ASC

OPERATION AND MAINTENANCE
INSTRUCTIONS
PERIPHERAL PROCESSOR
VOLUME 1 OF 2

Equipment Group
P.O. Box 2909
Austin, Texas 78767

TEXAS INSTRUMENTS
INCORPORATED

930182-2
December 1973

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

Note: The portion of the text affected by the changes is indicated by a vertical line in the outer margins of the page.

DATES OF ISSUE FOR ORIGINAL AND CHANGED PAGES ARE:

ORIGINAL . . . . . 0 . . . . . DECEMBER 1973

Page No.	Change No.	Page No.	Change No.	Page No.	Change No.
Title	0				
A Page	0				
Update	0				
iii - xvi	0				
1-1 - 1-30	0				
2-1 - 2-2	0				
3-1 - 3-3	0				
4-1 - 4-359	0				
I-1 - I-3	0				
Update	0				

*The asterisk indicates pages changed, added, or deleted by the current change.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

INTRODUCTION

PURPOSE OF MANUAL

This manual provides the information and instructions necessary for maintenance personnel to operate and maintain the Peripheral Processor, Texas Instruments part number 921444-1. This manual is one of a series of manuals prepared for the Advanced Scientific Computer (ASC) system.

CONTENT

This manual consists of seven sections and appendices, divided into two volumes. A brief description of each section is provided in the following paragraphs.

SECTION I (VOLUME I) GENERAL DESCRIPTION

This section provides a brief functional description of the Peripheral Processor, the manner in which the Peripheral Processor interfaces with the ASC system, and the part the Peripheral Processor plays in the ASC system. This section also contains a list of general operating characteristics, a brief introduction to the instruction repertoire, and a paragraph on the unique method of Peripheral Processor instruction processing.

SECTION II (VOLUME I) INSTALLATION

This section references the ASC System Installation Manual for Peripheral Processor installation instructions.

SECTION III (VOLUME I) OPERATING INSTRUCTIONS

This section references the ASC Maintenance Console OMI for the controls and indicators related to Peripheral Processor operations and the ECL Regulators OMI for the controls and indicators associated with the Peripheral Processor power supply regulators.

SECTION IV (VOLUME I) PRINCIPLES OF OPERATION

This section provides both a general and detailed description of the Peripheral Processor theory of operation. Both descriptions are based on the eight basic functional areas of the Peripheral Processor. The general description covers the eight functional areas at the block diagram level and the detailed description is based on logic diagrams and detailed block diagrams.

SECTION V (VOLUME II) MAINTENANCE

This section will be provided at a later date.

SECTION VI (VOLUME II) PARTS LISTING

This section will be provided at a later date.

SECTION VII (VOLUME II) DIAGRAMS

This section will be provided at a later date.

APPENDICES (VOLUME II)

This manual contains the following three appendices:

APPENDIX A - Peripheral Processor Transfer Tables

APPENDIX B - Communications Register File Map

APPENDIX C - External Maintenance System

SECTION I

GENERAL DESCRIPTION

1-1 GENERAL

This manual provides operation and maintenance instructions for the Peripheral Processor (figure 1-1) manufactured by Texas Instruments Incorporated as part of the Advanced Scientific Computer (ASC) system.

1-2 PURPOSE OF EQUIPMENT

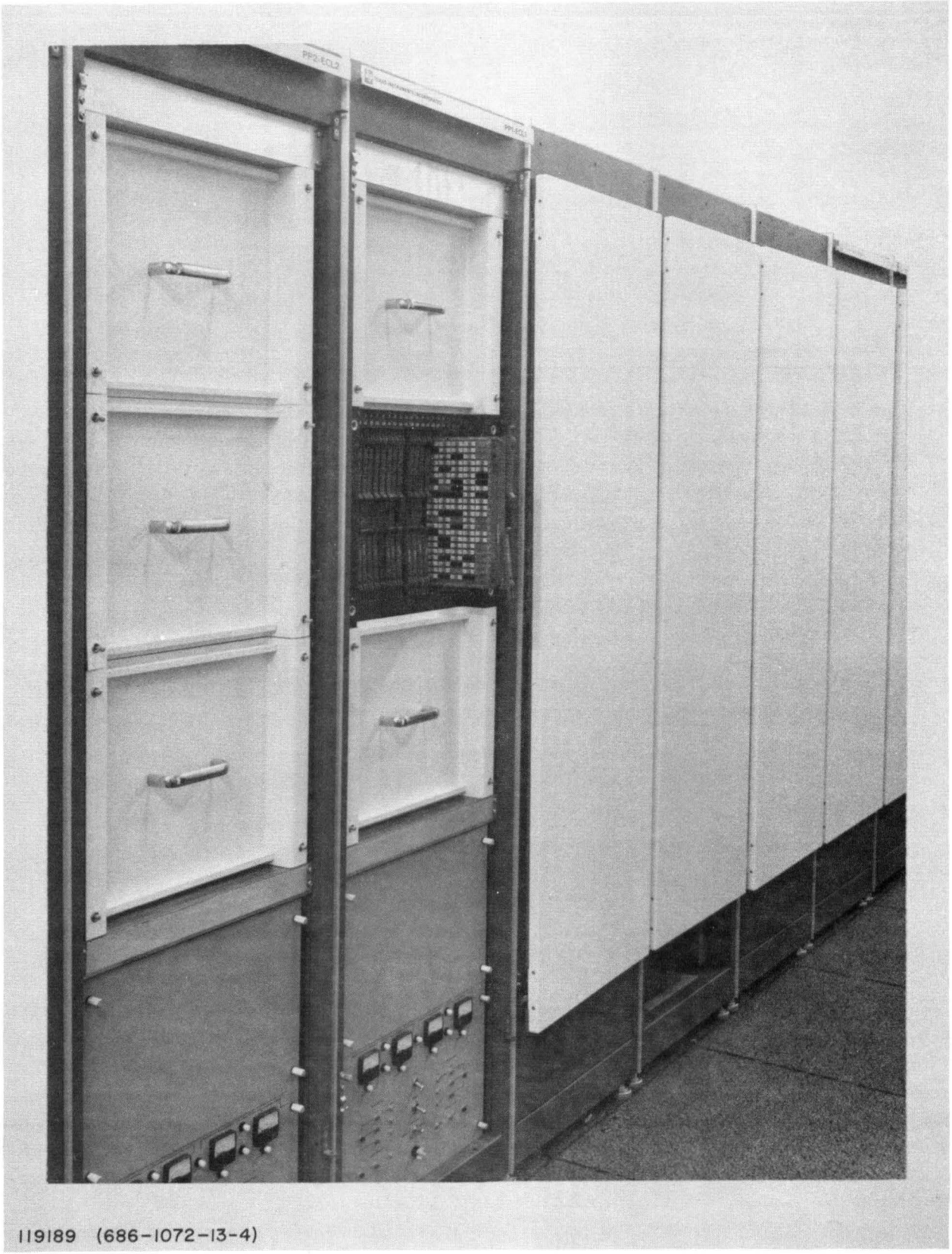
The Peripheral Processor (PP) is a versatile multiprocessor designed to control a variety of peripheral devices and perform management functions for the ASC system. The primary device interfaces of a typical ASC system are shown in figure 1-2. Within this configuration, the Peripheral Processor performs the following:

- Controls all ASC operations required to process User Programs
- Communicates with all peripheral devices
- Schedules tasks for the Central Processor
- Fulfills job requests that do not require the high arithmetic capabilities of the Central Processor
- Provides central control for ASC operational checkout and maintenance

The Peripheral Processor is readily adaptable to future changes in the peripheral scheme due to the large file of Communications Registers that provide the primary interface to the peripherals and the rich instruction repertoire designed to control these Communications Registers.

1-3 EQUIPMENT OVERVIEW

The Peripheral Processor structure includes eight identical processors, designated Virtual Processors (VP's), each capable of executing a separate program from Central Memory or the Peripheral Processor Read Only Memory. The Virtual Processors execute programs on a time-sharing basis under the influence of a time slot table that allocates real time to 16 table entries, each approximately 85 nanoseconds long. The time slot table is located in Communications Registers (CR's) eight and nine, as shown in figure 1-3. As real time progresses, the time slots are examined in the sequence shown in figure 1-3. Each time slot entry consists of an active bit and a three-bit Virtual Processor identification code. The active bit is



119189 (686-1072-13-4)

Figure 1-1. Peripheral Processor

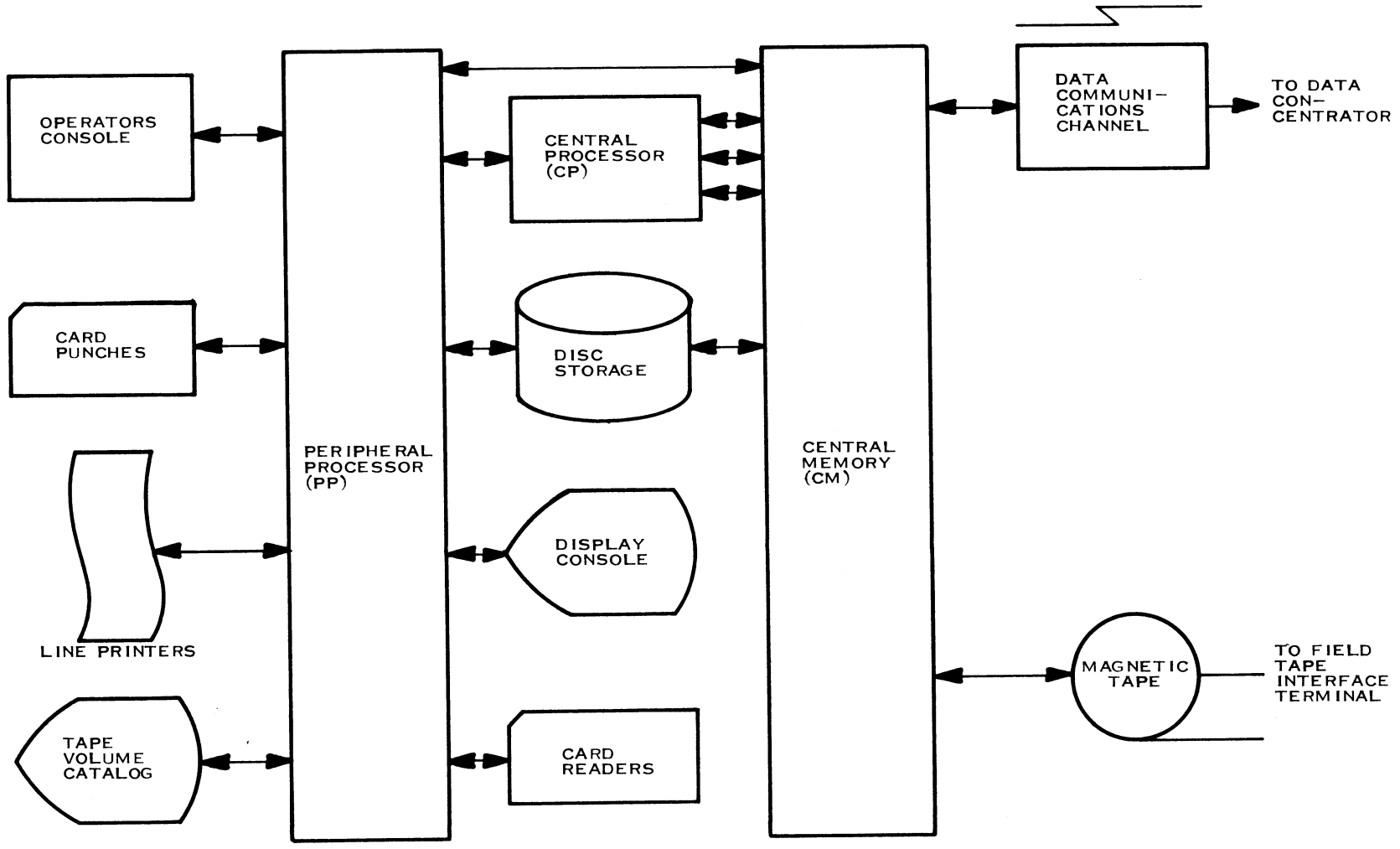
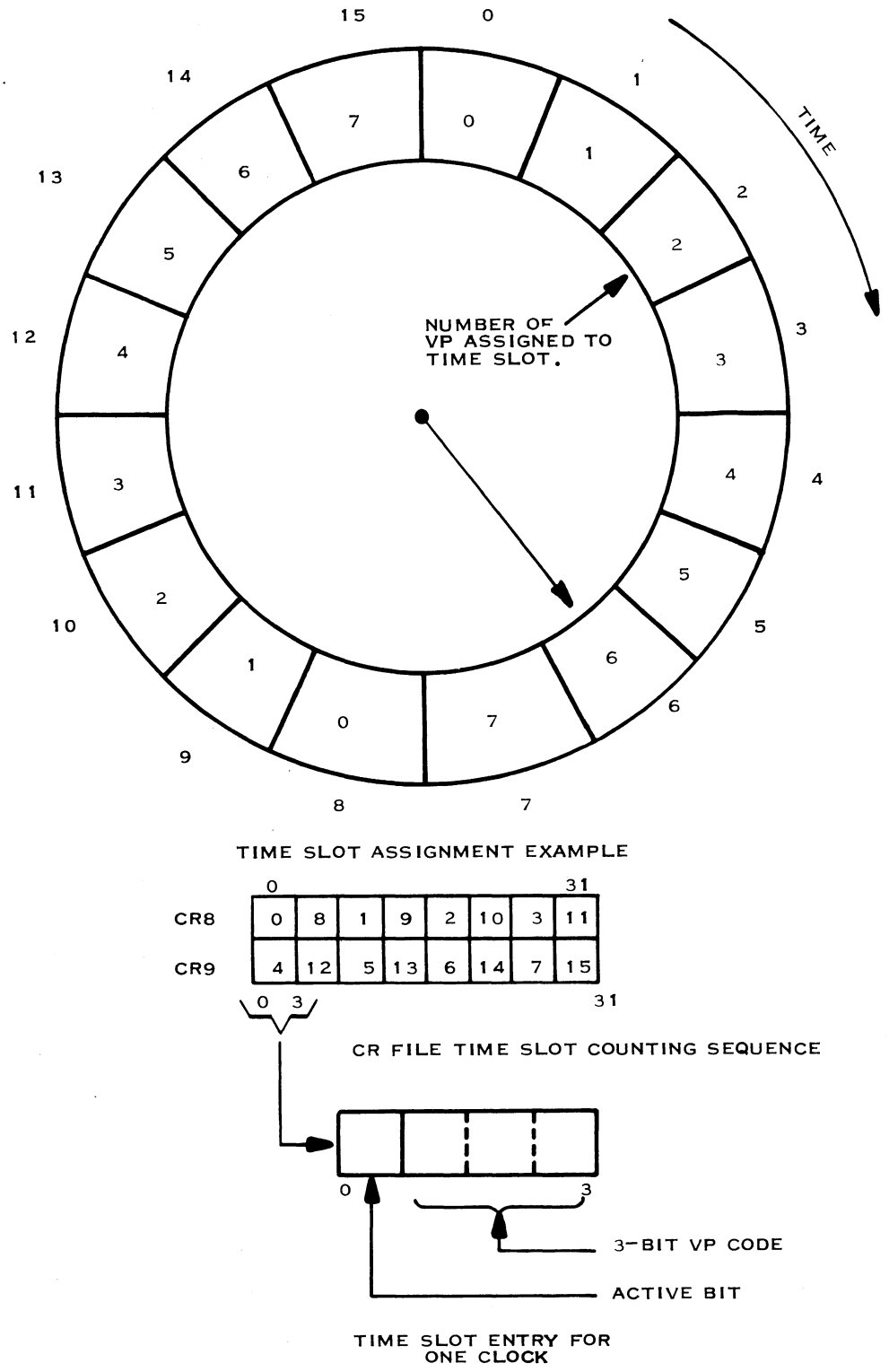


Figure 1-2. ASC System Simplified Block Diagram



(A) 124725

Figure 1-3. CR File Time Slot Assignments

set to indicate when a Virtual Processor is to use the time slot for program execution and the three-bit identification code identifies one of eight Virtual Processors to perform the execution. The example in figure 1-3 is one case of time slot assignment where each of the Virtual Processors is provided with two execution periods for one pass through the time slot table. Any other combination of Virtual Processor time slot assignments can be made; however, if one Virtual Processor is assigned two consecutive time slots, the second assignment will be voided due to hardware limitations.

One of the eight Virtual Processors is designated the master Virtual Processor and is assigned time slot zero. The master Virtual Processor then executes the Master Controller function (this includes making the initial time slot assignments) of the ASC Operating System (software) and assigns the remaining controller functions to other Virtual Processors. Any Virtual Processor not assigned a block of the Operating System can be used to control the individual jobs required to execute ASC system jobs. This type of job includes reading cards from one of the system card readers, printing out a memory buffer on one of the line printers, or initializing a Central Processor job. Virtual Processors not assigned work remain idle until activated by the master Virtual Processor.

1-4 SYSTEM INTERFACE

The eight Virtual Processors that make up the Peripheral Processor interface with the Central Processor and all other peripherals in the ASC system through a group of 64 32-bit registers designated the Communications Register (CR) file. Each hardware device in the system is assigned a portion of the CR file as detailed in Appendix B of this manual. The assigned bits in the CR file may be set or read by the associated peripheral devices (including the Central Processor) and all CR file bits may be set or read by any of the Virtual Processors.

The CR file provides the data paths for the paper peripheral devices and the control interface between the Virtual Processors and all units in the system (refer to figure 1-4). The CR file also monitors and processes system hardware and software interrupts, holds control information necessary to exercise the Peripheral Processor maintenance logic, and serves as the maintenance interface with the system whereby Virtual Processors execute diagnostic programs and monitor data, status, and control bits for each device in the system.

1-5 FUNCTIONAL DESCRIPTION

The Peripheral Processor (PP) consists of eight identical Virtual Processors (VP's) and the following major components that are time-shared by the VP's:

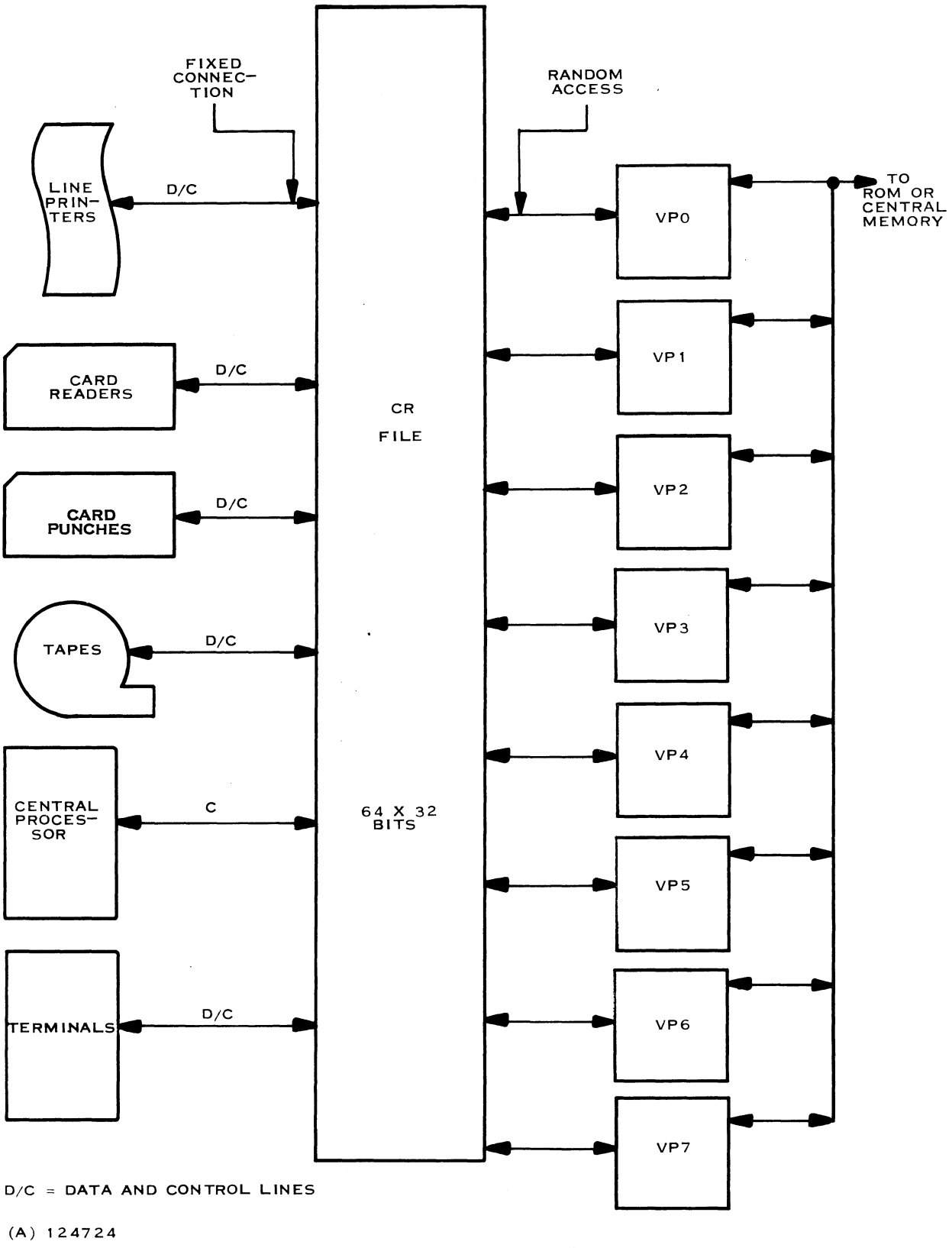


Figure 1-4. Simplified System Interface Diagram

- Arithmetic Unit
- Indexer
- Communications Register File
- Read Only Memory
- Single Word Buffer Controller
- Control
- Maintenance Logic

A simplified block diagram of the PP detailing the interfaces between the major components is shown in figure 1-5. A brief description of each component is given in the following paragraphs.

1-6 VIRTUAL PROCESSORS

The PP contains eight identical Virtual Processors (VP's), designated VP0 through VP7, used to execute the software necessary to control all ASC system operations.

The VP's execute their respective code according to the time-sharing method provided by the time slot table (described in the equipment overview paragraph of this section). Since only one of the eight VP's can be active at any given time, the other seven major components of the PP can be time-shared between the eight VP's. The net effect is eight separate processors, but with a considerable savings in logic.

Each VP has a large instruction repertoire (219 basic instructions and one no-operation instruction) and employs three-level instruction look-ahead to facilitate high-speed instruction processing. The instruction word retrieved from either Central Memory or Read Only Memory is 32 bits, but is expanded to 64 bits within each VP prior to instruction execution. Each of the VP's consists of the following register types:

- Program Counter Register
- Next Instruction Register
- Instruction Register
- Virtual Processor Register File
- Central Memory Base Register
- Single Word Buffer Address Register
- Single Word Buffer Data Register

Figure 1-6 ties all of the VP register types together, along with some of the other major PP components. The Program Counter (PC) is a 32-bit register

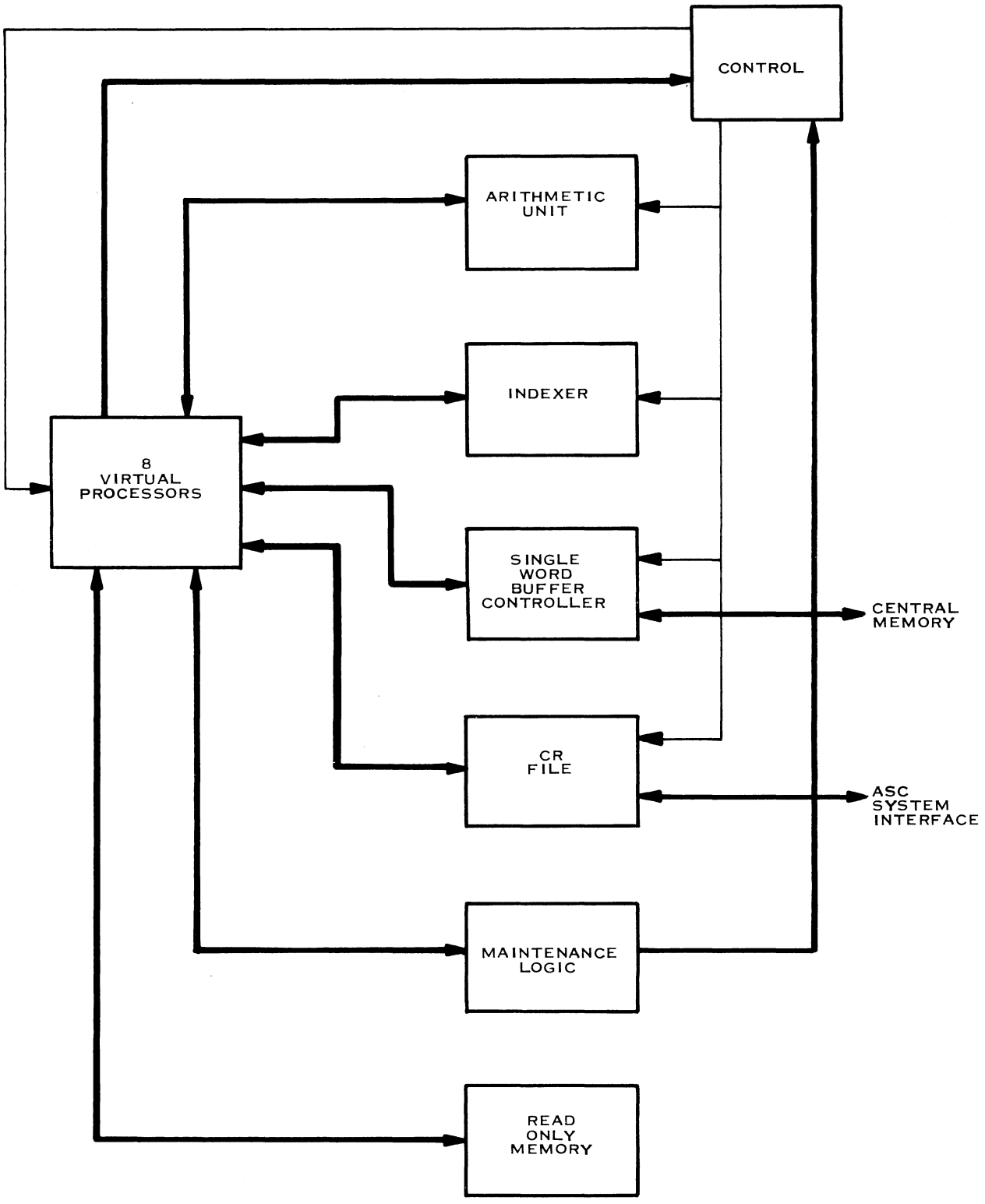


Figure 1-5. Peripheral Processor Simplified Block Diagram

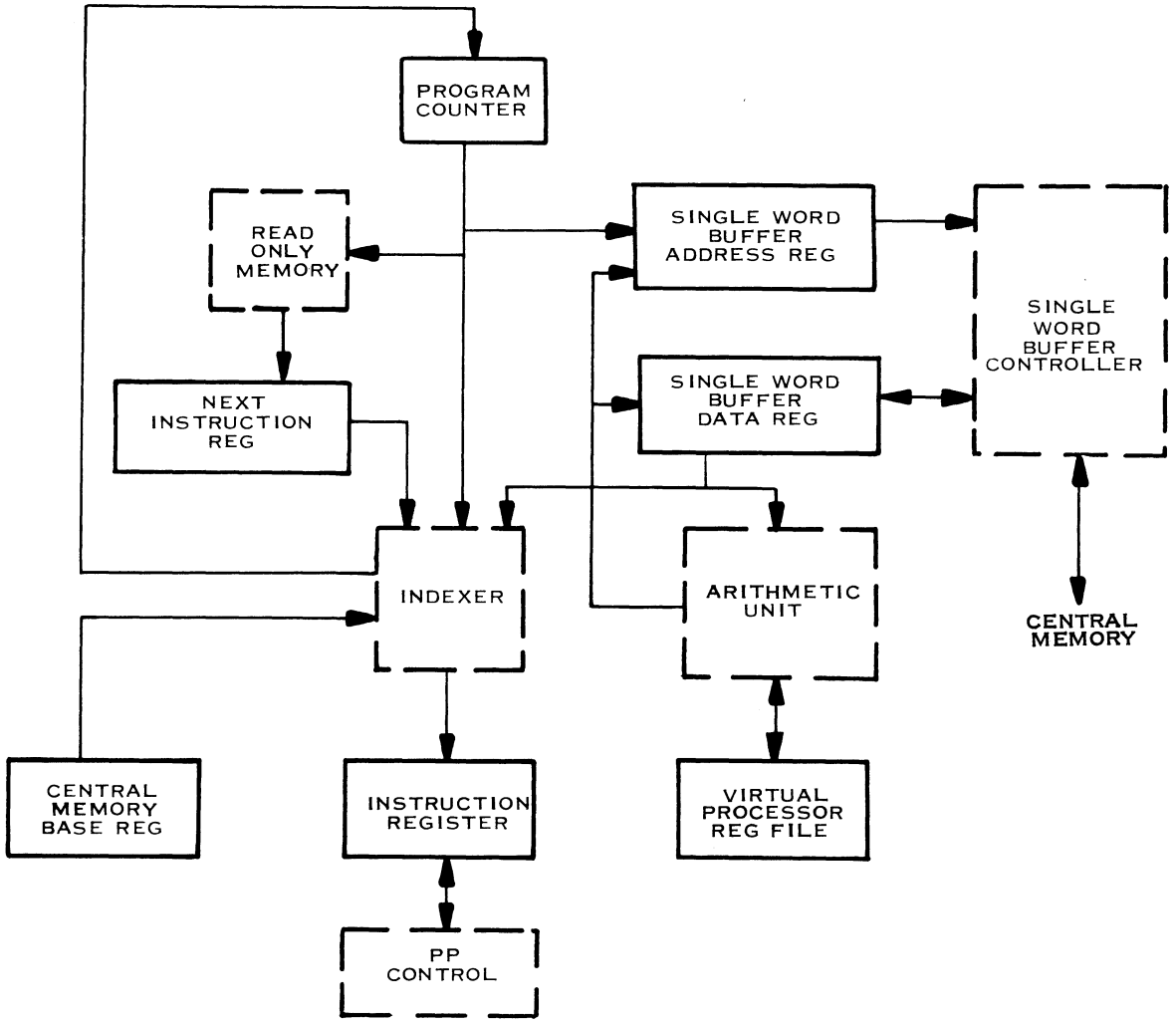


Figure 1-6. Virtual Processor Simplified Block Diagram

used to hold the address of the next instruction to be retrieved from Central Memory or Read Only Memory. The program counter is updated by the Indexer during each active time slot period to point to the next sequential instruction in the executing program. The Single Word Buffer Address (SWBA) register is a 32-bit register used to hold the address necessary to access Central Memory. The SWBA accepts PC data during the normal instruction acquisition process and PP Control data via the Arithmetic Unit when the executing instruction requires use of Central Memory for a store, load, or branch type instruction. The Single Word Buffer Data (SWBD) register is a 32-bit register that provides temporary storage of data being written to or read from Central Memory. The SWBD accepts data from Central Memory

via the Single Word Buffer Controller when a read operation is being performed or from the VP register addressed by PP Control via the Arithmetic Unit when a write operation is being performed. When a write is executed, the SWBD data is input to the Single Word Buffer Controller and eventually, Central Memory. When a read is executed, the SWBD data is distributed to the Indexer (for Instruction Register address development) and Arithmetic Unit (for arithmetic and load type operations).

The Next Instruction Register (NIR) is a 32-bit register used to hold instructions retrieved from Read Only Memory prior to their transfer to the Instruction Register via the Indexer. The Instruction Register (IR) is a 64-bit register used to hold the expanded instruction developed by the Indexer. The expanded instruction contains the control information necessary for execution, and is input to PP Control when the associated VP is active. The Virtual Processor Register (VPR) file consists of four 32-bit registers, designated VPR0 through VPR3, used as general purpose accumulator registers. The VPR file is different from the other VP registers in that they can be addressed on the byte, halfword, or word level. The Central Memory (CM) Base register is a 24-bit register located in the Communications Register file and used to hold a base value for base relative address development.

1-7 ARITHMETIC UNIT

The Arithmetic Unit (AU) is time shared by the eight Virtual Processors to perform addition, subtraction, logical functions (AND, OR, EXCLUSIVE OR, and EQUIVALENCE), data shifting, data testing, and bit setting and resetting. The AU is capable of handling two 32-bit words at one time (for addition, subtraction, and logical functions) or a single 32-bit word and operating on it down to the bit level. The data handled by the AU is supplied by the active VP and the operation performed is under direction of PP Control. The AU accepts one operand from the VPR file and a second operand from a different VPR, a Communications Register, or the SWBD.

1-8 INDEXER

The Indexer is time shared by the eight Virtual Processors to perform Program Counter (PC) indexing and to develop addresses for the Instruction Register (IR). The PC related portion of the Indexer increments the current PC value by one during the normal instruction acquisition process, decrements the current PC value by one when the address of the next instruction needs to be saved (this is due to the three-level look-ahead feature of the PP), or decrements the current PC value by two when an interrupt occurs (this is done so that the instruction following the interrupted instruction is not skipped). The IR address development portion of the Indexer adds various combinations of the PC, CM base register, NIR, SWBD, and VPR file under the influence of PP control to develop source, destination, and effective addresses for the IR. This address development is the primary reason

for the expansion of the 32-bit SWBD or NIR instruction to the 64-bit IR instruction.

1-9 COMMUNICATIONS REGISTER FILE

The Communications Register (CR) file consists of 64 32-bit registers accessible to all eight Virtual Processors. Each of the 64 CR's can be addressed down to the bit level. The CR file holds the necessary data to provide the PP/ASC system interface, maintenance control for the PP, system interrupt monitoring and control, VP time slot and priority assignments (each VP is assigned a high or low priority for use in honoring CM access requests), real time clock information, CM base operands, and temporary storage as required by the PP for normal operation.

1-10 READ ONLY MEMORY

The Read Only Memory (ROM) is time shared by the eight Virtual Processors to reduce Central Memory requirements by providing storage for up to 4K of 32-bit instructions, used primarily for control programs associated with input/output devices of the ASC system. The ROM is extremely fast (25 nanosecond access time) and is addressed by the PC under the influence of PP Control. The memory is organized into 16 256-word modules, so that portions of the contained programs can be modified without complete refabrication of the memory.

1-11 SINGLE WORD BUFFER CONTROLLER

The Single Word Buffer Controller (SWBC) is time shared by the eight Virtual Processors to provide an interface to the Memory Control Unit (MCU) and Central Memory (CM) for CM read and write operations. The SWBC accepts memory access requests from the active VP (assuming the active VP does not already have a request pending), notifies the MCU when a request is present, and provides a data and address path to CM to execute the highest priority request under direction of the MCU. The SWBC/MCU interface honors all VP's assigned a high priority on a first come, first served basis, and then honors all VP's assigned a low priority in the same manner. The ASC Operating System (software) is responsible for making the priority assignments.

1-12 CONTROL

The PP Control is time shared by the eight Virtual Processors to provide the controls and enables necessary to develop and execute PP instructions. The center of PP control, the Main Instruction Register (MIR), holds the IR of the active VP for the duration of the assigned time slot so the Control logic can expand the IR to 256 bits and distribute the resulting control signals

throughout the PP. PP Control constantly monitors the CR file for reported interrupts and the SWBC for the status of the active VP when generating the controls and enables for instruction execution. The normal operation of the PP Control logic can be disabled by the Maintenance Logic, for example, when a VP or the ASC Maintenance Console operator places some portion of the PP under test. When some portion of the PP is placed under test, the Maintenance Logic replaces the IR of the active VP as the source of control data.

1-13 MAINTENANCE LOGIC

The PP Maintenance Logic provides a means of checking the operation of the other seven major components of the PP (VP's, AU, Indexer, CR file, ROM, SWBC, and PP Control). In addition, when the PP is operating normally (versus the test mode when some portion of the PP is under test), primary functions of the Maintenance Logic include supplying the PP with the active VP code (VP code is the number associated with a VP) and the priority of the active VP as it relates to the SWBC. Control of the PP Maintenance Logic is provided by a group of registers in the CR file called maintenance registers. The method of entering data in the maintenance registers is controlled by the ASC Maintenance Console. When the manual mode of operation is selected, the operator enters data directly into the maintenance registers via the console. When the semi-automatic mode of operation is selected, a card reader supplies data to the maintenance registers. When the automatic mode of operation is selected, an active VP supplies data to the maintenance registers. The maintenance command repertoire can be used in any of these three methods and is versatile enough to completely check the PP.

1-14 INSTRUCTION REPERTOIRE

The Peripheral Processor instruction repertoire applies to all eight VP's. There are 219 basic instructions (and one no-operation instruction) that fall into one of the following major instruction groups:

- Stores
- Loads
- Arithmetic
- Logical
- Compare and skip
- Unconditional branches
- Stack
- Set/reset CR bits
- Test CR bits and skip

- Shifts
- Test CR bits, set/reset, and skip
- Increment/decrement and test conditional branches
- Arithmetic conditional branches
- Miscellaneous

A listing of the instruction types and the data handled within each of these major groups is provided in table 1-1.

1-15 INSTRUCTION FORMAT

The basic Peripheral Processor instruction consists of 32 bits divided into four fields, as shown in figure 1-7. The operation code field (op-code, bits 0 through 7) is an 8-bit code, usually represented as a two-character hexadecimal (base 16) number, used to identify one of 220 (including the no-operation instruction) valid instructions. Each of the instructions is assigned a unique hardware mnemonic (versus the software mnemonics listed in table 1-1 that group instructions together when they perform the same operation with different data sources) but can be grouped with other instructions, as shown in the transfer tables in appendix B, when the execution steps involved are examined.

The R field (bits 8 through 11) is a 4-bit code used to specify a VPR or CR at the word, halfword, or byte level, depending on the accompanying op-code. When a VPR needs to be specified, the R field is all that is necessary. When a CR needs to be specified, the R field must be added to byte 3 of VPR3 so that all 64 CR's can be addressed down to the byte level (the 4-bit R field can only address four 32-bit words to the byte level by itself). The R field also serves as a mask for some instructions when bits within a CR half byte (hex) need designating.

The T field (bits 12 through 15) is four bits in length and is used to specify both indirect addressing and whether a VPR halfword is to be added to the quantity designated by the N field (this add operation is referred to as indexing). Since only eight VPR halfwords are available to any one VP, only the three least significant bits of the T field are used to specify a VPR halfword and the most significant bit is used to indicate when the current instruction is indirect. The three least significant bits set to zero reflects no indexing, rather than indexing with the left half of VPR0. The left half of VPR0 can not be used for indexing.

The N field (bits 16 through 31) consists of 16 bits used to specify an immediate operand, Central Memory address, ROM address, branch address, VPR, or CR, depending on the accompanying op-code.

Table 1-1. Peripheral Processor Instructions

Group/Software Mnemonic	Description
Store Instructions	
ST	Store word from {VPR or CR}* to {CM VPR or CR}*
STA	Store word from VPR to CM absolute
STH	Store halfword from {VPR or CR} to {VPR or CR}
STB	Store byte from {VPR or CR} to {VPR or CR}
STL	Store from {VPR or CR} to left half CM
STR	Store from {VPR or CR} to right half CM
STF	Store file from VPR to CM
Load Instructions	
LD	Load word to {VPR CR} from {CM VPR or CR}
LDA	Load word to VPR from CM absolute
LDI	Load immediate word into {VPR or CR}
LDH	Load halfword to {VPR or CR} from {VPR or CR}
LDHI	Load immediate halfword into {VPR or CR}
LDB	Load byte to {VPR or CR} from {VPR or CR}
LDBI	Load immediate byte into {VPR or CR}

*VPR - Virtual Processor Register CM - Central Memory
CR - Communication Register

Table 1-1. Peripheral Processor Instructions (Continued)

Group/Software Mnemonic	Description
Load Instructions (continued)	
LDL	Load to {VPR or CR} from left half CM
LDR	Load to {VPR or CR} from right half CM
LDF	Load file from CM to VPR
Arithmetic Instructions	
AD	Add word from {CM or VPR} to VPR
ADI	Add immediate word to VPR
ADH	Add halfword in VPR to VPR
ADHI	Add immediate halfword to VPR
ADB	Add byte in VPR to VPR
ADBI	Add immediate byte to VPR
ADL	Add left half in CM to VPR
ADR	Add right half in CM to VPR
SU	Subtract word in {CM or VPR} from VPR
SUI	Subtract immediate word from VPR
SUH	Subtract halfword in VPR from VPR
SUHI	Subtract immediate halfword from VPR
SUB	Subtract byte in VPR from VPR
SUBI	Subtract immediate byte from VPR
SUL	Subtract left half in CM from VPR
SUR	Subtract right half in CM from VPR
Logical Instructions	
OR	Logical OR word in {CM VPR or CR} to VPR

Table 1-1. Peripheral Processor Instructions (Continued)

Group/Software Mnemonic	Description
Logical Instructions (continued)	
ORH	Logical OR halfword in {VPR or CR} to VPR
ORHI	Logical OR immediate halfword to VPR
ORB	Logical OR byte in {VPR or CR} to VPR
ORBI	Logical OR immediate byte to VPR
ORL	Logical OR left half in CM to VPR
ORR	Logical OR right half in CM to VPR
EX	Logical exclusive OR word in {CM VPR or CR} to VPR
EXH	Logical exclusive OR halfword in {VPR or CR} to VPR
EXHI	Logical exclusive OR immediate halfword to VPR
EXB	Logical exclusive OR byte in {VPR or CR} to VPR
EXBI	Logical exclusive OR immediate byte to VPR
EXL	Logical exclusive OR left half CM to VPR
EXR	Logical exclusive OR right half CM to VPR
AN	Logical AND word in {CM VPR or CR} to VPR
ANH	Logical AND halfword in {VPR or CR} to VPR
ANHI	Logical AND immediate halfword to VPR
ANB	Logical AND byte in {VPR or CR} to VPR
ANBI	Logical AND immediate byte to VPR
ANL	Logical AND left half in CM to VPR
ANR	Logical AND right half in CM to VPR

Table 1-1. Peripheral Processor Instructions (Continued)

Group/Software Mnemonic	Description
Logical Instructions (continued)	
EQ	Logical EQUIVALENCE word {CM / VPR or / CR} to VPR
EQH	Logical EQUIVALENCE halfword {VPR or / CR} to VPR
EQHI	Logical EQUIVALENCE immediate halfword to VPR
EQB	Logical EQUIVALENCE byte {VPR or / CR} to VPR
EQBI	Logical EQUIVALENCE immediate byte to VPR
EQL	Logical EQUIVALENCE left half CM to VPR
EQR	Logical EQUIVALENCE right half CM to VPR
Compare and Skip Instructions	
CE	Compare word {CM / VPR or / CR} to VPR, skip if equal
CEI	Compare immediate word with VPR, skip if equal
CEH	Compare halfword {VPR or / CR} to VPR, skip if equal
CEHI	Compare immediate halfword with VPR, skip if equal
CEB	Compare byte {VPR or / CR} to VPR, skip if equal
CEBI	Compare immediate byte with VPR, skip if equal
CEL	Compare left half of CM to VPR, skip if equal
CER	Compare right half CM to VPR, skip if equal
CN	Compare word {CM / VPR or / CR} to VPR, skip if not equal
CNI	Compare immediate word with VPR, skip if not equal

Table 1-1. Peripheral Processor Instructions (Continued)

Group/Software Mnemonic	Description
Compare and Skip Instructions (continued)	
CNH	Compare halfword {VPR or CR} to VPR, skip if not equal
CNHI	Compare immediate halfword with VPR, skip if not equal
CNB	Compare byte {VPR or CR} to VPR, skip if not equal
CNBI	Compare immediate byte with VPR, skip if not equal
CNL	Compare left half CM to VPR, skip if not equal
CNR	Compare right half CM to VPR, skip if not equal
Unconditional Branch Instructions	
BC	Branch unconditionally to CM, base relative
BCS	Branch unconditionally to CM relative base, save PC in VPR
BCA	Branch unconditionally to absolute CM
BCAS	Branch unconditionally to CM absolute, save PC in VPR
BPC	Branch unconditionally to CM, relative PC
BPCS	Branch unconditionally to CM, relative PC, save PC in VPR
BR	Branch unconditionally to ROM
BRS	Branch unconditionally to ROM, save PC in VPR
BRSM	Branch unconditionally to absolute ROM, save PC in fixed CM location
Stack Instructions	
PUSH	Push word from VPR into stack
PULL	Pull word from stack into VPR
MOD	Modify stack

Table 1-1. Peripheral Processor Instructions (Continued)

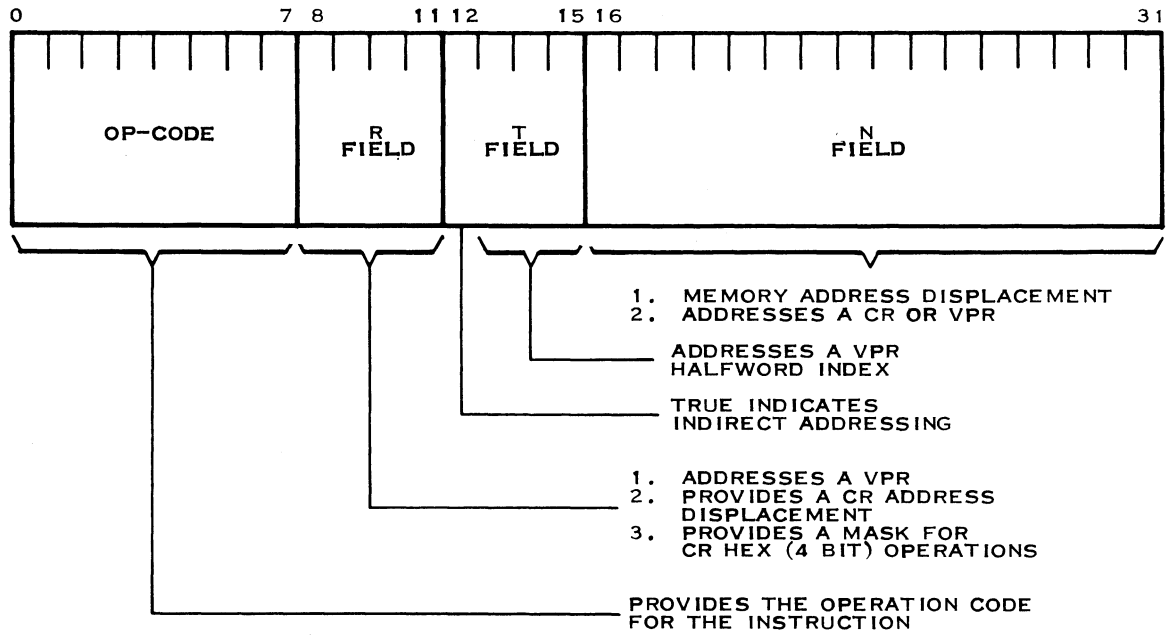
Group/Software Mnemonic	Description
Set/Reset CR Bits Instructions	
VPS	Set VP flag in CR
VPR	Reset VP flag in CR
VPTO	Test VP flag in CR, skip if equal to one
VPTZ	Test VP flag in CR, skip if equal to zero
SL	Set mask bits in left half of CR byte
SR	Set mask bits in right half of CR byte
RL	Reset mask bits in left half of CR byte
RR	Reset mask bits in right half of CR byte
Test CR Under Mask and Skip Instructions	
TOL	Test under mask for any ones in left half of CR byte and skip if true
TOR	Test under mask for any ones in right half of CR byte and skip if true
TZL	Test under mask for any zeros in left half of CR byte and skip if true
TZR	Test under mask for any zeros in right half of CR byte and skip if true
TAOL	Test under mask for all ones in left half of CR byte and skip if true
TAOR	Test under mask for all ones in right half of CR byte and skip if true
TAZL	Test under mask for all zeros in left half of CR byte and skip if true
TAZR	Test under mask for all zeros in right half of CR byte and skip if true
Shift Instructions	
SHL	Shift logical word in VPR
SHA	Shift arithmetic word in VPR
SHC	Shift cyclic word in VPR

Table 1-1. Peripheral Processor Instructions (Continued)

Group/Software Mnemonic	Description
Test CR Bits, Set/Reset, and Skip Instructions	
TSZL	Test under mask for any zeros in left half of CR byte and set; then skip if true
TSZR	Test under mask for any zeros in right half of CR byte and set; then skip if true
TSOL	Test under mask for any ones in left half of CR byte and set; then skip if true
TSOR	Test under mask for any ones in right half of CR byte and set; then skip if true
TRZL	Test under mask for any zeros in left half of CR byte and reset; then skip if true
TRZR	Test under mask for any zeros in right half of CR byte and reset; then skip if true
TROL	Test under mask for any ones in left half of CR byte and reset; then skip if true
TROR	Test under mask for any ones in right half of CR byte and reset; then skip if true
Increment/Decrement and Test Conditional Branch Instructions	
IBZ	Increment VPR by one; branch if result equal to zero
IBN	Increment VPR by one; branch if result not equal to zero
DBZ	Decrement VPR by one; branch if result equal to zero
DBN	Decrement VPR by one; branch if result not equal to zero
Arithmetic Conditional Branch Instructions	
TZ	Test {VPR or CR} word arithmetically; branch if equal to zero
TZH	Test {VPR or CR} halfword arithmetically; branch if equal to zero

Table 1-1. Peripheral Processor Instructions (Continued)

Group/Software Mnemonic	Description
Arithmetic Conditional Branch Instructions (continued)	
TZB	Test {VPR or CR} byte arithmetically; branch if equal to zero
TN	Test {VPR or CR} word arithmetically; branch if not equal to zero
TNH	Test {VPR or CR} halfword arithmetically; branch if not equal to zero
TNB	Test {VPR or CR} byte arithmetically; branch if not equal to zero
TP	Test {VPR or CR} word arithmetically; branch if greater than or equal to zero
TPH	Test {VPR or CR} halfword arithmetically; branch if greater than or equal to zero
TPB	Test {VPR or CR} byte arithmetically; branch if greater than or equal to zero
TM	Test {VPR or CR} word arithmetically; branch if less than zero
TMH	Test {VPR or CR} halfword arithmetically; branch if less than zero
TMB	Test {VPR or CR} byte arithmetically; branch if less than zero
Miscellaneous Instructions	
LDEA	Load effective address into VPR
ANAZ	Analyze CM
POLL	Poll CR and set VPR
EXEC	Execute CM
LDMB	Load VP base from VPR
NOP	No operation



(A) 111645

Figure 1-7. Peripheral Processor Instruction Format

1-16 DATA FORMATS

The Peripheral Processor handles data at the word (32 bits), halfword (16 bits), and byte (8 bits) levels and encounters the indirect cell format when an instruction (or other indirect cell) specifies indirect addressing. Figure 1-8 presents all four of these types of formats. The sign bits used in the word, halfword, and byte formats reflect whether the associated data is positive or negative (in two's complement form). When any of these three formats is involved in an arithmetic operation, any overflow conditions are ignored. The Peripheral Processor is also capable of addressing CR file data down to the individual bits, but this is only for testing and setting/resetting purposes (the smallest unit of data that can be transferred between registers is the byte).

The most significant bit of the indirect cell is used during the terminal (last) level of indirect addressing to reflect the source (Central Memory or ROM) of a branch address. The T field in the indirect cell serves the same purpose as the T field in the instruction format, and the 24-bit address field specifies the base operand address subject to modification by the T field. The operand address developed from the indirect cell always references Central Memory, whereas the first level of indirect addressing from the instruction format can reference a CR, VPR, or Central Memory.

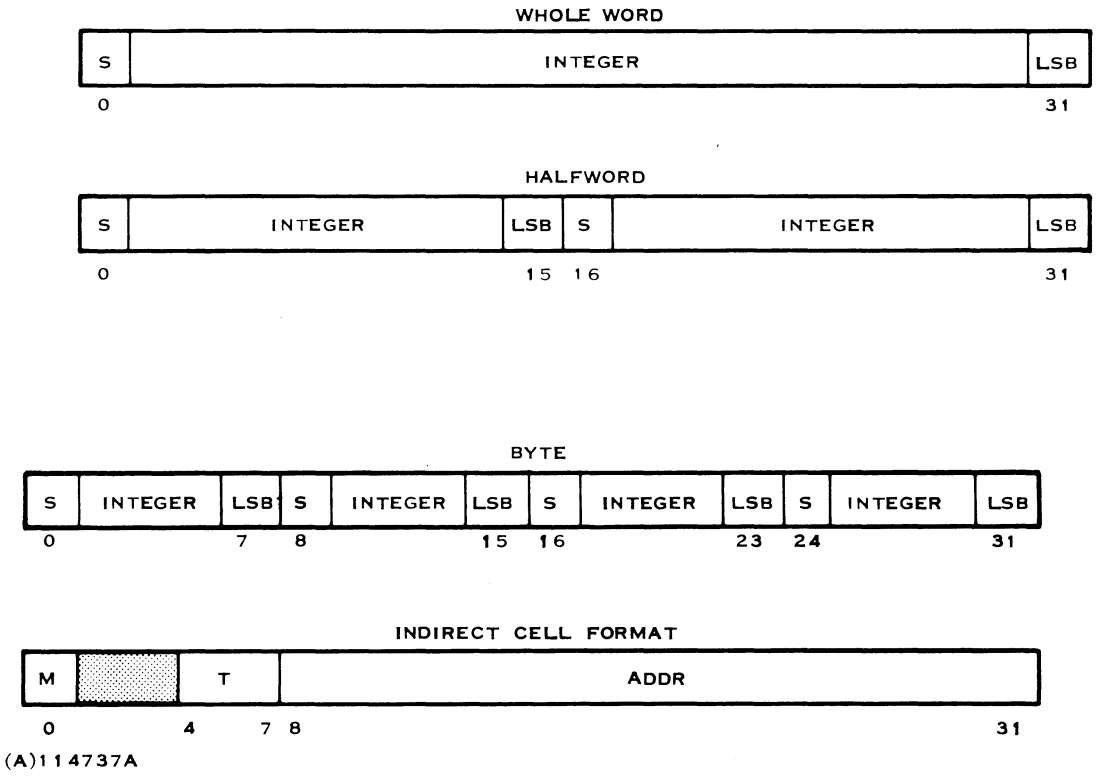


Figure 1-8. Peripheral Processor Data Formats

1-17 INSTRUCTION PROCESSING

Each of the active Virtual Processors retrieves, expands, and executes program instructions residing in either Central Memory (CM) or ROM in a continuous three-step procedure. Refer to the Virtual Processor (VP) block diagram (figure 1-6) during the following discussion. The first step, addressing the instruction to be retrieved, involves operation of the Program Counter (PC). When an instruction terminates in the VP on which this discussion is based, the PC address is input to either ROM or the Single Word Buffer Address (SWBA) Register, under the influence of PP Control. ROM responds to the address by supplying the Next Instruction Register (NIR) with an instruction and CM responds to the read request from the Single Word Buffer Controller (SWBC) and the SWBA address by supplying the Single Word Buffer Data (SWBD) Register with an instruction. When another instruction terminates, the second of three steps (instruction expansion) occurs. The retrieved instruction, from either the NIR or SWBD, combines with the Indexer and PP Control to develop the state, control flags, and address information necessary for execution of the first step of the instruction. All of this data is input to the Instruction Register (IR) for the third step (instruction execution). At the next active time slot, the IR data is input to PP Control, where the instruction is expanded again to perform the execution step. If the instruction requires more than one step (as is the case with most of

the PP instructions due to their involvement), PP Control updates the IR so that the next step will be executed at the next active time slot. When the last step of an instruction is executed by PP Control, the VP is notified so that a new instruction can be expanded into the IR, the following instruction can be retrieved from one of the two memories, and the third succeeding instruction can be addressed by the PC. In this manner, all of the active VP's are able to maintain a continual flow of instructions so that little time is wasted in program execution.

1-18 GENERAL CHARACTERISTICS

A condensed listing of Peripheral Processor general characteristics is provided in table 1-2.

Table 1-2. Peripheral Processor General Characteristics

- Eight medium-power time-shared processors (called Virtual Processors)
- Provides system control via ASC Operating System (software)
- Interfaces with ASC system via group of 64 Communications Registers
- Contains 364 synchronizers for asynchronous operation with peripheral devices
- Contains built-in maintenance logic to facilitate checkout and trouble-shooting
- Implemented with ECL2500 series high-speed logic (logic 0 ≈ +400mv, logic 1 ≈ -400mv)
- Constructed with multilayer etched motherboards and plug-in multilayer printed wiring boards
- Basic 32-bit instruction, expanded to 64 bits at execution
- Versatile data-handling instruction set of 219 basic instructions and one no-operation instruction
- Capable of handling and operating on data in 32, 16, 8, and 1-bit groups
- Performs following arithmetic operations:
    - Addition
    - Subtraction (2's complement)
    - Logical AND, OR, EXCLUSIVE OR, and EQUIVALENCE
    - Shifts
    - Bit setting/resetting

Table 1-2. Peripheral Processor General Characteristics (Continued)

- Two priority levels for Virtual Processor access requests to Central Memory
- Monitors and processes following types of interrupts:
  - System A/C power failure
  - Pressing of Operator's Console STOP button
  - Central Memory parity error
  - Central Memory protect violation
  - Disc protect violation
  - Illegal Peripheral Processor instruction
  - Central Processor interrupt
- Clock rate of 85 nanoseconds
- Contains 4K by 32 bits of Read Only Memory with 25 nanoseconds access time
- Relies on asynchronous interface with Central Memory for primary instruction source

1-19 PHYSICAL DESCRIPTION

The Peripheral Processor consists of two Emitter Coupled Logic (ECL) columns in the ASC system complex. Each ECL column contains three motherboards, with a maximum of 22 logic cards per motherboard, and a voltage regulator. Figure 1-9 illustrates the placement of the two ECL columns, the six motherboards and their associated logic cards, and the voltage regulators. Figure 1-10 shows the actual logic card locations on the six motherboards. Table 1-3 groups the logic cards with the eight major Peripheral Processor components.

The motherboards are multilayer etched boards with 22 connectors designed to accept plug-in multilayer logic cards. The logic card connectors mounted in the motherboards have pins that extend through to the back of the motherboards for use as oscilloscope connections. The input/output lines for the motherboards are handled by 24 pin card edge connectors usually referred to as "spade" connectors. The wiring to the spade connectors is all coaxial and is routed around the motherboards to provide access to the pins on the back. The portion of this coax harness that provides input/output to the Peripheral Processor is routed to either of the two ECL column side panels (usually referred to as "bulkheads").

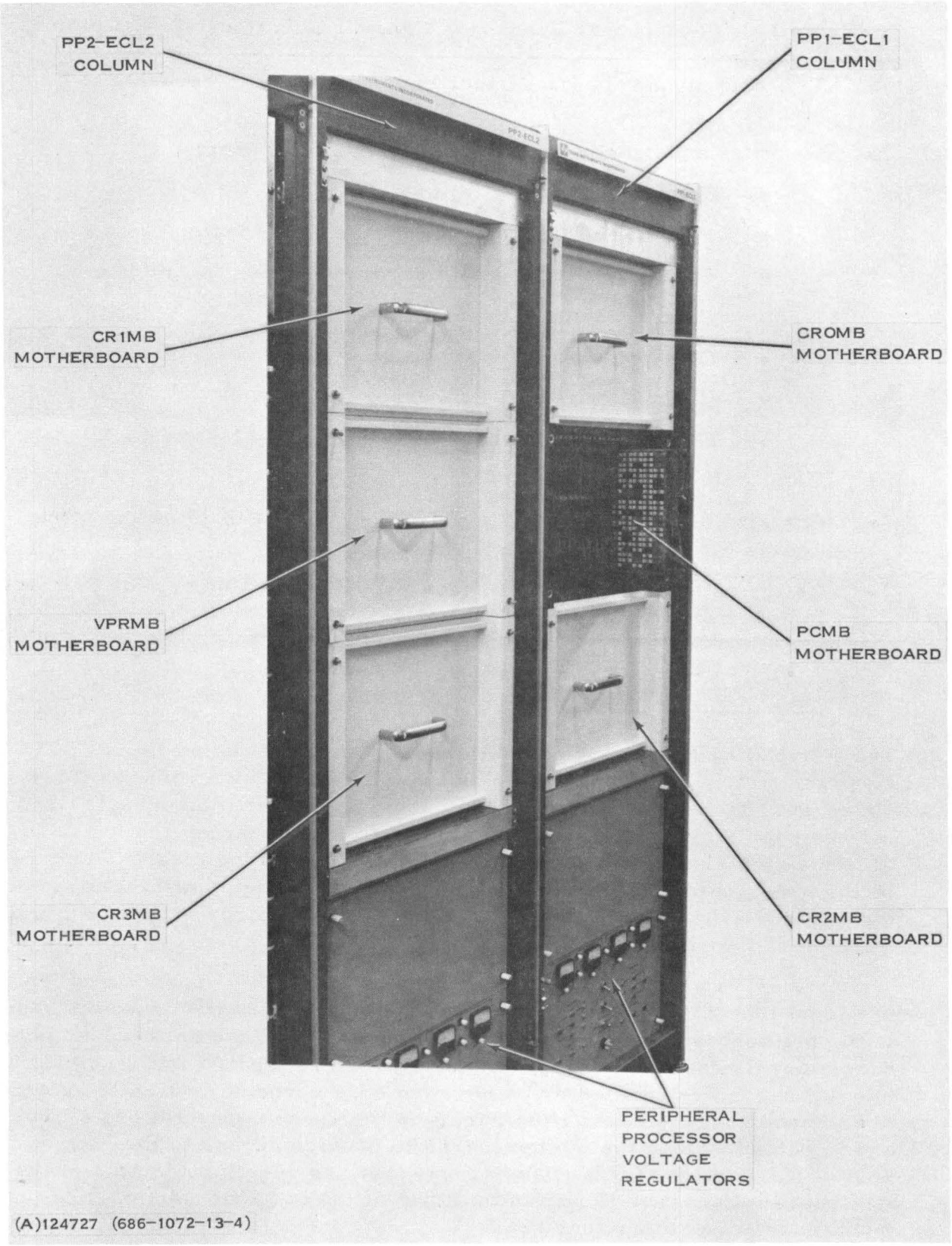


Figure 1-9. Peripheral Processor Assemblies

Motherboard Locations	Motherboards		
	CR0MB	PCMB	CR2MB
LA		TERMCARD	
LB		INDEXER(1)	
LC	CRCELLY	INDEXER(0)	CRBASE2
LD	PBOCRD(0)	PCCARDA(7)	PB2CRD(0)
LE	CRCELL0(0)	PCCARDA(6)	CRCELL2(0)
LF	CRCELL0(1)	PCCARDA(5)	CRCELL2(1)
LG	PBOCRD(1)	PCCARDA(4)	PB2CRD(1)
LH	CRCELL0(2)	PCCARDA(3)	CRCELL2(2)
LI	CRCELL0(3)	PCCARDA(2)	CRCELL2(3)
LJ	PBOCRD(2)	PCCARDA(1)	PB2CRD(2)
LK	CRCELL0(4)	PCCARDA(0)	CRCELL2(4)
LL	CRCELL0(5)	PCCTL	CRCELL2(5)
LM	PBOCRD(3)	SWBASY	PB2CRD(3)
LN	CRCELL0(6)	SWBSYNC	CRCELL2(6)
LO	CRCONT0	PPCTL2	CRCONT2
LP	MLCTL	PPCTL1	ML2
LQ	ROMCRD(8)	MIRMRGB	ROMCRD(12)
LR	ROMCRD(9)	IRCARD(3)	ROMCRD(13)
LS	ROMCRD(4)	IRCARD(2)	ROMCRD(6)
LT	CRROMRG(0)	IRCARD(1)	CRROMRG(2)
LU	CRMIRLDR	IRCARD(0)	
LV		TERMCARD	

Figure 1-10. Peripheral Processor Logic Card Locations (Sheet 1 of 2)

Motherboard Locations	Motherboard		
	CR1MB	VPRMB	CR3MB
LA		TERMCARD	
LB		ROMCRD(3)	
LC	CRBASE1	ROMCRD(2)	CRBASE3
LD	PB1CRD(0)	ROMCRD(1)	PB3CRD(0)
LE	CRCELL1(0)	ROMCRD(0)	CRCELL3(0)
LF	CRCELL1(1)	ROMMRG	CRCELL3(1)
LG	PB1CRD(1)	PPAUCD(3)	PB3CRD(1)
LH	CRCELL1(2)	PPAUCD(2)	CRCELL3(2)
LI	CRCELL1(3)	PPAUCD(1)	CRCELL3(3)
LJ	PB1CRD(2)	PPAUCD(0)	PB3CRD(2)
LK	CRCELL1(4)	CONTAU	CRCELL3(4)
LL	CRCELL1(5)	AU2XFER	CRCELL3(5)
LM	PB1CRD(3)	VPRCONT	PB3CRD(3)
LN	CRCELL1(6)	VPRCARD(7)	CRCELL3(6)
LO	CRCONT1	VPRCARD(6)	CRCONT3
LP	ML1(1)	VPRCARD(5)	ML1(0)
LQ	ROMCRD(10)	VPRCARD(4)	ROMCRD(14)
LR	ROMCRD(11)	VPRCARD(3)	ROMCRD(15)
LS	ROMCRD(5)	VPRCARD(2)	ROMCRD(7)
LT	CRROMRG(1)	VPRCARD(1)	CRROMRG(3)
LU	LOGCLK	VPRCARD(0)	
LV		TERMCARD	

Figure 1-10. Peripheral Processor Logic Card Locations (Sheet 2 of 2)

Table 1-3. Peripheral Processor Logic Card Functions

PP Component	Logic Cards	PP Component	Logic Cards
Virtual Processors	CRBASE(1-3) IRCARD(0-3) PCCARDA(0-7) VPRCARD(0-7) AU2XFER	Communications Register File	CRMIRLDR CRCONT(0-3) CRCELLY CRCELL0(0-6) CRCELL1(0-6) CRCELL2(0-6) CRCELL3(0-6) PB0CRD(0-3) PB1CRD(0-3) PB2CRD(0-3) PB3CRD(0-3) } Patch-boards
Arithmetic Unit	CONTAU PPAUCD(0-3)		
Indexer	INDEXER(0, 1)		
Read Only Memory	CRROMRG(0-3) ROMCRD(0-15) ROMMRG	Single Word Buffer Controller	SWBSYNC SWBASY
PP Control	PCCTL PPCTL1 PPCTL2 VPRCONT IRCARD(0-3)	Maintenance Logic	MLCTL ML1(0, 1) ML2 MIRMRGB

1-20 INTEGRATED CIRCUIT TYPES

The Peripheral Processor is implemented with the ECL logic set shown in figure 1-11. Refer to appendix E of the ASC System Manual for System No. 2 (Texas Instruments Incorporated part number 930005-1) for the Boolean equations and truth tables describing operation of the individual logic modules.

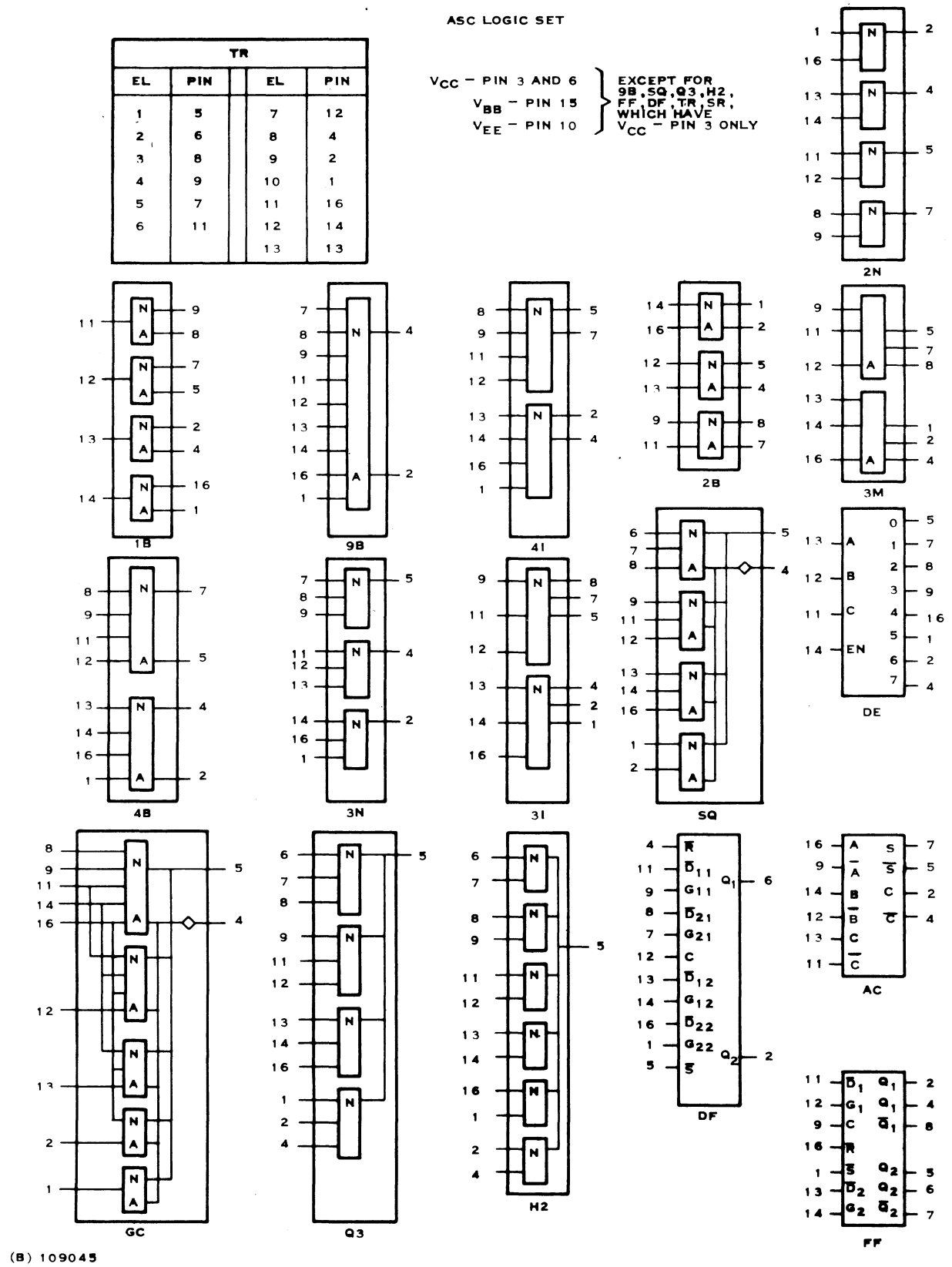


Figure 1-11. Peripheral Processor ECL Logic Set

SECTION II

INSTALLATION

2-1 GENERAL

Installation instructions for the Peripheral Processor are provided in the ASC System Installation Manual, part number 929980-1.

SECTION III

OPERATING INSTRUCTIONS

3-1 GENERAL

The two ECL columns that comprise the Peripheral Processor have no controls and indicators other than those associated with the ECL power supply regulators located on the lower front portion of both ECL columns. Refer to the ECL Regulators OMI, part number 930194-1, for a description of the regulator controls and indicators.

Control of Peripheral Processor operations is provided by the ASC Maintenance Console. The external maintenance system described in appendix C of this manual contains a brief introduction to the ASC Maintenance Console controls and indicators and how they affect the Peripheral Processor. For a more detailed discussion of the ASC Maintenance Console, refer to the ASC Maintenance Console OMI, part number 930009-1.

SECTION IV
PRINCIPLES OF OPERATION

4-1 GENERAL

This section provides both a general and detailed description of the Peripheral Processor (PP) theory of operation. Descriptions of the PP instruction repertoire, instruction processing, and instruction transfer table usage are also included in this section. The general description presented first is based on the PP detailed block diagram in figure 4-1, but is supplemented with additional diagrams covering the more involved components of the PP. Next, the instruction repertoire and instruction processing relate the hardware mentioned in the general description to the instruction set and the unique method of instruction development. The transfer table introduction provides a basic understanding of instruction execution and the capability to trace the data paths involved. A detailed description, accompanied by simplified block diagrams, timing diagrams, flow charts, logic diagrams, and transfer tables, is presented last. This section should be used with the PP logic card set provided in Section VII of this manual.

4-2 GENERAL DESCRIPTION

The Peripheral Processor (PP) is composed of the following eight major components:

- Virtual Processors
- Arithmetic Unit
- Indexer
- Communications Register File
- Read Only Memory
- Single Word Buffer Controller
- Control
- Maintenance Logic

The data path relationships between the PP components are shown in the PP detailed block diagram of figure 4-1. Distribution of the control necessary for PP operation is not shown in figure 4-1 (for simplicity); however, the control paths are discussed in detail in the detailed description of the PP control. The shaded blocks in figure 4-1 represent data buses that multiplex several inputs to a single output. An example of this is the Instruction

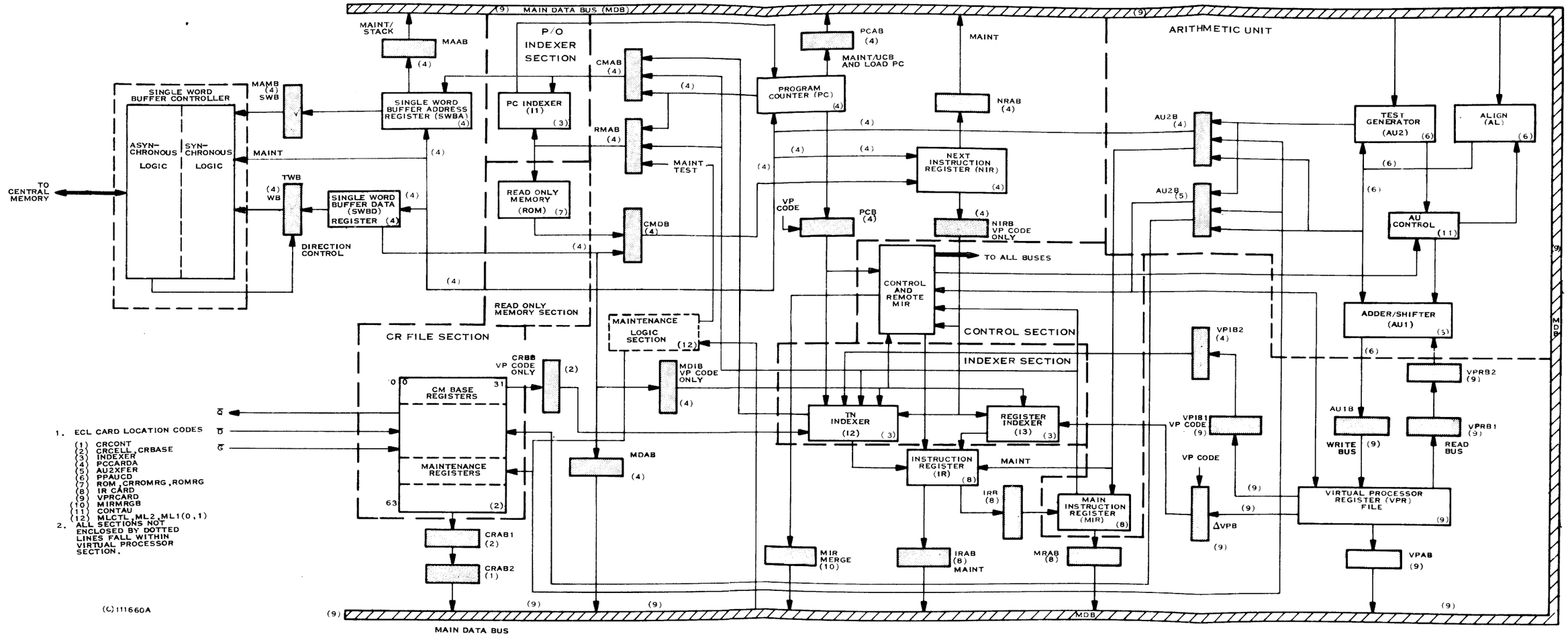


Figure 4-1. Peripheral Processor Detailed Block Diagram

Register Bus (IRB), used to enable the Instruction Register (IR) of the active Virtual Processor (VP) to the Main Instruction Register (MIR). Figure 4-1 shows only one input to IRB from one IR, when in reality there are eight inputs to IRB (one input for each of the eight IR's) with an associated control consisting of the active VP code (number of the active VP).

Figure 4-1 also contains hardware location information in the form of digits ranging from 1 to 12, in parenthesis next to each data bus and inside each functional block. These location digits are keyed to the ECL card location index on the left side of the detailed block diagram. The location of these cards in the PP ECL columns can be found by referring to the ECL card location map in Section I of this manual.

4-3 VIRTUAL PROCESSORS

The eight identical Virtual Processors (VP's) of the PP, designated VP0 through VP7, each consist of the following registers and their associated loading and distribution logic:

- Program Counter Register
- Next Instruction Register
- Instruction Register
- Virtual Processor Register File
- Central Memory Base Register
- Single Word Buffer Address Register
- Single Word Buffer Data Register

Each of the eight VP's shares the other seven components of the PP via the time sharing method described in the equipment overview in Section I of this manual. In figure 4-1, one block represents eight of a given type of register (there are seven types). Each of eight VP's uses one register from each block. The following paragraphs describe each VP register type.

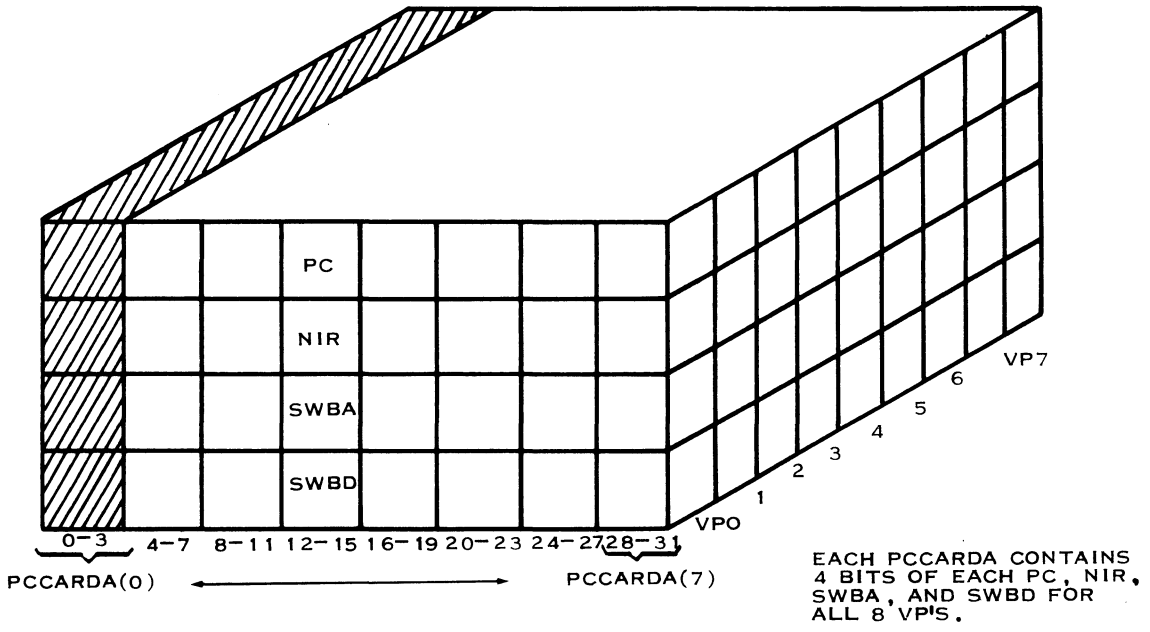
4-4 PROGRAM COUNTER REGISTER. The Program Counter (PC) is a 32-bit register used to address instructions in the Read Only Memory (ROM) or Central Memory (CM). Bit 0 of the PC specifies the memory source (CM or ROM) and bits 8 through 31 identify the instruction in the specified source. The eight PC's exist on the PCCARDA(0-7) cards, with four bits of each PC on every PCCARDA card as shown in figure 4-2.

The mode bit (bit 0) of the PC is supplied by PP control and the memory address is supplied by the PC indexer (part of the Indexer). The PC value is

then distributed to the PC indexer (for indexing to the next sequential instruction), the TN field indexer (for address development of PC relative instructions), the Single Word Buffer Address register (SWBA, for CM access), ROM, and the Main Data Bus (MDB, for PC saving instructions), all under direction of PP control.

4-5 NEXT INSTRUCTION REGISTER. The Next Instruction Register (NIR) is a 32-bit register used, primarily, to hold instructions retrieved from ROM prior to their transfer to the Instruction Register (IR). The NIR is also used for temporary storage of the Single Word Buffer Data register (SWBD) instruction when the SWBD is required to interface with CM during the execution of an instruction. The eight NIR's exist on the PCCARDA(0-7) cards, with four bits of each NIR on every PCCARDA card as shown in figure 4-2. The NIR accepts ROM or SWBD data, under direction of PP control, and distributes the loaded value to both the TN and R field indexers (for source, destination, and address development).

4-6 INSTRUCTION REGISTER. The Instruction Register (IR) is a 64-bit register used to hold the control information necessary to execute a step of



(A) 124728

Figure 4-2. PCCARDA(0-7) Registers

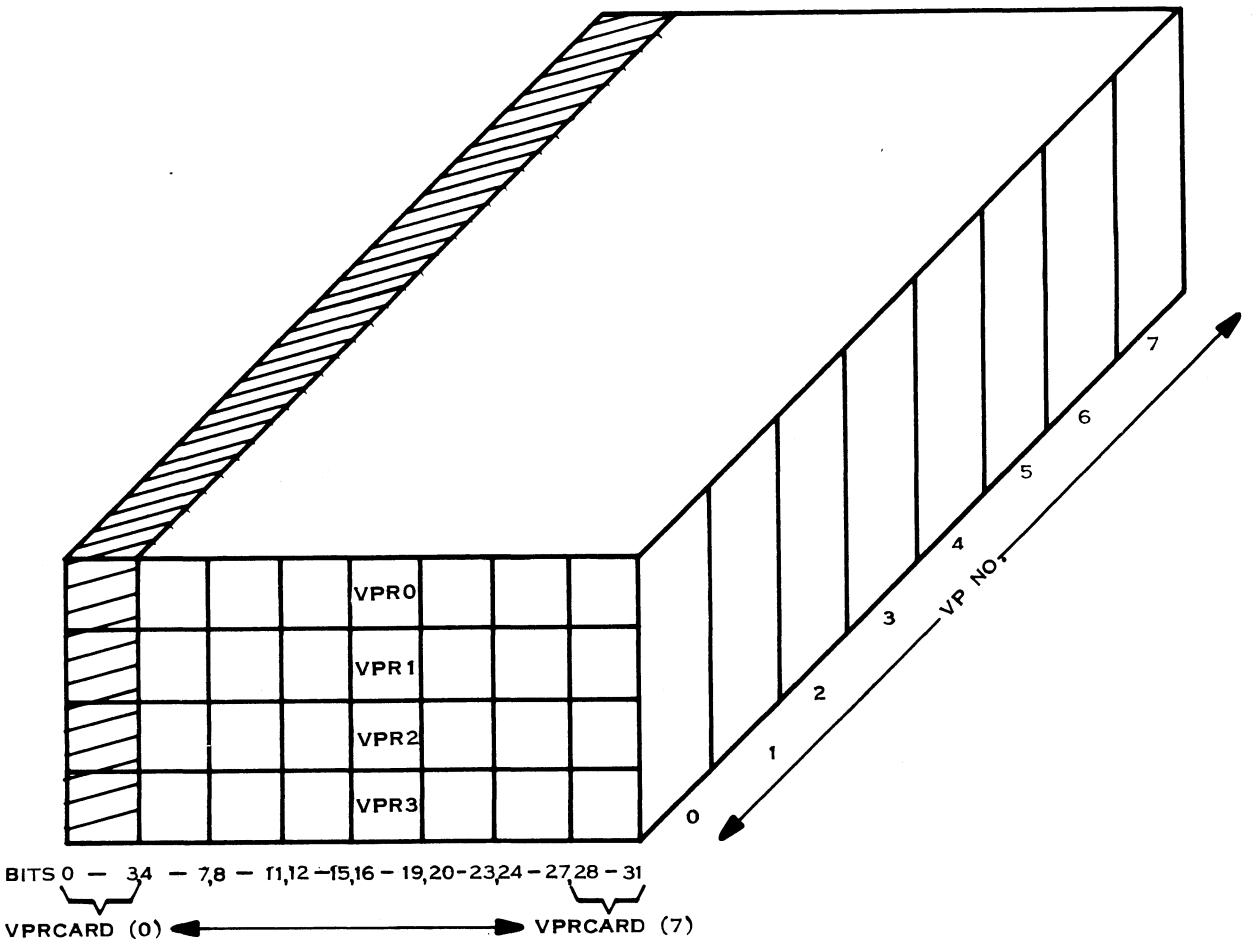
an instruction. The control information includes the op-code, state, source, destination, special control flags, and effective address. The op-code identifies the type of instruction (store, load, etc.); the state sequences the instruction through its various steps; the source (when used) identifies a Communications Register (CR) or Virtual Processor Register (VPR) used in a data transfer; the destination (when used) identifies a CR or VPR used in a data transfer; the special control flags provide miscellaneous control (next instruction source (SWBD or NIR) indicator, interrupt cycle initiation, etc.); and the effective address (when used) provides a CM or ROM address, immediate data, or possibly a shift count as mentioned in the Arithmetic Unit detailed description. The eight IR's exist on the IRCARD(0-3) cards, with 16 bits of each IR on every IRCARD (bits 0-15 on IRCARD(0), bits 16-31 on IRCARD(1), bits 32-47 on IRCARD(2), and bits 48-63 on IRCARD(3)).

The IR accepts op-code, state, and control flag data from PP control, source and destination data from the TN or R field indexers, and effective address data from the TN field indexer or the Arithmetic Unit (if a shift instruction is involved). The loaded IR data is then input to PP control to direct execution of the instruction.

4-7 VIRTUAL PROCESSOR REGISTER FILE. The Virtual Processor Register (VPR) file consists of four 32-bit registers, designated VPR0 through VPR3, used as general purpose accumulator registers. The eight sets of the VPR file are located on the VPRCARD(0-7) cards, with each card containing four bits of the VPR file for all eight VP's (refer to figure 4-3). The VPR file accepts data from the Arithmetic Unit (AU) and distributes data to the TN and R field indexers (for source, destination, and effective address development) and to the AU (for arithmetic operations and data transfers to other areas of the PP).

4-8 CENTRAL MEMORY BASE REGISTER. The Central Memory (CM) Base Register is a 24-bit register, located in the Communications Register (CR) file and used to hold a base value for base relative instructions. The CM Base Registers for VP0 through VP7 are located in bits 8 through 31 of the first eight CR's. A byte of each of the eight CM base Registers is located on each of the three CRBASE cards (bits 8-15 on CRBASE1, bits 16-23 on CRBASE2, and bits 24-31 on CRBASE3). The CM Base Register accepts data from the AU and distributes the CM base value to the TN field indexer (for address development of base relative instructions) and the Main Data Bus (MDB, for CR file read operations).

4-9 SINGLE WORD BUFFER ADDRESS REGISTER. The Single Word Buffer Address (SWBA) register is a 32-bit register used to hold the address necessary to access CM. The eight SWBA's are located on the PCCARDA(0-7) cards, with four bits of each SWBA on every PCCARDA card as shown in figure 4-2. The SWBA accepts data from the PC (normal



(A) 111682

Figure 4-3. VPRCARD(0-7) VPR File

instruction sequencing), the TN field indexer (for stack pointer modification during stack instructions (refer to the instruction repertoire of this section)), and the IR effective address (for CM stores, loads, branches, etc.). The data loaded in the SWBA is distributed to the Memory Control Unit (MCU, for CM access) and the MDB (for stack instructions).

4-10 SINGLE WORD BUFFER DATA REGISTER. The Single Word Buffer Data (SWBD) register is a 32-bit register that provides temporary storage for data being written to or read from CM. The eight SWBD's are located on the PCCARDA(0-7) cards, with four bits of each SWBD on every

PCCARDA card, as shown in figure 4-2. The SWBD accepts data from CM when a read is performed, and from the associated PC, SWBA, VPR file, or Communications Register (CR) file via the MDB and AU when a write is performed. The loaded SWBD data is distributed to CM (for CM write operations), the TN field indexer (for effective address development), the NIR (for temporary storage when the instruction source is CM and the SWBD is being used to do more than hold the next instruction), and the AU via the MDB (for loads, stores, logicals, etc., involving CM).

4-11 ARITHMETIC UNIT

The Arithmetic Unit (AU) is time shared by the eight Virtual Processors to perform addition, subtraction, logical functions (AND, OR, EXCLUSIVE OR, and EQUIVALENCE), data alignment, data shifting, data testing, and bit setting and resetting. The AU is capable of handling two 32-bit words at one time (for addition, subtraction, logical functions, or data comparisons) or a single 32-bit word and operating on it down to the bit level. The data handling portion of the AU is located on the PPAUCD(0-3) cards, with a byte of each logic function on each of the four PPAUCD cards (byte 0 is on PPAUCD(0), byte 1 is on PPAUCD(1), byte 2 is on PPAUCD(2), and byte 3 is on PPAUCD(3)). The complete control portion of the AU is located on the CONTAU card.

The AU accepts one operand from the VPR file and a second operand from the Main Data Bus (MDB). The MDB data may be supplied by a different VPR, a Communications Register (CR), the SWBD, or the Main Instruction Register (MIR) effective address when an immediate operand is involved. When the AU performs an addition, subtraction (by two's complement), logical function, poll operation (refer to paragraph 4-61), or shift, the output data is input to the VPR file over the AU1B bus. When a test is performed on one of the input operands to the AU, the AU responds with test result control signals to PP control. When the AU is used for alignment, setting or resetting of operand bits, or just data transfer, the AU output is transferred to its destination via the AU2B transfer bus. Functionally, the AU is divided into the following logic omponents:

- Aligner
- Complement or constant generator
- Unload box
- Double rail generator
- Adder
- Shifter
- Bit picker

- Test box 1, 2, and 3 logic
- Comparator
- Data manipulator
- Skip taken and branch taken logic
- AU control

The relationships between the listed AU components are shown in figure 4-4, and a brief description of each is given in the following paragraphs.

4-12 ALIGNER. The aligner is used to perform right-end-around shifts on the 32-bit MDB input to the AU (CR file, VPR file, SWBD, or MIR effective address data) in byte increments. This type of operation is necessary when halfword or byte level instructions involve data groups that are out of position in relation to the operation desired. An example of this problem is the addition of byte one of a VPR to byte two of a VPR. In this case, the CR (the MDB data) is shifted right one byte by the aligner. The aligner is capable of performing zero, one, two, or three byte shifts in this manner under direction of AU control. The aligner output is applied to the complement or constant generator, the shifter (for 16-bit cyclic shifts), the comparator, and the AU2B transfer bus. Data on the MDB is transferred to its desired destination via the AU2B bus, passing through the aligner with no shift.

4-13 COMPLEMENT OR CONSTANT GENERATOR. The complement or constant generator develops the true form of the 32-bit aligner output for addition and logical instructions, the complement form of the aligner output for subtraction instructions, plus one in both halfwords of the 32-bit output when an increment by one and test (IBZ or IBN) instruction is executing, or minus one in both halfwords of the output when a decrement by one and test (DBZ or DBN) instruction is executing. The output of the complement or constant generator is one of the 32-bit inputs to the adder.

4-14 UNLOAD BOX. The unload box distributes the true and complement form of the MDB data (from the CR file, VPR file, SWBD, or MIR effective address) to the shifter, bit picker, data manipulator, test box 1, and test box 2. In addition, the unload box accepts the active VP code and R field from AU control and generates the true and complement form of both for the data manipulator and test box 2.

4-15 DOUBLE RAIL GENERATOR. The double rail generator develops the true and complement form of the 32-bit operand input to the AU via the VPRB bus. The output of the double rail generator is distributed to the adder, comparator, and test box 3.

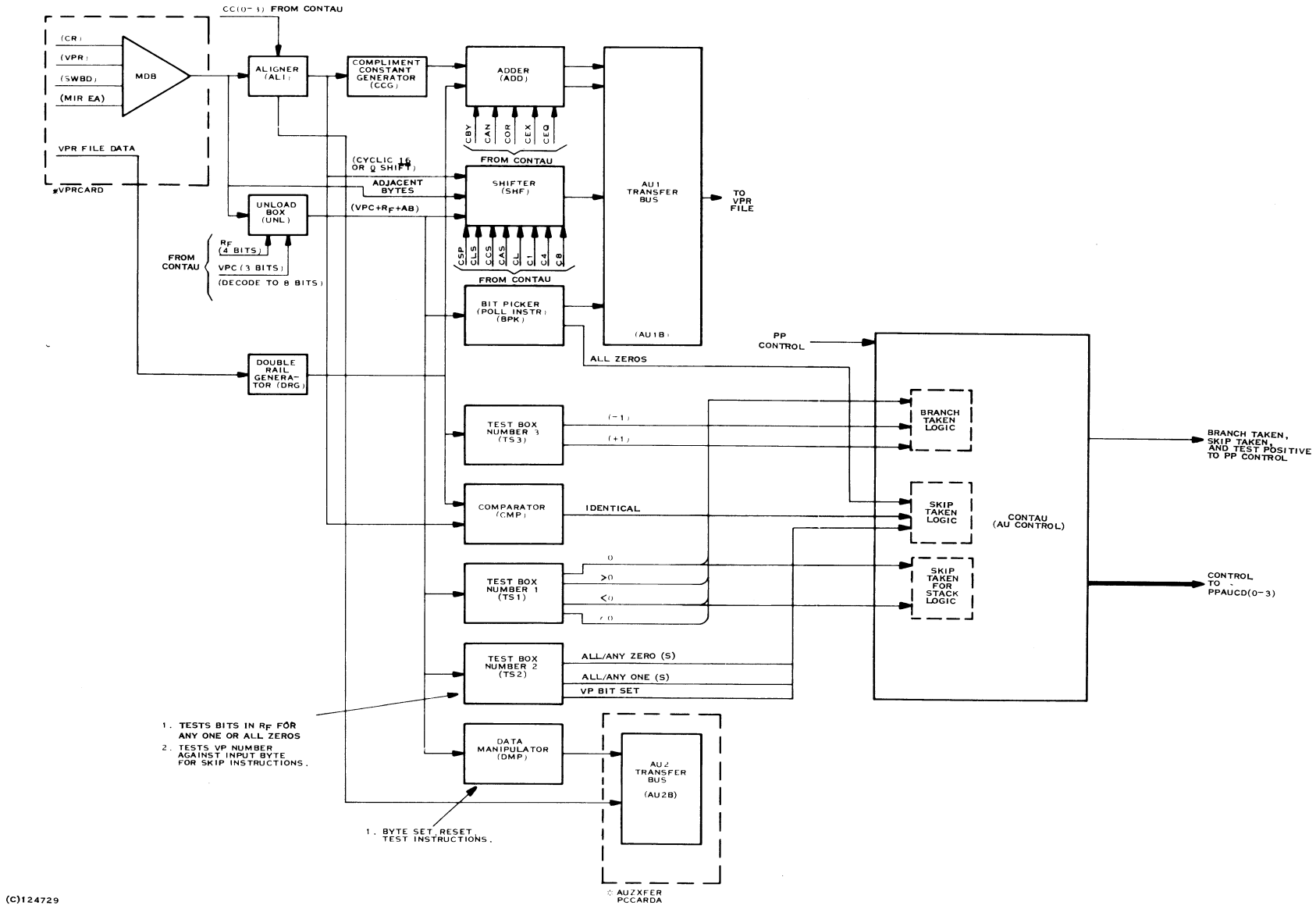


Figure 4-4. Arithmetic Unit Detailed Block Diagram

4-16 ADDER. The adder performs addition, subtraction (two's complement addition), and logical functions (AND, OR, EXCLUSIVE OR, and EQUIVALENCE) using the two 32-bit words supplied by the complement or constant generator and the double rail generator. The operation performed is under direction of AU control, and the resulting output is applied to the AU1B bus and, ultimately, the VPR file.

4-17 SHIFTER. The shifter performs right and left arithmetic, logical, and cyclic shifts in increments of one, four, and eight bits on the 32-bit MDB word supplied by the unload box. In addition, the aligner input to the shifter is passed through to the output when a 16-bit cyclic shift is executed. Refer to figure 4-5 for a definition of the three basic shift types used by the PP When the amount of shift is other than the increments mentioned, the shift is executed by first performing the largest possible incremental shift without exceeding the desired end amount and then following up with the necessary series of equal or smaller shift increments on the succeeding assigned time slots. An example would be a shift of 13, which is executed in increments of 8, 4, and 1, using three time slots. The result of the desired shift operation, which is under direction of AU control, is output to the VPR file over the AU1B bus.

4-18 BIT PICKER. The bit picker is used during POLL instructions to scan a byte of data (from a CR) from the most-significant bit to the least-significant bit, in order to determine the number of zeros from the most significant bit to the first one. The bit picker operates on all four bytes of the MDB word supplied by the unload box and PP control selects the count of the desired byte for transfer to the VPR file over the AU1B bus. The bit picker also develops an all-zero signal (indicates when all bits of the byte under examination are zero) used by the AU control skip taken logic.

4-19 TEST BOX 1, 2, AND 3 LOGIC. The test box logic performs various tests on the 32-bit words from the unload box and the double rail generator. The portion (byte, hex, or bit) of the word under test specified by the instruction requiring the test is selected by the AU control skip and branch-taken logic. The test box 1 logic utilizes the MDB word supplied by the unload box during test positive (TP), test negative (TM), test zero (TZ), and test nonzero (TN) type instructions to test each of the four input bytes for positive, negative, zero, and nonzero data, respectively. The test box 1 logic is also used during stack (PUSH, PULL, and MOD) instructions to test for negative and zero data. The test box 2 logic utilizes the word, R field, and VP code supplied by the unload box and is used during test-for-any-one (TO, TSO, and TRO), test-for-any-zero (TZ, TSZ, and TRZ), test-for-all-ones (TAO), and test-for-all-zeros (TAZ) type instructions. The test box 2 logic is used to test all eight input hex groups for any one, any zero, all ones, and all zeros in only those bit positions marked by ones in the R field. In addition, the test box 2 logic is used during the test VP flag for one (VPTO)

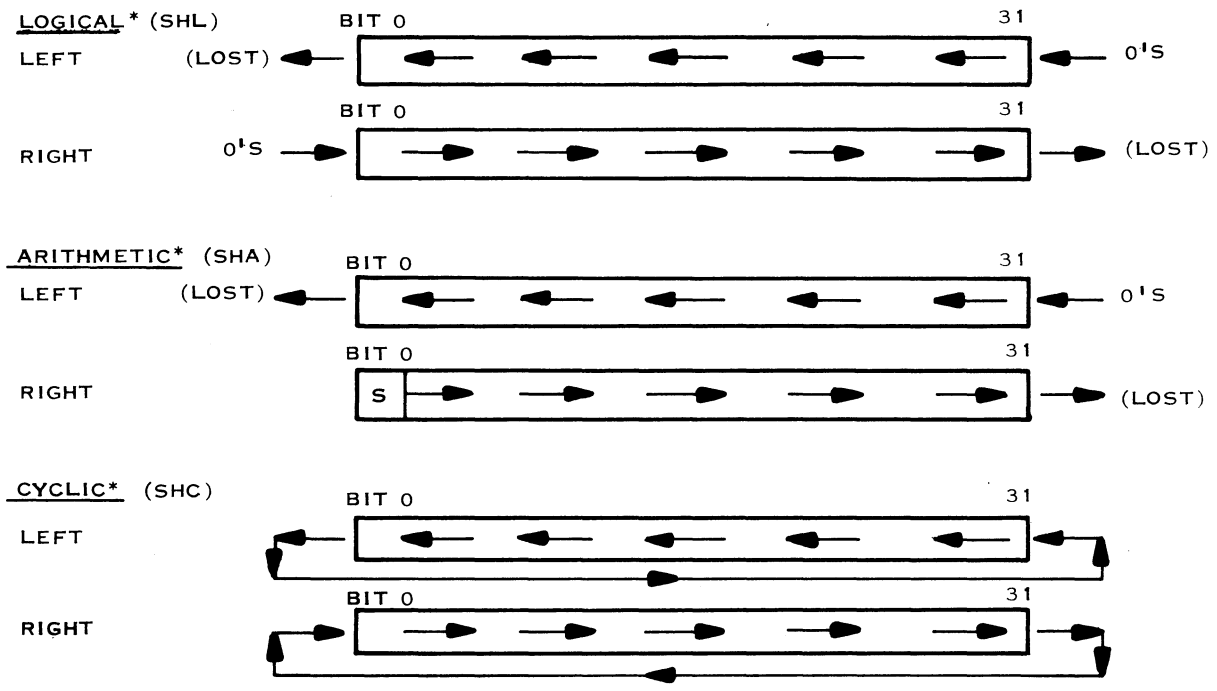


Figure 4-5. Peripheral Processor Shift Basics

and test VP flag for zero (VPTZ) instructions to test each of the four input bytes for a set bit and cleared bit, respectively, in the position designated by the VP code. The test box 3 logic utilizes the VPRB bus word from the double rail generator during the decrement and branch if zero/nonzero (DBZ and DBN) and increment and branch if zero/nonzero (IBZ and IBN) instructions to test each of the four input bytes for plus and minus one, respectively. This test box 3 byte information is then used to determine when a VPR halfword is zero or nonzero. The test box logic outputs are input to the AU control skip taken for stack logic, branch taken logic, and skip taken logic for development of the test positive, branch taken, and skip taken control signals.

4-20 COMPARATOR. The comparator performs a bit-for-bit comparison of the two 32-bit words input to the AU and generates a signal for each byte indicating whether or not the two inputs are identical. The results of the comparison are input to the AU control skip taken logic, where the data group (word, halfword, or byte) specified by the instruction requiring the compare is selected.

4-21 DATA MANIPULATOR. The data manipulator sets or resets bits in the MDB word supplied by the unload box in accordance with the R field or VP code, depending on the executing instruction. When a set (SL, SR), reset (RL, RR), test and set (TSOL, TSOR, TSZL, and TSZR), or test and reset (TROL, TROR, TRZL, and TRZR) type instruction is executing, the data manipulator sets or resets (depending on the op-code) bits in all eight input hex groups marked by ones in the R field. When a set VP flag (VPS) or reset VP flag (VPR) instruction is executing, the data manipulator sets or resets, respectively, one bit in each of the input bytes according to the VP code. The output of the data manipulator is input to the CR file over the AU2B bus, where the hex (for TS and TR type instructions) or byte (for VPS and VPR instructions) specified by the instruction requiring the setting or resetting is selected.

4-22 SKIP TAKEN AND BRANCH TAKEN LOGIC. The skip taken and branch taken logic, located on the CONTAU card, consists of skip taken for stack logic, branch taken logic, and skip taken logic. The skip taken for stack logic utilizes the zero and negative test signals from the test box 1 logic to determine if the instruction following the current stack instruction is to be skipped (normal operation of a stack instruction) or not (invalid stack parameter encountered). The output of the skip taken for stack logic is used by PP control to direct the stack instruction skipping action. The branch taken logic uses the test box 1 and test box 3 outputs during the test and branch (TP, TM, TN, and TZ) and increment/decrement test and branch (IBZ, IBN, DBZ, and DBN) type instructions, respectively, to determine if the specified branch should be taken. The output of the branch taken logic is used by PP control to direct the branching action. The skip taken logic determines whether the next instruction should be skipped. To do this, it uses the comparator outputs for compare (CE and CN) type instructions, the test box 2 outputs for test and skip (TO, TA, TR, TS, TZL, TZR, and VPT) type instructions, and the all-zero outputs from the bit picker for the POLL instruction. The output of the skip taken logic is used by PP control to direct the skipping action.

4-23 AU CONTROL. The AU control logic, located on CONTAU, accepts PP control information in the AUMIR format, shown in figure 4-78 of the PP control detailed description, translates the AUMIR data to a more usable form, and distributes the results to the AU components described in the

previous paragraphs. The aligner control logic (part of AU control) utilizes the aligner reference and object portions of the AUMIR format to develop the byte increment shift controls necessary for aligner operation. The aligner reference and object are also used by the aligner control logic to develop the byte and halfword signals used by the skip and branch taken logic to determine what data is specified by the current instruction for testing purposes. The shifter control logic uses the shift count and op-code portions of the AUMIR format to develop the shifter controls, the control signal for the aligner that results in a two byte shift when a cyclic shift of 16 bits is desired, and the updated shift count for PP control when additional shift is necessary to complete a multistep shift instruction. The remainder of the AU control logic is primarily concerned with decoding the instruction op-code and state in order to enable the proper adder, complement or constant generator, data manipulator, and skip and branch taken logic.

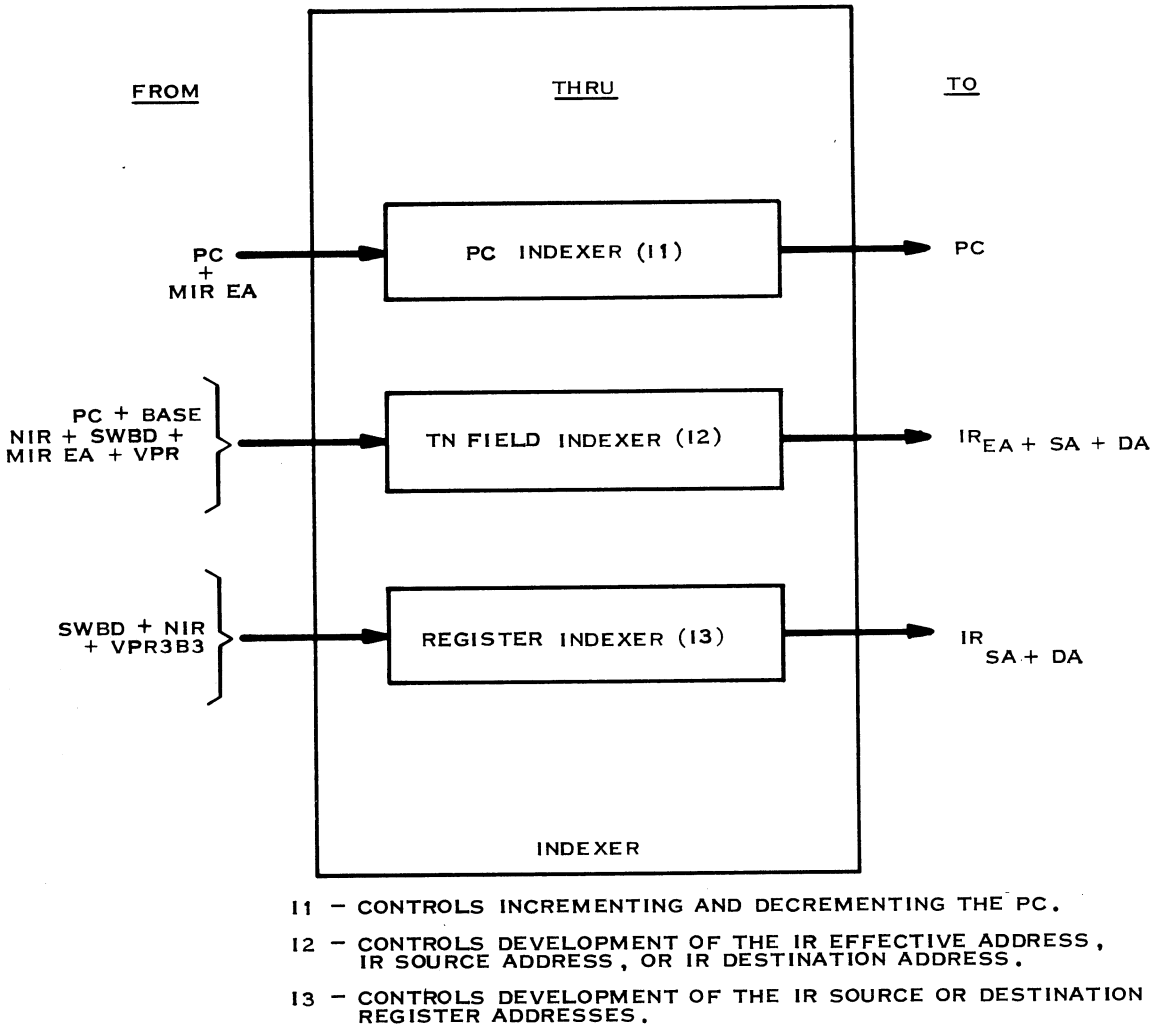
4-24 INDEXER

The Indexer is time shared by the eight Virtual Processors to perform Program Counter (PC) indexing and to develop the Instruction Register (IR) effective address, source address, and destination address. The Indexer is also involved in miscellaneous operations such as Write Cycle Equality checking (WCE occurs when the next instruction to be executed is modified by the current instruction) and stack parameter modification. The Indexer is functionally divided into the following components (they are described in the next three paragraphs):

- PC indexer (I1)
- TN field indexer (I2)
- Register indexer (I3)

The Indexer is located on the INDEXER(0, 1) cards, with 16-bits of each functional component on both INDEXER cards (bits 0-15 are on INDEXER(0) and bits 16-31 are on INDEXER(1)). The inputs and outputs of the three Indexer components are shown in figure 4-6.

4-25 PC INDEXER. The PC indexer is primarily used to increment the current PC value by one to advance the third level of the PP three-level pipe. (The PP three-level pipe concept is described in detail in paragraph 4-62.) The incremented result is returned to the PC to locate the next sequential instruction. The PC indexer decrements the current PC value by one when the address of the next instruction needs to be saved (store and load PC instructions). When an interrupt occurs, the PC indexer decrements the current PC value by two so that the instruction following the interrupted instruction is not skipped after the interrupt is honored. If a branch is taken out of the current instruction stream, the branch address in the Main Instruction Register (MIR) effective address, rather than the current PC value, is used for indexing.



(A) 111650

Figure 4-6. Indexer Block Diagram

4-26 TN FIELD INDEXER. The TN field indexer develops the IR effective address when the T and N fields of the instruction being indexed specify a CM or ROM location or an immediate operand. When the T and N fields specify a register (VPR or CR), the TN field indexer develops a source or destination address for the IR. The TN field indexer is capable of adding up to three different items in the development of a CM or ROM address and up to two different items in the development of an immediate operand, register operand, or absolute operand. The items that may be involved in the development of a memory location include the PC value or CM base value (for CM addresses only) for PC and base relative instructions, respectively, the N field of the next instruction (from the SWBD or NIR), and the VPR halfword specified by the T field of the next instruction. In some cases (WCE test,

locating a stack pointer, and incrementing the branch address during an unconditional branch to ROM and save PC (BRSM) instruction), the TN field indexer develops a memory address by incrementing the MIR effective address. The two items that may be involved in the development of an immediate operand, absolute operand, or register operand include the N field of the next instruction and the VPR halfword specified by the T field of the next instruction.

The TN field indexer is also capable of developing an IR effective address from an indirect cell. The two items that may be involved in the indirect case include the 24-bit address field in the SWBD and the VPR halfword specified by the T field of the indirect cell. A miscellaneous operation provided by the TN field indexer is the modification of the stack word and space count parameters (refer to paragraph 4-46) input in the SWBD. The output of the TN field indexer is inserted in the IR effective address, source address, or destination address field under direction of PP control.

4-27 REGISTER INDEXER. The register indexer develops the IR source or destination address specifying a VPR or CR. When a VPR is the source or destination, the R field of the instruction being indexed (from the SWBD or NIR) is passed through the register indexer to the IR. When a CR is the source or destination, the register indexer adds byte 3 of VPR3 to the R field to develop the desired address. The addition is necessary because the four-bit R field is not large enough to specify each individual byte of the 64 register CR file. The output of the register indexer is inserted in the IR source or destination address field under direction of PP control.

4-28 COMMUNICATIONS REGISTER FILE

The Communications Register (CR) file consists of 64 32-bit registers, accessible to all eight VP's and addressable down to the individual bit level. Primary functions of the CR file include the following:

- PP to ASC system interface
- PP maintenance control
- System control
- Interrupt monitoring and control
- VP time slot assignments
- VP CM priority assignments
- Source of real time clock information
- Source of CM base operands
- Temporary storage area

The CR file is implemented on four motherboards (CR0MB, CR1MB, CR2MB, and CR3MB), each motherboard containing a byte of all 64 registers. Refer to figure 4-63 for the card layout in relation to the motherboards. The CR file is functionally divided into the following components and described in the next three paragraphs:

- CR file control
- Input synchronizers
- Communications Registers

The relationships between these components is shown in figure 4-7.

4-29 CR FILE CONTROL. The CR file control logic is located on the CRMIRLDR, CRCONT(0-3), and CRCELLY cards. The CRMIRLDR card accepts op-code groupings and the IR source and destination addresses from PP control, and generates write controls capable of addressing from the word to the hex level and read controls capable of addressing a word. The CRCONT(0-3) cards utilize the CRMIRLDR output to distribute the write and read select enables to the intended CR. The write select provides the PP software with the capability of writing in any word (down to the individual bit)

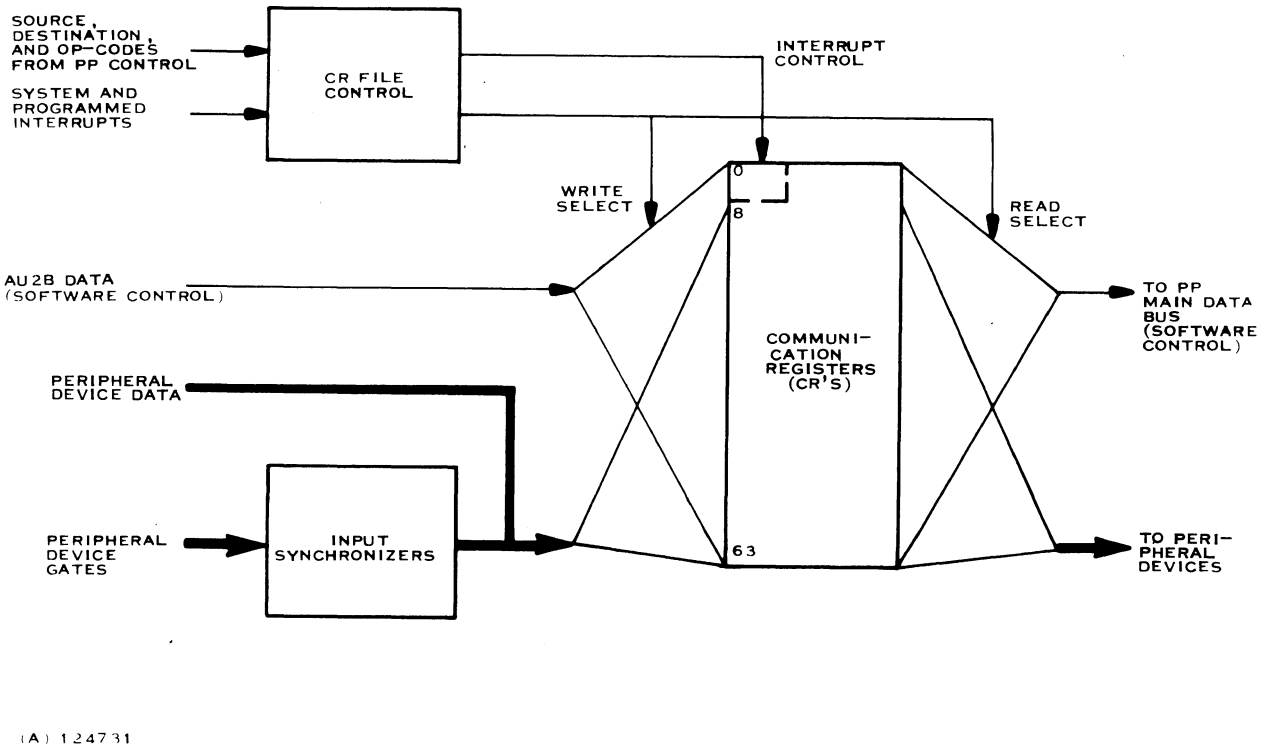


Figure 4-7. Communications Register File Block Diagram

of the CR file, with the approval of the CR protect mechanism (the CR protect mechanism inhibits writing into the first 10_{16} words of the CR file when enabled). The read select provides the PP software with the capability of reading any word from the CR file. The CRCELLY card monitors system interrupts (ac power failure, Central Processor interrupt, activation of the ASC Operator's Console STOP button, disc protect violation, CM parity error, CM protect violation, or illegal op-code in the selected VP (the selected VP is the master controller VP used to direct PP operations)) and the software initiated (programmed) interrupts to control the interrupt associated bytes of the CR file.

4-30 INPUT SYNCHRONIZERS. The CR file input synchronizers are used to synchronize gating signals from peripheral devices with the PP clock so that data from the peripheral devices is not lost at the interface with the CR file. All of the synchronizers are identical, except for the number of outputs, and are located on the CRCELLY card and each CRCONT and CRCELL card. The single output synchronizers are used with the gating signals associated with data on one motherboard and the multiple output synchronizers are used when one gate is associated with data on more than one motherboard. The peripheral device data may access all CR's except the first eight (interrupt control bytes and the CM base registers) and is continually monitored by the device associated with the data.

4-31 COMMUNICATIONS REGISTERS. The Communications Registers (CR's) provide the actual storage capability (64 32-bit registers) for the CR file data and control information. All 64 CR's are subject to modification via the PP software (under the influence of the CR protect mechanism), whereas the peripheral device data has access to the CR bits as detailed in the PP Fixed Variable List. Refer to appendix B of this manual for the CR file map and to the Description of the ASC CR File, part number 930207-1, for a detailed description of all data in the CR file.

4-32 READ ONLY MEMORY

The PP Read Only Memory (ROM) provides storage for up to 4096 32-bit words, used primarily to hold control programs for the input/output devices of the ASC system. The actual ROM is located on the 16 ROMCRD(0-15) cards (256 32-bit words per card) and the decoding and merging logic is on the AU2XFER, CRROMRG(0-3), and ROMMRG cards. Refer to figure 4-8 for a block diagram illustrating the relationships between the logic cards.

The twelve bit ROM address supplied by the PC is applied directly to the ROMCRD(0-15) cards. The four most significant bits of the address are used to enable one of the 16 ROMCRD cards and the eight least significant bits of the address select one of the 256 words on the enabled card. The addressed word is passed through the merging logic to the NIR. (The merging logic consists of the ROMMRG and AU2XFER cards when the addressed word

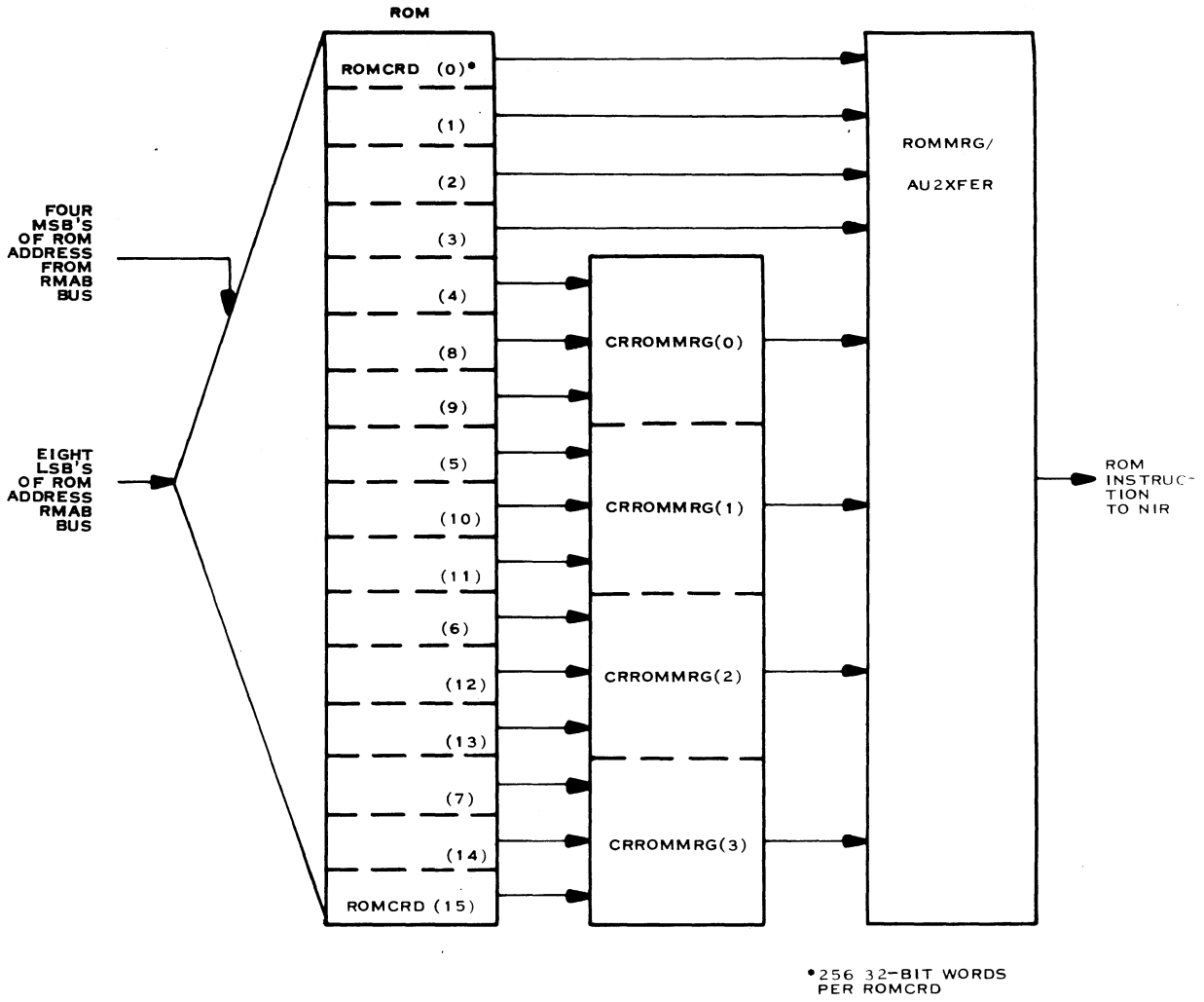


Figure 4-8. Read Only Memory Block Diagram

is on ROMCRD(0-3) and a CRROMRG card and the ROMMRG and AU2XFER cards when the addressed word is on ROMCRD(4-15).

4-33 SINGLE WORD BUFFER CONTROLLER. The Single Word Buffer Controller (SWBC) is time shared by the eight Virtual Processors to provide an interface to the Memory Control Unit (MCU) and Central Memory (CM) necessary for CM read and write operations. The SWBC accepts memory access requests from the active VP, notifies the MCU when a request is present, and provides a data path to CM to execute the highest priority request under direction of the MCU. Functionally, the SWBC is divided into the following components:

- Synchronous logic
- Asynchronous logic
- Two Way Bus

The synchronous logic is located on the SWBSYNC card, the asynchronous logic is located on the SWBASY card, and the Two Way Bus (TWB) is located on the PCCARDA(0-7) cards, with four bits of the 32-bit TWB on each PCCARDA. The relationships between the components of the SWBC are shown in figure 4-9.

4-34 SYNCHRONOUS LOGIC. The synchronous logic portion of the SWBC provides the interface between the eight VP's and the asynchronous logic portion of the SWBC. When a CM read or write operation is desired by the active VP, the memory request and VP priority (assigned by software in the CR file) combine to enable the VP code into either the high priority or low priority queue on SWBSYNC. At the same time, the memory request enables status information describing the request into the assigned status file entry. The status information includes the memory zone (three least significant bits of the PC), the type of operation (read or write), and the CM protect enable. Each status file entry has a busy bit associated with the assigned VP so that no more than one outstanding memory request can exist for any one VP. The priority queue and status file data is input to the asynchronous logic so that any outstanding memory access requests can be honored.

4-35 ASYNCHRONOUS LOGIC. The SWBC asynchronous logic is the PP interface with the MCU. The asynchronous logic monitors the synchronous logic priority queue and status file data, indicates to the MCU when a PP memory access request is pending, and controls the necessary transfer of data and address information in order to execute the desired read or write operation.

The priority queue select logic on SWBASY monitors entries in both synchronous logic priority queues and selects the entry with the highest priority. Queue entries in the same priority queue are selected on a first come, first served basis. When a queue entry does exist, the VP code associated with the selected entry is enabled to the VP code distribution logic and is used to enable the associated status file entry in the synchronous logic to the asynchronous status file output logic. At the same time, the memory request control logic issues an access request (AR) signal to the MCU to initiate the memory cycle. When the MCU responds with the request accepted (RA) signal, the following takes place: The VP code associated with the request and the status file data are both input to the MCU, and the write control (other than the write enable from the MCU) from the VP code distribution logic is input to the TWB. If a write operation is specified by the status file entry, the MCU develops a write enable that combines with the other mentioned

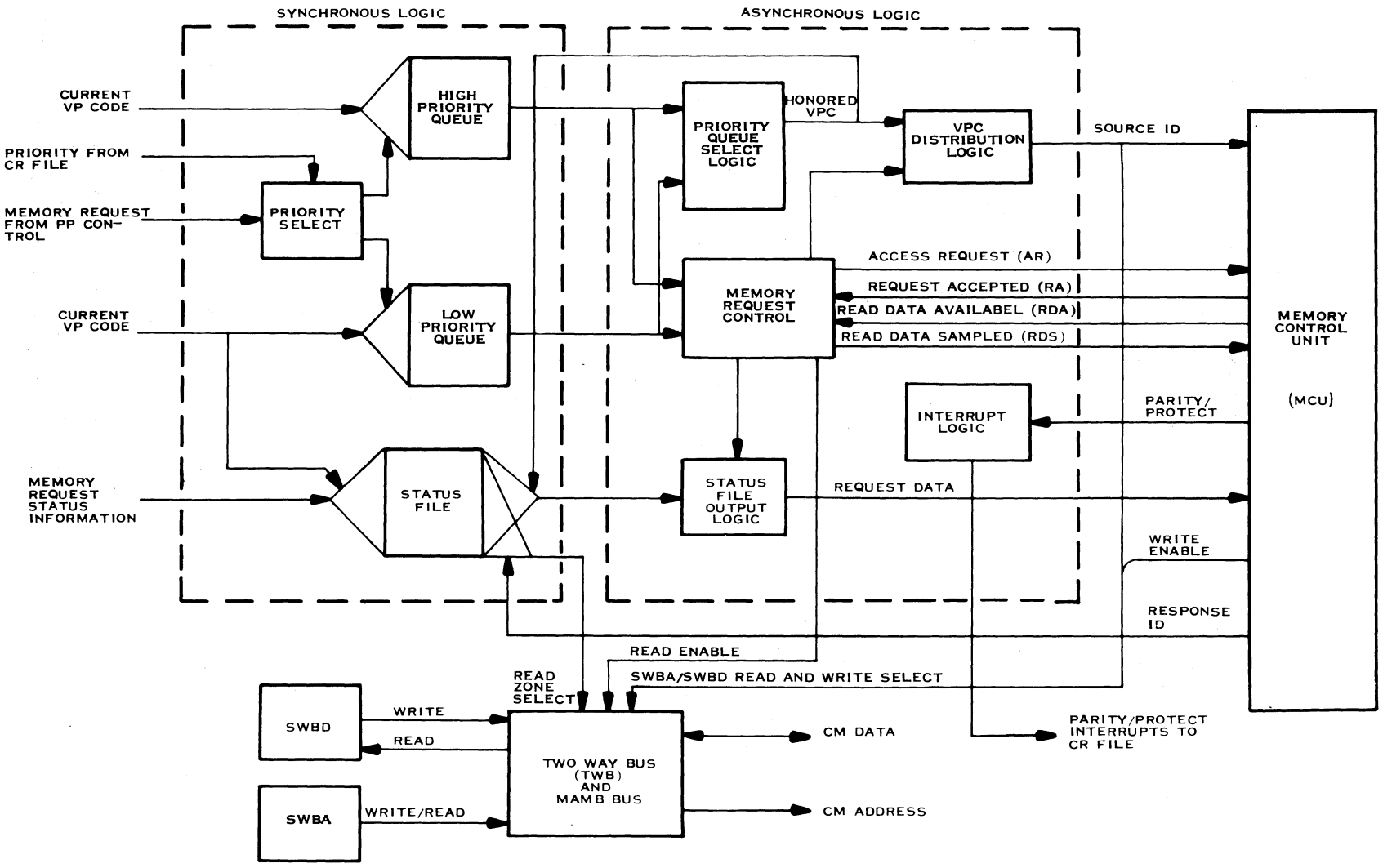


Figure 4-9. Single Word Buffer Controller Block Diagram

write control signals to execute the write. If a read operation is specified by the status file entry, the address of the data to be read is input to CM and the VP code input to the MCU is used to select the zone data in the corresponding status file entry for use by the TWB. When the MCU responds to the read request with the read data available (RDA) signal, the memory request control logic acknowledges the response with the read data sampled (RDS) signal and enables the CM read by providing the necessary controls to the TWB. If a CM parity error or protect violation occurs during the memory cycle, the MCU response reflecting the error is transferred to the CR file interrupt logic via the asynchronous logic.

4-36 TWO-WAY BUS. The Two-Way Bus (TWB) and MAMB bus interface the SWBD and SWBA, respectively, with CM. When a write operation is to be performed, the write control signals from the asynchronous logic enable the SWBD over the 32-bit bidirectional TWB to the CM data lines and the SWBA over the MAMB bus to the CM address lines. During these transfers, the SWBD data is expanded eight times to match the CM eight word data port. When a read operation is to be performed, the SWBA is transferred over the MAMB bus to the CM address lines. When the MCU and CM respond to the read request, the TWB accepts data from CM (the TWB is normally in the read mode). In addition, the TWB zone select from the synchronous logic enables one of the eight words read to the SWBD after the read memory cycle.

4-37 PERIPHERAL PROCESSOR CONTROL

The Peripheral Processor (PP) control is time shared by the eight Virtual Processors (VP's) to provide the controls and enables necessary to execute PP instructions loaded in the Instruction Registers (IR's). The heart of PP control, the Main Instruction Register (MIR), is used to hold the expanded IR of the active VP to control PP operations for the duration of a time slot. The MIR consists of 256 bits. Words 0 and 1, the IRMIR, are located on IRCARD(0-3); words 2, 5, and 6, the CRMIR, are located on CRCONT(0-3); word 3, the VPRMIR, is located on VPRCONT and PCCTL; word 4, the AUMIR is located on CONTAU; word 7 is the VP codes used by IRCARD(0-3), and is located on IRCARD(0-3). The remaining PP control logic is located on the PCCTL, PPCTL1, PPCTL2, and VPRCONT cards. The control logic on the CONTAU card of the AU and the CRMIRLDR and CRCONT(0-3) cards of the CR file is discussed in the general description of the AU and CR file, respectively.

The major control paths in the PP are shown in figure 4-10, and should be referenced in the following discussion. The VP code associated with the next active VP is used to transfer the IR of the next active VP to the 64-bit IRMIR and to the control logic on VPRCONT associated with the development of the remaining portions of the MIR.

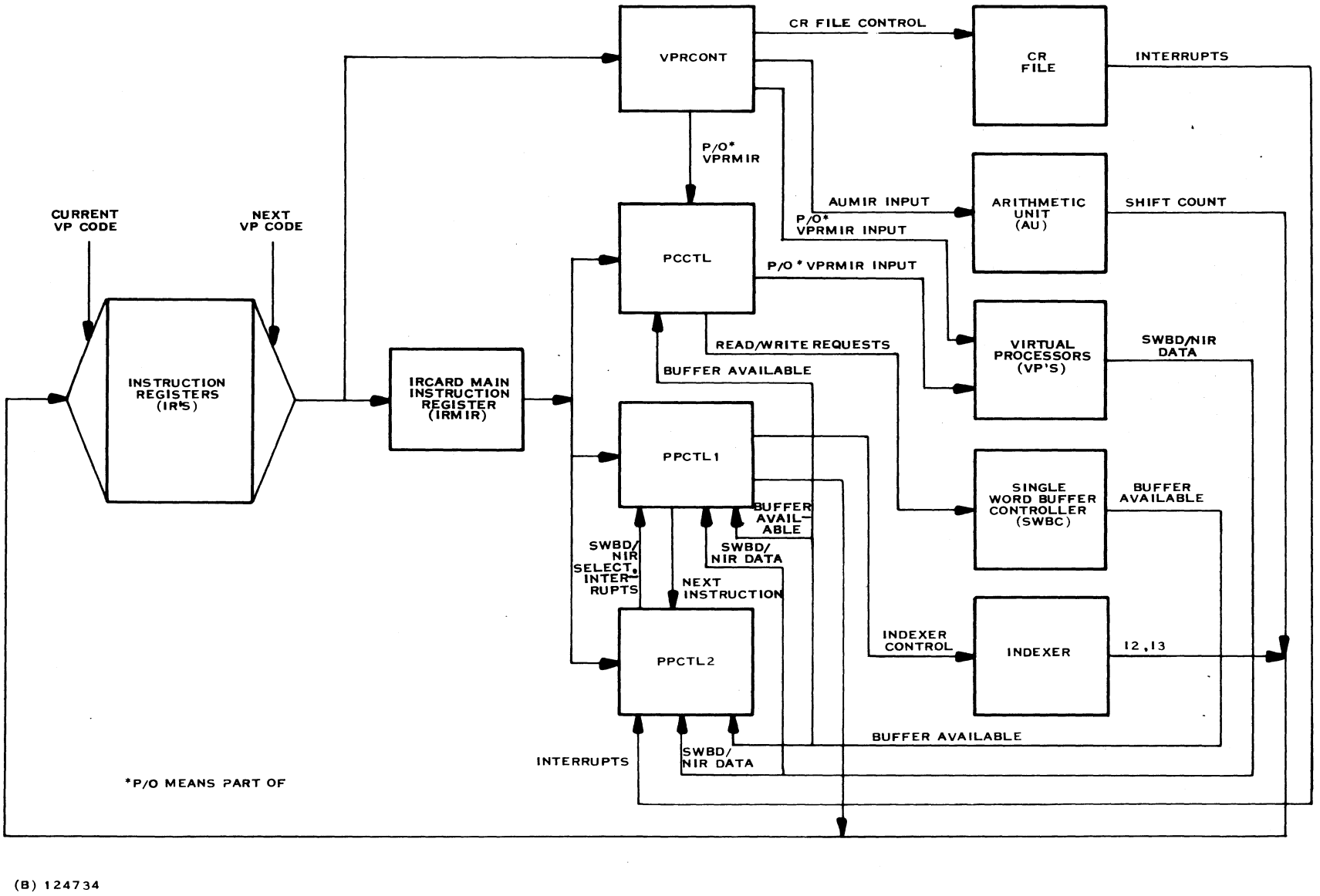


Figure 4-10. Peripheral Processor Control Block Diagram

The VPRCONT card utilizes the applied IR data to develop the CR file control signals necessary for the CRMIRLDR card (referenced in paragraph 4-29) to generate the CRMIR input for the CR file, the AUMIR input for the AU, and the VPRMIR input (with the aid of the PCCTL card) for the active VP. Refer to figure 4-78 for the CRMIR, AUMIR, and VPRMIR input formats. When the next clock occurs and the described VP becomes active, the IR is loaded in the IRMIR, and the CRMIR, AUMIR, and VPRMIR are expanded to the formats shown in figure 4-79. The MIR (IRMIR, CRMIR, AUMIR, and VPRMIR) is now used to control the operations necessary to execute a step of the current instruction. The PCCTL card uses the IRMIR data and the buffer available signal from the SWBC (indicates that the active VP has no memory requests pending) to develop additional controls for the active VP and read and write requests for SWBC. A new read or write request can only be made when the SWBC is available.

The PPCTL1 card uses the IRMIR data, SWBD or NIR data (as determined by the select signal from the PPCTL2 card) from the active VP, and the SWBC buffer available signal to develop indexer controls, update the IR state at the conclusion of the time slot, and generate a signal used to indicate when a new instruction should be transferred to the IR from the SWBD or NIR (termination of the current instruction). The PPCTL2 card uses the IRMIR data, SWBD or NIR data from the active VP, the SWBC buffer available signal, and the next instruction indicator from PPCTL to update the IR op-code and control flags at the conclusion of the time slot. The IR also receives update information from the TN (I2) and R (I3) field indexers (source, destination, or effective address) and the AU (effective address) when a shift instruction is being executed. Automatic and programmed interrupts recorded by the CR file are processed by the PPCTL2 card at the conclusion of the instruction during which the interrupt occurred and by the PPCTL1 card when the interrupt is honored by branching to ROM.

4-38 MAINTENANCE LOGIC

The Peripheral Processor (PP) maintenance logic provides a means of checking the operation of the PP ECL circuitry previously discussed in this section. In addition, during normal operation of the PP, the maintenance logic supplies the VP code of the active VP, the Single Word Buffer (SWB) priority of the active VP, and a Communications Register (CR) indicator used to enable or disable the CR protect logic provided by PP control. The checkout capability provided by the maintenance logic is exercised by the ASC Maintenance Console in the manual mode, a card reader in the semiautomatic mode, or by a VP in the automatic or normal mode. The maintenance system external to the PP (this includes the ASC Maintenance Console, the card reader, and Test Control Logic) necessary for selecting one

of the three operating modes and controlling both the manual and semi-automatic modes is briefly described in appendix C of this manual. In the automatic mode, any VP is capable of placing any other VP (except the VP designated by the VP SELECT switch on the ASC Maintenance Console) under test to execute maintenance commands supplied by the controlling VP.

A block diagram showing the interrelationships between the four basic areas of the maintenance logic is presented in figure 4-11. The "heart" of the maintenance logic is in the CR file; specifically, the maintenance registers beginning at bit 17 of word C_{16} and extending through word F_{16}. The data entered in these registers, either via the ASC Maintenance Console, the card reader, or the PP software controlling the VP responsible for exercising the maintenance logic, is used to control the PP maintenance logic during the execution of maintenance commands. When a maintenance command is initiated (by the ASC Maintenance Console in the manual or semi-automatic modes or by the occurrence of a time slot associated with the VP under test in the automatic mode), the control information and data from the maintenance registers are distributed to the hardcore maintenance logic on the

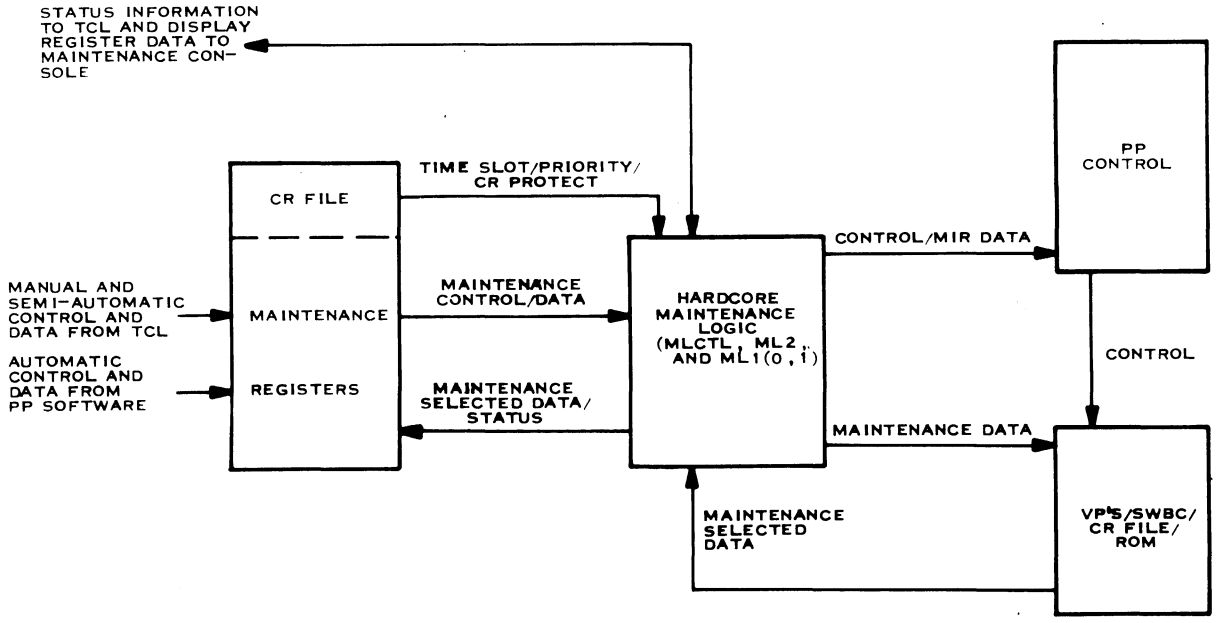


Figure 4-11. Peripheral Processor Maintenance Logic Block Diagram

MLCTL, ML2, and ML1(0, 1) cards. The hardcore maintenance logic uses the maintenance control and data to direct operation of one of the following types of maintenance commands:

- Switch register to display register transfer (these are the two data-holding registers of the maintenance registers)
- PP register (includes any of the PC's, NIR's, IR's, SWBA's, SWBD's, VPR's, or CR's, the MIR, or an entry in the SWBC) to display register transfer
- Switch register specified CM word to display register transfer
- Switch register specified ROM word to display register transfer
- Switch register to PP register transfer
- Display register to PP register transfer
- Storage of the display register in CM at the location specified by the switch register
- Lock or unlock the PC's specified by the maintenance registers
- Set or reset the flip-flops associated with the VP's specified by the maintenance registers
- Advance all (or one) of the VP's specified by the maintenance registers by the number of time slots specified by the maintenance registers (this is called a burst operation)
- Advance all (or one) of the VP's specified by the maintenance registers until they have all completed their current instruction

The hardcore maintenance logic distributes control gates and enables (and MIR data, when the current maintenance command transfers data to the MIR) to PP control, and data to the eight VP's, the SWBC, the CR file, and the ROM (ROM is supplied data only during the ROM to display register maintenance command previously mentioned). When the maintenance command involves a data transfer, PP control provides the VP's, SWBC, CR file, or ROM with the control enables necessary to accept maintenance register data or to distribute currently held quantities to the display maintenance register. The maintenance commands not involving a maintenance register transfer are also under direction of PP control, but execute without any other interface to the maintenance logic. By executing the proper combination of maintenance commands, any area of the PP can be checked out. The hardcore maintenance logic also reports PP status information (including the current VP code and time slot) to the Test Control Logic (TCL), the display register contents directly to the ASC Maintenance Console, and

maintenance command status information (includes maintenance logic busy, illegal command, and PC lock indicators) to the CR file maintenance registers.

4-39 INSTRUCTION REPERTOIRE

The Peripheral Processor (PP) instruction repertoire consists of 219 basic instructions and one no-op instruction, as shown in the Karnaugh map of figure 4-12. Each of these instructions (except the no-op) deals with immediate data or data in Central Memory (CM), Read Only Memory (ROM), the VPR file, or the CR file. The R, T, and N fields of the instruction (refer to paragraph 1-15) specify the data involved in the instruction as follows:

- R field addressing - The 4-bit R field (ABCD) is used to specify a VPR or CR at the word, halfword, or byte level. When a VPR is specified, the first two bits (AB) identify the VPR word (00 identifies VPR0, 01 identifies VPR1, etc.), the third bit (C) identifies the halfword (0 for right and 1 for left), and the third and fourth together (CD) identify the byte (00 for byte 0, 01 for byte 1, etc.). When a CR is specified, byte 3 of VPR3 is added to the R field and the result (ABCDEFGH) identifies the CR (four bits is not enough to identify 64 registers down to the byte level). The first six bits (ABCDEF) of the sum identify the word (000000 identifies CR0, 000001 identifies CR1, etc.) and the last two bits identify the halfword and byte as mentioned for the VPR case.
- T field addressing - The 4-bit T field is used to specify the VPR halfword to be added to the quantity specified by the N field (indexing). The first bit is used to indicate whether the current instruction is direct (0) or indirect (1) and the last three bits specify the VPR halfword involved. The left half of VPR0 is not used, however, because 000 specifies no indexing (0000 and 1000 specify the direct and indirect cases, respectively, of no indexing, 0001 and 1001 specify the direct and indirect cases of indexing with the right halfword of VPR0, ... 0111 and 11111 specify the direct and indirect cases of indexing with the right halfword of VPR3)).
- N field addressing - The 16-bit N field is used to specify an immediate operand, CM address (α), branch address (β), VPR, or CR. The quantity specified is determined by the op-code of the instruction. A VPR or CR is identified by the N field as described in the R field addressing (the four LSB's identify a VPR and the eight LSB's identify a CR).

$C_2C_3 \rightarrow 00$ | 01 | 11 | 10

C_6C_7

C_4C_5

$C_0C_1 = 00$

C_4C_5 \ C_6C_7	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	00 NOP NOOP	01	03	BC 02 UC UCB	ST 10 SFW STCM	STL 11 SFLH STHCM	STR 13 SFRH STHCM	BCS 12 UCV UCBLP	CE 30 CWE SKUCM	CEL 31 CLHE SKUCM	CER 33 CRHE SKUCM	BC 32 UCX UCB	20	21	23	22
01	LD 04 LW LDCM	LDL 05 LLH LDCM	LDR 07 LRH LDCM	LDA 06 LAB LDCM	ST 14 SW STCM	STL 15 SLH STHCM	STR 17 SRH STHCM	STA 16 SAB STCM	CN 34 CWN SKUCM	CNL 35 CLHN SKUCM	CNR 37 CRHN SKUCM	36	24	25	27	26
11	LD 0C LXW LDCM	LDL 0D LXLH LDCM	LDR 0F LXRH LDCM	LDA 0E LXAB LDCM	ST 1C SXW STCM	STL 1D SXLH STHCM	STR 1F SXRH STHCM	STA 1E SXAB STCM	3C	3D	3F	3E	2C	2D	2F	2E
10	LD 08 LXFW LDCM	LDL 09 LXFLH LDCM	LDR 0B LXFRH LDCM	LDF 0A LXVP LDUF	ST 18 SXFW STCM	STL 19 SXFLH STHCM	STR 1B SXFRH STHCM	STF 1A SXVP STUF	LD 38 LFW LDCM	LDL 39 LFLH LDCM	LDR 3B LFRH LDCM	LDF 3A LVP LDUF	28	29	2B	STF 2A SVP STUF

$C_0C_1 = 01$

C_4C_5 \ C_6C_7	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	AN 40 BAW CMLOU	ANL 41 BALH CMLOU	ANR 43 BARH CMLOU	BR 42 UR UCB	AD 50 AW CMAU	ADL 51 ALH CMAU	ADR 53 ARH CMAU	BCAS 52 UCAV UCBLP	ADI 70 AIW IMAU	ADHI 71 AIH IMAU	ADBI 73 AIB IMAU	LDI 72 LIW LDIM	SHA 60 SHA SHFT	ANHI 61 BAIH IMLOU	ANBI 63 BAIB IMLOU	LDI 62 LFIW LDIM
01	OR 44 BOW CMLOU	ORL 45 BOLH CMLOU	ORR 47 BORH CMLOU	BRS 46 URV UCBLP	SU 54 MW CMAU	SUL 55 MLH CMAU	SUR 57 MRH CMAU	BRSM 56 UCAS UCBSP	SUI 74 MIW IMAU	SUHI 75 MIH IMAU	SUBI 77 MIB IMAU	LDHI 76 LIM LDIM	SHL 64 SHL SHFT	ORHI 65 BOIH IMLOU	ORBI 67 BOIB IMLOU	LDHI 66 LFIH LDIM
11	EX 4C BXW CMLOU	EXL 4D BXLH CMLOU	EXR 4F BXRH CMLOU	BCA 4E UCAX UCB	EXEC 5C EXCM EXCM	LDEA 5D LEA LEA	ANAZ 5F ANCM ANCM	BPC 5E UX UCB	CNI 7C CINW SKUIM	CNHI 7D CINH SKUIM	CNBI 7F CINB SKUIM	LDBI 7E LIB LDIM	SHC 6C SHC SHFT	EXHI 6D BXIH IMLOU	EXBI 6F BXIB IMLOU	LDBI 6E LFIB LDIM
10	EQ 48 BQW CMLOU	EQL 49 BQLH CMLOU	EQR 4B BQRH CMLOU	BCA 4A UCA UCB	PUSH 58 PSH PUSH	PULL 59 PUL PULL	MOD 5B MDF MDF	BPC 5A U UCB	CEI 78 CIEW SKUIM	CEHI 79 CIEH SKUIM	CEBI 7B CIEB SKUIM	7A	68	EQHI 69 BQIH IMLOU	EQBI 6B BQIB IMLOU	6A

$C_0C_1 = 11$

C_4C_5 \ C_6C_7	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	AN C0 BAVW PPULO	ANH C1 BAVH PPULO	ANB C3 BAVB PPULO	TZL C2 TZL CRLO	AD D0 AVW UAU	ADH D1 AVH UAU	ADB D3 AVB UAU	TSZL D2 TSZL CRTSR	F0	F1	F3	RL F2 CBZL CRSRT	AN E0 BACW PPULO	ANH E1 BACH PPULO	ANB E3 BACB PPULO	TAZL E2 TAZL CRLO
01	OR C4 BOVW PPULO	ORH C5 BOVH PPULO	ORB C7 BOVB PPULO	TZR C6 TZR CRLO	SU D4 MVW UAU	SUH D5 MVH UAU	SUB D7 MVB UAU	TSZR D6 TSZR CRTSR	LDMB F4 LFA LDLFA	PULL F5 TPOL TPOL	F7	RR F6 CBZR CRSRT	OR E4 BOCW PPULO	ORH E5 BOCH PPULO	ORB E7 BOCB PPULO	TAZR E6 TAZR CRLO
11	EX CC BXVW PPULO	EXH CD BXVH PPULO	EXB CF BXVB PPULO	TOR CE TOR CRLO	CN DC CVNW SKUPP	CNH DD CVNH SKUPP	CNB DF CVNB SKUPP	TSOR DE TSOR CRTSR	CN FC CCNW SKUPP	CNH FD CCNH SKUPP	CNB FF CCNB SKUPP	SR FE CBOR CRSRT	EX EC BXCW PPULO	EXH ED BXCH PPULO	EXB EF BXCB PPULO	TAOR EE TAOR CRLO
10	EQ C8 BQVW PPULO	EQH C9 BQVH PPULO	EQB CB BQVB PPULO	TOL CA TOL CRLO	CE D8 CVEW SKUPP	CEH D9 CVEH SKUPP	CEB DB CVEB SKUPP	TSOL DA TSOL CRTSR	CE F8 CCEW SKUPP	CEH F9 CCEH SKUPP	CEB FB CCEB SKUPP	SL FA CBOL CRSRT	EQ E8 BQCW PPULO	EQH E9 BQCH PPULO	EQB EB BQCB PPULO	TAOL EA TAOL CRLO

$C_0C_1 = 10$

C_4C_5 \ C_6C_7	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	LD 80 LVW LDPPU	LDH 81 LVH LDPPU	LDB 83 LVB LDPPU	VPR 82 VPRT UCSRT	ST 90 SVW STPTP	STH 91 SVH STPTP	STB 93 SVB STPTP	TRZL 92 TRZL CRTSR	TZ B0 TZW CBAT	TZH B1 TZH CBAT	TZB B3 TZB CBAT	IBZ B2 XIZ CBIMD	TZ A0 TFZW CBAT	TZH A1 TFZH CBAT	TZB A3 TFZB CBAT	A2
01	LD 84 LFVW LDPPU	LDH 85 LFVH LDPPU	LDB 87 LFVB LDPPU	VPS 86 VPST UCSRT	ST 94 SFVW STPTP	STH 95 SFVH STPTP	STB 97 SFVB STPTP	TRZR 96 TRZR CRTSR	TN B4 TNW CBAT	TNH B5 TNH CBAT	TNB B7 TNB CBAT	IBN B6 XIN CBIMD	TN A4 TFNW CBAT	TNH A5 TFNH CBAT	TNB A7 TFNB CBAT	A6
11	LD 8C LFCW LDPPU	LDH 8D LFCH LDPPU	LDB 8F LFCB LDPPU	VPTO 8E VPTO UCT	ST 9C SFCW STPTP	STH 9D SFCH STPTP	STB 9F SFCB STPTP	TROR 9E TROR CRTSR	TM BC TMW CBAT	TMH BD TMH CBAT	TMB BF TMB CBAT	DBN BE XDN CBIMD	TM AC TFMW CBAT	TMH AD TFMH CBAT	TMB AF TFMB CBAT	BPCS AE UV UCBLP
10	LD 88 LCW LDPPU	LDH 89 LCH LDPPU	LDB 8B LCB LDPPU	VPTZ 8A VPTZ UCT	ST 98 SCW STPTP	STH 99 SCH STPTP	STB 9B SCB STPTP	TROL 9A TROL CRTSR	TP B8 TPW CBAT	TPH B9 TPH CBAT	TPB BB TPB CBAT	DBZ BA XDZ CBIMD	TP A8 TFPW CBAT	TPH A9 TFPH CBAT	TPB AB TFPB CBAT	AA

(B)124736

Figure 4-12. Peripheral Processor Instruction Set

The data specified by the T and N fields combines with the instruction opcode to dictate one of the operand types (the R field may or may not specify a second operand) listed in table 4-1.

Table 4-1. Peripheral Processor Operand Types

Operand Type	Description
Immediate Operand	An immediate operand is developed by sign extending the 16-bit N field and the VPR halfword specified by the T field to 32 bits (N^{32} and T^{32}, respectively), and adding the results ($N^{32}+T^{32}$, where $T^{32}=0$ for no indexing).
CM Address (α) Base Relative Branch	A CM address or base relative branch address is developed by adding the 16-bit N field (N^{16}), the 24-bit CM base register value (B^{24}), and VPR halfword specified by the T field sign extended to 24 bits (T^{24}) ($N^{16}+B^{24}+T^{24}$, where $T^{24}=0$ for no indexing).
CM Absolute Operand VPR/CR Operand ROM Branch Address CM Absolute Branch	These operand types are developed by adding the 16-bit N field to the 24-bit sign extended VPR halfword specified by the T field ($N^{16}+T^{24}$, where $T^{24}=0$ for no indexing). The LSB's (four for a VPR operand and eight for a CR operand) are used in the VPR/CR operand case as described in the R field addressing.
PC Relative Branch	A PC relative branch address is developed by adding the 24-bit sign extended N field, the 24-bit PC value (PC^{24}), and the 24-bit sign extended VPR halfword specified by the T field ($N^{24}+PC^{24}+T^{24}$, where $T^{24}=0$ for no indexing).

When an indirect instruction retrieves an indirect cell via one of the operands just described, the indirect cell operand address is developed by adding the 24-bit address field (ADR^{24}, refer to paragraph 1-16) and the 24-bit sign extended VPR halfword specified by the T field ($ADR^{24}+T^{24}$). A variation to the normal T and N field operand development occurs for the augmented instructions. In these cases, the three LSB's of the developed effective

address (CM or ROM address) are replaced by the active VP code. When augmenting occurs in combination with indirect, only the first level of indirect is augmented.

Each of the basic instruction groups utilizes some portion of the instruction field addressing and T and N field operand development procedures and is described in the order listed below:

- Stores
- Loads
- Arithmetic
- Logical
- Compare and skip
- Shifts
- Stack
- Set/reset CR bits
- Test CR bits and skip
- Test CR bits, set/reset, and skip
- Set/reset CR VP flag
- Test CR VP flag and skip
- Arithmetic conditional branches
- Increment/decrement and test conditional branches
- Unconditional branches
- Unconditional branch and load PC
- Unconditional branch to ROM and store PC
- Analyze effective address
- Load effective address
- Load CM base register
- Execute CM
- Test poll bits

4-40 STORE INSTRUCTIONS (ST, STA, STH, STB, STL, STR, and STF)

The store instructions store the VPR or CR operand specified by the R field in the VPR, CR, or CM location developed by the T and N fields. Both word and halfword stores may involve a register to register or register to CM data transfer (the store CM absolute instruction is legal only on the word

level, however), but the byte stores can only be a register to register transfer (the PP is capable of modifying CM data at the word and halfword level). All register to CM stores have an identical augmented counterpart and indirect addressing is illegal for the halfword and byte register to register stores.

A special case store instruction is the store VPR file instruction (STF), which stores all four VPR's of the VP executing the instruction in four consecutive CM locations, the first of which is specified by the T and N field operand. If the specified CM address is not a multiple of four, it is forced to a multiple of four by zeroing the two LSB's. Augmenting occurs in the three bits adjacent to the two zeroed LSB's.

4-41 LOAD INSTRUCTIONS (LD, LDA, LDH, LDB, LDL, LDR, LDF, LDI, LDHI, and LDBI)

The load instructions load the VPR, CR, CM, or immediate operand developed by the T and N fields into the VPR or CR specified by the R field. Word and halfword load instructions can use all combinations of both operands, but byte loads cannot involve a CM operand. An exception is the load CM absolute instructions, which are legal only on the word level. All CM loads have an identical augmented counterpart and indirect addressing is undefined for all immediate and halfword and byte register-to-register loads.

A special case load instruction is the load VPR file instruction (LDF), which loads all four VPR's with the contents of four consecutive CM locations, the first of which is specified by the T and N field operand. If the specified CM address is not a multiple of four, it is forced to a multiple of four by zeroing the two LSB's. Augmenting occurs in the three bits adjacent to the two zeroed LSB's.

4-42 ARITHMETIC INSTRUCTIONS (AD, ADH, ADB, ADL, ADR, ADI, ADHI, ADBI, SU, SUH, SUB, SUL, SUR, SUI, SUHI, SUBI)

The arithmetic instructions add/subtract the VPR, CM, or immediate operand developed by the T and N fields to/from the VPR specified by the R field. The result of the operation replaces the contents of the VPR specified by the R field. Word and halfword add/subtract instructions can use all combinations of both operands, but byte add/subtract instructions cannot involve a CM operand. Indirect addressing is undefined for all immediate arithmetic instructions and the halfword and byte arithmetic instructions involving two VPR operands.

4-43 LOGICAL INSTRUCTIONS (OR, ORH, ORB, ORL, ORR, ORHI, ORBI, AN, ANH, ANB, ANL, ANR, ANHI, ANBI, EX, EXH, EXB, EXL, EXR, EXHI, EXBI, EQ, EQH, EQB, EQL, EQR, EQHI, EQBI)

The logical instructions logically combine (OR, AND, EXCLUSIVE OR, or EQUIVALENCE) the VPR, CR, CM, or immediate operand developed by the T and N fields with the VPR specified by the R field. The result of the logical operation replaces the contents of the VPR specified by the R field. Word level logical instructions can combine all combinations of both operands except for immediate operands; halfword level logical instructions can combine all combinations of both operands without exception. Byte level logical instructions can combine all combinations of both operands except for CM operands. Indirect addressing is undefined for all immediate logical instructions and for the halfword and byte logical instructions involving two registers.

4-44 COMPARE AND SKIP INSTRUCTIONS (CE, CEH, CEB, CEL, CER, CEI, CEHI, CEBI, CN, CNH, CNB, CNL, CNR, CNI, CNHI, CNBI)

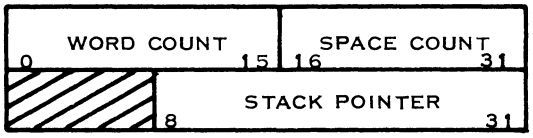
The compare and skip instructions compare the VPR, CR, CM, or immediate operand developed by the T and N fields to the VPR operand specified by the R field. The next instruction is skipped if the comparison for equality or non-equality evaluates true. Word and halfword compare instructions can use all combinations of both operands, but byte compare instructions cannot involve a CM operand. Indirect addressing is undefined for all immediate compare and skip instructions and for the halfword and byte compare and skip instructions involving two registers.

4-45 SHIFT INSTRUCTIONS (SHL, SHA, SHC)

The shift instructions perform right and left logical, arithmetic, and cyclic shifts (paragraph 4-17) on the VPR word specified by the R field in the direction and amount of the immediate operand developed by the T and N fields. The immediate operand is a signed number (positive for left shifts and negative for right shifts) in the range of +31 to -32. Indirect addressing is undefined for all shift instructions.

4-46 STACK INSTRUCTIONS (PUSH, PULL, MOD)

The stack instructions are used to maintain an operand stack by modifying the status parameters to reflect any change. The format of the status parameters is as follows:



The 16-bit word count indicates the number of operands in the stack, the 16-bit space count indicates the number of available 32-bit slots remaining in the stack (up to a maximum of 32, 767), and the 24-bit stack pointer is the address of the next available slot. Both parameter words are located in adjacent memory locations.

The push stack instruction reads the first parameter word from the CM address developed by the T and N fields, checks the space count for zero, and terminates execution if the space count is zero (the stack is full). If the space count is nonzero, the space count is decremented by one, the word count is incremented by one, and the first parameter word is replaced by the modified quantities. The second parameter word is read from CM, the contained stack pointer is used to store the VPR word specified by the R field in the operand stack, and the stack pointer is incremented by one and stored back in the second parameter word. The next sequential instruction is skipped.

The pull stack instruction reads the first parameter word from the CM address developed by the T and N fields, checks the word count for zero, and terminates execution if the word count is zero (the stack is empty). If the word count is nonzero, the word count is decremented by one, the space count is incremented by one, and the first parameter word is replaced by the modified parameters. The second parameter word is read from CM, the contained stack pointer is decremented by one and then used to read the last operand in the stack into the VPR specified by the R field, and the decremented stack pointer replaces the original. The next sequential instruction is skipped.

The modify stack instruction reads the first parameter word from the CM address developed by the T and N fields, adds the modification value in the VPR specified by the R field to the word count and subtracts the modification value from the space count (a positive modification value generates a gap of unused stack locations and a negative modification value deletes the most recent stack entries), checks both resulting quantities for a negative value, and terminates execution if either count is negative. If both counts are nonnegative, they replace the original word and space count, the second parameter word is read from CM, the modification value is added to the retrieved stack pointer, and the modified stack pointer replaces the original stack pointer. The next sequential instruction is skipped.

4-47 SET/RESET CR BIT INSTRUCTIONS (SL, SR, RL, RR)

The set/reset CR bit instructions set or reset (depending on the op-code) those bits, (marked by ones in the R field), in the right or left half of the CR byte specified by the T and N fields. The R field is used as a mask in this group of instructions and indirect addressing is undefined.

4-48 TEST CR BITS AND SKIP INSTRUCTIONS (TOL, TOR, TZL, TZR, TAOL, TAOR, TAZL, TAZR)

The test CR bits and skip instructions test the bit positions marked by ones in the R field in the left or right half of the CR byte specified by the T and N fields for any or all one(s) or zero(s). The desired test (any one, any zero, all ones, or all zeros) is determined by the op-code. If the test is not satisfied, the next instruction is executed. If the test is satisfied, the next instruction is skipped. Indirect addressing is undefined.

4-49 TEST CR BITS, SET/RESET, AND SKIP INSTRUCTIONS (TSZL, TSOL, TRZL, TROL, TSZR, TSOR, TRZR, TROR)

The test CR bits, set/reset, and skip instructions test the bit positions marked by ones in the R field in the left or right half of the CR byte specified by the T and N fields for any one or zero. If the desired test is satisfied, the next instruction is skipped. Independent of the test, the bit positions marked by ones in the R field are set or reset (depending on the op-code). Indirect addressing is undefined.

4-50 SET/RESET CR VP FLAG INSTRUCTIONS (VPS, VPR)

The set/reset CR VP flag instructions set or reset the flag bit in the CR byte specified by the T and N fields. The flag bit under consideration in the byte is determined by the number of the executing VP. Indirect addressing is undefined.

4-51 TEST CR VP FLAG AND SKIP INSTRUCTIONS (VPTO, VPTZ)

The test CR VP flag and skip instructions test the flag bit in the CR byte specified by the T and N fields for one or zero and skip the next instruction if the desired test is satisfied. The flag bit under test in the byte is determined by the number of the executing VP. Indirect addressing is undefined.

4-52 ARITHMETIC CONDITIONAL BRANCH INSTRUCTIONS (TZ, TZH, TZB, TN, TNH, TNB, TP, TPH, TPB, TM, TMH, TMB)

The arithmetic conditional branch instructions test the VPR or CR word, halfword, or byte specified by the R field for zero, nonzero, greater than zero, or less than zero. If the desired test is satisfied, a PC relative branch is taken to the location specified by the T and N fields.

4-53 INCREMENT/DECREMENT AND TEST CONDITIONAL BRANCH INSTRUCTIONS (IBZ, IBN, DBZ, DBN)

The increment/decrement and test conditional branch instructions increment or decrement by one the VPR halfword specified by the R field and test the

result for zero or nonzero. If the desired test is satisfied, a PC relative branch is taken to the location specified by the T and N fields.

4-54 UNCONDITIONAL BRANCH INSTRUCTIONS (BPC, BR, BC, BCA)

The unconditional branch instructions branch to the PC relative, ROM, base relative, or CM absolute operand address developed by the T and N fields, as determined by the op-code. The unconditional branch to CM instructions (PC relative, base relative, and CM absolute) all have an identical augmented counterpart.

4-55 UNCONDITIONAL BRANCH AND LOAD PC INSTRUCTIONS (BPCS, BCS, BRS, BCAS)

The unconditional branch and load PC instructions branch to the PC relative, base relative, ROM, or CM absolute operand address developed by the T and N fields and load the address of the next instruction in the current instruction stream in the VPR specified by the R field. The most significant bit of the VPR is set to indicate which instruction stream is currently being accessed (one for CM and zero for ROM).

4-56 UNCONDITIONAL BRANCH TO ROM AND STORE PC INSTRUCTION (BRSM)

The unconditional branch to ROM and store PC instruction branches to the ROM address specified by the T and N fields and stores the address of the next instruction in the current instruction stream in one of the eight contiguous CM locations beginning at 20_{16}. The identity of the VP executing the instruction is added to the 20_{16} base to determine the exact CM location. Indirect addressing is undefined.

4-57 ANALYZE EFFECTIVE ADDRESS INSTRUCTION (ANAZ)

The analyze effective address instruction retrieves an object instruction from the CM address developed by the T and N fields. The T and N field operand of the object instruction is developed in the normal manner and the result is stored in the VPR specified by the R field of the analyze instruction. The result of the object instruction T and N field operand development is that the object instruction is effectively in the location of the analyze instruction, with the following exception: a PC relative branch address is developed with a PC value that is one greater than it would be if the PC relative branch was in the analyze instruction location.

4-58 LOAD EFFECTIVE ADDRESS INSTRUCTION (LDEA)

The load effective address instruction loads the CM effective address developed by the T and N fields in the VPR specified by the R field.

4-59 LOAD CM BASE REGISTER INSTRUCTION (LDMB)

The load CM base register instruction loads the CM base register associated with the active VP with the three least significant bytes of the VPR specified by the T and N fields. The LDMB instruction may be indirect. When this is the case, the first level of indirect is through a VPR and any additional levels use CM. This instruction is exempt from the CR protect mechanism (no interrupt will occur when the CR protect logic is enabled and this instruction executes).

4-60 EXECUTE CM INSTRUCTION (EXEC)

The execute CM instruction executes the CM object instruction specified by the T and N fields as though it were in the location of the original execute CM instruction, except when the object instruction is a PC relative branch. When the object instruction is a PC relative branch, the PC value used in the development of the branch address is one greater than that used if the object instruction actually replaced the execute CM instruction.

4-61 TEST POLL BITS INSTRUCTION (POLL)

The test poll bits instruction tests the CR byte specified by the T and N fields for a one in any of the bit positions and skips the next instruction if a one is found. The number of bit positions to the most significant one is inserted in the VPR halfword specified by the R field. (If no one is found, the VPR halfword is cleared and no skip is taken). Indirect addressing is undefined.

4-62 INSTRUCTION PROCESSING

Each of the eight Virtual Processors (VP's) is capable of executing an independent program residing in Central Memory (CM) or Read Only Memory (ROM). The hardware directly involved in retrieving, holding, expanding, and executing program instructions includes a Program Counter (PC), Single Word Buffer Address register (SWBA), Single Word Buffer Data register (SWBD), Next Instruction Register (NIR), Instruction Register (IR), and the time-shared Indexer and Main Instruction Register (MIR).

The PC, SWBA, SWBD, NIR, and IR are discussed in the general and detailed description of the VP's, the Indexer (including the PC indexer, TN field indexer, and register indexer) is discussed in the general and detailed description of the Indexer, and the MIR is discussed in the general and detailed description of PP control. Refer to figure 4-13 for a diagram relating all of these components.

The procedure required to retrieve and prepare an instruction for execution involves three basic steps (N+2, N+1, and N in figure 4-13), which give rise to the phrase "PP three-level pipe". The first basic step, instruction acqui-

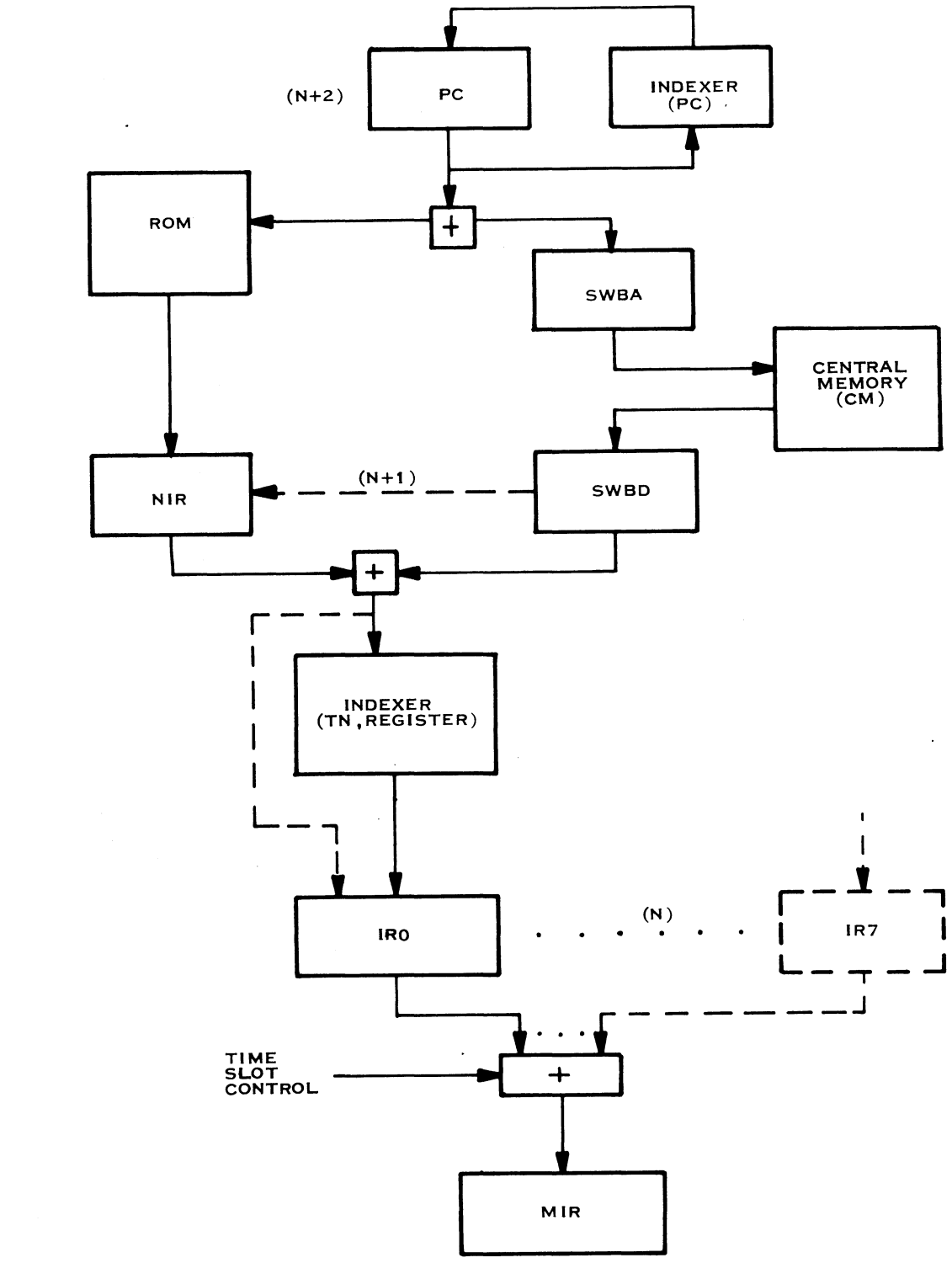


Figure 4-13. Peripheral Processor Instruction Processing

sition, centers around the PC operation. When PP control determines that the PP three-level pipe is to be advanced one level, and the time slot for the VP being discussed occurs, the PC address is applied to ROM or the SWBD as directed by PP control. If the instruction source is ROM, ROM immediately responds by transferring the addressed instruction to the NIR. If the instruction source is CM, the VP issues a read request and CM responds by returning the addressed instruction to the SWBD. The second basic step, instruction expansion, is now possible. When the next PP three-level pipe advancement occurs, the SWBD or NIR R, T, and N fields (paragraph 1-15) are expanded by the TN field and register indexers into the source, destination, and effective addresses for the IR. The dotted line around these two indexers, in figure 4-13, represents the op-code, state, and control flags developed by PP control for the IR. The third basic step, instruction execution, is now possible. When the next time slot occurs for this VP, the IR data is transferred to the MIR for the duration of the time slot (approximately 85 nanoseconds). During this 85 nanoseconds, the MIR data is used by PP control to direct execution of one step of the instruction. If termination of the instruction occurs at this step, PP control initiates advancement of the PP three-level pipe so a new instruction can be brought into the IR. If termination of the instruction does not occur at this step, PP control updates the IR, at the conclusion of the time slot, to the next step of the multistep instruction. When the next time slot for this VP occurs, the next step of the same instruction is executed. The MIR is continually receiving IR data in various steps of execution from all VP's that are executing programs in this manner.

The terms N, N+1, and N+2 in figure 4-13 represent three sequential instructions in the PP three-level pipe at any one time from either ROM or CM. The instruction from location N has been in the PP three-level pipe during two level advances (the instructions from locations N-2 and N-1 have both terminated while N was in the pipe), the instruction from location N+1 has been in the PP three-level pipe during one level advance (termination of N-1), and the instruction at location N+2 is currently addressed by the PC. When execution of instruction N terminates, instruction N+1 goes through the indexing phase to the IR, instruction N+2 is retrieved from either ROM or CM and inserted in the NIR or SWBD, respectively, and the PC indexer increments the PC to location N+3. This PP three-level pipe advancement occurs every time an instruction terminates. This rather complicated method of instruction processing necessitates a few data transfers and time delaying techniques that interrupt this smooth flow (the dotted line between the SWBD and NIR in figure 4-13 is one of these interruptions). These situations are called sequential dependencies and are discussed in the following paragraphs.

4-63 SEQUENTIAL DEPENDENCIES

Sequential dependencies that do interrupt the smooth flow of the PP three-level pipe include the following situations:

- CM instruction requires CM access
- Current instruction modifies next instruction
- Current instruction modifies next instruction index
- Unconditional branch and load PC instruction followed by PC relative branch

Each of these hazards, with its associated solution, is described in the following paragraphs.

4-64 CM INSTRUCTION REQUIRES CM ACCESS. This problem occurs when a VP is executing a program resident in CM and one of the program instructions is required to read from or write to CM. The problem exists because the SWBD contains the next instruction to be executed and it is needed for a CM read or write operation. The solution is provided by transferring the next instruction to the NIR and setting the NIL bit in the current instruction to reflect the transfer. The SWBD is now free to engage in the CM read or write operation without destroying data.

4-65 CURRENT INSTRUCTION MODIFIES NEXT INSTRUCTION. This problem occurs when the current instruction stores data in the CM location of the next instruction. The problem exists because the next instruction already resides in the NIR (due to the saving procedure described in the previous paragraph) in the unmodified form. The solution is provided by first comparing the operand address plus one with the current PC value (the PC has been incremented one location past the address of the next instruction). This check is called the Write Cycle Equality (WCE) test. If WCE does exist (the next instruction is modified), the modified data to be stored in the SWBD is transferred to the NIR. This action replaces the "old" next instruction with the "new" next instruction, but also adds an extra step to the store instruction.

4-66 CURRENT INSTRUCTION MODIFIES NEXT INSTRUCTION INDEX. This problem occurs when the current instruction loads data into a CR or VPR that is used by the next instruction for indexing purposes (address development) during the terminating step of the current instruction. The problem exists because both the indexing and the register loading occur during the same step, so the next instruction will be indexed with the unmodified register value. The solution is provided by first checking to see if the register to be loaded by the current instruction enters into the indexing of the next instruction. This check is used to develop the Dependency (D) signal. If Dependency does exist (the indexing register is to be modified), the load is executed but the indexing operation is delayed one step. This action allows for the indexing register modification but also adds one step to the current instruction.

4-67 UNCONDITIONAL BRANCH AND LOAD PC INSTRUCTION FOLLOWED BY PC RELATIVE BRANCH. This problem occurs when an unconditional branch and load PC instruction attempts to save the address of the next instruction at the same time a PC relative branch (next instruction) is in the indexing phase. The problem exists because the PC relative branch instruction develops its branch address using the address following the unconditional branch and load PC instruction address as the PC value (the PC was decremented for the save operation) rather than one greater than the branch address of the unconditional branch and load PC instruction. The problem can be more readily understood by stepping through the unconditional branch and load PC instruction.

Step 1 The branch address from the IR is transferred to the SWBA to retrieve the branched to instruction, and the branch address plus one is temporarily stored in the effective address portion of the IR. The PC is decremented by one to the address of the next instruction.

Step 2 The decremented PC is saved in the designated VPR; the IR effective address (containing the branch address plus one) is transferred to the SWBA to retrieve the instruction following the branched to instruction; and the IR effective address is incremented by one and stored in the PC. In addition, the instruction retrieved in step 1 is indexed into the IR.

If the instruction retrieved in step 1 and indexed in step 2 is a PC relative branch, the saved PC value would be used in the development of the PC relative branch address. The solution to this problem is provided by first checking to see if a PC relative branch does follow the unconditional branch and load PC instruction. When this is the case, the next instruction BTN (NIBTN) signal is used to delay the PC relative branch indexing operation until the PC holds the address following the PC relative branch instruction address. This procedure modifies step 2 and adds two additional steps to the execution of the unconditional branch and load PC instruction.

Step 2 The decremented PC is saved in the designated VPR and the IR effective address is incremented by one (it is now two greater than the original branch address) and inserted in the PC.

Step 3 The PC is decremented by one to point to the instruction following the PC relative branch.

Step 4 The PC relative branch is indexed and input to the IR, the PC value is impressed upon memory to retrieve the next instruction, and the PC value is incremented to complete the PP three-level advance.

4-68 INSTRUCTION TRANSFER TABLES

The instruction transfer tables provide a step-by-step summary of instruction execution for each of the instruction subgroups (refer to appendix A). Each subgroup has two transfer tables, one for CM source and one for ROM source instructions. The header information on each of the transfer tables includes a general description of the subgroup, the subgroup mnemonic and source, and the hexadecimal representation, software mnemonic, and hardware mnemonic of each instruction in the subgroup. A description of the transfer table columns is provided in table 4-1A.

Table 4-1A. Transfer Table Column Description

Column Name	Description
Step	Sequential numbering of instruction states.
Present State	State the instruction is currently in.
Next State	State the instruction will be in after the next time slot.
Transfers	Events that will take place when a time slot occurs and the proper conditions exist for any given present state.
Mode	Indicates instruction source CM (M) or ROM (¬M)).
SWBC	The OUT column indicates if a CM read or write request is possible (BA) or not (¬BA) and the IN column indicates when a read (RC) or write (WC) request is being made.
Op Code	Subgroup mnemonic.
Facility	Indicates the hardware involved in performing the data transfer on the same line of the transfer column.
Source-Destination	Indicates the indexer involved in developing source and/or destination addresses used in the data transfer on the same line of the transfer column.
Conditions	Indicates hardware conditions that must exist for the events in the transfer column to take place.

When an instruction is indexed and expanded from the SWBD or NIR to the IR, PP control is responsible for development of the op-code, initial state, and control flags and the Indexer is responsible for development of the source, destination, and effective addresses. When a time slot for the VP being discussed occurs, the IR data is transferred to the MIR for the duration of the time slot in order to direct the events listed in the transfer column of the transfer table associated with the IR instruction. If the instruction is not complete at the conclusion of the time slot, the IR present state is modified to reflect the next state via PP control and the VP waits for the next time slot. When the instruction does terminate (indicated by NIN in the transfer tables), the PP three-level pipe is advanced one level (a new instruction is brought into the IR).

The state information presented in the transfer tables represents six bits in the IR, three to define the state class and three to define the step within the state class. Each of the state class bits has a definition, and, when combined with the other two state class bits and three step bits, gives a fair description of what is actually happening in the PP. Table 4-2 contains some useful combinations of state class and step, with their associated generalized descriptions.

The actual state transformation from the present state to the next state is directed by PP control (paragraphs 4-37 and 4-154), which utilizes the present state, various test results (this includes the Write Cycle Equality (WCE) signal, Dependecny (D) signal, Skip Taken (ST) signal, Branch Taken (BT) signal, etc.), and the relevent control flags to direct the transformation. When the current instruction terminates, a signal called NINS is developed by PP control to indicate the use of the initial state of the next instruction in the development of the present state. The initial state of an instruction is primarily determined by the op-code.

The following paragraphs, accompanied by figure 4-1, provide a few examples of tracing instruction execution in the PP via the transfer tables.

4-69 NO OPERATION INSTRUCTION

Refer to page A-74 of appendix A for a CM source transfer table of the no-op instruction. The no-op instruction begins execution in state class 3, step 2. If the SWBC buffer is not available (¬BA) to the VP executing the no-op and an interrupt did not occur during the previous instruction (¬INTF), the no-op remains in state class 3, step 2 without initiating any data transfers. When the buffer does become available, the following events take place during the next time slot:

(PC) ⟶ SWBA
(NIL) ⟶ IR
(PC)+1 ⟶ PC
0 ⟶ NIL

Table 4-2. Instruction States

EX*	LA*	LC*	BC*	General Description
0	1	0	1	When direct, a branch state with PC indexing and instruction retrieval. When indirect, the first step of the indirect cycle.
0	1	0	2	When direct, a stack instruction skip state with PC indexing and instruction retrieval. When indirect, the second step of the indirect cycle.
0	1	0	>2	Successive indexing steps of multiple-step instructions.
0	1	1	1	Indexing with write cycle compare.
0	1	1	2	Indexing with no termination inhibits except ¬BA* and interrupts.
0	1	1	3	Indexing on β* with no termination inhibits except ¬BA and interrupts.
1	0	0	1	Step 1 of multistep instructions.
1	0	0	>1	Successive execution steps of multistep instructions.
1	1	1	1	Execution with indexing and no write cycle compare.
1	1	1	2	Execution with indexing and dependency, skip, or branch testing.
1	1	1	3	Indexing on β with no termination inhibits except ¬BA and interrupts.

* EX-Execution State LA-Look Ahead (Indexing) State LC-Last Cycle State
BC-Bit Count (Step) BA-Buffer Available
β-Branch Address

The PC is transferred over the CMAB bus to the SWBA, the PC indexer uses the PC value on the CMAB bus to increment the PC, the SWBD or NIR (as determined by the NIL bit of the MIR) is expanded by PP control and the TN field and register indexers and input to the IR, and the IR NIL bit is zeroed by PP control to indicate the next instruction will be in the SWBD (because of the CM instruction source). In addition, a read request is issued by PP

control to the SWBC to retreive the instruction addressed by the SWBA. The net effect of the described events is advancement of one level in the PP three-level pipe.

4-70 STORE WORD TO CENTRAL MEMORY INSTRUCTION

Refer to page A-1 of appendix A for a CM source transfer table of the store word to CM instructions. The store word to CM instructions begin execution in state class 4, step 1. When the buffer becomes available (BA), the following events take place:

$$(PPU)R \longrightarrow SWBD$$
$$(IR)TN \longrightarrow SWBA$$
$$(SWBD) \longrightarrow NIR$$
$$1 \longrightarrow NIL$$
$$(IR)TN+1 \longrightarrow IR$$

If a VPR is specified by the R field, the desired VPR is enabled over the VPAB bus, over the Main Data Bus (MDB), through the AU aligner, and over the AU2B bus to the SWBD. If a CR is specified by the R field, the desired CR is enabled over the CRAB buses, over the MDB, through the AU aligner, and over the AU2B bus to the SWBD. The IR effective address developed by the T and N fields is enabled over the CMAB bus to the SWBA; the next instruction in the SWBD is saved in the NIR via the CMDB bus; the IR NIL bit is set by PP control to reflect the save; and the IR effective address is incremented by one by the TN field indexer and stored back into the IR. (This is done for the WCE test mentioned in paragraph 4-63.) A write request is issued by PP control to the SWBC to store the VPR or CR to the desired CM address and the IR state class and step are advanced to three and one, respectively, by PP control.

The next time this instruction receives a time slot, the WCE indicator is used to determine whether termination of this instruction should occur now or later. When the buffer is available and the current instruction has not modified the next instruction (¬WCE), the PP three-level pipe is advanced one level as described in the CM source no-op instruction. When the current instruction does modify the next instruction (WCE), the SWBD is enabled over the CMDB bus to the NIR so that the modified instruction replaces the old instruction. The IR state class and step are advanced to three and two, respectively, by PP control. At the next time slot, the store word to CM instruction terminates by advancing the PP three-level pipe.

4-71 COMPARE CENTRAL MEMORY TO VPR INSTRUCTION

Refer to page A-45 of appendix A for a CM source transfer table of the compare CM to VPR instructions. The compare CM to VPR instructions begin execution in state class 4, step 1. When the buffer becomes available (BA), the following events take place:

(IR)TN ⟶ SWBA
(SWBD) ⟶ NIR
1 ⟶ NIL

The IR effective address of the CM quantity to be compared is enabled over the CMAB bus to the SWBA; the next instruction in the SWBD is saved in the NIR via the CMDB bus and the IR NIL bit is set by PP control to reflect the save; and a read request is issued by PP control to the SWBC to retrieve the CM quantity for comparison. The IR state class and step are advanced to seven and two, respectively, by PP control.

At the next time slot for which the buffer is available (BA), the retrieved CM quantity and VPR specified by the R field are both applied to the skip taken (ST) logic in the AU in the following manner: The desired VPR is enabled over the VPRB buses to the AU and the CM quantity in the SWBD is enabled over both the MDAB bus and the MDB to the AU. If the comparison is satisfied (a check for equality or inequality, depending on the instruction in the KSKUCM subgroup), the ST signal is true and the following takes place:

(PC) ⟶ SWBA
(PC)+1 ⟶ PC
0 ⟶ NIL

The PC is enabled over the CMAB bus to the SWBA; the PC indexer uses the PC value on the CMAB bus to increment the PC; the IR NIL bit is zeroed so that the next instruction previously saved in the NIR is skipped; and a read request is issued by PP control to the SWBC to retrieve the skipped-to instruction. The IR state class and step are advanced to three and two, respectively, by PP control and the PP three-level pipe is advanced one level using the skipped-to instruction at the next time slot for which the buffer is available (BA). If the comparison is not satisfied (¬ST), the PP three-level pipe is advanced one level without taking any skip.

4-72 INDIRECT CYCLE

Refer to page A-72 of appendix A for a CM source transfer table of the indirect cycle. The indirect cycle is executed when the next instruction to be executed (in the SWBD or NIR) has the first bit of its T field set (first indirect cycle) or the indirect cell retrieved from CM has the first bit of its T field set (multiple level indirect cycle) and the current instruction has completed execution (NIN in the transfer table). When indirect addressing is specified, the indirect cycle(s) is executed to develop the IR source, destination, or effective address prior to the normal execution of the instruction for which indirect addressing is defined. This indirect cycle is not valid for the conditional branch instructions (they have their own unique indirect cycle, as described in paragraphs 4-182 and 4-183) or for the instructions for which indirect addressing is undefined.

When the next instruction indirect indicator (DB) from PP control indicates the next instruction is indirect, the ignore indirect indicator (IGI) from PP control indicates indirect addressing is possible for the next instruction, and the instruction termination indicator (NINS) indicates that the current instruction has terminated, the IR DC bit is set by PP control and the indirect cycle enters execution in state class 2, step 1. If the PPTN flag associated with the indirect instruction indicates the indirect cell is located in CM (¬PPTN), the following events take place:

(IR)TN ⟶ SWBA
(SWBD) ⟶ NIR
1 ⟶ NIL

The IR effective address developed by the TN field indexer pointing to the indirect cell is enabled over the CMAB bus to the SWBA; the instruction following the indirect instruction in the SWBD is saved in the NIR via the CMDB bus and the IR NIL bit is set by PP control to reflect the save; and a read request is issued by PP control to the SWBC to retrieve the indirect cell. The IR state class and step are both advanced to two by PP control. If the PPTN flag associated with the indirect instruction indicates the indirect cell is located in a register (PPTN), the following events take place:

(PPU)TN ⟶ SWBD
SWBD ⟶ NIR
1 ⟶ NIL

The VPR or CR specified by the T and N fields is transferred to the SWBD as follows: If a VPR is specified, the desired VPR is enabled over the VPAB, over the MDB through the AU aligner, and over the AU2B bus to the SWBD. If a CR is specified, the desired CR is enabled over the CRAB busses, over the MDB, through the AU aligner, and over the AU2B bus to the SWBD. The next instruction is saved in the NIR as previously mentioned for the indirect cell-in-CM. The IR state class and step are both advanced to two by PP control.

At the next time slot for which the buffer is available (BA) and the next instruction indirect indicator signals termination of indirect addressing (¬DB), the TN field indexer develops an IR source, destination, or effective address from the SWBD indirect cell and the IR DC bit is zeroed by PP control. The former indirect instruction now enters its normal execution sequence. If the next instruction indirect indicator signals another level of indirect addressing (DB), the TN field indexer develops the indirect cell address for the IR from the current indirect cell in the SWBD and PP control sets the IR DC bit to reflect another level of indirect addressing. The retreiving of indirect cells continues until the termination level is reached (indirect addressing through a register is only possible at the first level, however).

4-73 INTERRUPT CYCLE

Refer to page A-76 of appendix A for a CM source transfer table of the interrupt cycle. The interrupt cycle is executed at the conclusion of the instruction during which a programmed or automatic interrupt occurred. PP control directs execution of the interrupt cycle, which executes the interrupt instruction at ROM location 10_{16} if an automatic interrupt occurred, or at ROM location 11_{16} if a programmed interrupt occurred (the interrupt instructions are the branch and save PC type). When the interrupt servicing routine terminates, control resumes with the instruction following the interrupted instruction. The net result is the squeezing of an interrupt servicing routine in the normal flow of instruction processing after the instruction during which the interrupt occurred.

When a programmed or automatic interrupt has been recorded and the associated IR interrupt bit has been set (INT), the current instruction has terminated (NINS), and the terminated instruction is not execute CM (¬EXCM), the following events take place:

$$1 \longrightarrow \text{INTF}$$
$$1 \longrightarrow \text{NIL}$$
$$\text{NO-OP} \longrightarrow \text{IR}$$
$$(\text{PC})+1 \longrightarrow \text{PC}$$

The check for ¬EXCM is necessary because the execute CM instruction has not really terminated until its object instruction has terminated. PP control sets the IR INTF bit to initiate the interrupt cycle and the IR NIL bit to indicate that the interrupt instruction to be executed will be retrieved from ROM. PP control zeros the IR op-code and the current PC value is incremented by the PC indexer to two instructions past the interrupted instruction. The IR state class and step are advanced to two and three, respectively, by PP control. When the interrupted VP receives its next time slot, the following events take place:

$$\text{E.A.} \longrightarrow \text{NIR}$$
$$(\text{PC})-2 \longrightarrow \text{PC}$$

The IR effective address reflecting the interrupt type (10_{16} for automatic and 11_{16} for programmed) is applied to ROM from PP control via the RMAB bus and the PC value is decremented by two via the PC indexer (the PC indexer receives the PC value to be modified via the CMAB bus). The PC value now points to the instruction following the interrupted instruction and the NIR contains the interrupt servicing instruction. The IR state class and step are advanced to three and two, respectively, by PP control.

At the next time slot for which the buffer is available (BA), the PP three-level pipe, with the no-op instruction at the execution level, the interrupt servicing instruction at the address preparation level, and the instruction following the interrupted instruction at the acquisition level, is advanced one level as described in the CM source no-op instruction. In addition, the IR INTF bit is zeroed to reflect the honored interrupt.

4-74 DETAILED DESCRIPTION

The remaining paragraphs in this section provide a detailed look at the eight major components of the PP (VP's, AU, Indexer, CR file, ROM, SWBC, PP Control, and Maintenance Logic). The detailed descriptions covering the eight major components are supplemented with detailed block diagrams and logic diagrams that aid in understanding the logic card diagrams in section VII of this manual. Timing diagrams are provided for the more involved timing circuits and transfer tables accompany the PP Control description of instruction execution.

4-75 VIRTUAL PROCESSORS

Each of the eight Virtual Processors (VP's) of the PP consists of a Program Counter Register (PC), Next Instruction Register (NIR), Instruction Register (IR), Virtual Processor Register File (VPR File), Central Memory Base Register (CM Base), Single Word Buffer Address Register (SWBA), and a Single Word Buffer Data Register (SWBD). A detailed description of each register type, supplemented with block diagrams and/or logic diagrams, is presented in the following paragraphs. The integrated operation of these areas is described in paragraph 4-62.

4-76 PROGRAM COUNTER REGISTER. Refer to figure 4-14 for a simplified block diagram of the eight PC's and the associated input and output logic. The primary function of each PC is to provide the associated VP with a pointer to the next instruction in Central Memory (CM) or Read Only Memory (ROM). Secondary functions include distribution of the PC to the MDB, Indexer I1, Indexer I2, PP Control, and the Single Word Buffer Controller, in support of the PP three-level pipe.

4-77 PC Loading Logic. During normal PP operation (maintenance logic not used), the source of the updated PC value is Indexer I1. When a PC is to be updated, the PC load enable line (PPI1PCE) from the PCCTL card permits the decoding of the PC load select lines (PPWMSC(0-2)) to develop the pointer to the PC of the active VP. The PPWMSC(0-2) lines are also supplied by the PCCTL card and contain the VP number of the active VP. The developed pointer is used to insert the updated PC value (32 bits) in the PC of the current active VP. Bit 0 of the updated PC value (indicates mode) is supplied by the PCCTL card, bits 1 ghrough 7 are set to zero, and bits 8 through 31 (the address) are supplied by Indexer I1.

4-78 PC Distribution Logic. When an unconditional branch and save PC instruction (BPCS, BCS, BRS, or BCAS) or the unconditional branch and store PC instruction (BRSM) is executing, the PCAB bus is used to route the PC of the active VP to the MDB. The PC to MDB enable line (PPPCABE)

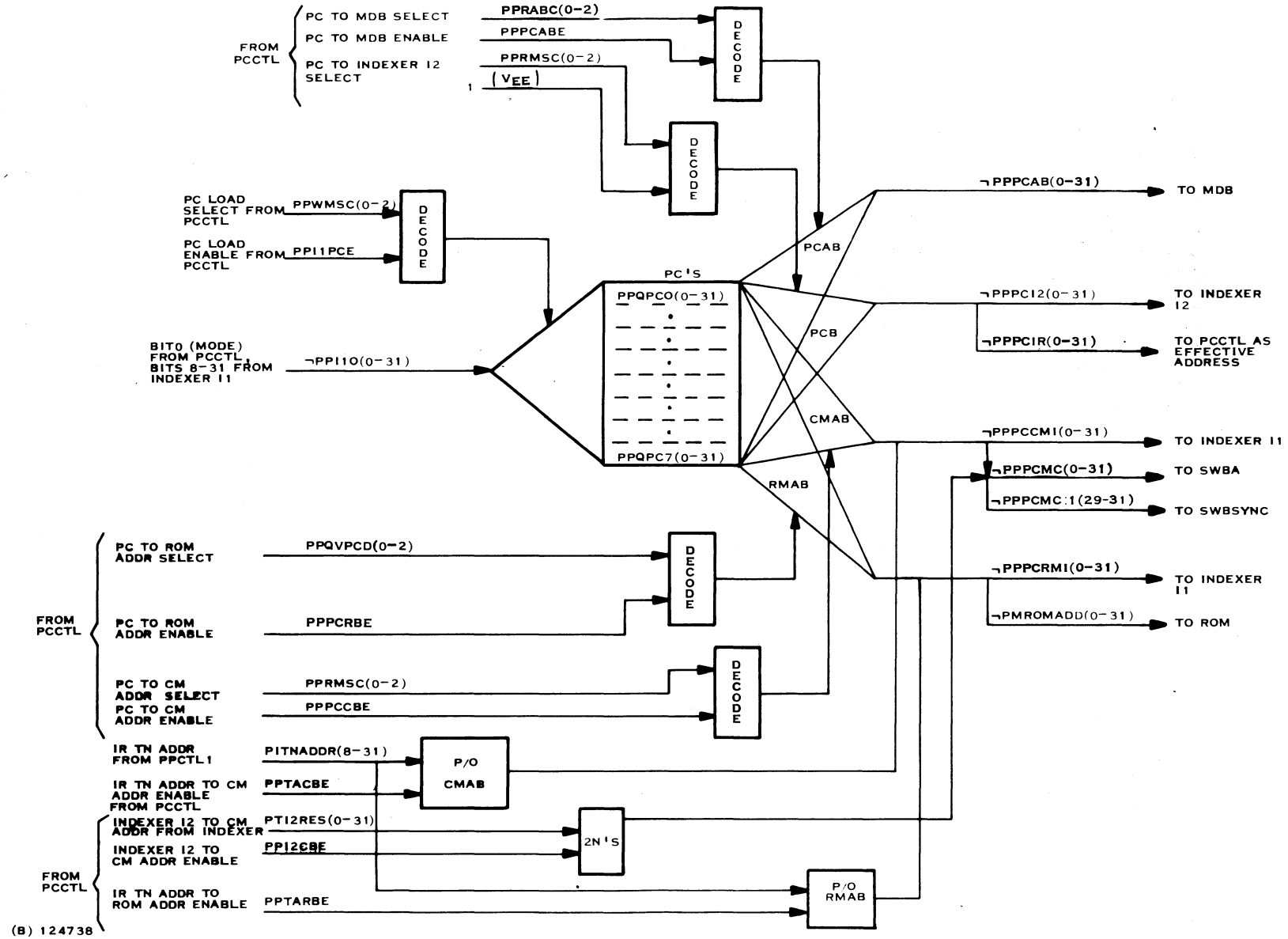
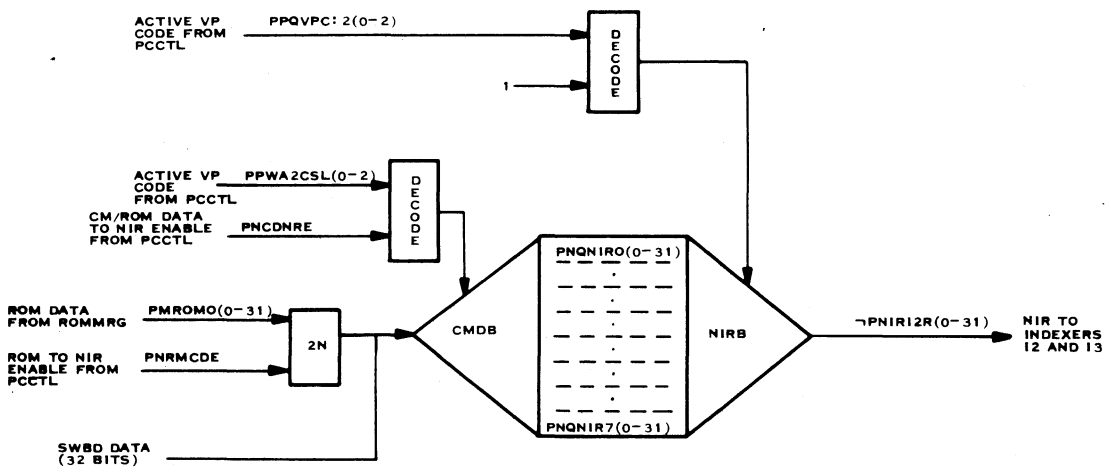


Figure 4-14. Program Counter Registers

permits the decoding of the PC to MDB select lines (PPRABC(0-2)) to select and route the PC of the active VP to the MDB (¬PPPCAB(0-31)). When any instruction is executing, the PCB bus is used to route the PC of the active VP to Indexer I2 and the PP Control logic used in performing the Write Cycle Equality (WCE) test. The PC to Indexer I2 select lines (PPRMSC(0-2)) are decoded and used to select and route the PC of the active VP to Indexer I2 (where the ¬PPPCI2(0-31) lines are used in PC relative instructions) and the PP Control logic on the PCCTL card (where the ¬PPPCIR(0-31) lines are used in performing the WCE test). The CMAB bus is used to route the PC of the active VP to Indexer I1 for indexing, to the associated SWBA for instruction location in CM, and to the Single Word Buffer Controller logic for CM zone selection. In these three cases, the PC to CM address enable line (PPPCCBE) permits the decoding of the select lines (PPRMSC(0-2)) so that the necessary distribution is possible. The CMAB bus is also used to route the IR TN address (PITNADDR(8-31)) to the SWBA when the IR TN address to CM address enable line (PPTACBE) goes to one or the output of Indexer I2 (PTI2RES(0-31)) to the SWBA when the Indexer I2 to CM address enable line (PPI2CBE) goes to one (stack instructions). The RMAB bus is used to route the PC of the active VP to Indexer I1 for indexing and to ROM for instruction location when the PC to ROM address enable line (PPPCRBE) permits the decoding of the select lines. The RMAB bus is also used to route the IR TN address to ROM when the IR TN address to ROM address enable line (PPTARBE) goes to one (branch to ROM type instructions).

4-79 NEXT INSTRUCTION REGISTER. Refer to figure 4-15 for a simplified block diagram of the eight NIR's and the associated input and output logic. The primary function of each NIR is to provide the associated VP with a 32-bit register for holding words retrieved from ROM. A secondary function is to provide temporary storage of SWBD data when the SWBD is being used during the execution of an instruction. When data is being read from ROM(PMROMO(0-31)), the ROM to NIR enable line (PNRMCDE) gates the ROM data to the NIR selection logic. The CM/ROM data to NIR enable line (PNCDNRE) permits the decoding of the active VP code lines (PPWA2CSL(0-2)) in order to develop a pointer to the NIR of the active VP. The pointer is then used to insert the ROM data into the proper NIR. When data from the SWBD of the active VP is to be transferred to the associated NIR, the SWBD data is applied to the NIR selection logic (instead of ROM data) and inserted in the proper NIR via the mentioned pointer. During the execution of all instructions, the NIR data is distributed to Indexers I2 and I3 over the NIRB bus. The active VP code lines (PPQVPD:2(0-2)) are decoded at each execution period and used to select the NIR of the active VP. The selected NIR is then transferred over the NIRB bus and input to Indexers I2 and I3 (¬PNIRI2R(0-31)) for IR development.



(B) 124739

Figure 4-15. Next Instruction Registers

4-80 INSTRUCTION REGISTER. Refer to figure 4-16 for the format of one of the eight 64-bit IR's. The IR data is grouped as follows:

- Operation code (bits 0 through 7)

The eight bit op-code specifies the instruction to be executed. The source of the op-code is the associated SWBD or NIR when the previous instruction has terminated, the MIR when the current instruction is not complete or when a test and skip or test and branch instruction evaluates true, or the remap logic when the first level of indirect for an indirect or indirect-augmented instruction is through a CR or VPR.

- State classes (bits 8 through 10)

The three state class bits (EX for execute, LA for look ahead, and LC for last cycle) provide a broad definition of the state of the current instruction in order for the PP three-level pipe to operate properly. The EX bit indicates the current instruction is in an execution state; the LA bit indicates the possiblility of indexing the next instruction on this step; the LC bit indicates that it is possible to terminate the current instruction on this execution cycle.

- Bit count (bits 11 through 13)

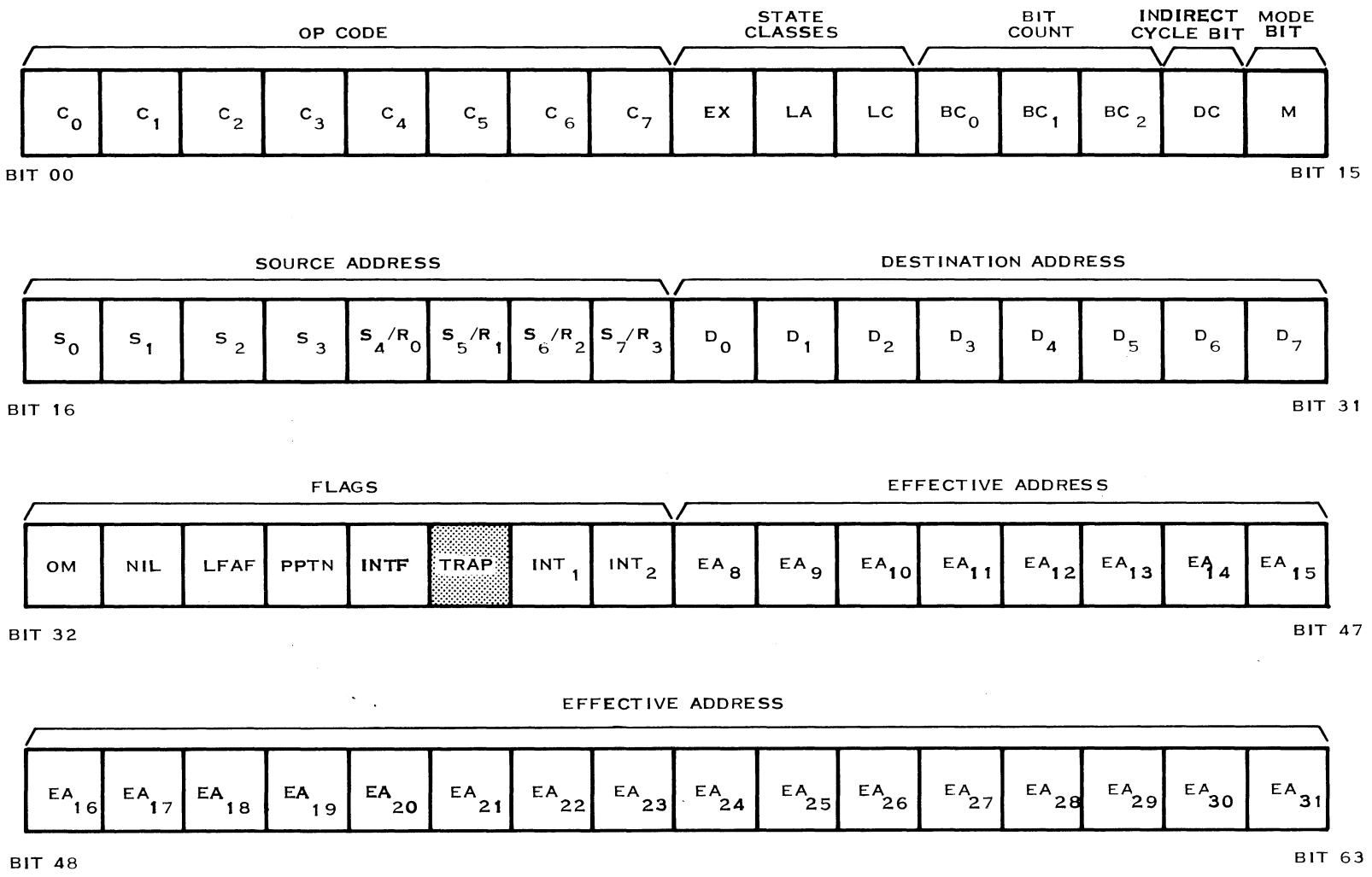


Figure 4-16. Instruction Register Format

The bit count (BC_0 is the MSB and BC_2 is the LSB) is a modifier of the state class and provides a counter for multiple steps in one state class.

- Indirect cycle bit (bit 14)

The indirect cycle bit (DC) is set to indicate that the current instruction in the IR is an indirect instruction.

- Mode bit (bit 15)

The mode bit (M) is set to indicate the source of the current instruction is Central Memory (CM) and reset to indicate Read Only Memory (ROM).

- Source address (bits 16 through 23)

The eight bit source address specifies a CR or VPR from which data is to be retrieved.

- Destination address (bits 24 through 31)

The eight bit destination address specifies a CR or VPR in which data is to be stored.

- Flags (bits 32 through 39)

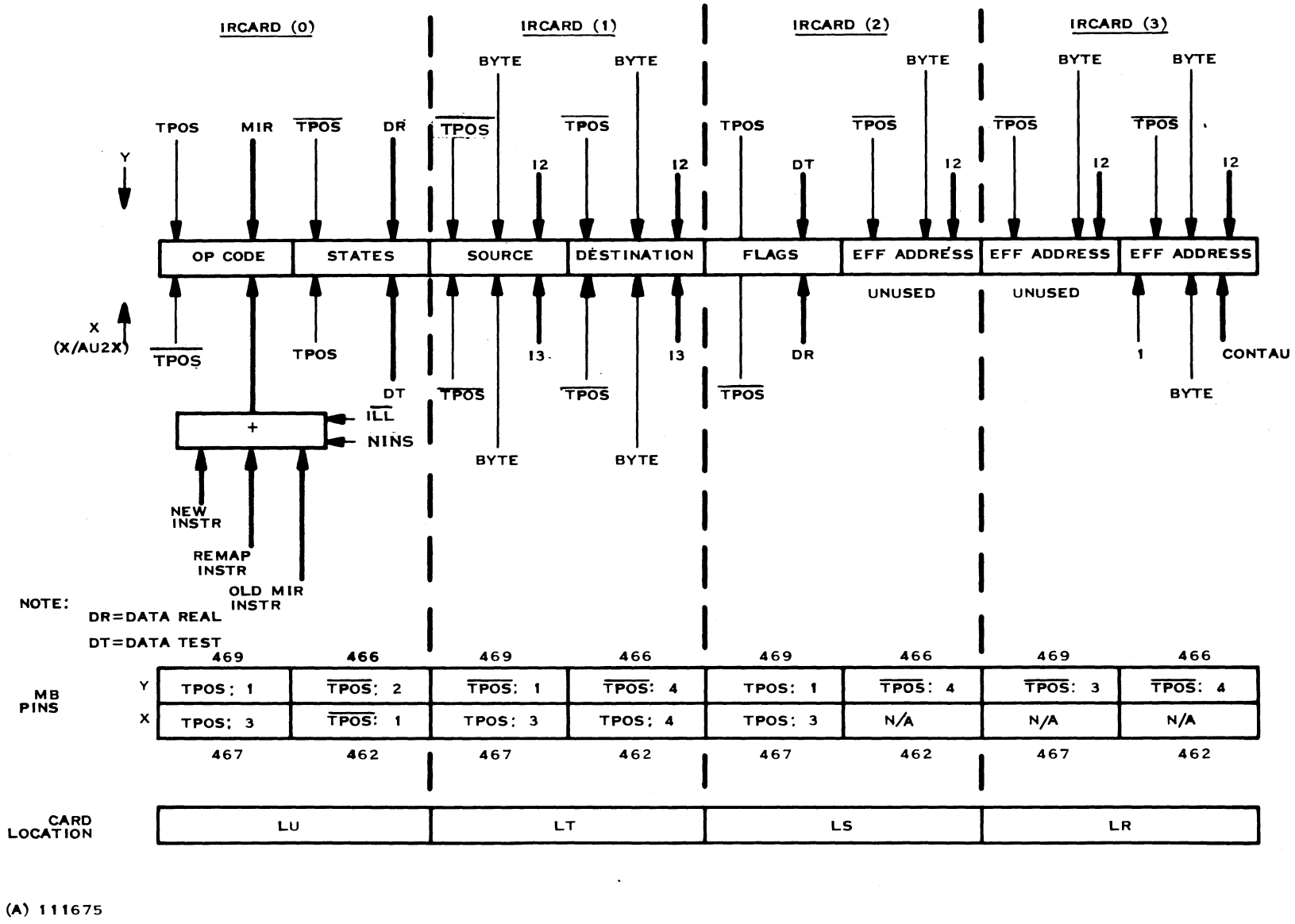
The object mode bit (OM) is the mode bit for the object instruction pointed to by an analyze instruction. The next instruction location (NIL) is set to indicate the next instruction is in the NIR, or reset to indicate the next instruction is in the SWBD. The LFAF flag is set when the current instruction is LDMB indirect and indicates the base value to be loaded is in CM rather than the VPR specified by the T and N fields. The PPTN flag is set when the current instruction is indirect through a CR or VPR specified by the T and N fields. The interrupt flag (INTF) is set at the last step of execution of the current instruction when an automatic or programmed interrupt occurs during the same instruction. The INTF flag is used to trap the VP with the interrupt to an interrupt routine in ROM. The TRAP flag is not used by the IR, and the INT1 flag is set when a programmed interrupt is initiated (a bit is set in the Interrupt Control byte of the CR file) by the executing VP. The INT2 flag is set (only in the IR of the selected VP) when any one of the following occurs: A/C power failure, activation of the STOP button on the Operator'sConsole, disc protect violation, CP interrupt, CM parity error in selected VP, CM protect violation in selected VP, illegal op-code in selected VP.

- Effective Address (bits 40 through 63)

The 24-bit effective address is developed by the TN field indexer (indexer I2) and indicates an immediate operand, a direct or indirect operand address, or a direct or indirect branch address. The only exception to this effective address development occurs when a shift instruction is executing and the current shift count update is supplied by the CONTAU card.

The loading logic and flip-flops for each of the eight IR's is distributed on four IRCARD cards (IRCARD(0) through IRCARD(33)), as shown in figure 4-17. The op-code and state class bits exist on IRCARD(0) and their source depends on the test positive signal from the CONTAU card. If a test and skip or test and branch instruction evaluates true (skip or branch taken), the test positive signal (TPOS) enables the op-code of the instruction (located in the MIR during the test cycle) and the data test states (DT) into the associated IR. These loading sources are required because the three-level pipe requires updating before the current instruction terminates. If a skip or branch is not taken, the inverted test positive signal ($\overline{\text{TPOS}}$) enables the op-code of a new instruction, remapped instruction, or old partially complete instruction and the data real states (DR) into the associated IR. The source and destination addresses exist on IRCARD(1) and they are supplied (as long as the $\overline{\text{TPOS}}$ signal is true) by the TN field indexer (indexer I2) or the register indexer (indexer I3), depending on the instruction being executed. The control flags and the first byte of the effective address exist on IRCARD(2). If the TPOS signal is true, the DT control flags are inserted in the IR; if the $\overline{\text{TPOS}}$ signal is true, the DR control flags are inserted in the IR. The first byte of the effective address is supplied by indexer I2 when the $\overline{\text{TPOS}}$ signal is true. The second and third bytes of the effective address exist on IRCARD(3) and are supplied by indexer I2 when the $\overline{\text{TPOS}}$ signal is true. When a shift instruction is in the process of shifting data in the AU and the desired shift count has not been completed, the selected shift increment (1, 4, 8, or 16) is inserted in the third byte of the effective address by the CONTAU card. A more detailed description of the loading and distribution logic on the IRCARD cards is presented in the following paragraphs.

4-81 IRCARD(0) Loading and Distribution Logic. Refer to figure 4-18 for a simplified logic diagram of the IR op-code and state loading and distribution logic. The active VP code lines (¬PILVPC(0-2)) from the MLCTL card are temporarily stored in a group of three flip-flops on IRCARD(0) during every execution cycle. The true outputs of the flip-flops are directly applied to a DE module for decoding purposes and the complement outputs are inverted by a group of 2N logic modules (during normal PP processing the maintenance logic holds one of the two 2N module inputs at one) before being applied to a second DE module. At the conclusion of the current execution cycle, the MIR op-code lines (¬PICDT(000-007)) from PPCTL2 are applied to the test positive inputs of the IR flip-flops. At this same time, the new op-code from the associated SWBD or NIR, the old op-code from an instruction currently in the MIR, or the remapped op-code from the remapping logic, is supplied by PPCTL2 (¬PICDR(000-007)) and inverted twice (once due to the inhibited maintenance logic). The result is applied to the test positive complement inputs of the IR flip-flops. If the test positive signal (PACTSPOS) developed by the skip taken and branch taken logic of the AU indicates an instruction is to be skipped, or control is to branch out of the



(A) 111675

Figure 4-17. Instruction Register Loading Format

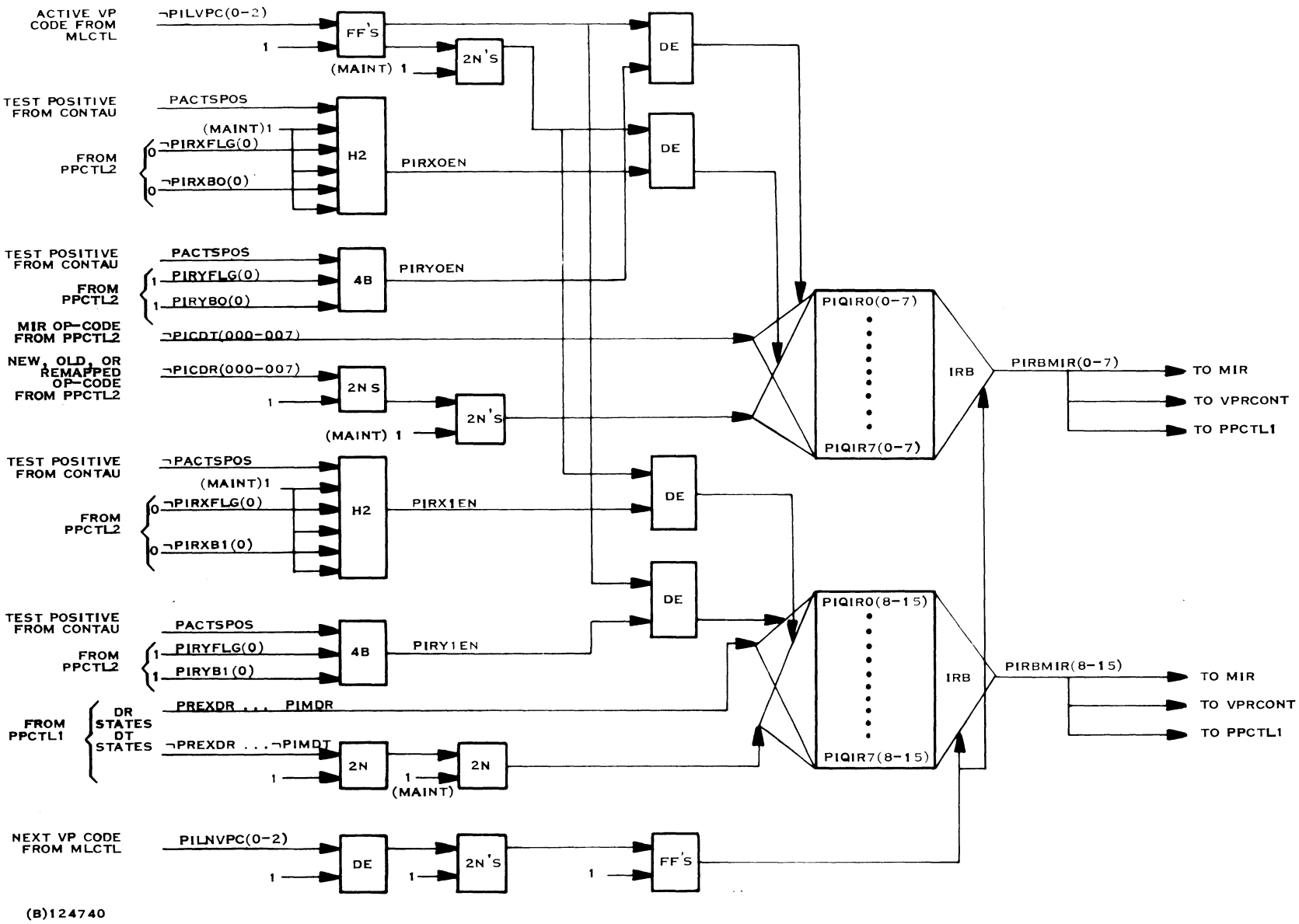


Figure 4-18. Instruction Register (IRCARD(0))

current instruction stream, the PACTSPOS signal, combined with the PIRYFLG(0) and PIRYBO(0) signals (these last two signals are hard wired on PPCTL2 at a logic one for IRCARD(0)), develop the PIRYOEN signal. The PIRYOEN signal enables the decoding of the VP code in order to generate a gate for the MIR op-code applied to the test positive inputs of the IR's. The net result is insertion of the MIR op-code in the IR of the active VP at the conclusion of the execution cycle. If the test positive signal indicates no skip or branch is to be taken, the PACTSPOS signal combines with the PPCTL2 hard-wired signals ¬PIRXFLG(0) and ¬PIRXBO(0) to develop the PIRXOEN signal. The PIRXOEN signal enables the decoding of the VP code and the resulting insertion of the new, old, or remapped op-code in the IR of the active VP.

The state bits (EX, LA, LC, BC_0, BC_1, BC_2, DC, and M) are inserted in the second byte of the IR of the active VP at the same time and in a manner similar to that described for the op-code. If the inverted test positive signal (¬PACTSPOS) indicates a skip or branch is to be taken, the PIRX1EN signal is developed to enable the decoding of the active VP code (gate generation) and to permit the insertion of the data test states (¬PREXDT, PRLADT, PRLCDT, ¬PRBCODT, ¬PRBC1DT, PRBC2DT, ¬PIDCDT, and ¬PIMDT) in the IR of the active VP. If the inverted test positive signal indicates no skip or branch, the PIRY1EN signal is developed to generate the gate used in inserting the data real states (¬PREXDR, PRLADE, PRLCDR, ¬PRBCODR, ¬PRBC1DR, PRBC2DR, ¬PIDCDR, and ¬PIMDR) in the IR of the active VP.

The next VP code (PILNVPC(0-2)) from the MLCTL card is decoded and stored in a group of eight flip-flops at the conclusion of an execution cycle. The true output of the flip-flop set, due to the decoding process, is used to enable the op-code and states of the IR associated with the active VP over the IRB bus at the beginning of the next execution cycle. The enabled data (PIRBMIR(0-15)) is output to the MIR, the VPRCONT card, and the PPCTL1 card.

4-82 IRCARD(1) Loading and Distribution Logic. The IR loading and distribution logic on IRCARD(1) is identical to that on IRCARD(0), however, the data handled and the loading control signals differ as shown in figure 4-19. Source data from indexer I2 or indexer I3 may be loaded in the first byte of IRCARD(1) and destination data from one of the same two sources may be loaded in the second byte. In order for any data to be loaded in either byte, the test positive signal must indicate no skip or branch is to be taken (¬PACTSPOS is true). When this is the case and the source address is to be supplied by indexer I2, the I2 enable signal (PIRYBO(1)) from PPCTL2 permits the development of the PIRYOEN signal. The PIRYOEN signal enables the decoding of the VP code and the resulting insertion of the I2 source address (¬PTI2RES(024-031)) in the IR of the active VP. When the source

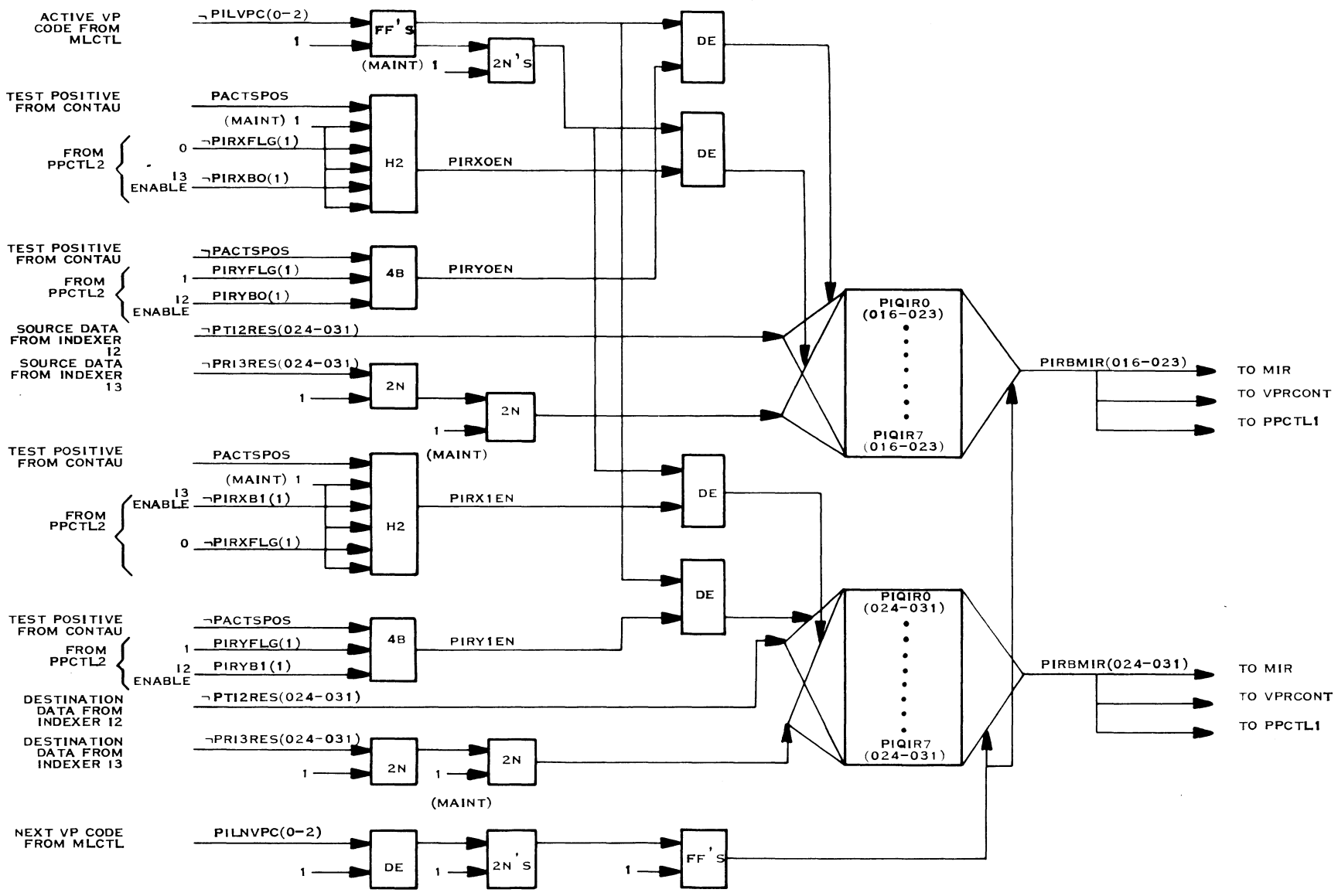


Figure 4-19. Instruction Register (IRCARD(1))

address is to be supplied by indexer I3, the I3 enable signal (¬PIRXBO(1)) develops the PIRXOEN signal, which in turn enables the I3 source address (¬PRI3RES(024-031)) into the IR of the active VP. The source of the destination address is determined in a similar manner, except the PIRYB1(1) signal provides the I2 enable and the ¬PIRXB1(1) signal provides the I3 enable. The source and destination addresses (PIRBMIR(016-031)) and the op-code and states on IRCARD(0) are distributed simultaneously.

4-83 IRCARD(2) Loading and Distribution Logic. The control flags and first byte of effective address loading and distribution logic on IRCARD(2) (the logic is again identical to that on IRCARD(0)) is shown in figure 4-20. If the test positive signal is true, the PIRYOEN signal is developed in order to enable the data test flags (¬PIOMDT, ¬PINILDT, ¬PILFAFDT, ¬PIPPTNDT, ¬PIINTFDT, ¬PINT1DT, and ¬PINT2DT) from PPCTL2 into the IR of the active VP. If the inverted test positive signal is true, the PIRXOEN signal is developed to enable the data real flags (¬PIOMDR, ¬PINILDR, ¬PILFAFDR, ¬PIPPTNDR, ¬PIINTFDR, ¬PINT1DR, and ¬PINT2DR) from PPCTL2 into the IR of the active VP. If the inverted test positive signal is true and PPCTL2 activates the indexer I2 enable signal (PIRYB1(2)), the PIRY1EN signal is developed to enable the first byte of the effective address generated by indexer I2(¬PTI2RES(008-015)) into the IR of the active VP. The PIRX1EN signal is held to a logic zero at all times by the ¬PIRXB1(2) signal from PPCTL2. (This disables all gates associated with one of the two inputs to the dual flip-flops composing the second byte on IRCARD(2).) The control flags, the first byte of the effective address (PIRBMIR(032-047)), and the IR data on IRCARD(0) and IRCARD(1) are distributed simultaneously.

4-84 IRCARD(3) Loading and Distribution Logic. The second and third bytes of effective address loading and distribution logic on IRCARD(3) (the logic is identical to that on the other IRCARD cards) are shown in figure 4-21. If the inverted test positive signal is true and PPCTL2 activates the indexer I2 enable signals (PIRYB0(3) and PIRYB1(3)), the PIRYOEN signal is developed to enable the second byte of the effective address from I2 (¬PTI2RES(016-023)) into the IR of the active VP, and the PIRY1EN signal is developed to enable the third byte of the effective address from I2 (¬PTI2RES(024-031)) into the same IR. The PIRX1EN signal is held to a logic zero at all times by the ¬PIRXB0(3) signal from PPCTL2, so one set of the inputs to the dual flip-flops composing the first byte on IRCARD(3) is not used. When the shift count needs updating during a shift instruction, the ¬PIRXB1(3) signal from PPCTL2 is used to drive the PIRX1EN signal to a logic one, so a new shift count (¬PACSHOB(000-005)) from CONTAU is inserted in the IR of the active VP. The second and third bytes of the effective address (PIRBMIR(048-063)) and the IR data on the other IRCARD cards are distributed simultaneously.

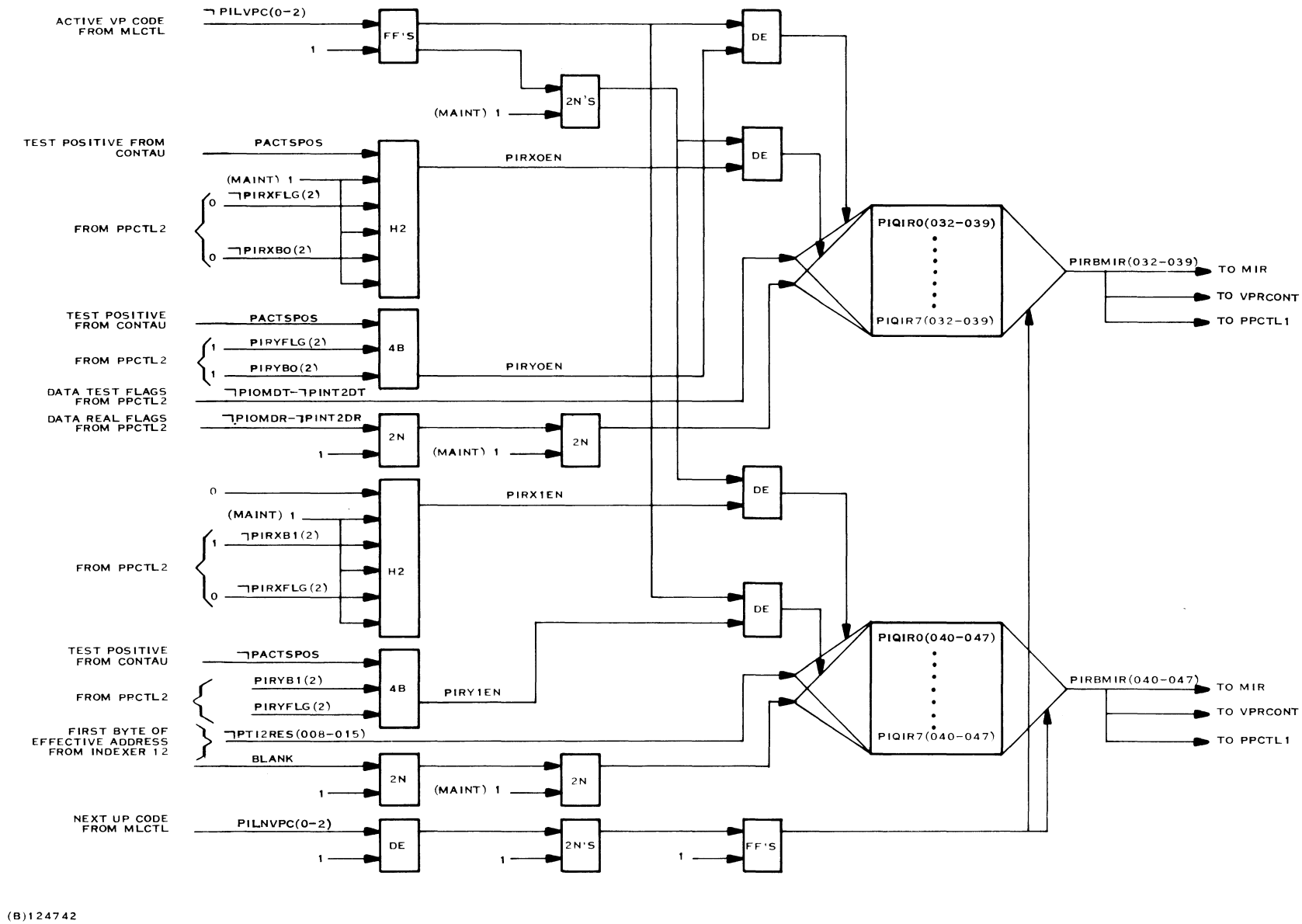


Figure 4-20. Instruction Register (IRCARD(2))

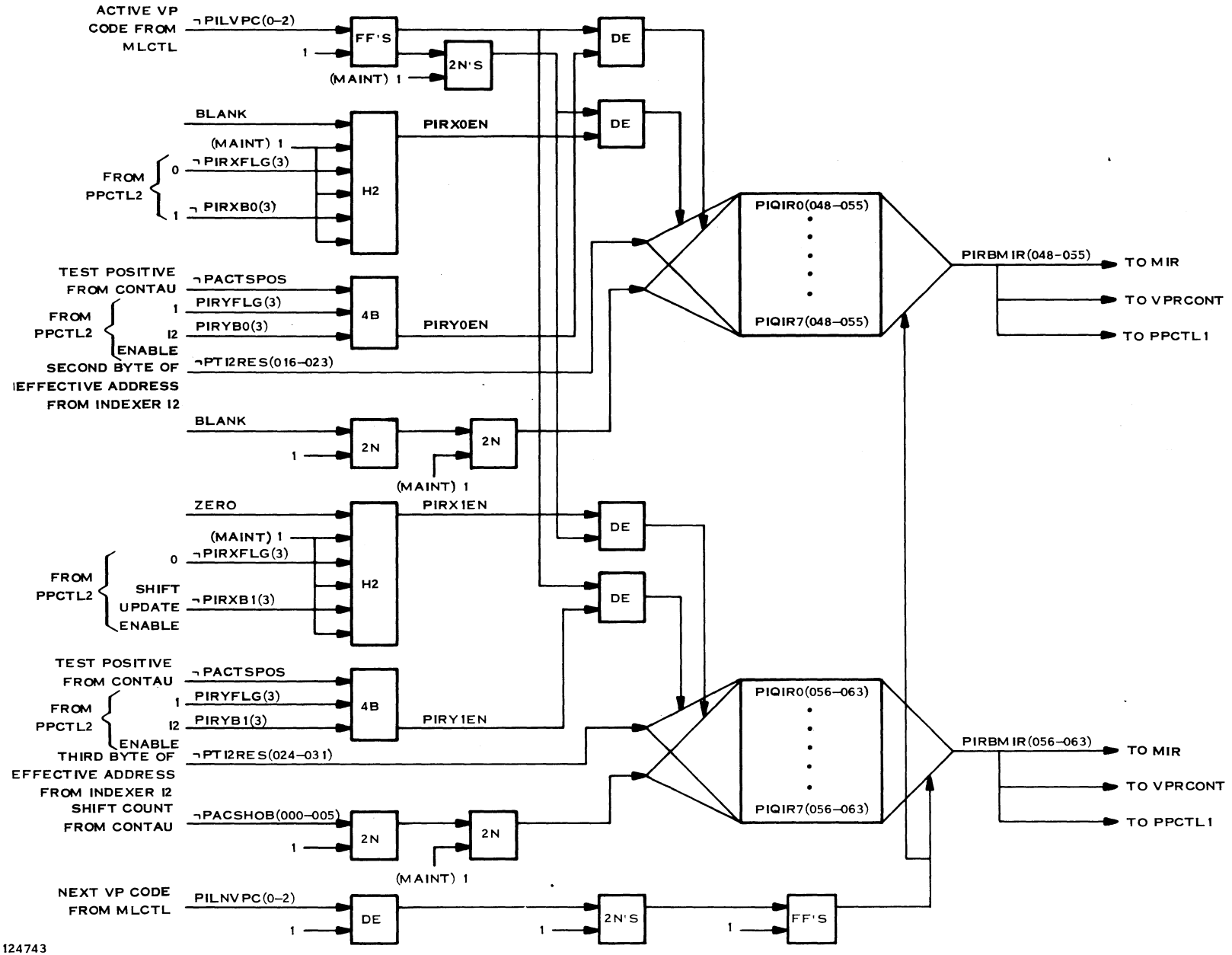


Figure 4-21. Instruction Register (IRCARD(3))

4-85 VIRTUAL PROCESSOR REGISTER FILE. Refer to figure 4-22 for a simplified block diagram of the eight VPR files and the associated input and output logic. Each of the eight VPR files provides the associated VP with four 32-bit general accumulator registers that can be addressed to the byte, halfword, or word level. Each VPR file accepts data from the AU and distributes data to Indexer I2, Indexer I3, and the AU.

4-86 VPR Loading Logic. Data is input to a VPR of the active VPR file via the AU1B or AU2B transfer buses of the AU. When the source of data is the AU1B bus, the appropriate AU1B enable signals (PUWA1P01(0-3) for VP0 and VP1, PUWA1P23(0-3) for VP2 and VP3, PUWA1P45(0-3) for VP4 and VP5, or PUWA1P67(0-3) for VP6 and VP7) from the VPRCONT card enable the decoding of the AU1B word code (PUAW1C(0-2), the word code points to one VPR out of two VPR files) and determine what portion of the destination VPR (byte, halfword, or word) is to be used. The decoded pointer is used to direct the AU1B data (¬PAU10(0-31)) to the destination VPR and the active AU1B enable signals are used to determine what part of the destination VPR is to be filled with new data. When the source of data is the AU2B bus, the appropriate AU2B enable signals (PUWA2P01(0-3) for VP0 and VP1, PUWA2P23(0-3) for VP2 and VP3, PUWA2P45(0-3) for VP4 and VP5, or PUWA2P67(0-3) for VP6 and VP7) from the VPRCONT card enable decoding of the AU2B word code (PUWA2C(0-2)) so that the AU2B data (¬PAU20(0-31)) is inserted in the intended portion of the destination VPR (this processing parallels that for the AU1B bus).

4-87 VPR Distribution logic. The selected VPR data is distributed to the AU over the VPAB bus in combination with the MDB or over the VPRB1 and VPRB2 buses. When VPR data is to be transferred to the AU by way of the MDB, the MDB enable signal (PURABP01 for VP0 and VP1, PURABP23 for VP2 and VP3, PURABP45 for VP4 and VP5, or PURABP67 for VP6 and VP7) from the VPRCONT card enables the decoding of the MDB word code (PURABC(0-2)). The resulting pointer enables one VPR of two VPR files over the VPAB bus to the MDB (¬PUPOR1AB(0-31) for VP0 and VP1, ¬PUP2R3AB(0-31) for VP2 and VP3, ¬PUP4R5AB(0-31) for VP4 and VP5, or ¬PUP6R7AB(0-31) for VP6 and VP7). When VPR data is to be transferred to the AU by way of the VPRB1 and VPRB2 buses, the VPRB1 enable signal (PURA1P01 for VP0 and VP1, PURA1P23 for VP2 and VP3, PURA1P45 for VP4 and VP5, or PURA1P67 for VP6 and VP7) from the VPRCONT card enables the decoding of the VPRB1 word code (PURA1C(0-2)). The resulting pointer enables one VPR of two VPR files over the VPRB1 bus to the VPRB2 bus. The VPR input to the VPRB2 bus is inverted by a group of 4B logic modules and then routed to the AU (PUVPRA1(0-31)).

The VPR file of the active VP is transferred to Indexer I2 via the VPIB1 bus and VPR3 of the active VP is transferred to Indexer I3 via the ΔVPB bus.

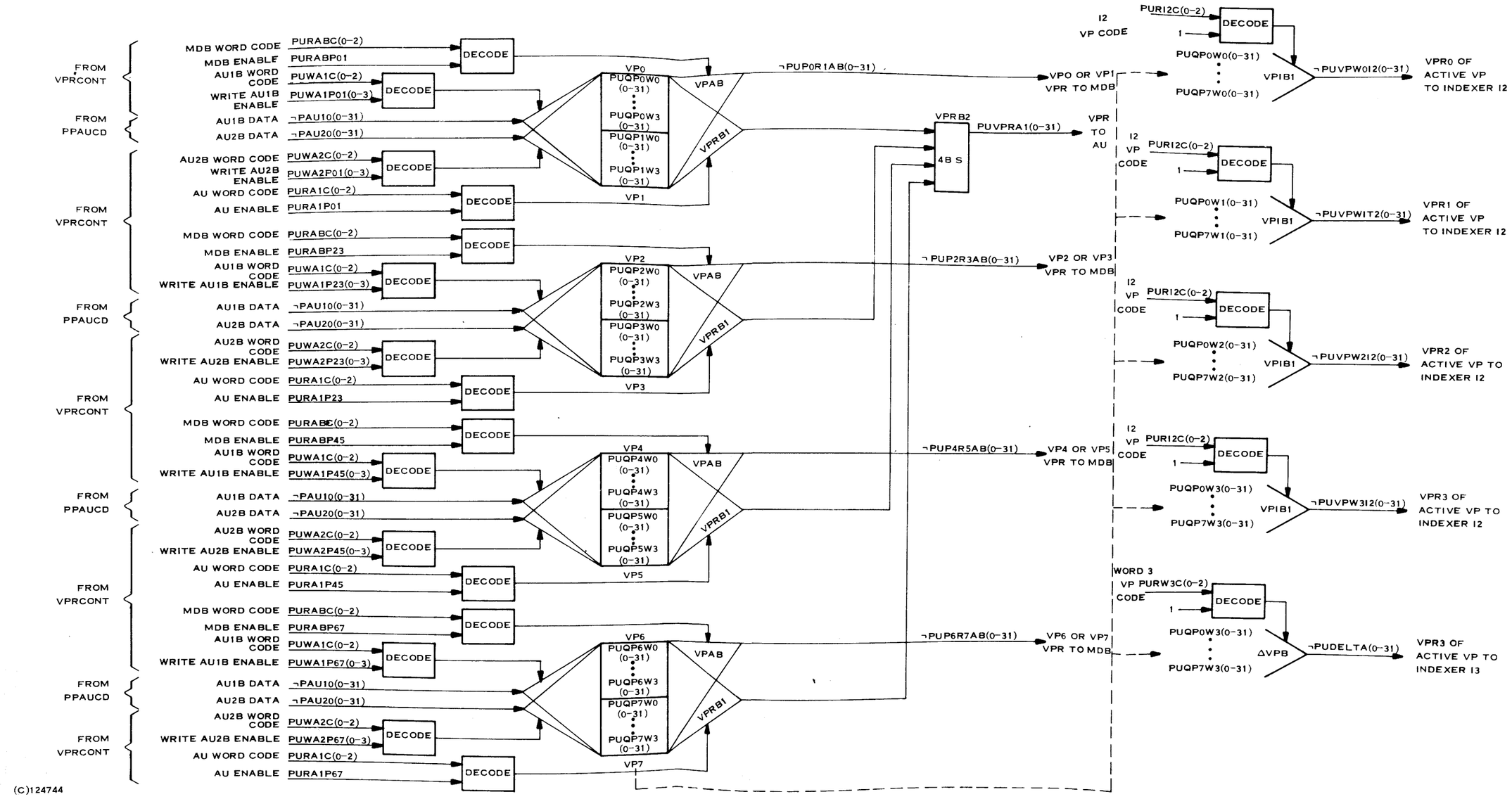


Figure 4-22. Virtual Processor Register Files

During the execution of all instructions, the VPR to Indexer I2 VP code (PURI2C(0-2), same as the active VP code) is decoded and used to enable the four VPR's of the active VP over the VPIB1 bus to the VPIB2 bus on the PCCARDA(0-7) cards. The middle two bits of the T field of the instruction being executed are used to enable one of the four VPR's over the VPIB2 bus to Indexer I2. The VPR3 VP code (PURW3C(0-2), same as the active VP code) is also continuously decoded and used to enable VPR3 of the active VP over the ΔVPB bus to Indexer I3. Indexer I3 uses only byte three of VPR3, therefore control at the motherboard level is necessary because all VPRCARD cards are identical.

4-88 CENTRAL MEMORY BASE REGISTER. Refer to figure 4-23 for a simplified block diagram of the eight CM base registers and the associated input and output logic. Each of the eight CM base registers provides the associated VP with 24 bits of temporary storage used to hold a base value for base relative instructions. The CM base register accepts data from the AU and outputs data to Indexer I2.

4-89 CM Base Register Loading Logic. Data is input to a CM base register via the AU2B bus of the AU in byte or multiple byte increments. When data from the AU is to be written to a CM base register, three signals are necessary from the CRCONT card. These signals include a write card zero (PCWA2B1E(0)), which designates the CM base registers of the CR file; a word select (¬PCWA2CB1(3-5)), which selects one of the eight CM base registers; and a write right (PCWA2CR(1-3)) or left (PCWA2CL(1-3)) hex, which is used to select the right or left half of bytes one, two and three of the selected CM base register. In addition, the R field mask signals (PCRFDMSK(0-3)) from the CONTAU card provide additional control on data written to a CM base register when a test and set (TS), test and reset (TR), set (S), or reset (R) instruction is being executed. When a load central memory base (LDMB) instruction is executed, all three bytes of the base register associated with the active VP are loaded simultaneously. This is accomplished by forcing the right and left hex enables and the R field mask to all ones. When a TS, TR, S, or R type instruction is being executed and data is to be written to the right half of the first byte of a CM base register, the write card zero and write right hex of CRBASE1 signals combine to enable the decoding of the word select signals. The result of the decode is masked by the R field so the AU2B data (¬PAU20(12-15)) marked by ones in the R field is inserted in the right half of the first byte of the intended CM base register. When AU2B data is to be written to the left half of the first byte of a CM base register, the write left hex of CRBASE1 signal replaces the write right hex of CRBASE1 signal so that the net result is data storage in the left half of the first byte of the intended CM base register. Data is written to the second and third bytes (CRBASE2 and CRBASE3, respectively) of the intended CM base register in a similar manner via the PCWA2CR(2),

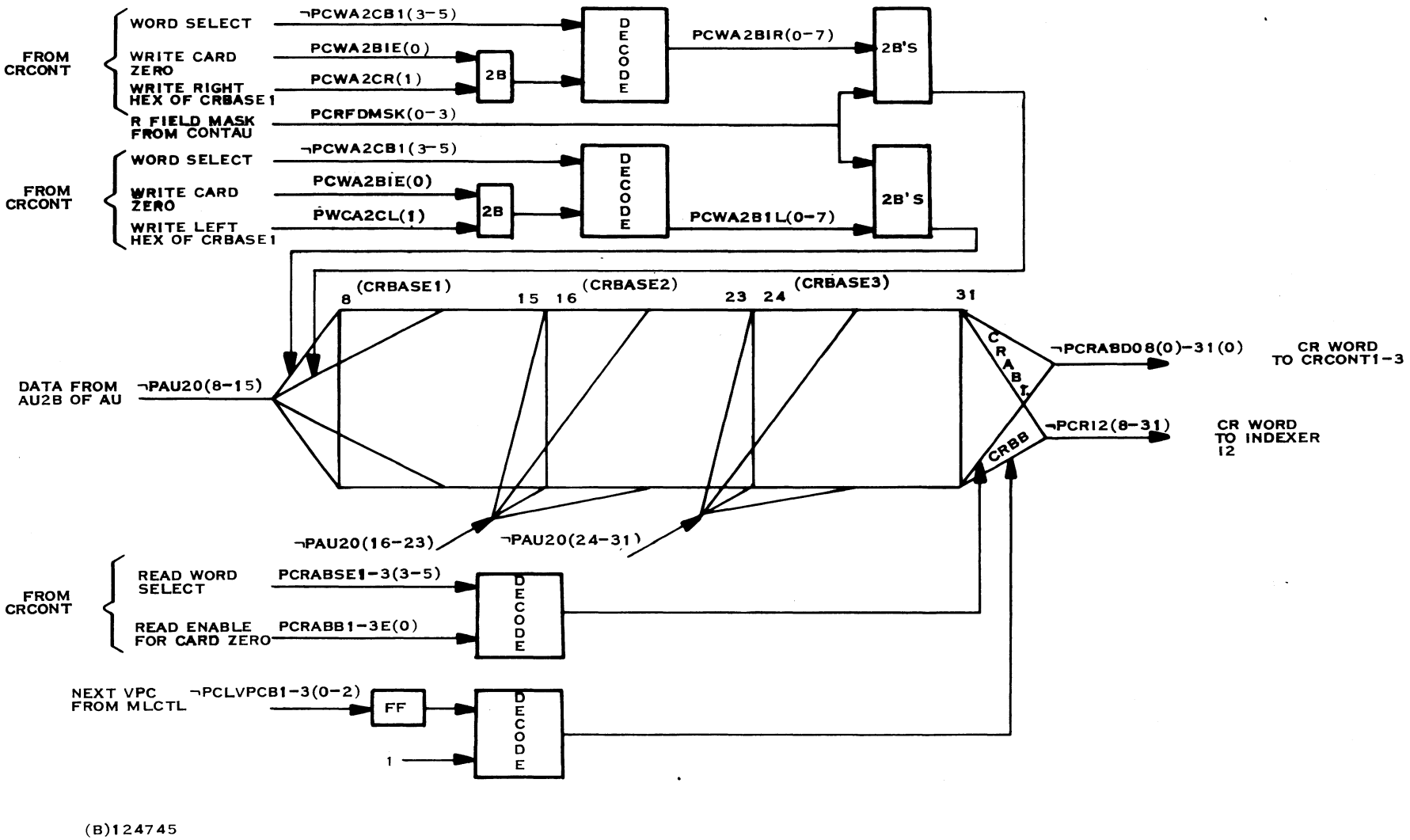


Figure 4-23. Central Memory Base Registers

PCWA2CL(2), PCWA2CR(3), and PCWA2CL(3) signals. When a group of data larger than a hex is under consideration (byte or halfword), the proper combination of hex controls enable the desired write operation.

4-90 CM Base Register Distribution Logic. CM base register data is read in 24-bit groups (the entire CM base register) and is distributed to both Indexer I2 over the CRBB bus and to the MDB (and eventually the AU) over the CRAB1 and CRAB2 buses. During the execution of all instructions, the VP code of the active VP is decoded and used to enable the selected CM base register over the CRBB bus to Indexer I2. Indexer I2 requires the CM base register of the active VP in case a base relative effective address needs to be developed. When CM base register data is required on the MDB, the read enable for card zero signals from CRCONT1, CRCONT2, and CRCONT3 (PCRABB1E(0), PCRABB2E(0), and PCRABB3E(0), respectively), enable the decoding of the read select signals (PCRABSE1(3-5) PCRABSE2(3-5) and PCRABSE3(3-5)) so that the selected CM base register is enabled over the CRAB1 bus to the CRAB2 bus and eventually to the MDB.

4-91 SINGLE WORD BUFFER ADDRESS REGISTER. Refer to figure 4-24 for a simplified block diagram of the eight SWBA's and the associated input

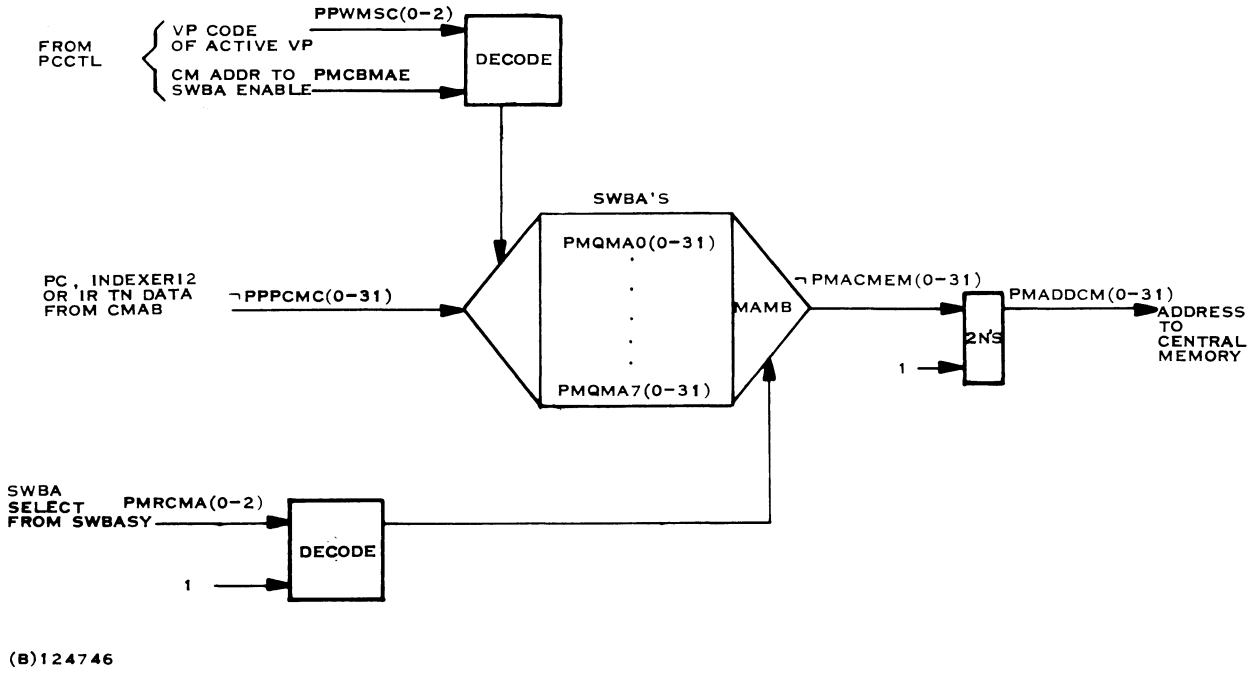


Figure 4-24. Single Word Buffer Address Registers

and output logic. Each of the eight SWBA's provides the associated VP with an address register that supplies Central Memory with an address when a read or write operation is to be performed. When PC, Indexer I2, or IR TN data (¬PPPCMC(0-31)) is to be written in a SWBA, the CM address to SWBA enable signal (PMCBMAE) from PCCTL permits the decoding of the active VP code in order to develop a pointer to the SWBA of the active VP. The developed pointer is then used to insert the source data into the proper SWBA. When a memory access request has been accepted by the MCU, the SWBA select signals (PMRCMA(0-2)) from the SWBASY card are decoded and used to enable the SWBA of the VP making the request over the MAMB bus. The selected address is inverted back to its true form and input to Central Memory.

4-92 SINGLE WORD BUFFER DATA REGISTER. Refer to figure 4-25 for a simplified block diagram of the eight SWBD's and the associated input and output logic. Each of the eight SWBD's provide the associated VP with a data register that is used to hold data to be written to or read from Central Memory. The SWBD's accept data from Central Memory when a read is performed and from the associated PC, SWBA, VPR file, or CM base register, via the MDB and AU2B bus, when a write is to be performed. The SWBD's distribute data to Indexer I2 via the MDIB bus, to the MDB via the MDAB bus, to the associated NIR via the CMDB bus, and to Central Memory via the TWB.

When a read request has been accepted by the MCU and read data is available to the PP, the SWBD load enable signal (PMWCMD) from the SWBASY card permits the decoding of the SWBD load select signals (PMWCMC(0-2)) so that the Central Memory data word is inserted in the SWBD of the VP making the read request. When PC, SWBA, VPR file, or CM base register data is to be written to Central Memory, the AU2B bus to SWBD enable signals (PMA2MDLE and PMA2MDRE for left half and right half, respectively) from the PCCTL card permit the decoding of the write AU2B bus to SWBD select signals (PPWA2C(0-2)). The resulting pointers are used to enable the MDB supplied data over the AU2B bus to the associated SWBD in halfword or word groups. During the execution of all instructions, the active VP code is continuously decoded and used to enable the associated SWBD over the MDIB bus to Indexer I2 for effective address development. When SWBD data is to be output to the MDB, the SWBD to MDB enable signal (PMMDABE) permits the decoding of the active VP code (PPRABC(0-2)) so that the associated SWBD is enabled over the MDAB bus to the MDB. When SWBD data is to be transferred to the associated NIR, the SWBD to NIR enable signal (PNMDCDE) permits the decoding of the active VP code (PPRMSC(0-2)) so that the SWBD is enabled over the CMDB bus to the NIR. When SWBD data is to be written to Central Memory, the TWB write selector (PMRCMD(0-2)) from SWBASY is decoded and used to enable the SWBD of the VP executing

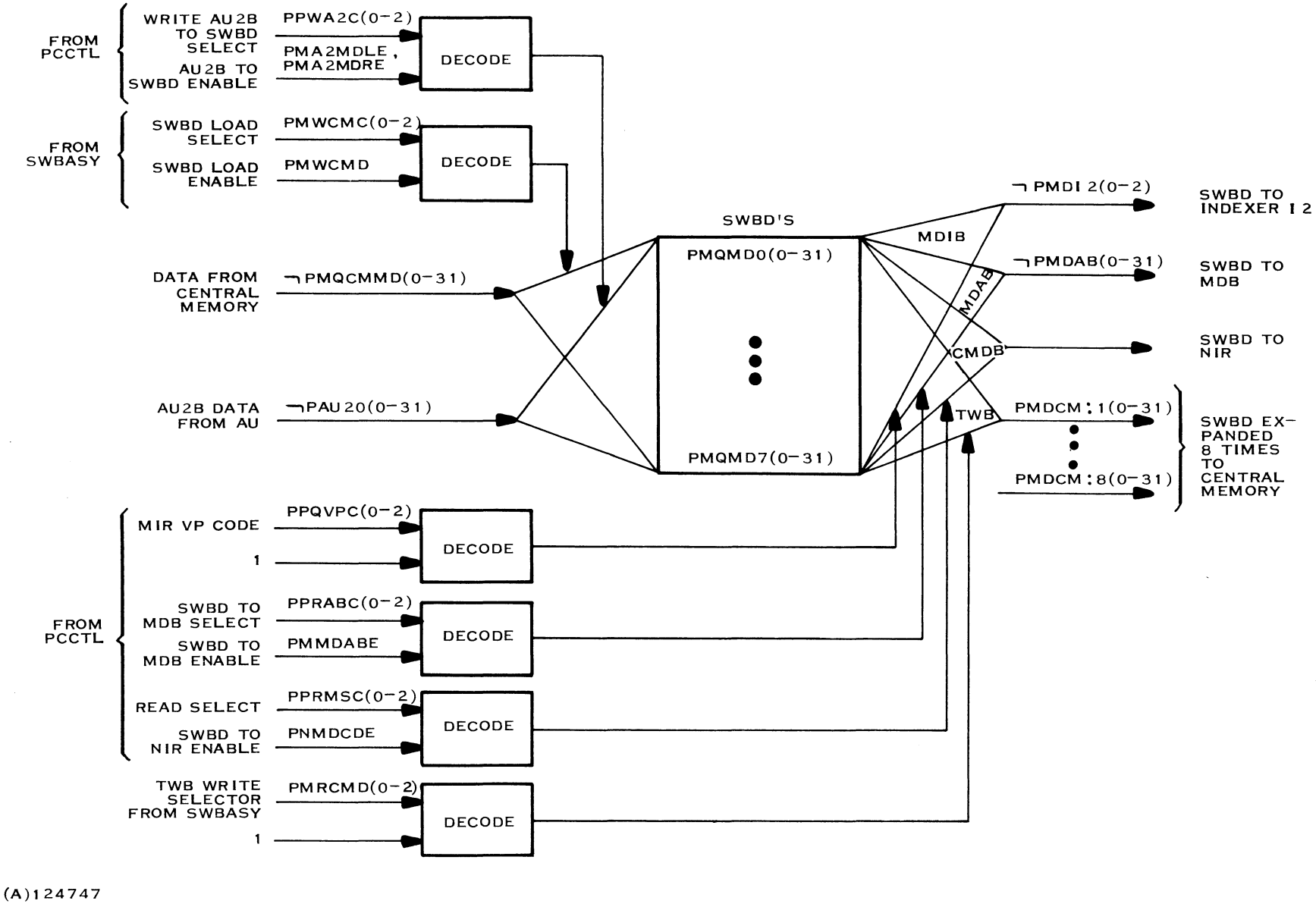


Figure 4-25. Single Word Buffer Data Registers

the write over the TWB to Central Memory. Refer to the description of the Single Word Buffer Controller TWB for additional information on the writing process.

4-93 ARITHMETIC UNIT

The arithmetic unit (AU) of the PP may be divided into the following major functions:

- Aligner
- Complement or constant generator
- Unload box
- Double rail generator
- Adder
- Shifter
- Bit picker
- Test box 1, 2, and 3 logic
- Comparator
- Data manipulator
- Skip taken and branch taken logic
- AU control

A detailed description supplemented with block diagrams, logic diagrams, and/or equations is provided for each of these functional areas in the following paragraphs.

4-94 ALIGNER. Refer to figure 4-26 for a simplified block diagram of the AU aligner. The aligner is used to perform a right end-around (cyclic) shift on a data word from a CR, VPR, the SWBD of the active VP, or the MIR effective address for immediate operands, in byte increments. The amount of shift is controlled by the AU control logic and can be zero, one, two, or three bytes, depending on the enabled control line. The selected source of input data to the aligner is supplied by the Main Data Bus (MDB) and is distributed to the aligner select logic in complemented byte groups. The select logic enables the appropriate byte of data to be output as determined by the enabled shift control line. The selected byte is routed to the AU2B transfer bus, the comparator, the shifter, and the complement or constant generator. The complemented selected byte is routed only to the complement or constant generator. The described data processing occurs simultaneously in each of the four PPAUCD cards so that the net result is a byte increment data word shift in true and complement form distributed throughout the AU. Refer to figure 4-27 for two examples of aligner data shifting.

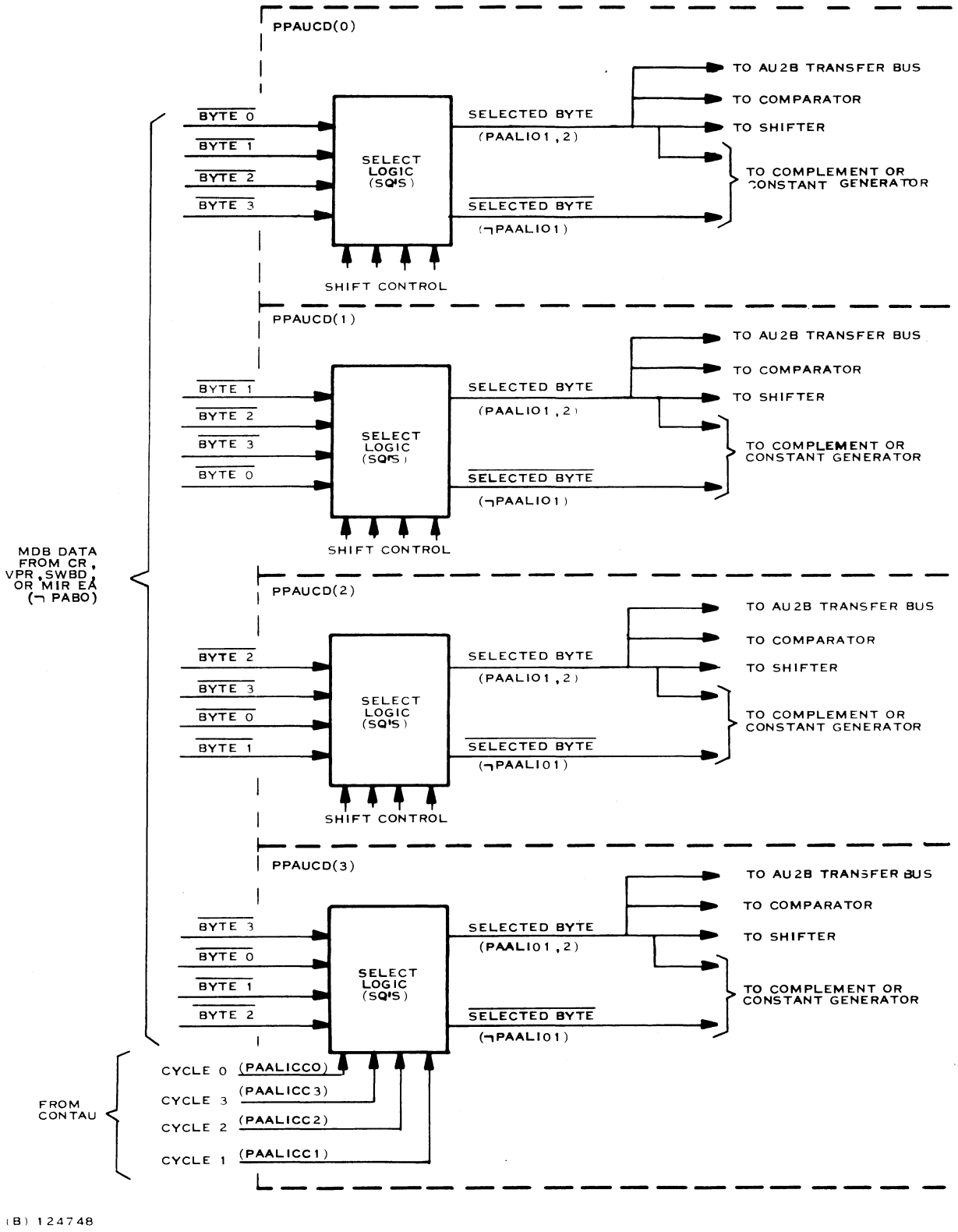
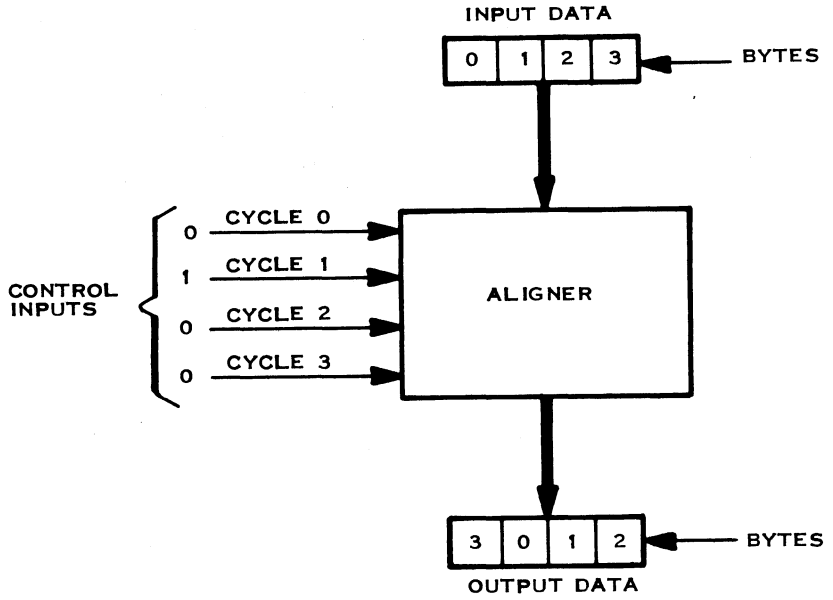
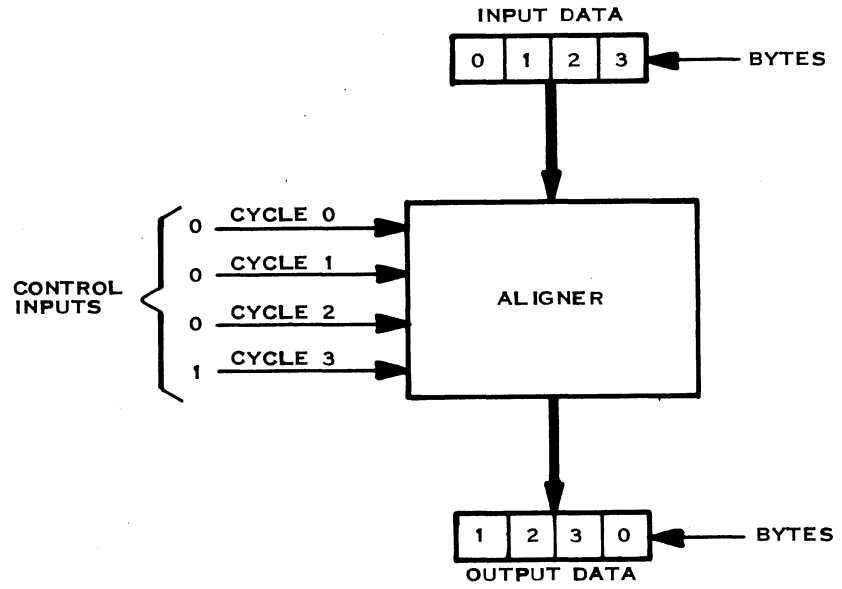


Figure 4-26. Aligner Logic on PPAUCD(0-3)



A. DATA ROTATED ONE BYTE



B. DATA ROTATED THREE BYTES

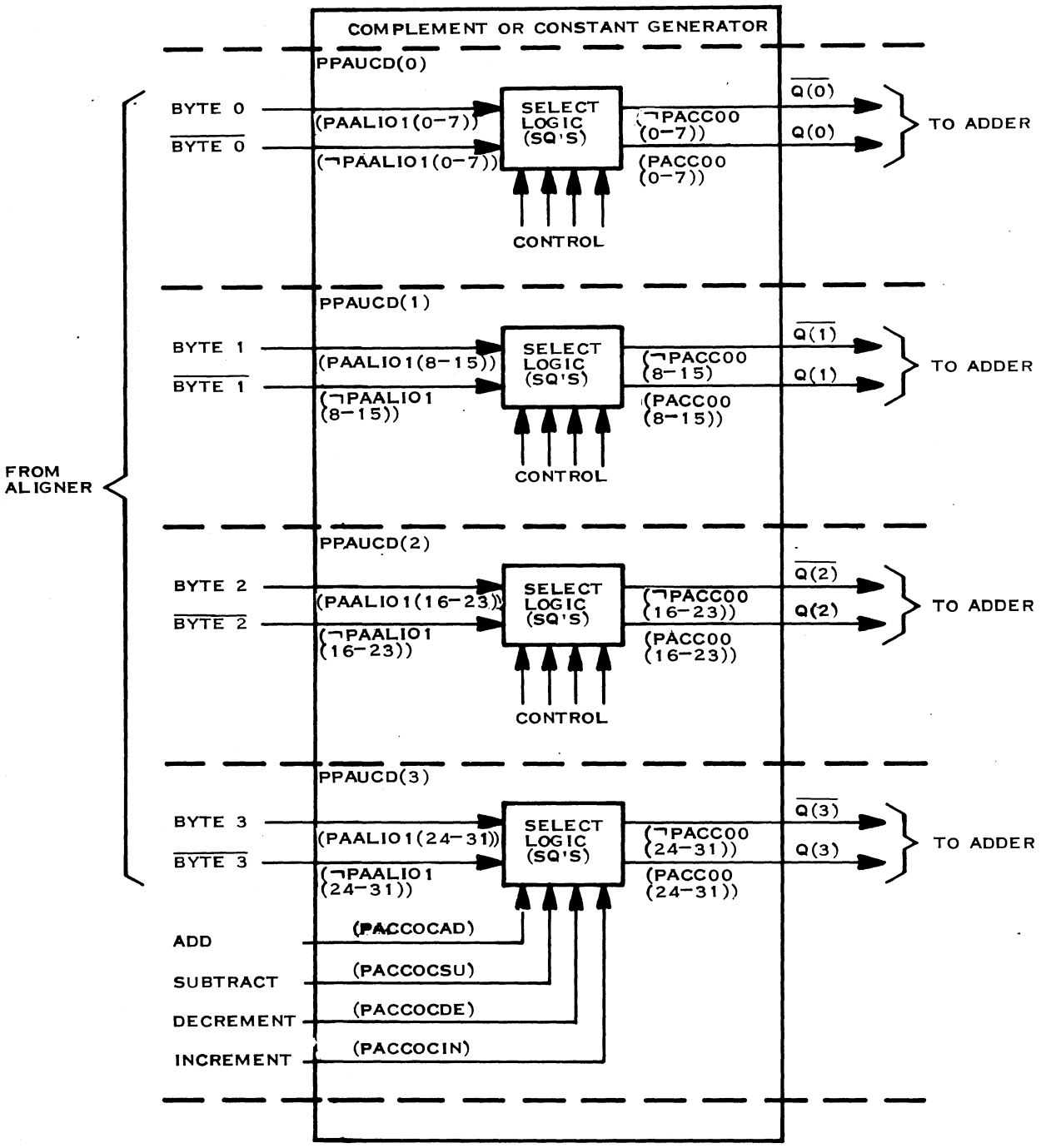
(A)124749

Figure 4-27. Aligner Byte Rotation

4-95 COMPLEMENT OR CONSTANT GENERATOR. Refer to figure 4-28 for a simplified block diagram of the complement or constant generator. The complement or constant generator accepts true and complement data from the aligner in byte groups and generates true and complement data for use in the adder when addition or subtraction is to be performed. When the adder is to be used for the incrementing and test or decrementing and test instructions, the complement or constant generator supplies plus or minus one, respectively, to the adder. The function of the complement or constant generator is controlled by the AU control logic via the add, subtract, increment, and decrement lines. The true and complement data supplied by the aligner is distributed to the select logic in byte groups. The control lines then enable the true form of the input data if an add instruction is being executed or the complement form of the input data if a subtract instruction is being executed. When a decrement and test instruction is being executed, the select logic and decrement control lines are used to generate the quantity $FFFFFFFF_{16}$ at the true output of the select logic on the word level. When an increment and test instruction is being executed, the select logic and increment control lines are used to generate the quantity 00010001_{16} at the true output of the select logic on the word level. The output data from the complement or constant generator is input to the first level of the adder. Operation of the complement or constant generator occurs simultaneously on each of the four PPAUCD cards (one byte per card) so that the net result is a 32-bit word output to the adder.

4-96 UNLOAD BOX. Refer to figure 4-29 for a simplified block diagram of the unload box. The unload box is used to develop the true and complement form of the VP code from the maintenance logic, the R field from the MIR, and the data word from a CR, VPR, the SWBD of the active VP, or the MIR effective address for immediate operand instructions. The 3-bit VP code from the maintenance logic is decoded into 8 bits (one bit per VP) and each of the 8 bits is developed in the true and complement form. The resulting VP code data is distributed to the data manipulator and test box 2. The 4-bit R field from the MIR is also developed in the true and complement form for use in the data manipulator and test box 2. Each of the four PPAUCD cards are operating on the same VP code and R field simultaneously to implement the byte bit-slice partitioning technique in the AU. The input data word to the unload box is supplied by the MDB in complement form. The unload box develops the true and complement form for each byte and then distributes the results to the shifter, bit picker, data manipulator, test box 1, and test box 2. Each of the PPAUCD cards are operating on a byte of the input data word simultaneously, so that the net result is a 32-bit data word.

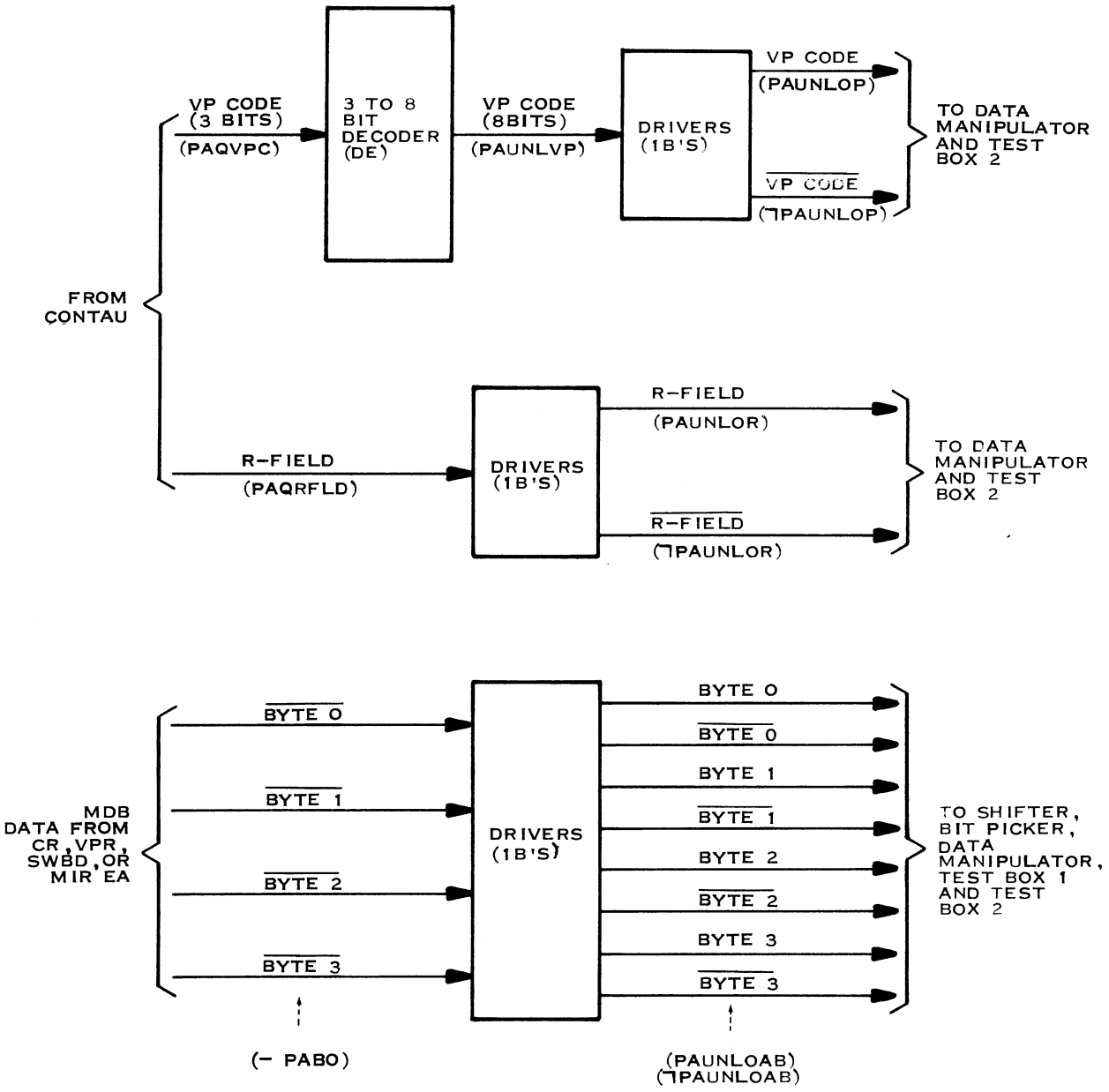
4-97 DOUBLE RAIL GENERATOR. The double rail generator is used to develop the true and complement form of the selected data word from the



WHERE, Q(N) = BYTE N FOR ADD, $\overline{\text{BYTE N}}$ FOR SUBTRACT, FF_{16} FOR DECREMENT
Q(0) AND Q(2) = 00_{16}, Q(1) AND Q(3) = 01_{16} FOR INCREMENT

(A)124750

Figure 4-28. Complement or Constant Generator



(A)124751

Figure 4-29. Unload Box

active VPR file. The data word from the selected VPR is input in byte groups to the double rail generator via the VPRB bus. A series of 1B logic modules provides the necessary drive and true and complement data for use in the adder, the comparator, and test box 3. The four bytes of input data are operated on in parallel by the four PPAUCD cards in order to supply a 32-bit word.

4-98 ADDER. Refer to figure 4-30 for a simplified block diagram of the five-level look-ahead adder. The adder performs addition, subtraction, and logical functions (AND, OR, EXCLUSIVE OR, and EQUIVALENCE) on 32-bit data words from the complement or constant generator and the double rail generator. The op-code from the MIR may specify adder operation to the byte, halfword, or word level. The following paragraphs provide a detailed description of each level in the five-level adder, and figure 4-31 presents a summary of each level.

4-99 Adder Level 1. Refer to figure 4-32 for the logic module connections required to implement the equations necessary for operation of adder level 1. The first level of the adder is used to develop the carry generated functions (D_i and $\overline{D}_i$), the carry propagate enables (T_i and $\overline{T}_i$), and the logical instruction outputs (LOG_i). Each PPAUDC card develops these signals for one byte of input data. A carry is generated (developed) whenever the two bits being added are both one. This fact results in the following equations:

$$D_i = a_i \cdot b_i$$

$$\overline{D}_i = \overline{a_i \cdot b_i} = \overline{a}_i + \overline{b}_i$$

where

a_i = the quantity at bit position i of the complement or constant generator output

b_i = the quantity at bit position i of the double rail generator output

A carry propagate (transfer) enable is generated whenever one of the two bits being added is one. This results in the following equations:

$$T_i = a_i \cdot \overline{b}_i + \overline{a}_i \cdot b_i$$

$$\overline{T}_i = a_i \cdot b_i + \overline{a}_i \cdot \overline{b}_i$$

An exception to the developed equations occurs for halfword instructions because the carry in the most significant bit of the right halfword must be disabled. This is accomplished with the following carry transfer (T_o) and carry develop (D_o) equations for the most significant bit of the right halfword.

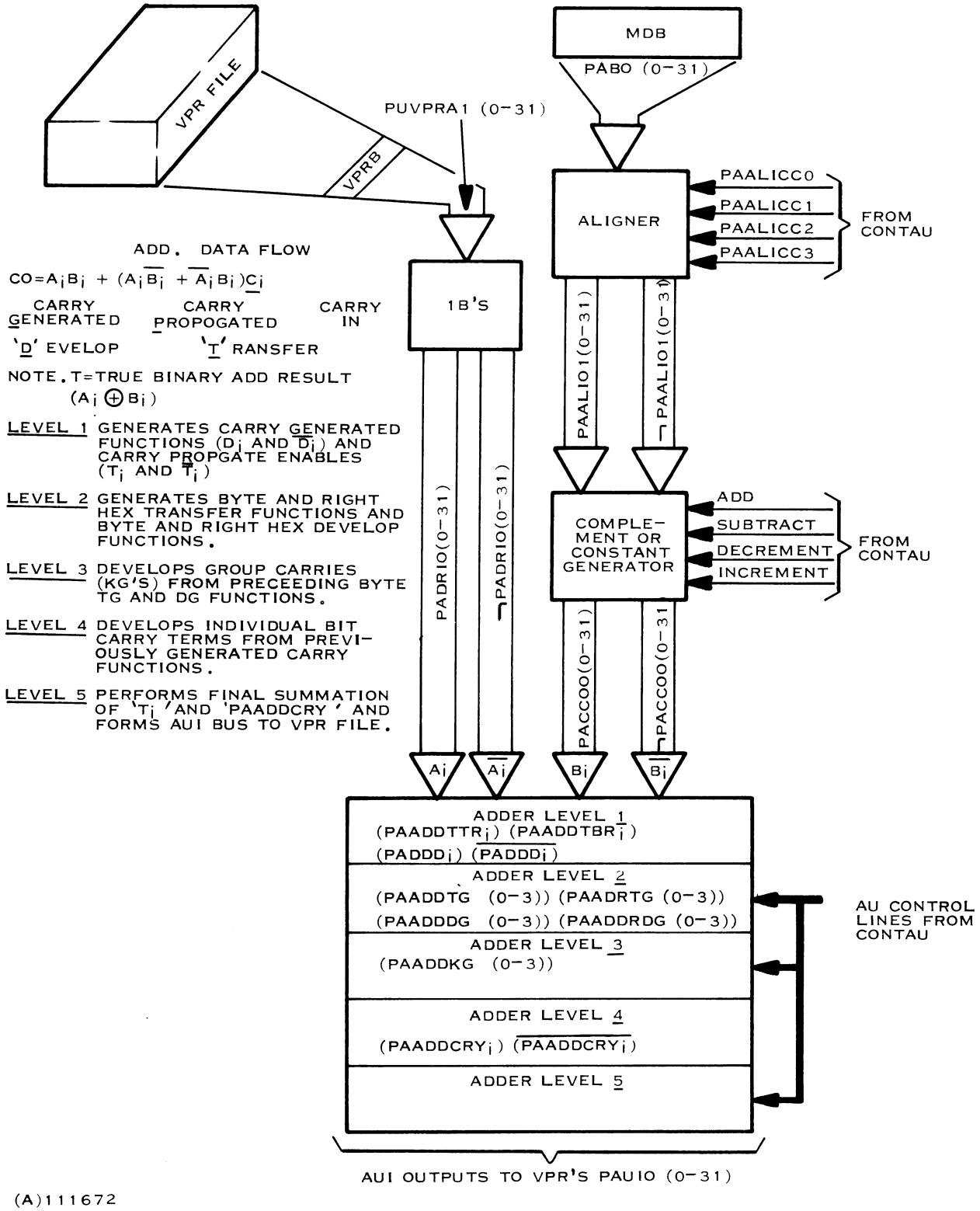
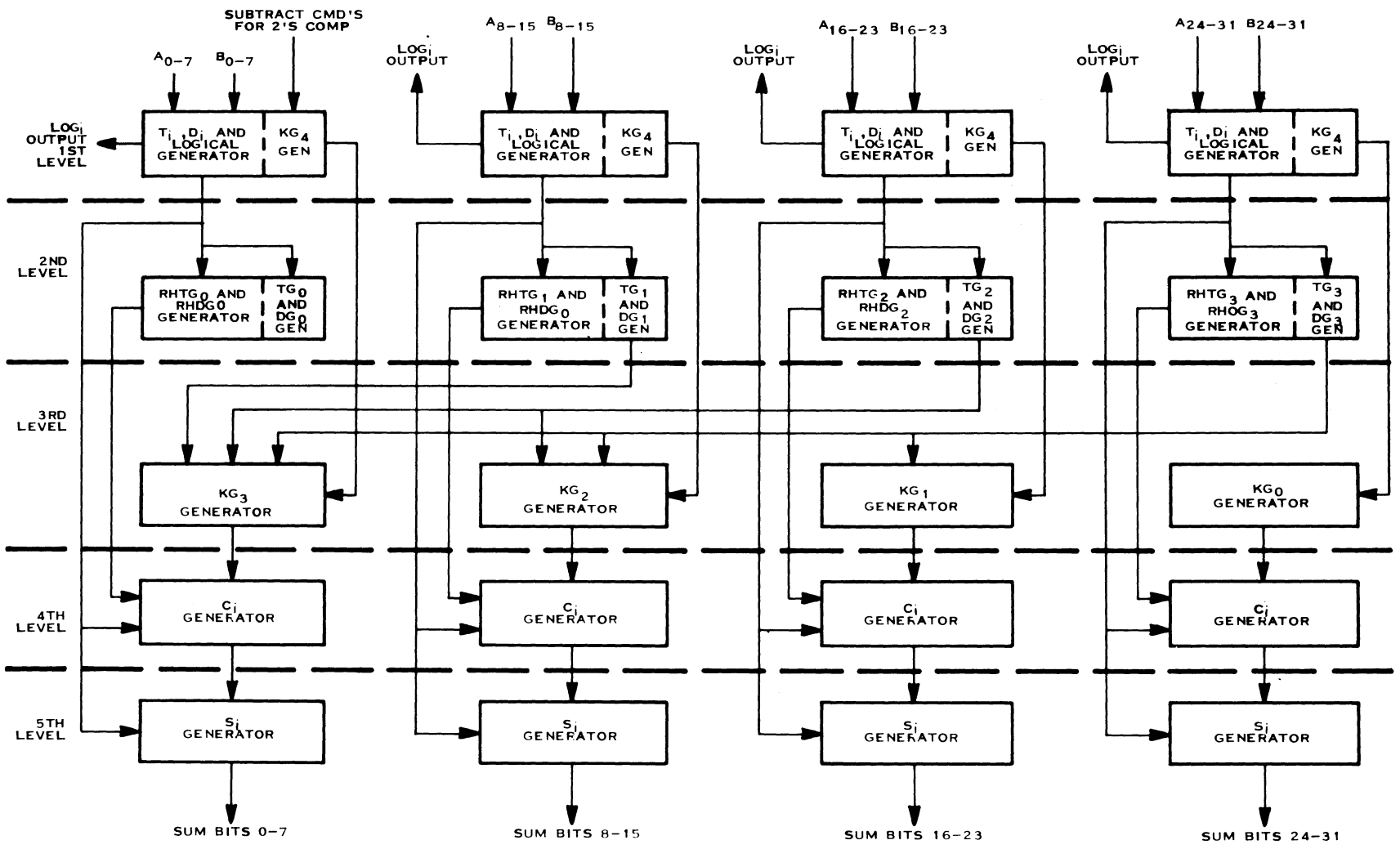
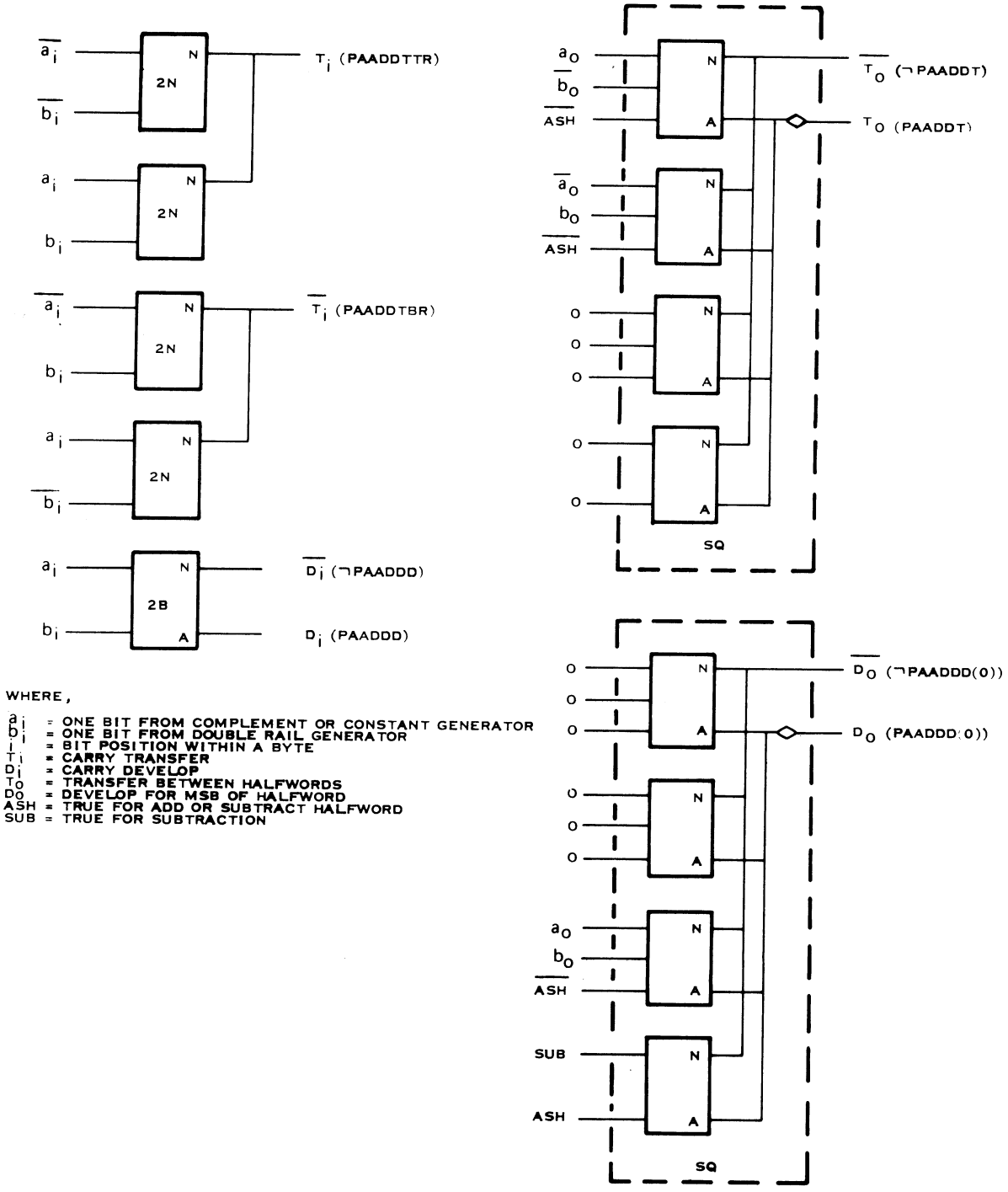


Figure 4-30. Five-Level Look-Ahead Adder



(B) 111671

Figure 4-31. Five-Level Look-Ahead Adder Detailed Block Diagram

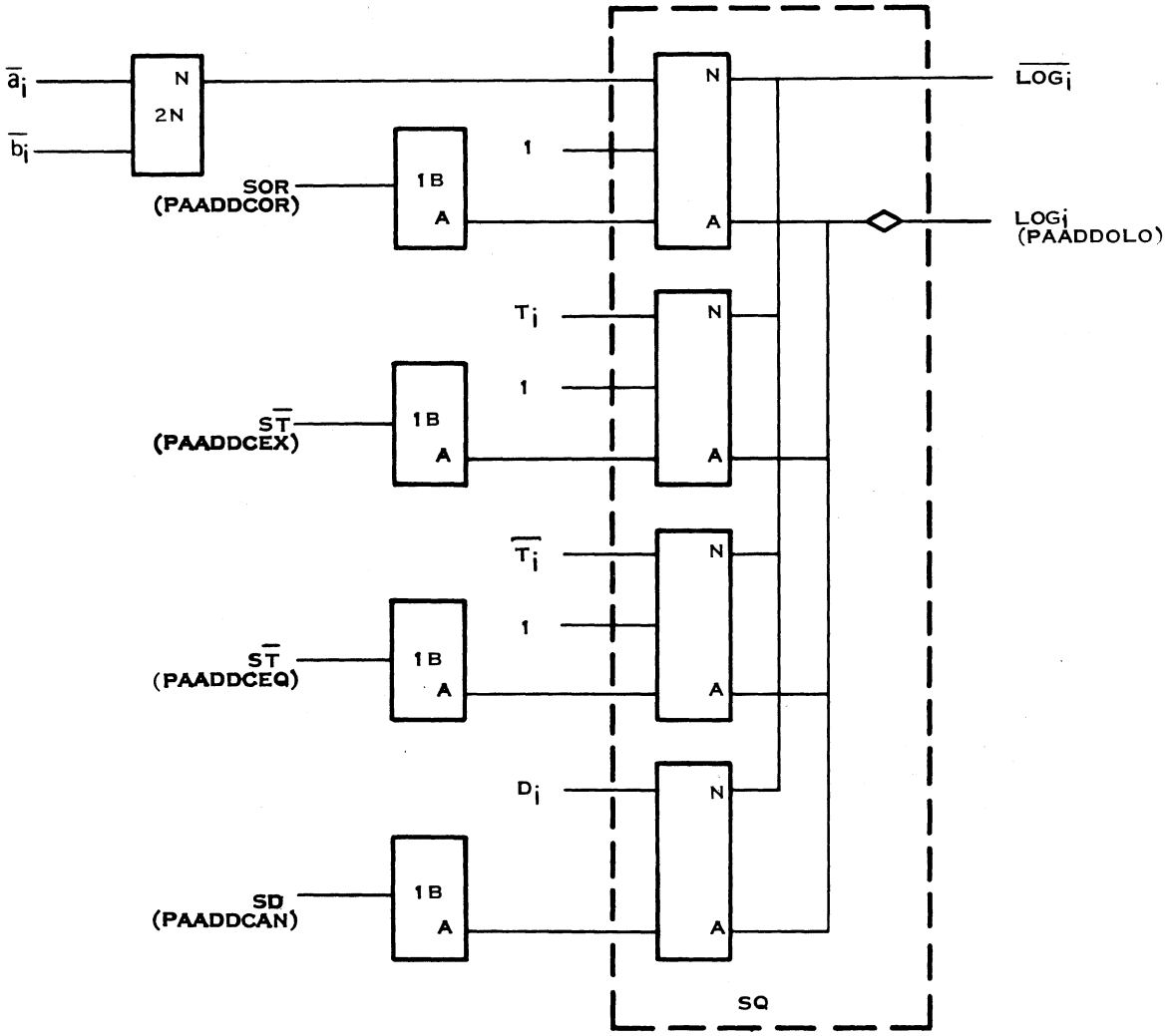


WHERE,

a_i = ONE BIT FROM COMPLEMENT OR CONSTANT GENERATOR
b_i = ONE BIT FROM DOUBLE RAIL GENERATOR
i = BIT POSITION WITHIN A BYTE
T_i = CARRY TRANSFER
D_i = CARRY DEVELOP
T_o = TRANSFER BETWEEN HALFWORDS
D_o = DEVELOP FOR MSB OF HALFWORD
ASH = TRUE FOR ADD OR SUBTRACT HALFWORD
SUB = TRUE FOR SUBTRACTION

(B)124752

Figure 4-32. Adder Level 1 Equation Implementation (Sheet 1 of 2)



WHERE:

LOG = OUTPUT FOR LOGICAL INSTRUCTIONS

SOR = TRUE FOR LOGICAL "OR" INSTRUCTIONS

ST = TRUE FOR LOGICAL "EXCLUSIVE OR" INSTRUCTIONS

$\overline{ST}$ = TRUE FOR LOGICAL "EQUIVALENCE" INSTRUCTIONS

SD = TRUE FOR LOGICAL AND INSTRUCTIONS.

(A)124753

Figure 4-32. Adder Level 1 Equation Implementation (Sheet 2 of 2)

$$T_o = (\overline{ASH}) \cdot a_o \cdot \overline{b}_o + (\overline{ASH}) \cdot \overline{a}_o \cdot b_o$$

$$D_o = SUB \cdot ASH + a_o \cdot b_o \cdot (\overline{ASH})$$

where,

ASH = true for add or subtract halfword instructions

SUB = true for subtract instructions

All the logical functions, except the OR function, are implemented using the carry transfer and develop equations and the logical function enables. The OR function is defined by the following equation:

$$OR_i = a_i + b_i$$

The exclusive OR function is true only when the two bits compared are different, so the carry transfer equation (T_i) is used. The equivalence function is true only when the two bits compared are the same, so the negated carry transfer equation ($\overline{T}_i$) is used. The AND function is true only when both bits compared are ones, so the carry develop equation (D_i) is used. The output equation for all logical instructions, therefore, is formed as follows:

$$LOG_i = SOR \cdot OR_i + ST \cdot T_i + S\overline{T} \cdot \overline{T}_i + SD \cdot D_i$$

where,

LOG_i = output at bit position i for all logical instructions

SOR = enable for OR functions

ST = enable for exclusive OR functions

$S\overline{T}$ = enable for equivalence functions

SD = enable for AND functions

The logical function output data (byte, halfword, or word, as determined by the logical instruction op-code) is distributed to the fifth level of the adder, where it is gated through to the AU1 transfer bus (AU1B).

4-100 Adder Level 2. Refer to figure 4-33 for the logic module connections required to implement the equations necessary for operation of adder level 2. The second level of the adder uses the carry transfer and develop equations from level 1 to generate the right hex group carries (RHTG and RHDG) and byte group carries (TG and DG). The carry equations generated by the second level of the adder exist on each of the four PPAUCD cards and apply to only one byte of data per card. A right hex group carry transfer is

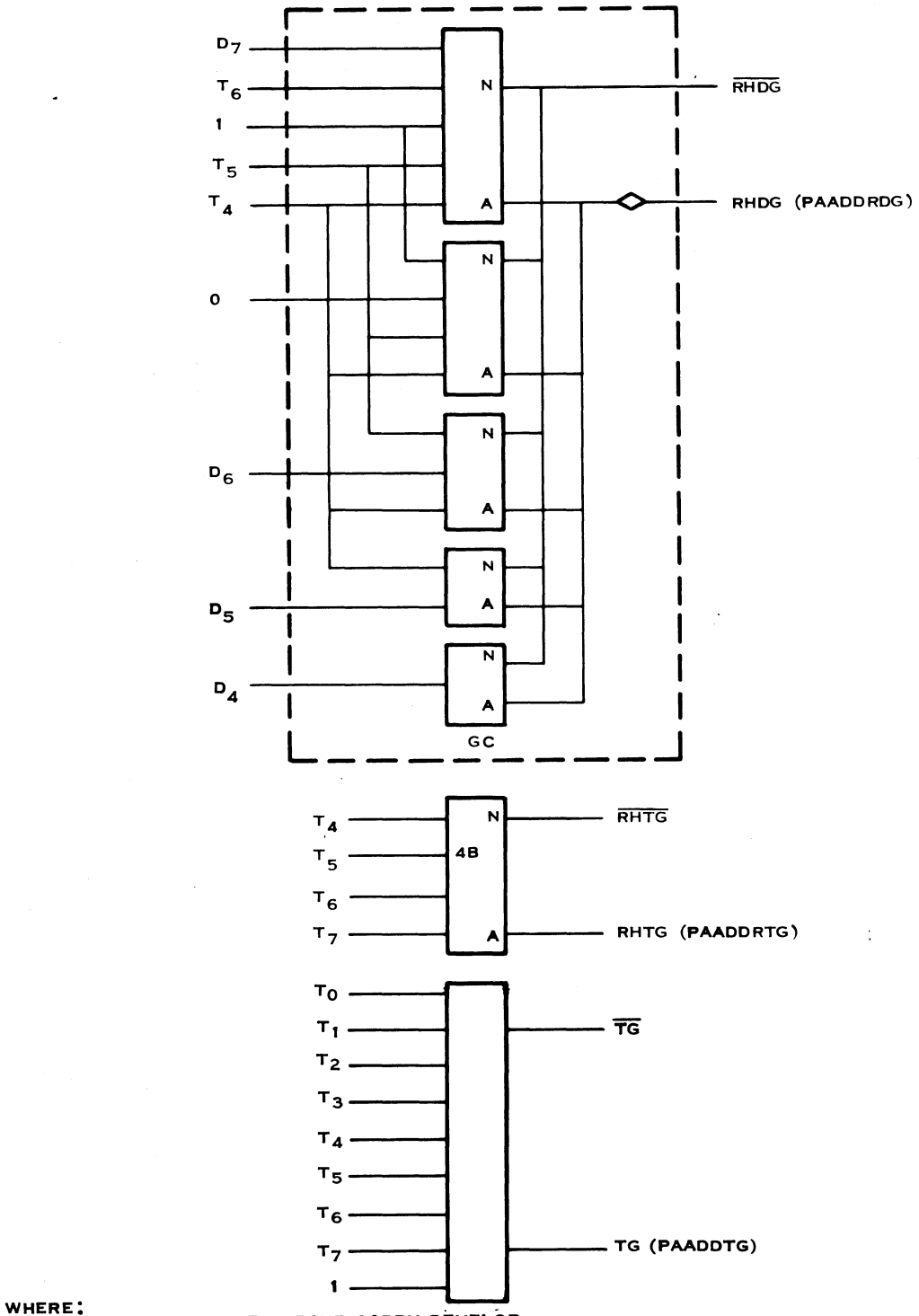


Figure 4-33. Adder Level 2 Equation Implementation (Sheet 1 of 2)

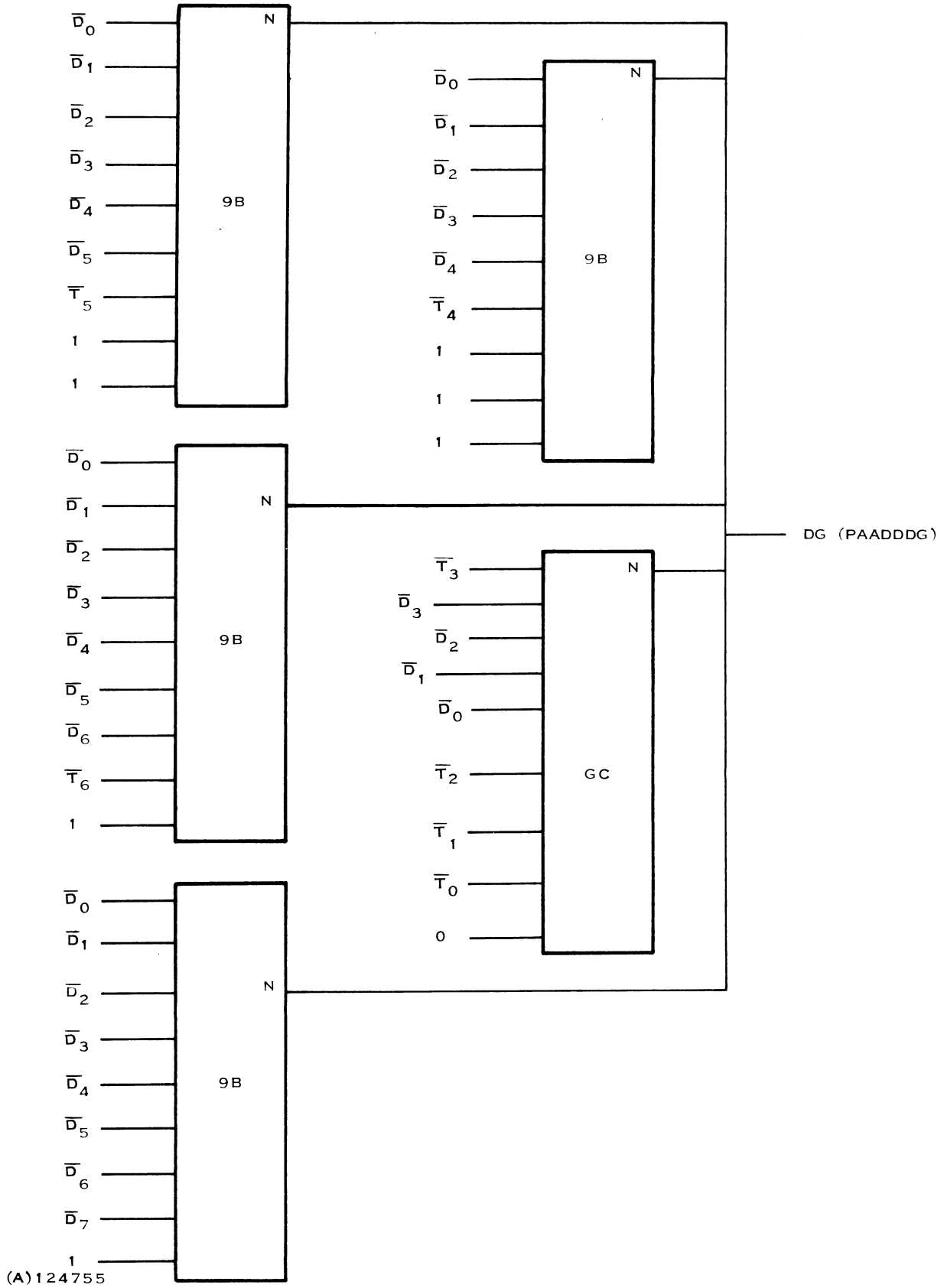


Figure 4-33. Adder Level 2 Equation Implementation (Sheet 2 of 2)

generated whenever all four carry transfers in the right hex group are true. This results in the following equation:

$$RHTG = T_4 \cdot T_5 \cdot T_6 \cdot T_7$$

A right hex group carry develop is generated whenever a carry is developed in the right hex group and the carry transfers propagate (propagation is not necessary when the MSB of the hex develops the carry) the carry through the right hex group. This results in the following equation:

$$RHDG = D_4 + D_5 \cdot T_4 + D_6 \cdot T_4 \cdot T_5 \cdot T_6 + D_7 \cdot T_4 \cdot T_5 \cdot T_6 \cdot T_7$$

A byte group carry transfer is generated whenever all eight carry transfers in the byte group are true. This results in the following equation:

$$TG = T_0 \cdot T_1 \cdot T_2 \cdot T_3 \cdot T_4 \cdot T_5 \cdot T_6 \cdot T_7$$

A byte group carry develop is generated whenever a carry is developed in a byte group and the carry transfers propagate the carry through the byte. Propagation is not required if the MSB of the byte develops the carry. This results in the following equation:

$$DG = D_0 + D_1T_0 + D_2T_0T_1 + D_3T_0T_1T_2 + D_4T_0T_1T_2T_3 + D_5T_0T_1T_2T_3T_4 + D_6T_0T_1T_2T_3T_4T_5 + D_7T_0T_1T_2T_3T_4T_5T_6$$

The equation for DG is implemented by forming $\overline{DG}$ and using the inverted output of the resulting logic network to extract DG. The equation for $\overline{DG}$ is as follows:

$$\overline{DG} = \overline{T}_0\overline{D}_0 + \overline{T}_1\overline{D}_0\overline{D}_1 + \overline{T}_2\overline{D}_0\overline{D}_1\overline{D}_2 + \overline{T}_3\overline{D}_0\overline{D}_1\overline{D}_2\overline{D}_3 + \overline{T}_4\overline{D}_0\overline{D}_1\overline{D}_2\overline{D}_3\overline{D}_4 + \overline{T}_5\overline{D}_0\overline{D}_1\overline{D}_2\overline{D}_3\overline{D}_4\overline{D}_5 + \overline{T}_6\overline{D}_0\overline{D}_1\overline{D}_2\overline{D}_3\overline{D}_4\overline{D}_5\overline{D}_6 + \overline{D}_0\overline{D}_1\overline{D}_2\overline{D}_3\overline{D}_4\overline{D}_5\overline{D}_6\overline{D}_7$$

This equation is formed by inverting the previous equation for DG, converting the result to the product of sums form, and multiplying out the resulting products. The byte group carries (TG and DG) are used in the third level of the adder and the right hex group carries (RHTG and RHDG) are used in the fourth level of the adder.

4-101 Adder Level 3. Refer to figure 4-34 for the logic module connections required to implement the equation necessary for operation of adder level 3. The third level of the adder uses the byte group carries from the second level of the adder to generate the carry into a byte (KG) from a previous byte or bytes. A carry is input to a byte whenever a carry is developed in a less

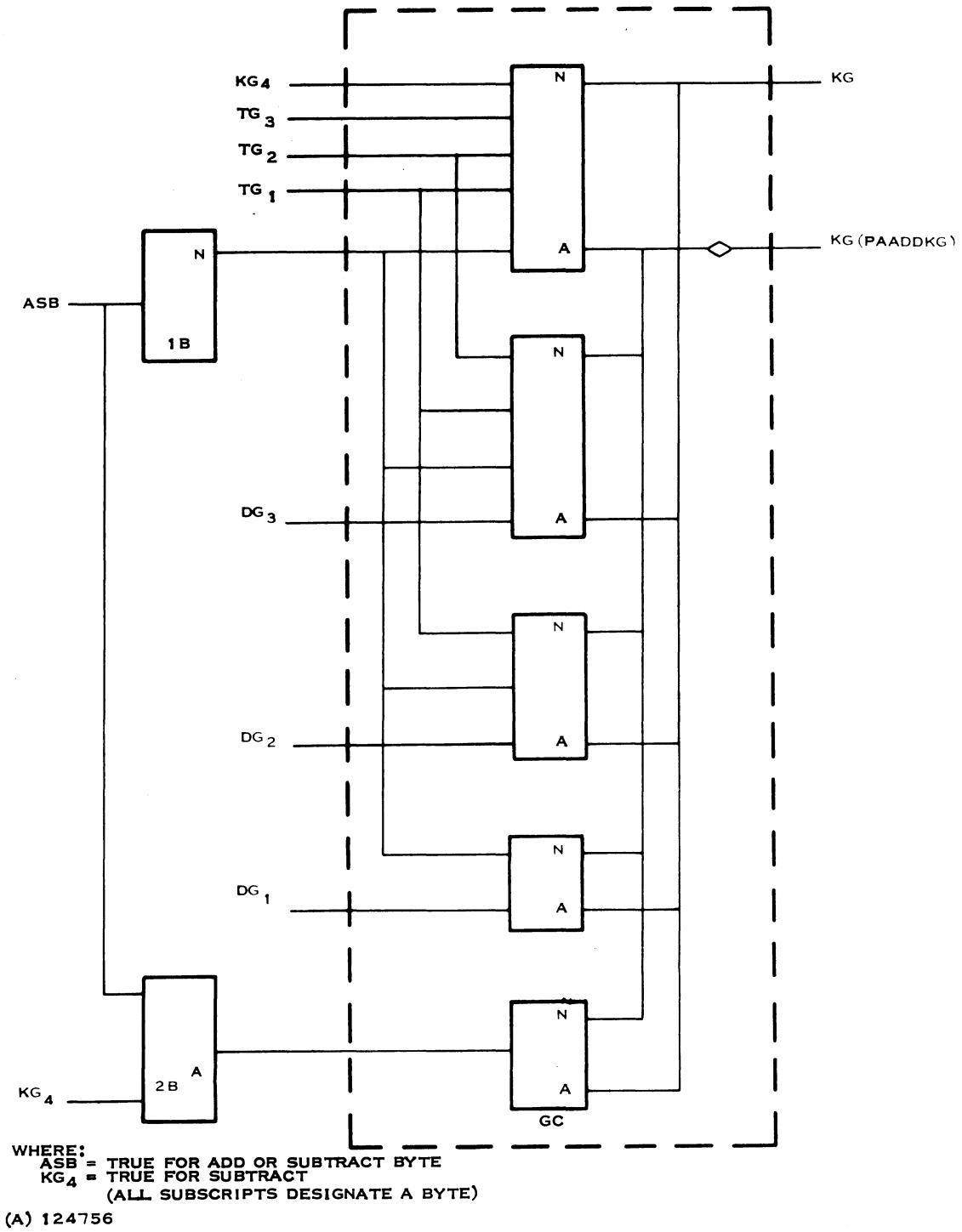


Figure 4-34. Adder Level 3 Equation Implementation

significant byte and the carry is propagated through to the byte in question. Propagation is not required if the carry is developed in the right adjacent byte. This results in the following four equations for the four bytes in a word.

$$KG_0 = DG_1 + DG_2TG_1 + DG_3TG_1TG_2 + KG_4TG_1TG_2TG_3$$

$$KG_1 = DG_2 + DG_3TG_2 + KG_4TG_2TG_3$$

$$KG_2 = DG_3 + KG_4TG_3$$

$$KG_3 = KG_4 = \text{true for all subtracts}$$

When a halfword add or subtract instruction is being executed, carries are not propagated from the right halfword to the left halfword, so the carry into byte equations must be modified.

$$KG_0 = DG_1 + KG_4TG_1$$

$$KG_1 = KG_4$$

$$KG_2 = DG_3 + KG_4TG_3$$

$$KG_3 = KG_4$$

When a byte add or subtract instruction is being executed, all carry into byte equations are set equal to KG_4 (true for subtract instructions). Since the PPAUCD cards are identical, they must contain the same logic yet generate the appropriate KG equation. This is accomplished by implementing the KG_0 equation for the word level instruction and routing inputs to each byte via the VPRMB motherboard to develop the appropriate KG equation. The KG_0 equation is modified as follows so that it can be used to develop the correct halfword and byte level KG equations.

$$KG_0 = KG_4(ASB) + DG_1(\overline{ASB}) + DG_2TG_1(\overline{ASB}) + DG_3TG_1TG_2(\overline{ASB}) + KG_4TG_1TG_2TG_3(\overline{ASB})$$

where,

ASB = true for add or subtract byte level instructions

The KG_0 equation for word level instructions is developed because ASB is zero. For word level instructions, KG_1 is developed by setting TG_1 to one and DG_1 to zero; KG_2 is developed by setting TG_1 and TG_2 to one and DG_1 and DG_2 to zero; KG_3 is developed by setting TG_1, TG_2, and TG_3 to one and DG_1, DG_2, and DG_3 to zero. The correct KG equations for bytes zero and one are developed for halfword instructions by using the control mentioned for the word level instructions and setting TG_2 to zero for both adds and

subtracts and DG_2 to zero for adds and one for subtracts. In reality, TG_2 and DG_2 are set equal to T_0 and D_0, respectively, of byte two for halfword instructions. This is done because T_0 of byte two is zero for halfword instructions and D_0 of byte two is zero for halfword adds and one for halfword subtracts. The developed KG equation for each byte is input to the fourth level of the adder.

4-102 Adder Level 4. Refer to figure 4-35 for the logic module connections required to implement the equations necessary for operation of adder level 4. The fourth level of the adder uses the carry develop, carry transfer, right hex group carry, and the carry into byte equations to generate the carry equations for each bit in the 32-bit adder result. Each PPAUCD card generates the carries for one byte of data. A carry is generated for a bit whenever a carry is developed in the word and the carry is propagated through to the bit in question. Propagation is not required if the carry is developed in the right adjacent bit. This results in the following carry equations for a byte of data:

$$C_0 = D_1 + D_2T_1 + D_3T_1T_2 + (RHDG)T_1T_2T_3 + KG(RHTG)T_1T_2T_3$$

$$C_1 = D_2 + D_3T_2 + (RHDG)T_2T_3 + KG(RHTG)T_2T_3$$

$$C_2 = D_3 + (RHDG)T_3 + KG(RHTG)T_3$$

$$C_3 = RHDG + KG(RHTG)$$

$$C_4 = D_5 + D_6T_5 + D_7T_5T_6 + (KG)T_5T_6T_7$$

$$C_5 = D_6 + D_7T_6 + (KG)T_6T_7$$

$$C_6 = D_7 + (KG)T_7$$

$$C_7 = KG$$

The results from the carry equations for four bytes of data are input to the fifth level of the adder.

4-103 Adder Level 5. Refer to figure 4-36 for the logic module connections required to implement the equations and gating necessary for operation of adder level 5. The fifth level of the adder uses the carry equations from level 4 and the carry transfer equations from level 1 to generate the sum outputs. In addition, level 5 includes the output gating associated with the AU1 transfer bus for the AU. Each PPAUCD card generates sum outputs for one byte of data. The sum output is formed by an exclusive OR between the carry (C_i) for a bit and the carry transfer (T_i) for a bit and is expressed by the following equation.

$$S_i = T_i \cdot \overline{C}_i + \overline{T}_i \cdot C_i$$

where,

S_i = sum output at bit position i

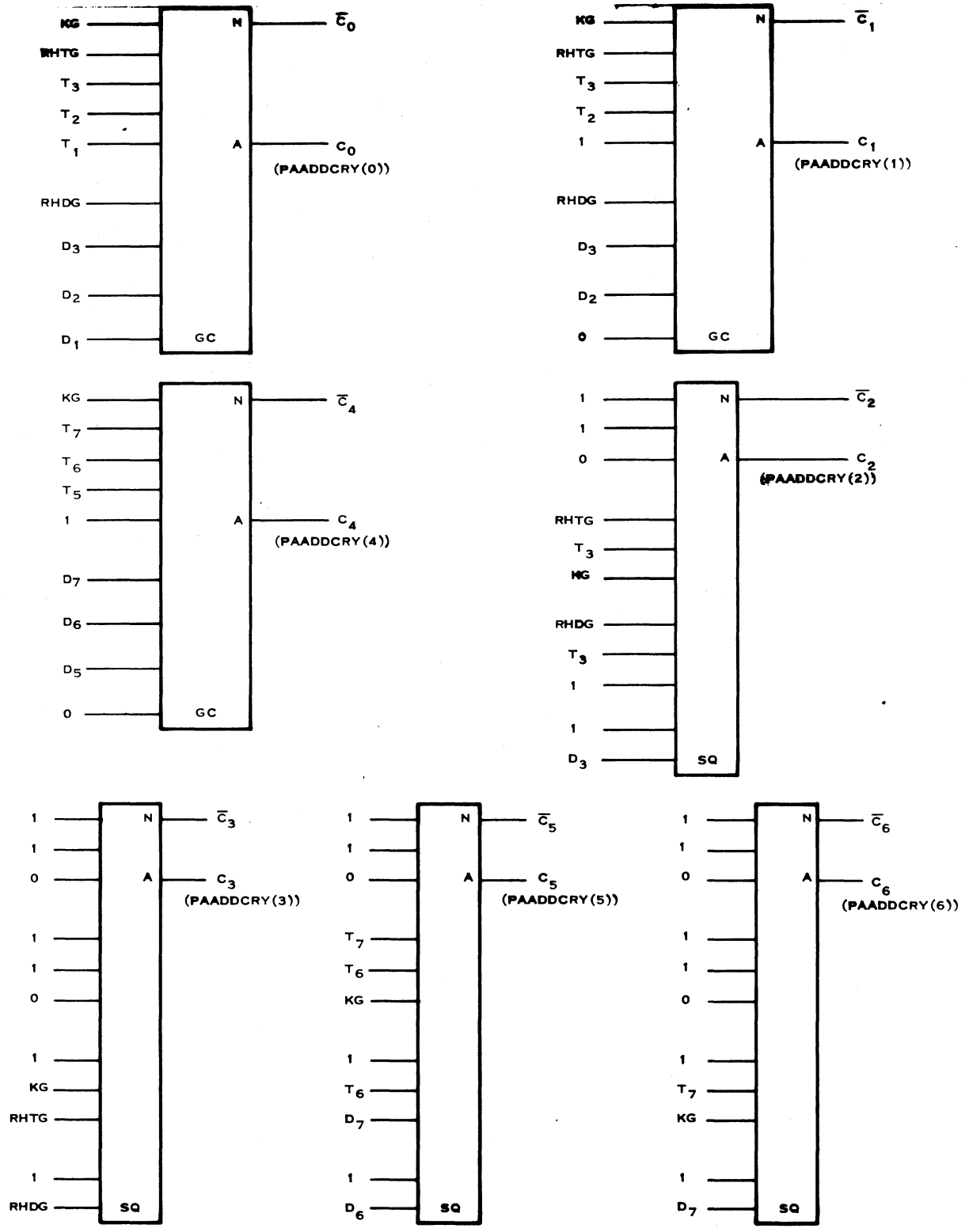
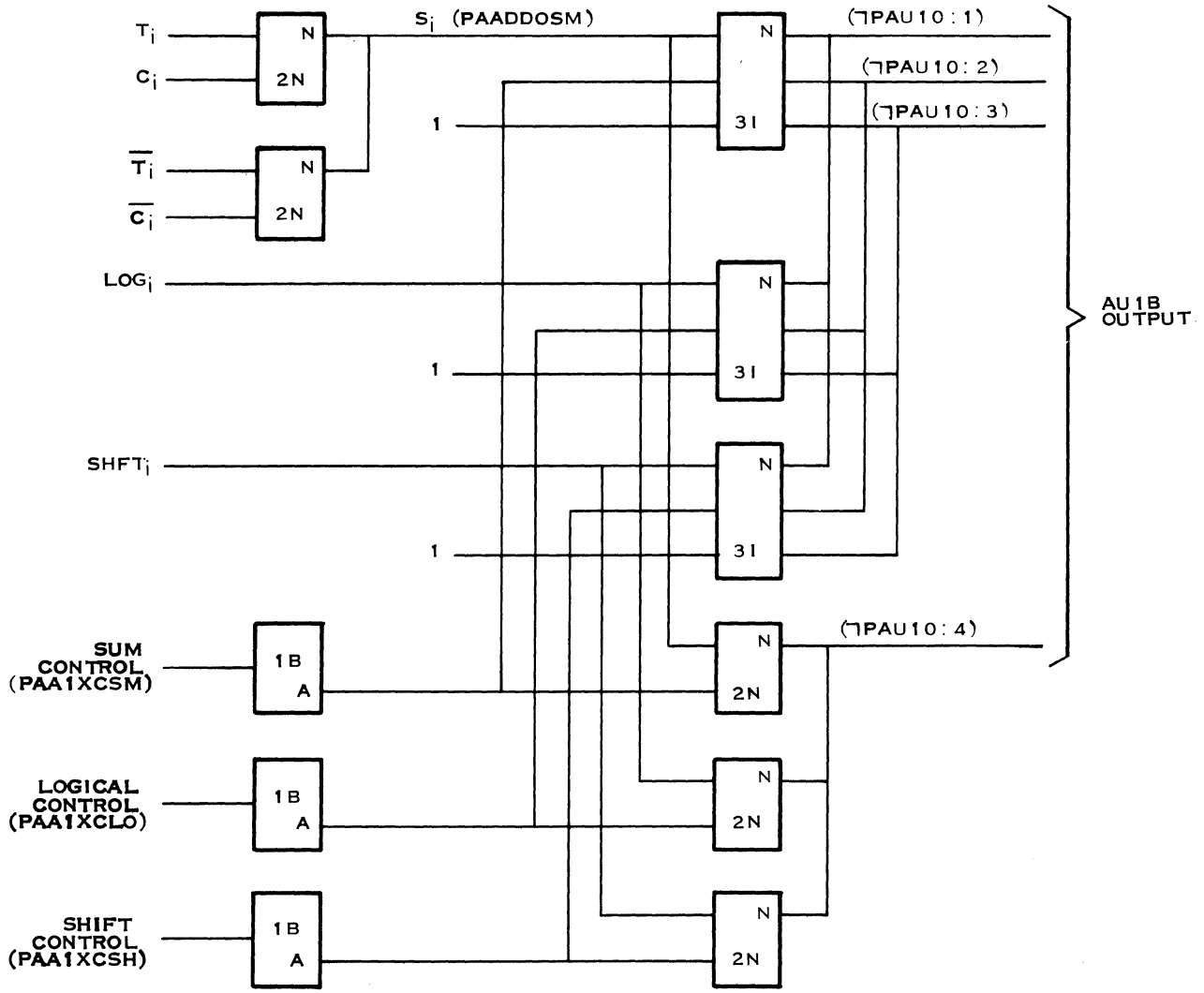


Figure 4-35. Adder Level 4 Equation Implementation



WHERE:

S_i = SUM OUTPUT AT BIT POSITION i
$SHFT_i$ = SHIFT OUTPUT AT BIT POSITION i
LOG_i = LOGICAL OUTPUT AT BIT POSITION i

(A)124758

Figure 4-36. Adder Level 5 Equation Implementation (Sheet 1 of 2)

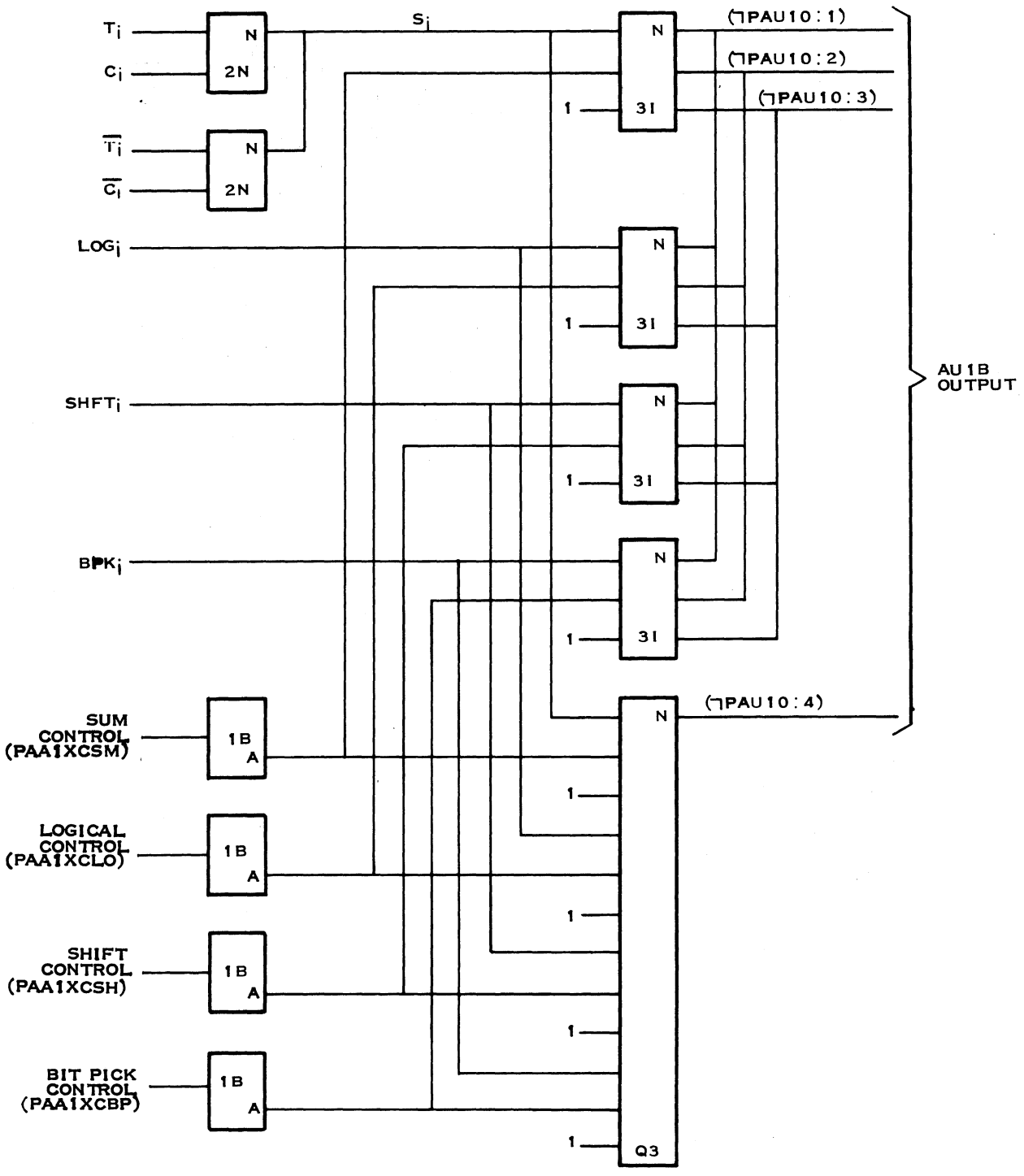


Figure 4-36. Adder Level 5 Equation Implementation (Sheet 2 of 2)

When a sum (add or subtract) is being executed, the S_i terms from the adder are gated through the AU1 transfer bus (AU1B) by the sum enable. When a logical function is being executed, the LOG_i terms from the first level of the adder are gated through AU1B by the logical enable. When a shift or poll instruction is being executed, the $SHFT_i$ or BPK_i terms, respectively, are gated through AU1B by the appropriate enable.

4-104 SHIFTER. Refer to figure 4-37 for a simplified block diagram of one byte of the AU shifter representing the logic on each PPAUCD card. The shifter performs right or left arithmetic, logical, or cyclic shifts in increments of 1, 4, or 8 bits on a 32-bit word from the MDB. In addition, the shifter and aligner combine to perform right or left cyclic shifts in increments of 16 bits. The byte of data corresponding to the PPAUCDM card number and its right and left adjacent bytes are applied to the shifter bit select logic to provide for the 1, 4, or 8 bit shifts. The selected byte from the aligner on the same PPAUCDM card is applied to the shifter bit select logic to provide for the 16 bit shifts. The remainder of the inputs to the shifter are supplied by the CONTAU card and provide the control necessary to generate a byte of shifted data.

The byte 0 fix control is used in the shift size and direction logic to generate the zero fill in byte zero of the shifter word output for right logical shifts. The same control is used in the sign propagation logic to provide sign propagation in byte zero of the shifter word output for right arithmetic shifts. The byte 3 fix control is used in the shift size and direction logic to generate the zero fill in byte three of the shifter word output for left arithmetic and left logical shifts. The shift size controls (1, 4, and 8) are used in both blocks of logic to generate the proper enables so that the correct amount of shift and sign propagation is gated through the shifter bit select logic. The shift type controls (cyclic, arithmetic, and logical) are used in the shift size and direction logic to control whether or not zero fill is necessary in bytes zero and three of the shifter word output. The arithmetic shift control and the right shift control are used in the sign propagation logic to determine when sign propagation is necessary. The shift direction controls (left and right) are used in the shift size and direction logic to generate the proper enables so that bits from the correct adjacent byte are gated through the shifter bit select logic. The cyclic shift of 16 control is used in the shifter bit select logic to enable the byte from the aligner through to the shifter output. The remaining paragraphs on the shifter give a more detailed description of the shift size and direction logic, the sign propagation logic, and shifter bit select logic.

4-105 Shift Size and Direction Logic. The shift size and direction logic uses the inputs shown in figure 4-37 to develop the following equations:

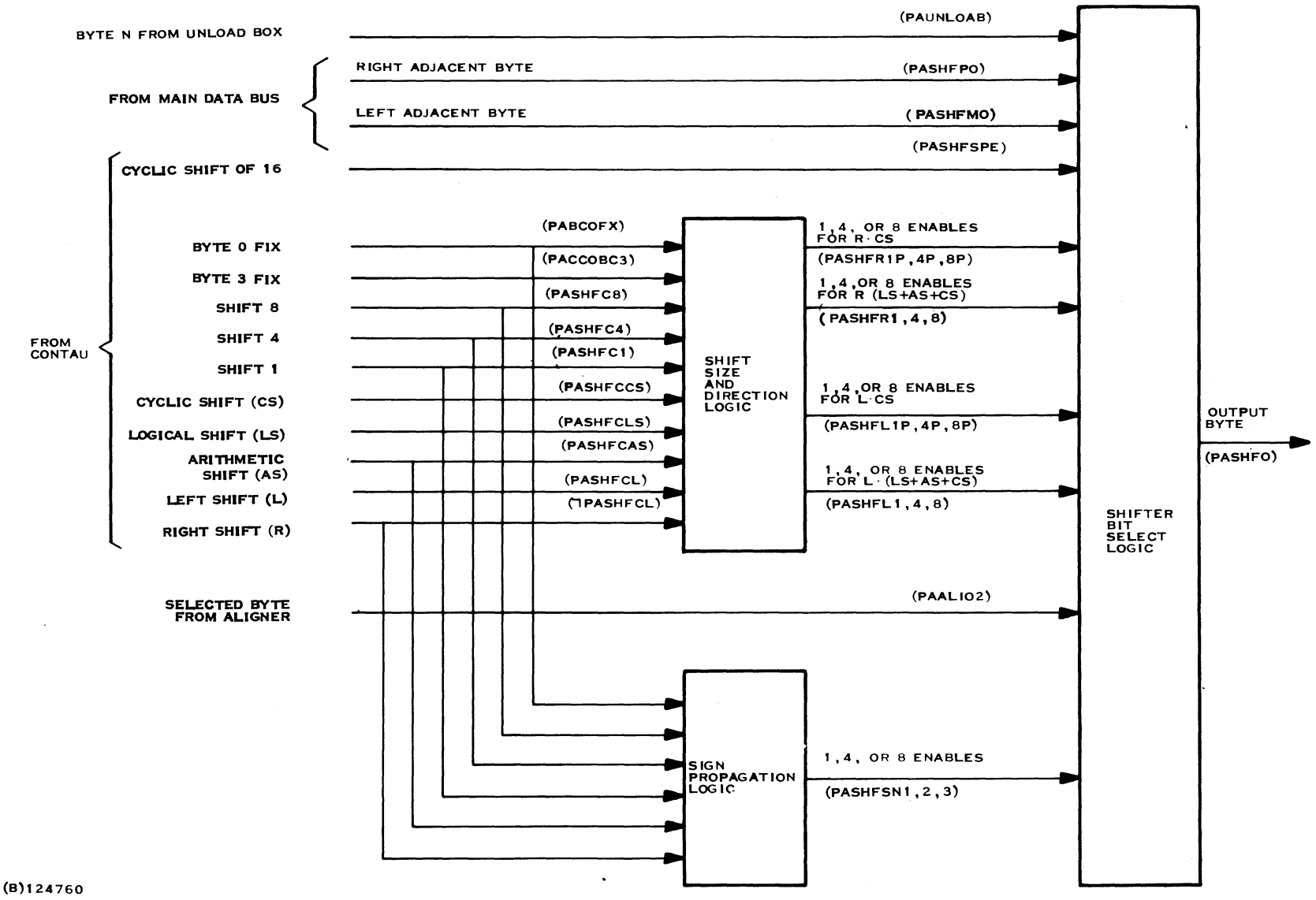


Figure 4-37. Shifter Logic on PPAUCD (N)

$$PASHFR1P = \overline{B0} \cdot \overline{B3} \cdot S1 \cdot R \cdot (AS+LS) + S1 \cdot R \cdot CS$$

$$PASHFR1 = (AS+LS+CS)(R \cdot S1)$$

$$PASHFR4P = \overline{B0} \cdot \overline{B3} \cdot S4 \cdot R \cdot (AS+LS) + S4 \cdot R \cdot CS$$

$$PASHFR4 = (AS+LS+CS)(R \cdot S4)$$

$$PASHFR8P = \overline{B0} \cdot \overline{B3} \cdot S8 \cdot R \cdot (AS+LS) + S8 \cdot R \cdot CS$$

where,

B0 = byte 0 fix control

B3 = byte 3 fix control

S1 = shift 1 bit

S4 = shift 4 bits

S8 = shift 8 bits

R = right shift

AS = arithmetic shift

LS = logical shift

CS = cyclic shift

This set of equations provides the enables necessary for the shifter bit select logic to perform right arithmetic, right logical, or right cyclic shifts in increments of 1, 4, or 8 bits. A similar set of equations (PASHFL1P, PASHFL1, PASHFL4P, PASHFL4, and PASHFL8P) are developed for left shifts. The PASHFR1P and PASHFR1 equations combine to provide the enables required for all types of right shift of 1 bit. The first term in the PASHFR1P equation enables zeroing of the first bit in byte zero of the shifter output word when a right logical or right arithmetic shift of 1 bit is being performed (the sign propagation logic overrides all zero fill of byte zero when a right arithmetic shift is being performed). This same term in the PASHFL1P equation enables zeroing of the last bit in byte three when a left logical or left arithmetic shift of 1 bit is being performed. The first term in the PASHFR1P (PASHFL1P) equation provides the PPAUCD(1-3) (PPAUCD(0-2)) cards with the enables necessary for the shifting of bit 7 (bit 0) from the left (right) adjacent byte to bit 0 (bit 7) of the output byte when a right (left) logical or right (left) arithmetic shift of 1 bit is being performed. The second term in the PASHFR1P (PASHFL1P) equation enables shifting of bit 7 (bit 0) from the left (right) adjacent byte to bit 0 (bit 7) of the output byte when a right (left) cyclic shift of 1 bit is being performed. The PASHFR1 (PASHFL1) equation enables the shifting of bits 0 through 6 (1 through 7) to bits 1 through 7 (0 through 6) of the output byte when a right (left) arithmetic, right (left) logical, or right (left) cyclic shift of 1 bit is being performed.

The PASHFR4P (PASHFL4P) and PASHFR4 (PASHFL4) equations combine to provide the enables required for all types of right (left) shift or 4 bits. The first term in the PASHFR4P (PASHFL4P) equation enables the zeroing of the first (last) four bits in byte zero (three) of the shifter output word when a right (left) logical or right (left) arithmetic shift of 4 bits is being performed. In addition, the first term in the PASHFR4P (PASHFL4P) equation is used on the PPAUCD(1-3) (PPAUCD(0-2)) cards to enable the shifting of the four last (first) bits from the left (right) adjacent byte to the four first (last) bits of the output byte when a right (left) logical or right (left) arithmetic shift of 4 bits is being performed. The second term in the PASHFR4P (PASHFL4P) equation enables the shifting of the four last (first) bits from the left (right) adjacent byte to the four first (last) bits of the output byte when a right (left) cyclic shift of 4 bits is being performed. The PASHFR4 (PASHFL4) equation enables the shifting of the four first (last) bits of the output byte to the four last (first) bits of the output byte when a right (left) arithmetic, right (left) logical, or right (left) cyclic shift of 4 bits is being performed.

The PASHFR8P (PASHFL8P) equation provides the enables required for all types of right (left) shift of 8 bits. The first term in the PASHFR8P (PASHFL8P) equation enables the zeroing of byte 0 (three) when a right (left) logical or right (left) arithmetic shift of 8 bits is being performed. In addition, the first term in the PASHFR8P (PASHFL8P) equation is used on the PPAUCD(1-3) (PPAUCD(0-2)) cards to enable the shifting of the left (right) adjacent byte to the output byte when a right (left) logical or right (left) arithmetic shift of 8 bits is being performed. The second term in the PASHFR8P (PASHFL8P) equation enables the shifting of the left (right) adjacent byte to the output byte when a right (left) cyclic shift of 8 bits is being performed.

4-106 Sign Propagation Logic. The sign propagation logic uses the inputs shown in figure 4-37 to develop the following equations:

$$PASHFSN1 = S8 \cdot AS \cdot R \cdot B0$$

$$PASHFSN2 = (S4 + S8)\ (AS \cdot R \cdot B0)$$

$$PASHFSN3 = (S1 + S4 + S8)\ (AS \cdot R \cdot B0)$$

This set of equations provides the enables necessary for the shifter bit select logic to sign extend the quantity being shifted. Examination of the three equations indicates that sign propagation occurs only when a right arithmetic shift is being performed and the logic using the equations is on the PPAUCDM(0) card. When this is the case, the PASHFSN3 equation is used to enable the sign of the data to be shifted into the bit 0 of byte 0, the PASHFSN2 equation is used to enable the sign of the data to be shifted into bits 1 through 3 of byte 0, and the PASHFSN1 equation is used to enable the

sign of the data to be shifted into bits 4 through 7 of byte 0. When sign propagation is necessary, only equation PASHFSN3 is true for a shift of one, equations PASHFSN3 and PASHFSN2 are true for a shift of four, and all three equations (PASHFSN3, PASHFSN2, and PASHFSN1) are true for a shift of eight.

4-107 Shifter Bit Select Logic. The shifter bit select logic uses the enables from the shift size and direction logic and the sign propagation logic to select and gate data from the unload box and the MDB through to the output when a shift of 1, 4, or 8 is specified. In addition, the cyclic shift of 16 control from the CONTAU card is used to select and gate data from the aligner through to the output when a cyclic shift of 16 is specified. The ten previously-mentioned shift size and direction enables from the shift size and direction logic, the three previously-mentioned sign propagation enables from the sign propagation logic, and the cyclic shift of 16 enable from the CONTAU card are all paired with an appropriate bit of data from one of the three data sources. When the control signals from the CONTAU card drive an enable signal true, the data bit paired with the true enable is output to the proper bit in the output byte. Each PPAUCD card produces a byte of shifted data for output to the AU1 transfer bus in this manner.

4-108 BIT PICKER. Refer to figure 4-38 for a diagram of the data flow required for bit picker operation. The bit picker is used during poll instructions to scan a byte of data (typically from a CR register) from the MSB to the LSB in order to determine the number of zeroes from the MSB of the byte to the first one. The bit picker accepts the true and complement form of the word supplied by the MDB from the unload box. All four bytes of the word are operated on in parallel (one byte per PPAUCD card) by the bit picker and the necessary selection of the byte specified by the poll instruction is supplied by the PPCTL2 card.

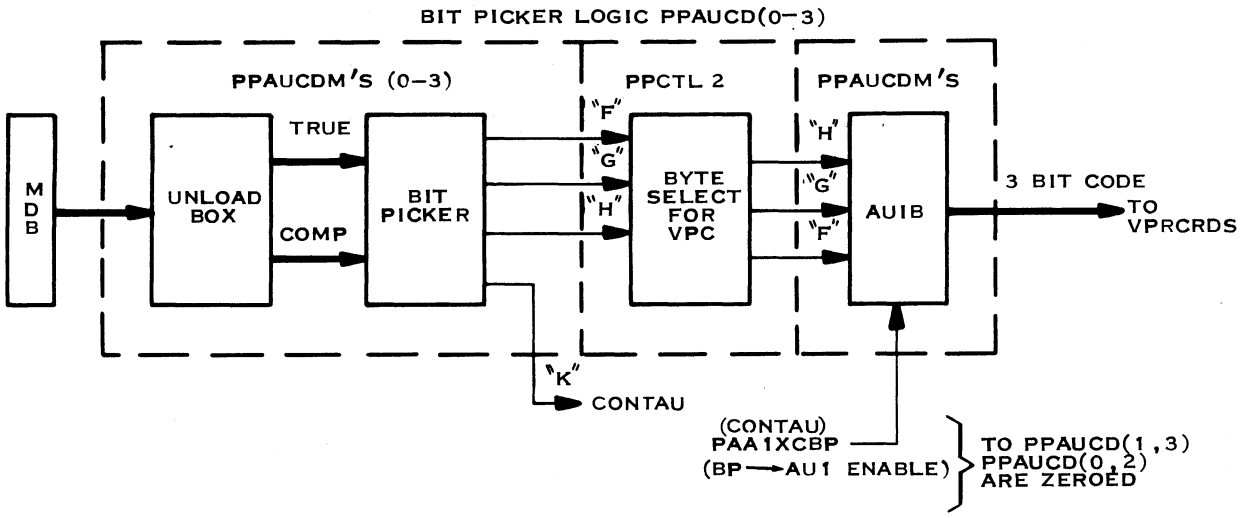
A three-bit code (H G F) is developed by the pit picker on each PPAUCD card to indicate the zero count to the first one for each byte of the input word. The equations formed to generate the three-bit code are based on the unload byte table in figure 4-38. In addition, an equation is formed (K) to indicate when all bits of a byte are zero. Refer to figure 4-39 for the logic module connections required to implement the following equations:

$$F_j = \bar{a}_0 a_1 + \bar{a}_0 \bar{a}_2 a_3 + \bar{a}_0 \bar{a}_2 \bar{a}_4 a_5 + \bar{a}_0 \bar{a}_2 \bar{a}_4 \bar{a}_6 a_7$$

$$G_j = \bar{a}_0 \bar{a}_1 a_2 + \bar{a}_0 \bar{a}_1 a_3 + \bar{a}_0 \bar{a}_1 \bar{a}_4 \bar{a}_5 a_6 + \bar{a}_0 \bar{a}_1 \bar{a}_4 \bar{a}_5 a_7$$

$$H_j = \bar{a}_0 \bar{a}_1 \bar{a}_2 \bar{a}_3 a_4 + \bar{a}_0 \bar{a}_1 \bar{a}_2 \bar{a}_3 a_5 + \bar{a}_0 \bar{a}_1 \bar{a}_2 \bar{a}_3 a_6 + \bar{a}_0 \bar{a}_1 \bar{a}_2 \bar{a}_3 a_7$$

$$K_j = \bar{a}_0 \bar{a}_1 \bar{a}_2 \bar{a}_3 \bar{a}_4 \bar{a}_5 \bar{a}_6 \bar{a}_7$$



UNLOAD BYTE

	0	1	2	3	4	5	6	7
"F" IS TRUE IF:	0	1	X	X	X	X	X	X
OR	0	X	0	1	X	X	X	X
ODD BITS OR	0	X	0	X	0	1	X	X
OR	0	X	0	X	0	X	0	1
"G" IS TRUE IF:	0	0	X	X	0	0	X	1
OR	0	0	X	X	0	0	1	X
OR	0	0	X	1	X	X	X	X
OR	0	0	1	X	X	X	X	X
"H" IS TRUE IF:	0	0	0	0	1	X	X	X
OR	0	0	0	0	X	1	X	X
OR	0	0	0	0	X	X	1	X
OR	0	0	0	0	X	X	X	1
"K" IS TRUE IF:	0	0	0	0	0	0	0	0

V	P	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
H	G	F

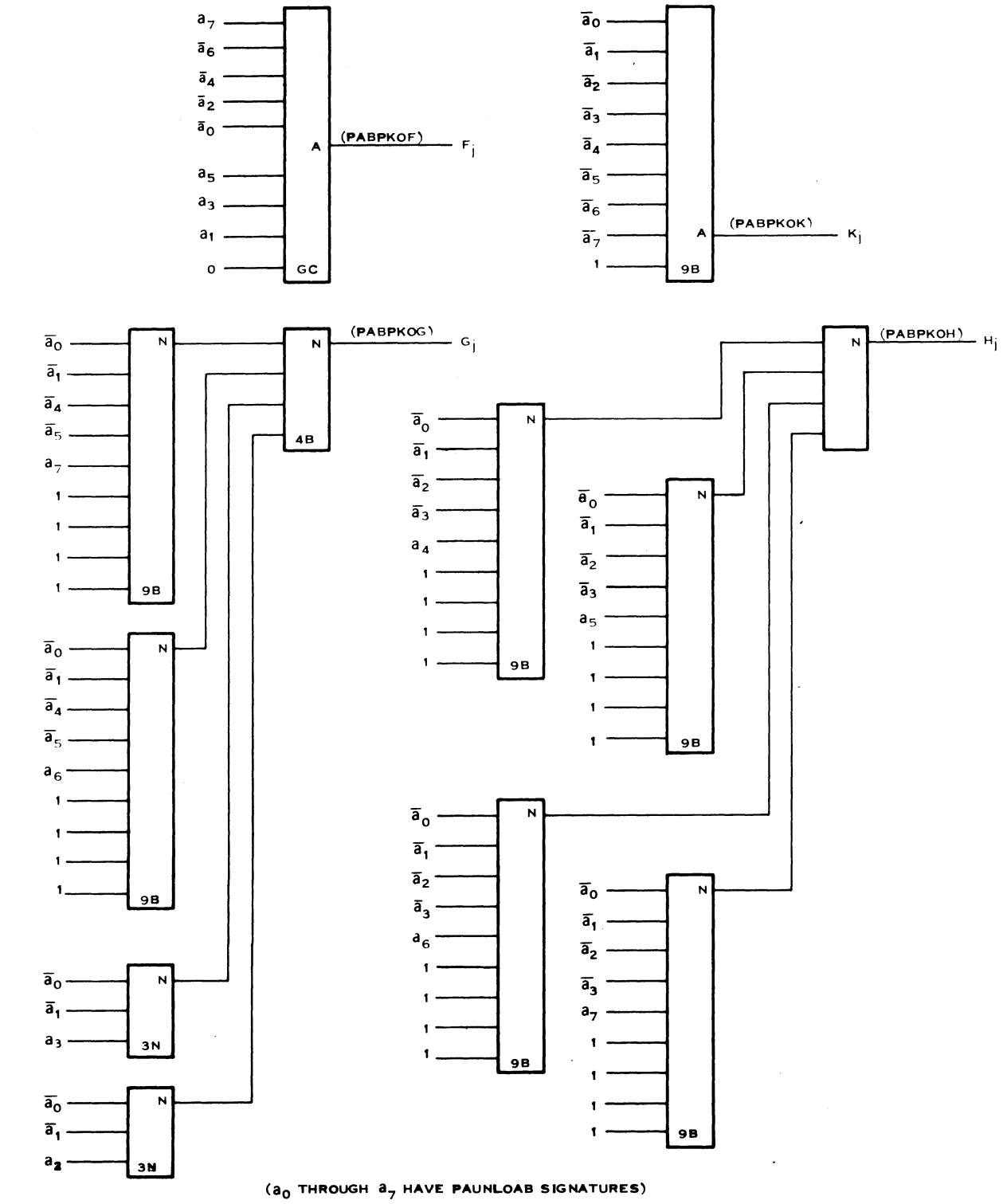
REFER TO LOGIC DIAG PG 12 AND 13

PPCTL2:

1. DECODES POLL INSTRUCTION
2. LOOKS AT DESTINATION ADDRESS FOR WHICH HALFWORD
3. LOOKS AT SOURCE ADDRESS FOR WHICH BYTE TO BE SELECTED FROM THE BIT PICKER

(A) 111699

Figure 4-38. Bit Picker Data Flow



(a_0 THROUGH a_7 HAVE PAUNLOAB SIGNATURES)

(B)124761

Figure 4-39. Bit Picker Equation Implementation

where,

F_j is the LSB of the three-bit code for byte j

G_j is the LSB+1 of the three-bit code for byte j

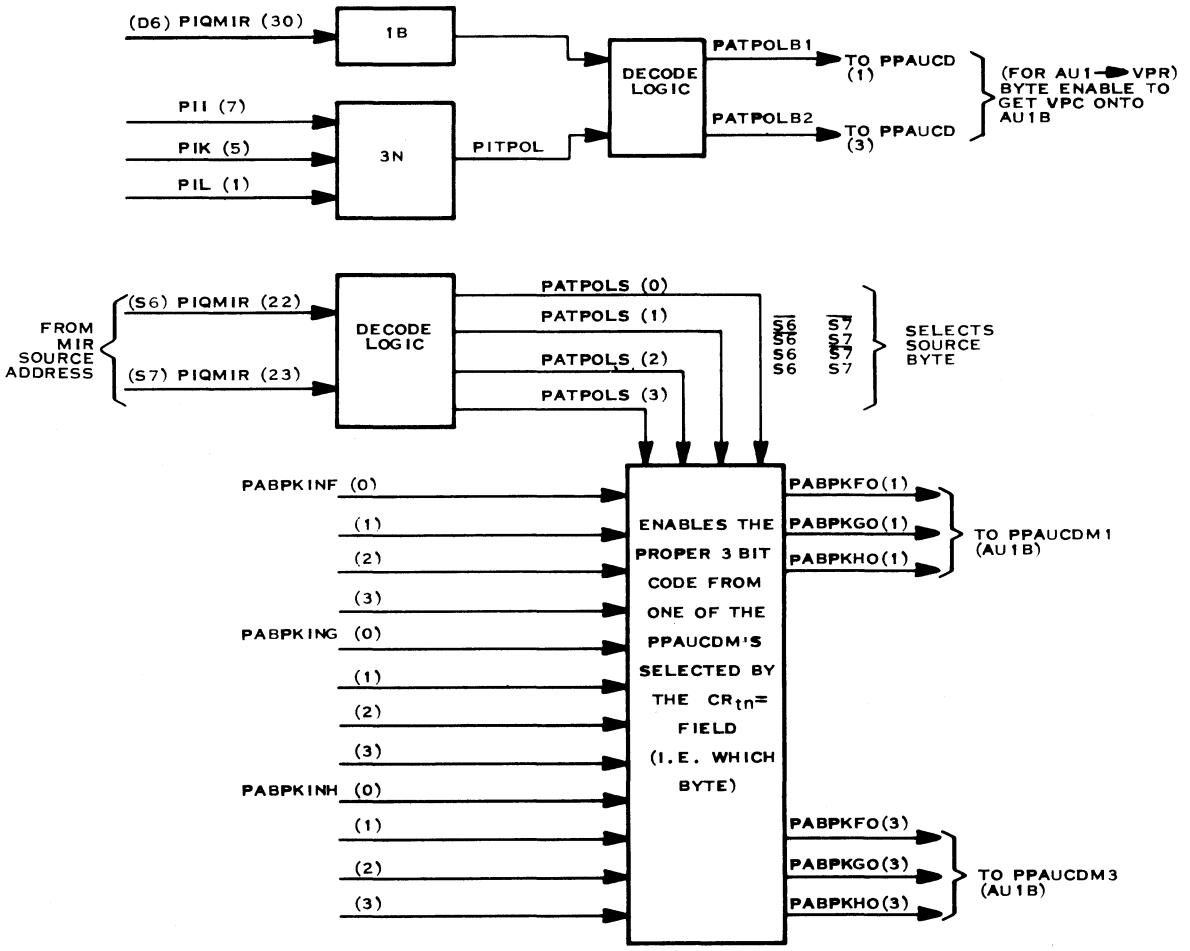
H_j is the MSB of the three-bit code for byte j

K_j is the no ones control signal for byte j

$a_0a_1a_2a_3a_4a_5a_6a_7$ is a byte of data from the unload box

The four sets of H, G, and F lines (one set per PPAUCD card) are input to the PPCTL2 card and the four K lines are input to the CONTAU card. The K line of the byte specified by the poll instruction is used to determine if the next instruction is to be skipped (K=0) or not (=1). The four sets of H, G, and F lines are applied to the byte select logic as shown in figure 4-40. Bits 6 and 7 from the MIR source address are decoded and used to enable the three-bit code from the specified byte to the AU1 transfer bus select logic (mentioned in the description of adder level 5) on PPAUCD1 and PPAUCD3. Bit 6 from the MIR destination address and the AU1 transfer bus select logic enable signal (PAA1XCBP) from the CONTAU card combine to select the right or left halfword three-bit code and to enable the three-bit code of the selected halfword to the AU1 transfer bus. The result is input to a VPR halfword.

4-109 TEST BOX 1, 2, AND 3 LOGIC. The test box logic is used to perform various tests on data from the unload box and double rail generator. The test box 1 logic is used during the execution of TP, TM, TZ, and TN instructions to test data from the unload box for positive, negative, zero, and nonzero data, respectively. In addition, the test box 1 logic is used during the execution of stack instructions to test the stack parameters for zero, nonzero, negative, and positive data. The test box 2 logic is used during the execution of TO (test for any one), TZ (test for any zero), TAO (test for all ones), TAZ (test for all zeroes), TSZ (test for any zero and set), TSO (test for any one and set), TRZ (test for any zero and reset), and TRO (test for any one and reset) type instructions to test data from the unload box in only those bit positions specified by the R field (the R field is also supplied by the unload box). The test box 2 logic is also used during the execution of VPTO and VPTZ (test CR for one and zero, respectively) instructions to test data from the unload box only in the bit position of the active VP code. The test box 3 logic is used during the execution of IBN (increment and branch if nonzero), IBZ (increment and branch if zero), DBN (decrement and branch if nonzero), and DBZ (decrement and branch if zero) instructions to test data from the double rail generator for zero and nonzero data. A detailed description of each test box is given in the following paragraphs.



(A)111661

Figure 4-40. Bit Picker Support Logic on PPCTL2

4-110 Test Box 1 Logic. The test box 1 logic accepts VPR or CR data in complement form from the unload box when a test instruction is being executed or SWBD data in complement form when a stack instruction is being executed. All four bytes of the input data word are operated on in parallel by the four PPAUCD cards and the CONTAU card uses the generated signals as necessary for word, halfword, and byte level instructions. Using only the data from the unload box, the test box 1 logic implements the following equations to indicate the byte of data tested is zero or nonzero and positive or negative.

$$\text{PATS1OSN (Negative)} = a_0$$

$$\overline{\text{PATS1OSN}} \text{ (Positive)} = \bar{a}_0$$

$$\text{PATS1OZ (Zero)} = \bar{a}_0\bar{a}_1\bar{a}_2\bar{a}_3\bar{a}_4\bar{a}_5\bar{a}_6\bar{a}_7$$

$$\overline{\text{PATS1OZ}} \text{ (Nonzero)} = a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$$

where,

$a_0a_1a_2a_3a_4a_5a_6a_7$ is a byte of data from the unload box

The four sets of negative, positive, zero, and nonzero lines (one set per PPAUCD card) are input to the CONTAU card for use in the skip taken for stack logic and the branch taken logic.

4-111 Test Box 2 Logic. The test box 2 logic uses CR data in true and complement form and the R field in true form, both from the unload box, when the R field is used as a mask in testing a half byte of data. When the VP code is used as a mask in testing one bit, the test box 2 logic uses CR data in complement form and the decoded VP code in true form, both from the unload box. All four bytes of the CR word are operated on in parallel by the four PPAUCD cards and the CONTAU card uses only the generated signals applicable to the byte specified by the T and N fields of the executing instruction. The test box 2 logic implements the following equation to indicate that the bit specified by the active VP code is set.

$$\text{VP Bit Set} = (a_0+\overline{VP}_0)(a_1+\overline{VP}_1)(a_2+\overline{VP}_2)(a_3+\overline{VP}_3)(a_4+\overline{VP}_4)(a_5+\overline{VP}_5)(a_6+\overline{VP}_6)(a_7+\overline{VP}_7)$$

where,

$VP_0VP_1VP_2VP_3VP_4VP_5VP_6VP_7$ is the decoded VP code

Each term in this equation, except the term representing the active VP, is equal to one because only one VP can be active at any execution time. This means the VP bit set equation is only true when the bit corresponding to the active VP is set, because the VP element of the active VP term is zero. The following equations are implemented to indicate that the bits specified by the R field in the byte of data tested are all zeroes, all ones, any zero, or any one. Left and right half byte equations are necessary because the R field is only four bits.

$$\text{All Zeroes (left half)} = (\bar{a}_0+\bar{R}_0)(\bar{a}_1+\bar{R}_1)(\bar{a}_2+\bar{R}_2)(\bar{a}_3+\bar{R}_3)$$

$$\text{All Zeroes (right half)} = (\bar{a}_4+\bar{R}_0)(\bar{a}_5+\bar{R}_1)(\bar{a}_6+\bar{R}_2)(\bar{a}_7+\bar{R}_3)$$

$$\text{All Ones (left half)} = (a_0+\bar{R}_0)(a_1+\bar{R}_1)(a_2+\bar{R}_2)(a_3+\bar{R}_3)$$

All Ones (right half) = $(a_4+\overline{R}_0)(a_5+\overline{R}_1)(a_6+\overline{R}_2)(a_7+\overline{R}_3)$

Any Zero (left half) = $\overline{a}_0R_0+\overline{a}_1R_1+\overline{a}_2R_2+\overline{a}_3R_3$

Any Zero (right half) = $\overline{a}_4R_0+\overline{a}_5R_1+\overline{a}_6R_2+\overline{a}_7R_3$

Any One (left half) = $a_0R_0+a_1R_1+a_2R_2+a_3R_3$

Any One (right half) = $a_4R_0+a_5R_1+a_6R_2+a_7R_3$

where,

$R_0R_1R_2R_3$ is the R field from the executing instruction

The all zero equations are true only when the ones in the R field match with zeroes in the half byte tested. The all one equations are true only when the ones in the R field match with ones in the half byte tested. The any zero equations are true only when the ones in the R field match with at least one zero in the half byte tested. The any one equations are true only when the ones in the R field match with at least one one in the half byte tested. The four sets of all zero, all one, any zero, any one, and VP bit set lines are input to the CONTAU card for use in the skip taken logic.

4-112 Test Box 3 Logic. The test box 3 logic uses VPR data in true and complement form from the double rail generator when increment or decrement and branch if zero or nonzero instructions are being executed. All four bytes of the VPR word are operated on in parallel by the four PPAUCD cards and the CONTAU card uses only the generated signals applicable to the halfword specified by the R field of the executing instruction. The test box 3 logic implements the following equations to determine when a VPR byte is equal to plus or minus one.

PATS3OP1 = $\overline{b}_0\overline{b}_1\overline{b}_2\overline{b}_3\overline{b}_4\overline{b}_5\overline{b}_6$

PATS3OP7 = b_7

PATS3OM1 = $b_0b_1b_2b_3b_4b_5b_6b_7$

where,

$b_0b_1b_2b_3b_4b_5b_6b_7$ is a byte of data from the double rail generator

A VPR byte of data is equal to plus one only when both the PATS3OP1 and PATS3OP7 equations are true. A VPR byte of data is equal to minus one only when the PATS3OM1 equation is true. The four sets of PATS3OP1, PATS3OP7, and PATS3OM1 lines (one set per PPAUCD card) are input to the CONTAU card for use in the branch taken logic.

4-113 COMPARATOR. The comparator is used during the execution of compare and skip if equal (CE) and compare and skip if not equal (CN) instructions to perform a comparison between data from the aligner and data from the double rail generator. The data from the aligner is supplied by a CR, VPR, or the SWBD of the active VP and the data from the double rail generator is supplied by a VPR. Using the two sources of input data, the comparator implements the following equation on each PPAUCD card to determine if two bytes of data are identical.

$$PAC020 = (a_0b_0+\overline{a}_0\overline{b}_0)(a_1b_1+\overline{a}_1\overline{b}_1)(a_2b_2+\overline{a}_2\overline{b}_2)(a_3b_3+\overline{a}_3\overline{b}_3)$$
$$(a_4b_4+\overline{a}_4\overline{b}_4)(a_5b_5+\overline{a}_5\overline{b}_5)(a_6b_6+\overline{a}_6\overline{b}_6)(a_7b_7+\overline{a}_7\overline{b}_7)$$

where,

$a_0a_1a_2a_3a_4a_5a_6a_7$ is a byte of data from the aligner

$b_0b_1b_2b_3b_4b_5b_6b_7$ is a byte of data from the double rail generator

If all bits in the two compared bytes are the same, all eight terms in the PAC020 equation evaluate to one and PAC020 is true. Four sets of the PAC020 equation and its complement (one set per PPAUCD card) are input to the CONTAU card for use in the skip taken logic.

4-114 DATA MANIPULATOR. The data manipulator is used during the execution of set (SL, SR), reset (RL, RR), test and set (TS), and test and reset (TR) instructions to set or reset (depending on the op-code) bits in a CR byte according to the ones in the R field of the executing instruction. It is also used during the execution of set VP flag (VPS) and reset VP flag (VPR) instructions to set and reset, respectively, a bit in a CR byte according to the decoded VP code. The unload box supplies the data manipulator with CR data in true form and the R field and decoded VP code in both true and complement form. Refer to figure 4-41 for a simplified block diagram of the data manipulator illustrating the four PPAUCD cards operating on a CR word (one byte per PPAUCD card). Each PPAUCD card utilizes a CR byte, the R field of the executing instruction, the decoded VP code, and op-code generated control signals from the CONTAU card to develop the following equations (one equation for each bit in the byte):

$$\text{Bit 0} = CZL\ (a_0\overline{R}_0) + CPR\ (a_0\overline{VP}_0) + COL\ (a_0+R_0) + CPS\ (a_0+VP_0)$$
$$\text{Bit 1} = CZL\ (a_1\overline{R}_1) + CPR\ (a_1\overline{VP}_1) + COL\ (a_1+R_1) + CPS\ (a_1+VP_1)$$
$$\text{Bit 2} = CZL\ (a_2\overline{R}_2) + CPR\ (a_2\overline{VP}_2) + COL\ (a_2+R_2) + CPS\ (a_2+VP_2)$$
$$\text{Bit 3} = CZL\ (a_3\overline{R}_3) + CPR\ (a_3\overline{VP}_3) + COL\ (a_3+R_3) + CPS\ (a_3+VP_3)$$
$$\text{Bit 4} = CZR\ (a_4\overline{R}_0) + CPR\ (a_4\overline{VP}_4) + COR\ (a_4+R_0) + CPS\ (a_4+VP_4)$$

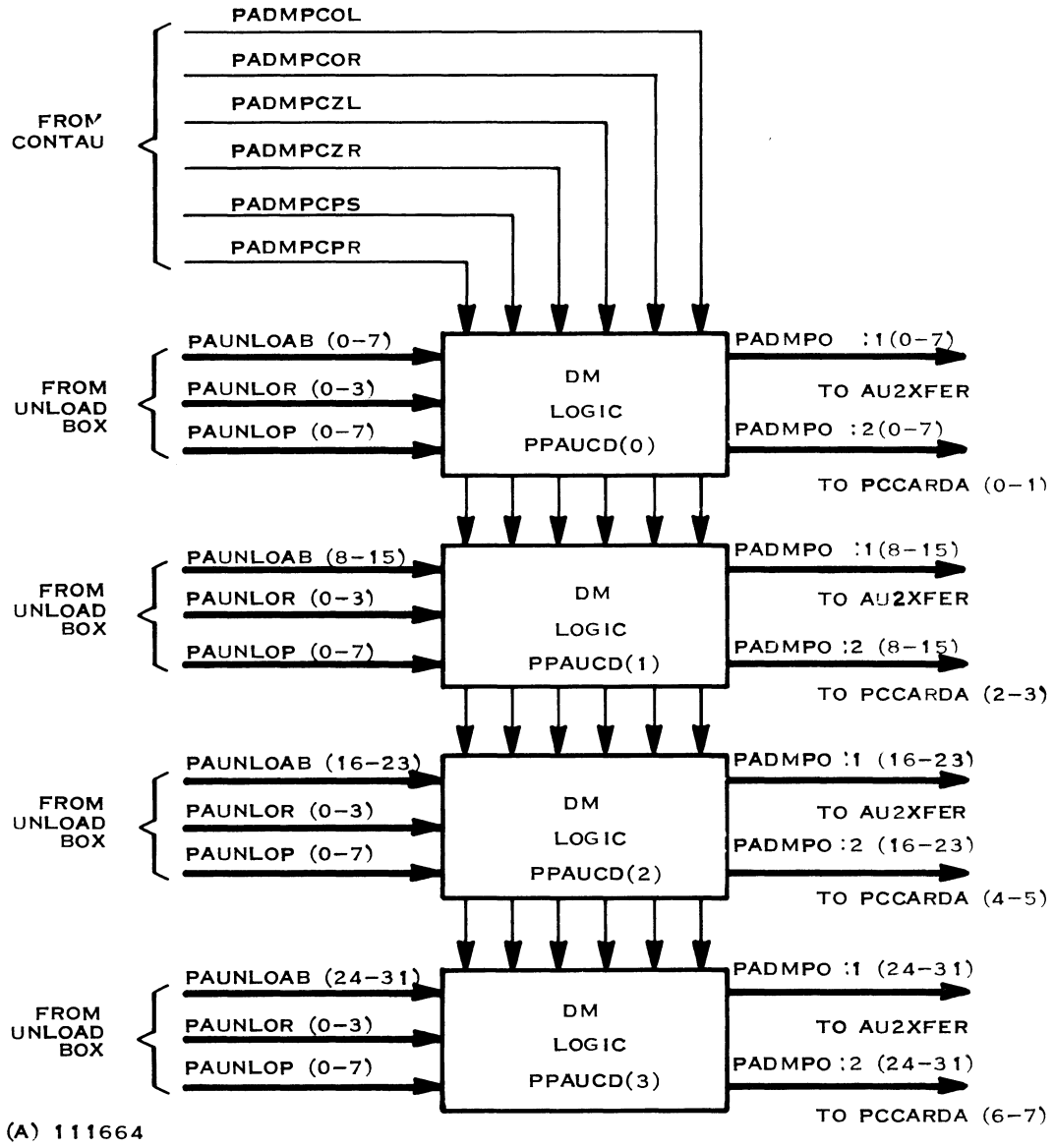


Figure 4-41. Data Manipulator

$$\text{Bit } 5 = CZR\ (a_5\overline{R}_1) + CPR\ (a_5\overline{VP}_5) + COR\ (a_5{+}R_1) + CPS\ (a_5{+}VP_5)$$

$$\text{Bit } 6 = CZR\ (a_6\overline{R}_2) + CPR\ (a_6\overline{VP}_6) + COR\ (a_6{+}R_2) + CPS\ (a_6{+}\ VP_6)$$

$$\text{Bit } 7 = CZR\ (a_7\overline{R}_3) + CPR\ (a_7\overline{VP}_7) + COR\ (a_7{+}R_3) + CPS\ (a_7{+}VP_7)$$

where,

CZL enables the resetting of bits in the left half of a byte according to the R field (TRZL, TROL, and RL instructions)

CZR enables the resetting of bits in the right half of a byte according to the R field (TRZR, TROR, and RR instructions)

COL enables the setting of bits in the left half of a byte according to the R field (TSZL, TSOL, and SL instructions)

COR enables the setting of bits in the right half of a byte according to the R field (TSZR, TSOR, and SR instructions)

CPR enables the resetting of a bit in a byte according to the decoded VP code (VPR instruction)

CPS enables the setting of a bit in a byte according to the decoded VP code (VPS instruction)

The equations for bits 0 through 3 use the left half byte controls and the equations for bits 4 through 7 use the right half byte controls. The first term in all of the equations is used to reset a CR bit if the corresponding R field bit is set; the second term in all of the equations is used to reset a CR bit if the corresponding decoded VP code bit is set; the third term in all of the equations is used to set a CR bit if the corresponding R field bit is set; the fourth term in all of the equations is used to set a CR bit if the corresponding decoded VP code bit is set. Refer to figure 4-42 for a flow chart of the processing accomplished by each of the four types of terms. Four sets of the bit 0 through 7 lines (one set per PPAUCD card) are input to the AU2 transfer bus (AU2B) for presentation to the CR byte specified by the executing instruction.

4-115 SKIP TAKEN AND BRANCH TAKEN LOGIC. The skip taken and branch taken logic consists of skip taken for stack logic, skip taken logic, and branch taken logic (refer to figure 4-43). The skip taken for stack logic is used during the execution of stack instructions to determine whether the next instruction is to be skipped (normal execution of a stack instruction) or not (invalid stack parameter encountered). The branch taken logic is used during the execution of test and branch (TP, TM, TN, and TZ type instructions) and increment or decrement test and branch (IBZ, IBN, DBZ, and DBN instructions) instructions to determine if a branch should be taken to the location specified by the T and N fields (a branch is taken when the test performed is true). The skip taken logic is used during the execution of compare and skip (CE and CN type instructions) and test and skip (TO, TZ (L and R), TA, TS, TR, and VPT type instructions) instructions to determine if the next instruction is to be skipped (a skip is taken when the compare or test performed is true). A detailed description of the skip taken for stack logic, branch taken logic, and stack taken logic is given in the following paragraphs.

4-116 Skip Taken for Stack Logic. The skip taken for stack logic is on the CONTAU card and uses push, pull, and modify stack control lines and test

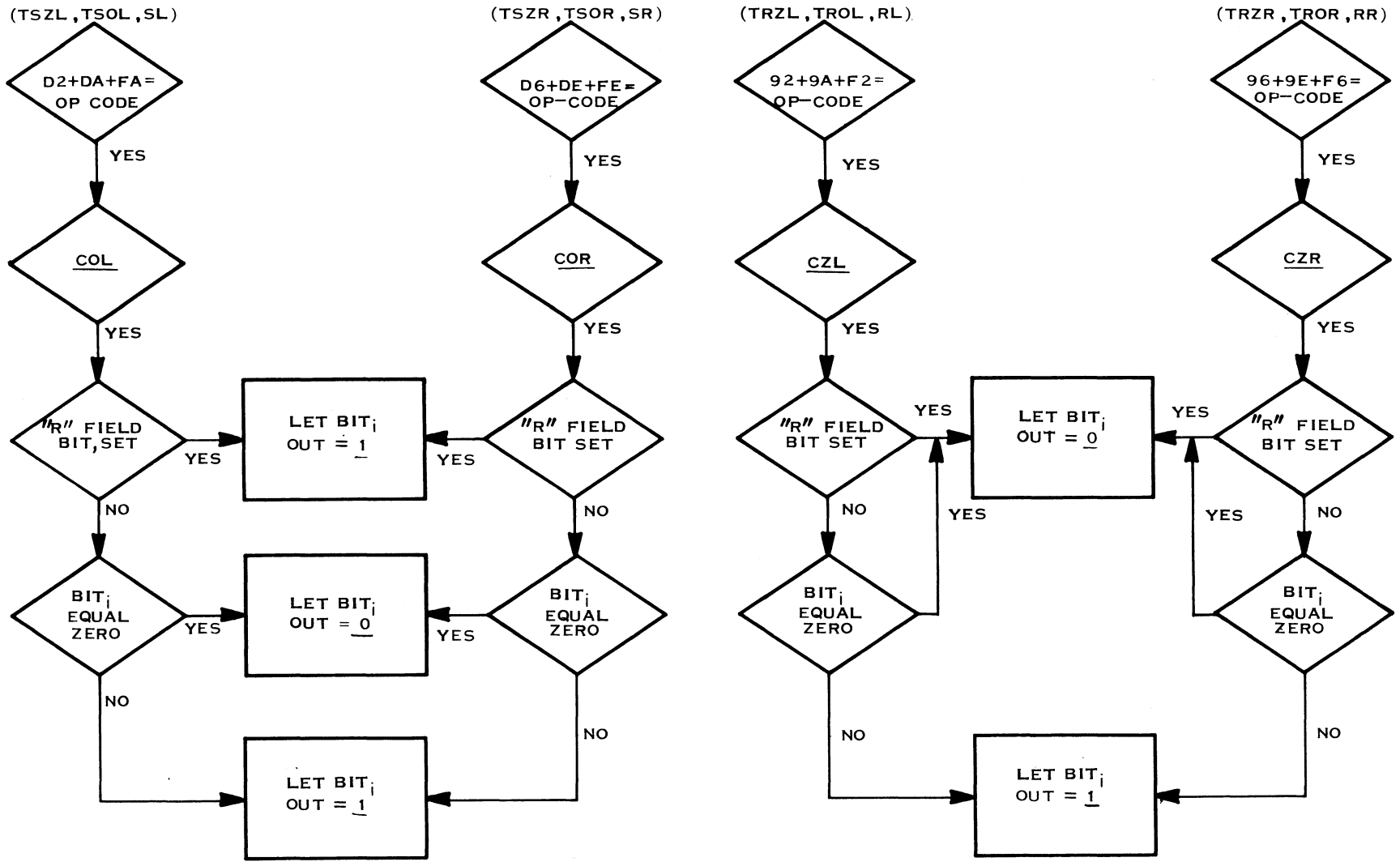


Figure 4-42. Data Manipulator Flow Chart (Sheet 1 of 2)

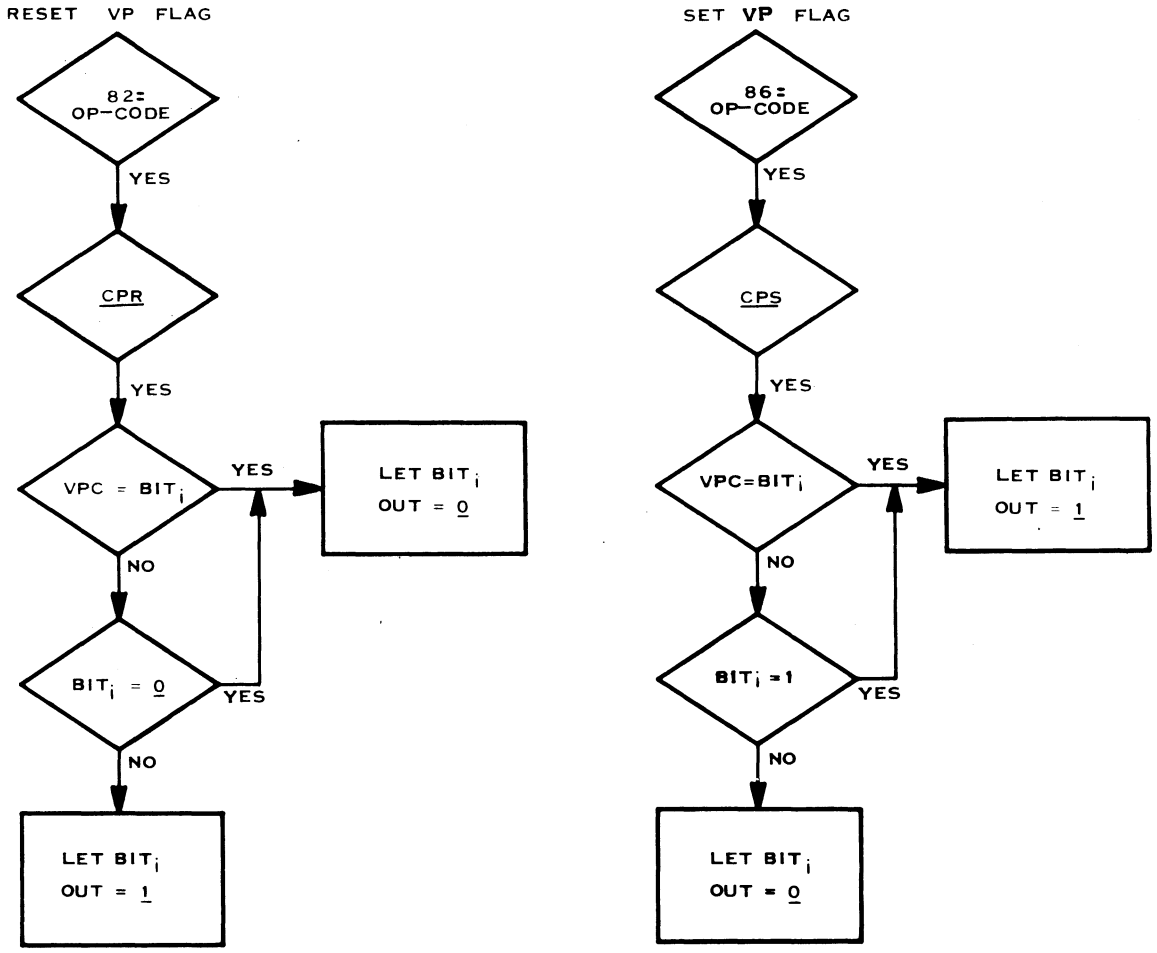


Figure 4-42. Data Manipulator Flow Chart (Sheet 2 of 2)

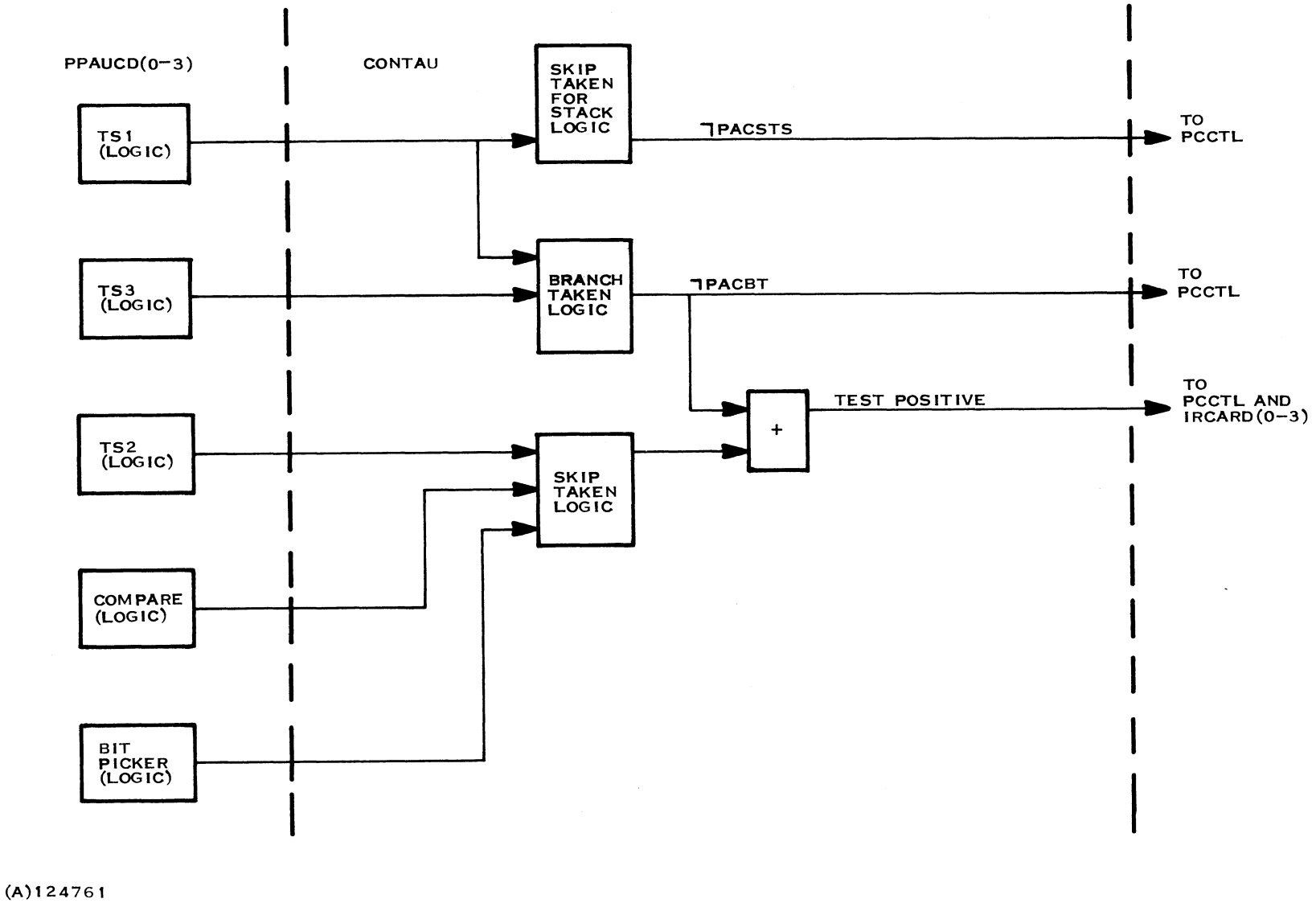


Figure 4-43. Arithmetic Unit Test Functions

signals from the test box 1 logic to determine whether an instruction is to be skipped or not. The following equation is implemented to make this decision.

$$\overline{PACSTS} = (\overline{PUL}+PATS10Z(0))(\overline{PUL}+PATS10Z(1))(\overline{PSH}+PATS10Z(2))$$
$$(\overline{PSH}+PATS10Z(3))(\overline{MDF}+PATS10SN(0)+PATS10SN(2))$$

where,

PUL is true for pull stack instructions in state class 4 (execute) step 2

PSH is true for push stack instructions in state class 4 step 2

MDF is true for modify stack instructions in state class 4 step 2

PATS10Z(i) and PATS10SN(i) are test signals from the test box 1 logic for byte i

The first two terms of the $\overline{PACSTS}$ equation represent the zero check on the word count parameter for pull stack instructions. When the pull stack instruction is executing in state class 4 step 2 and the word count is zero (PATS10Z(0) and PATS10Z(1) are true), the first two terms in the equation, and the equation itself, go to one ($\overline{PACSTS}$ equals one for no skip and zero for skip). When the word count is nonzero, the first and/or second terms in the equation go to zero so that a skip is made. When the push stack instruction is executing in state class 4 step 2 and the space count is zero (PATS10Z(2) and PATS10Z(3) are true), the third and fourth terms in the equation go to one so that no skip is made. When the space count is nonzero, the third and/or fourth terms in the equation force a skip condition. When the modify stack instruction is executing in state class 4 step 2 and either the word count or space count is negative (PATS10SN(0) and/or PATS10SN(2) are true), the last term in the equation goes to one so that no skip is made. When both the word count and space count are non-negative, the last term in the equation forces a skip condition. The $\overline{PACSTS}$ signal is used on the CONTAU card to aid in developing the test positive signal and is also input to the PCCTL card.

4-117 Branch Taken Logic. The branch taken logic is on the CONTAU card and uses test and branch and increment or decrement test and branch instruction control lines, along with test signals from the test box 1 logic and test box 3 logic, to determine if a branch is to be taken to the location specified by the T and N fields. The following equations are implemented to make this decision.

$$\overline{PACBT} = (\overline{PACSF1BT})(\overline{PACSF2BT})(\overline{PACSF3BT})(\overline{PACSF4BT})$$
$$(\overline{PACSF5BT})$$

$$\overline{PACSF1BT} = (\overline{DZ}+\overline{LH}+\overline{(+1)})(\overline{DZ}+\overline{RH}+\overline{(+1)}(\overline{DN}+\overline{LH}+(+1))(\overline{DN}+\overline{RH}+(+1))$$

$$\overline{PACSF2BT} = (\overline{IZ}+\overline{LH}+(\overline{-1}))(\overline{IZ}+\overline{RH}+(\overline{-1}))(\overline{IN}+\overline{LH}+(-1))(\overline{IN}+\overline{RH}+(-1))$$

$$\overline{PACSF3BT} = (\overline{TPB}+\overline{B0}+\overline{POS0})(\overline{TPB}+\overline{B1}+\overline{POS1})(\overline{TPB}+\overline{B2}+\overline{POS2})(\overline{TPB}+\overline{B3}+\overline{POS3})(\overline{TMH}+\overline{LH}+\overline{NEG0})(\overline{TMH}+\overline{RH}+\overline{NEG2})(\overline{TPH}+\overline{LH}+\overline{POS0})(\overline{TPH}+\overline{RH}+\overline{POS2})(\overline{TMW}+\overline{NEG0})(\overline{TPW}+\overline{POS0})$$

$$\overline{PACSF4BT} = (\overline{TNW}+\overline{NZW})(\overline{TNH}+\overline{LH}+\overline{NZLH})(\overline{TNH}+\overline{RH}+\overline{NZRH})(\overline{TZW}+\overline{ZW})(\overline{TZH}+\overline{LH}+\overline{ZLH})(\overline{TZH}+\overline{RH}+\overline{ZRH})$$

$$\overline{PACSF5BT} = (\overline{TNB}+\overline{B0}+\overline{NZB0})(\overline{TNB}+\overline{B1}+\overline{NZB1})(\overline{TNB}+\overline{B2}+\overline{NZB2})(\overline{TNB}+\overline{B3}+\overline{NZB3})(\overline{TZB}+\overline{B0}+\overline{ZB0})(\overline{TZB}+\overline{B1}+\overline{ZB1})(\overline{TZB}+\overline{B2}+\overline{ZB2})(\overline{TZB}+\overline{B3}+\overline{ZB3})(\overline{TMB}+\overline{B0}+\overline{NEG0})(\overline{TMB}+\overline{B1}+\overline{NEG1})(\overline{TMB}+\overline{B2}+\overline{NEG2})(\overline{TMB}+\overline{B3}+\overline{NEG3})$$

where,

DZ is true for DBZ instructions in step 2 of the execute state

DN is true for DBN instructions in step 2 of the execute state

IZ is true for IBZ instructions in step 2 of the execute state

IN is true for IBN instructions in step 2 of the execute state

TPB is true for TPB instructions in step 2 of the execute state

TMH is true for TMH instructions in step 2 of the execute state

TPH is true for TPH instructions in step 2 of the execute state

TMW is true for TM instructions in step 2 of the execute state

TPW is true for TP instructions in step 2 of the execute state

TNW is true for TN instructions in step 2 of the execute state

TNH is true for TNH instructions in step 2 of the execute state

TZW is true for TZ instructions in step 2 of the execute state

TZH is true for TZH instructions in step 2 of the execute state

TNB is true for TNB instructions in step 2 of the execute state

TZB is true for TZB instructions in step 2 of the execute state

TMB is true for TMB instructions in step 2 of the execute state

+1 is true when the test box 3 logic indicates the applicable halfword is equal to plus one

-1 is true when the test box 3 logic indicates the applicable halfword is equal to minus one

LH is true for instructions involving the left half of a word

RH is true for instructions involving the right half of a word

BN is true for instructions involving byte N of a word (N=0, 1, 2, or 3)

POSN is true when the test box 1 logic indicates byte N is positive (N=0, 1, 2, or 3)

NEGN is true when the test box 1 logic indicates byte N is negative (N=0, 1, 2, or 3)

NZN is true when the test box 1 logic indicates a word (N=W), halfword (N=LH or RH), or byte (N=B0, B1, B2, or B3) is nonzero

ZN is true when the test box 1 logic indicates a word (N=W), halfword (N=LH or RH), or byte (N=B0, B1, B2, or B3) is zero

The equation for $\overline{\text{PACBT}}$ goes to zero whenever a branch is to be taken and stays at one if no branch is to be taken. If any one of the five terms in the $\overline{\text{PACBT}}$ equation evaluates to zero, a branch is necessary. The $\overline{\text{PACSF1BT}}$ equation provides the branch or no-branch decision for the DBZ and DBN instructions. When the DBZ instruction is in step 2 of the execute state and the left half of the word being tested is equal to plus one (plus one is the quantity tested for because the VPR halfword to be decremented has not been decremented in step 2 of the execute state), the first term in the $\overline{\text{PACSF1BT}}$ equation goes to zero to cause a branch. If the left half of the word being tested is not plus one, no branch is taken. The second term performs the same function as the first when the right half of a word is being tested for a DBZ instruction. The third and fourth terms provide the branch or no-branch decision for DBN instructions. The $\overline{\text{PACSF2BT}}$ equation provides the branch or no-branch decision for the IBZ and IBN instructions in a manner similar to that described for the DBZ and DBN instructions. Minus one is tested for in the IBZ and IBN instructions because the VPR halfword to be incremented has not been incremented in step 2 of the execute state. The $\overline{\text{PACSF3BT}}$ equation provides the branch or no-branch decision for the TPB, TMH, TPH, TM, and TP instructions. The first four terms apply to the four possible bytes that can be used by the TPB instruction. The fifth and sixth terms apply to the two possible halfwords that can be used by the TMH instruction and the seventh and eighth terms apply to the two possible halfwords that can be used by the TPH instruction. The next to last and last terms apply to the TM and TP instructions, respectively. The $\overline{\text{PACSF4BT}}$ equation provides the branch or no-branch decision for the TN, TNH, TZ, and TZH instructions and the $\overline{\text{PACSF5BT}}$ equation provides the branch or no-branch decision for the TNB, TZB, and TMB instructions. In the last three equations, the term corresponding to the instruction being performed goes to zero when the quantity being tested for is found. The $\overline{\text{PACBT}}$ signal is used on the CONTAU card to aid in developing the test positive signal and is also input to the PCCTL card.

4-118 Skip Taken Logic. The skip taken logic is on the CONTAU card and uses compare and skip and test and skip instruction control lines along with test signals from the test box 2 logic, comparator, and bit picker to determine if the next instruction is to be skipped. The following equations are implemented to make this decision.

$$\overline{\text{Skip Taken}} = (\overline{\text{PACSF1ST}})(\overline{\text{PACSF2ST}})(\overline{\text{PACSF3ST}})(\overline{\text{PACSF4ST}})(\overline{\text{PACSF5ST}})(\overline{\text{PACSF6ST}})(\overline{\text{PACSF7ST}})(\overline{\text{PACSTS}})$$

$$\begin{aligned}\overline{\text{PACSF1ST}} = {} & (\overline{\text{TSOL}}+\overline{\text{B0}}+\overline{\text{AYOLB0}})(\overline{\text{TSOL}}+\overline{\text{B1}}+\overline{\text{AYOLB1}})\\ & (\overline{\text{TSOL}}+\overline{\text{B2}}+\overline{\text{AYOLB2}})(\overline{\text{TSOL}}+\overline{\text{B3}}+\overline{\text{AYOLB3}})\\ & (\overline{\text{TSOR}}+\overline{\text{B0}}+\overline{\text{AYORB0}})(\overline{\text{TSOR}}+\overline{\text{B1}}+\overline{\text{AYORB1}})\\ & (\overline{\text{TSOR}}+\overline{\text{B2}}+\overline{\text{AYORB2}})(\overline{\text{TSOR}}+\overline{\text{B3}}+\overline{\text{AYORB3}})\\ & (\overline{\text{TSZL}}+\overline{\text{B0}}+\overline{\text{AYZLB0}})(\overline{\text{TSZL}}+\overline{\text{B1}}+\overline{\text{AYZLB1}})\\ & (\overline{\text{TSZL}}+\overline{\text{B2}}+\overline{\text{AYZLB2}})(\overline{\text{TSZL}}+\overline{\text{B3}}+\overline{\text{AYZLB3}})\end{aligned}$$

$$\begin{aligned}\overline{\text{PACSF2ST}} = {} & (\overline{\text{TSZR}}+\overline{\text{B0}}+\overline{\text{AYZRB0}})(\overline{\text{TSZR}}+\overline{\text{B1}}+\overline{\text{AYZRB1}})\\ & (\overline{\text{TSZR}}+\overline{\text{B2}}+\overline{\text{AYZRB2}})(\overline{\text{TSZR}}+\overline{\text{B3}}+\overline{\text{AYZRB3}})\\ & (\overline{\text{TPOL}}+\overline{\text{B0}}+\overline{\text{AYOB0}})(\overline{\text{TPOL}}+\overline{\text{B1}}+\overline{\text{AYOB1}})\\ & (\overline{\text{TPOL}}+\overline{\text{B2}}+\overline{\text{AYOB2}})(\overline{\text{TPOL}}+\overline{\text{B3}}+\overline{\text{AYOB3}})\end{aligned}$$

$$\begin{aligned}\overline{\text{PACSF3ST}} = {} & (\overline{\text{CWN}}+\overline{\text{CN0}}(\overline{\text{CN1}})\overline{\text{CN2}}(\overline{\text{CN3}}))(\overline{\text{CRHN}}+\overline{\text{LH}}+\overline{\text{CN0}}(\overline{\text{CN1}}))\\ & (\overline{\text{CRHN}}+\overline{\text{RH}}+(\overline{\text{CN2}})\overline{\text{CN3}})(\overline{\text{CHE}}+\overline{\text{LH}}+\overline{\text{CE0}}+\overline{\text{CE1}})\\ & (\overline{\text{CHE}}+\overline{\text{RH}}+\overline{\text{CE2}}+\overline{\text{CE3}})(\overline{\text{CWE}}+\overline{\text{CE0}}+\overline{\text{CE1}}+\overline{\text{CE2}}+\overline{\text{CE3}})\end{aligned}$$

$$\begin{aligned}\overline{\text{PACSF4ST}} = {} & (\overline{\text{TSAZL}}+\overline{\text{B0}}+\overline{\text{ALZLB0}})(\overline{\text{TSAZL}}+\overline{\text{B1}}+\overline{\text{ALZLB1}})\\ & (\overline{\text{TSAZL}}+\overline{\text{B2}}+\overline{\text{ALZLB2}})(\overline{\text{TSAZL}}+\overline{\text{B3}}+\overline{\text{ALZLB3}})\\ & (\overline{\text{TSAZR}}+\overline{\text{B0}}+\overline{\text{ALZRB0}})(\overline{\text{TSAZR}}+\overline{\text{B1}}+\overline{\text{ALZRB1}})\\ & (\overline{\text{TSAZR}}+\overline{\text{B2}}+\overline{\text{ALZRB2}})(\overline{\text{TSAZR}}+\overline{\text{B3}}+\overline{\text{ALZR33}})\\ & (\overline{\text{TSAOL}}+\overline{\text{B0}}+\overline{\text{ALOLB0}})(\overline{\text{TSAOL}}+\overline{\text{B1}}+\overline{\text{ALOLB1}})\\ & (\overline{\text{TSAOL}}+\overline{\text{B2}}+\overline{\text{ALOLB2}})(\overline{\text{TSAOL}}+\overline{\text{B3}}+\overline{\text{ALOLB3}})\end{aligned}$$

$$\begin{aligned}\overline{\text{PACSF5ST}} = {} & (\overline{\text{CIEW}}+\overline{\text{CE0}}+\overline{\text{CE1}}+\overline{\text{CE2}}+\overline{\text{CE3}})(\overline{\text{CIEH}}+\overline{\text{LH}}+\overline{\text{CE0}}+\overline{\text{CE1}})\\ & (\overline{\text{CIEH}}+\overline{\text{RH}}+\overline{\text{CE2}}+\overline{\text{CE3}})(\overline{\text{CIEB}}+\overline{\text{B0}}+\overline{\text{CE0}})(\overline{\text{CIEB}}+\overline{\text{B1}}+\overline{\text{CE1}})\\ & (\overline{\text{CIEB}}+\overline{\text{B2}}+\overline{\text{CE2}})(\overline{\text{CIEB}}+\overline{\text{B3}}+\overline{\text{CE3}})(\overline{\text{CINB}}+\overline{\text{B0}}+\overline{\text{CN0}})\\ & (\overline{\text{CINB}}+\overline{\text{B1}}+\overline{\text{CN1}})(\overline{\text{CINB}}+\overline{\text{B2}}+\overline{\text{CN2}})(\overline{\text{CINB}}+\overline{\text{B3}}+\overline{\text{CN3}})\end{aligned}$$

$$\begin{aligned}\overline{\text{PACSF6ST}} = {} & (\overline{\text{TSAOR}}+\overline{\text{B0}}+\overline{\text{ALORB0}})(\overline{\text{TSAOR}}+\overline{\text{B1}}+\overline{\text{ALORB1}})\\ & (\overline{\text{TSAOR}}+\overline{\text{B2}}+\overline{\text{ALORB2}})(\overline{\text{TSAOR}}+\overline{\text{B3}}+\overline{\text{ALORB3}})\\ & (\overline{\text{CINW}}+\overline{\text{CN0}}(\overline{\text{CN1}})\overline{\text{CN2}}(\overline{\text{CN3}}))(\overline{\text{CINH}}+\overline{\text{LH}}+(\overline{\text{CN0}})\overline{\text{CN1}})\\ & (\overline{\text{CINH}}+\overline{\text{RH}}+(\overline{\text{CN2}})\overline{\text{CN3}})\end{aligned}$$

$$\begin{aligned}\overline{\text{PACSF7ST}} = {} & (\overline{\text{VPTZ}}+\overline{\text{B0}}+\overline{\text{VPZB0}})(\overline{\text{VPTZ}}+\overline{\text{B1}}+\overline{\text{VPZB1}})\\ & (\overline{\text{VPTZ}}+\overline{\text{B2}}+\overline{\text{VPZB2}})(\overline{\text{VPTZ}}+\overline{\text{B3}}+\overline{\text{VPZB3}})\\ & (\overline{\text{VPTO}}+\overline{\text{B0}}+\overline{\text{VPOB0}})(\overline{\text{VPTO}}+\overline{\text{B1}}+\overline{\text{VPOB1}})\\ & (\overline{\text{VPTO}}+\overline{\text{B2}}+\overline{\text{VPOB2}})(\overline{\text{VPTO}}+\overline{\text{B3}}+\overline{\text{VPOB3}})\end{aligned}$$

where,

$\overline{\text{PACSTS}}$ is the signal generated by the skip taken for stack logic

TSOL is true for TSOL, TOL, and TROL instructions in step 2 of the execute state

TSOR is true for TSOR, TOR, and TROR instructions in step 2 of the execute state

TSZL is true for TSZL, TZL, and TRZL instructions in step 2 of the execute state

TSZR is true for TSZR, TZR, and TRZR instructions in step 2 of the execute state

TSAZL is true for TAZL instructions in step 2 of the execute state

TSAZR is true for TAZR instructions in step 2 of the execute state

TSAOL is true for TAOL instructions in step 2 of the execute state

TSAOR is true for TAOR instructions in step 2 of the execute state

TPOL is true for POLL instructions in step 2 of the execute state

VPTO is true for VPTO instructions in step 2 of the execute state

VPTZ is true for VPTZ instructions in step 2 of the execute state

CWN is true for CN instructions in step 2 of the execute state

CWE is true for CE instructions in step 2 of the execute state

CHE is true for CEH, CER, and CEL instructions in step 2 of the execute state

CRHN is true for CNH, CNL, and CNR instructions in step 2 of the execute state

CIEW is true for CEI instructions in step 2 of the execute state

CINW is true for CNI instructions in step 2 of the execute state

CIEH is true for CEHI instructions in step 2 of the execute state

CIEB is true for CEBI and CEB instructions in step 2 of the execute state

CINB is true for CNB and CNBI instructions in step 2 of the execute state

CINH is true for CNHI instructions in step 2 of the execute state

LH is true for instructions involving the left half of a word

RH is true for instructions involving the right half of a word

BN is true for instructions involving byte N of a word (N=0, 1, 2, or 3)

CEN is true when the comparator indicates byte N of the compared quantities are identical (N=0, 1, 2, or 3)

CNX is true when the comparator indicates byte X of the compared quantities are different (X=0, 1, 2, or 3)

AYORBN is true when the test box 2 logic indicates the right half of byte N has a one in one of the bit positions marked by the R field (N=0, 1, 2, or 3)

AYOLBN is true when the test box 2 logic indicates the left half of byte N has a one in one of the bit positions marked by the R field (N=0, 1, 2, or 3)

AYZLBN is true when the test box 2 logic indicates the left half of byte N has a zero in one of the bit positions marked by the R field (N=0, 1, 2, or 3)

AYZRBN is true when the test box 2 logic indicates the right half of byte N has a zero in one of the bit positions marked by the R field (N=0, 1, 2, or 3)

ALORBN is true when the test box 2 logic indicates the right half of byte N has a one in each bit position marked by the R field (N=0, 1, 2, or 3)

ALOLBN is true when the test box 2 logic indicates the left half of byte N has a one in each bit position marked by the R field (N=0, 1, 2, or 3)

ALZRBN is true when the test box 2 logic indicates the right half of byte N has a zero in each bit position marked by the R field (N=0, 1, 2, or 3)

ALZLBN is true when the test box 2 logic indicates the left half of byte N has a zero in each bit position marked by the R field (N=0, 1, 2, or 3)

AYOBN is true when the bit picker indicates byte N has at least one one (N=0, 1, 2, or 3)

VPZBN is true when the test box 2 logic indicates the bit in byte N marked by the decoded VP code is zero (N=0, 1, 2, or 3)

VPOBN is true when the test box 2 logic indicates the bit in byte N marked by the decoded VP code is one (N=0, 1, 2, or 3)

The equation for $\overline{\text{Skip Taken}}$ goes to zero whenever an instruction is to be skipped and stays at one if no skip is to be made. If any one of the eight terms in the $\overline{\text{Skip Taken}}$ equation evaluates to zero, a skip is taken. The $\overline{\text{PACSF1ST}}$ equation provides the skip or no-skip decision for TSOL, TROL, TOL, TSOR, TROR, TOR, TSZL, TRZL, and TZL instructions. When a TSOL, TROL, or TOL instruction is in step 2 of the execute state and the left half of byte zero has at least one logical one in the bit position(s) marked by the R field, the first term in the $\overline{\text{PACSF1ST}}$ equation goes to zero to cause

a skip. If no one(s) are found during the test, no skip is made. The second, third, and fourth terms of the same equation provide the skip or no-skip decision for the same instructions when byte one, two, and three, respectively, are used. The middle four terms of the equation provide the skip or no-skip decision for the TSOR, TROR, and TOR instructions and the last four terms provide the skip or no-skip decision for the TSZL, TRZL, and TZL instructions. The first four terms of the $\overline{\text{PACSF2ST}}$ equation apply to the TSZR, TRZR, and TZR instructions and the last four terms apply to the POLL instruction. The first term of the $\overline{\text{PACSF3ST}}$ equation applies to the CN instruction; the second and third terms apply to the CNH, CNL, and CNR instructions; the fourth and fifth terms apply to the CEH, CER, and CEL instructions; and the last term applies to the CE instruction. The first four terms of the $\overline{\text{PACSF4ST}}$ equation apply to the TAZL instruction; the middle four terms apply to the TAZR instruction; and the last four terms apply to the TAOL instruction. The first term of the $\overline{\text{PACSF5ST}}$ equation applies to the CEI instruction; the second and third terms apply to the CEHI instruction; the fourth through seventh terms apply to the CEBI and CEB instructions; and the last four terms apply to the CNB and CNBI instructions. The first four terms in the $\overline{\text{PACSF6ST}}$ equation apply to the TAOR instruction, the fifth term applies to the CNI instruction, and the last two terms apply to the CNHI instruction. The first four terms in the $\overline{\text{PACSF7ST}}$ equation apply to the VPTZ instruction and the last four terms apply to the VPTO instruction. The developed $\overline{\text{Skip Taken}}$ equation is used on the CONTAU card to aid in developing the test positive signal.

The test positive signal is developed utilizing the results of the branch taken logic and skip taken (including the skip taken for stack) logic in the following manner:

$$\text{PACTSPOS} = \overline{\overline{\text{PACBT}} \cdot \overline{\text{Skip Taken}}} = \text{PACBT} + \text{Skip Taken}$$

The PACTSPOS signal is used on the PCCTL and IRCARD(0-3) cards.

4-119 AU CONTROL. Refer to figure 4-44 for a simplified block diagram of the AU control on the CONTAU card. The branch and skip test translator block was discussed in detail in paragraphs 4-115 through 4-118. The remainder of the logic on the CONTAU card is functionally divided and discussed in detail in the following paragraphs.

4-120 Aligner Control. Refer to figure 4-45 for a simplified block diagram of the aligner control logic. The inputs to the aligner control logic ($\overline{\text{PALALIGN}}$, $\overline{\text{PALRSWB(0-3)}}$, and $\overline{\text{PALWSWB(0-3)}}$) are supplied by the AUMIR portion of the MIR via the transformation logic on the VPRCONT card. The $\overline{\text{PALALIGN}}$ signal indicates when alignment of data from the MDB is necessary and is used on the CONTAU card to set a flip-flop and enable

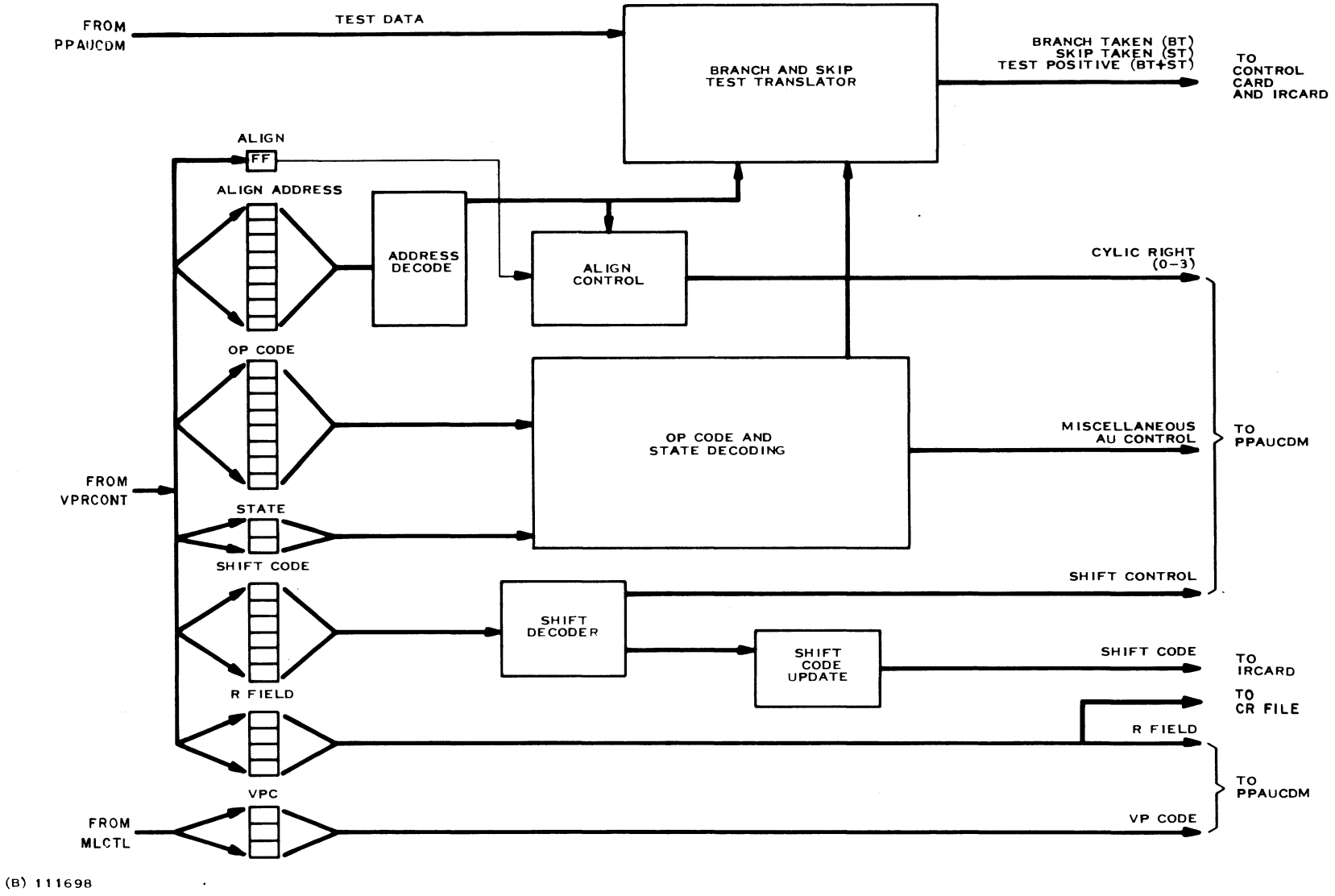


Figure 4-44. AU Control on CONTAU

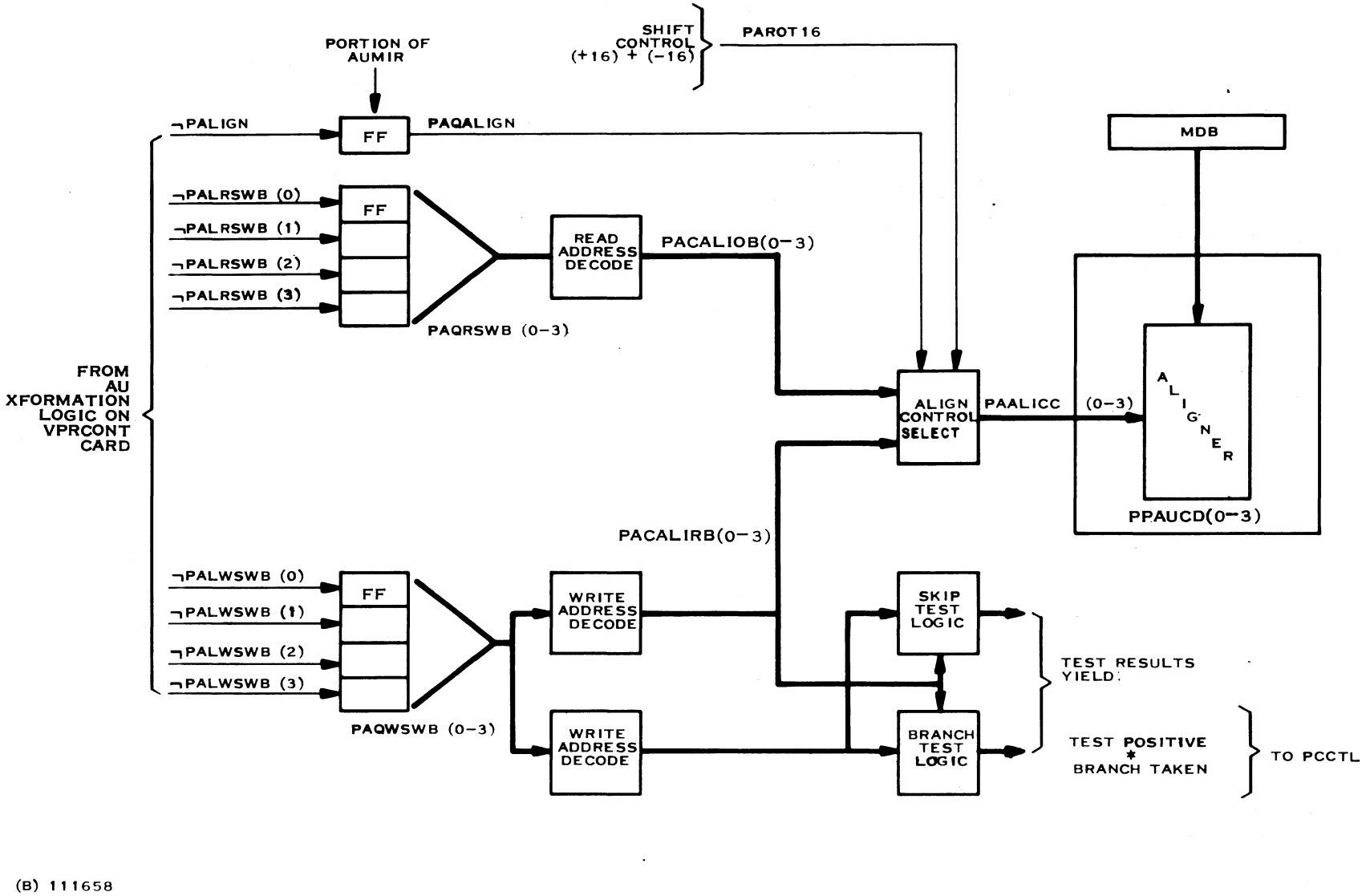


Figure 4-45. Aligner Control Logic on CONTAU

the aligner control logic. The $\overline{\text{PALRSWB(0-3)}}$ signals combine to represent a word (one bit represents a byte) of data that requires aligning (align object). The $\overline{\text{PALWSWB(0-3)}}$ signals combine to represent the result of the aligning procedure (align reference). Both of these sets of inputs are temporarily stored (for the duration of the current clock) for decoding purposes. Refer to figure 4-46 for the possible relationships between the $\overline{\text{PALRSWB(0-3)}}$ and $\overline{\text{PALWSWB(0-3)}}$ signals. The read address decode logic on CONTAU utilizes the $\overline{\text{PAQRSWB(0-3)}}$ signals to develop the following equations:

$$\text{PACALIOB(0)} = (\text{PAQRSWB(0)})(\overline{\text{PAQRSWB(1)}})(\overline{\text{PAQRSWB(2)}})(\overline{\text{PAQRSWB(3)}})$$

$$\text{PACALIOB(1)} = (\overline{\text{PAQRSWB(0)}})(\text{PAQRSWB(1)})(\overline{\text{PAQRSWB(2)}})(\overline{\text{PAQRSWB(3)}})$$

$$\text{PACALIOB(2)} = (\overline{\text{PAQRSWB(0)}})(\overline{\text{PAQRSWB(1)}})(\text{PAQRSWB(2)})(\overline{\text{PAQRSWB(3)}})$$

$$\text{PACALIOB(3)} = (\overline{\text{PAQRSWB(0)}})(\overline{\text{PAQRSWB(1)}})(\overline{\text{PAQRSWB(2)}})(\text{PAQRSWB(3)})$$

$$\text{PACALIOH(0)} = (\text{PAQRSWB(0)})(\text{PAQRSWB(1)})(\overline{\text{PAQRSWB(2)}})(\overline{\text{PAQRSWB(3)}})$$

$$\text{PACALIOH(1)} = (\overline{\text{PAQRSWB(0)}})(\overline{\text{PAQRSWB(1)}})(\text{PAQRSWB(2)})(\text{PAQRSWB(3)})$$

The write address decode logic on CONTAU uses the $\overline{\text{PAQWSWB(0-3)}}$ signals to develop two sets (two sets are required for fan out purposes) of the following equations:

$$\text{PACALIRB(0)} = (\text{PAQWSWB(0)})(\overline{\text{PAQWSWB(1)}})(\overline{\text{PAQWSWB(2)}})(\overline{\text{PAQWSWB(3)}})$$

$$\text{PACALIRB(1)} = (\overline{\text{PAQWSWB(0)}})(\text{PAQWSWB(1)})(\overline{\text{PAQWSWB(2)}})(\overline{\text{PAQWSWB(3)}})$$

$$\text{PACALIRB(2)} = (\overline{\text{PAQWSWB(0)}})(\overline{\text{PAQWSWB(1)}})(\text{PAQWSWB(2)})(\overline{\text{PAQWSWB(3)}})$$

$$\text{PACALIRB(3)} = (\overline{\text{PAQWSWB(0)}})(\overline{\text{PAQWSWB(1)}})(\overline{\text{PAQWSWB(2)}})(\text{PAQWSWB(3)})$$

$$\text{PACALIRH(0)} = (\text{PAQWSWB(0)})(\text{PAQWSWB(1)})(\overline{\text{PAQWSWB(2)}})(\overline{\text{PAQWSWB(3)}})$$

$$\text{PACALIRH(1)} = (\overline{\text{PAQWSWB(0)}})(\overline{\text{PAQWSWB(1)}})(\text{PAQWSWB(2)})(\text{PAQWSWB(3)})$$

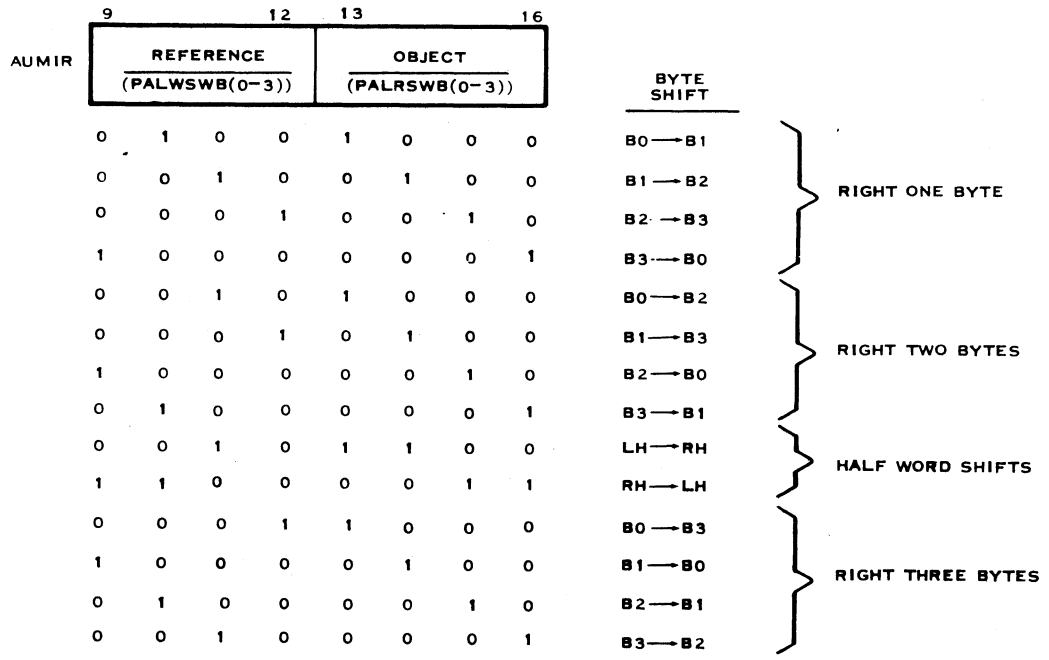


Figure 4-46. Aligner Control Inputs ($\overline{\text{PALWSWB(0-3)}}$ and $\overline{\text{PALRSWB(0-3)}}$)

The PACALIRB(0-3) and PACALIRH(0, 1) signals from the write address decode logic are used in the skip taken and branch taken logic to indicate the byte and half word of data, respectively, under test. The PACALIOB(0-3), PACALIOH(0, 1), PACALIRB(0-3), and PACALIRH(0, 1) signals are all input to the aligner control select logic along with the left and right 16-bit shift control lines. These inputs are used by the aligner control select logic to develop the no alignment (PAALICC0), right shift one byte (PAALICC1), right shift two bytes (PAALICC2), and right shift three bytes (PAALICC3) commands for use by the aligner on the PPAUCD(0-3) cards. The following equations are implemented to generate these commands.

$$\text{PAALICC0} = (\overline{\text{PAALICC1}})(\overline{\text{PAALICC2}})(\overline{\text{PAALICC3}})$$

$$\text{PAALICC1} = (\text{RB0}\cdot\text{OB3}+\text{RB1}\cdot\text{OB0}+\text{RB2}\cdot\text{OB1}+\text{RB3}\cdot\text{OB2})\text{PAQALIGN}$$

$$\text{PAALICC2} = (\text{RB0}\cdot\text{OB2}+\text{RB1}\cdot\text{OB3}+\text{RB2}\cdot\text{OB0}+\text{RB3}\cdot\text{OB1}+\text{RH0}\cdot\text{OH1}+\text{RH1}\cdot\text{OH0})\text{PAQALIGN}+\text{PAROT16}$$

$$\text{PAALICC3} = (\text{RB0}\cdot\text{OB1}+\text{RB1}\cdot\text{OB2}+\text{RB2}\cdot\text{OB3}+\text{RB3}\cdot\text{OB0})\text{PAQALIGN}$$

$$\text{PAROT16} = \text{SHP16}+\text{SHM16}$$

where,

RBN equals PACALIRB(N)

OBN equals PACALIOB(N)

RHN equals PACALIRH(N)

OHN equals PACALIOH(N)

SHP16 is true for 16-bit cyclic shifts to the left

SHM16 is true for 16-bit cyclic shifts to the right

The PAALICC(0-3) signals are input to the four PPAUCDM cards where they are used by the aligner to control the byte increment data shift.

4-121 Shifter Control. Refer to figure 4-47 for a simplified block diagram of the shifter control logic. Inputs to the shifter control logic include a shift operand (indicates shift count and direction) from the AUMIR and the MIR op-code (indicates an arithmetic, logical, or cyclic shift). The six-bit shift operand is gated into a set of flip-flops with the PAPUGAT:3 signal for use in the shift decode logic and the shift code update logic. The shift decode logic uses the shift operand data (PAQSHFBI(0-5)) in true and complement form and the decoded cyclic shift control lines to generate add ($\overline{\text{PACSHP16}}$, $\overline{\text{PACSHP8}}$, $\overline{\text{PACSHP4}}$, and $\overline{\text{PACSHP1}}$) and subtract ($\overline{\text{PACSHM16}}$, $\overline{\text{PACSHM8}}$, $\overline{\text{PACSHM4}}$, and $\overline{\text{PACSHM1}}$) shift control lines according to the map in figure 4-48. The add and subtract terms apply to the use of these signals in the shift code update logic. The shift decode logic also generates a right shift line ($\overline{\text{PAQSHFBI(0)}}$) by inverting the sign bit of the negated shift operand (the sign is positive for left shifts and negative for right shifts) and a zero shift line by AND'ing the individually negated bits of the shift operand. The 16-bit add and subtract cyclic shift lines ($\overline{\text{PACSHP16}}$ and $\overline{\text{PACSHM16}}$) and the zero shift line combine to develop the special shift line (PASHFCSP) used in gating the aligner output through the shifter. The 16-bit add and subtract cyclic shift lines are also used to develop the negated PAROT16 signal used in generating the two byte shift command in the aligner control logic. The 8-bit add and subtract shift lines ($\overline{\text{PACSHP8}}$ and $\overline{\text{PACSHM8}}$) combine to develop the 8-bit shift line (PASHFC8); the 4-bit add and subtract shift lines ($\overline{\text{PACSHP4}}$ and $\overline{\text{PACSHM4}}$) combine to develop the 4-bit shift line (PASHFC4); the single bit add and subtract shift lines ($\overline{\text{PACSHP1}}$ and $\overline{\text{PACSHM1}}$) combine to develop the single bit shift line (PASHFC1); the right shift line ($\overline{\text{PAQSHFBI(0)}}$) is inverted to develop the left shift line (PASHFCL). The developed shift control lines (PASHFCSP, PASHFC8, PASHFC4, PASHFC1, and PASHFCL) are distributed to the four PPAUCD cards, where they control the shifting.

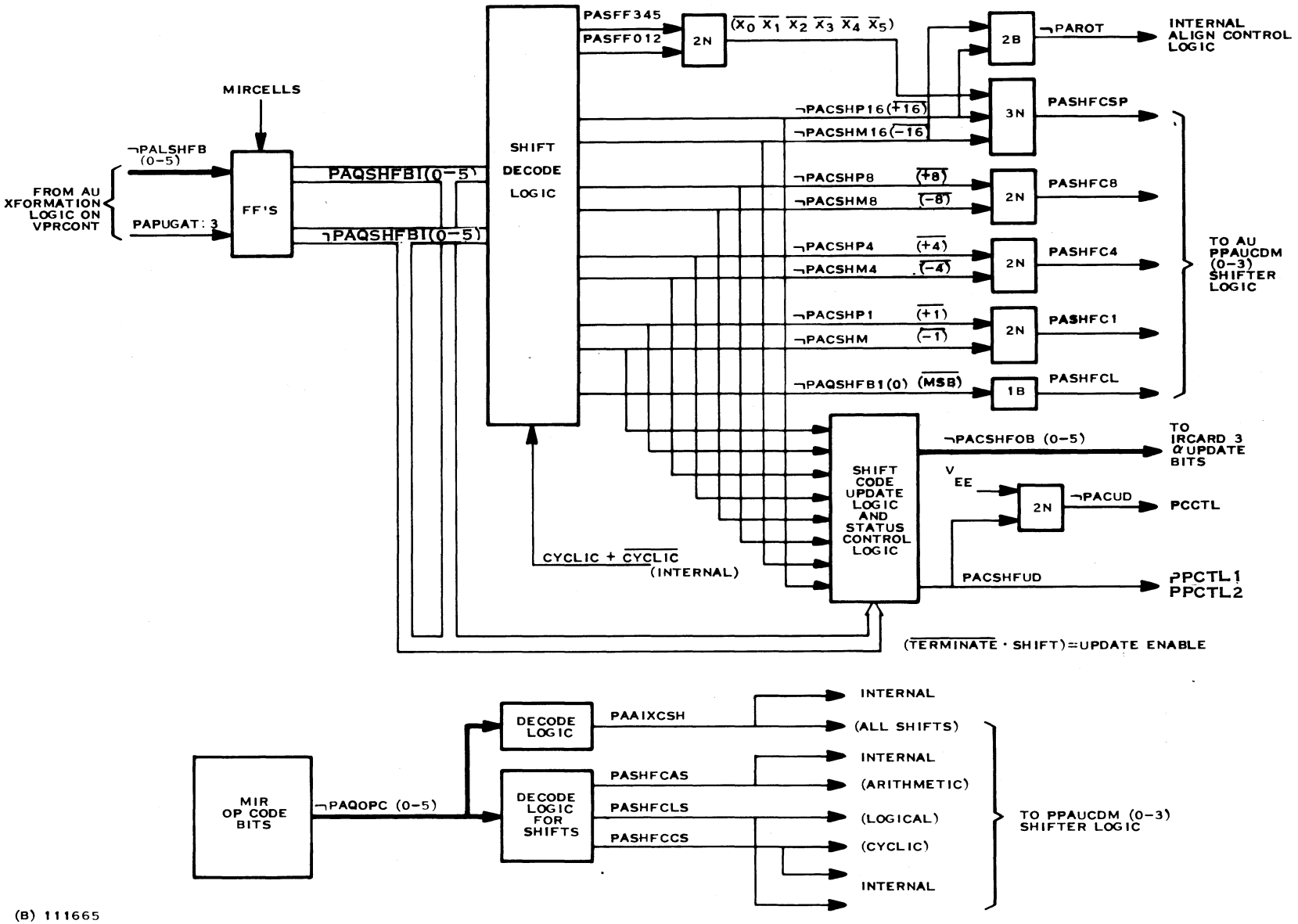
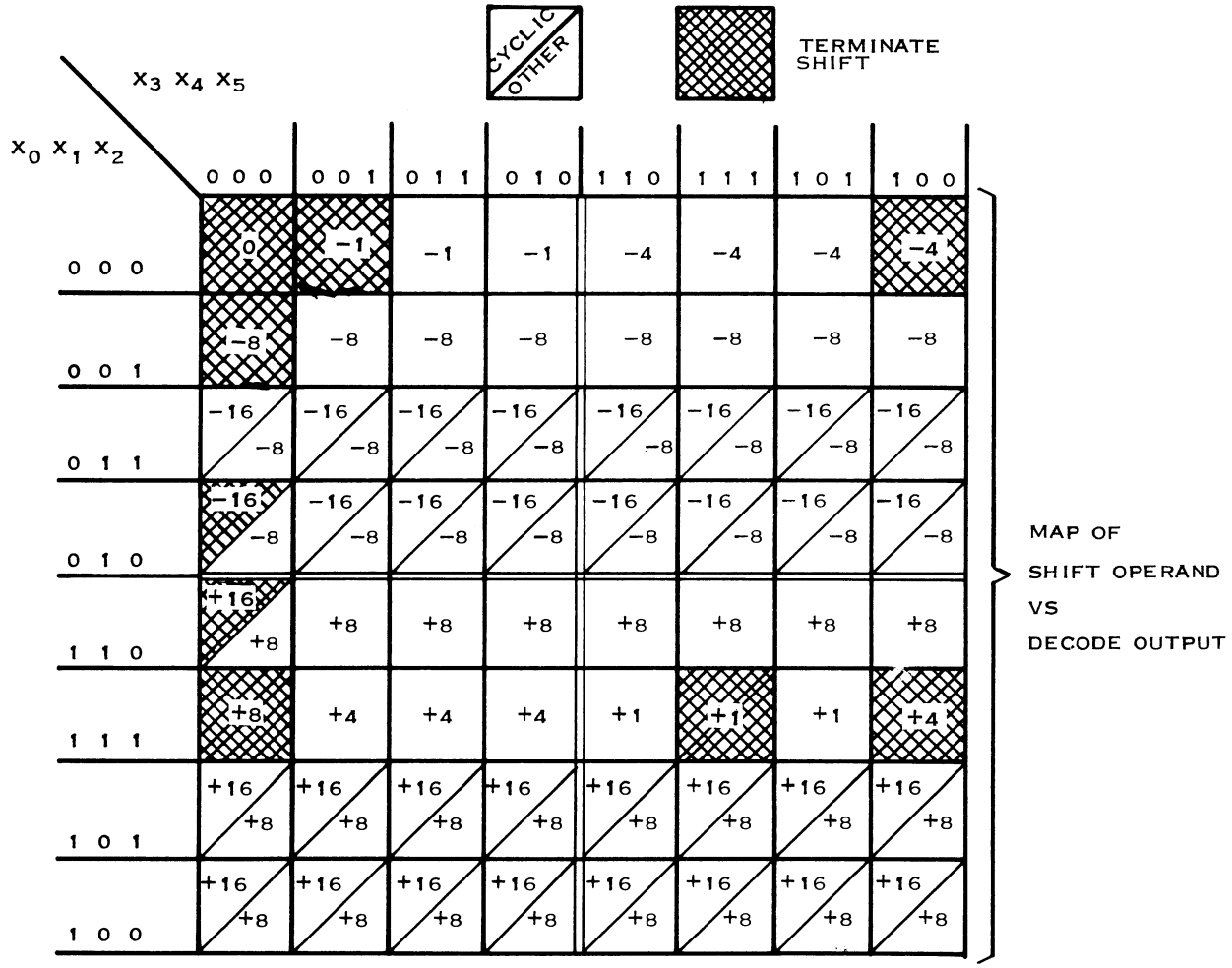


Figure 4-47. Shift Control Logic on CONTAU



ADD $16 = CS\ (x_0\ \overline{x_1} + x_0\ \overline{x_2}\ \overline{x_3}\ \overline{x_4}\ \overline{x_5})$

SUB $16 = CS\ (\overline{x_0}\ x_1)$

ADD $8 = \overline{CS}\ (x_0\ \overline{x_1} + x_0\ \overline{x_2}\ \overline{x_3}\ \overline{x_4}\ \overline{x_5}) + x_0\ x_1\ x_2\ \overline{(\overline{x_3}\ \overline{x_4}\ \overline{x_5})} + x_0\ x_1\ x_2\ \overline{x_3}\ \overline{x_4}$

SUB $8 = \overline{CS}\ (\overline{x_0}\ x_1) + \overline{x_0}\ \overline{x_1}\ x_2$

ADD $4 = x_0\ x_1\ x_2\ (\overline{x_3}\ x_5 + \overline{x_3}\ x_4 + x_3\ \overline{x_4}\ \overline{x_5})$

SUB $4 = \overline{x_0}\ \overline{x_1}\ \overline{x_2}\ x_3$

ADD $1 = x_0\ x_1\ x_2\ x_3\ (x_4 + x_5)$

SUB $1 = \overline{x_0}\ \overline{x_1}\ \overline{x_2}\ \overline{x_3}\ (x_4 + x_5)$

(A) 111668

Figure 4-48. Shift Operand/Shift Decode Logic Output

The 16, 8, 4, and single bit add and subtract lines are also used in the shift code update logic illustrated in figure 4-49. The shift code update logic adds the current shift operand ($X_0X_1X_2X_3X_4X_5$) to the adder input code ($A_1A_2A_3A_4A_5$), developed from the 16, 8, 4, and single bit add and subtract lines, to generate an updated shift operand ($\overline{PACSHFOB(0-5)}$) for the AUMIR on IRCARD(3). Figure 4-49 shows the adder input code equations implemented as part of the shift code update logic. The $\overline{X}_0$ bit is used as a carry into the development of $\overline{PACSHFOB(5)}$ so that two's complement addition (subtraction) is performed when a subtract line ($\overline{PACSHM16}$, $\overline{8}$, $\overline{4}$, or $\overline{1}$) from the shift decode logic is true. The status control logic uses the shift operand in true and complement form, the cyclic shift control line, and the last cycle shift control line to develop the update enable signal (PACSHFUD). The update enable signal is true as long as the current shift operand cannot be reduced to zero during the current execution cycle. If the current shift operand is 1, 4, or 8 for any shift instruction or 16 for a cyclic shift instruction, the update enable signal goes to zero. The update enable signal is used on the PCCTL1 and PCCTL2 cards and its inverse, the shift complete signal ($\overline{PACUD}$), is used on the PCCTL card. The PACSHFUD signal is generated with the following equation:

$$\begin{aligned} PACSHFUD = {} & (\overline{X}_0+\overline{X}_1+\overline{X}_2+X_4+X_5)(X_0+X_1+X_3+X_4+X_5) \\ & (X_0+X_1+X_2+X_3+X_4)(X_0+X_1+X_2+X_4+X_5) \\ & (\overline{CS}+\overline{X}_1+X_2+X_3+X_4+X_5)(\overline{X}_0+\overline{X}_1+\overline{X}_2+\overline{X}_3+\overline{X}_4+\overline{X}_5) \\ & (PAA1XCSH) \end{aligned}$$

where,

CS = PACSHFCS and is true for cyclic shifts

PAA1XCSH is true for all shifts

The arithmetic shift, logical shift, cyclic shift, and all shift control lines used by the shifter are generated by decoding the 8-bit op-code from the AUMIR. The all shift control line (PAAIXCSH) is implemented with an equation that goes to one for op-codes of 60_{16}, 64_{16}, or $6C_{16}$. The arithmetic shift control line (PASHFCAS) is implemented with an equation that goes to one for an op-code of 60_{16}; the logical shift control (PASHFCLS) line is implemented with an equation that goes to one for an op-code of 64_{16}; the cyclic shift control line (PASHFCCS) is implemented with an equation that goes to one for an op-code of $6C_{16}$. These control lines, except for PAAIXCSH, are distributed to the four PPAUCD cards to direct the type of shift. The PAAIXCSH control line is used by the fifth level of the AU adder to enable the output of the shifter through the AU1 transfer bus.

4-122 Op-code Decoding and Support Control. The op-code decoding logic on the CONTAU card develops the necessary op-code enables for AU operation and the support control supplies the current R field and VP code. The

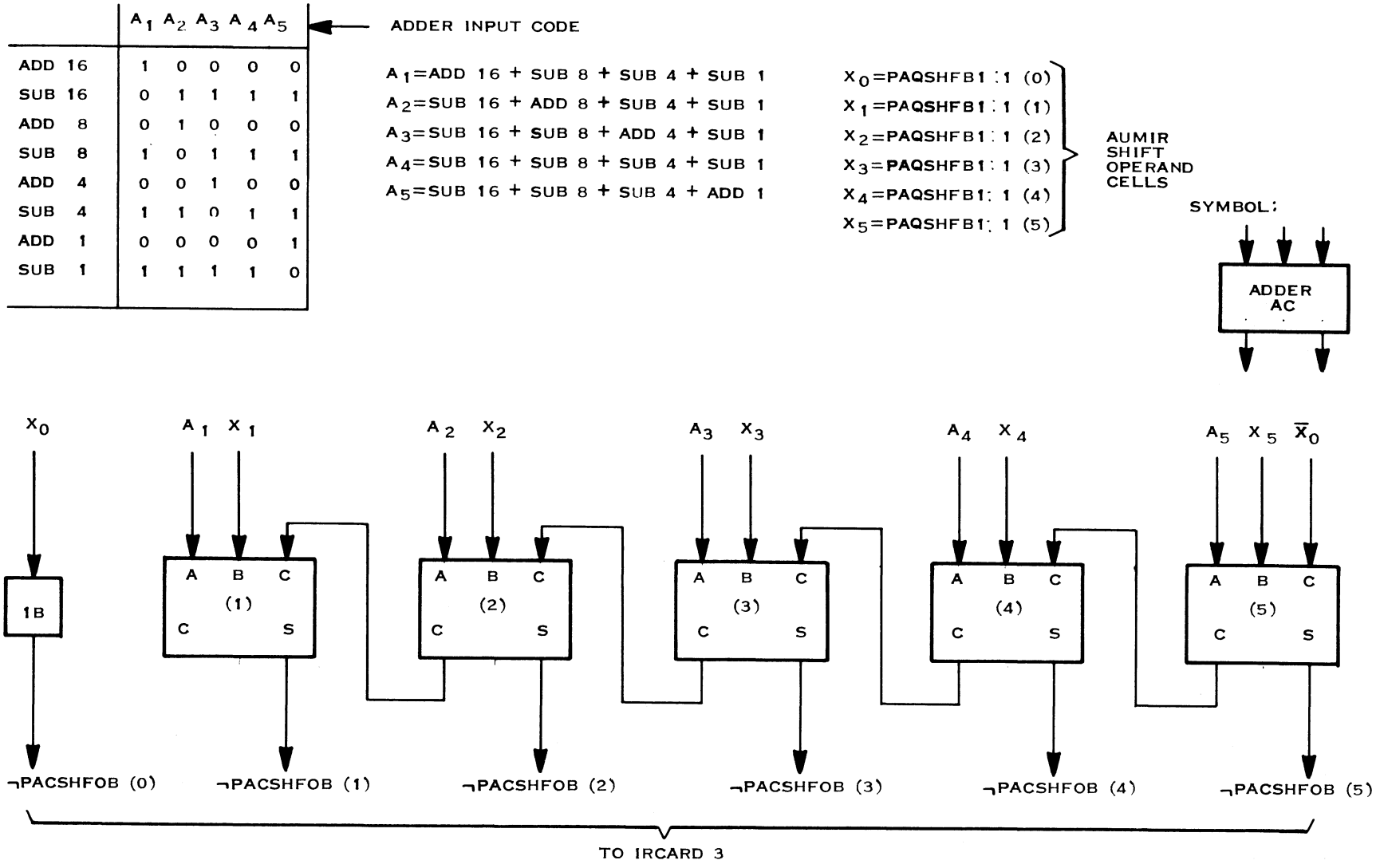


Figure 4-49. Shift Code Update Logic

R field and VP code from the AUMIR are temporarily stored in flip-flops on the CONTAU card for the duration of the execution cycle. These two quantities are distributed to the four PPAUCD cards where they are used by the unload box and test box 2 logic. The op-code decoding logic utilizes the 8-bit op-code and the 2-bit state code from the AUMIR to develop the following AU control lines (this list does not include those control lines already mentioned for the shifter):

- PAADDCAN - Used by the first level of the adder to enable the logical AND function.
- PAADDCBY - Used by the third level of the adder to develop the generalized carry into byte equation.
- PAADDCEQ - Used by the first level of the adder to enable the logical equivalence function.
- PAADDCEX - Used by the first level of the adder to enable the logical exclusive OR function.
- PAADDCHW - Used by the first level of the adder to develop the carry transfer and carry develop equations for the MSB of the right halfword.
- PAADDCOR - Used by the first level of the adder to enable the logical OR function.
- PAA1XCBP - Used by the fifth level of the adder to enable the output of the bit picker through to the AU1 transfer bus (POLL instruction).
- PAA1XCLO - Used by the fifth level of the adder to enable the logical output of the adder through to the AU1 transfer bus (logical instructions).
- PAA1CSM - Used by the fifth level of the adder to enable the sum output of the adder through to the AU1 transfer bus (add, subtract, increment, and decrement instructions).
- PABCOFX - Used in the shifter to generate zero fill in byte zero for right logical shifts and to generate sign propagation in byte zero for right arithmetic shifts.
- PABC3FX - Used in the shifter to generate zero fill in byte three for left arithmetic and left logical shifts.
- PACCHE - Used to enable the skip taken logic for CEH, CER, and CEL instructions.

- PACCIEB - Used to enable the skip taken logic for CEB and CEBI instructions.
- PACCIEH - Used to enable the skip taken logic for CEHI instructions.
- PACCIEW - Used to enable the skip taken logic for CEI instructions.
- PACCINB - Used to enable the skip taken logic for CNB and CNBI instructions.
- PACCINH - Used to enable the skip taken logic for CNHI instructions.
- PACCINW - Used to enable the skip taken logic for CNI instructions.
- PACCOCAD - Used in the complement or constant generator to enable true data to the adder (add instructions).
- PACCOCDE - Used in the complement or constant generator to enable minus one to the adder (decrement instructions).
- PACCOCIN - Used in the complement or constant generator to enable plus one to the adder (increment instructions).
- PACCOCSU - Used in the complement or constant generator to enable complement data to the adder (subtract instructions).
- PACCRHN - Used to enable the skip taken logic for CNH, CNL, and CNR instructions.
- PACCWE - Used to enable the skip taken logic for CE instructions.
- PACCWN - Used to enable the skip taken logic for CN instructions.
- PACMDF - Used to enable the skip taken for stack logic for MOD instructions.
- PACPSH - Used to enable the skip taken for stack logic for PUSH instructions.
- PACPUL - Used to enable the skip taken for stack logic for PULL instructions.
- PACTMB - Used to enable the branch taken logic for TMB instructions.

- PACTMH - Used to enable the branch taken logic for TMH instructions.
- PACTMW - Used to enable the branch taken logic for TM instructions.
- PACTNB - Used to enable the branch taken logic for TNB instructions.
- PACTNH - Used to enable the branch taken logic for TNH instructions.
- PACTNW - Used to enable the branch taken logic for TN instructions.
- PACTPB - Used to enable the branch taken logic for TPB instructions.
- PACTPH - Used to enable the branch taken logic for TPH instructions.
- PACTPOL - Used to enable the skip taken logic for POLL instructions.
- PACTPW - Used to enable the branch taken logic for TP instructions.
- PACTSAOL - Used to enable the skip taken logic for TAOL instructions.
- PACTSAOR - Used to enable the skip taken logic for TAOR instructions.
- PACTSAZL - Used to enable the skip taken logic for TAZL instructions.
- PACTSAZR - Used to enable the skip taken logic for TAZR instructions.
- PACTSOL - Used to enable the skip taken logic for TSOL, TOL, and TROL instructions.
- PACTSOR - Used to enable the skip taken logic for TSOR, TOR, and TROR instructions.
- PACTSZL - Used to enable the skip taken logic for TSZL, TZL, and TRZL instructions.
- PACTSZR - Used to enable the skip taken logic for TSZR, TZR, and TRZR instructions.
- PACTZB - Used to enable the branch taken logic for TZB instructions.

- PACTZH - Used to enable the branch taken logic for TZH instructions.
- PACTZW - Used to enable the branch taken logic for TZ instructions.
- PACVPTO - Used to enable the skip taken logic for VPTO instructions.
- PACVPTZ - Used to enable the skip taken logic for VPTZ instructions.
- PACXDN - Used to enable the branch taken logic for DBN instructions.
- PACXDZ - Used to enable the branch taken logic for DBZ instructions.
- PACXIN - Used to enable the branch taken logic for IBN instructions.
- PACXIZ - Used to enable the branch taken logic for IBZ instructions.
- PADMPCOL - Used in the data manipulator to enable the setting of bits for TSZL, TSOL, and SL instructions.
- PADMPCOR - Used in the data manipulator to enable the setting of bits for TSZR, TSOR, and SR instructions.
- PADMPCPR - Used in the data manipulator to enable the resetting of a bit for VPR instructions.
- PADMPCPS - Used in the data manipulator to enable the setting of a bit for VPS instructions.
- PADMPCZL - Used in the data manipulator to enable the resetting of bits for TRZL, TROL, and RL instructions.
- PADMPCZR - Used in the data manipulator to enable the resetting of bits for TRZR, TROR, and RR instructions.

4-123 INDEXER

The indexer section of the PP is composed of a PC indexer (I1), a TN field indexer (I2), and a register indexer (I3). A detailed description with accompanying block diagrams on each of the three indexer parts is presented in the following paragraphs.

4-124 PC INDEXER. Refer to figure 4-50 for a simplified block diagram of the PC indexer (I1). The PC indexer is used to increment or decrement the program counters (PC's) of the eight VP's or the effective address developed in the main instruction register (MIR). The VP PC to be operated on by the PC indexer is selected by the VP code (the VP code is the VP number (0 through 7) of the VP whose instruction is currently at the execution level) from the maintenance logic and the indexing action to be taken is controlled by the op-code from the MIR. The source of input to the PC indexer (CMAB bus or RMAB bus) is primarily determined by the mode bit from the MIR, except for the cases mentioned in the following paragraph.

When the current instruction is from central memory (CM) and is not a BPCS, BCS, BRS, or BCAS instruction in state class 2 step 2 or 3, the CMAB bus is enabled for PC indexing. When a BPCS, BCS, BRS, or BCAS instruction from CM is executing and is in state class 2 step 2 or 3, the RMAB bus is enabled for indexing because the CMAB bus is being used to handle other data flow. When the current instruction is from read only memory (ROM) and is not a BPCS, BCS, BRS, or BCAS instruction in state class 2 step 2 or 3, the BRSM instruction in state class 2 step 2, or the INTF cell is not set in state class 2 step 3, the RMAB bus is enabled for PC indexing.

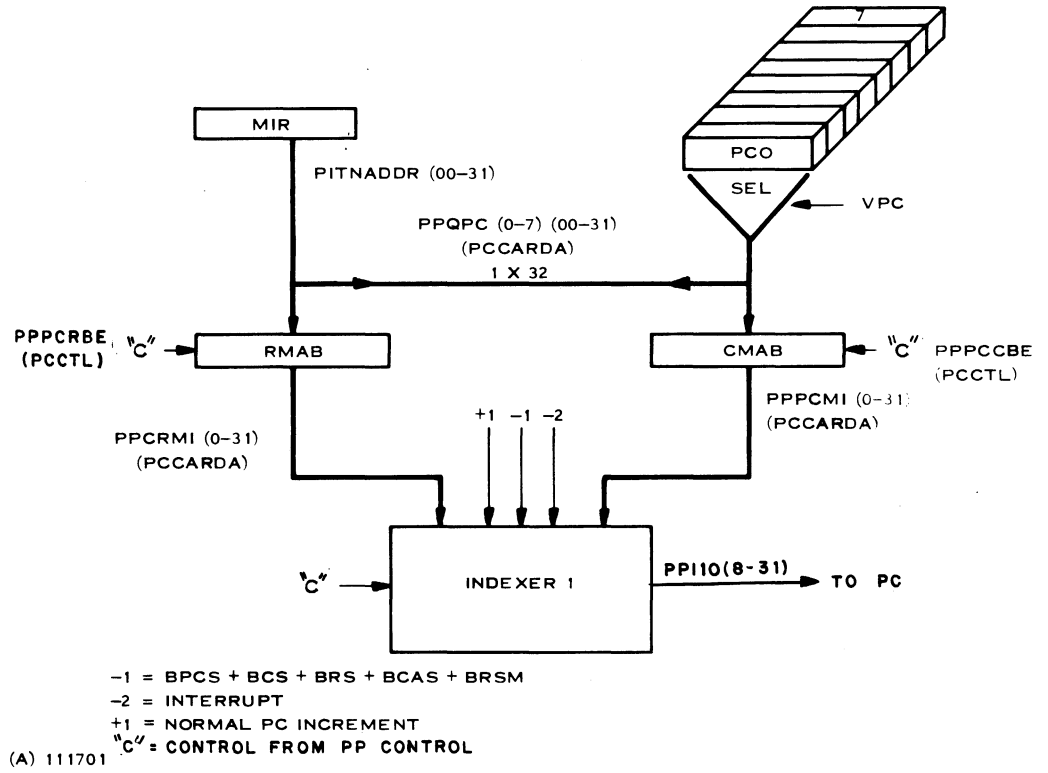


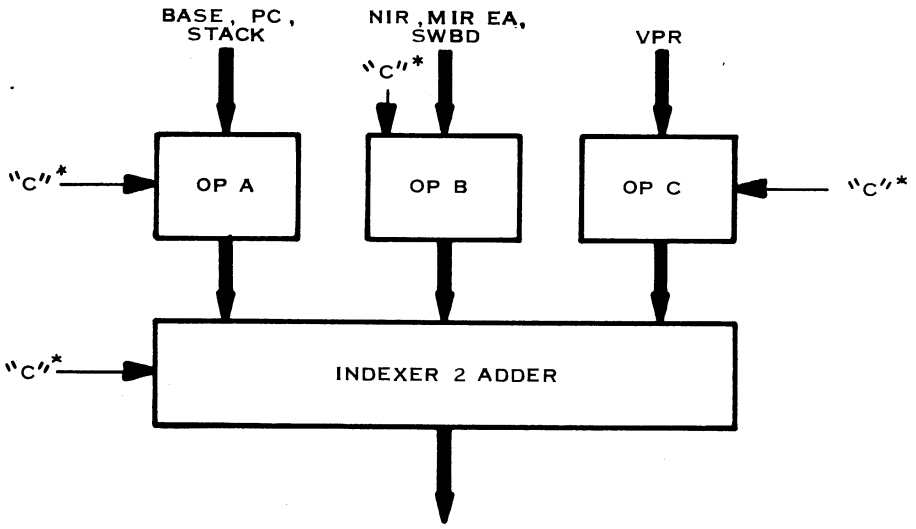
Figure 4-50. Program Counter Indexer

When a BPCS, BCS, BRS, or BCAS instruction from ROM is executing in state class 2, step 2 or 3, the BRSM instruction from ROM is executing in state class 2, step 2, or the INTF cell of the MIR is set in state class 2, step 3, the CMAB bus is enabled for indexing because the RMAB bus is being used to handle other data flow. The effective address in the MIR is used as the quantity to be indexed (rather than the PC) when a branch is taken in either CM or ROM. When the source of the branch instruction is CM, the CMAB bus is enabled for the indexing and when the source of the branch instruction is ROM, the RMAB bus is enabled for the indexing.

The indexing quantities (+1, -1, or -2) are hardwired in the PC indexer and are used to modify the input from the CMAB or RMAB bus, as shown in figure 4-50. In the normal case, the input to the PC indexer is incremented by one in order to advance the third level of the three level pipe in the PP. When a BPCS, BCS, BRS, BCAS, or BRSM instruction is encountered, the input to the PC indexer is decremented by one so that the PC value saved points to the instruction following the branch and save instruction executed. When an interrupt occurs, execution of the current instruction is completed and the resulting PC value is decremented by two so that the instruction following the interrupted instruction is not lost after the interrupt is honored. The 24-bit output of the PC indexer is inserted back in the PC of the current VP for use as a pointer to the next instruction to be retrieved from CM or ROM (as determined by bit 0 of the PC).

The PC indexer controls the setting and resetting of PC bit 0 during the same time frame as the indexing of the 24-bit PC value (bit 0 is set to one for CM or reset to zero for ROM). When the current instruction is not a conditional branch, unconditional branch, or branch and save, bit 0 of the PC will copy itself during the indexing operation. When a conditional or unconditional branch instruction is encountered, bit 0 of the PC is set or reset to reflect the mode bit of the MIR, if the branch is taken. If a conditional branch is not taken, bit 0 of the PC will copy itself. When a branch and save instruction is encountered, bit 0 of the PC is set or reset to reflect the mode bit of the MIR.

4-125 TN FIELD INDEXER. Refer to figure 4-51 for a simplified block diagram of the TN field indexer (I2). The TN field indexer is used to develop the effective address of each instruction from the eight VP's for use in the appropriate VP instruction register (IR). The OP A, OP B, and OP C blocks in figure 4-51 are selectors used to gate the proper quantities into the indexer adder. The OP A block selects a base quantity from the active VP Central Memory (CM) base register for base relative instructions, a PC quantity from the active VP PC for PC relative instructions, or the proper combination of plus and minus one for updating the stack parameters when a stack instruction is being executed. The OP B block selects the N field of the SWBD or NIR, depending on the instruction source or the effective address



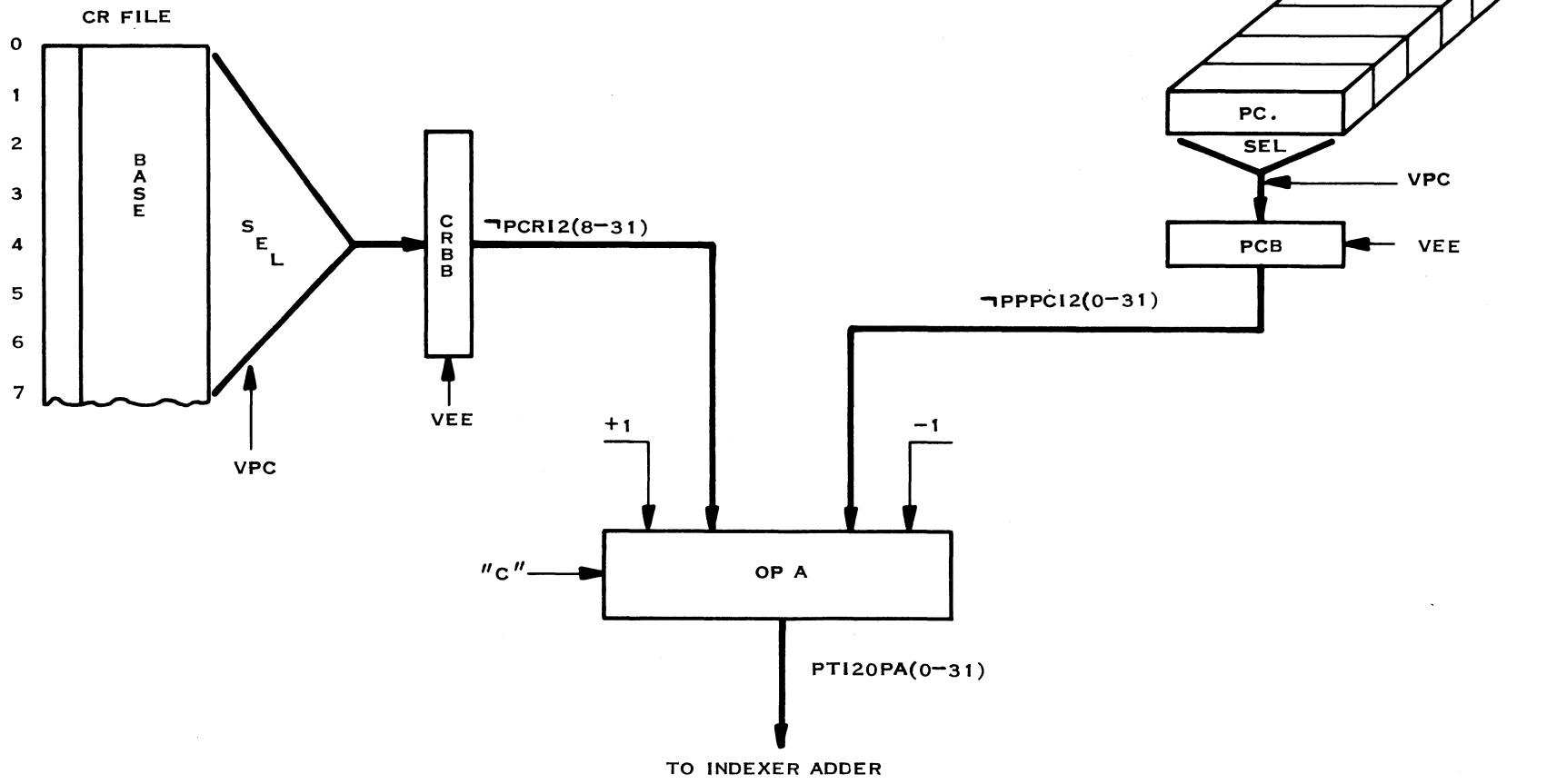
*"C" MEANS CONTROL

(A)124763

Figure 4-51. TN Field Indexer

of the MIR when the WCE test is to be performed. The OP C block selects the proper VP register (VPR) halfword quantity, when indexing is required, in order to develop an effective address. The outputs of OP A, OP B, and OP C are input to the 24-bit indexer adder. During normal operation, the 24-bit sum of the inputs to the indexer adder is output to the IR as the source, destination, or effective address of a new instruction to be executed. When the stack parameters are to be modified or a WCE test is to be performed, the indexer adder performs in a different manner. When the word count and space count stack parameters are to be modified, the carry bit between the most-significant halfword and the least-significant halfword of the indexer adder result is inhibited because the two halfwords contain different stack parameters. When the stack pointer parameter is to be incremented, the WCE test is to be performed, or the word count and space count parameters are to be modified by a modify stack instruction, the carry input to the least-significant halfword of the indexer adder is set so that the input to the indexer adder is incremented by one. A more detailed description of the OP A, OP B, and OP C portions of the TN field indexer is presented in the following paragraphs.

4-126 OP A Selector. Refer to figure 4-52 for a simplified block diagram of the OP A selector. The VP code from the MIR enables the CM base register of the active VP to the CRBB bus and the PC of the active VP to the



I. BASE SELECTION ENABLED FOR ALPHA OPERAND DEVELOPMENT AND BASE RELATIVE BRANCHES.
II. PC SELECTION ENABLED FOR PC RELATIVE BRANCHES.
III. (+1) AND (-1) ENABLED DURING STACK INSTRUCTIONS.
IV. VEE ENABLES THE ASSOCIATED BUS

(A) 111649

Figure 4-52. OP A Selector

PCB bus. Data exists on both of these buses whenever the PP is executing instructions; therefore, no control gating is required at this level. The control necessary to select a base register quantity, a PC quantity, or the proper combination of plus and minus one for stack instructions, is applied to the OP A selector block of figure 4-52. When a base relative instruction is to be indexed for use in the IR, the op-code of the base relative instruction is used to gate the 24-bit CM base quantity through OP A to the indexer adder. When a PC relative instruction is to be indexed for use in the IR, the op-code of the PC relative instruction is used to gate the 24-bit PC quantity through OP A to the indexer adder. When the stack parameters retrieved from CM by a stack instruction are to be modified, the op-code and state of the stack instruction are used to control the placement of plus and minus one in the output of OP A. The result is input to the indexer adder.

4-127 OB B Selector. Refer to figure 4-53 for a simplified block diagram of the OP B selector. The VP code from the maintenance logic enables the NIR of the active VP to the NIRB bus and the SWBD of the active VP to the MDIB bus. Data exists on both of these buses whenever the PP is executing instructions; therefore, no control gating is used at this level. The control necessary to select a NIR quantity, a SWBD quantity, or the effective address field of the MIR, is applied to the OP B selector block of figure 4-53. When the next instruction location flag (NIL) of the MIR indicates the next instruction is in the NIR and the op-code of the next instruction indicates neither a PC relative branch or an immediate operand, the N field of the NIR is gated through OP B to the indexer adder. When the NIL indicates that the next instruction is in the SWBD and the op-code of the next instruction indicates neither a PC relative branch, nor an immediate operand, the N field of the SWBD is gated through OP B to the indexer adder. When the source of the next instruction is either the NIR or the SWBD and the op-code of the next instruction indicates a PC relative branch or an immediate operand, the sign bit of the source register is extended 8 bits and gated through OP B along with the 16-bit N field of the source register. When a stack instruction is executing in a state class and step such that a stack parameter is to be modified, or when an analyze or indirect instruction is passing an indirect cell from the SWBD to the IR, the entire 32 bits of the SWBD are gated through OP B to the indexer adder. When the effective address of the current instruction requires incrementing to perform a WCE test, to locate the stack pointer parameter during a stack instruction, to locate an instruction following a branch location during an unconditional branch and save instruction, or to locate the next sequential address during a store VPR file instruction, the 24-bit effective address from the MIR is gated through OP B to the indexer adder. In all of the mentioned cases, only bits 8 through 31 of the OP B output are used by the indexer adder.

4-128 OP C Selector. Refer to figure 4-54 for a simplified block diagram of the OP C selector. The VP code from the maintenance logic enables the

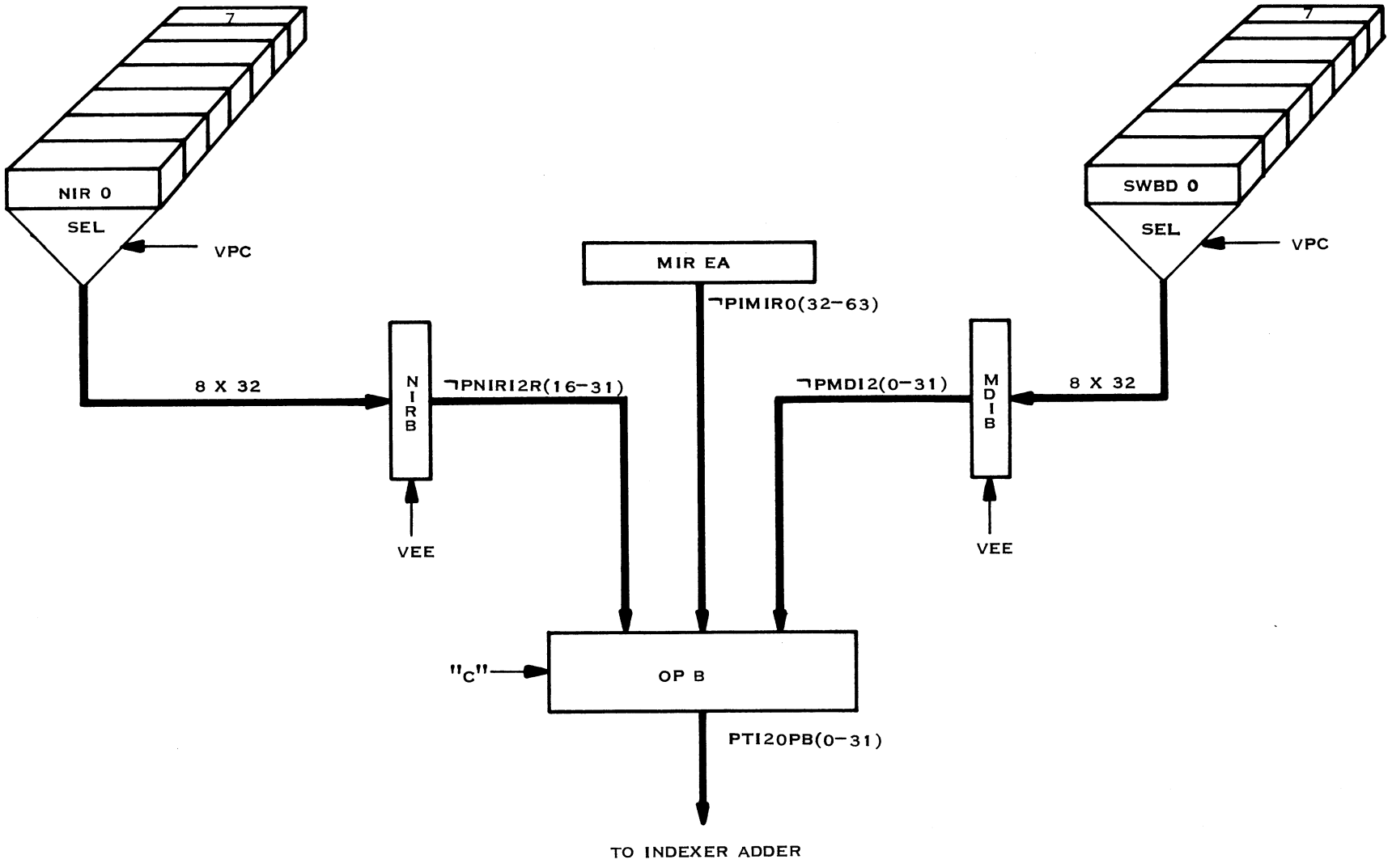
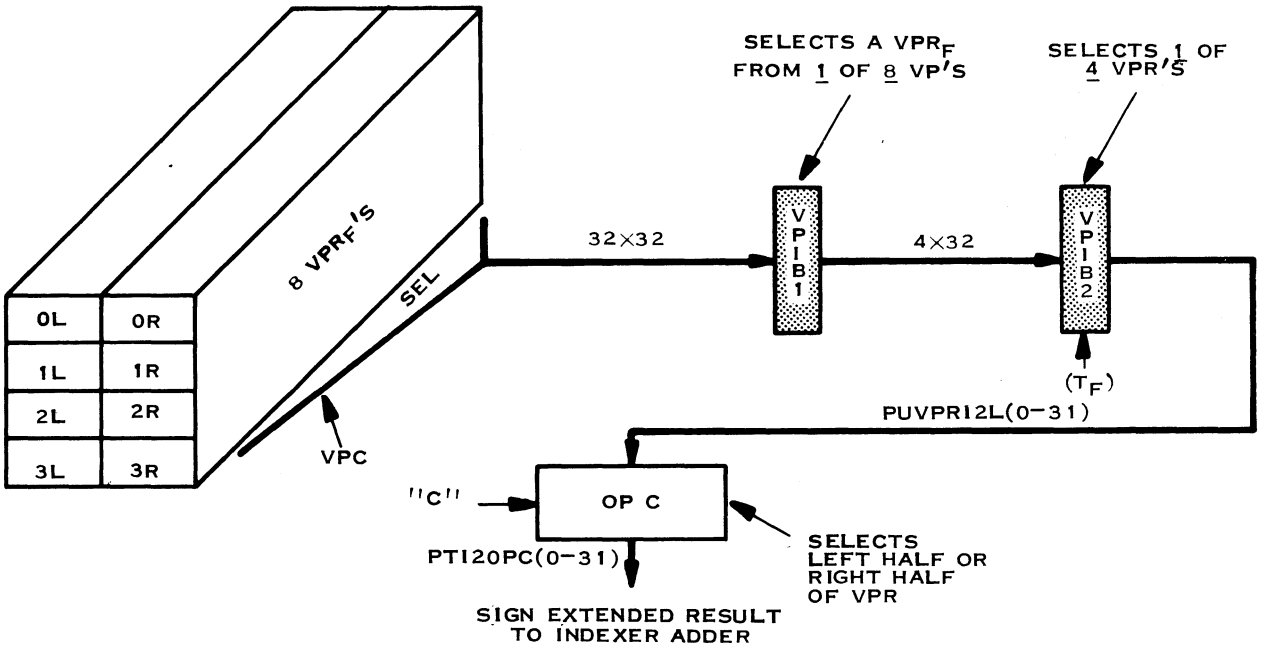


Figure 4-53. OP B Selector



I. SELECTION CONTROL LOGIC SELECTS WHICH VPR HALFWORD TO USE

II. OP C IS ENABLED INTO ADDER IF $\overline{T=0} + \overline{T=8}$

(A) 111696

Figure 4-54. OP C Selector

VPR file of the active VP to the VPIB1 bus, and the middle two bits from the T field of the instruction being indexed are used to enable one of the four VPR's of the VPR file to the VPIB2 bus. The selected 32-bit VPR quantity is input to the OP C selector where the necessary control is provided to select either the right or left halfword of the VPR, and to enable the selected halfword to the indexer adder or to set the input to the indexer adder to zero. When the T field of the instruction to be indexed is 0 or 8, a push or pull stack instruction is being executed, or the MIR effective address is being incremented for a WCE test, the output to the indexer adder from OP C is forced to zero (no indexing). When the output to the indexer adder from OP C is not forced to zero, the least-significant bit (LSB) from the T field of the instruction being indexed is used to select the right (T field LSB of one) or left (T field LSB of zero) halfword of the VPR on the VPIB2 bus. The selected halfword is sign extended 8 bits and the result is output to the indexer adder. An exception to the described OP C selector operation occurs when the word count and space count parameters are to be modified by the modify stack instruction. In this case, the two most significant bits from the R field (rather than the two middle bits from the T field) of the instruction being indexed are used to enable one of the four VPR's of the current VPR file to the VPIB2 bus. The right or left halfword of the resulting 32-bit VPR quantity is selected by bit 2 of the same R field. The selected halfword and its one's complement are inserted in bits 0 through 15 and bits 16 through 31, respectively, of the OP C output before being gated to the indexer adder.

4-129 REGISTER INDEXER. Refer to figure 4-55 for a simplified block diagram of the register indexer (I3). The register indexer is used to develop source and destination addresses for instructions from the eight VP's for use in the appropriate VP IR. The addresses developed specify a VPR or a Communications Register (CR). The VP code from the maintenance logic enables VPR3 byte 3 of the active VPR file to the ΔVPB bus, and the NIL flag from the MIR is used to select the R field from the SWBD or NIR. These two quantities are input to the register indexer, where the op-code of the instruction being indexed is used for control. When the op-code of the instruction being indexed specifies that the R field is addressing a VPR, the R field input to the register indexer is gated through to the IR as the developed address. When the op-code specifies that the R field is addressing a CR, the address is developed by summing the input R field and byte 3 of VPR3. The result is input to the IR as the developed source or destination address. The 8-bit developed address from the register indexer is used to specify one of four VPR's down to the byte level or one of 64 CR's down to the byte level. The amount of data addressed (word, halfword, or byte) is determined by the op-code of the instruction being indexed.

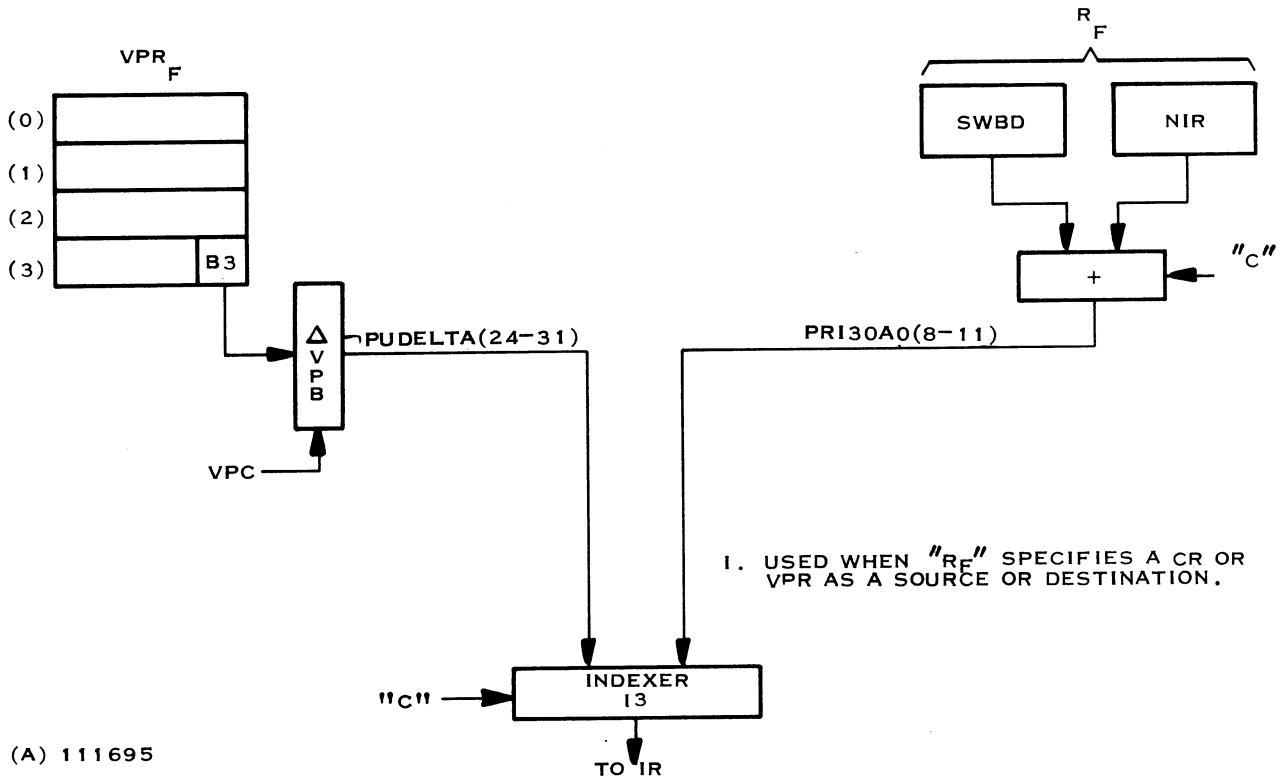


Figure 4-55. Register Indexer

4-130 COMMUNICATIONS REGISTER FILE

The Communications Register (CR) file of the PP is functionally divided into CR file control, input synchronizers, and CR registers. A detailed description, supplemented with block diagrams and/or logic diagrams, is presented for each of these three areas in the following paragraphs.

4-131 CR FILE CONTROL. Control of the CR file is physically divided into CRMIRLDR card control, CRCONT (0 through 3) card control, and CRCELLY card control. The CRMIRLDR card receives IR data via the VPRCONT card, expands the IR data into the CRMIR input format, and distributes the expanded control information to the four CRCONT cards. Each CRCONT card uses the control information from the CRMIRLDR card to generate the selects and enables needed for reading from and writing to one of 64 bytes of data. The CRCELLY card uses control from CRCONTO and interrupt information from various areas of the PP to control the setting and resetting of data in the interrupt associated bytes of the CR file (byte 0 of CR's 0 through 7). A detailed description of the three physical areas of CR file control is presented in the following paragraphs.

4-132 CRMIRLDR Control. Refer to figure 4-56 for a simplified logic diagram of the CRMIRLDR control logic. The CRMIRLDR card accepts source, destination, op-code group, and maintenance select data from the VPRCONT card, and maintenance data from the AU2B bus, and generates CR read and write signals capable of addressing CR data down to the hex level (this is necessary for writing to the CR file; reading is done at the word level). Refer to figure 4-57 for the format of the control data distributed to the four CRCONT cards.

When data is to be read from a CR, the source (¬PES(0-5)) and destination (¬PED(0-5)) bits from the VPRCONT card are applied to select logic composed of a group of SQ logic modules. If the executing instruction is an indirect store of a word via a CR from a VPR or another CR (op-codes of 98 or 9C), or tests and/or sets/resets bits in a CR byte or half-byte (hex) and skips the next instruction if a test made is true (op-codes of 82, 86, 8A, 8E, 92, 96, 9A, 9E, C2, C6, CA, CE, D2, D6, DA, DE, E2, E6, EA, EE, F2, F6, FA, and FE), the destination bits from the VPRCONT card are selected and output to the CRCONT cards as the CR read card and word select lines (¬PCLRW(0-5)). In these cases, the destination is used as the source of the read during the indirect cycle (for the indirect stores) or because the source contains the R field mask or is identical to the destination (for the test and/or set/reset type instructions mentioned). If the executing instruction is different from those mentioned and involves reading from a CR (stores, loads, test CR and branches, and test poll bits), the source bits from the VPRCONT card are selected and output to the CRCONT cards as the CR read card and word select lines. When a maintenance function is being performed,

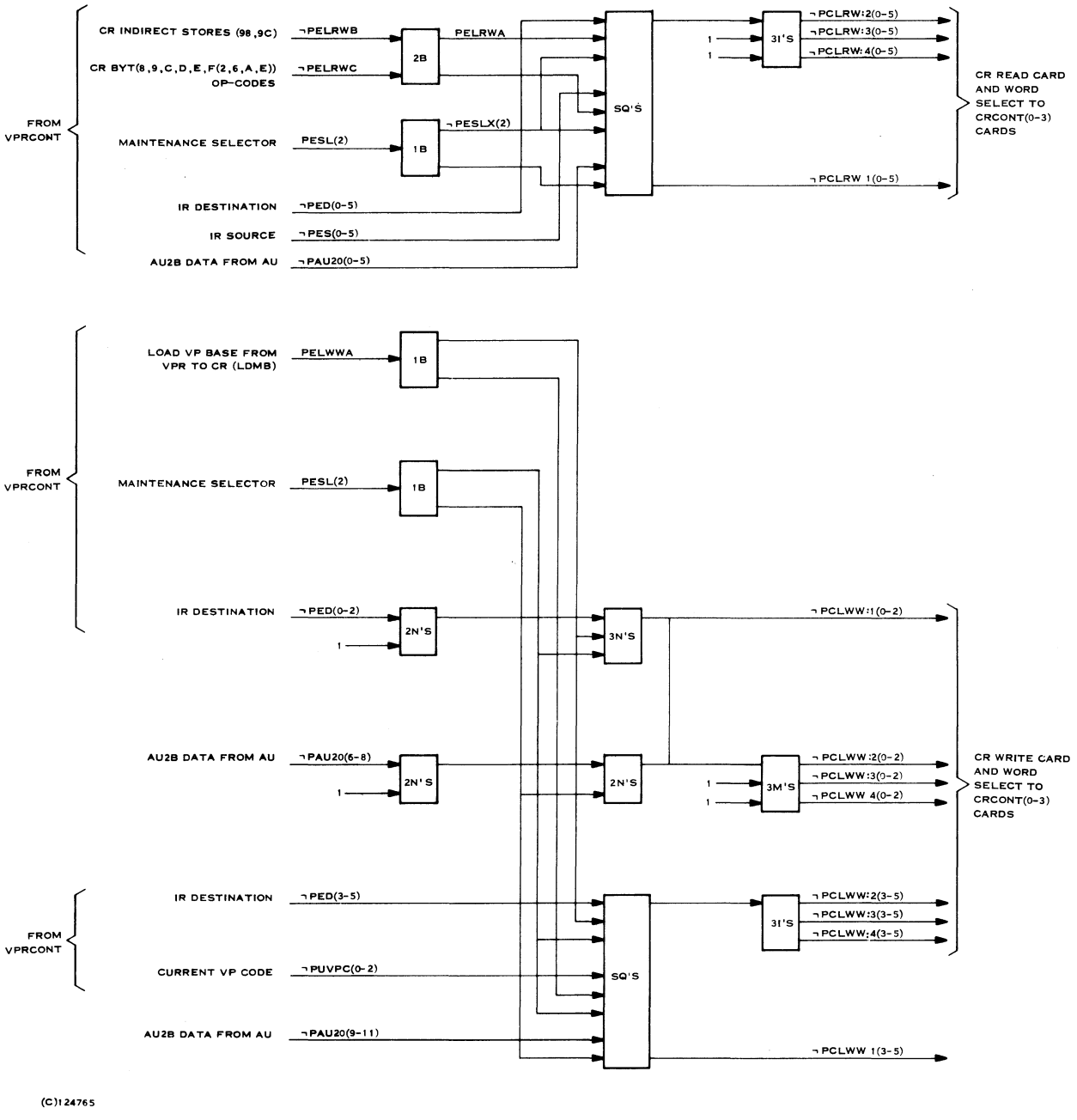


Figure 4-56. CRMIRLDR Control Logic (Sheet 1 of 2)

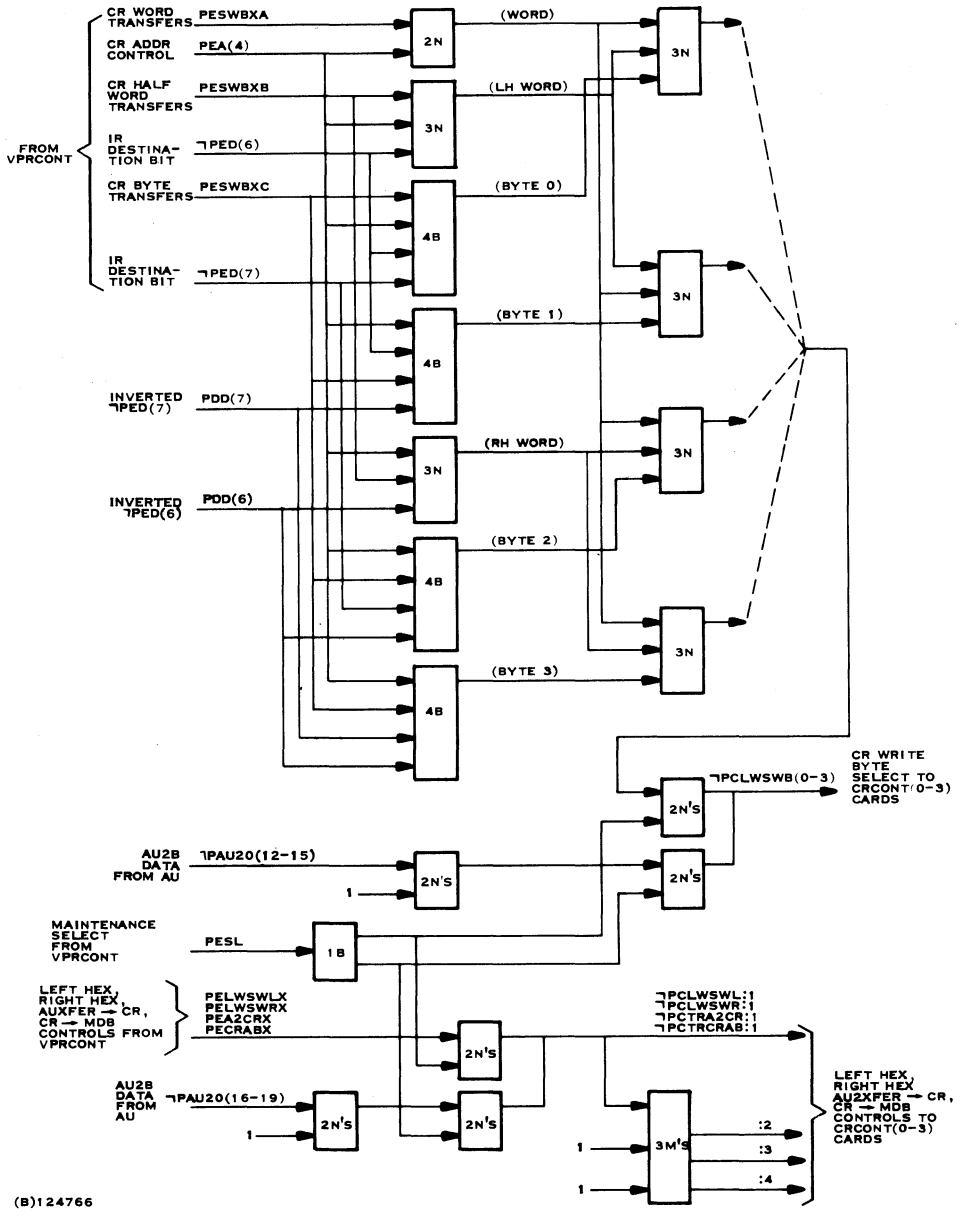


Figure 4-56. CRMIRLDR Control Logic (Sheet 2 of 2)

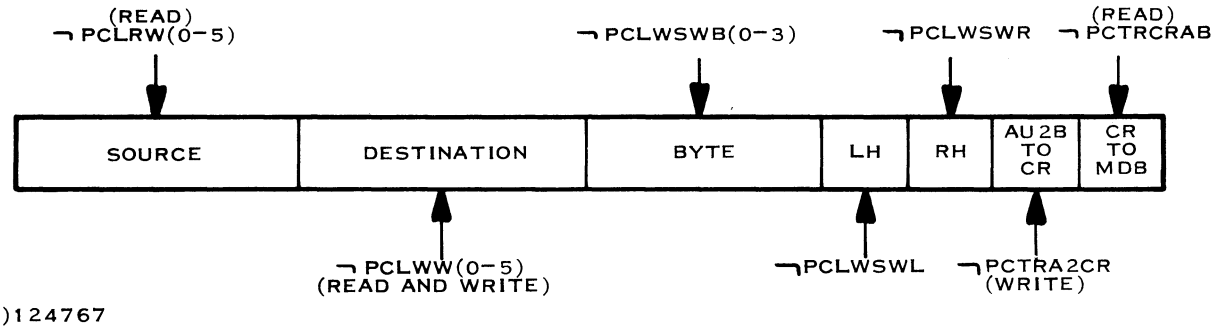


Figure 4-57. CRMIR Input Format From CRMIRLDR

the maintenance selector (PESL(2)) is used to disable selection of the source or destination bits from VPRCONT and to enable selection of the AU2B bus data (¬PAU20(0-5)). The AU2B bus data is output to the CRCONT cards as the CR read card and word select.

When data is to be written to a CR, the destination bits from the VPRCONT card are split into two groups (bits 0 through 2 and bits 3 through 5) and applied to select logic composed of 3N and SQ logic modules. If the executing instruction is load VP base in CR from VPR (LDMB), the PELWWA signal disables the destination bits and enables the active VP code (¬PUVPC(0-2)) into bits 3 through 5 of the output. The net result, the CR write card and word select, points to the CM base register corresponding to the active VP code and is output to the CRCONT cards. If the executing instruction is other than LDMB and involves writing into a CR, bits 0 through 2 of the destination are selected and output to the CRCONT cards as the CR write card select (¬PCLWW(0-2)) and bits 3 through 5 of the destination are selected and output to the CRCONT cards as the CR write word select (¬PCLWW(0-2)). When a maintenance function is being performed, the maintenance selector disables selection of the destination bits and enables selection of AU2B bus data (¬PAU20(6-11)). The AU2B bus data is output to the CRCONT cards as the CR write card and word select.

When data is to be written to a CR, the CR address control (PEA(4)) is used to enable the logic required in generating the byte select lines (¬PCLWSWB(0-3)). If a fullword CR write is to be executed, the PESWBXA signal from VPRCONT is inverted and used to drive the outputs of the four second level 3N logic modules to one (refer to sheet 2 of figure 4-56). The 3N outputs are inverted and distributed to the associated CRCONT cards (¬PCLWSWB(0) goes to CRCONT0, ¬PCLWSWB(1) goes to CRCONT1, ¬PCLWSWB(2) goes to CRCONT2, and ¬PCLWSWB(3) goes to CRCONT3). If a halfword CR write is to be executed, the PESWBXB signal from VPRCONT combines with bit 6 of the destination (bit 6 indicates left halfword when zero and right halfword when one) to develop a left halfword or right halfword signal. A left halfword write drives the outputs of the first two second-level 3N

logic modules to one and a right halfword write drives the outputs of the last two second-level 3N logic modules to one. In both of these cases, the 3N outputs are inverted and distributed to the associated CRCONT cards to indicate that two bytes of a CR are to be written to. If a byte CR write is to be executed, the PESWBXC signal from VPRCONT combines with bits 6 and 7 of the destination to develop the byte 0, byte 1, byte 2, and byte 3 signals. When bits 6 and 7 are both zero, the byte 0 signal drives the first second level 3N logic module to one; when only bit 7 is one, the byte 1 signal drives the second second-level 3N logic module to one; when only bit 6 is one, the byte 2 signal drives the third second-level 3N logic module to one; when both bits 6 and 7 are one, the byte 3 signal drives the fourth second-level 3N logic module to one. In all four of these cases, the 3N outputs are inverted and distributed to the associated CRCONT cards to indicate that only one byte of a CR is to be written to. When a maintenance function is being performed, the maintenance selector disables the outputs of the second level 3N logic modules and enables the AU2B bus data (¬PAU20(12-15)) through to the CRCONT cards on the byte select lines.

When data is to be written to the left hex of a CR byte, the PELWSWLX signal is developed on the VPRCONT card and applied to select logic composed of 2N logic modules on the CRMIRLDR card. During normal processing, the select logic inverts the PELWSWLX signal and distributes the result (¬PCLWSWL) to the four CRCONT cards. When data is to be written to the right hex of a CR byte, the ¬PCLWSWR signal is developed in a similar manner. Anytime data is to be written to a CR, the VPRCONT card develops the PEA2CRX signal, the 2N select logic on CRMIRLDR inverts it, and the result (¬PCTRA2CR) is distributed to the four CRCONT cards to enable the write operation. The ¬PCTRCRAB signal is developed in a similar manner anytime data is to be read from a CR. When a maintenance function is being performed, the maintenance selector disables the left hex, right hex, CR write, and CR read signals from VPRCONT and enables the AU2B bus data (¬PAU20(16-19)) to provide a substitute for these signals.

4-133 CRCONT Control. Refer to figure 4-58 for a simplified logic diagram of the CRCONT1 control logic (the synchronizers on the CRCONT card are discussed in a later paragraph) that applies to byte 1 of the CR file. The control logic on CRCONT0, CRCONT2, and CRCONT3 is identical to that on CRCONT1 and applies to CR bytes 0, 2, and 3, respectively. The detailed description of the CRCONT cards control logic is based on CRCONT1; however, the other three cards perform the same type of processing at the same time. Each CRCONT card accepts read and write card and word selectors, left and right hex indicators, read and write enables, and the associated byte bit from the CRMIRLDR card, to develop the read and write card enables, the read and write word selectors, and the write left hex and right hex enables for the associated CR byte. This same control information from all four CRCONT cards is combined to form the CRMIR output format shown in figure 4-59.

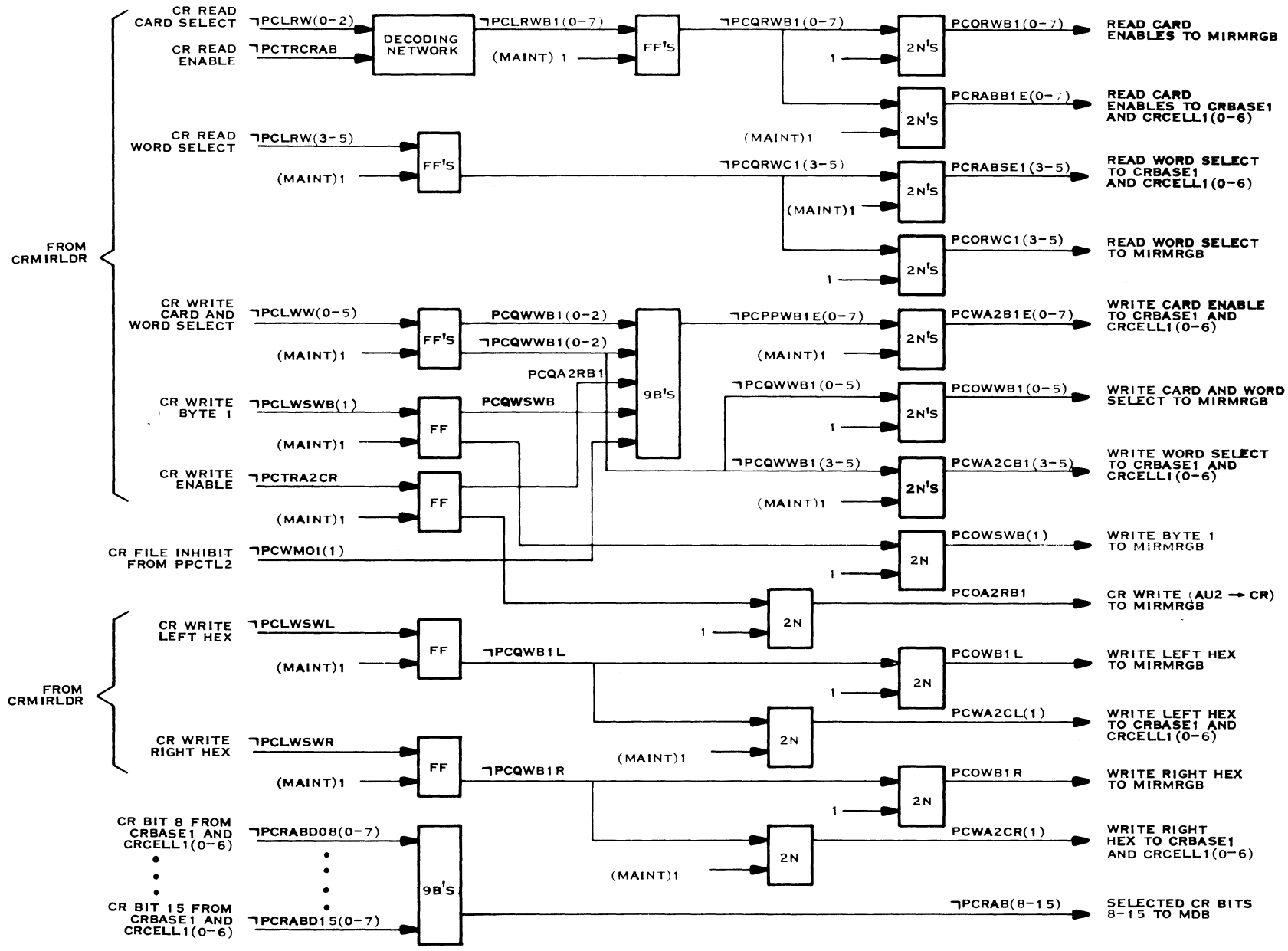
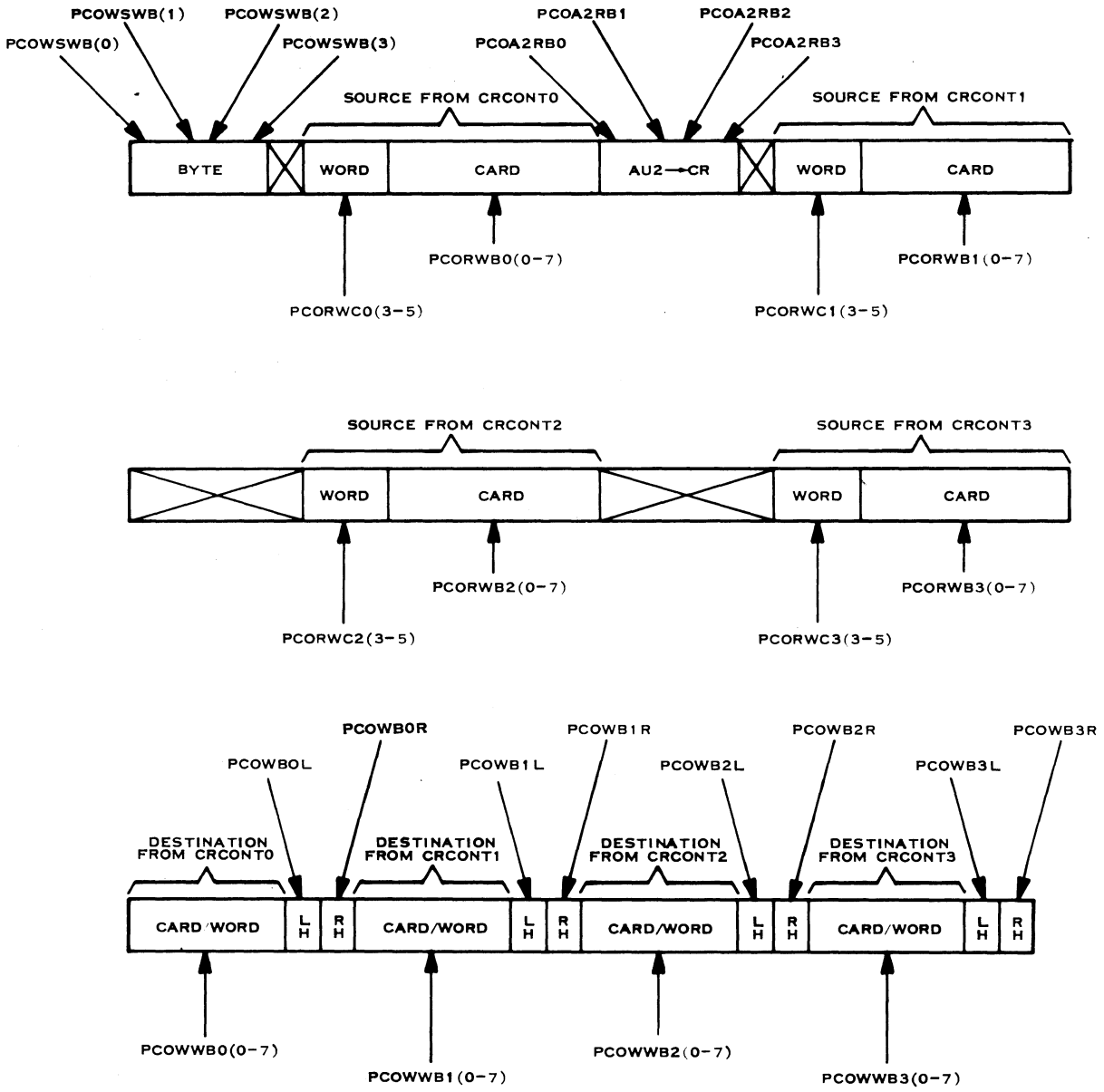


Figure 4-58. CRCONT1 Control Logic



(B)124769

Figure 4-59. CRMIR Output Format from CRCONT0 - CRCONT3

When data is to be read from a CR (all CR reads involve a fullword), the CR read enable signal (¬PCTRCRAB) from CRMIRLDR permits the decoding of the three read card select lines (¬PCLRW(0-2)) to eight read card enable lines. The eight read card enable lines are gated into a group of flip-flops (the gate remains set during normal PP processing) for the duration of the current execution cycle. The true flip-flop outputs are inverted by a group of 2N logic modules and the results (PCRABB1E(0-7)) are distributed to CRBASE1 and CRCELL1(0-6) (PCRABB1E(0) goes to CRBASE1, PCRABB1E(1) goes to CRCELL1(0), etc.). A different group of 2N logic modules invert the same flip-flop outputs in order to develop the CRMIR output format for the MIRMRGB card (PCORWB1(0-7)). The read word select lines from CRMIRLDR (¬PCLRW(3-5)) are gated into a group of flip-flops and the complement outputs are inverted by two groups of 2N logic modules in order to develop the read word select lines for CRBASE1 and CRCELL1(0-6) (PCRABSE1(3-5)) and for MIRMRGB (PCORWC1(3-5)). The other three CRCONT cards (CRCONT0, CRCONT2, and CRCONT3) perform the same processing at the same time, so the net result is the reading of one full CR from the CR file.

When data is to be written to a CR (it is possible to write single bits when the R field is used as a mask on a hex of data), the CR write enable signal (¬PCTRA2CR) from CRMIRLDR is gated into a flip-flop. The true output of the flip-flop and the CR file inhibit signal (¬PCWMOI(1)) from the PPCTL2 card combine to provide secondary enables for the group of 9B logic modules used in developing the write card enable lines. The primary enable is provided by the CR write byte 1 signal (¬PCLWSWB(1)) when byte 1 of the CR is involved in the write (when byte 0, 2, or 3 is involved in the write, the associated ¬PCLWSWB signal provides the primary enable used in developing the write card enable lines for the desired byte). When the primary and secondary enables are present, the group of 9B logic modules uses the first three bits of the CR write card and word select signals (¬PCLWW(0-5)) in true and complement form to develop the eight write card enable lines. These eight lines are inverted by a group of 2N logic modules and distributed to the CRBASE1 and CRCELL1(0-6) cards (PCWA2B1E(0) goes to CRBASE1, PCWA2B1E(1) goes to CRCELL1(0), etc.). The complement outputs of the flip-flops used to hold all six bits of the CR write card and word select signals are inverted and output to the MIRMRGB card as the write card and word selector (PCOWWB1(0-5)). The last three bits from these same flip-flops are inverted and output to CRBASE1 and CRCELL1(0-6) as the write word selector. The complement outputs of the flip-flops used to hold the write enable and the byte 1 enable are inverted and output to the MIRMRGB card (PCOA2RB1 and PCOWSWB(1), respectively). The left hex (¬PCLWSWL) and right hex (¬PCLWSWR) indicators from CRMIRLDR are gated into flip-flops, inverted twice (once in the flip-flops), and output to CRBASE1 and CRCELL1(0-6) as the write left hex enable and write right hex enable, respectively. This same process (a different set of 2N logic modules is used

to provide the second inversion) develops the write left hex (PCOWB1L) and write right hex (PCOWB1R) signals for MIRMRGB. The other three CRCONT cards perform similar processing at the same time, so the net result may be writing a fullword, halfword, byte, hex, or isolated bits when the R field is used as a mask.

4-134 CRCELLY control. Refer to figure 4-60 for a simplified logic diagram of the CRCELLY control logic that responds to hardware initiated interrupts by setting or resetting bits in the interrupt associated control bytes of the CR file (byte 0 of CR's 0 through 7). This control logic accepts interrupts generated internally by the PP or externally by other components of the ASC system and provides the necessary control over the interrupt associated control bytes of the CR file to carry out the processing of those interrupts. Refer to appendix B of this manual for the format of byte 0 of CR's 0 through 7.

The time slot override control byte and the time slot override reason byte of the CR file are under direct control of the CRCELLY control logic when a CM parity error, CM protect violation, CM breakpoint (not used), CR protect violation, or illegal op-code occurs in a VP other than the selected VP (the selected VP exercises the PP maintenance logic during normal processing). When one of these conditions exists, the associated signal (¬PMPAOS for parity error, ¬PMPROS for protect violation, ¬PMBPOS for breakpoint, ¬PXILLC for illegal op-code, or ¬PXINHIB for CR protect violation) is used to develop the gates for both the override control and override reason bytes and the data for the override reason byte only. The data for the override control byte (¬PFDD00CR(0) through ¬PFDD07CR(0)) is developed by holding the inputs to a group of 2N logic modules at the V_{EE} level so that a bit is set in the override control byte whenever the associated gate occurs. When a parity error, protect violation, or breakpoint occurs, the PYCMVIOL signal combines with the ¬PYERSWEQ signal (developed as long as the CM violation occurs in a VP other than the selected VP) to enable the DE logic module used in generating the gates for the override control byte. The PYCMVIOL signal also enables the VP code of the VP causing the CM violation (the ¬PMERRC(0-2) signals from SWBASY) through to the mentioned DE module and to a group of 2N logic modules for generation of the VP code data in the override reason byte. The enabled DE module decodes the applied VP code to generate a gate (and hence set the associated bit) for the VP in the override control byte that caused the CM violation. If a parity error occurred, none of the reason data bits in the override reason byte (¬PFDD05CR(1) through ¬PFDD07CR(1)) are affected (all zeroes). If a breakpoint occurred, the middle reason data bit is set. If a protect violation occurred, the LSB reason data bit is set. The data line for the control bit of the override reason byte (¬PFDD00CR(1)) is constantly held for a set condition and the data line for bit 4 of the override reason byte (¬PFDD04CR(1)) is constantly held for a clear condition. The gate lines for the override reason byte are developed only if the control bit (¬PCQD00CR(1)) of the current override reason

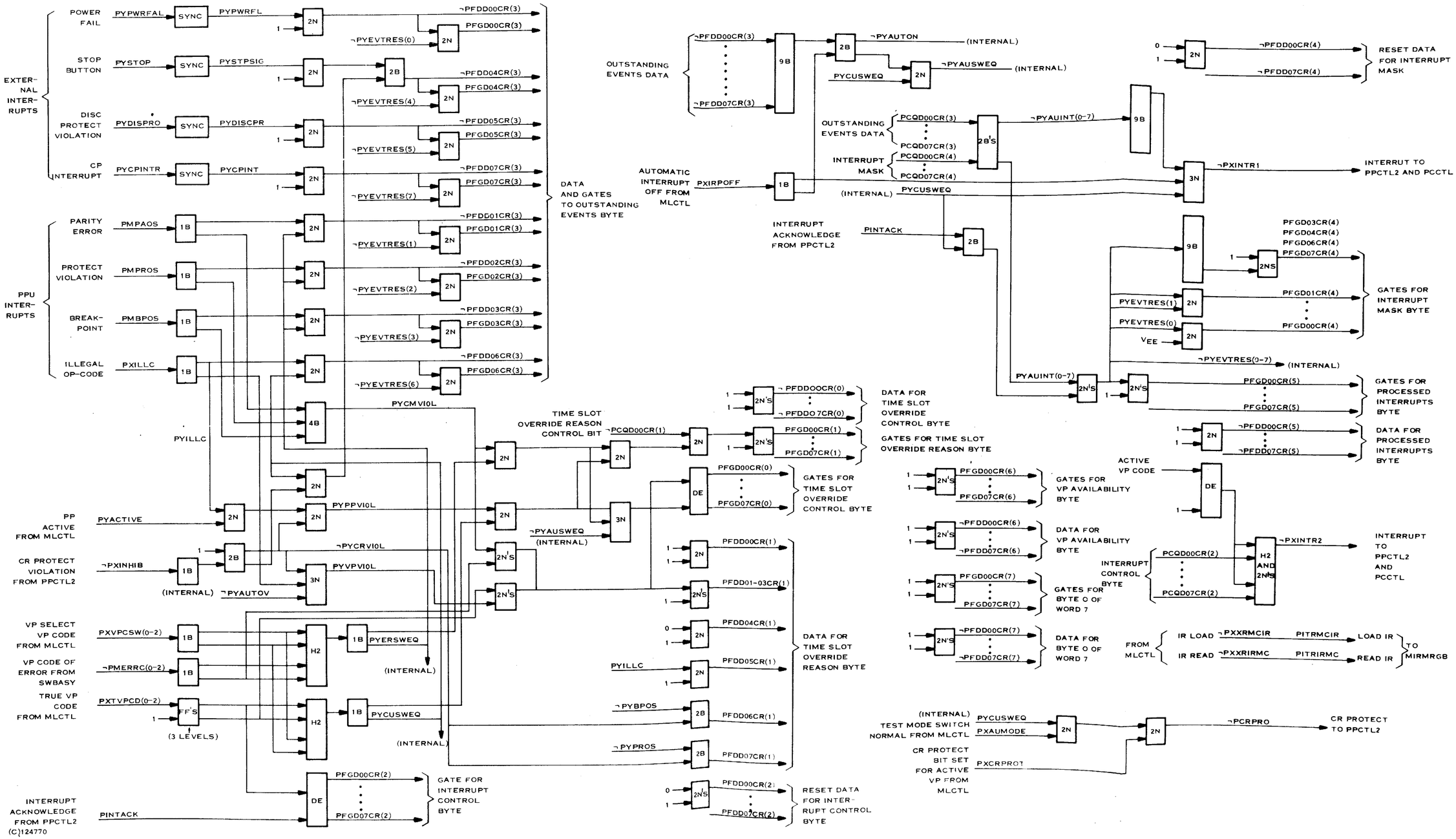


Figure 4-60. CRCELLY Control Logic

byte is clear. If it is clear, the mentioned PYCMVIOL and ¬PYERSWEQ signals combine with the control bit to generate a gate for each bit in the override reason byte. The net result is the updating of the override reason byte simultaneously with the setting of a bit in the override control byte to reflect the VP number and CM violation causing the error. When an illegal op-code or CR protect violation occurs, the PYPPVIOL signal replaces the PYCMVIOL signal in the development of the override control gate and enables the VP code reflecting the error through a group of 2N logic modules to the VP code lines of the override reason byte. If an illegal op-code occurred, the first reason data bit in the override reason byte is set by the PYILLC signal; if a CR protect violation occurred, the last two reason data bits are set by the ¬PYCRVIOL signal.

In addition to the cases mentioned in the previous paragraph, a bit in the time slot override control byte is set when the AUTO INTRPT OFF button on the ASC Maintenance Console is pressed and an interrupt external to the PP occurs, the selected VP causes a CM violation, an illegal op-code occurs in the selected VP, or a CR protect violation occurs in the selected VP when the PP is not operating in the normal mode. In these cases, the bit associated with the selected VP in the time slot override byte is set. When any one of these cases occurs, the ¬PYAUTOV signal is developed due to an entry in the outstanding events byte and the pressing of the AUTO INTRPT OFF button, while the ¬PYAUSWEQ signal is developed due to these two conditions only when the selected VP is active (PYCUSWEQ). The ¬PYAUTOV signal is used to generate the PYVPVIOL signal, which in turn enables the VP code reflecting the interrupt (in this case the selected VP code) through a group of 2N logic modules to the DE logic module used in generating the gate for the time slot override control byte. The ¬PYAUSWEQ signal enables the same DE module to permit the decoding of the applied VP code and the generation of the selected VP gate for the time slot override control byte.

When an interrupt external to the PP occurs (power fail, disc protect violation, Central Processor (CP) interrupt, or activation of the ASC Operator's Console STOP button), the associated synchronizer on CRCELLY is triggered (the synchronizers are discussed in a later paragraph). The triggered synchronizer develops a pulse coincident with the second clock after the interrupt (PYPWRFL for power fail, PYDISCPR for disc protect violation, PYCPINTR for CP interrupt, or PYSTPSIG for STOP button interrupt). The pulse developed by the synchronizer is inverted by a 2N logic module and input to the associated bit of the outstanding events byte (byte 0 of CR 3) as data. The same pulse that is used as data by the outstanding events byte is inverted and used as a gate to set the bit in the outstanding events byte associated with the interrupt. When a CM violation (parity error, protect violation, or breakpoint) occurs in the selected VP, the interrupt signal combines with the select VP CM violation signal (PYERSWEQ) to develop the data and gate needed to set the bit in the outstanding events byte that reflects the interrupt. When an illegal op-code occurs in the selected VP, the interrupt

signal combines with the select VP active signal (PYCUSWEQ) to develop the data and gate needed to set the illegal op-code bit in the outstanding events byte.

Each bit in the outstanding events byte (¬PCQD00CR(3) through ¬PCQD07CR(3)) is paired with the associated bit in the interrupt mask (¬PCQD00CR(4) through ¬PCQD07CR(4)) on the input lines of a 2B logic module. If an outstanding event is recorded and the associated interrupt mask bit is set, the corresponding ¬PYAUINT signal combines with the select VP active signal to generate an interrupt signal (¬PXINTR1) for the selected VP. When the selected VP responds to the interrupt by pulsing the PINTACK line, the PYAUINT line associated with the interrupt is used to develop a processed interrupts byte gate so that the bit reflecting the interrupt is set. (The data lines to the processed interrupts byte are all constantly held for a set condition, so whenever a gate occurs, the associated bit is set.) The PYAUINT line associated with the interrupt is also used to develop a corresponding ¬PYEVTRES signal, which generates a clearing gate for the outstanding events bit responsible for the interrupt to the selected VP. The same ¬PYEVTRES signal is used to generate the gates needed to reset all bits in the interrupt mask byte that represent events of a priority equal to or lower than the event that caused the interrupt. If a power fail interrupt (highest priority) is honored by the selected VP, the ¬PYEVTRES(0) signal generates a gate for all bits of the interrupt mask byte. (The data lines to the interrupt mask byte are all constantly held for a clearing condition, so whenever a gate occurs, the associated bit is cleared.) If a parity error (middle priority) is honored, the ¬PYEVTRES(1) signal generates a gate for all bits of the interrupt mask byte except for the gate to the power fail bit. If the interrupt honored is neither a power fail nor a parity error, the associated ¬PYEVTRES signal (¬PYEVTRES(2) through ¬PYEVTRES(7)) generates a gate for only those bits of the interrupt mask with a low priority (CM protect violation in selected VP, CM breakpoint in selected VP, system STOP button interrupt, disc protect violation, illegal op-code in selected VP, and CP interrupt). At the conclusion of the software service routine, the processed interrupts byte should be reset and the interrupt mask should be restored so interrupt processing can return to normal.

Each VP may be interrupted by a bit in the interrupt control byte set via software (programmed interrupt). When a bit is set in the interrupt control byte, the active VP code is decoded and paired with the set bit to develop an interrupt (¬PXINTR2) for the VP designated by programmed interrupt bit. When the VP acknowledges the interrupt by pulsing the PINTACK line, the VP code reflecting the interrupted VP is decoded to generate a gate to the associated bit in the interrupt control byte. The hard wired data lines on CRCELLY to the interrupt control byte are all constantly held for a clearing condition, so whenever a gate occurs, the associated bit is cleared.

CRCELLY control of the VP availability byte and byte 0 of word 7 is inhibited by hard wiring the gates so that they are constantly disabled. This gives software total control over both of these bytes. CRCELLY also disables the CR protect logic on the PPCTL2 card when the selected VP is active and the TEST MODE switch on the ASC Maintenance Console is in the NORMAL position. When these two conditions occur, the select VP active signal (PYCUSWEQ) and the TEST MODE switch NORMAL position signal (PXAUMODE) combine to hold the ¬PCRPR0 line at one (thus disabling the CR protect logic on PPCTL2). If either of these two signals does not occur and the CR protect bit for the active VP is set (PXCRPROT), the ¬PCRPR0 line enables the CR protect logic on PPCTL2.

4-135 INPUT SYNCHRONIZERS. The CR file input synchronizers are used to synchronize gating signals from external TTL devices with the PP clock so that data from the external devices is not lost at the interface with the CR file. All of the synchronizers are identical and exist on the CRCELLY card and each CRCELL and CRCONT card. The CRCELLY card has four single output synchronizers that are used to interface external interrupts (power fail, disc protect violation, CP interrupt, or STOP button interrupt) with the synchronous control logic used in setting bits in the outstanding events byte. Each CRCELL card has ten single output synchronizers that are used to interface data from external devices with a bit or bits in the CR file. Each CRCONT card has twelve single output synchronizers, three two output synchronizers, four four output synchronizers, and one five output synchronizer. All of these synchronizers are used to interface data from external devices with a bit or bits in the CR file. The multiple output synchronizers are used when one gate is required to enable data to more than one CR motherboard. Refer to the PP Fixed Variable List and the Patch Card specification drawing for the assignment of the synchronizers to the bits in the CR file. Section XI of the TTL Designer's Manual, ASC part number 930004-1, provides a more detailed look at the patch card interface between the synchronizers and the CR file gates.

A simplified logic diagram of the input synchronizers and the associated timing diagram are shown in figures 4-61 and 4-62, respectively. During the quiescent state of the synchronizer (at least two PP clocks after the last synchronous gate), all three inputs to the DE logic module are zero. When an asynchronous gate or external interrupt occurs, (the ¬$SYBMIN(P) signal makes a one-to-zero transition), the zero output of the DE module and the inverting output of the 1B module combine to set the number two DF flip-flop. The flip-flop output drives the number one output of the DE module to one, so when the first PP clock occurs, the number one DF flip-flop is set. The number one and two DF flip-flops (both are now set) combine to drive the number three output of the DE module to one. At the conclusion of the first clock, DF flip-flop zero is set and the synchronous gate (¬$XSYBMON(P)) makes a one-to-zero transition (DF flip-flops zero and one are set). At this instant, all three DF flip-flops are set so that the number seven output of the

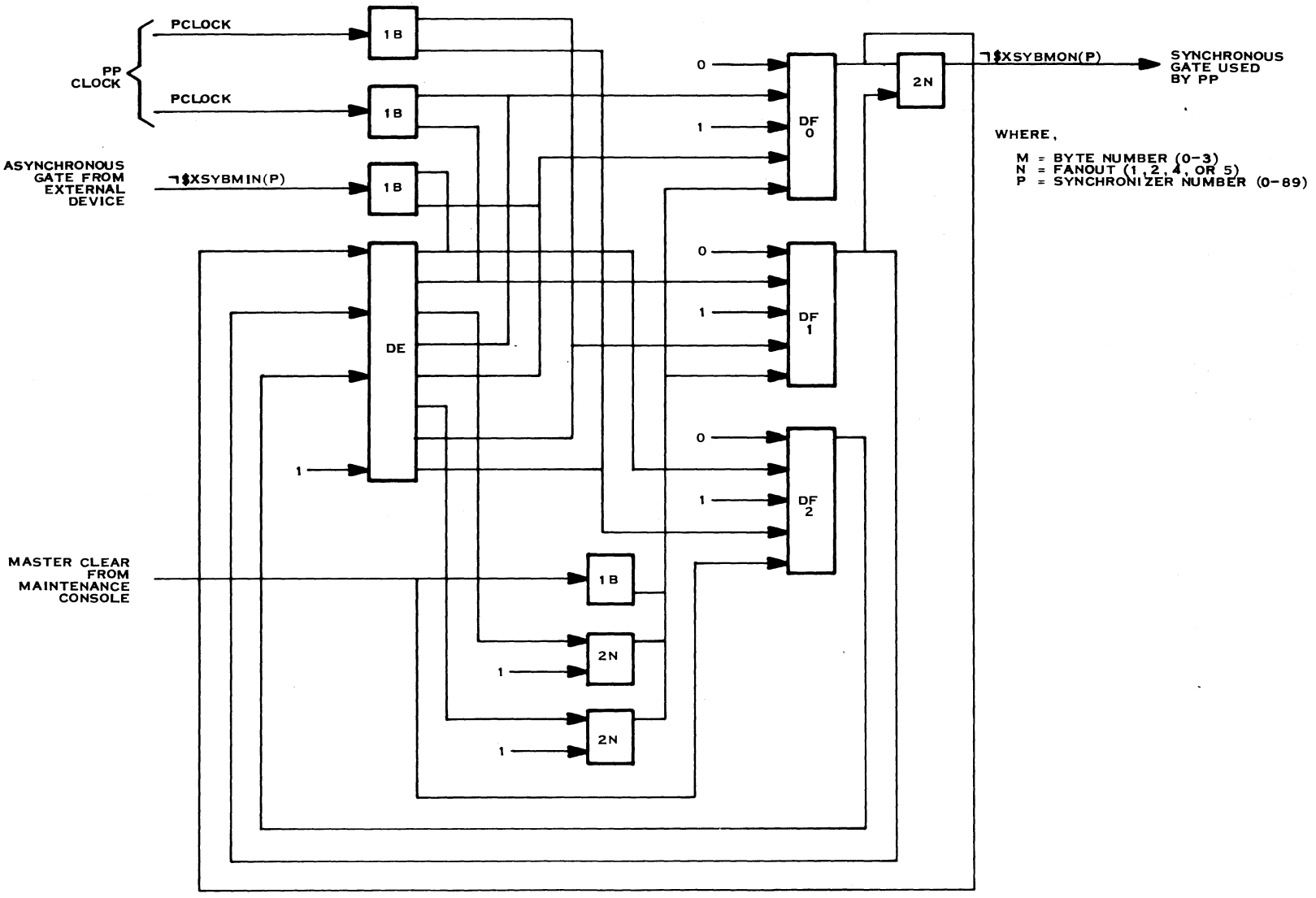


Figure 4-61. Input Synchronizers

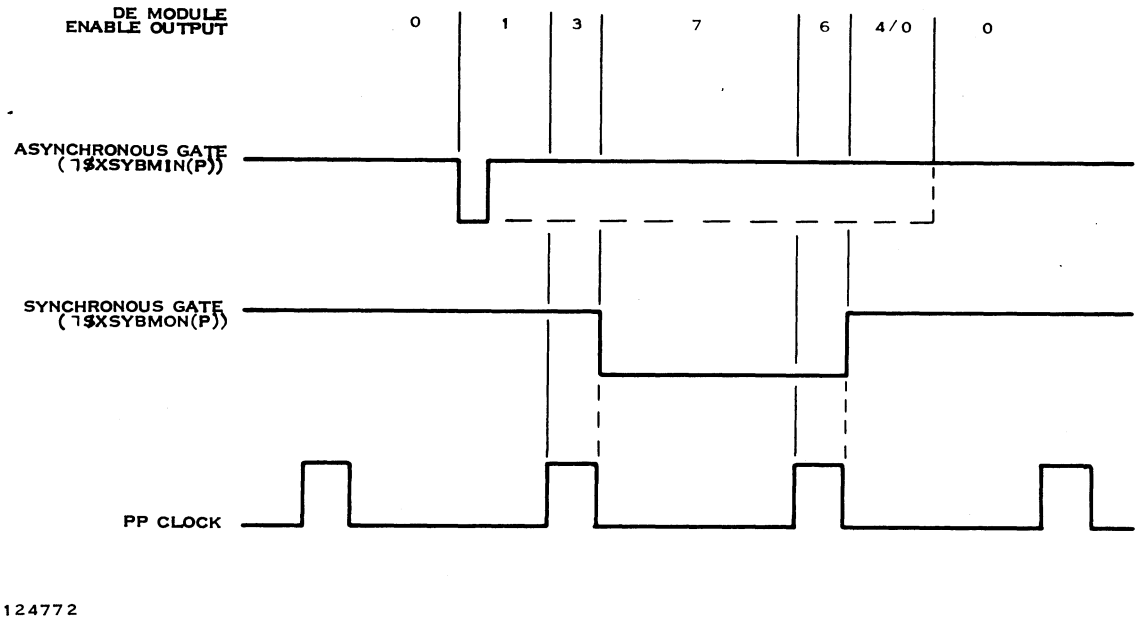


Figure 4-62. Input Synchronizer Timing Diagram

DE module is one. When the second clock occurs, the number two DF flip-flop is cleared and the number six output of the DE module goes to one. At the conclusion of the second clock, DF flip-flop one is cleared and the synchronous gate returns to one (the number four output of the DE module is now one). The number four output of the DE module combines with the true output of the 1B module handling the asynchronous gate to clear DF flip-flop zero. The synchronizer is now back in its quiescent state. If an invalid DE module output occurs (the number two or number five output goes to one), the invalid output is input to a 2N logic module, which in turn clears DF flip-flops zero and one. If the MASTER CLEAR button on the Maintenance Console is pressed when the PP is operating in the manual mode, all three DF flip-flops are cleared.

4-136 CR REGISTERS. The CR registers consist of 64 32-bit registers situated on 32 cards as shown in figure 4-63 (each card is capable of holding a byte of data for eight CR's). Refer to the Central Memory Base Register portion of the VP detailed theory for a description of the registers on the CRBASE cards. The registers on the CRCELLY card are used to process hardware detected and software initiated interrupts. The registers on the CRCELL cards provide control for and data links to the peripheral equipment, in addition to some control over system operation. A detailed description of the registers and associated loading and distribution logic on the CRCELLY

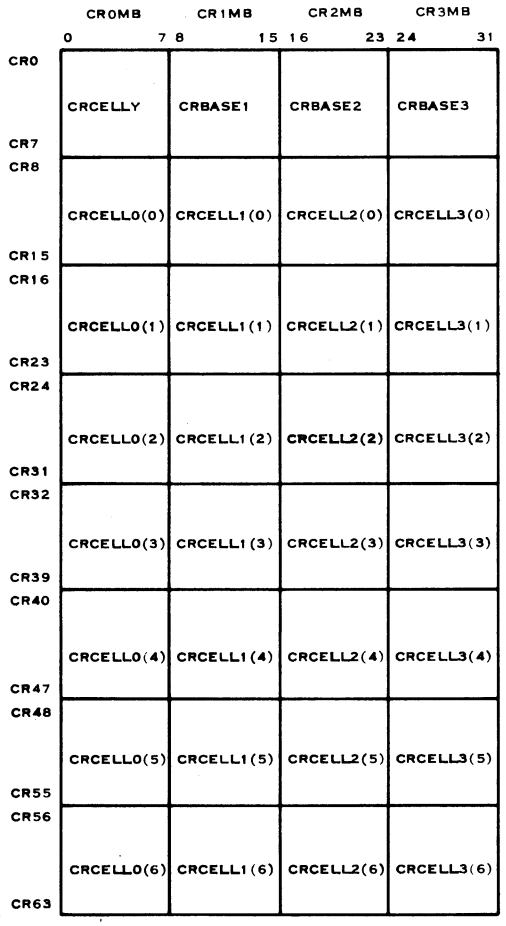


Figure 4-63. CR File Card Layout

card and the CRCELL cards is presented in the following paragraphs. A final paragraph is presented to describe the merging of data from the CRBASE, CRCELLY, and CRCELL cards to the MDB.

4-137 CRCELLY Registers. Refer to figure 4-64 for a simplified logic diagram of the CRCELLY registers (each register consists of one byte) and the associated input and output logic. The PP software is capable of setting or resetting every bit in all eight registers, whereas the CRCELLY control logic has direct control over only the time slot override control byte, time slot override reason byte, interrupt control byte, outstanding events byte, interrupt mask byte, and the processed interrupts byte. The software also has the capability of monitoring any of the eight CRCELLY registers via the MDB.

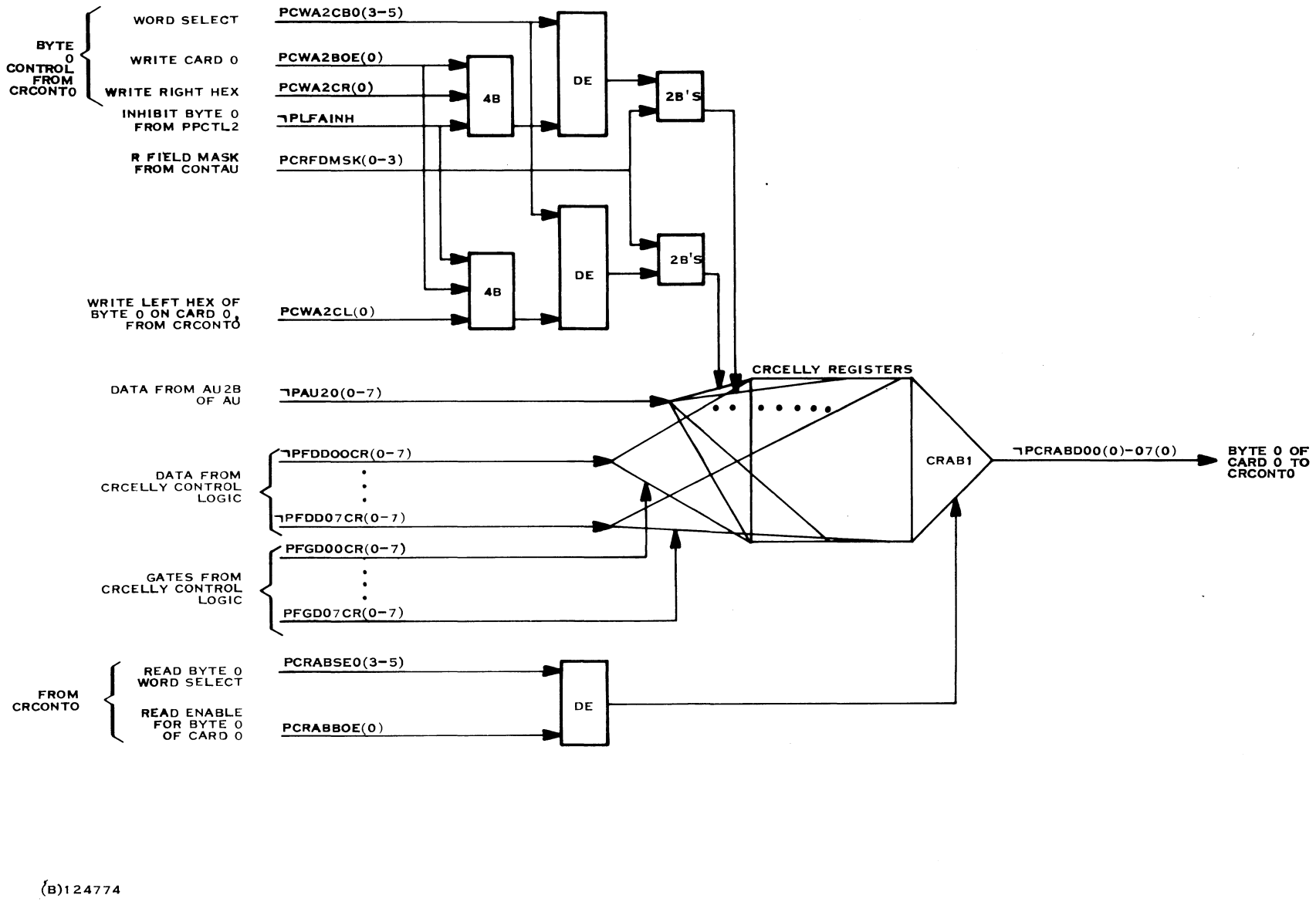


Figure 4-64. CRCELLY Registers

When the executing instruction requires data to be written to one of the CRCELLY registers (via the AU2B bus), the CRCONT0 card supplies the CRCELLY card with the necessary control lines to locate a hex of data. These control lines include a write byte 0 of card 0 (PCWA2B0E(0)) signal used to designate the CRCELLY registers, a byte 0 word select (PCWA2CB0(3-5)) signal used to select one of the eight CRCELLY registers, and the write right (PCWA2CR(0)) and/or left (PCWA2CL(0)) hex signals used to select the right and/or left half of the selected byte. The PPCTL2 card provides an inhibit byte 0 of CM base load signal (¬PLFAINH) used to enable the loading of data into the CRCELLY registers when data is not being loaded in a CM base register. The CONTAU card supplies the R field mask signals (PCRFDMSK(0-3)) to provide additional control over data written to a CRCELLY register when a test and set (TS), test and reset (TR), set (S), or reset (R) instruction is being executed. When a TS, TR, S, or R type instruction is being executed and data is to be written to the right half of a CRCELLY byte, the write byte 0 of card 0 signal, write right hex signal, and the complement of the inhibit byte 0 of CM base load signal combine to enable the decoding of the word select signals. The result of the decode is masked by the R field such that the AU2B data (¬PAU20(4-7)), marked by ones in the R field, is inserted in the right half of the intended CRCELLY byte. When an instruction requires data to be written to the left half of a CRCELLY byte, the write left hex signal replaces the write right hex signal so the appropriate AU2B data (¬PAU20(0-3)) is inserted in the left half of the intended CRCELLY byte. A byte of data from the AU2B bus is written to the intended CRCELLY byte when both the left and right hex signals are activated.

The CRCELLY control logic supplies separate data and gate lines for each bit in all eight of the CRCELLY bytes. When the time slots of a VP are to be voided, the associated time slot override control byte data and gate lines combine to set the bit in the override control byte reflecting the VP to be deactivated. If the control bit of the time slot override reason byte is not set, the CRCELLY control logic develops a gate for each bit in the time slot override reason byte so new data reflecting the VP number and reason for the time slot voiding can be inserted. When an automatic interrupt occurs, the outstanding events byte data and gate lines reflecting the interrupt combine to set the associated bit in the outstanding events byte. If the corresponding interrupt mask bit is set and the selected VP acknowledges the recorded event, data and gate lines reflecting the event are developed for the processed interrupts byte. In addition, data and gate lines for the interrupt mask byte bits with a priority equal to or lower than that of the event are developed to reset those bits, and a gate is developed to reset the event recorded in the outstanding events byte that caused the interrupt. When software generates a programmed interrupt and the executing VP acknowledges that interrupt, a gate and data line are developed to reset the associated bit in the interrupt control byte. The gates developed by the CRCELLY control logic for

the VP availability byte and byte 7 on CRCELLY are disabled, so software has total control over these two bytes.

When a byte of data is to be read from CRCELLY, the read enable for byte 0 of card 0 signal (PCRABB0E(0)) permits the decoding of the read byte 0 word select signals (PCRABSE0(3-5)). The resulting select line enables the intended byte over the CRAB1 bus to the CRAB2 bus on the CRCONT0 card.

4-138 CRCELL Registers. The CR file contains 28 CRCELL cards, each card containing a byte of storage capability for eight words. A simplified logic diagram of the CRCELL1(0) registers (each register consists of one byte) and the associated input and output logic is shown in figure 4-65. The other CRCELL cards contain the same logic as that shown in figure 4-65, but use different sets of control signals and handle different data. Each of the four groups of seven CRCELL cards that have the same byte number (CRCELL0(0-6), CRCELL1(0-6), CRCELL2(0-6), and CRCELL3(0-6)) is controlled by software via the associated CRCONT card (CRCONT0, CRCONT1, CRCONT2, and CRCONT3, respectively).

Each CRCONT card provides its group of CRCELL cards (and CRCELLY or CRBASE card, whichever is applicable) with a write card enable (CRCONT0 develops PCWA2B0E(0-7), CRCONT1 develops PCWA2B1E(0-7), CRCONT2 develops PCWA2B2E(0-7), and CRCONT3 develops PCWA2B3E(0-7)) which is used to select one of eight possible cards. A write word select (CRCONT0 develops PCWA2CB0(3-5), CRCONT1 develops PCWA2CB1(3-5), etc.) is used to select one of eight bytes on the chosen card. The write left hex and right hex selectors (CRCONT0 develops PCWA2CL(0) and PCWA2CR(0), CRCONT1 develops PCWA2CL(1) and PCWA2CR(1), etc.) are used to enable the write operation to the left and/or right half of the chosen byte. A read card enable (CRCONT0 develops PCRABB0E(0-7), CRCONT1 develops PCRABB1E(0-7), etc.) is used to select one of eight possible cards. A read word select (CRCONT0 develops PCRABSE0(3-5), CRCONT1 develops PCRABSE1(3-5), etc.) is used to select one of eight bytes on the chosen card.

The write controls from the CRCONT cards direct the storage of the AU2B bus data in byte increments (CRCONT0 controls ¬PAU20(0-7), CRCONT1 controls ¬PAU20(8-15), CRCONT2 controls ¬PAU20(16-23), and CRCONT3 controls ¬PAU20(24-31)) and the read controls supply the CRAB2 bus with CR file data in byte increments (CRCONT0 supplies ¬PCRABD00(0-7) through ¬PCRABD07(0-7), CRCONT1 supplies ¬PCRABD08(0-7) through ¬PCRABD15(0-7), etc.). The following paragraphs apply to the CRCELL1(0) card; however, operation of the other 27 CRCELL cards is identical except for the mentioned control and data differences.

When a test and set (TS), test and reset (TR), set (S), or reset (R) type instruction is being executed and data is to be written to the right half of a CRCELL byte, the write byte 1 of card 1 signal (PCWA2B1E(1)) and the write right hex of

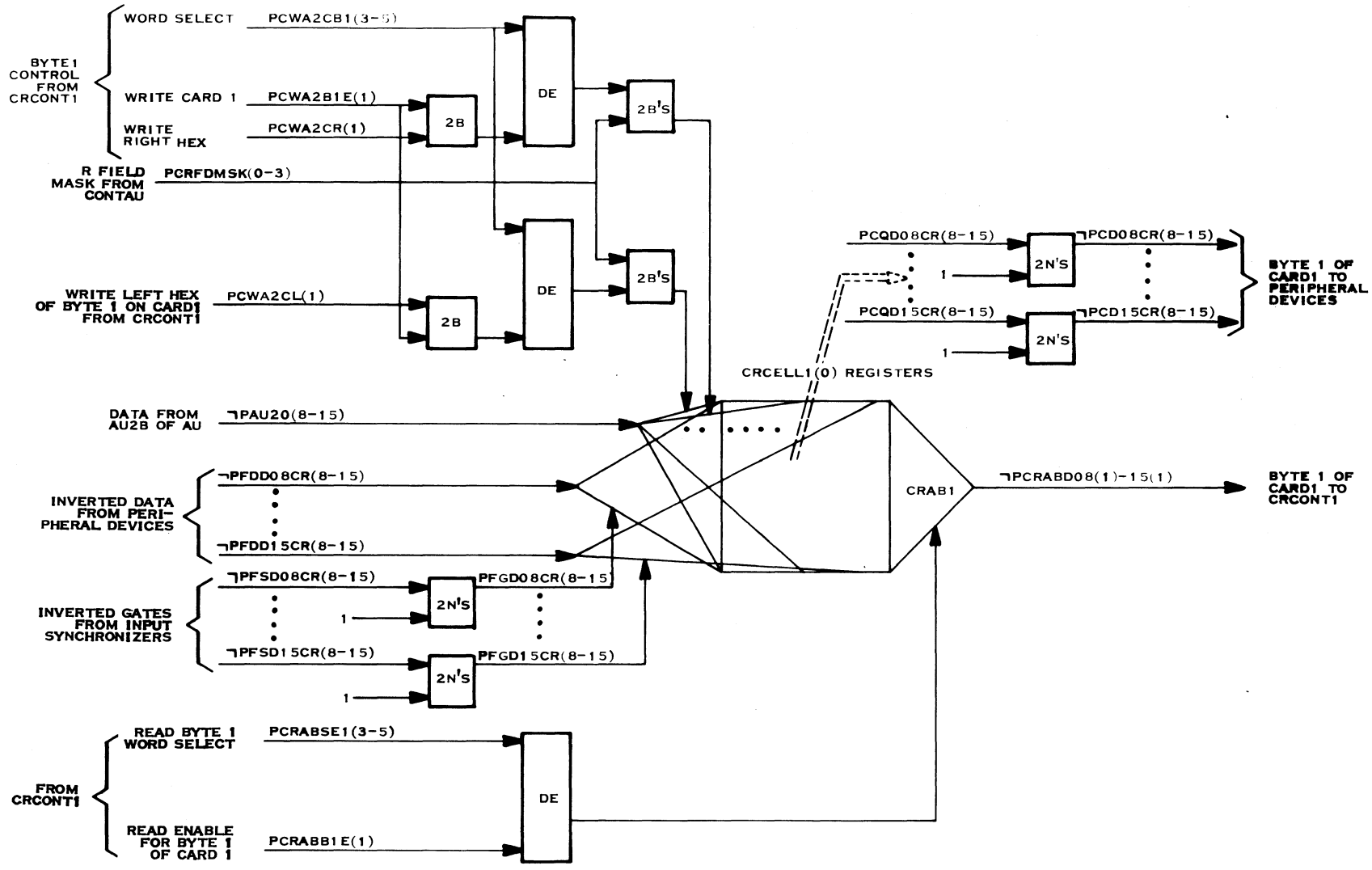


Figure 4-65. CRCELL1(0) Registers

byte 1 signal (PCWA2CR(1)) combine to enable the decoding of the word select signals (PCWA2CB1(3-5)). The result of the decode is masked by the R field such that the AU2B data (¬PAU20(12-15)) marked by ones in the R field is inserted in the right half of the intended CRCELL byte. When an instruction requires data to be written to the left half of a CRCELL byte, the write left hex signal replaces the write right hex signal such that the appropriate AU2B data (¬PAU20(8-11)) is inserted in the left half of the intended CRCELL byte. A byte of data from the AU2B bus is written to the intended CRCELL byte when both the left and right hex signals are activated. In addition to the PP software, the peripheral devices of the ASC system have direct access to the CRCELL registers via the inverted data (PFDD08CR(8-63) through PFDD31CR(8-63)) and synchronized gate (¬PFSD08CR(8-63) through ¬PFSD31CR(8-63)) lines. Refer to appendix B of this manual for the data format of CR's 8 through 63 (the Description of the ASC CR File, part number 930207-1, provides more detailed coverage of the same information) and to the PP Fixed Variable List and Patch Card specification drawing for the assignment of the gate lines to the data in these registers.

When the PP software requires data to be read from one of the bytes on the CRCELL1(0) card, the read enable for byte 1 of card 1 signal (PCRABB1E(1) permits the decoding of the read byte 1 word select signals (PCRABSE1(3-5)). The resulting select line enables the intended byte over the CRAB1 bus to the CRAB2 bus on the CRCONT1 card. The output from each of the CRCELL1(0) flip-flops is also inverted by a 2N logic module and routed to the applicable peripheral device.

4-139 CR File Output Merging. Refer to figure 4-66 for a simplified logic diagram of the merging logic on CRCONT0, CRCONT1, CRCONT2, and CRCONT3. Each of the CRCONT cards uses eight 9B logic modules to select a byte of data from one of the eight cards having the same byte number as the associated CRCONT card. Inputs to each of the 9B logic modules consist of eight data lines representing the output from the eight associated cards having the same bit (and byte) number. During a CR read operation, seven of the eight data lines to each of the 9B modules are held at one (only one CR word at a time can be read from the CR file). The remaining line is enabled through the 9B module to the MDB on one of the VPRCARD cards. Each CR read operation involves a full word, so 32 bits of data are enabled through the 9B modules on the CRCONT cards (eight bits per CRCONT card) at each read. The output lines ¬PCRAB(0-3) are routed to VPRCARD(0), ¬PCRAB(4-7) are routed to VPRCARD(1), ¬PCRAB(8-11) are routed to VPRCARD(2), ¬PCRAB(12-15) are routed to VPRCARD(3), ¬PCRAB(16-19) are routed to VPRCARD(4), ¬PCRAB(20-23) are routed to VPRCARD(5), ¬PCRAB(24-27) are routed to VPRCARD(6), and ¬PCRAB(28-31) are routed to VPRCARD(7).

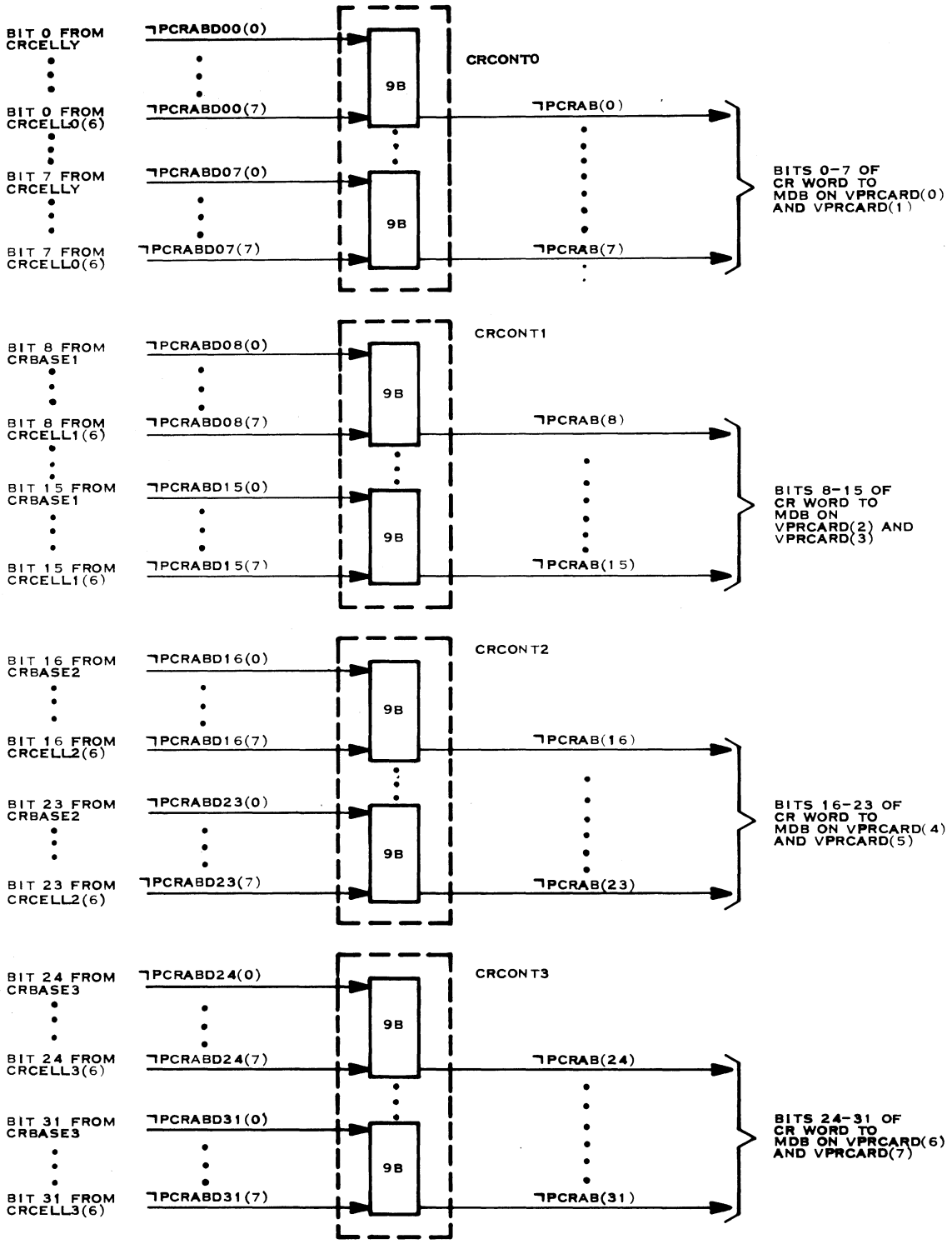


Figure 4-66. CR File Output Merging Logic (CRAB2 Bus)

4-140 READ ONLY MEMORY

The Read Only Memory (ROM) of the PP is functionally divided into ROM addressing logic and ROM merging logic for description purposes. A detailed description of these two functional areas is presented in the following paragraphs. Refer to the ROM simplified block diagram in figure 4-67 for aid in following the detailed description.

4-141 ROM ADDRESSING LOGIC. When the mode bit of the instruction register of the active VP indicates that the next instruction is to be retrieved from ROM, the ROM address is split into two groups (PMROMADD(20-23) and PMROMADD(24-31)) and input to the ROM addressing loigc. Bits 20 through 23 of the ROM address are decoded to 16 enable lines (PMRMSSEN(0-15)) that are used to enable one of the 16 ROMCRD cards. Bits 24 through 31 of the ROM address are distributed to each of the 16 ROMCRD cards, where they are decoded to 256 word select lines (¬MRW000-255) and applied to 192 diode matrices (capable of holding 256 32-bit words with a complement bit for each word). Each ROMCRD card responds to its own set of word select lines with a 32-bit data word and complement bit, which are applied to the ROM merging logic along with the associated enable (MRDEN).

4-142 ROM MERGING LOGIC. The ROM retreived word and complement bit associated with the activated enable are output through a group of 9B logic gates on the selected ROMCRD card. If the selected ROMCRD card is ROMCRD(0), ROMCRD(1), ROMCRD(2), or ROMCRD(3), bits 0 through 7 of the retrieved word are input to the merging logic on the AU2XFER card, bits 8 through 31 of the retreived word are input to the merging logic on the ROMMRG card, and the complement bit is input to the complement control merging logic on the ROMMRG card. If the selected ROMCRD card is ROMCRD(4), ROMCRD(8), or ROMCRD(9), the retrieved word and complement bit are gated through the merging logic on CRROMRG(0) to the mentioned merging logic on AU2XFER and ROMMRG. The CRROMRG(1) card provides merging for ROMCRD(5), ROMCRD(10), and ROMCRD(11), CRROMRG(2) provides merging for ROMCRD(6), ROMCRD(12), and ROMCRD(13), and CRROMRG(3) provides merging for ROMCRD(7), ROMCRD(14), and ROMCRD(15).

The merging logic on the AU2XFER card (9B logic gates) generates the true and complement form of bits 0 through 7 of the word from the selected ROMCRD card and the merging logic on the ROMMRG card generates the true and complement form of bits 8 through 31. On both cards, the true and complement data are applied to separate groups of 2N logic gates along with the appropriate complement control signal. When the word retrieved from the selected ROMCRD card is in true form (the associated complement bit is

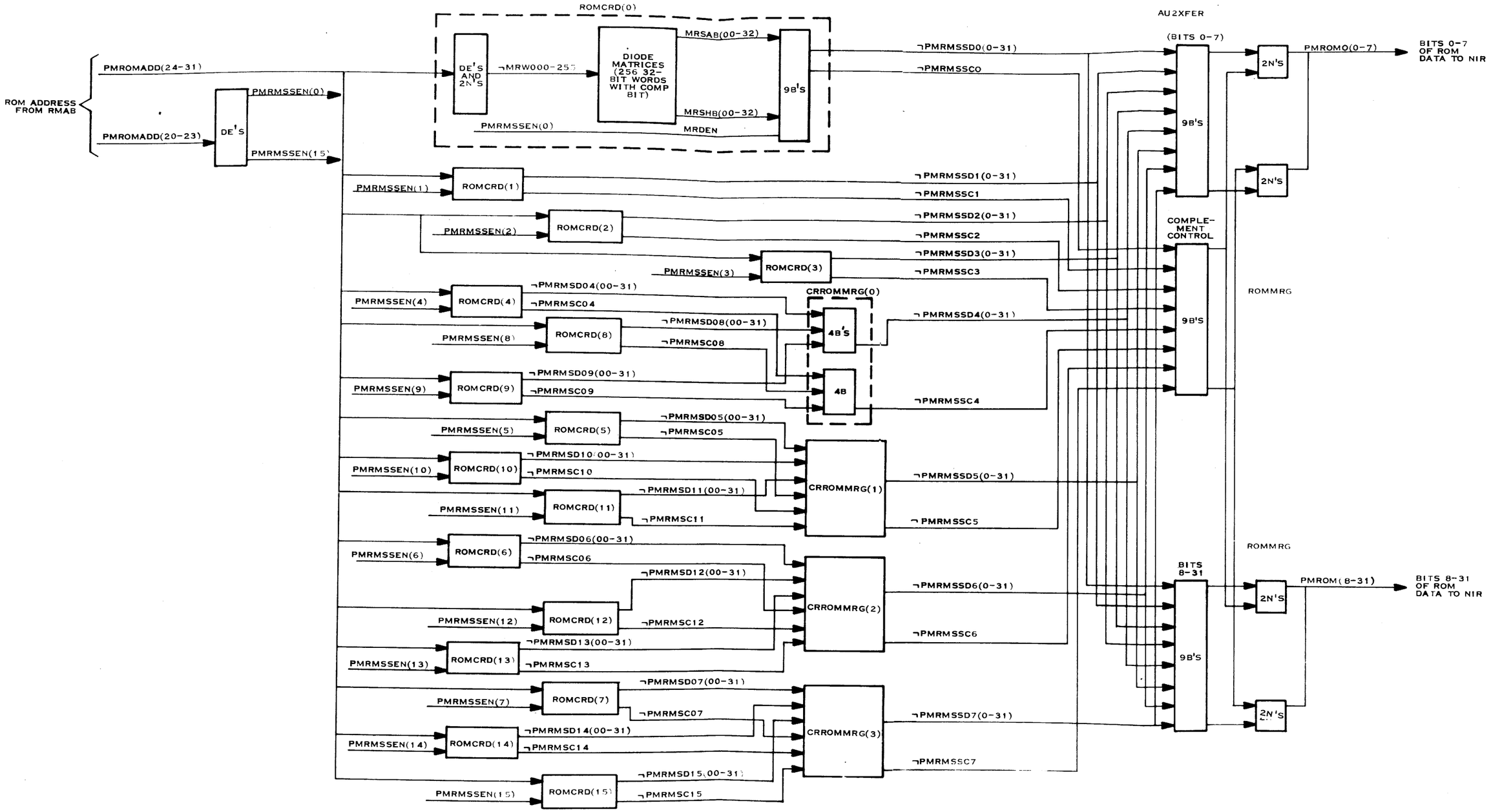


Figure 4-67. Read Only Memory (ROM)

zero), the true output of the 9B logic gates on the AU2XFER and ROMMRG cards is enabled (and inverted) through a group of 2N logic gates for a net result of two inversions (the first level of inversion was provided in the 9B logic gates of the selected ROMCRD card). The result, therefore, is 32 bits of ROM data in true form (PMROM0(0-31)) routed to the NIR of the active VP. When the word retrieved from the selected ROMCRD card is in complement form (the associated complement bit is set), the complement output of the 9B logic gates on the AU2XFER and ROMMRG cards is enabled (and inverted) through a group of 2N logic gates for a net result of three inversions. The result, as before, is 32 bits of ROM data in true form routed to the NIR of the active VP.

4-143 SINGLE WORD BUFFER CONTROLLER

The single word buffer controller of the PP is functionally divided into synchronous logic, asynchronous logic, and a two-way bus (TWB). A detailed description supplemented with block diagrams and/or logic diagrams is presented on each of these three functional areas in the following paragraphs.

4-144 SYNCHRONOUS LOGIC. Refer to figure 4-68 for a simplified block diagram of the synchronous logic portion of the single word buffer controller. The synchronous logic accepts memory access requests from executing instructions, stores the current active VP code in a high or low priority queue and the corresponding status data in a status file, and makes the necessary queue and file data available to the asynchronous logic portion of the single word buffer controller. Refer to figure 4-69 for the registers used by the synchronous logic.

4-145 High and Low Priority Queues. A memory access request is initiated when the decoded op-code and state indicate a read request, write request, or execute request is necessary (the execute request occurs when an instruction is to be retrieved from Central Memory). When a request is present, the active VP priority bits from the maintenance logic (¬PMNPBIT(0, 1)) and the PMRC and PMWC lines are combined to develop the high request (PMGIC(0)), low request (PMGIC(2), high enable (PMGC:2(0)), and low enable (PMGC:2(2)) signals. When a high priority VP is executing, the high request signal is used to increment the high priority input counter and the high enable signal is used to enable the decoding of the incremented input counter result (refer to figure 4-70 for a closer look at the input counter and its eight states). The decoded input counter result is used to locate the next available entry in the high priority queue. When a low priority VP is executing, a similar process occurs in order to locate the next available entry in the low priority queue. The next VP code signal from the maintenance logic (¬PMNVPC(0-2)) is applied to both the high and low priority queue selection logic at the end of the previous execution clock period. When a decoded input

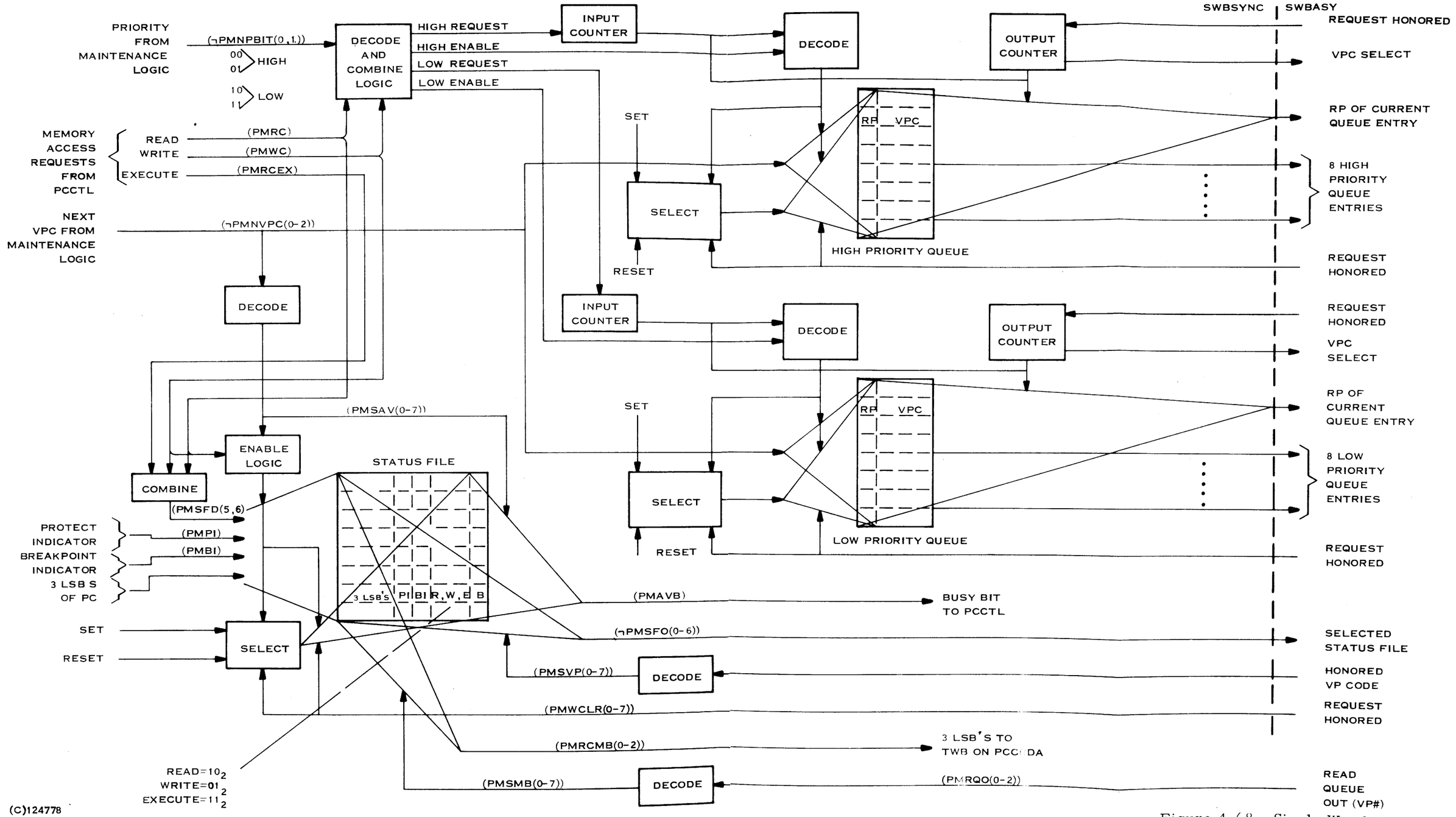


Figure 4-68. Single Word Buffer Controller Synchronous Logic

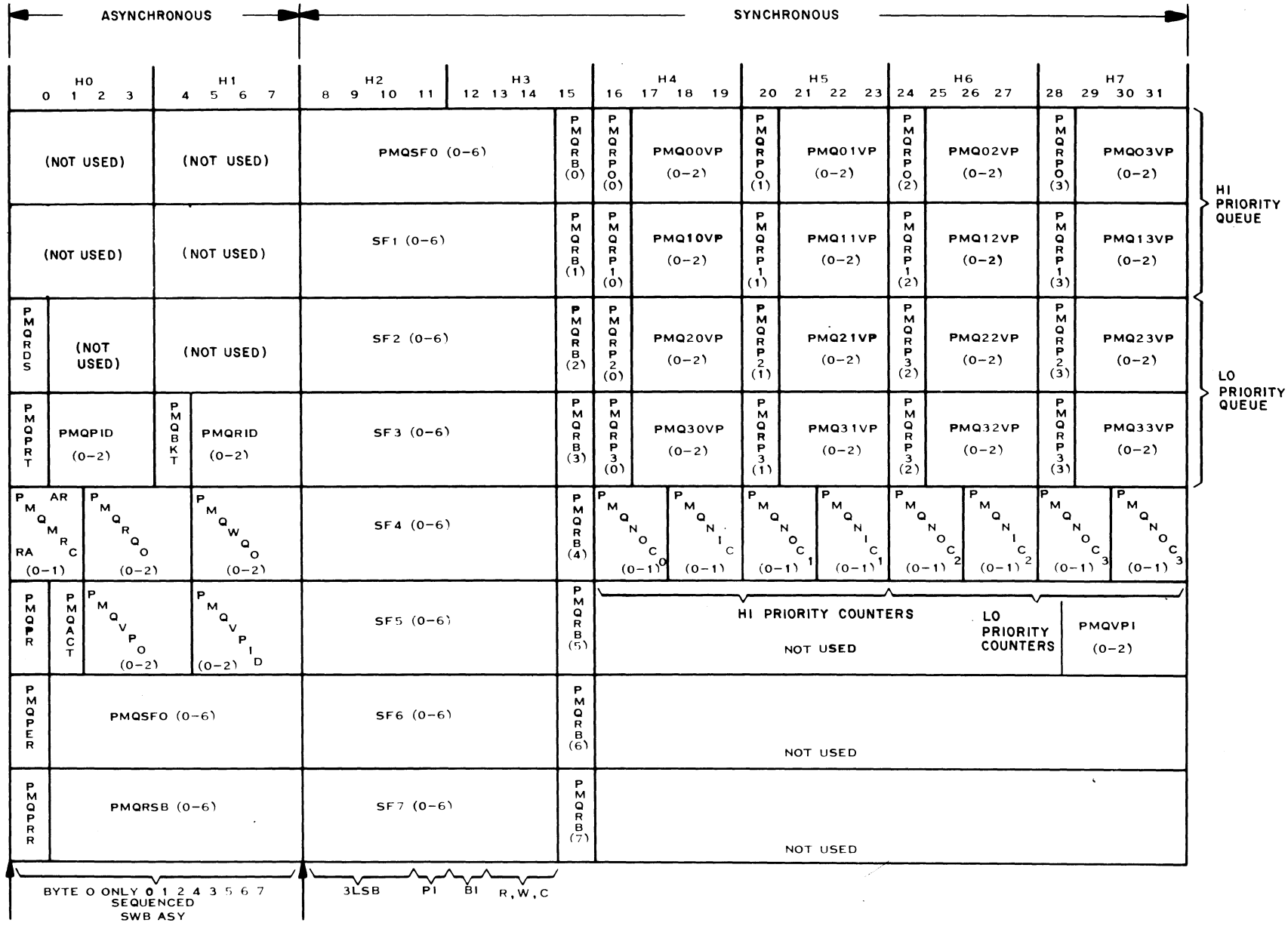
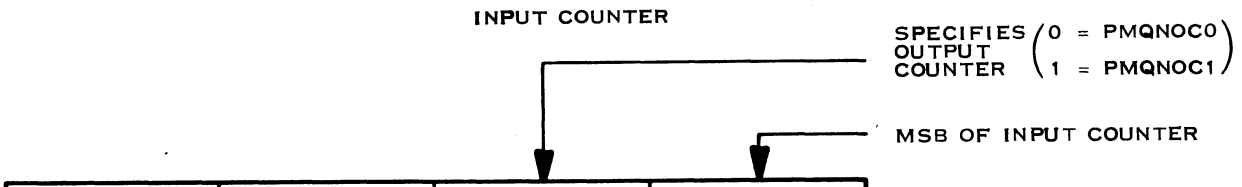


Figure 4-69. Single Word Buffer Controller Registers



PMQNIC0(0)	PMQNIC0(1)	PMNQNIC1(0)	PMNQNIC1(1)	COUNTER STATE
0	0	0	0	0
0	1	0	0	1
1	0	0	0	2
1	1	0	0	3
0	0	1	1	4
0	1	1	1	5
1	0	1	1	6
1	1	1	1	7

OUTPUT COUNTER

PMQNOC0(0)	PMQNOC0(1)	PMQNOC1(0)	PMQNOC1(1)	COUNTER STATE
0	0	0	0	0
0	1	0	0	1
1	0	0	0	2
1	1	0	0	3
0	0	0	0	4
0	0	0	1	5
0	0	1	0	6
0	0	1	1	7

(A)124779

Figure 4-70. Single Word Buffer Controller Input/Output Counters

counter result is detected, the VP code is inserted in the next available entry of the appropriate queue and the request present (RP) flag associated with the filled entry is set using the decoded input counter result. Each queue (high priority and low priority) has eight entries. Queue overflow is not a problem because each VP can have only one entry (the reason for this is described later). An individual output counter is associated with each of the two priority queues and is used in the asynchronous logic as a pointer to the current entry under examination and in the synchronous logic as part of a pointer to the current RP bit (refer to figure 4-68). When a memory access request has been processed by the asynchronous logic, the output counter of the appropriate queue is incremented to point to the next queue entry and the RP bit of the entry just processed is reset. Besides being used in the asynchronous logic, the output counter result is combined with the output counter

bit of the input counter in order to develop a pointer to the next RP bit. The selected RP bit is input to the asynchronous logic, where it is used to determine if more entries exist (RP set) in the current queue under examination. When all entries in the high priority queue have been processed (RP of the high priority queue reset), entries in the low priority queue are processed.

4-146 Status File. The status file consists of eight registers, each register corresponding to a specific VP number. A status file entry is made simultaneously with each high or low priority queue entry and includes the three LSB's of the PC (identify the memory zone specified by the PP), a protect indicator (PMPI) from the CR file, a breakpoint indicator (PMBI, not used by the PP), and the memory access request type bits (specify read, write, or execute). In addition, a busy (B) bit is set whenever a status file entry is made. When a memory access request occurs, the next VP code and the PMRC and PMWC lines combine to develop a pointer to the status file entry corresponding to the VP that made the request. Using the developed status file pointer, the entry data is inserted into the status file and the associated busy bit is set. During the succeeding execution clock periods, the decoded next VP code is used to select the busy bit of the active VP. The result is input to the PCCTL card, where it is used to develop the buffer available (BA) signal. If the busy bit is set ($\overline{BA}$ is true), the status file is not available for new data so an additional memory access request cannot be made by the active VP. This checking of the BA signal is necessary because only one entry in the status file is allotted to each VP. When the asynchronous logic is processing an entry in one of the synchronous logic priority queues, the VP number of the queue entry is routed back to the synchronous logic, where it is decoded and used as a pointer to an entry in the status file. The selected entry data (⌐PMSFO(0-6)) is input to the asynchronous logic. When a read request is being processed, the number of the VP making the request (PMRQO(0-2)) is input to the synchronous logic from the asynchronous logic. The VP number is decoded and used as a pointer in selecting the proper zone bits (three LSB's from the PC) from the status file. The zone data (PMRCMB(0-2)) is input to the PCCARDA(0-7) cards, where it is used to enable the reading of one of eight memory zones via the two way bus (TWB). When a memory access request has been processed, the PMWCLR(0-7) signals from the asynchronous logic are used to reset the busy bit in the status file entry corresponding to the honored request.

4-147 ASYNCHRONOUS LOGIC. Refer to figure 4-71 for a detailed block diagram of the asynchronous logic portion of the single word buffer controller. The asynchronous logic accepts memory access request data (VP codes, priority queue output counter results, RP bits, and status file data) from the synchronous logic, indicates to the Memory Control Unit (MCU) when a memory access request is being made by the PP, and controls the necessary transfer of data and address information in order to complete the requested

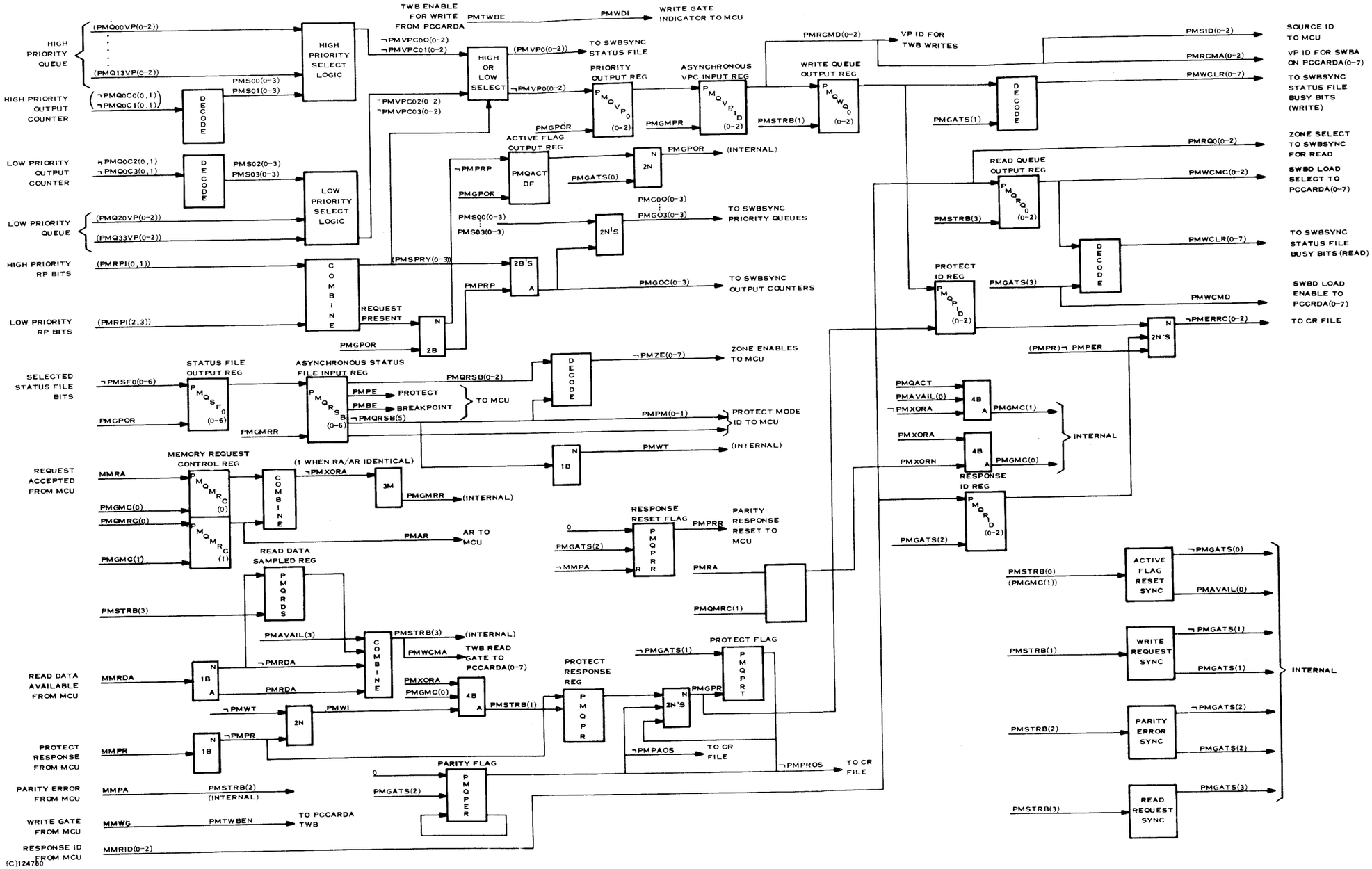


Figure 4-71. Single Word Buffer Controller Asynchronous Logic

memory access process (read, write, or execute). In short, the asynchronous logic provides the required interface between the PP and the Central Memory MCU. Refer to figure 4-69 for the registers used by the asynchronous logic.

4-148 General Request Processing. The eight high and low priority queue entries from the synchronous logic are applied to separate blocks of select logic where the respective output counters are used to select two of the eight entries from each of the queues. Two entries are selected from each of the queues because the first two bits of each output counter apply only to the first four queue entries of the respective queues and the last two bits of each output counter apply only to the last four queue entries of the respective queues. For this reason, entries from the first and last half of each queue are applied to the high or low priority select logic. The high and low priority RP bits from the synchronous logic are input to a combinational logic network that develops the set of signals (PMSPRY(0-3)) used in selecting one of the four queue entries applied to the high or low priority select logic. The combinational logic network develops the PMSPRY(0-3) signals such that the high priority queue entries are processed before any low priority queue entry. The VP code of the queue entry enabled through the high or low priority select logic is applied to the priority output register (PMQVP0(0-2)) in complement form and to the synchronous logic in true form. The synchronous logic uses the enabled VP code to select one of the eight status file entries for use in the asynchronous logic. In the asynchronous logic, the selected status file entry is applied to the status file output register (PMQSF0(0-6)). In addition to the PMSPRY(0-3) signals, the combinational logic network develops a request present signal that indicates when any one of the priority queue RP bits is set.

When the asynchronous logic is ready to accept another memory access request (PMGPOR is one and PMQMRC(0) (request accepted) equals PMQMRC(1) (access request)), the active flag output register (PMQACT) is set, the selected VP code is gated into the priority output register (PMQVP0(0-2)), the selected status file data (¬PMSF0(0-6)) is gated into the status file output register (PMQSF0(0-6)), the appropriate synchronous logic output counter is incremented by one of the PMG0C(0-3) signals, and the RP bit of the current request is reset by one of the PMG00, PMG01, PMG02, or PMG03 signals. In addition, the access request (AR) register (PMQMRC(1)) is toggled to indicate to the MCU that the PP is making a memory access request. The active flag reset synchronizer is strobbed via PMSTRB(0) to determine if additional memory request entries exist in the priority queues (refer to figure 4-72 for a timing diagram of the active flag reset synchronizer operation). The PMGMRR gate, developed when the previous request

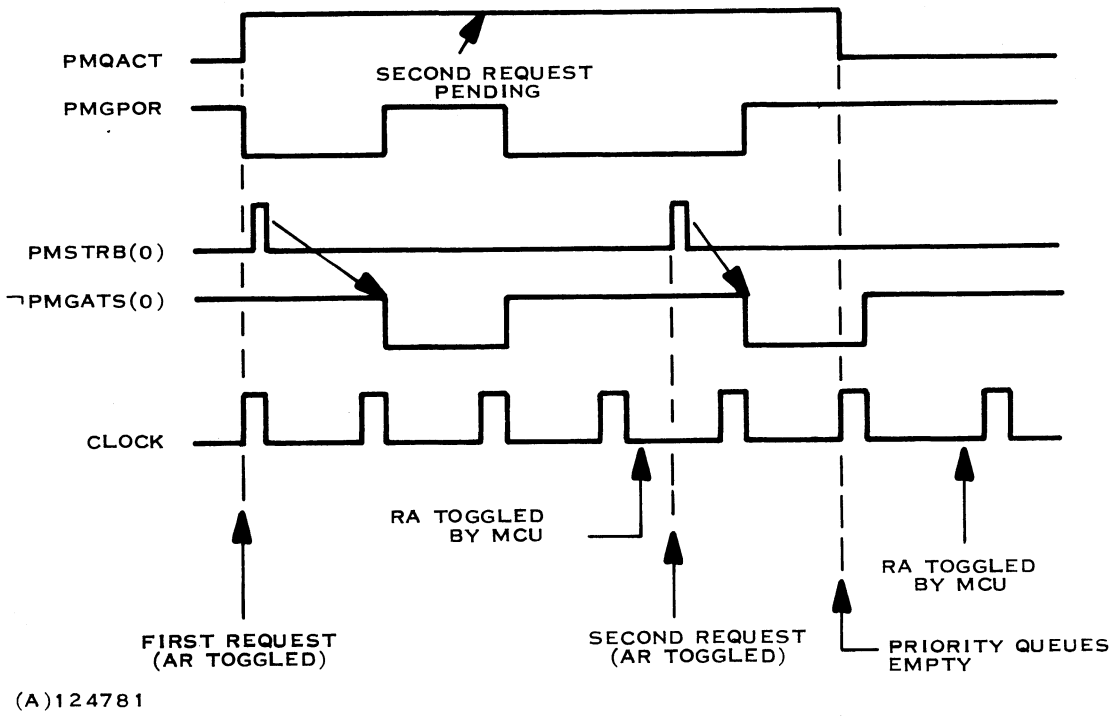


Figure 4-72. Active Flag Reset Synchronizer

has been handled, is used to enable the VP code in the priority output register into the asynchronous VP code input register (PMQVPID(0-2)) and the status file entry in the status file output register into the asynchronous status file input register (PMQRSB(0-6)). The VP code gated into the asynchronous VP code input register is used by the MCU to indicate the source of the request and by the PCCARDS(0-7) cards as a TWB write selector (PMRCMD(0-2)) and a singleword buffer address (SWBA) register selector (PMRCMA(0-2)). On the SWBASY card, the same VP code is applied to the write queue output register (PMQWQ0(0-2)). The status file entry gated into the asynchronous status file input register is used by the MCU and the SWBASY card. When a write request is made, the three LSB's from the PC (¬PMQRSB(0-2)) are decoded to eight lines (¬PMZE(0-7)) and input to the MCU where they are used to enable one of eight zones in Central Memory. When a read request is made, bit 5 of the status file entry disables the zone enable transfer. The protect enable (PMPE) and breakpoint enable (PMBE) signals (the breakpoint enable signal is not used by the MCU when the PP is requesting access) are direct inputs to the MCU. Bits 5 and 6 of the status file entry (indicate read, write, or execute) are input to the MCU as the protect mode ID (PMPM(0-1)). This two-bit code is used by the MCU to define the protected memory segment when the protect enable (PE) bit is set. The complement of bit 5 (¬PMQRSB(5)) is used by the SWBASY card to determine whether a read or write access is being made.

4-149 Write Request Processing. When bit 5 from the status file entry indicates a write request is being made, and the MCU responds by toggling the request accepted signal (MMRA), the request accepted (RA) flag (PMQMRC(0)) is toggled and the PMSTRB(1) strobe is developed. The PMSTRB(1) strobe is used to gate the VP code in the asynchronous VP code input register into the write queue output register (PMQWQ0(0-2)) and to initiate the write request synchronizer (refer to figure 4-73). The MCU reacts to the PP write request by developing a write gate (MMWG) for the asynchronous logic. The asynchronous logic uses this write gate to generate a TWB enable signal (PMTWBEN) for use on the PCCARDA(0-&) cards. The PMTWBEN signal and the previously mentioned TWB write selector signals (PMRCMD(0-2) combine to provide the control necessary to write the contents of the selected SWBD to Central Memory (the address to be written to in the enabled zone is supplied by the SWBA selected by the previously mentioned PMRCMA(0-2) signals). The TWB enable signal on the PCCARDA(0-7) cards (PMTWBE) is routed back to the asynchronous logic, where it is used to develop the write gate indicator response (PMWDI) for the MCU. When the PMGATS(1) signal from the write request synchronizer responds to the PMSTRB(1) strobe, the VP code in the write queue output register is decoded (PMWCLR(0-7)) and input to the synchronous logic. The synchronous logic uses the PMWCLR(0-7) signals to reset the busy bit in the status file entry that results in the described write request.

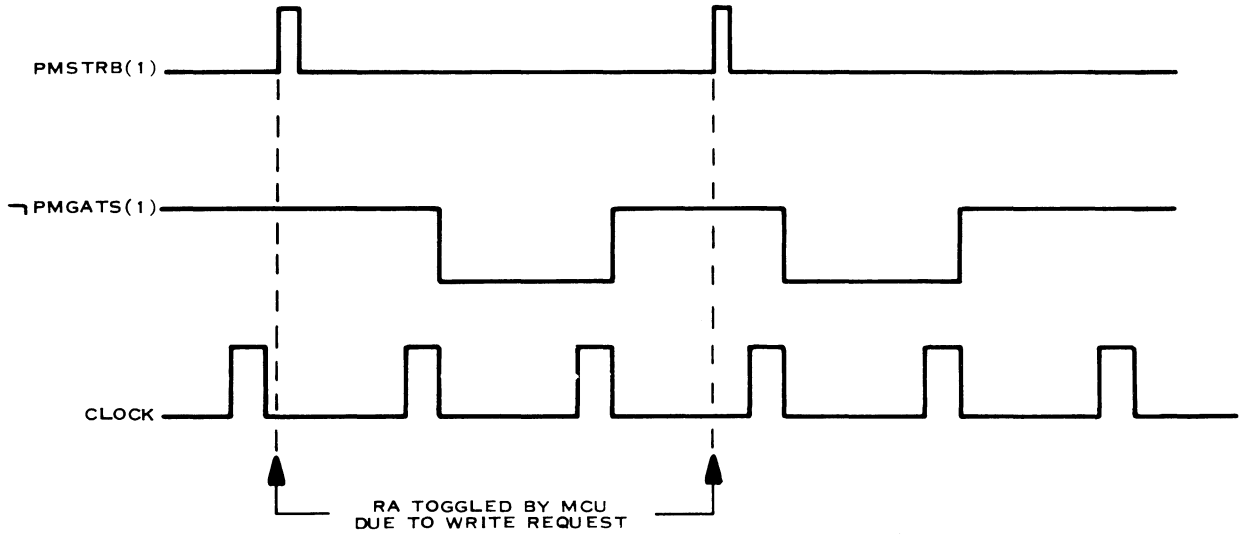
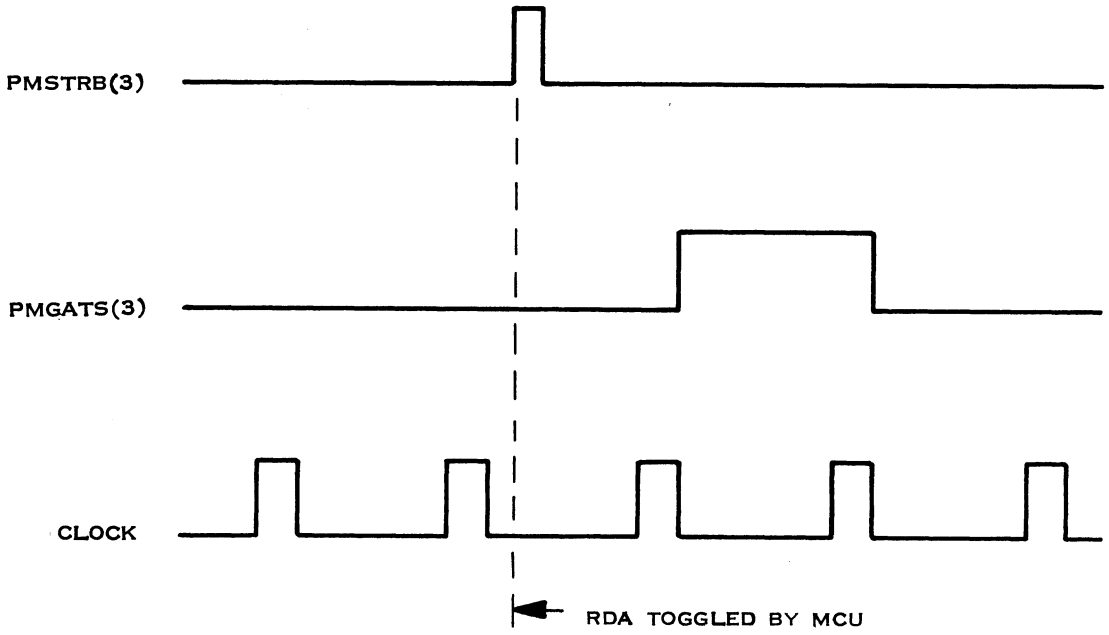


Figure 4-73. Write Request Synchronizer

When the write request requires access to a protected area of the Central Memory (the protected area is defined by the protect mode ID only when the protect indicator is set by the PP), the MCU responds with a protect response signal (MMPR) to the asynchronous logic (MMPR is only valid when AR equals RA). The MMPR and PMSTRB(1) signals combine to set the protect response register (PMQPR) with the toggling of RA. The set protect response register is used to gate the VP code from the write queue output register into the protect ID register (PMQPID) and to set the protect flag (PMQPRT) when the synchronizer signal (¬PMGATS(1)) from the write request synchronizer occurs. The VP code from the protect ID register and the set protect flag are used in the time slot override reason byte and outstanding events byte of the CR file for interrupt purposes. The protect flag is reset at the next clock pulse and the protect response register is reset when the next request is made (MMPR goes to zero).

4-150 Read Request Processing. When bit 5 from the status file entry disables the zone enable transfer to the MCU to indicate that a read request is being made, and the MCU responds by toggling the request accepted signal (MMRA), the asynchronous logic must wait for a read data available (RDA) signal from the MCU. At this time, the response ID from the MCU (MMRID, developed from the previously mentioned source ID signal) is applied to the read queue output register (PMQRQ0(0-2)) in the asynchronous logic and the zone select logic in the synchronous logic. The selected zone from the status file in the synchronous logic is used by the TWB on the PCCARDA(0-7) cards to enable one of the eight Central Memory zones during the reading process. When the MCU toggles the RDA bit via the MMRDA signal to signify the start of the reading process, the PMSTRB(3) strobe is developed and used to toggle the read data sampled (RDS) register (PMQRDS). In addition, the PMSTRB(3) strobe initializes the read request synchronizer operation (refer to figure 4-74 for a timing diagram), supplies the TWB read gate (PMWCMA) to the PCCARDA(0-7) cards, and gates the response ID into the read queue output register. The response ID gated in the read queue output register is used on the PCCARDA(0-7) cards to select the SWBD used in the read operation (the PMWCMC(0-2) signals). When the read request synchronizer responds with the PMGATS(3) signal, the response ID in the read queue output register is decoded (PMWCLR(0-7)) and input to the synchronous logic where it is used to reset the busy bit in the status file entry that caused the described read request. The PMGATS(3) signal is also used on the PCCARDA(0-7) cards to enable the loading of the data read into the selected SWBD (the PMWCMD signal).

When the read request requires access to a protected area of Central Memory, the asynchronous logic processes the protect response from the MCU as described for a write request. When the read request results in a parity error, the MCU responds with a parity error signal (MMPA) to the asynchronous logic. The parity error signal is used to strobe (PMSTRB(2)) the



(A)124784

Figure 4-74. Read Request Synchronizer

parity error synchronizer. Refer to figure 4-75 for a timing diagram of the parity error synchronizer. When the parity error synchronizer responds with the PMGATS(2) and ¬PMGATS(2) signals, the parity flag (PMQPER) is set, the response reset flag (PMQPRR) is set, and the response ID is gated into the response ID register (PMQRID). The parity flag and the VP code from the response ID register are used in the time slot override reason byte and outstanding events byte of the CR file for interrupt purposes. The response reset flag indicates to the MCU that the asynchronous logic has received the parity error signal. The parity flag is reset at the next clock pulse and the response reset flag is reset when the parity error signal goes to zero.

4-151 TWO WAY BUS. Refer to figure 4-76 for a simplified block diagram of the TWB (the MAMB bus is also included because the singleword buffer controller provides the control for MAMB). For write operations, the TWB accepts data from the SWBD of the VP executing the write and inputs that same data to the eight zones of Central Memory. For read operations, the TWB accepts data from the eight zones of Central Memory and inputs data from the selected zone to the SWBD of the VP executing the read. For both write and read operations, the MAMB supplies Central Memory with the address from the SWBA of the VP executing the write or read.

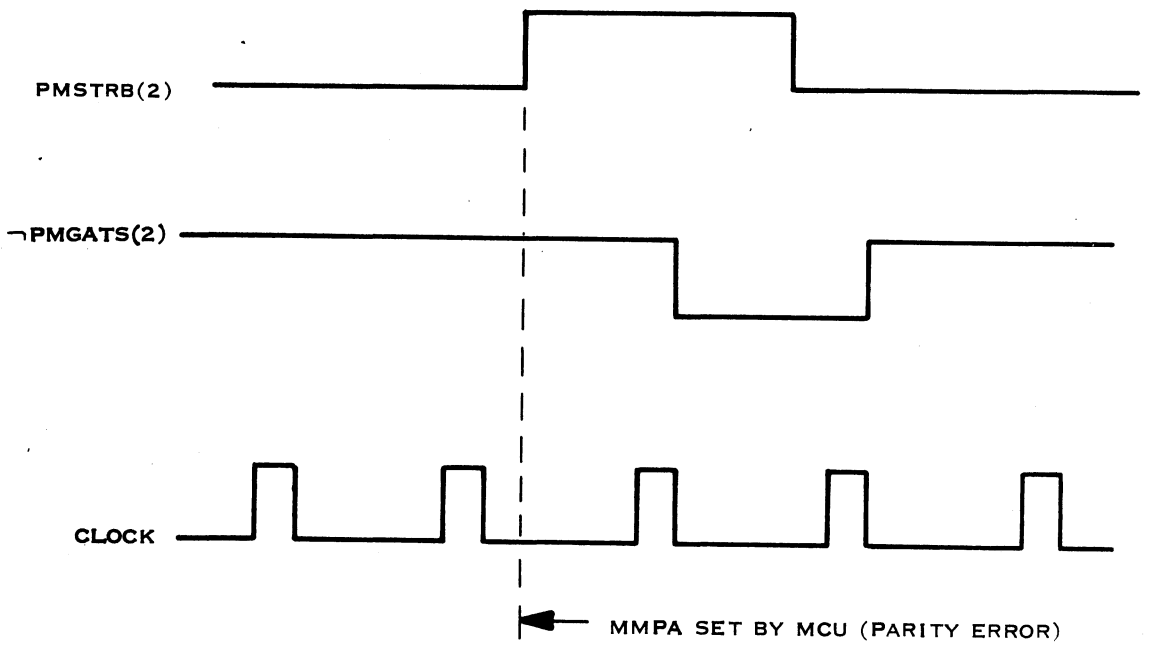


Figure 4-75. Parity Error Synchronizer

4-152 Write Operations. When a write request has been accepted by the MCU, the three SWBA select lines (PMRCMA(0-2)) from the asynchronous logic are decoded to eight lines (PMACMMD(00-07)) and used to enable the SWBA of the VP executing the write through to Central Memory. The TWB write selector (PMRCMD(0-2)) from the asynchronous logic is decoded and used to enable the SWBD of the VP executing the write through the TWB write selection logic. The selected SWBD is expanded to eight identical words and enabled through a group of 2N logic gates to Central Memory via the TWB write enable signal (PMTWBEN).

4-153 Read Operations. When a read request has been accepted by the MCU, the SWBA select lines from the asynchronous logic are decoded and used to enable the SWBA of the VP executing the read through to Central Memory (same process as for write operations). The TWB zone select lines (PMRCMB(0-2)) from the synchronous logic are decoded and used to enable one of the eight sets of word lines from Central Memory in the TWB read select logic. The TWB write enable signal is disabled to prevent a write operation and the selected word from Central Memory is gated into a group of flip-flops (via the PMWCMA signal from the asynchronous logic) when the MCU toggles the RDA bit. When the read request synchronizer on SWBASY responds to the RDA toggle, the SWBD load enable signal (PMWCMD) permits

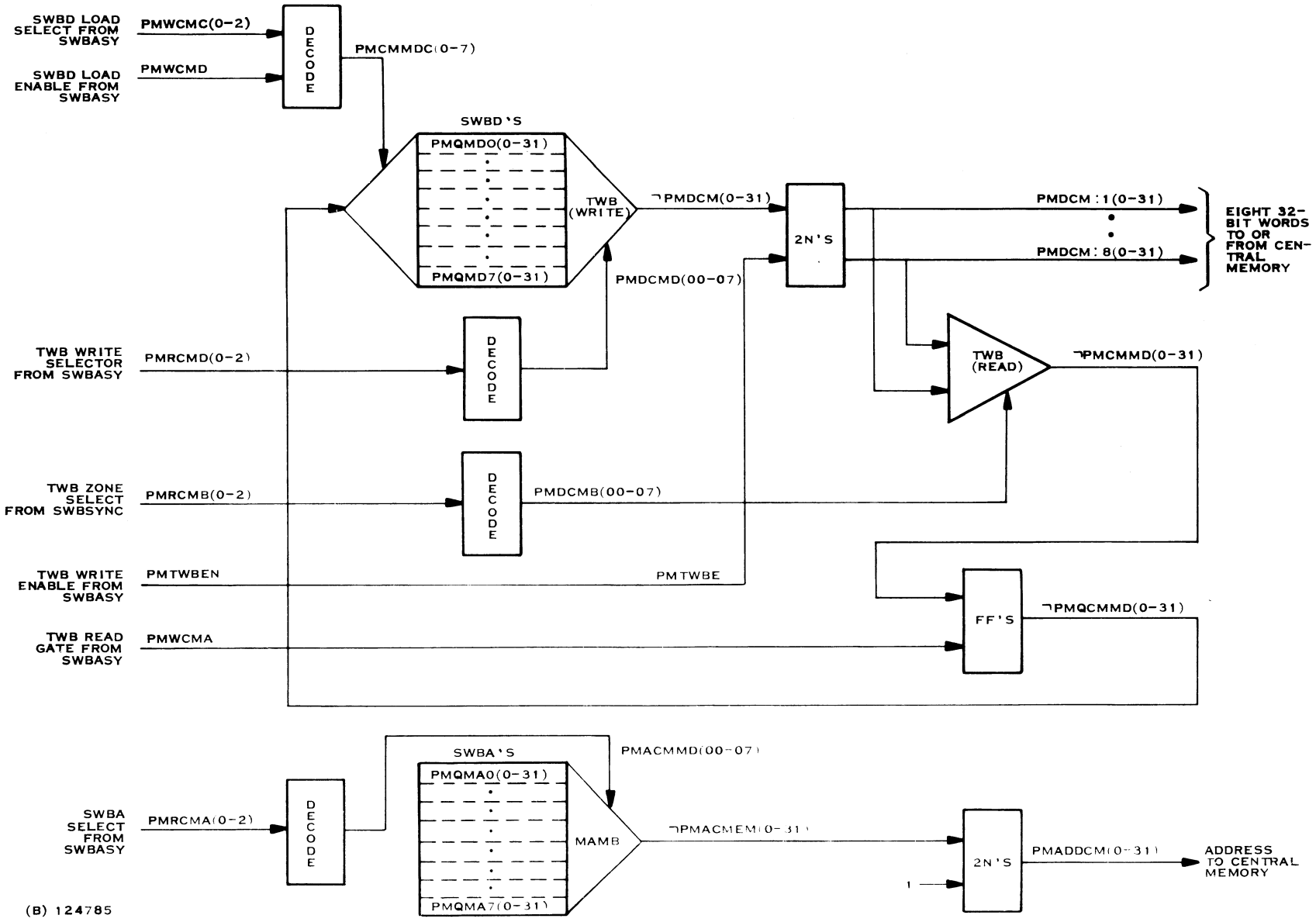


Figure 4-76. Single Word Buffer Controller TWB and MAMB Buses

the decoding of the SWBD load select signals (PMWCMC(0-2)) so that the selected word from Central Memory is inserted in the SWBD of the VP executing the read.

4-154 PERIPHERAL PROCESSOR CONTROL INTRODUCTION

The heart of PP control is the Main Instruction Register (MIR), but is physically situated on the VPRCONT, PCCTL, PPCTL1, PPCTL2, CONTAU, CRMIRLDR, CRCONT(0-3), and IRCARD(0-3) cards. The interrelationships between these control cards and the cards whose responses are directly involved in the control (SWBSYNC, PPAUCDM(0-3), CRCELLY, INDEXER(0, 1), and PCCARDS(0-7)) are shown in figure 4-76. A description of the major control lines is given in the following paragraphs and a more detailed discussion of instruction execution, supplemented with detailed block diagrams and transfer tables, follows the block diagram discussion.

4-155 PERIPHERAL PROCESSOR CONTROL. Refer to figures 4-77 and 4-16 (Instruction Register format) for aid in following this description. One PP clock prior to the command execution time on which this discussion is based, the next VP code (VPC) lines from the MLCTL card enable the contents of the next Instruction Register (IR) over the IRB bus to the IRMIR and the VPRCONT and PPCTL1 cards. The PPCTL1 card uses the next IR data to modify the five LSB's of the new effective address (EA) if a store VP file (STF), load VP file (LDF), or unconditional branch to ROM (BRSM) instruction is to be executed during the command execution period. The PPCTL1 card also sets both interrupt (automatic and programmed) bits in order to enable the development of the 20_{16} CM address (if the BRSM instruction is in the appropriate state) and sets the interrupt trap bit if termination of the previous instruction caused the INTF bit to set. All of this data is applied to the IRMIR. The next IR data is also used by the VPRCONT card to generate the remainder of the MIR shown in the AUMIR, CRMIR, and VPRMIR formats in figure 4-78. This is done to minimize the delay in developing the controls necessary to execute the new instruction when the command execution clock occurs.

The VPRCONT card supplies the PCCTL card with the data bus enable and select lines shown in the PCCTL portion of the VPRMIR format of figure 4-78, the CONTAU card with the data shown in the AUMIR format of figure 4-78, and the CRMIRLDR card with the source, destination, and op-code groupings necessary to develop the CRMIR format of figure 4-78. When the clock signaling command execution time occurs, the next IR data from the IRB bus and the mentioned bits supplied by PPCTL1 are loaded into the IRMIR. At the same time, the VPRCONT card supplies the VPRCARD(0-7) cards with the data bus enable and select lines shown in the VPRCONT portion of the VPRMIR format of figure 4-78. The PCCTL card distributes the PCCTL portion of the VPRMIR (except for the EA and immediate (IM) to MDB

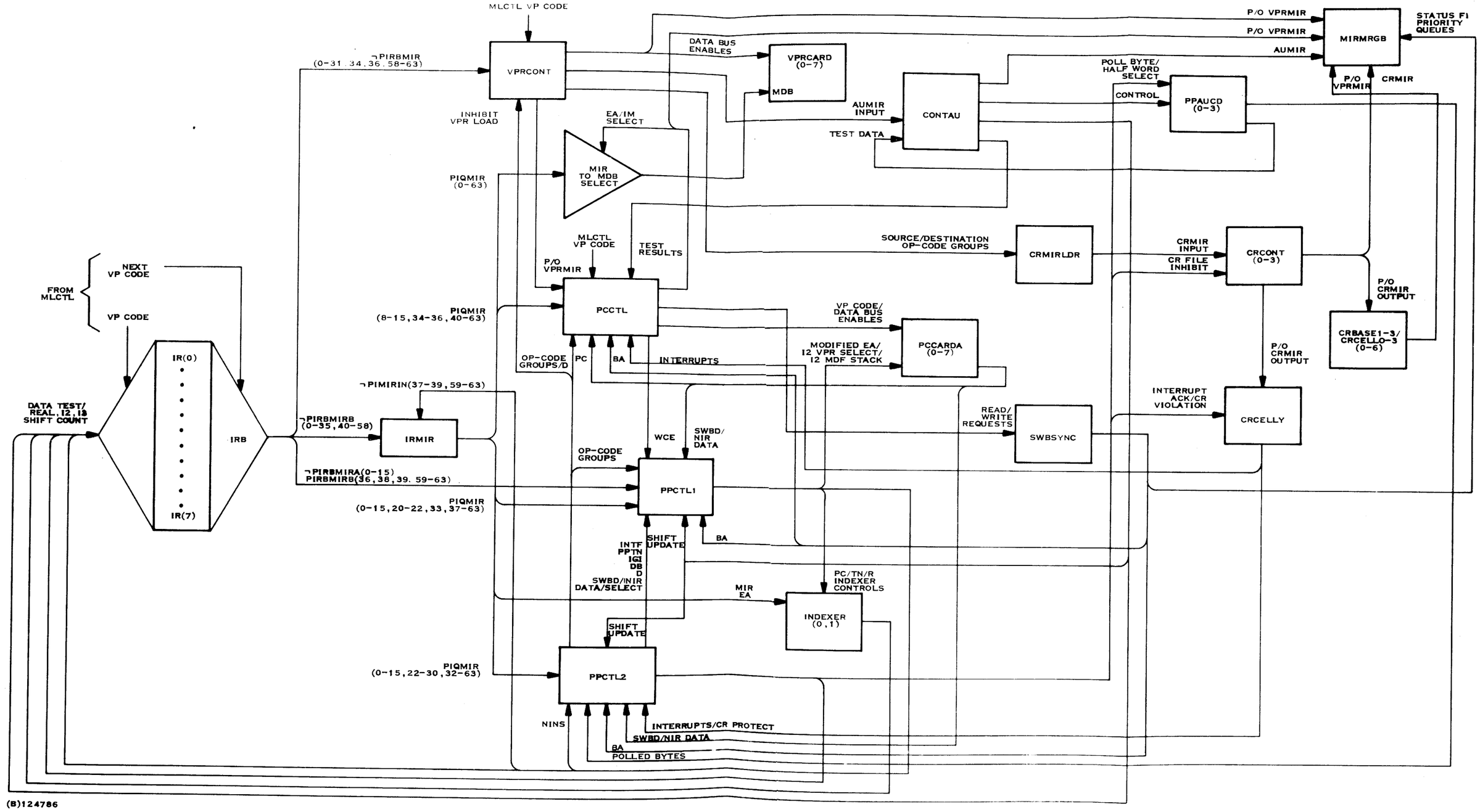


Figure 4-77. Peripheral Processor Control Block Diagram

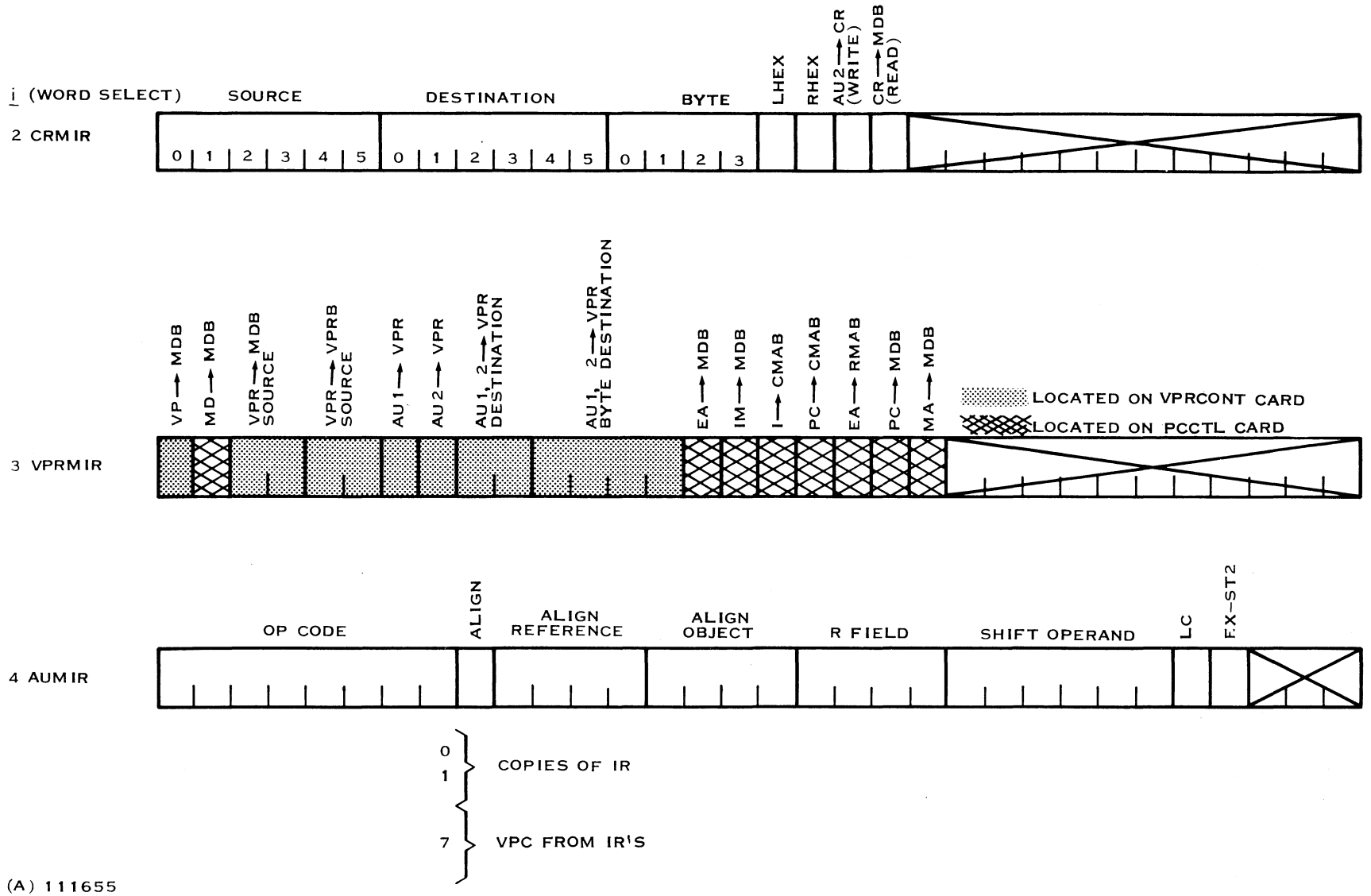


Figure 4-78. MIR Input Format

enables), the EA to CMAB and PC to RMAB enables, and the VPC from MLCTL to PCCARDA(0-7). The EA and IM to MDB enables are input to the MIR to MDB select logic on IRCARD(2, 3). The CONTAU card utilizes the AUMIR format to supply the PPAUCD(0-3) cards with the control lines necessary to execute the desired arithmetic operation, and the PPAUCD(0-3) cards feed test data back to CONTAU when a skip or branch decision needs to be made. The CRMIRLDR developed CRMIR format is input to the CRCONT(0-3) cards, which in turn, develop the CRMIR output format (shown in figure 4-79) necessary to control reading from or writing to the CR file. (The CR file consists of the CRCELLY, CRBASE(1-3), CRCELL0(0-6), CRCELL1(0-6), CRCELL2(0-6), and CRCELL3(0-6) cards.)

The loaded MIR data is distributed to the INDEXER(0, 1), PCCTL, PPCTL1, and PPCTL2 cards, in addition to the MIR to Main Data Bus (MDB) select logic on IRCARD(2, 3). The majority of the MIR and SWBD/NIR op-code decoding logic is contained on PPCTL2. Refer to figures 4-80 and 4-81 for a closer look at the op-code groups involved in the decoding, and to figures 4-82 through 4-104 for the Karnaugh map groupings of the same signals. In addition to being used on PPCTL2, the MIR op-code groups are input to PCCTL and PPCTL1 for further development of control lines. The PCCTL card uses MIR op-code groupings and the dependency indicator (D, the signal generated to indicate that the current instruction must complete execution before the SWBD/NIR instruction can be indexed) from PPCTL2; the MIR state class and step, DC bit, mode digit, LFAF bit, PPTN bit, and INTF bit from IRCARD(0, 2); the test results generated by CONTAU from the mentioned test data supplied by PPAUCDM(0-3); the interrupt lines from CRCELLY; the mode digit from the current PC value on PCCARDA(0); the write cycle equality indicator (WCE, the signal generated by PCCTL to indicate the EA of the current instruction is identical to the address of the next instruction to be executed) developed on PCCTL from the MIR EA and the current PC value on PCCARDA(2-7); the buffer available indicator (BA, the signal generated by the SWBSYNC card to reflect whether or not the status file entry for the executing VP is occupied) from SWBSYNC. The PCCTL card responds with data bus enables for PCCARDA(0-7) that are not included in the VPRMIR format (CMDB to NIR, PC load and mode control, CMAB to SWBA, ROM to CMDB, SWBD to CMDB, data manipulator to AU2B, and aligner to AU2B). In addition, PCCTL uses the same inputs to generate read and write requests for SWBSYNC and routes the WCE indicator to PPCTL1.

The PPCTL1 card uses MIR data (including MIR op-code groups from PPCTL2); SWBD/NIR data and select lines (including SWBD/NIR op-code groups from PPCTL2); the WCE indicator from PCCTL; the BA indicator from SWBSYNC; the shift update indicator (this signal is developed to reflect when additional data shift is necessary to complete the shift specified by a shift instruction) from CONTAU; the dependency (D), ignore indirect (IGI, the signal generated to indicate the next instruction is illegal for the indirect

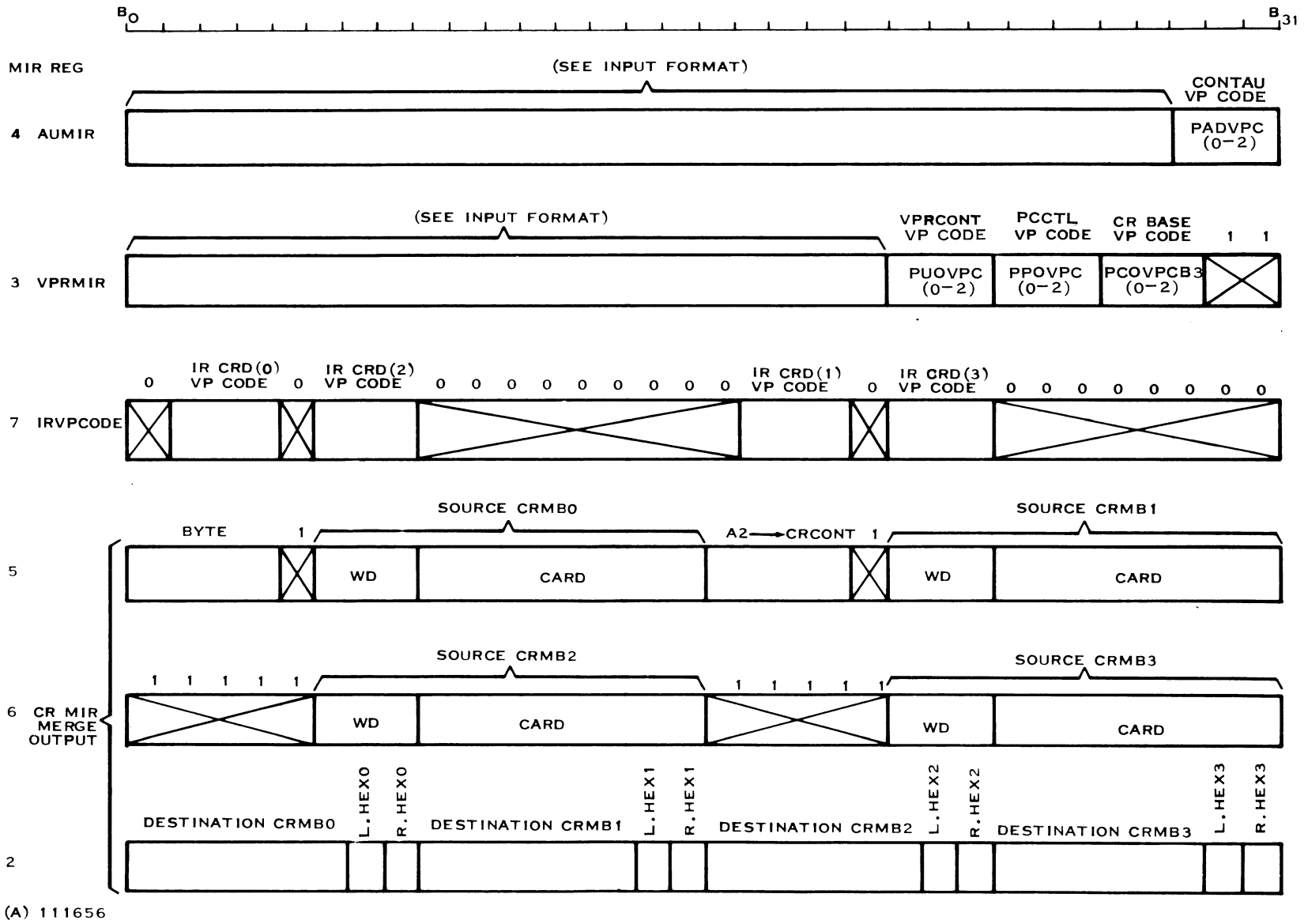


Figure 4-79. MIR Output Format

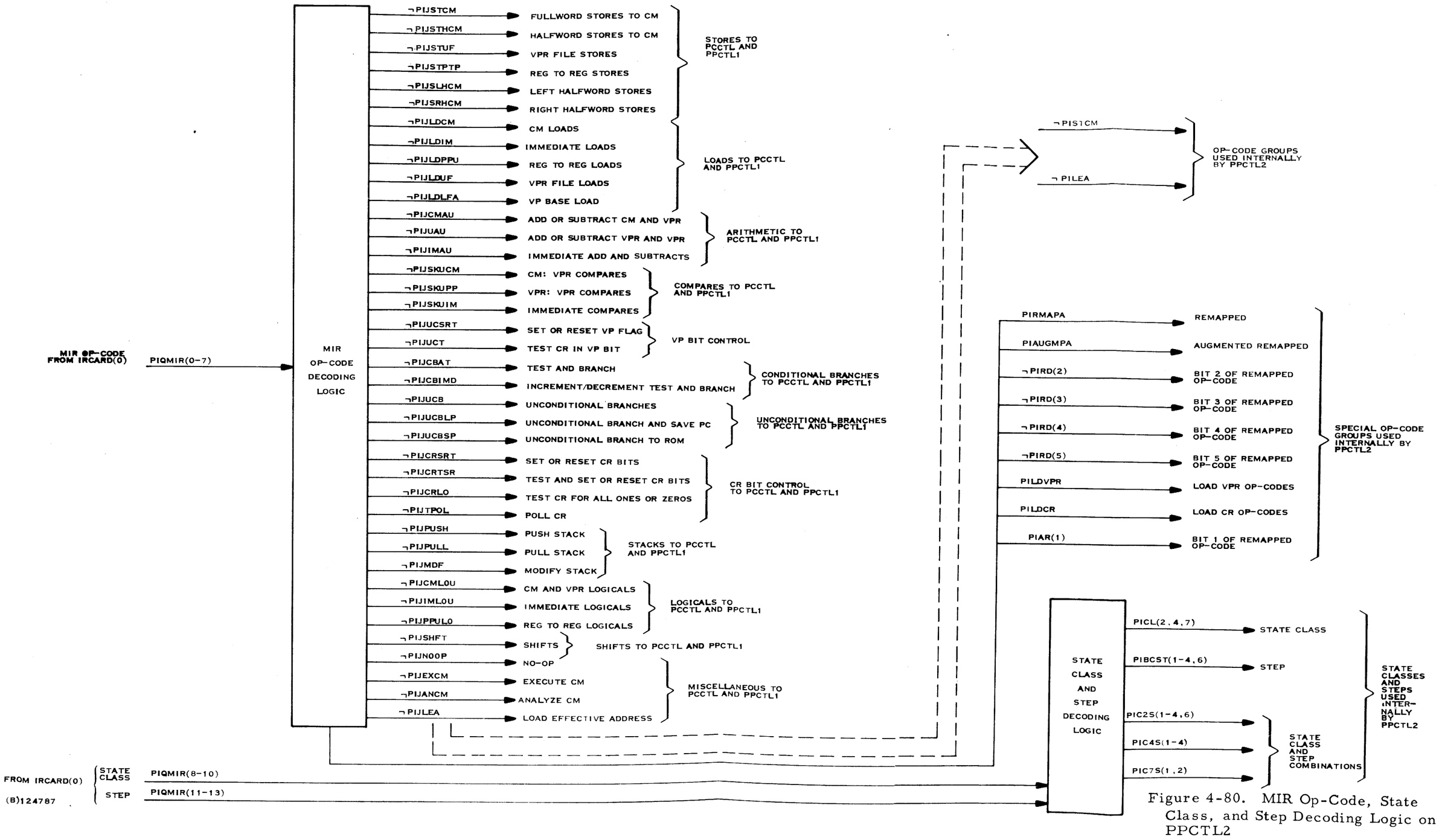


Figure 4-80. MIR Op-Code, State Class, and Step Decoding Logic on PPCTL2

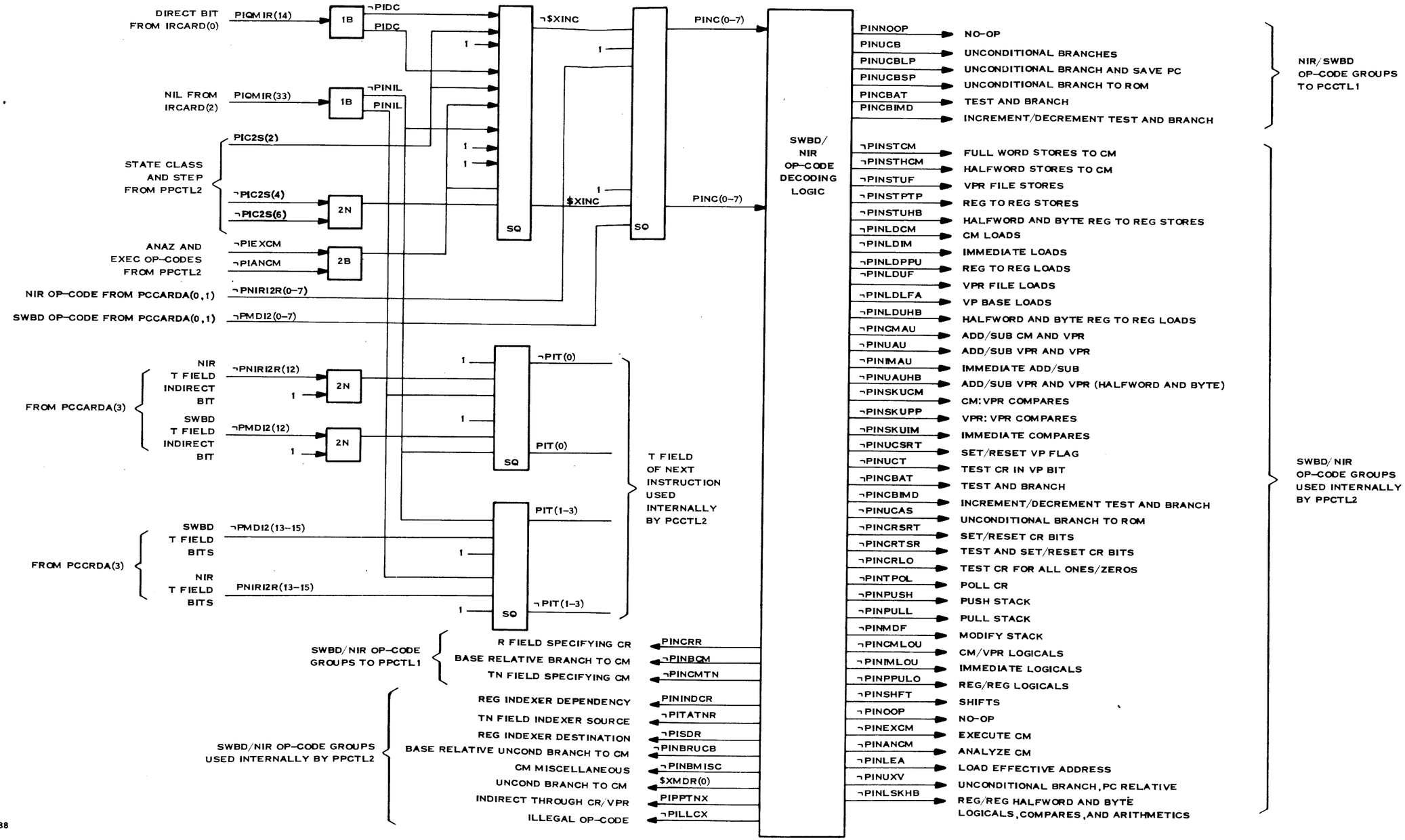
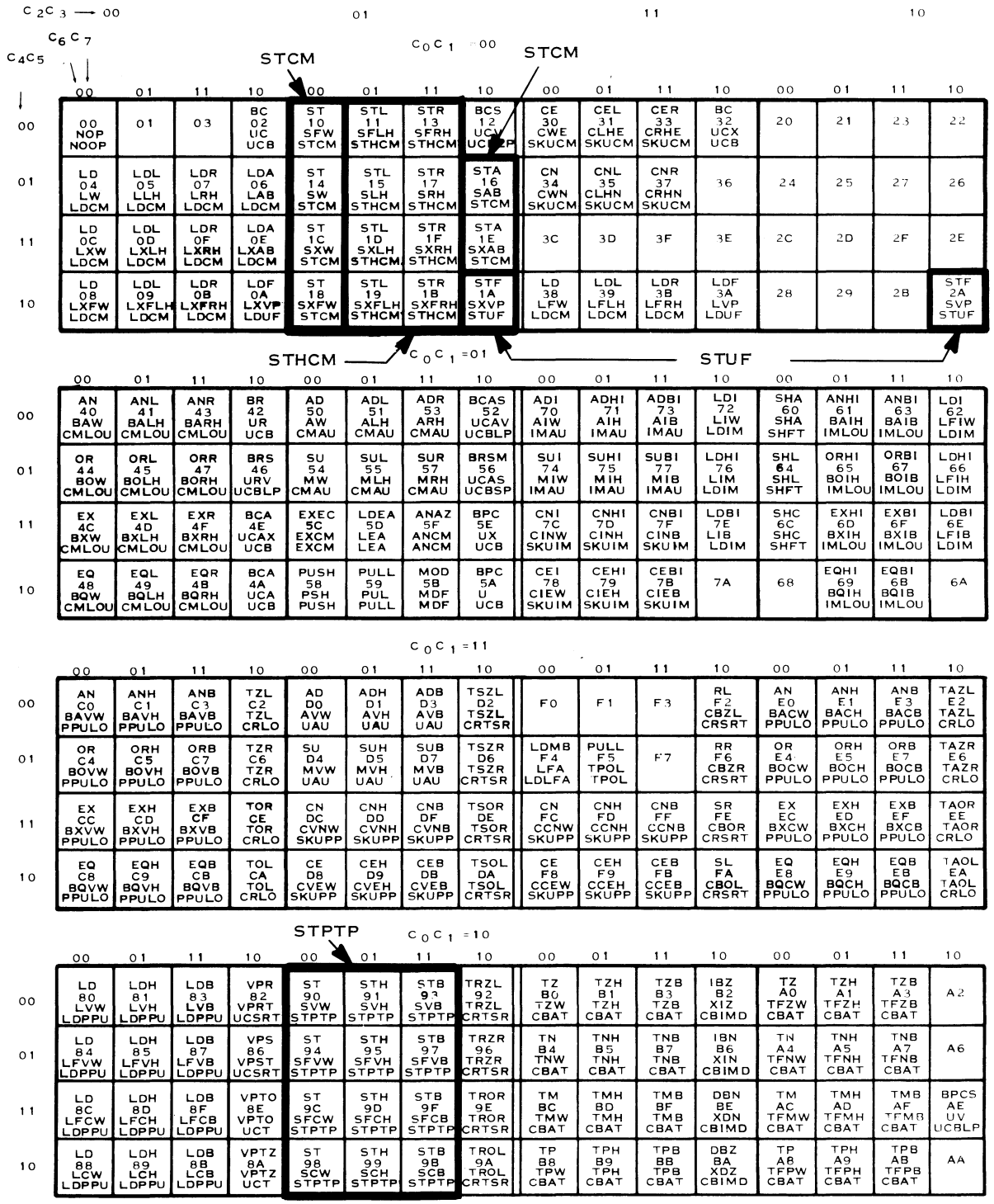
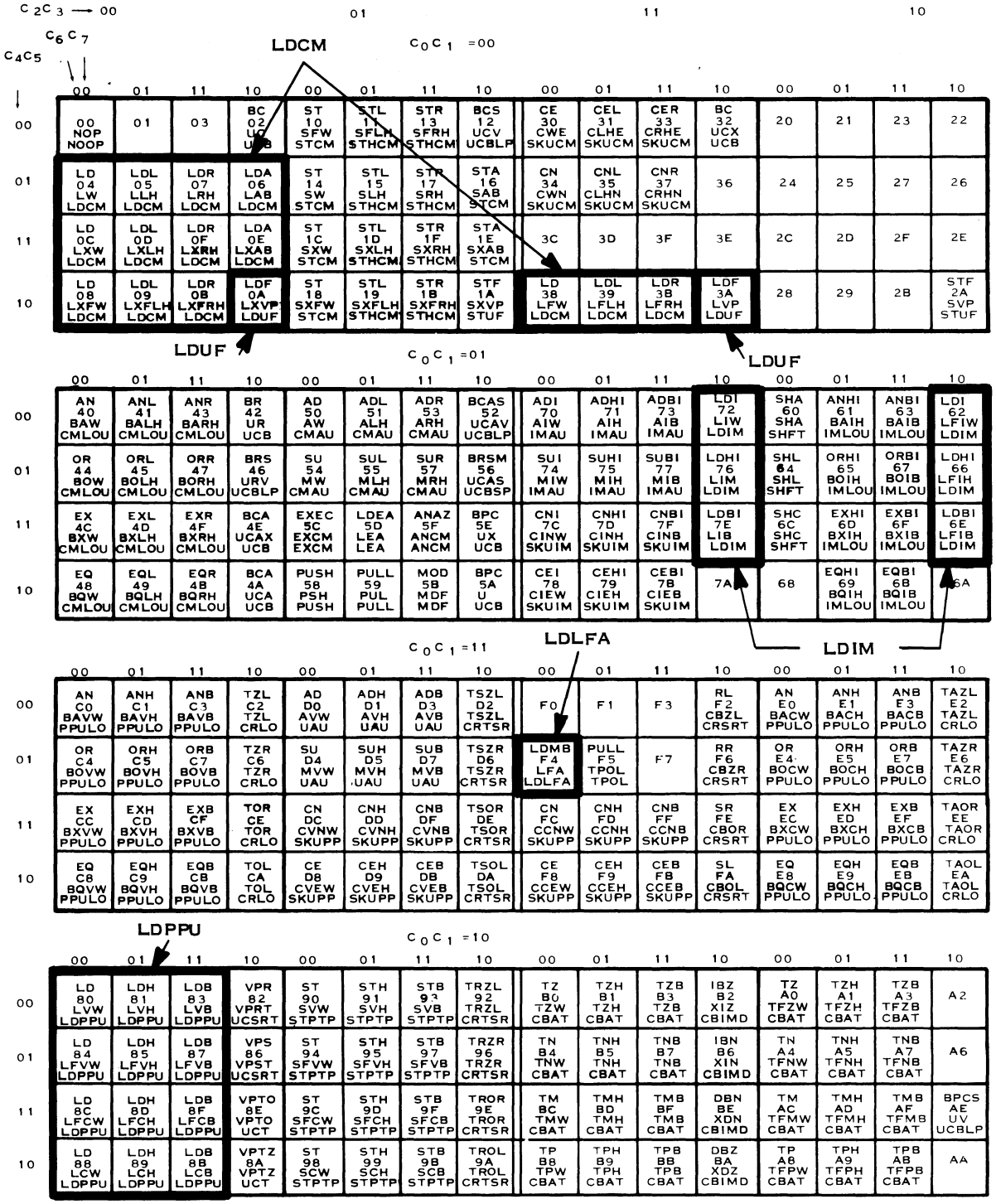


Figure 4-81. SWBD/NIR Op-Code and T Field Selection and Decoding Logic on PPCTL2



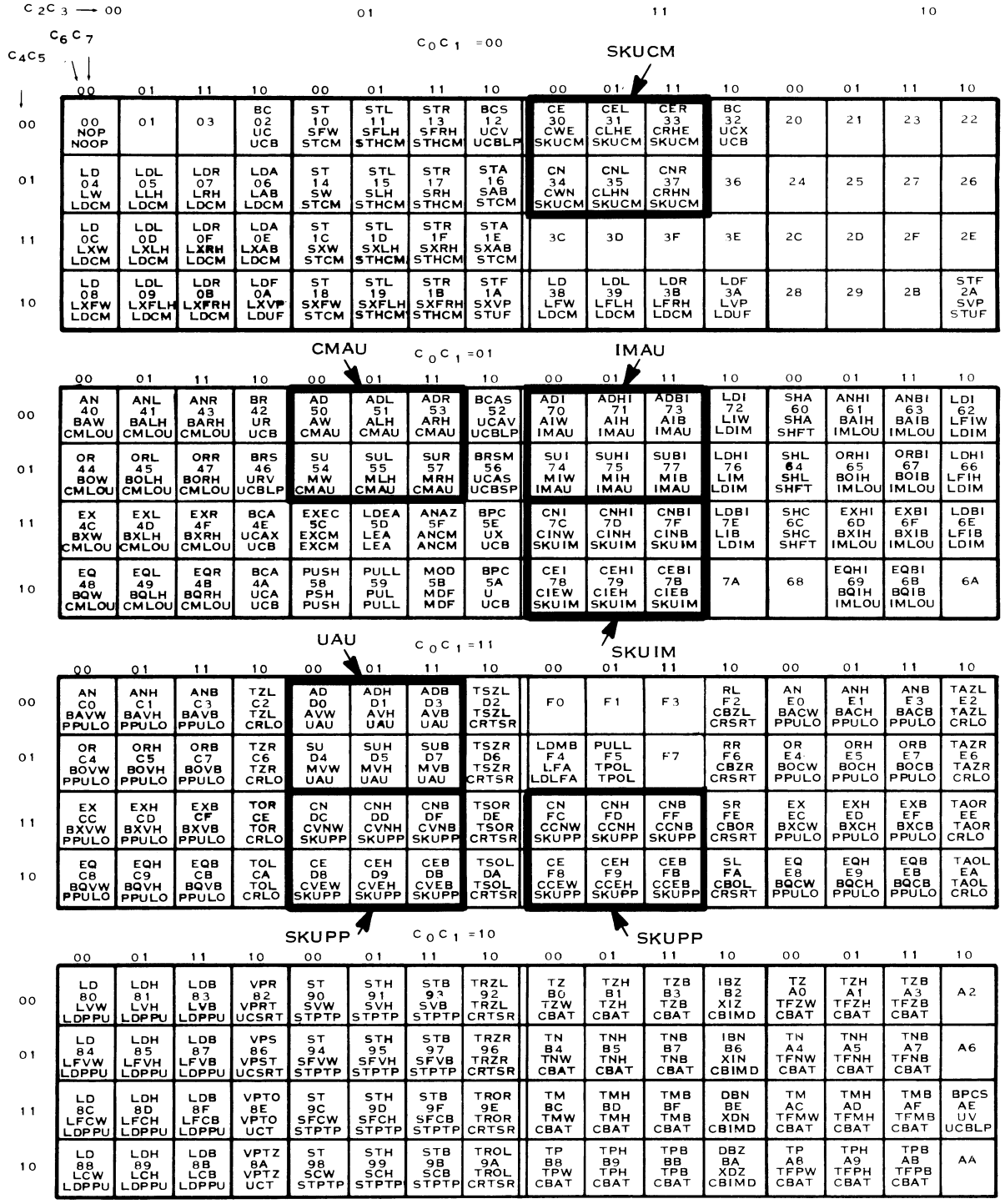
(A)124789

Figure 4-82. Store Op-Code Groups



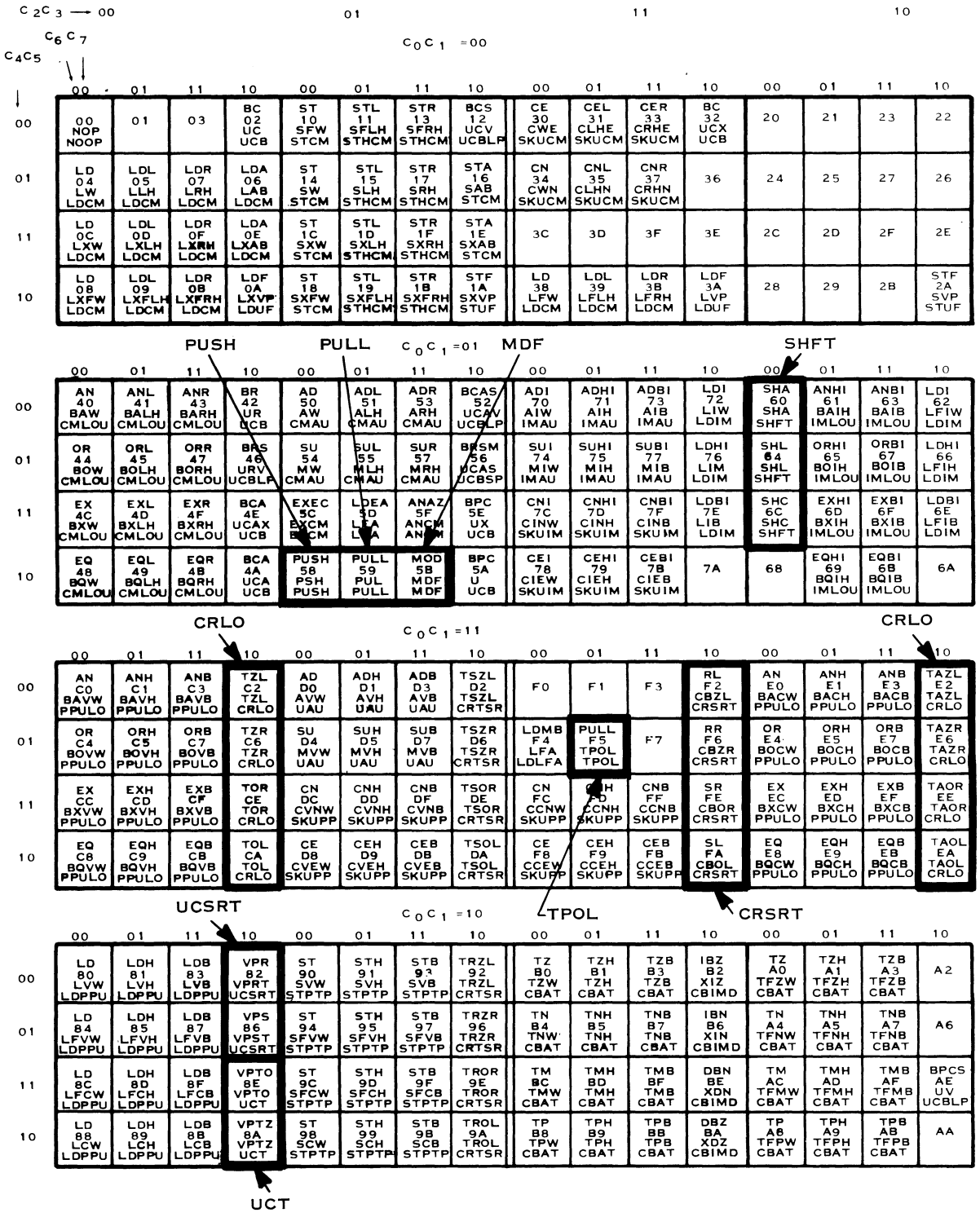
(A)124790

Figure 4-83. Load Op-Code Groups



(A)124791

Figure 4-84. Arithmetic and Compare Op-Code Groups



(A)124792

Figure 4-85. Shift, Stack, VP Bit Control, and CR Bit Control Op-Code Groups

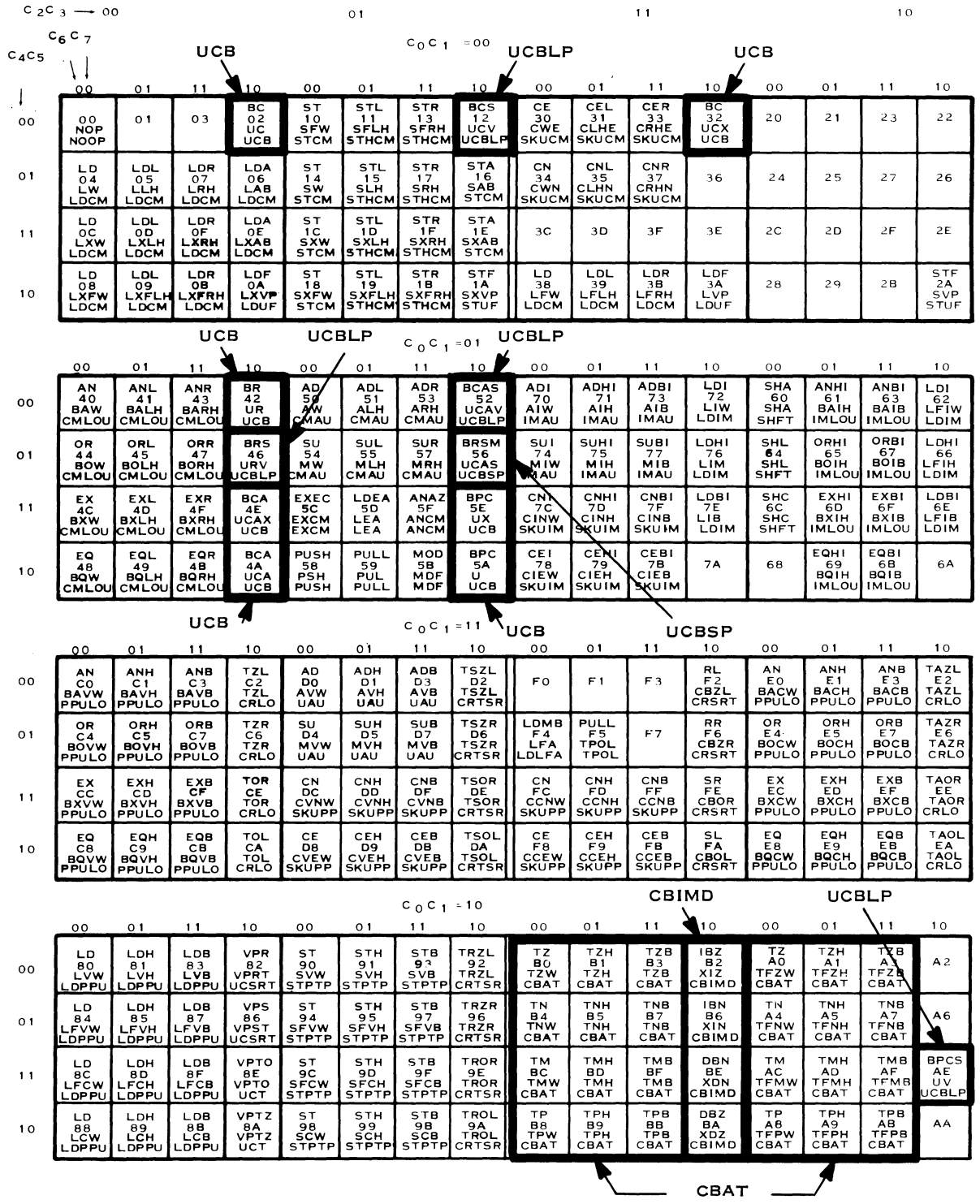
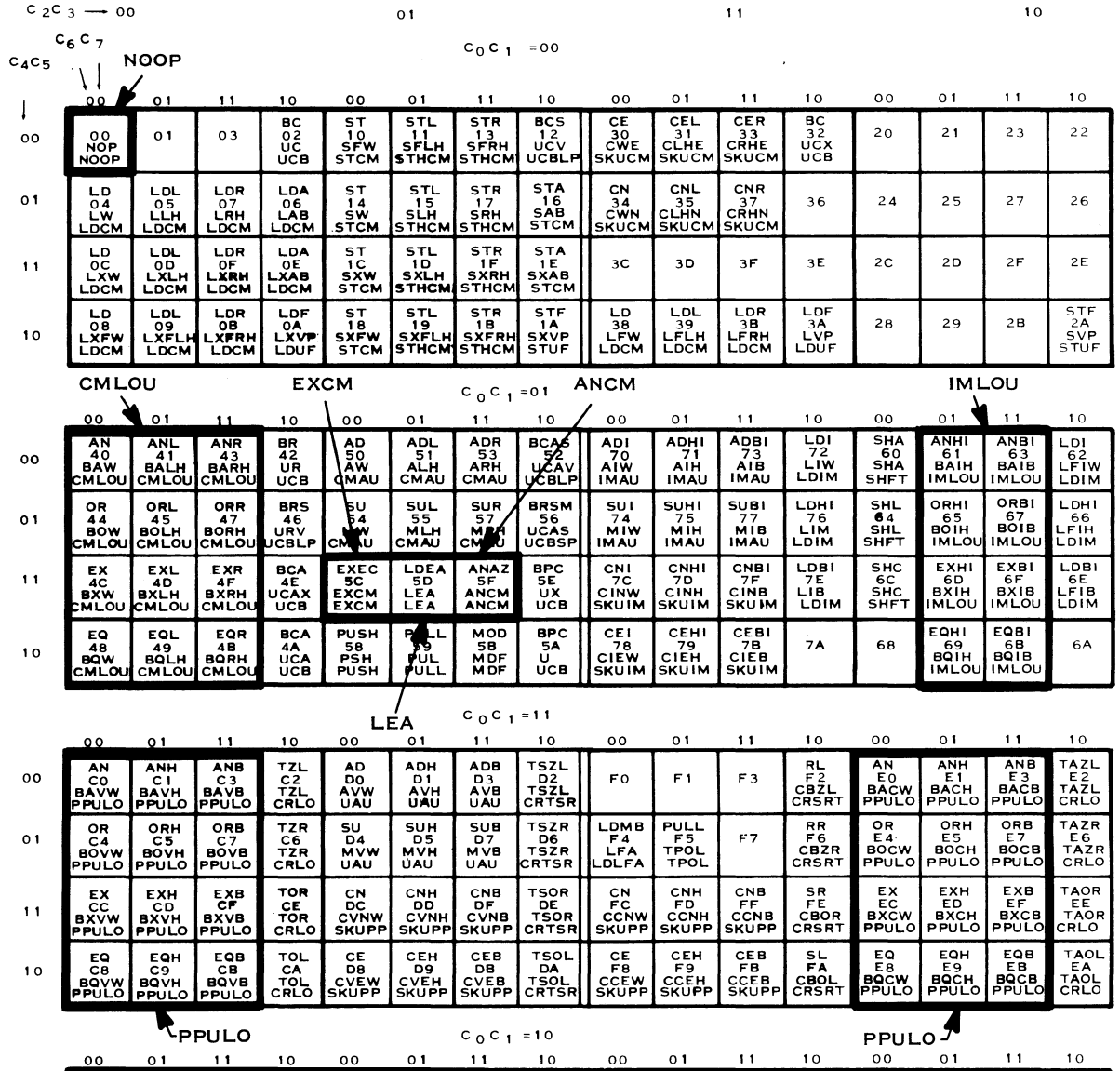
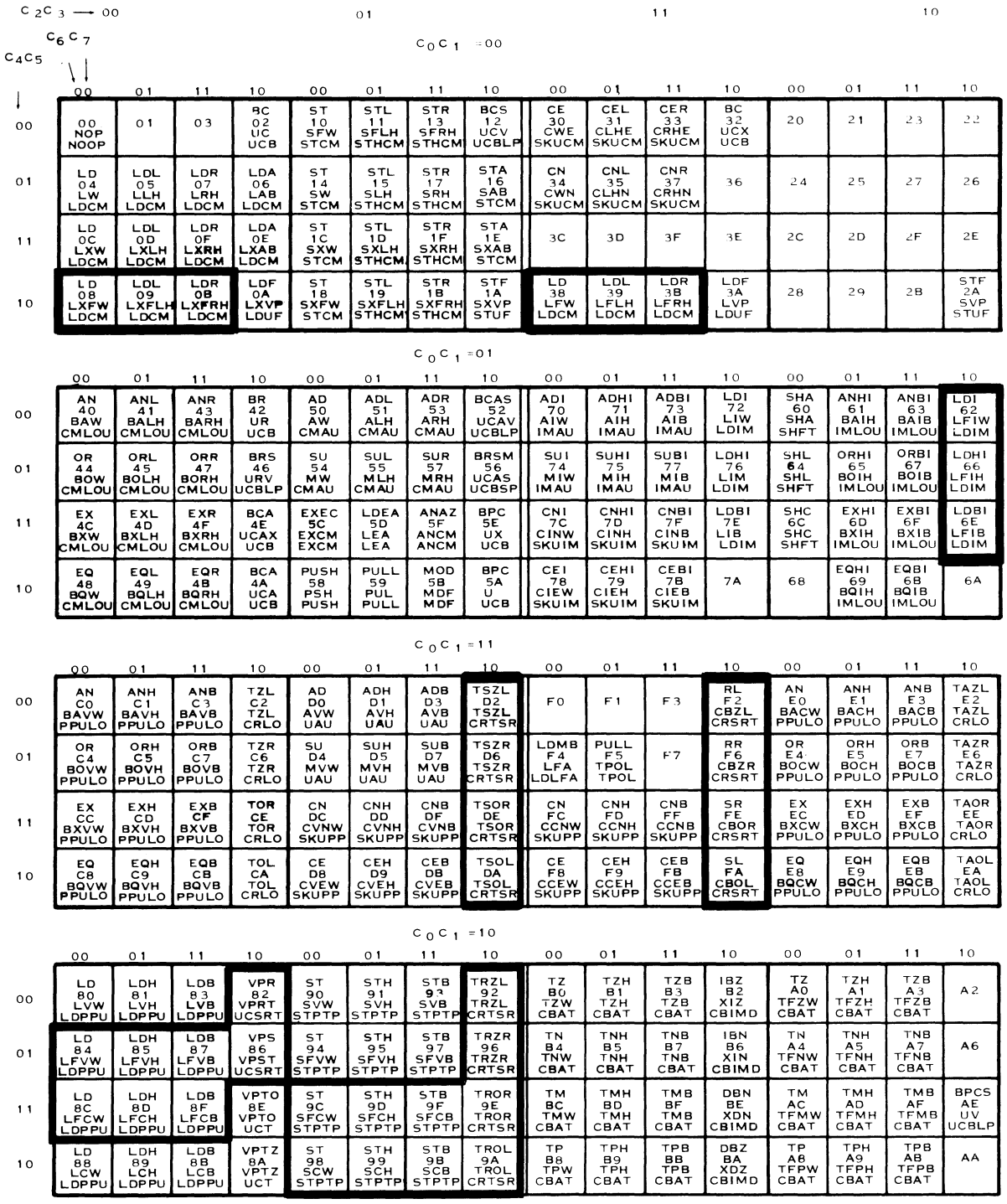


Figure 4-86. Conditional and Unconditional Branch Op-Code Groups



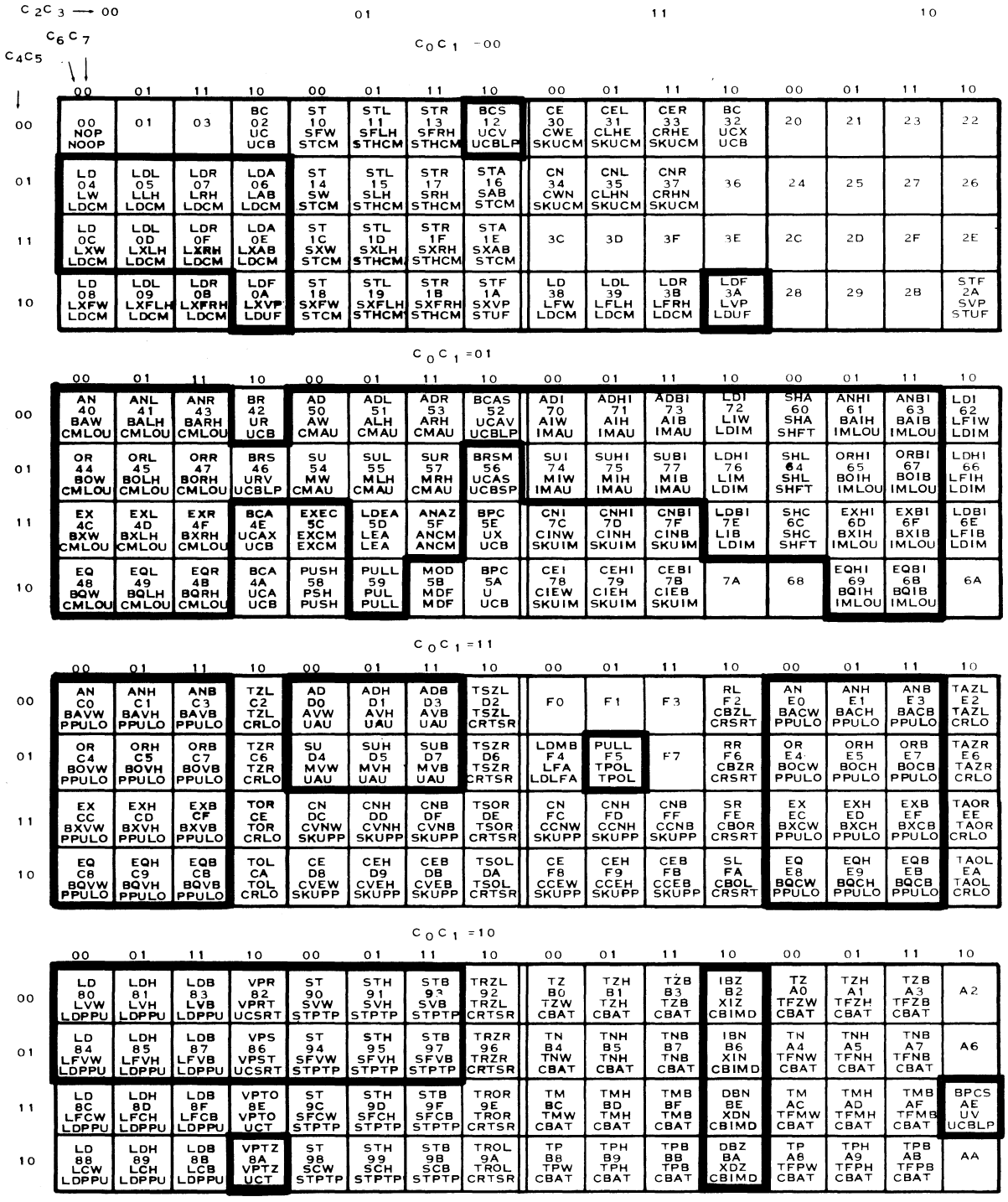
(A)124794

Figure 4-87. Logical and Miscellaneous Op-Code Groups



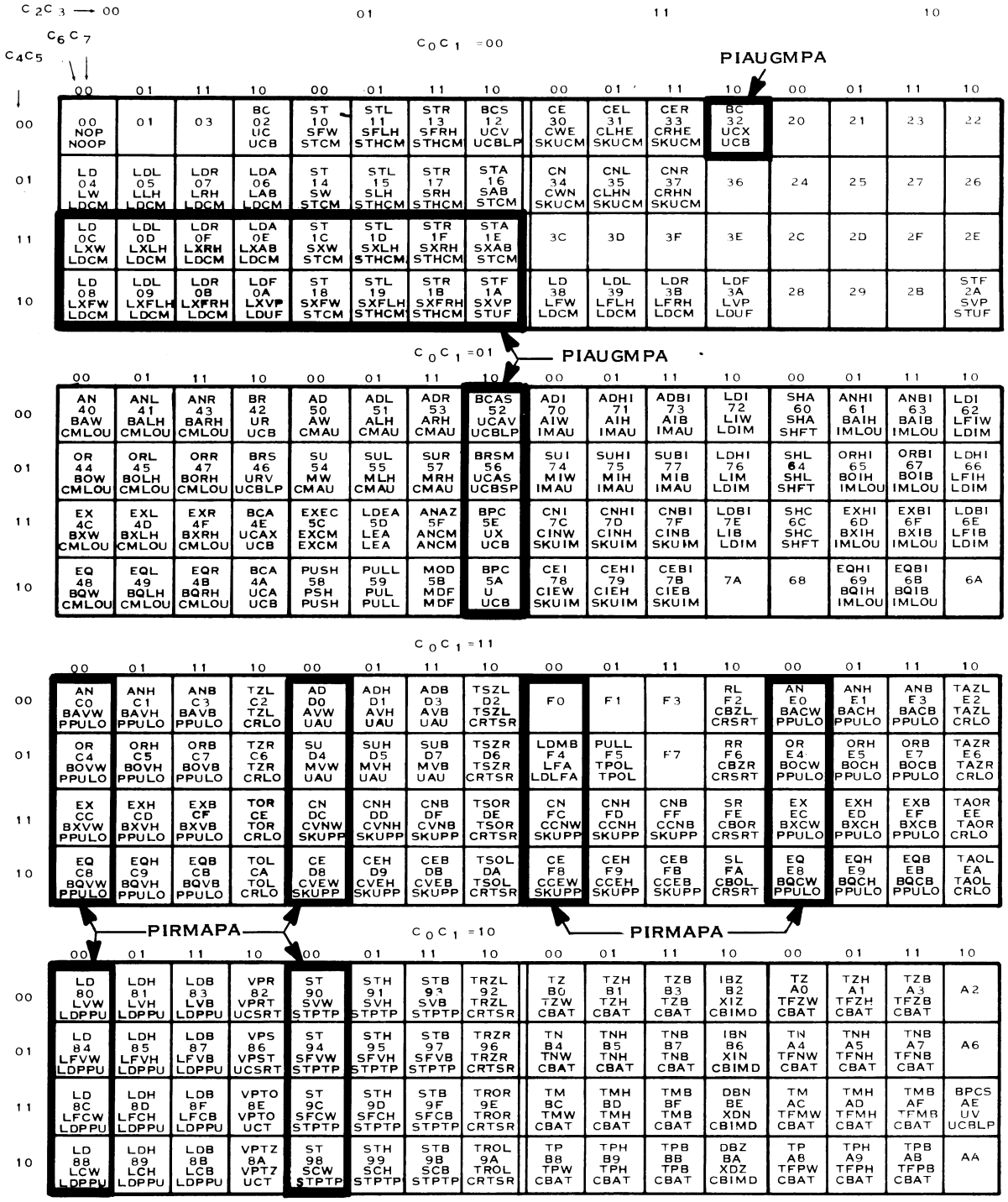
(A)124795

Figure 4-88. Load CR (PILDCR) Op-Code Groups



(A)124796

Figure 4-89. Load VPR (PILDVPR) Op-Code Groups



(A)124797

Figure 4-90. Remapped and Augmented Remapped Op-Code Groups

$C_2C_3 \rightarrow 00$ 01 11 10

C_6C_7

C_4C_5

$C_0C_1 = 00$

C_4C_5	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	00 NOP NOOP	01	03	BC 02 UC UCB	ST 10 SFW STCM	STL 11 SFLH STHCM	STR 13 SFRH STHCM	BCS 12 UCV UCBLP	CE 30 CWE SKUCM	CEL 31 CLHE SKUCM	CER 33 CRHE SKUCM	BC 32 UCX UCB	20	21	23	22
01	LD 04 LW LDCM	LDL 05 LLH LDCM	LDR 07 LRH LDCM	LDA 06 LAB LDCM	ST 14 SW STCM	STL 15 SLH STHCM	STR 17 SRH STHCM	STA 16 SAB STCM	CN 34 CWN SKUCM	CNL 35 CLHN SKUCM	CNR 37 CRHN SKUCM	36	24	25	27	26
11	LD 0C LXW LDCM	LDL 0D LXLH LDCM	LDR 0F LXRH LDCM	LDA 0E LXAB LDCM	ST 1C SXW STCM	STL 1D SXLH STHCM	STR 1F SXRH STHCM	STA 1E SXAB STCM	3C	3D	3F	3E	2C	2D	2F	2E
10	LD 08 LXFW LDCM	LDL 09 LXFLH LDCM	LDR 0B LXFRH LDCM	LDF 0A LXVP LDUF	ST 18 SXFW STCM	STL 19 SXFLH STHCM	STR 1B SXFRH STHCM	STF 1A SXVP STUF	LD 38 LFW LDCM	LDL 39 LFLH LDCM	LDR 3B LFRH LDCM	LDF 3A LVP LDUF	28	29	2B	STF 2A SVP STUF

$C_0C_1 = 01$

	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	AN 40 BAW CMLOU	ANL 41 BALH CMLOU	ANR 43 BARH CMLOU	BR 42 UR UCB	AD 50 AW CMAU	ADL 51 ALH CMAU	ADR 53 ARH CMAU	BCAS 52 UCAV UCBLP	ADI 70 AIW IMAU	ADHI 71 AIH IMAU	ADBI 73 AIB IMAU	LDI 72 LIW LDIM	SHA 60 SHA SHFT	ANHI 61 BAIH IMLOU	ANBI 63 BAIB IMLOU	LDI 62 LFIW LDIM
01	OR 44 BOW CMLOU	ORL 45 BOLH CMLOU	ORR 47 BORH CMLOU	BRS 46 URV UCBLP	SU 54 MW CMAU	SUL 55 MLH CMAU	SUR 57 MRH CMAU	BRSM 56 UCAS UCBSP	SUI 74 MIW IMAU	SUHI 75 MIH IMAU	SUBI 77 MIB IMAU	LDHI 76 LIM LDIM	SHL 64 SHL SHFT	ORHI 65 BOIH IMLOU	ORBI 67 BOIB IMLOU	LDHI 66 LFIH LDIM
11	EX 4C BXW CMLOU	EXL 4D BXLH CMLOU	EXR 4F BXRH CMLOU	BCA 4E UCAX UCB	EXEC 5C EXCM EXCM	LDEA 5D LEA LEA	ANAZ 5F ANCM ANCM	BPC 5E UX UCB	CNI 7C CINW SKUIM	CNHI 7D CINH SKUIM	CNBI 7F CINB SKUIM	LDBI 7E LIB LDIM	SHC 6C SHC SHFT	EXHI 6D BXIH IMLOU	EXBI 6F BXIB IMLOU	LDBI 6E LFIB LDIM
10	EQ 48 BQW CMLOU	EQL 49 BQLH CMLOU	EQR 4B BQRH CMLOU	BCA 4A UCA UCB	PUSH 58 PSH PUSH	PULL 59 PUL PULL	MOD 5B MDF MDF	BPC 5A U UCB	CEI 78 CIEW SKUIM	CEHI 79 CIEH SKUIM	CEBI 7B CIEB SKUIM	7A	68	EQHI 69 BQIH IMLOU	EQBI 6B BQIB IMLOU	6A

$C_0C_1 = 11$

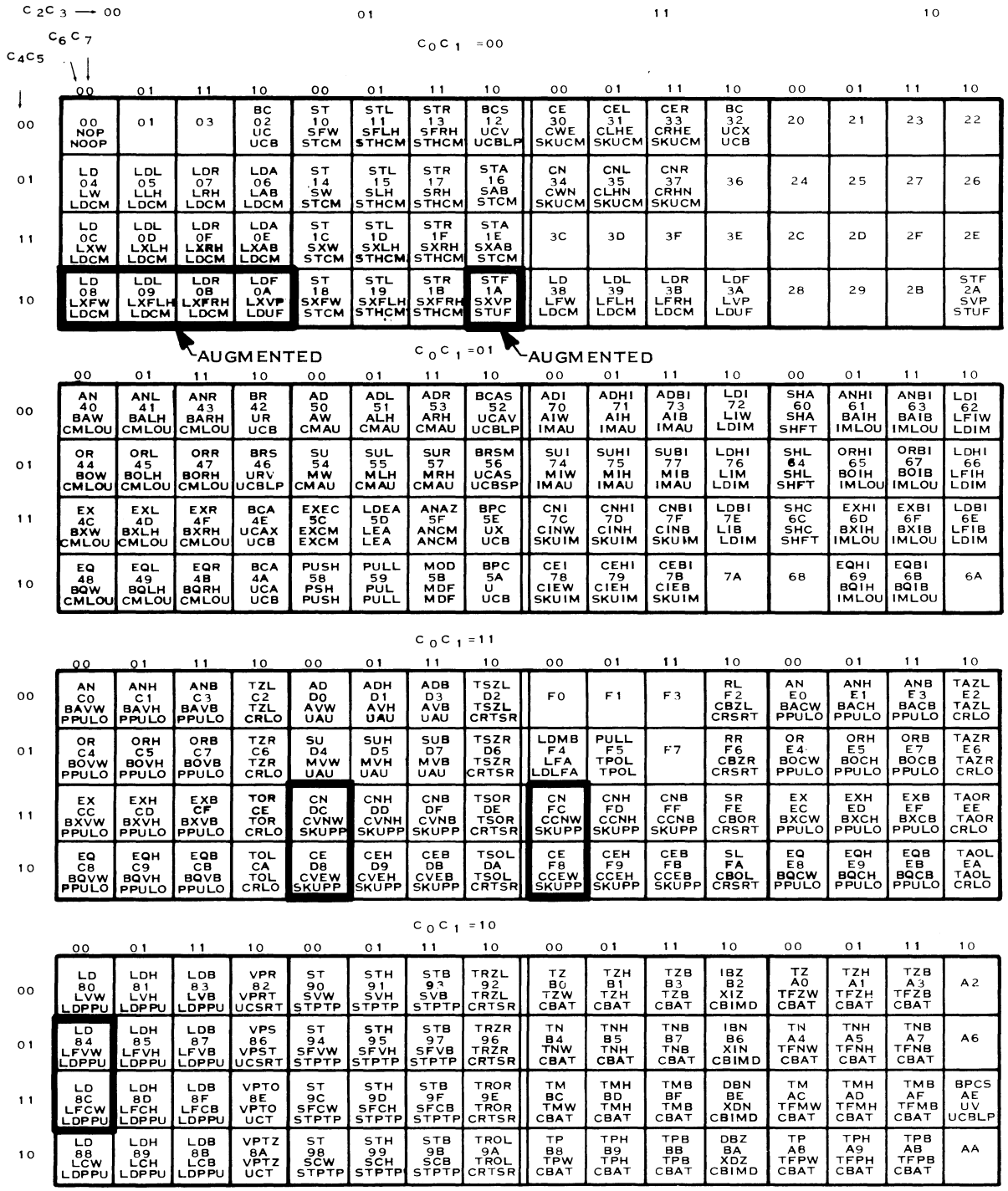
	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	AN C0 BAVW PPULO	ANH C1 BAVH PPULO	ANB C3 BAVB PPULO	TZL C2 TZL CRLO	AD D0 AVW UAU	ADH D1 AVH UAU	ADB D3 AVB UAU	TSZL D2 TSZL CRTSR	F0	F1	F3	RL F2 CBZL CRSRT	AN E0 BACW PPULO	ANH E1 BACH PPULO	ANB E3 BACB PPULO	TAZL E2 TAZL CRLO
01	OR C4 BOVW PPULO	ORH C5 BOVH PPULO	ORB C7 BOVB PPULO	TZR C6 TZR CRLO	SU D4 MVW UAU	SUH D5 MVH UAU	SUB D7 MVB UAU	TSZR D6 TSZR CRTSR	LDMB F4 LFA LDLFA	PULL F5 TPOL TPOL	F7	RR F6 CBZR CRSRT	OR E4 BOCW PPULO	ORH E5 BOCH PPULO	ORB E7 BOCB PPULO	TAZR E6 TAZR CRLO
11	EX CC BXVW PPULO	EXH CD BXVH PPULO	EXB CF BXVB PPULO	TOR CE TOR CRLO	CN DC CVNW SKUPP	CNH DD CVNH SKUPP	CNB DF CVNB SKUPP	TSOR DE TSOR CRTSR	CN FC CCNW SKUPP	CNH FD CCNH SKUPP	CNB FF CCNB SKUPP	SR FE CBOR CRSRT	EX EC BXCW PPULO	EXH ED BXCH PPULO	EXB EF BXCB PPULO	TAOR EE TAOR CRLO
10	EQ C8 BQVW PPULO	EQH C9 BQVH PPULO	EQB CB BQVB PPULO	TOL CA TOL CRLO	CE D8 CVEW SKUPP	CEH D9 CVEH SKUPP	CEB DB CVEB SKUPP	TSOL DA TSOL CRTSR	CE F8 CCEW SKUPP	CEH F9 CCEH SKUPP	CEB FB CCEB SKUPP	SL FA CBOL CRSRT	EQ E8 BQCW PPULO	EQH E9 BQCH PPULO	EQB EB BQCB PPULO	TAOL EA TAOL CRLO

$C_0C_1 = 10$

	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	LD 80 LVW LDPPU	LDH 81 LVH LDPPU	LDB 83 LVB LDPPU	VPR 82 VPRT UCSRT	ST 90 SVW STPTP	STH 91 SVH STPTP	STB 93 SVB STPTP	TRZL 92 TRZL CRTSR	TZ B0 TZW CBAT	TZH B1 TZH CBAT	TZB B3 TZB CBAT	IBZ B2 XIZ CBIMD	TZ A0 TFZW CBAT	TZH A1 TFZH CBAT	TZB A3 TFZB CBAT	A2
01	LD 84 LFVW LDPPU	LDH 85 LFVH LDPPU	LDB 87 LFVB LDPPU	VPS 86 VPST UCSRT	ST 94 SFVW STPTP	STH 95 SFVH STPTP	STB 97 SFVB STPTP	TRZR 96 TRZR CRTSR	TN B4 TNW CBAT	TNH B5 TNH CBAT	TNB B7 TNB CBAT	IBN B6 XIN CBIMD	TN A4 TFNW CBAT	TNH A5 TFNH CBAT	TNB A7 TFNB CBAT	A6
11	LD 8C LFCW LDPPU	LDH 8D LFCH LDPPU	LDB 8F LFCB LDPPU	VPTO 8E VPTO UCT	ST 9C SFCW STPTP	STH 9D SFCH STPTP	STB 9F SFCB STPTP	TROR 9E TROR CRTSR	TM BC TMW CBAT	TMH BD TMH CBAT	TMB BF TMB CBAT	DBN BE XDN CBIMD	TM AC TFMW CBAT	TMH AD TFMH CBAT	TMB AF TFMB CBAT	BPCS AE UV UCBLP
10	LD 88 LCW LDPPU	LDH 89 LCH LDPPU	LDB 8B LCB LDPPU	VPTZ 8A VPTZ UCT	ST 98 SCW STPTP	STH 99 SCH STPTP	STB 9B SCB STPTP	TROL 9A TROL CRTSR	TP B8 TPW CBAT	TPH B9 TPH CBAT	TPB BB TPB CBAT	DBZ BA XDZ CBIMD	TP A8 TFPW CBAT	TPH A9 TFPH CBAT	TPB AB TFPB CBAT	AA

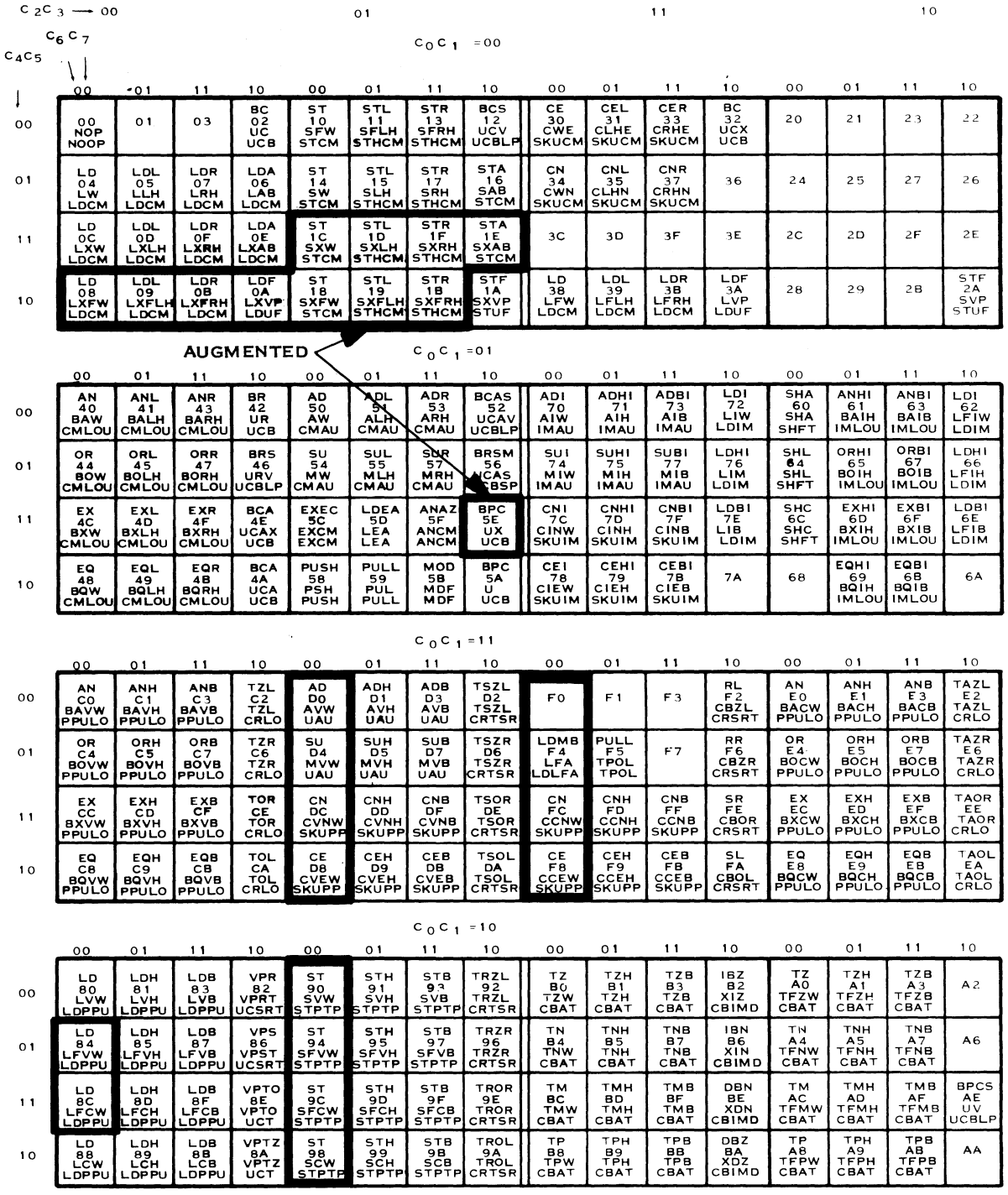
(A)124798

Figure 4-91. PIRMAPA · ┐PIAR(1) Remapped Op-Code Groups



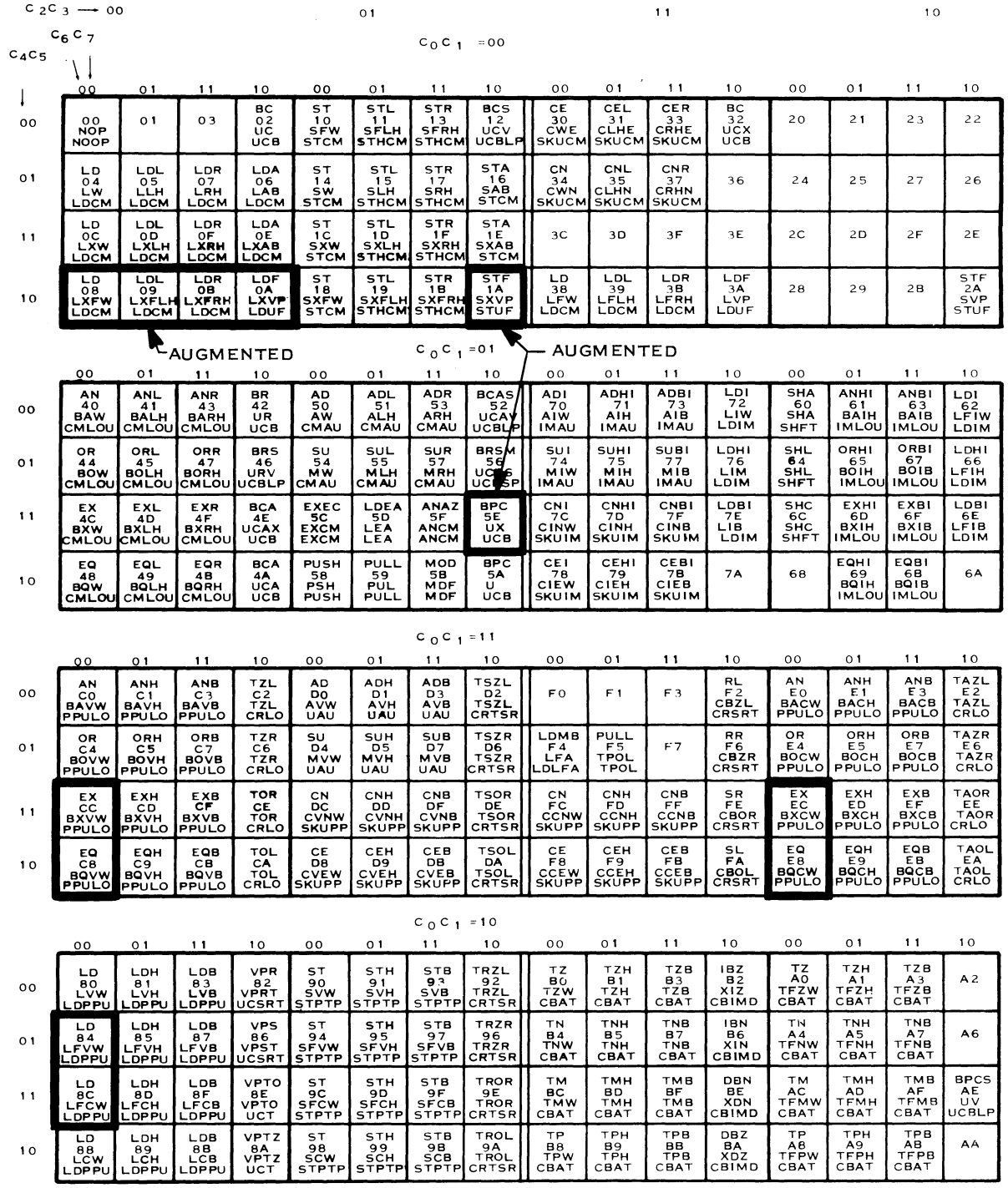
(A)124799

Figure 4-92. ¬PIRD(2) Remapped Op-Code Groups



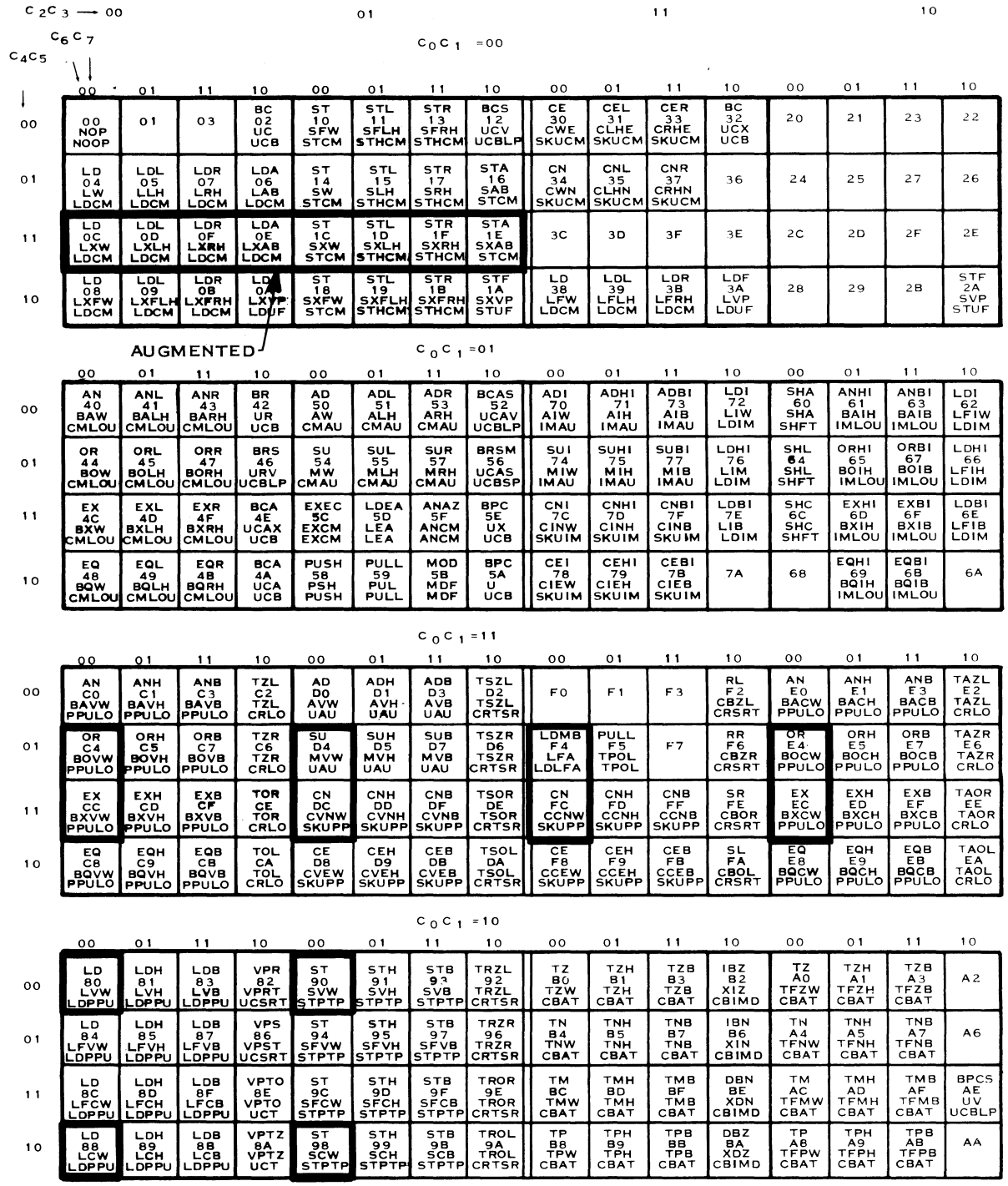
(A)124800

Figure 4-93. ¬PIRD(3) Remapped Op-Code Groups



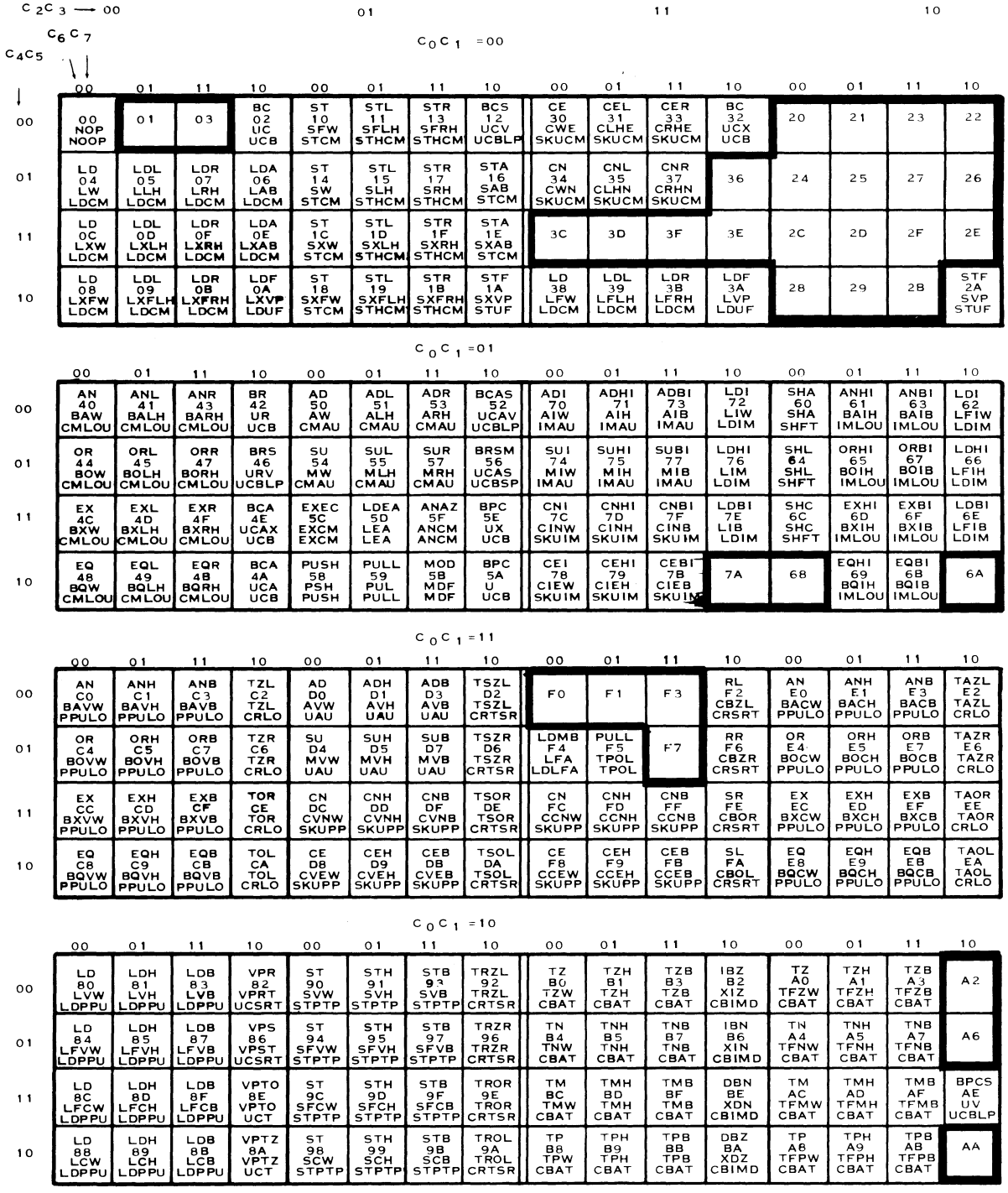
(A)124801

Figure 4-94. ¬PIRD(4) Remapped Op-Code Groups



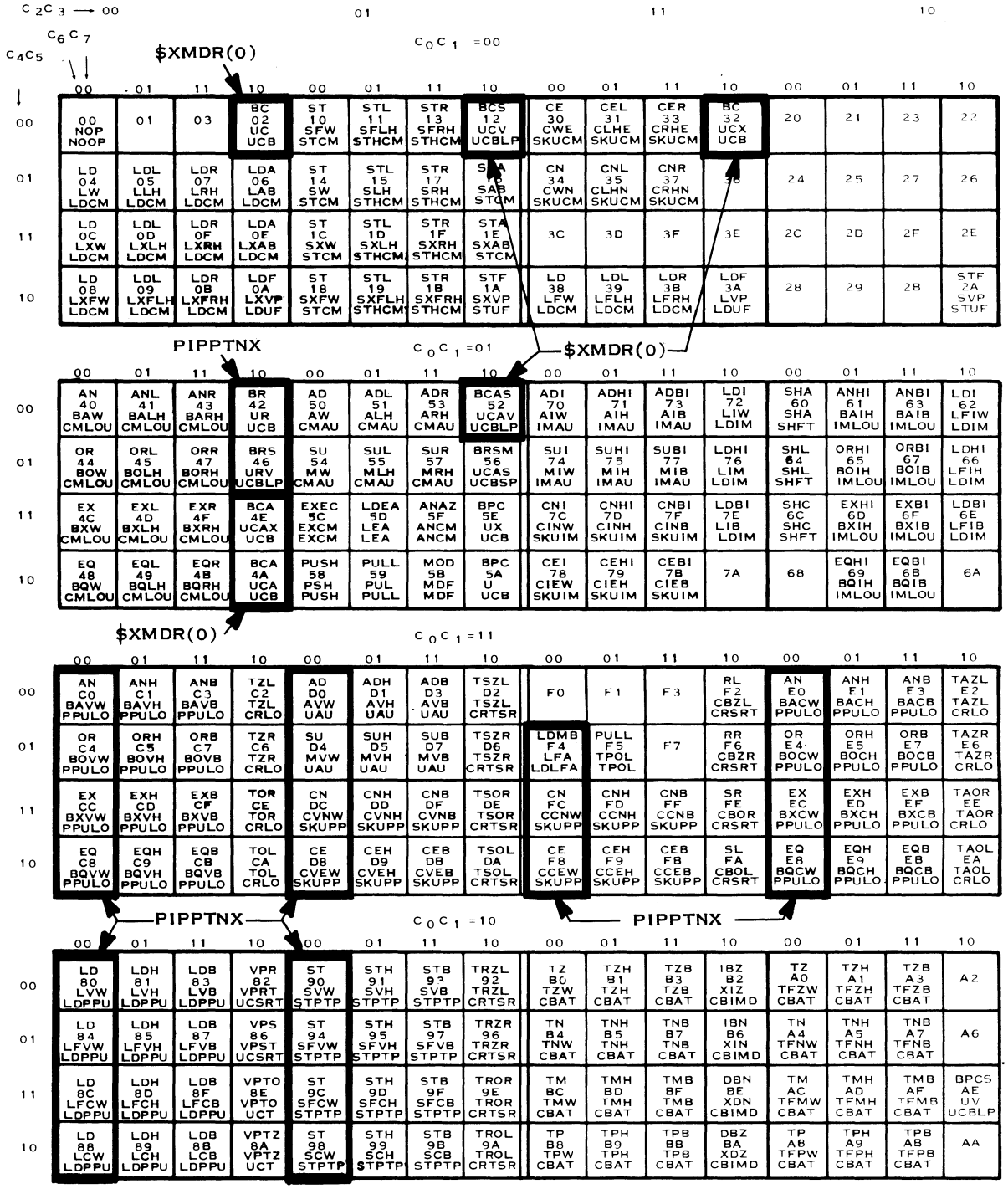
(A)124802

Figure 4-95. ¬PIRD(5) Remapped Op-Code Groups



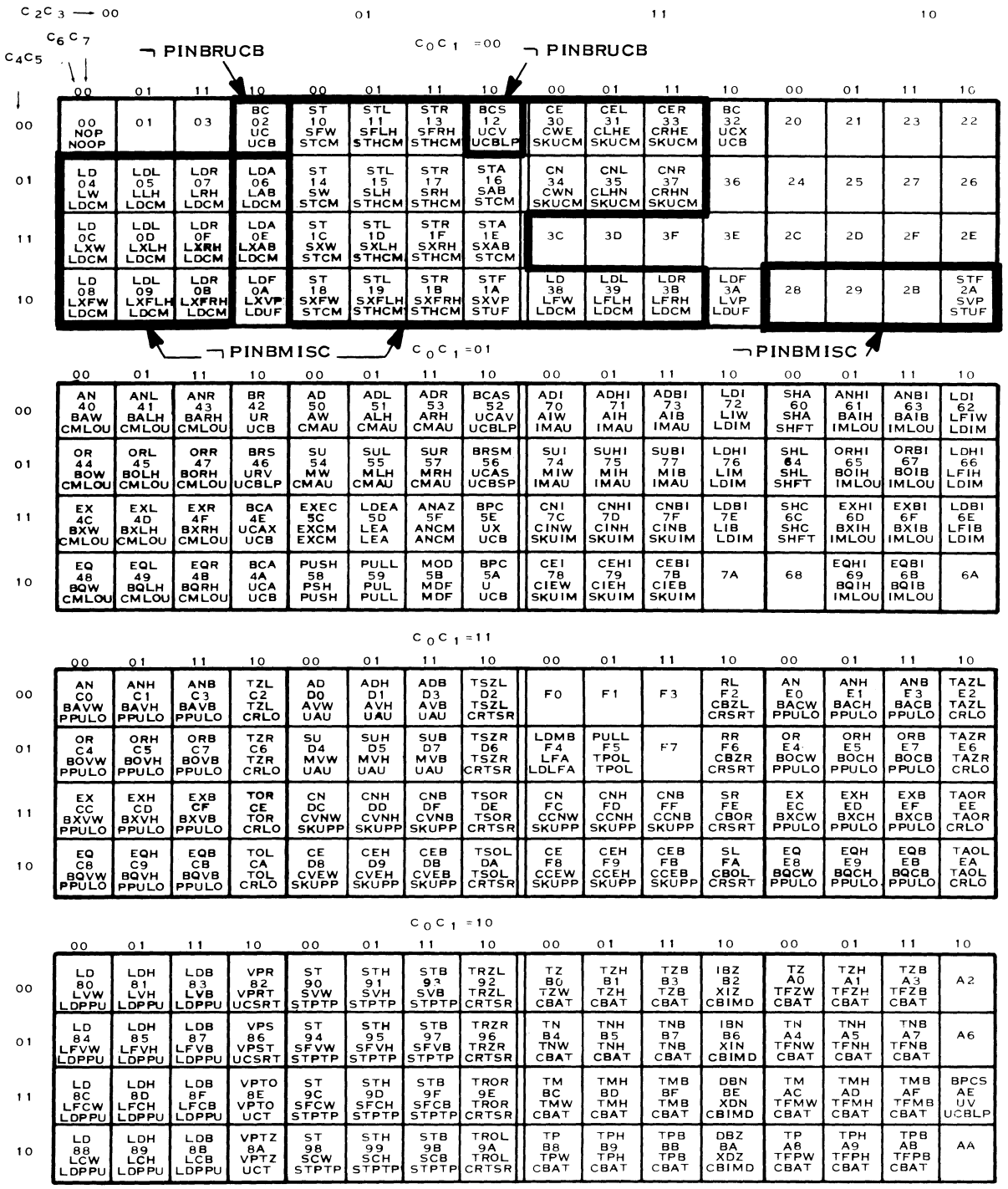
(A)124803

Figure 4-96. Illegal Op-Codes



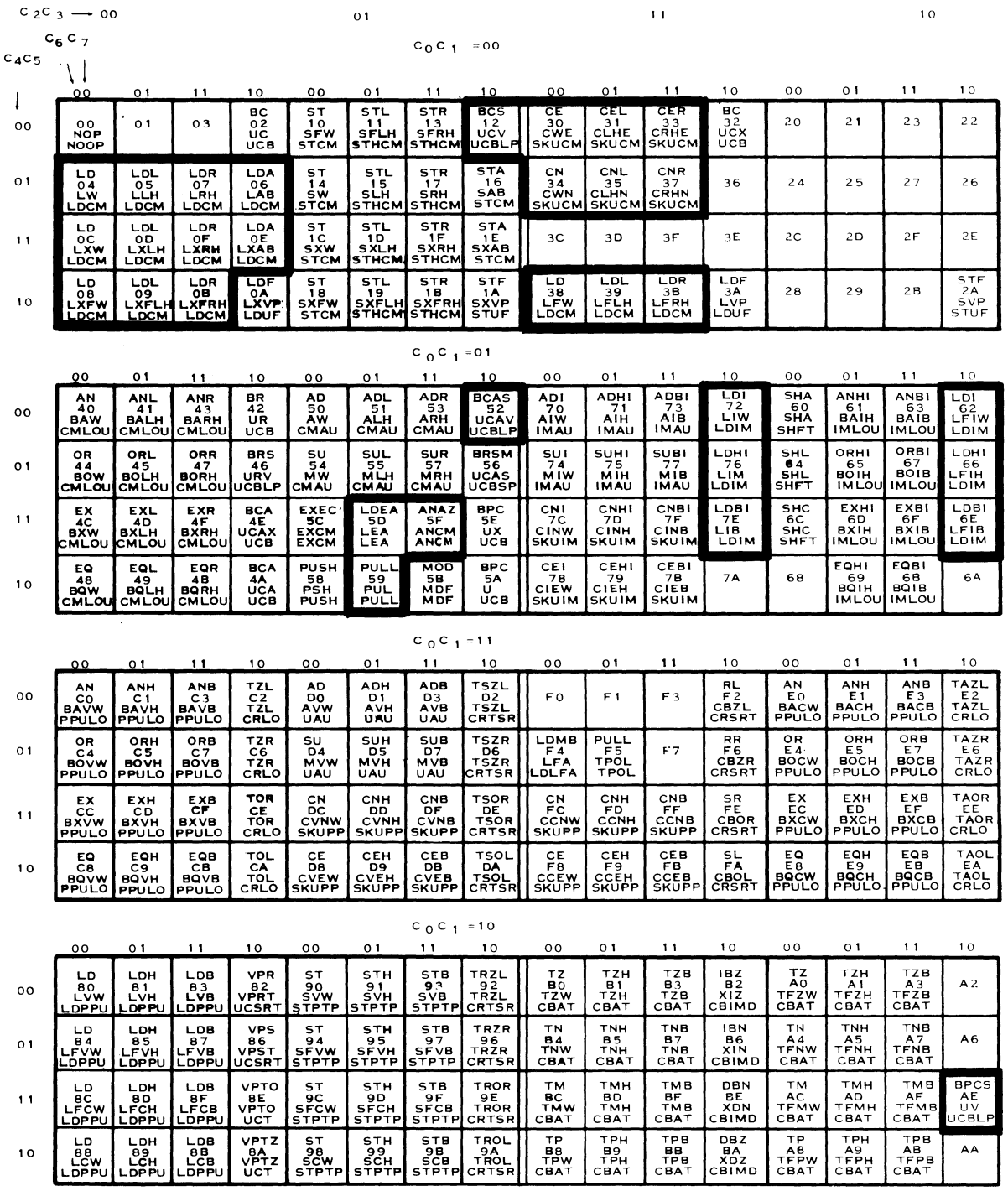
(A)124804

Figure 4-97. Unconditional Branch to Central Memory ($XMDR(0)) and Indirect Through CR or VPR (PIPPTNX) Op-Code Groups



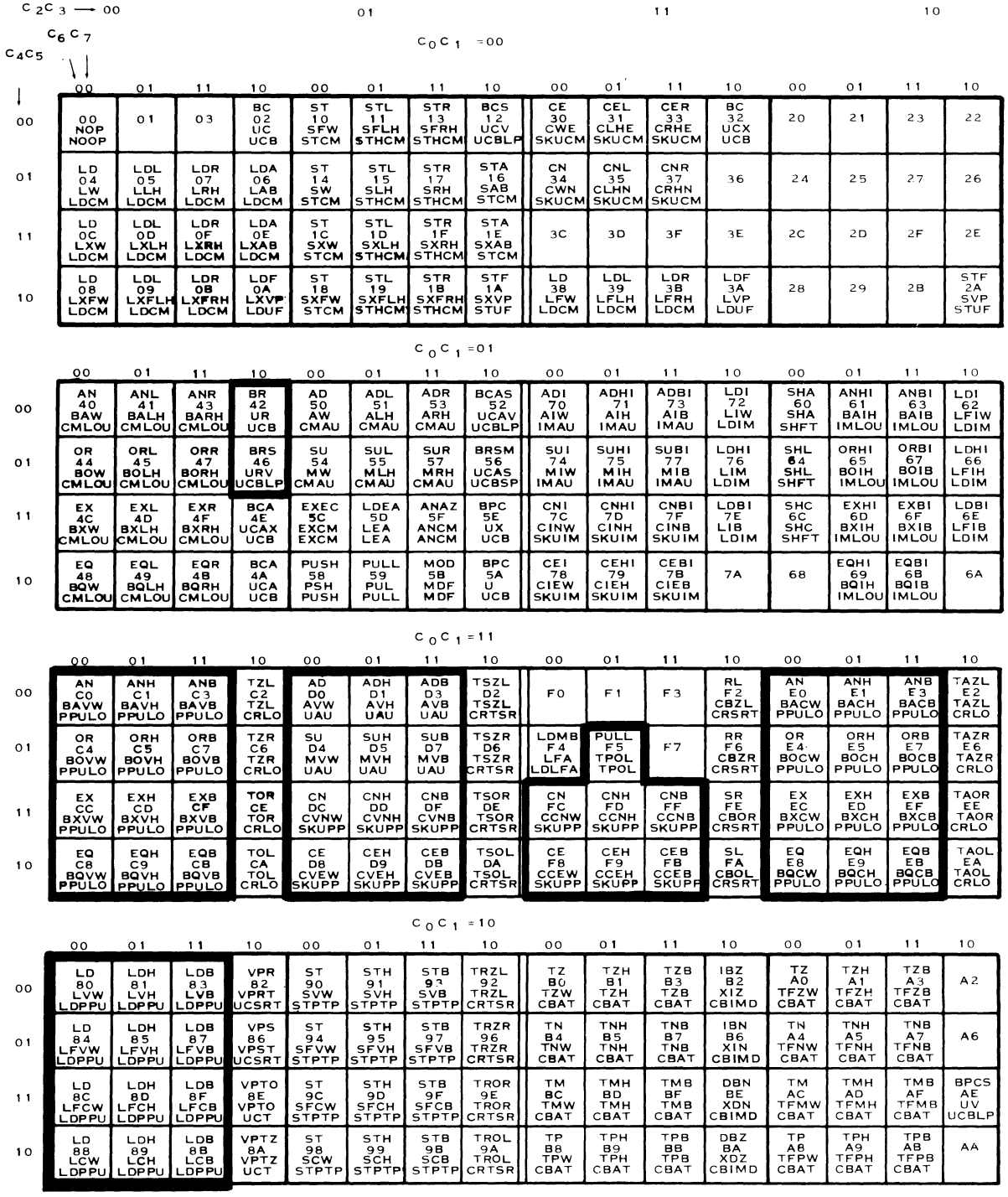
(A)124805

Figure 4-98. Base Relative Unconditional Branch to Central Memory (¬PINBRUCB) and Miscellaneous Central Memory (¬PINBMISC) Op-Code Groups



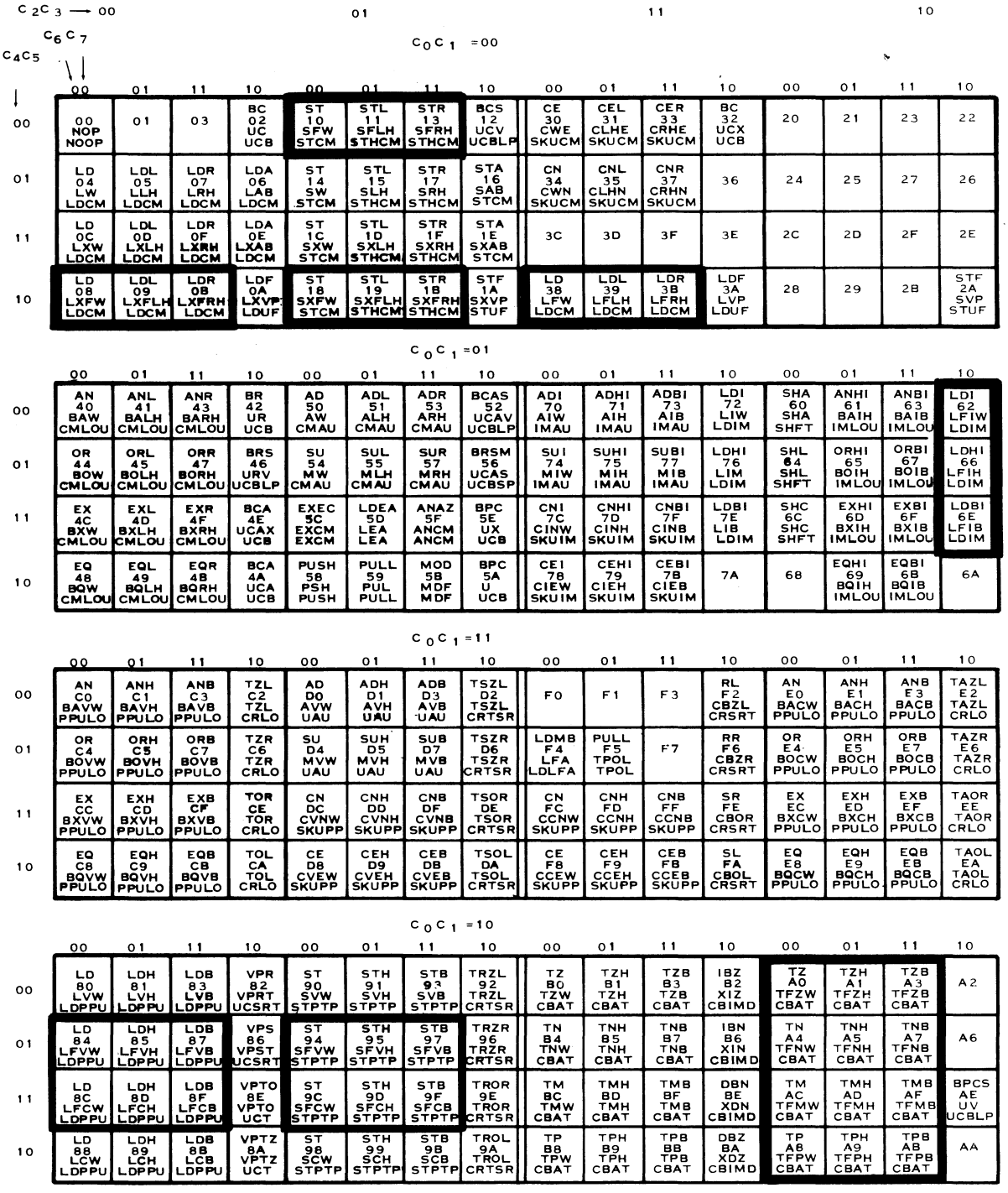
(A)124806

Figure 4-99. Register Indexer Supplied Destination (¬PISDR) Op-Code Groups (Source is Central Memory or Immediate)



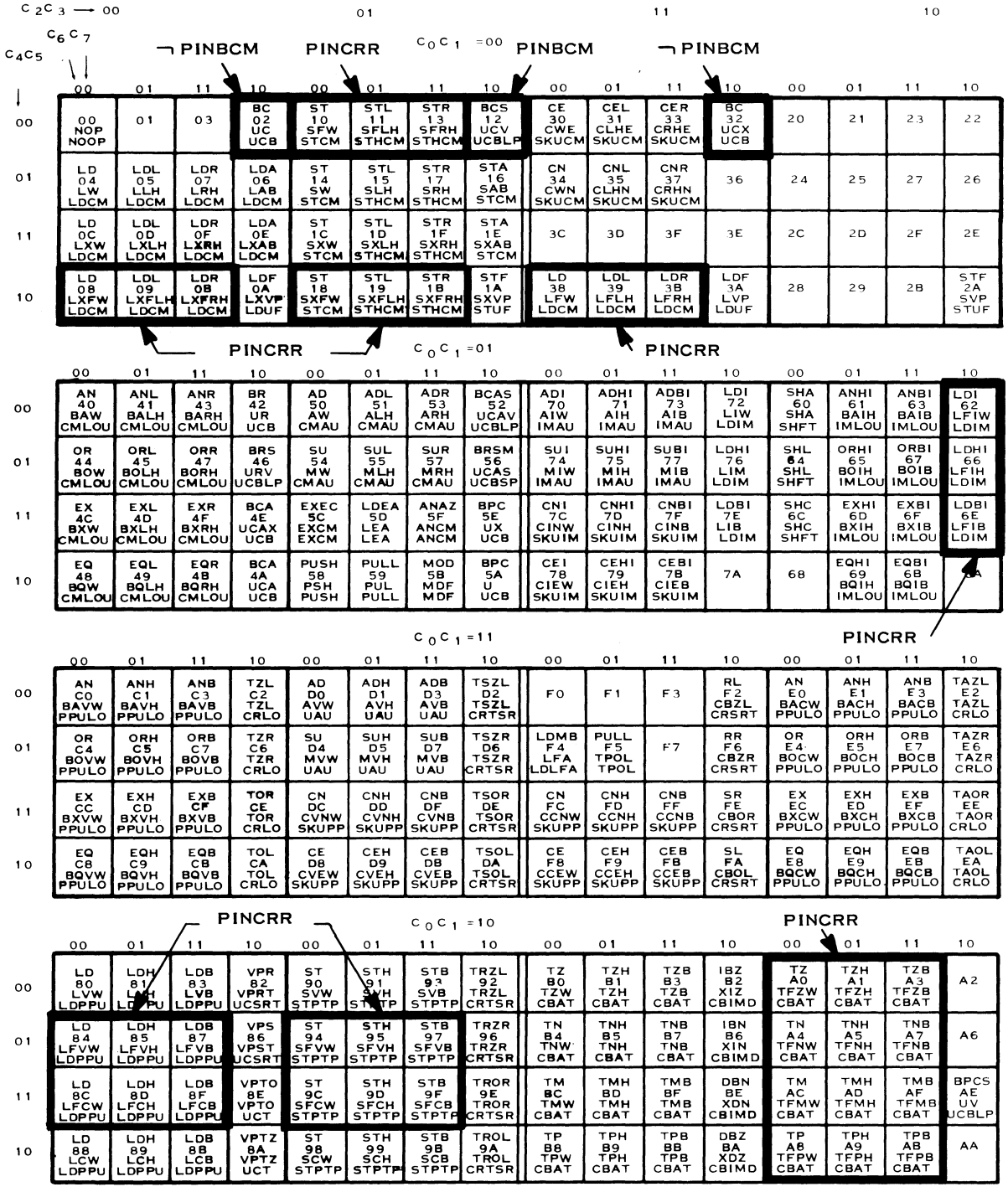
(A)124807

Figure 4-100. TN Field Indexer Supplied Source (¬PITATNR) Op-Code Groups



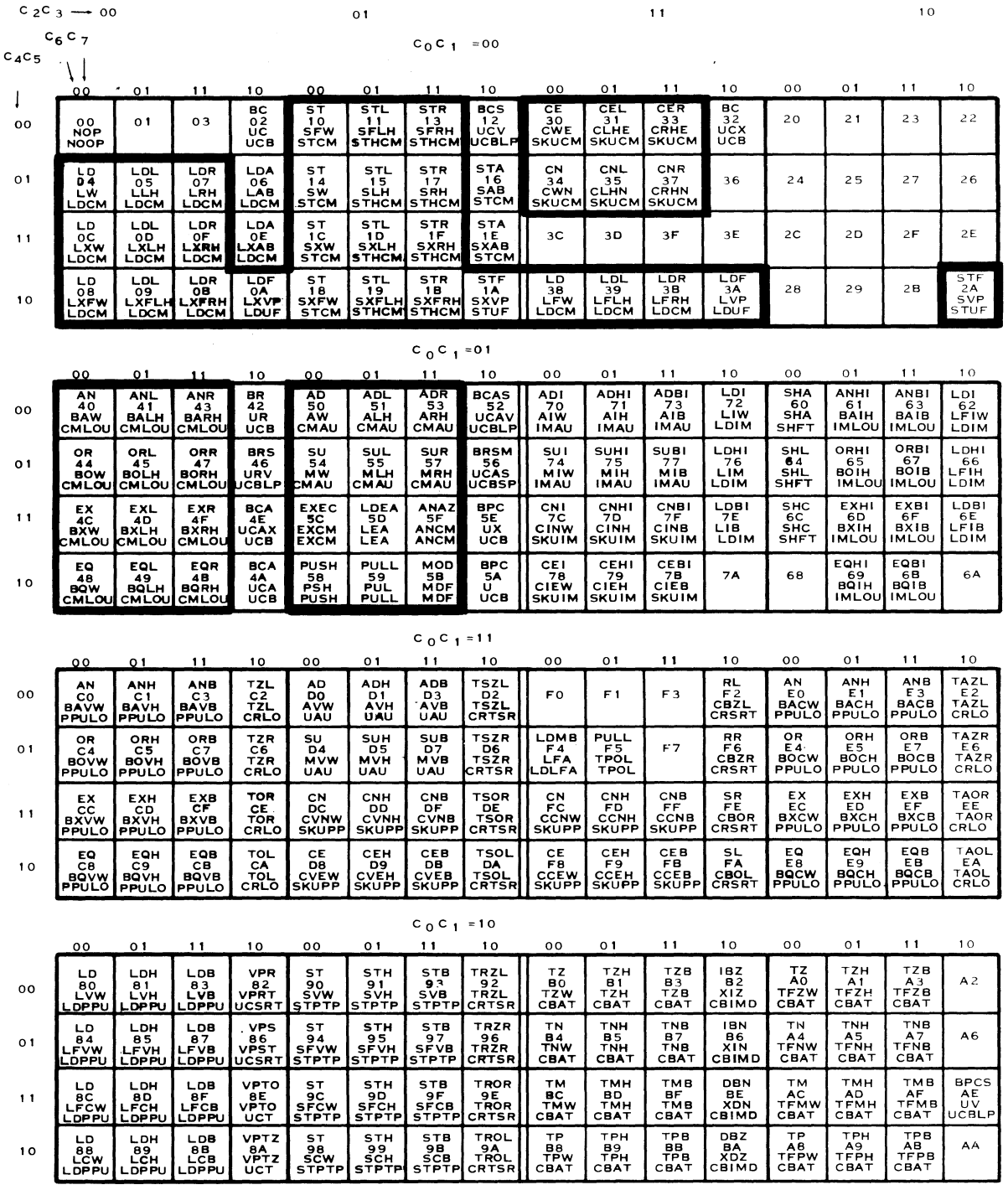
(A)124808

Figure 4-101. Register Indexer Dependent (PININDCR) Op-Code Groups



(A)124809

Figure 4-102. Base Relative Branch to Central Memory (¬PINBCM) and Register Indexer Specifying CR (PINCRR) Op-Code Groups



$C_2C_3 \rightarrow 00$ | 01 | 11 | 10

C_6C_7

C_4C_5

$C_0C_1 = 00$

C_4C_5 \ C_6C_7	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	00 NOP NOOP	01	03	BC 02 UC UCB	ST 10 SFW STCM	STL 11 SFLH STHCM	STR 13 SFRH STHCM	BCS 12 UCV UCBLP	CE 30 CWE SKUCM	CEL 31 CLHE SKUCM	CER 33 CRHE SKUCM	BC 32 UCX UCB	20	21	23	22
01	LD 04 LW LDCM	LDL 05 LLH LDCM	LDR 07 LRH LDCM	LDA 06 LAB LDCM	ST 14 SW STCM	STL 15 SLH STHCM	STR 17 SRH STHCM	STA 16 SAB STCM	CN 34 CWN SKUCM	CNL 35 CLHN SKUCM	CNR 37 CRHN SKUCM	36	24	25	27	26
11	LD 0C LXW LDCM	LDL 0D LXLH LDCM	LDR 0F LXRH LDCM	LDA 0E LXAB LDCM	ST 1C SXW STCM	STL 1D SXLH STHCM	STR 1F SXRH STHCM	STA 1E SXAB STCM	3C	3D	3F	3E	2C	2D	2F	2E
10	LD 08 LXFW LDCM	LDL 09 LXFLH LDCM	LDR 0B LXFRH LDCM	LDF 0A LXVP LDUF	ST 18 SXFW STCM	STL 19 SXFLH STHCM	STR 1B SXFRH STHCM	STF 1A SXVP STUF	LD 38 LFW LDCM	LDL 39 LFLH LDCM	LDR 3B LFRH LDCM	LDF 3A LVP LDUF	28	29	2B	STF 2A SVP STUF

$C_0C_1 = 01$

C_4C_5 \ C_6C_7	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	AN 40 BAW CMLOU	ANL 41 BALH CMLOU	ANR 43 BARH CMLOU	BR 42 UR UCB	AD 50 AW CMAU	ADL 51 ALH CMAU	ADR 53 ARH CMAU	BCAS 52 UCAV UCBLP	ADI 70 AIW IMAU	ADHI 71 AIH IMAU	ADBI 73 AIB IMAU	LDI 72 LIW LDIM	SHA 60 SHA SHFT	ANHI 61 BAIH IMLOU	ANBI 63 BAIB IMLOU	LDI 62 LFIW LDIM
01	OR 44 BOW CMLOU	ORL 45 BOLH CMLOU	ORR 47 BORH CMLOU	BRS 46 URV UCBLP	SU 54 MW CMAU	SUL 55 MLH CMAU	SUR 57 MRH CMAU	BRSM 56 UCAS UCBSP	SUI 74 MIW IMAU	SUHI 75 MIH IMAU	SUBI 77 MIB IMAU	LDHI 76 LIM LDIM	SHL 64 SHL SHFT	ORHI 65 BOIH IMLOU	ORBI 67 BOIB IMLOU	LDHI 66 LFIH LDIM
11	EX 4C BXW CMLOU	EXL 4D BXLH CMLOU	EXR 4F BXRH CMLOU	BCA 4E UCAX UCB	EXEC 5C EXCM EXCM	LDEA 5D LEA LEA	ANAZ 5F ANCM ANCM	BPC 5E UX UCB	CNI 7C CINW SKUIM	CNHI 7D CINH SKUIM	CNBI 7F CINB SKUIM	LDBI 7E LIB LDIM	SHC 6C SHC SHFT	EXHI 6D BXIH IMLOU	EXBI 6F BXIB IMLOU	LDBI 6E LFIB LDIM
10	EQ 48 BQW CMLOU	EQL 49 BQLH CMLOU	EQR 4B BQRH CMLOU	BCA 4A UCA UCB	PUSH 58 PSH PUSH	PULL 59 PUL PULL	MOD 5B MDF MDF	BPC 5A U UCB	CEI 78 CIEW SKUIM	CEHI 79 CIEH SKUIM	CEBI 7B CIEB SKUIM	7A	68	EQHI 69 BQIH IMLOU	EQBI 6B BQIB IMLOU	6A

$C_0C_1 = 11$

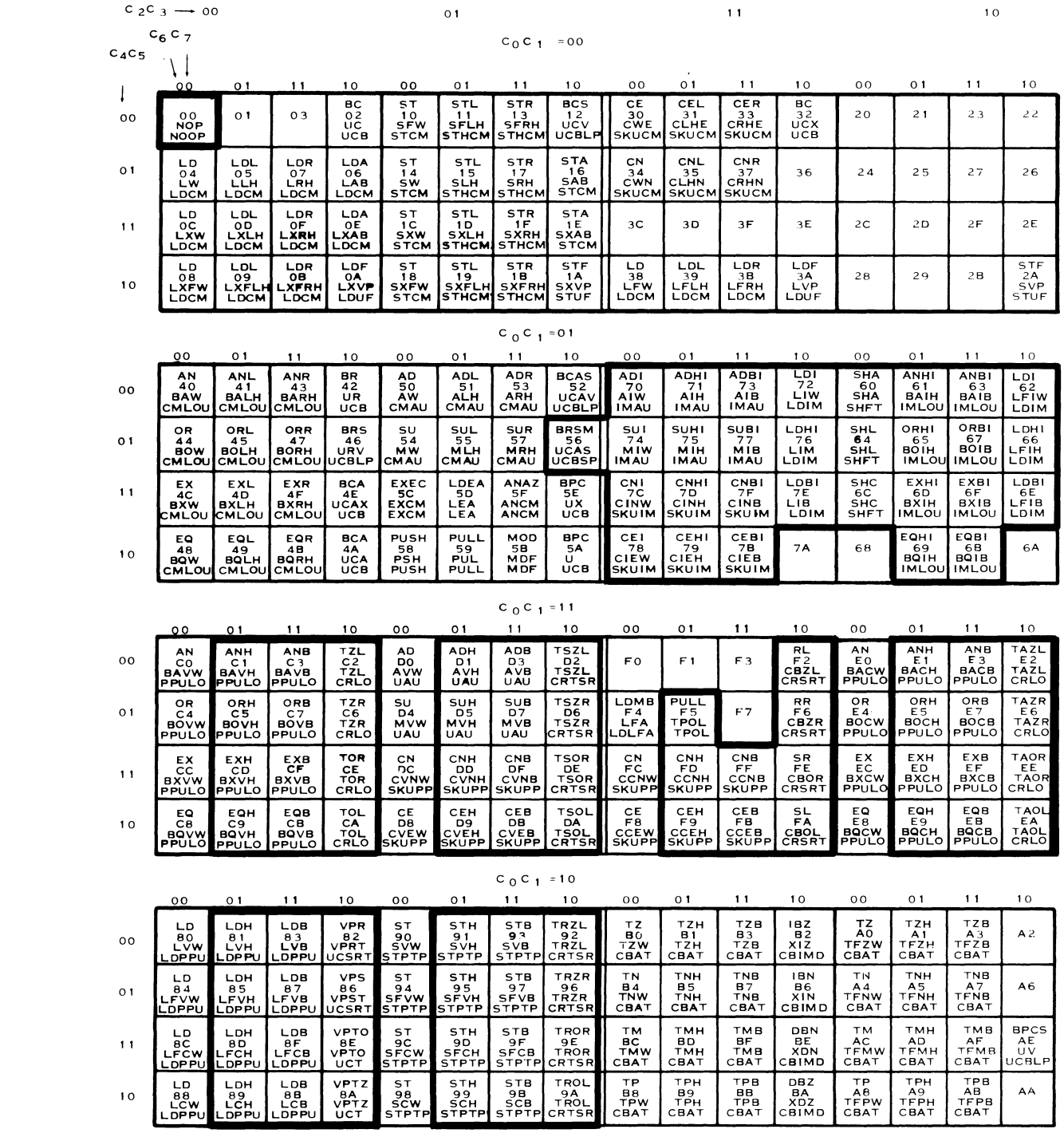
C_4C_5 \ C_6C_7	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	AN C0 BAVW PPULO	ANH C1 BAVH PPULO	ANB C3 BAVB PPULO	TZL C2 TZL CRLO	AD D0 AVW UAU	ADH D1 AVH UAU	ADB D3 AVB UAU	TSZL D2 TSZL CRTSR	F0	F1	F3	RL F2 CBZL CRSRT	AN E0 BACW PPULO	ANH E1 BACH PPULO	ANB E3 BACB PPULO	TAZL E2 TAZL CRLO
01	OR C4 BOVW PPULO	ORH C5 BOVH PPULO	ORB C7 BOVB PPULO	TZR C6 TZR CRLO	SU D4 MVW UAU	SUH D5 MVH UAU	SUB D7 MVB UAU	TSZR D6 TSZR CRTSR	LDMB F4 LFA LDLFA	PULL F5 TPOL TPOL	F7	RR F6 CBZR CRSRT	OR E4 BOCW PPULO	ORH E5 BOCH PPULO	ORB E7 BOCB PPULO	TAZR E6 TAZR CRLO
11	EX CC BXVW PPULO	EXH CD BXVH PPULO	EXB CF BXVB PPULO	TOR CE TOR CRLO	CN DC CVNW SKUPP	CNH DD CVNH SKUPP	CNB DF CVNB SKUPP	TSOR DE TSOR CRTSR	CN FC CCNW SKUPP	CNH FD CCNH SKUPP	CNB FF CCNB SKUPP	SR FE CBOR CRSRT	EX EC BXCW PPULO	EXH ED BXCH PPULO	EXB EF BXCB PPULO	TAOR EE TAOR CRLO
10	EQ C8 BQVW PPULO	EQH C9 BQVH PPULO	EQB CB BQVB PPULO	TOL CA TOL CRLO	CE D8 CVEW SKUPP	CEH D9 CVEH SKUPP	CEB DB CVEB SKUPP	TSOL DA TSOL CRTSR	CE F8 CCEW SKUPP	CEH F9 CCEH SKUPP	CEB FB CCEB SKUPP	SL FA CBOL CRSRT	EQ E8 BQCW PPULO	EQH E9 BQCH PPULO	EQB EB BQCB PPULO	TAOL EA TAOL CRLO

$C_0C_1 = 10$

C_4C_5 \ C_6C_7	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	LD 80 LVW LDPPU	LDH 81 LVH LDPPU	LDB 83 LVB LDPPU	VPR 82 VPRT UCSRT	ST 90 SVW STPTP	STH 91 SVH STPTP	STB 93 SVB STPTP	TRZL 92 TRZL CRTSR	TZ B0 TZW CBAT	TZH B1 TZH CBAT	TZB B3 TZB CBAT	IBZ B2 XIZ CBIMD	TZ A0 TFZW CBAT	TZH A1 TFZH CBAT	TZB A3 TFZB CBAT	A2
01	LD 84 LFVW LDPPU	LDH 85 LFVH LDPPU	LDB 87 LFVB LDPPU	VPS 86 VPST UCSRT	ST 94 SFVW STPTP	STH 95 SFVH STPTP	STB 97 SFVB STPTP	TRZR 96 TRZR CRTSR	TN B4 TNW CBAT	TNH B5 TNH CBAT	TNB B7 TNB CBAT	IBN B6 XIN CBIMD	TN A4 TFNW CBAT	TNH A5 TFNH CBAT	TNB A7 TFNB CBAT	A6
11	LD 8C LFCW LDPPU	LDH 8D LFCH LDPPU	LDB 8F LFCB LDPPU	VPTO 8E VPTO UCT	ST 9C SFCW STPTP	STH 9D SFCH STPTP	STB 9F SFCB STPTP	TROR 9E TROR CRTSR	TM BC TMW CBAT	TMH BD TMH CBAT	TMB BF TMB CBAT	DBN BE XDN CBIMD	TM AC TFMW CBAT	TMH AD TFMH CBAT	TMB AF TFMB CBAT	BPCS AE UV UCBLP
10	LD 88 LCW LDPPU	LDH 89 LCH LDPPU	LDB 8B LCB LDPPU	VPTZ 8A VPTZ UCT	ST 98 SCW STPTP	STH 99 SCH STPTP	STB 9B SCB STPTP	TROL 9A TROL CRTSR	TP B8 TPW CBAT	TPH B9 TPH CBAT	TPB BB TPB CBAT	DBZ BA XDZ CBIMD	TP A8 TFPW CBAT	TPH A9 TFPH CBAT	TPB AB TFPB CBAT	AA

(A)124810

Figure 4-103. TN Field Specifying Central Memory (¬PINCMTN) Op-Code Groups



$C_2C_3 \rightarrow 00$ | 01 | 11 | 10

C_6C_7

C_4C_5

$C_0C_1 = 00$

	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	00 NOP NOOP	01	03	BC 02 UC UCB	ST 10 SFW STCM	STL 11 SFLH STHCM	STR 13 SFRH STHCM	BCS 12 UCV UCBLP	CE 30 CWE SKUCM	CEL 31 CLHE SKUCM	CER 33 CRHE SKUCM	BC 32 UCX UCB	20	21	23	22
01	LD 04 LW LDCM	LDL 05 LLH LDCM	LDR 07 LRH LDCM	LDA 06 LAB LDCM	ST 14 SW STCM	STL 15 SLH STHCM	STR 17 SRH STHCM	STA 16 SAB STCM	CN 34 CWN SKUCM	CNL 35 CLHN SKUCM	CNR 37 CRHN SKUCM	36	24	25	27	26
11	LD 0C LXW LDCM	LDL 0D LXLH LDCM	LDR 0F LXRH LDCM	LDA 0E LXAB LDCM	ST 1C SXW STCM	STL 1D SXLH STHCM	STR 1F SXRH STHCM	STA 1E SXAB STCM	3C	3D	3F	3E	2C	2D	2F	2E
10	LD 08 LXFW LDCM	LDL 09 LXFLH LDCM	LDR 0B LXFRH LDCM	LDF 0A LXVP LDUF	ST 18 SXFW STCM	STL 19 SXFLH STHCM	STR 1B SXFRH STHCM	STF 1A SXVP STUF	LD 38 LFW LDCM	LDL 39 LFLH LDCM	LDR 3B LFRH LDCM	LDF 3A LVP LDUF	28	29	2B	STF 2A SVP STUF

$C_0C_1 = 01$

	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	AN 40 BAW CMLOU	ANL 41 BALH CMLOU	ANR 43 BARH CMLOU	BR 42 UR UCB	AD 50 AW CMAU	ADL 51 ALH CMAU	ADR 53 ARH CMAU	BCAS 52 UCAV UCBLP	ADI 70 AIW IMAU	ADHI 71 AIH IMAU	ADBI 73 AIB IMAU	LDI 72 LIW LDIM	SHA 60 SHA SHFT	ANHI 61 BAIH IMLOU	ANBI 63 BAIB IMLOU	LDI 62 LFIW LDIM
01	OR 44 BOW CMLOU	ORL 45 BOLH CMLOU	ORR 47 BORH CMLOU	BRS 46 URV UCBLP	SU 54 MW CMAU	SUL 55 MLH CMAU	SUR 57 MRH CMAU	BRSM 56 UCAS UCBSP	SUI 74 MIW IMAU	SUHI 75 MIH IMAU	SUBI 77 MIB IMAU	LDHI 76 LIM LDIM	SHL 64 SHL SHFT	ORHI 65 BOIH IMLOU	ORBI 67 BOIB IMLOU	LDHI 66 LFIH LDIM
11	EX 4C BXW CMLOU	EXL 4D BXLH CMLOU	EXR 4F BXRH CMLOU	BCA 4E UCAX UCB	EXEC 5C EXCM EXCM	LDEA 5D LEA LEA	ANAZ 5F ANCM ANCM	BPC 5E UX UCB	CNI 7C CINW SKUIM	CNHI 7D CINH SKUIM	CNBI 7F CINB SKUIM	LDBI 7E LIB LDIM	SHC 6C SHC SHFT	EXHI 6D BXIH IMLOU	EXBI 6F BXIB IMLOU	LDBI 6E LFIB LDIM
10	EQ 48 BQW CMLOU	EQL 49 BQLH CMLOU	EQR 4B BQRH CMLOU	BCA 4A UCA UCB	PUSH 58 PSH PUSH	PULL 59 PUL PULL	MOD 5B MDF MDF	BPC 5A U UCB	CEI 78 CIEW SKUIM	CEHI 79 CIEH SKUIM	CEBI 7B CIEB SKUIM	7A	68	EQHI 69 BQIH IMLOU	EQBI 6B BQIB IMLOU	6A

$C_0C_1 = 11$

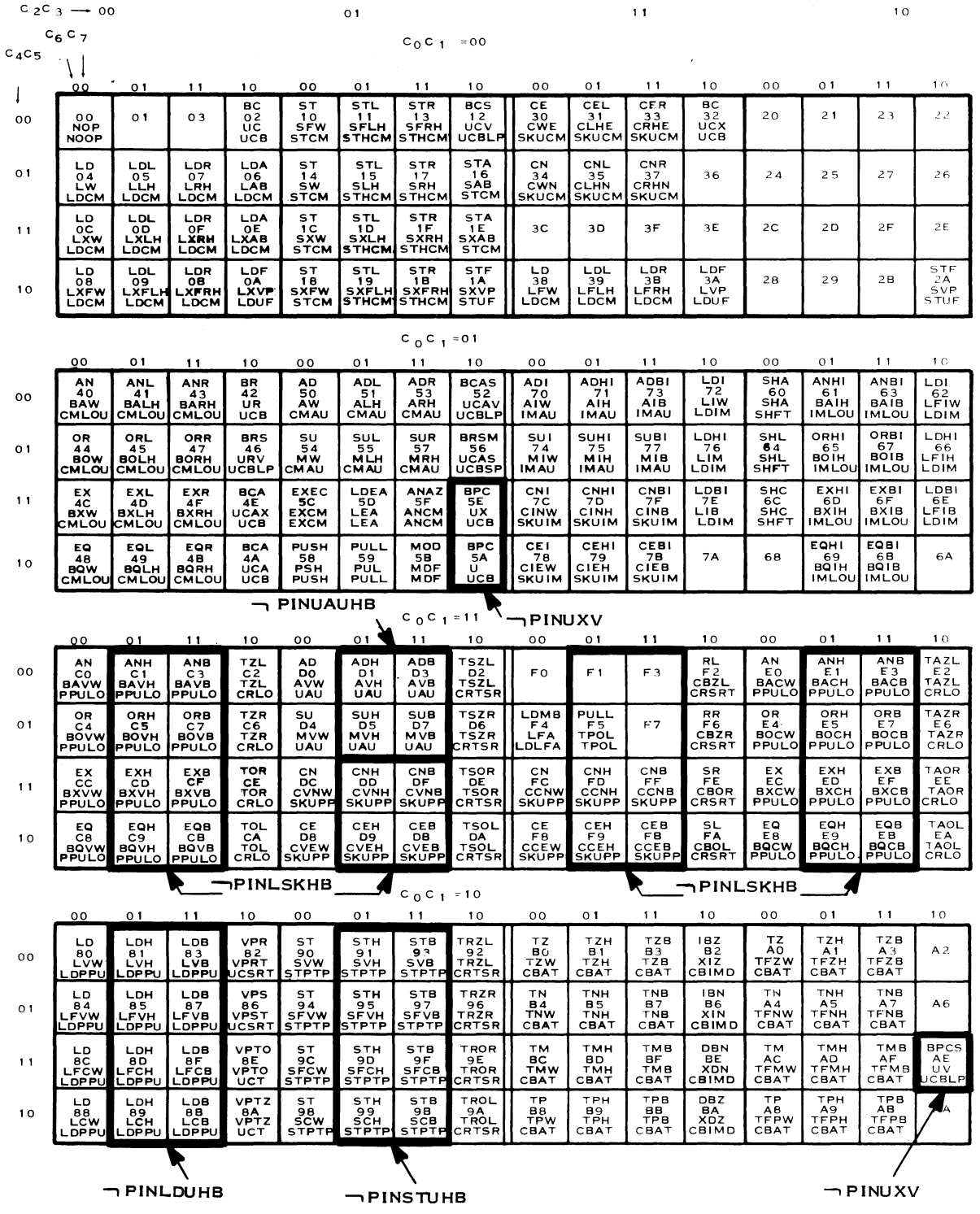
	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	AN C0 BAVW PPULO	ANH C1 BAVH PPULO	ANB C3 BAVB PPULO	TZL C2 TZL CRLO	AD D0 AVW UAU	ADH D1 AVH UAU	ADB D3 AVB UAU	TSZL D2 TSZL CRTSR	F0	F1	F3	RL F2 CBZL CRSRT	AN E0 BACW PPULO	ANH E1 BACH PPULO	ANB E3 BACB PPULO	TAZL E2 TAZL CRLO
01	OR C4 BOVW PPULO	ORH C5 BOVH PPULO	ORB C7 BOVB PPULO	TZR C6 TZR CRLO	SU D4 MVW UAU	SUH D5 MVH UAU	SUB D7 MVB UAU	TSZR D6 TSZR CRTSR	LDMB F4 LFA LDLFA	PULL F5 TPOL TPOL	F7	RR F6 CBZR CRSRT	OR E4 BOCW PPULO	ORH E5 BOCH PPULO	ORB E7 BOCB PPULO	TAZR E6 TAZR CRLO
11	EX CC BXVW PPULO	EXH CD BXVH PPULO	EXB CF BXVB PPULO	TOR CE TOR CRLO	CN DC CVNW SKUPP	CNH DD CVNH SKUPP	CNB DF CVNB SKUPP	TSOR DE TSOR CRTSR	CN FC CCNW SKUPP	CNH FD CCNH SKUPP	CNB FF CCNB SKUPP	SR FE CBOR CRSRT	EX EC BXCW PPULO	EXH ED BXCH PPULO	EXB EF BXCB PPULO	TAOR EE TAOR CRLO
10	EQ C8 BQVW PPULO	EQH C9 BQVH PPULO	EQB CB BQVB PPULO	TOL CA TOL CRLO	CE D8 CVEW SKUPP	CEH D9 CVEH SKUPP	CEB DB CVEB SKUPP	TSOL DA TSOL CRTSR	CE F8 CCEW SKUPP	CEH F9 CCEH SKUPP	CEB FB CCEB SKUPP	SL FA CBOL CRSRT	EQ E8 BQCW PPULO	EQH E9 BQCH PPULO	EQB EB BQCB PPULO	TAOL EA TAOL CRLO

$C_0C_1 = 10$

	00	01	11	10	00	01	11	10	00	01	11	10	00	01	11	10
00	LD 80 LVW LDPPU	LDH 81 LVH LDPPU	LDB 83 LVB LDPPU	VPR 82 VPRT UCSRT	ST 90 SVW STPTP	STH 91 SVH STPTP	STB 93 SVB STPTP	TRZL 92 TRZL CRTSR	TZ B0 TZW CBAT	TZH B1 TZH CBAT	TZB B3 TZB CBAT	IBZ B2 XIZ CBIMD	TZ A0 TFZW CBAT	TZH A1 TFZH CBAT	TZB A3 TFZB CBAT	A2
01	LD 84 LFVW LDPPU	LDH 85 LFVH LDPPU	LDB 87 LFVB LDPPU	VPS 86 VPST UCSRT	ST 94 SFVW STPTP	STH 95 SFVH STPTP	STB 97 SFVB STPTP	TRZR 96 TRZR CRTSR	TN B4 TNW CBAT	TNH B5 TNH CBAT	TNB B7 TNB CBAT	IBN B6 XIN CBIMD	TN A4 TFNW CBAT	TNH A5 TFNH CBAT	TNB A7 TFNB CBAT	A6
11	LD 8C LFCW LDPPU	LDH 8D LFCH LDPPU	LDB 8F LFCB LDPPU	VPTO 8E VPTO UCT	ST 9C SFCW STPTP	STH 9D SFCH STPTP	STB 9F SFCB STPTP	TROR 9E TROR CRTSR	TM BC TMW CBAT	TMH BD TMH CBAT	TMB BF TMB CBAT	DBN BE XDN CBIMD	TM AC TFMW CBAT	TMH AD TFMH CBAT	TMB AF TFMB CBAT	BPCS AE UV UCBLP
10	LD 88 LCW LDPPU	LDH 89 LCH LDPPU	LDB 8B LCB LDPPU	VPTZ 8A VPTZ UCT	ST 98 SCW STPTP	STH 99 SCH STPTP	STB 9B SCB STPTP	TROL 9A TROL CRTSR	TP B8 TPW CBAT	TPH B9 TPH CBAT	TPB BB TPB CBAT	DBZ BA XDZ CBIMD	TP A8 TFPW CBAT	TPH A9 TFPH CBAT	TPB AB TFPB CBAT	AA

(A)126211

Figure 4-103A. Ignore Indirect (IGI) Op-Codes



(A)124811

Figure 4-104. SWBD/NIR Unique Op-Code Groups

case), and indirect indicator (DB, the signal generated to indicate the instruction being loaded in the IR of the active VP is indirect) from PPCTL2. The PPCTL1 card responds with indexing controls for INDEXER(0, 1); an EA subject to modifications if an interrupt has occurred or a BRSM instruction is in the process of saving the PC, the VPR select lines ultimately used by the TN field indexer, and a modify stack parameter enable, all input to PCCARDA(0-7); a next instruction indicator (NINS) for PPCTL2; state class and step data test and data real information for the IR of the active VP on IRCARD(0-3). The PPCTL2 card uses MIR data; SWBD/NIR data from PCCARDA(0-7); the shift update indicator from CONTAU; the buffer available indicator (BA) from SWBSYNC; the next instruction indicator (NINS) from PPCTL1. The PPCTL2 card responds with D, DB, and IGI for PPCTL1; the inhibit CR file signal for CRCONT(0-3); the inhibit VPR load signal for VPRCONT; the op-code, DC, mode, object mode, next instruction location (NIL), interrupt, LFAF, and PPTN data test and data real information, the TN and R indexer source and destination enables, the EA byte enables, and the shift count for the IR of the active VP on IRCARD(0-3). In addition, PPCTL2 monitors interrupts and the CR protect enable from CRCELLY and data generated by the bit picker on PPAUCD(0-3). When an automatic or programmed interrupt occurs, the PPCTL2 card responds by setting the appropriate interrupt data real bit in the associated IR. At the conclusion of the instruction during which the interrupt occurred, the INTF bit of the same IR is set by PPCTL2 so the interrupt cycle is initiated and the interrupt can be acknowledged. When CRCELLY supplies a CR protect enable to PPCTL2 and PPCTL2 detects an attempt to write in one of the first 10_{16} CR's, a protect violation signal is directed back to CRCELLY for interrupt purposes. This same signal is also used in the development of the CR file and VPR load inhibit signals previously mentioned. The bit picker on PPAUCD(0-3) supplies PPCTL2 with the bit pick count on all four bytes of the data input to the arithmetic unit via the MDB. When a POLL instruction is being executed, PPCTL2 utilizes the MIR source data to select the desired byte for use on PPAUCD(0-3).

The MIRMRGB card of the PP is used as a maintenance tool to monitor the AUMIR merge, VPRMIR merge, and CRMIR merge formats of figure 4-79, in addition to the SWBC status file and priority queue information. The AUMIR merge format is supplied by CONTAU and consists of the AUMIR format developed by VPRCONT and the active VPC supplied by MLCTL. The VPRMIR merge format is supplied by VPRCONT, PCCTL, and CRBASE2. The first part of the format is identical to the VPRMIR input format developed by VPRCONT and PCCTL and the last three blocks of data contain the active VPC being used by the VPRCONT, PCCTL, and CRBASE2 cards, respectively. The three words of CRMIR merge format are supplied by the CRCONT(0-3) cards and are described in detail in the CRCONT portion of the CR file detailed theory. The monitoring of the SWBSYNC data is controlled by the maintenance logic and gives the operator the capability of

viewing the contents of any one status file entry and four priority queue entries of the same queue, all at one time. If desired, the priority queue entries may be replaced by the contents of the high and low input and output priority queue counters. The IRVPCODE format shown in figure 4-79 is not input to the MIRMRGB card, but contains the VPC used by IRCARD(0-3) and is developed by IRCARD(0-3).

4-156 DETAILED TRANSFER TABLE ANALYSIS. The remainder of the detailed theory on the control section of the PP consists of a step-by-step description of execution of each of the instruction groups in the PP repertoire. Each instruction group description is accompanied by a unique transfer table (appendix A) and references to control signals presented in the detailed block diagrams of figures 4-105 through 4-108. Separate descriptions are presented for the instruction groups that are valid in both CM and ROM. The following abbreviations are used throughout the transfer tables and are listed here for reference.

- BA — Buffer available is set to indicate the status file entry for the active VP is ready to accept another CM request.
- BT — Branch taken is set to indicate the branch in question is to be taken.
- D — Dependency is set to indicate the current instruction must complete execution before the SWBD/NIR instruction can be indexed.
- DB — Next indirect is set to indicate the instruction being loaded in the IR of the active VP is indirect.
- DC — The IR DC bit is set when the current instruction in the MIR terminates and the DB bit is true.
- EXCM — EXCM is set to indicate the current instruction is execute central memory.
- IGI — Ignore indirect is set to indicate the next instruction is illegal for the indirect case.
- INT — Interrupt is set to indicate an automatic or programmed interrupt has occurred.
- INTF — The IR INTF bit is set at the termination of the current instruction if an interrupt has occurred during the instruction.
- KNOOP — KNOOP is set to indicate the next instruction is no-op.
- LFAF — The IR LFAF bit is set to indicate the base value involved in the LDMB indirect instruction is supplied by CM rather than a VPR.

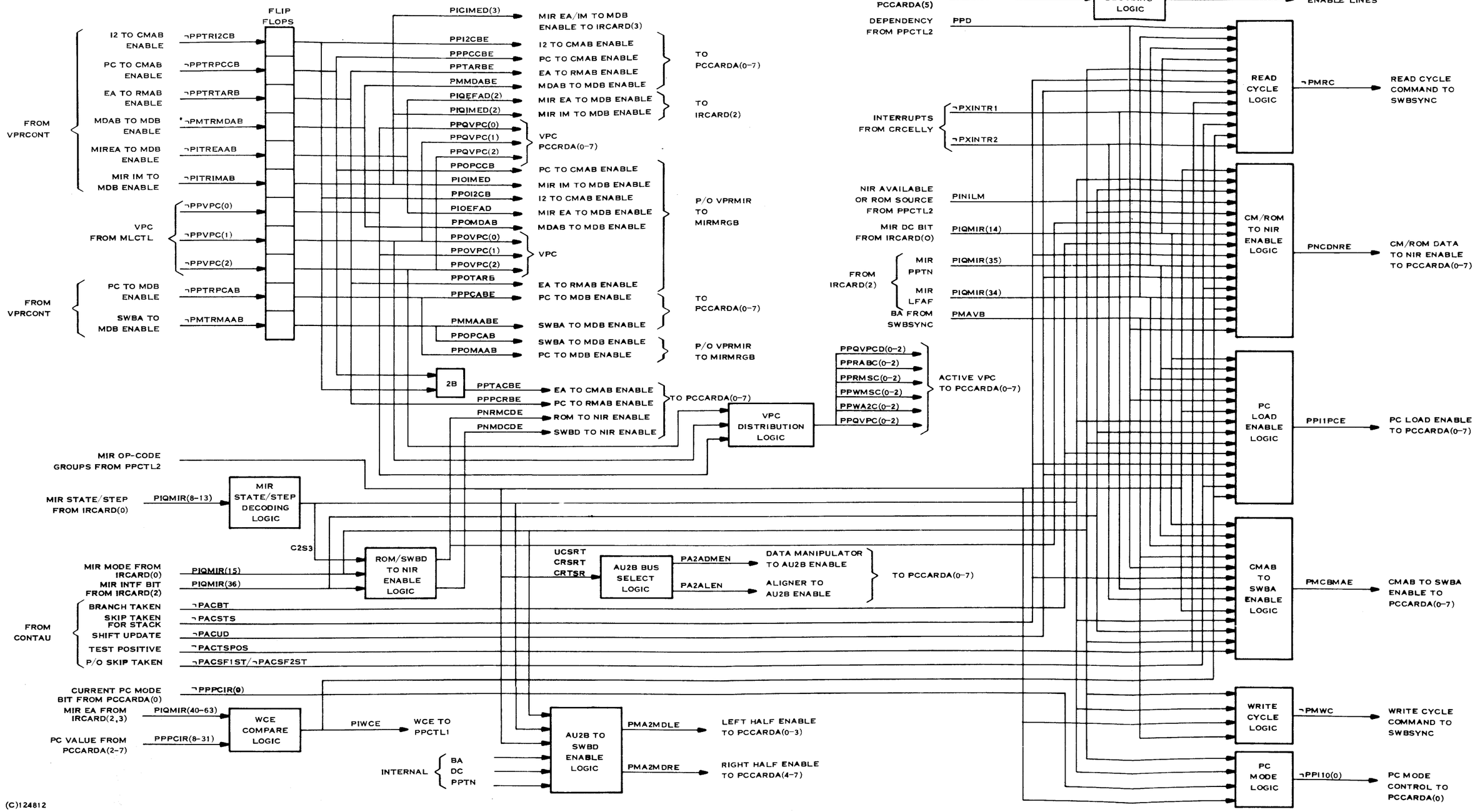


Figure 4-105. PCCTL Detailed Block Diagram

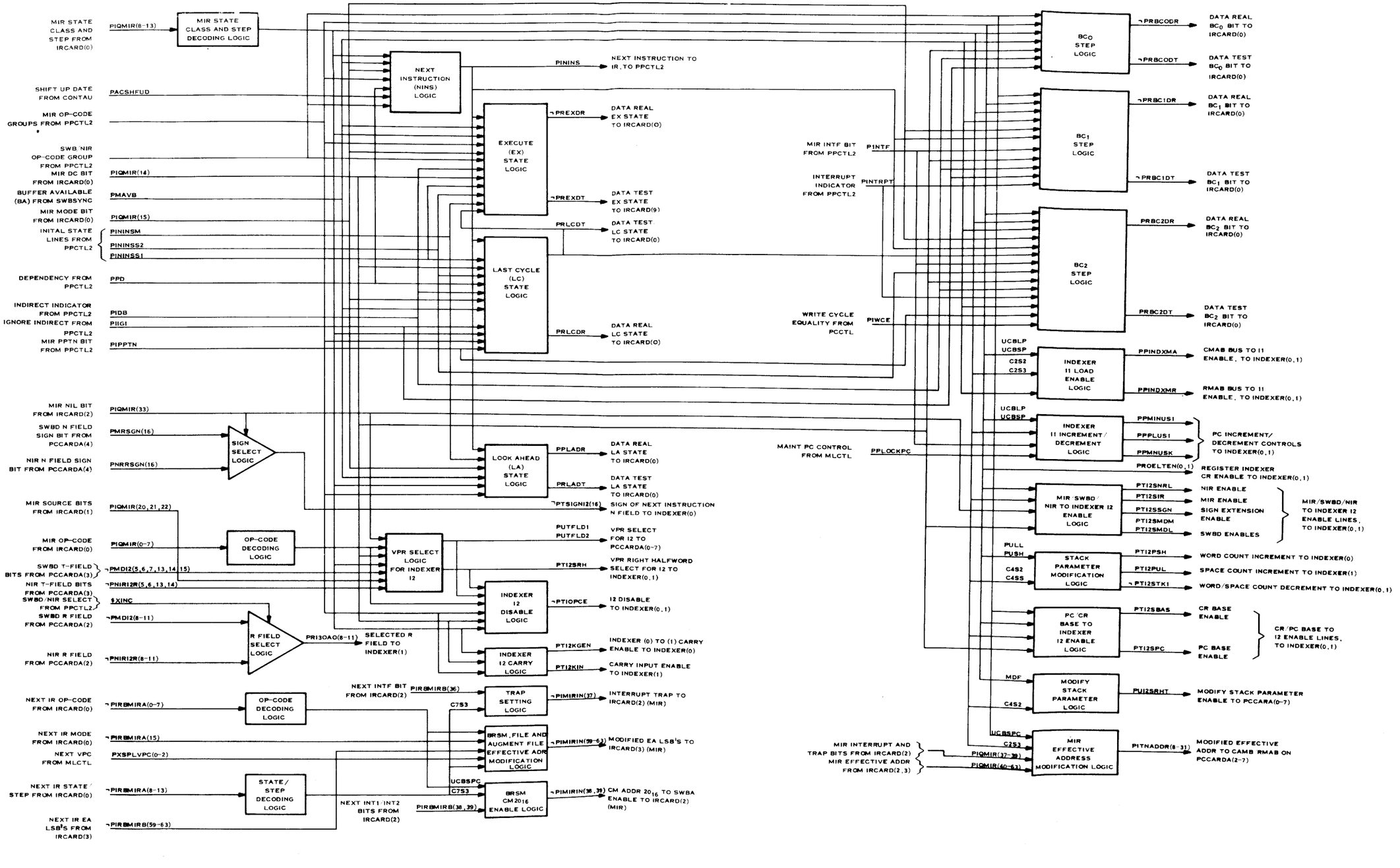


Figure 4-106. PPCTL1 Detailed Block Diagram

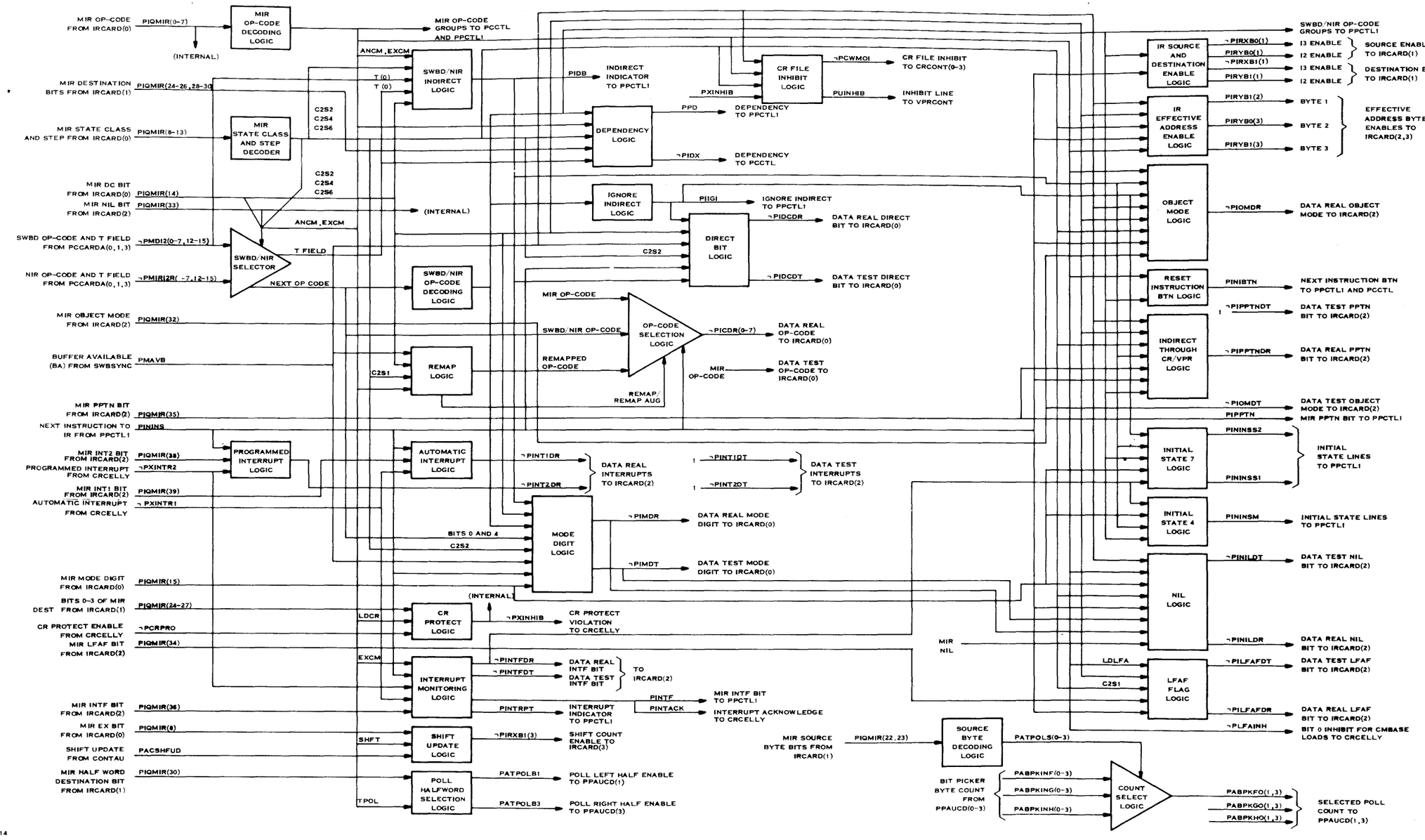


Figure 4-107. PPCTL2 Detailed Block Diagram

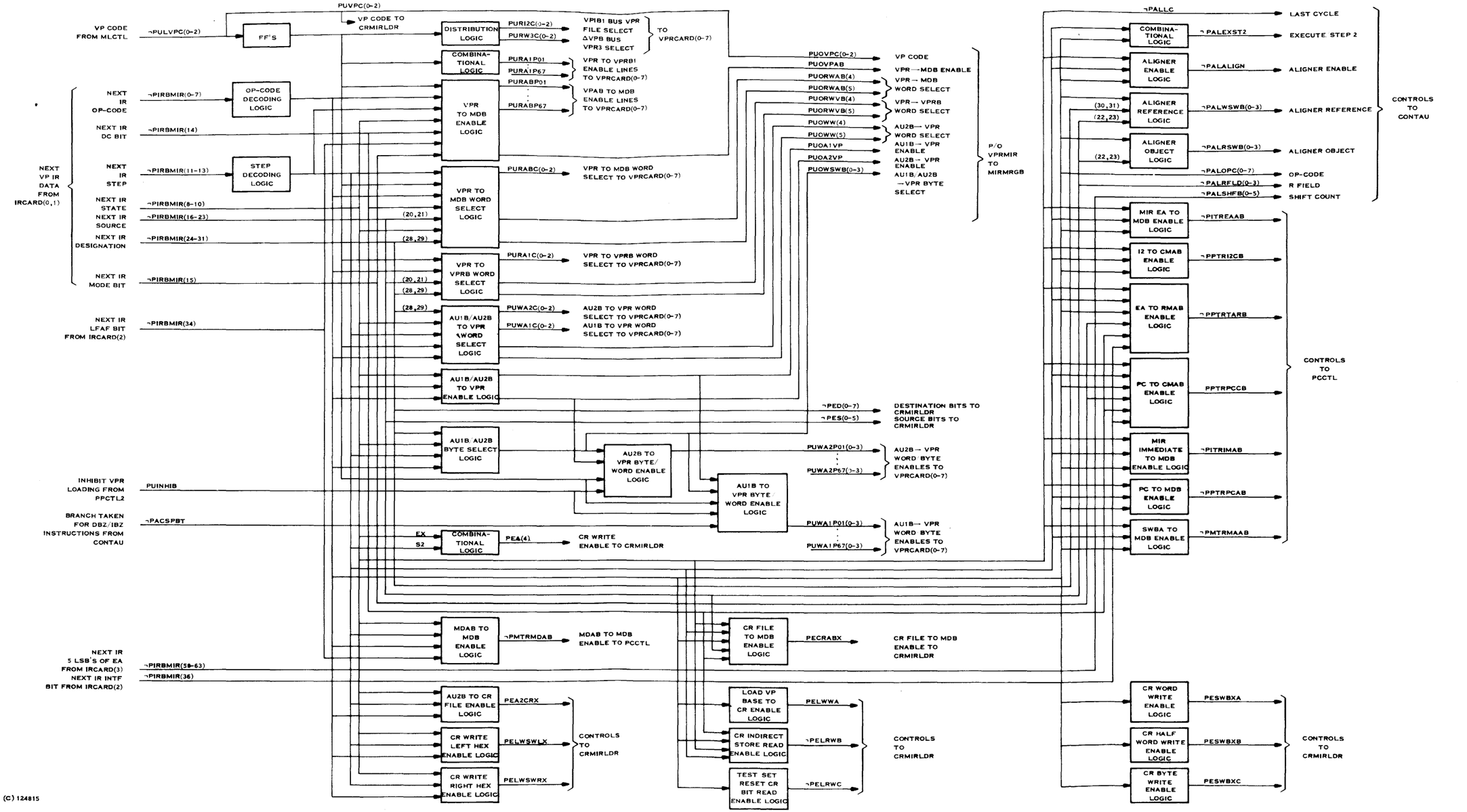


Figure 4-108. VPRCONT Detailed Block Diagram

- M — Mode is set to indicate the current instruction source is CM and reset to indicate ROM.
- NIBTN — Next instruction BTN is set to indicate the current unconditional branch and load PC instruction is followed by a PC relative branch instruction.
- NIL — Next instruction location is set to indicate the source of the next instruction is the NIR and reset to indicate SWBD.
- NINS — Termination is set to indicate completion of the current instruction.
- PPTN — The IR PPTN bit is set to indicate the instruction is indirect through a CR or VPR.
- RC — Read command is set to indicate the initiation of a CM read cycle.
- ST — Skip taken is set to indicate the next instruction is to be skipped.
- UD — Shift update is set to indicate additional data shift is necessary to complete the shift specified by a shift instruction.
- WC — Write command is set to indicate the initiation of a CM write cycle.
- WCE — Write cycle equality is set to indicate the EA of the current instruction is identical to the address of the next instruction to be executed.

Prior to each step description under all of the instruction groups, the IR containing the instruction that is the subject of the description is enabled into the MIR at the beginning of the command execution time (this signals the start of the active period for the VP associated with the IR). During the active period, the same IR is updated with data real, data test, TN field indexer, R field indexer, and shift count information necessary to carry out complete execution of the original instruction or initiation of a new instruction. When a VP register (PC, SWBD, IR, etc.) is mentioned in a step description, the register associated with the active VP is the one being referenced. The state class (EX, LA, and LC) and step (BC_0, BC_1, and BC_2) data in the transfer tables do not represent the actual IR contents but reflect the state class and step definitions presented in the IR portion of the VP detailed theory. Table 4-3 shows the relationships of the state class and step definitions to the actual inputs to (from PPCTL1) and contents of the IR. Table 4-4 contains the instruction groups (stores, loads, etc.), the subgroup mnemonics associated with each group, and the paragraph number associated with each subgroup transfer table analysis.

Table 4-3. State Class and Step Defined/Actual Relationships

State Class							Step							
Definition	IR Inputs			IR Contents			BC	Definition	IR Inputs			IR Contents		
	¬PREX D(D) R(T)	PRLA D(D) R(T)	PRLC D(D) R(T)						¬PRBC 0 D(D) R(T)	¬PRBC 1 D(D) R(T)	PRBC 2 D(D) R(T)			
0 1 0	1	1	0	0 0 1			1	0 0 0	1	1	0	0 0 1		
0 1 1	1	1	1	0 0 0			2	0 0 1	1	1	1	0 0 0		
1 0 0	0	0	0	1 1 1			3	0 1 0	1	0	0	0 1 1		
1 1 1	0	1	1	1 0 0			4	0 1 1	1	0	1	0 1 0		
							5	1 0 0	0	1	0	1 0 1		
							6	1 0 1	0	1	1	1 0 0		
							7	1 1 0	0	0	0	1 1 1		
							8	1 1 1	0	0	1	1 1 0		

Table 4-4. Instruction Transfer Table Analysis Index

Instruction Groups	Subgroup Mnemonic/Definition	Paragraph Number
Store	KSTCM/Store Word to CM	4-157
	KSTHCM/Store Halfword to CM	4-158
	KSTPTP/Store Register to Register	4-159
	KSTUF/Store UPR File	4-160
Load	KLDCM/Load From CM	4-161
	KLDIM/Load Immediate	4-162
	KLDPPU/Load Register to Register	4-163
	KLDUF/Load VPR File	4-164
	KLDLFA/Load CM Base Register	4-165
Add/Subtract	KCMAU/Add/Subtract CM to/from VPR	4-166

Table 4-4. Instruction Transfer Table Analysis Index (Continued)

Instruction Groups	Subgroup Mnemonic/Definition	Paragraph Number
Add/Subtract (Continued)	KIMAU/Add/Subtract Immediate to/from VPR	4-167
	KUAU/Add/Subtract VPR to/from VPR	4-168
Logical	KCMLOU/Logical CM to VPR	4-169
	KIMLOU/Logical Immediate to VPR	4-170
	KPPULOU/Logical VPR/CR to VPR	4-171
Shift	KSHFT/Shift	4-172
CR Bit Test/ Control	KUCSRT/Set/Reset CR VP Flag	4-173
	KUCT/Test CR VP Flag and Skip	4-174
	KCRSRT/Set/Reset CR Bits	4-175
	KCRTSRT/Test CR Bits, Set/Reset and Skip	4-176
	KCRLO/Test CR Bits and Skip	4-177
Poll	KTPOL/Test Poll Bits	4-178
Compare	KSKUCM/Compare CM to VPR	4-179
	KSKUIM/Compare Immediate to VPR	4-180
	KSKUPPU/Compare VPR/CR to VPR	4-181
Branch	KCBAT/Arithmetic Conditional Branch	4-182
	KCBIMDT/Increment/Decrement Conditional Branch	4-183
	KUCB/Unconditional Branch	4-184
	KUCBLPC/Unconditional Branch and Load PC	4-185
	KUCBSPC/Unconditional Branch to ROM, Save PC	4-186

Table 4-4. Instruction Transfer Table Analysis Index (Continued)

Instruction Groups	Subgroup Mnemonic/Definition	Paragraph Number
Stack	KPUSH/Push Stack	4-187
	KPULL/Pull Stack	4-188
	KMDF/Modify Stack	4-189
Miscellaneous	KEXCM/Execute CM	4-190
	KLEA/Load Effective Address	4-191
	KNDIREC/Indirect Cycle	4-192
	KNOOP/No Operation	4-193
	INTRPT/Interrupt Cycle	4-194

4-157 Store Word to CM (KSTCM). This instruction group stores the R field specified VPR or CR word to CM at the EA developed by the T and N fields. Execution of the KSTCM instruction group originating from CM is shown in the transfer table on page 1 of appendix A.

Step 1 The multistep CM source KSTCM instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability and the word to be stored is from a VPR, the appropriate VPAB to MDB enable (PURABP01, PURABP23, PURABP45, or PURABP67) and the VPR to MDB word select (PURABC(0-2)) from VPRCONT enable the desired VPR to the MDB. If the word to be stored is from a CR, the MIR source bits (¬PES(0-5)) routed through VPRCONT are used to select the CR word to be read and the CR file to MDB enable (PECRABX from VPRCONT) transfers the desired CR to the MDB. The aligner enable (PA2ALEN) from PCCTL transfers the VPR/CR over AU2B and the AU2B to SWBD enables (PMA2MDLE and PMA2MDRE) from PCCTL permit completion of the VPR/CR to SWBD transfer. At the same time, the MIR EA is passed through PPCTL1 and enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL; the SWBD is transferred over CMDB to the NIR via the SWBD to NIR enable (PNMDCDE) from PCCTL (to make room for the

VPR/CR); the data real NIL bit from PPCTL2 is set in the IR to indicate the next instruction is in the NIR; the MIR EA is indexed by one and input to the IR for the WCE test by enabling the MIR EA to the TN field indexer (via PTI2SIR on PPCTL1) and generating the carry input enable (PTI2KIN on PPCTL1); a write request (¬PMWC) is made from PCCTL to write the desired VPR/CR to the CM location specified by the MIR EA; and the state class and step data real logic on PPCTL1 inputs state class 3, step 1 to the IR.

Step 2 When the BA signal from SWBSYNC indicates buffer availability but the WCE logic on PCCTL evaluates true, the SWBD (containing the stored VPR/CR) is transferred to the NIR via the SWBD to NIR enable (PNMDCDE) and NIR load enable (PNCDNRE) from PCCTL. The state class and step data real logic on PPCTL1 inputs state class 3, step 2 to the IR (step 3 of this instruction group). If WCE is not true, the PP three-level pipe is advanced one level, with the aid of the NINS (PININS) enable from PPCTL1, as described in subparagraphs (1) through (5) below:

(1) The NIR-to-IR transfer is accomplished by enabling the following:

- The NIR op-code through the data real logic on PPCTL2 to the IR.
- The developed data real state class and step on PPCTL1 to the IR.
- The developed data real DC bit, M digit, and flags on PPCTL2 to the IR.
- The outputs of the TN and R field indexers to the source and destination fields of the IR as determined by the source enables (¬PIRXB0(1) for R and PIRYB0(1) for TN) and destination enables (¬PIRXB1(1) for R and PIRYB1(1) for TN) supplied by PPCTL2.
- The output of the TN field indexer or shift update logic on CONTAU to the EA of the IR as determined by the TN field indexer byte enables (PIRYB1(2), PIRYB0(3), and PIRYB1(3)) or the shift count enable (¬PIRXB1(3)) supplied by PPCTL2.

(2) The PC is transferred to the SWBA by the developed PC to CMAB enable (PPPCCBE) and CMAB to SWBA enable (PMCBMAE) from PCCTL.

(3) The PC is incremented by one by first enabling the PC to the PC indexer (via PPINDXMA from PPCTL1), then supplying the PC indexer with the increment by one signal (PPPLUS1 from PPCTL1), and finally enabling the PC indexer result to the PC via PPI1PCE from PCCTL.

(4) The NIL bit of the IR is zeroed via the data real logic on PPCTL2 to indicate the next instruction is in the SWBD.

(5) The PCCTL card issues a read request (¬PMRC) to fetch the next instruction.

Step 3 The PP three-level pipe is advanced one level as described in step 2 of this instruction group using the instruction retrieved from the VPR/CR in step 1.

Execution of the KSTCM instruction group originating from ROM is shown in the transfer table on page 5 of appendix A.

Step 1 The single step ROM source KSTCM instructions begin execution in state class 7, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the desired VPR/CR is transferred to the SWBD, the MIR EA is transferred to the SWBA, and a write request is issued, all as described in step 1 of the CM source KSTCM instruction group. At this same time, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exceptions: the PC is applied to RMAB and ROM instead of the SWBA via the PC to RMAB enable (PPPCRBE) from PCCTL (the PC to CMAB enable is still used, however, because of the PC indexing operation) and the NIL bit of the IR is set via the data real logic on PPCTL2 to indicate the next instruction is in the NIR.

4-158 Store Halfword to CM (KSTHCM). This instruction group stores a VPR or CR halfword, specified by the R field, to the left or right half of the CM location developed by the T and N fields. Execution of the KSTHCM instruction group originating from CM is shown in the transfer table on page 2 of appendix A.

Step 1 The multistep CM source KSTHCM instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL; the SWBD is saved in the NIR via the SWBD to NIR enable (PNMDCDE) from PCCTL; the IR NIL bit is set via the data real logic on PPCTL2 to indicate the next instruction is in the NIR; a read request (¬PMRC) is made from PCCTL to read the word addressed by the MIR EA; and the state class and step data real logic on PPCTL1 inputs state class 4, step 2 to the IR.

Step 2 When the BA signal from SWBSYNC indicates buffer availability and the halfword to be stored is from a VPR, the appropriate VPAB to MDB enable (PURABP01, PURABP23, PURABP45, or PURABP67) and VPR to MDB word select (PURABC(0-2)) from VPRCONT enable the VPR containing the desired halfword to the MDB. If the halfword to be stored is from a CR, the MIR source bits (¬PES(0-5)) from VPRCONT are used to select the CR containing the desired halfword and the CR file to MDB enable (PEACRABX) from VPRCONT transfers the selected CR to the MDB. The aligner enable (¬PALALIGN from VPRCONT) combines with the aligner object (¬PALRSWB(0-3) from VPRCONT) and aligner reference (¬PALWSWB(0-3) from VPRCONT) to rotate, if necessary, and transfer the MDB word to the AU2B. (Rotation is necessary if the register halfword is to be stored in the opposite halfword of the CM destination.) The appropriate AU2B to SWBD enable (PMA2MDLE for left halfword stores and PMA2MDRE for right halfword stores) from PCCTL transfers the VPR/CR halfword to the SWBD to complete the VPR/CR to SWBD transfer. At the same time, the MIR EA from PPCTL1 is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL, the MIR EA is indexed by one and input to the IR for the WCE test by enabling the MIR EA to the TN field indexer (via PTI2SIR on PPCTL1) and generating the carry input enable (PTI2KIN on PPCTL1), a write request (¬PMWC) is made from PCCTL to write the register and remaining CM halfwords to the CM location specified by the MIR EA, and the state class and step data real logic on PPCTL1 inputs state class 3, step 1 to the IR.

Step 3 Identical to step 2 of the CM source KSTCM instruction group.

Step 4 Identical to step 3 of the CM source KSTCM instruction group.

Execution of the KSTHCM instruction group originating from ROM is shown in the transfer table on page 6 of appendix A.

Step 1 The multistep ROM source KSTHCM instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL, the state class and step data real logic on PPCTL1 inputs state class 7, step 1 to the IR, and a read request (¬PMRC) is made from PCCTL to read the word addressed by the MIR EA.

Step 2 When the BA signal from SWBSYNC indicates buffer availability, the desired VPR/CR halfword is transferred to the SWBD, the MIR EA is transferred to the SWBA, and a write request is made, all as described in step 2 of the CM source KSTHCM instruction group. In addition, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exceptions: the PC is applied to RMAB and ROM instead of the SWBA via the PC to RMAB enable (PPPCRBE) from PCCTL (the PC to CMAB enable is still used, however, because of the PC indexing operation) and the NIL bit of the IR is set via the data real logic on PPCTL2 to indicate the next instruction is in the NIR.

4-159 Store Register-to-Register (KSTPTP). This group of instructions stores a VPR or CR word, halfword, or byte, as specified by the R field, to the VPR or CR word, halfword, or byte specified by the T and N fields. Execution of the KSTPTP instruction group originating from CM is shown in the transfer table on page 3 of appendix A.

Step 1 The single step CM source KSTPTP instructions begin execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability but the D logic on PPCTL2 indicates the next instruction depends on the current instruction, the VPR or CR containing the data to be transferred is enabled to both

AU1B and AU2B as described in step 2 of the CM source KSTHCM instruction group. If the data is to be stored in a VPR, the appropriate AU1B to VPR byte enables (PUWA1P01(0-3), PUWA1P23(0-3), PUWA1P45(0-3), or PUWA1P67(0-3)) combine with the AU1B to VPR word select (PUWA1C(0-2)), both from VPRCONT, to transfer the VPR/CR data to the desired VPR. If the data is to be stored in a CR, the AU2B to CR file enable (PEA2CRX), destination bits (¬PED(0-7)), and word (PESWBXA), halfword (PESWBXB), byte (PESWBXC), and hex (PELWSWLX for left hex and PELWSWRX for right hex) enables from VPRCONT combine to transfer the VPR/CR data to the desired CR. The state class and step data real logic on PPCTL1 inputs state class 3, step 2 to the IR. If the next instruction does not depend on the current instruction (¬D), the previously described VPR/CR to VPR/CR transfer is enabled and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: the SWBD is the source of the next instruction so it must be used in the indexing cycle.

Step 2 The PP three-level pipe is advanced one level using the SWBD as the source of the next instruction.

Execution of the KSTPTP instruction group originating from ROM is shown in the transfer table on page 7 of appendix A, and is identical to the KSTPTP instruction group except for the source of the next instruction (ROM) used in the three-level pipe advance and the setting of the NIL bit to indicate NIR.

4-160 Store VPR File (KSTUF). This instruction group stores the VPR file in four consecutive CM locations beginning at the EA developed by the T and N fields. If the EA is not a multiple of four, it is forced to a multiple of four. Execution of the KSTUF instruction group originating from CM is shown in the transfer table on page 4 of appendix A.

Step 1 The multistep CM source KSTUF instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the appropriate VPAB to MDB enable (PURABP01 for VP0 and VP1, PURABP23 for VP2 and VP3, PURABP45 for VP4 and VP5, and PURABP67 for VP6 and VP7) and VPR to MDB word select (PURABC(0-2)) from VPRCONT enable VPR0 to the MDB. The aligner enable (¬PALALIGN) from VPRCONT transfers VPR0 to AU2B and the AU2B to SWBD enables (PMA2MDLE and

PMA2MDRE) from PCCTL permit completion of the VPR0 to SWBD transfer. The MIR EA, with the last two bits set to zero by the ¬PIMIRIN(62, 63) lines from PPCTL1, is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL, the SWBD is transferred over CMDB to the NIR via the SWBD to NIR enable (PNMDCDE) from PCCTL in order to make room for VPR0, the IR NIL bit is set via the data real logic on PPCTL2 to indicate the next instruction is in the NIR, a write request (¬PMWC) is made from PCCTL to write VPR0 to CM, and the state class and step data real logic on PPCTL1 inputs state class 4, step 2 to the IR.

Step 2 — When the BA signal from SWBSYNC indicates buffer availability and no WCE exists (as indicated by PCCTL), VPR1 is written to CM at the location specified by the MIR EA (with the last two bits set to one) in a manner similar to that described in step 1. If the WCE logic evaluates true (the MIR EA with the last two bits zeroed is identical to the address of the next instruction), VPR1 is written to CM as mentioned and VPR0 is saved in the NIR via the SWBD to NIR enable (PNMDCDE) from PCCTL. In either case (¬WCE or WCE), the state class and step of the IR are advanced to four and three, respectively.

Step 3 — When the BA signal from SWBSYNC indicates buffer availability and no WCE exists, VPR2 is written to CM at the location specified by the MIR EA (with the last two bits set to two) in a manner similar to that described in step 1. If the WCE logic evaluates true (the MIR EA with the last two bits set to one is identical to the address of the next instruction), VPR2 is written to CM as mentioned and VPR1 is saved in the NIR via the SWBD to NIR enable (PNMDCDE) from PCCTL. In either case, the state class and step of the IR are both advanced to four.

Step 4 — When the BA signal from SWBSYNC indicates buffer availability and no WCE exists, VPR3 is written to CM at the location specified by the MIR EA (with the last two bits set to three) and the MIR EA is indexed by one and input to the IR for the next WCE test by enabling the MIR EA to the TN field indexer (via PTI2SIR on PPCTL1) and generating the carry input enable (PTI2KIN on PPCTL1). If the WCE logic evaluates true (the MIR EA with the last two bits set to two is identical to the address of the next instruction), VPR3 is written to CM as mentioned, VPR2

is saved in the NIR via the PNMDCDE enable from PCCTL, and the MIR EA is indexed by one and input to the IR as previously described. In either case (¬WCE or WCE), the state class and step of the IR are advanced to three and one, respectively.

Step 5 When the BA signal from SWBSYNC indicates buffer availability and no WCE exists, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group. When WCE does exist, VPR3 is saved in the NIR via the PNMDCDE enable from PCCTL and the state class and step of the IR are advanced to three and two, respectively.

Step 6 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Execution of the KSTUF instruction group originating from ROM is shown in the transfer table on page 8 of appendix A.

Step 1 The multistep ROM source KSTUF instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, VPR0 is written to CM and the state class and step of the IR are updated as described in step 1 of the CM source KSTUF instruction group.

Step 2 When the BA signal from SWBSYNC indicates buffer availability, VPR1 is written to CM and the state class and step of the IR are updated as described in step 2 of the CM source KSTUF instruction group.

Step 3 When the BA signal from SWBSYNC indicates buffer availability, VPR2 is written to CM as described in step 3 of the CM source KSTUF instruction group and the IR state class and step are advanced to seven and one, respectively, by the data real logic on PPCTL1.

Step 4 When the BA signal from SWBSYNC indicates buffer availability, VPR3 is written to CM as described in step 4 of the CM source KSTUF instruction group and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exceptions: the PC is applied to RMAB and ROM instead of the SWBA via the PC to RMAB enable (PPPCRBE) from PCCTL (the PC to CMAB enable is still

used, however, because of the PC indexing operation) and the IR NIL bit is set via the PPCTL2 data real logic to indicate the next instruction is in the NIR.

4-161 Load from CM (KLDCM). This group of instructions loads the word or halfword specified by the T and N fields into the register specified by the R field. Execution of the KLDCM instruction group originating from CM is shown in the transfer table on page 9 of appendix A.

Step 1 The multistep CM source KLDCM instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL, the SWBD is transferred over CMDB to the NIR via the PNMDCDE enable from PCCTL to make room for the CM word to be read, the IR NIL bit is set to one by the data real logic on PPCTL2 to indicate the next instruction is in the NIR, a read request (¬PMRC) is issued from PCCTL to retrieve the CM word addressed by the MIR EA, and the state class and step of the IR are advanced to seven and two, respectively, by the data real logic on PPCTL1.

Step 2 When the BA signal from SWBSYNC indicates buffer availability but the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing, the SWBD to MDB enable (PMMDABE) and word select (PPRABC(0-2)) from PCCTL combine to enable the active SWBD to the MDB, and the aligner enable (¬PALALIGN from VPRCONT) transfers the MDB to AU1B and AU2B (if a CM halfword is to be loaded in the opposite half of the destination register, the aligner object (¬PALRSWB(0-3)) and aligner reference (¬PALWSWB(0-3)) combine to rotate the CM MDB word before it is transferred to AU1B and AU2B). If the CM data is to be stored in a VPR, the appropriate AU1B to VPR byte enables (PUWA1P01(0-3) for VP0 and VP1, PUWA1P23(0-3) for VP2 and VP3, PUWA1P45(0-3) for VP4 and VP5, and PUWA1P67(0-3) for VP6 and VP7) combine with the AU1B to VPR word code (PUWA1C(0-2)), both from VPRCONT, to complete the SWBD to VPR transfer. If the CM data is to be stored in a CR, the AU2B to CR file enable (PEA2CRX), destination bits (¬PED(0-7)), and word (PESWBXA) and

halfword (PESWBXB) enables from VPRCONT combine to complete the SWBD to CR transfer. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If there is no dependency (¬D) between the next and current instructions, the SWBD is transferred to the desired register as described above and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Step 3 — The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Execution of the KLDCM instruction group originating from ROM is shown in the transfer table on page 14 of appendix A.

Step 1 — The multistep ROM source KLDCM instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL, a read request (¬PMRC) is issued from PCCTL to retrieve the CM word addressed by the MIR EA, and the state class and step of the IR are advanced to seven and two, respectively, by the data real logic on PPCTL1.

Step 2 — Identical to step 2 of the CM source KLDCM instruction group, with the following exception: the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not occur.

Step 3 — Identical to step 3 of the CM source KLDCM instruction group except for the PP three-level advance difference due to the ROM source.

4-162 Load Immediate (KLDIM). This group of instructions loads the immediate operand (word, halfword, or byte) indicated by the T and N fields into the register specified by the R field. Execution of the KLDIM instruction group originating from CM is shown in the transfer table on page 10 of appendix A.

Step 1 — The single step CM source KLDIM instructions begin execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability but the dependency logic on PPCTL2 indicates the

immediate operand to register transfer must be complete before indexing, the MIR immediate to MDB enables (PIQIMED(2) and PIQIMED(3)) from PCCTL transfer the immediate operand to the MDB, the immediate operand is transferred to the desired VPR/CR from the MDB in the manner described for the CM word in step 2 of the CM source KLDCM instruction group, and the state class and step of the IR are advanced to three and two, respectively, by the data real logic on PPCTL1. If no dependency exists, the immediate operand to VPR/CR transfer mentioned above is enabled and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: the next instruction source is the SWBD rather than the NIR.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KLDIM instruction group originating from ROM is shown in the transfer table on page 15 of appendix A.

Step 1 Identical to step 1 of the CM source KLDIM instruction group, with the following exceptions: the BA signal from SWBSYNC is not necessary for instruction execution and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-163 Load Register-to-Register (KLDPPU). This group of instructions loads (word, halfword, or byte) the register specified by the T and N fields into the register specified by the R field. Execution of the KLDPPU instruction group originating from CM is shown in the transfer table on page 11 of appendix A.

Step 1 The single step CM source KLDPPU instructions begin execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability but the dependency logic on PPCTL2 evaluates true and the source of the word to be loaded is a VPR, the appropriate VPAB to MDB enable (PURABP01, PURABP23,

PURABP45, or PURABP67) and VPR to MDB word select (PURABC(0-2)) from VPRCONT enable the VPR containing the desired data to the MDB. If the source of the word to be loaded is a CR, the MIR source bits (¬PES(0-5)) from VPRCONT are used to select the CR containing the data to be loaded and the CR file to MDB enable (PEACRABX) from VPRCONT transfers the selected CR to the MDB. The aligner enable (¬PALALIGN from VPRCONT) combines with the aligner object (¬PALRSWB(0-3) from VPRCONT) and aligner reference (¬PALWSWB(0-3) from VPRCONT) to align the MDB data (alignment is necessary if the halfword or byte on the MDB is to be loaded in a different halfword or byte of the destination register) and transfer it to AU1B and AU2B. The VPR/CR data is now loaded in the destination VPR/CR as described in step 1 of the CM source KSTPTP instruction group and the IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If dependency does not exist, the register to register transfer is enabled as described above and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: the next instruction source is the SWBD rather than the NIR.

Step 2 — The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KLDPPU instruction group originating from ROM is shown in the transfer table on page 16 of appendix A.

Step 1 — Identical to step 1 of the CM source KLDPPU instruction group, with the following exceptions: the BA signal from SWBSYNC is not necessary for instruction execution and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist.

Step 2 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-164 Load VPR File (KLDUF). This instruction group loads the VPR file with the contents of four consecutive CM locations, the first of which is specified by the EA developed by the T and N fields. If the EA is not a multiple of four, it is forced to a multiple of four during execution. The KLDUF instruction group originating from CM is shown in the transfer table on page 12 of appendix A.

Step 1 The multistep CM source KLDUF instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA, with the last two bits set to zero by the ¬PIMIRIN(62,63) lines from PPCTL1, is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL, the SWBD is transferred over CMDB to the NIR via the PNMDCDE enable from PCCTL to make room for the first CM word to be read, the IR NIL bit is set to one by the data real logic on PPCTL2 to indicate the next instruction in the NIR, a read request (¬PMRC) is issued from PCCTL to retrieve the CM word addressed by the MIR EA, and the state class and step of the IR are advanced to four and two, respectively, by the data real logic on PPCTL1.

Step 2 When the BA signal from SWBSYNC indicates buffer availability, the SWBD (containing the CM word read) is transferred to VPR0 in a manner similar to that described in step 2 of the CM source KLDCM instruction group for SWBD to VPR transfers, the MIR EA with the last two bits set to one is transferred to the SWBA in a manner similar to that described in step 1, a read request (¬PMRC) is issued from PCCTL to retrieve the second word of the VPR file group, and the state class and step of the IR are advanced to four and three, respectively.

Step 3 Identical to step 2 except for the following: SWBD is transferred to VPR1, the third word of the VPR file group is retrieved from CM, and the state class and step of the IR are both advanced to four.

Step 4 Identical to step 2 except for the following: SWBD is transferred to VPR2, the fourth word of the VPR file group is retrieved from CM, and the state class and step of the IR are advanced to seven and two, respectively.

Step 5 When the BA signal from SWBSYNC indicates buffer availability but the next instruction is dependent (D) on the current instruction, the SWBD is transferred to VPR3 and the state class and step of the IR are advanced to three and two, respectively. If dependency does not exist, the SWBD is transferred to VPR3 and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Step 6 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Execution of the KLDUF instruction group originating from ROM is shown in the transfer table on page 17 of appendix A.

Setp 1 Identical to step 1 of the CM source KLDUF instruction group except for the following: the SWBD to NIR transfer and the setting of the IR NIL bit are not necessary because the instruction source is ROM.

Step 2 Identical to step 2 of the CM source KLDUF instruction group.

Step 3 Identical to step 3 of the CM source KLDUF instruction group.

Step 4 Identical to step 4 of the CM source KLDUF instruction group.

Step 5 Identical to step 5 of the CM source KLDUF instruction group except for the following when dependency does not exist: the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

Step 6 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-165 Load CM Base Register (KLDLFA). This instruction loads the CM base register associated with the active VP with the three least significant bytes of the VPR specified by the T and N fields. If the MIR LFAF bit is set, the source of the load is the CM location specified by the T and N fields. Execution of the KLDLFA instruction originating from CM is shown in the transfer table on page 13 of appendix A.

Step 1 The multistep CM source KLDLFA instruction begins execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. Initially, the IR LFAF flag is checked to determine the source of the load. If the flag is not set (¬LFAF), the IR state class and step are advanced to seven and two, respectively, by the data real logic on PPCTL1. If the flag is set (LFAF), the availability of the SWB is checked. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE)

on PCCTL, the SWBD is transferred over CMDB to the NIR via the PNMDCDE enable from PCCTL to make room for the CM word to be read, the IR NIL bit is set to one by the data real logic on PPCTL2 to indicate the next instruction is in the NIR, a read request (¬PMRC) is issued from PCCTL to retrieve the CM word addressed by the MIR EA, and the IR state class and step are advanced to seven and two, respectively, by the data real logic on PPCTL1.

Step 2 When the BA signal from SWBSYNC indicates buffer availability but the next instruction is dependent on the current instruction (D) and the MIR LFAF bit is not set, the appropriate VPAB to MDB enable (PURABP01, PURABP23, PURABP45, or PURABP67) and VPR to MDB word select (PURABC(0-2)), both from VPRCONT, enable the VPR containing the desired data to the MDB. The aligner enable (¬PALALIGN from VPRCONT) transfers the selected VPR to AU2B and the AU2B-to-CR file enable (PEA2CRX), load CM base enable (PELWWA), and word enable (PESWBXA), all from VPRCONT, combine with the byte zero inhibit for CM base load signal (¬PLFAINH) from PPCTL2 to complete the VPR to CM base register transfer. The state class and step of the IR are advanced to three and two, respectively, by the data real logic on PPCTL1. If dependency does not exist and the MIR LFAF bit is not set, the selected VPR data is transferred to the CM base register as previously mentioned and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group. If the next instruction is dependent on the current instruction and the MIR LFAF bit is set, the SWBD to MDB enable (PMMDABE) and word select (PPRABC(0-2)), both from PCCTL, combine to enable the SWBD to the MDB. The remainder of the data transfer, from MDB to CM base register, is identical to that previously mentioned in this step, as is the state class and step advance. If dependency does not exist and the MIR LFAF bit is set, the SWBD is transferred to the CM base register as previously mentioned and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Step 3 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Execution of the KLDLFA instruction group originating from ROM is shown in the transfer table on page 18 of appendix A.

Step 1 — Identical to step 1 of the CM source KLDLFA instruction except for the following: the SWBD to NIR transfer and the setting of the IR NIL bit are not necessary when LFAF is set because the instruction source is ROM.

Step 2 — Identical to step 2 of the CM source KLDLFA instruction except for the following when dependency does not exist; the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

Step 3 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-166 Add/Subtract CM To/From VPR (KCMAU). This group of instructions adds/subtracts the word or halfword operand specified by the T and N fields to/from the VPR contents specified by the R field. Execution of the KCMAU instruction group originating from CM is shown in the transfer table on page 19 of appendix A.

Step 1 — The multistep CM source KCMAU instruction group begins execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA is transferred to the SWBA, the SWBD is transferred to the NIR, the IR NIL bit is set, a read request is issued, and the IR state class and step are updated, all as described in step 1 of the CM source KLDLFA instruction.

Step 2 — When the BA signal from SWBSYNC indicates buffer availability but the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing, the SWBD and desired VPR are added/subtracted in the following manner:

- The appropriate VPR to VPRB1 enable (PURA1P01 for VP0 and VP1, PURA1P23 for VP2 and VP3, PURA1P45 for VP4 and VP5, or PURA1P67 for VP6 and VP7) and word select (PURA1C(0-2)), both from VPRCONT, combine to transfer the desired VPR to the Arithmetic Unit (AU).

- The SWBD to MDB enable (PMMDABE) and word select (PPRABC(0-2)), both from PCCTL, combine to enable the SWBD to the MDB, and therefore the AU.
- The aligner enable (¬PALALIGN), aligner reference (¬PALWSWB(0-3)), aligner object (¬PALRSWB(0-3)), and MIR op-code from VPRCONT are used by CONTAU to align the SWBD data, if necessary, (alignment is required if the CM and VPR halfwords involved are in opposite halves); add or subtract the SWBD and VPR as determined by the MIR op-code; and enable the result to AU1B.
- The appropriate AU1B to VPR byte enables (PUWA1P01(0-3), PUWA1P23(0-3), PUWA1P45(0-3), or PUWA1P67(0-3)) and word select (PUWA1C(0-2)), both from VPRCONT, combine to transfer the AU1B result to the VPR whose contents are added/subtracted.

The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When dependency does not exist, the contents of the R field specified VPR are replaced by the sum/difference of the VPR and SWBD as previously mentioned in this step and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Step 3 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Execution of the KCMAU instruction group originating from ROM is shown in the transfer table on page 22 of appendix A.

Step 1 Identical to step 1 of the CM source KCMAU instruction group except that the SWBD to NIR transfer and the setting of the IR NIL bit are not necessary because the instruction source is ROM.

Step 2 Identical to step 2 of the CM source KCMAU instruction group except that, when dependency does not exist, the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

Step 3 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-167 Add/Subtract Immediate to/from VPR (KIMAU). This group of instructions adds/subtracts the word, halfword, or byte immediate operand developed by the T and N fields to/from the VPR contents specified by the R field. Execution of the KIMAU instruction group originating from CM is shown in the transfer table on page 20 of appendix A.

Step 1 The single step CM source KIMAU instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, but the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing, the immediate operand and desired VPR are added/subtracted in the following manner: The desired VPR is transferred to the AU via VPRB as described in step 2 of the CM source KCMAU instruction group; the MIR immediate to MDB enables (PIQIMED(2) and PIQIMED(3)) from PCCTL transfer the immediate operand to the MDB; the aligner enable (¬PALALIGN), aligner reference (¬PALWSWB(0-3)), aligner object (¬PALRSWB(0-3)), and MIR op-code, all from VPRCONT, combine to align the immediate operand if necessary (alignment is required if the halfwords or bytes involved are in different parts of their respective words), add or subtract the immediate operand and VPR as determined by the MIR op-code, and enable the result to AU1B; the AU1B result is transferred to the source VPR as described in step 2 of the CM source KCMAU instruction group. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When dependency does not exist, the contents of the VPR specified by the R field are replaced by the sum/difference of the VPR and the immediate operand as previously mentioned in this step and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: the source of the next instruction is the SWBD rather than the NIR.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KIMAU instruction group originating from ROM is shown in the transfer table on page 23 of appendix A.

Step 1 Identical to step 1 of the CM source KIMAU instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary for instruction execution and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-168 Add/Subtract VPR to/from VPR (KUAU). This group of instructions adds/subtracts the VPR word, halfword, or byte specified by the T and N fields to/from the VPR specified by the R field. Execution of the KUAU instruction group originating from CM is shown in the transfer table on page 21 of appendix A.

Step 1 The single step CM source KUAU instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability but the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing, the two VPR quantities are added/subtracted in the following manner: The VPR specified by the R field (destination of the addition/subtraction) is transferred to the AU via VPRB as described in step 2 of the CM source KCMAU instruction group; the VPR specified by the T and N fields is input to the AU over the MDB via the appropriate VPAB to MDB enable (PURABP01, PURABP23, PURABP45, or PURABP67) and VPR to MDB word select (PURABC(0-2)) from VPRCONT; the two VPR quantities are added/subtracted in the AU as described in step 1 of the CM source KIMAU instruction group; the result of the arithmetic operation is input to the VPR specified by the R field as described in step 2 of the CM source KCMAU instruction group. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When dependency does not exist, the contents of the VPR specified by the R field are replaced by the sum/difference of the T and N field VPR and R field VPR as previously mentioned in this step. The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KUAU instruction group originating from ROM is shown in the transfer table on page 24 of appendix A.

Step 1 Identical to step 1 of the CM source KUAU instruction group, except that the BA signal from SWBSYNC is not necessary for instruction execution and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-169 Logical CM to VPR (KCMLOU). This group of instructions logically combines (AND, OR, EXCLUSIVE OR, or EQUIVALENCE) the word or half-word CM operand specified by the T and N fields with the contents of the VPR specified by the R field. The result replaces the original contents of the VPR. Execution of the KCMLOU instruction group originating from CM is shown in the transfer table on page 25 of appendix A.

Step 1 The multistep CM source KCMLOU instruction group begins execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA is transferred to the SWBA, the SWBD is transferred to the NIR, the IR NIL bit is set, a read request is issued, and the IR state class and step are advanced to seven and two, respectively, all as described in step 1 of the CM source KLDCM instruction group.

Step 2 When the BA signal from SWBSYNC indicates buffer availability but the dependency (D) logic on PPCTL2 indicates that the next instruction requires complete execution of the current instruction before indexing, the SWBD and desired VPR are logically combined in the following manner: The VPR and SWBD are input to the AU as described in step 2 of the CM source KCMAU instruction group; the AU performs the necessary alignment and executes the desired logical operation (this depends on the MIR op-code) as described in step 2 of the CM source KCMAU instruction group; the AU result is transferred to the VPR involved in the logical operation as described

in step 2 of the CM source KCMAU instruction group. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When dependency does not exist, the contents of the VPR specified by the R field are replaced by the result of the logical operation as previously mentioned in this step and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Step 3 — The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Execution of the KCMLOU instruction group originating from ROM is shown in the transfer table on page 28 of appendix A.

Step 1 — Identical to step 1 of the CM source KCMLOU instruction group except that the SWBD to NIR transfer and the setting of the IR NIL bit are not necessary because the instruction source is ROM.

Step 2 — Identical to step 2 of the CM source KCMLOU instruction group except that, when dependency does not exist, the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

Step 3 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-170 Logical Immediate to VPR (KIMLOU). This group of instructions logically combines (AND, OR, EXCLUSIVE OR, or EQUIVALENCE) the halfword or byte immediate operand specified by the T and N fields with the contents of the VPR specified by the R field. The result replaces the original contents of the VPR. Execution of the KIMLOU instruction group originating from CM is shown in the transfer table on page 26 of appendix A.

Step 1 — The single step CM source KIMLOU instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability but the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing, the immediate operand and desired VPR are logically combined in the following manner: The immediate operand and VPR are input to the AU as described in step 1 of the CM source KIMAU instruction group; the AU performs the necessary

alignment and executes the desired logical operation (this depends on the MIR op-code) as described in step 1 of the CM source KIMAU instruction group; the AU result is transferred to the VPR involved in the logical operation as described in step 2 of the CM source KCMAU instruction group. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When dependency does not exist, the contents of the VPR specified by the R field are replaced by the result of the logical operation as previously mentioned in this step. The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: the source of the next instruction is the SWBD rather than the NIR.

Step 2 — The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KIMLOU instruction group originating from ROM is shown in the transfer table on page 29 of appendix A.

Step 1 — Identical to step 1 of the CM source KIMLOU instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary for instruction execution and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist.

Step 2 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-171 Logical VPR/CR to VPR (KPPULOU). This group of instructions logically combines (AND, OR, EXCLUSIVE OR, or EQUIVALENCE) the word, halfword, or byte of the VPR/CR specified by the T and N fields with the VPR specified by the R field. The result replaces the original contents of the VPR specified by the R field. Execution of the KPPULOU instruction group originating from CM is shown in the transfer table on page 27 of appendix A.

Step 1 — The single step CM source KPPULOU instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability but the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing, the VPR/CR

specified by the T and N fields is logically combined with the VPR specified by the R field in the following manner: The VPR specified by the R field is input to the AU via VPRB as described in step 2 of the CM source KCMAU instruction group; if the T and N fields specify a VPR, it is input to the AU via the MDB as described in step 1 of the CM source KUAU instruction group; if the T and N fields specify a CR, the MIR source bits (¬PES(0-5)) from VPRCONT are used to select the CR containing the data for the logical operation and the CR file to MDB enable (PEACRABX) from VPRCONT transfers the selected CR to the MDB; the AU performs the necessary alignment and executes the desired logical operation (this depends on the MIR op-code) as described in step 1 of the CM source KIMAU instruction group; the AU result is transferred to the VPR specified by the R field as described in step 2 of the CM source KCMAU instruction group. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When dependency does not exist, the contents of the VPR specified by the R field are replaced by the result of the logical operation as previously mentioned in this step. In addition, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR.

Step 2 — The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KPPULOU instruction group originating from ROM is shown in the transfer table on page 30 of appendix A.

Step 1 — Identical to step 1 of the CM source KPPULOU instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary for instruction execution and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist.

Step 2 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-172 Shift (KSHFT). This group of instructions performs right and left logical, arithmetic, and cyclic shifts on the VPR word specified by the R field. The amount of shift is specified by the immediate operand developed

by the T and N fields. Execution of the KSHFT instruction group originating from CM is shown in the transfer table on page 31 of appendix A.

Step 1 The single step CM source KSHFT instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. The VPR is input to the AU via VPRB as described in step 2 of the CM source KCMAU instruction group and the MIR op-code and shift count from VPRCONT are input to the AU to control the shift. If shift update is necessary, as indicated by the PACSHFUD signal from CONTAU when it takes more than the current shift cycle to complete the desired shift, the AU responds by executing a shift of 1, 4, or 8 (or 16, if a cyclic shift is executing) and generating an updated shift count. The result of the shift is transferred to the VPR specified by the R field as described in step 2 of the CM source KCMAU instruction group and the updated shift count (¬PACSHFOB(0-5)) from CONTAU is input to the IR EA via the shift count enable (¬PIRXB1(3)) from PPCTL2. The IR state class and step are returned to seven and two, respectively, by the data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability and the shift update signal from CONTAU indicates only one cycle of shift update is remaining (¬UD), the last shift cycle is executed as previously mentioned in this step (except no shift count is returned to the IR) and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR. If the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing when the last shift cycle is to be executed (¬UD), the last shift cycle is executed as previously mentioned in this step and the IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KSHFT instruction group originating from ROM is shown in the transfer table on page 32 of appendix A.

Step 1 Identical to step 1 of the CM source KSHFT instruction group, with the following exceptions: the BA signal from

SWBSYNC is not necessary to advance the PP three-level pipe and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist at the last shift cycle.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-173 Set/Reset CR VP Flag (KUCSRT). This group of instructions sets/resets the flag bit in the CR byte specified by the T and N fields. The flag bit under consideration is determined by the number of the executing VP. Execution of the KUCSRT instruction group originating from CM is shown in the transfer table on page 32 of appendix A.

Step 1 The single step CM source KUCSRT instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability but the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing, the CR byte specified by the T and N fields is modified in the following manner: The MIR source bits (¬PES(0-5)) from VPRCONT are used to select the CR containing the desired byte and the CR file to MDB enable (PEACRABX) from VPRCONT transfers the selected CR to the MDB; the AU data manipulator uses the selected CR, the MIR op-code from VPRCONT, and the VPC from MLCTL to set/reset the bit in each of the four CR bytes associated with the number of the executing VP; the desired byte of the data manipulator result on AU2B is transferred back to the source CR via the destination bits (¬PED(0-7)), AU2B to CR file enable (PEA2CRX), and CR byte write enable (PESWBXC), all from VPRCONT. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When dependency does not exist, the T and N field specified CR byte is modified as previously mentioned in this step and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KUCSRT instruction group originating from ROM is shown in the transfer table on page 35 of appendix A.

Step 1 Identical to step 1 of the CM source KUCSRT instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary for advance of the PP three-level pipe and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-174 Test CR VP Flag and Skip (KUCT). This instruction group tests the flag bit in the CR byte specified by the T and N fields for one/zero and skips an instruction if the test is satisfied. The flag bit under test is determined by the number of the executing VP. Execution of the KUCT instruction group originating from CM is shown in the transfer table on page 34 of appendix A.

Step 1 The single step CM source KUCT instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the AU makes the skip taken (ST) decision in the following manner: The CR containing the byte to be tested is input to the AU as described in step 1 of the CM source KUCSRT instruction group; the AU test box 2 logic uses the selected CR and the VPC from MLCTL to determine if the bit in question in all four bytes of the CR are set; the skip taken (ST) logic on CONTAU uses the test box 2 output, the MIR op-code from VPRCONT (to indicate whether zero or one is being tested for), and the aligner reference (¬PALWSWB(0-3)) from VPRCONT (to indicate which byte is being tested) to develop ST. If the test performed was not satisified (¬ST), the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR. If the test performed was satisfied (ST), the instruction in the SWBD is skipped by reading the next instruction into the SWBD (by transferring the PC to the SWBA and issuing a read request) and incrementing the PC (both of these operations are described in step 2 of the CM source KSTCM instruction group). The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KUCT instruction group originating from ROM is shown in the transfer table on page 36 of appendix A.

Step 1 Identical to step 1 of the CM source KUCT instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary to read from ROM; the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when no instruction is skipped; the PC is applied to ROM rather than the SWBA when an instruction is to be skipped.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-175 Set/Reset CR Bits (KCRSRT). This instruction group sets/resets the bit positions in the left or right half of the CR byte specified by the T and N fields in accordance with the mask provided by the R field. Execution of the KCRSRT instruction group originating from CM is shown in the transfer table on page 37 of appendix A.

Step 1 The single step CM source KCRSRT instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability but the dependency (D) logic on PPCTL2 indicates the next instruction requires complete execution of the current instruction before indexing, the AU sets/resets the CR bits in the following manner: The test/set/reset CR bit enable (¬PELRWC) from VPRCONT permits use of the MIR destination bits (¬PED(0-5)) in the selection of the CR containing the desired half byte and the CR file to MDB enable (PEACRABX) from VPRCONT transfers the selected CR to the MDB; the AU data manipulator uses the selected CR and the MIR op-code and source bits (R field) from VPRCONT to set/reset bits in each half byte of the CR using the R field mask; the desired half byte is transferred back to the source CR via the destination bits (¬PED(0-7)), AU2B to CR file enable (PEA2CRX), CR byte write enable (PESWBXC), and appropriate CR hex write enable (PELWSWLX for left hex and PELWSWRX for right hex), all from VPRCONT. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When dependency does not exist, the CR half byte bit positions are set/reset as previously mentioned in this step and the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KCRSRT instruction group originating from ROM is shown in the transfer table on page 41 of appendix A.

Step 1 Identical to step 1 of the CM source KCRSRT instruction group, with the following exceptions: The BA signal from

SWBSYNC is not necessary to read from ROM and the PP three-level pipe is advanced one level, as described in step 1 of the ROM source KSTCM instruction group, when dependency does not exist.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-176 Test CR Bits, Set/Reset and Skip (KCRTSRT). This instruction group tests the bit positions marked by ones in the R field in the left or right half of the CR byte specified by the T and N fields for any one/zero. If the test is satisfied, the next instruction is skipped. Independent of the test, the bit positions marked by the R field are set to one/zero. Execution of the KCRTSRT instruction group originating from CM is shown in the transfer table on page 38 of appendix A.

Step 1 The single step CM source KCRTSRT instruction group begins execution in step class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the AU develops the skip taken (ST) decision in the following manner: The CR containing the half byte to be operated on is transferred to the AU via the MDB as described in step 1 of the CM source KCRSRT instruction group; the AU test box 2 logic uses the selected CR and the MIR source bits (R field) from VPRCONT to determine if there are any ones/zeroes in the R field marked bits in each of the CR half bytes; the skip taken (ST) logic on CONTAU uses the test box 2 outputs, the MIR op-code from VPRCONT (to determine whether ones or zeroes are being tested for and to indicate the half byte being tested), and the aligner reference (¬PALWSWB(0-3)) from VPRCONT (to indicate which byte is being tested) to develop ST. Independent of the development of the ST signal, the AU data manipulator uses the selected CR and the MIR op-code and source bits (R field) from VPRCONT to set/reset bits in each half byte of the CR word using the R field as a mask. The desired half byte from the data manipulator is transferred back to the source CR via the destination bits (¬PED(0-7)), AU2B to CR file enable (PEA2CRX), CR byte write enable (PESWBXC), and appropriate CR hex write enable (PELWSWLX for left hex and PELWSWRX for right hex), all from VPRCONT. If the test performed was satisfied (ST), the instruction in the SWBD is skipped by reading the next instruction into the SWBD (by transferring the PC to the SWBA and issuing

a read request) and incrementing the PC (both of these operations are described in step 2 of the CM source KSTCM instruction group). The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If the test performed was not satisfied and no dependency exists (¬D), the PP three-level pipe is advanced as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR. If the test performed was not satisfied and dependency does exist, the IR state class and step are advanced to three and two, respectively.

Step 2 — When the BA signal indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KCRTSRT instruction group originating from ROM is shown in the transfer table on page 42 of appendix A.

Step 1 — Identical to step 1 of the CM source KCRTSRT instruction group, with the following exceptions: The BA signal from SWBSYNC is not required when reading from ROM and the PC is applied to ROM (via the PPPCRBE enable from PCCTL) rather than the SWBA prior to a read cycle.

Step 2 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-177 Test CR Bits and Skip (KCRLO). This instruction group tests the bit positions (marked by ones in the R field) in the left or right half of the CR byte specified by the T and N fields for any or all of one(s)/zero(s). If the test is satisfied, the next instruction is skipped. Execution of the KCRLO instruction group originating from CM is shown in the transfer table on page 39 of appendix A.

Step 1 — The single step CM source KCRLO instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the AU develops the skip taken (ST) decision in the following manner: The CR containing the half byte to be tested is transferred to the AU via the MDB as described in step 1 of the CM source KCRSRT instruction group; the AU test box 2 logic uses the selected CR and the MIR source bits (R field) from VPRCONT to determine if there are

any or all one(s)/zero(s) in the R field marked bit positions in each of the CR half bytes; the skip taken (ST) logic on CONTAU uses the test box 2 outputs, the MIR op-code from VPRCONT (to determine whether any or all one(s)/zero(s) are being tested for and to indicate the half byte being tested), and the aligner reference (¬PALWSWB(0-3)) from VPRCONT (to indicate which byte is being tested) to develop ST. If the test performed was satisfied (ST), the instruction in the SWBD is skipped by reading the next instruction into the SWBD (by transferring the PC to the SWBA and issuing a read request) and incrementing the PC (both of these operations are described in step 2 of the CM source KSTCM instruction group). The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If the test performed was not satisfied (¬ST), the PP three-level pipe is advanced as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR.

Step 2 — When the BA signal indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KCRLO instruction group originating from ROM is shown in the transfer table on page 43 of appendix A.

Step 1 — Identical to step 1 of the CM source KCRLO instruction group, with the following exceptions: The BA signal from SWBSYNC is not required when reading from ROM and the PC is applied to ROM (via the PPPCRBE enable from PCCTL) rather than the SWBA prior to a read cycle.

Step 2 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-178 Test Poll Bits (KTPOL). This instruction tests the CR byte specified by the T and N fields for a one in any of the bit positions and skips the next instruction if a one is found. The binary representation of the number of bit positions to the most significant one is inserted in the VPR halfword specified by the R field. (If a one is not found, the VPR halfword is cleared and no skip is taken.) Execution of the KTPOL instruction originating from CM is shown in the transfer table on page 40 of appendix A.

Step 1 — The single step CM source KTPOL instruction begins execution in state class 7, step 2, as determined by the

state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the AU develops the skip taken (ST) decision in the following manner: The CR containing the byte to be polled is transferred to the AU via the MDB as described in step 1 of the CM source KUCSRT instruction group; the AU bit picker logic develops the count to the most significant one and an all zero indicator for each of the four CR bytes; the skip taken logic on CONTAU uses the all zero indicators from the bit picker, the MIR op-code from VPRCONT (to determine a POLL instruction is executing), and the aligner reference (¬PALWSWB(0-3)) from VPRCONT (to indicate which byte is being polled) to develop ST. Independent of the development of the ST signal, PPCTL2 selects the byte bit picker count specified by the T and N fields (using the MIR source bits) and transfers the result to both halves of AU1B. The byte count is transferred to the VPR halfword specified by the R field via the appropriate AU1B to VPR byte enables (PUWA1P01(0-3), PUWA1P23(0-3), PUWA1P45(0-3), or PUWA1P67(0-3)) and the AU1B to VPR word select (PUWA1C(0-2)), both from VPRCONT. If a one was found during the poll (ST), the instruction in the SWBD is skipped by reading the next instruction into the SWBD (by transferring the PC to the SWBA and issuing a read request) and incrementing the PC (both of these operations are described in step 2 of the CM source KSTCM instruction group). The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If no one was found during the poll and no dependency exists (¬D), the PP three-level pipe is advanced as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR. If no one was found during the poll and dependency does exist, the IR state class and step are advanced to three and two, respectively.

Step 2 — When the BA signal indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KTPOL instruction originating from ROM is shown in the transfer table on page 44 of appendix A.

Step 1 — Identical to step 1 of the CM source KTPOL instruction, with the following exceptions: The BA signal from

SWBSYNC is not necessary to read from ROM and the PC is applied to ROM (via the PPPCRBE enable from PCCTL) rather than the SWBA prior to a read cycle.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-179 Compare CM to VPR (KSKUCM). This instruction group compares the CM word or halfword specified by the T and N fields with the VPR specified by the R field and skips the next instruction if the two quantities compared are equal/not equal. Execution of the KSKUCM instruction group originating from CM is shown in the transfer table on page 45 of appendix A.

Step 1 The multistep CM source KSKUCM instructions begin execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to SWBA via the EA to CMAB enable (PPTACBE) on PCCTL and the CMAB to SWBA enable (PMCBMAE) on PCCTL, the SWBD is transferred over CMDB to the NIR via the PNMDCDE and PNCDNRE enables from PCCTL to make room for the CM word to be read, the IR NIL bit is set to one by the data real logic on PPCTL2 to indicate the next instruction is in the NIR, a read request (¬PMRC) is issued from PCCTL to retrieve the CM word addressed by the MIR EA, and the IR state class and step are advanced to seven and two, respectively, by the data real logic on PPCTL1.

Step 2 When the BA signal from SWBSYNC indicates buffer availability, the AU develops the skip taken (ST) decision in the following manner: The appropriate VPR to VPRB1 enable (PURA1P01, PURA1P23, PURA1P45, or PURA1P67) and word select (PURA1C(0-2)), both from VPRCONT, combine to enable the desired VPR to the AU; the SWBD to MDB enable (PMMDABE) and word select (PPRABC(0-2)), both from PCCTL, combine to enable the SWBD to the AU via the MDB; the AU comparator uses the VPR and CM word to determine if each of the four bytes compared are identical; the skip taken (ST) logic on CONTAU utilizes the comparator outputs, the MIR op-code from VPRCONT (to determine if the instruction is a skip if equal or skip if not equal), and the aligner reference (¬PALWSWB(0-3)) from VPRCONT (to indicate which halfword is being compared) to develop ST. If the comparison was satisfied (ST), the instruction in the

SWBD is skipped by reading the next instruction into the SWBD (by transferring the PC to the SWBA and issuing a read request) and incrementing the PC (both of these operations are described in step 2 of the CM source KSTCM instruction group). The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If the comparison was not satisfied (¬ST), the PP three-level pipe is advanced as described in step 2 of the CM source KSTCM instruction group.

Step 3 — When the BA signal indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Execution of the KSKUCM instruction group originating from ROM is shown in the transfer table on page 48 of appendix A.

Step 1 — Identical to step 1 of the CM source KSKUCM instruction group except for the following: The SWBD to NIR transfer and the setting of the IR NIL bit are not necessary because the instruction source is ROM.

Step 2 — Identical to step 2 of the CM source KSKUCM instruction group except for the following: The PC is applied to ROM (via the PPPCRBE enable from PCCTL) rather than the SWBA prior to a read cycle.

Step 3 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-180 Compare Immediate to VPR (KSKUIM). This instruction group compares the word, halfword, or byte immediate operand specified by the T and N fields with the VPR specified by the R field and skips the next instruction if the two quantities compared are equal/not equal. Execution of the KSKUIM instruction group originating from CM is shown in the transfer table on page 46 of appendix A.

Step 1 — The single step CM source KSKUIM instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the AU develops the skip taken (ST) decision in the following manner: The VPR containing data to be compared is input to the AU as described in step 2 of the CM source KSKUCM instruction group; the immediate to MDB enables (PIQIMED(2) and PIQIMED(3)) from PCCTL transfer the MIR immediate operand to the AU via the MDB; the AU

comparator uses the immediate operand and the VPR to determine if each of the four bytes compared are identical; the skip taken (ST) logic on CONTAU uses the comparator outputs, the MIR op-code from VPRCONT (to determine if the instruction is a skip if equal or skip if not equal), and the aligner reference (¬PALWSWB(0-3)) from VPRCONT (to indicate which halfword or byte is the subject of the instruction) to develop ST. If the comparison was satisfied (ST), the instruction in the SWBD is skipped by reading the next instruction into the SWBD (by transferring the PC to the SWBA and issuing a read request) and incrementing the PC (both of these operations are described in step 2 of the CM source KSTCM instruction group). The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If the comparison was not satisfied (¬ST), the PP three-level pipe is advanced as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR.

Step 2 When the BA signal indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KSKUIM instruction group originating from ROM is shown in the transfer table on page 49 of appendix A.

Step 1 Identical to step 1 of the CM source KSKUIM instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary to read from ROM and the PC is applied to ROM (via the PPCRBE enable from PCCTL) rather than the SWBA prior to a read cycle.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-181 Compare VPR/CR to VPR (KSKUPPU). This instruction group compares the VPR/CR word, halfword, or byte specified by the T and N fields with the VPR specified by the R field and skips the next instruction if the two quantities compared are equal/not equal. Execution of the KSKUPPU instruction group originating from CM is shown in the transfer table on page 47 of appendix A.

Step 1 The single step CM source KSKUPPU instruction group begins execution in state class 7, step 2, as determined

by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the AU develops the skip taken (ST) decision in the following manner: The VPR specified by the R field is input to the AU as described in step 2 of the CM source KSKUCM instruction group; if the T and N fields specify a VPR, the appropriate VPAB to MDB enable (PURABP01, PURABP23, PURABP45, or PURABP67) and VPR to MDB word select (PURABC(0-2)), both from VPRCONT, transfer the selected VPR to the AU via the MDB; if the T and N fields specify a CR, the MIR source bits (¬PES(0-5)) from VPRCONT are used to select the CR containing the data to be compared and the CR file to MDB enable (PEACRABX) from VPRCONT transfers the selected CR to the AU via the MDB; the aligner enable (¬PALALIGN), aligner reference (¬PALWSWB(0-3)), and aligner object (¬PALRSWB(0-3)), all from VPRCONT, combine to align the MDB VPR/CR if necessary (alignment is required if the halfwords or bytes involved in the comparison are in different parts of their respective words); the AU comparator uses the aligned VPR/CR and the R field specified VPR to determine if each of the four bytes compared are identical; the skip taken (ST) logic on CONTAU uses the comparator outputs, the MIR op-code from VPRCONT (to determine if the instruction is a skip if equal or skip if not equal), and the aligner reference from VPRCONT (to indicate which halfword or byte is the subject of the instruction) to develop ST. If the comparison was satisfied (ST), the instruction in the SWBD is skipped by reading the next instruction into the SWBD (by transferring the PC to the SWBA and issuing a read request) and incrementing the PC (both of these operations are described in step 2 of the CM source KSTCM instruction group). The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If the comparison was not satisfied (¬ST), the PP three-level pipe is advanced as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR.

Step 2 When the BA signal indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KSKUPPU instruction group originating from ROM is shown in the transfer table on page 50 of appendix A.

Step 1 Identical to step 1 of the CM source KSKUPPU instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary to read from ROM and the PC is applied to ROM (via the PPCRBE enable from PCCTL) rather than the SWBA prior to a read cycle.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-182 Arithmetic Conditional Branch (KCBAT). This instruction group tests the VPR/CR word, halfword, or byte specified by the R field for zero, nonzero, greater than zero, or less than zero. If the test is satisfied, a PC relative branch is taken as specified by the T and N fields. Execution of the KCBAT instruction group originating from CM is shown in the transfer table on page 51 of appendix A.

Step 1 The single step CM source KCBAT instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. Prior to any decision making, the AU develops the branch taken (BT) decision in the following manner: If the R field specifies a VPR, the VPR is input to the AU over the MDB via the appropriate VPAB to MDB enable (PURABP01, PURABP23, PURABP45, or PURABP67) and word select (PURABC(0-2)) from VPRCONT; if the R field specifies a CR, the MIR source bits (¬PES(0-5)) from VPRCONT are used to select the CR containing the data to be tested and the CR file to MDB enable (PEACRABX) from VPRCONT transfers the selected CR to the AU via the MDB; the AU test box 1 logic uses the input VPR/CR to develop zero, nonzero, negative, and positive indicators for each of the four bytes; the branch taken (BT) logic on CONTAU uses the test box 1 outputs, the MIR op-code from VPRCONT (to determine if the test is for zero, nonzero, negative, or positive), and the aligner reference from VPRCONT (to indicate which halfword or byte is the subject of the instruction) to develop BT. If the test is not satisfied (¬BT) when the BA signal from SWBSYNC indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR. If the test is satisfied (BT) and the KCBAT instruction is indirect (DC), the IR state class and step are advanced to two and one, respectively, by the data real logic on PPCTL1. If the

test is satisfied and the KCBAT instruction is direct, the IR state class and step are both advanced to two.

Step 2 — When the BA signal indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to the SWBA via the EA to CMAB enable (PPTACBE) and the CMAB to SWBA enable (PMCBMAE), both on PCCTL. A read request (¬PMRC) is issued from PCCTL to retrieve the indirect cell addressed by the MIR EA and the IR state class and step are both advanced to two.

Step 3 — When the BA signal indicates the buffer is available and the MIR DC bit indicates the current MIR data is not indirect, the branch address (MIR EA) from PPCTL1 is transferred to the SWBA via the PPTACBE and PMCBMAE enables on PCCTL and the branch address plus one is transferred to the PC by enabling CMAB to the PC indexer via PPINDXMA on PPCTL1, supplying the PC indexer with the increment by one signal (PPPLUS1 from PPCTL1), and enabling the PC indexer result to the PC via PPI1PCE on PCCTL. A read request (¬PMRC) is issued from PCCTL to retrieve the branched to instruction and the IR state class and step are advanced to three and two, respectively. When the MIR instruction is indirect (DC) but the SWBD/NIR indirect indicator from PPCTL2 indicates the SWBD instruction is not indirect (¬DB), the IR data is modified as follows: The DC bit is cleared by the data real logic on PPCTL2 (¬PIDCDR) and the EA is updated by the TN indexer output via the byte enables (PIRYB1(2), PIRYB0(3), and PIRYB1(3)) from PPCTL2. The remainder of the IR data is not changed. When the MIR instruction is indirect (DC) and the SWBD/NIR indirect indicator from PPCTL2 indicates the SWBD instruction is also indirect (DB), the IR data is modified as follows: The DC bit is set by the data real logic on PPCTL2, the state class and step are advanced to two and one, respectively, by the data real logic on PPCTL1, and the EA is updated by the TN indexer output via the byte enables previously mentioned.

Step 4 — The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR.

Execution of the KCBAT instruction group originating from ROM is shown in the transfer table on page 53 of appendix A.

Step 1 Identical to step 1 of the CM source KCBAT instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary to read from ROM and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when no branch is to be taken.

Step 2 Identical to step 2 of the CM source KCBAT instruction group.

Step 3 Identical to step 3 of the CM source KCBAT instruction group, with the following exceptions when the current MIR data is not indirect: The branch address (MIR EA) from PPCTL1 is transferred to ROM via the PPTARBE enable on PCCTL, the branch address plus one is developed by enabling RMAB to the PC indexer via PPINDXMR on PPCTL1 (rather than PPINDXMA), and the BA signal is not necessary to read from ROM.

Step 4 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-183 Increment/Decrement Conditional Branch (KCBIMDT). This instruction group increments/decrements by one the VPR halfword specified by the R field and takes a PC relative branch as specified by the T and N fields if the result is zero/nonzero. Execution of the KCBIMDT instruction group originating from CM is shown in the transfer table on page 52 of appendix A.

Step 1 The single step CM source KCBIMDT instruction group begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. Initially, the AU develops the branch taken (BT) decision in the following manner: The appropriate VPR to VPRB enable (PURA1P01, PURA1P23, PURA1P45, or PURA1P67) and the word select (PURA1C(0-2)), both from VPRCONT, combine to transfer the VPR containing the halfword to be incremented/decremented to the AU; the AU test box 3 logic uses the input VPR to develop plus one and minus one indicators for each of the four bytes; the branch taken (BT) logic on CONTAU uses the test box 3 outputs, the MIR op-code from VPRCONT (to determine if the instruction is an increment or decrement and whether the test is for zero or nonzero), and the aligner reference from VPRCONT (to indicate which half of the VPR is under consideration) to develop BT. In addition to the development of BT, the AU uses the MIR op-code to add or subtract one from each half of the input VPR. The

resulting VPR halfword specified by the R field is transferred back to its source via the appropriate AU1B to VPR byte enables (PUWA1P01(0-3), PUWA1P23(0-3), PUWA1P45(0-3), or PUWA1P67(0-3)) and word select (PUWA1C(0-2)), both from VPRCONT. When the BA signal from SWBSYNC indicates buffer availability, but the test is not satisfied (¬BT) and the next instruction is not dependent (¬D) on the modified VPR, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR. When dependency does exist and the test is not satisfied, the IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. When the test is satisfied (BT) but the current instruction is indirect (DC), the IR state class and step are advanced to two and one, respectively. When the test is satisfied and the current instruction is not indirect, the IR state class and step are both advanced to two.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Step 3 When the BA signal indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to the SWBA via the EA to CMAB enable (PPTACBE) and the CMAB to SWBA enable (PMCBMAE), both from PCCTL. A read request (¬PMRC) is issued from PCCTL to retrieve the indirect cell addressed by the MIR EA and the IR state class and step are both advanced to two.

Step 4 Identical to step 3 of the CM source KCBAT instruction group.

Step 5 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR.

Execution of the KCBIMDT instruction group originating from ROM is shown in the transfer table on page 54 of appendix A.

Step 1 Identical to step 1 of the CM source KCBIMDT instruction group, with the following exceptions: The BA signal from SWBSYNC is not necessary to read from ROM and the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when no branch is to be taken.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

Step 3 Identical to step 3 of the CM source KCBIMDT instruction group.

Step 4 Identical to step 3 of the ROM source KCBAT instruction group.

Step 5 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-184 Unconditional Branch (KUCB). This instruction group branches (PC relative, base relative, CM absolute, and ROM) as specified by the T and N fields. Execution of the KUCB instruction group originating from CM is shown in the transfer table on page 55 of appendix A.

Step 1 The branch CM source KUCB instruction group begins execution in state class 2, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability and the current instruction is not indirect (¬DC), the branch address (MIR EA) from PPCTL1 is transferred to the SWBA via the PPTACBE and PMCBMAE enables on PCCTL and the branch address plus one is transferred to the PC by enabling CMAB (branch address) to the PC indexer via PPINDXMA on PPCTL1, supplying the PC indexer with the increment-by-one signal (PPPLUS1 from PPCTL1), and enabling the PC indexer result to the PC via PPI1PCE on PCCTL. A read request (¬PMRC) is issued from PCCTL to retrieve the branched to instruction and the IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR.

Execution of the KUCB instruction group originating from ROM is shown in the transfer table on page 57 of appendix A.

Step 1 Identical to step 1 of the CM source KUCB instruction group, except for the following: The branch address (MIR EA) from PPCTL1 is transferred to ROM via the PPTARBE enable from PCCTL, the branch address plus one is developed by enabling RMAB to the PC indexer via

PPINDXMR on PPCTL1 (rather than PPINDXMA), and the BA signal is not necessary to read from ROM.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-185 Unconditional Branch and Load PC (KUCBLPC). This instruction group branches (PC relative, base relative, CM absolute, and ROM) as specified by the T and N fields and loads the address of the next instruction in the stream in the VPR specified by the R field. The most significant bit of the VPR is set to indicate which instruction stream is currently being accessed (one for CM and zero for ROM). Execution of the KUCBLPC instruction group originating from CM is shown in the transfer table on page 56 of appendix A.

Step 1 The branch CM source KUCBLPC instruction group begins execution in state class 2, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability and the current instruction is not indirect (¬DC), the branch address (MIR EA) from PPCTL1 is transferred to the SWBA via the PPTACBE and PMCBMAE enables from PCCTL. The branch address is incremented by one in the IR by enabling the MIR EA to the TN field indexer (via PTI2SIR from PPCTL1), generating the carry input enable (PTI2KIN on PPCTL1) for the TN field indexer, and transferring the TN field indexer result to the IR EA via the EA byte enables (PIRYB1(2), PIRYB0(3), and PIRYB1(3)) from PPCTL2. The PC is decremented by one by enabling the PC to RMAB (via PPPCRBE on PCCTL), enabling RMAB to the PC indexer (via PPINDXMR on PPCTL1), supplying the PC indexer with the decrement by one signals (PPMINUS1 and PPMNUSK from PPCTL1), and enabling the PC indexer result to the PC via PPI1PCE from PCCTL. A read request (¬PMRC) is issued from PCCTL to retrieve the branched to instruction and the IR state class and step are advanced to seven and three, respectively, by the data real logic on PPCTL1.

Step 2 When the BA signal indicates buffer availability, if the dependency logic on PPCTL2 (PPD) indicates the next instruction is not dependent (¬D) on the VPR to be loaded with the PC and the next instruction BTN logic on PPCTL2 (PINIBTN) indicates the instruction following the current KUCBLPC instruction is not a PC relative branch, the PC value that follows the current instruction is transferred to

the VPR specified by the R field in the following manner: The PC is enabled to the MDB over PCAB via PPPCABE on PCCTL; the MDB data is passed through the AU aligner to AU2B and the appropriate AU2B to VPR byte enables (PUWA2P01(0-3), PUWA2P23(0-3), PUWA2P45(0-3), or PUWA2P67(0-3)) and word select (PUWA2C(0-2)), both from VPRCONT, transfer the AU2B data to the destination VPR. The MIR EA (containing the original branch address plus one) is transferred to the SWBA (as described in step 1 of this instruction group) and incremented by one and transferred to the PC (as described in step 1 of the CM source KUCB instruction group). The MIR mode bit is transferred to the PC via the ¬PPI10(0) signal from PCCTL, the NIL bit is zeroed to indicate the next instruction is in the SWBD via the ¬PINILDR signal from PPCTL2, a read request (¬PMRC) is issued by PCCTL to retrieve the instruction following the branched to instruction, and the branched to instruction is transferred to the IR with the aid of the next instruction indicator (PININS) from PPCTL1 in the following manner: The SWBD op-code is enabled through the data real logic on PPCTL2 to the IR; the data real state class and step developed on PPCTL1 are enabled to the IR; the data real DC bit, M digit, and flags developed on PPCTL2 are enabled to the IR; the outputs of the TN and R field indexers are input to the source and destination fields of the IR via the source enables (¬PIRXB0(1) for R and PIRYB0(1) for TN) and destination enables (¬PIRXB1(1) for R and PIRYB1(1) for TN) supplied by PPCTL2; the output of the TN field indexer or shift update logic on CONTAU is input to the EA of the IR via the TN field indexer byte enables (PIRYB1(2), PIRYB0(3), and PIRYB1(3)) or shift count enable (¬PIRXB1(3)) supplied by PPCTL2. If dependency does exist and the instruction following the current KUCBLPC instruction is not a PC relative branch, the PC value that follows the current instruction is transferred to the VPR specified by the R field as previously described in this step. The IR state class and step are both advanced to three. When a PC relative instruction does follow the current KUCBLPC instruction (NIBTN), the PC following the current instruction is transferred to the VPR specified by the R field as previously described in this step, the branch address (containing the original branch address plus one) plus one is transferred to the PC by enabling the branch address (MIR EA) to CMAB via PPTACBE on PCCTL, transferring

CMAB to the PC indexer via PPINDXMA on PPCTL1, supplying the PC indexer with the increment by one signal (PPPLUS1 from PPCTL1), and enabling the PC indexer result to the PC via PPI1PCE on PCCTL, the SWBD is transferred to the NIR over CMDB via PNMDCDE and PNCDNRE from PCCTL, the MIR mode bit is transferred to the PC via the ¬PPI10(0) signal from PCCTL, and the IR NIL bit is set to one via ¬PINILDR from PPCTL2 to indicate the next instruction is in the NIR. The IR state class and step are advanced to two and three, respectively.

Step 3 The MIR EA (containing the original branch address plus one) is transferred to the SWBA and incremented by one and transferred to the PC (both as described in step 1 of the CM source KUCB instruction group). The MIR mode bit is transferred to the PC, a read request is issued, the IR NIL bit is zeroed, and the branched to instruction is transferred to the IR, all as described in step 2 of this instruction group.

Step 4 The PC is decremented by one as described in step 1 of this instruction so the instruction following the branched to instruction is not missed in the NIBTN case. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1.

Step 5 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Execution of the KUCBLPC instruction group originating from ROM is shown in the transfer table on page 58 of appendix A.

Step 1 Identical to step 1 of the CM source KUCBLPC instruction group, except for the following: The branch address (MIR EA) from PPCTL1 is transferred to ROM via the PPTARBE enable from PCCTL, the PC is decremented by one by enabling the PC to CMAB (rather than RMAB) via PPPCCBE on PCCTL and CMAB to the PC indexer via PPINDXMA on PPCTL1, and the BA signal is not necessary to read from ROM.

Step 2 When dependency does not exist and the instruction following the KUCBLPC instruction is not PC relative (¬D·¬NIBTN), the PC value that follows the current instruction is transferred to the VPR specified by the R field as described in step 2 of the CM source KUCBLPC instruction group; the MIR EA (branch address plus one) is transferred to ROM via the PPTARBE enable from

PCCTL and is incremented by one and inserted in the PC by enabling RMAB (branch address plus one) to the PC indexer via PPINDXMR on PPCTL1, supplying the PC indexer with the increment by one signal (PPPLUS1 from PPCTL1), and enabling the PC indexer result to the PC via PPI1PCE on PCCTL; the MIR mode bit is transferred to the PC via the ¬PPI10(0) signal from PCCTL; the IR NIL bit is set via ¬PINILDR on PPCTL2 to indicate the next instruction is in the NIR; and the branched to instruction is transferred to the IR as described in step 2 of the CM source KUCBLPC instruction group, except that the source of the branched-to instruction is the NIR rather than the SWBD. If dependency does exist and the instruction following the current KUCBLPC instruction is not a PC relative instruction (D·¬NIBTN), the PC value following the current instruction is transferred to the VPR specified by the R field as described in step 2 of the CM source KUCBLPC instruction group and the IR state class and step are both advanced to three. When a PC relative instruction does follow the current KUCBLPC instruction (NIBTN), the PC is loaded in the specified VPR, the branch address (equal to the original branch address plus one) plus one is transferred to the PC, the MIR mode bit is transferred to the PC and the IR state class and step are advanced to two and three, respectively, all as described in step 2 of the CM source KUCBLPC instruction group.

Step 3 The MIR EA (containing the original branch address plus one) is transferred to ROM and is incremented by one and transferred to the PC (both as described in step 2 of this instruction group). The MIR mode bit is transferred to the PC, the IR NIL bit is set, and the branched to instruction is transferred to the IR, all as described in step 2 of this instruction group.

Step 4 The PC is decremented by one by enabling the PC to CMAB via PPPCCBE on PCCTL, enabling CMAB to the PC indexer via PPINDXMA on PPCTL1, supplying the PC indexer with the decrement by one signal (PPMINUS1 from PPCTL1), and enabling the PC indexer result to the PC via PPI1PCE from PCCTL. The IR state class and step are advanced to three and two, respectively.

Step 5 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-186 Unconditional Branch to ROM, Store PC (KUCBSPC). This instruction branches to the ROM address specified by the T and N fields and stores the address of the next instruction in the stream in one of the eight contiguous CM locations beginning at 20_{16}. The identity of the VP executing the instruction determines the exact CM location. The mode digit of all KUCBSPC instructions is cleared to indicate ROM, so only one transfer table (page 59 of appendix A) is applicable.

Step 1 The branch ROM source KUCBSPC instruction begins execution in state class 2, step 2, as determined by the state class and step data real logic on PPCTL1. The branch address (MIR EA) from PPCTL1 is transferred to ROM via the PPTARBE enable from PCCTL, the branch address is incremented by one and inserted back into the IR for temporary storage as described in step 1 of the CM source KUCBLPC instruction group, and the PC is decremented by one to locate the address to be stored as described in step 4 of the ROM source KUCBLPC instruction group. The IR state class and step are advanced to seven and three, respectively, by the data real logic on PPCTL1.

Step 2 When the BA signal from SWBSYNC indicates buffer availability, the PC is transferred to the SWBD in the following manner: The PC is enabled to the MDB over PCAB via PPPCABE on PCCTL; the MDB data is passed through the AU aligner to AU2B and the AU2B to SWBD enables (PMA2MDLE and PMA2MDRE) from PCCTL transfer the AU2B data to the SWBD. The MIR EA (PITNADDR(8-31)) reflecting the VP number from PPCTL1 is transferred to the SWBA via the PPTACBE and PMCBMAE enables from PCCTL and the MIR mode bit is transferred to the PC via the ¬PPI10(0) signal from PCCTL. A write request (¬PMWC) is issued from PCCTL to store the PC value following that for the current instruction at the CM address specified by the MIR EA and the IR state class and step are advanced to two and three, respectively, by the data real logic on PPCTL1.

Step 3 The MIR EA (containing the original branch address plus one) is transferred to the PC in the following manner: The MIR EA is enabled over CMAB via the PPTACBE enable from PCCTL and input to the PC indexer via the PPINDXMA enable on PPCTL1; the MIR EA is passed through the PC indexer without modification due to the zeroed PPPLUS1 and PPMINUS1 signals from PPCTL1; the PC indexer output is input to the PC via the PPI1PCE load enable from PCCTL.

Step 4 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-187 Push Stack (KPUSH). This instruction stores the VPR specified by the R field in the CM stack specified by the T and N fields at the location specified by the stack pointer retrieved from the stack parameters. The space count parameter is decremented and the word count parameter is incremented to reflect the push operation. If the stack is full, it remains unmodified and the next instruction executes. If the push operation does take place, the next instruction is skipped. Execution of the KPUSH instruction originating from CM is shown in the transfer table on page 60 of appendix A.

Step 1 The multistep CM source KPUSH instruction begins execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA is transferred to the SWBA, the SWBD is saved in the NIR, the IR NIL bit is set to reflect the save, and a read request is issued to retrieve the word count and space count for the stack, all as described in step 1 of the CM source KSKUCM instruction group. The IR state class and step are advanced to four and two, respectively, by the data real logic on PPCTL1.

Step 2 When the BA signal indicates buffer availability, the space count is checked for zero in the following manner: The SWBD (containing the word count and space count) is transferred to the MDB over MDAB via the PMMDABE enable from PCCTL; the AU test box 1 logic uses the MDB data to determine if each of the four bytes is zero; the skip taken for stack logic (ST) on CONTAU uses the test box 1 byte zero indicators and the MIR op-code from VPRCONT (to determine if a push, pull, or modify stack instruction is executing) to develop the skip taken (ST) signal. If the space count is zero (¬ST), the IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1. If there is room to execute the push (ST), the TN field indexer increments the word count and decrements the space count (both parameters are in the SWBD) via PTI2PSH and ¬PTI2STK1 from PPCTL1. The result is transferred to the SWBA (for temporary storage) over CMAB via the PPI2CBE and PMCBMAE enables from PCCTL. The IR NIL bit is zeroed via ¬PINILDT from PPCTL2 to reflect the skip of the NIR instruction and the IR state class and step are advanced to four and three, respectively.

Step 3 When the BA signal indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Step 4 The MIR EA from PPCTL1 is enabled over CMAB to the SWBA via PPTACBE and PMCBMAE from PCCTL and is incremented by one (to locate the address of the stack pointer) and inserted in the IR by enabling the MIR EA to the TN field indexer (via PTI2SIR from PPCTL1), generating the carry input enable (PTI2KIN on PPCTL1) for the TN field indexer, and transferring the TN field indexer result to the IR EA via the EA byte enables (PIRYB1(2), PIRYB0(3), and PIRYB1(3)) from PPCTL2. The SWBA is transferred to the SWBD by enabling the SWBA to the MDB (via PPOMAAB on PCCTL) over MAAB, passing the MDB data through the AU aligner to AU2B, and transferring the AU2B data to the SWBD via the PMA2MDLE and PMA2MDRE enables from PCCTL. A write request (¬PMWC) is issued from PCCTL to update the modified word and space count and the IR state class and step are both advanced to four.

Step 5 When the BA signal indicates buffer availability, the MIR EA (locating the stack pointer) from PPCTL1 is enabled over CMAB to the SWBA via the PPTACBE and PMCBMAE enables from PCCTL. A read request (¬PMRC) is issued from PCCTL to retrieve the stack pointer and the IR state class and step are advanced to four and five, respectively.

Step 6 When the BA signal indicates buffer availability, the SWBD (stack pointer) is transferred to the SWBA in the following manner: The SWBD is input to the TN field indexer via the PTI2SMDM and PTI2SMDL enables from PPCTL1; the input SWBD is passed through the TN field indexer without modification and enabled to the SWBA over CMAB via PPI2CBE and PMCBMAE on PCCTL. The VPR specified by the R field is transferred to the SWBD by enabling the desired VPR to the MDB via the appropriate VPR to MDB enable (PURABP01, PURABP23, PURABP45, or PURABP67) and word select (PURABC(0-2)), both from VPRCONT, passing the MDB data through the AU aligner to AU2B, and enabling the AU2B data to the SWBD via the PMA2MDLE and PMA2MDRE enables from PCCTL. A write request (¬PMWC) is issued from PCCTL to add the VPR to the stack and the IR state class and step are advanced to four and six, respectively.

Step 7 When the BA signal indicates buffer availability, the SWBA (stack pointer) is transferred to the SWBD as described in step 4 of this instruction. The IR state class and step are advanced to four and seven, respectively.

Step 8 The stack pointer (SWBD) is incremented by one and temporarily saved in the SWBA in the following manner: The SWBD is input to the TN field indexer via the PTI2SMDM and PTI2SMDL enables from PPCTL1; the input SWBD is incremented by one via the PTI2KIN (carry input) and PTI2GEN enables from PPCTL1; the result is transferred to the SWBA over CMAB via the PPI2CBE and PMCBMAE enables from PCCTL. The IR state class and step are advanced to four and eight, respectively.

Step 9 The updated stacker pointer (in the SWBA) is transferred to the SWBD as described in step 4 of this instruction and the stack pointer address from PPCTL1 (the MIR EA calculated in step 4 of this instruction) is enabled over CMAB to the SWBA via PPTACBE and PMCBMAE from PCCTL. A write request (¬PMWC) is issued from PCCTL to insert the updated stack pointer in the second parameter word of the stack and the IR state class and step are advanced to two and one, respectively.

Step 10 When the BA signal indicates buffer availability, the PC is input to the SWBA and is incremented by one as described in step 2 of the CM source KSTCM instruction group. A read request is issued to retrieve the skipped to instruction and the IR state class and step are advanced to three and two, respectively.

Execution of the KPUSH instruction originating from ROM is shown in the transfer table on page 63 of appendix A.

Step 1 The multistep ROM source KPUSH instruction begins execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA from PPCTL1 is enabled over CMAB to the SWBA via the PPTACBE and PMCBMAE enables from PCCTL, a read request (¬PMRC) is issued from PCCTL to retrieve the word count and space count for the stack, and the IR state class and step are advanced to four and two, respectively, by the data real logic on PPCTL1.

Step 2 Identical to step 2 of the CM source KPUSH instruction, with the following exception: The IR NIL bit is not zeroed because the instruction source is ROM.

Step 3 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

Step 4 Identical to step 4 of the CM source KPUSH instruction.

Step 5 Identical to step 5 of the CM source KPUSH instruction.

Step 6 Identical to step 6 of the CM source KPUSH instruction.

Step 7 Identical to step 7 of the CM source KPUSH instruction.

Step 8 Identical to step 8 of the CM source KPUSH instruction.

Step 9 Identical to step 9 of the CM source KPUSH instruction.

Step 10 The PC is input to ROM over RMAB via the PC to RMAB enable (PPPCRBE) from PCCTL and is incremented by one by enabling the PC to the PC indexer (via PPINDXMR from PPCTL1), supplying the PC indexer with the increment by one signal (PPPLUS1 from PPCTL1), and enabling the PC indexer result to the PC via PPI1PCE from PCCTL. The IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1.

4-188 Pull Stack (KPULL). This instruction retrieves the word located by the stack pointer minus one from the CM stack specified by the T and N fields and saves the result in the VPR specified by the R field. The space count is incremented, the word count is decremented, and the stack pointer is decremented to reflect the pull operation. If the stack is initially empty, it remains unmodified and the next instruction executes. If the pull operation does take place, the next instruction is skipped. Execution of the KPULL instruction originating from CM is shown in the transfer table on page 61 of appendix A.

Step 1 Identical to step 1 of the CM source KPUSH instruction.

Step 2 Identical to step 2 of the CM source KPUSH instruction, except for the following: The skip taken for stack logic on CONTAU develops the skip taken decision (ST or ¬ST) by testing the word count (rather than the space count) for zero; if a pull is to be executed, the TN field indexer decrements the word count and increments the space count (rather than the opposite) via PTI2PUL and ¬PTI2STK1 from PPCTL1.

Step 3 When the BA signal from SWBSYNC indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group.

Step 4 Identical to step 4 of the CM source KPUSH instruction.

Step 5 Identical to step 5 of the CM source KPUSH instruction.

Step 6 When the BA signal indicates buffer availability, the stack pointer (in the SWBD) is decremented to locate the last word in the stack and transferred to the SWBA in the following manner: The SWBD is input to the TN field indexer via the PTI2SMDM and PTI2SMDL enables from PPCTL1; the input SWBD is decremented by one via the ¬PTI2STK1 and PTI2KGEN enables from PPCTL1; the result is transferred to the SWBA over CMAB via the PPI2CBE and PMCBMAE enables from PCCTL. A read request (¬PMRC) is issued from PCCTL to retrieve the last word in the stack and the IR state class and step are advanced to four and six, respectively.

Step 7 When the BA signal indicates buffer availability, the last word in the stack is transferred to the VPR specified by the R field in the following manner: The SWBD is transferred to the MDB over MDAB via the PMMDABE enable from PCCTL; the MDB data is passed through the AU aligner to AU2B; the appropriate AU2B to VPR byte enables (PUWA2P01(0-3), PUWA2P23(0-3), PUWA2P45(0-3), or PUWA2P67(0-3)) combine with the word select (PUWA2C(0-2)), both from VPRCONT, to transfer the AU2B data to the desired VPR. The IR state class and step are advanced to four and seven, respectively.

Step 8 Identical to step 9 of the CM source KPUSH instruction.

Step 9 Identical to step 10 of the CM source KPUSH instruction.

Execution of the KPULL instruction originating from ROM is shown in the transfer table on page 64 of appendix A.

Step 1 Identical to step 1 of the ROM source KPUSH instruction.

Step 2 Identical to step 2 of the CM source KPULL instruction, except for the following: The IR NIL bit is not zeroed because the instruction source is ROM.

Step 3 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

Step 4 Identical to step 4 of the CM source KPUSH instruction.

Step 5 Identical to step 5 of the CM source KPUSH instruction.

Step 6 Identical to step 6 of the CM source KPULL instruction.

Step 7 Identical to step 7 of the CM source KPULL instruction.

Step 8 Identical to step 9 of the CM source KPUSH instruction.

Step 9 Identical to step 10 of the ROM source KPUSH instruction.

4-189 Modify Stack (KMDF). This instruction modifies the parameters of the CM stack specified by the T and N fields by the amount contained in the VPR halfword specified by the R field. If the modification value is positive, a gap of unused stack locations is created. If the modification value is negative, the most recent stack entries are deleted. The modification value is added to the word count and stack pointer and subtracted from the space count. If either of the counts is negative after the modification, the stack parameters remain unmodified and the next instruction is executed. If both counts are non-negative after the modification, the modify operation does take place and the next instruction is skipped. Execution of the KMDF instruction originating from CM is shown in the transfer table on page 62 of appendix A.

Step 1 Identical to step 1 of the CM source KPUSH instruction.

Step 2 When the BA signal from SWBSYNC indicates buffer availability, the word and space count parameters are modified by the VPR halfword in the following manner: The SWBD is input to the TN field indexer via the PTI2SMDM and PTI2SMDL enables from PPCTL1; the desired VPR is input to the TN field indexer via the PUTFLD1 and PUTFLD2 select lines from PPCTL1; the TN field indexer uses the halfword select (PTI2SRH from PPCTL1) to enable the desired VPR halfword, PUI2SRHT and ¬PTI2SVPL from PPCTL1 to develop the desired halfword and its one's complement, and the carry input (PTI2KIN from PPCTL1) to develop the two's complement from the one's complement, all in order to add the modification value to the word count (most significant half of the SWBD) and subtract the modification value from the space count (least significant half of the SWBD); the result is transferred to the SWBA over CMAB via the PPI2CBE and PMCBMAE enables from PCCTL. The IR state class and step are advanced to seven and two, respectively, by the data real logic on PPCTL1.

Step 3 The modified word and space count are both checked for negative quantities as follows: The SWBA (containing both counts) is transferred to the MDB over MAAB via PMMAABE from PCCTL; the AU test box 1 logic uses the MDB data to determine if each of the four bytes is negative; the skip taken for stack logic on CONTAU utilizes the test box 1 byte negative indicators and the MIR op-code from VPRCONT to develop the skip taken (ST) signal. If one of the count parameters was negative

after the modification (¬ST), the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group. If neither of the count parameters is negative, the IR NIL bit is zeroed via ¬PINILDT from PPCTL2 to reflect the skip of the NIR instruction and the IR state class and step are advanced to four and three, respectively.

Step 4 Identical to step 4 of the CM source KPUSH instruction.

Step 5 Identical to step 5 of the CM source KPUSH instruction.

Step 6 When the BA signal indicates buffer availability, the stack pointer (in the SWBD) is modified by the desired VPR halfword in the following manner: The SWBD is input to the TN field indexer via the PTI2SMDM and PTI2SMDL enables from PPCTL1; the desired VPR is input to the TN field indexer via the PUTFLD1 and PUTFLD2 select lines from PPCTL1; the TN field indexer uses the VPR halfword select (PTI2SRH from PPCTL1) and PTI2KGEN from PPCTL1 to add the SWBD to the sign extended VPR halfword; the result is transferred to the SWBA over CMAB via the PPI2CBE and PMCBMAE enables from PCCTL. The IR state class and step are advanced to four and six, respectively.

Step 7 Identical to step 9 of the CM source KPUSH instruction.

Step 8 Identical to step 10 of the CM source KPUSH instruction.

Step 9 When the BA signal indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group except that the source of the next instruction is the SWBD rather than the NIR.

Execution of the KMDF instruction originating from ROM is shown in the transfer table on page 65 of appendix A.

Step 1 Identical to step 1 of the ROM source KPUSH instruction.

Step 2 Identical to step 2 of the CM source KMDF instruction.

Step 3 Identical to step 3 of the CM source KMDF instruction, except as follows: When one of the count parameters is negative (¬ST), the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group; when both count parameters are non-negative, the IR NIL bit is not zeroed because the instruction source is ROM.

Step 4 Identical to step 4 of the CM source KPUSH instruction.

Step 5 Identical to step 5 of the CM source KPUSH instruction.

Step 6 Identical to step 6 of the CM source KMDF instruction.

Step 7 Identical to step 9 of the CM source KPUSH instruction.

Step 8 Identical to step 10 of the ROM source KPUSH instruction.

Step 9 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-190 Execute CM (KEXCM). This instruction executes the CM instruction specified by the T and N fields (object instruction) as if it were in the location of the original KEXCM instruction, except when the object instruction is a PC relative branch. When the object instruction is a PC relative branch, the PC value used in the development of the branch address is one greater than that used if the object instruction actually replaced the KEXCM instruction. Execution of the KEXCM instruction originating from CM is shown in the transfer table on page 66 of appendix A.

Step 1 The multistep CM source KEXCM instruction begins execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA is transferred to the SWBA, the SWBD is saved in the NIR, and a read request is issued to retrieve the object instruction, all as described in step 1 of the CM source KSKUCM instruction group. The IR state class and step are both advanced to two by the data real logic on PPCTL1.

Step 2 When the object instruction has been successfully retrieved from CM (BA), it (in the SWBD) is indexed and input to the IR in the following manner: the SWBD op-code is enabled through the data real logic on PPCTL2 to the IR; the data real state class and step developed on PPCTL1 are enabled to the IR; the data real DC bit, M digit, and flags developed on PPCTL2 are enabled to the IR; the outputs of the TN and R field indexers are enabled to the source and destination fields of the IR as determined by the source enables (¬PIRXB0(1) for R and PIRYB0(1) for TN) and destination enables (¬PIRXB1(1) for R and PIRYB1(1) for TN) supplied by PPCTL2; the output of the TN field indexer or shift update logic on CONTAU is enabled to the IR EA as determined by the TN field indexer byte enables (PIRYB1(2), PIRYB0(3), and PIRYB1(3)) or shift count enable (¬PIRXB1(3)) supplied by PPCTL2. The IR NIL bit is set in the operation just described to reflect the saving of the next instruction in the NIR in the

previous step. At the next execution clock, the object instruction begins execution.

Execution of the KEXCM instruction originating from ROM is shown in the transfer table on page 69 of appendix A.

Step 1 The multistep ROM source KEXCM instruction begins execution in state class 4, step 1, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability, the MIR EA is transferred to the SWBA and a read request is issued to retrieve the object instruction, both as described in step 1 of the CM source KSKUCM instruction group. The IR state class and step are both advanced to two by the data real logic on PPCTL1.

Step 2 Identical to step 2 of the CM source KEXCM instruction, except the IR NIL bit is set because the instruction source is ROM (not because an instruction was saved in the NIR).

4-191 Load Effective Address (KLEA). This instruction loads the effective address developed by the T and N fields in the VPR specified by the R field. Execution of the KLEA instruction originating from CM is shown in the transfer table on page 68 of appendix A.

Step 1 The single step CM source KLEA instruction begins execution in state class 7, step 2, as determined by the state class and step data real logic on PPCTL1. When the BA signal from SWBSYNC indicates buffer availability and dependency logic (PPD) on PPCTL2 indicates the VPR to be loaded is not involved in the next instruction (¬D), the MIR EA developed by the TN field indexer is transferred to the VPR specified by the R field as follows: The MIR EA is enabled to the MDB via the PIQEFAD(2) and PICIMED(3) enables from PCCTL; the MDB data is passed through the AU aligner to AU2B and the appropriate AU2B to VPR byte enables (PUWA2P01(0-3), PUWA2P23(0-3), PUWA2P45(0-3), or PUWA2P67(0-3)) and word select (PUWA2C(0-2)), both from VPRCONT, transfer the AU2B data to the desired VPR. The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, with the following exception: The source of the next instruction is the SWBD rather than the NIR. When dependency does exist (D), the MIR EA is transferred to the desired VPR as previously mentioned in this step and the IR state class and step are advanced to three and two, respectively, by the data real logic on PPCTL1.

Step 2 The PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group, except for the SWBD source difference.

Execution of the KLEA instruction originating from ROM is shown in the transfer table on page 71 of appendix A.

Step 1 Identical to step 1 of the CM source KLEA instruction, except for the following: The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group when dependency does not exist.

Step 2 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group.

4-192 Indirect Cycle (KNDIREC). The indirect cycle is executed when the next instruction to be executed (in the SWBD or NIR) has the first bit of its T field set (first indirect cycle), or the indirect cell retrieved from CM has the first bit of its T field set (multiple level indirect cycle), and the current instruction has completed execution (as indicated by PININS on PPCTL1). This indirect cycle is not valid for the conditional branch instructions (they have their own unique indirect cycle, as described in the KCBAT and KCBIMDT instruction groups) or the instructions for which indirect is undefined (this applies to the instructions for which the ignore indirect signal (PIIGI) from PPCTL2 is true). Execution of the CM indirect cycle is shown in the transfer table on page 72 of appendix A.

Step 1 When the SWBD/NIR indirect indicator (PIDB) from PPCTL2 indicates the next instruction is not indirect (¬DB), or the ignore indirect signal (PIIGI) from PPCTL2 indicates indirect is undefined for the SWBD/NIR instruction (IGI), and the instruction termination signal (PININS) from PPCTL1 indicates the current instruction has completed execution (NINS), the IR DC bit is cleared via ¬PIDCDR from PPCTL2 to indicate the SWBD/NIR instruction being transferred to the IR is not indirect. The IR NIL bit is cleared via ¬PINILDR from PPCTL2 to indicate the read cycle initiated to retrieve the instruction following the SWBD/NIR instruction involves CM, and the SWBD/NIR instruction begins execution at the next assigned clock. When the SWBD/NIR indirect indicator indicates indirect and indirect is possible (DB·¬IGI) on termination of the current instruction, the IR DC bit is set and the IR state class and step are advanced to two and one, respectively, by the data real logic on PPCTL1.

Step 2 When the BA signal from SWBSYNC indicates buffer availability and the MIR PPTN flag indicates the indirect cell is not (¬PPTN) located in a register (VPR or CR), the MIR EA containing the indirect cell address is transferred to the SWBA, the SWBD is saved in the NIR, the IR NIL bit is set to reflect the save, and a read request is issued to retrieve the indirect cell, all as described in step 1 of the CM source KSKUCM instruction group. The IR state class and step are both advanced to two. When the MIR PPTN flag indicates the indirect cell is in a VPR or CR (PPTN), the SWBD is saved in the NIR, the NIL bit is set to reflect the save, and the register specified by the T and N fields is transferred to the SWBD in the following manner: If the indirect cell is located in a VPR, the appropriate VPR to MDB enable (PURABP01, PURABP23, PURABP45, or PURABP67) combines with the VPR to MDB word select (PURABC(0-2)) to transfer the desired VPR to the MDB; if the indirect cell is located in a CR, the MIR source bits (¬PES(0-5)), or the MIR destination bits (¬PED(0-5)) in the case of an indirect store to a CR, combine with the CR file to MDB enable (PEACRABX), both from VPRCONT, to transfer the desired CR to the MDB; the MDB data is passed through the aligner to the AU2B bus and the AU2B to SWBD enables (PMA2MDLE and PMA2MDRE) from PCCTL enable completion of the register to SWBD transfer. The IR state class and step are both advanced to two.

Step 3 When the BA signal indicates buffer availability (this check is only necessary when the indirect cell is in CM (¬PPTN)), the TN field indexer uses the indirect cell (in the SWBD) to develop an EA, source, or destination for the IR (as determined by the source enable (PIRYB0(1)), destination enable (PIRYB1(1), or EA enables (PIRYB1(2), PIRYB0(3), and PIRYB1(3))). If the T field of the indirect cell indicates multiple level indirect addressing (DB), the IR DC bit is set via ¬PIDCDR from PPCTL2 and the IR state class and step are returned to two and one, respectively. If the terminal level of indirect addressing has been reached (¬DB), the IR DC bit is cleared via ¬PIDCDR and the original indirect instruction begins execution at the next time slot.

Execution of the ROM indirect cycle is shown in the transfer table on page 73 of appendix A.

Step 1 Identical to step 1 of the CM indirect cycle, except for the following: The NIL bit is set (rather than cleared) when there is no indirect cycle to reflect the ROM instruction source.

Step 2 Identical to step 2 of the CM indirect cycle, except for the following: The saving of the SWBD and setting of the IR NIL bit is not necessary because of the ROM instruction source.

Step 3 Identical to step 3 of the CM indirect cycle.

4-193 No Operation (KNOOP). The no operation (no-op) instruction is a default condition that jumps to the next instruction in the PP three-level pipe. Execution of the KNOOP instruction originating from CM is shown in the transfer table on page 74 of appendix A.

Step 1 When the BA signal from SWBSYNC indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group (assuming no interrupt (¬INTF) occurs during the execution clock), except for the following: The source of the next instruction is the SWBD rather than the NIR.

Execution of the KNOOP instruction originating from ROM is shown in the transfer table on page 75 of appendix A.

Step 1 The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group, assuming no interrupt (¬INTF) occurs during the execution clock.

4-194 Interrupt Cycle (INTRPT). The interrupt cycle is executed at the conclusion of the instruction during which a programmed or automatic interrupt occurred. In general, the interrupt cycle directs PP control to ROM location 10_{16} if an automatic interrupt occurs or ROM location 11_{16} if a programmed interrupt occurs. Execution of the CM interrupt cycle is shown in the transfer table on page 76 of appendix A.

Step 1 If an automatic or programmed interrupt has not occurred at the conclusion of the current instruction (¬INT • NINS), the PP three-level pipe is advanced one level as described in step 2 of the KSTCM instruction group. If an automatic or programmed interrupt has occurred (INT), the MIR INTF bit is not set (¬INTF), the current instruction has terminated (NINS), and the current instruction is not execute CM (¬EXCM), the IR INTF bit is set via ¬PINTFDR from PPCTL2 to initiate the interrupt cycle, the IR NIL

bit is set via ¬PINILDR from PPCTL2 to point to the instruction retrieved from ROM locations 10_{16} or 11_{16}, the IR op-code is forced to the no-op case via the data real logic on PPCTL2, and the PC is incremented by one as described in step 2 of the CM source KSTCM instruction group. The IR state class and step are advanced to two and three, respectively, by the data real logic on PPCTL1.

Step 2 — The MIR EA, 10_{16} if an automatic interrupt has been recorded or 11_{16} if a programmed interrupt has been recorded, is transferred to the NIR via the PPTARBE and PNCDNRE enables from PCCTL. The PC is decremented by two to point to the instruction following the interrupted instruction in the following manner: The PC is transferred to the PC indexer over CMAB via PPPCCBE and PPINDXMA from PCCTL and PPCTL1, respectively; the PC indexer decrements the input quantity via the zeroed PPPLUS1 and PPMINUS1 signals and the set PPMNUSK signal, all from PPCTL1; the PC indexer result is transferred back to the PC via PPI1PCE from PCCTL. The IR state class and step are advanced to three and two, respectively.

Step 3 — When the BA signal from SWBSYNC indicates buffer availability, the PP three-level pipe is advanced one level as described in step 2 of the CM source KSTCM instruction group (this procedure transfers the ROM service instruction to the IR for execution). The IR INTF bit is zeroed via ¬PINTFDR from PPCTL2 to indicate the interrupt processing is complete.

Execution of the ROM interrupt cycle is shown in the transfer table on page 77 of appendix A.

Step 1 — If an automatic or programmed interrupt has not occurred at the conclusion of the current instruction (¬INT·NINS), the PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group. If an automatic or programmed interrupt has occurred (INT), the MIR INTF bit is not set (¬INTF), the current instruction has terminated (NINS), and the current instruction is not execute CM (¬EXCM), the IR INTF bit is set via ¬PINTFDR from PPCTL2 to initiate the interrupt cycle, the IR op-code is forced to the no-op case via the data real logic on PPCTL2, the PC is applied to ROM via PPPCRBE on PCCTL, and the PC is incremented by one via the RMAB to PC indexer enable

(PPINDXMR from PPCTL1), the PC indexer increment by one signal (PPPLUS1 from PPCTL1), and the PC load enable (PPI1PCE from PCCTL). The IR state class and step are advanced to two and three, respectively, by the data real logic on PPCTL1.

Step 2 — Identical to step 2 of the CM interrupt cycle.

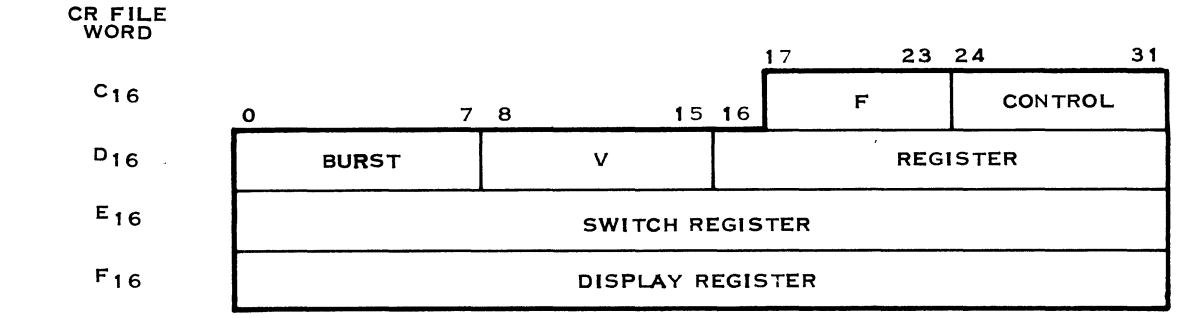
Step 3 — The PP three-level pipe is advanced one level as described in step 1 of the ROM source KSTCM instruction group (this procedure transfers the ROM service instruction to the IR for execution) and the IR INTF bit is zeroed via ¬PINTFDR from PPCTL2 to indicate the interrupt processing is complete.

4-195 MAINTENANCE LOGIC

Control of the PP maintenance logic centers around the CR file maintenance registers, which are controlled by the external maintenance system described in appendix C of this manual. The data entered in the maintenance registers (via the ASC Maintenance Console in the manual mode, the card reader in the semi-automatic mode, or PP software in the automatic mode) is interpreted by the hardcore maintenance logic (located on the MLCTL, ML2, and ML1(0,1) cards) and distributed to the remainder of the PP in order to execute the desired operation. The maintenance command repertoire provides the necessary range of operations so that any area of the PP can be checked out by executing the proper combination of maintenance commands. The following paragraphs give a detailed description of the PP maintenance logic by covering the maintenance registers, the maintenance logic control circuitry, the maintenance logic data path structure, the maintenance logic timing as it relates to the determination of the active VP, and the maintenance command transfer tables.

4-196 MAINTENANCE REGISTERS. The CR file maintenance registers, beginning at bit 17 of word C_{16} in the CR file and extending through word F_{16}, accept data from the Test Control Logic (TCL) in the manual or semi-automatic modes or from the PP ECL circuitry (PP software) in the automatic mode. The maintenance registers are divided into seven basic fields, as shown in figure 4-109. The control field contains the maintenance command to be executed and the status of the maintenance logic and the remaining fields contain the indicators and data necessary for execution of the command. A description of the basic maintenance register fields is given in the following paragraphs.

4-197 Control Field. The maintenance register control field consists of bits 24 through 31 of CR file word C_{16}. A descriptive breakdown of each bit or bit group in the control field is presented in table 4-5. A list of command codes (part of the control field) with the associated PP reaction is detailed in table 4-6.



(A)124816

Figure 4-109. Peripheral Processor Maintenance Registers

Table 4-5. Maintenance Register Control Field Breakdown

Bit(s)	Name	Description
24	Busy	Set by the maintenance logic on the clock immediately following receipt of any nonzero command code in bits 28 through 31 of the control field. Reset by the maintenance logic on the clock immediately following completion of the command. Resetting does not occur if the command is illegal.
25	PC Lock	Set by the maintenance logic at the completion of command code 8 to reflect a locked program counter. Reset by the maintenance logic at the completion of command code 9 to reflect an unlocked program counter.
26	Illegal	Set by maintenance logic if an illegal command is inserted in the maintenance registers.
27	Spare	Not used.
28-31	Command Code	Set to one of 15 possible codes ($1\text{-}F_{16}$) to initiate a maintenance command. At completion of the command, the command code bits are reset.

Table 4-6. Maintenance Command Codes

Code	Description
0	This is a no-op command inserted in the command code field at the completion of any command, unless the command is illegal or can not be completed.
1	(SR)→DR The switch register is transferred to the display register via the hardcore maintenance logic.
2	(REG)→DR The PP register specified by the register field is transferred to the display register. This command is illegal if the register field specifies the MIR in the automatic mode.
3	$((SR))_{CM}$→DR The CM word addressed by the switch register is transferred to the display register. The VP portion of the register field is used by the SWBC in making the read request.
4	$((SR))_{ROM}$→DR The ROM word addressed by the switch register is transferred to the display register. The VP portion of the register field is used to specify the NIR used in making the read.
5	(SR)→REG The switch register is transferred to the PP register specified by the register field. If the register field specifies a CR or VPR, the portion of the destination register affected is controlled by the F field. This command is illegal if the register field specifies a SWBC or MIR in the automatic mode. If the register field specifies $SWBC_8$ or $SWBC_9$, command code 5 reacts differently and controls the asynchronous portion of the SWBC.
6	(DR)→REG The display register is transferred to the PP register specified by the register field. If the register field specifies a CR or VPR, the portion of the destination register affected is controlled by the F field. This command is illegal if the register field specifies a SWBC or MIR in the automatic mode.
7	$(DR)\rightarrow(SR)_{CM}$ The display register is stored in CM at the address specified by the switch register. The VP portion of the register field is used by the SWBC in making the write request.
8	This command locks the PC's of the VP's designated by the V field. When this command occurs, the designated PC's are held at their current value until command code 9 is issued.
9	This command unlocks the PC's designated by the V field.

Table 4-6. Maintenance Command Codes (Continued)

Code	Description
A_{16}	This command resets the data holding flip-flops (PC, NIR, SWBD, SWBA, VPR, IR, and CM base) of the VP's designated by the V field.
B_{16}	This command sets the data holding flip-flops (PC, NIR, SWBD, SWBA, VPR, IR, and CM base) of the VP's designated by the V field.
C_{16}	(PP Burst) In the automatic mode, the VP's designated by the V field are subject to advancement, along with the VP's that are active, under the influence of the time slot table. Counting begins at time slot zero and continues until the number of executed time slots equals the value in the burst field. In the semi-automatic and manual modes the reaction to command C_{16} depends on whether or not the SWB has been placed under test by the register field or SWB LOCK switch on the ASC Maintenance Console, respectively. If the SWB is not under test, the PP is advanced the number of time slots indicated by the burst field (this is equivalent to normal PP operation except that it occurs for a limited time). If the SWB is under test, the SWB is advanced (entries are made in the SWBC for the VP's requiring memory access) the number of time slots indicated by the burst field.
D_{16}	(PP Cycle) In the automatic mode, the VP's designated by the V field are advanced, along with the active VP's, under the influence of the time slot table until all V field designated VP's have completed their current instruction. If more than one VP is designated by the V field, termination of advancement of the designated VP's does not necessarily occur at the same time. In the semi-automatic mode, the VP's are advanced as in the automatic mode until time slot zero is reached. When time slot zero is reached, advancement is stopped. In the manual mode, the one VP designated by the V field is advanced (independent of the time slot table) until its current instruction is completed.
E_{16}	(VP Burst) This command is illegal in the automatic and semi-automatic modes. In the manual mode, the single V field specified VP is advanced (independent of the time slot table) as indicated by the burst field.
F_{16}	(VP Continuous) This command is illegal in the automatic and semi-automatic modes. In the manual mode, the VP designated

Table 4-6. Maintenance Command Codes (Continued)

Code	Description
F_{16} (Cont.)	by the V field is advanced (independent of the time slot table) until a new maintenance command code is received. If the new command is other than F_{16}, then, in addition to halting the designated VP, the new command is performed.

4-198 F Field. The maintenance register F field consists of bits 17 through 23 of CR file word C_{16}. The F field is used during the execution of maintenance command codes 5 and 6 when a portion of a CR or VPR needs specifying. Refer to table 4-7 for the manner in which the maintenance logic interprets the F field (the X's indicate don't cares).

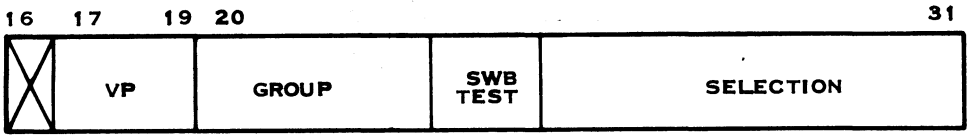
Table 4-7. Maintenance Register F Field Breakdown

F Field	CR/VPR Portion
XXXX000	Byte 0
XXXX001	Byte 1
XXXX010	Byte 2
XXXX011	Byte 3
XXXX100	Left Half Word
XXXX101	Right Half Word
XXXX110	Whole Word
XXXX111	

4-199 Burst Field. The maintenance register burst field consists of bits 0 through 7 of CR file word D_{16}. The burst field is used in conjunction with maintenance command codes C_{16} and E_{16} to specify the number of time slots of advancement.

4-200 V Field. The maintenance register V field consists of bits 8 through 15 of CR file word D_{16}. The V field is used by maintenance command codes 8, 9, A_{16}, B_{16}, C_{16}, D_{16}, E_{16}, and F_{16} to designate the VP's under test. In the manual mode, only one of the bits will be set.

4-201 Register Field. The maintenance register register field consists of bits 16 through 31 of CR file word D_{16}, as shown in figure 4-110. The VP, group, and selection portions of the register field designate a register used



(A)124817

Figure 4-110. Maintenance Register Register Field Format

by the data transfer maintenance command codes 2, 5, and 6. Bit 16 is not used and the SWB test portion of the register field is used in conjunction with command code C_{16} in the semi-automatic mode to affect interpretation of command code C_{16}. A setting of 10_2 places the SWB under test, 01_2 places the SWB in the normal condition, and 00_2 and 11_2 have no effect. The VP, group, and selection portions of the register field designate a register as detailed in table 4-8. If the group and selection designate $SWBC_8$ or $SWBC_9$ in conjunction with command code 5, then the command is interpreted differently and controls the asynchronous portion of the SWBC ($SWBC_8$ stops the asynchronous-MCU interface on an even state and $SWBC_9$ stops the interface on an odd state).

Table 4-8. Register Field Register Designation

VP	Group	Selection	Designated Register
0-7	0000	Not Used	PC in VP(0-7)
0-7	X001	Not Used	NIR in VP(0-7)
0-7	X010	0, 1	IR(0, 1) in VP(0-7)
0-7	X011	Not Used	SWBA in VP(0-7)
0-7	X100	Not Used	SWBD in VP(0-7)
0-7	X101	0-3	VPR(0-3) in VP(0-7)
Not Used	X110	0-63	CR(0-63)
Not Used	X111	0-9	SWBC(0-9)
Not Used	1000	0-7	MIR (0-7)

4-202 Switch Register Field. The maintenance register switch register field consists of all 32 bits of CR file word E_{16}. The switch register field provides data for maintenance command codes 1 and 5 and a memory address for command codes 3, 4, and 7.

4-203 Display Register Field. The maintenance register display register field consists of all 32 bits of CR file word F_{16}. The display register field provides data for maintenance command codes 6 and 7 and is used as the destination register in command codes 1, 2, 3, and 4.

4-204 MAINTENANCE LOGIC CONTROL. The maintenance logic control circuitry is primarily concerned with determining when the PP is executing in the test mode or the normal active mode and controlling the execution of maintenance commands when the test mode is selected. Figure 4-111 is a detailed block diagram of the maintenance logic control system and illustrates the logic cards and major interface signals involved. For presentation purposes, the maintenance logic control system is divided into hardcore maintenance logic and maintenance/PP interface logic. The hardcore maintenance logic consists of the CR file maintenance registers and the MLCTL, ML2, and ML1(0, 1) cards; the maintenance/PP interface logic consists of the remaining cards in figure 4-111.

4-205 Hardcore Maintenance Logic. The hardcore maintenance logic accepts maintenance register data from the Test Control Logic (TCL) in the manual or semi-automatic (card) modes or from a controlling VP (PP software) in the automatic mode; mode, VP select, automatic interrupt off, and SWB lock ASC Maintenance Console switch indications from the TCL; the ASC Maintenance Console master clear switch indication from the Power Control Unit; and the CR protect byte, CM protect byte, SWB priority halfword, and time slot table from the CR file. In return, the hardcore maintenance logic distributes the VP code, next VP code, test mode indicator, master clear, current priority, CR protect enable, CM protect enable, and maintenance command control signals to the maintenance/PP interface logic; the current time slot, current VP code, current VP active indication, compare error, and SWB lock illumination signals to the TCL; the display register data to the ASC Maintenance Console; and the busy, PC lock, and illegal gate to the patchboard for use in setting or resetting those bits in the CR file maintenance registers.

The MLCTL card, detailed in figure 4-112, provides the majority of the hardcore maintenance control interface with the remainder of the PP. A new maintenance command from the TCL (PXCONTD(4-7) in manual or semi-automatic modes) is enabled through MLCTL to the CR file. PP software enters maintenance commands directly in the CR file maintenance registers when the busy bit of the maintenance registers indicates a new command can be accepted. When the current command completes execution, the new command is inserted in the appropriate maintenance register bits and made available to MLCTL as the new current command. At this point in time, the zeroed current state and presence of the new command combine to set the busy bit to indicate execution of a maintenance command is in progress. A state diagram of the three flip-flop maintenance controller used to execute

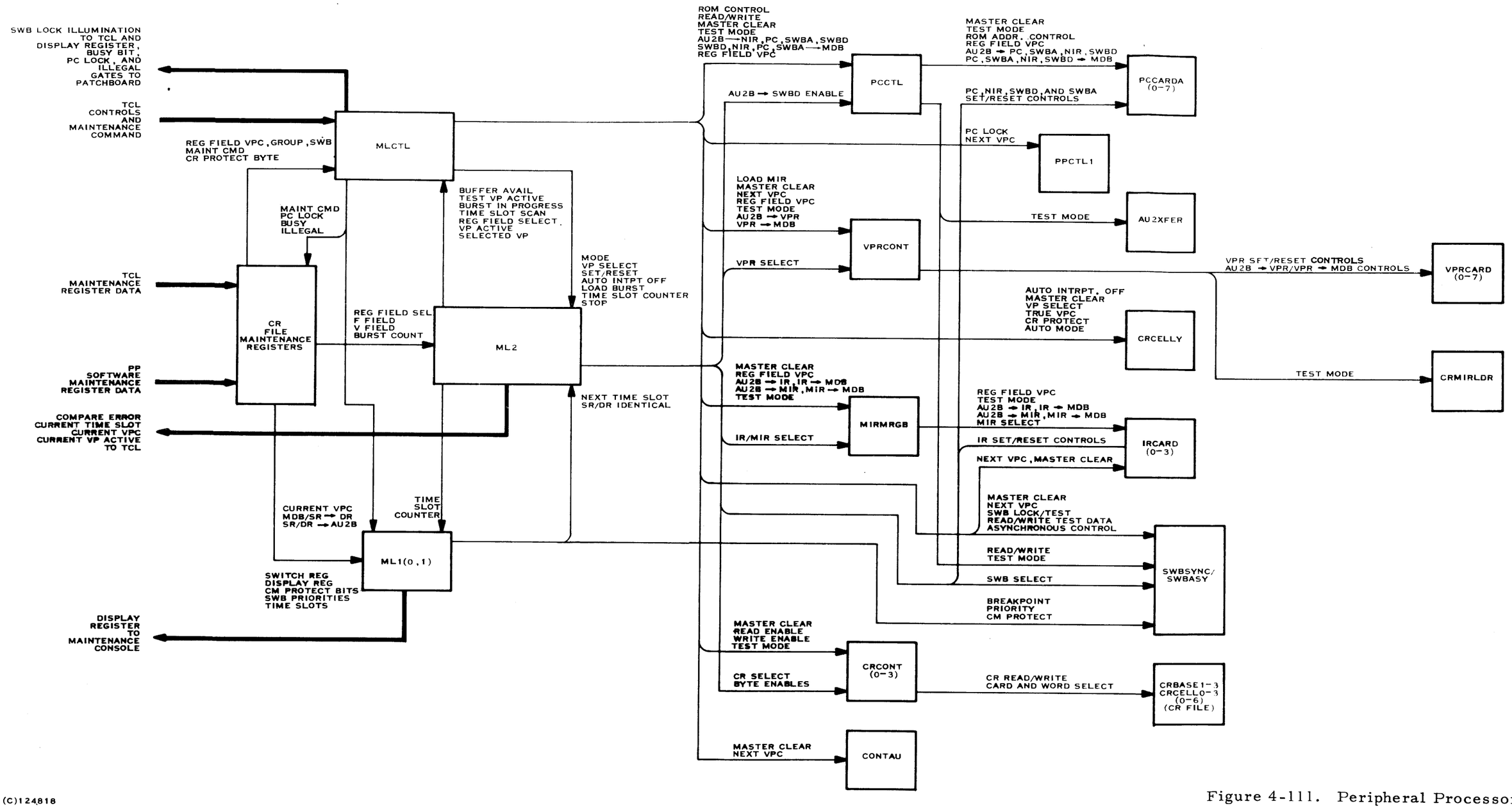


Figure 4-111. Peripheral Processor Maintenance Logic Control Detailed Block Diagram

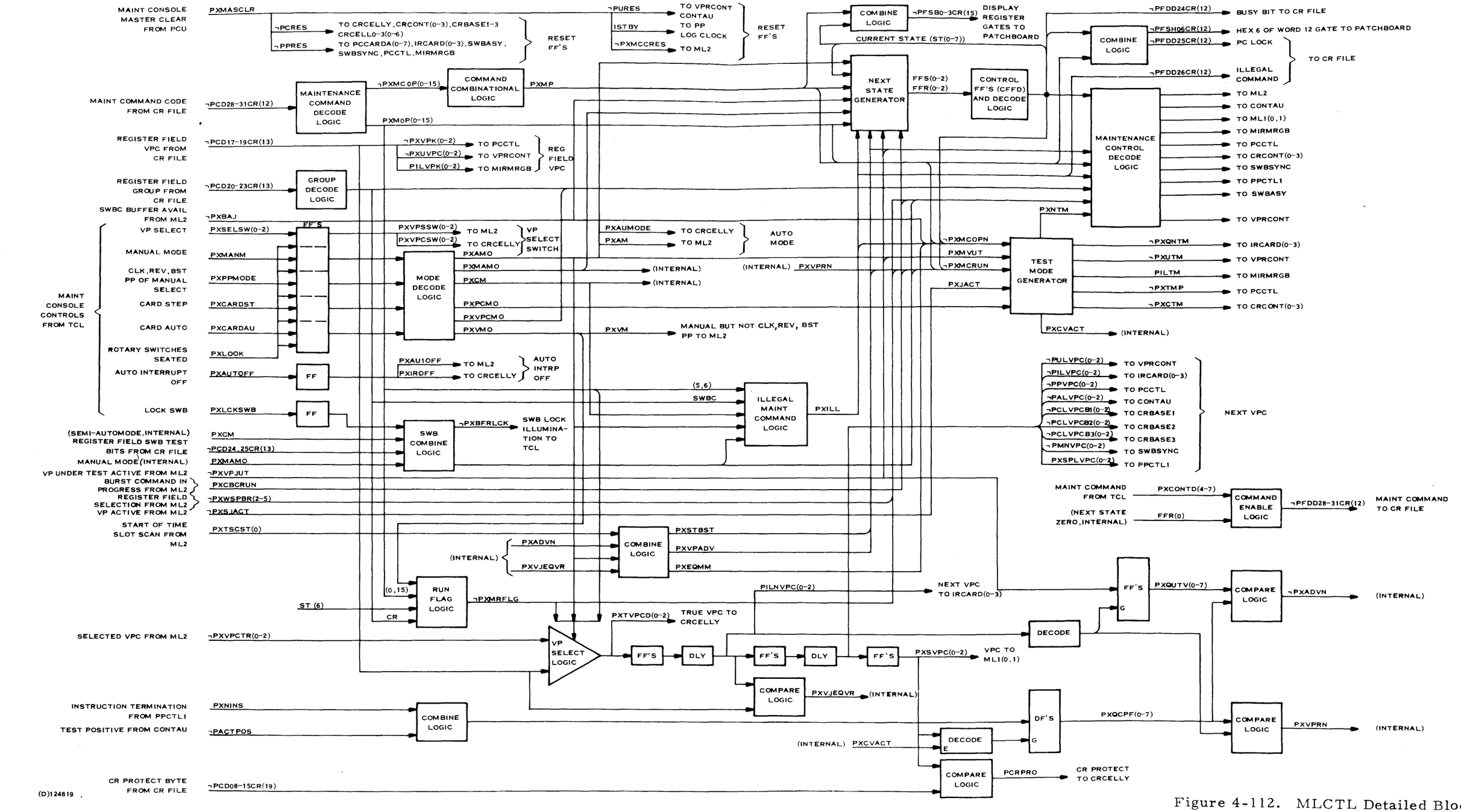


Figure 4-112. MLCTL Detailed Block Diagram

the new maintenance command is shown in figure 4-113. Each of the three bits in the controller takes on a significant meaning as the command progresses to completion: The first bit is true when the controller is busy, the second bit is true for execution states, and the third bit is true for the second step of a two step command.

Beginning in the ready state, the executing command can take one of three steps. If the automatic mode has been chosen and the maintenance logic is waiting for a time slot, the controller is advanced to step 1 of the wait state. If the manual or semi-automatic modes have been chosen and the proper ASC Maintenance Console pushbutton has been pressed to initiate the command, or the lock/unlock PC commands are initiated in any mode, the controller

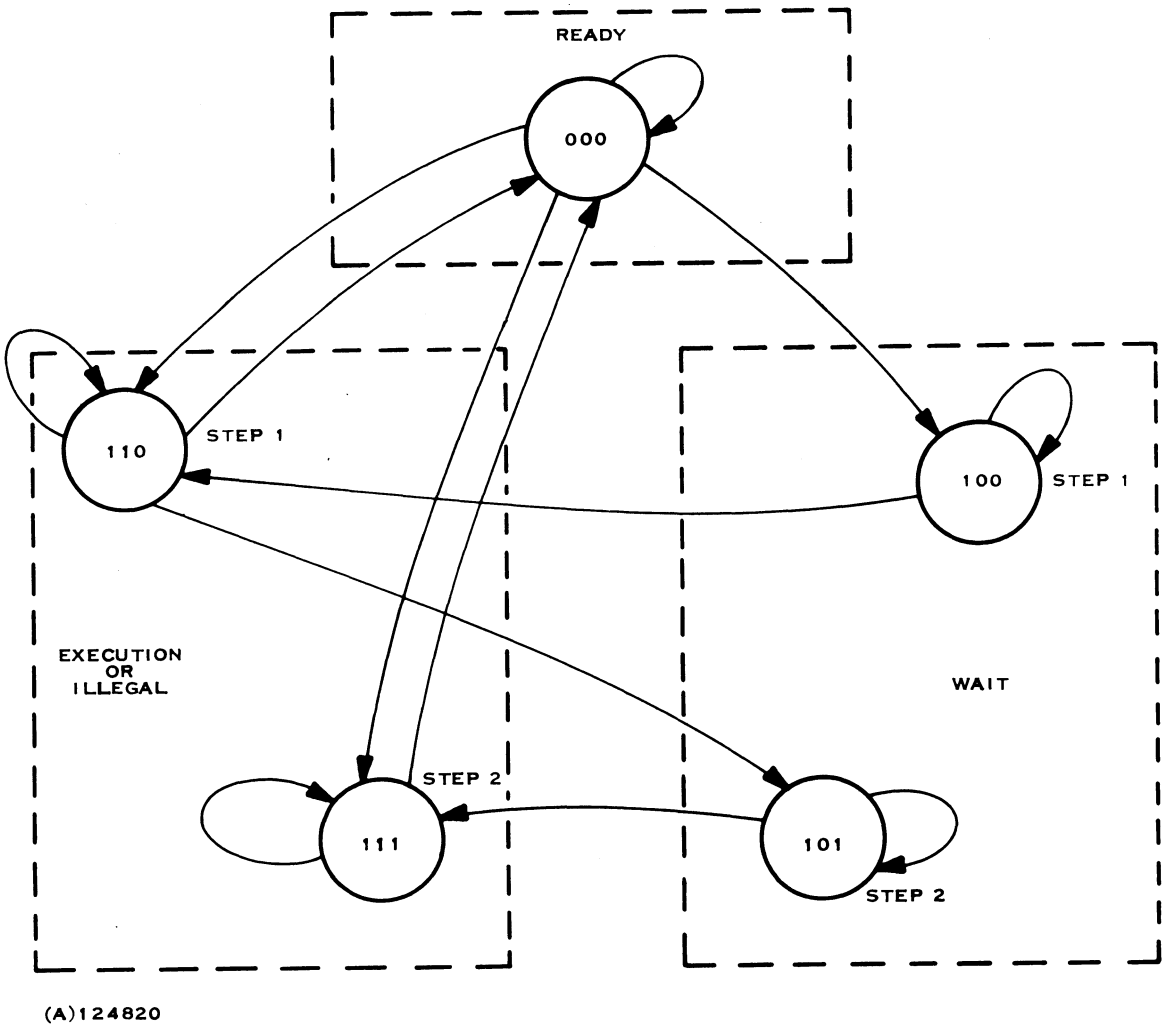


Figure 4-113. Maintenance Controller State Diagram

is advanced to step 1 of the execute state. If the command is illegal, the controller is advanced to step 2 of the execute state (this is also the illegal state). In the first case, the controller remains in step 1 of the wait state until the VP specified by the register field is identical to the VP that will receive a time slot in two clock periods (this is for the data transfer type maintenance commands), or until time slot zero occurs (this is for the type of maintenance commands that release the VP's under test to execute for some period of time). When either of these situations arises for the associated types of commands, the controller is advanced to step 1 of the execute state. In step 1 of the execute state, the MLCTL and ML2 cards combine to develop the controls to lock or unlock the specified PC(s), to enable the specified data transfer, or to release the VP's under test for the period of time specified by the command code. In addition, the controller is returned to the ready state and the busy bit is cleared if a PC(s) was locked or unlocked or the specified data transfer did not involve either CM or ROM. At the completion of a run command (VP's under test released for a specified period of time), the controller is prepared for a new command in the same manner. If a memory is involved in the data transfer command, the controller is advanced to step 2 of the wait state. If data is being read from CM, the wait is necessary for some indication that the read data is available. If data is being written to CM, the wait is necessary until the SWB is available for the write. If data is being read from ROM, the wait is necessary in the automatic mode for the next time slot assigned to the VP under test. In the manual and semi-automatic modes, the second wait state is necessary only because two data transfers are necessary to perform the read and read data transfer. When the mentioned conditions occur for the memory commands, the controller is advanced to step 2 of the execute state to complete the command and prepare the controller for the next command. When the controller is advanced to step 2 of the execute state due to an illegal command, the controller remains in the same state until the illegal conditions are removed by entering a legal command in the maintenance registers. When this happens, the controller is returned to its ready state.

In addition to controlling execution of the maintenance commands and providing the majority of the control signals necessary to make the execution possible (refer to table 4-9 for a list of these control signals and their functions), the MLCTL card provides the remainder of the hardcore maintenance logic and the maintenance/PP interface logic with other essential controls. The MLCTL card uses the ASC Maintenance Console switch settings to supply the ML2 card with the mode, the VP select VP code, the automatic interrupt off indicator, and the master clear. The maintenance command and its state are reflected in the busy, PC lock, and illegal command status indicators returned to the maintenance registers. The ML1(0, 1) cards are supplied with the current VP code from the final level of selection and delay circuitry on MLCTL. The MLCTL card supplies the maintenance/PP interface logic cards with the register field VPC, VP select VP code, mode, automatic interrupt off indicator, master clear, test mode indicator, three next VP codes, and current CR protect indicator.

Table 4-9. MLCTL Maintenance Command Control Signals

Destination Card	Signature	Description
CONTAU	PXAPGATE	Disables all normal control flip-flops in the test mode.
CRCELLY	⌝PXTRIRMC	Enables data to be read from the IR (this signal is inverted by CRCELLY and passed to MIRMRGB).
	⌝PXTRMCIR	Enables data to be loaded in the IR (this signal is inverted by CRCELLY and passed to MIRMRGB).
CRCONT(0-3)	PXCMGATE	Enables all test mode control flip-flops.
	PXCPGATE	Disables all normal control flip-flops in the test mode.
	⌝PXTRCRMC	Enables data to be read from the CR file.
	⌝PXTRMCCR	Enables data to be loaded in the CR file.
IRCARD(0-3)	PIUPGATE	Enables all flip-flops concerned with the next VP code.
MIRMRGB	PIPUGATE	Enables all test mode control flip-flops.
	PITRMCMR	Enables data to be loaded in the MIR.
	PITRMRMC	Enables data to be read from the MIR.
ML1(0,1)	PXSRA2EN:1,2	Selects the switch register when true and the display register when false, and enables the selected register to the AU2XFER bus.
	PXSRDREN:1,2	Selects the switch register when true and the MDB when false, and enables the selected data to the display register.

Table 4-9. MLCTL Maintenance Command Control Signals (Continued)

Destination Card	Signature	Description
ML2	⌉PXCBLOAD	Enables the loading of the burst field in the clock burst counter.
	PXTSCSTP	Stops the time slot counter.
	PXVPRES	Reset control developed for command code A_{16}.
	PXVPSET	Set control developed for command code B_{16}.
PCCTL	PXCMRC	Read request.
	⌉PXCMWC	Write request.
	PXPMGATE	Enables all test mode control flip-flops.
	PXPPGATE	Disables all normal control flip-flops in the test mode.
	⌉PXROMNR	Enables ROM data into a NIR.
	⌉PXTRMAMC	Enables the SWBA to MDB transfer.
	⌉PXTRMCMA	Enables the AU2B to SWBA transfer.
	⌉PXTRMCMD	Enables the AU2B to SWBD transfer.
	⌉PXTRMCNR	Enables the AU2B to NIR transfer.
	⌉PXTRMCPC	Enables the AU2B to PC transfer.
	⌉PXTRMDMC	Enables the SWBD to MDB transfer.
	⌉PXTRNRMC	Enables the NIR to MDB transfer.
	⌉PXTRPCMC	Enables the PC to MDB transfer.
PPCTL1	PPLOCKPC	Locks the PC(s) when true and unlocks the PC(s) when false.
SWBASY	PMCMTLE2	Enables asynchronous control of the SWBC.
	PMMME	Stops the PP asynchronous/MCU interface on an even state.
	PMMMT	Stops the PP asynchronous/MCU interface on an odd state.

Table 4-9. MLCTL Maintenance Command Control Signals (Continued)

Destination Card	Signature	Description
SWBSYNC	PMCMTLE1	Enables test mode control of the SWBC data.
	PMLTMR	Enables data to be read from the SWBC.
	PMLTMW	Enables data to be written to the SWBC.
	PMTRMCBC	Locks the SWBC for the test mode.
VPRCONT	⌐PITRMCMR	Enables the AU2B to MIR transfer.
	⌐PXTRMCVP	Enables the AU2B to VPR transfer.
	⌐PXTRVPMC	Enables the VPR to MDB transfer.
	PXUPGATE	Disables all normal control flip-flops in the test mode.
	PXUMGATE	Enables all test mode control flip-flops.

The ML2 card, detailed in figure 4-114, supplies the MLCTL card with some of the signals necessary for the maintenance controller operation, the ML1(0, 1) cards with the current time slot entry, the TCL with PP status indicators, and the maintenance/PP interface logic with the remaining maintenance controls not supplied by MLCTL. The individual signals supplied to MLCTL include the VP code and active indicator to be used at the third succeeding time slot, the SWBC availability indicator to be used at the second succeeding time slot, an indication of whether or not the VP under consideration in two time slots is under test, the register field selection bits, an indication of whether or not a burst type command is in progress, and an indication of when a new time slot table scan is starting. The PP status indicators supplied to the TCL include the time slot number associated with the current VP, the current VP code, an indication of whether or not the current VP is active, and the compare error signal used to reflect when the contents of the switch and display registers are identical. The maintenance control supplied to the maintenance/PP interface logic includes the register selection controls developed from the register field selection bits, the CR and VPR byte and halfword enables developed from the F field, and the register reset and set controls developed from the signal supplied by MLCTL when command codes A_{16} and B_{16}, respectively, are detected.

The ML1(0, 1) cards, detailed in figure 4-115, provide the hardcore maintenance logic data handling capabilities, but also supply some support to the

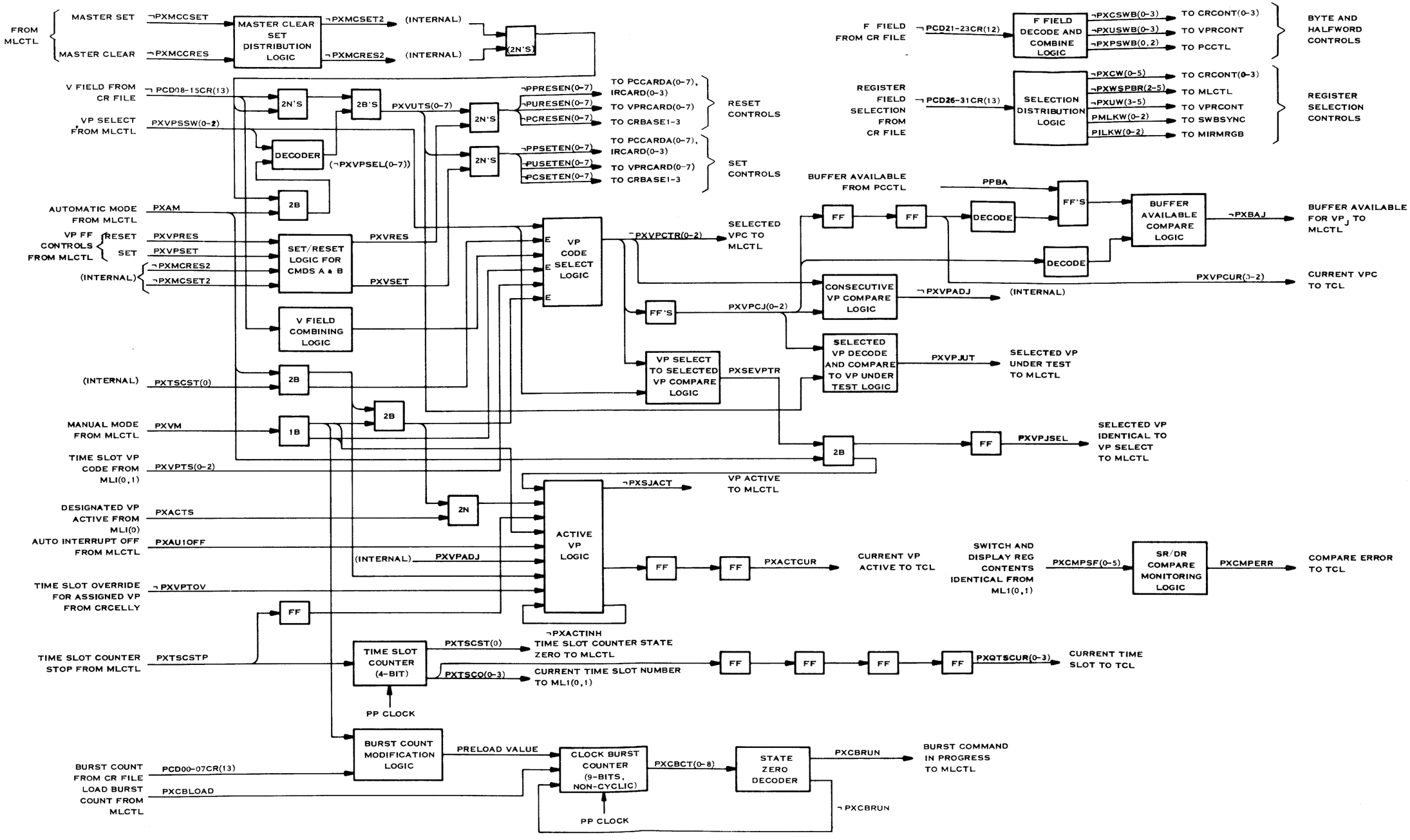


Figure 4-114. ML2 Detailed Block Diagram

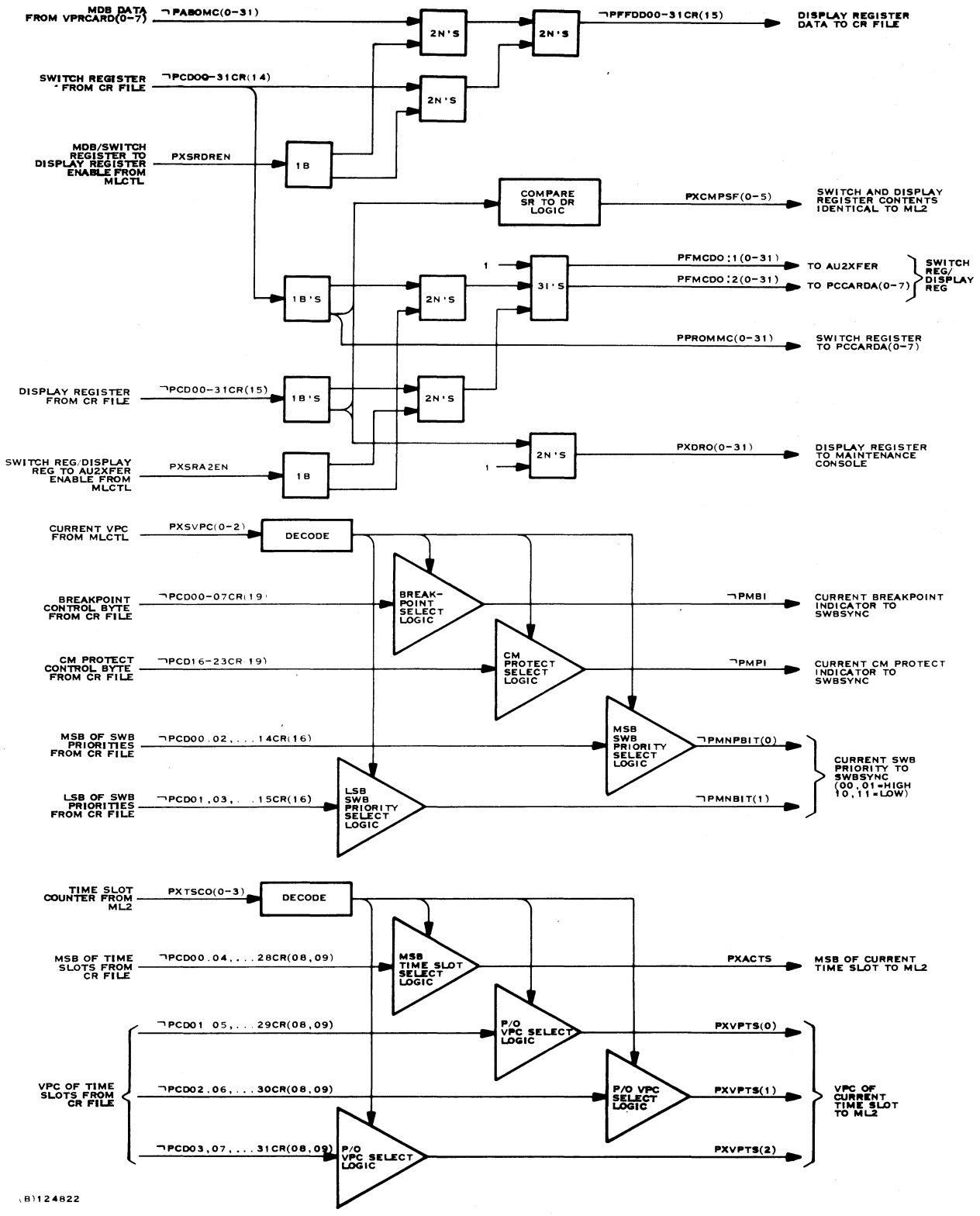


Figure 4-115. ML1(0, 1) Detailed Logic Diagram

control circuitry on ML2 and SWBSYNC. The ML1(0, 1) to ML2 interface consists of the time slot data pointed to by the time slot counter on ML2 and the ML1(0, 1) to SWBSYNC interface includes the current priority, CM protect indicator, and breakpoint indicator (not used) selected with the current VP code.

4-206 Maintenance/PP Interface Logic. The maintenance/PP interface logic that involves control is located on the PCCTL, VPRCONT, MIRMRGB, CRCONT(0-3), CONTAU, PPCTL1, CRCELLY, SWBSYNC, SWBASY, PCCARDA(0-7), VPRCARD(0-7), IRCARD(0-3), AU2XFER, CRMIRLDR, CRBASE1-3, and CRCELL0-3(0-6) cards. The primary purpose of the maintenance/PP interface logic on each of these cards is to accept and interpret maintenance related control signals from the hardcore maintenance logic so the desired maintenance operation can be performed. A few of the signals supplied by the hardcore maintenance logic to the maintenance/PP interface logic in figure 4-111 are essential to normal PP operation and do not involve maintenance control (these include the CM protect indicator, CR protect indicator, and SWBC priority).

The maintenance related logic on the PCCTL card, detailed in figure 4-116, is enabled by the test mode indicator (⅂PXTMP) simultaneously with the disabling of the normal control flip-flops (via PXPPGATE). The gates to all of the test mode flip-flops are continuously held to a logic one by the PXPMGATE signal, so when the hardcore maintenance logic indicates test mode operation, the maintenance control signals are interpreted and routed to their destination. The majority of the control signals developed on PCCTL are input to the PCCARDA(0-7) cards (refer to figure 4-117) to read from and write to the SWBD, SWBA, PC, and NIR. The PCCTL card also supplies PCCARDA(0-7) with the controls necessary to read from ROM. The PCCTL card contains the logic to transfer the test mode indicator to SWBSYNC and to develop the maintenance read and write request signals for SWBSYNC. The master clear signal input to PCCTL is used to clear both the normal and test mode control flip-flops.

The maintenance related logic on the VPRCONT card, detailed in figure 4-118, is enabled and the normal control logic is disabled in a manner similar to that described for PCCTL. In the test mode, the register field VP code combines with the VPR to MDB enable to develop one of four possible VPR to MDB enables for VPRCARD(0-7), with the three LSB's of the register field selection bits to develop the VPR to MDB word select and the AU2B to VPR word select, both for VPRCARD(0-7), and with the AU2B to VPR enable and the individual byte enables to develop the AU2B to VPR byte enables for VPRCARD(0-7). The MIR load enable combines with the two LSB's of the register field selection bits to develop the CRMIR load enable (for CRMIRLDR), VPRMIR load enable, and AUMIR load enable. Each of these MIR type enables gates AU2B data through to the associated MIR control flip-flops.

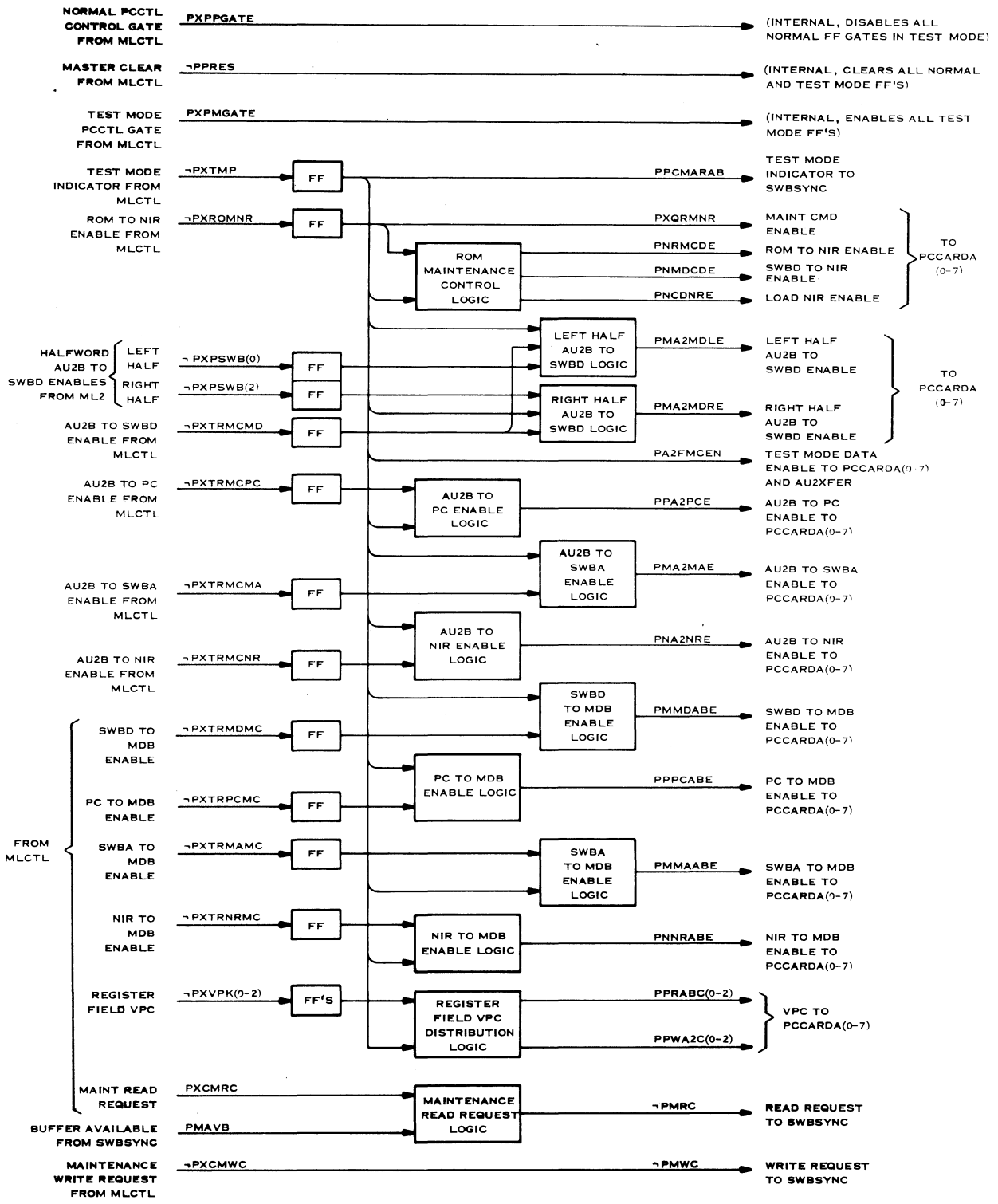


Figure 4-116. Maintenance Related Logic on PCCTL

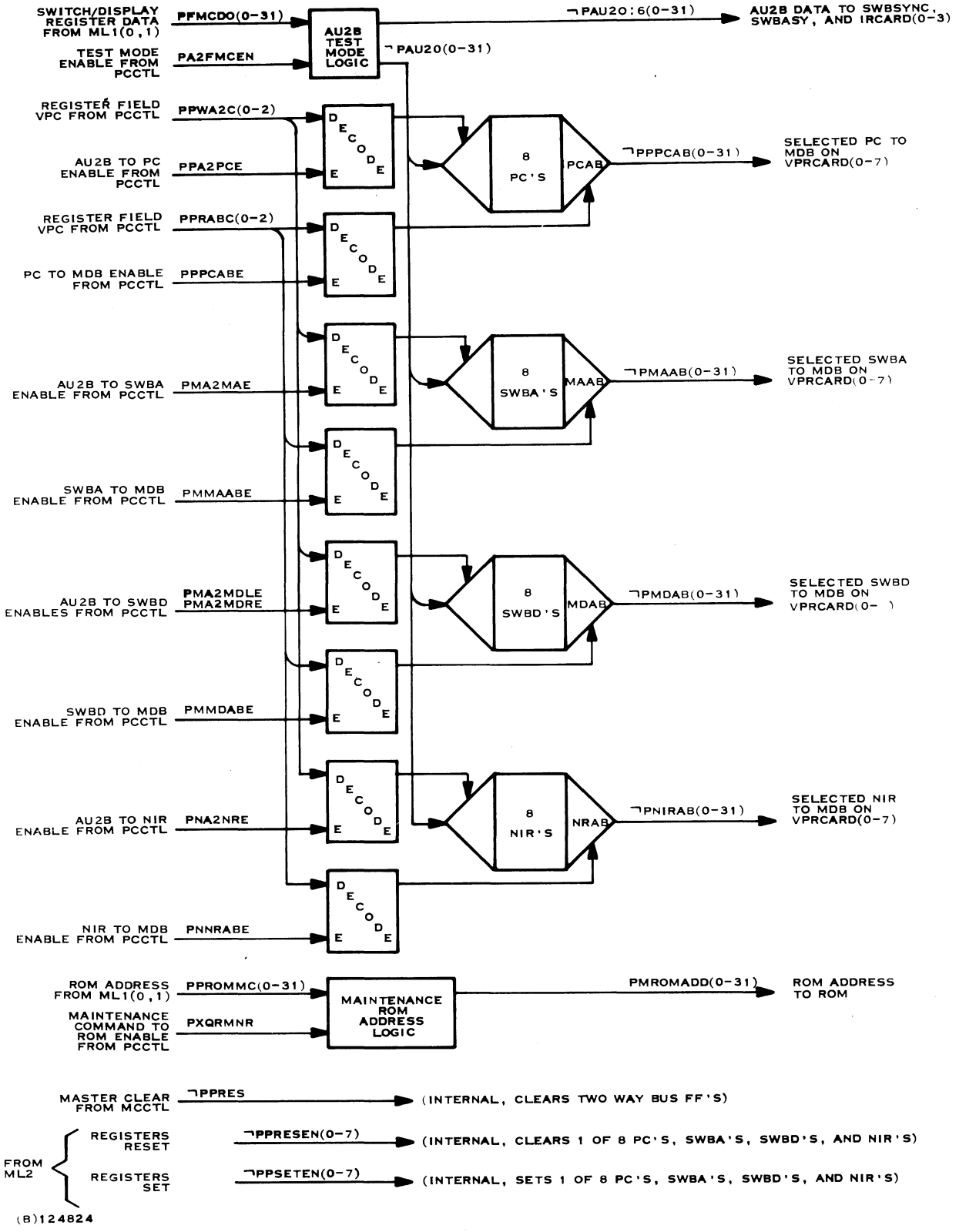


Figure 4-117. Maintenance Related Logic on PCCARDA(0-7)

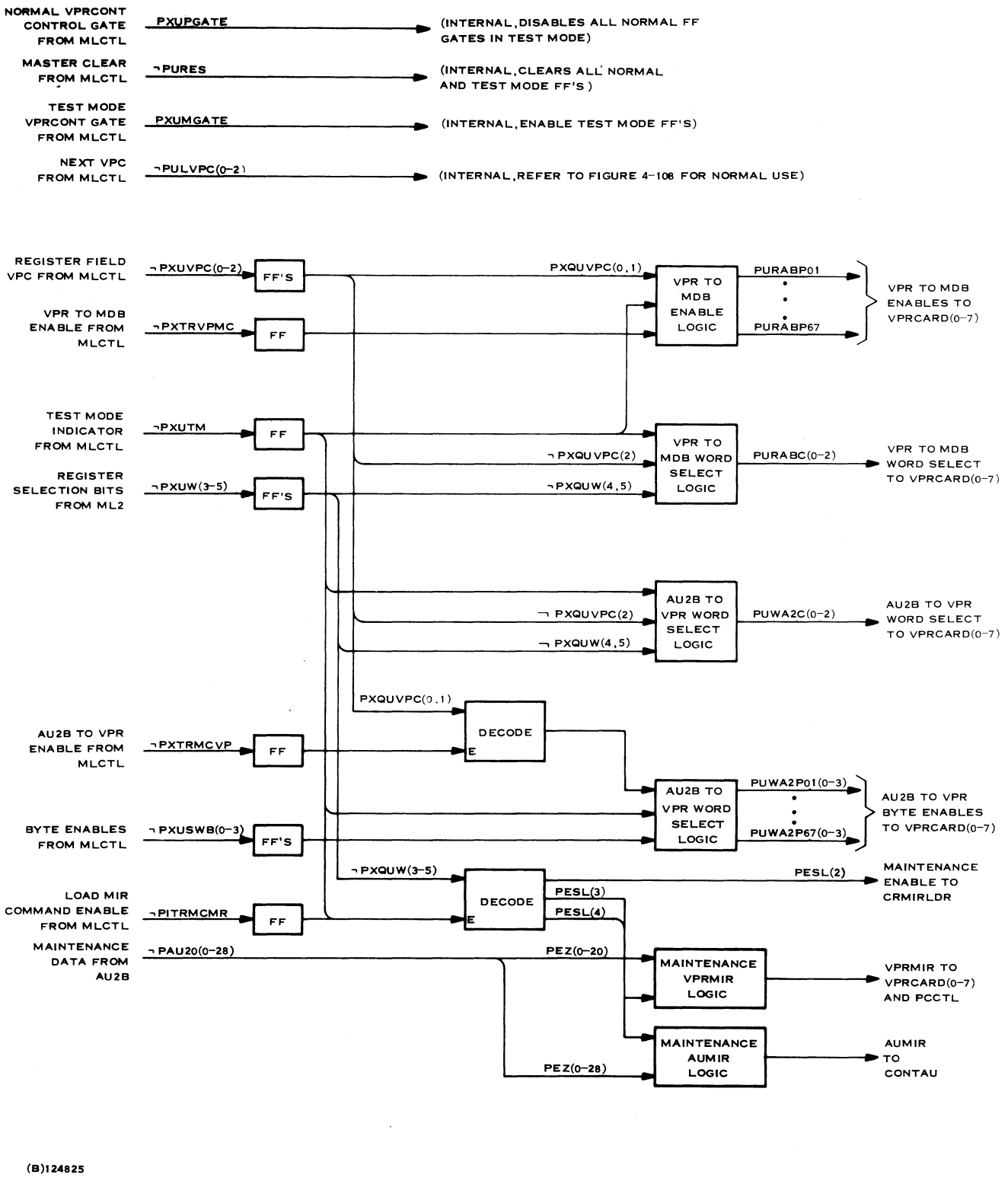


Figure 4-118. Maintenance Related Logic on VPRCONT

The maintenance logic on the MIRMRGB card, detailed in figure 4-119, is primarily concerned with transferring and expanding the maintenance controls necessary for handling the IR's and MIR located on IRCARD(0-3). The register field VP code, read and write IR and MIR enables, and test mode indicator are all gated through a single level of flip-flops on MIRMRGB to IRCARD(0-3). The three LSB's of the register field selection bits are expanded to develop the MIR word 0, 1, and 7 read and write enables. The LSB of the same field is passed to IRCARD(0-3) in true and complement form to develop the IR left and right half read and write enables (only half of the selected 64-bit IR can be read from or written to at one time because the maintenance logic data transfers can handle only 32 bits). The MIR read enable combines with the three LSB's of the register field selection bits to develop the MIR read enables necessary for selection of MIR words two through six.

The maintenance related logic on the CRCONT(0-3) cards, detailed in figure 4-120, expands the hardcore maintenance logic control signals concerning the CR file to produce the CR file read and write card and word select controls. In the test mode, the three MSB's of the register field selection bits combine with the maintenance CR file read enable to develop the CR read card enable. The same bits combine with the CR file write enable and associated byte enable to develop the CR write card enable. The three LSB's from the same maintenance register field are enabled by the test mode indicator for the development of the CR read and write word select. The read and write card and word select signals developed on each of the CRCONT cards are distributed to the associated CR file data holding cards (CRCONT0 controls go to CRCELLY and CRCELL0(0-6), CRCONT1 controls go to CRBASE1 and CRCELL1(0-6), etc.).

The maintenance related logic on the SWBSYNC and SWBASY cards, detailed in figure 4-121, provides the control necessary to read from and write to the SWBC flip-flops in the test mode. In addition, the asynchronous control logic adds the option of stopping the SWBC/MCU asynchronous interface at different steps in the communication procedure (communications between the SWBC and MCU are described in both the general and detailed SWBC theory). The priority, CM protect, and breakpoint (not used) signals input to SWBSYNC are essential to normal PP operations and are included in figure 4-121 only because they are developed by the maintenance logic. When maintenance data is to be written to the SWBC, the write enable (PMLTMW), SWB lock/test control (PMTRMCBC, this signal indicates when the SWB LOCK switch on the ASC Maintenance Console is in the lock position when the manual mode has been selected or when the SWB is under test in the semi-automatic mode), and three LSB's of the register field selection bits (PMLKW(0-2)) combine to enable AU2B data into the selected SWBC flip-flops. Refer to figure 4-69 for a map of the synchronous and asynchronous flip-flops that are affected when $SWBC_0$ through $SWBC_7$ are selected as the destination of the write. When data is to be read from the SWBC, the read enable (PMLTMR) combines

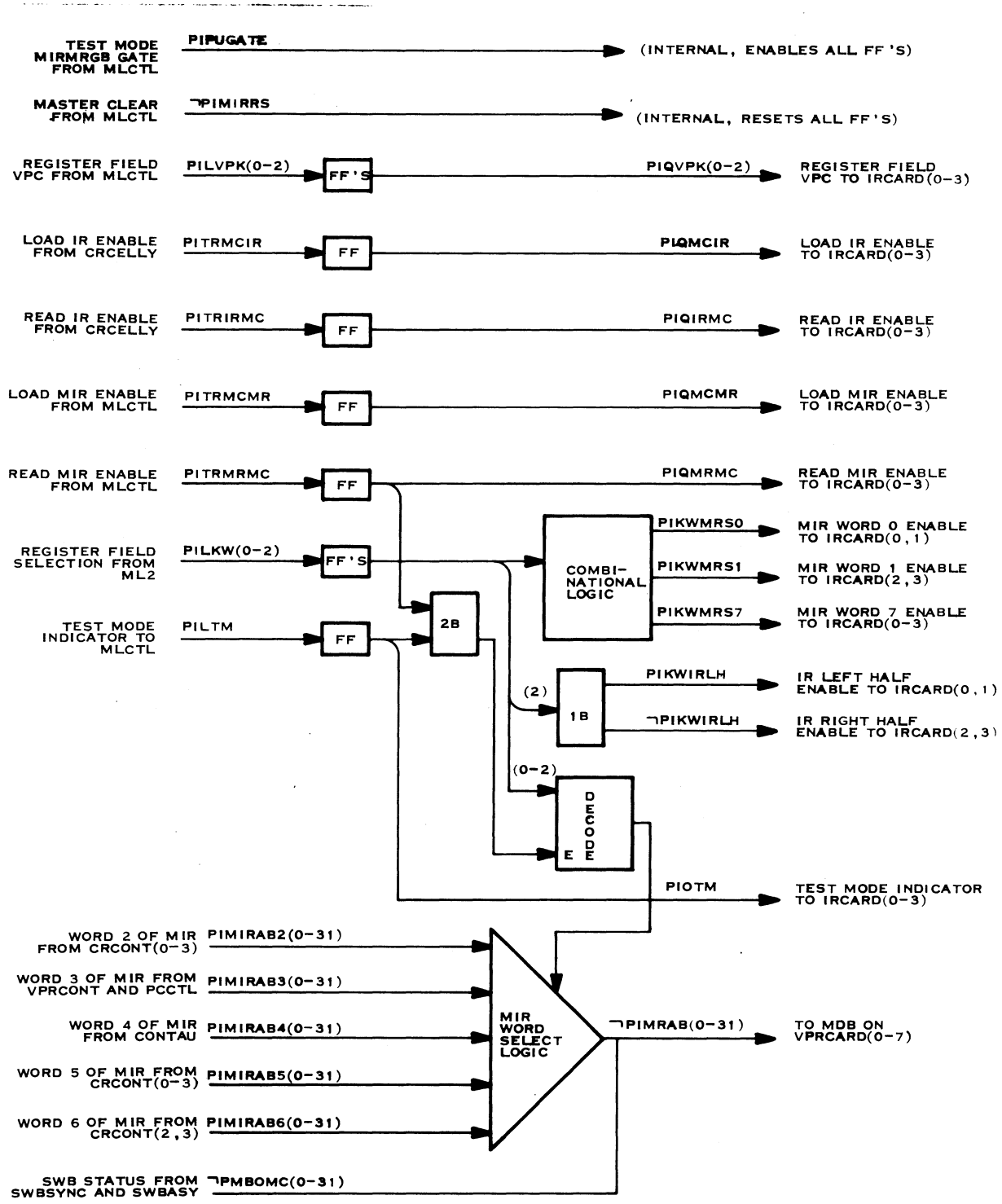


Figure 4-119. Maintenance Logic On MIRMRGB

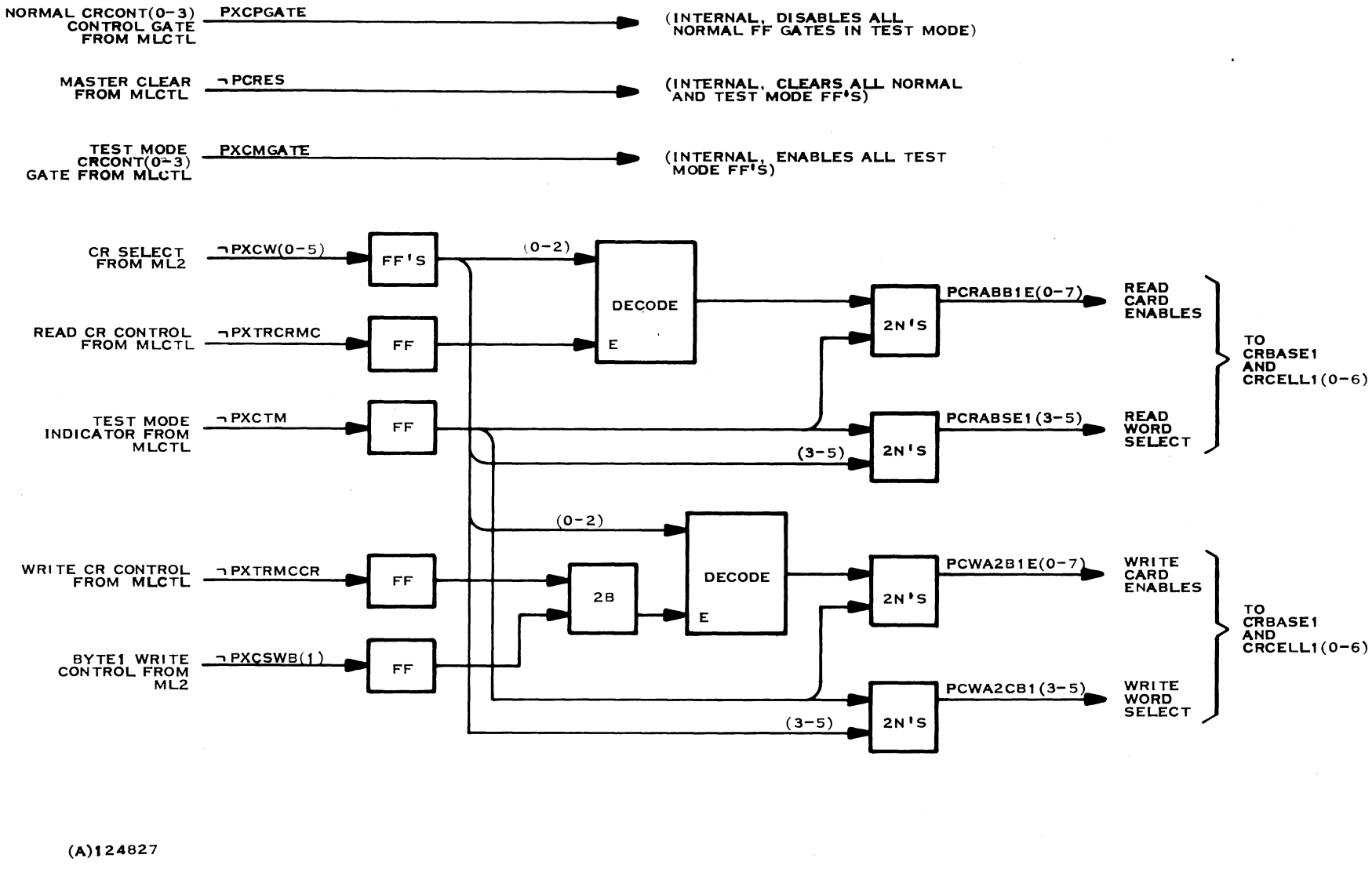


Figure 4-120. Maintenance Related Logic on CRCONT1

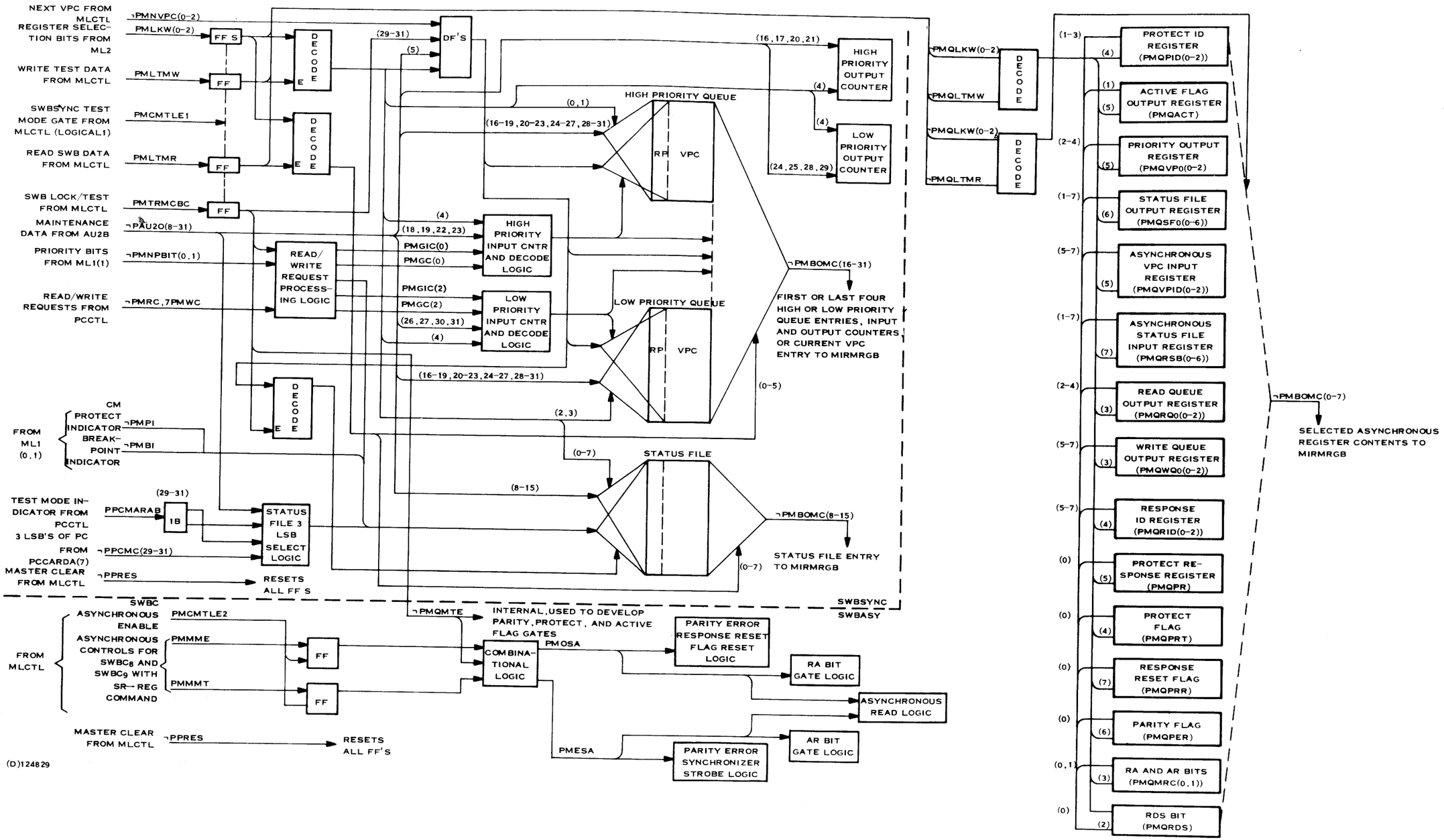


Figure 4-121. Maintenance Related Logic on SWBSYNC/SWBASY

with the three LSB's of the register field selection bits to select one of the eight SWBC flip-flop combinations ($SWBC_0$ through $SWBC_7$) for output to the MDB. When the asynchronous SWBC/MCU interface is to be temporarily halted for examination, the PMCMTLE2, PMMME, and PMMMT control signals combine to disable the memory access request (AR) flip-flop gate or the memory request accepted (RA) flip-flop gate when $SWBC_9$ or $SWBC_8$, respectively, is used in conjunction with maintenance command code 5.

The maintenance related logic on the IRCARD(0-3) cards, detailed in figure 4-123, interprets the IR and MIR read and write control signals supplied by the MIRMRGB card. When data is to be written to half of one of the eight IR's, the three LSB's of the register field selection bits (PIQVPK(0-2)), the write IR enable (PIQMCIR), and the desired left (PIKWIRLH) or right (¬PIKWIRLH) half enable combine to transfer AU2B data into the left or right half of the desired IR. When data is to be read from half of one of the eight IR's, the three LSB's of the register field selection bits, the read IR enable (PIQIRMC), and the left or right half enable combine to perform the read. When data is to be written to or read from the IRMIR (words 0 and 1 of the MIR) or word 7 of the MIR, the PIQMCMR signal enables a write, the PIQMRMC signal enables a read, and the PIKWMRS0, PIKWMRS1, and PIKWMRS7 signals select word 0, 1, or 7, respectively.

The maintenance related logic on the CRMIRLDR card, detailed in figure 4-56 of the CR file detailed theory, is enabled by the previously mentioned CRMIR load enable developed by VPRCONT to insert the AU2B test data in the CRMIR flip-flops. The maintenance logic on the AU2XFER card, detailed in figure 4-122, enables maintenance data to the AU2B bus under control of the test mode indicator supplied by PCCTL. The logic on the VPRCARD(0-7) cards that responds to the VPR read and write maintenance controls is also used during normal PP processing and is described in the VPR file portion of the VP detailed theory. The control lines in figure 4-111 extending to the CONTAU, CRCELLY, and PPCTL1 cards are discussed in the detailed theory covering AU control, CR file control, and PP control, respectively.

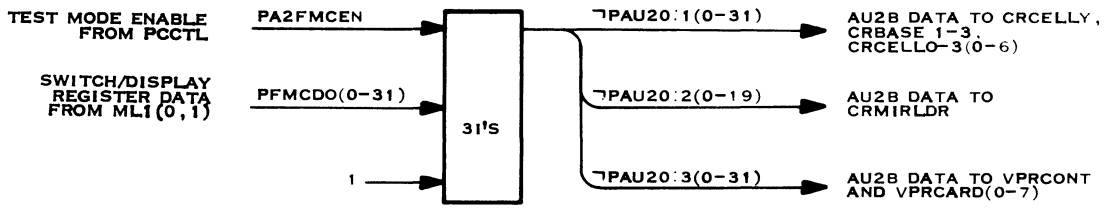


Figure 4-122. Maintenance Related Logic on AU2XFER

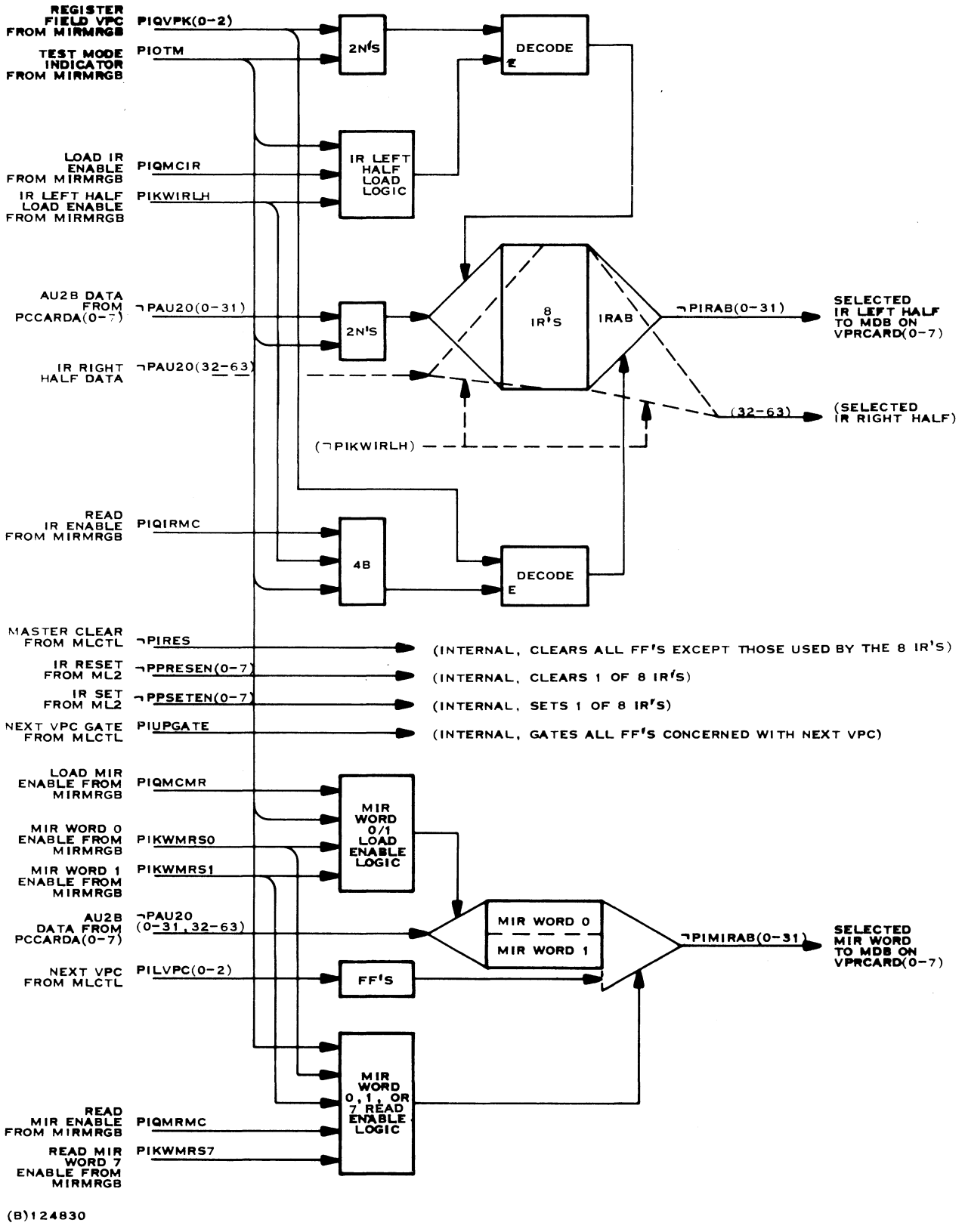


Figure 4-123. Maintenance Related Logic on IRCARD(0-3)

4-207 MAINTENANCE LOGIC DATA PATHS. The maintenance logic data path structure provides the maintenance logic with the data paths necessary to carry out execution of the maintenance command codes 1 through 7. Figure 4-124 shows the logic cards and the type of data that is transferred between the logic cards in the maintenance data path system. Most of the lines connecting the logic cards in figure 4-124 have a capacity of 32 bits, the only exceptions being AUMIR input, CRMIR input, and VPRMIR input lines supplied by the VPRCONT and CRMIRLDR cards and the CRMIR line input to the MIRMRGB card. These MIR type formats are described in the PP control detailed theory.

In the manual or semi-automatic modes of operation, data is entered in the PP maintenance logic system via the switch register portion of the maintenance registers. In the automatic mode of operation, PP software is capable of entering data in either the switch register or display register. Both the switch and display register plus MDB data are applied to the ML1(0, 1) card, as shown in figure 4-115. In the case of maintenance command code 1 (switch register transferred to the display register), maintenance logic control enables the switch register data through ML1(0, 1) back to the display register. In the case of maintenance command codes 2, 3, or 4, the MDB data is selected for transfer to the display register. For commands 3 and 4, this data transfer represents the retrieval of CM and ROM data, respectively. In the case of command codes 3, 4, 5, 6, and 7, switch or display register data is transferred through the ML1(0, 1) cards to the AU2XFER or PCCARDA(0-7) cards. For commands 3, 4, and 7, this data transfer routes an address to the associated memory. For command 7, this transfer is used a second time when the display register is written to CM. In addition to the mentioned data transfers that occur under direction of the maintenance logic control, the display register is continuously applied to the ASC Maintenance Console in case the DISPLAY REGISTER switch is pressed and the switch register is continuously applied to the ROM address logic on PCCARDA(0-7) in case data is to be read from ROM.

The PCCARDA(0-7) cards, shown in figure 4-117, accept switch register data in case a maintenance ROM read is to be performed and switch or display register data when a transfer to one of the associated PP registers is desired. When ROM data is to be retrieved (command 4), the switch register address is enabled through PCCARDA(0-7) to the ROMCRD(0-15) cards. When ROM responds to the address, the retrieved word is transferred back to the NIR on PCCARDA(0-7) associated with the VP specified by the maintenance registers. When switch or display register data is to be written to one of the PC's, SWBA's, SWBD's, or NIR's (commands 5 and 6), the input data is enabled into the desired register via maintenance logic control. The switch or display register data is also distributed to the SWBSYNC/SWBASY and IRCARD(0-3) cards from PCCARDA(0-7). When PC, NIR, SWBA, or SWBD data is to be transferred to the display register (commands 2, 3, or 4), maintenance logic control enables the desired PCCARDA(0-7) register to the MDB for input to the ML1(0, 1) cards.

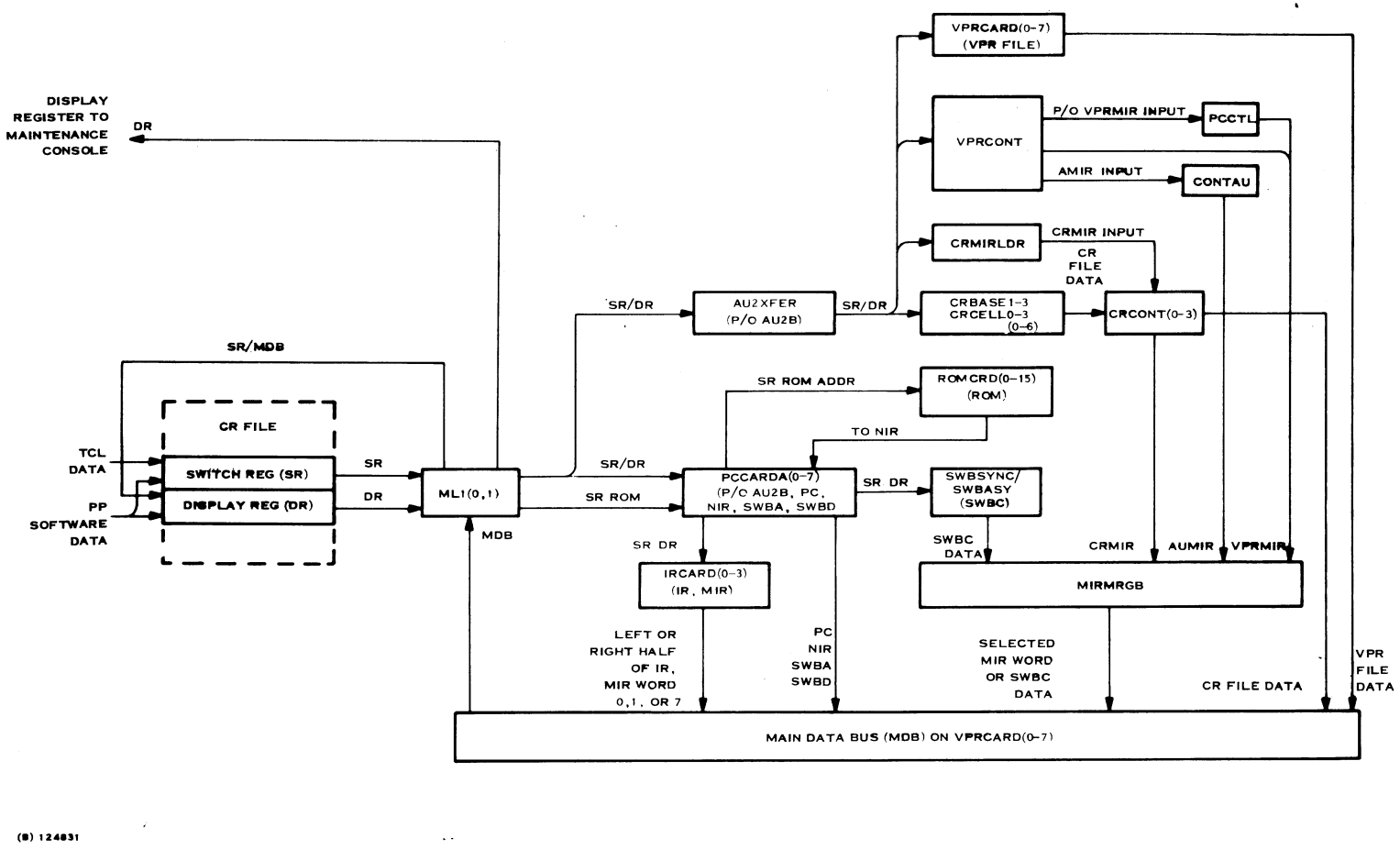


Figure 4-124. Peripheral Processor Maintenance Logic Data Path Detailed Block Diagram

The SWBSYNC and SWBASY cards, shown in figure 4-122, accept AU2B bus switch or display register data from PCCARDA(0-7). When a SWBC is specified as the destination in conjunction with command codes 5 or 6, the input switch or display register data is enabled into the desired combination of SWBC synchronous and asynchronous flip-flops. When SWBC data is to be transferred to the display register (command 2), the desired data is enabled through the MIRMRGB card to the MDB for transfer to the ML1(0,1) cards. The IRCARD(0-3) cards, shown in figure 4-123, also accept AU2B bus switch or display register data from PCCARDA(0-7). When half of an IR or words 0 or 1 of the MIR is specified as the destination in conjunction with command codes 5 or 6, the input switch or display register data is enabled into the desired IRCARD(0-3) register. When data from one of the same registers (or the VP code in the register assigned as MIR word 7) is to be transferred to the display register (command 2), the desired register is transferred to the MDB for input to the ML1(0,1) cards.

The AU2XFER card, shown in figure 4-121, accepts switch or display register data from the ML1(0,1) cards when a CR or VPR or the AUMIR, CRMIR, or VPRMIR is specified as the destination register in conjunction with command codes 5 or 6. When maintenance data is to be written to a VPR, the switch or display register data is enabled into the desired VPR on the VPRCARD(0-7) cards. When maintenance data is to be written to a CR, the switch or display register data is enabled into the desired CR on the CRBASE1-3, CRCELLY, or CRCELL0-3(0-6) cards. Switch or display register data is enabled into the AUMIR or VPRMIR logic on VPRCONT or the CRMIR logic on CRMIRLDR when one of the MIR type registers is specified as the destination. The input forms of AUMIR, CRMIR, and part of VPRMIR are transferred to CONTAU, CRCONT(0-3), and PCCTL, respectively, during normal processing of the PP.

The MIRMRGB card, shown in figure 4-119, monitors the output forms of AUMIR, CRMIR, and VPRMIR. When the AUMIR, part of the CRMIR (the complete CRMIR output format consists of three 32-bit words), or VPRMIR is specified in conjunction with command code 2, the maintenance logic control on MIRMRGB enables the desired portion of the MIR to the MDB for input to ML1(0,1). When a CR or VPR is specified in conjunction with command code 2, the maintenance logic control enables a VPR from VPRCARD(0-7) or a CR from CRCONT(0-3) to the MDB for input to ML1(0,1).

4-208 MAINTENANCE LOGIC/PP TIMING. The PP maintenance logic, in addition to providing the controls and data paths necessary to execute maintenance commands, provides the remainder of the PP with the current VP code and an indication of whether the PP is operating in the normal mode or under control of the maintenance logic. A timing diagram illustrating the procedure involved in determining and distributing the current VP code and mode of operation (normal or test) is presented in figure 4-125. The abbreviations that occur in the diagram are explained in table 4-10.

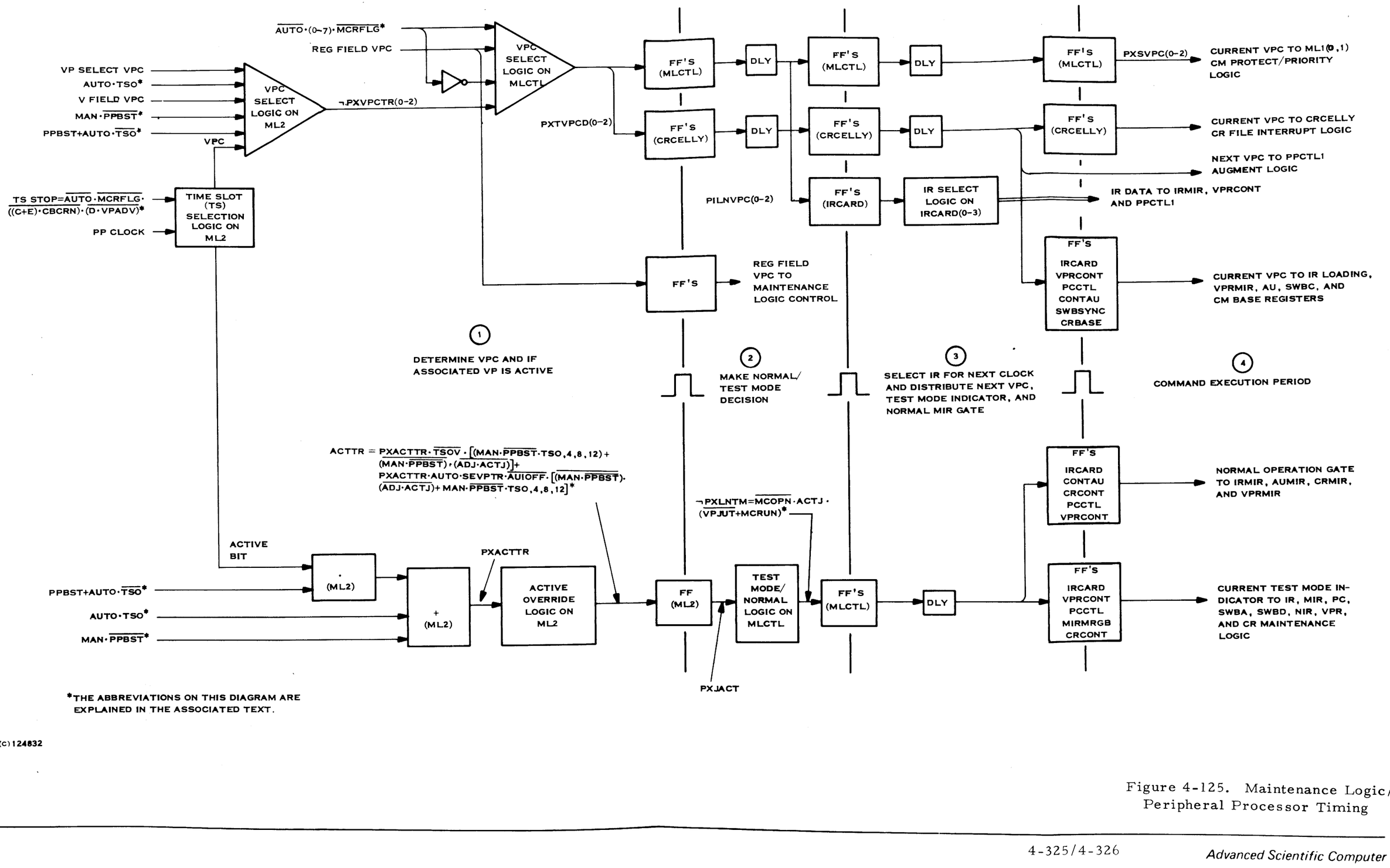


Figure 4-125. Maintenance Logic/ Peripheral Processor Timing

Table 4-10. Maintenance Logic Abbreviations

Abbreviated Term	Signature	Description
ACTJ	PXACTSJ	The active bit associated with VP_J (VP_J will be the active VP on the second succeeding clock period).
ACTTR	PXACTJD1	The active bit associated with VP_{TR} (VP_{TR} will be the active VP on the third succeeding clock period).
ADJ	PXVPADJ	Indicates when VP_J and VP_{TR} are identical (one VP assigned to two consecutive time slots).
AUIOFF	PXAU1OFF	Indicates the status of the AUTO INTERRUPT OFF switch on the ASC Maintenance Console.
AUTO	PXAM	Indicates when the PP is operating in the automatic or normal mode.
CBCRN	PXCBCRN	Indicates when the clock burst counter is counting (burst in progress) in conjunction with maintenance command codes C_{16} and E_{16}.
MAN	PXMAMO	Indicates when the PP is operating in the manual mode.
MCOPN	PXMCOPN	Indicates when maintenance command codes 1 through 7 and A_{16} and B_{16} are ready to execute. In effect, this signal reserves the second succeeding clock period for execution of the command. When one of the mentioned commands has been entered in the maintenance registers (either via TCL or PP software), the PXMCOPN signal follows the PXEQMM signal, where, $PXEQMM=\overline{AUTO}+(\overline{TSO}+\overline{AUTO})\cdot PXVJEQVR$ The PXVJEQVR signal indicates when VP_J is identical to the VP specified by the register field.

Table 4-10. Maintenance Logic Abbreviations (Continued)

Abbreviated Term	Signature	Description
MCRFLG	PXMCRFLG	Indicates when maintenance command code F_{16} is running in the continuous mode. This signal is reset as soon as a new maintenance command is received.
MCRUN	PXMCRUN	Indicates when a maintenance controlled run or burst is executing (commands C_{16}, D_{16}, E_{16}, and F_{16}). PXMCRUN=[((C+E)· CBCRN· ST6) +(D· PXVPADV· PXVPRN) +(O· STO· MCRFLG)] ·[C· PXSWBCHK] where; ST signifies state. O, C, D, and E are command codes. PXVPADV is true if the VP's under test have not completed their current instruction. PXVPRN is true if VP_J is still executing the instruction that was executing when maintenance command code D_{16} was initiated. PXSWBCHK is true when the SWBC is locked in the manual mode or placed under test in the semi-automatic mode.
PPBST	PXPPMODE	Indicates when the ASC Maintenance Console MANUAL SELECT switch is set to the CLK PP, REV PP, or BST PP positions in the manual mode of operation.
SEVPTR	PXSEVPTR	Indicates when the VP SELECT VP is identical to VP_{TR}.
TSOV	PXVPTOV	Indicates when a time slot assigned to an active VP is overriden (reasons for time slot override are discussed in the CR file detailed theory).

Table 4-10. Maintenance Logic Abbreviations (Continued)

Abbreviated Term	Signature	Description
TS0, 4, 8, 12	(none)	Indicates the occurrence of time slot 0, 4, 8, or 12.
VPADV	PXVPADV	Indicates when the VP's under test have not completed their current instruction (this signal is used in conjunction with command code D_{16}).
VPJUT	PXVPJUT	Indicates when VP_J is under test (the VP's under test are marked by logical one's in the V field).
0-7, C, D, E	⌐PXCF4, PXMPCE, PXMOP(13)	Indicates the occurrence of maintenance command codes 0 through 7, C, D, or E in the command code field of the maintenance registers.

4-209 VP Code Timing Logic. The VP code select logic on ML2 accepts mode and time slot data for use in selecting one of three VP code sources as the supplier of the active VP on the third succeeding clock. When the manual mode has been selected and a PP burst operation has not been chosen (MAN• $\overline{\text{PPBST}}$), the V field specified VP code is enabled in the development of VP_{TR} (PXVPCTR(0-2), the active VP code on the third succeeding clock). When the automatic or normal mode has been selected and the current time slot under consideration is time slot zero (AUTO• TSO), the VP code associated with the VP SELECT switch setting (on the ASC Maintenance Console) is used in the development of VP_{TR}. When the manual mode has been selected in conjunction with a PP burst operation or the normal mode has been selected and the current time slot is other than time slot zero, the VP code used in the development of VP_{TR} is retrieved from the time slot table in accordance with the current time slot counter value. The time slot counter is clocked by the PP clock in the normal mode or in the semi-automatic or manual modes when a PP burst (command C_{16}), PP cycle (command D_{16}), VP burst (command E_{16}), or VP continuous (command F_{16}) maintenance command has been issued.

A second level of VP code selection is provided on the MLCTL card. When the semi-automatic or manual mode of operation has been selected and the maintenance command entered in the command field is 0 through 7, the VP code select logic on MLCTL overrides the VP code developed on ML2 and substitutes the register field VP code. This override feature is necessary because the VP code associated with commands 2 through 7 (command 0 and 1 do not require a VP code) is determined by the register field and not the V

field. The register field VP code is also delayed one clock through a set of flip-flops and distributed to the maintenance logic control circuitry for use with commands 2 through 7. In all other cases of mode and maintenance command, the VP code developed on ML2 is selected by MLCTL. The output of the VP code select logic on MLCTL must now be clocked through three levels of flip-flops before use as the current VP code. As shown in figure 4-125, during the clock period immediately preceding the command execution period, the next VP code is used to select the Instruction Register (IR) for insertion in the Main Instruction Register (MIR) and to perform any effective address augmenting that is necessary. During the command execution period, the current VP code is distributed throughout the PP to control and monitor instruction execution.

4-210 Normal/Test Mode Timing Logic. The VP active bit logic on ML2 accepts mode and time slot data to determine if a VP will be active during the third succeeding clock (assuming the active indication is not overridden by logic to be described later). When the manual mode has been selected but a PP burst operation has not been chosen or the normal mode has been selected and the current time slot under consideration is time slot zero, the PXACTTR (active bit) signal is set to a logical one. When the manual mode has been selected in conjunction with a PP burst operation or the normal mode has been selected and the current time slot is other than time slot zero, the active indicator is retrieved from the time slot table in accordance with the current time slot counter value. The resulting PXACTTR signal is input to the active override logic on ML2.

The active override logic on ML2 implements the equation shown in figure 4-125. Basically, the logic implementation controls the disabling of the active indicator for a VP under the direction of the time slot override byte of the CR file. If the VP under consideration is the selected VP, however, the AUTO INTERRUPT OFF switch on the ASC Maintenance Console must be activated before the active indicator can be overridden. Independent of any consideration for the selected VP, the active indicator is overridden on three out of every four clock periods when a VP burst or VP continuous maintenance command is issued (these two commands are legal only in the manual mode). This is accomplished by enabling the active indicator only on time slots 0, 4, 8, and 12. Finally, the active indicator is overridden in the normal mode or the PP burst option of the manual mode when a VP is assigned to two consecutive time slots and is active in both (in this case, the active indicator is overridden on the second of the two time slots, because updating of an IR requires the full command execution period). The active indicator, modified by the override logic, is delayed one clock period by a single flip-flop and then input to the test mode/normal logic on MLCTL. The test mode/normal logic develops the ¬PXLNTM signal to indicate when VPJ will execute in the normal case (versus the test mode case). The first term in the equation shown in figure 4-125, $\overline{\text{MCOPN}}$, is used to switch operation to the test mode case when maintenance commands 1 through 7 and

A_{16} and B_{16} are ready to execute for the cases listed in table 4-9. The second term in the equation (ACTJ) represents the active indicator previously discussed in detail. The third and last term in the equation ($\overline{VPJUT}$ + MCRUN) is used to switch operation to the test mode case when VP_J is going to execute commands 1 through B_{16} or hold operation to the normal mode when a burst or run command is scheduled (commands C_{16} through F_{16}). The output of the test mode/normal logic on MLCTL is delayed one clock and distributed to PP control as the test mode indicator and normal MIR gate. When normal operation is to occur, the normal MIR gate enables the transfer of the selected IR control data into the various MIR areas (AUMIR, CRMIR, IRMIR, and VPRMIR) with the occurrence of the clock signaling the command execution period. When test mode operation is to occur, the test mode indicator combines with the maintenance logic controls necessary to execute the current maintenance command.

The net result of the normal/test mode timing logic is selection of either PP control or maintenance logic control for all command execution periods. In the normal mode of operation, the active VP's execute under the influence of the time slot table and any maintenance commands execute either in the time slot associated with the VP under test or in the first available time slot that is not associated with an active VP. In the manual and semi-automatic modes of operation, initiation of the maintenance commands is controlled by the ASC Maintenance Console.

4-211 MAINTENANCE COMMAND TRANSFER TABLES. The maintenance command repertoire consists of 15 basic command codes and a no-op. This paragraph presents the general concepts necessary for a full understanding of the maintenance command transfer tables and the remaining paragraphs in this section provide a step-by-step analysis for each of the 15 command codes. The maintenance command transfer tables are an abbreviated form of the transfer tables associated with the PP instruction repertoire and discussed in the PP control detailed theory, but they present the same type of information. Each of the maintenance commands begins execution with the maintenance controller in state zero, performs the desired operation in a series of nonzero states, and returns the maintenance controller to state zero at completion. The maintenance controller remains in the ready state (state zero) until a new command is entered in the maintenance registers. The abbreviated terms used in the maintenance command transfer tables and their actual logic card signatures and associated descriptions are listed in table 4-11. The no-op maintenance command (command zero) is not discussed in the following paragraphs because it is the result of zeroing the command code field and represents the lack of a valid maintenance command. In cases where a time slot is selected for test mode operation and the maintenance command is no-op, the PP and maintenance logic remain idle.

Table 4-11. Maintenance Command Transfer Table Terms

Abbreviated Term	Signature	Description
AMO	PXAMO	Indicates selection of the automatic mode.
BAJ	PXBAJ	Developed on ML2 to indicate when the SWB is available to VP_J.
CBCLOAD	PXCBCLOAD	Developed on MLCTL to enable the loading of the burst counter for commands C_{16} and E_{16}.
CBCRN	PXCBCRN	Developed on ML2 to indicate when the clock burst counter is counting (burst in progress).
ILL	PXILL	Developed on MLCTL to indicate when an illegal command exists in the maintenance registers.
LPCF	PXQLPCF	A flip-flop on MLCTL set to indicate locking of a PC(s) and reset to indicate unlocking of a PC(s).
MCOPN	PXMCOPN	Developed on MLCTL to indicate when VP_J is ready to execute command codes 1 through 7 and A_{16} and B_{16}.
MCRFLG	PXMCRFLG	Developed on MLCTL to indicate when command code F_{16} is running in the continuous mode.
MCRUN	PXMCRUN	Developed on MLCTL to indicate when a maintenance controlled run or burst is executing (commands C_{16}, D_{16}, E_{16}, and F_{16}).
QCPF(0-7)	PXQCPF(0-7)	A group of flip-flops on MLCTL initially zeroed when command D_{16} is issued and then set as the active VP's complete their current instruction.
QLDR	PXQLDR	A flip-flop on MLCTL set to develop the gate necessary for loading display register data in the CR file.
QSRA2	PXQSRA2	A flip-flop on MLCTL set to enable switch register data to the AU2B bus and reset to enable display register data to the AU2B bus.

Table 4-11. Maintenance Command Transfer Table Terms (Continued)

Abbreviated Term	Signature	Description
QSRDR	PXQSRDR	A flip-flop on MLCTL set to enable switch register data to the display register and reset to enable MDB data to the display register.
QVPRES	PXQVPRES	A flip-flop on MLCTL set to enable the resetting of the data holding registers associated with the VP(s) designated by the V field.
QVPSET	PXQVPSET	A flip-flop on MLCTL set to enable the resetting of the data holding registers associated with the VP(s) designated by the V field.
RC	PXQMRC	A flip-flop on MLCTL set to initiate a CM read request.
TRMCMA	PXTRMCMA	Developed on MLCTL to enable AU2B data to the desired SWBA.
TRMDMC	PXTRMDMC	Developed on MLCTL to enable the desired SWBD to the MDB.
TRROMNR	PXROMNR	Developed on MLCTL to enable ROM data into the desired NIR.
TSCST(0)	PXTSCST(0)	Developed on ML2 to indicate when time slot zero is under examination.
VJEQVR	PXVJEQVR	Developed on MLCTL to indicate when VP_J is identical to the VP specified by the register field.
VPADV	PXVPADV	Developed on MLCTL to indicate the VP's under test have not completed their current instruction.
VPJUT	PXVPJUT	Developed on MLCTL to indicate when VP_J is under test.
VPRN	PXVPRN	Developed on MLCTL to indicate VP_J is still executing the instruction that was executing when maintenance command code D_{16} was initiated.
WC	PXQMWC	A flip-flop on MLCTL set to initiate a CM write request.

4-212 Switch Register to Display Register Command. This maintenance command transfers the contents of the switch register to the display register. Execution of command code 1 is shown in the transfer table of table 4-12.

Step 1 — When command code 1 is entered in the maintenance registers, the CR file busy bit is set via the ¬PFDD24CR(12) signal from MLCTL and the mode is examined. If the manual or semi-automatic mode has been selected, the MCOPN flag is set and the maintenance controller is advanced to state 6. If the automatic mode has been selected but VP_J (VP_J will be the current VP on the second succeeding clock) is not the VP specified by the register field ($\overline{VJEQVR}$), the maintenance controller is advanced to state 4. If VP_J is identical to the VP specified by the register field, the MCOPN flag is set to save the second succeeding time slot for the actual data transfer described in step 3 and the maintenance controller is advanced to state 6.

Step 2 (State 4) — Command code 1 remains in state 4 when the automatic mode has been selected until VP_J is identical to the VP specified by the register field (VJEQVR). When the equality occurs, the MCOPN flag is set to save the second succeeding time slot for the data transfer and the maintenance controller is advanced to state 6.

Step 3 (State 6) — On the first clock after reaching state 6 in the manual or semi-automatic modes, or the second clock after reaching state 6 in the automatic mode, the following maintenance control signals are activated: The QSRDR flip-flop is set to enable the switch register to display register transfer via the PXSRDREN signal from MLCTL and the QLDR flip-flop is set to develop the CR file gates for the display register (¬PFSB0-3CR(15) from MLCTL). In addition, the busy bit and command code field are zeroed via the ¬PFDD24CR(12) and ¬PFDD28-31CR(12) signals, respectively, from MLCTL.

4-213 PP Register to Display Register Command. This maintenance command transfers the contents of the PP register (PC, NIR, IR, SWBA, SWBD, VPR, CR, SWBC, or MIR) specified by the register field to the display register. This command is illegal if the register field specifies the MIR in the automatic mode. Execution of command code 2 is shown in the transfer table of table 4-13.

Step 1 — When command code 2 is entered in the maintenance registers, the CR file busy bit is set via the ¬PFDD24CR(12)

Table 4-12. Switch Register to Display Register Maintenance Command Transfer Table

Description: (SR) → DR			Mnemonic: XSD			Code: 1
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 0 0	1 → BUSY			$\overline{\text{VJEQVR}}$·AMO
		1 1 0	1 → BUSY 1 → MCOPN			VJEQVR+$\overline{\text{AMO}}$
2	1 0 0	1 0 0				$\overline{\text{VJEQVR}}$·AMO
		1 1 0	1 → MCOPN			VJEQVR·AMO
3	1 1 0	0 0 0	1 → QSRDR 1 → QLDR 0 → BUSY 0 → COMMAND		SR/MDB → DR	

Table 4-13. PP Register to Display Register Maintenance Command Transfer Table

Description: (REG) → DR				Mnemonic: XRD		Code: 2
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 0 0	1 → BUSY			$\overline{VJEQVR} \cdot AMO \cdot \overline{ILL}$
		1 1 0	1 → BUSY 1 → MCOPN			$VJEQVR + \overline{AMO}$
		1 1 1	1 → BUSY 1 → ILLEGAL			ILL
2	1 0 0	1 0 0				$\overline{VJEQVR} \cdot AMO$
		1 1 0	1 → MCOPN			$VJEQVR \cdot AMO$
3	1 1 0	0 0 0	0 → QSRDR 1 → QLDR 1 → TR__MC 0 → BUSY 0 → COMMAND		SR/MDB → DR MDB	
4	1 1 1	1 1 1				ILL
		0 0 0	0 → ILLEGAL 0 → BUSY 0 → COMMAND			$\overline{ILL}$

signal from MLCTL and the mode and illegal flag are examined. If the command is illegal, the CR file illegal bit is set via the ¬PFDD26CR(12) signal from MLCTL and the maintenance controller is advanced to state 7. If the command is not illegal and the manual or semi-automatic mode has been selected, the MCOPN flag is set and the maintenance controller is advanced to state 6. If the automatic mode has been selected but VP_J is not the VP specified by the register field, the maintenance controller is advanced to state 4. If VP_J is identical to the VP specified by the register field, the MCOPN flag is set to save the second succeeding time slot for the actual data transfer described in step 3 and the maintenance controller is advanced to state 6.

Step 2 (State 4) Command code 2 remains in state 4 when the automatic mode has been selected until VP_J is identical to the VP specified by the register field. When the equality occurs, the MCOPN flag is set to save the second succeeding time slot for the data transfer and the maintenance controller is advanced to state 6.

Step 3 (State 6) On the first clock after reaching state 6 in the automatic or semi-automatic modes, or the second clock after reaching state 6 in the automatic mode, the PP register to display register transfer is enabled in the following manner: The appropriate PP register to MDB enable (PXTRPCMC for the PC, PXTRNRMC for the NIR, etc.) from MLCTL transfers the register data to the MDB; the QSRDR flip-flop is reset to enable the MDB data to the display register; the QLDR flip-flop is set to gate the data into the display register. In addition, the busy bit and command code field are zeroed via the ¬PFDD24CR(12) and ¬PFDD28-31CR(12) signals, respectively, from MLCTL.

Step 4 (State 7) The illegal form of command code 2 remains in state 7 until the illegal flag is cleared by zeroing the command. When this is done, the illegal flag is cleared via the ¬PFDD26CR(12) signal from MLCTL and the busy bit and command code field are cleared as described in the previous step.

4-214 Central Memory to Display Register Transfer. This maintenance command transfers the CM word addressed by the switch register to the display register. Execution of command code 3 is shown in the transfer table of table 4-14.

Table 4-14. Central Memory to Display Register Maintenance Command Transfer Table

Description: $((SR))_{CM} \rightarrow DR$			Mnemonic: XCD			Code: 3
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 0 0	1 → BUSY			AMO• ($\overline{VJEQVR}$+$\overline{BAJ}$)
		1 1 0	1 → BUSY 1 → MCOPN			VJEQVR• BAJ+AMO
2	1 0 0	1 0 0				AMO• ($\overline{VJEQVR}$+$\overline{BAJ}$)
		1 1 0	1 → MCOPN			VJEQVR• BAJ
3	1 1 0	1 0 1	1 → QSRA2 1 → TRMCMA	RC	SR/DR → AU2B AU2B	
4	1 0 1	1 0 1				AMO• ($\overline{VJEQVR}$+$\overline{BAJ}$)
		1 1 1	1 → MCOPN			VJEQVR• BAJ+$\overline{AMO}$
5	1 1 1	0 0 0	0 → QSRDR 1 → TRMDMC 1 → QLDR 0 → BUSY 0 → COMMAND		SR/MDB → DR MDB	

Step 1 — When command code 3 is entered in the maintenance registers, the CR file busy bit is set via the ⌐PFDD24CR(12) signal from MLCTL and the mode and buffer available signal are examined. If the manual or semi-automatic mode has been selected, the MCOPN flag is set and the maintenance controller is advanced to state 6. If the automatic mode has been selected, but VP_J is not the VP specified by the register field, or the SWBC is not available to VP_J ($\overline{BAJ}$), the maintenance controller is advanced to state 4. If VP_J is identical to the VP specified by the register field and the buffer is available to VP_J, the MCOPN flag is set to save the second succeeding time slot for the address transfer described in step 3 and the maintenance controller is advanced to state 6.

Step 2 (State 4) — Command code 3 remains in state 4 when the automatic mode has been selected until VP_J is identical to the VP specified by the register field and the SWBC is available to VP_J. When this is the case, the MCOPN flag is set to save the second succeeding time slot for the address transfer and the maintenance controller is advanced to state 6.

Step 3 (State 6) — On the first clock after reaching state 6 in the manual or semi-automatic mode, or the second clock after reaching state 6 in the automatic mode, the switch register to SWBA address transfer is accomplished in the following manner: The QSRA2 flip-flop is set to enable the switch register data to the AU2B bus; the PXTRMCMA enable from MLCTL is developed to transfer the AU2B address to the SWBA associated with the VP specified by the register field; and a read request is initiated by MLCTL to retrieve the addressed word. The maintenance controller is advanced to state 5.

Step 4 (State 5) — When the manual or semi-automatic mode has been selected, the maintenance controller is advanced from state 5 to state 7 on the first clock. In the automatic mode, the maintenance controller is advanced from state 5 to state 7 only after VP_J is identical to the VP specified by the register field and the SWBC is available to VP_J (read cycle complete). The MCOPN flag is set to save the second succeeding time slot for the data transfer described in step 5.

Step 5 (State 7) — On the first clock after reaching state 7 in the manual or semi-automatic mode, or the second clock after reaching state 7 in the automatic mode, the retrieved CM word is

transferred to the display register in the following manner: The PXTRMDMC enable from MLCTL is developed to transfer the retrieved CM word from the SWBD associated with the VP specified by the register field to the MDB; the QSRDR flip-flop is reset to enable the MDB data to the display register; the QLDR flip-flop is set to gate the transferred data into the display register. The busy bit and command code field are cleared via the ⌝PFDD24CR(12) and ⌝PFDD28-31CR(12) signals, respectively, from MLCTL.

4-215 ROM to Display Register Transfer. This maintenance command transfers the ROM word addressed by the switch register to the display register. Execution of command code 4 is shown in the transfer table of table 4-15.

Step 1 — When command code 4 is entered in the maintenance registers, the CR file busy bit is set via the ⌝PFDD24CR(12) signal from MLCTL and the mode is examined. If the manual or semi-automatic mode has been selected, the MCOPN flag is set and the maintenance controller is advanced to state 6. If the automatic mode has been selected but VP_J is not the VP specified by the register field, the maintenance controller is advanced to state 4. If VP_J is identical to the VP specified by the register field, the MCOPN flag is set to save the second succeeding time slot for the address transfer described in step 3 and the maintenance controller is advanced to state 6.

Step 2 (State 4) — Command code 4 remains in state 4 when the automatic mode has been selected until VP_J is identical to the VP specified by the register field. When the equality occurs, the MCOPN flag is set to save the second succeeding time slot for the address transfer and the maintenance controller is advanced to state 6.

Step 3 (State 6) — On the first clock after reaching state 6 in the manual or semi-automatic mode, or the second clock after reaching state 6 in the automatic mode, the PXROMNR enable is developed on MLCTL to direct a ROM read in the following manner: The switch register ROM address is applied to ROM over RMAB via the PPMCRBE enable from PCCTL; the data from ROM is transferred to the NIR associated with the VP specified by the register field over CMDB via the PNRMCDE and PNCDNRE enables from PCCTL. The maintenance controller is advanced to state 5.

Table 4-15. ROM to Display Register Maintenance Command Transfer Table

Description: $((SR))_{ROM} \longrightarrow DR$			Mnemonic: XMD			Code: 4
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 0 0	1 → BUSY			$\overline{VJEQVR} \cdot AMO$
		1 1 0	1 → BUSY 1 → MCOPN			$VJEQVR + \overline{AMO}$
2	1 0 0	1 0 0				$\overline{VJEQVR} \cdot AMO$
		1 1 0	1 → MCOPN			$VJEQVR \cdot AMO$
3	1 1 0	1 0 1	1 → TRROMNR		RMAB CMDB	
4	1 0 1	1 0 1				$\overline{VJEQVR} \cdot AMO$
		1 1 1	1 → MCOPN			$VJEQVR + \overline{AMO}$
5	1 1 1	0 0 0	1 → TRNRMC 0 → QSRDR 1 → QLDR 0 → BUSY 0 → COMMAND		MDB SR/MDB → DR	

Step 4 (State 5) When the manual or semi-automatic mode has been selected, the maintenance controller is advanced from state 5 to state 7 on the first clock. In the automatic mode, the maintenance controller is advanced from state 5 to state 7 only after VP_J is identical to the VP specified by the register field. When this is the case, the MCOPN flag is set to save the second succeeding time slot for the data transfer described in step 5.

Step 5 (State 7) On the first clock after reaching state 7 in the manual or semi-automatic mode, or the second clock after reaching state 7 in the automatic mode, the word retrieved from ROM is transferred to the display register in the following manner: The PXTRNRMC enable from MLCTL is developed to transfer the ROM word from the NIR associated with the VP specified by the register field to the MDB; the QSRDR flip-flop is reset to enable the MDB data to the display register; the QLDR flip-flop is set to gate the transferred data into the display register. The busy bit and command code field are cleared via the ⌝PFDD24CR(12) and ⌝PFDD28-31CR(12) signals, respectively, from MLCTL.

4-216 Switch Register to PP Register Command. This command transfers the contents of the switch register to the PP register (PC, NIR, IR, SWBA, SWBD, VPR, CR, SWBC, or MIR) specified by the register field. This command is illegal if the register field specifies a SWBC or MIR in the automatic mode. Execution of command 5 is shown in the transfer table of table 4-16.

Step 1 Identical to step 1 of command code 2.

Step 2 Identical to step 2 of command code 2.

Step 3 (State 6) On the first clock after reaching state 6 in the manual or semi-automatic mode, or the second clock after reaching state 6 in the automatic mode, the switch register is transferred to the desired PP register in the following manner: The QSRA2 flip-flop is set to enable the switch register to the AU2B bus; the appropriate AU2B to PP register enable (PXTRMCPC for PC's, PXTRMCNR for NIR's, etc.) from MLCTL combines with the register field VP code to transfer AU2B data to the desired PP register. The busy bit and command code field are cleared via the ⌝PFDD24CR(12) and ⌝PFDD28-31CR(12) signals, respectively, from MLCTL.

Step 4 Identical to step 4 of command code 2.

Table 4-16. Switch Register to PP Register Maintenance Command Transfer Table

Description: (SR)→REG			Mnemonic: XSR			Code: 5
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 0 0	1→BUSY			$\overline{VJEQVR} \cdot AMO \cdot \overline{ILL}$
		1 1 0	1→BUSY 1→MCOPN			$VJEQVR + \overline{AMO}$
		1 1 1	1→BUSY 1→ILLEGAL			ILL
2	1 0 0	1 0 0				$\overline{VJEQVR} \cdot AMO$
		1 1 0	1→MCOPN			$VJEQVR \cdot AMO$
3	1 1 0	0 0 0	1→QSRA2 1→TRMC.___ 0→BUSY 0→COMMAND		SR/DR→AU2B AU2B	
4	1 1 1	1 1 1				ILL
		0 0 0	0→ILLEGAL 0→BUSY 0→COMMAND			$\overline{ILL}$

4-217 Display Register to PP Register Command. This command transfers the contents of the display register to the PP register (PC, NIR, IR, SWBA, SWBD, VPR, CR, SWBC, or MIR) specified by the register field. This command is illegal if the register field specifies a SWBC or MIR in the automatic mode. Execution of command code 6 is shown in the transfer table of table 4-17.

Step 1	Identical to step 1 of command code 2.
Step 2	Identical to step 2 of command code 2.
Step 3	Identical to step 3 of command code 5 except the QSRA2 flip-flop is reset to enable the display register to the AU2B bus.
Step 4	Identical to step 4 of command code 2.

4-218 Display Register to CM Command. This maintenance command transfers the contents of the display register to CM at the address specified by the switch register. Execution of command code 7 is shown in the transfer table of table 4-18.

Step 1	Identical to step 1 of command code 3.
Step 2	Identical to step 2 of command code 3.
Step 3	Identical to step 3 of command code 3 except no read request is initiated.
Step 4	Identical to step 4 of command code 3.
Step 5 (State 7)	On the first clock after reaching state 7 in the manual or semi-automatic mode, or the second clock after reaching state 7 in the automatic mode, the display register data is written to CM at the address established in step 3 in the following manner: The QSRA2 flip-flop is reset to enable display register data to the AU2B bus; the PXTRMCMD enable from MLCTL is developed to transfer AU2B data to the SWBD associated with the VP specified by the register field; a write request (PXQMWC) is initiated by MLCTL to store the SWBD (display register) at the address specified by the SWBA (switch register). The busy bit and command code field are cleared via ⌝PFDD24CR(12) and PFDD28-31CR(12) signals, respectively, from MLCTL.

4-218 Lock Program Counter (PC) Command. This maintenance command locks the PC's of the VP's designated by the V field (the PC's are held at their current value). Execution of command code 8 is shown in the transfer table of table 4-19.

Table 4-17. Display Register to PP Register Maintenance Command Transfer Table

Description: (DR)→REG			Mnemonic: XDR			Code: 6
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 0 0	1→BUSY			$\overline{VJEQVR}$·AMO·$\overline{ILL}$
		1 1 0	1→BUSY 1→MCOPN			VJEQVR+$\overline{AMO}$
		1 1 1	1→BUSY 1→ILLEGAL			ILL
2	1 0 0	1 0 0				$\overline{VJEQVR}$·AMO
		1 1 0	1→MCOPN			VJEQVR·AMO
3	1 1 0	0 0 0	0→QSRA2 1→TRMC___ 0→BUSY 0→COMMAND		SR/DR→AU2B AU2B	
4	1 1 1	1 1 1				ILL
		0 0 0	0→ILLEGAL 0→BUSY 0→COMMAND			$\overline{ILL}$

Table 4-18. Display Register to CM Maintenance Command Transfer Table

Description: (DR)⟶(SR)$_{CM}$			Mnemonic: XDC			Code: 7
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 0 0	1⟶BUSY			AMO• ($\overline{VJEQVR+BAJ}$)
		1 1 0	1⟶BUSY 1⟶MCOPN			VJEQVR• BAJ+$\overline{AMO}$
2	1 0 0	1 0 0				AMO• ($\overline{VJEQVR+BAJ}$)
		1 1 0	1⟶MCOPN			VJEQVR• BAJ
3	1 1 0	1 0 1	1⟶QSRA2 1⟶TRMCMA		SR/DR⟶AU2B AU2B	
4	1 0 1	1 0 1				AMO• ($\overline{VJEQVR+BAJ}$)
		1 1 1	1⟶MCOPN			VJEQVR• BAJ+$\overline{AMO}$
5	1 1 1	0 0 0	0⟶QSRA2 1⟶TRMCMD 0⟶BUSY 0⟶COMMAND	WC	SR/DR⟶AU2B AU2B	

Table 4-19. Lock Program Counter Maintenance Command Transfer Table

Description: Lock PC			Mnemonic: LPC			Code: 8
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 1 0	1→BUSY			
2	1 1 0	0 0 0	1→LPCF 0→BUSY 0→COMMAND			

Step 1 — When command code 8 is entered in the maintenance registers, the CR file busy bit is set via the ⌝PFDD24CR(12) signal from MLCTL and the maintenance controller is advanced to state 6.

Step 2 (State 6) — Following the first clock after reaching state 6, the LPCF flip-flop is set to initiate the locking of PC's designated by the V field. In the manual mode, the PC associated with the VP selected by the ASC Maintenance Console VP SELECT switch is locked. In the semi-automatic and automatic modes, the PC's associated with the VP's designated by the V field are locked under influence of the time slot table. The busy bit and command code field are cleared via the ⌝PFDD24CR(12) and ⌝PFDD28-31CR(12) signals, respectively, from MLCTL.

4-220 Unlock PC Command. This maintenance command unlocks the PC's of the VP's designated by the V field. Execution of command code 9 is shown in the transfer table of table 4-20.

Step 1 — Identical to step 1 of command code 8.

Step 2 — Identical to step 2 of command code 8 except the LPCF flip-flop is reset to unlock the PC's designated by the V field.

4-221 Reset Registers Command. This maintenance command resets the PC, NIR, SWBD, SWBA, VPR, IR, and CM base registers of the VP's designated by the V field. Execution of command code A_{16} is shown in the transfer table of table 4-21.

Step 1 — When command code A_{16} is entered in the maintenance registers, the CR file busy bit is set via the ⌝PFDD24CR(12) signal from MLCTL and the maintenance controller is advanced to state 6.

Step 2 (State 6) — On the first clock after reaching state 6, the QVPRES flip-flop on MLCTL is set. The resulting PXVPRES signal combines with the V field on ML2 to reset the PC's, NIR's, SWBD's, SWBA's, VPR's, IR's, and CM base registers associated with the VP's designated by the V field. The busy bit and command code field are cleared via the ⌝PFDD24CR(12) and ⌝PFDD28-31CR(12) signals, respectively, from MLCTL.

4-222 Set Registers Command. This maintenance command sets the PC, NIR, SWBD, SWBA, VPR, IR, and CM base registers of the VP's designated by the V field. Execution of command code B_{16} is shown in the transfer table of table 4-22.

Table 4-20. Unlock Program Counter Maintenance Command Transfer Table

Description: Unlock PC			Mnemonic: UPC			Code: 9
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 1 0	1→BUSY			
2	1 1 0	0 0 0	0→LPCF 0→BUSY 0→COMMAND			

Table 4-21. Reset PP Registers Maintenance Command Transfer Table

Description: Reset Registers			Mnemonic: RST			Code: A_{16}
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 1 0	1 → BUSY			
2	1 1 0	0 0 0	1 → QVPRES 0 → BUSY 0 → COMMAND			

Table 4-22. Set PP Registers Maintenance Command Transfer Table

Description: Set Registers			Mnemonic: SET			Code: B_{16}
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 1 0	1 → BUSY			
2	1 1 0	0 0 0	1 → QVPSET 0 → BUSY 0 → COMMAND			

Step 1 Identical to step 1 of command code A_{16}.

Step 2 Identical to step 3 of command code B_{16} except the QVPSET flip-flop is set and the resulting PXVPSET signal is developed to set the mentioned registers.

4-223 PP Burst Command. In the automatic mode, this maintenance command advances the VP's designated by the V field, along with those that are active, under the influence of the time slot table. Counting begins at time slot zero and continues until the number of examined time slots equals the burst count. In the manual or semi-automatic mode, the PP is advanced the number of time slots indicated by the burst field. Execution of command code C_{16} is shown in the transfer table on table 4-23.

Step 1 When command code C_{16} is entered in the maintenance registers, the CR file busy bit is set via the ⌝PFDD24CR(12) signal from MLCTL and the mode is examined. If the manual or semi-automatic mode has been selected, the PXCBCLOAD control on MLCTL is developed to enable the loading of the burst counter with the burst field. The maintenance controller is advanced to state 6. If the automatic mode has been selected but the current time slot is not time slot zero ($\overline{\text{TSCST(0)}}$), the maintenance controller is advanced to state 4. If time slot zero is current in the automatic mode, the PXCBCLOAD control is developed to enable loading of the burst counter and the maintenance controller is advanced to state 6.

Step 2 (State 4) Command code C_{16} remains in state 4 when the automatic mode has been selected until time slot zero is current. When time slot zero is current, the PXCBCLOAD control is developed to enable loading of the burst counter and the maintenance controller is advanced to state 6.

Step 3 (State 6) As soon as the burst counter is loaded, the PXCBCRN signal is developed to indicate that the burst counter is counting on each clock. When PXCBCRN goes to logical one, the PXMCRUN control is developed on MLCTL to advance all active VP's, including those under test, under the influence of the time slot table. Advancement continues with each clock as long as the burst counter remains in a nonzero state. When the burst is complete, as indicated by a cleared PXCBCRN control on ML2, the busy bit and command code field are cleared via the ⌝PFDD24CR(12) and ⌝PFDD28-31CR(12) signals, respectively, from MLCTL.

Table 4-23. PP Burst Maintenance Command Transfer Table

Description: PP Burst			Mnemonic: CLK			Code: C_{16}
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 0 0	1→BUSY			$\overline{TSCST(0)}$·AMO
		1 1 0	1→BUSY 1→CBCLOAD			TSCST(0)+$\overline{AMO}$
2	1 0 0	1 0 0				$\overline{TSCST(0)}$·AMO
		1 1 0	1→CBCLOAD			TSCST(0)·AMO
3	1 1 0	1 1 0	1→MCRUN			CBCRN
		0 0 0	0→BUSY 0→COMMAND			$\overline{CBCRN}$

4-224 PP Cycle Command. In the automatic mode, this maintenance command advances the VP's designated by the V field, along with those that are active, under the influence of the time slot table until all V field designated VP's have completed their current instruction. In the semi-automatic mode, the VP's are advanced as in the automatic mode until time slot zero is reached. In the manual mode, the one VP designated by the V field is advanced until its current instruction is complete. Execution of command code D_{16} is shown in the transfer table of table 4-24.

Step 1 — When command code D_{16} is entered in the maintenance registers, the CR file busy bit is set via the ⌝PFDD24CR(12) signal from MLCTL and the PXQCPF(0-7) flags are cleared. The maintenance controller is advanced to state 6.

Step 2 (State 6) — The cleared PXQCPF(0-7) flags combine with the VP's under test and VP_J to develop PXVPADV and PXVPRN, respectively (both of these controls are described in table 4-10). As long as both of these controls are logical one, the PXMCRUN control is developed to advance the VP's under test (and all other active VP's in the automatic and semi-automatic modes). In the automatic and semi-automatic modes, as the VP's under test complete their instructions, the PXVPRN signal clears the PXMCRUN control so no additional advancement of these VP's is made. When the last VP under test completes its instruction and time slot zero occurs, the PXVPADV signal clears the PXMCRUN control for the remainder of the maintenance command. In the automatic case, the PP continues running in wait for another maintenance command. In the semi-automatic case, the PP stops. In the manual mode only one VP is selected for advancement, so when that VP completes its current instruction the PP is stopped. In all three cases of mode, the busy bit and command code field are cleared when the PXVPADV signal goes to zero.

4-225 VP Burst Command. In the manual mode (this command is illegal in the automatic and semi-automatic modes), the single VP specified by the V field is advanced the number of steps indicated by the burst field. Execution of command code E_{16} is shown in the transfer table of table 4-25.

Step 1 — When command code E_{16} is entered in the maintenance registers, the CR file busy bit is set via the ⌝PFDD24CR(12) signal from MLCTL and the PXCBCLOAD control is developed to load the burst counter with twice the burst field. The maintenance controller is advanced to state 6. If command code E_{16} is entered in the maintenance registers in

Table 4-24. PP Cycle Maintenance Command Transfer Table

Description: PP Cycle			Mnemonic: CYC			Code: D_{16}
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 1 0	1⟶BUSY 0⟶QCPF(0-7)			
2	1 1 0	1 1 0	1⟶MCRUN			VPADV•VPRN
		0 0 0	0⟶BUSY 0⟶COMMAND			$\overline{\text{VPADV}}$

Table 4-25. VP Burst Maintenance Command Transfer Table

Description: VP Burst			Mnemonic: VPBST			Code: E_{16}
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 1 0	1⟶BUSY 1⟶CBCLOAD			$\overline{\text{ILL}}$
		1 1 1	1⟶BUSY 1⟶ILLEGAL			ILL
2	1 1 0	1 1 0	1⟶MCRUN			CBCRN
		0 0 0	0⟶BUSY 0⟶COMMAND			$\overline{\text{CBCRN}}$
3	1 1 1	1 1 1				ILL
		0 0 0	0⟶ILLEGAL 0⟶BUSY 0⟶COMMAND			$\overline{\text{ILL}}$

an illegal mode, the CR file illegal bit is set via the ⌉PFDD26CR(12) signal from MLCTL and the maintenance controller is advanced to state 7.

Step 2 (State 6) — As soon as the burst counter is loaded, the PXCBRN signal is developed to indicate the burst counter is counting on each clock. When PXCBCRN goes to a logical one, the PXMCRUN control is developed on MLCTL to advance the VP SELECT switch VP. The active override logic on ML2 disables advancement of the test VP except on every fourth clock, so a burst field of eight (doubled to 16 before being loaded in the burst counter) will advance the test VP four steps. When the burst is complete, as indicated by a cleared PXCBCRN signal on ML2, the busy bit and command code field are cleared via the ⌉PFDD24CR(12) and ⌉PFDD28-31CR(12) signals, respectively, from MLCTL.

Step 3 — Identical to step 4 of command code 2.

4-226 VP Continuous Command. In the manual mode (this command is illegal in the automatic and semi-automatic modes), the VP selected by the VP SELECT switch on the ASC Maintenance Console and specified by the V field is advanced until a new maintenance command code is received. If the new command is other than F_{16}, then, in addition to halting the specified VP, the new command is performed. Execution of command code F_{16} is shown in the transfer table of table 4-26.

Step 1 — When command code F_{16} is entered in the maintenance registers, the CR file busy bit is set via the ⌉PFDD24CR(12) signal from MLCTL. If command code F_{16} is entered in the maintenance registers in an illegal mode, the CR file illegal bit is set via the ⌉PFDD26CR(12) signal from MLCTL and the maintenance controller is advanced to state 7; otherwise, the maintenance controller is advanced to state 6.

Step 2 (State 6) — When state 6 is reached and the PXMCRFLG signal indicates the previous maintenance command was not F_{16} ($\overline{\text{MCRFLG}}$), the PXMCRFLG signal goes to a logical one, the maintenance controller is returned to state zero, and the busy bit and command code field are cleared. This new set of conditions is used to develop the PXMCRUN control, which in turn, advances the VP under test until a new maintenance command is entered in the maintenance registers. When a new command code F_{16} is received, step 1 is executed as previously mentioned and step 2 clears PXMCRFLG to stop advancement of the VP under test.

Table 4-26. VP Continuous Maintenance Command Transfer Table

Description: VP Continuous			Mnemonic: VPCNT			Code: F_{16}
Step	Present State	Next State	Transfers	SWBC	Facility	Transfer Conditions
1	0 0 0	1 1 0	1⟶BUSY			$\overline{ILL}$
		1 1 1	1⟶BUSY 1⟶ILLEGAL			ILL
2	1 1 0	0 0 0	1⟶MCRFLG			$\overline{MCRFLG}$
			0⟶MCRFLG			MCRFLG
			0⟶BUSY 0⟶COMMAND			
3	0 0 0	X X X	0⟶MCRFLG			MCRFLG($\overline{0+F}$)
4	1 1 1	1 1 1				ILL
		0 0 0	0⟶ILLEGAL 0⟶BUSY 0⟶COMMAND			$\overline{ILL}$

Step 3 When a maintenance command other than F_{16} is entered in the maintenance registers after the first F_{16}, the PXMCRFLG signal is cleared to stop advancement of the VP under test and the new command is executed as detailed in its associated transfer table.

Step 4 Identical to step 4 of command code 2.

INDEX

TEXAS INSTRUMENTS
INCORPORATED
EQUIPMENT GROUP
AUSTIN, TEXAS

PUBLICATION UPDATE

PUBLICATION

PROGRAM ASC PUBLICATION NO. 930182-2

TITLE OMI: Peripheral Processor

DATE Dec., 1973 JOB NO. 930182

TYPE OF CHANGE

☐ IMMEDIATE
(MAY CAUSE PERSONAL INJURY OR EQUIPMENT DAMAGE/FAILURE)

☐ ROUTINE
(BATCH PROCESSED)

SUBMITTED BY

NAME _____ PHONE _____

ADDRESS _____

MAIL STATION _____ DATE _____

LIST PAGE AND PARAGRAPH OR FIGURE NUMBERS AND DESCRIBE RECOMMENDED CHANGES.

FORWARD CHANGES BY FOLDING THIS SHEET AND STAPLING. RETURN ADDRESS IS ON BACK OF SHEET.

TEXAS INSTRUMENTS INCORPORATED
EQUIPMENT GROUP
P.O. BOX 2909
AUSTIN, TEXAS 78767

ATTENTION: TECHNICAL DATA BRANCH
MAIL STATION 2146