OPERATION AND MAINTENANCE INSTRUCTIONS:

ASC-4X CENTRAL PROCESSOR (CP-4X) VOLUME 1

TEXAS INSTRUMENTS
INCORPORATED

ASC

OPERATION AND MAINTENANCE INSTRUCTIONS:

ASC-4X CENTRAL PROCESSOR (CP-4X)
VOLUME 1

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

TEXAS INSTRUMENTS
INCORPORATED

931443-2
January 1976

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INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES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 bar in the outer margins of the page.

Operation and Maintenance Instructions: ASC-4X Central Processor (CP-4X), Vol. 1, P/N 931443-2

Original Issue . . . . May 1974
Revised and Reissued . . . . January 1976

Total number of pages in this publication is 298 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Title	0				
Effective Pages	0				
iii - x	0				
1-1 - 1-32	0				
2-1/2-2	0				
3-1/3-2	0				
4-1 - 4-252	0				

TABLE OF CONTENTS

Paragraph | Title | Page

SECTION I. GENERAL DESCRIPTION

SECTION II. INSTALLATION

SECTION III. OPERATING INSTRUCTIONS

SECTION IV. PRINCIPLES OF OPERATION

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Paragraph **Title** **Page**

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

SECTION I

GENERAL DESCRIPTION

1.1 GENERAL

This section describes the operation of the 4-Pipe Central Processor (CP) of the Texas Instruments Advanced Scientific Computer (ASC-4X). It includes a brief system overview of the ASC, a general functional block diagram description of the CP, a physical description of the CP, plus information about the instruction set and words used in the CP. Section 4 of this manual provides a detailed discussion of the CP theory of operation. Other useful charts and data are contained in the appendices of this manual. This manual applies to 4-pipe CP configurations only.

1.2 PURPOSE

The ASC CP accesses program instructions from Central Memory, executes those instructions, and stores the results either within the CP or back into Central Memory. In performing this function, it also monitors program status to detect errors, branches and conflicts, and informs the Peripheral Processor if it is unable to continue a particular operation. The Peripheral Processor controls the selection of programs executed by the CP.

1.3 ASC SYSTEM OVERVIEW

Besides the CP, the ASC includes the following major units:

- Peripheral Processor (PP)
- Central Memory System (CM)
- System Clock
- Disc Storage System
- Magnetic Tape System
- Data Communications Channel
- Paper Peripheral Channels
- Operators Console
- Display Console
- Power System
- Maintenance System.

The relationship of these components is shown in figure 1-1. The CP interfaces directly with Central Memory for instruction and operand fetching, as well as for maintenance purposes. Initial programming sequences are determined by the PP, which also controls CP reaction to certain status conditions and calls. The CP, however, executes programs under its own control.

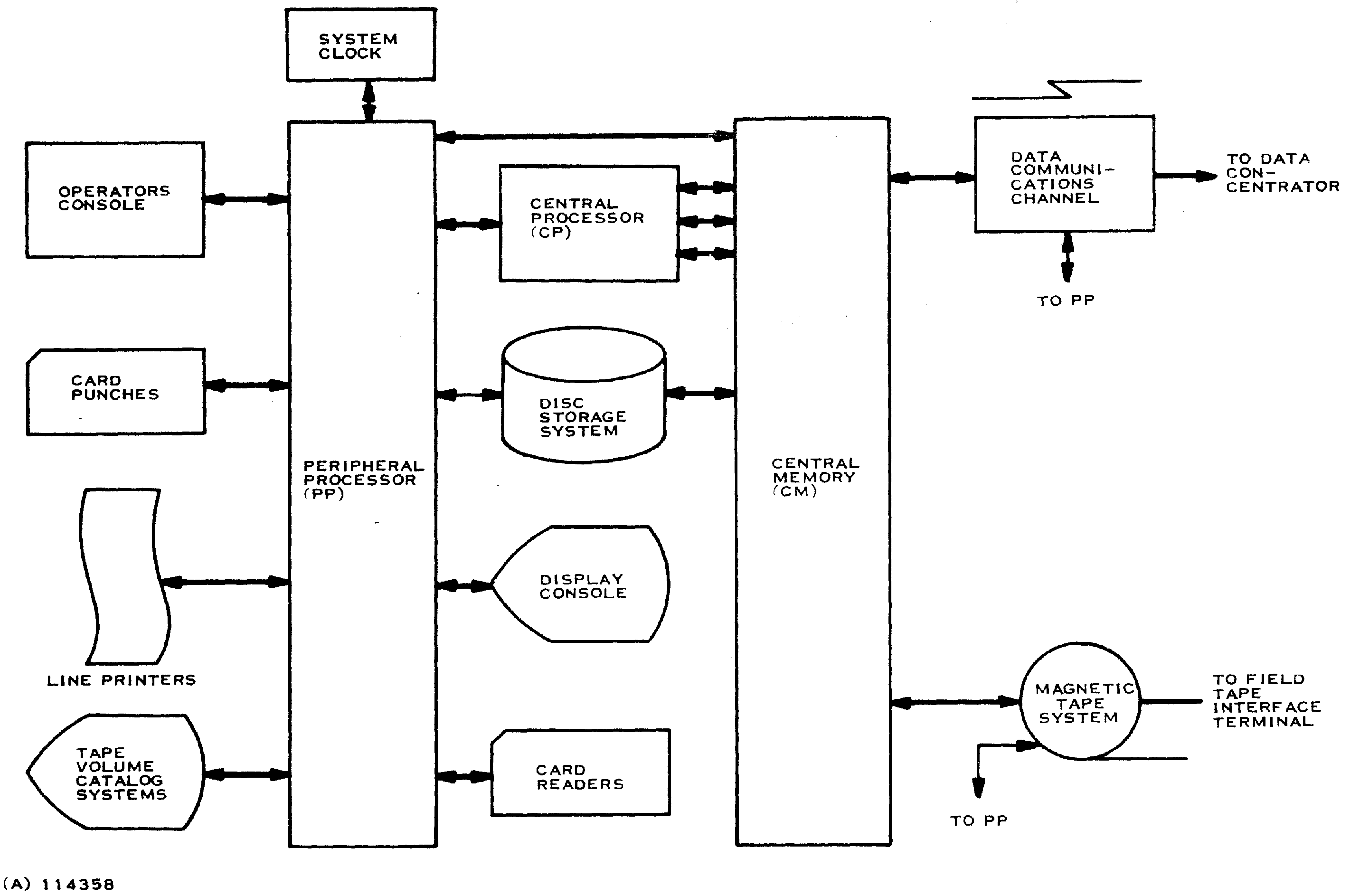


Figure 1-1. ASC Simplified Block Diagram

1.4 FUNCTIONAL DESCRIPTION

The ASC-4X Central Processor (4XCP) is comprised of nine units - one Instruction Processing Unit (IPU4) to process the CP commands, four Memory Buffer Units (MBU's) to provide central memory operands, and four Arithmetic Units (AU's) to perform the specified arithmetic operations. The structuring of these units is shown in figure 1-2. Figure 1-3 is a block diagram of the 4XCP.

The 4XCP provides four parallel execution pipelines below the IPU. Any mixture of scalar or vector instructions may be in execution simultaneously in the four pipes. In any of the CP configurations, the interaction between an IPU, MBU, and AU is equivalent to that of one pipeline. The flow of data is from the IPU to the MBU, from the MBU to the AU, and then from the AU back to the MBU for stores to memory or back to the IPU for arithmetic results to the register file. The IPU performs all decisions pertaining to the routing of instructions to various pipes. MBU's and AU's are not aware of other MBU-AU pairs.

In the multiple-pipeline CP's, each MBU has its own dedicated memory port. The times-four CP, for example, uses five memory ports; one for the IPU4 and four for the MBU's. The AU details information is loaded from or stored into memory only during maintenance commands and context switching. AU memory requests, therefore, occur infrequently and are routed through an expander cascaded on another expander.

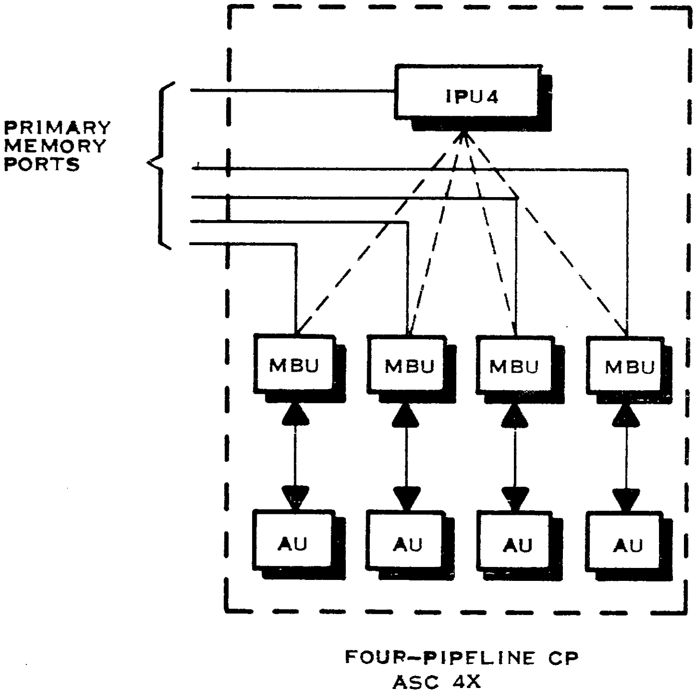


Figure 1-2. 4XCP Unit Structure

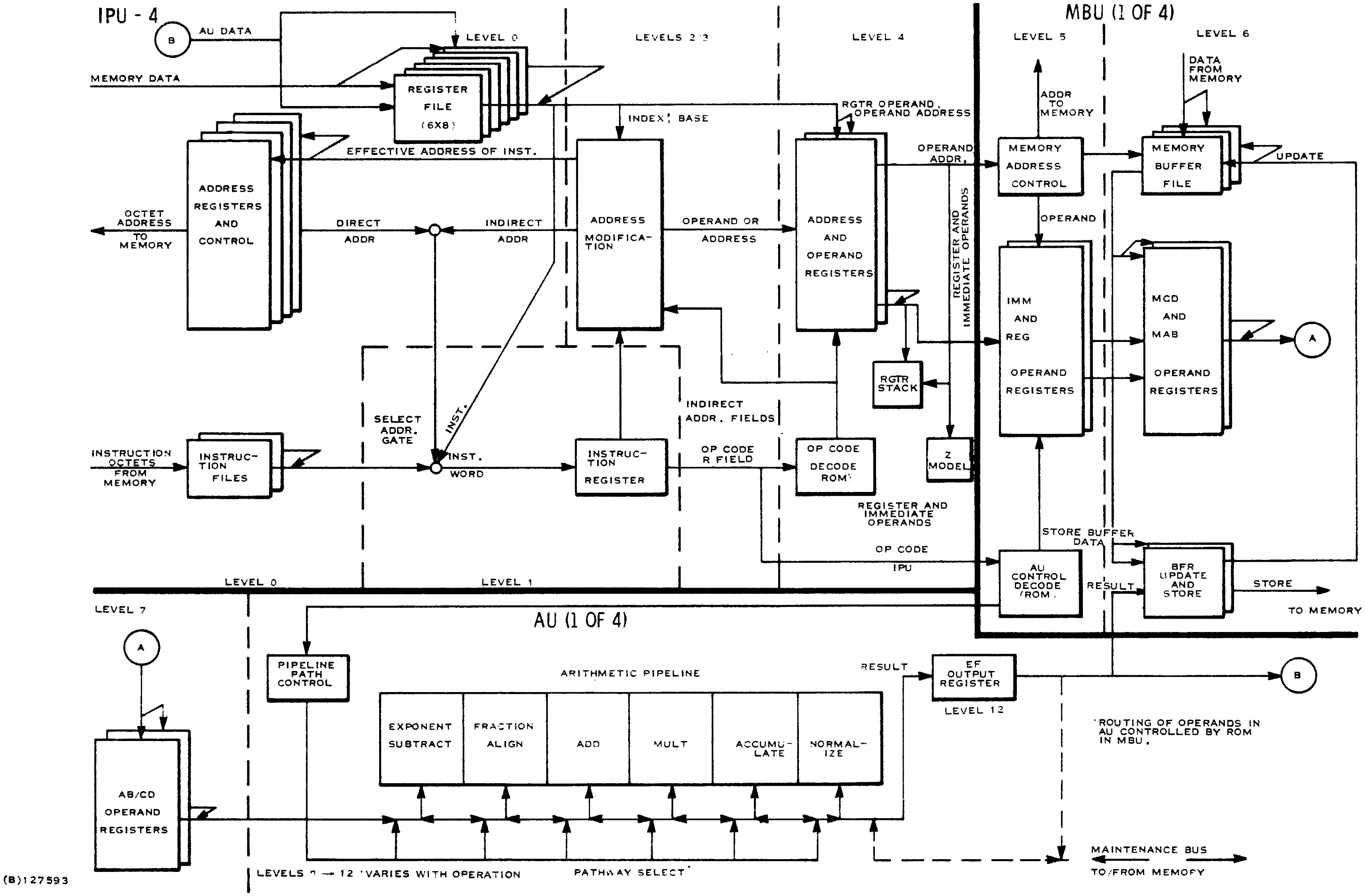


Figure 1-3. Central Processor Block Diagram

1.4.1 ADDRESS REGISTERS AND CONTROL. Four address registers control the acquisition of instruction word octets (8-word groups) from Central Memory. These registers select the proper instruction word for processing, call up a new octet while the current one is being processed, and provide for branch address acquisition. During indirect addressing, the output of the Address Modification network updates the Output Address Register in this circuit for each new address developed by the network until the terminal effective address is reached. The other registers maintain the program address so that the program resumes when the effective address is reached.

1.4.2 REGISTER FILE. The Register File is a memory source contained within the IPU. These registers are loaded by program instructions with data from either memory or the AU output. The file consists of six sets of eight 32-bit register (six octets). Each area in the file has a primary function, such as base addresses for developing effective addresses (15 words), general arithmetic use (16 words), seven index registers, and eight vector parameter registers to define the scope of a specific vector instruction. They may, however, be used for other processes.

1.4.3 INSTRUCTION FILES. Two instruction files, each containing one octet (eight words), supply a continuous source of instructions to the Instruction Register. The Address Registers and Control block controls loading and selection from these registers. It first loads one file and begins drawing instructions from the octet in that file. Address Control then loads the second file while the first one empties. Consecutive addresses supply a smooth transition from one file to the next. During indirect addressing, the effective address of an instruction from the Address Modification block selects the output from the instruction files if the address is currently in the files.

1.4.4 INSTRUCTION REGISTER. The Instruction Register receives the selected word from the instruction file and holds it for processing. Depending on the instruction format, the register may contain address bits, address modifiers, and operand and/or an operation code. The register output drives instruction decode and address generation networks.

1.4.5 ADDRESS MODIFICATION. When the Instruction Register specifies either direct or indexed addressing, the Address Modification block performs the operations required to generate a new address. This block provides for base address (from Register File) plus displacement modification and/or additional of the contents of one of the seven index registers in the Register File. The circuit permits direct or indirect addressing with or without modification, or the development of an immediate operand. Operands, direct operand addresses, and terminal operand addresses transfer to the MBU's to provide operands for the AU's. If an indirect address develops, it returns to the Output Address Register to retrieve a new instruction word for further address generation. The modification hardware includes input registers for indexing, base address and displacement, an adder, plus a result holding and output register.

1.4.6 ADDRESS AND OPERAND REGISTERS. These two registers are the IPU output register. They provide each MBU (according to pre-determined priorities) with either two operands, one operand and an operand address, or just one address.

1.4.7 IMM/REG REGISTERS. The Immediate (IMM) and Register (REG) Operand Registers receive operands from the IPU. During vector initialization, the IMM Register also transmits the vector parameters to the MBU Registers to set up the beginning vector conditions. Once a vector operation begins, neither of these registers is used until the next operation begins. Control signals, generated within the MBU, transfer data that is in these registers to the output registers of the MBU during scalar operations.

1.4.8 MEMORY ADDRESS CONTROL. This circuit supplies addresses to memory for storing results from the AU vector and store operations and for accessing new operand octets from memory for input to the AU. During scalar operations, operand addresses are supplied from the IPU. If the desired operand is already in the Memory Buffer File, the IPU sends only a 4-bit address to select the output from one of the file registers. If the operand is not in the buffer file, the IPU sends a full 21-bit address to fetch the octet containing the operand from Central Memory and loads it into the buffer file before transferring the operand to the output register. During vector operations, Memory Address Control generates the address of each octet in the vector after the address is initially loaded by the IPU.

1.4.9 MEMORY BUFFER FILE. The Memory Buffer File consists of six octet buffers plus an octet receiver/synchronizer register. The buffers are arranged in two three-stage buffers with the output of the final stage available to the output registers. Inputs to the buffers may enter the final file to bypass the delay in the buffering sequence. During scalar operations, Memory Address Control can select the output from either buffer and transfer it to the MCD Operand Register. During vector operations each buffer set supplies a stream of operands to one of the MAB/MCD Operand Registers. Either buffer set may be modified by the result output from the Arithmetic Pipeline (update) during scalars.

1.4.10 MAB/MCD OPERAND REGISTERS. These registers supply two operands simultaneously to the AU for processing. The MAB Register receives registers operands from the REG Register during scalar operations, and vector operands from the buffer file during vector operations. The MCD Register receives either immediate operands from the IMM Register or operands from either set of the buffer file during scalar operations. During vector operations the buffer file supplies a stream of operands to the MCD Register.

1.4.11 AU CONTROL DECODE. The AU Control Decode is a Read Only Memory (ROM) that designates to the AU which processes must be performed to accomplish the function specified by the Op Code. The decode circuit also supplies control signals to aid in selection of operands for the MAB/MCD registers.

1.4.12 BUFFER UPDATE AND STORE. The buffer update provides temporary retention of an octet of AU output. This octet may change the contents of the buffer file, or may be stored into Central Memory when the AU begins to produce results for a new octet.

1.4.13 AB/CD OPERAND REGISTERS. These registers are the input phase to the arithmetic pipeline. They receive two operands from the MBU and transfer them to the pipeline when the pipeline segment that performs the first operational step becomes available. Other inputs to these registers come from within the AU to provide a feedback path.

1.4.14 PIPELINE PATH CONTROL. This circuit follows the directions of the AU Control ROM in the MBU to perform the gating and sequencing functions required to develop a complete process in the pipeline.

1.4.15 ARITHMETIC PIPELINE. The Arithmetic Pipeline is a segmented arithmetic processor whose sequence is determined by the MBU ROM signals. Six segments of the pipe perform independent operations on up to six different sets of operands simultaneously. Each segment is a basic function that, combined in a specific order with other segments, performs arithmetic operations from scalar addition to complex vector operations on both fixed and floating point operands.

1.4.16 EF OUTPUT REGISTER. The EF Output Register receives a result from any segment of the pipeline, except the multiplication segment (output of multiplier is two partial products that must be added to produce a result). The output of this register may return to the Register File in the IPU (scalar operations), may update the data in the Memory Buffer File, or may be stored in memory (vectors and store operations).

1.5 GENERAL CHARACTERISTICS

Table 1-1 lists some of the general characteristics of the ASC Central Processor.

1.6 CP INSTRUCTION SET

The ASC Central Processor performs scalar and vector operations through a powerful array of instructions. The instruction set includes Load and Store functions, arithmetic scalar operations, scalar logical instructions, and branching capabilities. Two special instructions, VECT and VECTL, expand the ASC instructions into the vector mode by loading a new set of parameters into the IPU from the Vector Parameter File. The set of vector parameters includes a vector operation code. The function of the vector operation is defined by an additional set of vector instructions that can be loaded only through this vector mode. Table 1-2 lists the instructions in the normal ASC instruction set with their mnemonic code and operation code; figure 1-4 supplies a mapping of scalar Op Codes. Table 1-3 and figure 1-5 contain similar information for the vector mode instructions. Refer to the ASC programming manuals for a more detailed explanation of the uses of each instruction.

Table 1-1. Central Processor General Characteristics

Item	Characteristic
Construction	Layered pipeline
Word Size	16 bits (halfword) -fixed point only 32 bits (singleword) -fixed or floating point 64 bits (doubleword) -floating point only
Instruction word size	32 bits (8 Op Code, 4 R-field, 4 T-field, 4 M-field, 12 N-field)
Memory address size:	
Octet	21 bits (sent to CM)
Word	24 bits (internal to CP)
Memory transfer size	1 octet (256 bits)
Number of memory paths	9: 1-IPU (instruction fetch), 4-MBU's (operand fetch/store) 4-AU's (maintenance - Load/Store Details)
Operation Modes	2: Scalar and Vector
Control:	
Initiate/Terminate	Through CR File in the Peripheral Processor
Operating	Individual pipe level controllers in CP
CP Clock Period	65 nanoseconds

Table 1-2. Vector Instruction Set

ASSMB CODE	MCHN CODE	INSTRUCTION
VA	40	Vector add, fixed point singleword
VAF	42	Vector add, floating point singleword
VAFD	43	Vector add, floating point doubleword
VAH	41	Vector add, fixed point halfword
VAM	44	Vector add magnitude, fixed point singleword
VAMF	46	Vector add magnitude, floating point singleword
VAMFD	47	Vector add magnitude, floating point doubleword
VAMH	45	Vector add magnitude, fixed point singleword
VAND	E0	Vector logical AND, singleword
VANDD	E1	Vector logical AND, doubleword
VC	D0	Vector arithmetic comparison, fixed point singleword
VCAB	EA	Vector compare AND singleword boolean
VCADB	EB	Vector compare AND doubleword boolean
VCAND	E2	Vector logical comparison using AND, singleword
VCANDD	E3	Vector logical comparison using AND, doubleword
VCB	F0	Vector compare fixed point singleword boolean
VCF	D2	Vector arithmetic comparison, floating point singleword
VCFB	F2	Vector compare floating point singleword boolean
VCFD	D3	Vector arithmetic comparison, floating point doubleword
VCFDB	F3	Vector compare floating point doubleword boolean
VCH	D1	Vector arithmetic comparison, fixed point halfword
VCHB	F1	Vector compare fixed point halfword boolean
VCOR	E6	Vector logical comparison using OR, singleword
VCORB	EE	Vector compare OR singleword boolean
VCORD	E7	Vector logical comparison using OR, doubleword
VCORDB	EF	Vector compare OR doubleword boolean
VD	64	Vector divide fixed point, singleword
VDF	66	Vector divide floating point, singleword
VDFD	67	Vector divide floating point, doubleword
VDH	65	Vector divide fixed point, halfword
VDP	68	Vector dot product, fixed point singleword
VDPF	6A	Vector dot product, floating point singleword
VDPFD	6B	Vector dot product, floating point doubleword
VDPH	69	Vector dot product, fixed point halfword
VEQC	EC	Vector logical Equivalence, singleword
VEQCD	ED	Vector logical Equivalence, doubleword
VFDFX	A2	Vector convert floating point doubleword to fixed point singleword

Table 1-2. Vector Instruction Set (Continued)

ASSMB CODE	MCHN CODE	INSTRUCTION
VFHFD	AB	Vector convert fixed point halfword to floating point doubleword
VFHFL	A9	Vector convert fixed point half length to floating point singleword
VFLFH	A1	Vector convert floating point singleword to fixed point halfword
VFLFX	A0	Vector convert floating point singleword to fixed point singleword
VFXFD	AA	Vector convert fixed point singleword to fixed point doubleword
VFXFL	A8	Vector convert fixed point singleword to floating point singleword
VL	50	Vector search for largest arithmetic element, fixed point singleword
VLF	52	Vector search for largest arithmetic element floating point singleword
VLFD	53	Vector search for largest arithmetic element, floating point doubleword
VLH	51	Vector search for largest arithmetic element, fixed point halfword
VLM	54	Vector search for largest magnitude, fixed point singleword
VLMF	56	Vector search for largest magnitude, floating point singleword
VLMFD	57	Vector search for largest magnitude, floating point doubleword
VLMH	55	Vector search for largest magitude, fixed point halfword
VM	6C	Vector multiply, fixed point singleword
VMAP	F8	Vector map singleword
VMAPB	FC	Vector map singleword boolean
VMAPD	FB	Vector map doubleword
VMAPDB	FF	Vector map doubleword boolean
VMAPH	F9	Vector map halfword
VMAPHB	FD	Vector map halfword boolean
VMAX	F4	Vector maximum/minimum fixed point singleword
VMAXF	F6	Vector maximum/minimum floating point singleword
VMAXFD	F7	Vector maximum/minimum floating point doubleword
VMAXH	F5	Vector maximum/minimum fixed point halfword
VMF	6E	Vector multiply, floating point singleword
VMFD	6F	Vector multiply, floating point doubleword

Table 1-2. Vector Instruction Set (Continued)

ASSMB CODE	MCHN CODE	INSTRUCTION
VMG	D8	Vector merge singlewords
VMGD	DB	Vector merge doublewords
VMGH	D9	Vector merge halfwords
VMH	6D	Vector multiply, fixed point halfword
VNFH	AD	Vector normalize fixed point halfword
VNFX	AC	Vector normalize fixed point singleword
VO	D4	Vector order singlewords, fixed point
VOF	D6	Vector order singlewords, floating point
VOFD	D7	Vector order doublewords, floating point
VOH	D5	Vector order halfwords, fixed point
VOR	E4	Vector logical OR, singleword
VORD	E5	Vector logical OR, doubleword
VPP	DC	Vector peak, fixed point singleword
VPPF	DE	Vector peak, floating point singleword
VPPFD	DF	Vector peak, floating point doubleword
VPPH	DD	Vector peak, fixed point halfword
VREP	B8	Replace singlewords in vector $\vec{C}$
VREPB	BC	Vector replace singleword boolean
VREPD	BB	Replace doublewords in vector $\vec{C}$
VREPDB	BF	Vector replace doubleword boolean
VREPH	B9	Replace halfwords in vector $\vec{C}$
VREPHB	BD	Vector replace halfword boolean
VS	48	Vector subtract, fixed point singleword
VSA	C0	Vector arithmetic shift, fixed point singleword
VSAD	C3	Vector arithmetic shift, fixed point doubleword
VSAH	C1	Vector arithmetic shift, fixed point halfword
VSC	CC	Vector circular shift, singleword
VSCD	CF	Vector circular shift, doubleword
VSCH	CD	Vector circular sluift, halfword
VSEL	B0	Select singlewords from vector $\vec{B}$
VSELB	B4	Vector select singleword boolean
VSELD	B3	Select doublewords from vector $\vec{B}$
VSELDB	B7	Vector select doubleword boolean
VSELH	B1	Select halfwords from vector $\vec{B}$
VSELHB	B5	Vector select halfword boolean
VSF	4A	Vector subtract, floating point singleword
VSFD	4B	Vector subtract, floating point doubleword
VSH	49	Vector subtract, fixed point halfword

Table 1-2. Vector Instruction Set (Continued)

ASSMB CODE	MCHN CODE	INSTRUCTION
VSL	C4	Vector logic shift, singleword
VSLD	C7	Vector logical shift, doubleword
VSLH	C5	Vector logical shift, halfword
VSM	4C	Vector subtract magnitude, fixed point singleword
VSMF	4E	Vector subtract magnitude, floating point singleword
VSMFD	4F	Vector subtract magnitude, floating point doubleword
VSMH	4D	Vector subtract magnitude, fixed point halfword
VSS	58	Vector search for smallest arithmetic element, fixed point singleword
VSSF	5A	Vector search for smallest arithmetic element, floating point singleword
VSSFD	5B	Vector search for smallest arithmetic element, floating point doubleword
VSSH	59	Vector search for smallest arithmetic element, fixed point halfword
VSSM	5C	Vector search for smallest magnitude, fixed point singleword
VSSMF	5E	Vector search for smallest magnitude, floating point singleword
VSSNFD	5F	Vector search for smallest magnitude, floating point doubleword
VSSMH	5D	Vector search for smallest magnitude, fixed point halfword
VXOR	E8	Vector logical Exclusive OR, singleword
VXORD	E9	Vector logical Exclusive OR, doubleword

OP BITS 0–3

OP BITS 4–7	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LRL	STZ	LN	A	AI	A	AI	ISE	MCP	FLFX	VECT	SAH		AND	ANDI
1		LEM	STZH	LNH	AH	AIH			ISNE	BCC	FLFH		SAH		ANDD	
2		LAM	SPS	LNF	AF	LEA	A	AI	DSE	INT	FDFX				CAND	CANDI
3		LAC	STZD	LND	AFD				DSNE	PSH			SAD		CANDD	
4		L	ST	STN	AM	LI	D	DI	BCLE	MCW			SL		OR	ORI
5		LLL	STLL	STNH	AMH	LIH	DH	DIH	BCG	BRC			SLH		ORD	
6		LLA	STRL	STNF	AMF	LEA	DF		BCLE	XEC			RVS		COR	CORI
7		LD	STD	STND	AMFD		DFD		BCG	PUL			SLD		CORD	
8		L	ST	LNM	S	SI	M	MI	IBZ	BLB	FXFL		C	CI	XOR	XORI
9		LLR	STLR	LNMH	SH	SIH			IBNZ	BLX	FHFL		CH	CIH	XORD	
A		XCH	STOH	LNMF	SF		M	MI	DBX	FORK	FXFD		CF			
B		LF	STF	LNMD	SFD				DBNZ	JOIN	FHFD		CFD			
C		L	ST	LM	SM	LI	M	MI	IBZ	BXEC	NFX		SC		EQC	EQCI
D		LRR	STRR	LMH	SMH		MH	MIH	IBNZ	BAE	NFH		SCH		EQCD	
E		LO	STO	LMF	SMF		MF		DBZ	PB	SCLK		C	CI		
F		LFM	STFM	LMD	SMFD		MFD		DBNZ	MOD			SCD			

(A)132345

Figure 1-4. Scalar Op Code Map

Table 1-3. Scalar Instruction Set

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
A	Add to arithmetic register, fixed point singleword	40	r,[@] [=] n[,x]
A	Add to base register, fixed point singleword	60	r,[@] [=] n[,x]
A	Add to index or vector parameter register	62	r,[@] [=] n[,x]
AF	Add to arithmetic register, floating point singleword	42	r,[@] [=] n[,x]
AFD	Add to arithmetic register, floating point doubleword	43	r,[@] [=] n[,x]
AH	Add to arithmetic register, fixed point halfword	41	r,[@] [=] n[,x]
AI	Add immediate to arithmetic register, fixed point singleword	50	r,i[,x]
AI	Add immediate to base register, fixed point singleword	70	r,i[,x]
AI	Add immediate to index or vector parameter register, fixed point singleword	72	r,i[,x]
AIH	Add immediate to arithmetic register, fixed point halfword	51	r,i[,x]
AM	Add magnitude to arithmetic register, fixed point singleword	44	r,[@] [=] n[,x]
AMF	Add magnitude to arithmetic register, floating point singleword	46	r,[@] [=] n[,x]
AMFD	Add magnitude to arithmetic register, floating point doubleword	47	r,[@] [=] n[,x]
AMH	Add magnitude to arithmetic register, fixed point halfword	45	r,[@] [=] n[,x]
AND	AND, singleword - arithmetic register	E0	r,[@] [=] n[,x]
ANDD	AND, doubleword - arithmetic register	E1	r,[@] [=] n[,x]
ANDI	AND immediate, singleword - arithmetic register	F0	r,i[,x]
B	Unconditional branch, Assembler supplies R field of 7	91	[@[=]] n[,x]
BAE	Branch on arithmetic exception condition true	9D	m,[@[=]] n[,x]
BCC	Branch on compare code true	91	m,[@[=]] n[,x]
BCG	Branch on arithmetic register greater than	85	r,r,n
BCG	Branch on index or vector register greater than	87	r,r,n
BCLE	Branch on arithmetic register less than or equal	84	r,r,n
BCLE	Branch on index or vector register less than or equal	86	r,r,n
BCM	Branch on compare code of mixed zeros and ones, Assembler supplies R field of 4	91	[@[=]] n[,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
BCNM	Branch on compare code of not mixed, Assembler supplies R field of 3	91	[@[=]]n[,x]
BCNO	Branch on compare code of not all ones, Assembler supplies R field of 5	91	[@[=]]n[,x]
BCNZ	Branch on compare code of not all zeros, Assembler supplies the R field of 6	91	[@[=]]n[,x]
BCO	Branch on compare code of all bits are one, Assembler supplies R field of 2	91	[@[=]]n[,x]
BCZ	Branch on compare code of all bits are zero, Assembler supplies R field of one	91	[@[=]]n[,x]
BD	Branch on divide check, Assembler supplies R field of 8	9D	[@[=]]n[,x]
BDO	Branch on divide check or floating point exponent overflow, Assembler supplies R field of A	9D	[@[=]]n[,x]
BDU	Branch on divide check or floating point exponent underflow, Assembler supplies R field of 9	9D	[@[=]]n[,x]
BDUO	Branch on divide check or floating point exponent overflow or underflow, Assembler supplies R field of B	9D	[@[=]]n[,x]
BDX	Branch on divide check or fixed point overflow, Assembler supplies R field of C	9D	[@[=]]n[,x]
BDXO	Branch on divide check or fixed point overflow or floating point exponent overflow, Assembler supplies R field of E	9D	[@[=]]n[,x]
BDXU	Branch on divide check or fixed point overflow or floating point exponent underflow, Assembler supplies R field of D	9D	[@[=]]n[,x]
BDXUO	Branch on divide check or fixed point overflow or floating point exponent overflow or underflow, Assembler supplies R field of F	9D	[@[=]]n[,x]
BE	Branch on compare code of equal, Assembler supplies R field of one	91	[@[=]]n[,x]
BG	Branch on compare code of greater than, Assembler supplies R field of 2	91	[@[=]]n[,x]
BGE	Branch on compare code of greater than or equal, Assembler supplies R field of 3	91	[@[=]]n[,x]
BL	Branch on compare code of less than, Assembler supplies R field of 4	91	[@[=]]n[,x]
BLB	Branch and load base register with program counter	98	r,[@[=]]n[,x]
BLE	Branch on compare code of less than or equal, Assembler supplies R field of 5	91	[@[=]]n[,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
BLR	Branch on logical result	95	m,[@[=]]n[,x]
BLX	Branch and load index or vector register with program counter	99	r,[@[=]]n[,x]
BMI	Branch on result code of negative, Assembler supplies the R field of 4	95	[@[=]]n[,x]
BNE	Branch on compare code of not equal, Assembler supplies R field of 6	91	[@[=]]n[,x]
BNZ	Branch on result code of not zero, Assembler supplies the R field of 6	95	[@[=]]n[,x]
BO	Branch on floating point exponent overflow, Assembler supplies R field of 2	9D	[@[=]]n[,x]
BPL	Branch on result code of positive, Assembler supplies the R field of 2	95	[@[=]]n[,x]
BRC	Branch on result code true	95	m,[@[=]]n[,x]
BRM	Branch on result code of bits mixed zeros and ones, Assembler supplies the R field of 4	95	[@[=]]n[,x]
BRNM	Branch on result code of bits not mixed zeros and ones, Assembler supplies the R field of 3	95	[@[=]]n[,x]
BRNO	Branch on result code of not all bits ones, Assembler supplies the R field of 5	95	[@[=]]n[,x]
BRNZ	Branch on result code of not all bits zeros, Assembler supplies the R field of 6	95	[@[=]]n[,x]
BRO	Branch on result code of all bits are one, Assembler supplies the R field of 2	95	[@[=]]n[,x]
BRZ	Branch on result code of all bits are zero, Assembler supplies the R field of one	95	[@[=]]n[,x]
BU	Branch on floating point exponent underflow, Assembler supplies R field of one	9D	[@[=]]n[,x]
BUO	Branch on floating point exponent underflow or overflow, Assembler supplies R field of 3	9D	[@[=]]n[,x]
BX	Branch on fixed point overflow, Assembler supplies R field of 4	9D	[@[=]]n[,x]
BXEC	Branch on Execute branch condition true, Assembler supplies R field of one or odd	9C	[@]n[,x]
BXO	Branch on fixed point overflow or floating point exponent overflow, Assembler supplies R field of 6	9D	[@[=]]n[,x]
BXU	Branch on fixed point overflow or floating point exponent underflow, Assembler supplies R field of 5	9D	[@[=]]n[,x]
BXUO	Branch on fixed point overflow or floating point exponent overflow or underflow, Assembler supplies R field of 7	9D	[@[=]]n[,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
BZ	Branch on result code of zero, Assembler supplies the R field of one	95	[@[=]]n[,x]
BZM	Branch on result code of zero or negative, Assembler supplies the R field of 5	95	[@[=]]n[,x]
BZP	Branch on result code of zero or positive, Assembler supplies the R field of 3	95	[@[=]]n[,x]
C	Compare arithmetic register, fixed point singleword	C8	r,[@] [=]n[,x]
C	Compare index or vector register, fixed point singleword	CE	r,[@] [=]n[,x]
CAND	Compare logical AND, singleword - arithmetic register	E2	r,[@] [=]n[,x]
CANDD	Compare logical AND, doubleword - arithmetic register	E3	r,[@] [=]n[,x]
CANDI	Compare immediate logical AND, singleword - arithmetic register	F2	r,i[,x]
CF	Compare arithmetic register, floating point singleword	CA	r,[@] [=]n[,x]
CFD	Compare arithmetic register, floating point doubleword	CB	r,[@] [=]n[,x]
CH	Compare arithmetic register, fixed point halfword	C9	r,[@] [=]n[,x]
CI	Compare immediate arithmetic register, fixed point singleword	D8	r,i[,x]
CI	Compare index or vector register with immediate singleword	DE	r,i[,x]
CIH	Compare arithmetic register immediate, fixed point halfword	D9	r,i[,x]
COR	Compare logical OR, singleword - arithmetic register	E6	r,[@] [=]n[,x]
CORD	Compare logical OR, doubleword - arithmetic register	E7	r,[@] [=]n[,x]
CORI	Compare immediate logical OR, singleword - arithmetic register	F6	r,i[,x]
D	Divide into arithmetic register, fixed point singleword	64	r,[@] [=]n[,x]
DBNZ	Decrement arithmetic register, test, and branch on not zero	8B	r,[@[=]]n[,x]
DBNZ	Decrement index or vector register, test, and branch on not zero	8F	r,[@[=]]n[,x]
DBZ	Decrement arithmetic register, test, and branch on zero	8A	r,[@[=]]n[,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
DBZ	Decrement index or vector register, test, and branch on zero	8E	r,[@[=]] n[,x]
DF	Divide into arithmetic register, floating point singleword	66	r,[@] [=] n[,x]
DFD	Divide into arithmetic register, floating point doubleword	67	r,[@] [=] n[,x]
DH	Divide into arithmetic register, fixed point halfword	65	r,[@] [=] n[,x]
DI	Divide immediate into arithmetic register, fixed point singleword	74	r,i[,x]
DIH	Divide immediate into arithmetic register, fixed point halfword	75	r,i[,x]
DSE	Decrement arithmetic register, test, and skip on equal	82	r,[@] [=] n[,x]
DSNE	Decrement arithmetic register, test, and skip on not equal	83	r,[@] [=] n[,x]
EQC	Equivalence, singleword - arithmetic register	EC	r,[@] [=] n[,x]
EQCD	Equivalence, doubleword - arithmetic register	ED	r,[@] [=] n[,x]
EQCI	Equivalence immediate, singleword - arithmetic register	FC	r,i[,x]
FDFX	Convert floating point doubleword to fixed point singleword	A2	r,[@] n[,x]
FHFD	Convert fixed point halfword to floating point doubleword	AB	r,[@] n[,x]
FHFL	Convert fixed point halfword to floating point singleword	A9	r,[@] n[,x]
FLFH	Convert floating point singleword to fixed point halfword	A1	r,[@] n[,x]
FLFX	Convert floating point singleword to fixed point singleword	A0	r,[@] n[,x]
FORK	Set fork indicator	A9	---
FXFD	Convert fixed point singleword to floating point doubleword	AA	r,[@] n[,x]
FXFL	Convert fixed point singleword to floating point singleword	A8	r,[@] n[,x]
IBNZ	Increment arithmetic register, test, and branch on not zero	89	r,[@[=]] n[,x]
IBNZ	Increment index or vector register, and branch on not zero	8D	r,[@[=]] n[,x]
IBZ	Increment arithmetic register, test, and branch on zero	88	r,[@[=]] n[,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
IBZ	Increment index or vector register, test, and branch on zero	8C	r,[@[=]]n[,x]
INT	Interpret - arithmetic register	92	r,[@] [=] n [,x]
ISE	Increment arithmetic register, test and skip on equal	80	r,[@] [=] n [,x]
ISNE	Increment arithmetic register, test, and skip on not equal	81	r,[@] [=] n [,x]
JOIN	Reset fork indicator	9B	---
L	Load arithmetic register, singleword	14	r,[@] [=] n [,x]
L	Load base register, singleword	18	r,[@] [=] n [,x]
L	Load index register or vector parameter register, singleword	1C	r,[@] [=] n [,x]
LAC	Load arithmetic exception condition	13	[@[=]] n [,x]
LAM	Load arithmetic mask	12	[@[=]] n [,x]
LD	Load arithmetic register doubleword	17	r,[@] [=] n [,x]
LEA	Load effective address into base register	52	r,[@] [=] n [,x]
LEA	Load effective address into index or vector register	56	r,[@] [=] n [,x]
LEM	Load arithmetic exception mask and condition	11	[@[=]] n [,x]
LF	Load base register file A, m=0	1B	m,[@] n [,x]
LF	Load base register file B, m=1	1B	m,[@] n [,x]
LF	Load arithmetic register file C, m=2	1B	m,[@] n [,x]
LF	Load arithmetic register file D, m=3	1B	m.[@] n [,x]
LF	Load index register file X, m=4	1B	m,[@] n [,x]
LF	Load vector parameter register file V, m=5	1B	m,[@] n [,x]
LFM	Load all six eight-word register files	1F	[@] n [,x]
LI	Load immediate into arithmetic register singleword	54	r,i [,x]
LI	Load immediate into index register, or vector parameter register, singleword	5C	r,i [,x]
LIH	Load immediate into arithmetic register, halfword	55	r,i [,x]
LLA	Load look ahead	16	i
LLL	Load memory left halfword, indexed, into arithmetic register left halfword	15	r,[@] [=] n [,x]
LLR	Load memory right halfword, indexed, into arithmetic register left halfword	19	r,[@] [=] n [,x]
LM	Load magnitude, fixed point singleword, arithmetic register	3C	r,[@] [=] n [,x]
LMD	Load magnitude, floating point doubleword arithmetic register	3F	r,[@] [=] n [,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
LMF	Load magnitude, floating point singleword, arithmetic register	3E	r,[@] [=] n [,x]
LMH	Load magnitude, fixed point halfword, arithmetic register	3D	r,[@] [=] n [,x]
LN	Load negative, fixed point singleword, arithmetic register	30	r,[@] [=] [,x]
LND	Load negative, floating point doubleword, arithmetic register	33	r,[@] [=] n [,x]
LNF	Load negative, floating point singleword, arithmetic register	32	r,[@] [=] n [,x]
LNH	Load negative, fixed point halfword, arithmetic register	31	r,[@] [=] n [,x]
LNM	Load negative magnitude, fixed point singleword, arithmetic register	38	r,[@] [=] n [,x]
LNMD	Load negative magnitude, floating point doubleword, arithmetic register	3B	r,[@] [=] n [,x]
LNMF	Load negative magnitude, floating point singleword, arithmetic register	3A	r,[@] [=] n [,x]
LNMH	Load negative magnitude, fixed point halfword, arithmetic register	39	r,[@] [=] n [,x]
LO	Load arithmetic register with ones complement singleword	1E	r,[@] [=] n [,x]
LRL	Load memory left halfword, indexed, into arithmetic register right halfword	10	r,[@] [=] n [,x]
LRR	Load memory right halfword, indexed, into arithmetic register right halfword	1D	r,[@] [=] n [,x]
M	Multiply fixed point singleword - arithmetic register	6C	r,[@] [=] n [,x]
M	Multiply, fixed point singleword - base register	68	r,[@] [=] n [,x]
M	Multiply, fixed point singleword - index or vector parameter register	6A	r,[@] [=] n [,x]
MCP	Monitor call and proceed	90	i [,x]
MCW	Monitor call and wait	94	i [,x]
MF	Multiply, floating point singleword - arithmetic register	6E	r,[@] [=] n [,x]
MFD	Multiply, floating point doubleword - arithmetic register	6F	r,[@] [=] n [,x]
MH	Multiply, fixed point halfword - arithmetic register	6D	r,[@] [=] n [,x]
MI	Multiply immediate, fixed point singleword - arithmetic register	7C	r,i [,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
MI	Multiply immediate, fixed point singleword - base register	78	r,i[,x]
MI	Multiply immediate, fixed point singleword - index or vector parameter register	7A	r,i[,x]
MIH	Multiply immediate, fixed point halfword - arithmetic register	7D	r,i[,x]
MOD	Modify stack parameter doubleword	9F	r,[@]n[,x]
NFH	Normalize fixed point halfword	AD	r,[@]n[,x]
NFX	Normalize fixed point singleword	AC	r,[@]n[,x]
NOP	Take next instruction, Assembler supplies R field of zero	91	[@[=]]n[,x]
OR	OR, singleword - arithmetic register	E4	r,[@] [=]n[,x]
ORD	OR, doubleword - arithmetic register	E5	r,[@] [=]n[,x]
ORI	OR immediate, singleword - arithmetic register	F4	r,i[,x]
PB	Prepare to Branch	9E	m,[@[=]]n[,x]
PSH	Push word into last-in-first-out stack	93	r,[@]n[,x]
PUL	Pull word from last-in-first-out stack	97	r,[@]n[,x]
RVS	Bit reversal, singleword - arithmetic register	C6	r,i[,x]
S	Subtract from arithmetic register, fixed	48	r,[@] [=]n[,x]
SA	Arithmetic shift, fixed point singleword - arithmetic register	C0	r,i[,x]
SAD	Arithmetic shift, fixed point doubleword - arithmetic register	C3	r,i[,x]
SAH	Arithmetic shift, fixed point halfword - arithmetic register	C1	r,i[,x]
SC	Circular shift, singleword - arithmetic register	CC	r,i[,x]
SCD	Circular shift, doubleword - arithmetic register	CF	r,i[,x]
SCH	Circular shift, halfword - arithmetic register	CD	r,i[,x]
SCLK	Store 32-bit fixed point clock	AE	[@]n[,x]
SF	Subtract from arithmetic register, floating point singleword	4A	r,[@] [=]n[,x]
SFD	Subtract from arithmetic register, floating point doubleword	4B	r,[@] [=[n[,x]
SH	Subtract from arithmetic register, fixed point halfword	49	r,[@] [=]n[,x]
SI	Subtract immediate from arithmetic register, fixed point singleword	58	r,i[,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
SIH	Subtract immediate from arithmetic register, fixed point halfword	59	r,i[,x]
SL	Logical shift, singleword - arithmetic register	C4	r,i[,x]
SLD	Logical shift, doubleword - arithmetic register	C7	r,i[,x]
SLH	Logical shift, halfword - arithmetic register	C5	r,i[,x]
SM	Subtract magnitude from arithmetic register, fixed point singleword	4C	r,[@] [=[n[,x]
SMF	Subtract magnitude from arithmetic register, floating point singleword	4E	r,[@] [=] n [,x]
SMFD	Subtract magnitude from arithmetic register, floating point doubleword	4F	r,[@] [=] n [,x]
SMH	Subtract magnitude from arithmetic register, fixed point halfword	4D	r,[@] [=] n [,x]
SPS	Store program status word	22	[@] n [,x]
ST	Store arithmetic register, singleword	24	r,[@] n [,x]
ST	Store base register, singleword	28	r,[@] n [,x]
ST	Store index register or vector parameter register, singleword	2C	r,[@] n [,x]
STD	Store arithmetic register, doubleword	27	r,[@] n [.x]
STF	Store base register file A, M=0	2B	m,[@] n [,x]
STF	Store base register file B, M=1	2B	m,[@] n [,x]
STF	Store arithmetic register file C, M=2	2B	m,[@] n [,x]
STF	Store arithmetic register file D, M=3	2B	m,[@] n [,x]
STF	Store index register file X, M=4	2B	m,[@] nl,x]
STF	Store vector parameter register file V, M=5	2B	m,[@] n [,x]
STFM	Store all six eight word register files	2F	[@] n [,x]
STLL	Store arithmetic left halfword into memory left halfword, indexed	25	r,[@] n [,x]
STLR	Store arithmetic register left halfword into memory right halfword, indexed	29	r,[@] n [,x]
STN	Store negative, fixed point word	34	r,[@] n [,x]
STND	Store negative, floating point doubleword	37	r,[@] n [,x]
STNF	Store negative, floating point word	36	r,[@] n [,x]
STNH	Store negative, fixed point halfword	35	r,[@] n [,x]
STO	Store ones complement, word	2E	r,[@] n [,x]
STOH	Store ones complement, halfword	2A	r,[@] n [,x]
STRL	Store arithmetic right halfword into memory left halfword, indexed	26	r,[@] n [,x]

Table 1-3. Scalar Instruction Set (Continued)

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT
STRR	Store arithmetic register right halfword into memory right halfword, indexed	2D	r,[@]n[,x]
STZ	Store zero, word	20	[@]n[,x]
STZD	Store zero, doubleword	23	[@]n[,x]
STZH	Store zero, halfword	21	[@]n[,x]
VECT	Execute vector parameter file, Assembler supplies R field of one	B0	[@]n[,x]
VECTL	Load and execute vector parameter file, Assembler supplies R field of zero	B0	[@]n[,x]
XCH	Exchange - arithmetic register with effective address	1A	r,[@]n[,x]
XEC	Execute addressed instruction in line	96	[@[=]]n[,x]
XOR	Exclusive OR, singleword - arithmetic register	E8	r,[@][=]n[,x]
XORD	Exclusive OR, doubleword - arithmetic register	E9	r,[@][=]n[,x]
XORI	Exclusive OR immediate, singleword - arithmetic register	F8	r,i[,x]

OP BITS 0-3

OP BITS 4-7	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					VA	VL					VFLFX	VSEL*	VSA	VC	VAND	VCB*
1					VAH	VLH					VFLFH	VSELH*	VSAH	VCH	VANDD	VCHB*
2					VAF	VLF					VFDFX			VCF	VCAND	VCFB*
3					VAFD	VLFD						VSELD*	VSAD	VCFD	VCANDD	VCFDB
4					VAM	VLM	VD					VSELB	VSL	VO	VOR	VMAX*
5					VAMH	VLMH	VDH					VSELHB*	VSLH	VOH	VORD	VMAXH*
6					VAMF	VLMF	VDF							VOF	VCOR	VMAXF*
7					VAMFD	VLMFD	VDFD					VSELDB*	VSLD	VOFD	VCORD	VMAXD*
8					VS	VSS	VDP				VFXFL	VREP*		VMG	VXOR	VMAP*
9					VSH	VSSH	VDPH				VFHFL	VREPH*		VMGH	VXORD	VMAPH*
A					VSF	VSSF	VDPF				VFXFD				VCAB	
B					VSFD	VSSD	VDPFD				VFHFD	VREPD*		VMG	VCADB*	VMAPD*
C					VSM	VSSM	VM				VNFX	VREPB*	VCS	VPP	VEQC	VMAPB*
D					VSMH	VSSMH	VMH				VNFH	VREPHB	VCSH	VPPH	VEQCD	VMAPBH*
E					VSMF	VSSMF	VMF							VPPF	VCORB*	
F					VSMFD	VSSMFD	VMFD					VREPDB*	VCSD	VPPFD	VCORDB*	VMAPDB*

(A)132346

NOTE: BLANK BOXES REPRESENT ILLEGAL OP CODES.
*4 PIPE ONLY

Figure 1-5. Vector Op Code Map

1.7 INSTRUCTION FORMAT

The instruction word of the Central Processor contains 32 bits and is divided into five fields as shown in figure 1-6.

- *Op-Field.* The Op-Field specifies the machine instruction to be executed.

- *R-Field.* The R-Field addresses one of 16 registers from the arithmetic, base, or index register group.

- *T-Field.* The T-field is an address modifier tag that has the following interpretation:

T	ADDRESSING TYPE	VIRTUAL ADDRESS, α, OF MEMORY OPERAND
0	DIRECT ADDRESS	N + (M)
1–7	INDEXED ADDRESS	N + (M) + (T)
8	INDIRECT	(N + (M))
9–F	INDEXED INDIRECT ADDRESS	(N + (M) + (T − 8))

(A)132473

A symbol of expression enclosed by parentheses () represents "the contents of".

The T-field (figure 1-7) may be divided into an I-bit and an X-field where the most significant I-bit designates indirect addressing and the 3-bit X-field specifies one of seven index registers used in the indexing operation. The index registers are physically assigned to register file address locations 21 through 27 (hexadecimal). A special set of index instructions are used to load, store, modify, and test the index registers.

Displacement indexing is provided such that the indexing operation is compatible with word size; i.e., the index registers are automatically aligned according to word size. If an index register contains the value K, the Kth element of an array is accessed, whether it is a halfword, singleword, or doubleword.

- *M-Field.* The M-field is a base register designator. It is used to extend the addressing range capability of the ASC to a potential 16.7 million words. The M-field selects one of fifteen 24-bit base registers to be added to the N-field displacement before indexing or indirect addressing. No base addressing is used when M equals 0.

- *N-Field.* The N-field is the address displacement relative to the base address contained in M.

The M- and N-fields also may be interpreted as immediate operands when immediate instructions are specified by the operation code.

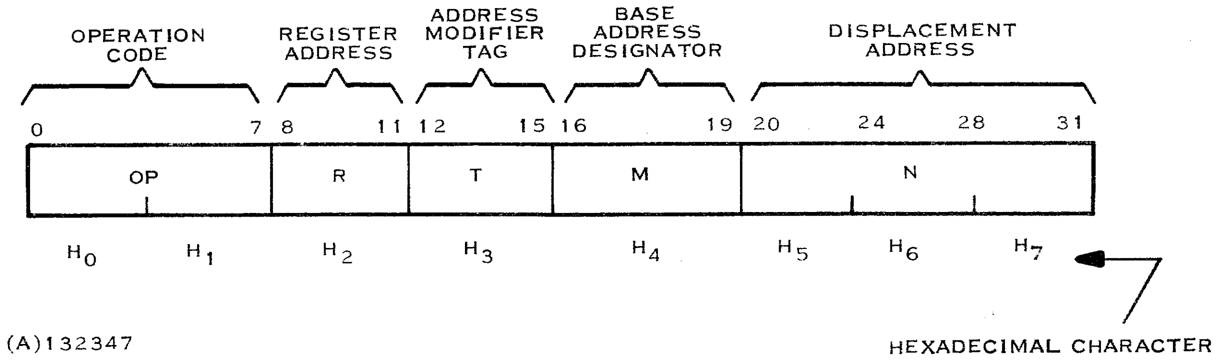


Figure 1-6. ASC Instruction Word Format

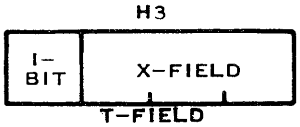


Figure 1-7. T-Field Subdivision

1.8 DATA FORMATS

Four data format representations may be used in the ASC:

- *Fixed point, single length, 32-bit word* (see figure 1-8).

  The sign bit is zero for positive numbers and one for negative. Negative numbers are represented in twos complement notation. The binary point is to the right of the least significant bit (LSB), particularly for multiplication or division. The result after addition is the same as though two binary fractions were added.

- *Fixed point, half length, 16-bit word* (two half length words are shown in figure 1-9).

  The sign bit is zero for positive numbers and one for negative. Negative numbers are represented in twos complement notation. The binary point is to the right of the LSB. Numbers are in fixed point signed integer notation.

- *Floating point, single length, 32-bit word* (see figure 1-10).

  The sign bit is zero for positive numbers and one for negative. Sign and magnitude representation is used for the fractional portion, bits 0, 8 through 31. The binary point is to the left of the MSB of the fraction (between bits 7 and 8).

  The biased hexedecimal exponent has the range of 00_{16} to $7F_{16}$, which covers the base 16 exponent range 16^{-64} to 16^{+63}.

  If the value 40 hex is subtracted from the biased exponent, a number is obtained which is signed integer twos complement notation (sign in bit position 1) can be converted to its equivalent decimal value. Sixteen raised to this decimal power gives a number which when multiplied by the fraction produces the number that was represented in floating point notation.

Examples:

Floating point		Decimal value
4110	0000	$(1/16) \times 16^1 = 1$
4210	0000	$(1/16) \times 16^2 = 16$
C110	0000	$-(1/16) \times 16^1 = -1$
7FF0	0000	$(15/16) \times 16^{+63}$
0010	0000	$(1/16) \times 16^{-64} = 16^{-65}$

By definition:

0000	0000	zero
7FFF	FFFF	$+\infty$
FFFF	FFFF	$-\infty$
7F00	0000	Indefinite (machine generated)
XX00	0000	Indefinite (dirty zero)

- *Floating point, double length, 64-bit word* (see figure 1-11).

The sign bit is zero for positive numbers and one for negative. Sign and magnitude representation is used for the fractional portion, bits 0, 8 through 63. The binary point is to the left of the MSB of the fraction (between bits 7 and 8).

The biased hexadecimal exponent has the range 00_{16} to $7F_{16}$, which covers the base sixteen exponent range 16^{-64} to 16^{+63}.

Subtracting the value 40_{16} from the biased exponent yields a number which, in signed integer twos complement notation (sign in bit position 1), can be converted to its equivalent decimal value. Sixteen raised to this power gives a number which when multiplied by the fraction produces the number that was represented in floating point notation.

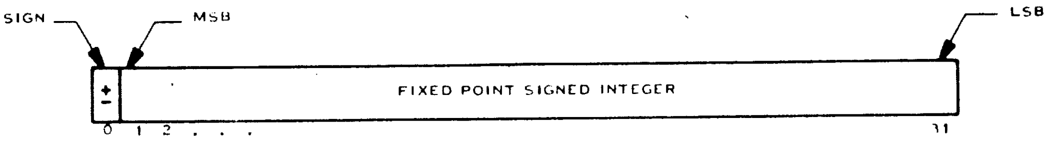


Figure 1-8. 32-Bit, Fixed Point Data Word Format

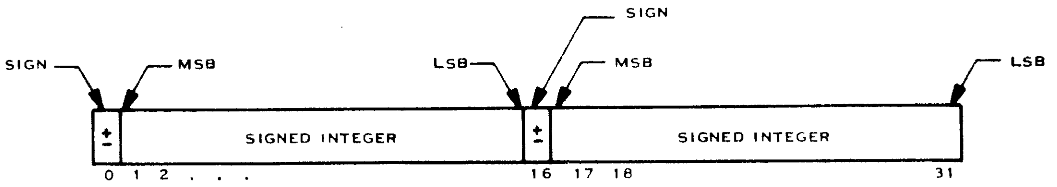


Figure 1-9. 16-Bit, Fixed Point Data Word Format

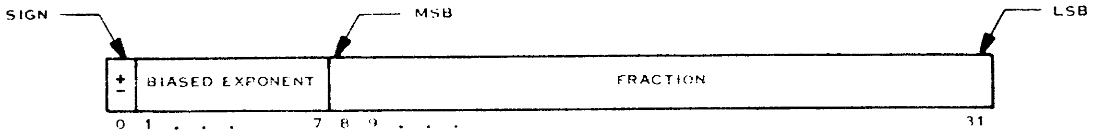


Figure 1-10. 32-Bit, Floating Point Data Word Format

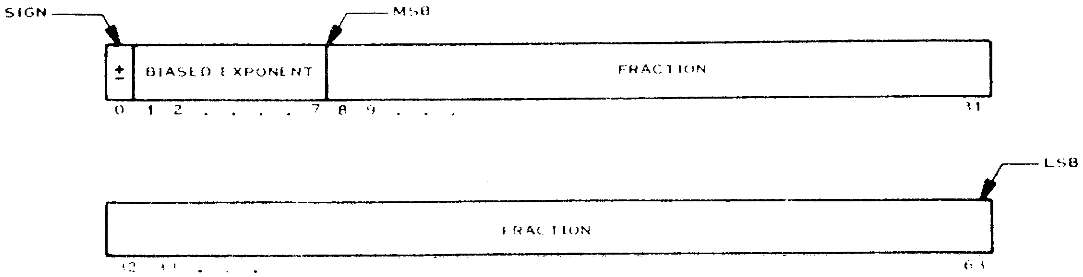


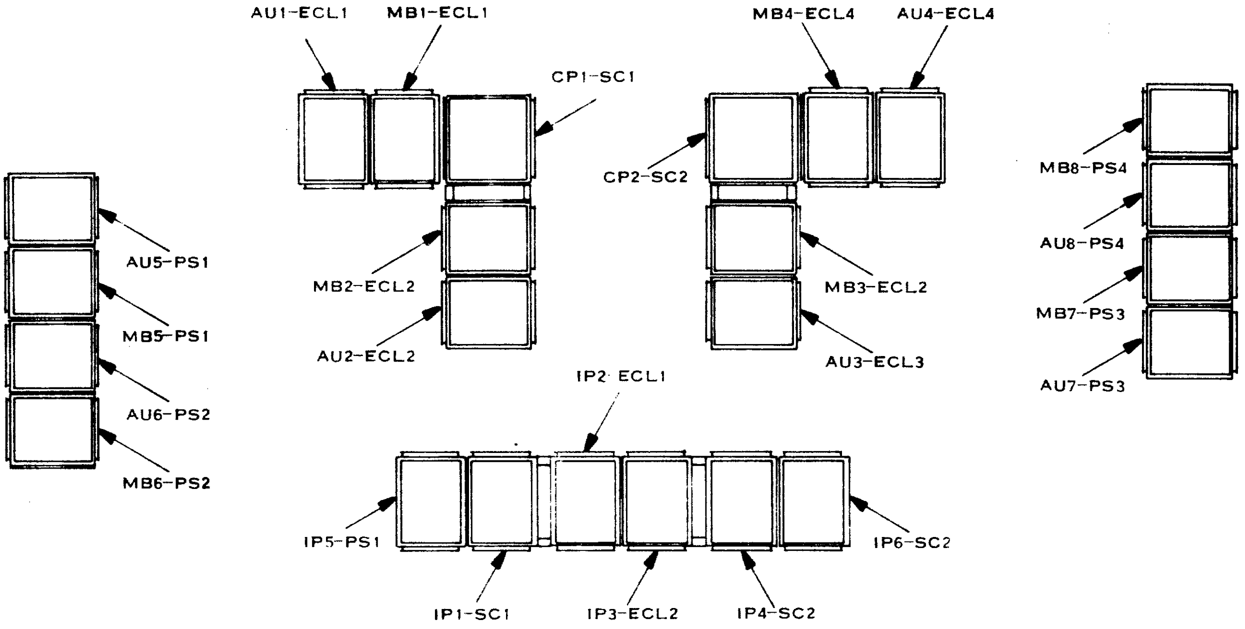
Figure 1-11. 64-Bit, Floating Point Data Word Format

1.9 PHYSICAL DESCRIPTION

The ASC-4X Central Processor in a four-pipe configuration is housed in a series of vertical logic and service columns. Figure 1-12 illustrates a typical layout for these columns; their actual arrangement may be changed to meet the physical requirements of the particular site. Each vertical logic column (IPU, MBU or AU) contains a three motherboard chassis capable of accepting up to 66 logic cards. Other vertical columns containing mounting space for power supplies provide the dc power requirements for their respective CP unit. Between the three main logic columns is a service column that contains the connector panels for the cable connections between the logic columns. In addition to this central service column, other non-logic service columns provide electrical output busses and water input plumbing for the CP cooling system.

1.9.1 COOLING SYSTEM. The Central Processor cooling system consists of a combination of forced air and circulated, cooled water to dissipate heat generated by the logic circuits. This cooling system is represented schematically in figure 1-13. The cold plate between each set of logic cards is a copper plate with small tubes running through it. Cooled water pumped through these tubes absorbs heat from the air surrounding the cold plate and carries the heat away from the logic card area to a heat exchanger. The heat exchanger releases the heat to the surrounding air and returns the cooled water to the logic chassis to complete the cycle. A blower assembly in each logic column aids cooling by circulating room temperature air past the logic cards.

1.9.2 LOGIC CIRCUITS. Central Processor logic is implemented on 9-1/2 inch by 7-1/2 inch printed circuit boards using Emitter Coupled Logic (ECL) integrated circuit packages. A 272-pin connector on one end of the circuit board mates with a corresponding receptacle in one of three motherboards in a vertical logic column. The motherboard supplies inter-chassis wiring connections plus a bus of common signals, bias voltages and ground. Figure 1-14 illustrates the different logic circuits in the ECL logic set. Table 1-4 defines the function of each of these logic circuits. Refer to Section VI, Parts Listing, of this manual for a listing of the logic cards by chassis location.



(B)128426

Figure 1-12. Typical ASC Central Processor Four-Pipe Configuration

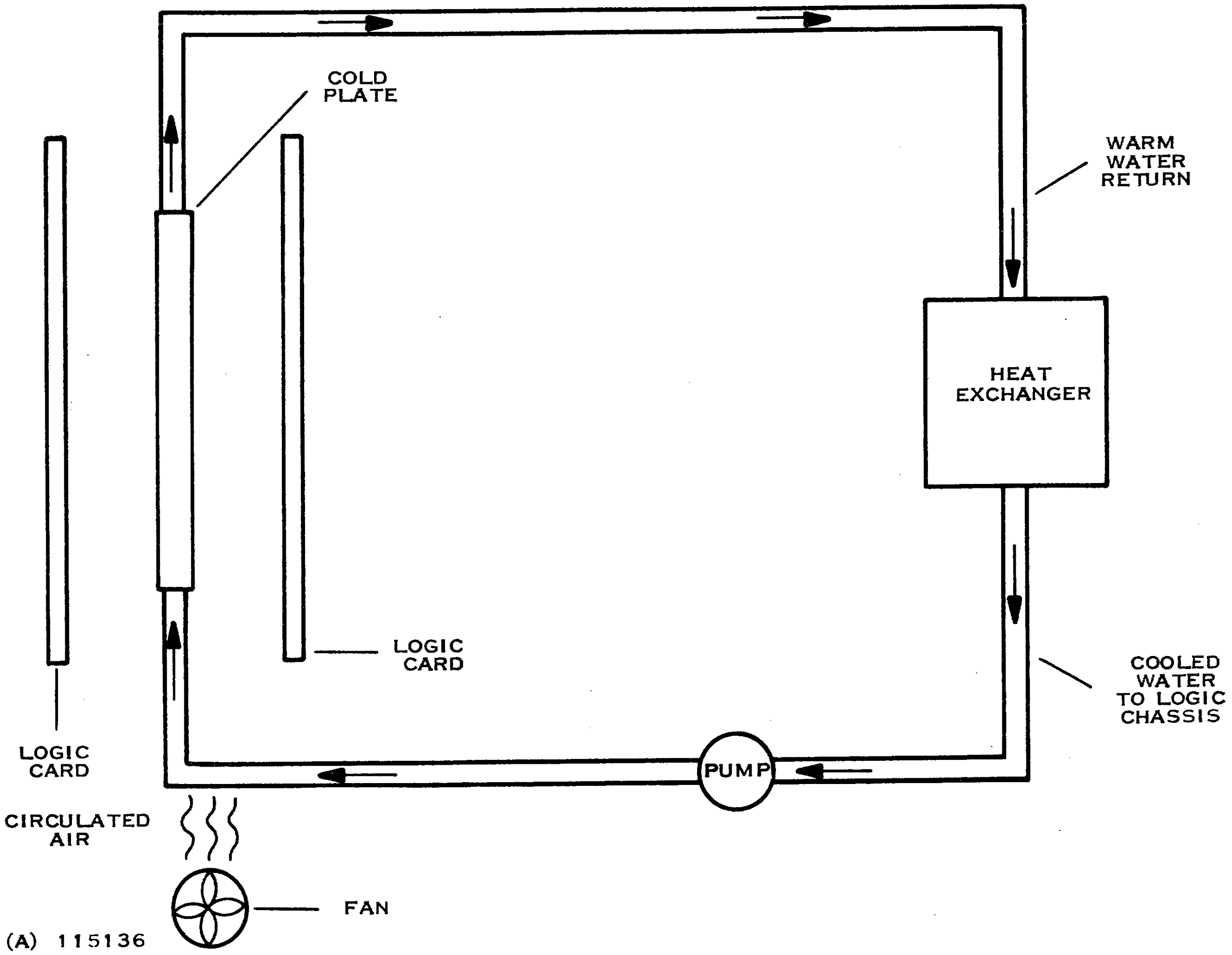
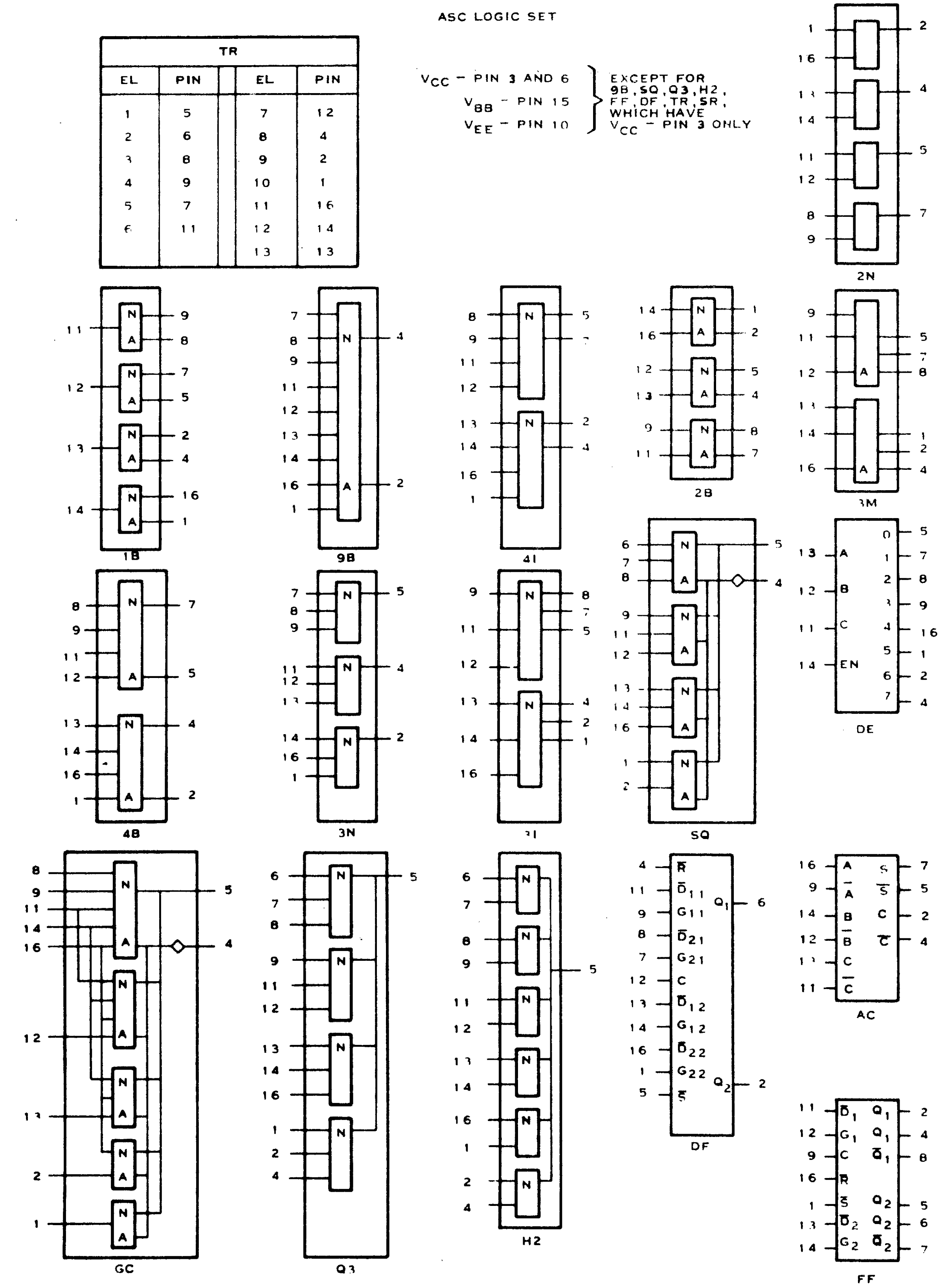


Figure 1-13. Schematic Representation of CP Cooling System

ASC LOGIC SET

TR			
EL	PIN	EL	PIN
1	5	7	12
2	6	8	4
3	8	9	2
4	9	10	1
5	7	11	16
6	11	12	14
		13	13

V_{CC} – PIN 3 AND 6
V_{BB} – PIN 15
V_{EE} – PIN 10
} EXCEPT FOR 9B, SQ, Q3, H2, FF, DF, TR, SR, WHICH HAVE V_{CC} – PIN 3 ONLY



(B) 109045

Figure 1-14. ECL Circuits

Table 1-4. ECL Circuit Types

Type	Title
1B	Four single to double ended converters
2N	Four 2-input inverting gates
2B	Three 2-input complementary gates
3N	Three 3-input inverting gates
4B	Two 4-input complementary gates
9B	9-input complementary gate
3I	Two 3-input, 3-output inverting gates
3M	Two 3-input, 3-output non-inverting gates
4I	Two 4-input, 2-output inverting gates
SQ	Three 3-input, One 2-input gates with dotted complementary outputs
GC	Four bit group carry gate structure
Q3	Four 3-input with dotted inverted output
DE	Three bit decoder with enable
AC	Full sum-carry with complementary outputs
H2	Six 2-input with dotted inverted outputs
FF	Two single-input gated clocked latches
DF	Two 2-input gated clocked latches
TR	Termination resistors (40Ω)
SR	Termination resistors (80Ω)
TE	TTL/ECL level converters W/ECL enable
ET	ECL/TTL level converters
OD	TTL output drivers
RS	Termination resistors (400Ω pulldown)
RD	Termination resistors (80Ω TTL)
AD	Two 2-input ECL/MOS level converters
DD	Four single-input TTL/MOS level converters
MA	MOS 256 X 8 memory array
2S	Two 2-input line receivers

SECTION II

INSTALLATION

2.1 GENERAL

Installation information for the CP is provided in the *ASC System Installation* manual, Texas Instruments part number 929980-1.

SECTION III

OPERATING INSTRUCTIONS

3.1 GENERAL

Operating instructions are not included in this publication. Refer to the *ASC Operator's Manual*, Texas Instruments part number 931433-1.

SECTION IV

PRINCIPLES OF OPERATION

4.1 GENERAL

The ASC-4X Central Processor is a layered, four-pipeline processor. As such, the CP contains distinct levels, or stages, in the development of an instruction in the IPU, of operands in the MBU's, and of results in the AU's. Each of these levels can hold and simultaneously operate on a separate instruction or set of operands, unless the level has been reserved by a previous instruction. The IPU contains five levels for instruction development (levels 0-4), each MBU has an input and an output level for operand selection (levels 5 and 6), and the AU's have a minimum of two levels (input and output). The number of effective levels in an AU varies with the operations being performed. Figure 4-1 illustrates the basic components of the Central Processor, their interconnections, and their relation to the levels of each CP pipe. The following theory discussion centers around this block diagram and explains the major functions of each block in the Central Processor. Additional maintenance data is included in the appendices to this manual. Detailed controller flowcharts and discussions follow the block diagram description.

4.2 IPU LEVEL 0

Level 0 of the IPU generates addresses to central memory to request instruction octets (eight word groups), receives the octets from memory, and selects one word instructions from the octets for transfer to the Instruction Register (IR) in level 1. The addressing portion consists of the Look-Ahead (LA) Register, the Present Address (PA) Register, the Output Address (OA) Register and the Branch Address (BA) Register. These registers ensure that the correct address will be in OA to access the next octet of instructions for the IPU. The Memory Interface File (KCM) and the two Current Instruction Files (KA and KB) receive and hold instruction octets from memory so that the selection circuits may access words from the octets. The File and Word select circuits use the address in PA to select an instruction from either KA or KB. While instructions are being drawn from either KA or KB, the other unused file can receive a new octet from memory. This latter file can then supply the next series of instructions without delay to the IPU. The following paragraphs describe the function of each of these level 0 components.

4.2.1 LOOK-AHEAD REGISTER (LA). LA is a 24-bit register that normally holds the address of the octet that is currently being requested from memory. When central memory accepts that request, the output from LA is fed through an adder to increase the address by eight to form the address of next octet in sequence. This new octet address enters the OA register for transfer to central memory, and also the LA register for the next look ahead cycle. At the start of an instruction sequence, the first address to be fetched from memory is in the P3 register (P3 receives this address during initial CP loading, since the addressing registers at level 0 are used to load the CP with the new program). To initiate the new program the address in P3 transfers into OA, LA and PA. The IPU issues a memory request for the octet indicated by the address in OA, and transfers the address in LA through the adder to OA and LA. The address in PA selects an instruction from the octet when it returns from memory. LA continues to supply addresses through the adder to OA until the end of the program sequence if no cycle interruptions occur.

4.2.1.1 Cycle Interruptions. The normal processing cycle for the LA register may be broken by either a branch instruction, a Load Look-Ahead (LLA) instruction, or an instruction hazard at level 3 of the IPU. When a branch instruction reaches level 3 of the IPU and the address of the branch target is not already in the pipe, the address of the new instruction transfers from the AR register in level 3 to LA, OA and PA so that instructions from the branch path may be accessed from memory and loaded into the IPU.

An LLA instruction prepares the IPU for a branch back to a point in the program sequence occupied by the LLA. When the LLA reaches level 3 of the IPU, the address of the LLA in the P3 register is stored into the BA register. When the indicated branch instruction enters the pipe, the address in BA transfers to LA and OA to fetch the octet containing the LLA from memory and continue to access instructions from that instruction path.

If an instruction reaches level 3 of the IPU and a hazard has occurred that makes the instruction invalid, the address of that instruction is transferred from P3 to LA and OA to re-fetch that instruction octet from memory to obtain valid information for that instruction. When memory returns the valid instruction, the look ahead cycle continues in the normal manner.

4.2.1.2 **Output Compare.** The output of the LA Register feeds two compare circuits. One network uses the output to determine if a far range instruction hazard exists in the LA octet. The other network determines if the LA octet contains the object address of a branch or execute instruction or an indirect address. Refer to the discussion of these networks for further explanation of the comparisons.

4.2.2 **LOAD LOOK-AHEAD COUNTER.** The Load Look-Ahead Counter is a 12-bit, decrementing counter used only during a Load Look-Ahead instruction. When the LLA instruction reaches Level 3 of the IPU, the N field of that instruction enters the LLA Counter. The N field specifies the number of instructions to be executed before the required branch occurs. The counter then decrements by one for each instruction that reaches Level 1. When the LLA count minus the number of active IPU levels (at the time of the LLA) is equal to zero, the counter transfers the address in the Branch Address Register to the Look-Ahead Register (LA) request to memory. Refer to the Load Look-Ahead controller discussion for a flowchart and theory of the look-ahead process.

4.2.3 **BRANCH ADDRESS REGISTER (BA).** BA is a 24-bit register that is used only during a Load Look-Ahead operation. When the LLA instruction reaches Level 3 of the IPU, the instruction address at that level transfers from the P3 Register to BA. BA then holds that address until the LLA Counter transfers the address to the LA register.

4.2.4 **PRESENT ADDRESS REGISTER (PA).** The PA Register is a 24-bit address register that holds the address of the next word to be transferred from the instruction file to the Instruction Register (IR). The address in PA increments by one word address each time a word enters the Instruction Register. The three least significant bits of the PA Register select the word from the Instruction File during normal instruction processing. These bits also determine when the last bit in an octet has been accessed and are, therefore, used to gate input to the PA Register and to toggle the File Select network from one instruction file to the other. As a new instruction enters IR, the word address in PA transfers to P1 Register to accompany the instruction through the IPU.

4.2.4.1 **PA Inputs.** The PA Register is normally loaded from the LA Register when the three LSB's of PA are all 1's. However, the output from P3 can enter PA at the start of an instruction sequence if an instruction hazard is detected at Level 3 of the IPU. The output from the AR register can also load PA during a branch instruction, an execute instruction, or for indirect addressing, providing that the object address of the operation is contained in the current octet as determined by the Branch, Execute, Indirect comparison network at Level 3.

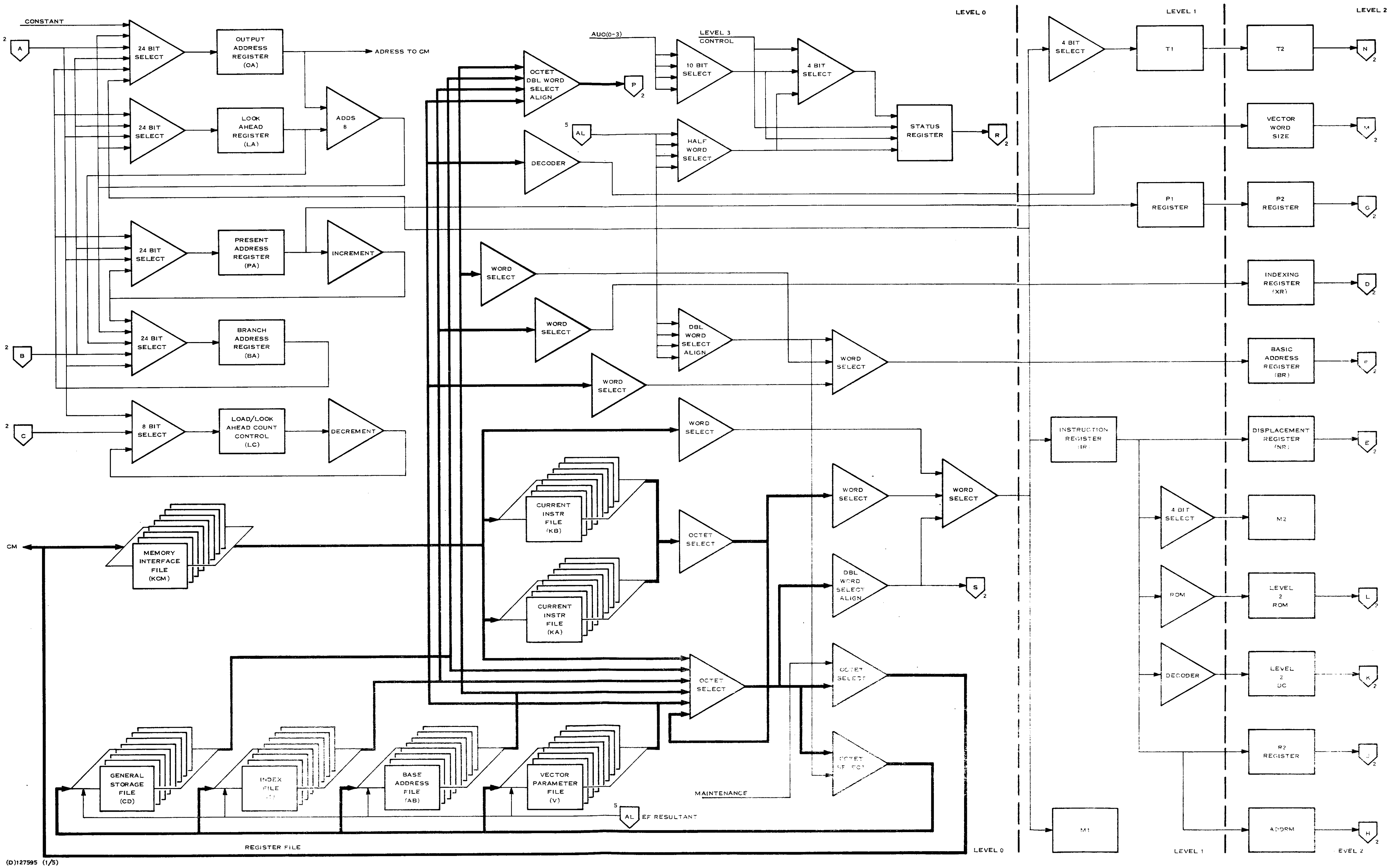


Figure 4-1. Central Processor Block Diagram (Sheet 1 of 5)

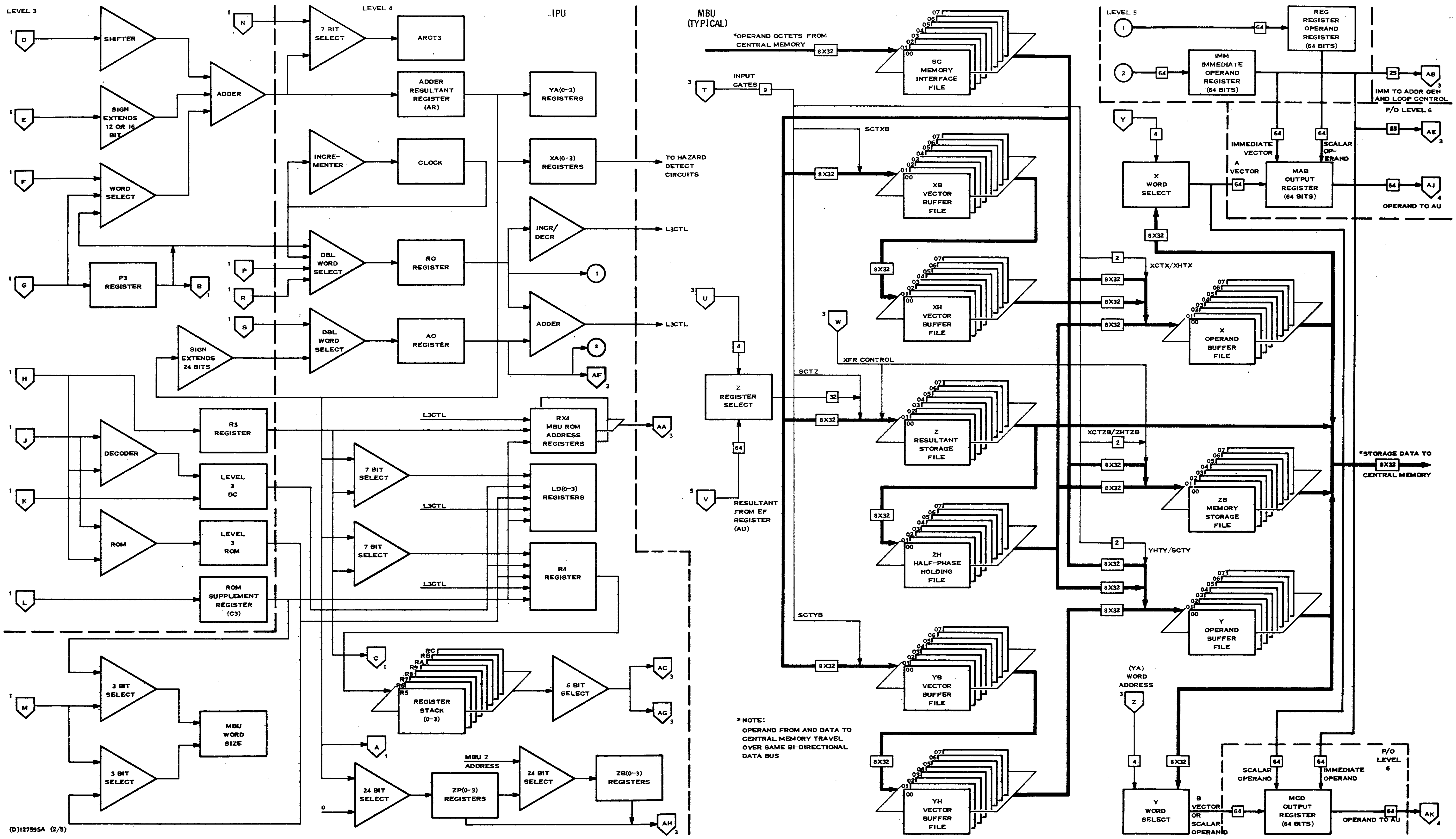


Figure 4-1. Central Processor Block Diagram (Sheet 2 of 5)

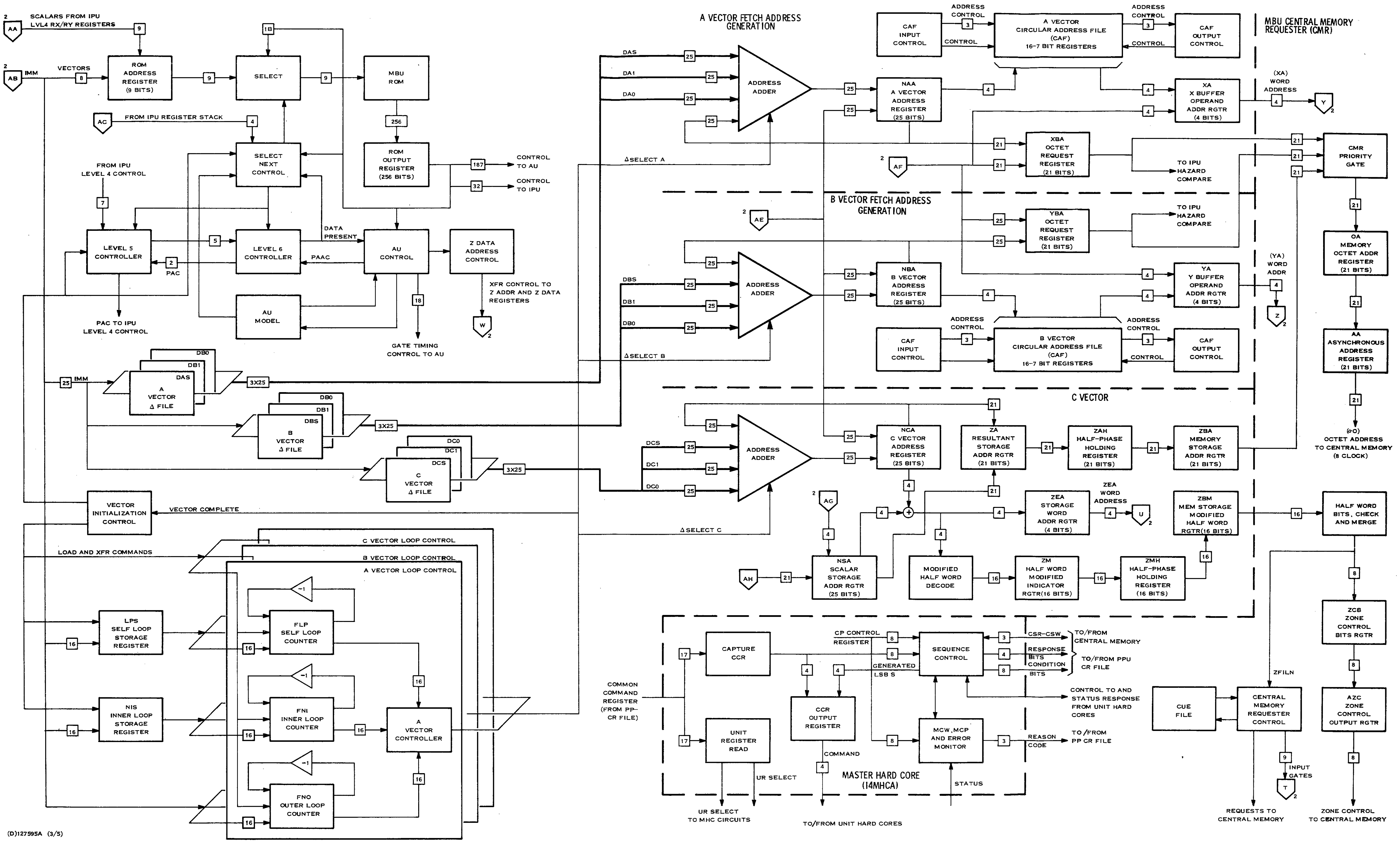

SCALARS FROM IPU LVL4 RX/RY REGISTERS
AA
AB
IMM
VECTORS
ROM ADDRESS REGISTER (9 BITS)
SELECT
MBU ROM
256
ROM OUTPUT REGISTER (256 BITS)
187
CONTROL TO AU
32
CONTROL TO IPU
AC
FROM IPU REGISTER STACK
SELECT NEXT CONTROL
FROM IPU LEVEL 4 CONTROL
LEVEL 5 CONTROLLER
LEVEL 6 CONTROLLER
DATA PRESENT
AU CONTROL
Z DATA ADDRESS CONTROL
PAC
PAAC
AU MODEL
XFR CONTROL TO Z ADDR AND Z DATA REGISTERS
W
PAC TO IPU LEVEL 4 CONTROL
GATE TIMING CONTROL TO AU
A VECTOR Δ FILE
DAS
DB1
DB0
3X25
B VECTOR Δ FILE
DBS
DB1
DB0
3X25
C VECTOR Δ FILE
DCS
DC1
DC0
3X25
VECTOR INITIALIZATION CONTROL
VECTOR COMPLETE
LOAD AND XFR COMMANDS
C VECTOR LOOP CONTROL
B VECTOR LOOP CONTROL
A VECTOR LOOP CONTROL
LPS SELF LOOP STORAGE REGISTER
-1
FLP SELF LOOP COUNTER
NIS INNER LOOP STORAGE REGISTER
FNI INNER LOOP COUNTER
A VECTOR CONTROLLER
FNO OUTER LOOP COUNTER
(D)127595A (3/5)
A VECTOR FETCH ADDRESS GENERATION
DAS
DA1
DA0
ADDRESS ADDER
ΔSELECT A
CAF INPUT CONTROL
ADDRESS CONTROL
CONTROL
A VECTOR CIRCULAR ADDRESS FILE (CAF) 16-7 BIT REGISTERS
CAF OUTPUT CONTROL
NAA A VECTOR ADDRESS REGISTER (25 BITS)
XA X BUFFER OPERAND ADDR RGTR (4 BITS)
XBA OCTET REQUEST REGISTER (21 BITS)
AF
TO IPU HAZARD COMPARE
B VECTOR FETCH ADDRESS GENERATION
AE
YBA OCTET REQUEST REGISTER (21 BITS)
NBA B VECTOR ADDRESS REGISTER (25 BITS)
Y BUFFER OPERAND ADDR RGTR (4 BITS)
B VECTOR CIRCULAR ADDRESS FILE (CAF) 16-7 BIT REGISTERS
ΔSELECT B
C VECTOR
NCA C VECTOR ADDRESS REGISTER (25 BITS)
ZA RESULTANT STORAGE ADDR RGTR (21 BITS)
ZAH HALF-PHASE HOLDING REGISTER (21 BITS)
ZBA MEMORY STORAGE ADDR RGTR (21 BITS)
AG
ZEA STORAGE WORD ADDR RGTR (4 BITS)
ZEA WORD ADDRESS
U
ZBM MEM STORAGE MODIFIED HALF WORD RGTR(16 BITS)
ΔSELECT C
AH
NSA SCALAR STORAGE ADDR RGTR (25 BITS)
MODIFIED HALF WORD DECODE
ZM HALF WORD MODIFIED INDICATOR RGTR(16 BITS)
ZMH HALF-PHASE HOLDING REGISTER (16 BITS)
MBU CENTRAL MEMORY REQUESTER (CMR)
(XA) WORD ADDRESS
Y
CMR PRIORITY GATE
OA MEMORY OCTET ADDR REGISTER (21 BITS)
(YA) WORD ADDR
Z
AA ASYNCHRONOUS ADDRESS REGISTER (21 BITS)
(σO) OCTET ADDRESS TO CENTRAL MEMORY (8 CLOCK)
HALF WORD BITS, CHECK AND MERGE
ZCB ZONE CONTROL BITS RGTR
ZFILN
CUE FILE
CENTRAL MEMORY REQUESTER CONTROL
INPUT GATES
T
AZC ZONE CONTROL OUTPUT RGTR
REQUESTS TO CENTRAL MEMORY
ZONE CONTROL TO CENTRAL MEMORY
COMMON COMMAND REGISTER (FROM PP-CR FILE)
CAPTURE CCR
CP CONTROL REGISTER
GENERATED LSB S
SEQUENCE CONTROL
CSR-CSW
RESPONSE BITS
CONDITION BITS
TO/FROM CENTRAL MEMORY
TO/FROM PPU CR FILE
CONTROL TO AND STATUS RESPONSE FROM UNIT HARD CORES
CCR OUTPUT REGISTER
UNIT REGISTER READ
MCW,MCP AND ERROR MONITOR
REASON CODE
TO/FROM PP CR FILE
COMMAND
UR SELECT
MASTER HARD CORE (14MHCA)
STATUS
UR SELECT TO MHC CIRCUITS
TO/FROM UNIT HARD CORES

Figure 4-1. Central Processor Block Diagram (Sheet 3 of 5)

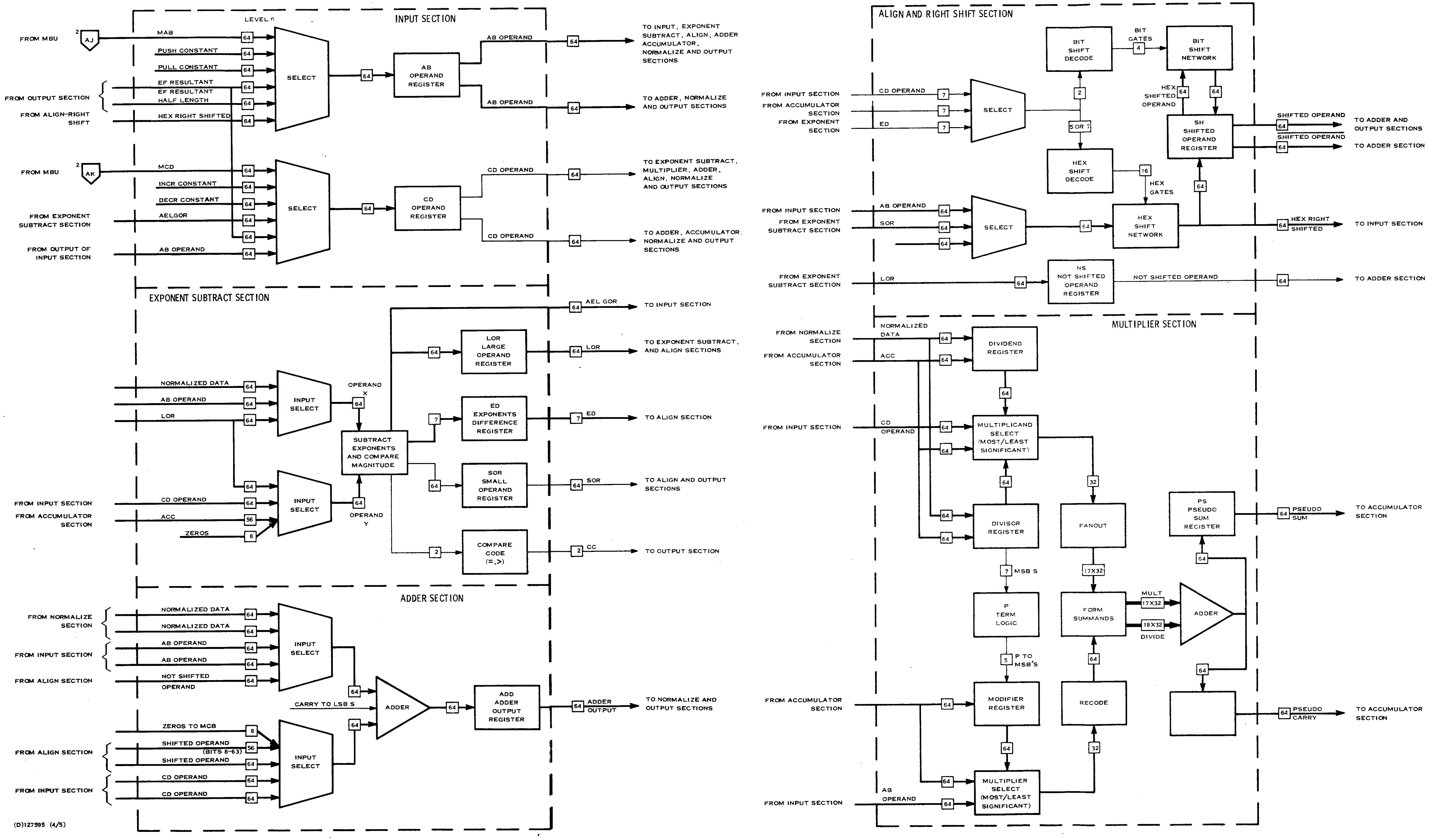


Figure 4-1. Central Processor Block Diagram
(Sheet 4 of 5)

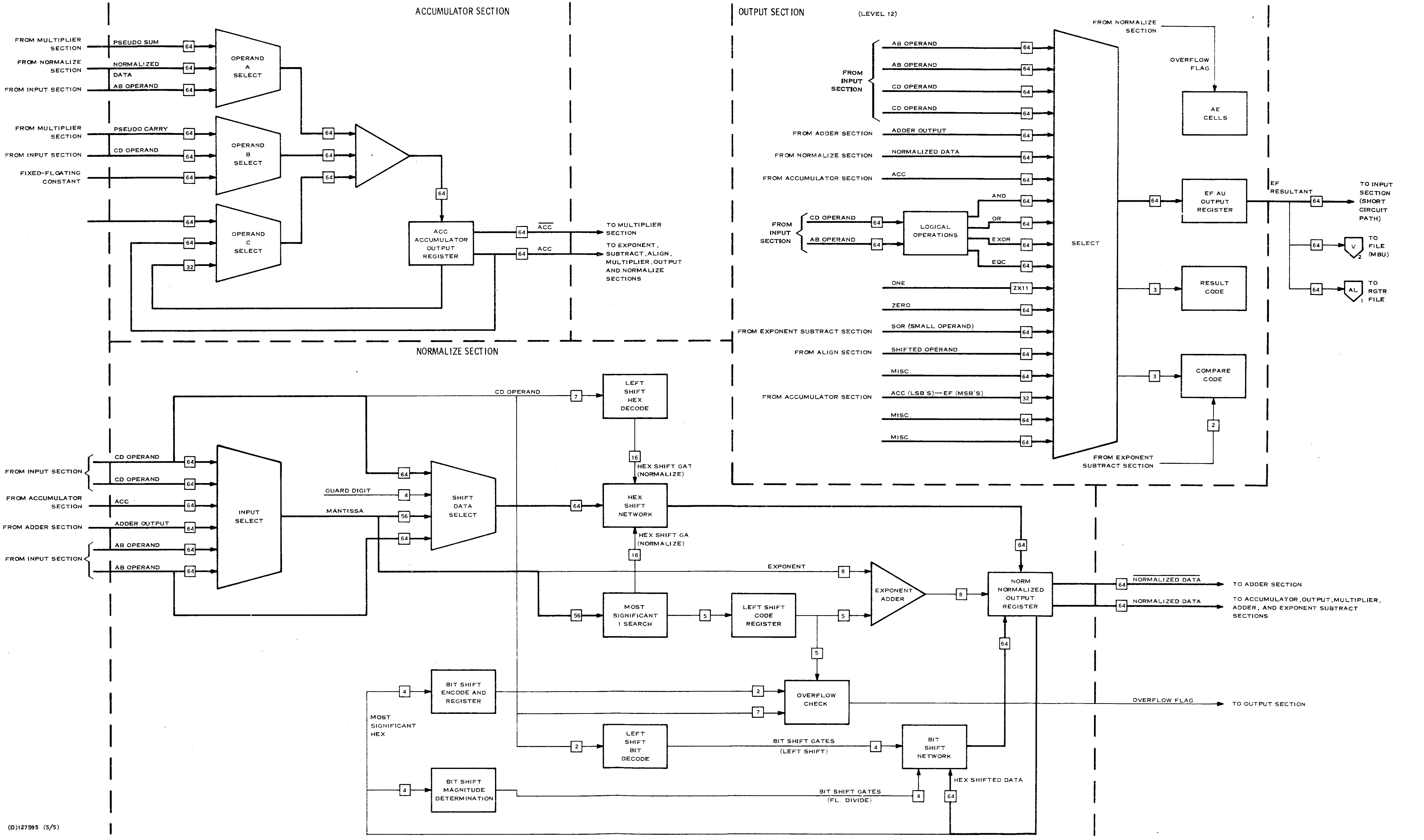


Figure 4-1. Central Processor Block Diagram (Sheet 5 of 5)

4.2.5 OUTPUT ADDRESS REGISTER (OA). The Output Address Register is a 24-bit register that relays 21-bit octet addresses to Central Memory for data transfer to/from KCM. All memory accesses from the I PU must transmit an address to memory through the OA register. Three input paths to the OA register provide addressing capability for all IPU communication to Central Memory.

4.2.5.1 P3 Register Output. During an instruction sequence start-up, or if an instruction hazard is detected at Level 3. The contents of the P3 Register (Level 3 Program Address Register) transfer into the OA Register. P3 holds either the first address of the instruction sequence in the case of a start-up operation, or the address of the instruction that must be re-fetched due to an instruction hazard. In either case, the OA Register transmits that address to Central Memory to begin the instruction sequence.

4.2.5.2 LA Register Output. During normal instruction processing, new instruction addresses enter the OA Register through the octet adder circuit (+8) from the Look-Ahead Register. The LA register provides a continuous source of instruction addresses to be fetched from memory.

4.2.5.3 AR Register Output. For indirect addresses or for branch or execute instructions, the output of the AR Register may transfer to the OA Register if the required address is not already in the pipe. Comparison circuits at Level 3 determine if it is necessary to access memory for the desired word.

4.2.5.4 Load/Store Details. The OA Register transmits sequential addresses to memory to Load or Store the contents of the IPU from or into memory. The details instruction from the peripheral processor loads the OA Register with a pointer address that points to indicate the address of the first octet of the details map in memory. A partial adder then increments the address by one octet (addition of 8) to provide sequential octet addresses to memory.

4.2.6 KCM MEMORY INTERFACE FILE. KCM is an octet register file containing eight 32-bit registers. This file performs a buffer function between ASC Central Memory and the IPU registers and flip-flops. All IPU data transfer operations to and from Central Memory must pass through KCM. KCM holds the data until it can be synchronized with clock pulses for orderly transfer through the IPU, or until memory accepts the data to be stored.

4.2.6.1 Instruction Processing. During instruction processing, the KCM file receives instruction octets from Central Memory and transfers the octets to one of the two current instruction files: KA or KB. The KCM file is transferred to whichever current instruction file is not being accessed by the current instruction address. For indirect addressing, a direct path from KCM to the instruction word select circuit by-passes the current instruction files to avoid alteration of the files. This path allows an instruction from Central Memory to be loaded directly into the Instruction Register.

4.2.6.2 Load/Store Details. Each bit in the KCM file connects directly to numerous bits throughout the IPU for use in a Load or Store Details operation. Each octet of the details map in Central Memory transfers sequentially to the KCM file (Load Details). The position of the octet in the details map determines which of the KCM output paths will be enabled for each bit of the octet until all flip-flops and registers in the IPU reflect the condition specified in the details map. The transfer path is similar, but in the opposite direction for a Store Details operation. Certain Details paths are also used in Load/Store Status or Intermediate commands. The process is the same for these operations as for the Details operation, but limited in scope.

4.2.6.3 **Store File.** The Store File operation passes through KCM for transfer to Central Memory. The output from the Register File fills KCM and the octet transfers to memory. Load File enters data into KCM and then to the Register File.

4.2.7 **KA/KB CURRENT INSTRUCTION FILES.** The Current Instruction Files are two octet files containing eight 32-bit registers each. During normal operation they receive alternate, synchronized octets from KCM that contain instruction words to be accessed by the IPU. The first octet enters the KA instruction file. While addresses are selecting words from the KA file, KCM loads the next octet into the KB file. This alternate loading process allows the IPU to proceed uninterrupted through an instruction sequence without the delay required to access a new octet from Central Memory. This time advantage is lost, however, when a Branch instruction jumps to an instruction that is not resident in either the KA or the KB file.

4.2.8 **FILE SELECT.** The File Select circuit controls the sequencing of the Current Instruction Files and relays the file status to the Level 0 Controller. When the first instruction octet from memory enters KA, the File Select circuit gates the output from the KA registers to the Word Select network. File Select then monitors the three least significant bits (LSB) from the Present Address Register and enables the KB registers to the Word Select network on the clock after the three LSB's of PA are all ones (hexadecimal 7). File selection alternates in a like manner until the instruction set is complete or a Branch instruction alters the order of instruction processing. File Select also notifies the Level 0 Controller when either instruction file is full and which file is selected. This enables the controller to determine if valid data is available for transfer to Level 1.

4.2.9 **WORD SELECT.** The Word Select circuit enables the proper instruction word from either KA or KB to be transferred to the Instruction Register. The Look-Ahead Controller determines when the transfer will take place. Only one octet is active to the input of the word select circuits at any one time. During sequential instruction fetching, the file select circuit supplies one octet to the word select network. When the object address of an indirect address or an Execute instruction is not resident in the KA or KB files, either the KCM octet or an octet from the Register File supplies inputs to the word select network, depending upon the origin of the instruction octet. The select circuit then monitors the three LSB's from either the PA register (sequential instruction acquisition) or the AR register (indirect addressing or Execute instruction). These bits designate a particular word within the active octet.

4.2.10 **LEVEL 0 CONTROLLER.** The Level 0 Controller monitors the status of the instruction files to determine if the valid data is present in Level 0, checks the status of the Level 1 Controller to determine if that level can accept a new instruction, and receives instruction status from Level 3 to determine if an instruction in Level 3 affects the actions required by Level 0. Level 0 Controller then issues a transfer signal to gate the Level 0 instruction and program address into the Level 1 registers. Refer to the Level 0 Controller flowchart and description later in this section for a detailed representation of controller functions.

4.3 IPU LEVEL 1

Level 1 of the IPU is a passive level. It receives an instruction word from the Level 0 selection network and holds it until Level 2 is ready to accept the new instruction. While in Level 1, the instruction is checked for an indirect address or an Execute instruction, either of which disables instruction reception for Level 1 until the object of those functions passes through Level 1. The following paragraphs describe the major components of Level 1.

4.3.1 P1 REGISTER. The P1 Register is a 24-bit register that holds the address of the instruction currently in the Instruction Register of Level 1 of the IPU. The address transfers into P1 from the PA register when the instruction enters the Instruction Register and leaves P1 when the Level 1 Controller gates the instruction to Level 2.

4.3.2 INSTRUCTION REGISTER (IR). The Instruction Register is a 32-bit register that receives an instruction word that has been selected from the instruction file, from Central Memory directly through KCM, or from the output of the Register File. IR holds the instruction until Level 1 Controller transfers it to Level 2. If IR contains an Executive instruction, or one containing an indirect address, Level 1 Controller prevents further instructions from entering the Instruction Register until the object of that instruction is retrieved from memory and passes into the Instruction Register.

4.3.3 LEVEL 1 CONTROLLER. The Level 1 Controller monitors the hazard detection circuit to detect a far range hazard, checks the status of Level 2 Controller to determine if that level can accept a transfer, and samples the instruction in Level 3 to determine its effect on Level 1. The controller then gates the contents of Level 1 into Level 2 and sets the active bit in the Level 2 Controller. Refer to the Level 1 Controller flowchart and description later in this section for a complete representation of the controller's functions.

4.4 REGISTER FILE

The Register File is a storage area in the IPU that is loaded by either a direct memory transfer or from the output of the Arithmetic Unit of the Central Processor. The file consists of forty-eight 32-bit registers grouped into six octets. The octets are designated by the letters A, B, C, D, I and V, and respond to the hexadecimal addresses 01 through 2F if the "M" field of the addressing instruction is equal to zero. The output of the Register File is available to three levels of the IPU: Level 0 for indirect addressing and Execute instructions, Level 2 for base addresses and indexing, and Level 4 for operands and vector parameters except X, Y, and Z addresses. The following paragraphs provide an outline of the contents and function of the octets in the Register File.

4.4.1 BASE ADDRESS FILE, A AND B. Octets A and B of the Register File (addresses 01 through 0F) are used for base addressing. Their output is selected by the 4-bit "M" field in the instruction containing base addressing. Since an "M" field of zero indicates no base addressing is to be done, Register File address 00 is inaccessible by this network. No register resides in location 00 of the Register File.

4.4.2 GENERAL STORAGE FILE, C AND D. Octets C and D of the Register File (addresses 10 through 1F) provide general storage for arithmetic operations or for quick access by instructions. These files can be loaded directly from memory to provide a source of instructions or operands to the IPU.

4.4.3 INDEX FILE, I. Octet I of the Register File (addresses 20 through 27) holds the index registers for indexing an address of an instruction. The T field of that instruction selects the proper register from the Index File to be used in the indexing process. Since a T field of zero indicates that no indexing will be performed, address 20 of the I File is inaccessible to the indexing network. Address 20 provides an additional general storage register.

4.4.4 VECTOR PARAMETER FILE, V. Octet V of the Register File (addresses 28 through 2F) supplies eight words that define the parameters used in a vector operation. A vector instruction results in reading the entire contents of the file. The words of the file are assigned as shown in figure 4-2. Table 4-1 defines the word fields. Words 29, 2A, 2B of the file, the starting addresses of the vectors, enter Level 2 of the IPU pipe for possible address modification. The remaining five words enter directly into Level 4 for transfer to the MBU.

REGISTER	H_0	H_1	H_2	H_3	H_4	H_5	H_6	H_7
28	OPR		ALCT	SV	L			
29	–	XA	SAA					
2A	HS	XB	SAB					
2B	VI	XC	SAC					
2C	DAI				DBI			
2D	DCI				NI			
2E	DAO				DBO			
2F	DCO				NO			

114314

Figure 4-2. Vector Parameter File Format

4.5 IPU LEVEL 2

Level 2 of the IPU is a selection and holding level in preparation for address modification. If index or base plus displacement addressing is indicated by the incoming instruction, this level channels the proper index and base address values into their respective holding registers, modifies the register outputs as required by the operation to be performed, and places the resulting 24-bit words for input to the Level 3 Modification Adder. The following paragraphs describe functions of the major components of Level 2 of the IPU.

4.5.1 LEVEL 2 CONTROLLER. The Level 2 Controller monitors the hazard detection circuit to detect a far range hazard, checks the status of Level 3 Controller to determine if that level can accept a transfer, and samples the instruction in Level 3 to determine its effect on Level 2. The controller then gates the contents of Level 2 into Level 3 after any necessary address modification has been performed and sets the active bit in the Level 3 Controller. Other control functions performed by this circuit are determined by the specific operation being processed. Refer to the Level 2 Controller flowcharts and description later in this section for a complete representation of controller functions.

4.5.2 LEVEL 2 ROM. The Level 2 ROM receives the 8-bit operation code of the instruction word as it enters Level 2. Depending upon the Op Code, the ROM generates 32 control bits that are used in Level 2 to control instruction processing or that transfer to the C3 ROM Supplement Register in Level 3.

4.5.3 R2 REGISTER. The R2 Register is a 4-bit register that receives the R field bits of the incoming instruction word and transfers them to Level 3 when the instruction enters Level 3. The output of this register is also used in Level 4 hazard detection logic to find a register hazard.

Table 4-1. V-File Field Descriptions

Reg	Hex Character	Field	Description
28	H_0,H_1	OPR	Operation code
28	H_2	ALCT	Arith. & Log. Comparison Term
28	H_3	SV	Single-valued vector
28	H_4-H_7	L	Vector dimension
29	H_1	XA	Initial index A
2A	H_1	XB	Initial index B
2B	H_1	XC	Initial index C
29	H_2-H_7	SAA	Starting address A
2A	H_2-H_7	SAB	Starting address B
2B	H_2-H_7	SAC	Starting address C
29	H_0-H_7	(29)	Immediate operand A
2A	H_0-H_7	(2A)	Immediate operand B
2A	H_0	HS	Halfword starting address
2B	H_0	VI	Vector increment direction
2C	H_0-H_3	DAI	$\pm\Delta A_i$, inner loop
2C	H_4-H_7	DBI	$\pm\Delta B_i$, inner loop
2D	H_0-H_3	DCI	$\pm\Delta C_i$, inner loop
2D	H_4-H_7	NI	Inner loop count
2E	H_0-H_3	DAO	$\pm\Delta A_O$, outer loop
2E	H_4-H_7	DBO	$\pm\Delta B_O$, outer loop
2F	H_0-H_3	DCO	$\pm\Delta C_O$, outer loop
2F	H_4-H_7	NO	Outer loop count

4.5.4 INDEXING REGISTER (XR). The Indexing Register is a 32-bit register that receives input from one of the seven index registers in the Register File. If the instruction entering Level 2 from Level 1 indicates that indexing will be required, the 4-bit T field of that instruction selects one register in the I File for transfer to the Indexing Register. The output from XR enters a shift network. Control bits from the Level 2 ROM indicate whether the index word will be left-shifted one bit (doubleword addresses), right-shifted one bit (halfword addresses), or remain unaltered (single word addresses). The output from the shift network enters the modification adder.

4.5.5 DISPLACEMENT REGISTER (NR). The Displacement Register is a 32-bit register that receives the instruction word from the Level 1 Instruction Register. Only part of the instruction is used by the displacement circuit, however. The instruction word from the Displacement Register enters a sign extension circuit. Control bits from the Level 2 ROM then determine one of two possible places for sign extension to occur. The LSB's of the resulting 24-bit word that enters the modification adder contain either the N field (bits 20 to 31) of the instruction word, or both the M and the N field (bits 16 to 31) of the instruction word. The remaining bits to the left of these fields are the result of sign extension.

4.5.6 P2 REGISTER. The P2 Register is a 24-bit register that holds the address of the instruction that currently resides in IPU Level 2. The address enters the P2 Register when the Level 1 Controller transfers the instruction into Level 2 and leaves the P2 Register when the Level 2 Controller transfers the instruction into Level 3. The output of the P2 Register may also transfer to the AR Register in Level 3 through the address modification network. The two comparison networks, hazard and branch, examine the contents of P2.

4.5.7 BASE ADDRESS REGISTER (BR). The Base Address Register is a 32-bit register that receives the base address word from file A or B of the Register File. If the instruction word that enters Level 2 contains an M field that is not zero, the M field bits select the output from one of the Register File registers and transfer that 32-bit word to the Base Address Register. When selected for base addressing, the Base Address Register inputs to the address modification adder. The Base Address Register is also used to transfer the first three words of the Vector Parameter File through the address modification network, and into the MBU. A select network at the output of this register allows control signals from the Level 2 ROM to select either the Base Address Register output or the output from the P2 register as the base address used in the modification addition.

4.6 IPU LEVEL 3

IPU Level 3 develops the effective address of the operand to be sent to the MBU. It receives input from the Level 2 instruction registers, adds the applicable base, displacement, and/or index, and holds the resultant address for use in Level 4. Level 3 also checks for hazards and reprocesses an instruction if a hazard exists concerning that instruction. The following paragraphs describe the function of the major blocks in Level 3.

4.6.1 MODIFICATION ADDER. The Modification Adder is a 32-bit (24 effective bits) parallel adder circuit with a double-level look-ahead, carry determination circuit. The adder receives inputs from the Base Address, Displacement, and Indexing Registers and adds them to form one 24-bit resultant that transfers to the Adder Resultant (AR) register when the Level 2 Controller enables the transfer. A feedback path from the AR register to the adder allows for incrementing the AR register to provide continuous octet addresses to Central Memory for Load File Multiple or Store File Multiple instructions.

4.6.2 ADDER RESULTANT (AR) REGISTER. The AR Register is a 32-bit (24 effective bits) register that receives the modified operand address from the Modification Adder. The output from this register may load the Level 0 addressing registers during a branch operation, indirect addressing, or an execute instruction. A feedback path to the Modification Adder provides for incrementing the address in the AR register for loading or storing multiple Register Files. If the address that enters the AR Register is an effective address of an operand (address) or an immediate operand, the contents of AR transfer to the AO Register in Level 4 under control of the Level 3 Controller. The address from AR also enters the Z model Stack (Store operation) or the Register Stack and is available to the hazard detection circuits in Level 4.

4.6.3 P3 REGISTER. The P3 Register is a 24-bit register that contains the address of the instruction that is currently in Level 3 of the IPU. It receives the address from the P2 Register when the instruction enters the AR Register after undergoing any indicated modifications. The output of this register can be used to load the BA Register in Level 0, or can be transferred to the RO Register in Level 4 as a direct operand. In all cases, the output from this register is available to the hazard detection circuits in Level 4.

4.6.4 LEVEL 3 ROM. The Level 3 ROM receives the Operation Code portion of the instruction word from Level 2 as that instruction word enters Level 3 through the Modification Adder. The 8-bit Op Code produces a 32-bit output from the Level 3 ROM. This output, in conjunction with the C3 Register outout, provides control bits for coordination of Level 3 processes and supplies bits to complete the address stored in the Register Stack in Level 4. If the Op Code indicates a branch, indirect or execute instruction, the Level 3 ROM triggers a comparison circuit for those operations.

4.6.5 ROM SUPPLEMENT REGISTER (C3) The C3 Register is a 24-bit register that stores control bits from the Level 2 ROM to be used as control bits in supplement to those produced by the Level 3 ROM. The control bits enter the C3 Register when the Level 2 Controller transfers the particular instruction into Level 3. The output is immediately available to the Level 3 circuits for gating and control purposes. C3 output bits also transfer to the Register Stack in Level 4 to complete the address stored in that stack.

4.6.6 R3 REGISTER. The R3 Register is a 4-bit register that receives the R field bits of the incoming instruction word and transfers them to the Register Stack in Level 4 when the operand or operand address from the AR Register transfers to Level 4. The output of this register also selects a word from the Register File to enter into the RO Register in Level 4 as one of the operands needed by the selected MBU for transmission to its AU.

4.6.7 LEVEL 3 CONTROLLER. The Level 3 Controller monitors the hazard detection circuits to determine if a hazard exists for the instruction that is now in Level 3. If the hazard bit sets, the operand or address in the AR Register may not be valid. This condition causes the instruction to be re-addressed by transferring the contents of the P3 Register to the Level O Addressing Registers to begin processing that instruction again. The instructions currently in Levels 1 and 2 will also be re-addressed by the addressing registers following fetching of the Level 3 instruction from its memory location.

In addition, the Level 3 Controller monitors the status of the Level 4 Controller to determine if Level 4 can accept a transfer and then gates the contents of Level 3 to Level 4. Refer to the Level 3 Controller flowcharts and description later in this section for a detailed representation of the controller's functions.

4.6.8 BRANCH, INDIRECT, EXECUTE COMPARISONS. Whenever a Branch or Execute instruction or an address requiring indirect processing reaches Level 3 of the IPU, the IPU must examine the addresses of the instructions currently in the registers to determine if a new memory fetch will be necessary to obtain the desired word. The Branch, Indirect, Execute Comparisons circuit performs this function in the following sequence (refer to figure 4-3):

- Compare AR with P2 (24 bits). If AR = P2, transfer Level 2 to Level 3.
- Compare AR with P1 (24 bits). If AR = P1, sequence Level 1 to Level 3.
- Compare AR octet with PA octet (21 bits). If equal, force AR to PA and LA to access new word (Branch), or use AR to select from Current Instruction File (Indirect or Execute).
- Compare AR octet with LA octet (21 bits). If equal, force AR to PA and LA to begin new sequence (Branch), or use AR to select from waiting Current Instruction File (Indirect or Execute).
- If all comparisons fail, transfer AR to OA, use LA or PA to access new octet (Branch), or transfer AR to OA and use AR to select word from KCM (Indirect or Execute).

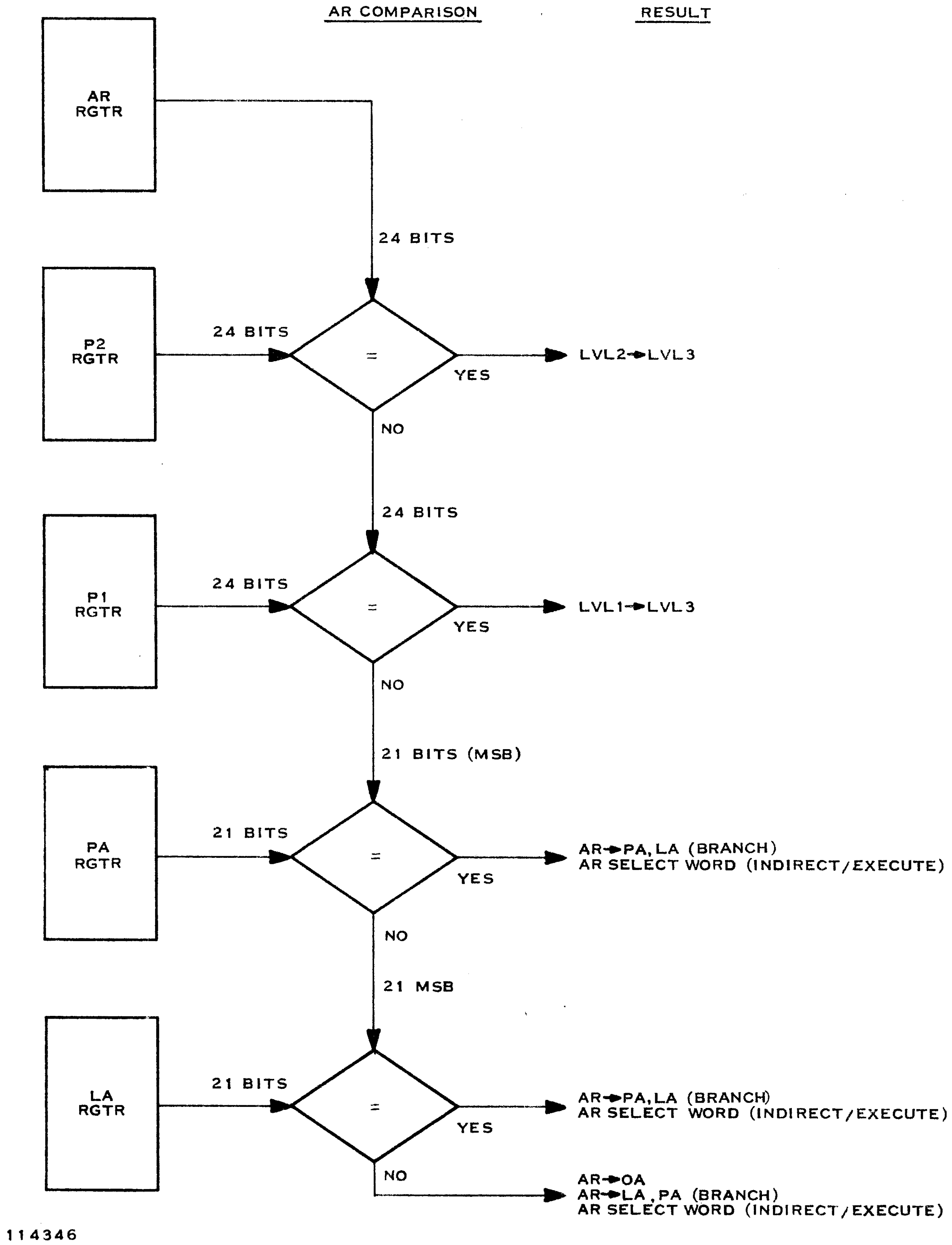


Figure 4-3. Branch, Indirect, Execute Comparisons

4.7 IPU LEVEL 4

IPU Level 4 is the IPU output level to the MBU. It includes an address and an operand output register, word selection logic, and a controller. Also included within the Level 4 circuits, but not solely operational within Level 4, are the hazard detection circuits. These circuits protect the IPU from processing potentially faulty instructins or operands. The following paragraphs briefly describe the major components included in Level 4 of the IPU.

4.7.1 LEVEL 4 CONTROLLER. The Level 4 Controller monitors the Level 5 status of the MBU's to determine if any MBU is ready to accept new data from the IPU. If an MBU can accept a transfer and the active bit in Level 4 Controller is set, the Level 4 Controller enables the output from the RO and AO Registers, along with control signals, to the MBU. Refer to the Level 4 Controller flowcharts and description later in this section for a complete representation of the Level 4 Controller functions.

4.7.2 REGISTER STACK. The Register Stack stores the resultant storage addresses of the operands in each level of the CP from IPU Level 4 through AU Level 12 (nine levels maximum). The stack is four-wide from Level 5 through Level 12 to allow storage of operand addresses for four pipes. The stack is used for any instruction that passes through an AU and has a storage destination in the Register File. The Register Stack registers contain the storage address of the result as well as control bits. The destination address is normally developed from the output of the R3 Register and specific control bits from the Level 3 ROM and ROM Supplement Register (C3). However, during a Store (R) into α when α is less than or equal to 2F (in the Register File), the output from the AR Register in Level 3 supplies the destination address to the Register Stack. The output from the Register Stack is used for hazard detection. The Register Hazard Comparison circuit compares the contents of the Register Stack with various addresses in the IPU to determine whether an instruction will draw from a location that is to be modified by the operands preceding it. Refer to the Register Hazard Comparison description for a more detailed discussion of the comparison circuitry.

4.7.3 REGISTER HAZARD COMPARISON. A register hazard exists when an instruction in the IPU accesses a register in the Register File and that register will be modified by an operation being processed elsewhere in the CP. The contents of that register will not be valid data until the modification operation is complete and the result has been stored in the Register File. The Register Hazard Comparison circuit prevents access to a register in the Register File until any instruction that modifies that register has cleared the CP. The comparison circuit performs this safeguard functions through the series of register comparisons illustrated in figure 4-4.

As an instruction enters Level 1 of the IPU, the compare circuit monitors both the T field (index register select) and the M field (base register select) and compares these fields with the contents of the Register Stack registers to determine if the T or M field registers will be modified by an instruction in Levels 4 through 12 of the IPU. Registers R2 and R3 are also compared with the two fields to determine if the instruction in Level 2 or 3 will modify the Register File register. The AR Register in Level 3 is also compared with the T and M fields of Level 1 to detect a hazard during a Load Register File operation, where the AR Register holds the address of the register in the Register File to be loaded. If the instruction passes these tests, it moves to Level 2. If not, the instruction must wait until the hazard condition drops before it can transfer to Level 2 to select the registers from the Register File.

At Level 2 the comparison circuit checks only the AR Register in Level 3 for a hazard against the T and M fields of the instruction at Level 2. This comparison checks an address that was not generated when the instruction was in Level 1. If the instruction passes this test, it may move to Level 3.

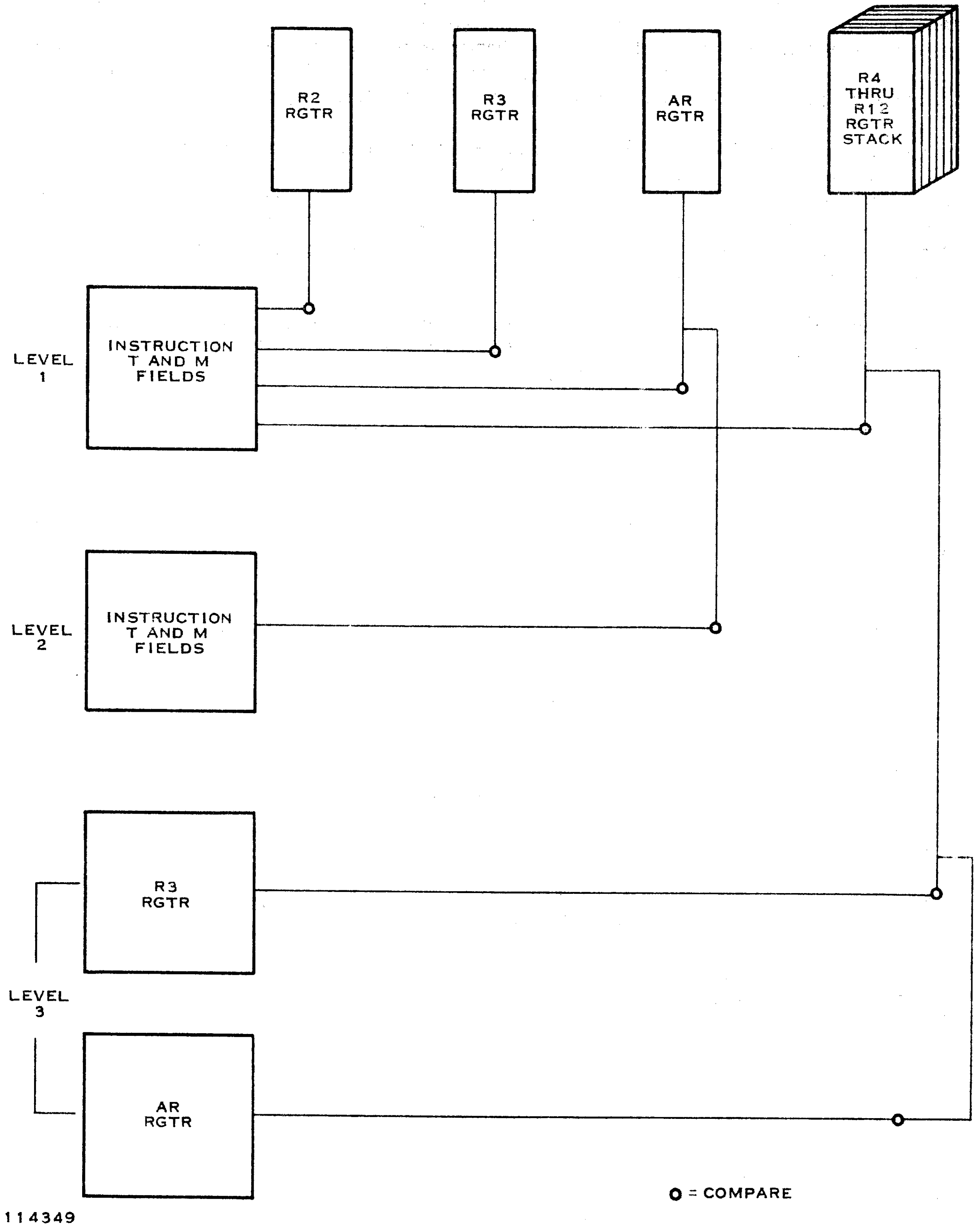


Figure 4-4. Register Hazard Comparisons

Since the output of the R3 Register may select a Register File register for input to the RO Register in Level 4, and the AR Register may select a Register File register during indirect addressing, these two registers are checked for a hazard conflict at Level 3 before being allowed to access the Register File. The addresses specified by these two registers are compared with the addresses stored in the Register Stack to detect a hazard condition.

4.7.4 **AO REGISTER.** The AO Register is a 64-bit register that receives the output from the AR Register in Level 3. This output may be either the memory address of an operand for use by an MBU or a direct operand (either immediate or from Register File) for transfer to an AU. Before entering the AO Register, the AR Register output (24 bits) undergoes a sign extension process to create a 64-bit input to the AO Register. The AO Register transfers its 64-bit word to an MBU when directed by the Level 4 Controller.

4.7.5 **Z MODEL STACK.** The Z Model is a 5-register stack that contains the destination address of all Store operations to Central Memory in the CP. The address input is from the AR Register in Level 3 and enters the Z Model only if the Op Code of the instruction specifies a Store operation. The address then moves through the stack registers as the operand moves through the CP. The registers in the Z Model correspond to levels as follows:

- *ZP Register.* Contains the destination address of a Store operation that is currently in pipe levels 4 through 12.

- *ZA Register.* Contains the destination address of a Store operation that is in the MBU Z Register, having been processed by the CP.

- *ZB Register.* Contains the destination address of a Store operation that has transferred from the Z File to the AB File in the MBU, and is no longer available for X and Y update.

- *ZO Register.* Contains the destination address of a Store operation that is being sent to Central Memory.

- *MA Register.* Contains the destination address of a Store operation that is in the Memory Control Unit, but has not been written into its addressed location of the Memory Module.

The output from the Z Model is used to determine if a requested operand from memory is to be changed by a Store operation currently in a pipe (operand hazard). The Instruction and Operand Hazard Comparison circuits determine if any hazards exist with respect to the contents of the Z Model.

4.7.6 **α OPERAND HAZARD COMPARISON.** An Operand hazard exists when the operand addressed by the AR Register is about to be altered by a Store instruction that is farther along in the CP. The hazard indicates that if the operand is acquired at the present moment, before the Store instruction is complete, a faulty operand may be obtained from memory. To avoid accessing a faulty operand, the Operand Hazard Comparison monitors the Z Model and compares its contents with the address indicated by the AR Register. This comparison is illustrated in figure 4-5.

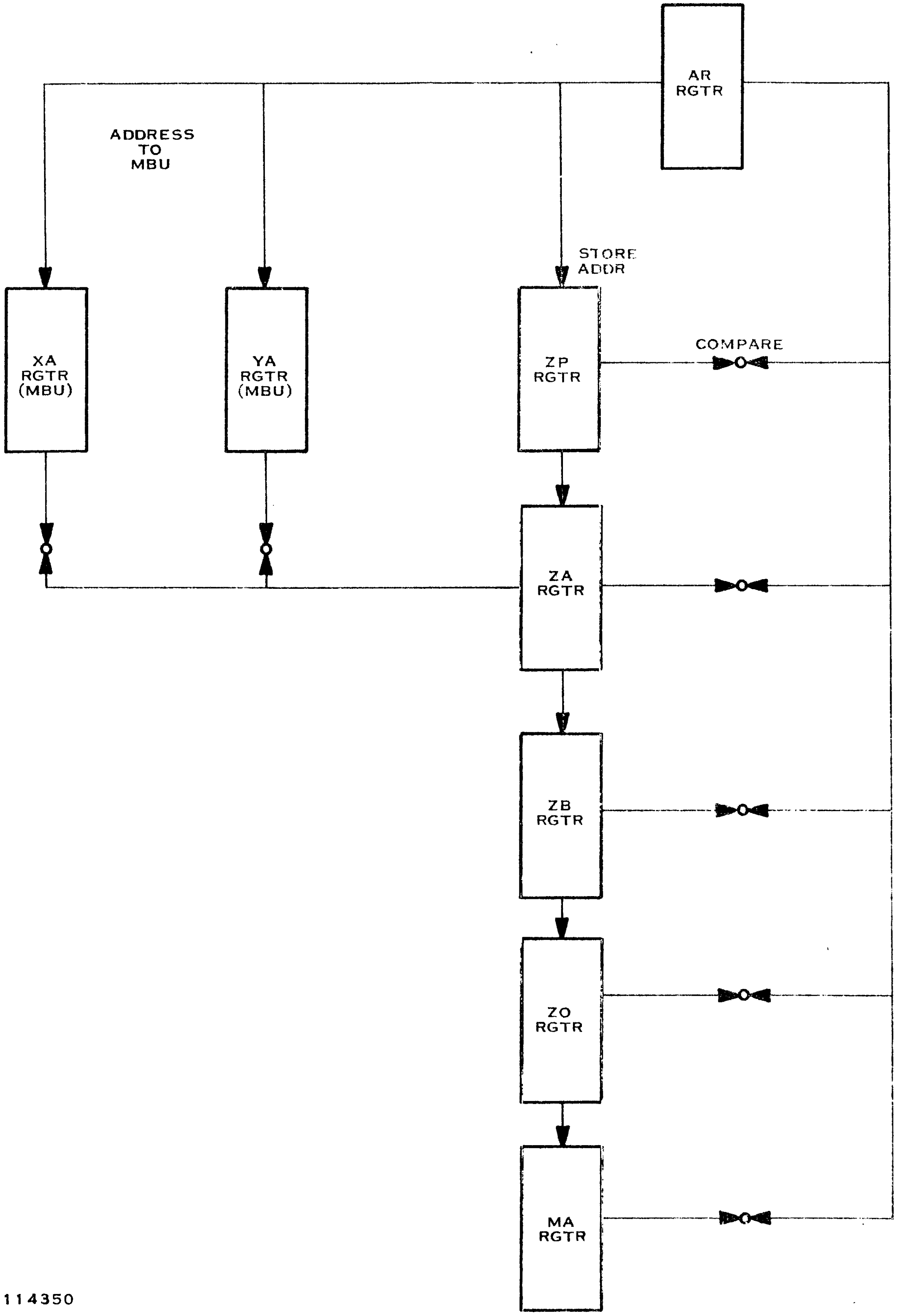
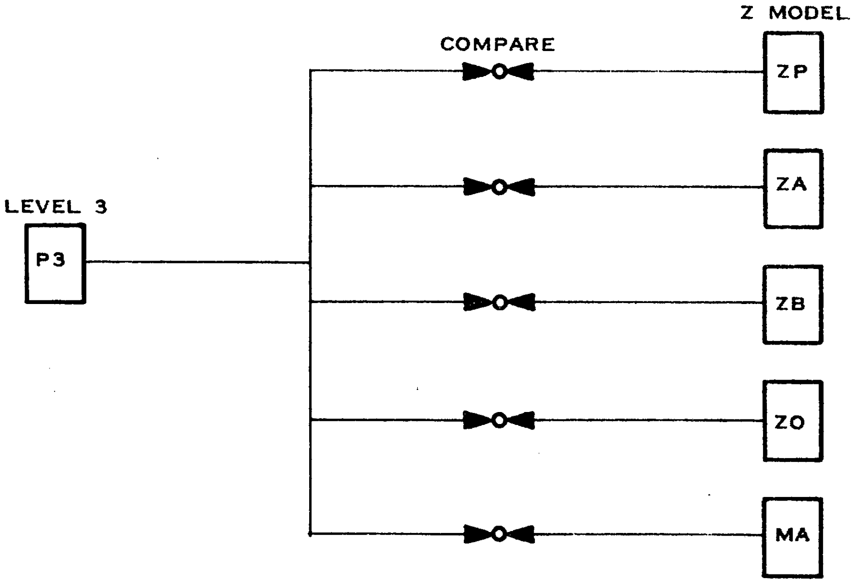


Figure 4-5. Operand Hazard Comparisons

An additional comparison is performed by this circuit to indicate whether a Z to X or Y update is necessary. The address in the ZA Register is compared with the address of the octets in the X and Y Buffers of the MBU. If the address in the ZA Register is within the octet in either the Y or the X Buffer, the IPU may choose to update the information in the buffers with the resultant data found in the MBU Z Register.

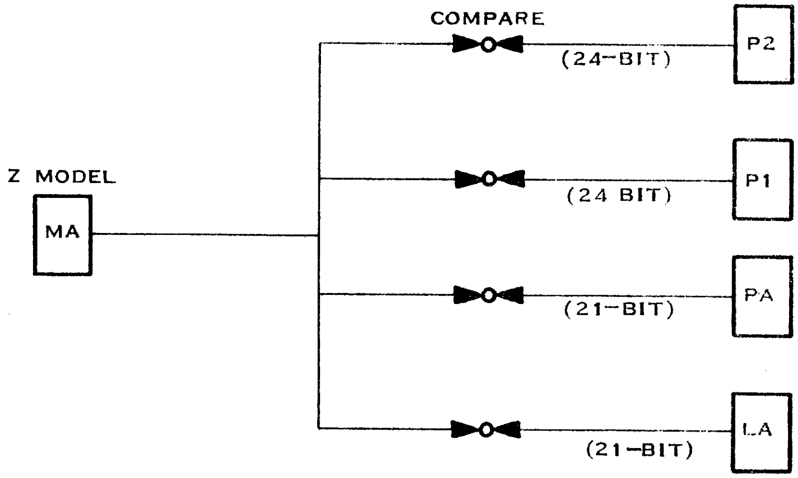
4.7.7 NEAR RANGE INSTRUCTION HAZARD COMPARISON. An Instruction hazard exists when an instruction that has been accessed by the IPU is to be altered by a Store instruction that is already in a pipe. The Near Range Instruction Hazard Comparison circuit detects an imminent instruction hazard by comparing the store operation address record in the Z Model with the address of the instruction that is about to be executed in Level 3 of the IPU. This address is contained in the P3 Register. If a near range hazard is detected by this comparison, the hazard flag is set. When the offending store instruction is finished, P3 transfers its contents to LA, OA, and PA to begin another pass at the instruction in memory. The near range comparison is illustrated in figure 4-6.

4.7.8 FAR RANGE INSTRUCTION HAZARD COMPARISON. A far range instruction hazard indicates that an instruction in the IPU before Level 3 has been fetched from a memory location that is being changed by a previous store instruction that is writing into memory. This condition means that the instruction in the pipe is not valid. To detect a far range hazard, the comparison circuit monitors the address in the MA Register of the Z Model and compares that address with the addresses of the instructions in the P1 ad P2 Registers and the octet address contained in PA and the LA Registers (refer to figure 4-7). Detection of a far range instruction hazard has no immediate effect on the IPU, as the invalid instruction may be disregarded by a branch, skip, or



114351

Figure 4-6. Near Range Instruction Hazard Comparisons



114352

Figure 4-7. Far Range Instruction Hazard Comparisons

other diversion before it reaches Level 3. Instead of an immediate reaction, a far range hazard flag sets in the controller corresponding to the invalid instruction. This flag passes from controller to controller as the instruction moves through the IPU levels. When the instruction reaches Level 3, the Level 3 Controller checks the far range hazard flag. If that flag is set, the controller loads the P3 Register into PA, LA, and OA Registers to restart the instruction sequence.

4.7.9 **RO REGISTER.** The RO Register is a 64-bit register that holds an operand for transmission to an MBU. The 4-bit R field from the R3 Register selects one word from the Register File for entry into the RO Register when the Level 3 Controller transfers information from Level 3 into Level 4. The RO Register may also be loaded from the P3 program address register for operations using a direct operand contained in the address registers. The output from the RO Register transfers to the selected MBU for input to the AU after the MBU fetches the second operand from memory.

4.8 MASTER HARD CORE (I4MHCA)

Master Hard Core (MHC) is the communications port and control channel between the Master Controller (MC) in the Peripheral Processor (PP) and the Central Processor (CP). MHC contains connections to the Unit Hard Cores (UHC) for each CP unit, the IPU, the MBU(s) and the AU(s). These connections carry maintenance commands and switch commands to the UHCs and status, response and data feedback lines from the UHCs. The CP Block Diagram (figure 4-1, sheet 3) illustrates the major circuits involved in the Master Hard Core. The following paragraphs describe the functions of each of these components. Refer to Appendix D for further detail regarding MHC.

4.8.1 CAPTURE COMMON COMMAND REGISTER (CAPTURE CCR). Capture CCR monitors the Common Command Register in the PP Communication Register File (CR File) in an asynchronous mode. The CCR (figure 4-8) is a 16-bit portion of the CR File Register C_{16}. It contains a 4-bit unit identifier, a 4-bit Op Code, and 8 bits of address that further define the function to be performed. In order for this register to be active, the Transfer Bit (bit 16 of CR File Register C) must be set.

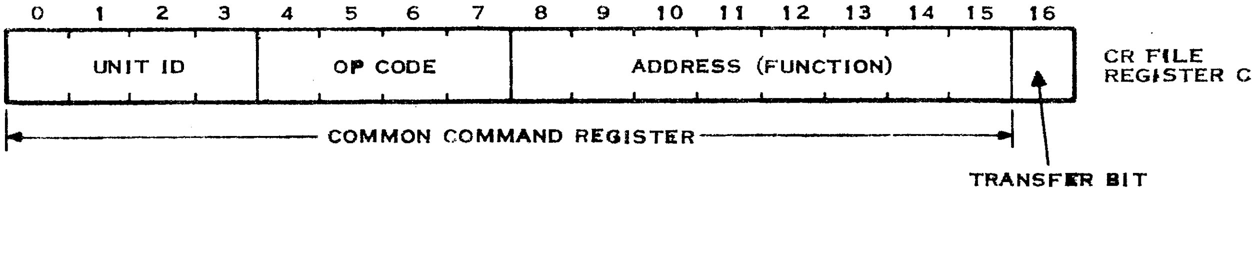
Capture CCR first monitors the Transfer Bit for an active PP command. When the Transfer Bit sets, Capture CCR examines CCR bits 0-7 to determine if the active command is intended for the CP. If these bits represent a hexadecimal 41, Capture CCR transfers the five least significant bits of CCR to a holding register and clears the Transfer Bit in the PP. The function code transfers to Sequence Control, Sequence Control decides whether to gate the four least significant bits to the CCR Output Register, or if it must generate a new code to the UHC's.

4.8.2 MCW, MCP AND ERROR MONITOR. This circuit receives error indications from all UHC's of the CP, examines the IPU for a Monitor Call and Wait (MCW) or a Monitor Call and Proceed (MCP) instruction, and checks the 8-bit CP Control Register in the CR File. From these inputs, the circuit determines when switching of CP contents should be performed. If the switch is not possible, it generates a three bit reason code to the PP. Sequence Control enables the reason code to the PP, or performs the actual switching operation when the Monitor indicates that the switch is necessary.

4.8.2.1 CP Control Register. The CP Control Register consists of eight flag bits residing in bits 24-31 of the CR File Register A (see figure 4-9). Table 4-2 lists each of these bits and their definitions. These bits are used by the PP to condition the responses of the CP Master Hard Core to status conditions.

4.8.2.2 CP Switches. Besides the switch initiated by an MCW instruction, the Monitor circuit checks the CP UHCs for the following error conditions and performs a switch when it detects an error if permitted by the PP:

- CM Protect Violation (PV)
- CM Parity Error (PE)
- Illegal Opcode Detection (IL)
- Arithmetic Exception (AE).



(A)115836

Figure 4-8. Common Command Register and Transfer Bit

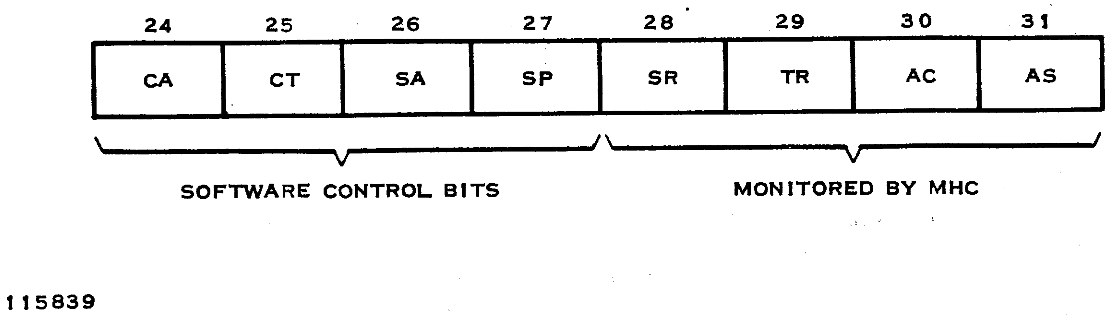


Figure 4-9. CP Control Register

4.8.2.3 **MHC Response.** When the MHC detects an error or call, it responds by setting bits in the CP Response Register (see figure 4-10), and initiates an operation in the Sequence Control circuit if the operation is allowed by the MC. If the context switch cannot be performed, it sets a reason code in the CP Response Register to tell the PP why the switch is not performed. In this case, only the reason code bits of the response byte are modified and the CP run bit is reset, which halts normal CP operation. Table 4-3 defines the bits of the CP Response Byte, including a decoding of the Reason Code bits. Refer to the controller descriptions later in this section for a flowchart of the PP status checking cycle of the response bits.

4.8.3 **SEQUENCE CONTROL.** Sequence Control monitors error and call status from the Monitor circuit, status and response bits from CP UHCs and four control bits from the CR File CP Control Register. It also receives the 5-bit function code from the Capture CCR circuit. From these inputs, Sequence Control determines the required steps to carry out the function prescribed by the PP. In performing this function, Sequence Control generates condition and response bits to inform the PP of current conditions. Four response bits transfer to the CR File CP Response Register. These bits are described under the Monitor circuit description. Eight condition bits, generated by Sequence Control transfer to the CP Condition Register in the CR File, Register 12_{16}. Table 4-4 lists and defines these condition bits; figure 4-11 illustrates the CP Condition Byte.

In addition to status reports, Sequence Control generates control signals to the CP Unit Hard Cores to coordinate performance of the PP Functions. If the hard core function originates from a CP error, Sequence Control produces a 4-bit code that transfers to the CCR Output Register in lieu of a PP produced command.

4.8.4 **CCR OUTPUT REGISTER.** CCR Output is a 4-bit register that holds the operation code required by the CP Unit Hard Cores to perform any maintenance or switch function. The input may be directly from the Capture CCR circuit. However, when an error produces a switch condition, Sequence Control generates the 4-bit input to the CCR Output Register. Input from Capture CCR is a transfer of the four least significant bits of the CCR command. In addition, special fanouts of this register are sent to the AU UHCs for "LOAD/STORE CP Details". Since the X7 and X4 AU UHCs are identical, and CCR commands 410B-D are not the same for X1 and X4 systems, the MHC in the X4 system records 410B to 410E, and reads 410C to 410F. In the case of MC issued CCR exchange commands (410A, 410D), the MHC issues the appropriate "LOAD - followed by Store" CCR commands.

Table 4-2. CP Control Byte Bit Definitions

Bit	Name	Function
CA	CP Available	Set by Master Controller when no CP step is primed. Indicates the need to poll for activity on the CP execution queue. Reset by Master Controller when a CP step is primed.
CT	CP Test	Set by Master Controller to indicate CP control is being relinquished to MCD. When set, the Master Controller will not respond to any other activity in the CP Response or Condition bytes.
SA	Step Active	Set by Master Controller when a CP step is initiated. Reset when a step terminates and no no step is primed for execution.
SP	Step Primed	Set by Master Controller to switch a CP step into execution. Reset by MC when CP begins executing step and no other step is ready for priming.
SR	System Reset	Set by Master Controller to initiate a CP reset. Must be reset before any other CP action can be taken.
TR	Terminate Request	Set by the Master Controller to terminate outstanding CCR or Automatic Switches in the CP. Reset by the MC.
AC	Allow Call	Set by the Master Controller to permit Automatic MCP and MCW calls. Reset by MC to inhibit these calls. Should be reset anytime a CCR command is used that invalidates the next job step status defined by pointers 16, 17, and 28.
AS	Allow Switch	Set by the Master Controller to permit automatic MCW and Error context switching. Reset to inhibit these switches. In the case of MCW's, no action can be taken if either AC=0 or AS=0.

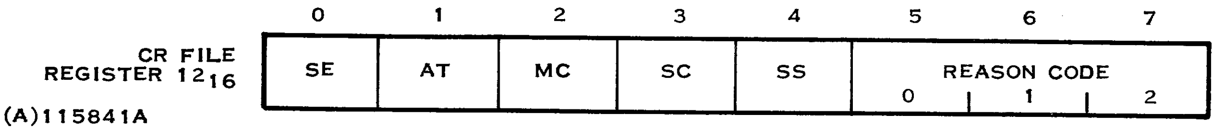


Figure 4-10. CP Response Byte

Table 4-3. CP Response Byte Bit Definitions

Bit	Name	Function
SE	System Error	Set by CP to indicate a Parity Error during normal CP operation. Reset by Master Controller.
AT	Attention	Set by the CP to indicate an abnormal termination (as defined by the Condition Byte) of an automatic call or switch (as defined by the MC and and SC bits). Reset by the Master Controller.
MC	Message Complete	Set by the CP to indicate the completion of an MCP or MCW. Reset by the Master Controller after operation on the message.
SC	Switch Complete	Set by the CP to indicate the completion of an MCW or Error switch. Reset by the Master Controller after priming the next switch.
SS	Stored Status	Set by MHC to indicate a CP Status word has been stored on an automatic switch or CCR command.
RZ(0-2)	Reason Codes	Set by the CP to inform the Master Controller of the following context switch conditions:

CODE	INTERPRETATION
000	NOOP
001	MCP inhibited by MC or SC bits being set.
010	MCW inhibited by MC or SC bits being set.
011	Error switch inhibited by MC or SC bits being set.
100	MCW inhibited by AC = 0.
101	MCP inhibited by AC = 0.
110	MCW inhibited as AS = 0.
111	Error switch inhibited by AS = 0.

The Master Controller resets these bits and sets the CP Run Bit via a CCR command after preparing for the indicated condition.

Table 4-4. CP Conditions Byte Bit Definitions

Bit	Name	Function
CC	Command Complete	Set by MHC to indicate the last CCR command Servicing has completed. Gated to Master Controller with I4QGCC. Reset in MC by MC.
AB	Abnormal Terminate	Set by MHC to indicate the last CCR command Servicing terminated abnormally because of at least one Unit Hard Core abnormally terminating, or because of TR being set by the MC. Gated to MC with I4QGCC.
ME	Memory Error	Set by MHC to indicate the cause of the abnormal termination was a protection violation or parity error encountered by at least one of the Unit Hard Cores. Gated to MC with I4QGCB.
PE	Parity Error	Set by MHC to indicate that a parity error caused or tried to cause an error context switch. Gated to MC with I4QGCB.
IL	Illegal Opcode	Set by MHC to indicate that an illegal opcode was encountered which caused or tried to cause an error context switch. Gated to MC with I4QGCB.
AE	Arithmetic Exception	Set by MHC to indicate an unmasked arithmetic exception caused or tried to cause an error context switch. Gated to MC with I4QGCB.
PV	Protection Violation	Set by MHC to indicate a C.M. Request resulted in a protection violation which caused or tried to cause an error context switch. Gated to MC with I4QGCB.
RB	CP Run bit	Continuously gated to MC to reflect the state of the CP: If RB=1, CP is running; If RB=0, CP is halted (frozen).

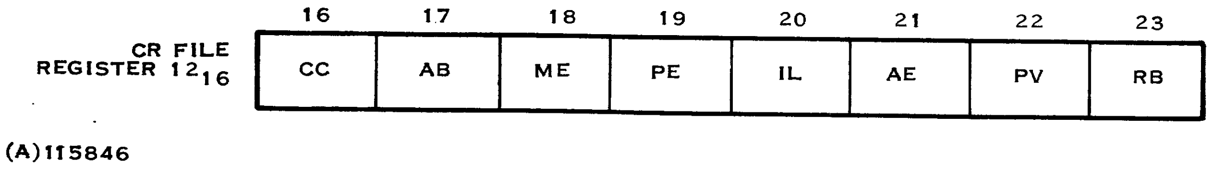


Figure 4-11. CP Condition Byte

4.8.5 UNIT REGISTER READ. Unit Register Read performs the maintenance function of addressing a hard core (unit or master) register or series of flip-flops, so that the contents of the selected register are transferred to the CR File for inspection by the PP. All bits selected in this manner transfer to the CP Unit Register in the CR File, Register A (bits 8-15). The selected byte transfers directly from its respective Unit Hard Core to the CP Unit Register in the CR file.

4.9 MEMORY BUFFER UNITS (MBUS)

The Memory Buffer Units (MBUs) receives addresses or immediate operands from the IPU. If the word is an address, the MBU requests the octet containing that address from central memory, and extracts the proper operand from that octet. In either case the MBU forwards the operand, immediate or addressed, to its AU for processing. The two operands within the MBU can be up to 64 bits long. During vector operations, each MBU buffers up to three octets from memory for each of two input buffers so that a steady input of data to its AU is ensured. The components of the MBU are illustrated in the detailed block diagram of the central processor in figure 4-1. The following paragraphs describe the function of each of those components within the MBU.

4.9.1 MEMORY INTERFACE FILE (SC). The MBU Memory Interface File (SC) receives all operands from Central Memory that the MBU transfers to the AU. The file is an eight-register (octet) group with 32 bits to each register. It receives data directly from the memory data lines and holds that data until it is synchronized with the CP clock pulses. The clock pulses then transfer the data through the remainder of the MBU, subject to gating signals from the MBU controllers. The output from the SC File may enter one of many places in the MBU. During scalar operations and during vector operations when the Vector Buffer Files are empty, the SC output transfers directly into either the X or Y operand buffer. When a vector operation is in progress and the X or Y operand buffers are full, the SC output enters either the YB or the XB Vector Buffer File. Two paths supply data to the Z Storage Files also. One path provides fill-in for partially filled words for storage into memory (ZB), while the other path is used exclusively during a Load Details operation (Z).

4.9.2 VECTOR BUFFER FILES (XB, XH, YB, YH). Each Vector Buffer File consists of eight 32-bit operand registers. During vector operations, the files provide continuous operands to the X and Y Operand Files for operand streaming into the AU. Two Vector Buffer Files supply two stages of octet buffering for each of the two Operand Files. Cue and control bits from the Central Memory Requester control entry into the Vector Buffer Files. The individual vector controllers (figure 4-1) gate the data between the files. When an octet arrives in the SC File, CMR determines which vector stream addressed that particular octet and gates the octet into either the X or Y data stream. The octet may enter either the B level or directly into the operand file, depending on the status of the operand file. If the octet enters the B level and the next buffer is clear (XH or YH) on the next clock pulse, a gate from the vector controller transfer the new octet into the H buffer. When the corresponding Operand Buffer File empties, another gate and a clock pulse transfer the octet from the H level to the Operand File.

4.9.3 OPERAND BUFFER FILES (X, Y). Both Operand Buffer Files consist of eight 32-bit registers. These files supply operands to the MBU output registers during both scalar and vector operations. The files receive their input octet from three sources: the SC Interface File, the XH or YH Vector Buffer Files during vector operations, and the ZH Holding File. This last source of operands is used when the Z pipe contains modified entries for storage in the octet that is resident in either of the Operand Buffer Files. Flag bits record the halfwords that have changed in the octet so that only the changed portions of the octet transfer to the Operand Buffer Files during this update procedure. The output from these files is available to the MAB and MCD output registers through a selection network for input to the AU.

4.9.4 X AND Y WORD SELECT. The word select circuits receive inputs from their respective Operand Buffer File and use a 4-bit word address (figure 4-1), Address Generation Circuit to select a half, single or doubleword entry from the operand octet. During vector operations, the output from the X select circuit is sent only to the MAB register and the output from the Y select circuits drives only the MCD register. No crossover of operands is possible. In scalar mode, both select circuits supply operands exclusively to the MCD register.

4.9.5 MAB/MCD OUTPUT REGISTERS. The MAB and MCD registers are two 64-bit registers that supply operands to the AU for processing. All operands, whether scalar or vector must pass through these registers for transmission to the AU.

4.9.5.1 Scalar Data Paths. During scalar processing, the MAB Register receives operands exclusively from the REG Register. This register supplies operands from the IPU Register File if a register operands is required. The MAB register may not be used during a scalar operation if no register operands are needed. The MCD Register, however, has three sources of operands during a scalar operation. It may receive operands from Central Memory through either the X or the Y Operand Buffers, or an immediate operand from the IMM Register.

4.9.5.2 Vector Data Paths. In vector mode each output register has one main operand source: the X Operand File for the MAB Register and the Y Operand File for the MCD Register. Either output register, however, may receive an immediate vector through the IMM Register.

4.9.6 REG REGISTER. The REG Register is a 64-bit register that receives operands directly from the IPU during scalar mode operation. REG then holds the operand until a corresponding operand from Central Memory, or an immediate operand is available, and the Level 5 Controller enables a transfer from Level 5 to Level 6 of the CP pipe. The contents of the REG register then transfer to the MAB Register at Level 6 of the MBU.

4.9.7 IMM REGISTER. The IMM Register is a 64-bit register that receives immediate operands from the IPU and transfers them to the Level 6 output registers at the direction of the Level 5 Controller. The IMM Register output may transfer only to the MCD Register during a scalar operation. However, during a vector operation, the IMM Register loads the vector parameters into the MBU vector processor circuits, and may also be used to load an immediate, one-value vector into either the MAB or the MCD output register.

4.9.8 Z REGISTER SELECT. The Z Register Select circuit receives the resultant data from the AU and routes it to a particular register in the Z Resultant Storage File. A 4-bit element address from the Z address generation circuit (figure 4-1) designates the particular register for storing the resultant data. This circuit and the Z File pipe are used for scalar store operations and all vector operations only. All other scalar operations store results in the IPU Register File.

4.9.9 Z RESULTANT STORAGE FILE. The Z File consists of eight 32-bit registers that receive input data from the AU for storage in Central Memory. The Z File represents one contiguous octet of Central Memory, the actual address of which is controlled by the Z Address Generation circuit (figure 4-1). The entire octet, therefore, may not be filled in any one operation if storage addresses do not indicate continuous memory locations. Whenever the storage addresses begin storing into a new octet, the Z Address Generation circuit transfers the contents of the Z File to the ZB File for storage and begins storing a new octet in the Z File. This transfer takes place regardless of the full status of the Z File. The input line to the Z File from the SC File is used during a Load Details operation only.

4.9.10 ZH HALF PHASE HOLDING FILE. The ZH File is an eight-register file used exclusively for a transfer delay between the Z File and the ZB File to avoid a premature transfer of contents from Z to ZB. Two clock pulses control the timing of transfers between the files, Phase 0 and Phase 1 clocks. Phase 0 enables input to the Z and the ZB files (along with other gating pulses from the control logic). The Phase 1 clock occurs at equal intervals to the Phase 0 clock, but the pulses are 180 degrees out of phase with the Phase 0 clock. Phase 1, therefore, represents a "half-phase" pulse with respect to the Phase 0 clock. This half-phase clock transfers the contents of the Z File into the ZH File, exclusive of all other control signals. ZH, therefore, always reflects the contents of the Z File after a half-clock delay. Because of this delay, ZH provides a stabilized output to the ZB File. At the end of an octet in the Z File, Phase 0 clock can simultaneously transfer the octet in the Z File, as reflected in ZH into the ZB File, and begin storing data for a new octet in the Z File. One half-clock later, Phase 1 clock changes the ZH File to reflect the new octet in the Z File.

If the octet in ZH is to be stored and the X or Y Operand File is currently using that octet, the changed words in the ZH octet transfer to the X or Y File to update that information before the store gate transfers the octet to the ZB File. This update path is also available if the operand address indicates an access to a word that is in the Z File. CMR will transfer the changed words from ZH to the requesting file, X or Y, to update the new octet as it enters X or Y from SC.

4.9.11 ZB MEMORY STORAGE FILE. The ZB File consists of eight 32-bit registers. The output from these registers supplies data to Central Memory for storage in octet transfers. Central Memory cannot store halfwords without destroying the second half of the word stored in that memory location. The ZB File receives either full or partially full octets from the Z File through the ZH File when address control begins storing AU results into a new octet. If the transfer contains a modified halfword that does not have a corresponding halfword to form a single word store, the Central Memory Requester circuit addresses memory to read the contents of the octet from its memory location into the SC Memory Interface File. Cue bits in CMR transfer the unmodified halfwords in the octet from the SC File to the ZB File to complete the storage octet. CMR then transfers the changed words into memory to replace the information stored at that location. If the AB File receives only single words from the Z File, CMR stores the words of that octet without the fill-in process.

4.9.12 ROM ADDRESS REGISTER. The ROM Address Register is a 9-bit register that holds control bits from the IPU to designate the starting address of the next ROM sequence for AU gating control. During scalar operations, the nine bits have the following sources:

- Bit 0 - Designates one of two halves of the ROM. May be set to allow a new sequence without modifying the remaining address bits.
- Bit 1-4 - The four most significant bits of the current instruction Op Code.
- Bits 5-8 - A recording of the four least significant bits of the current instruction Op Code.

The ROM Address Register receives this input during the Level 4 to Level 5 address transfer and holds the address bits until the Select Next Controller gates the output into the ROM circuits.

During vector operations, the ROM Address Register is loaded from the Vector Parameter File in the IPU. At the start of a vector operation, the operation code in the first word of the Vector Parameter File enters the ROM Address Register from the IMM Register. Since the operation code is only eight bits, the ninth bit is held at a constant "1" level.

4.9.13 ROM ADDRESS SELECT. Control bits from the Select Next Controller monitor the progress of the ROM sequence and designate to the Select circuit which address source to gate into the ROM circuit. The address may be derived either from the ROM Address Register or from the output of the ROM itself. The first source is used to initiate a ROM sequence; the second source continues the ROM sequence by supplying succeeding addresses from the ROM output to access the next ROM word. The ROM supplies two 9-bit addresses during sequencing (β_1 and β_2). The Select circuit may choose either of these addresses, depending upon conditions monitored by the Select Next Controller.

4.9.14 MBU ROM. The output from the MBU ROM controls the gating of operations through the AU pipe. The ROM supplies a 256-bit output from a 9-bit address input. In addition to AU Control bits, the ROM produces two 9-bit addresses that feed back to the Select circuit to address the next output from the ROM during sequences of more than one ROM instruction. An output map of the function codes is provided on site in computer printout form.

4.9.15 ROM OUTPUT REGISTER. The ROM Output Register is a 256-bit register that receives the code from the ROM and holds it for relay to the AU gating circuits. The contents of this register change each clock time to follow the output of the MBU ROM.

4.9.16 SELECT NEXT CONTROLLER. Select Next monitors the status of the MBU ROM sequence and determines when a new sequence may be started for the next instruction to enter the AU pipe. In streams of similar operations, a new operand may be entered into the AU before the previous operand has completed processing. The precise point of entry is determined by the nature of the operation. When a new operation can begin without disturbing the one in process, Select Next enters the new Op Code into the ROM. The ROM then produces gate signals for both instructions concurrently. Refer to the flowchart and description of the Select Next Controller that appears later in this section.

4.9.17 AU CONTROL. The AU control circuit produces timing signals to the AU to coordinate the gating signals produced by the MBU ROM. It receives control signals from the ROM, status from the Level 6 Controller, and a reflection of the contents of the AU from the AU Model. AU control then generates the proper enable pulses to the AU to route data to the proper pipe segment at the proper time in the ROM sequence. AU control also supplies the Level 6 Controller with a Path Ahead Clear (PAC) indication so that the controller can gate data from Level 6 into the AU. Refer to the AU Control flowchart and description that appears later in this section.

4.9.18 AU MODEL. The AU Model consists of a series of AU status flip-flops that mirror the current condition of each level of the AU. Timing and control signals from the AU Controller set and clear the respective level busy bits in the model to assure a current status picture in the model at all times. The output from the model supplies status signals to the AU Controller and to the Select Next Controller to aid in coordination of AU operations and instructions.

4.9.19 Z DATA AND ADDRESS CONTROL. This circuit controls the transfer of data between the Z, ZH, and ZB Files and the corresponding transfer of addresses from the NSA to the ZBA Registers through the ZA Register. To determine the timing of these transfers, the circuit monitors the status of data in the AU pipe to discover when a new octet of addresses will be started in the Z File. This circuit also controls the introduction of a new storage address into the NSA Register by monitoring status commands from the IPU and gating the new address from the Z Stack in the IPU when the operand and instruction reaches Level 7 of the CP. To provide immediate destination selection for resultant AU data, the addresses in the address registers precede their respective data by one clock period as they pass through the Z files. Flowcharts for the address and data flow through the Z file appear later in this section along with description of the major decision paths.

4.9.20 LEVEL 5 CONTROLLER. The Level 5 Controller receives status and transfer bits from the Level 4 Controller in the IPU, gates the IPU addresses into the Level 5 Registers, returns status bits to Level 4 and provides Level 6 with information for transfer coordination into Level 6. The Select Next Controller also provides a gating input to the Level 5 Controller to enable the controller to transfer new data into the next level of the pipe. A flowchart and description of the scalar input cycle appears later in this section. Vector inputs are under control of the separate vector controllers.

4.9.21 LEVEL 6 CONTROLLER. The Level 6 Controller receives transfer control signals from the Level 5 Controller, path status signals from the AU Controller, and a transfer enable signal from the Select Next Controller. By combining these signals, the Level 6 Controller determines when to transfer operands and ROM addresses such that no data or ROM control bits are lost. The Level 6 Controller also returns data present indications to the AU Controller to indicate an active level state. The logic flow of the Level 6 Controller in the scalar mode and vector mode is discussed later in this section.

4.9.22 INNER LOOP STORAGE REGISTER (NIS). NIS is a 16-bit storage register that receives the inner loop count portion of the Vector Parameter File at the beginning of a vector operation. NIS then holds this count for restoration to the inner loop counters when a new inner loop is begun. The count held in NIS is used by the inner loop counters for all three vectors.

4.9.23 SELF LOOP COUNT REGISTER (LPS). LPS is a 16-bit storage register that receives the Length portion of the Vector Parameter File at the beginning of a vector operation. LPS then holds this count for restoration to the self-loop counters when a new self-loop is begun. The count held in LPS is used by the self-loop counters for all three vectors.

4.9.24 VECTOR INITIALIZATION CONTROL. Vector Initialization Control performs the gating functions required to begin vector processing in the MBU/AU and also clears out the units at the completion of a vector to prepare the pipe for the next operation. At the start of a vector operation, this circuit distributes the sections of the Vector Parameter File from the IMM Register to their proper destinations within the vector control and generation circuits, starts the address generators, performs the first operand fetch of the vector, and aligns the first operands for processing. At that point vector control and generation circuits take control of operation and run until completion.

When the vector is complete, the Initialization Control circuit again assumes control to store the last results of the vector into Central Memory. Initialization Control also cycles the MBU ROM to a NOP (no operation) condition and sequences the AU to clear the gate and control flip-flops. This clean-up operation prevents alteration of the next operation that will be processed by the pipe. Refer to the flowchart and discussion of the Vector Initialization Controller that appears later in this section.

4.9.25 VECTOR LOOP CONTROL. Vector Loop Control consists of a 16-bit decrementing counter for each of the three address loops plus a vector controller. The controller monitors counter status and the status of the address generation network corresponding to that vector. It then controls counter decrementing and input gating to the address generator. Each of the three possible vectors, A, B, and C, have a separate and independent loop control circuit.

4.9.25.1 Self Loop Counter (FLP). FLP receives the 16-bit length field of the Vector Parameter File from LPS at the beginning of a vector operation. Control pulses from the vector controller then decrement the count in the counter each time an address is generated by the address generation circuits. If the FLP received a count of zero (NOP), or all loop counters receive a count of one (unit vector) to start the vector, the vector requires no address generation. When the counter reaches a count of one, it signals the vector controller that the FLP is "1". If additional self loops are to be executed, the vector controller loads the FLP with the original self-loop count from the LPS holding register.

4.9.25.2 Inner Loop Counter (FNI). FNI receives the 16-bit inner loop count field of the Vector Parameter File from the NIS register at the beginning of a vector operation. If the inner loop count is equal to zero or one, the inner loop will not be executed: the self loop will be executed once. If the inner count is greater than one, control pulses from the vector controller decrement the count in the counter each time a new self-loop of addresses begins. When the count in the counter reaches a one, the counter signals the vector controller that the FNI is equal to "1". If additional inner loops are to be executed, the vector controller loads the FNI with the original inner loop count from the NIS holding register.

4.9.25.3 Outer Loop Counter (FNO). FNO receives the 16-bit outer loop count field of the Vector Parameter File directly from the IMM Register at the beginning of a vector operation. If the outer loop count is equal to one or zero, the outer loop will not be executed: only the self-loop and inner loops will run to completion. If the outer loop count is greater than one, control pulses from the vector controller decrement the count in the counter each time a new inner loop of addresses begins. When the count in the counter reaches a one, the counter signals the vector controller that FNO is equal to "1". Another cycle of inner and self-loops is performed to complete the vector. A new outer loop count entered into the FNO counter indicates the beginning of a new vector.

4.9.25.4 Vector Controller. The Vector Controller supplies increment pulses to the loop counters and gating signals to the address generation circuits to select increment values between addresses. At the beginning of a vector operation, the controller checks the value in the FLP counter. If this value is a zero (NOP) or a one (unit vector), the incoming vector requires no address generation. If the self-loop count is greater than one, the controller enables the address generation adder to increment the address. Each new address produced by the generation circuit causes the controller to decrement the FLP counter. When FLP reaches a "1" count, the controller disables the increment input to the address adder and inspects the FNI counter.

If the inner loop count in the FNI counter is equal to zero or one, vector processing is finished. If the inner loop count is greater than one, the controller enables the inner loop displacement input to the address adder to produce the starting address of the next self-loop and decrements FNI. The controller then loads the self-loop count into FLP and enables the address adder to increment through the self-loop again. This process continues until both FLP and FNI contain a count of one. At that point the controller disables the address adder and inspects the FNO counter.

If the outer loop count in FNO is equal to zero or one, the vector operation is complete. If the outer loop count is greater than one, the controller enables the address adder to add the outer loop displacement to the last address and decrements the outer loop counter. Both the inner loop count and the self-loop count restore to FNI and FLP, respectively, and the self- and inner loop sequence repeats. When the outer loop counter reaches a count of one, a final repetition of the self- and inner loop sequence completes the vector.

When the controller determines that the vector is finished, it disables the address generation circuit and informs the Vector Initialization Controller of the status. Vector Initialization Control then stores the remaining results in memory and clears the pipe for the next instruction.

4.9.26 MBU UNIT HARD CORE. MBU Unit Hard Core is the maintenance function and context switching controller for the MBU. It receives maintenance and switching commands from the PP through Master Hard Core. It then performs the specific operation independent from the other two unit hard cores in the CP. When the operation is complete, MBU UHC reports the completed status to MHC. Among the functions performed by Unit Hard Core and Load and Store Details or Intermediate operations, and Unit Register Read when the register to be transferred to the PP is in the MBU Unit Hard Core. Refer to Appendix D for flowcharts and an explanation of hard core operations.

4.9.27 VECTOR ADDRESS GENERATION (A/B VECTORS). The address generation circuits for the A and B vectors are functionally identical. Each circuit produces addresses to access vector components from memory for their respective vectors. At the start of a vector operation, Vector Initialization Control loads the vector starting address field into the Vector Address Register (NAA/NBA). The octet address of this word transfers to the Octet Request Register (XBA/YBA). If it is a new octet, the vector controller generates a request to the Central Memory Requester (CMR). The 4-bit word address within an octet is stored into the vector Circular Address File (CAF). The word addresses are then retrieved sequentially (first in - first out) to select from their corresponding octet for input to the MAB/MCD Registers. As each word address enters CAF, the entire address feeds back through the address adder to create the next address in the vector. The following paragraphs describe each major component of the address generation circuit. Refer to the Address Generation Controllers discussion later in this section for description and flowcharts of the process.

4.9.27.1 Vector Address Register (NAA/NBA). This register receives the 25-bit vector starting address field from the Vector Parameter File at the start of a vector operation. This field enters from the IMM Register under control of the Vector Initialization Controller. The Vector Address Register then holds that address until modified by input from the Address Adder or a new starting address field. The 21 most significant bits of this register (octet address) transfer to the Octet Request Register for transmission to memory. The four least significant bits are stored in the CAF for use in routing the proper operand to the data stream. The output from the register is then applied to the input of the Address Adder for possible modification.

4.9.27.2 Address Adder. The Address Adder performs address modification on the address from the Vector Address Register to form a stream of addresses to access the component operands of a vector. Any one of three modification inputs may be added to the contents of the Vector Address Register to produce the next address to be accessed. Selection of the particular input is under control of the Vector Controller in the Vector Loop Control circuit. If the loop counters indicate a self-loop is to be performed, the controller enables the DAS input. This input is a fixed "1", so that it produces an increment or decrement of one: depending upon the sign. Similarly, for changing the address to begin a new self-loop sequence (inner loop) the DAI input is added to the address: for beginning a new inner loop sequence (outer loop) the DAO input is added to the address. The output from the adder transfers to the Vector Address Register to supply the next address in the stream.

4.9.27.3 Octet Request Register (SBA/YBA). The Octet Request Register receives the 21-bit octet address from the Vector Address Register and makes it available to the Central Memory Requester for transmission to Central Memory. The individual vector controller monitors the contents of this register and produces a request to CMR when a new octet address enters the register. When CMR accepts this request and transfers the octet address to the OA Register, the Octet Request Register is free to accept a new address. During scalar operations, a direct path from the IPU AO Register loads the Octet Request Register.

4.9.27.4 Circular Address File (CAF). CAF is a file of 16, 7-bit registers. Together with its input and output controllers, it keeps track of all vector components whose addresses have been requested from memory but have not been used by the operand selection circuit that loads the MAB and/or MCD Registers. The file has a capacity of 16 unused operands, so that when this limit is reached, the address generation circuit is disabled until one of the operands is used, creating a vacancy in the file. The seven bits of the file words are divided as illustrated in figure 4-12. The four most significant bits in the word are the element address bits from the Vector Address Register. Each time a new address enters the Vector Address Register, the four least significant bits of the address transfer to the CAF and fill the four MSB's of the next vacant file word. When the bits enter the file, the input controller sets bit 4 of the file word to indicate that the element address is active. When the word is used, the output controller clears bit 4 to indicate that that word is now vacant. Bit 5 is a control bit that indicates that the word is the end of a self-loop. This bit is set every time that the Vector Controller enables the DAI or DAO input to the Address Adder. Bit 6 is set by the input controller to indicate the first word of a new octet. When the octet address in the Octet Request Register differs from the octet address in the Vector Address Register, the controller sets bit 6. The words remain stored in sequence in the file until an operand request removes the address from the file.

Each time the MBU needs a new operand to send to the AU, the output controller transfers one of the file words to the Buffer Operand Address Register (XA/YA) and clears the active bit in the file corresponding to the word that was removed. The output controller assures that the file words are removed in exactly the same order that they were put into the file so that they remain matched with their proper octets that arrive from memory. Refer to the controller discussion later in this section for a flowchart and description of the address output cycle.

4.9.28 BUFFER OPERAND ADDRESS REGISTER (XA/YA). The Buffer Operand Address Register is a 4-bit register whose output selects the proper word from an octet in the X or Y Operand Buffer for transfer into the MAB or MCD output registers. This register receives its element address from either the CAF in the case of vector streams, or directly from the IPU AO Register during scalar operations.

4.9.29 C VECTOR AND STORAGE ADDRESS GENERATION. This network generates memory addresses for storing the resultant vector (C) from a vector operation and processes the storage address of a scalar Store operation. For either operation, the addresses are coordinated with their proper data octets in the Z File so that the address is sent to memory when the corresponding data octet is in the ZB File. In addition, this network records the modification status of each halfword in the Z files for use in an update of the X or Y Buffer Files. The

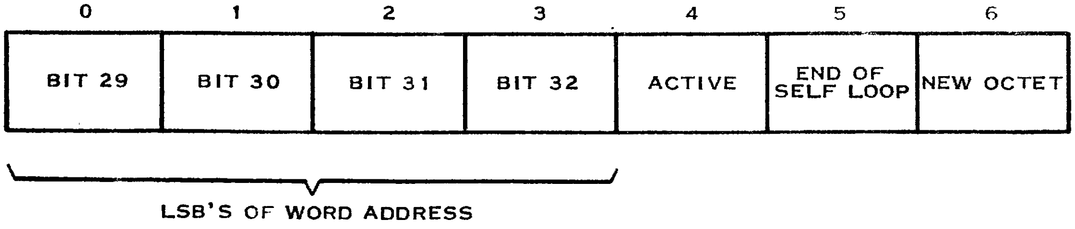


Figure 4-12. Typical CAF Word

following paragraphs describe the major component circuits of this network. Refer to the detailed theory discussion later in this section for a flowchart of the address generation circuit.

4.9.29.1 C Vector Address Register (NCA). NCA receives the 25-bit C Vector starting address field from the Vector Parameter File at the beginning of a vector operation. The Vector Initialization Controller enables this input through the IMM Register during vector start-up. The output of NCA is used to generate further addresses for storing the vector and also transfers the address to the CMR through the ZA/ZBA registers for use in storing the resultant vector in Central Memory.

4.9.29.2 Address Adder. The Address Adder modifies the address in the Vector Address Register, NCA, to form a stream of addresses for storing the resultants of a vector operation. Any one of three modification inputs may be added to the contents of NCA to produce the next address for resultant storage. Selection of the particular input is under control of the C Vector Controller in the C Vector Loop Control circuit. If the loop counters indicate a self-loop is to be performed, the controller enables the DCS input. This input is fixed "1" that produces an increment or decrement of one to the address in the register. Similarly, for changing the address to begin a new self-loop sequence (inner loop) the output from the DCI Register is added to the address; to begin a new inner loop sequence (outer loop), the DCO Register output is added to the address. The output from the adder transfers to the C Vector Address Register to supply the next address in the stream to the CMR.

4.9.29.3 Scalar Storage Address Register (NSA). NSA is a 25-bit register that contains the storage address of a scalar operation resultant that is currently in the AU pipe. A 21-bit input from the IPU ZP Register forms the octet address portion (most significant bits) of the storage address. Four bits from the IPU Register Stack (level R6) become the element address that is used to store the word within the octet contained in the Z File. The address enters NSA when the respective operands enter the AU and leaves NSA to transfer to ZA and ZEA one clock before the resultant of the operation leaves the AU EF Register.

4.9.29.4 Resultant Storage Address Register (ZA). ZA is a 21-bit octet address register. The contents of this register indicate the destination storage address of the octet that is currently in the Z File. The address enters ZA from NSA (scalar) or NCA (vector) one clock before the resultant begins storing into that octet in the Z File. The output of the register then transfers to the ZAH Register on the following Phase 1 clock pulse.

4.9.29.5 Half Phase Holding Register (ZAH). ZAH is a 21-bit octet address register that reflects the address in ZA after a one-half clock delay. Refer to the discussion of the ZH Holding File for a description of the half phase relationship of the two registers. The output of ZAH transfers to the ZBA Register when a new octet address enters the ZA Register.

4.9.29.6 Memory Storage Address Register (ZBA). ZBA is a 21-bit octet address register that contains the address of the octet that is currently in the ZB File awaiting transfer to Central Memory. The address transfers to ZBA from ZAH one clock before the corresponding octet transfers from the ZH File to the ZB File. An address that enters ZBA immediately results in a memory request to the CMR. Since store requests have priority over all other memory requests CMR immediately begins a memory cycle to store the contents of the ZB File into the memory location contained in the ZBA Register.

4.9.29.7 Halfword Modified Indicator Register (ZM). ZM is a 16-bit register that is used to keep a record of those halfwords in the Z File that have been modified by resultants from the AU. Each bit in ZM corresponds to one-half word in the Z File. When an entry is made into a particular halfword in the Z File, the corresponding indicator bit in the ZM Register is set. The

Modified Halfword Detect circuit monitors each element address sent to the Z File and generates the signal that sets the control bit in ZM.

4.9.29.8 Half Phase Holding Register (ZMH). ZMH is a 16-bit register that reflects the contents of the ZM Register after a one-half clock delay. The output bits gate the changed halfwords into the X or Y File during a Z to X (Y) update.

4.9.29.9 Memory Storage Modified Halfwords Register (ZBM). ZBM is a 16-bit register that stores the modification indicator bits for the octet of data in the ZB File. It receives its input from the ZMH Register one clock before the data file transfers from ZH to ZB. If an update of X or Y is needed during the time that its corresponding octet is still in ZH, the output of ZBM may be used to gate the modified words of that file to the corresponding words in the X or Y Files. Since memory can only consider whole word storage, the output of this register is merged into eight whole-word-modified indicator bits before being transferred to CMR as zone control bits.

4.9.29.10 Storage Word Address Register (ZEA). ZEA is a 4-bit register that receives the element address from either NCA (vector) or NSA (scalar). The output of this register directs the output from the AU EF Register into the proper word of the octet currently residing in the Z File. ZEA receives a new input address as soon as the old address has performed its gating function.

4.9.30 CENTRAL MEMORY REQUESTER (CMR). CMR performs the coordination, bookkeeping and transfer functions required to store or fetch octets in Central Memory. It assures that octets read from memory are routed to the data file that requested that particular octet. For store functions it also assures that only valid full words of data are stored into memory to avoid destroying information in Central Memory. Three addressing circuits provide requests for memory access to CMR. If conflicts occur between these requests, CMR resolves the conflict. Refer to the controller discussion that appears later in this section for flowcharts of CMR cycles.

4.9.30.1 CMR Priority Gate. This circuit monitors memory requests generated by the vector controllers during vector operation, or from the IPU during scalar operation, and resolves any conflicts that may occur over simultaneous memory requests. Since storage requests have no address buffer, storage requests involving the ZBA Register address always receive the highest priority for memory requests. Conflicts involving the addresses from XBA and YBA are resolved alternately; that is, a request for access to the location in XBA receives access to memory for the first conflict with a request from the Y pipe, but the second conflict is resolved in favor of the Y pipe request. The Priority Gate then transfers the address from the highest priority channel to the OA Address Register.

4.9.30.2 Memory Octet Address Register (OA). OA is a 21-bit register that receives memory addresses from one of three input registers and holds that address until CMR Control completes processing the request. Once the address enters OA, the output is immediately available to the AA Register.

4.9.30.3 Asynchronous Address Register (AA). AA is a 21-bit address register that provides request addresses to Central Memory. Transfer of an address from OA into AA depends only upon Central Memory's acceptance of the address previously held in AA. AA is not dependent upon clock pulses.

4.9.30.4 Halfword Bits Check and Merge. This circuit receives the 16 "halfword modified" flags from the ZBM Register and processes them for two purposes. First, the circuit checks each pair of bits that represents the two halves of a whole word. If only one of the pair of bits is set, it indicates that only half of that word contains valid data in the ZB File. The other half, being unwritten into, contains unknown and undesirable data. The circuit, therefore, signals the CMR Controller that a Z Fill-in process is required (refer to CMR Controller description). The circuit then examines the bit pairs again and inputs a "1" bit to the ZCB Register for each pair of bits having at least one bit set. This produces a set of flags in ZCB that indicates which whole words in the ZB File have been changed.

4.9.30.5 Zone Control Bit Registers (ZCB/AZC). ZCB and AZC are 8-bit registers that hold the zone control bits corresponding to the octet residing in the ZB File. Each zone control bit corresponds to one word in that octet. A set bit indicates that its word in ZB contains data that resulted from the AU pipe operations. A cleared bit indicates that no data is stored in that word. These bits enter ZCB when the address of the storage data enters the OA Register. AZC is an asynchronous register so that the control bits transfer to AZC as soon as Central Memory accepts the last group of control bits. Central Memory stores only those words in the ZB File whose corresponding zone control bit is set in the AZC Register. Valid data stored in memory is thereby not destroyed by writing unmodified (blank) words from ZB over the valid data.

4.9.30.6 CMR Control. CMR Control coordinates all memory requests, routes the resulting octets to the requesting file, and assures that only valid data is stored into Central Memory. During a fetch from memory, CMR gates the request address to memory and sends a 2-bit code to the Cue File that indicates the destination for the octet when it returns from memory (X, Y or Z pipes). When an octet enters the SC File from memory, CMR gates the octet to the proper file by accessing the cue bits in the Cue File.

At the start of a memory store operation in CMR, the controller inspects the halfword bits check circuit to determine if the octet to be stored contains any half complete words. If all the words in the octet are complete, the octet is written into memory using the zone control bits from the AZC Register to determine which words in the octet are to be written. If, however, one or more of the words in the octet are only partially filled with new data, CMR Control initiates a Z Fill-in operation.

Z Fill-in results in supplying complete words to memory. CMR Control issues a fetch request to Central Memory for the octet in the location corresponding to the address of the octet to be stored. When the octet enters the SC File, CMR Control transfers it to the ZB File using the cue bits from the Cue File. The "halfword modified" control bits from the ZBM Register prevent the incoming octet from replacing those halfwords that are already in the ZB File. The incoming octet, therefore, only stores into the vacant halfwords of the ZB File. When the ZB File is complete, CMR Control issues a request to memory to store the octet in ZB. The zone control bits sent with it to Central Memory, however, allow only the modified words to be stored, preventing alteration of valid data that is already stored in memory.

4.9.30.7 Cue File. The Cue File is a 2-bit circular file with an eight-entry capacity. Each time the CMR Controller sends a memory request for a read from memory, it also generates a 2-bit code, designating the destination of the requested octet, and enters that code in the Cue File. The cue codes are as follows:

00 = octet to X Buffer Files

01 = octet to Y Buffer Files

10 = octet to ZB File (Z Fill-in).

Since memory requests are processed by Central Memory on a first-in, first-out basis, no tag bits are required to identify the cue entries with their octet. As octets return from memory, the cue entries are accessed from the Cue File in the order that they were stored. They are then used to gate the incoming octet to the proper file.

4.10 AU INPUT

The Input section of the AU is pipe level 7 of the Central Processor. It receives the original operand inputs from the MBU, plus numerous other inputs that are developed within the AU and fed back for further processing. Of significant interest is the additional input from the EF Output Register. This input supplies the 'short-circuit' feedback that is used for performing consecutive instructions on the same set of operands. It allows the result of one operation to be used as the input for the next operation without the delay that would be required if the first result were stored and then re-accessed. Control signals from the AU Control ROM in the MBU select the proper input and transfer it to the AB or CD Operand Registers. These 64-bit registers hold the operands until further control signals route the output to the AU level corresponding to the first operation to be performed.

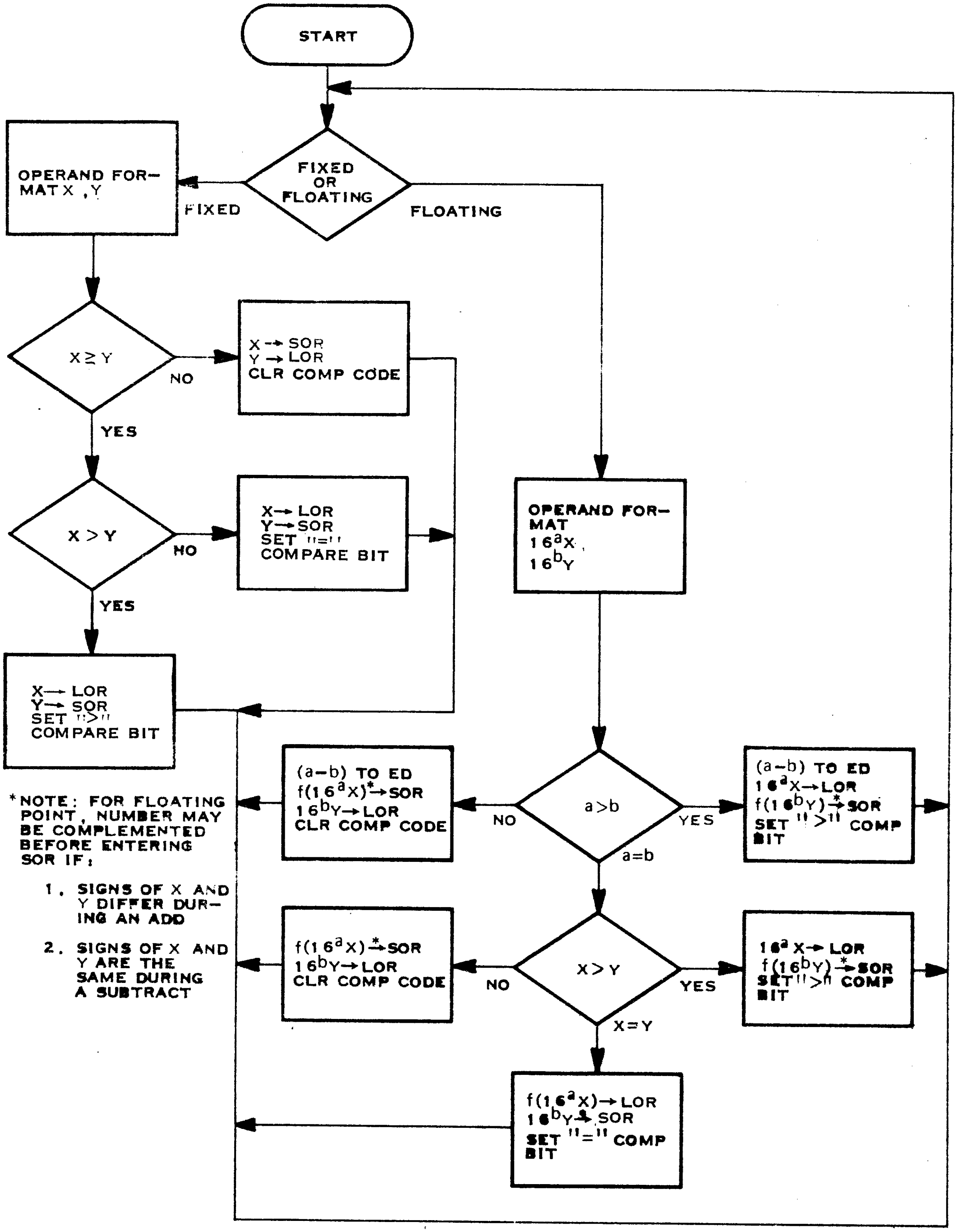
4.11 EXPONENT SUBTRACT

The Exponent Subtract section determines which of the two input operands is larger and, in the process, performs a 7-bit subtraction of the operand exponent bits. The circuit is used to input properly ordered data and a right shift count to the Aligner and Right Shift section for aligning the smaller operand to the larger. It may also be used in simple arithmetic compare operations to designate the larger operand. The following paragraphs summarize the functional blocks of the Exponent Subtract section. Figure 4-13 provides a flowchart of the comparison logic.

4.11.1 INPUT SELECT. Control bits from the AU Control ROM in the MBU select the proper pair of inputs for a designated operation. Each input is 64-bits; however, during half or single word operation, only 16 or 32 bits may be active, leaving the remaining bits ineffective.

4.11.2 SUBTRACT EXPONENTS AND COMPARE MAGNITUDE. This circuit performs a comparison of the magnitude of the input operands by subtracting the exponent portion of the data word. If the result of the subtraction (X - Y) is positive, then operand X is greater than operand Y. Operand X is then gated to the LOR Register; operand Y, to the SOR Register; and the "greater than" compare code bit is set. If the result of the subtraction is zero, the circuit compares the magnitude of the two mantissas to determine which operand is larger. If the two mantissas are equal, the "equal to" compare code bit sets, the X operand transfers to the LOR Register, and the Y operand transfers to the SOR Register. If they are unequal, either no bits or the "greater than" compare code bit sets; the incoming operands are routed to the proper output registers. If the result of the subtraction is negative, operand Y is greater than operand X. Operand X transfers to the SOR Register and Y enters the LOR Register. No compare bit sets, indicating that operand X is neither greater than nor equal to operand Y. The result of the exponent subtraction always enters the ED Register for output to the next stage of the AU.

This circuit also complements floating point numbers in preparation for effective subtraction. That is, if the signs of X and Y are different during a Floating Point Add, the operand will be complemented (one's complement) before entering the SOR Register. Similarly, if these signs of X and Y are the same during a Floating Point Subtract, the operand will also be complemented before entering the SOR Register. The carry-in bit to complete the two's complement is added in the Adder section.



*NOTE: FOR FLOATING POINT, NUMBER MAY BE COMPLEMENTED BEFORE ENTERING SOR IF:

1. SIGNS OF X AND Y DIFFER DURING AN ADD

2. SIGNS OF X AND Y ARE THE SAME DURING A SUBTRACT

(A) 117983

Figure 4-13. Exponent Subtract and Compare Logic Flowchart

4.11.3 LOR REGISTER. The LOR Register is a 64-bit register that holds the larger of the two input operands for output to the alignment section. Control signals from the Subtract and Compare circuit select which operand to transfer into the LOR Register.

4.11.4 SOR REGISTER. The SOR Register is a 64-bit register that holds the smaller of the two input operands for output to the alignment section. Control signals from the Subtract and Compare circuit select which operand to transfer into the SOR Register. The contents of SOR are right shifted in the Align circuit to make the exponent portion of the two operands (SOR and LOR) equal.

4.11.5 ED REGISTER. The ED Register is a 7-bit register that receives the result of the exponent subtraction process. The output of this register indicates the magnitude of the difference between the SOR and LOR operands and, therefore, indicates how many bits the SOR operand must be right shifted to align with the LOR operand.

4.11.6 COMPARE CODE. The Compare Code consists of two flag bits. One bit is set to indicate that the X operand is larger than the Y operand; the other bit sets to indicate that the two operands are equal. If neither bit is set, the X operand is neither larger than nor equal to the Y operand and must, therefore, be smaller than the Y operand.

4.12 ALIGN AND RIGHT SHIFT

This section is used for all floating point add instructions or for any right shift instruction. It performs floating point alignment in one pass through its circuitry and processes fixed point right shifts in two passes. A not-shifted holding stage is also supplied to maintain coordination of two corresponding operands as they pass through the section. The following paragraphs describe the major functional divisions of the Align and Right Shift circuit.

4.12.1 SELECT. The Align and Right Shift circuit has two select circuits. One select circuit supplies a 64-bit operand to the Hex Shift circuit for processing. The other select circuit provides a parameter input that indicates the magnitude of the required shift. This parameter may be specified by bits 25-31 of the CD Operand Register for a fixed point right shift, by bits 25-31 of the Accumulator Output Register for a fixed-floating or a floating-fixed conversion, or by the 7-bit output of the ED Register for a floating point add operation. The ED Register input is a base 16 number. It specifies only a shift of 4-bit multiples, or hex shift. The other inputs are base 2 numbers and specify both a hex shift and a bit shift. For this reason, floating point add alignment requires only one pass through the circuit for hex shifting, whereas the other shift operations require both a hex shift and a bit shift to complete the operation.

4.12.2 HEX SHIFT DECODE. This circuit receives the shift select bits from the select circuit and generates the required gating signals to perform the specified hex shift. Since there are 16 possible hexadecimal shifts for a 64-bit word, 16 separate gate signals are required to allow for any specified right shift. These 16 gates specify hex shifts from zero to 15 hexadecimal characters to the right.

4.12.3 BIT SHIFT DECODE. This circuit monitors the two least significant bits of the shift select network. During operations requiring a bit shift in addition to a hex shift, this circuit produces one of four possible gate signals to shift the hex shifted operand between one and four bits to the right.

4.12.4 SHIFT SEQUENCE. The Shift Sequence involves the bit shift network, the hex shift network, and the SH Shifted Operand Register. The incoming data word is first hex shifted, and the hex shifted result is stored in the SH Register. This output is available to the adder circuit for a floating point add at this time. If, however, a bit shift is required, the hex shifted operand in SH is fed back into the shift circuit gates. The bit shifted result then appears in SH.

Depending upon which hex shift gate bit is active, any one of 16 inputs to the ARSH16 FF (figure 4-14) may be enabled. The next clock pulse will transfer that enabled input to the ff and provide a hex shifted output. If a bit shifted result is required, one of the bit shift gate bits will enable the output from one of four of the SH Register ff's to be fed back as an input to the SH Register. The second clock period transfers that bit into the SH Register to complete the shift operation.

4.12.5 NOT SHIFTED REGISTER (NS). The pipeline structure of the AU requires that both operands of an operation be at the same level of the pipe at all times to avoid confusion or loss of data. Therefore, even though the larger operand of a floating point add requires no action of the Align and Right Shift section, a holding stage must be provided for that operand while the smaller operand is being hex shifted. For this purpose, NS receives the output from the LOR Register and holds that operand until the hex shift of the smaller operand is complete. Both operands then transfer to the Add section of the pipe for the addition portion of the floating point add instruction.

4.13 ADDER SECTION

The Adder section performs both addition and subtraction operations on either fixed or floating point operands. It is capable of processing two 64-bit operands during one clock period. The following paragraphs describe the functions of the major components of the AU Adder Section.

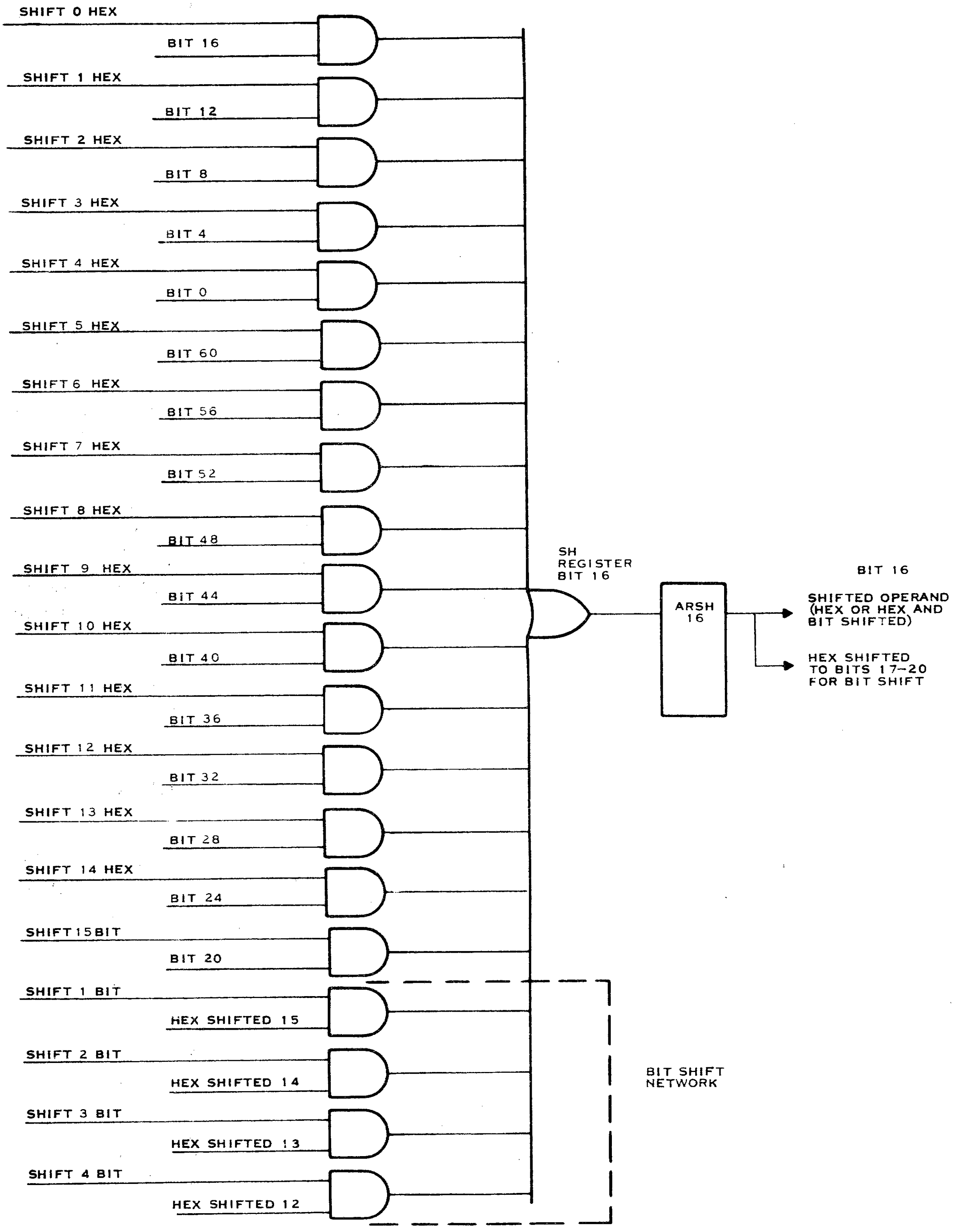
4.13.1 INPUT SELECT. Two select gates provide the pair of operands to the adder circuit. Control signals from the AU Control ROM in the MBU designate the proper gate signals to the Select circuits. For floating point adds and subtracts, the input from the Align and Right Shift section is selected. The Input section of the AU supplies operands for fixed point operations. If the adder is to perform a subtraction, control signals select the input corresponding to the complement of the subtrahend. This number will be a simple one's complement; an additional carry input to the adder creates the two's complement input required for a subtract operation.

4.13.2 ADDER. The adder performs 64-bit addition on two operands in a parallel mode. Two-level, look-ahead logic determines the carry of the total operation and adds it to the partial sum to form the sum that is transferred to the ADD Register. For subtraction operations, one of the input operands is in one's complement form. An extra bit is added to the least significant bit position of the addition to develop an equivalent to a two's complement for performing the subtraction.

4.13.3 ADDER OUTPUT REGISTER (ADD). The Adder Output Register is a 64-bit register that receives the resultant sum or difference of an adder operation and forwards that answer to the next level of the AU pipe as determined by control signals from the AU Control ROM in the MBU. The result from floating point operations is sent to the Normalize section before being placed into the EF Output Register. Fixed point results transfer directly to the EF Register.

4.14 ACCUMULATOR

The Accumulator is a special purpose adder section used to total the output of the Multiplier section for vector dot products and for other functions that require a running total. The following paragraphs describe the basic functions performed by the accumulator.



(A) 117984

Figure 4-14. Simplified Right Shift Network (Bit 16 of Operand)

4.14.1 **OPERAND SELECT.** The Accumulator has three operand select circuits that provide inputs to the adder portion of the circuit. Two or three of the select circuits may be enabled at one time. Control signals from the AU Control ROM in the MBU enable the select circuits to select the proper operands. One of the select gates provides a wrap-around path from the result sum in the ACC Register so that new incoming data may be added to the contents of ACC to form an accumulated total in ACC.

4.14.2 **ADDER.** The Accumulator adder is a 64-bit parallel adder with double level look-ahead logic for determining the carry of the addition operation. The adder has three inputs, all of which may be active at one time. Refer to the Multiplier section description for information regarding a similar 3-input adder tree used in that section. The output of the adder is stored in the ACC Register for output to the next level in the AU.

4.14.3 **ACCUMULATOR OUTPUT REGISTER (ACC).** ACC is a 64-bit register that receives the output from the accumulator adder and holds that output until a new result is entered from the adder. The output from this register is available to other stages of the AU as directed by the AU Control bits from the ROM in the MBU. A feedback path from this register to the input select circuit allows the contents of ACC to be added to incoming data to form an accumulated total.

4.15 OUTPUT

The AU output section gates all AU results to either the MBU for vectors and store operations or to the IPU Register File. It receives the output signals from all AU sections except the Multiplier section. Control signals from the AU Control ROM in the MBU select the input to this section and determine the destination of its output. This section also performs basic Boolean logical functions and reports status conditions to the IPU. The following paragraphs describe the major components of the AU Output section.

4.15.1 **LOGICAL OPERATIONS.** The Output section receives operands directly from the Input section to perform four Boolean logical operations on them. These operations are always performed on all operands that pass through the Input section. The output corresponding to the particular function need only be selected to place the logical result into the EF Output Register. The logical operations performed by this circuit are Logical AND, Logical OR, Exclusive OR, and Equivalence.

4.15.2 **OUTPUT SELECT.** The Output Select circuit uses control signals from the MBU ROM to select the proper resultant to place into the EF Output Register. The circuit may select the output from any AU section, except the Multiplier whose output is a partial result that must be added by the Accumulator to be meaningful. Three miscellaneous word inputs provide for transferring status bits or other messages to the IPU or to the MBU for storage into Central Memory. Other inputs provide a word of all zeroes, two 11-bit entries of ones, and 32 least significant bits of the ACC Register foı input to the 32 most significant bits of EF.

4.15.3 **EF REGISTER.** The AU Output Register (EF) is a 64-bit register that holds the resultant output of the AU process until it is either transferred to the Register File in the IPU (scalar operations) or to the MBU for storage into memory (vectors and store operations). The output of this register is also fed back to the AU input section for use by the next following instruction if that instruction addresses the register address of the result in the EF Register. This "short circuit" path saves the time required to fetch the result from its register storage location if the instruction immediately following it will require that same result as an operand.

4.15.4 COMPARE CODE. The Compare Code performs two functions of comparison. In both cases it provides three mutually exclusive flags to the IPU for determining whether two operands are equal or if one is larger than the other. The first usage receives the comparison flag bits from the Exponent Subtract section and uses those bits to set the X greater than Y, less than Y, or equal to Y flag. Only one flag can be set at one time. The second function monitors the logical compare circuit and sets the compare code to indicate if the result of that comparison is mixed ones and zeroes, all ones, or all zeroes.

4.15.5 RESULT CODE. The Result Code is a 3-bit, mutually exclusive flag set that indicates whether the result contained in the EF Register after an arithmetic operation is positive, negative, or equal to zero. During logical operations, these flags indicate if the result is all ones, all zeroes, or mixed ones and zeroes.

4.15.6 ARITHMETIC EXCEPTION CELLS (AE). This series of flags monitors various error bits in the AU and sets a flag if an arithmetic exception occurs. Types of arithmetic exceptions include: fixed point overflow, floating point overflow, floating point underflow, and divide check (attempt to divide by zero).

4.16 NORMALIZE SECTION

The Normalize section is employed for both floating point add instructions and fixed point left shift instructions. Divisors are routed through this section to assure bit normalized inputs for divide instructions. The section performs essentially the same as the Align and Right Shift section, except that this section must also determine the length of a hexadecimal floating point shift during normalization. During left shifts, however, the instruction word specifies the length of shift, so that the operation of this circuit is then analogous to the Align and Right Shift section. This circuit also employs a 7-bit adder to update the exponent for normalizing processes. The following paragraphs describe the major functional blocks of the Normalize section.

4.16.1 INPUT SELECT. The Normalize section has two input select circuits. The first circuit determines which input will be entered into the normalize logic to designate the length of shift required to normalize the input. Control bits from MBU ROM control the selection of the input word. The output of this select circuit is divided into two parts: the exponent and the mantissa.

The second select circuit determines the input to the shifter. The input may be the mantissa output from the first select circuit during a normalize operation. In this case a 4-bit guard digit is also added to the mantissa to avoid loss of data. During a fixed point left shift, the output from the input section is selected to the shifter. For these instructions, the AB operand is the number to be shifted and the CD operand contains the shift parameters.

The guard digit added in the second select circuit consists of the four least significant bits and is used to avoid the loss of one hexadecimal digit of accuracy resulting from truncation prior to double length addition or subtraction. Four bits are sufficient, since the only times normalization may produce a loss of accuracy, it requires a shift of only one hexadecimal digit. Normalized operands are required for the guard digit to be of maximum use. For example, when multiplying two normalized operands, the fractions will be between 2^{-4} and 2^{-1}. The result will be between 2^{-8} and 2^{-2}. Therefore, the result will always require no more than one 4-bit shift to normalize the fraction to between 2^{-4} and 2^{-1}. During an addition, if the exponents are equal, no alignment is required. Therefore, the guard digit is not necessary. If the exponents differ by one, the guard digit will retain significant information. Finally, if the exponents differ by more than one, it can be shown that the result to be normalized will require no more than one hexadecimal shift. Thus, the guard digit contains information that can be retrieved.

4.16.2 MOST SIGNIFICANT 1 SEARCH. This logic searches the incoming 56-bit mantissa, beginning with the most significant hexadecimal digit, for the first 1 bit in the number. While searching, it totals the number of hex digits that have been checked until the 1 bit is found. This total defines the number of left hex shifts required to normalize the number. From this total the circuit enables one of 16 hex shift gates in the Hex Shift Network to perform the hex shift for normalization. In order to adjust the exponent to fit the normalized mantissa, the shift count is fed to the Left Shift Code Register, where the count is added to the exponent.

4.16.3 LEFT SHIFT CODE REGISTER. This register is 5-bit holding register that stabilizes the shift code determined by the Significant 1 Search circuit and inputs that code into the exponent adder. The five bits in this register correspond to the five most significant bits of the exponent. The output from this register is also used during shift instructions to determine if the requested shift produces an overflow.

4.16.4 EXPONENT ADDER. The Exponent Adder is a 7-bit adder that receives the exponent portion of the data to be normalized and adds the hex shift count to it. The result represents the adjusted exponent that corresponds to the shifted mantissa. An additional flip-flop parallel to the adder holds the sign bit during the addition, so that the sign bit remains unmodified.

4.16.5 LEFT SHIFT HEX DECODE. During a Left Shift instruction, the CD operand from the AU Input section specifies the parameters of the left shift. This circuit receives the 7-bit shift count from that operand and enables one of 16 possible hex shift gates to the Hex Shift Network after decoding the incoming shift count to determine which shift gate to activate. The shift count parameter input to this circuit is also used to detect a possible overflow in the shift network.

4.16.6 HEX SHIFT NETWORK/BIT SHIFT NETWORK. Both shift networks are electrically integrated, but functionally separate. That is, although both shift networks employ the same type of circuitry and funnel through the same gates into the NORM Register, the hex shift must be performed before the bit shift. Refer to figure 4-15 for a simplified representation of the shift circuitry for one bit of the operand.

An incoming operand to be left shifted, whether for normalization of for bit shifting, must first pass through the hex shift network. One of 16 hex shift gates, turned on by the Significant 1 Search or the Left Shift Hex Decode circuit, enables an input gate to each bit of the NORM Register corresponding to the number of hexadecimal digits involved in the shift. For normalization, processes, except bit normalization, the process is then complete. The output of the NORM Register is available to the circuit requiring normalized data.

During a left shift or bit normalization, the output of the NORM Register is fed back through a four-gate input to the NORM Register. One of four select signals enables one of the four input gates for each bit of the operand, resulting in a shift of zero to three bits to the left. The bit shifted operand is then available to other AU levels from the output of the NORM Register.

4.16.7 NORMALIZED OUTPUT REGISTER (NORM). NORM is a 64-bit register that receives the results of the normalizer shift networks and holds them for output to the other levels of the AU, or to the bit shift network in the Normalize section. Inputs enter directly from the shift gates without enabling pulses, so that when an operand passes through the shift gates, it immediately enters the NORM Register. During floating point normalize operations, eight input bits (one sign bit plus seven exponent bits) from the Exponent Adder fill-in the exponent portion of the floating point data word in the NORM Register. The mantissa portion of the data word (56-bits) passes through the Hex Shift Network before entering the NORM Register. During left shift operations, the entire 64-bit word for the NORM Register enters from the Hex Shift Network, or from the Bit Shift Network following the second pass.

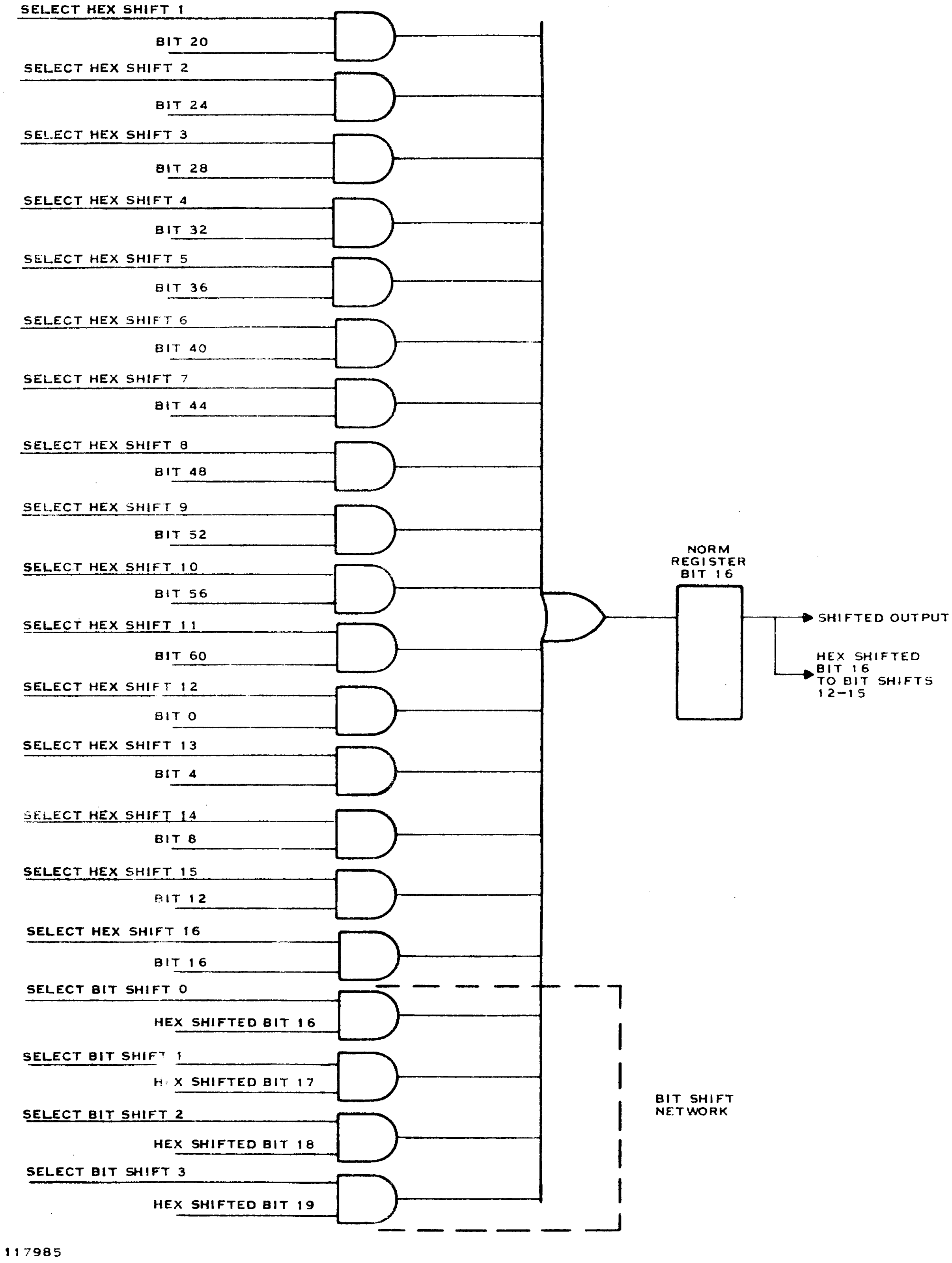


Figure 4-15. Simplified Left Shift (Normalize) Network (Bit 16 of Operand)

4.16.8 LEFT SHIFT BIT DECODE. Bit Decode receives the two least significant bits of the CD Operand Register and generates one of four input gating signals to the bit shift network. The input gating signals result in enabling a left shift of between zero and three bits. By combining the 4-bit shift with a hex shift from zero to 16 hexadecimal digits, any magnitude of left shift may be performed from zero to 63 bits.

4.16.9 BIT SHIFT MAGNITUDE DETERMINATION. During bit normalization, the Normalize section must determine the magnitude of the bit shift. To accomplish this function, a magnitude determination inspects the most significant hexadecimal digit of the hex shifted operand to locate the most significant 1 bit in that hex digit. By counting the number of zeroes preceding that 1 bit, this circuit determines the magnitude of left shift required to bit normalize the number. This circuit then generates one of four input gate signals to the Bit Shift Network to produce the required bit shift.

4.16.10 BIT SHIFT ENCODE AND REGISTER. This circuit monitors the most significant hexadecimal digit of the NORM Register during a left shift operation and produces a 2-bit code that indicates the number of bit shifts that can be performed on that number before an overflow will occur. The 2-bit code indicates a shift from 0 to 3 digits to the left, depending upon the position of the most significant one bit in the hexadecimal digit. A 2-bit register in this circuit holds the shift code for input to the Overflow Check circuit.

4.16.11 OVERFLOW CHECK. This circuit determines if an overflow of significant data occurs during a left shift instruction. By the nature of a normalize operation, no overflow can occur during that process. The seven bits of the CD operand that indicate the magnitude of the left shift enter this circuit at the beginning of the operation. Control signals then route the AB Operand through the Significant 1 Search network in addition to transferring it to the Hex Shift Network. The Search circuit creates a 5-bit code that indicates the maximum possible hex shift before data will be lost due to an overflow. This function is identical to determining the hex shift required for normalization. Overflow Check compares this code with the requested hex shift in the CD Operand. If the requested shift is greater than the maximum shift, the Overflow Flag sets. If a bit shift is to be performed, the Bit Shift Encode and Register circuit supplies a 2-bit code that specifies the maximum bit shift, the Overflow Flag sets. If a bit shift is to be performed, the Bit Shift Encode and Register circuit supplies a 2-bit code that specifies the maximum bit shift allowable before an overflow will occur. Overflow Check compares this code against the requested bit shift and sets the Overflow Flag if the requested shift is greater than the allowable shift. In any case, performance of the shift is not interrupted. An Arithmetic Exception cell in the Output section of the AU informs the IPU of the error.

4.17 MULTIPLIER SECTION

The Multiplier section of the AU performs both multiplication and division on 32-bit operands for the ASC Central Processor. Doubleword operands are processed as two single words; the results are then combined to form the doubleword result. Division is performed as a series of reciprocal multiplications. The following paragraphs describe the functional blocks that perform the multiplication. Following the block diagram description are two sections that discuss the theory of the multiplication and division processes in this section.

4.17.1 DIVIDEND REGISTER. The Dividend Register is a 64-bit register that holds the number to be divided during division. One of two inputs may supply the dividend to this register. The Normalizer section output supplies initial input for all division operations. The Accumulator input allows for loading the result of one multiply or divide immediately into the dividend register to start a new iteration.

4.17.2 DIVISOR REGISTER. The Divisor Register is a 64-bit register that holds the number to divide into the dividend. One of two inputs supply operands to this register: the normalizer section of the AU supplies normalized floating point inputs, while the Accumulator section supplies results from a previous multiplication or divide for starting a new iteration of the division process.

4.17.3 P-TERM LOGIC. The P-Term Logic uses the seven most significant bits of the divisor as an address to access a table location within the logic. The table contains 5-bit numbers that are approximations of the reciprocal of the input address. Therefore, each input of seven bits from the Divisor Register results in placing five bits into the Modifier Register that are an approximation of the reciprocal of the divisor. This reciprocal is accurate to five bits so that the maximum estimation error is less than or equal to 0.00001_2. The estimation error is eliminated in significance through several iterations in the division operation. The 5-bit P-term enters the five most significant bits of the Modifier Register; the remaining bits of the register are zeroes.

4.17.4 MODIFIER REGISTER. The Modifier Register is a 64-bit holding register for input to the Recode circuit during a division. At the start of a divide, the 5-bit approximate reciprocal of the divisor enters the Modifier Register in the five most significant bits. The multiplier multiplies the divisor and then the dividend by the fraction in the Modifier Register as the first steps in the production of a quotient. The Modifier Register also stores intermediate multiplication terms throughout the division process.

4.17.5 MULTIPLICAND/MULTIPLIER SELECT. The Multiplicand/Multiplier Select circuits provide 32-bit inputs to the Fanout and Recode circuits, respectively, during both multiply and divide operations. During multiplication, the AB operand input to the Multiplier Select and the CD Operand input to the Multiplicand Select circuits are enabled. These are, however, 64-bit inputs. Control signals from the AU Control ROM direct these circuits to enable either the most or the least significant half of the 64-bit input for output to their respective circuits.

During division, three inputs are available to the fanout circuit, depending upon the stage of the division process that has been reached. Refer to the discussion of the division process that has been reached. Refer to the discussion of the division process in the Multiplier circuit for the particular gating sequence. Three inputs are also available to the Recode circuit during a divide: two through the Modifier Register and one directly from the complemented output of the Accumulator Register. The gating of these signals is also discussed in the division process explanation.

4.17.6 RECODE. The Recode circuits inspect the incoming 32-bit multiplier word in three bit segments, as illustrated in figure 4-16. Sixteen separate recode circuits (R_0 - R_{15}) are required for a single word multiply; an additional circuit (R_{DV}) is used to generate a fraction summand during a division. R_{DV} checks the most significant bit of the multiplier. If it is a one it copies the multiplicand into the DV summard; if it is a zero, it complements the multiplicand and enters it into the DV summand. Each of the other recode circuits operates on its respective bits identically to the other decode circuits to activate one or none of four control lines to the Form Summand circuit. In general form, Recode circuit R_N monitors the 3-bit segment XYZ of the

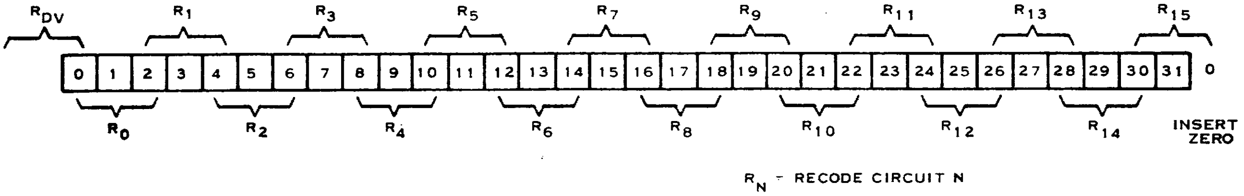


Figure 4-16. Multiplier Word Recode Bit Assignments

incoming multiplier word (Y is an even power of 2). The equivalent equations in table 4-5 define the states of X, Y and Z that produce each of four output control signals and the void state of no output control signals.

4.17.7 **FANOUT.** The Fanout circuit receives the 32-bit Multiplicand word from the Multiplicand Select circuit and duplicates it 16 times. The output from this circuit supplies 16 identical 32-bit words to the Summand Formation circuit to be used in creating the first 16 summands for input to the Adder Tree. Each of these words is a copy of the Multiplicand. During division, a 17th fanout is produced to generate the division summand in the Form Summands circuit.

4.17.8 **FORM SUMMANDS.** This circuit receives the Multiplicand words from the fanout circuit and manipulates each word, as directed by the recode bits before placing the words into their respective adder tree inputs. Figure 4-17 illustrates the arrangement of the 18 possible summands in a figurative addition array. The summand circuits are gating devices and contain no registers for the individual summands.

4.17.8.1 **Overflow Salvage.** During a left shift, a one bit from the data word may be shifted out of the summand. To avoid losing this bit, an extra bit position preceding each data word catches the shifted bit. This bit is not, however, added in the Adder Tree. It is significant only in its use to fill the Sign Extension summand, so that a shift will not change the effective sign of any summand.

Table 4-5. Recode Output Control Signal Definitions

Signal		Equation	Control Function
R_NP1	=	$X\ (Y \oplus Z)$	Copy Multiplicand into Summand N
R_NP2	=	$X \star Y \star Z$	Left shift Multiplicand by one bit and enter shifted word into Summand N
R_NM1	=	$X\ (Y \oplus Z)$	Enter the two's complement of the Multiplicand into Summand N
R_NM2	=	$X \star \bar{Y} \star Z$	Form the two's complement of the Multiplier, left shift the complement by one bit, and enter the result into Summand N★
Void	=	$X \star Y \star Z\ +$	Load zeroes into Summand N
(all signals are zero)		$X \star Y \star \bar{Z}$	

★Refer to Summand circuit description for a refinement of the complement process.

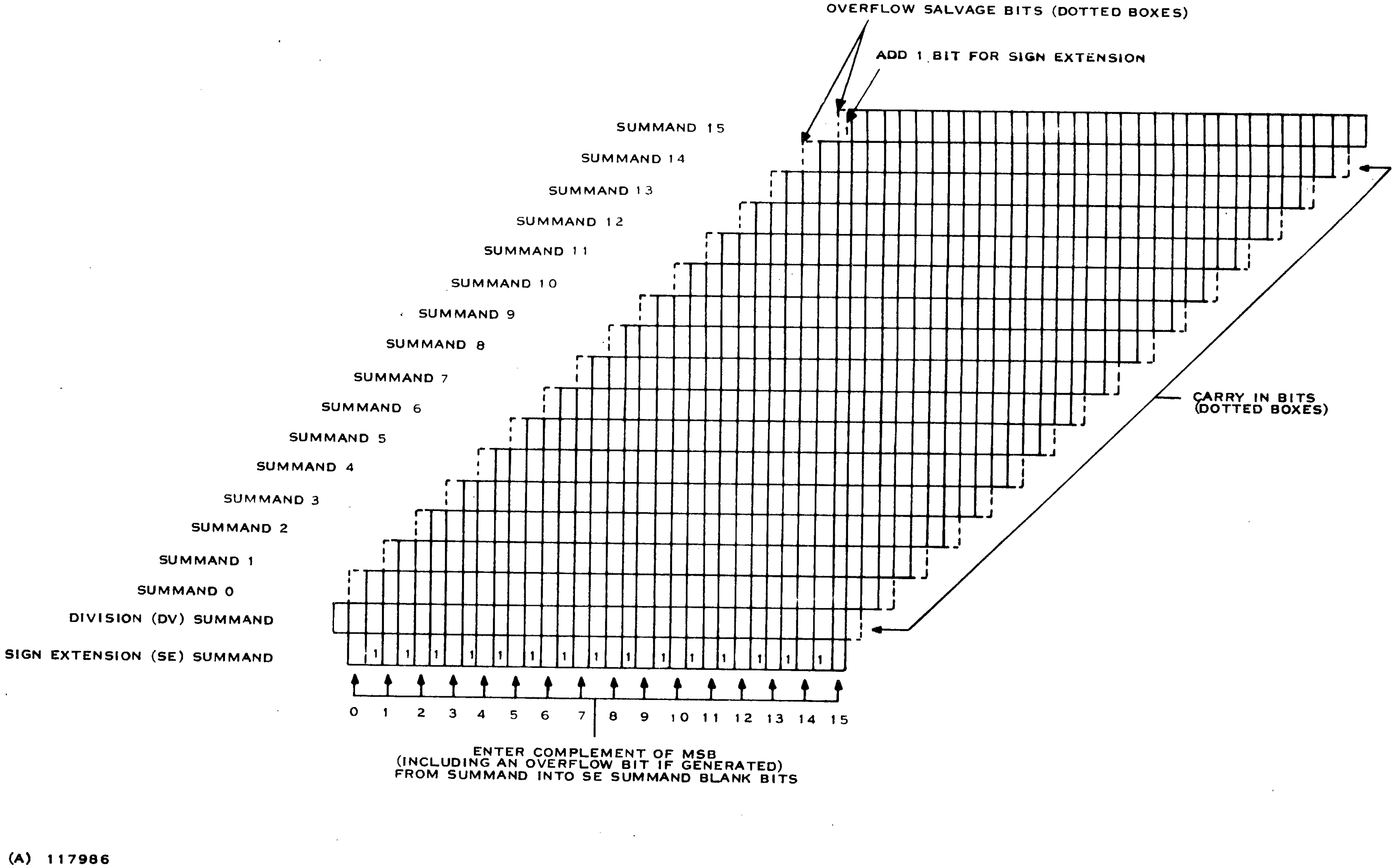


Figure 4-17. Summand Array

4.17.8.2 Sign Extension Summand. To avoid the requirement of extensive hardware to add sign extension bits of the summands, a single Sign Extension Summand represents the sum of all sign extension bits. Certain bits in the Summand are fixed at a one value; the remaining bits are governed by the complement of the most significant bit of the modified multiplicand in each preceding Summand. The sign bit will be in the overflow salvage bit if a shift occurred. Otherwise, it will be the MSB of each summand. An additional one bit, added to the least significant bit of the SE Summand, completes the sign extension total. This bit occupies the overflow salvage bit of summand 15. Refer to paragraphs 4-185 through 4-187 for further explanation of the sign extension algorithm.

4.17.8.3 Division Summand. During a divide operation a 17th multiplicand enters from the fanout circuit and is acted upon by the Division Recode bits (R_{DV}) to form the Division Summand. R_{DV} can specify only a load multiplicand ($R_{DV}P1$) or a load complemented multiplicand ($R_{DV}M1$). Addition of the Division Summand to the summand addition produces a multiplication by an unsigned fraction rather than a signed whole number.

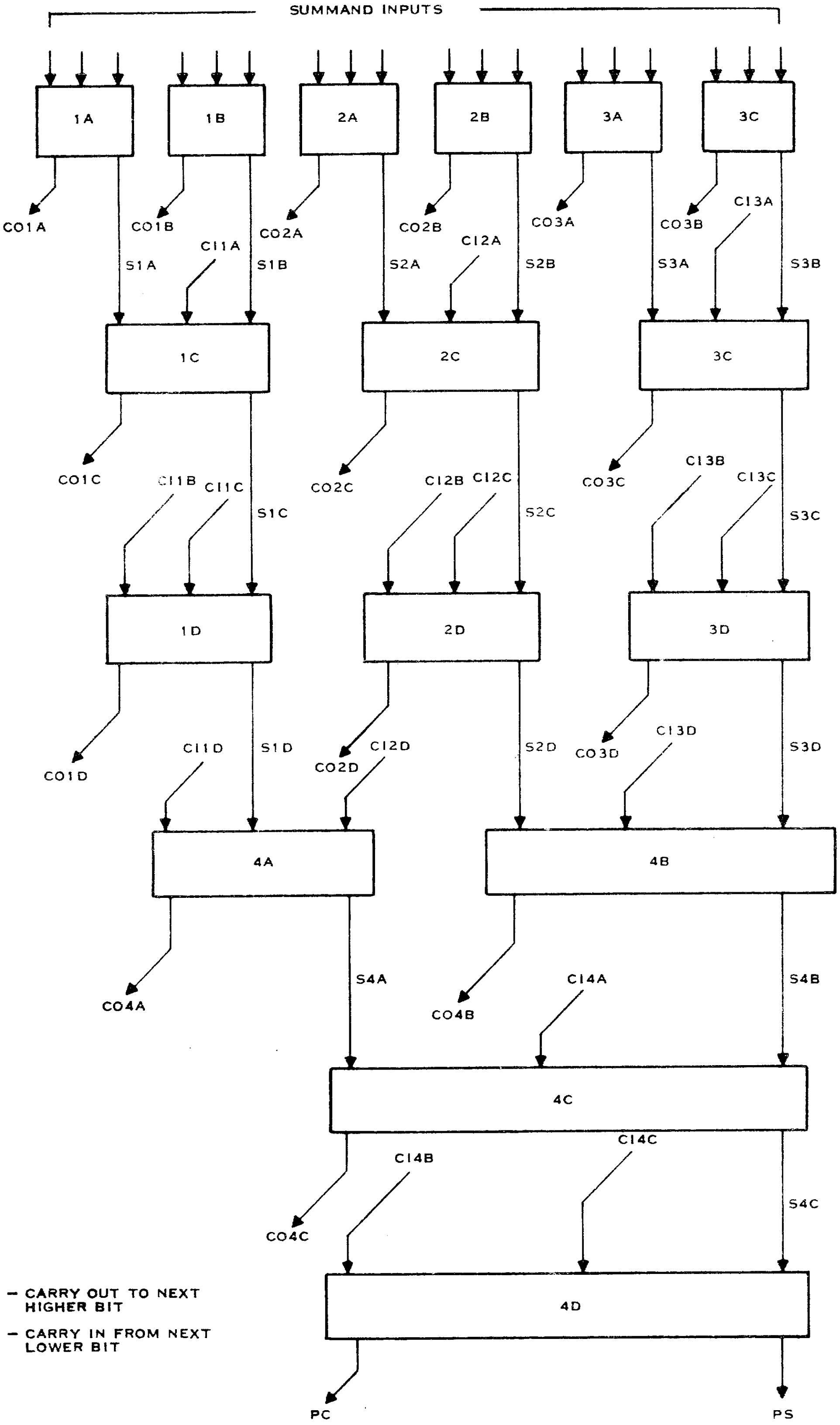
4.17.8.4 Two's Complement. To perform a two's complement, the Summand circuit forms the one's complement (invert each bit) of the word and adds one to the least significant bit. The mechanics of the operation and slightly different for each of the two complementing operations. The R_NM1 recode bit (enter two's complement) enters the one's complement of the 31 most significant bits and the true value of bit 31 into the proper Summand. If bit 31 is a zero, a carry would have been generated by the addition of one to bit 31. The circuit then enters a "carry-in" one bit into the bit 32 position of the next Summand. This results in adding that bit to bit 30 of the first summand when the Adder Tree forms the Pseudo-Sum and Pseudo-Carry.

The R_NM2 recode bit (enter two's complement and left shift) actually loads the one's complement into the proper Summand and shifts it left one bit. To complete the two's complement, it loads a one bit into the "carry-in" bit of the next Summand so that when the Adder Tree adds that column, the carry-in bit will be added to the shifted LSB of the multiplicand.

4.17.9 ADDER TREE. The Multiplier Adder Tree is an 18-input adder circuit that receives the 32-bit Summand words from the Form Summands circuit and produces two 64-bit results: a pseudo-sum and a pseudo-carry. Figure 4-18 provides a simplified circuit diagram of the Adder Tree for one bit of the addition process. The inputs represent a 1-bit vertical slice from the Summand array. There will be a similar circuit for each vertical column in the array, except that most will not be as extensive as the sample circuit due to a fewer number of elements in the column. The output from this adder in the form of two 64-bit words is not in itself meaningful. These two data words must be totaled in the Accumulator section of the AU to form the final produce or quotient.

4.17.10 PSEUDO-SUM (PS) REGISTER. PS is a 64-bit register that receives one of the resultant words from the Adder Tree during any operation that passes through the Multiplier section. The output from this register always goes to the Accumulator section of the AU to be added to the output from the PC Register.

4.17.11 PSEUDO-CARRY (PC) REGISTER. PC is a 64-bit register that receives one of the resultant words from the Adder Tree during a multiply or divide operation. The output from this register always goes to the Accumulator section of the AU to be added to the output from the PS Register.



NOTE:

1. CO – CARRY OUT TO NEXT HIGHER BIT
2. CI – CARRY IN FROM NEXT LOWER BIT

118046

Figure 4-18. Simplified Adder Tree Block Diagram

4.18 MULTIPLICATION THEORY

The AU Multiplier section multiplies two 32-bit operands to form two 64-bit partial results that are added to provide a final 64-bit product in the Accumulator section. Each 32-bit operand may be represented in the form:

$$2^n a_n + 2^{n-1} a_{n-1} + 2^{n-2} a_{n-2} + \ldots + 2^2 a_2 + 2a_1 + a_0$$

or in summation notation (for a 32-bit operand):

$$\sum_{n=0}^{31} 2^n a_n$$

Where *a* can be either a one or a zero bit coefficient, and *n* indicates the power of 2 position of that bit. The product of this operand and another 32-bit operand, B, is represented as:

$$\sum_{n=0}^{31} 2^n a_n B$$

One classical computer multiplication process performs this operation through a series of additions and register shifts. This process, for 32-bit operands, would require 32 add/shift cycles, or 32 partial Summands, depending on the desirability of execution time versus hardware bulk. The AU improves these extremes through the use of a recording algorithm that reduces the number of Summands required for a high-speed multiply.

4.18.1 ALGORITHM DERIVATION.

An equivalent binary expansion form of the multiplier is:

$$2^{2n+1} a_{2n+1} + 2^{2n} a_{2n} + 2^{2n-1} a_{2n-1} + \ldots + 2^2 a_2 + 2a_1 + a_0$$

This representation is equivalent to the first notation, but has the advantage of differentiating between odd and even terms. The summation can, therefore, be written defining half as many index values for n:

$$\sum_{n=0}^{15} 2^{2n+1} a_{2n+1} + 2^{2n} a_{2n}$$

By multiplying each odd term of the expansion by an equivalent of one, (2 - 1), a more useful expression evolves:

$$(2-1)\,2^{2n+1} a_{2n+1} + 2^{2n} a_{2n} + (2-1)\,2^{2n-1} a_{2n-1} + \ldots$$
$$+ 2^2 a_2 + (2-1)\,2a_1 + a_0$$

or, through simplification and replacement of each $2(2^{2n+1})$ with 2^{2n+2},

$$2^{2n+2} a_{2n+1} - 2^{2n+1} a_{2n+1} + 2^{2n}(a_{2n} + a_{2n-1}) - 2^{2n-1} a_{2n-1} + \ldots$$
$$+ 2^2 (a_2 + a_1) - 2a_1 + a_0$$

which can be written in summation form as (for a 32-bit operand):

$$2^{32} a_{31} + \sum_{n=0}^{15} -2^{2n+1} a_{2n+1} + 2^{2n} a_{2n} + 2^{2n} a_{2n-1}$$

The term $2^{32} a_{31}$ preceding the summation is actually a repetition of the sign bit, a_{31}, or a sign extension term that overflows the modulus of the multiplier. This term is, therefore, discarded to produce the expression

$$\sum_{n=0}^{15} -2^{2n+1} a_{2n+1} + 2^{2n} a_{2n} + 2^{2n} a_{2n-1}$$

that is the equivalent of the binary expansion for the 32-bit operand. Multiplication of the multiplicand by this representation of the multiplier requires only 16 summands (terms) instead of 32 for the normal expansion. An inspection of the multiplication using this term demonstrates the algorithm used in the AU multiplier.

To multiply multiplicand B by the multiplier A in the new format, each bit position of B is multiplied by the term

$$-2^{2n+1} a_{2n+1} + 2^{2n} a_{2n} + 2^{2n} a_{2n-1}$$

to form the 16 Summands, one for each value of n, needed to form the product. If B is represented by the expression

$$\sum_{k=0}^{31} 2^k b_k$$

then each bit of the *nth* Summand will be determined as follows:

Bit k of B = $2^k \ b_k$

times *nth* term of A = $\dfrac{-2^{2n+1} a_{2n+1} + 2^{2n} a_{2n} + 2^{2n} a_{2n-1}}{2^{2n+k^{th}}\ \text{bit} = -a_{2n+1} b_k 2^{2n+k+1} + a_{2n} b_k 2^{2n+k} + a_{2n-1} 2^{2n+k}}$

or,

$$-2a_{2n+1} b_k 2^{2n+k} + a_{2n} b_k 2^{2n+k} + a_{2n-1} b_k 2^{2n+k}$$

Which can be simplified to: $(-2a_{2n+1}\ t^a 2_n + a_{2n-1}) b_k 2^{2n+k}$

This expression indicates that the Nth Summand can be determined by multiplying each b_k (k from 0-31) by the constant inside parentheses and placing the resulting bits in the 2^{2n+k} bit position. The constant is determined by inspecting the even term (a_{2n}) and its neighboring odd terms (a_{2n-1}, a_{2n+1}) of the multiplier with respect to the relation: constant = $2a_{2n+1} + a_{2n} + a_{2n-1}$. The Recode circuit of the AU Multiplier performs this inspection to yield the results outlined in table 4-6.

4.19 AU DIVISION THEORY

The Multiplier section performs division by multiplying the dividend by the reciprocal of the divisor. That is, the division Y ÷ X is performed as the equivalent multiplication:

$$1/X \cdot Y.$$

Both X and Y are the mantissa portion of normalized floating point numbers. The Exponent Subtract section processes the exponent portion of the data. This division process, however, requires finding the reciprocal of X.

The AU approximates this reciprocal by providing a table of reciprocals (P-Term Logic) that is accurate to four binary places. This table, therefore, yields a reciprocal that is accurate within $\pm 0.00001_2$. For discussion, the Greek letter (Δ) represents the absolute value of this error. The reciprocal table provides a first approximation of the reciprocal of:

$$1/X \pm \Delta$$

to provide an approximation of the quotient (Q), through multiplication, of:

$$\begin{aligned} Y/X = Q &\approx Y(1/X \pm \Delta) \\ Q &\approx Q(1 \pm X\Delta). \end{aligned}$$

Table 4-6. Recode Circuit Data Analysis

Incoming Bits $a_{2n+1}, a_{2n}, a_{2n-1}$	**Resulting Constant (C)**	**Required Action to perform multiplication (C)*(b_k)**
000, or 111	0	Fill Summand with zeroes.
110, or 101	-1	Perform two's complement of B and enter into Summand.
010, or 001	+1	Copy each b_k into corresponding bit positions of Summand.
011	+2	Left shift the B operand one bit and enter into Summand.
100	-2	Complement B, left shift it one bit, and enter result into Summand.

Since X is normalized, it is of the form:

0.1——
0.01—
0.001–

or

0.0001

Therefore, the product $(X\Delta)$ can be no greater than Δ. For this worst case, as the error approaches Δ, the value of X in the product $X\Delta$ can be disregarded to yield an approximation of Q:

$$Q \approx Q(1 \pm\Delta).$$

To reach a more acceptable value of Q, further refinement is necessary. But multiplying the approximate quotient by an approximation of one, this refinement is achieved. This operation produces:

$$Q \approx Q(1 \pm\Delta)\ (1 \pm\Delta) = Q(1 - \Delta^2).$$

The error is reduced from $\pm\Delta$ (4-bit accuracy) to $-\Delta^2$ (8-bit accuracy). Extending this process develops the following approximations:

$$Q \approx Q(1 - \Delta^2)\ (1 + \Delta^2) = Q(1 - \Delta^4)$$

and

$$Q \approx Q(1 - \Delta^4)\ (1 + \Delta^4) = Q(1 - \Delta^8).$$

This last step reduces the error to Δ^8, (32-bit accuracy), which is not significant for single word (32 bit) operands. Double length operands require an additional step:

$$Q \approx Q(1 - \Delta^8)\ (1 + \Delta^8) = Q(1 - \Delta^{16})$$

to provide a value of Q that is accurate to 64 bits, or 2^{-65}.

Figure 4-19 illustrates the paths utilized by the AU to accomplish the division process. The vertical sequence on the left of the figure derives the multipliers for each step of the refinement process; the right column produces the approximations of Q. Notice that both the Accumulator and the Multiplier sections of the AU are in continuous use until a solution produced, each one being used alternately by one of the two derivation cycles.

4.20 SIGN EXTENSION ALGORITHM

The sign extension algorithm allows the replacement of 272 Summands and their associated adder tree logic with a single 32-bit Summand that uses existing inputs to the adder tree. The algorithm determines the sum of all sign extension bits by inspecting the most significant bit (or overflow bit if the summand was shifted) of each summand. The resulting sum of the sign extension bits is determined by complementing the most significant bit, including overflow, of Summand 15 and placing the result in the least significant bit of the SE Summand. The complemented MSB's of the remaining summands enter every other bit of the SE Summand in sequence, while one bits fill the intervening bit positions of the SE Summand. Finally, an additional one bit is added to the LSB of the SE Summand to complete the process (this bit actually occupies the "overflow salvage" bit position of Summand 15).

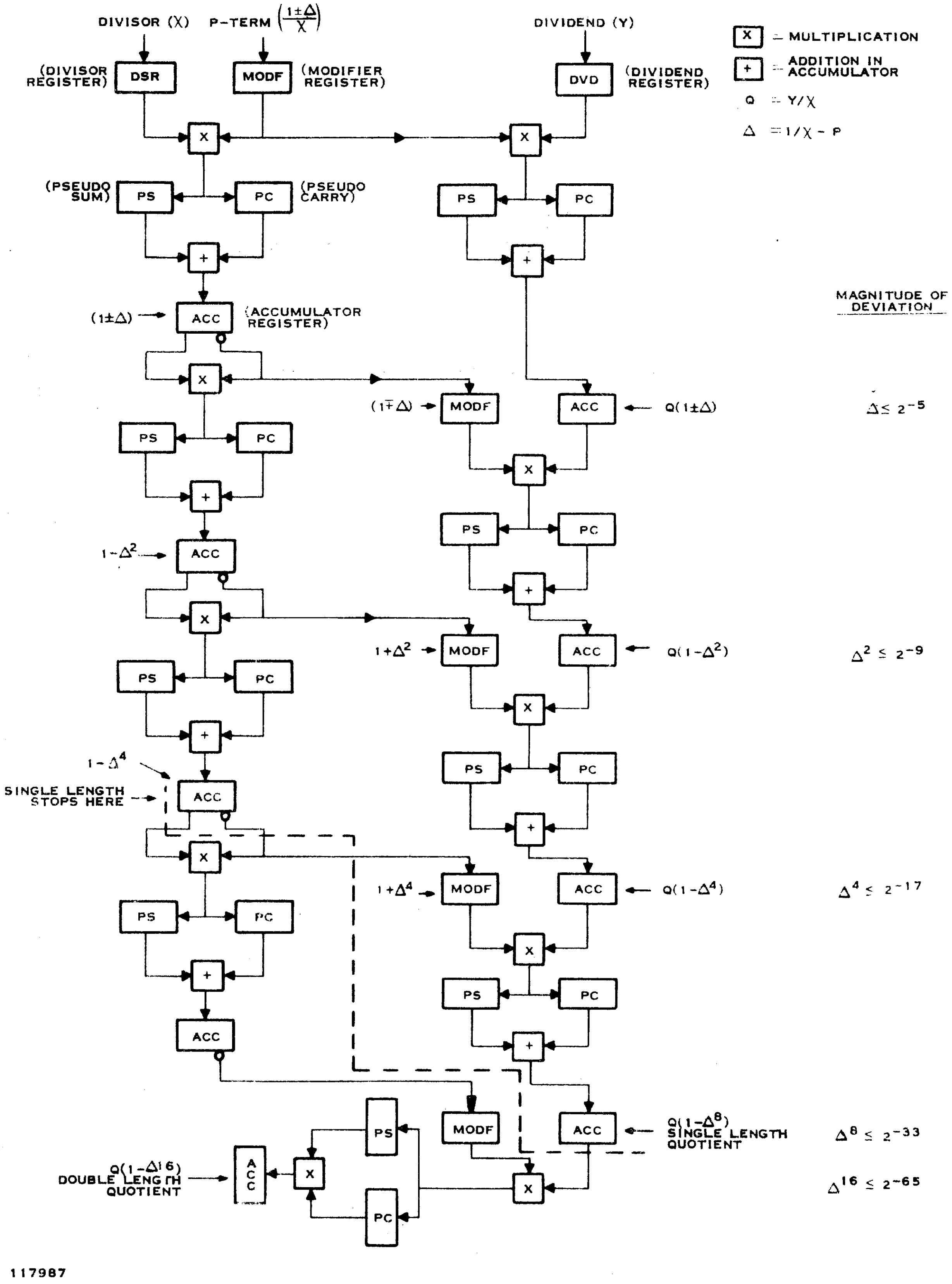


Figure 4-19. CP Hardware Utilization-Division Process

4.20.1 TWO'S COMPLEMENT FORMATION.

Before discussing the derivation of the sign extension algorithm, two methods of forming the two's complement of a binary number must be reviewed. The first method forms the two's complement of a number by starting with the LSB of the number and copying all bits of the number up to and including the first one bit. After that point, all bits of the number are inverted. That is, copy the LSB. If that bit is a one, complement thc remaining bits of the number. If it is a zero, copy the next bit, inspecting it in a like manner.

The second method of two's complementing forms the two's complement by creating the one's complement of each bit of the number and then adding one to the LSB of the complemented number and performing any carry addition that may be required.

4.20.2 ALGORITHM DERIVATION

NOTE

> The following derivation considers only a three-entry sign extension. Expansion of this explanation to a 16-entry extension as employed in the AU is valid, but for simplicity of presentation is not covered in this discussion.

Consider the sign extension bits from Summands 0, 1 and 2. Using letters to represent these bits, they would appear in the Summand array as:

```
        CCCCCC --------
        BBBB ----------
        AA ------------
      -------------------
Total = abcdef
```

By inspection of the addition and application of Boolean logical functions, the sum of the sign extension bits (abcdef) can be expressed in terms of the sign extension bits A, B, and C:

$f = C$
$e = C$
$d = B \oplus C$ *
$c = B + C$
$b = A \oplus (B + C)$
$a = A + B + C$

($\oplus$ is the symbol for exclusive OR).

This expression does not define a process for directly determining the sum. To further simplify this expression, the relation, $A \equiv (\bar{A}+6,)$, will be used. That is, a number is always equal to the complement of the complement of that number.

*d = 1 if B or C is a one, but not if both are ones. If B and C are ones, d = 0 and a carry is generated to the next bit, c.

Similarly,

c = 1 if B or C is a one, or if both are a one. If both are ones, however, the sum includes a carry from bit d, and in turn generates a carry to bit b.

Forming the two's complement of abcdef, using the first method outlined above, produces a new number, UVWXYZ, whose individual bits are defined as follows:

$Z = f$	(copy first bit)
$Y = \bar{e}f + e\bar{f}$ $= e \oplus f$	(if f = 1, complement e, if not, then copy e)
$X = \bar{d}(e + f) + d(\overline{e + f})$ $= d \oplus (e + f)$	(complement d if any preceding bit is a one; otherwise, copy d)
$W = \bar{c}(e + f + d) + c(\overline{d + e + f})$ $= c \oplus (d + e + f)$	(similar to d)
$V = b \oplus (c + d + e + f)$	(similar to d)
$U = a \oplus (b + c + d + e + f)$	(similar to d)

Substituting the equivalent forms of abcdef into the definitions for UVWXYZ yields the following relationships:

$$Z = C$$

$$Y = C \oplus C = 0$$

$$X = (B \oplus C) \oplus (C + C) = (B \oplus C) \oplus C = B$$

$$W = (B + C) \oplus [(B \oplus C) + C] = (B + C) \oplus (B + C) = 0$$

$$\begin{aligned} V &= [A \oplus (B + C)] \oplus [(B + C) + (B + C)] \\ &= [A \oplus (B + C)] \oplus (B + C) = A \end{aligned}$$

$$\begin{aligned} U &= (A + B + C) \oplus [A \oplus (B + C)] + (B + C) \\ &= (A + B + C) \oplus (A + B + C) = 0 \end{aligned}$$

These expressions indicate that the two's complement of the sum of the sign extension bits may be represented as:

0A0B0C

Forming the two's complement of this representation (using the second method explained above) returns to an equivalent expression of the sum of the sign extension bits:

$$\begin{array}{l} 1\,\bar{A}\,1\,\bar{B}\,1\,\bar{C} \\ \quad\quad\;\; +\,1 \end{array}$$

This expression defines the algorithm used to fill the Sign Extension Summand in the Summand array. Extending the SE Summand expression to 32-bits expands it to the right by adding bits of alternate ones and complemented sign extension bits. The additional one bit is always added to the last significant bit of the sum.

4.21 AU UNIT HARD CORE

Unit Hard Core performs context switches, power down sequences, and other maintenance functions, as instructed by the CP Master Hard Core. Refer to Appendix D for further information about AU hard core.

4.22 CONTROLLER DESCRIPTIONS AND FLOWCHARTS

The following pages contain information concerning the control circuits within the ASC Central Processor. Each control circuit is represented by a flowchart that outlines the decision paths within the controller. Text accompanying the flowcharts explains these paths in relation to the functions performed and signals generated by the controller.

IPU control can be viewed as shown in figure 4-20. The control of each level of the pipeline through the IPU is treated independently in the flowcharts that accompany each control circuit description. Each level controller serves three primary functions: data selection, register gating, and generation of the "path ahead clear" (PAC) function for that level. The data selection and register gating required for transfers between levels do not affect the progress of an instruction from one level to the next. The PAC, along with other circuits in the controller, determine when data gates into the next level.

In general, the flowcharts for level 0 control through level 4 control indicate the following actions:

1. The gating of data into the succeeding level is usually indicated by the statement "LVLn→LVL_{n+1}", and is accompanied by setting the activity bit at level n+1, "1→A_{n+1}". In general, this action occurs if the path ahead is clear into level n+1, level n is active, and hazards do not exist.

2. The progression of the inactivity to a succeeding level is usually indicated by resetting the activity bit at level n+1 "0→A_{n+1}".

   In general, this action occurs if the path ahead is clear into level n+1, but level n is inactive or a hazard exists.

3. The communication between a given controller and other controllers is indicated by output statements; for example, "PAC_n". This statement normally occurs if level n is inactive or if level n is active and is passing its data to level n+1.

4. Changing the primary control cells at a given level constitutes a state change at that level. These state changes normally occur when the subsequent actions at a given level are dependent on current conditions at that level.

5. State changes at a given level are sometimes accomplished by manipulation of secondary control cells. These cells are referred to as flags or counters. Changes in these flags are indicated by statements such as "1→HOLD FLAG", or "DEC COMP CTR."

4.22.1 INSTRUCTION FLOW. The data residing at a given level of the IPU is usually an instruction in a partially decoded and developed condition. This data usually passes from one level to the next, never occupying more than one level at any given time. For such cases, the flowchart for each level can easily be studied independently. There are, however, several situations for which the levels in the IPU do not operate independently. The following paragraphs supply an overview of the IPU objectives in order to more easily understand the flowcharts for these situations.

4.22.1.1 Indirect Addressing. As an indirect instruction proceeds through the IPU, it reserves the level 1 through level 3. Thus, when an indirect instruction reaches level 3, levels 1 and 2 are inactive. Level 3 control makes a request for the indirect cell via the IPU memory bus and then becomes inactive. When the indirect cell is available, it enters level 1, proceeds through level 2, and finally replaces the address associated with the original instruction which still resides at level 3. When the indirect cell enters level 3, the activity bit at level 3 is set again, and the level 3

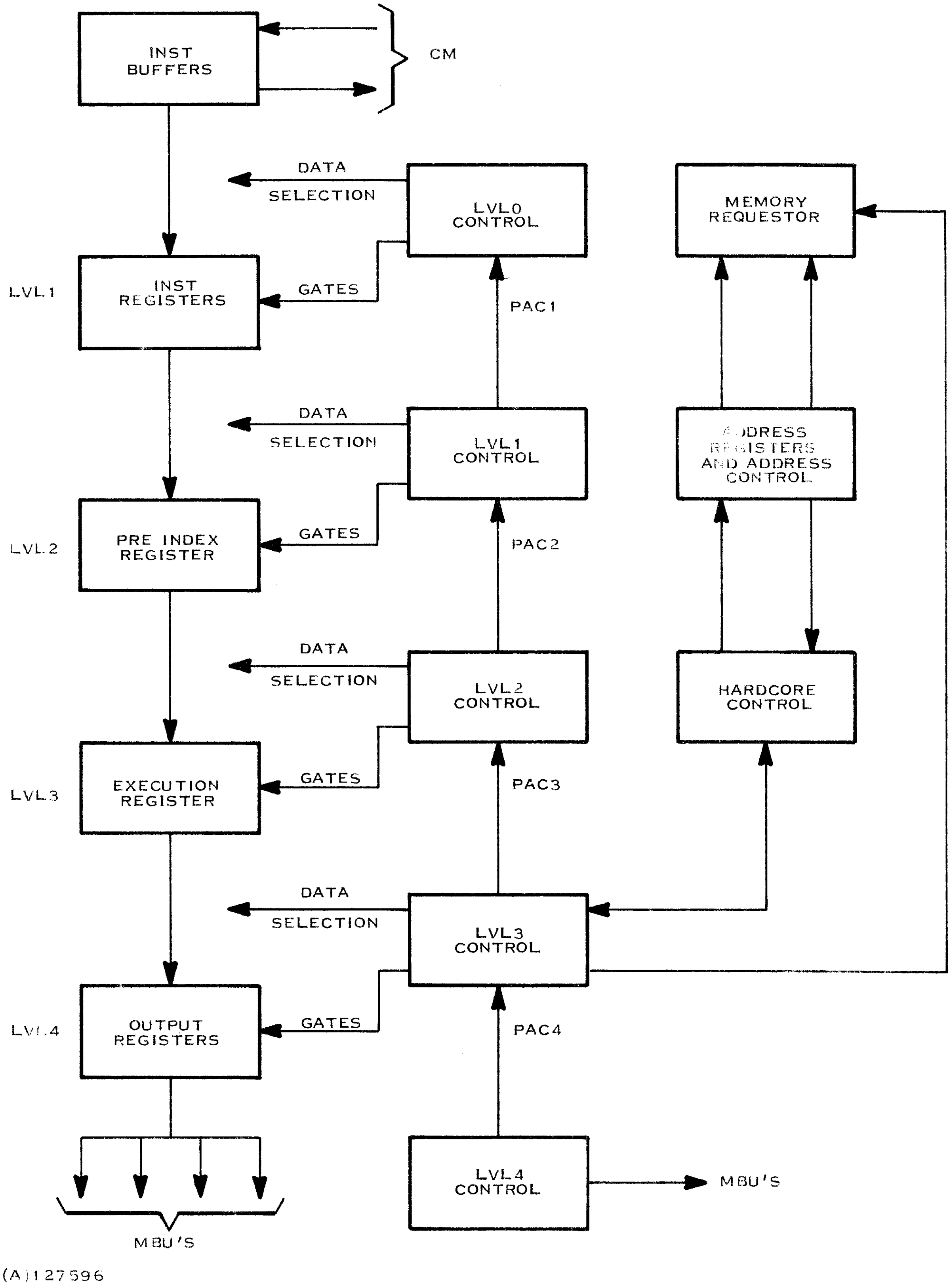


Figure 4-20. IPU Control

controller is again aware of the instruction. If the address is still indirect, then levels 1 and 2 were again reserved as the indirect cell passed through. As the terminal indirect cell progresses through levels 1 and 2, these levels revert to their usual condition, and take no further part in indirect cycling. Indirect addresses never advance down the pipe beyond level 3.

4.22.1.2 **Execute Instruction.** As an execute instruction (XEC) passes through the IPU to level 3 it reserves levels 1 and 2 of the pipe in a manner similar to an indirect instruction. Level 3 control makes a request for the object of the XEC, sets the XEC flag, and becomes inactive. When the object of the XEC instruction reaches level 3, all trace of the original instruction is gone except that the XEC flag is set and the address register at level 3 contains the address of the XEC. The object instruction is performed as if it had been in the program string in the position of the XEC. The XEC flag alters skips, branches, and calls such that the program string is not altered. At the conclusion of the instruction the XEC flag is reset.

4.22.1.3 **Skips.** Skips produce a SKIP signal from level 3. Each upstream level control observes pipe activity in the upper levels and the instruction to be skipped is inactivated if it is in the pipe. If it is not yet in the pipe, then this fact is recorded by the level 1 control. When the instruction does appear in level 1, it is discarded by the Level 1 controller.

4.22.1.4 **Branches.** A branch instruction produces commands from level 3 to the upstream levels. Each upstream level deactivates those instructions which are in the pipe but which are not desired because of the branch. If the instruction to which the branch is taken is not in the pipe upstream from level 3, then the branch address in level 3 is accepted by address control, and the pipe remains inactive through level 3 until the new instruction stream can be fetched and started down the pipe.

Indirect branches reserve levels 1 and 2 of the pipe as do all indirect instructions. However, indirect cycling does not begin until and unless the branch test is satisfied.

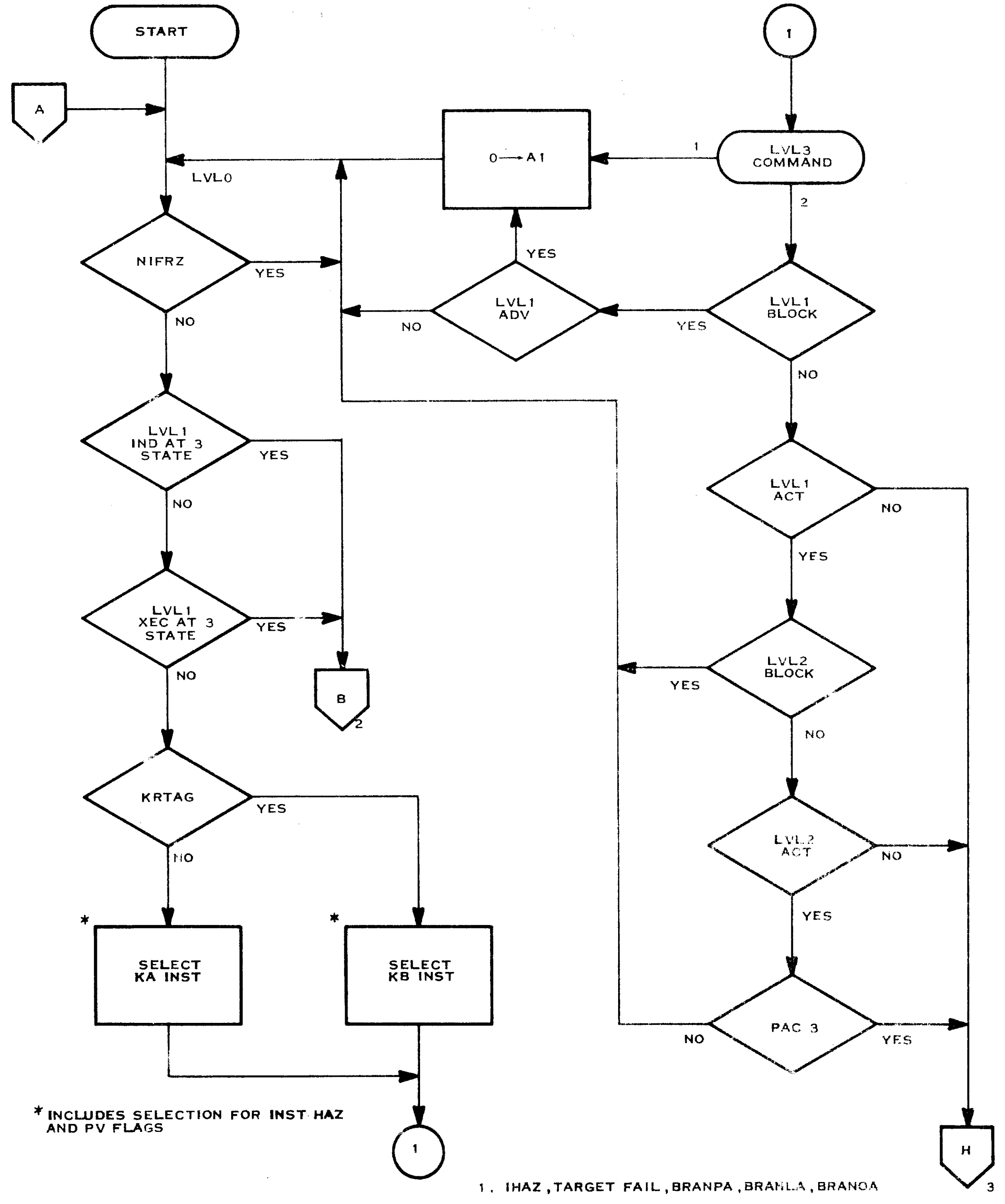
4.22.1.5 **Store File and Load File Instructions.** Store File and Load File instructions reserve level 2 as they pass down the pipe to level 3. This block at level 2 eliminates special and extensive hazard detection logic which would be required if an instruction were at level 2 during execution of file instructions at level 3.

Memory requests required for execution of the file instructions are initiated by level 3 control via the IPU memory bus. Normal instruction flow resumes through level 2 after the file instruction.

4.22.1.6 **Push, Pull Instructions.** Push and Pull instructions reserve level 2 as they pass down the pipe to level 3. The Push or Pull instruction occupies level 3 while the address of the stack parameters advances to the MBU. The MBU fetches/and transfers them to the AU for modification and testing. When available for the AU, the stack pointer is accepted into level 2, advances to level 3, and proceeds down the pipe, appearing to be a load (Pull) or a store (Push). As the pointer moves into level 3, normal instruction flow into level 2 resumes.

If the AU test indicates termination should result, level 3 control terminates the operation and resumes normal instruction flow. If termination is not necessary, then the address of the stack parameters advances to the MBU and the MBU stores the modified values.

Controller flowcharts for levels 0-4 and a level 3 controller state diagram are provided in figure 4-21 through 4-40. These include flowcharts for R/Z Join, Arithmetic Exception (AE) and AE Mask, along with those for Compare Code and Result Code controllers.



* INCLUDES SELECTION FOR INST HAZ AND PV FLAGS

1. IHAZ, TARGET FAIL, BRANPA, BRANLA, BRANOA
2. NONE OF THE ABOVE

(B)127597 (1/3)

Figure 4-21. Level 0 Controller Flowchart (Sheet 1 of 3)

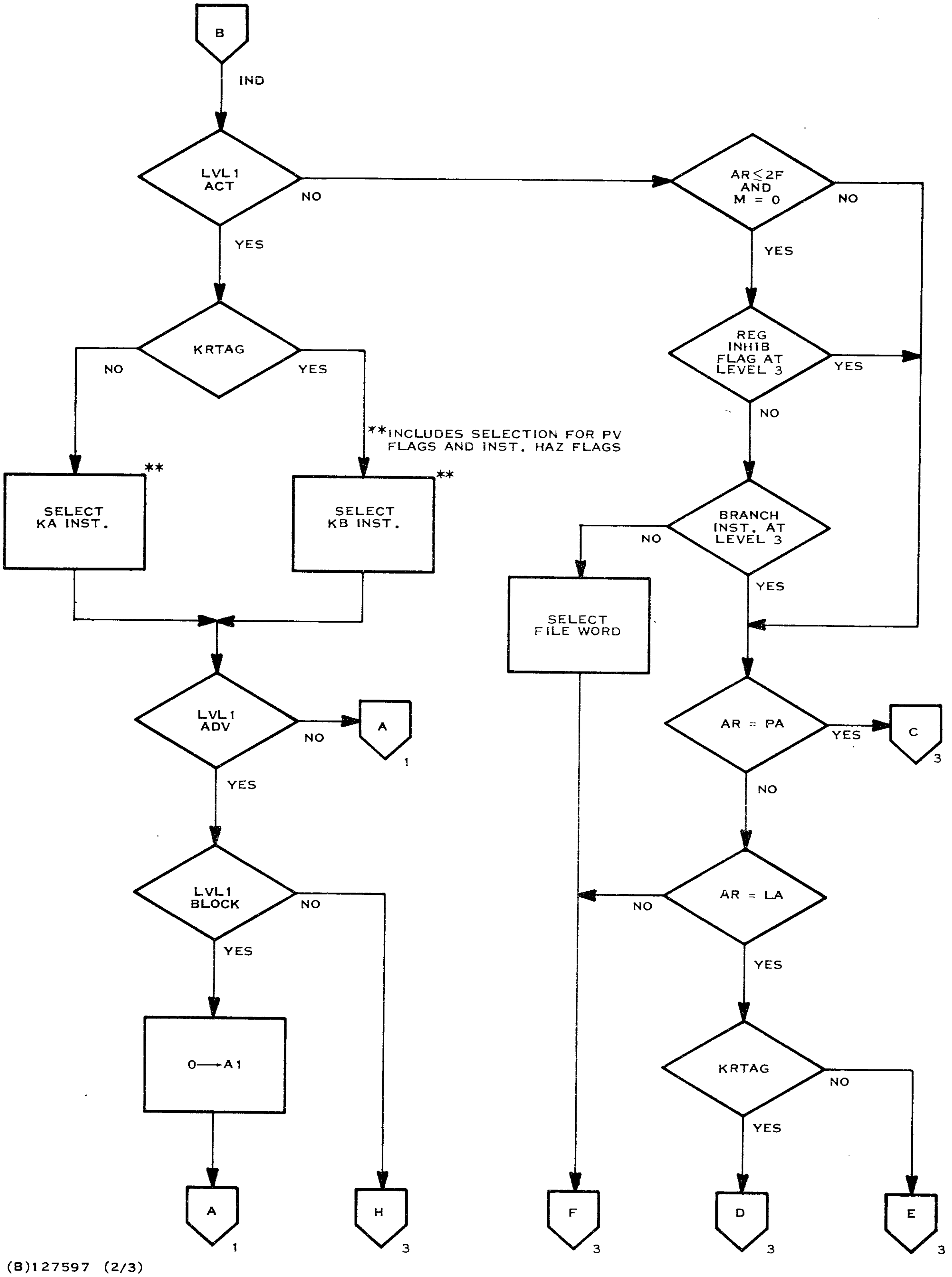


Figure 4-21. Level 0 Controller Flowchart (Sheet 2 of 3)

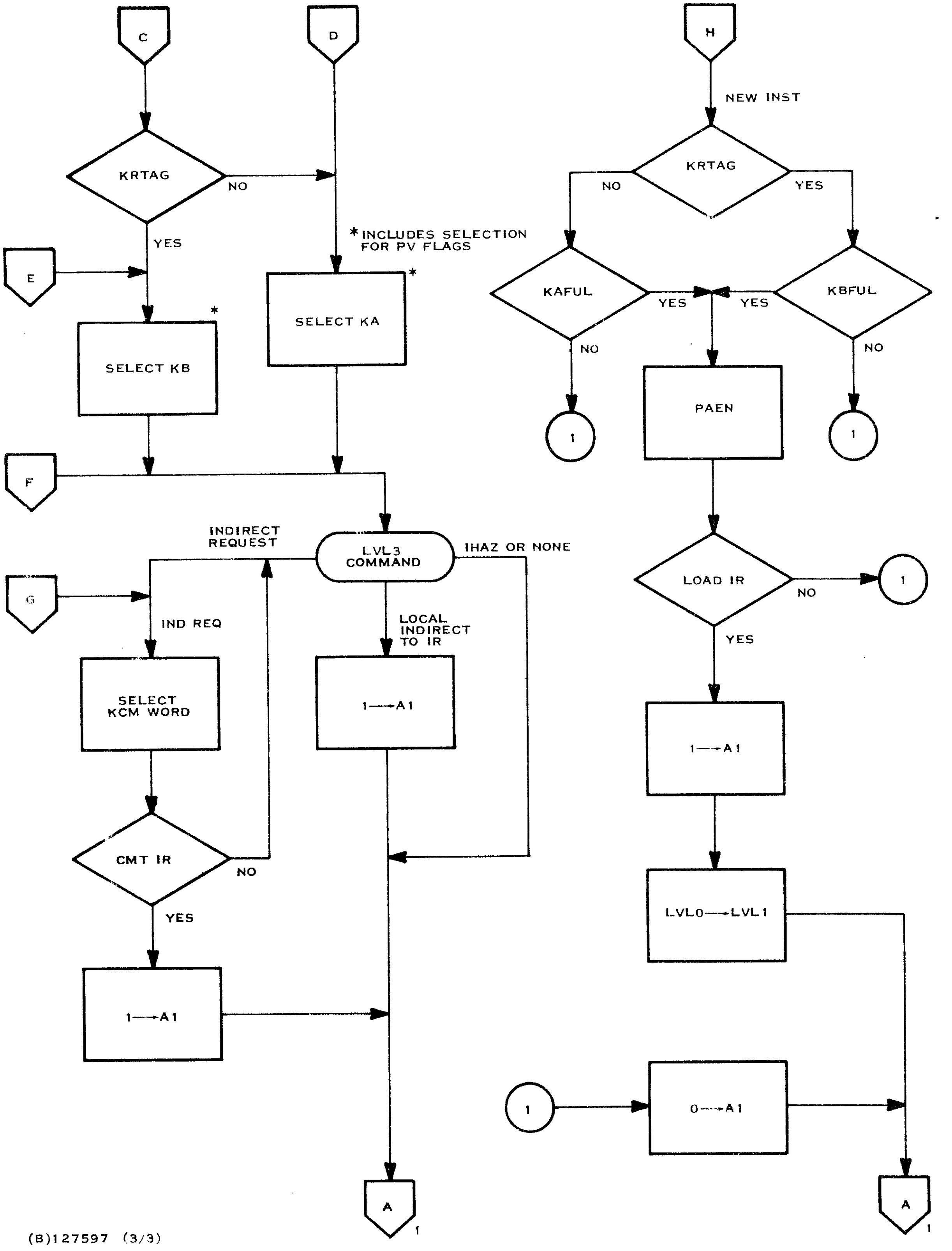
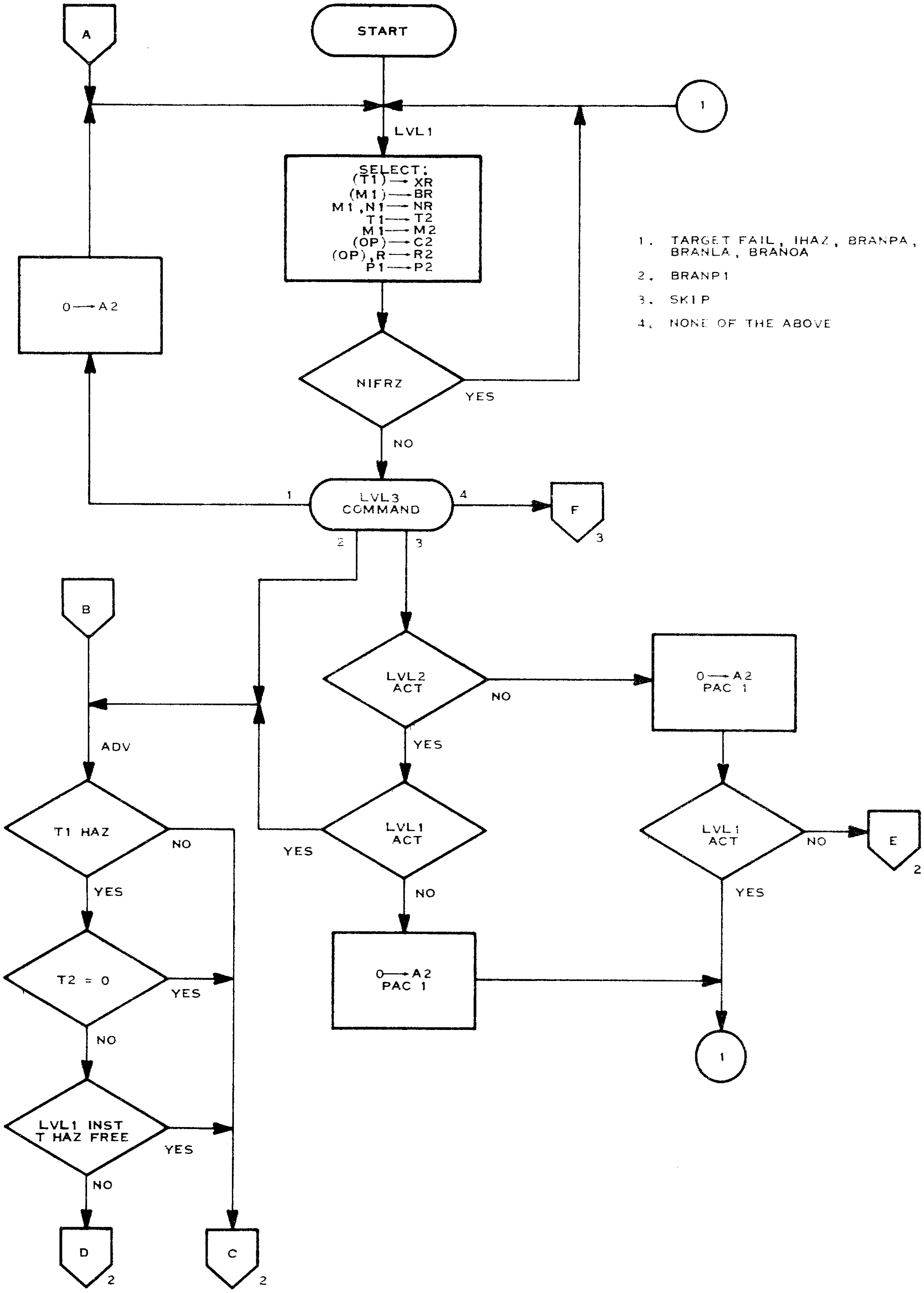
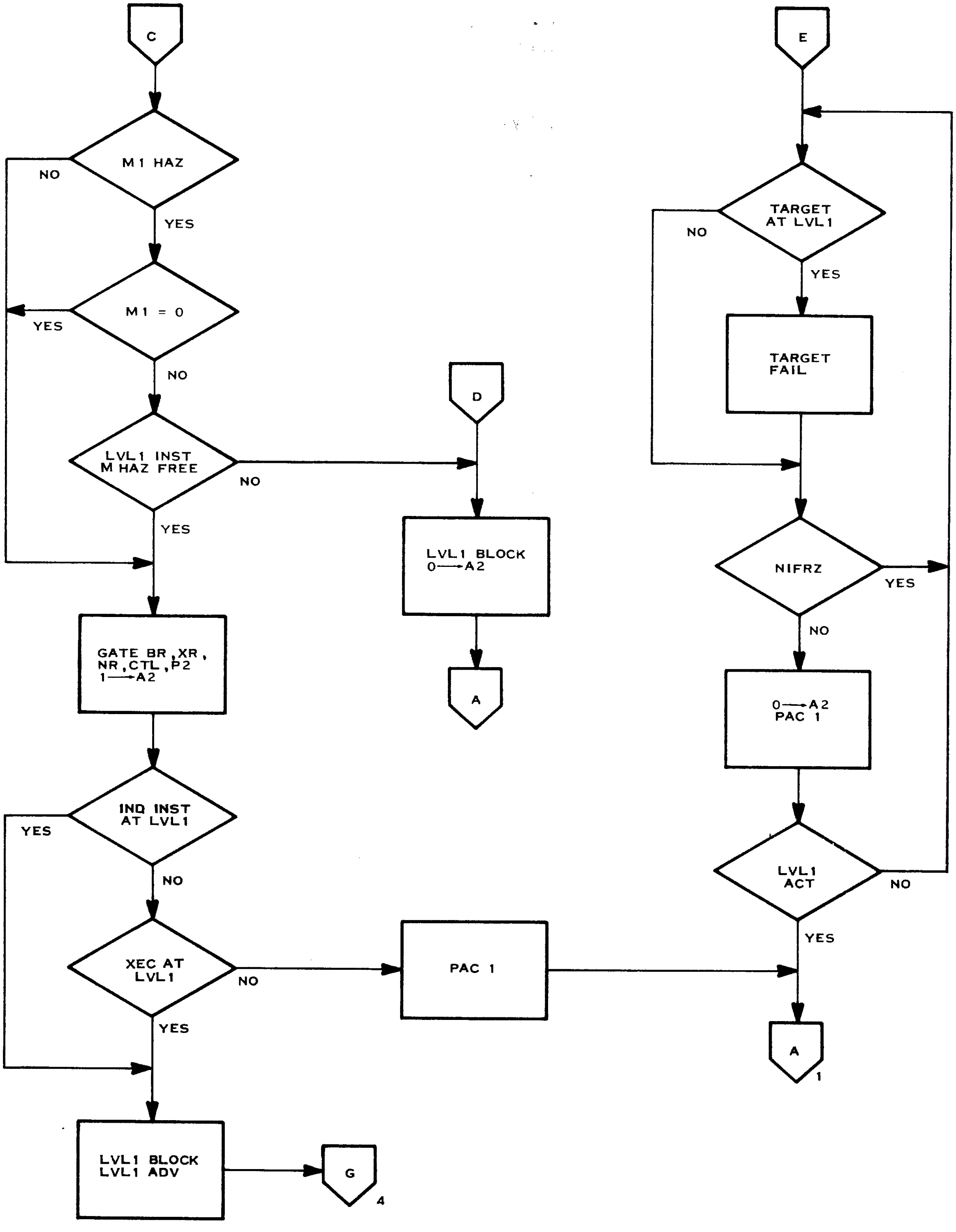


Figure 4-21. Level 0 Controller Flowchart (Sheet 3 of 3)



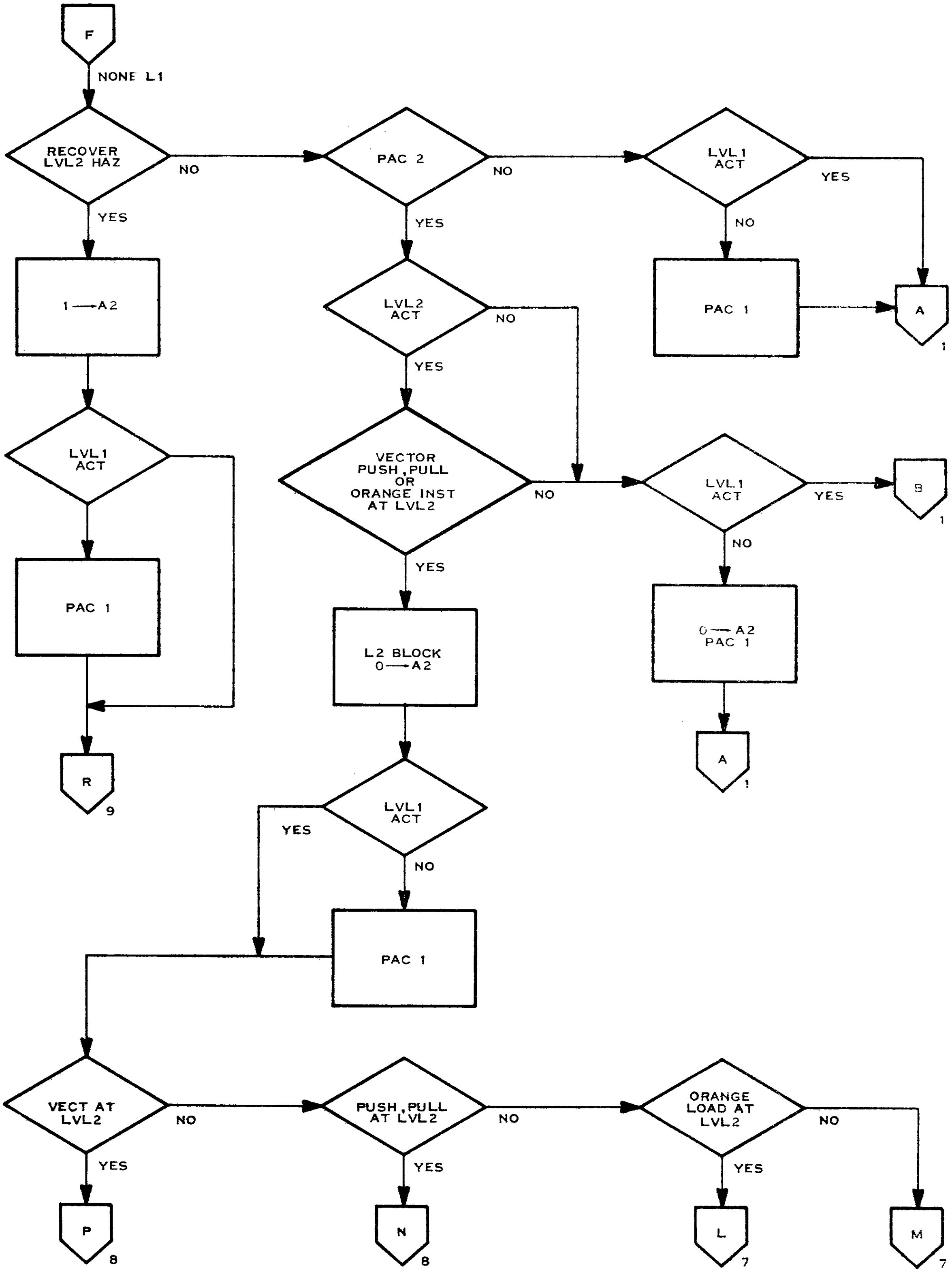
(B)127598 (1/9)

Figure 4-22. Level 1 Controller Flowchart (Sheet 1 of 9)



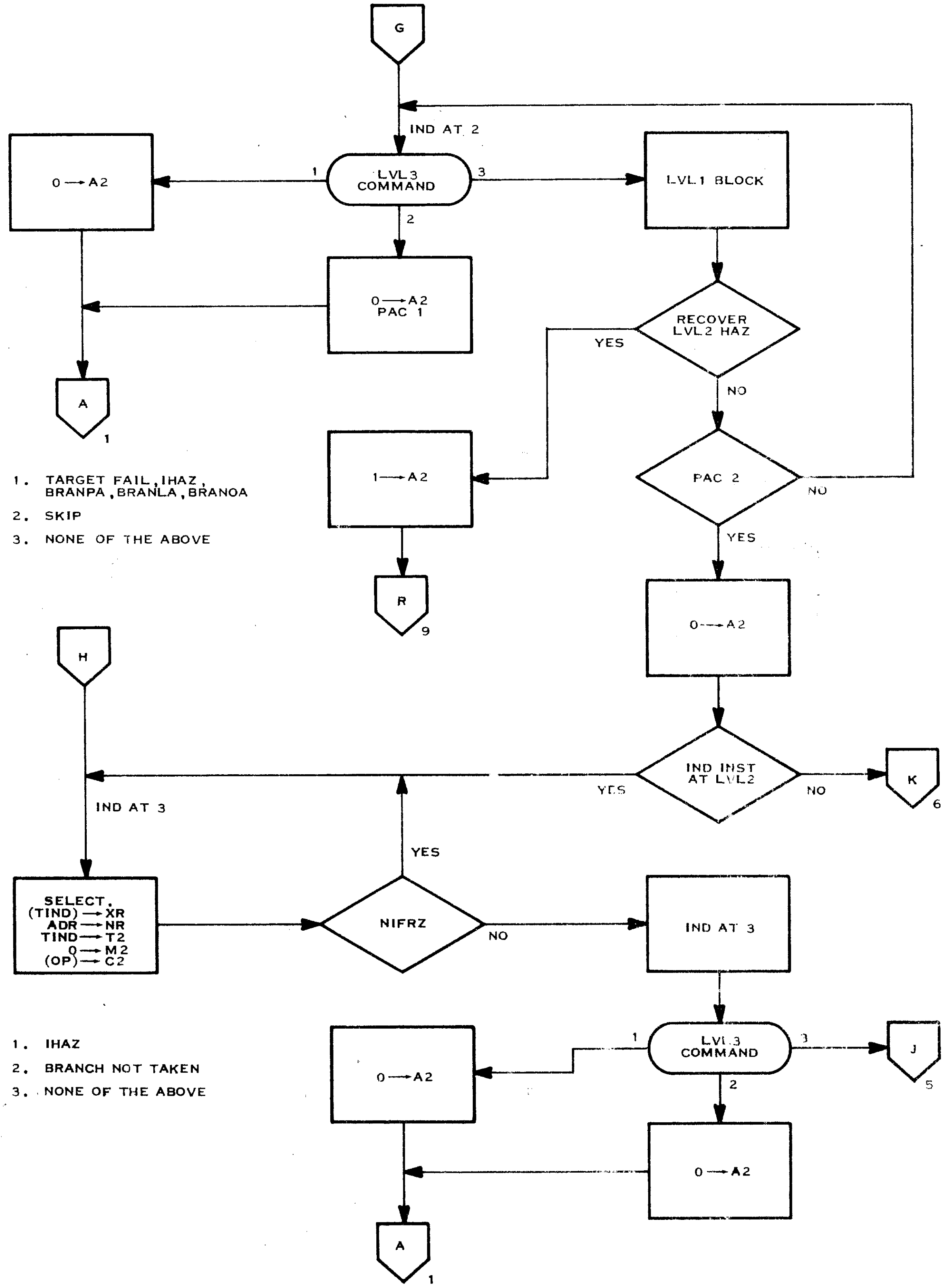
(B)127598 (2/9)

Figure 4-22. Level 1 Controller Flowchart (Sheet 2 of 9)



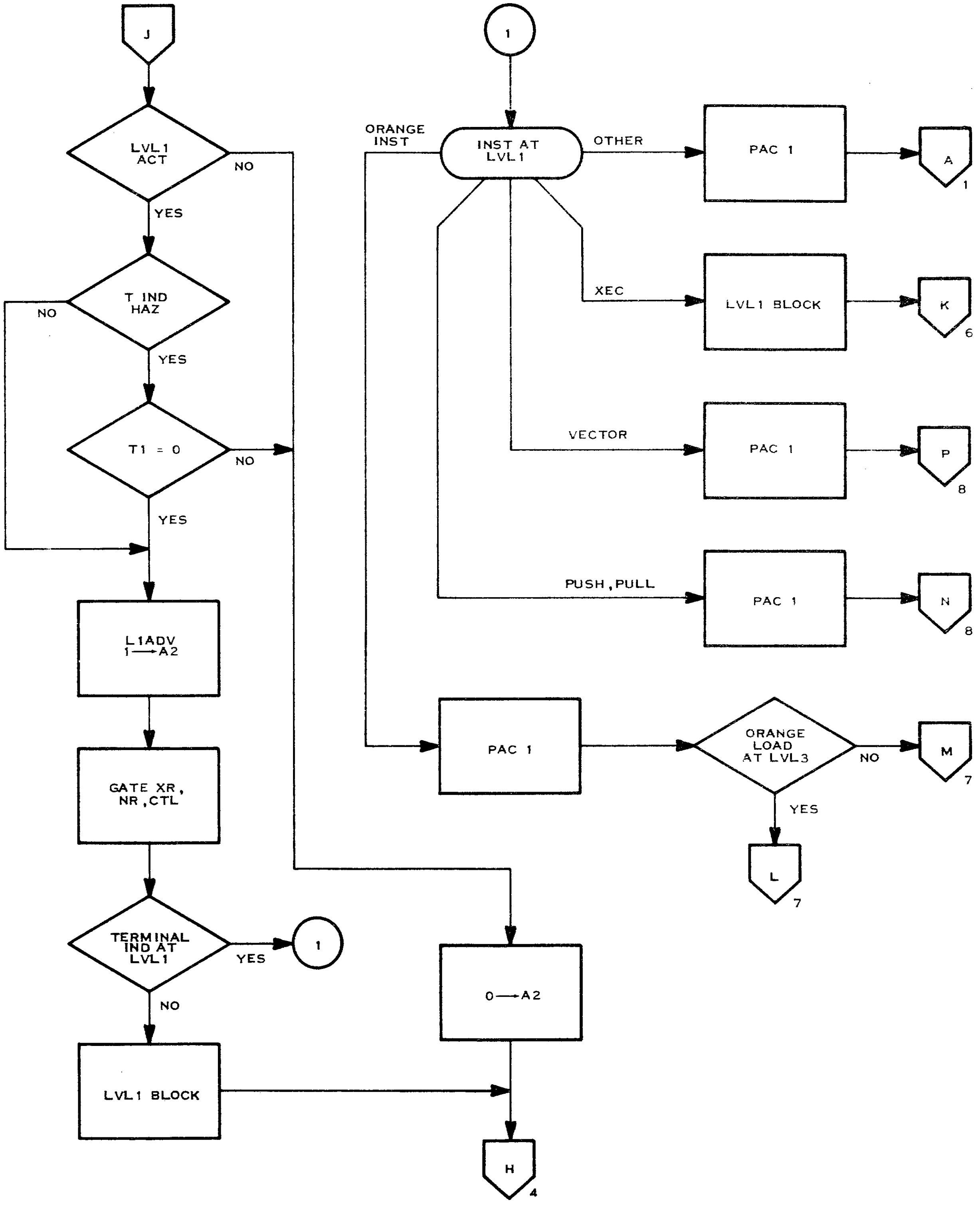
(B)127598 (3/9)

Figure 4-22. Level 1 Controller Flowchart (Sheet 3 of 9)



(B)127598 (4/9)

Figure 4-22. Level 1 Controller Flowchart (Sheet 4 of 9)



(B)127598 (5/9)

Figure 4-22. Level 1 Controller Flowchart (Sheet 5 of 9)

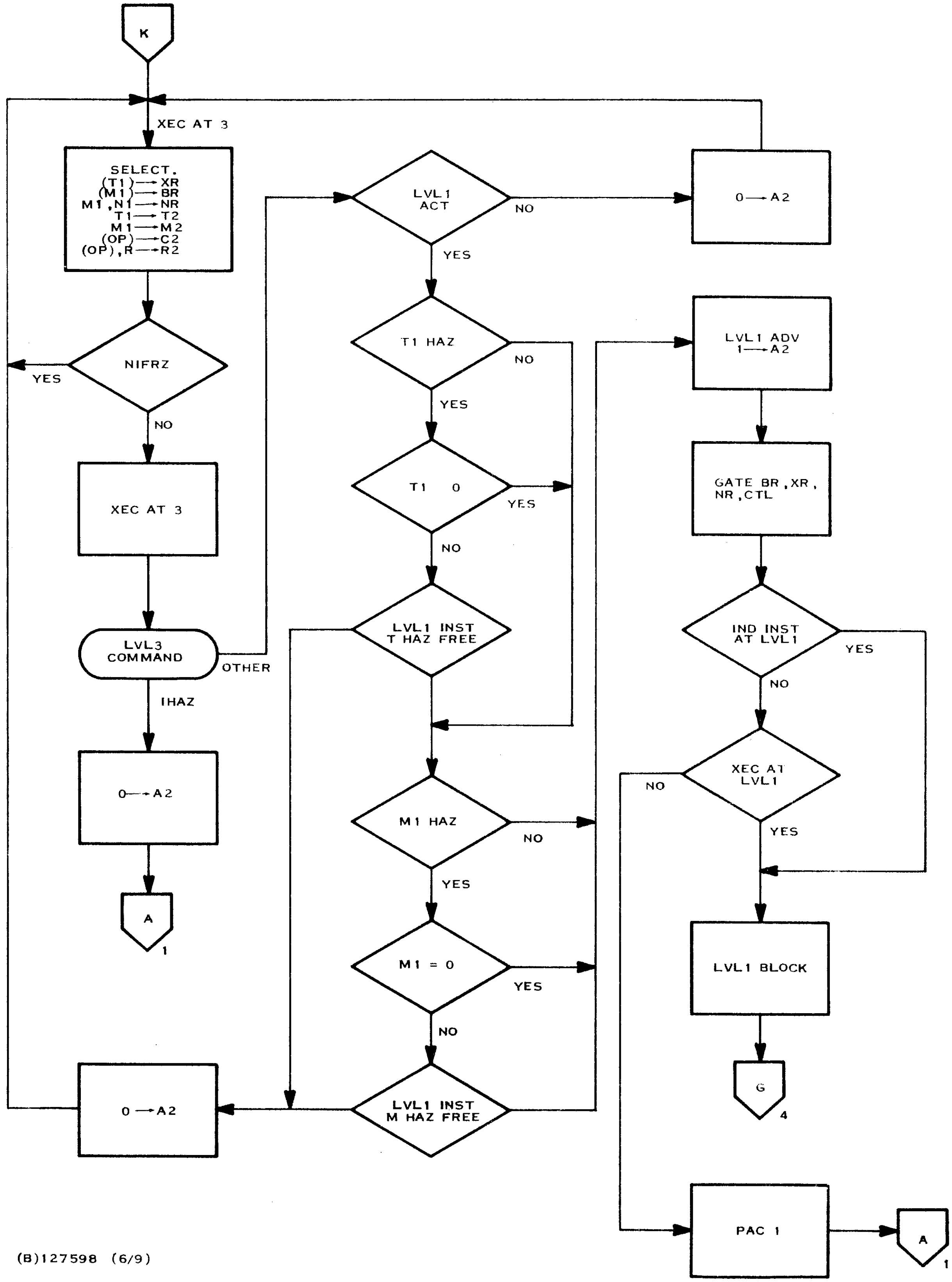
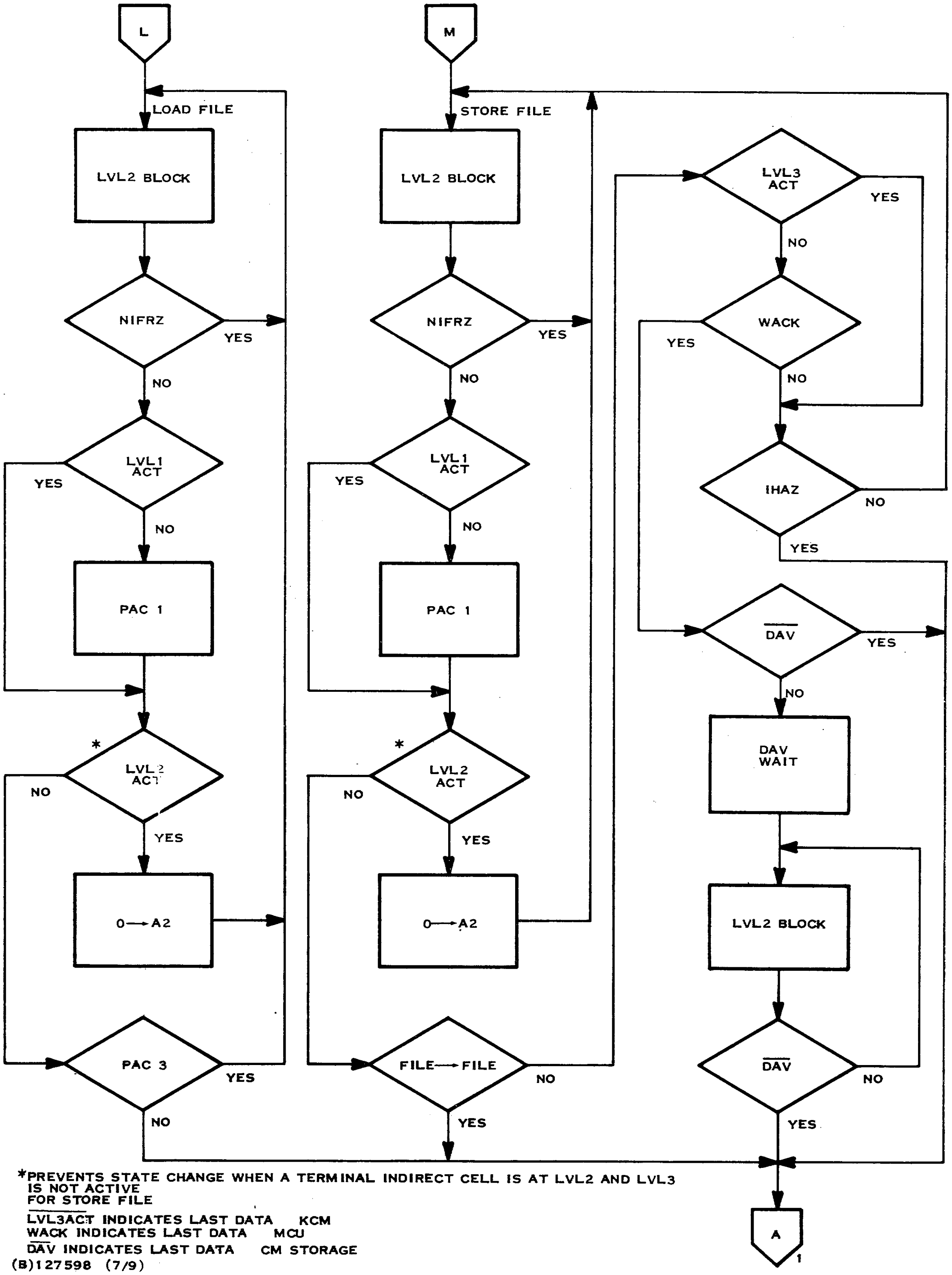


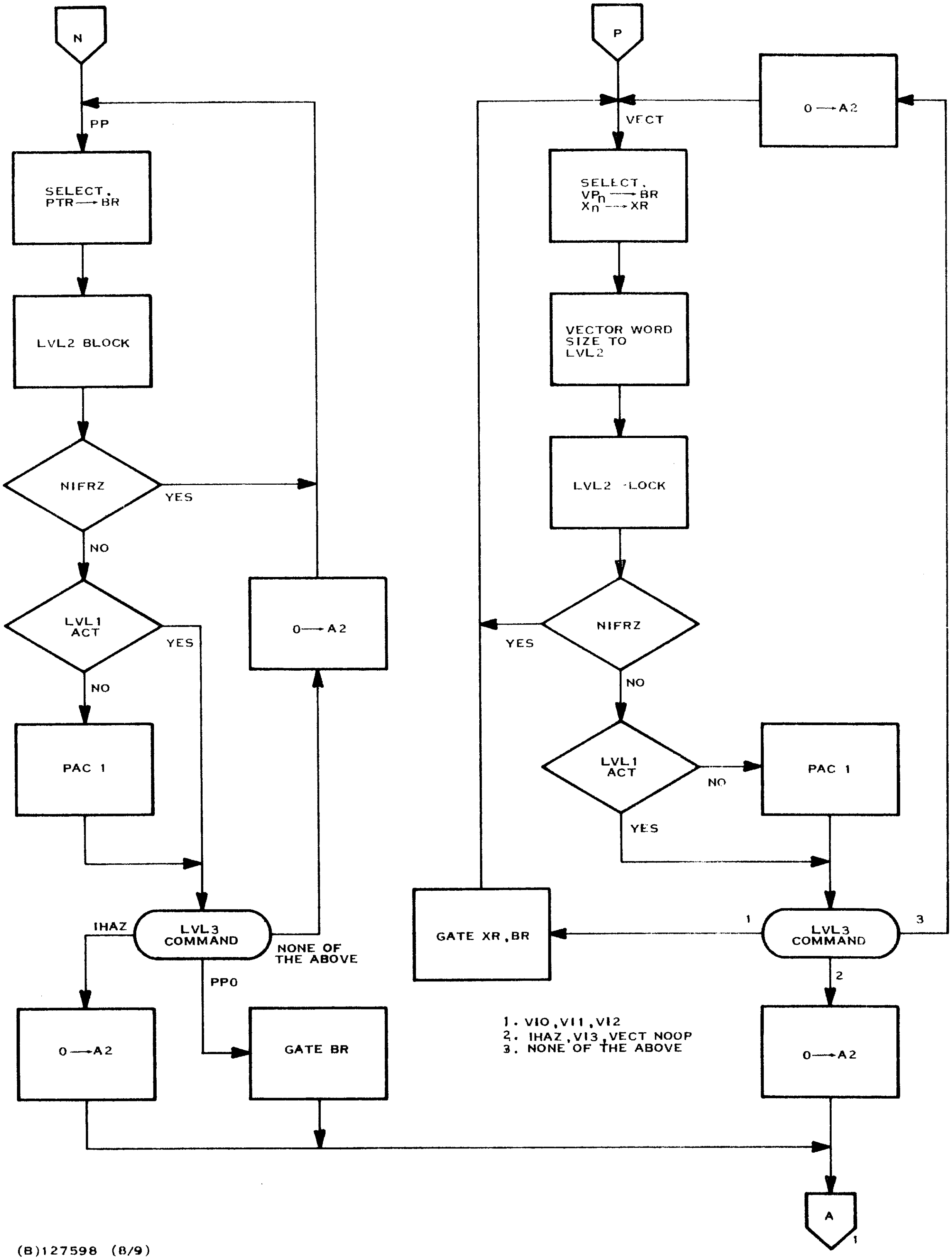
Figure 4-22. Level 1 Controller Flowchart (Sheet 6 of 9)



*PREVENTS STATE CHANGE WHEN A TERMINAL INDIRECT CELL IS AT LVL2 AND LVL3 IS NOT ACTIVE FOR STORE FILE

$\overline{\text{LVL3ACT}}$ INDICATES LAST DATA KCM
WACK INDICATES LAST DATA MCU
$\overline{\text{DAV}}$ INDICATES LAST DATA CM STORAGE
(B)127598 (7/9)

Figure 4-22. Level 1 Controller Flowchart (Sheet 7 of 9)



(B)127598 (8/9)

Figure 4-22. Level 1 Controller Flowchart (Sheet 8 of 9)

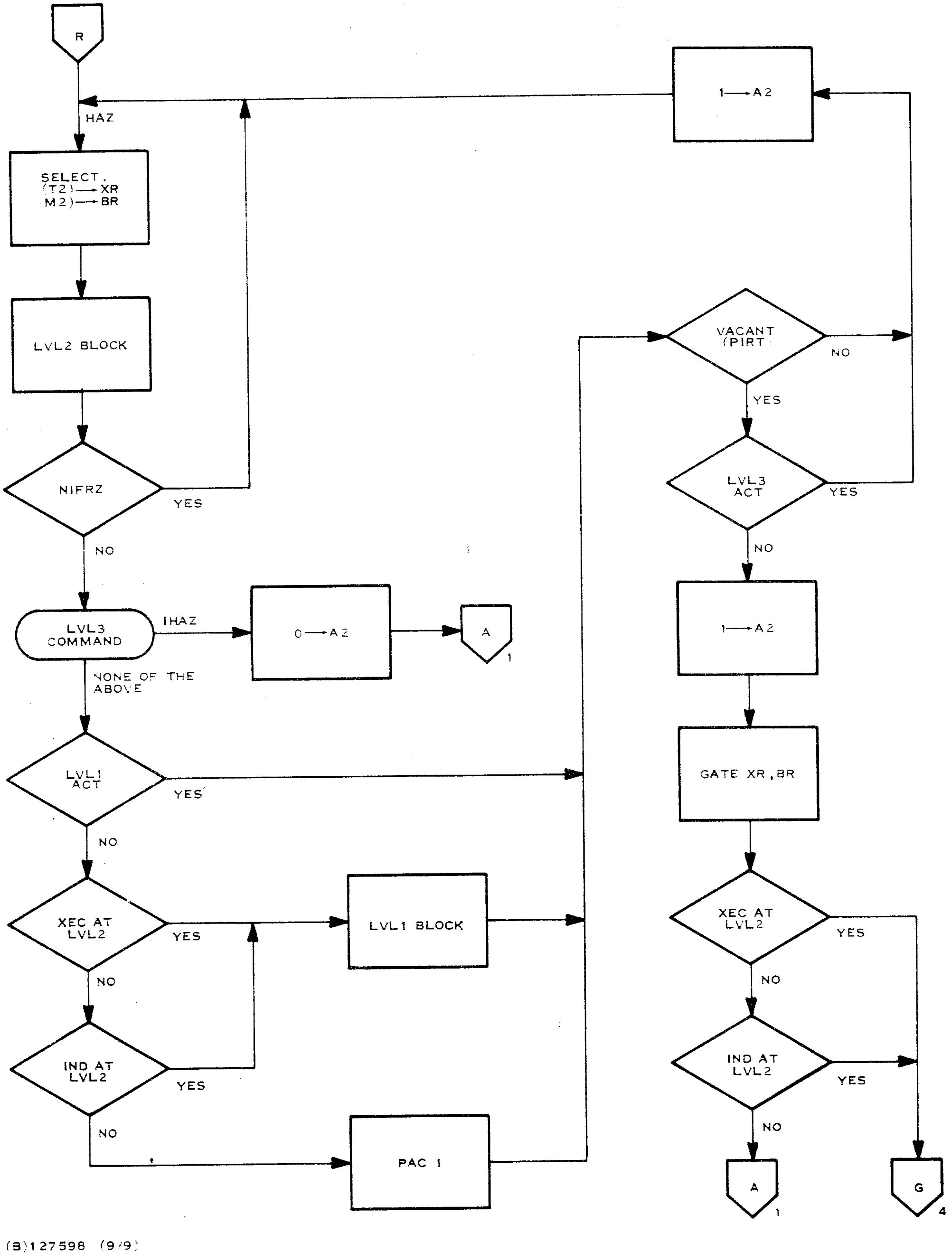
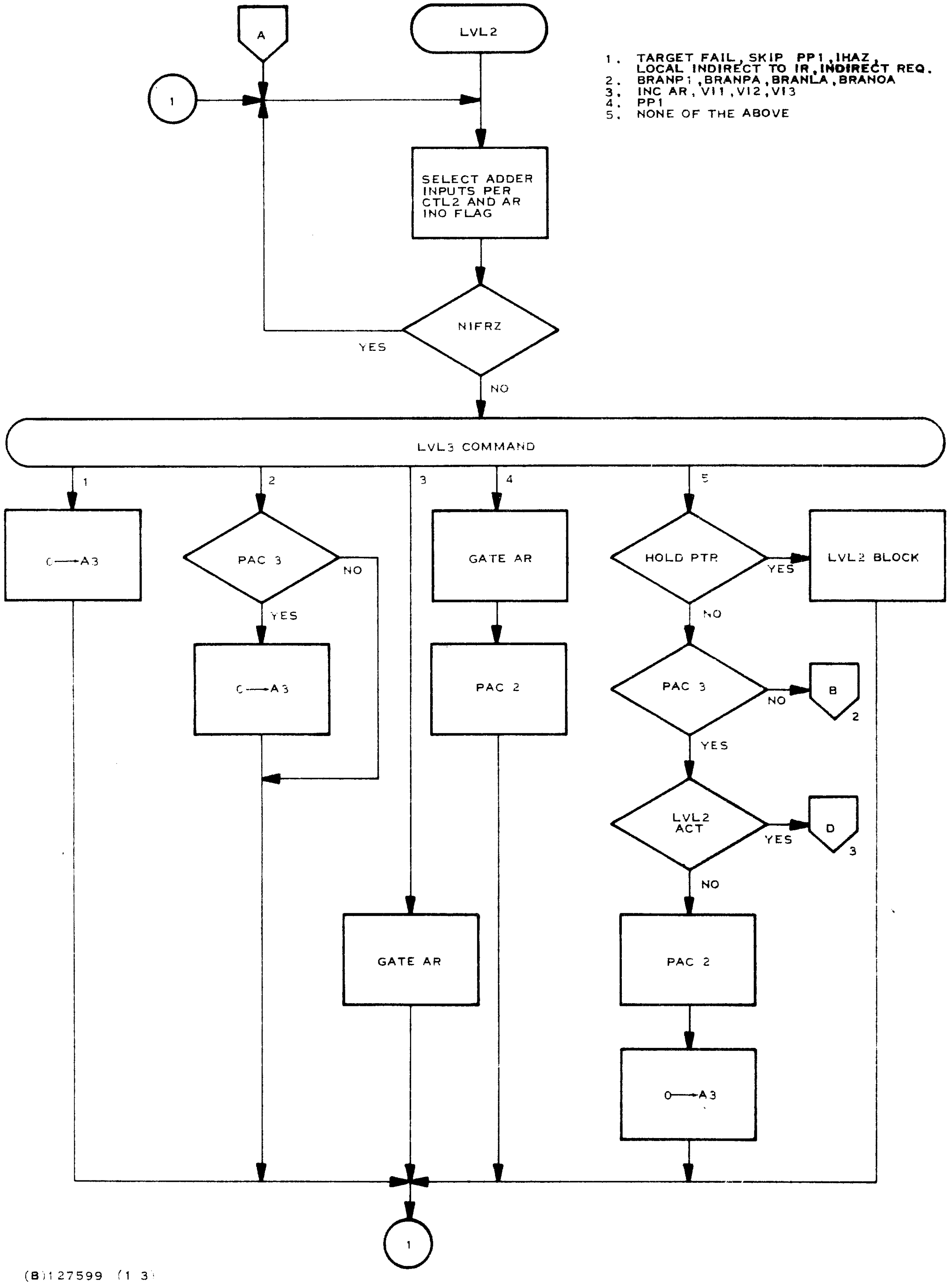


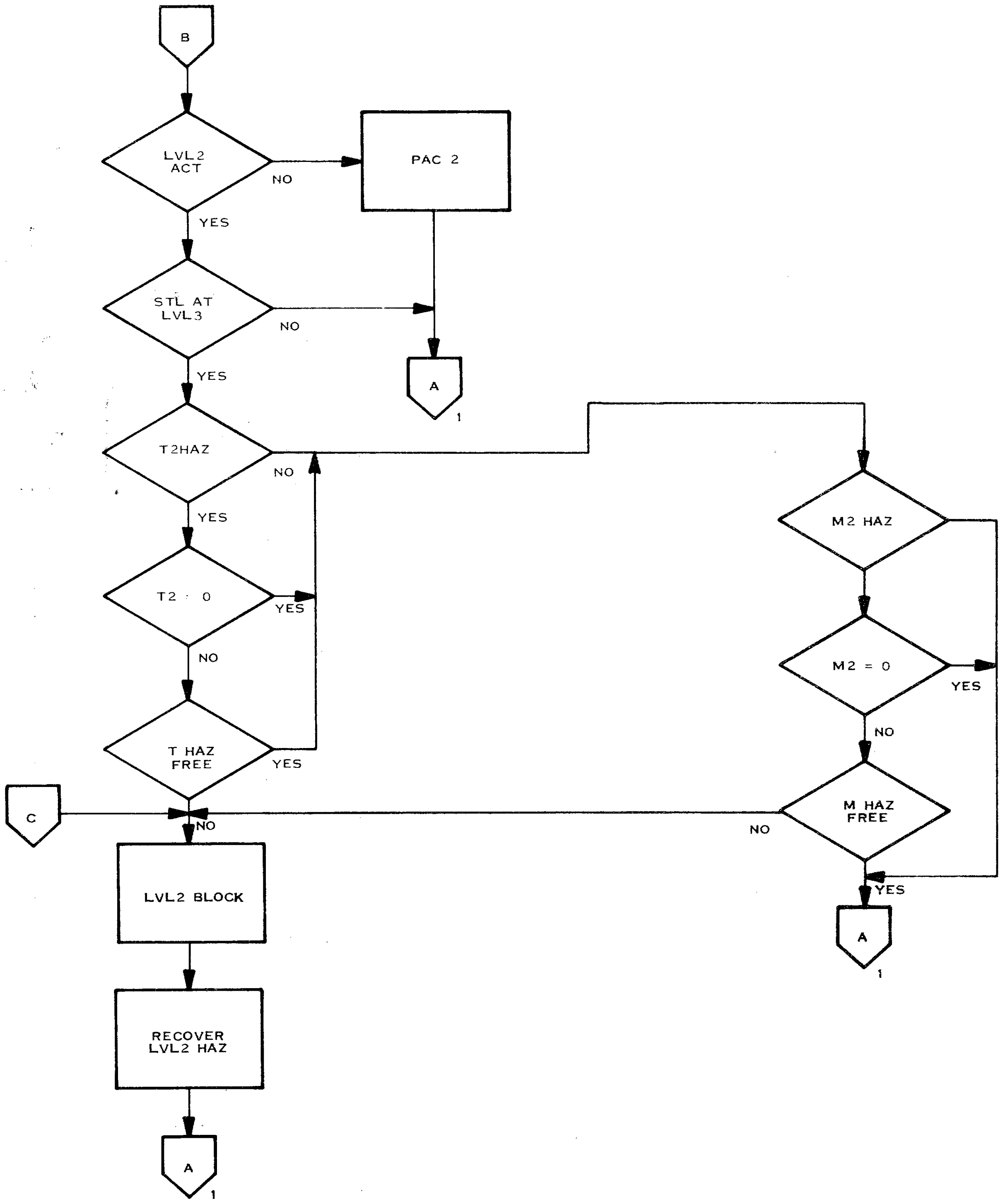
Figure 4-22. Level 1 Controller Flowchart (Sheet 9 of 9)



1. TARGET FAIL, SKIP PP1, IHAZ, LOCAL INDIRECT TO IR, INDIRECT REQ.
2. BRANP1, BRANPA, BRANLA, BRANOA
3. INC AR, VI1, VI2, VI3
4. PP1
5. NONE OF THE ABOVE

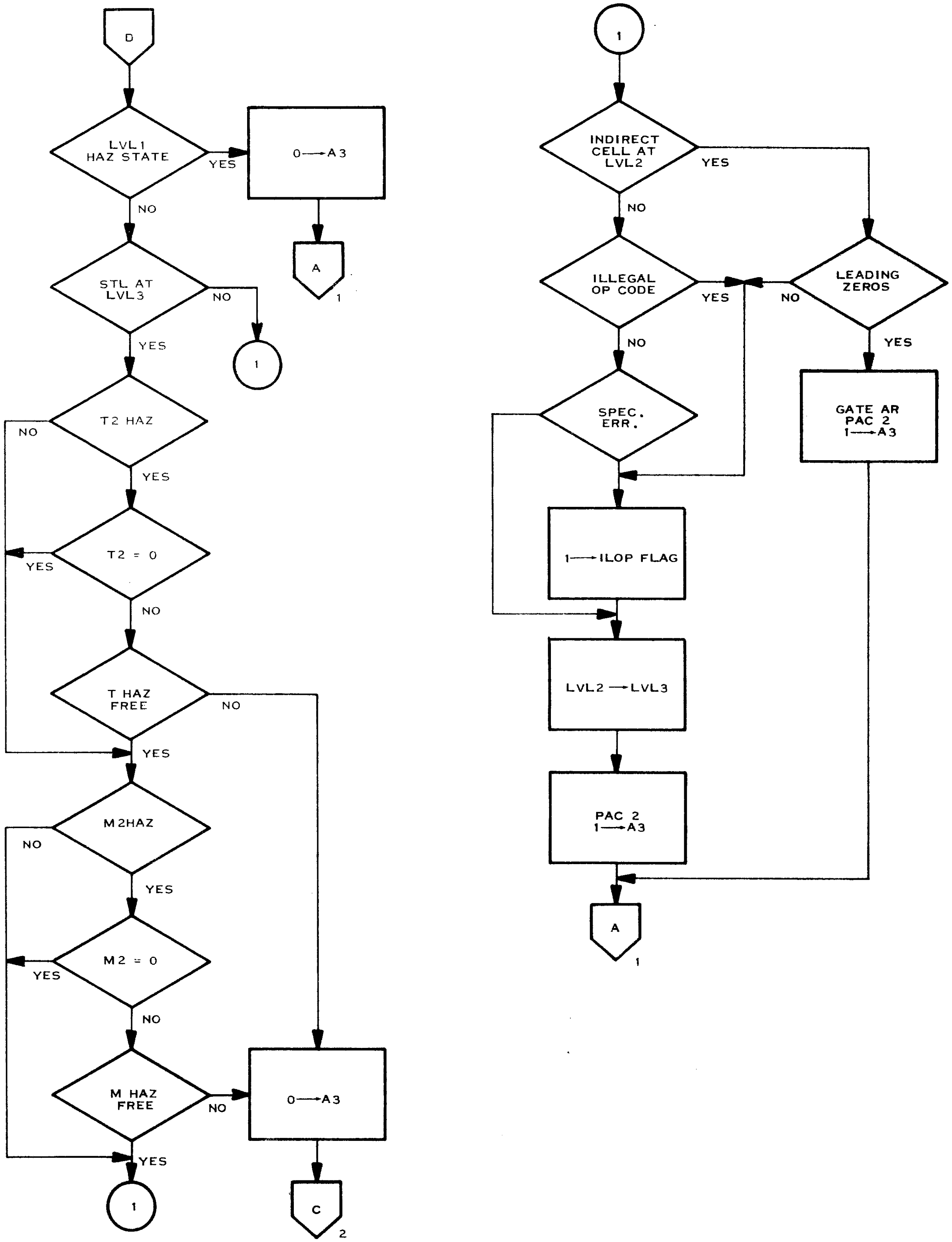
(B)127599 (1 3)

Figure 4-23. Level 2 Controller Flowchart (Sheet 1 of 3)



(B)127599 (2/3)

Figure 4-23. Level 2 Controller Flowchart (Sheet 2 of 3)



(B)127599 (3/3)

Figure 4-23. Level 2 Controller Flowchart (Sheet 3 of 3)

Scalar Op Code Map and Associated Controllers/Registers

OP BITS 0–3

OP BITS 4–7	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LRL^{11}	STZ^{9}	LN^{11}	A^{13}	AI^{10}	A(B)^{13}	AI(B)^{10}	ISE^{4}	MCP^{1}	FLFX^{13}	VECT^{8}	SAH^{10}		AND^{13}	ANDI^{10}
1		LEM^{11}	STZH^{9}	LNH^{11}	AH^{13}	AIH^{10}			ISNE^{4}	BCC^{3}	FLFH^{13}		SAH^{10}		ANDD^{13}	
2		LAM^{11}	SPS^{9}	LNF^{11}	AF^{13}	LEA(B)^{12}	A(XV)^{13}	AI(XV)^{10}	DSE^{4}	INT^{11}	FDFX^{13}				CAND^{13}	CANDI^{10}
3		LAC^{11}	STZD^{9}	LND^{11}	AFD^{13}				DSNE^{4}	PSH^{7}			SAD^{10}		CANDD^{13}	
4		L^{11}	ST^{9}	STN^{9}	AM^{13}	LI(XV)^{12}	D^{13}	DI^{10}	BCLE^{5}	MCW^{1}			SL^{10}		OR^{13}	ORI
5		LLL^{11}	STLL^{9}	STNH^{9}	AMH^{13}	LIH(XV)^{12}	DH^{13}	DIH^{10}	BCG (XV)^{5}	BRC^{3}			SLH^{10}		ORD^{13}	
6		LLA^{10}	STRL^{9}	STNF^{9}	AMF^{13}	LEA(XV)^{12}	DF^{13}		BCLE (XV)^{5}	XEC^{6}			RVS^{10}		COR^{13}	CORI^{10}
7		LD^{11}	STD^{9}	STND^{9}	AMFD^{13}		DFD^{13}		BCG^{5}	PUL^{7}			SLD^{10}		CORD^{13}	
8		L(B)^{11}	ST^{9}	LNM^{11}	S^{13}	SI^{10}	M(B)^{13}	MI(B)^{10}	IBZ(XV)^{5}	BLB^{3}	FXFL^{13}		C^{13}	CI^{10}	XOR^{13}	XORI^{10}
9		LLR^{11}	STLR^{9}	LNMH^{11}	SH^{13}	SIH^{10}			IBNZ^{5}	BLX^{3}	FHFL^{13}		CH^{13}	CIH^{10}	XORD^{13}	
A		XCH^{12}	STOH^{9}	LNMF^{11}	SF^{13}		M(XV)^{13}	MI(XV)^{10}	DBZ (XV)^{5}	FORK^{3}	FXFD^{13}		CF^{13}			
B		LF^{2}	STF^{2}	LNMD^{11}	SFD^{13}				DBNZ (XV)^{5}	JOIN^{3}	FHFD^{13}		CFD^{13}			
C		L(XV)^{11}	ST(XV)^{9}	LM^{11}	SM^{13}	LI(XV)^{12}	M^{13}	MI^{10}	IBZ^{5}	BXEC^{3}	NFX^{11}		SC^{10}		EQC^{13}	EQCI^{10}
D		LRR(XV)^{11}	STRR^{9}	LMH^{11}	SMH^{13}		MH^{13}	MIH^{10}	IBNZ^{5}	BAE^{3}	NFH^{11}		SCH^{10}		EQCD^{13}	
E		LO(XV)^{11}	STO^{9}	LMF^{11}	SMF^{13}		MF^{13}		DBZ^{5}	PB^{3}	SCLK^{9}		C(XV)^{13}	CI(XV)^{10}		
F		LFM^{2}	STFM^{2}	LMD^{11}	SMFD^{13}		MFD^{13}		DBNZ^{5}	MOD^{7}			SCD^{10}			

(A)132348

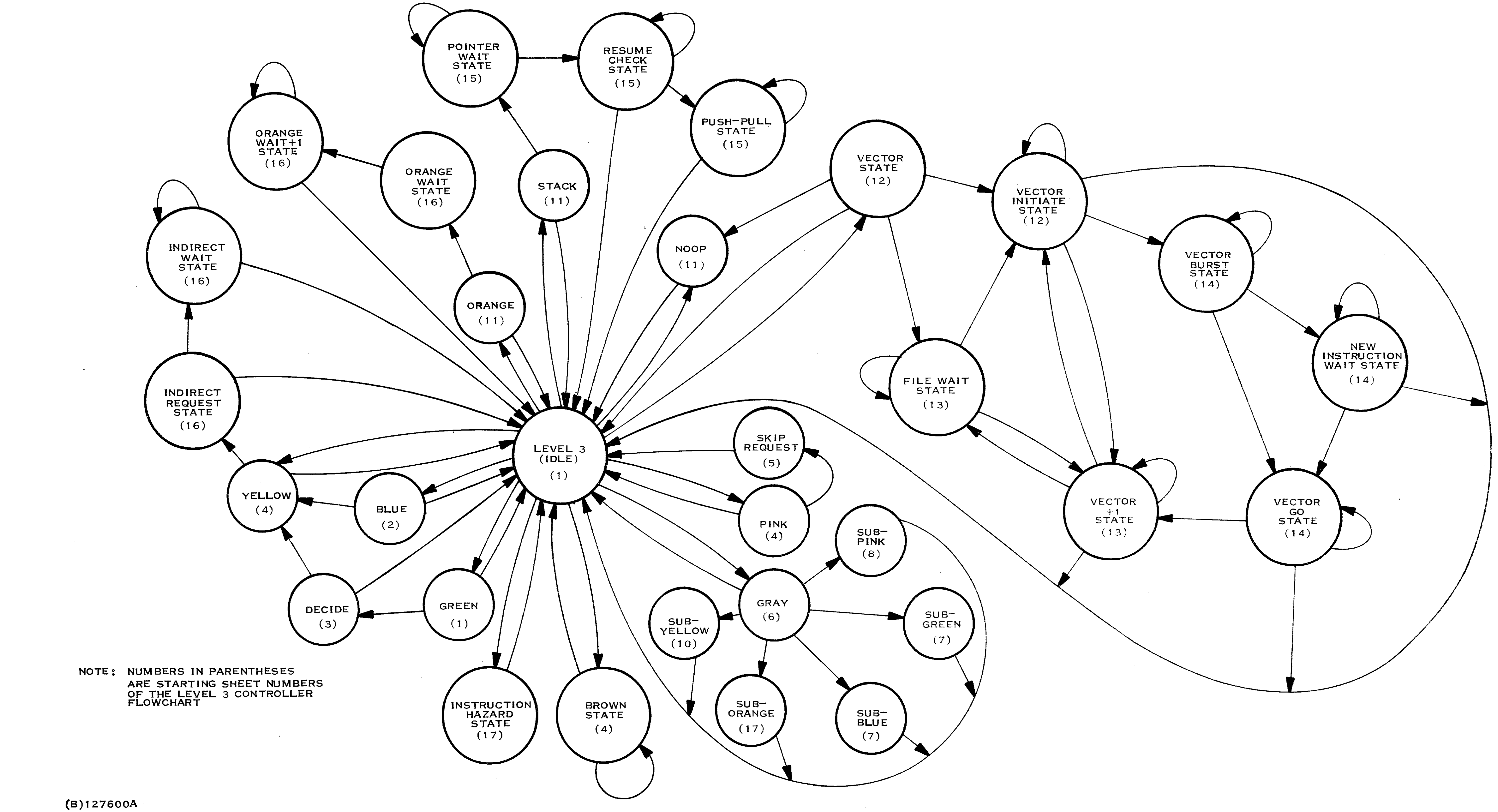


Figure 4-24. Level 3 Controller State Diagram

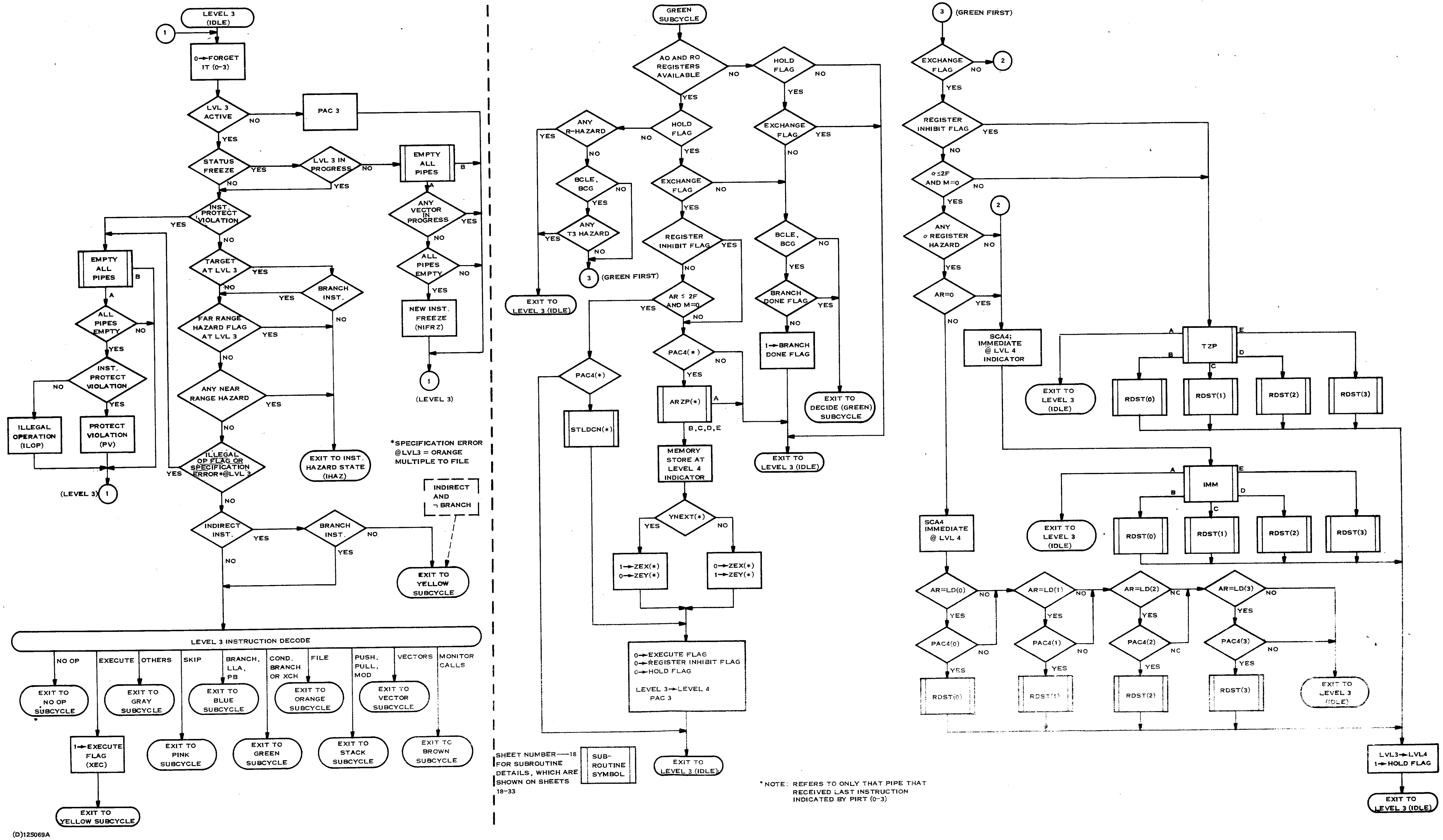


Figure 4-25. Level 3 Controller Flowchart (Sheet 1 of 33)

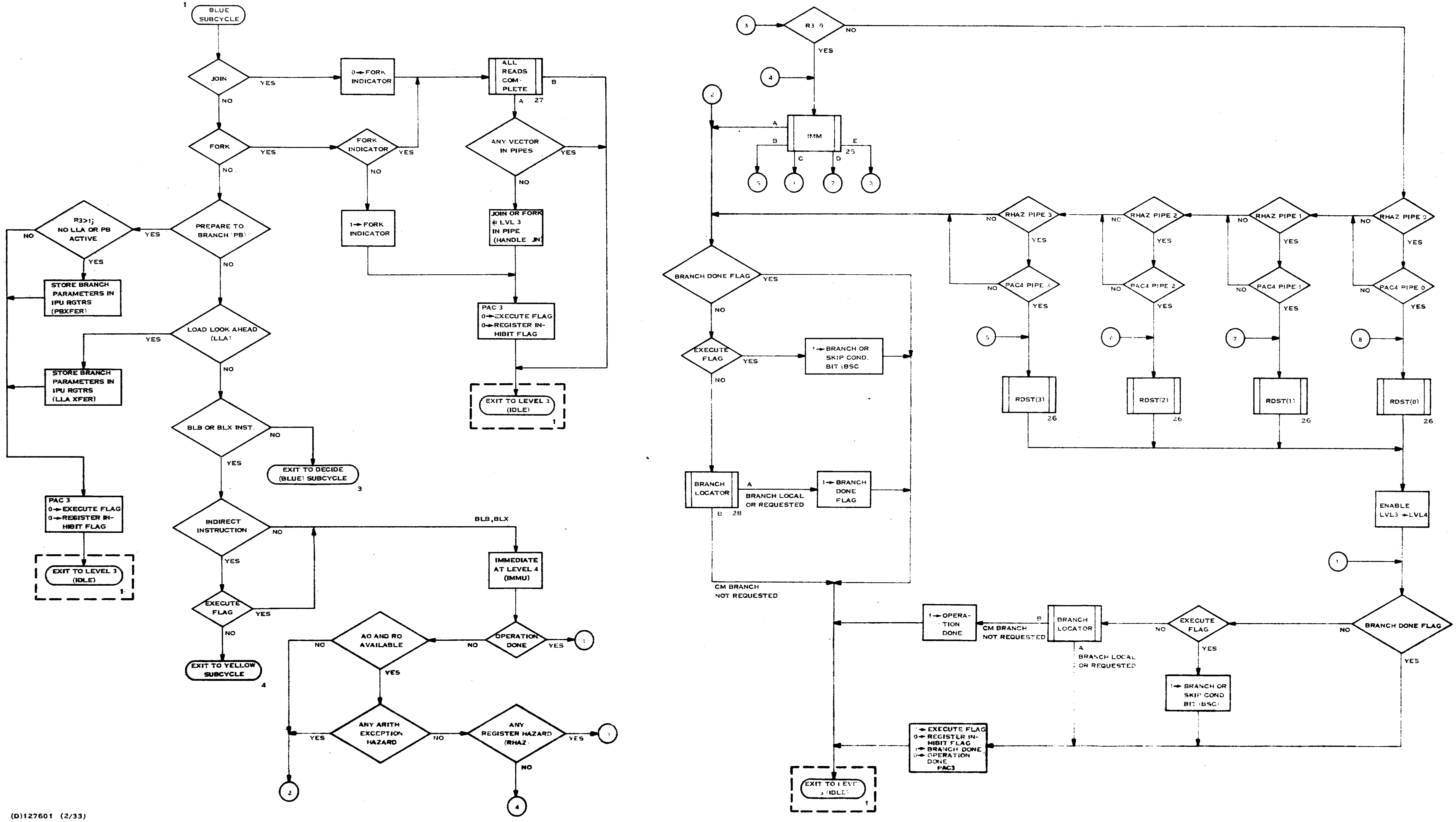
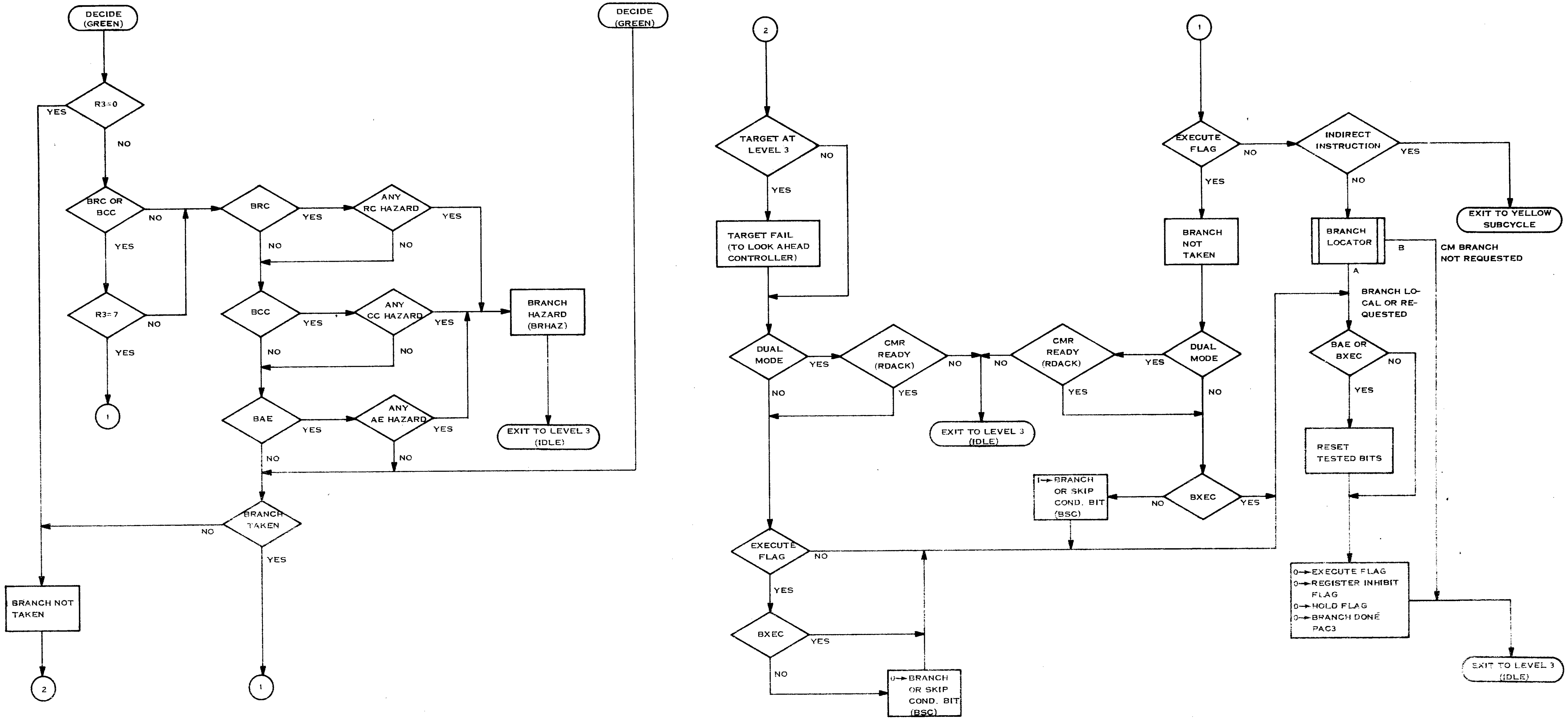
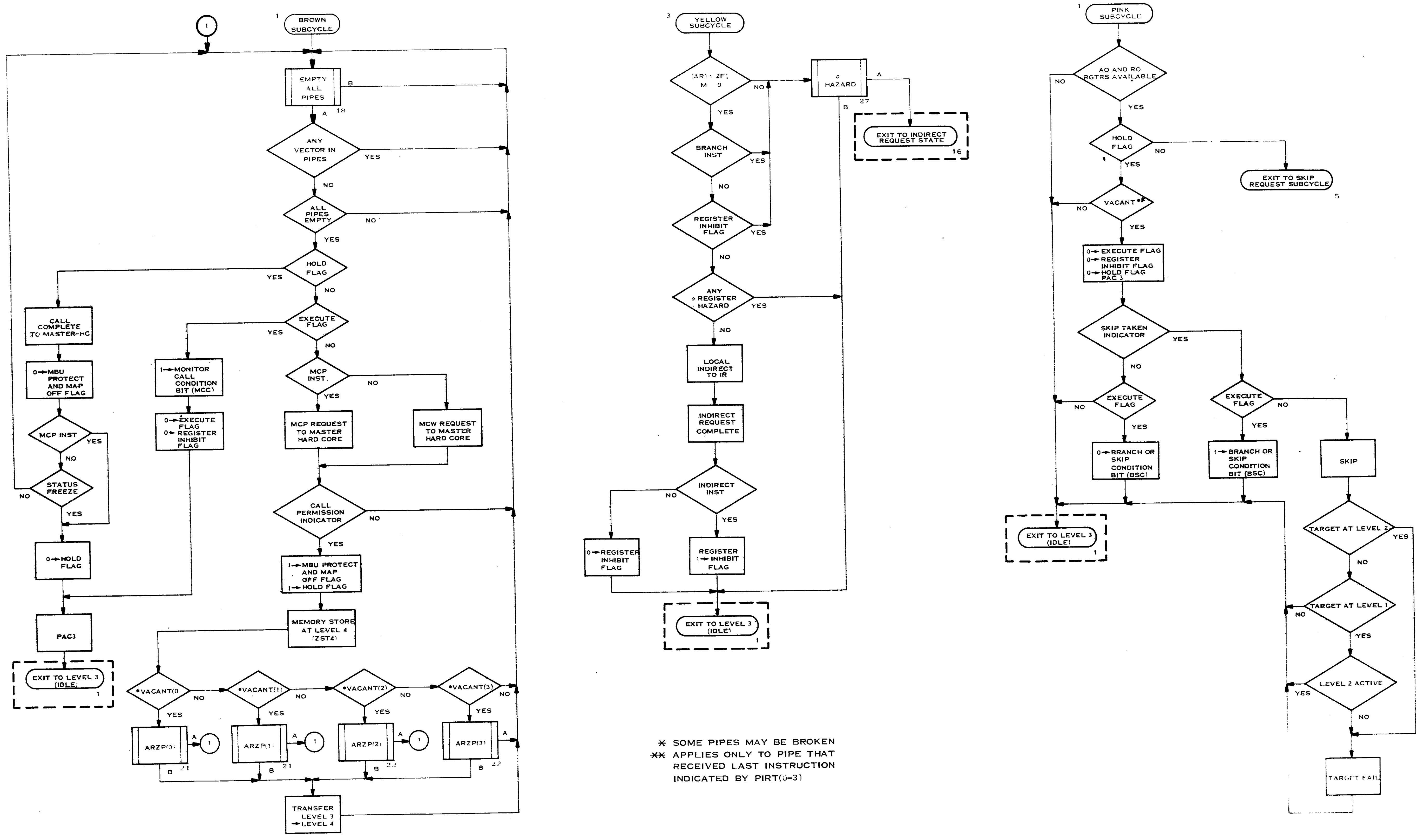


Figure 4-25. Level 3 Controller Flowchart (Sheet 2 of 33)



(D)125075A

Figure 4-25. Level 3 Controller Flowchart (Sheet 3 of 33)



✱ SOME PIPES MAY BE BROKEN
✱✱ APPLIES ONLY TO PIPE THAT RECEIVED LAST INSTRUCTION INDICATED BY PIRT(0-3)

(D) 127601 (4/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 4 of 33)

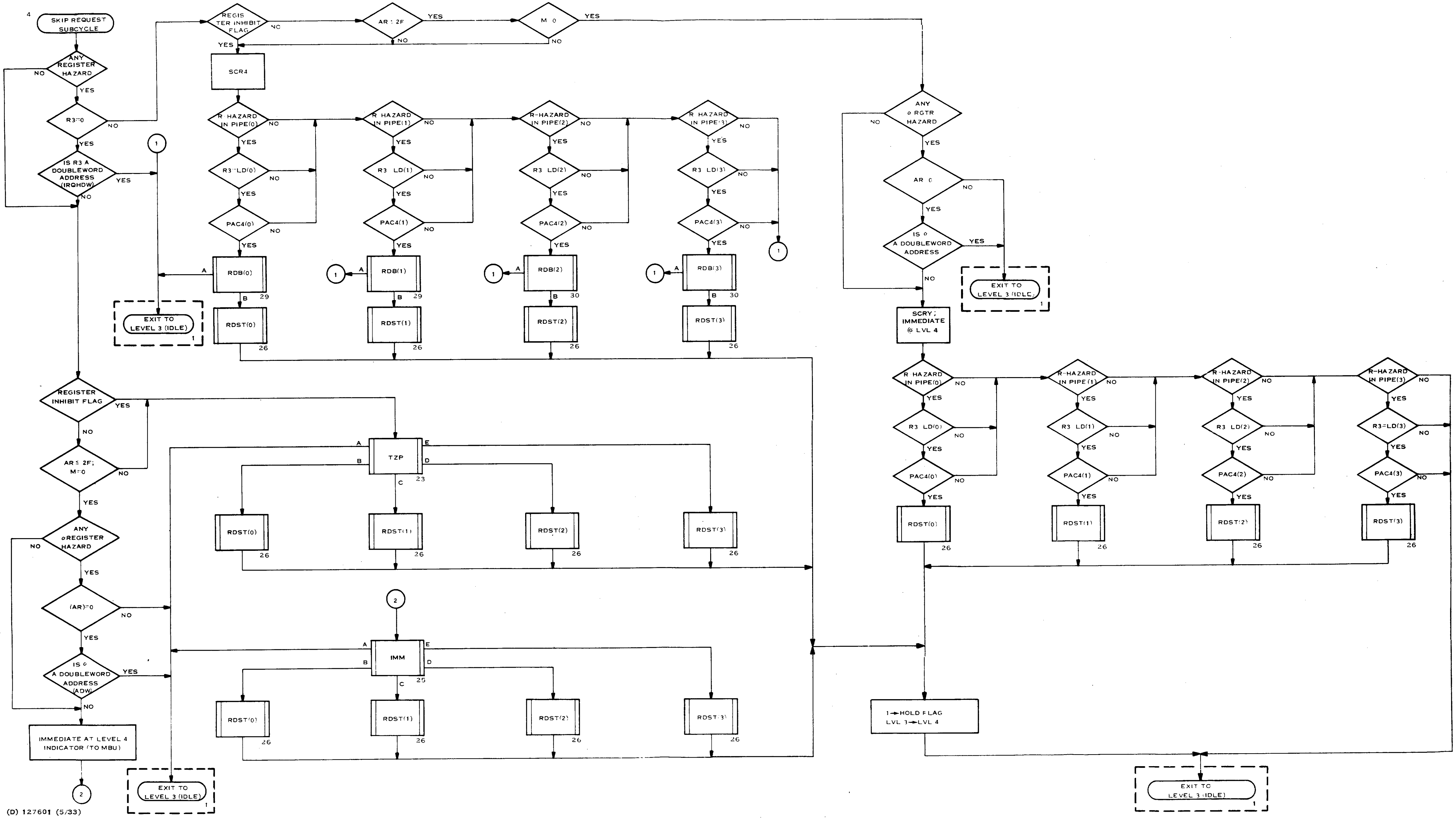
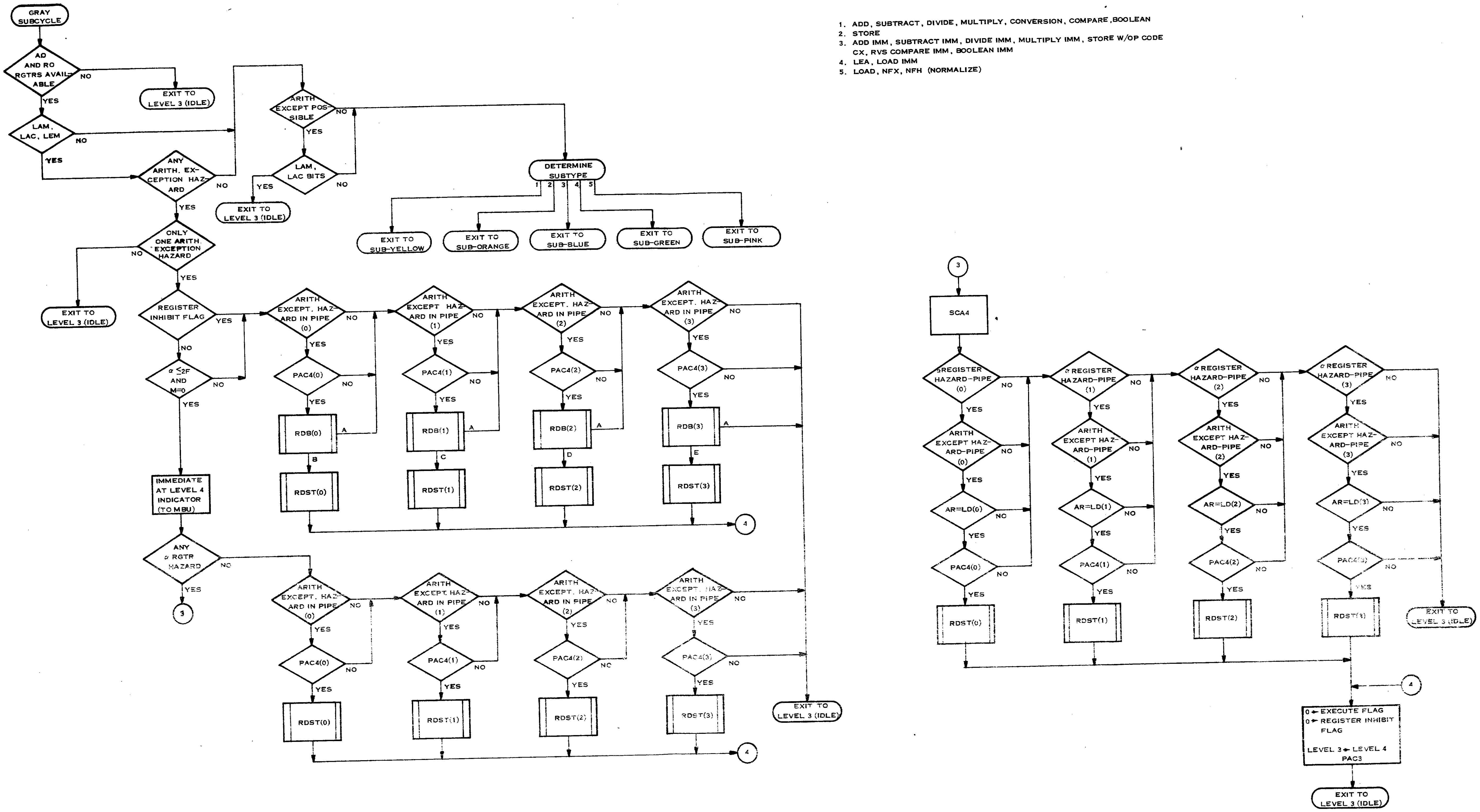


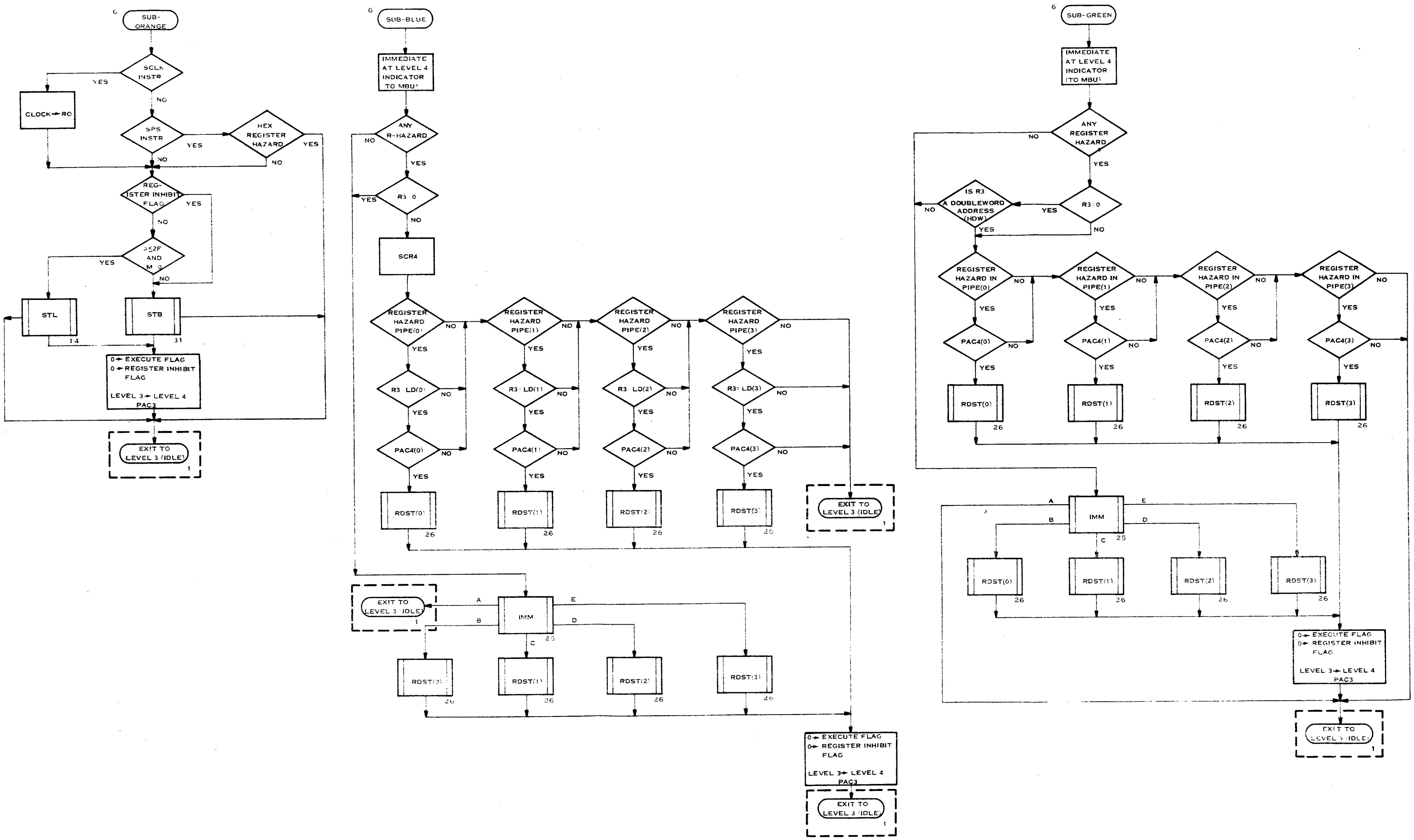
Figure 4-25. Level 3 Controller Flowchart (Sheet 5 of 33)



1. ADD, SUBTRACT, DIVIDE, MULTIPLY, CONVERSION, COMPARE, BOOLEAN
2. STORE
3. ADD IMM, SUBTRACT IMM, DIVIDE IMM, MULTIPLY IMM, STORE W/OP CODE CX, RVS COMPARE IMM, BOOLEAN IMM
4. LEA, LOAD IMM
5. LOAD, NFX, NFH (NORMALIZE)

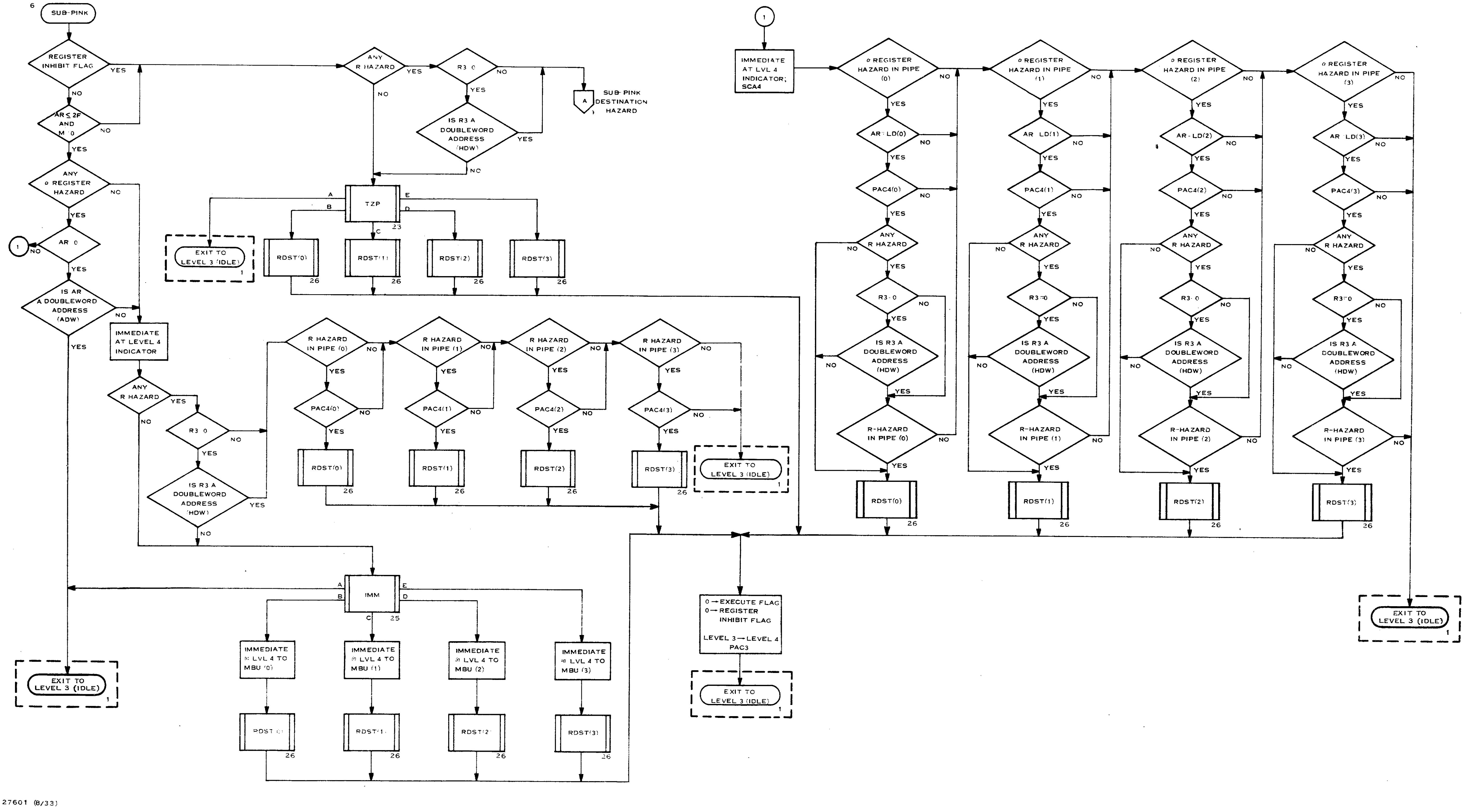
(D)125082A

Figure 4-25. Level 3 Controller Flowchart (Sheet 6 of 33)



(D)127601 (7/73)

Figure 4-25. Level 3 Controller Flowchart (Sheet 7 of 33)



(D)127601 (8/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 8 of 33)

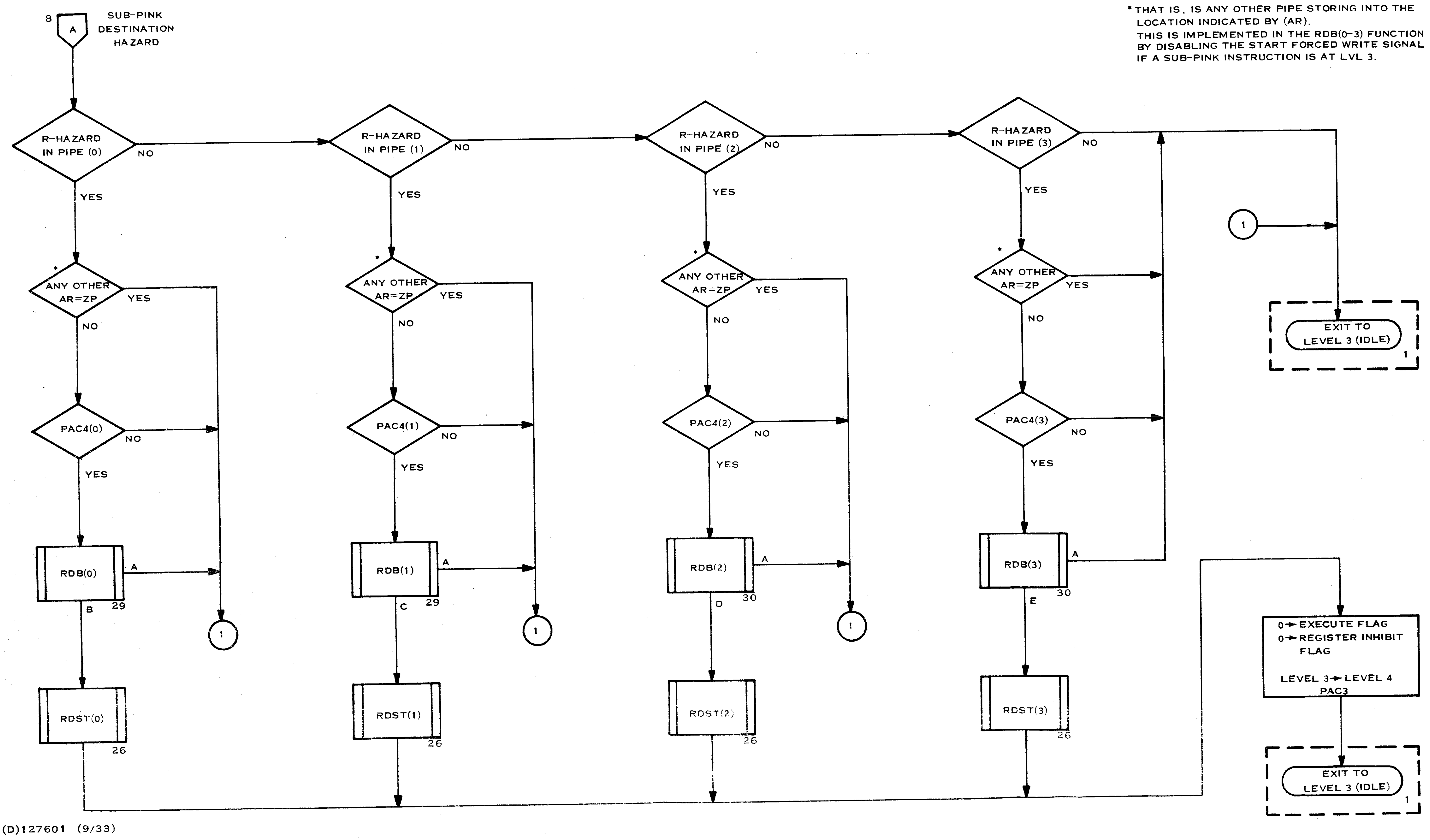


Figure 4-25. Level 3 Controller Flowchart (Sheet 9 of 33)

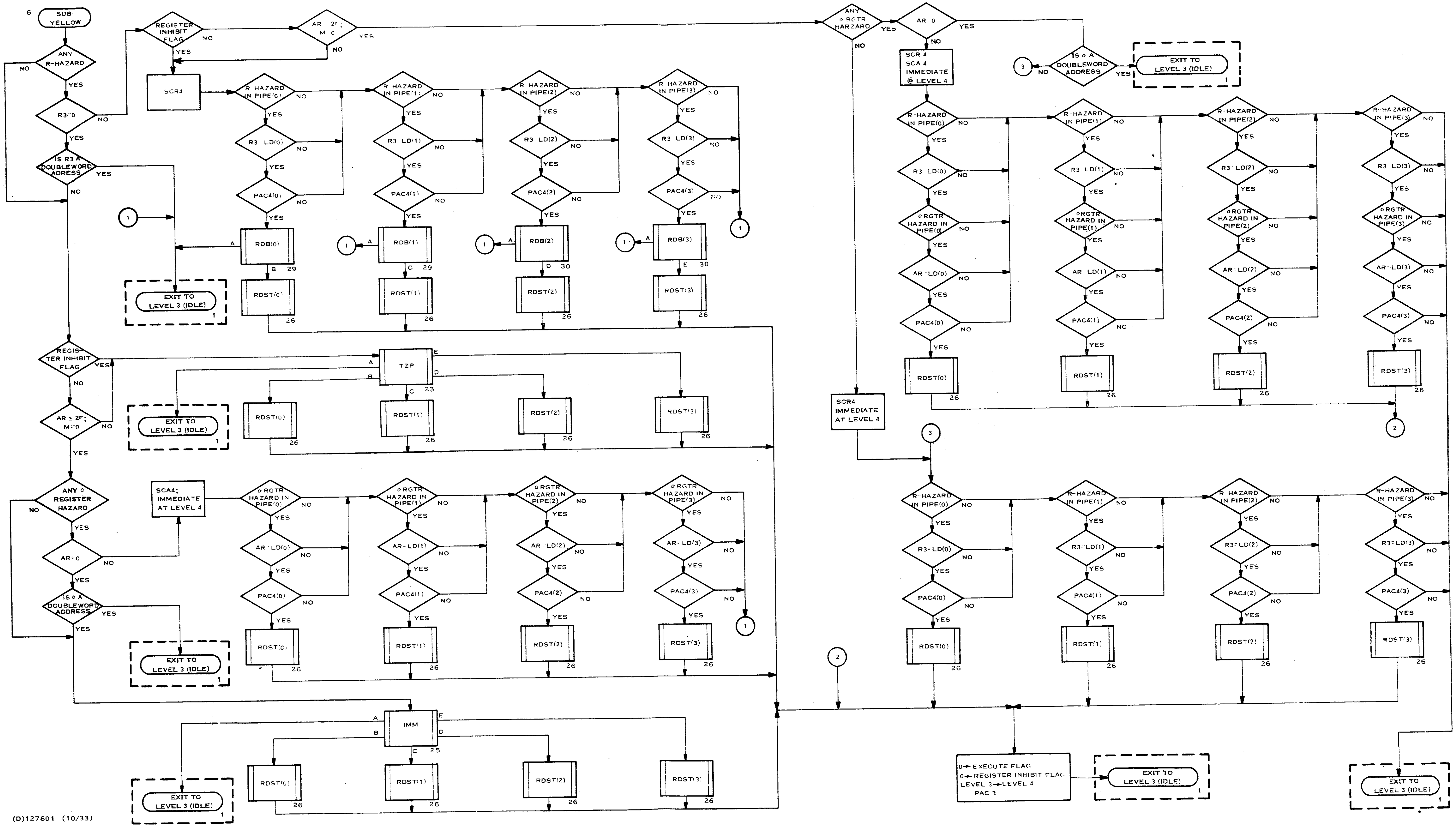


Figure 4-25. Level 3 Controller Flowchart (Sheet 10 of 33)

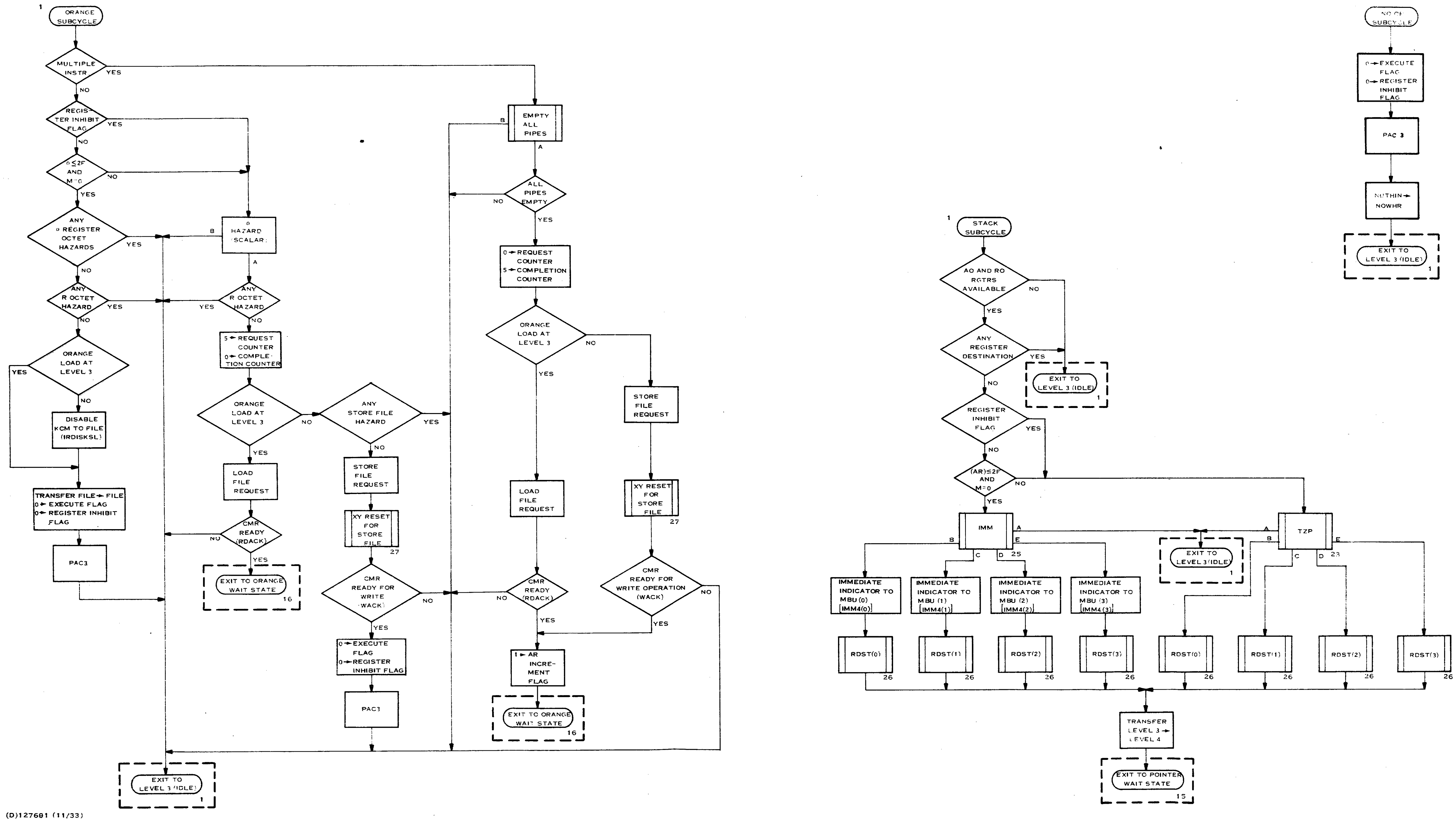
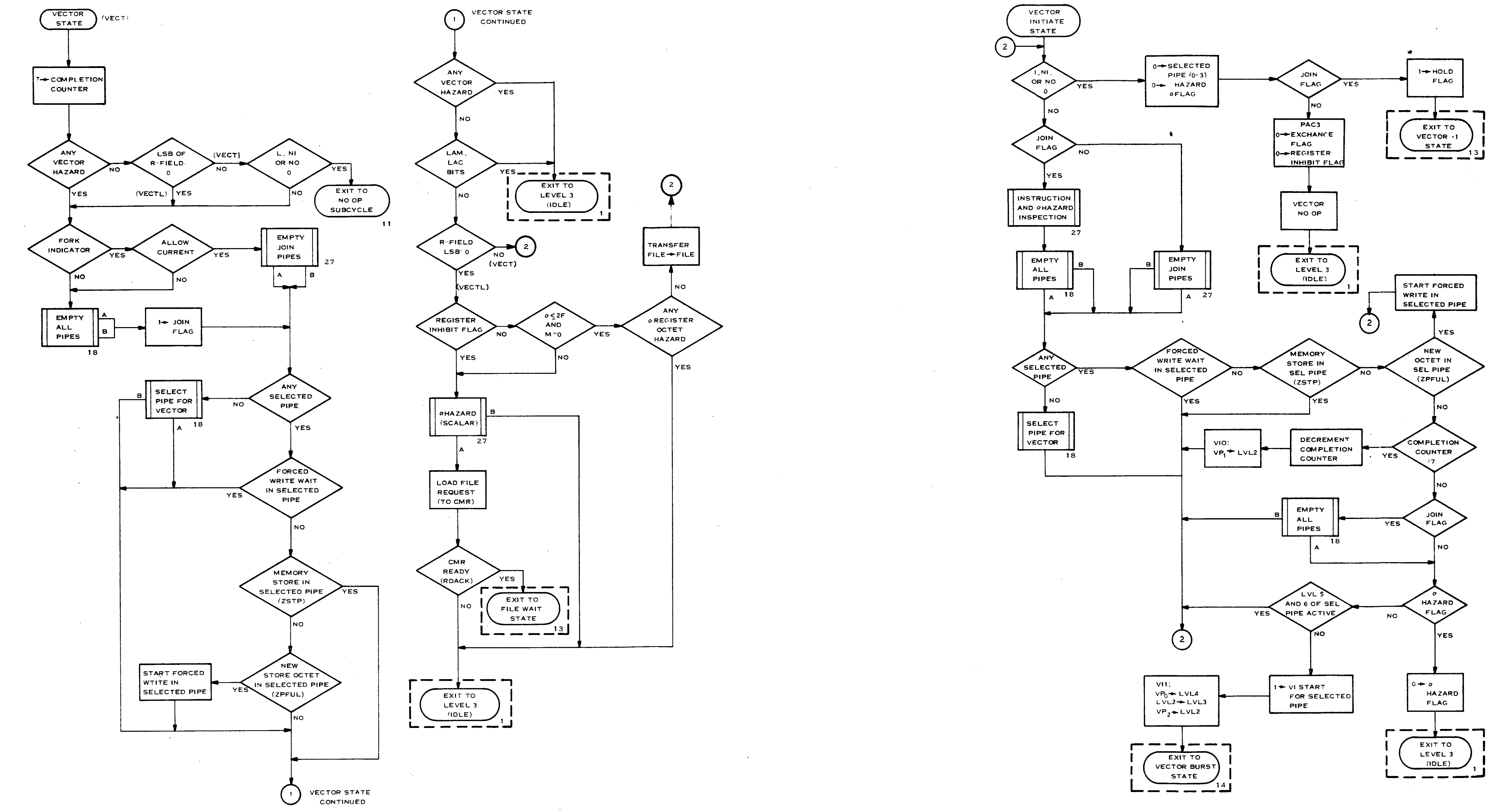


Figure 4-25. Level 3 Controller Flowchart (Sheet 11 of 33)



(D)127601 (12/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 12 of 33)

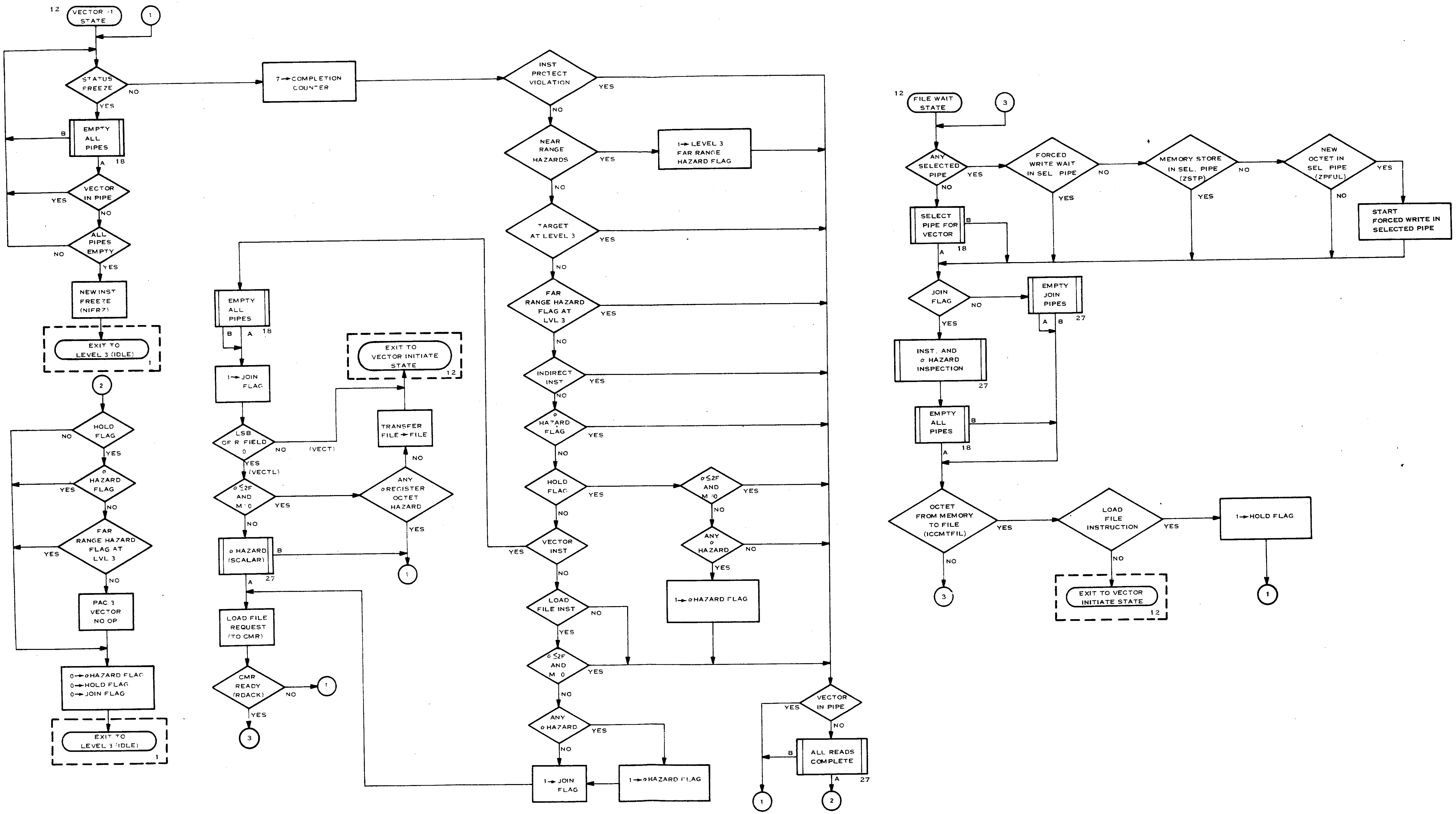
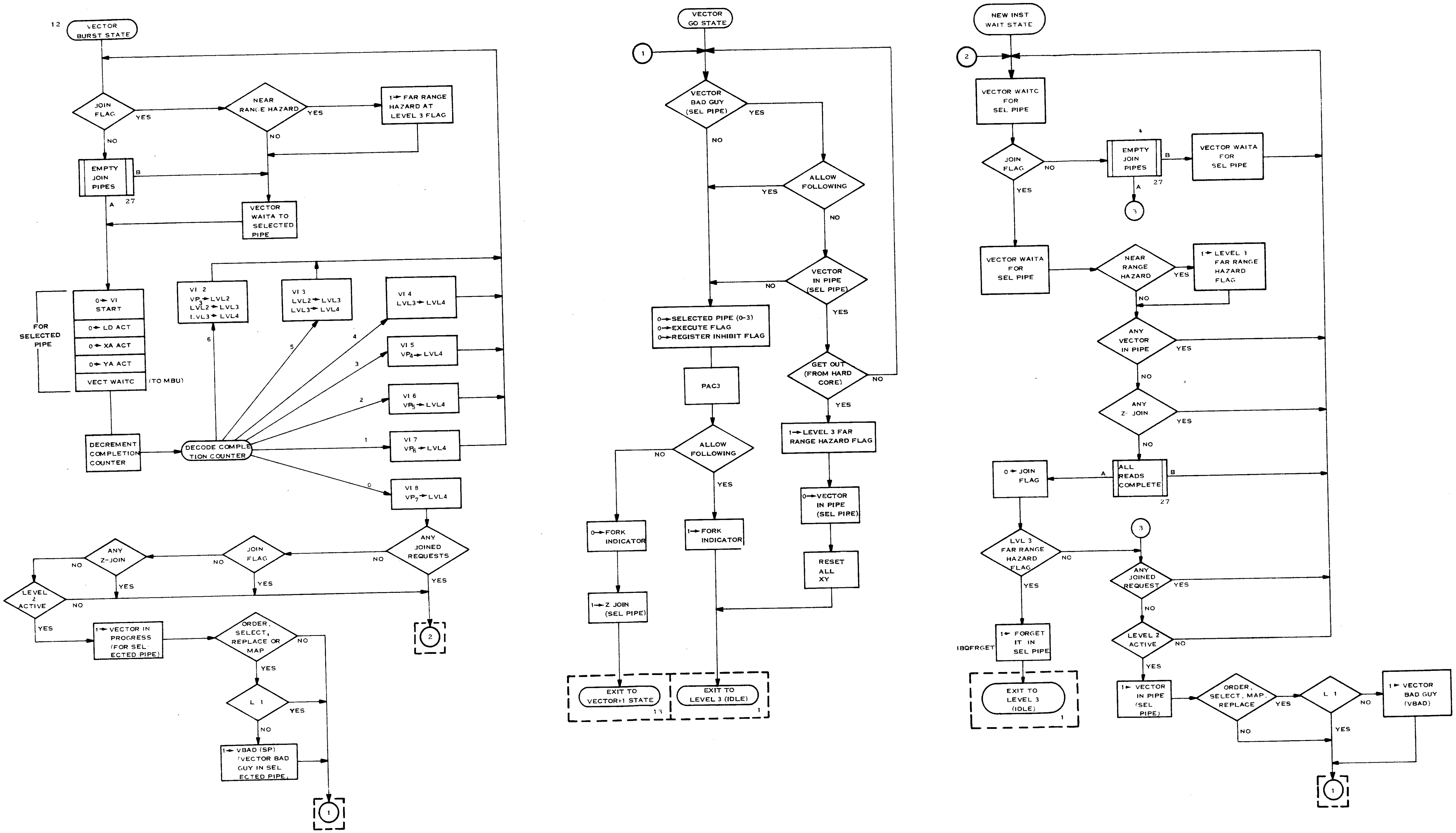
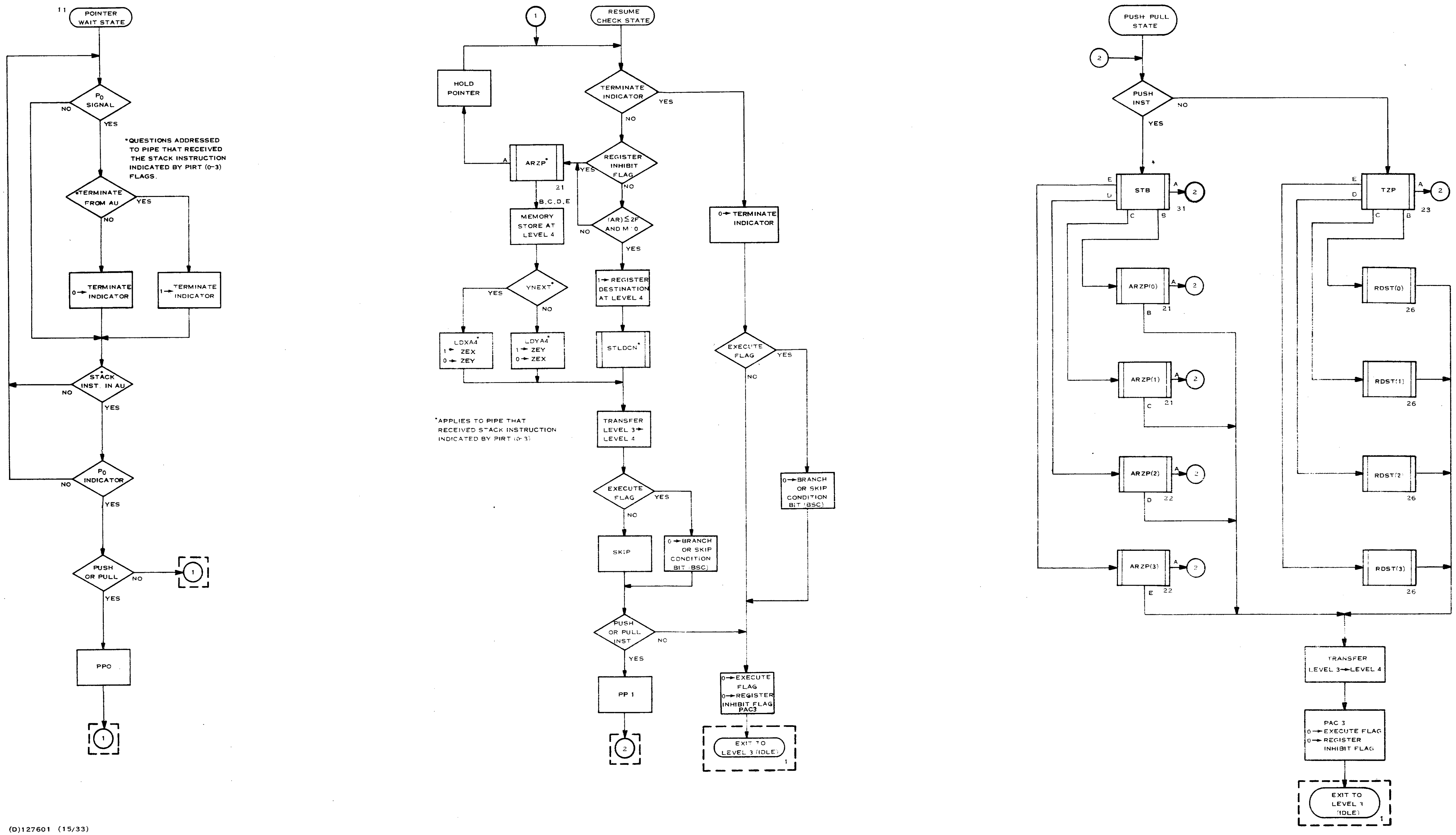


Figure 4-25. Level 3 Controller Flowchart (Sheet 13 of 33)



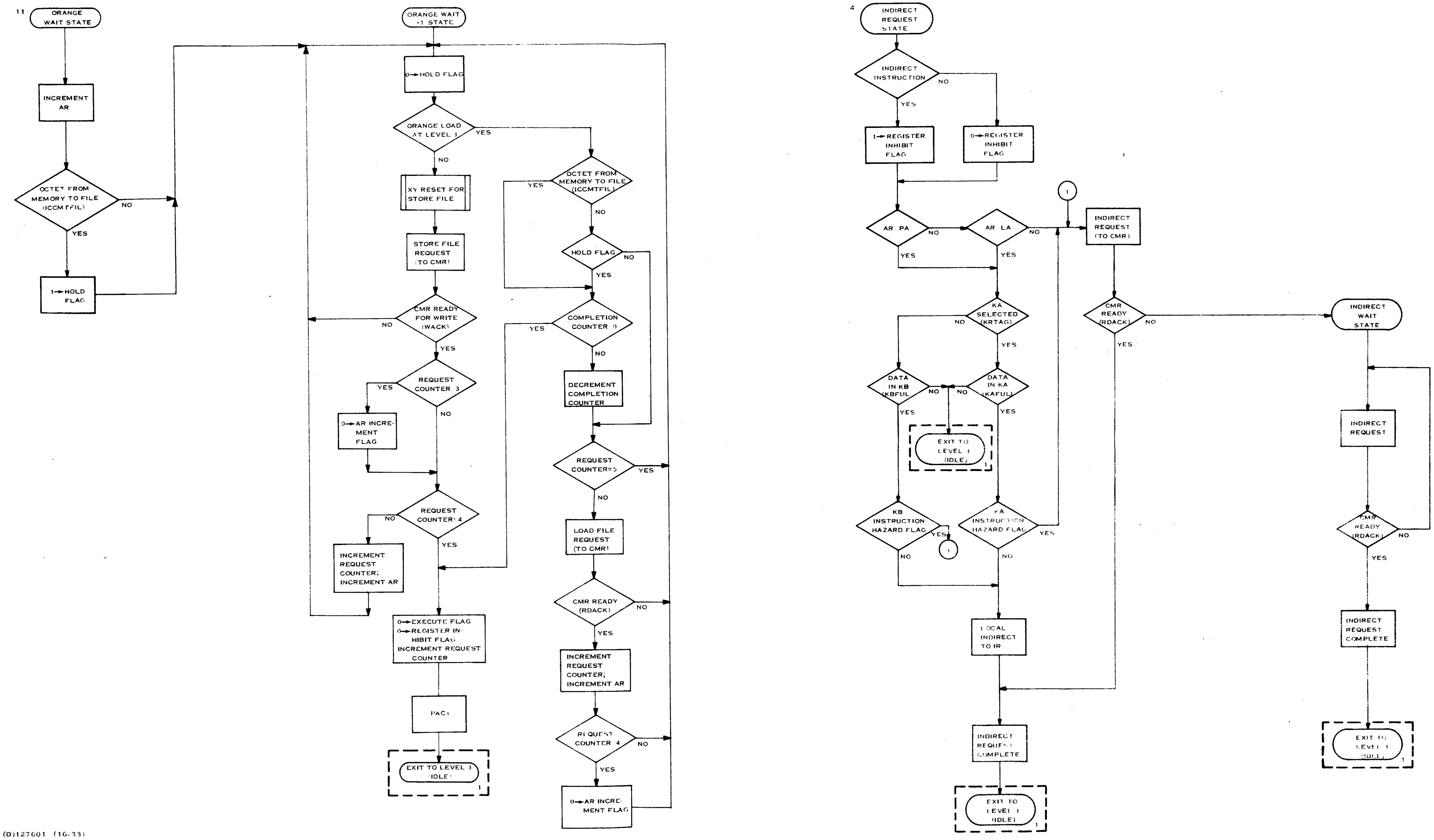
(D)127601 (14/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 14 of 33)



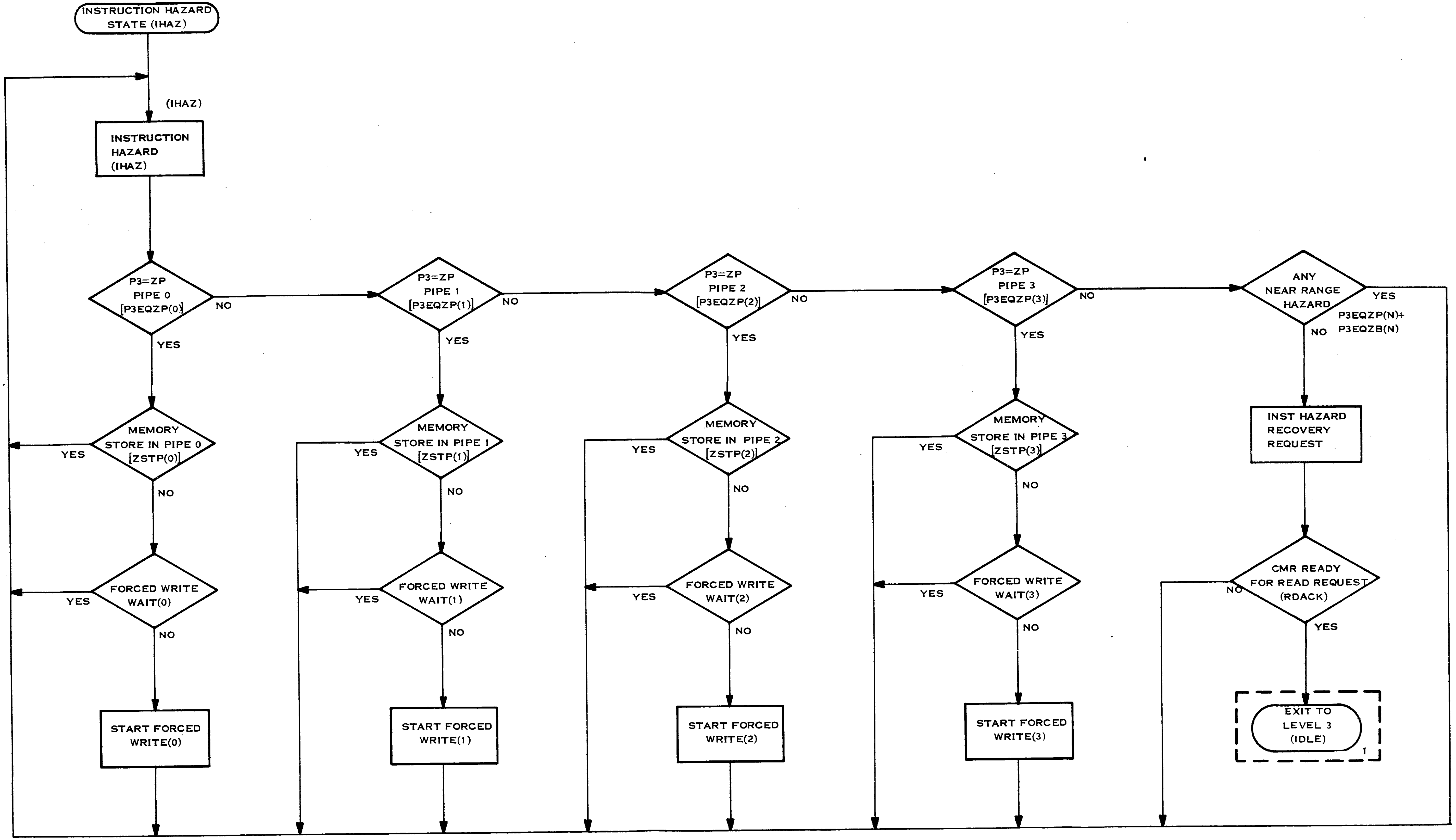
(D)127601 (15/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 15 of 33)



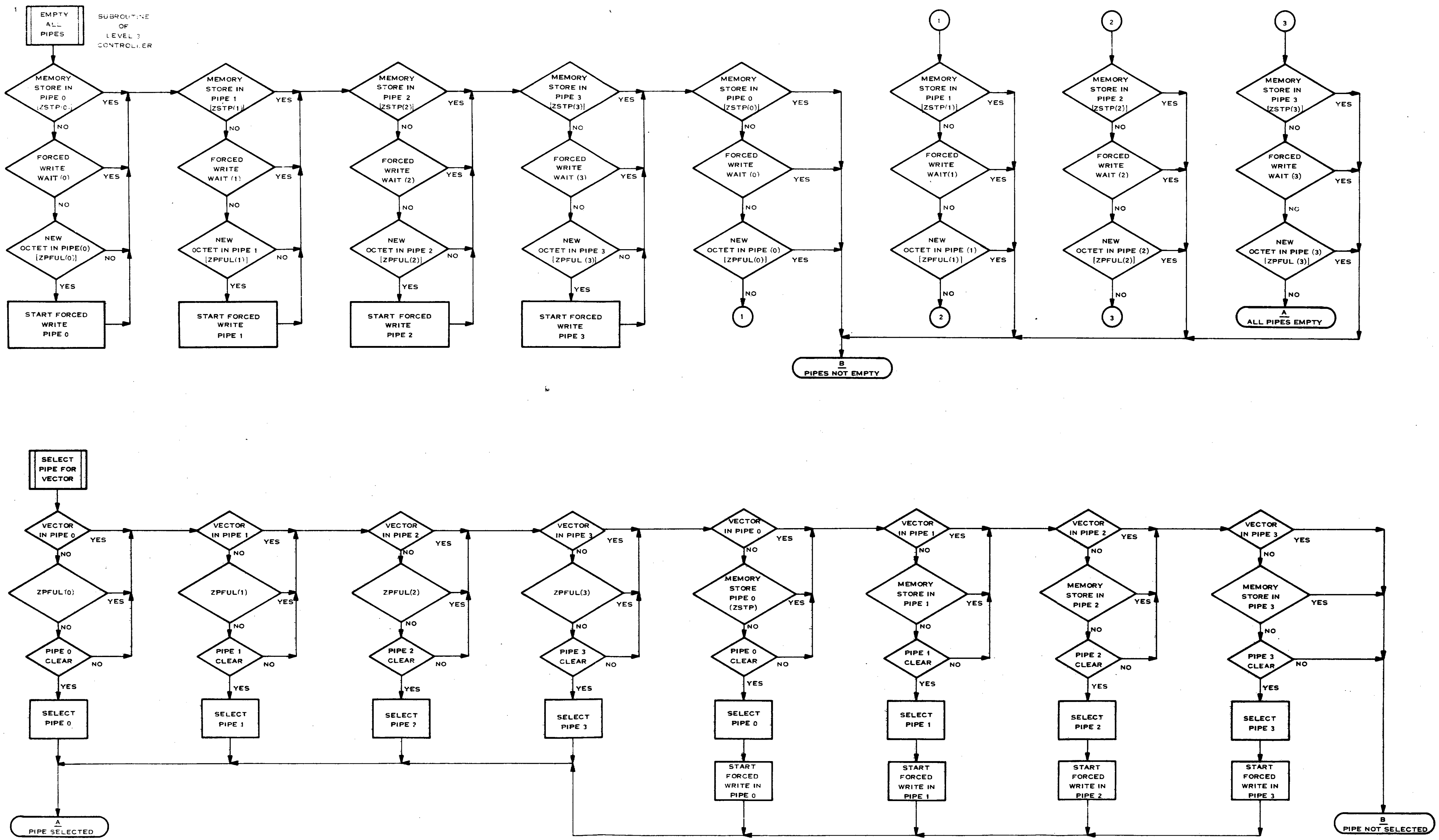
(D)127601 (16/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 16 of 33)



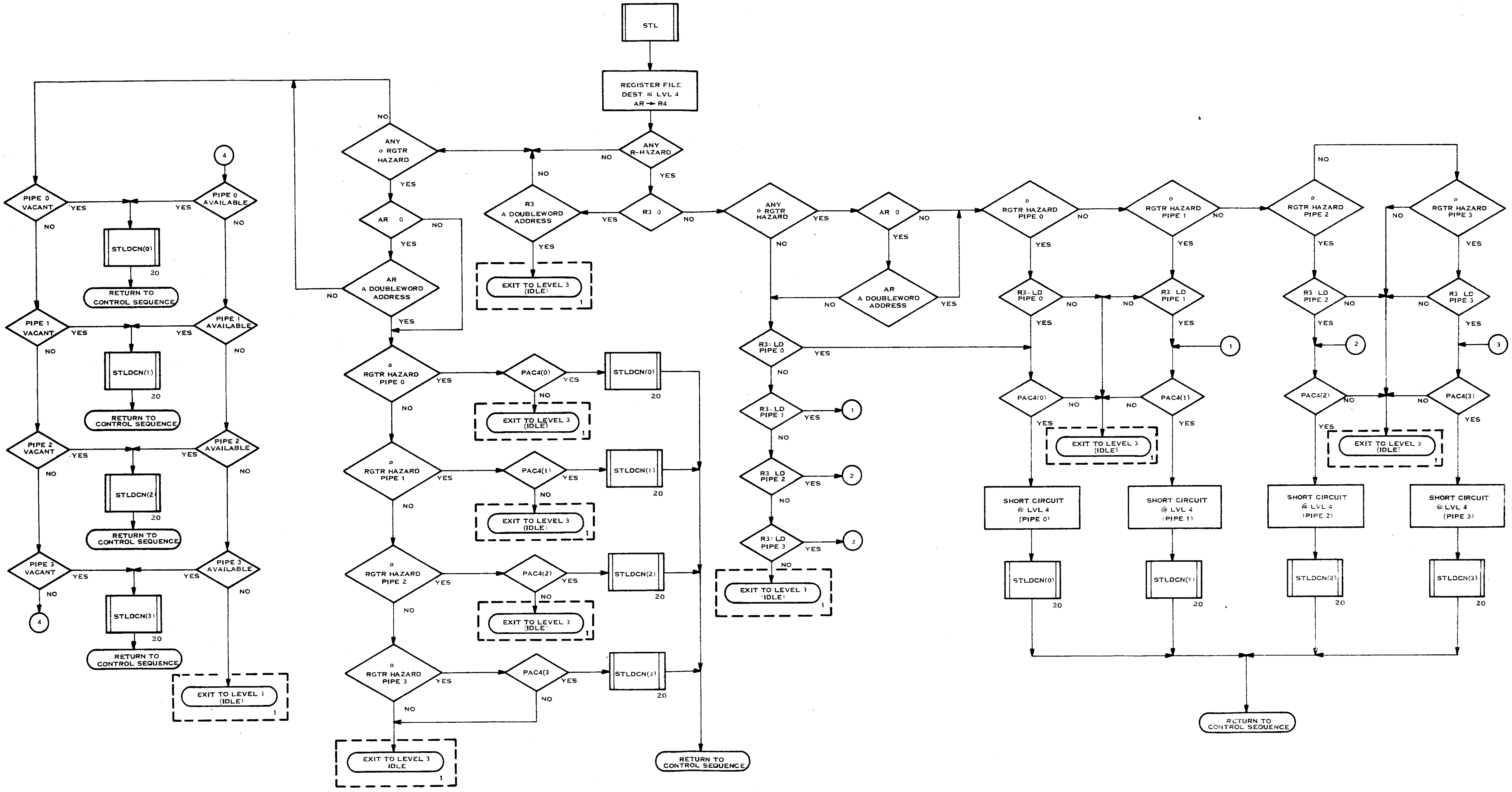
(D) 127601 (17/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 17 of 33)


1
EMPTY ALL PIPES
SUBROUTINE OF LEVEL 3 CONTROLLER
MEMORY STORE IN PIPE 0 [ZSTP(0)]
FORCED WRITE WAIT (0)
NEW OCTET IN PIPE(0) [ZPFUL(0)]
START FORCED WRITE PIPE 0
MEMORY STORE IN PIPE 1 [ZSTP(1)]
FORCED WRITE WAIT (1)
NEW OCTET IN PIPE 1 [ZPFUL(1)]
START FORCED WRITE PIPE 1
MEMORY STORE IN PIPE 2 [ZSTP(2)]
FORCED WRITE WAIT (2)
NEW OCTET IN PIPE 2 [ZPFUL(2)]
START FORCED WRITE PIPE 2
MEMORY STORE IN PIPE 3 [ZSTP(3)]
FORCED WRITE WAIT (3)
NEW OCTET IN PIPE 3 [ZPFUL (3)]
START FORCED WRITE PIPE 3
MEMORY STORE IN PIPE 0 [ZSTP(0)]
FORCED WRITE WAIT (0)
NEW OCTET IN PIPE (0) [ZPFUL(0)]
1
2
3
MEMORY STORE IN PIPE 1 [ZSTP(1)]
FORCED WRITE WAIT(1)
NEW OCTET IN PIPE (1) [ZPFUL(1)]
MEMORY STORE IN PIPE 2 [ZSTP(2)]
FORCED WRITE WAIT (2)
NEW OCTET IN PIPE (2) [ZPFUL(2)]
MEMORY STORE IN PIPE 3 [ZSTP(3)]
FORCED WRITE WAIT (3)
NEW OCTET IN PIPE (3) [ZPFUL (3)]
A ALL PIPES EMPTY
B PIPES NOT EMPTY
YES
NO
SELECT PIPE FOR VECTOR
VECTOR IN PIPE 0
ZPFUL(0)
PIPE 0 CLEAR
SELECT PIPE 0
VECTOR IN PIPE 1
ZPFUL(1)
PIPE 1 CLEAR
SELECT PIPE 1
VECTOR IN PIPE 2
ZPFUL(2)
PIPE 2 CLEAR
SELECT PIPE 2
VECTOR IN PIPE 3
ZPFUL(3)
PIPE 3 CLEAR
SELECT PIPE 3
MEMORY STORE PIPE 0 (ZSTP)
START FORCED WRITE IN PIPE 0
MEMORY STORE IN PIPE 1
START FORCED WRITE IN PIPE 1
MEMORY STORE IN PIPE 2
START FORCED WRITE IN PIPE 2
MEMORY STORE IN PIPE 3
START FORCED WRITE IN PIPE 3
A PIPE SELECTED
B PIPE NOT SELECTED

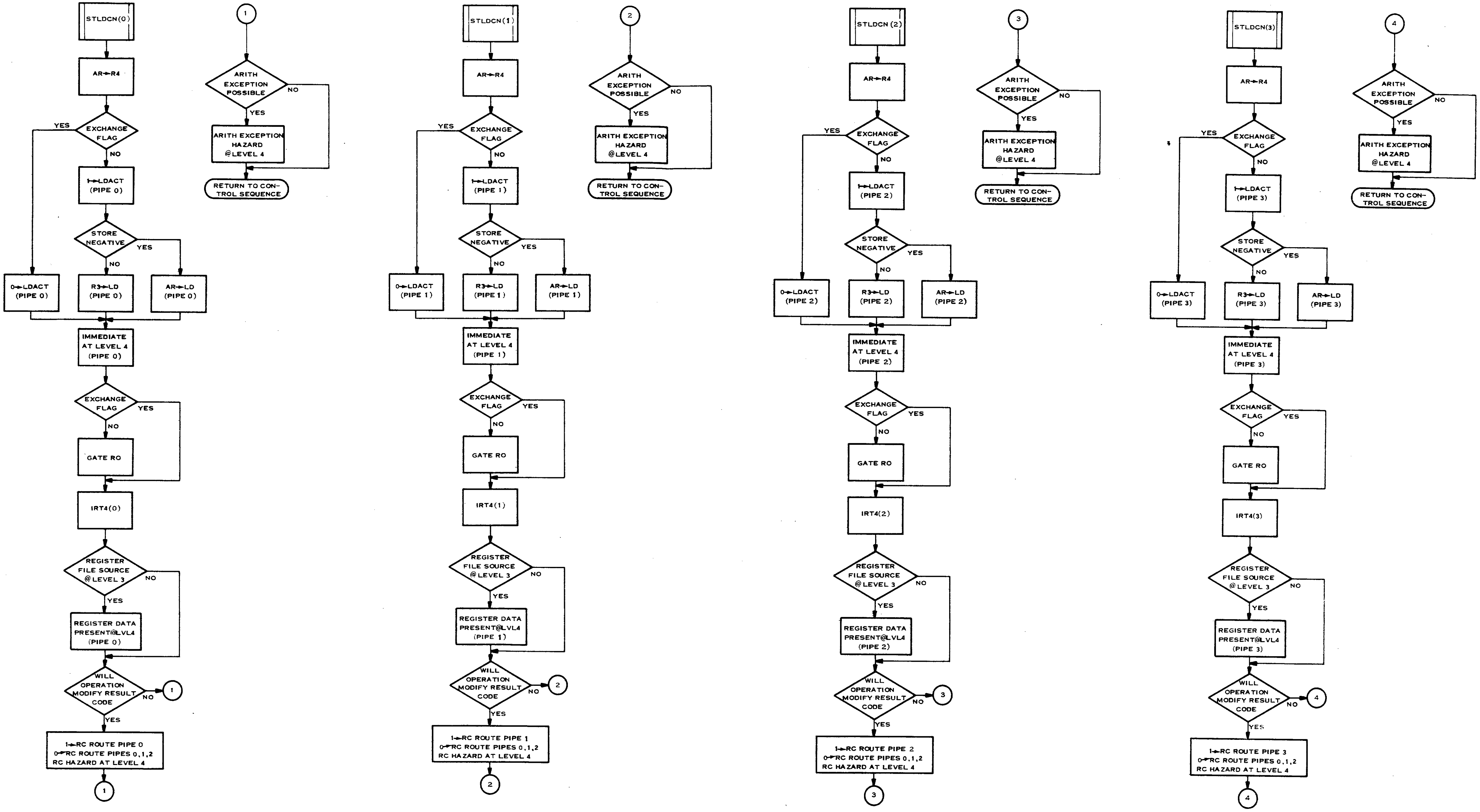
(D)127601 (18/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 18 of 33)



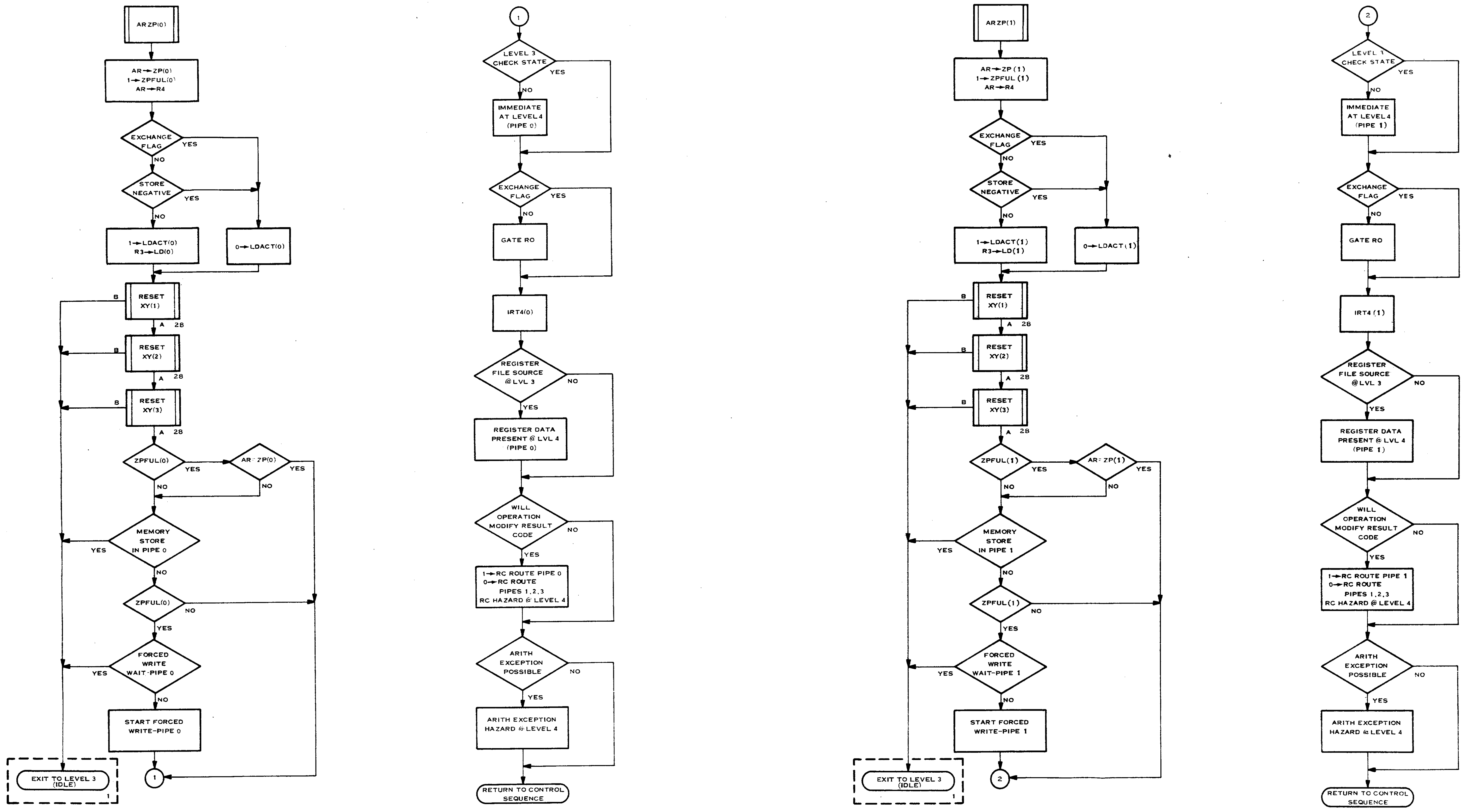
(D)127601 (19/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 19 of 33)



(D) 127601 (20/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 20 of 33)



(D) 127601 (21/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 21 of 33)

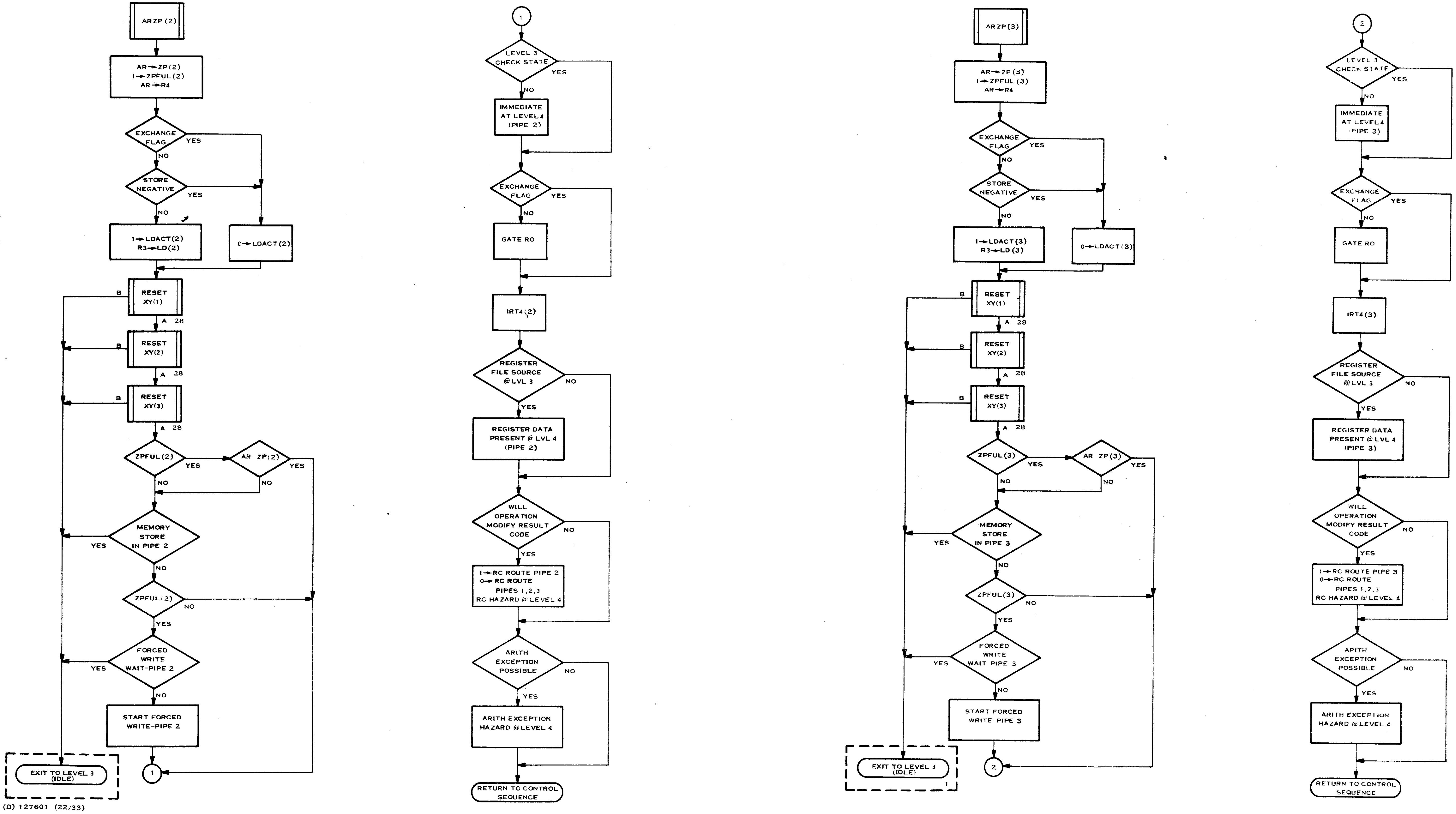


Figure 4-25. Level 3 Controller Flowchart (Sheet 22 of 33)

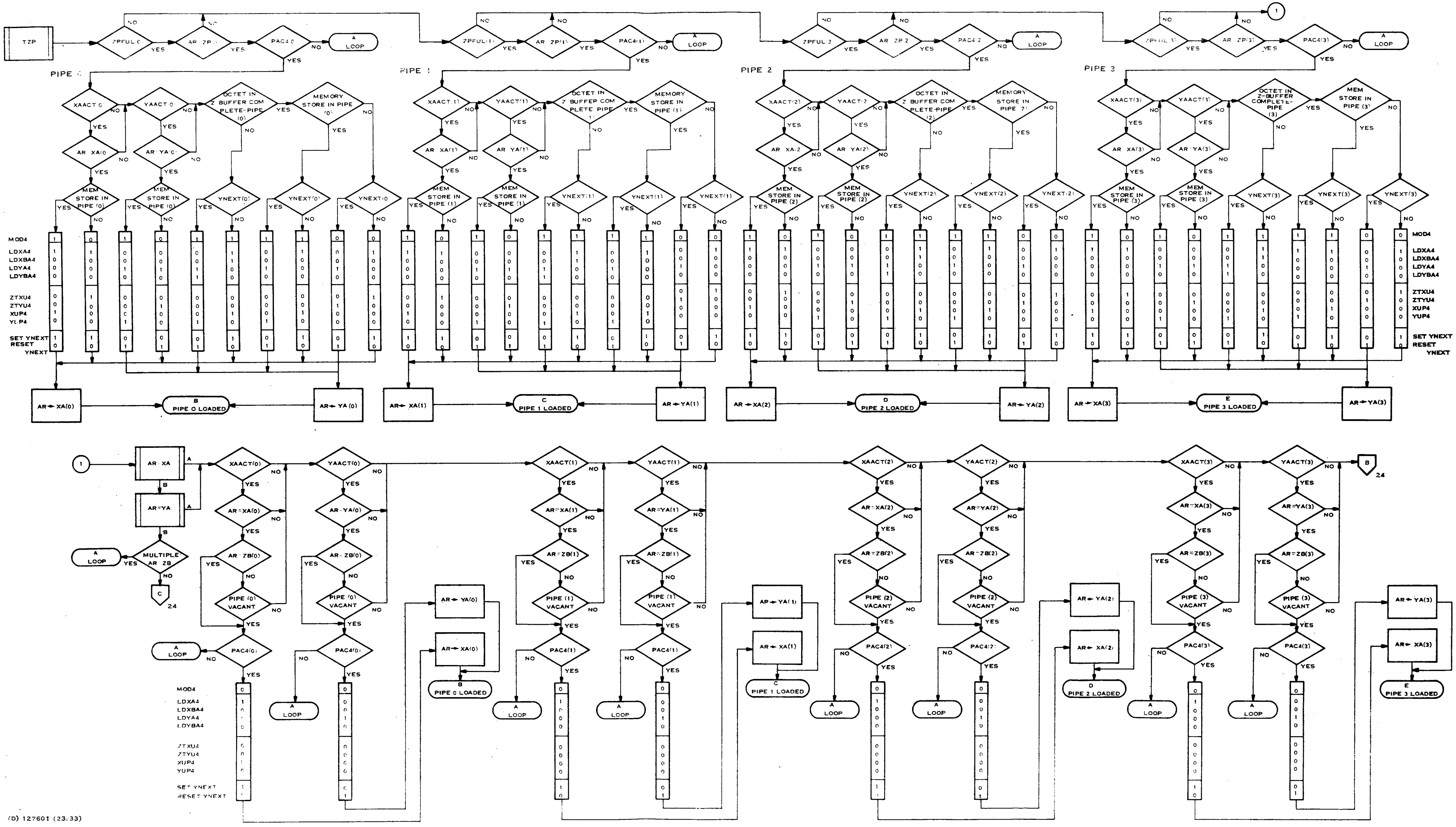
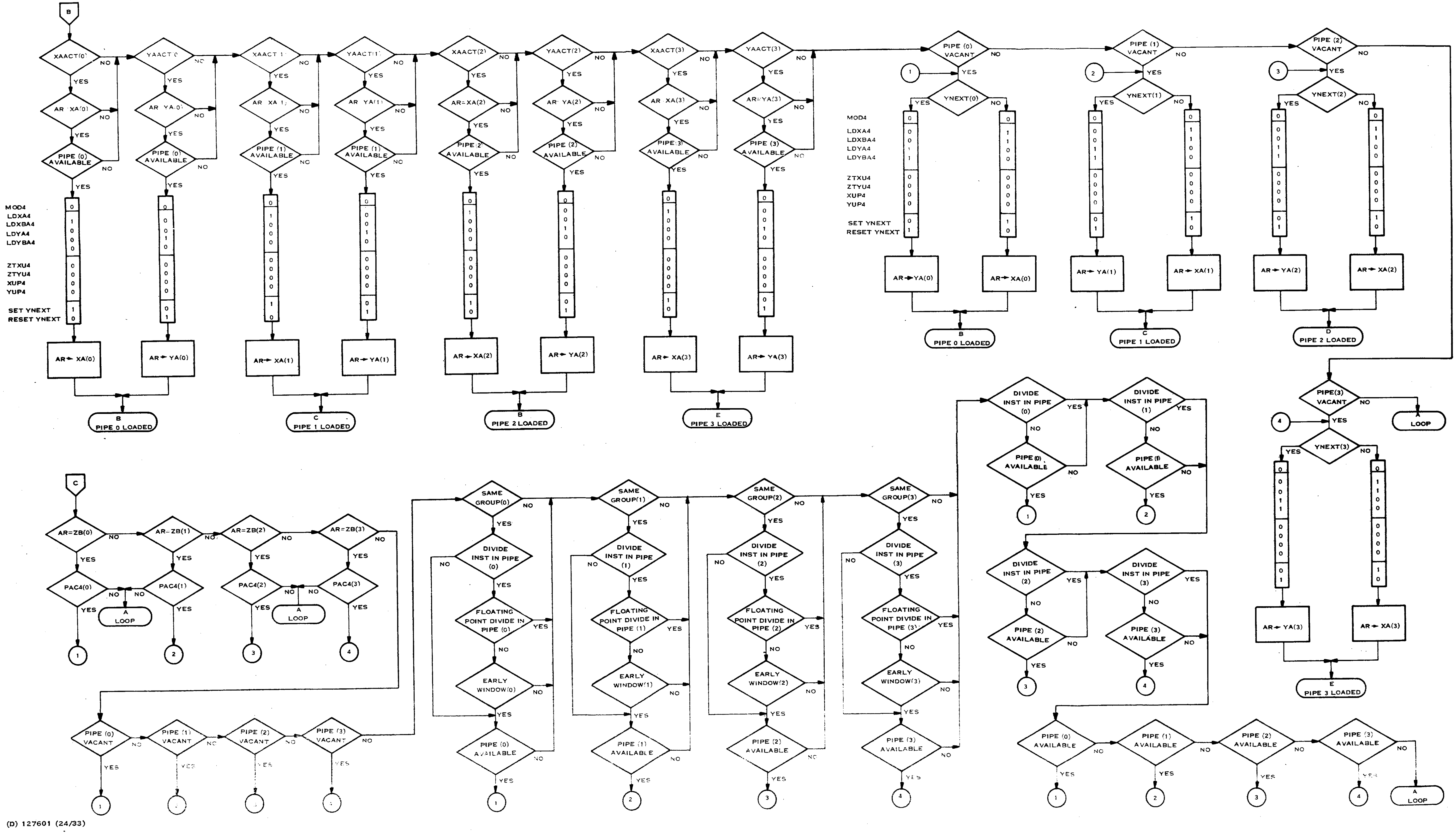
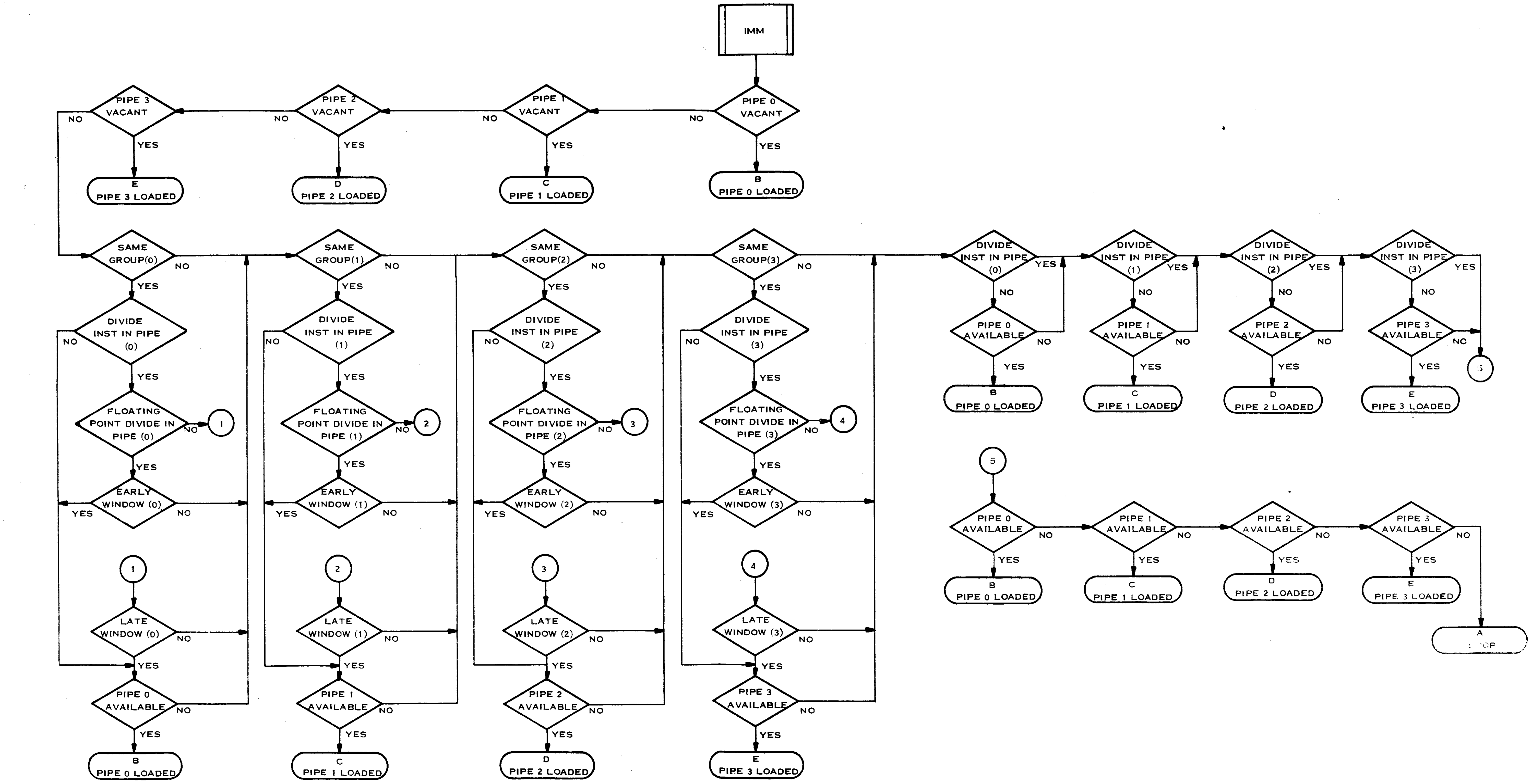


Figure 4-25. Level 3 Controller Flowchart (Sheet 23 of 33)



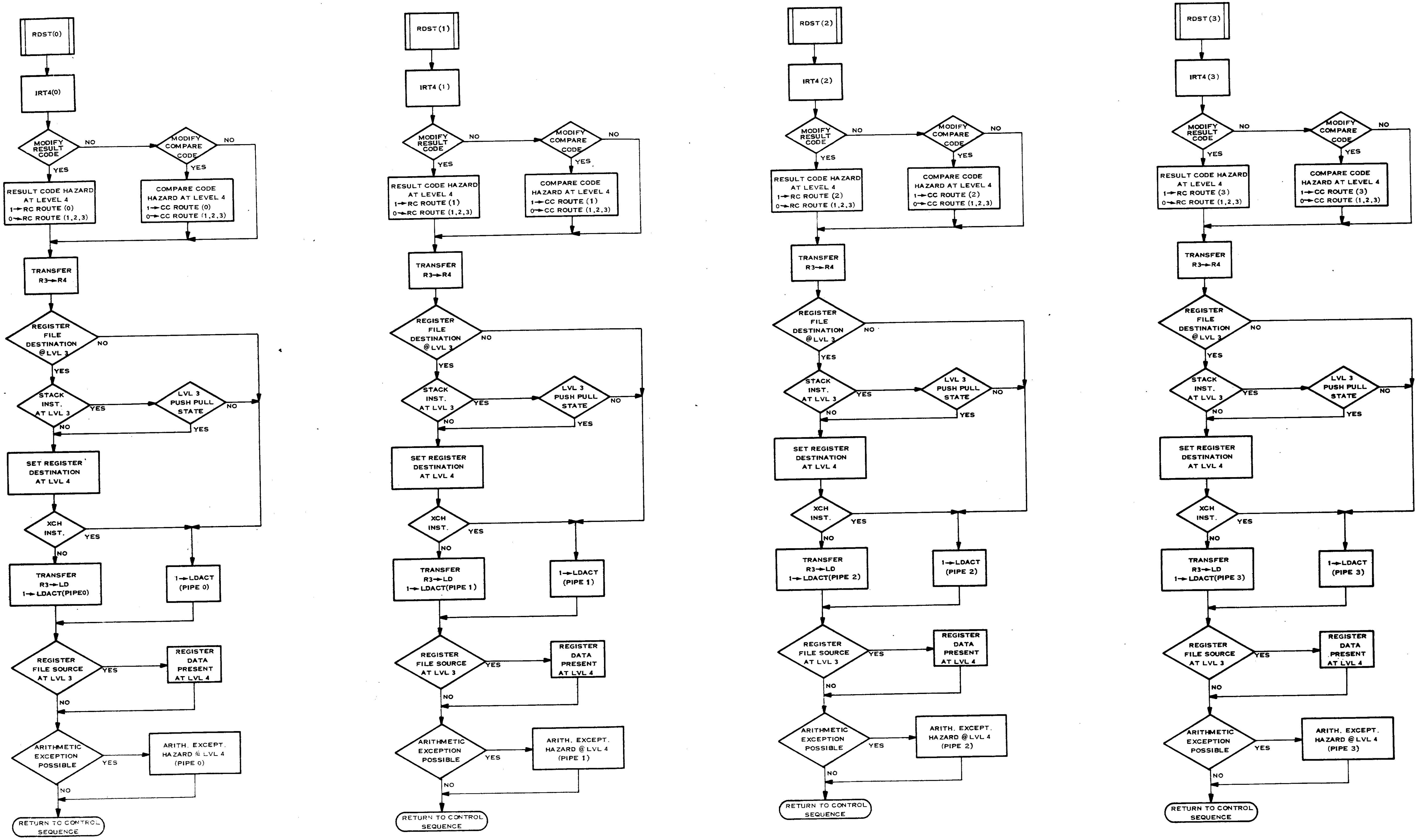
(D) 127601 (24/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 24 of 33)



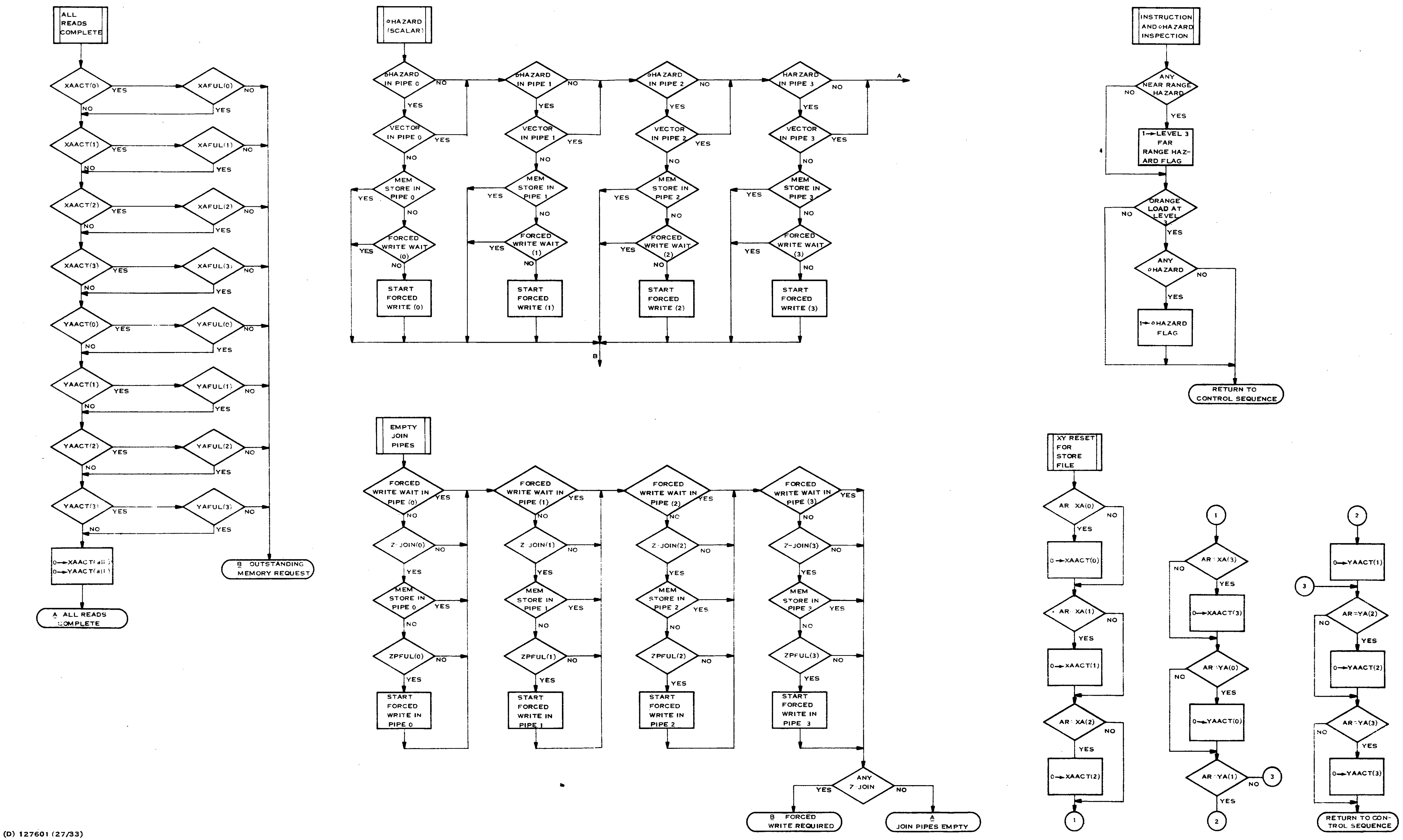
(D) 127601 (25/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 25 of 33)


RDST(0)
IRT4(0)
MODIFY RESULT CODE
NO
YES
MODIFY COMPARE CODE
NO
YES
RESULT CODE HAZARD AT LEVEL 4 1→RC ROUTE (0) 0→RC ROUTE (1,2,3)
COMPARE CODE HAZARD AT LEVEL 4 1→CC ROUTE (0) 0→CC ROUTE (1,2,3)
TRANSFER R3→R4
REGISTER FILE DESTINATION @LVL 3
NO
YES
STACK INST. AT LVL 3
YES
NO
LVL 3 PUSH PULL STATE
NO
YES
SET REGISTER DESTINATION AT LVL 4
XCH INST.
YES
NO
TRANSFER R3→LD 1→LDACT(PIPE0)
1→LDACT (PIPE 0)
REGISTER FILE SOURCE AT LVL 3
YES
NO
REGISTER DATA PRESENT AT LVL 4
ARITHMETIC EXCEPTION POSSIBLE
YES
NO
ARITH. EXCEPT. HAZARD @ LVL 4 (PIPE 0)
RETURN TO CONTROL SEQUENCE
RDST (1)
IRT4 (1)
MODIFY RESULT CODE
NO
YES
MODIFY COMPARE CODE
NO
YES
RESULT CODE HAZARD AT LEVEL 4 1→RC ROUTE (1) 0→RC ROUTE (1,2,3)
COMPARE CODE HAZARD AT LEVEL 4 1→CC ROUTE (1) 0→CC ROUTE (1,2,3)
TRANSFER R3→R4
REGISTER FILE DESTINATION @LVL 3
NO
YES
STACK INST. AT LVL 3
YES
NO
LVL 3 PUSH PULL STATE
NO
YES
SET REGISTER DESTINATION AT LVL 4
XCH INST.
YES
NO
TRANSFER R3→LD 1→LDACT(PIPE 1)
1→LDACT (PIPE 1)
REGISTER FILE SOURCE AT LVL 3
YES
NO
REGISTER DATA PRESENT AT LVL 4
ARITHMETIC EXCEPTION POSSIBLE
YES
NO
ARITH. EXCEPT. HAZARD @ LVL 4 (PIPE 1)
RETURN TO CONTROL SEQUENCE
RDST(2)
IRT4(2)
MODIFY RESULT CODE
NO
YES
MODIFY COMPARE CODE
NO
YES
RESULT CODE HAZARD AT LEVEL 4 1→RC ROUTE (2) 0→RC ROUTE (1,2,3)
COMPARE CODE HAZARD AT LEVEL 4 1→CC ROUTE (2) 0→CC ROUTE (1,2,3)
TRANSFER R3→R4
REGISTER FILE DESTINATION @LVL 3
NO
YES
STACK INST. AT LVL 3
YES
NO
LVL 3 PUSH PULL STATE
NO
YES
SET REGISTER DESTINATION AT LVL 4
XCH INST.
YES
NO
TRANSFER R3→LD 1→LDACT(PIPE 2)
1→LDACT (PIPE 2)
REGISTER FILE SOURCE AT LVL 3
YES
NO
REGISTER DATA PRESENT AT LVL 4
ARITHMETIC EXCEPTION POSSIBLE
YES
NO
ARITH. EXCEPT. HAZARD @LVL 4 (PIPE 2)
RETURN TO CONTROL SEQUENCE
RDST(3)
IRT4(3)
MODIFY RESULT CODE
NO
YES
MODIFY COMPARE CODE
NO
YES
RESULT CODE HAZARD AT LEVEL 4 1→RC ROUTE (3) 0→RC ROUTE (1,2,3)
COMPARE CODE HAZARD AT LEVEL 4 1→CC ROUTE (3) 0→CC ROUTE (1,2,3)
TRANSFER R3→R4
REGISTER FILE DESTINATION @LVL 3
NO
YES
STACK INST. AT LVL 3
YES
NO
LVL 3 PUSH PULL STATE
NO
YES
SET REGISTER DESTINATION AT LVL 4
XCH INST.
YES
NO
TRANSFER R3→LD 1→LDACT(PIPE 3)
1→LDACT (PIPE 3)
REGISTER FILE SOURCE AT LVL 3
YES
NO
REGISTER DATA PRESENT AT LVL 4
ARITHMETIC EXCEPTION POSSIBLE
YES
NO
ARITH. EXCEPT. HAZARD @LVL 4 (PIPE 3)
RETURN TO CONTROL SEQUENCE

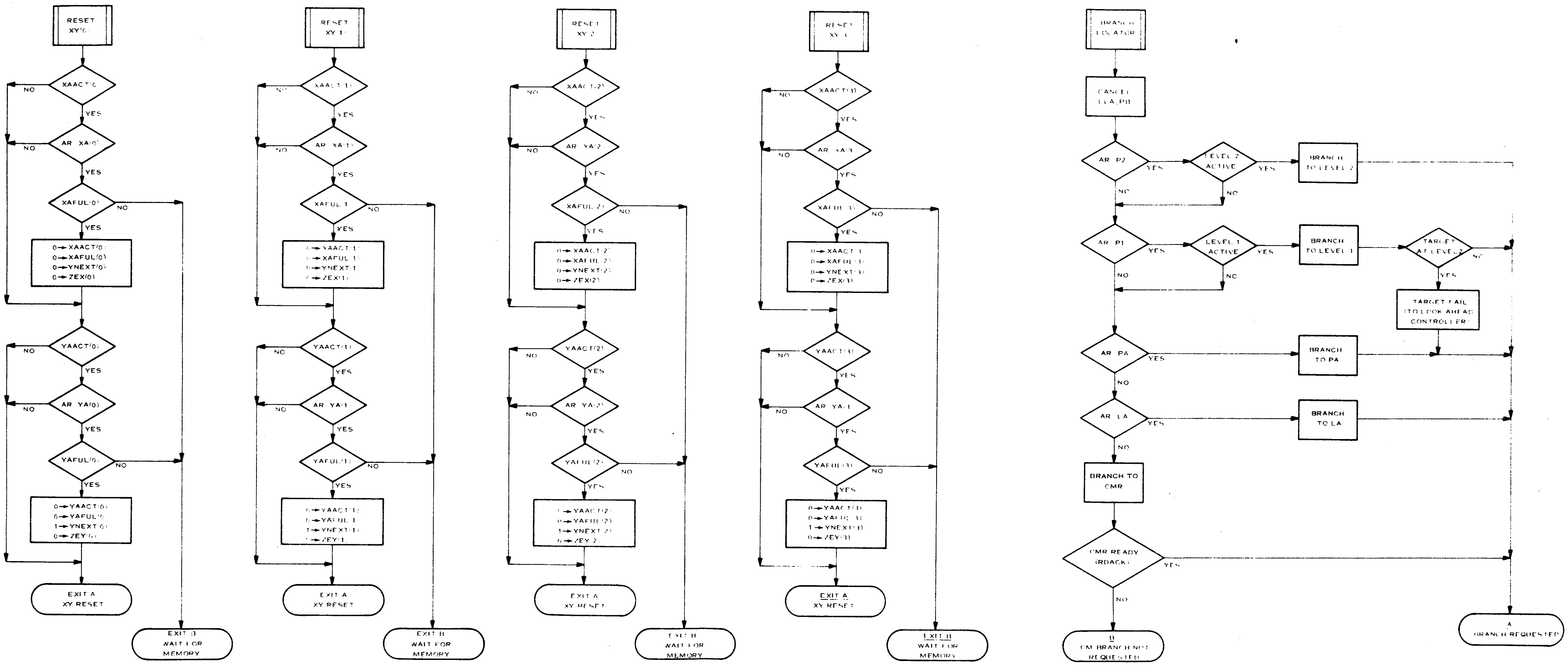
(D) 127601 (26/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 26 of 33)



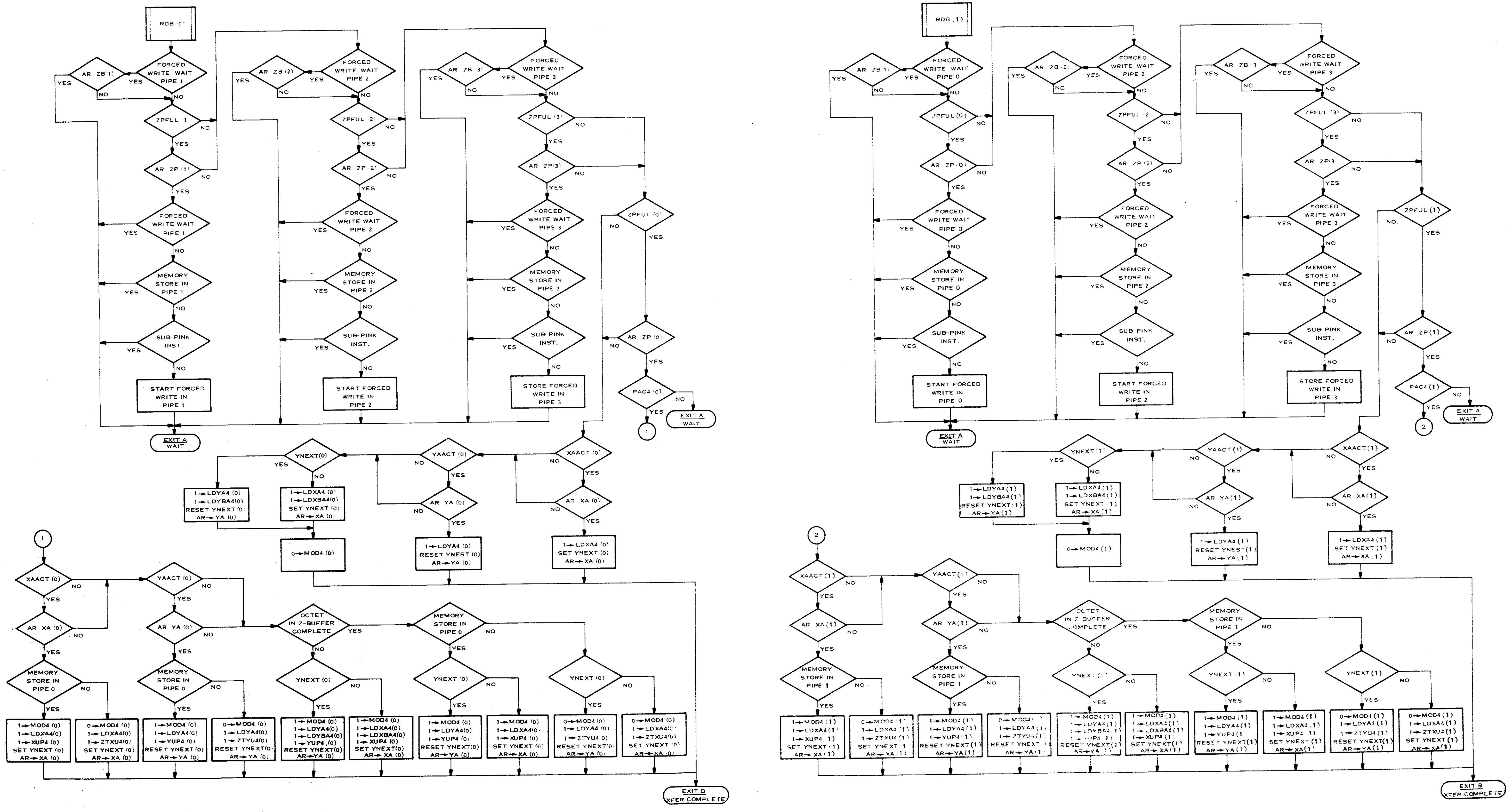
(D) 127601 (27/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 27 of 33)



(D) 127601 (28/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 28 of 33)


RDB (0)
FORCED WRITE WAIT PIPE 1
AR ZB (1)
ZPFUL (1)
AR ZP (1)
FORCED WRITE WAIT PIPE 1
MEMORY STORE IN PIPE 1
SUB-PINK INST.
START FORCED WRITE IN PIPE 1
FORCED WRITE WAIT PIPE 2
AR ZB (2)
ZPFUL (2)
AR ZP (2)
FORCED WRITE WAIT PIPE 2
MEMORY STORE IN PIPE 2
SUB-PINK INST.
START FORCED WRITE IN PIPE 2
FORCED WRITE WAIT PIPE 3
AR ZB (3)
ZPFUL (3)
AR ZP(3)
FORCED WRITE WAIT PIPE 3
MEMORY STORE IN PIPE 3
SUB-PINK INST.
STORE FORCED WRITE IN PIPE 3
ZPFUL (0)
AR ZP (0)
PAC4 (0)
EXIT A WAIT
EXIT A WAIT
YNEXT(0)
YAACT (0)
XAACT (0)
AR YA (0)
AR XA (0)
1→LDYA4 (0) 1→LDYBA4(0) RESET YNEXT (0) AR→YA (0)
1→LDXA4 (0) 1→LDXBA4(0) SET YNEXT (0) AR→XA (0)
0→MOD4 (0)
1→LDYA4 (0) RESET YNEST (0) AR→YA (0)
1→LDXA4 (0) SET YNEXT (0) AR→XA (0)
XAACT (0)
AR XA (0)
MEMORY STORE IN PIPE 0
YAACT (0)
AR YA (0)
MEMORY STORE IN PIPE 0
OCTET IN Z-BUFFER COMPLETE
YNEXT (0)
MEMORY STORE IN PIPE 0
YNEXT (0)
YNEXT (0)
1→MOD4 (0) 1→LDXA4(0) 1→XUP4 (0) SET YNEXT (0) AR→XA (0)
0→MOD4 (0) 1→LDXA4(0) 1→ZTXU4(0) SET YNEXT (0) AR→XA (0)
1→MOD4 (0) 1→LDYA4(0) 1→YUP4 (0) RESET YNEXT (0) AR→YA (0)
0→MOD4 (0) 1→LDYA4(0) 1→ZTYU4(0) RESET YNEXT(0) AR→YA (0)
1→MOD4 (0) 1→LDYA4(0) 1→LDYBA4(0) 1→YUP4 (0) RESET YNEXT(0) AR→YA (0)
1→MOD4 (0) 1→LDXA4(0) 1→LDXBA4(0) 1→XUP4 (0) SET YNEXT(0) AR→XA (0)
1→MOD4 (0) 1→LDYA4(0) 1→YUP4 (0) RESET YNEXT(0) AR→YA (0)
1→MOD4 (0) 1→LDXA4(0) 1→XUP4 (0) SET YNEXT (0) AR→XA (0)
0→MOD4 (0) 1→LDYA4 (0) 1→ZTYU4(0) RESET YNEXT(0) AR→YA (0)
0→MOD4 (0) 1→LDXA4(0) 1→ZTXU4(0) SET YNEXT (0) AR→XA (0)
EXIT B XFER COMPLETE
RDB (1)
FORCED WRITE WAIT PIPE 0
AR ZB (1)
ZPFUL (0)
AR ZP (0)
FORCED WRITE WAIT PIPE 0
MEMORY STORE IN PIPE 0
SUB-PINK INST.
START FORCED WRITE IN PIPE 0
FORCED WRITE WAIT PIPE 2
AR ZB (2)
ZPFUL (2)
AR ZP (2)
FORCED WRITE WAIT PIPE 2
MEMORY STORE IN PIPE 2
SUB-PINK INST.
START FORCED WRITE IN PIPE 2
FORCED WRITE WAIT PIPE 3
AR ZB (3)
ZPFUL (3)
AR ZP (3)
FORCED WRITE WAIT PIPE 3
MEMORY STORE IN PIPE 3
SUB-PINK INST.
STORE FORCED WRITE IN PIPE 3
ZPFUL (1)
AR ZP (1)
PAC4 (1)
EXIT A WAIT
EXIT A WAIT
YNEXT (1)
YAACT (1)
XAACT (1)
AR YA (1)
AR XA (1)
1→LDYA4 (1) 1→LDYBA4 (1) RESET YNEXT (1) AR→YA (1)
1→LDXA4 (1) 1→LDXBA4 (1) SET YNEXT (1) AR→XA (1)
0→MOD4 (1)
1→LDYA4 (1) RESET YNEST (1) AR→YA (1)
1→LDXA4 (1) SET YNEST (1) AR→XA (1)
XAACT (1)
AR XA (1)
MEMORY STORE IN PIPE 1
YAACT (1)
AR YA (1)
MEMORY STORE IN PIPE 1
OCTET IN Z BUFFER COMPLETE
YNEXT (1)
MEMORY STORE IN PIPE 1
YNEXT (1)
YNEXT (1)
1→MOD4 (1) 1→LDXA4 (1) 1→XUP4 (1) SET YNEXT (1) AR→XA (1)
0→MOD4 (1) 1→LDXA4 (1) 1→ZTXU4 (1) SET YNEXT (1) AR→XA (1)
1→MOD4 (1) 1→LDYA4 (1) 1→YUP4 (1) RESET YNEXT (1) AR→YA (1)
0→MOD4 (1) 1→LDYA4 (1) 1→ZTYU4 (1) RESET YNEXT (1) AR→YA (1)
1→MOD4 (1) 1→LDYA4 (1) 1→LDYBA4 (1) 1→YUP4 (1) RESET YNEXT (1) AR→YA (1)
1→MOD4 (1) 1→LDXA4 (1) 1→LDXBA4 (1) 1→XUP4 (1) SET YNEXT (1) AR→XA (1)
1→MOD4 (1) 1→LDYA4 (1) 1→YUP4 (1) RESET YNEXT (1) AR→YA (1)
1→MOD4 (1) 1→LDXA4 (1) 1→XUP4 (1) SET YNEXT (1) AR→XA (1)
0→MOD4 (1) 1→LDYA4 (1) 1→ZTYU4 (1) RESET YNEXT (1) AR→YA (1)
0→MOD4 (1) 1→LDXA4 (1) 1→ZTXU4 (1) SET YNEXT (1) AR→XA (1)
EXIT B XFER COMPLETE
YES
NO
1
2

(D) 127601 (29/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 29 of 33)

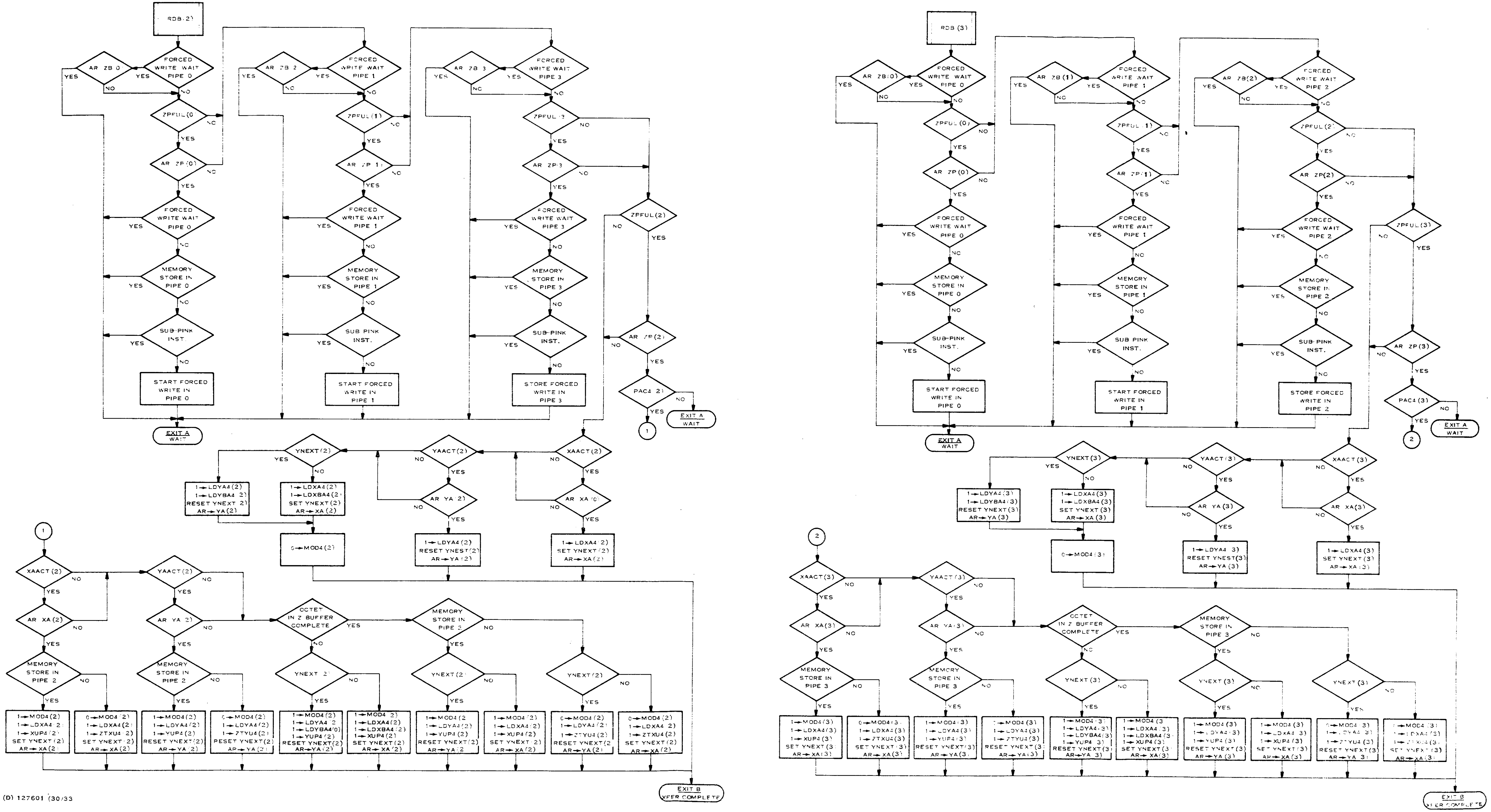

RDB(2)
AR ZB 0
FORCED WRITE WAIT PIPE 0
ZPFUL(0)
AR ZP(0)
FORCED WRITE WAIT PIPE 0
MEMORY STORE IN PIPE 0
SUB-PINK INST
START FORCED WRITE IN PIPE 0
AR ZB 2
FORCED WRITE WAIT PIPE 1
ZPFUL(1)
AR ZP 1
FORCED WRITE WAIT PIPE 1
MEMORY STORE IN PIPE 1
SUB PINK INST.
START FORCED WRITE IN PIPE 1
AR ZB 3
FORCED WRITE WAIT PIPE 3
ZPFUL 3
AR ZP 3
FORCED WRITE WAIT PIPE 3
MEMORY STORE IN PIPE 3
SUB-PINK INST.
STORE FORCED WRITE IN PIPE 3
ZPFUL(2)
AR ZP(2)
PAC4 2
EXIT A WAIT
1
YES
NO
YNEXT(2)
YAACT(2)
XAACT(2)
AR YA 2
AR XA(0)
1→LDYA4(2) 1→LDYBA4 2 RESET YNEXT 2 AR→YA(2)
1→LDXA4(2) 1→LDXBA4(2) SET YNEXT(2) AR→XA(2)
0→MOD4(2)
1→LDYA4(2) RESET YNEXT(2) AR→YA(2)
1→LDXA4 2 SET YNEXT(2) AR→XA(2)
XAACT(2)
AR XA(2)
MEMORY STORE IN PIPE 2
YAACT(2)
AR YA 2
MEMORY STORE IN PIPE 2
OCTET IN Z BUFFER COMPLETE
YNEXT 2
MEMORY STORE IN PIPE 2
YNEXT(2
YNEXT(2)
1→MOD4(2) 1→LDXA4 2 1→XUP4(2) SET YNEXT(2) AR→XA(2)
0→MOD4(2) 1→LDXA4(2) 1→ZTXU4(2) SET YNEXT(2) AR→XA(2)
1→MOD4(2) 1→LDYA4(2) 1→YUP4(2) RESET YNEXT(2) AR→YA(2)
0→MOD4(2) 1→LDYA4(2) 1→ZTYU4(2) RESET YNEXT(2) AR→YA(2)
1→MOD4(2) 1→LDYA4 2 1→LDYBA4(0) 1→YUP4(2) RESET YNEXT(2) AR→YA(2)
1→MOD4 2 1→LDXA4(2) 1→LDXBA4(2) 1→XUP4(2) SET YNEXT(2) AR→XA(2)
1→MOD4(2 1→LDYA4(2) 1→YUP4(2) RESET YNEXT(2) AR→YA(2
1→MOD4(2) 1→LDXA4(2) 1→XUP4(2) SET YNEXT 2) AR→XA(2)
0→MOD4(2) 1→LDYA4(2) 1→ZTYU4(2) RESET YNEXT(2 AR→YA(2)
0→MOD4(2) 1→LDXA4 2) 1→ZTXU4(2) SET YNEXT(2) AR→XA(2)
EXIT B XFER COMPLETE
(D) 127601 (30/33
RDB(3)
AR ZB(0)
FORCED WRITE WAIT PIPE 0
ZPFUL(0)
AR ZP(0)
FORCED WRITE WAIT PIPE 0
MEMORY STORE IN PIPE 0
SUB-PINK INST.
START FORCED WRITE IN PIPE 0
AR ZB(1)
FORCED WRITE WAIT PIPE 1
ZPFUL 1)
AR ZP(1)
FORCED WRITE WAIT PIPE 1
MEMORY STORE IN PIPE 1
SUB PINK INST.
START FORCED WRITE IN PIPE 1
AR ZB(2)
FORCED WRITE WAIT PIPE 2
ZPFUL(2)
AR ZP(2)
FORCED WRITE WAIT PIPE 2
MEMORY STORE IN PIPE 2
SUB-PINK INST.
STORE FORCED WRITE IN PIPE 2
ZPFUL(3)
AR ZP(3)
PAC4 (3)
EXIT A WAIT
2
YNEXT(3)
YAACT(3)
XAACT(3)
AR YA(3)
AR XA(3)
1→LDYA4(3) 1→LDYBA4(3) RESET YNEXT(3) AR→YA(3)
1→LDXA4(3) 1→LDXBA4(3) SET YNEXT(3) AR→XA(3)
0→MOD4(3)
1→LDYA4 3) RESET YNEST(3) AR→YA(3)
1→LDXA4(3) SET YNEXT(3) AR→XA(3)
XAACT(3)
AR XA(3)
MEMORY STORE IN PIPE 3
YAACT(3)
AR YA(3)
MEMORY STORE IN PIPE 3
OCTET IN Z BUFFER COMPLETE
YNEXT(3)
MEMORY STORE IN PIPE 3
YNEXT(3)
YNEXT(3)
1→MOD4(3) 1→LDXA4(3) 1→XUP4(3) SET YNEXT(3 AR→XA(3)
0→MOD4(3) 1→LDXA4(3) 1→ZTXU4(3) SET YNEXT(3) AR→XA(3)
1→MOD4(3) 1→LDYA4(3) 1→YUP4(3) RESET YNEXT(3) AR→YA(3)
1→MOD4(3) 1→LDYA4(3) 1→ZTYU4(3) RESET YNEXT(3) AR→YA(3)
1→MOD4 3 1→LDYA4(3) 1→LDYBA4(3) 1→YUP4 3) RESET YNEXT(3) AR→YA 3)
1→MOD4(3 1→LDXA4(3) 1→LDXBA4(3) 1→XUP4(3) SET YNEXT(3) AR→XA(3)
1→MOD4(3) 1→LDYA4(3) 1→YUP4(3) RESET YNEXT(3) AR→YA(3)
1→MOD4(3) 1→LDXA4 3) 1→XUP4(3) SET YNEXT(3) AR→XA(3)
0→MOD4 3) 1→LDYA4 3) 1→ZTYU4(3) RESET YNEXT(3) AR→YA 3)
1→MOD4 (3) 1→LDXA4(3) 1→ZTXU4(3) SET YNEXT(3) AR→XA(3)
EXIT B XFER COMPLETE

Figure 4-25. Level 3 Controller Flowchart (Sheet 30 of 33)

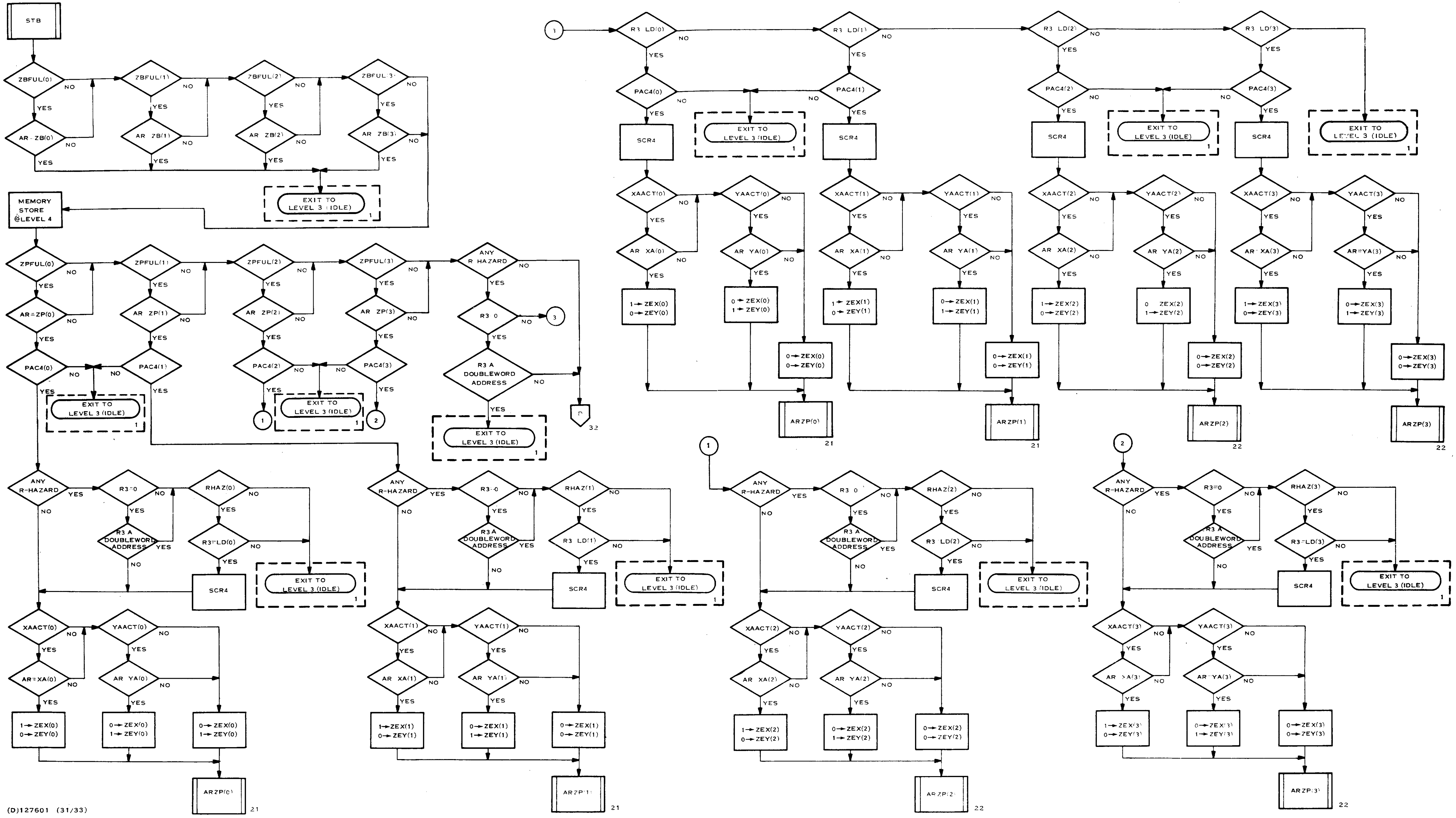
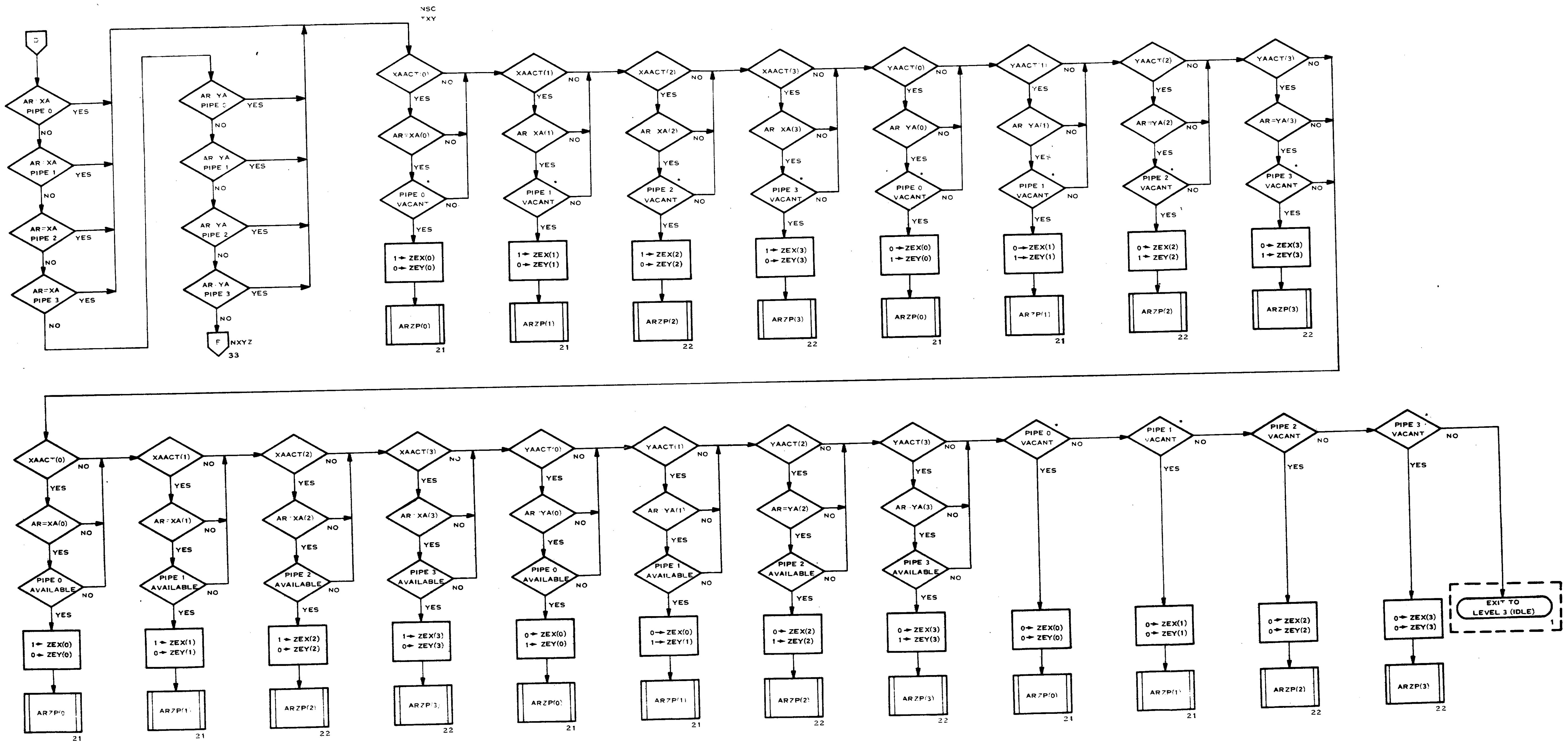


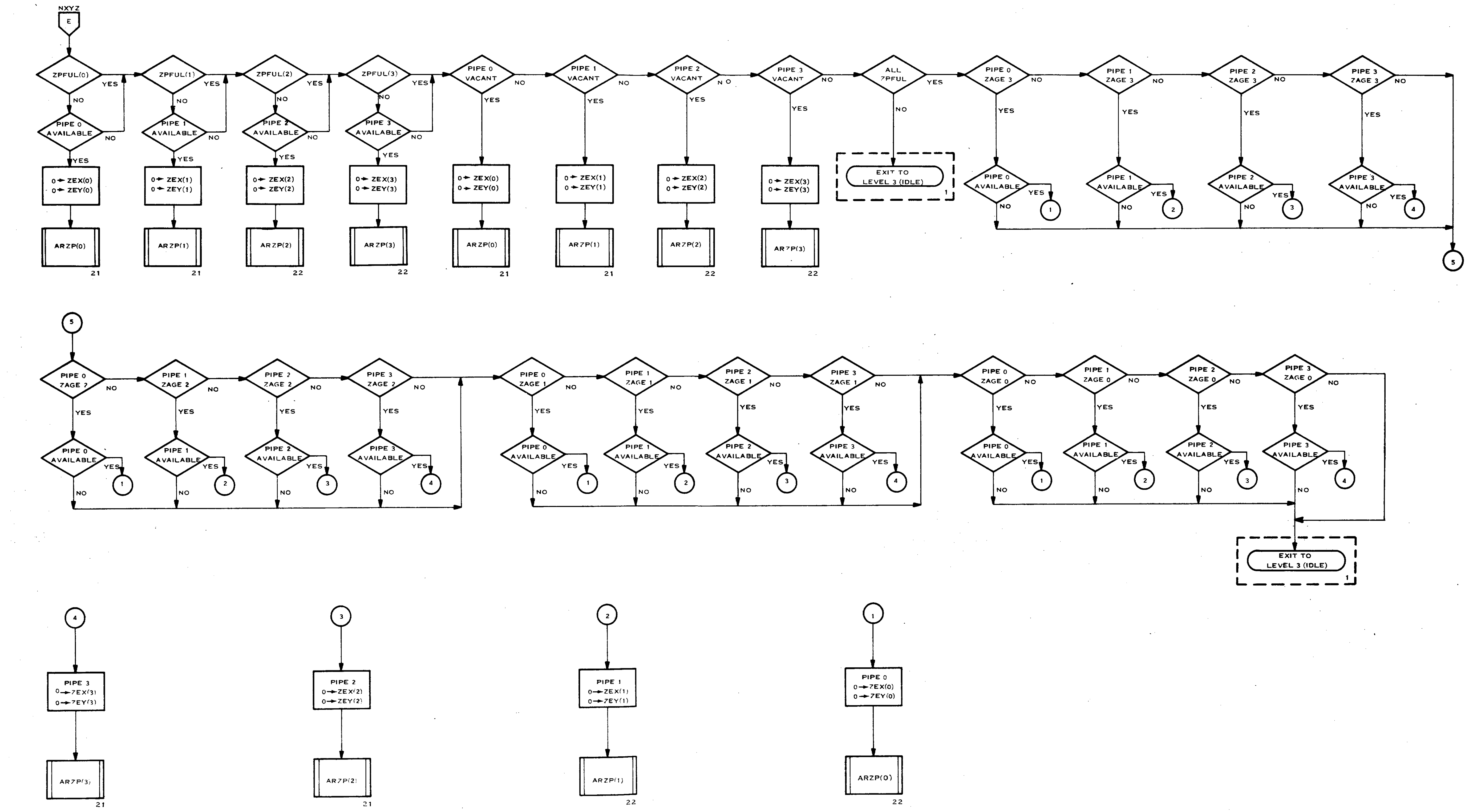
Figure 4-25. Level 3 Controller Flowchart (Sheet 31 of 33)



*IN "ONE PIPE MODE" OR IN "ONE OPERATIONAL SCALAS PIPE MODE,"
THE "ONE PIPE" IS ALWAYS VACANT AND PAC4 IS CHECKED.

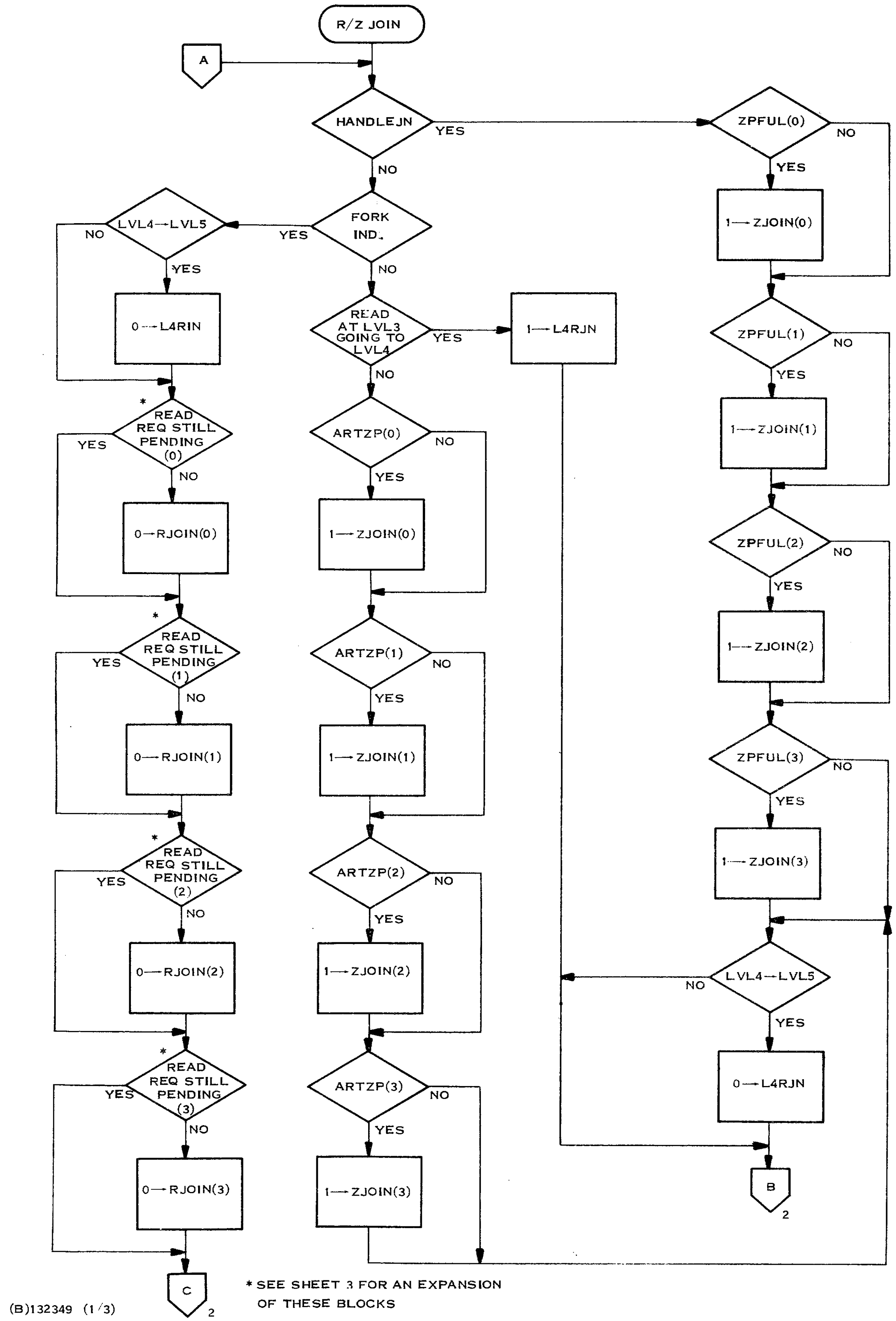
(D)127601 (32/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 32 of 33)



(D)127601 (33/33)

Figure 4-25. Level 3 Controller Flowchart (Sheet 33 of 33)



* SEE SHEET 3 FOR AN EXPANSION OF THESE BLOCKS

Figure 4-26. R/Z Join Flowchart (Sheet 1 of 3)

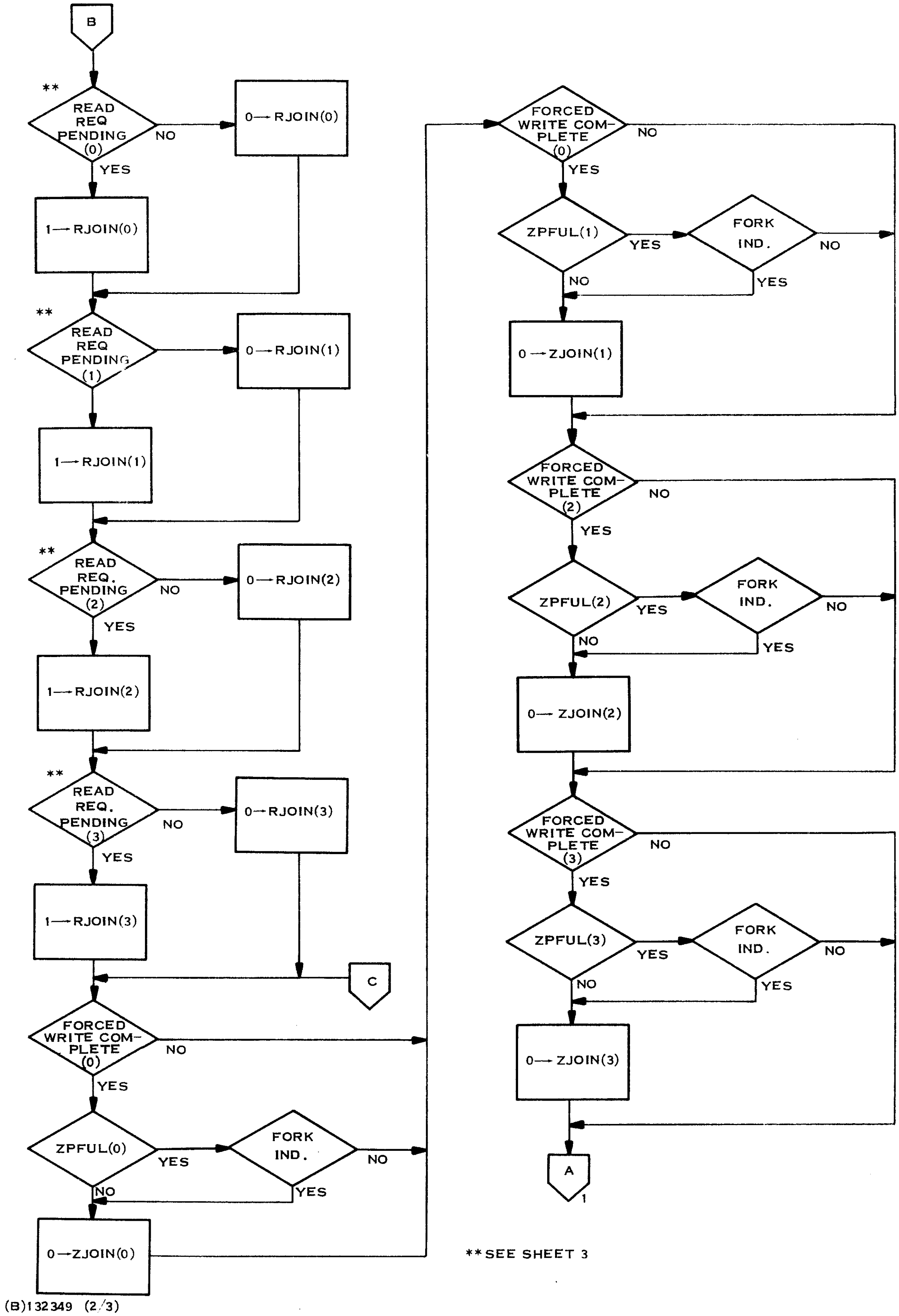


Figure 4-26. R/Z Join Flowchart (Sheet 2 of 3)

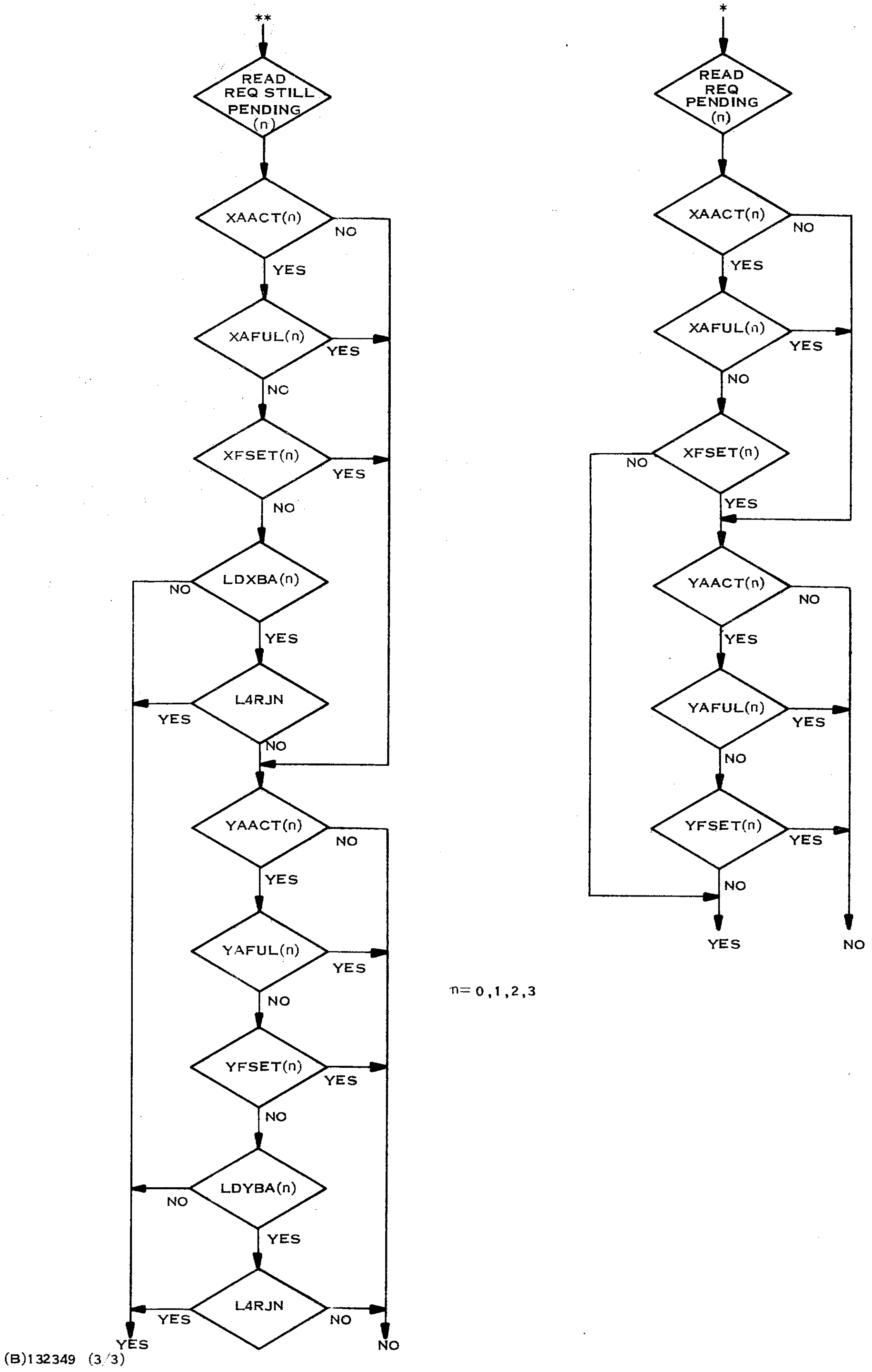
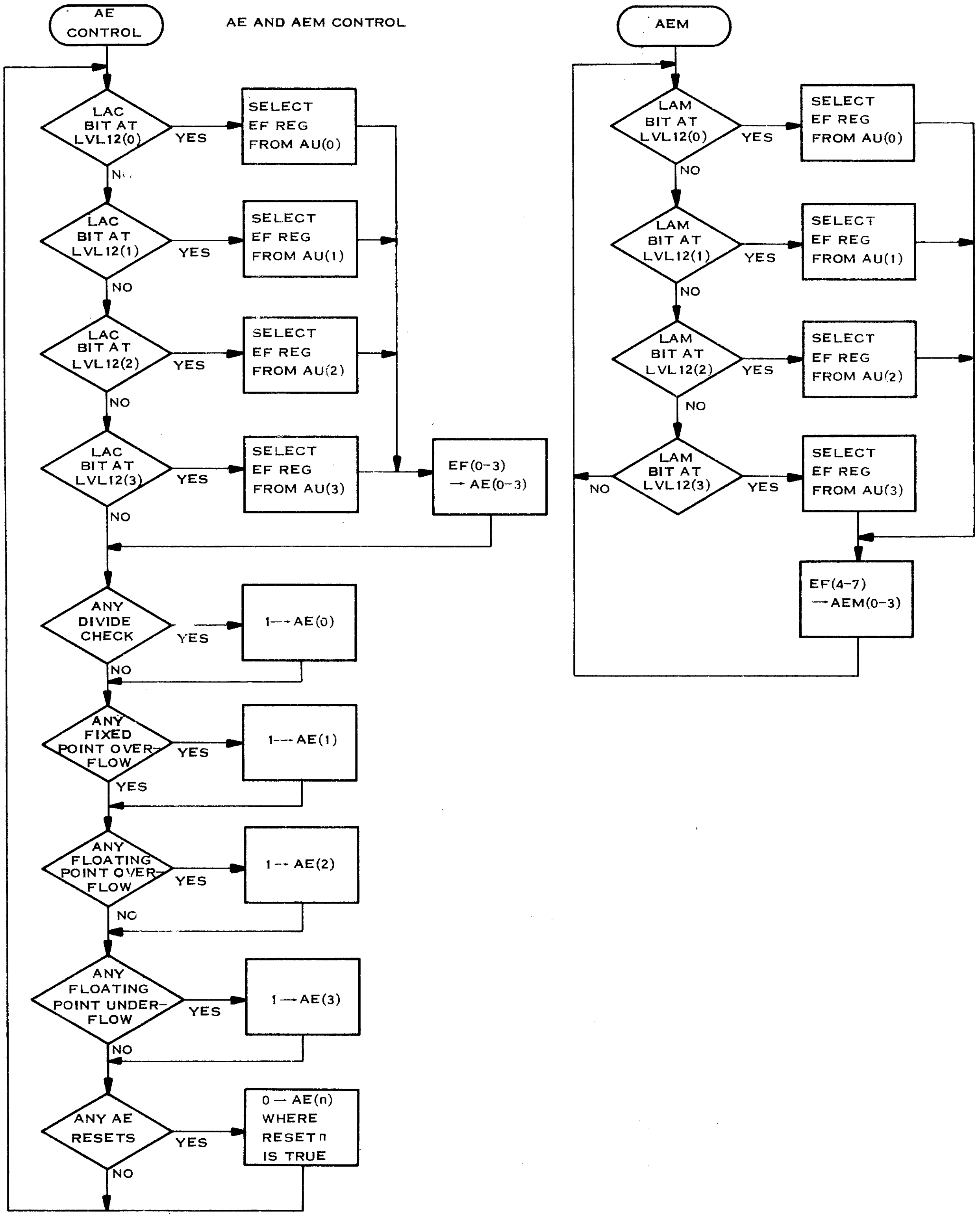
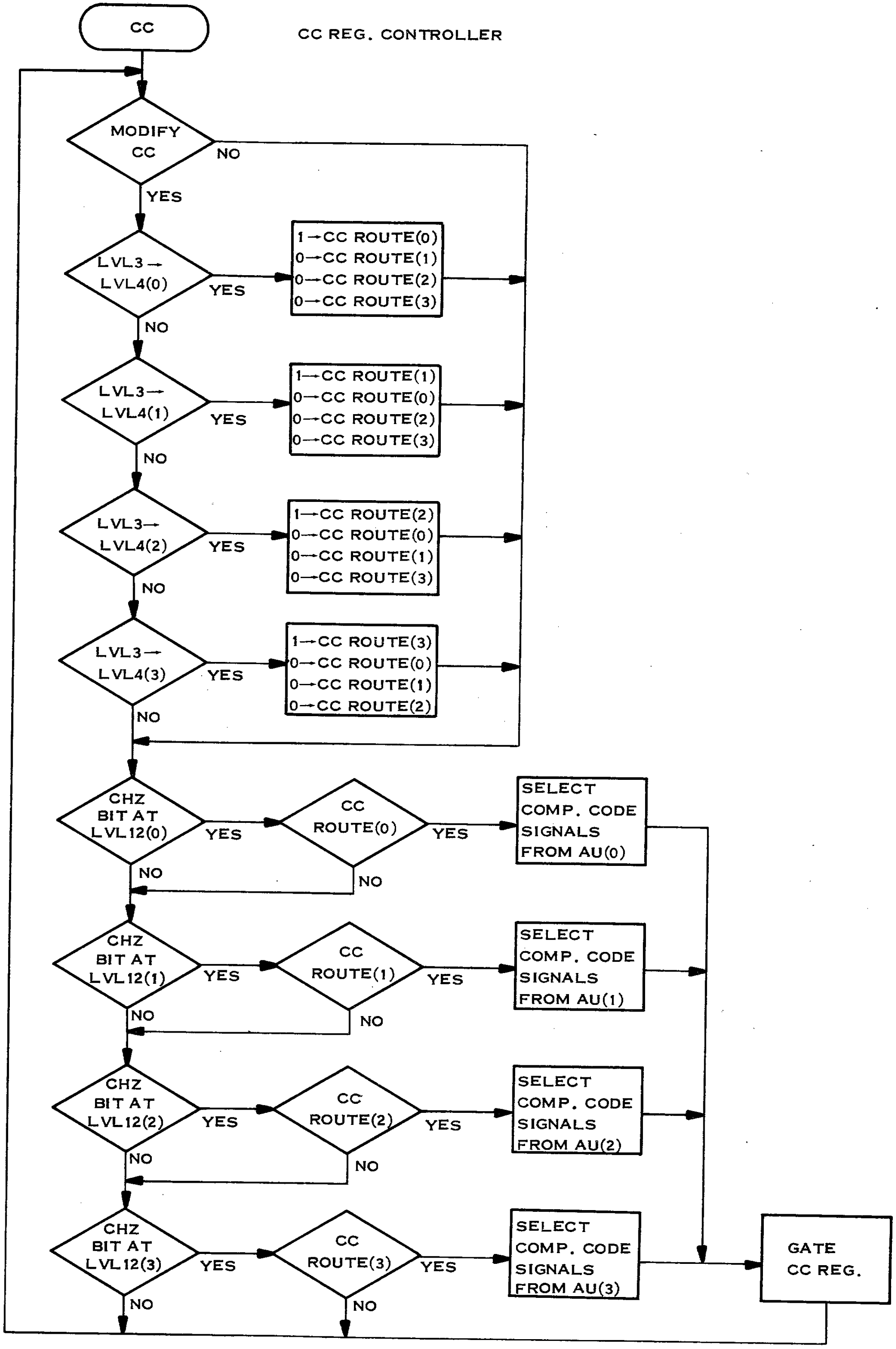


Figure 4-26. R/Z Join Flowchart (Sheet 3 of 3)



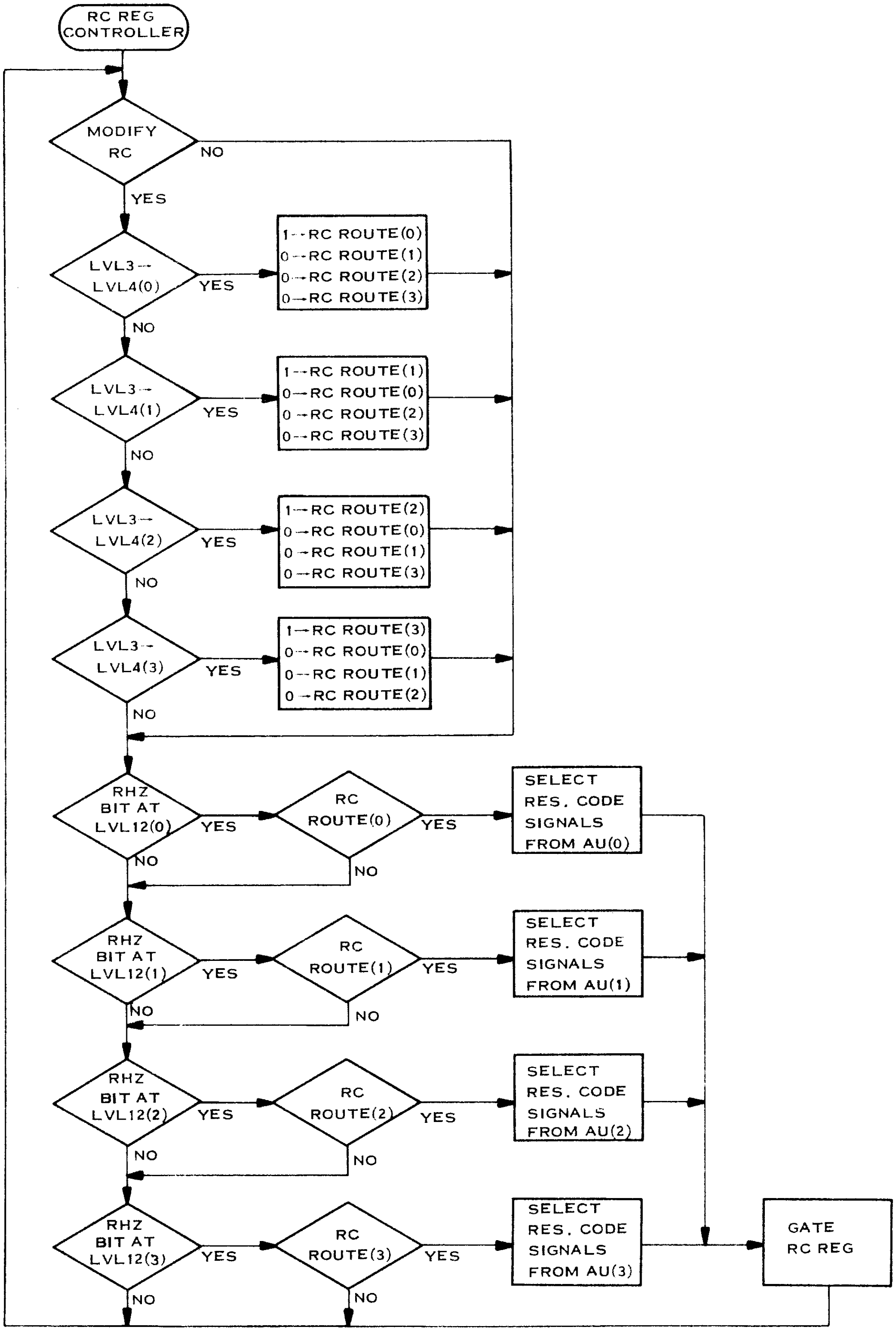
(B)132350

Figure 4-27. Arithmetic Exception and Arithmetic Exception Mask Control



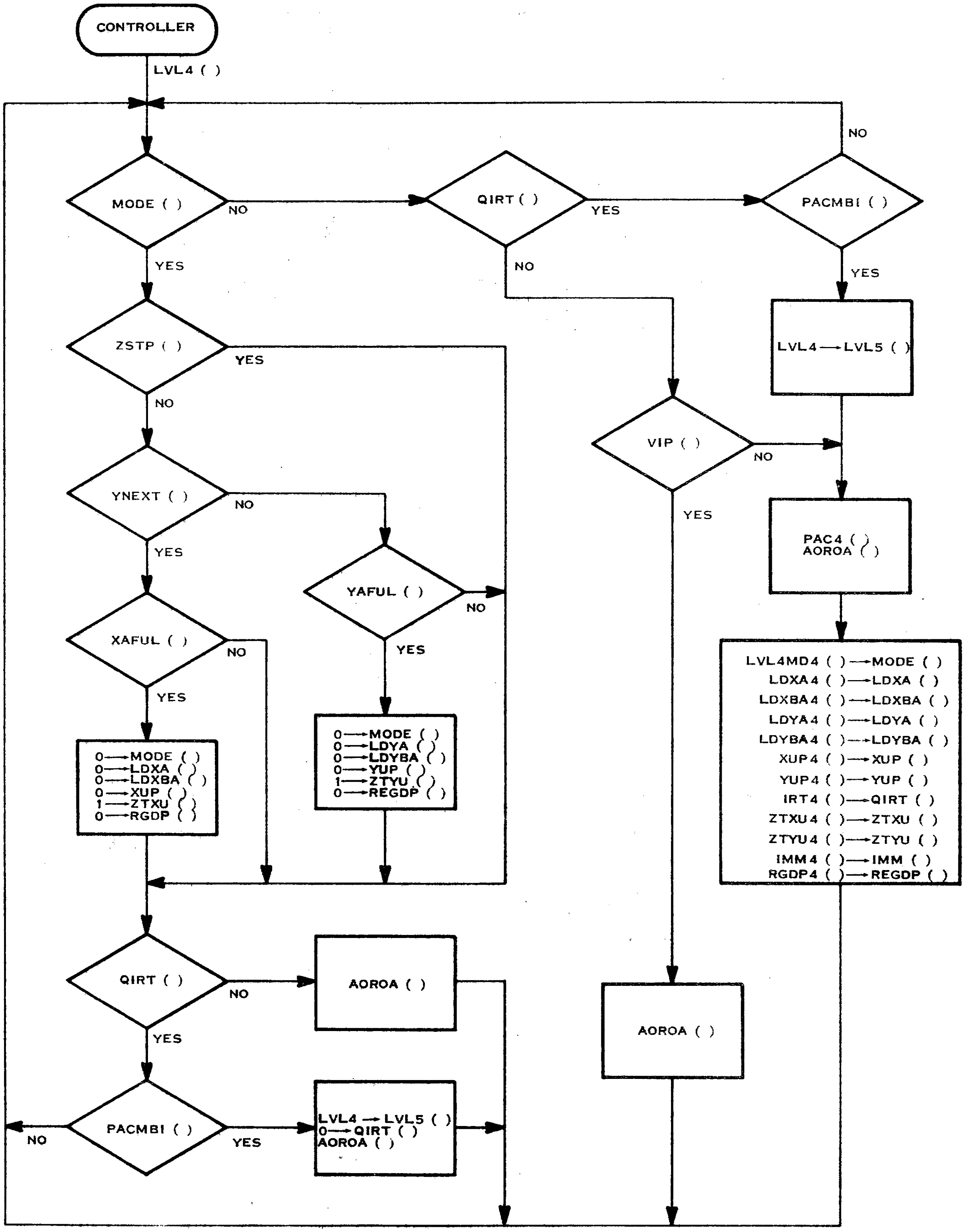
(B)132351

Figure 4-28. Compare Code Register Controller Flowchart



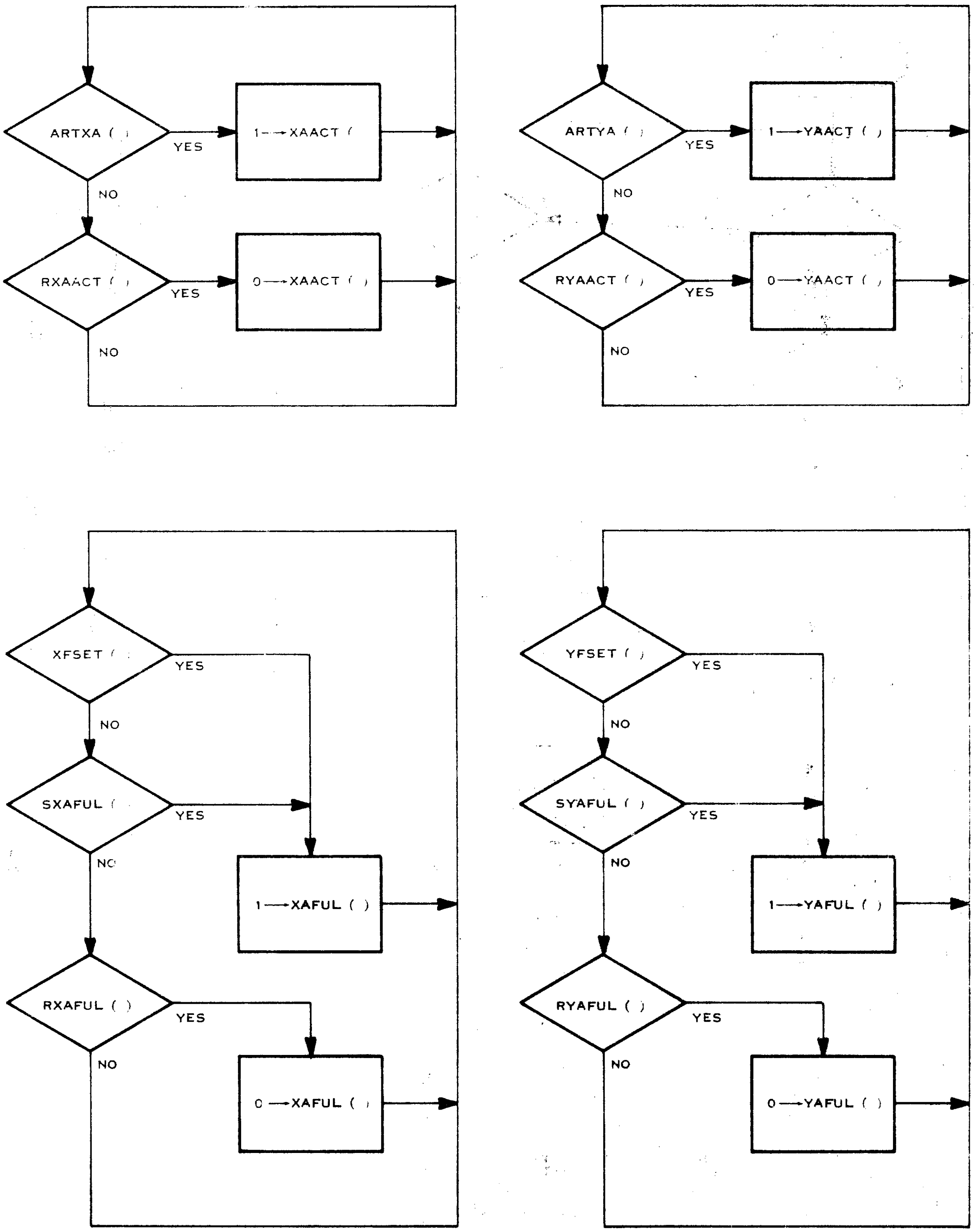
(B)132352

Figure 4-29. Result Code Register Controller Flowchart



(B)127602

Figure 4-30. Level 4 Controller Flowchart



(B)127603

Figure 4-31. Level 4 IPU Models

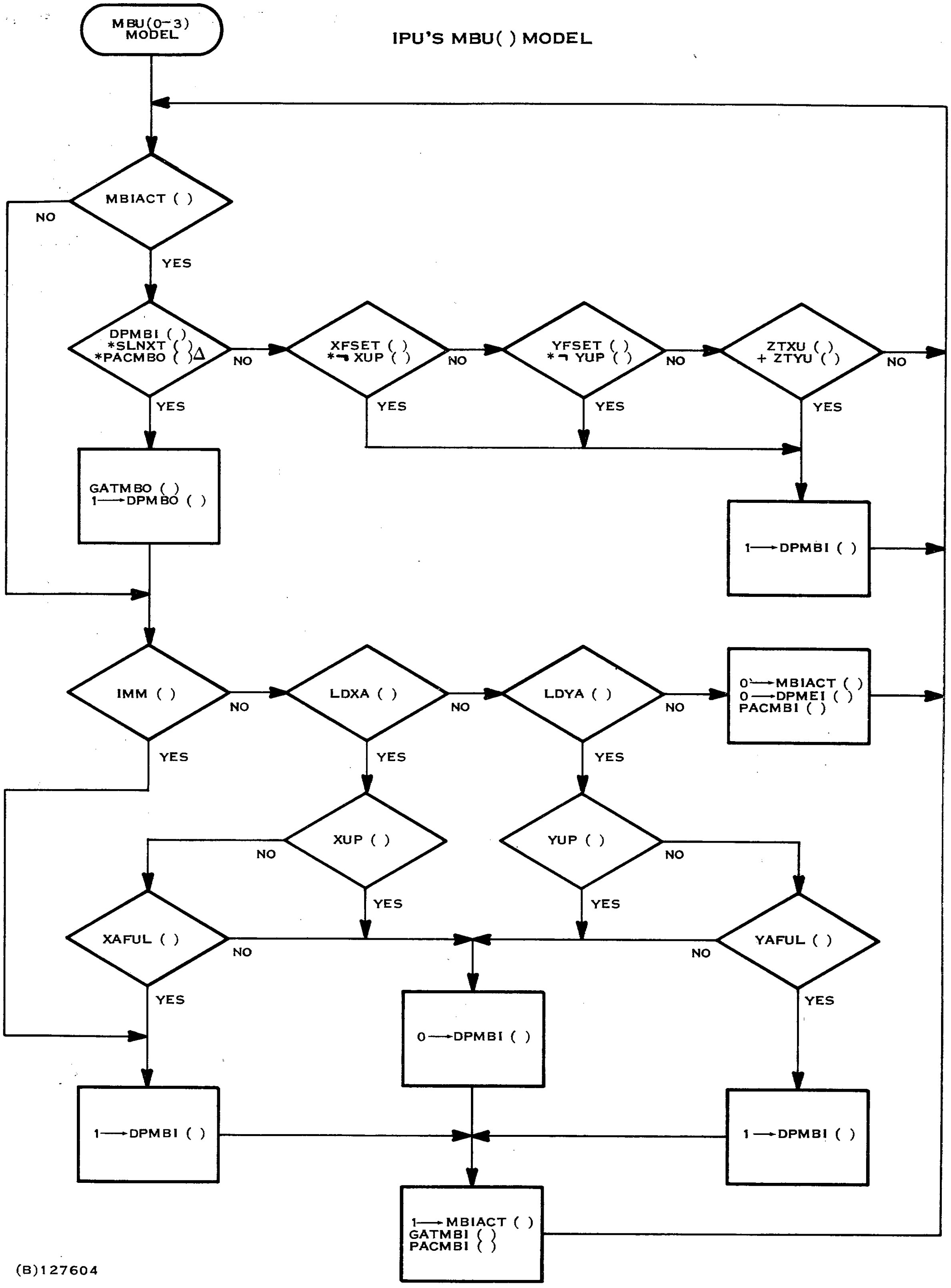


Figure 4-32. Level 4 MBU Model

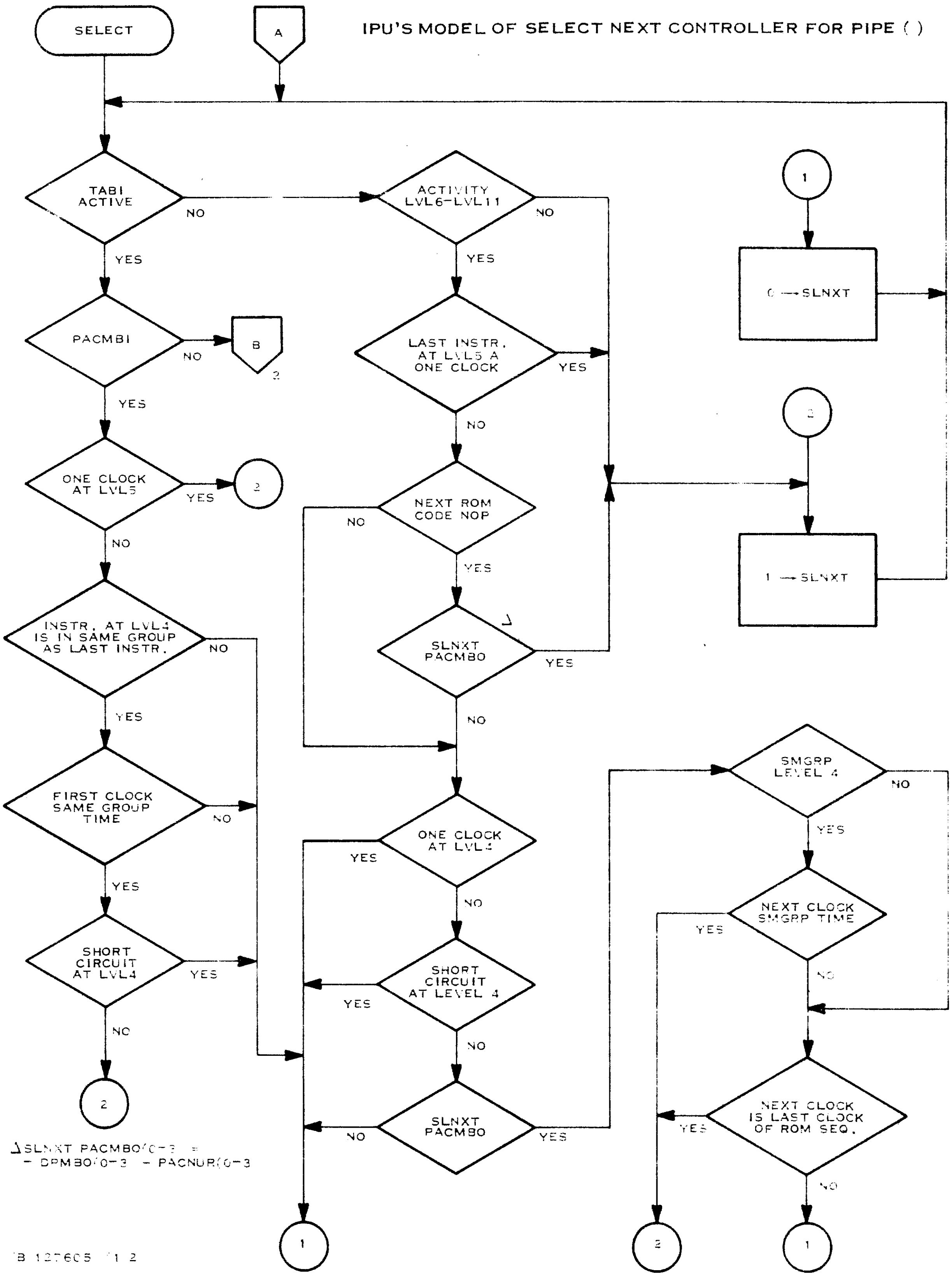
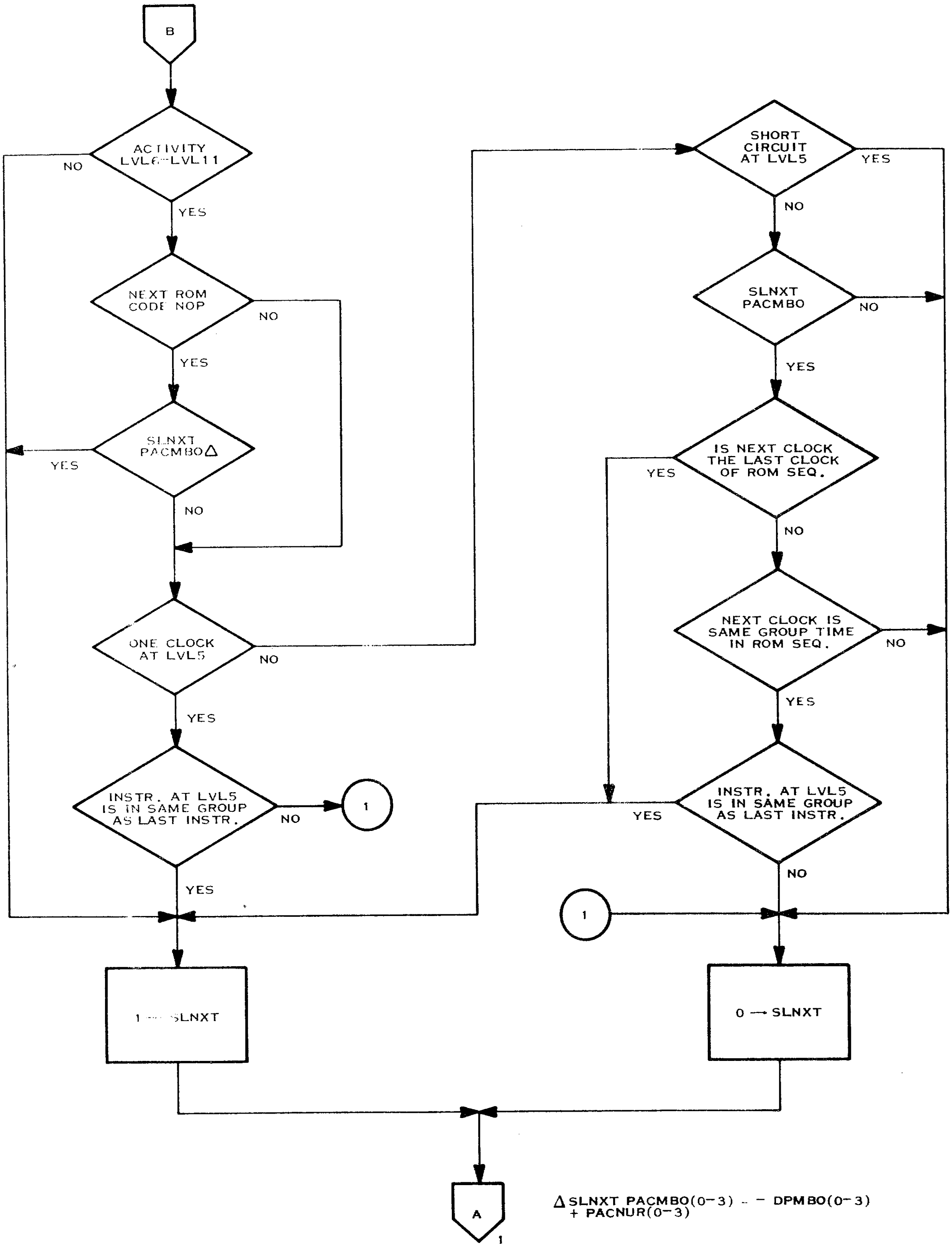


Figure 4-33. Level 4 Select Next Controller Model (Sheet 1 of 2)


B
ACTIVITY LVL6-LVL11
NO
YES
NEXT ROM CODE NOP
NO
YES
SLNXT PACMBOΔ
YES
NO
ONE CLOCK AT LVL5
NO
YES
INSTR. AT LVL5 IS IN SAME GROUP AS LAST INSTR.
NO
1
YES
1 → SLNXT
SHORT CIRCUIT AT LVL5
YES
NO
SLNXT PACMBO
NO
YES
IS NEXT CLOCK THE LAST CLOCK OF ROM SEQ.
YES
NO
NEXT CLOCK IS SAME GROUP TIME IN ROM SEQ.
NO
YES
INSTR. AT LVL5 IS IN SAME GROUP AS LAST INSTR.
YES
NO
1
0 → SLNXT
A
1

ΔSLNXT PACMBO(0-3) - - DPMBO(0-3)
+ PACNUR(0-3)

Figure 4-33. Level 4 Select Next Controller Model (Sheet 2 of 2)

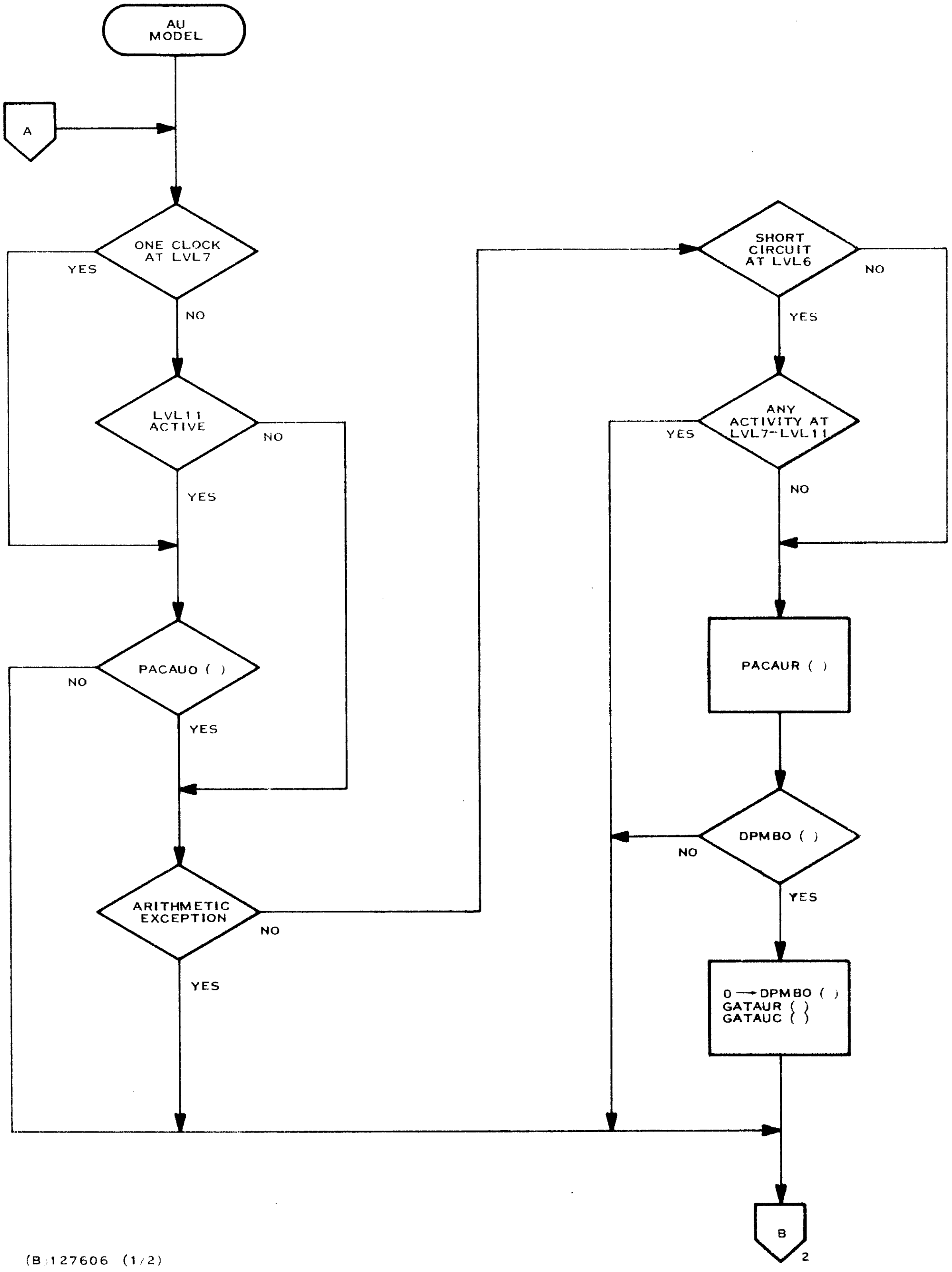
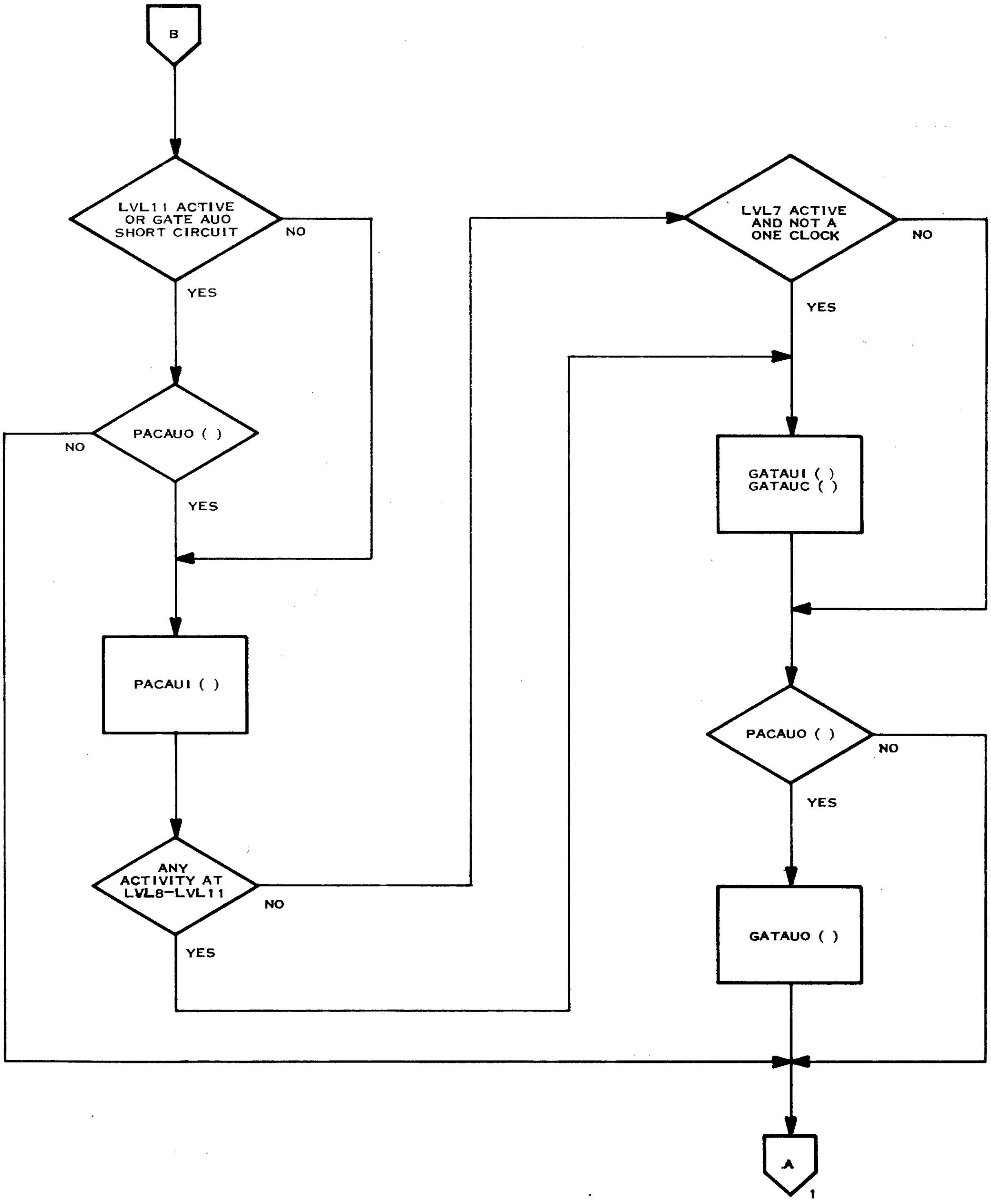


Figure 4-34. Level 4 IPU's AU Model (Sheet 1 of 2)


B
LVL11 ACTIVE OR GATE AUO SHORT CIRCUIT
NO
YES
PACAUO ( )
NO
YES
PACAUI ( )
ANY ACTIVITY AT LVL8–LVL11
NO
YES
LVL7 ACTIVE AND NOT A ONE CLOCK
NO
YES
GATAUI ( )
GATAUC ( )
PACAUO ( )
NO
YES
GATAUO ( )
A
1

(B)127606 (2/2)

Figure 4-34. Level 4 IPU's AU Model (Sheet 2 of 2)

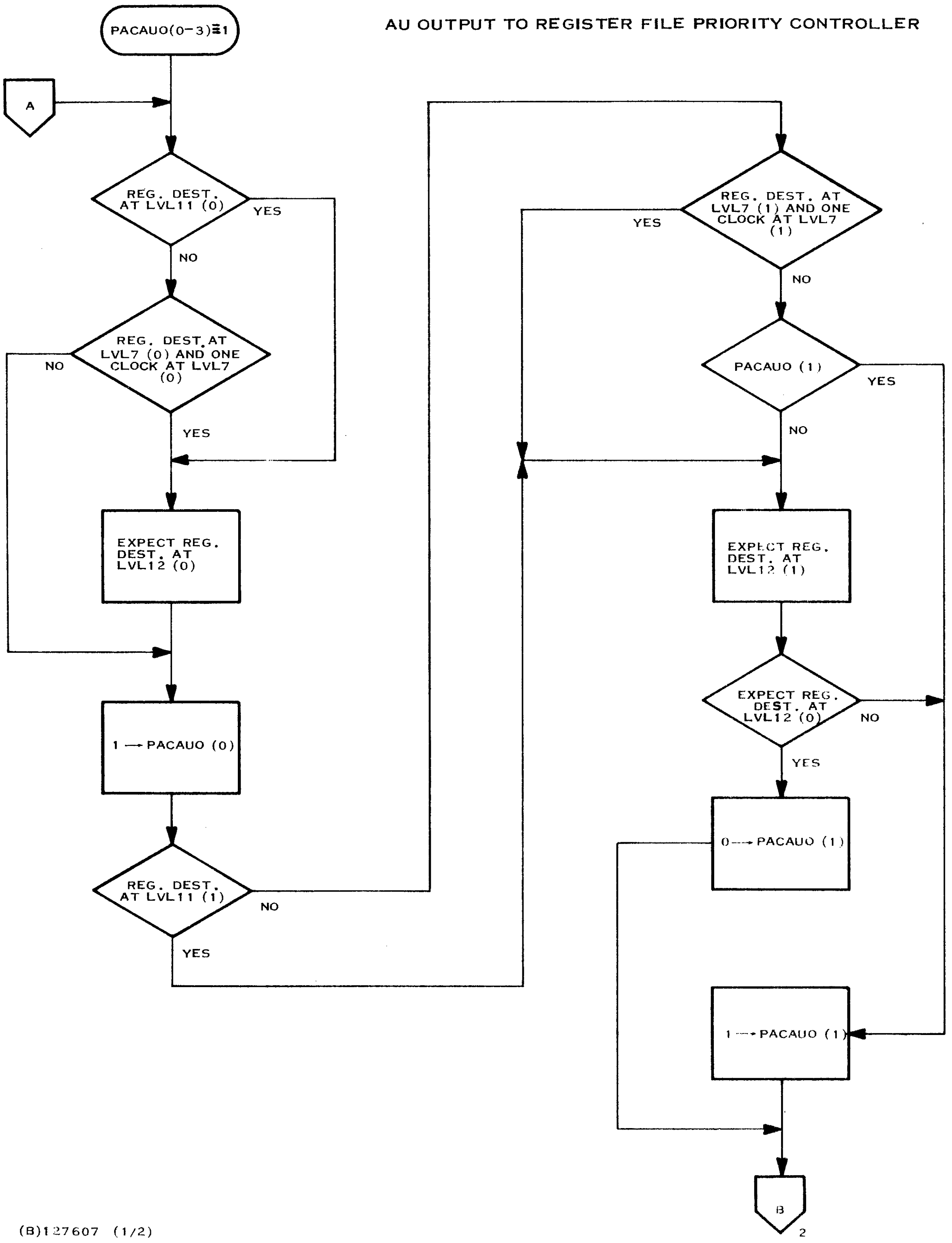
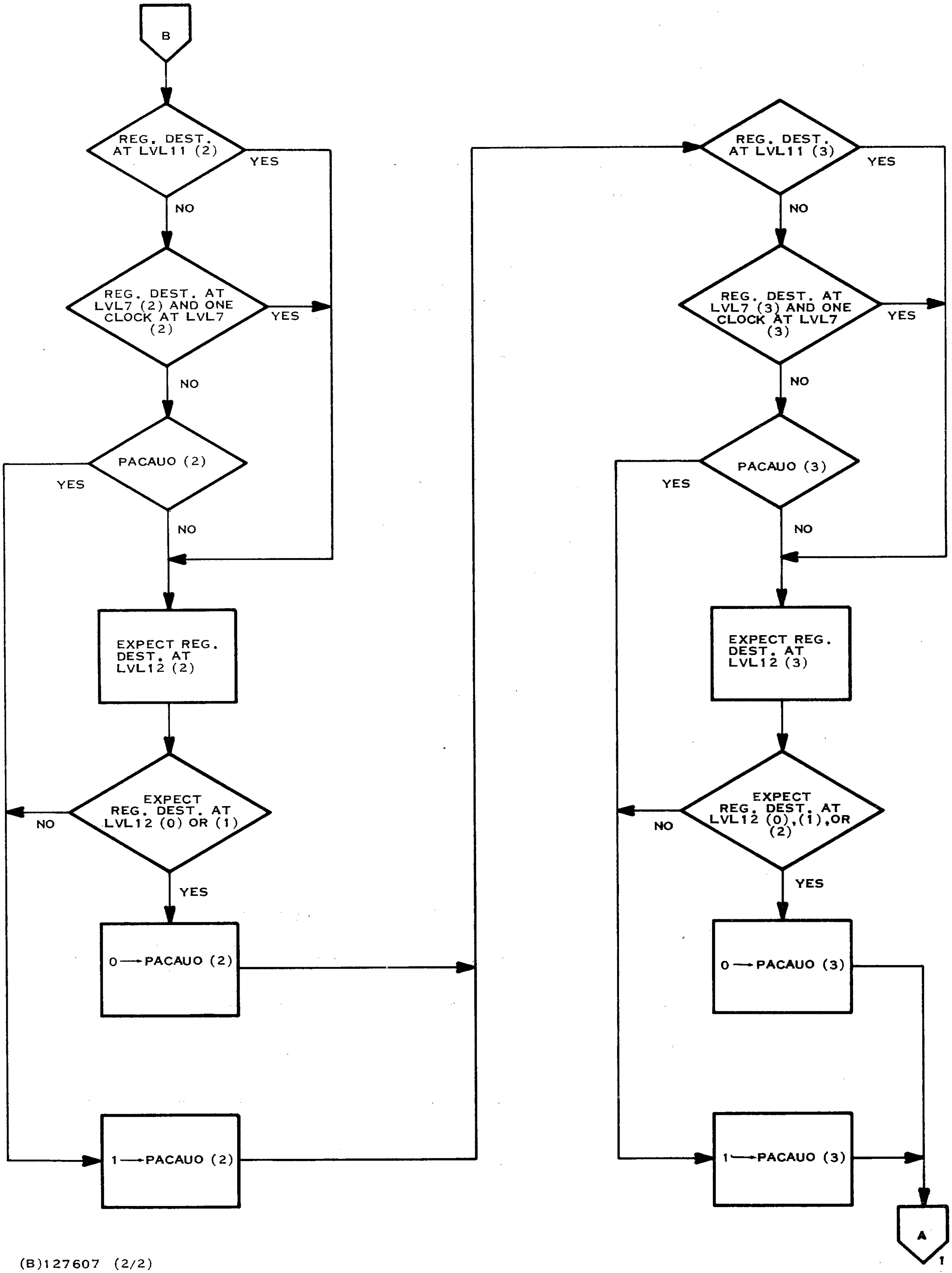
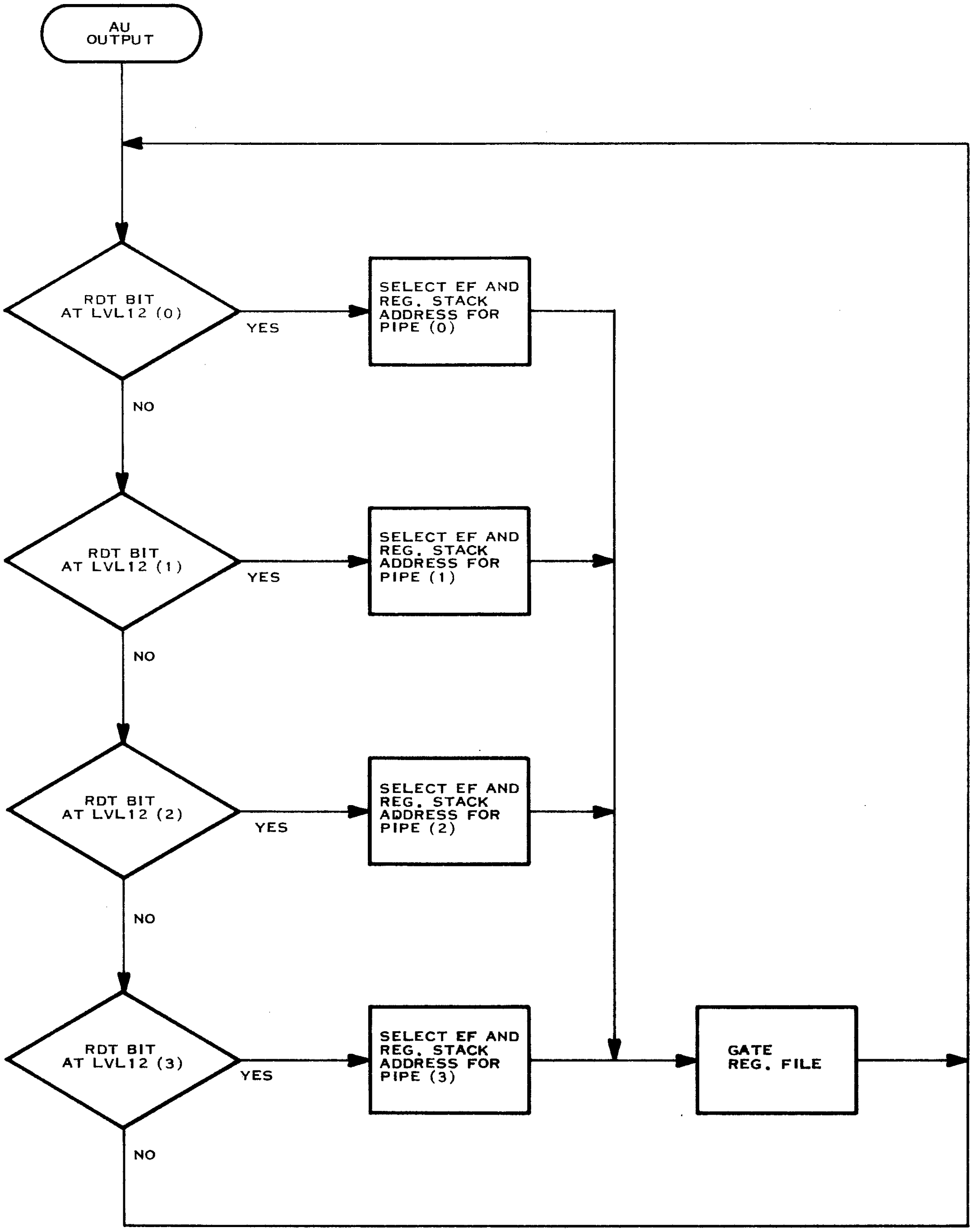

AU OUTPUT TO REGISTER FILE PRIORITY CONTROLLER
PACAUO(0-3)≡1
A
REG. DEST. AT LVL11 (0)
YES
NO
REG. DEST. AT LVL7 (0) AND ONE CLOCK AT LVL7 (0)
NO
YES
EXPECT REG. DEST. AT LVL12 (0)
1 → PACAUO (0)
REG. DEST. AT LVL11 (1)
NO
YES
REG. DEST. AT LVL7 (1) AND ONE CLOCK AT LVL7 (1)
YES
NO
PACAUO (1)
YES
NO
EXPECT REG. DEST. AT LVL12 (1)
EXPECT REG. DEST. AT LVL12 (0)
NO
YES
0 → PACAUO (1)
1 → PACAUO (1)
B
2
(B)127607 (1/2)

Figure 4-35. Level 4 PACAUO = 1 (Sheet 1 of 2)


B
REG. DEST. AT LVL11 (2)
YES
NO
REG. DEST. AT LVL7 (2) AND ONE CLOCK AT LVL7 (2)
YES
NO
PACAUO (2)
YES
NO
EXPECT REG. DEST. AT LVL12 (2)
EXPECT REG. DEST. AT LVL12 (0) OR (1)
NO
YES
0 → PACAUO (2)
1 → PACAUO (2)
REG. DEST. AT LVL11 (3)
YES
NO
REG. DEST. AT LVL7 (3) AND ONE CLOCK AT LVL7 (3)
YES
NO
PACAUO (3)
YES
NO
EXPECT REG. DEST. AT LVL12 (3)
EXPECT REG. DEST. AT LVL12 (0),(1),OR (2)
NO
YES
0 → PACAUO (3)
1 → PACAUO (3)
A
1

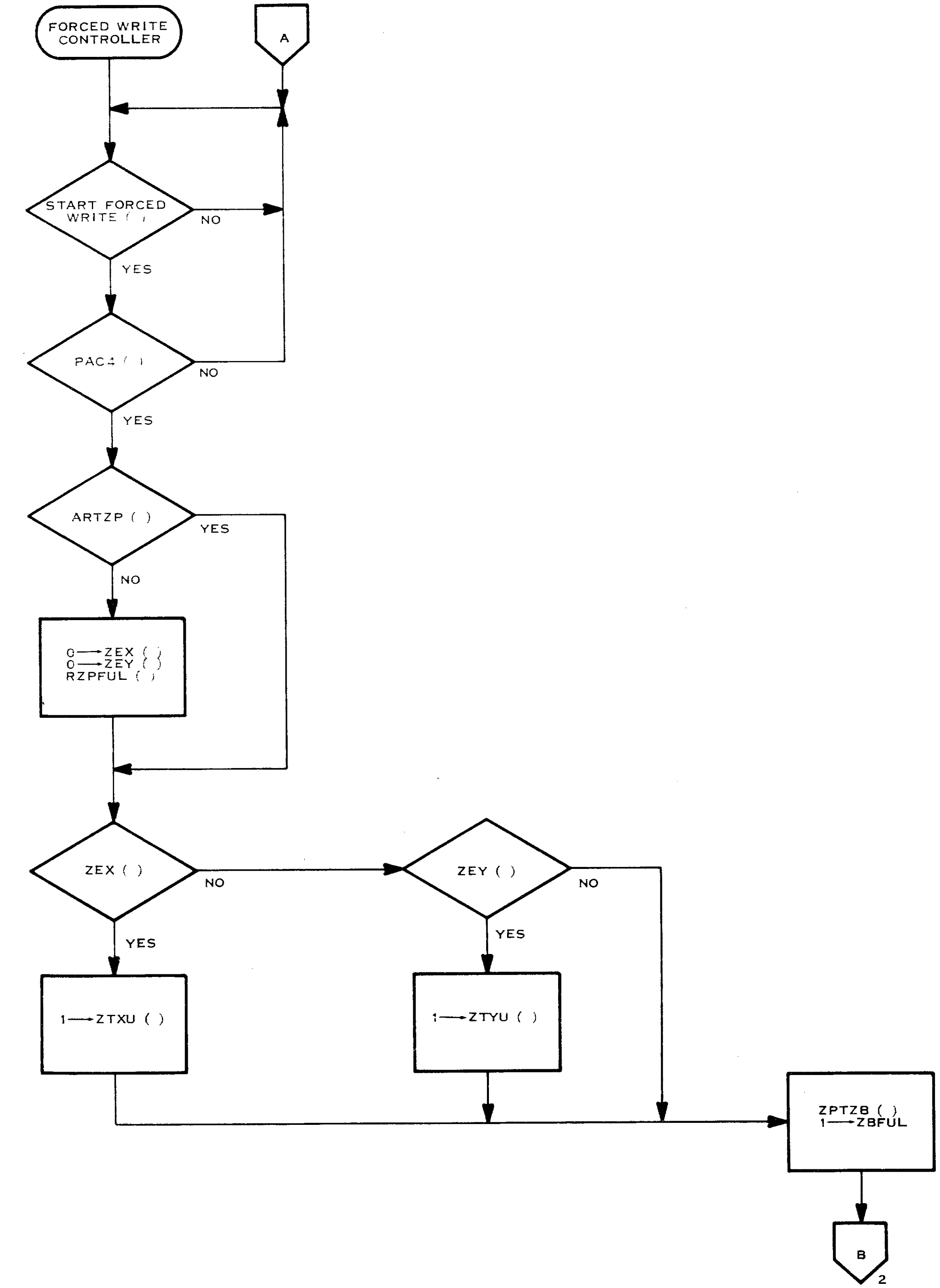
(B)127607 (2/2)

Figure 4-35. Level 4 PACAUO = 1 (Sheet 2 of 2)


AU OUTPUT
RDT BIT AT LVL12 (0)
YES
NO
SELECT EF AND REG. STACK ADDRESS FOR PIPE (0)
RDT BIT AT LVL12 (1)
YES
NO
SELECT EF AND REG. STACK ADDRESS FOR PIPE (1)
RDT BIT AT LVL12 (2)
YES
NO
SELECT EF AND REG. STACK ADDRESS FOR PIPE (2)
RDT BIT AT LVL12 (3)
YES
NO
SELECT EF AND REG. STACK ADDRESS FOR PIPE (3)
GATE REG. FILE

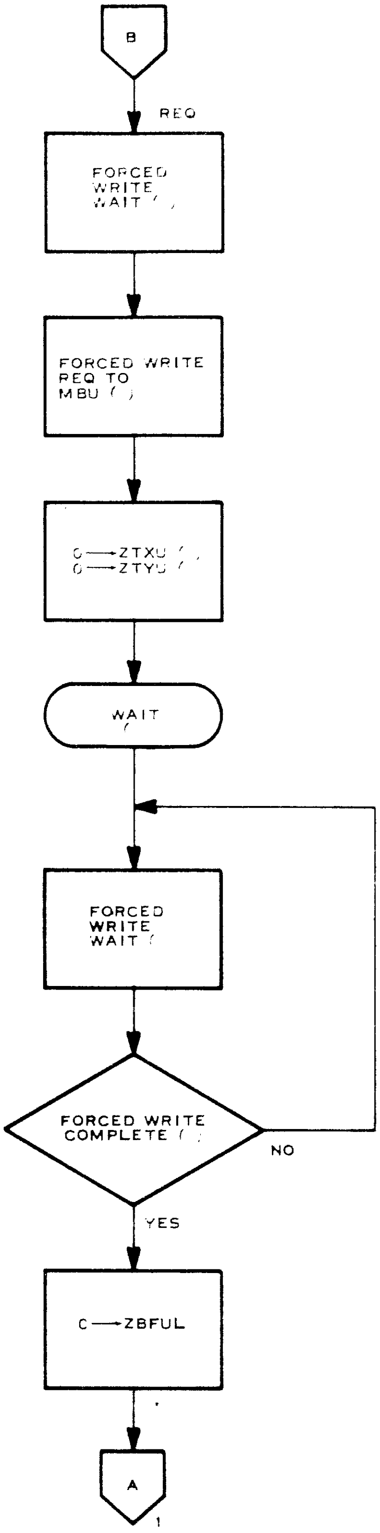
(B)127608

Figure 4-36. Level 4 AU Output to File



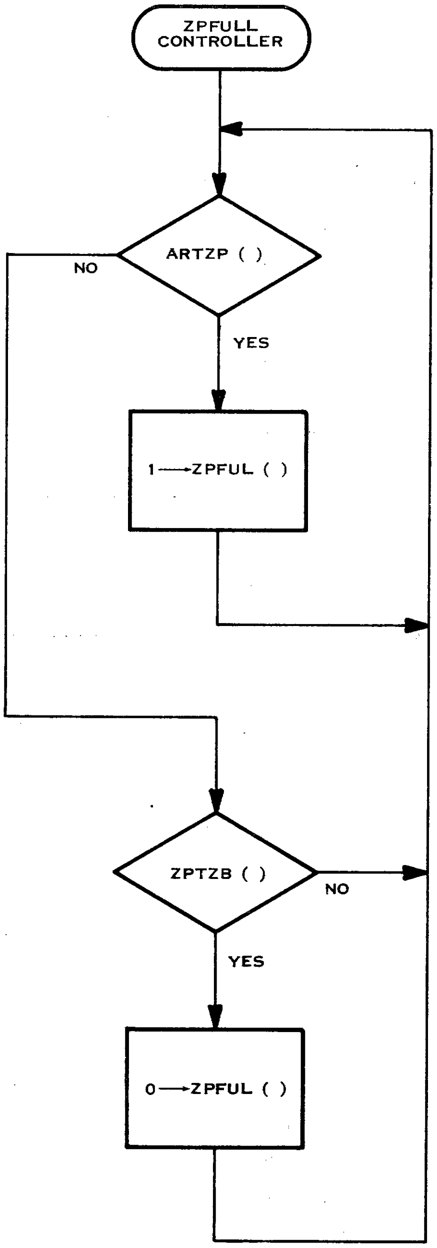
(B 127609 (1/2)

Figure 4-37. Level 4 Forced Write Controller (Sheet 1 of 2)



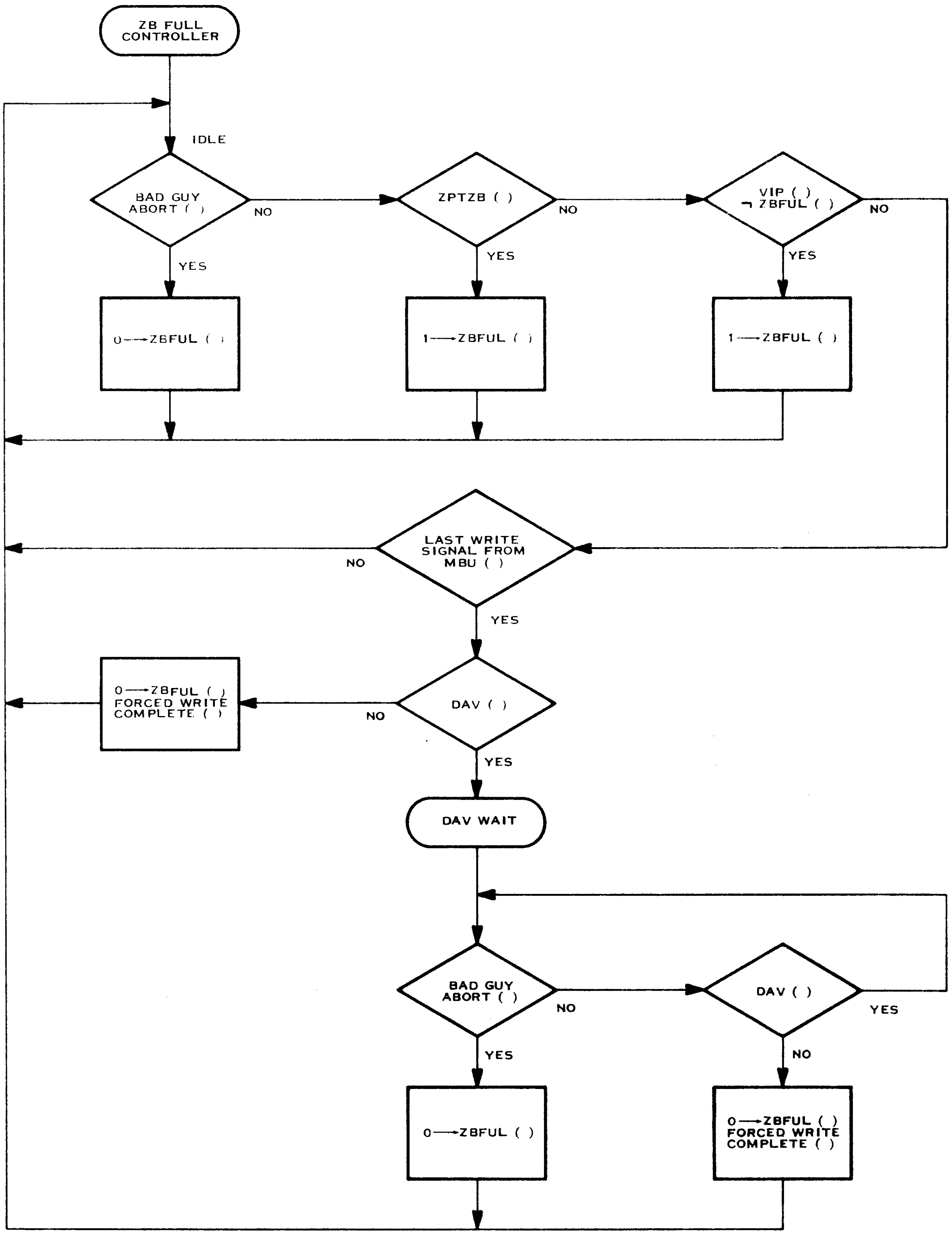
(B 127609 (2/2)

Figure 4-37. Level 4 Forced Write Controller (Sheet 2 of 2)



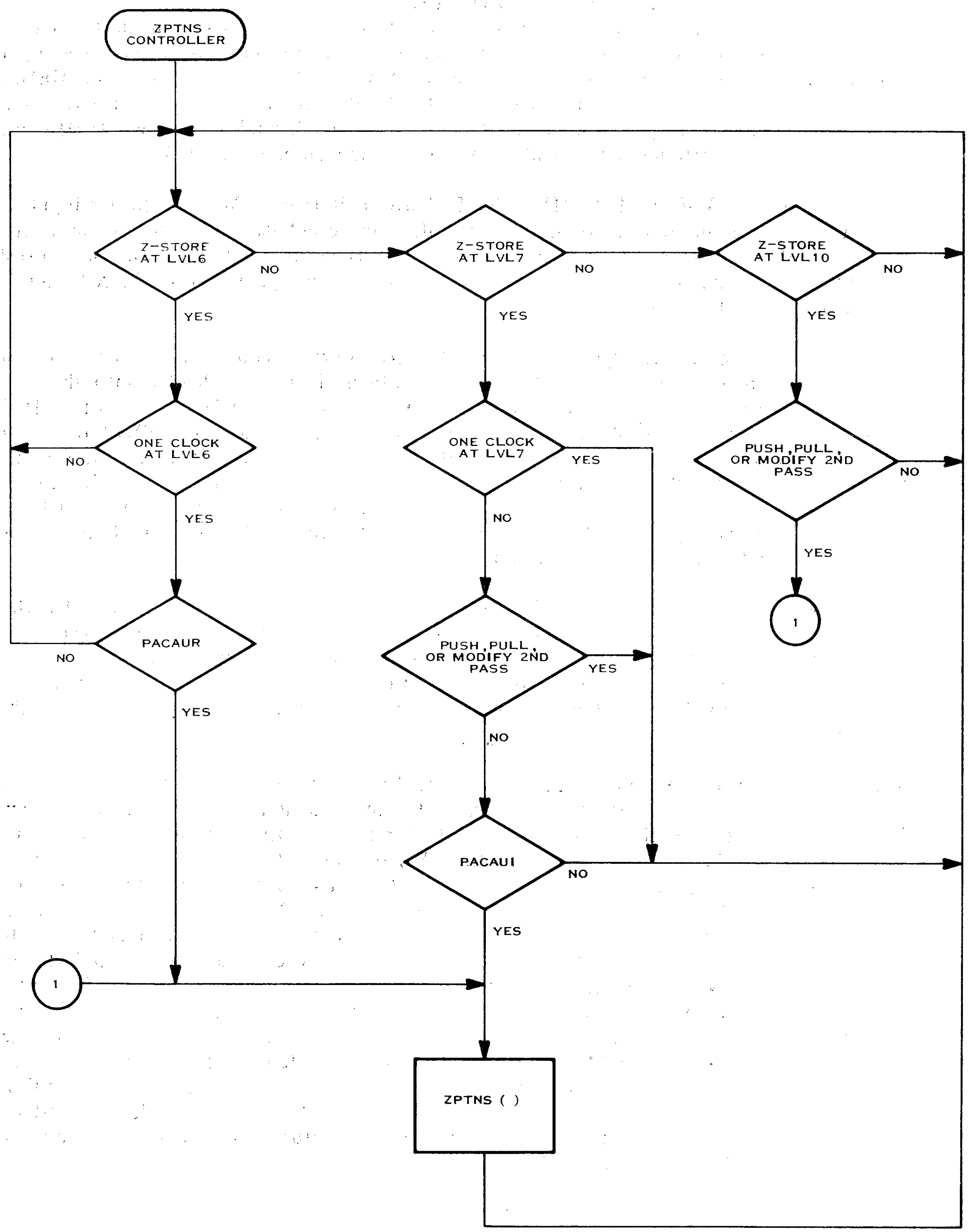
(B)127610

Figure 4-38. Level 4 ZPFULL Controller



(B)127611

Figure 4-39. Level 4 ZBFULL Controller



(B)127612

Figure 4-40. Level 4 ZPTNS Controller

4.23 LEVEL 5 SCALAR INPUT CONTROLLER

The level 5 scalar input controller monitors the interface control bits from the IPU levels 3 and 4 and enables data transfers to perform the specified loading operations. The operations performed are further qualified by status inputs from level 5 and level 6 of the MBU. Figure 4-41 illustrates the control paths involved in the level 5 input controller. Since vector loading operations are handled by a different controller that is not dependent on the IPU-MBU interface control bits, this controller is responsible only for scalar loading of the MBU.

4.23.1 INPUT STAGE NOT ACTIVE (NOT MBIAC). If level 5 (Memory Buffer Input, MBI) is not busy with an input operation, the controller can accept data and address inputs from the IPU. The IPU inputs can specify any one of three types of operations for the controller to perform: load an immediate operand from the IPU, load an operand from the X buffer, or load an operand from the Y buffer.

4.23.1.1 Load Immediate Operand. If the IMMED interface control bit is set from the IPU, then the AO register in level 4 of the IPU contains data to be loaded into the pipe. The controller enables the transfer of the data in AO directly into the IMM register in the MBU input stage, sets IMFUL to indicate that IMM contains valid data, and sets DPMBI to indicate the presence of data in the input stage to the MBU. The controller next examines the Register Data Present (REGDP) interface control bit to determine if the RO register in the IPU also contains data to be placed into the pipe. If REGDP is set, the controller enables transfer of the data in RO into the REG register in the input stage of the MBU, and sets RGFUL to indicate the presence of valid data in the REG register.

Regardless of the state of the REGDP bit, the controller sets MBIAC to indicate that the input stage contains valid operands to be transferred to level 6, enables the Op Code from the RX or RY register at IPU level 4 to be transferred to the ROM Address Register, allows the word size indicator from the IPU to select the operand word size to be used during the operation in the AU, and issues PACMBI to the IPU indicating that the MBU is ready to examine the next instruction.

4.23.1.2 Load From X Buffer (LDXA). If the LDXA interface control bit is set, the level 5 controller must select a word from the X buffer and load it into the MAB output register at level 6. To prepare for this transfer, the controller transfers the address of the word from the AO register into the XA register in the MBU. XA output will then be able to select the correct operand from the X buffer when level 5 to level 6 occurs. The controller then sets XAFUL to indicate that a valid address is in XA, and clears YFRST. YFRST is a pointer that indicates which buffer will receive the next input octet from memory during scalar operations. When set, YFRST indicates that the Y buffer will receive the input; when clear, YFRST indicates that the X buffer will receive the input. The controller must then determine if the X buffer contains the correct octet. If LDXBA is set, a memory octet must first be transferred into the X buffer before the word can be selected. The controller, therefore, transfers the address of the octet from the AO register to XBA for input to the Central Memory Requester, sets XBREQ to indicate that CMR will have to perform a fetch for the octet, and clears XFUL to indicate that the octet currently in the X buffer is not the desired octet. The controller sets MBIAC as it leaves this control loop so that it waits for the octet to return from memory before performing any further operations with the level 5 data.

If LDXBA was not set, then no memory fetch will be required to load the X buffer. The controller then checks that the X buffer contains valid data (XFUL). If XFUL is set, or if it is clear and a Z to X update is required (ZTXU), thereby setting XFUL, the controller checks the XUP interface control bit. If this bit is set, the update operation has not yet completed. The

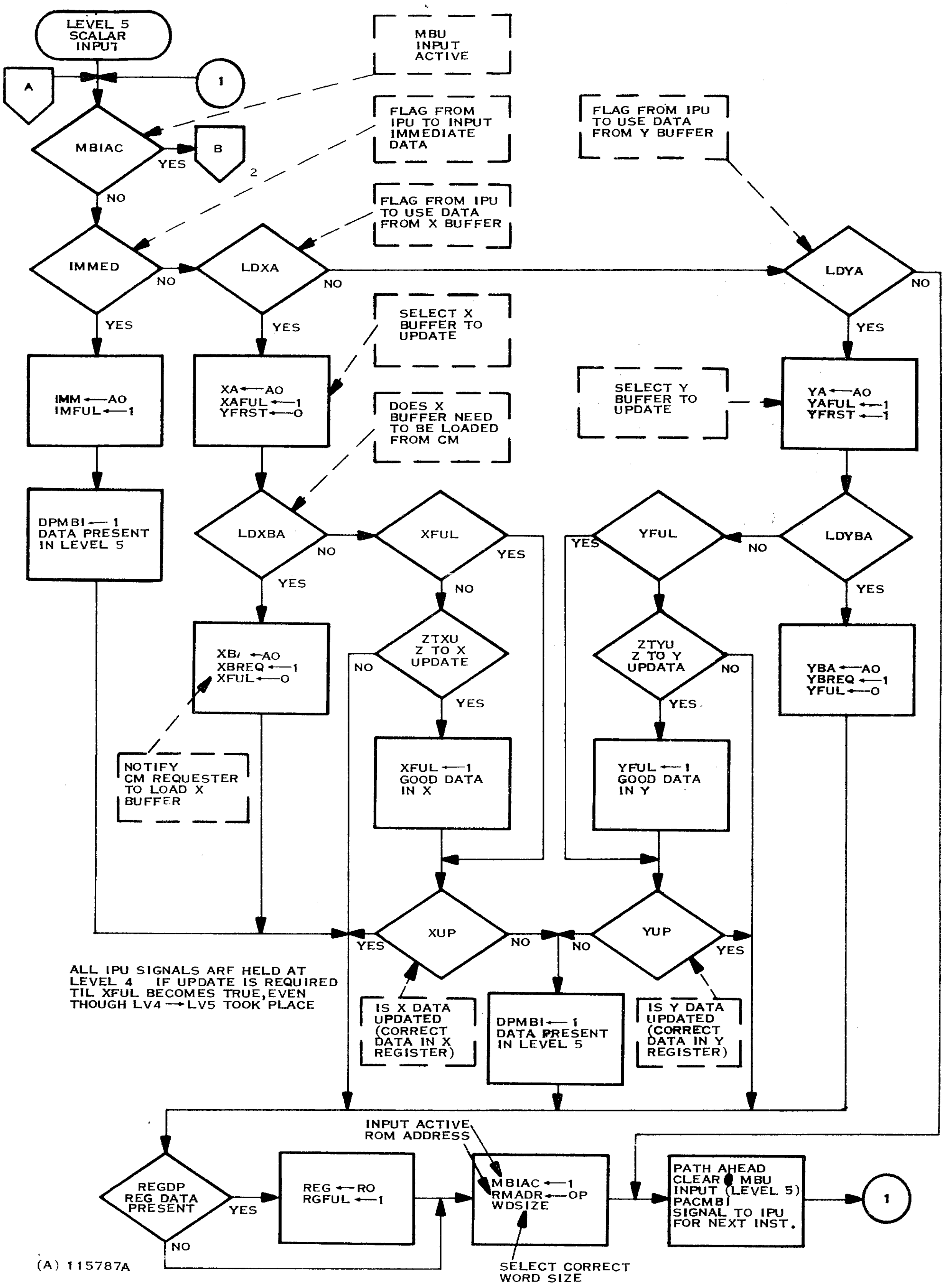

LEVEL 5 SCALAR INPUT
MBU INPUT ACTIVE
A
1
MBIAC
YES
B
2
NO
FLAG FROM IPU TO INPUT IMMEDIATE DATA
FLAG FROM IPU TO USE DATA FROM Y BUFFER
FLAG FROM IPU TO USE DATA FROM X BUFFER
IMMED
NO
LDXA
NO
LDYA
NO
YES
YES
YES
SELECT X BUFFER TO UPDATE
IMM←AO
IMFUL←1
XA←AO
XAFUL←1
YFRST←O
DOES X BUFFER NEED TO BE LOADED FROM CM
SELECT Y BUFFER TO UPDATE
YA←AO
YAFUL←1
YFRST←1
DPMBI←1
DATA PRESENT IN LEVEL 5
LDXBA
NO
XFUL
YES
YES
YFUL
NO
LDYBA
YES
NO
YES
XB/ ←AO
XBREQ←1
XFUL←O
ZTXU
Z TO X
UPDATE
NO
YES
ZTYU
Z TO Y
UPDATA
NO
YES
YBA←AO
YBREQ←1
YFUL←O
NOTIFY CM REQUESTER TO LOAD X BUFFER
XFUL←1
GOOD DATA IN X
YFUL←1
GOOD DATA IN Y
XUP
YES
NO
YUP
NO
YES
ALL IPU SIGNALS ARF HELD AT LEVEL 4 IF UPDATE IS REQUIRED TIL XFUL BECOMES TRUE,EVEN THOUGH LV4→LV5 TOOK PLACE
IS X DATA UPDATED (CORRECT DATA IN X REGISTER)
DPMBI←1
DATA PRESENT IN LEVEL 5
IS Y DATA UPDATED (CORRECT DATA IN Y REGISTER)
INPUT ACTIVE ROM ADDRESS
REGDP
REG DATA
PRESENT
YES
NO
REG←RO
RGFUL←1
MBIAC←1
RMADR←OP
WDSIZE
PATH AHEAD CLEAR MBU INPUT (LEVEL 5) PACMBI SIGNAL TO IPU FOR NEXT INST.
1
SELECT CORRECT WORD SIZE
(A) 115787A

Figure 4-41. Level 5 Scalar Input Controller Flowchart (Sheet 1 of 5)

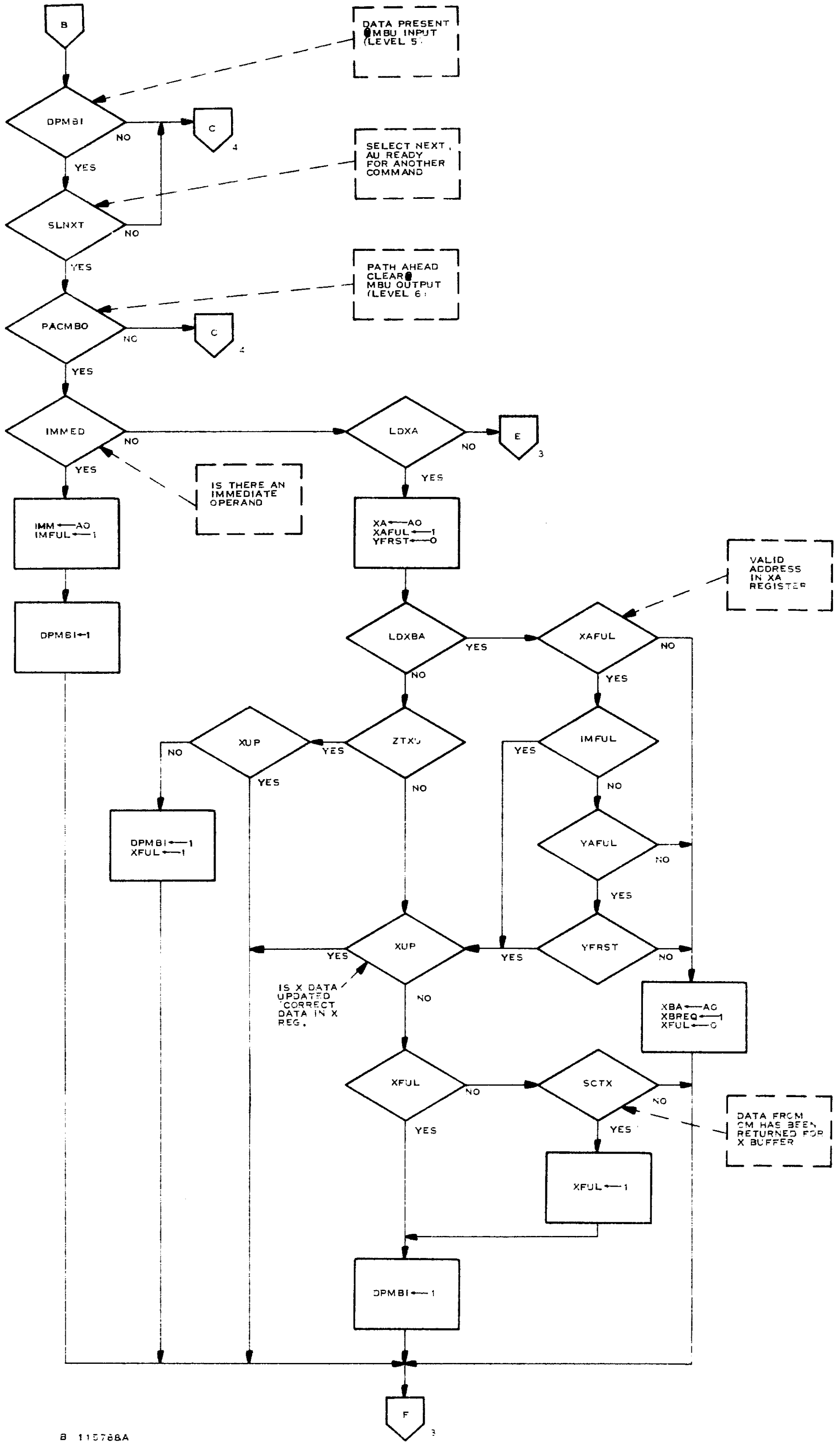
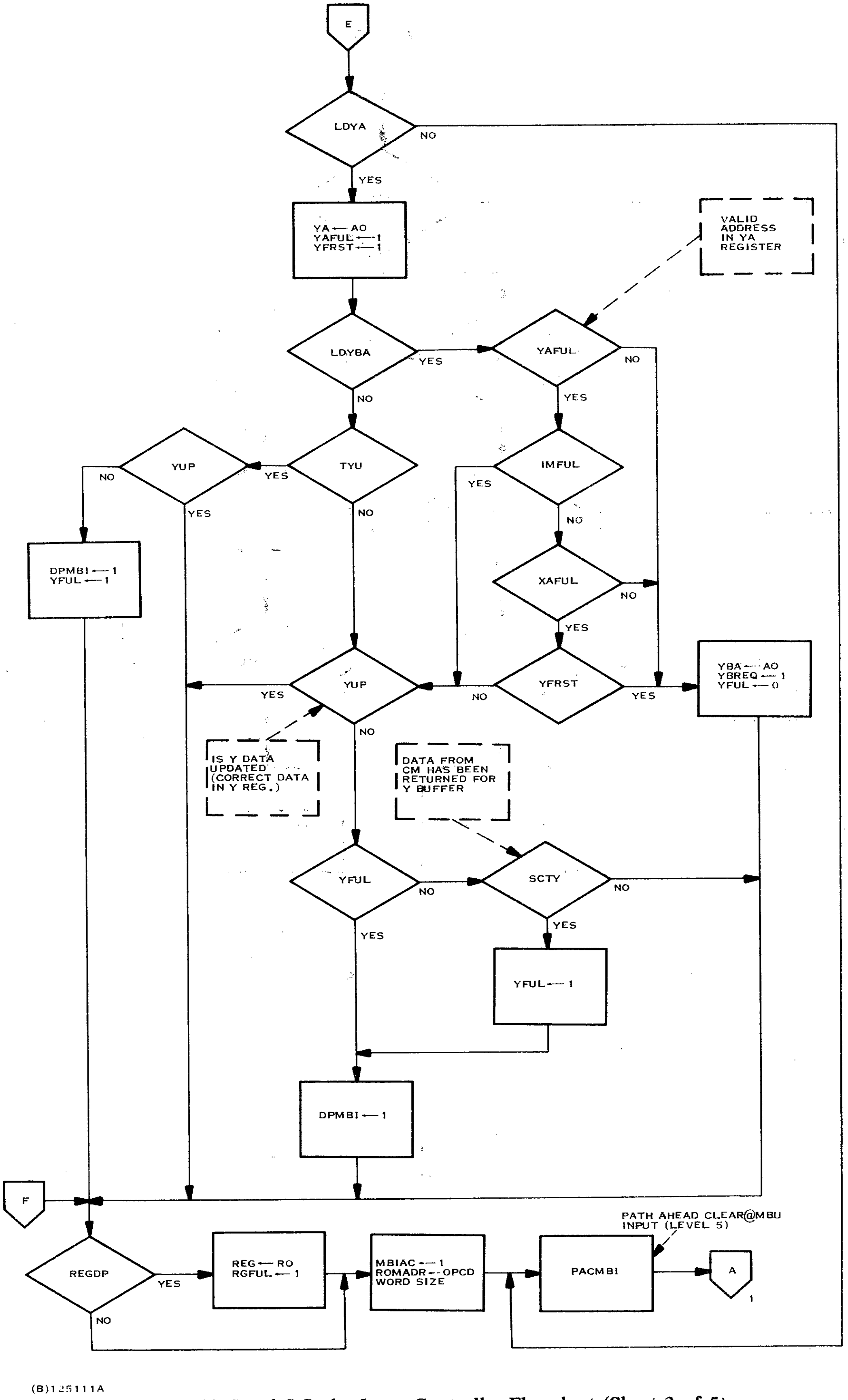
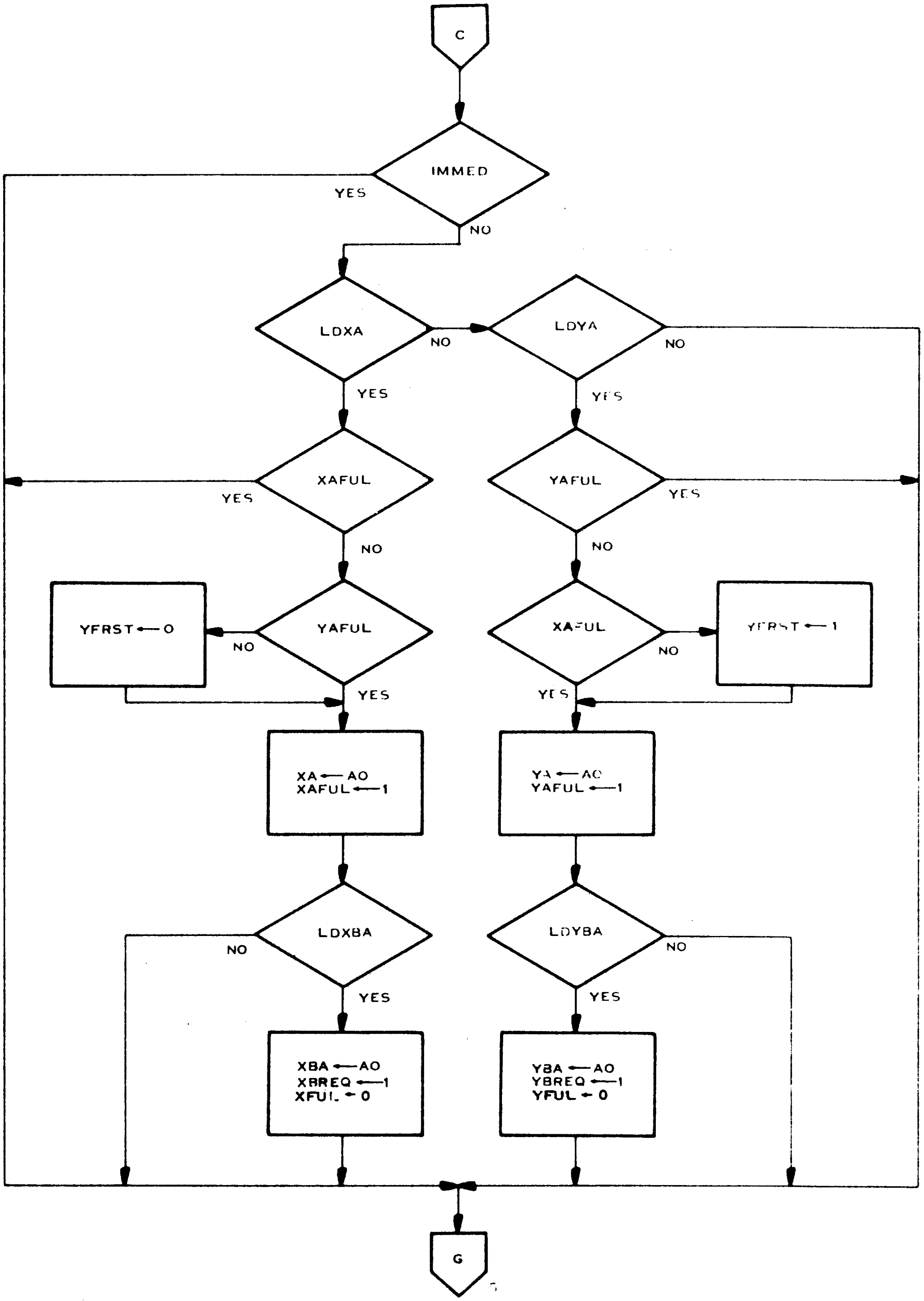


Figure 4-41. Level 5 Scalar Input Controller Flowchart (Sheet 2 of 5)



(B)125111A

Figure 4-41. Level 5 Scalar Input Controller Flowchart (Sheet 3 of 5)



(B) 115789

Figure 4-41. Level 5 Scalar Input Controller Flowchart (Sheet 4 of 5)

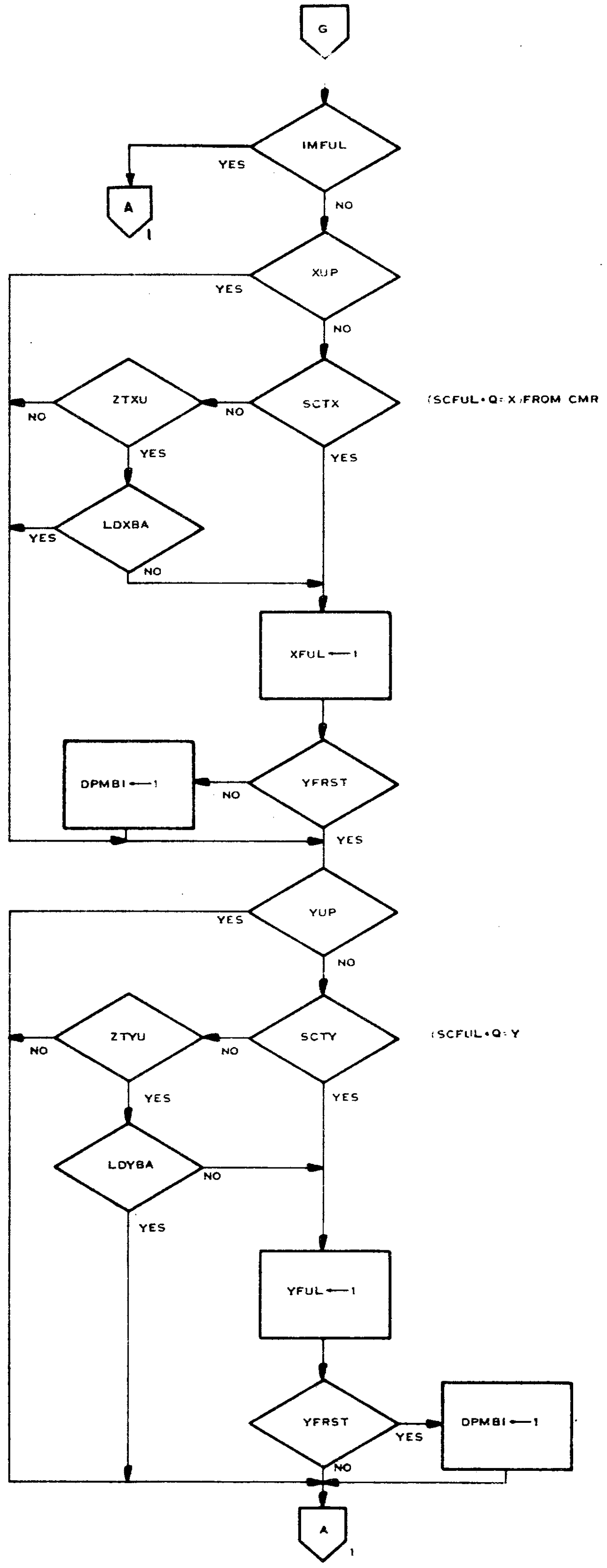


Figure 4-41. Level 5 Scalar Input Controller Flowchart (Sheet 5 of 5)

controller will wait for XUP to clear before performing any additional operations with level 5 operands. If XUP is not set, then the data in X will not be changed by an update in progress or is being changed by ZTXU update and will, therefore, represent a valid octet. The controller sets DPMBI to indicate the presence of valid data in the input stage. Regardless of the path taken through the load from X sequence, the controller completes the cycle by loading REG with data from RO and setting RGFUL (if the REGDP interface control bit is set), setting MBIAC to indicate activity at level 5, transferring the Op Code from RX to the ROM Address register, selecting the word size to be used for the operands from the word size indicator in the MBU, and generating PACMBI to indicate to the IPU that the controller can examine a new instruction.

4.23.1.3 Load From Y Buffer (LDYA). The LDYA control paths are identical to the LDXA control paths, except that the action and decision blocks involve the status and registers associated with the Y buffer instead of the X buffer. Refer to the description of the Load From X Buffer (LDXA) control sequence for an understanding of the LDYA sequence.

4.23.2 INPUT STAGE ACTIVE (MBIAC). If MBIAC is set when the controller begins the control cycle, then level 5 contains a valid operation to be performed. The controller can then enter one of two routines depending upon whether the operation at level 5 will be passed to level 6 during the next clock pulse. To determine this condition, the controller examines DPMBI (Data Present MBI), SLNXT (Select Next instruction) and PACMBO (PAC from level 6). If all of these signals are true, then the operation at level 5 will pass to level 6 during the next clock. If any one of these conditions is not met, the operation at level 5 will not pass to level 6.

4.23.2.1 Transfer OK. If the next clock pulse will transfer the operation in level 5 into level 6, then the controller can examine the incoming instruction and prepare to route it into level 6. The controller, therefore, determines if the instruction in the output of the IPU is an immediate operand, requires a load from the X buffer, or requires a load from the Y buffer.

For immediate operands, the controller enables the next clock pulse to transfer the contents of AO to the output stage of the IPU into the IMM register in level 5 of the MBU, and sets IMFUL to indicate the presence of valid data in the IMM register. The controller also sets DPMBI to indicate the presence of valid data in level 5, before proceeding to examine the REGDP control bit. If there is an operand in the RO register (REGDP set) the controller enables that operand into the REG register in the MBU and sets RGFUL to indicate the presence of data in that register. Regardless of the state of REGDP, the controller sets MBIAC to indicate the new operation in level 5, enables the op code portion of the instruction from RX or RY in level 4 to the ROM Address register in the MBU, and uses the word size indication from the IPU to select the operand size to be sent to the AU. The controller issues PACMBI to the IPU before returning to the start of the control cycle.

If the new instruction from the IPU requires a load into level 6 from either the X or the Y buffer (LDXA or LDYA), the controller transfers the address of the operand from the AO register in the IPU to the respective operand address register (XA or YA) to select the correct word from the output of the Buffer. The controller also sets the full flag (YAFUL or XAFUL) associated with that address register to indicate the presence of a valid address in that register and either sets or clears the YFRST pointer, depending upon whether the Y or the X buffer will receive the next operand from memory. If a memory request is required to load an octet into the selected buffer (LDYBA or LDXBA), YAFUL will not be set during the first control cycle for the instruction. The controller then indicates to CMR that a memory request is required (YBREQ or XBREQ), transfers the address from AO into the operand address register (YBA or XBA) and clears YFUL to indicate that the octet in the Y buffer is not valid. If YAFUL is set when the controller takes the LDYBA path, then the instruction in level 5 is using the YA register or the controller has already made the memory request for the level 4 instruction. If YAFUL is set, the controller makes three other inspections to determine if it can request a new octet from memory to satisfy the LDYBA bit. If YAFUL and IMFUL are both true, then the instruction in level 5 is an immediate and the controller has already requested the octet for the instruction at level 4. Therefore, the controller checks the YFUL flag and sets DPMBI if the octet for level 4 instruction has returned from memory. If YAFUL is set, but the instruction at level 5 is not an immediate operand, the controller checks the XAFUL flag. If XAFUL is not set, then the level 5 to level 6 transfer that will occur on the next clock pulse will select a word from the Y buffer, clearing YAFUL. The controller can therefore issue a memory request for an octet to be loaded into the Y buffer. If both YAFUL and XAFUL are set and level 5 is not immediate data, the controller checks the YFRST flag. If YFRST is not set, then the octet to be loaded into the Y buffer for this instruction has already been requested from memory (since the X buffer is first and XA corresponds to the operands in level 5, therefore YA corresponds to the instruction at level 4 and YAFUL indicates that the octet has been ordered). The controller then checks to see if the octet for the Y buffer has returned from memory (SCTY), and if so, it sets YFUL and DPMBI. The controller then sets MBIAC, transfers the op code from RY in the IPU to the ROM Address register and selects the word size of the operand to be used for the operation at level 4. If the REGDP (Register Data Present) flag is set, the controller enables the transfer of the register operand from RO to the REG register and sets RGFUL. Regardless, the controller sends PACMBI to the level 4 controller so that not only will the next clock pulse transfer the operands of the level 5 operation into level 6, but it will also transfer the instruction at level 4 into level 5.

If a memory fetch is not required to furnish the operand for the instruction at level 4, (not LDYBA), the controller checks the YUP control bit from the level 4 controller. If this bit is set, the controller waits until YUP clears indicating that the Z-to-Y update has been performed and the Y buffer contains the correct data. When YUP clears during a Z-to-Y update (ZTYU), the controller sets DPMBI and YFUL to indicate that the correct data is ready in the Y buffer. If there is no Z-to-Y update and YUP is clear, the controller waits for data to return from memory, sets YFUL, and also sets DPMBI (if data has already returned from memory or was resident in the current buffer, YFUL will already be set). The controller then enables the next clock pulse to transfer level 4 into level 5 along with any register operand that may be in the RO register, sets MBIAC and selects the operand word size.

4.23.2.2 **Transfer Not OK.** If MBIAC is set at the start of a control cycle, but either DPMBI, SLNXT (select next), or PACMBO is not active, then the instruction at level 5 will not transfer to level 6 during the next clock. Therefore, the level 5 controller will not be able to accept a new instruction from level 4. It can, however, request the octet containing the operand from memory if a request is required. If the instruction at level 4 is an immediate operand, then no memory fetch will be needed. The controller, therefore, waits until the instruction at level 5 moves to level 6 before processing an immediate operand.

If the instruction at level 4 requires an operand from either the X or Y buffer (LDXA or LDYA), the controller can prepare the registers for that operation. The requirements of either an LDXA or an LDYA are identical; only one cycle will be described. For an LDXA, the controller first inspects the XAFUL flag to determine if the instruction at level 5 is using the XA register to select its operand from the X buffer. If that is the case, the controller may not load a new address into XA until the level 5 to level 6 transfer occurs. If XA is not full, the controller inspects YAFUL. If YAFUL is not set, the controller clears YFRST to indicate that, since the instruction at level 4 is the only instruction in levels 4 and 5, the X buffer will be the next buffer to receive a memory transfer. If YAFUL is set, the Y buffer will be used first so the controller leaves YFRST set to select that buffer. Regardless of the state of YAFUL, as long as XAFUL is not set, the controller enables the next clock to transfer the word address of the operand from AO into the XA register and sets XAFUL to indicate the presence of a valid address in XA. If a memory request is required to fill the X buffer (LDXBA), the controller also enables the octet address in AO to the XBA register, clears XFUL to indicate that the X buffer does not contain the desired octet, and issues XBREQ to CMR to indicate the need for a memory request. The controller then examines the X buffer flags to determine if the buffer contains valid data for the instruction to be executed. If the XUP bit is still set, the buffer requires a Z-to-X update and the present data is not correct. The controller then waits for XUP to clear. When XUP has cleared, the controller determines if an octet has returned from memory (SCTX) into the X buffer. If SCTX is true, the controller sets XFUL to indicate the presence of data in the X buffer. Also, if new data did not arrive, but a Z-to-X update was performed (ZTXU), the controller also sets XFUL. If YFRST is not set, indicating that the X buffer will be the next source of operands for the MBU, the controller also sets DPMBI to indicate that the input stage is prepared to transfer data to level 6.

4.24 LEVEL 6 CONTROLLER - SCALAR MODE

The level 6 controller (figure 4-42) operating in the scalar mode monitors signals from the level 5 and Select Next controllers to gate data from level 5 into level 6 for output to the AU. For a transfer to occur, the following signals must be present:

- DPMBI - Data Present in Memory Buffer Interface
- SLNXT - Select Next; indicates that the next instruction can be input to the AU
- DPMBO - Data Present in Memory Buffer Output stage

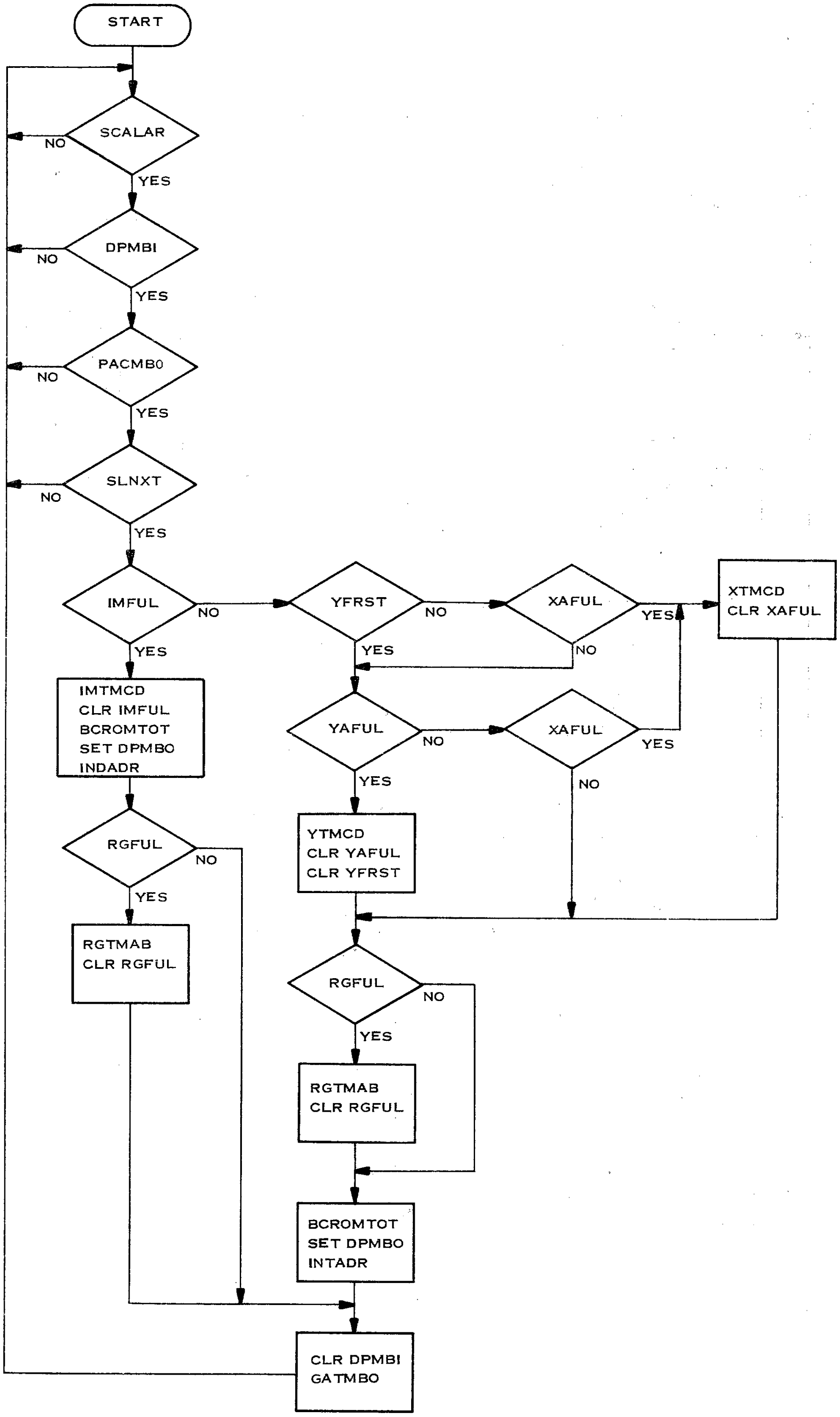
  or

- PACAUR - Path Ahead Clear in the AU Receiver section.

If these signals are present, the controller checks the level 5 register status flags. If an immediate operand is in level 5, the controller enables the contents of IMM into the MCD register, clears IMFUL and sets DPMBO. If a register operand is in level 5, the controller enables the contents of the REG register into the MAB register. For X or Y operands, the controller checks the YFRST flag. If it is set, the controller examines the Y buffer flags first. In either case, if the first inspection does not result in a transfer into MCD, the controller will check the flags associated with the other buffer and transfer data from that buffer into MCD if valid data is present. When all required transfers are complete during a control cycle, the controller issues PACMBO to the level 5 controller.

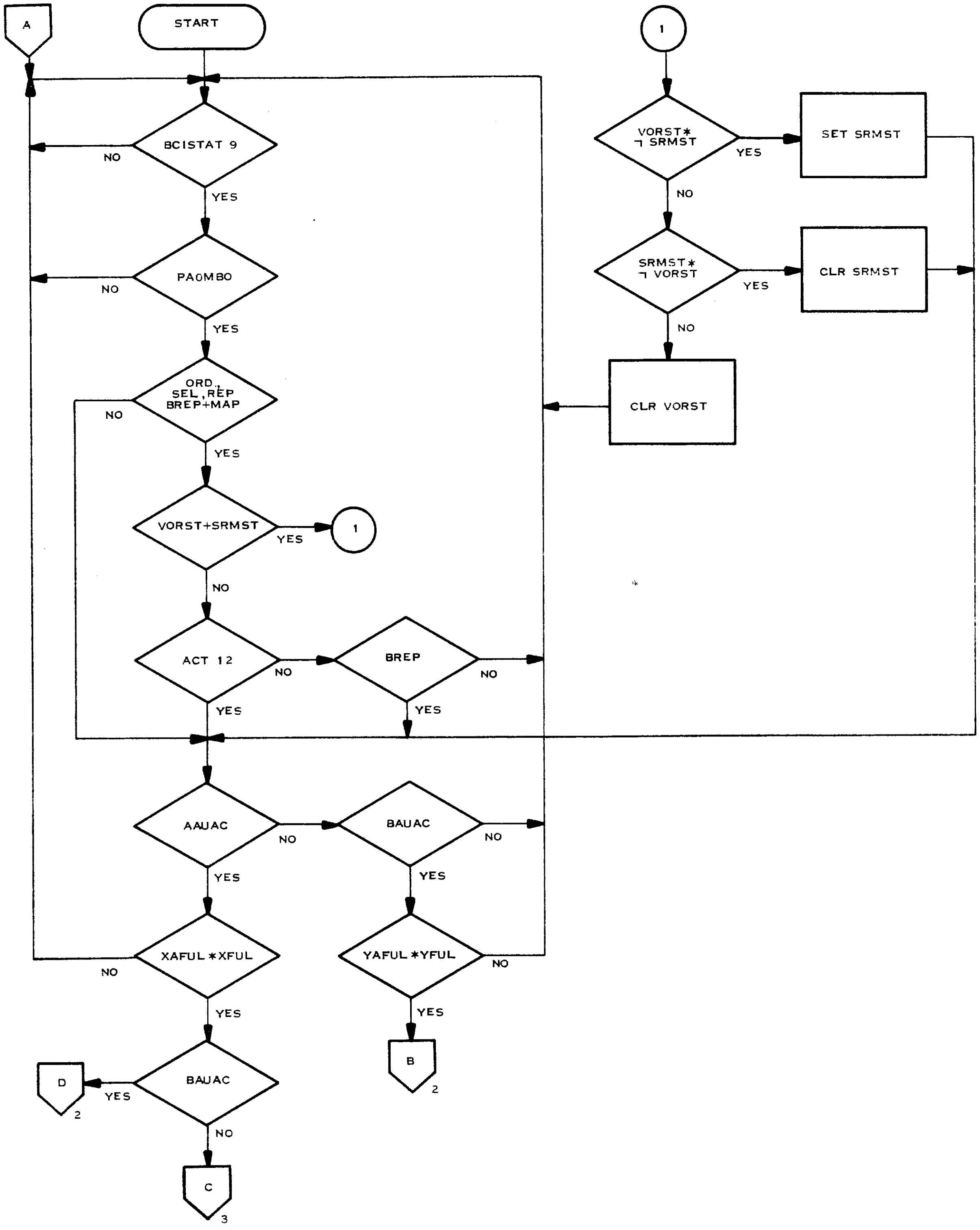
4.25 LEVEL 6 CONTROLLER - VECTOR MODE

Level 6 vector mode control is illustrated in figure 4-43. Description of this circuit will be supplied at a later date. Level 6 ROM address selection for both scalar and vector modes is illustrated in figure 4-44.



(B)132253

Figure 4-42. Level 6 MBU Output Control Flowchart - Scalar



(B)132354 (1/5)

Figure 4-43. Level 6 MBU Output Control Flowchart - Vector (Sheet 1 of 5)

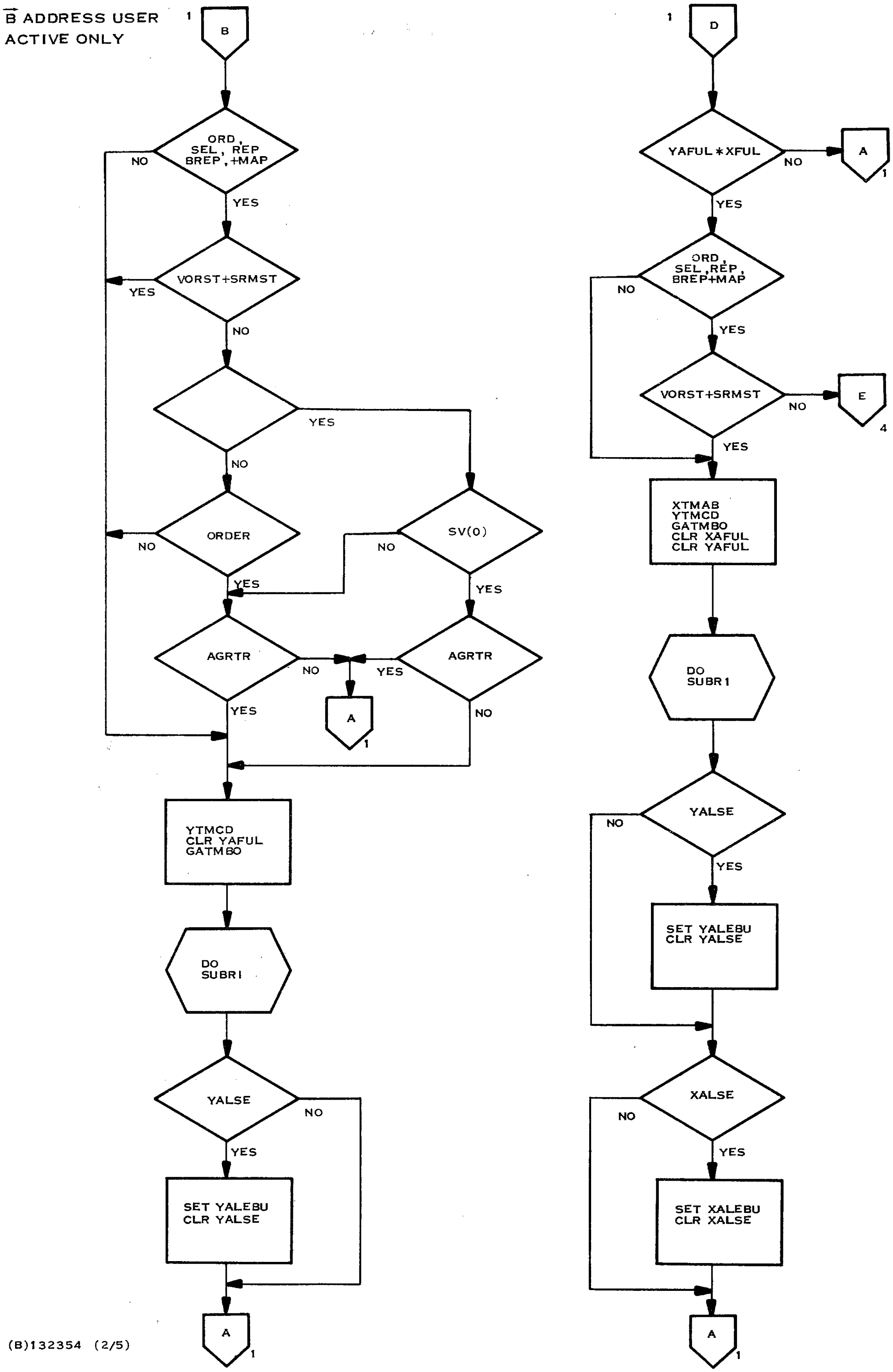


Figure 4-43. Level 6 MBU Output Control Flowchart - Vector (Sheet 2 of 5)

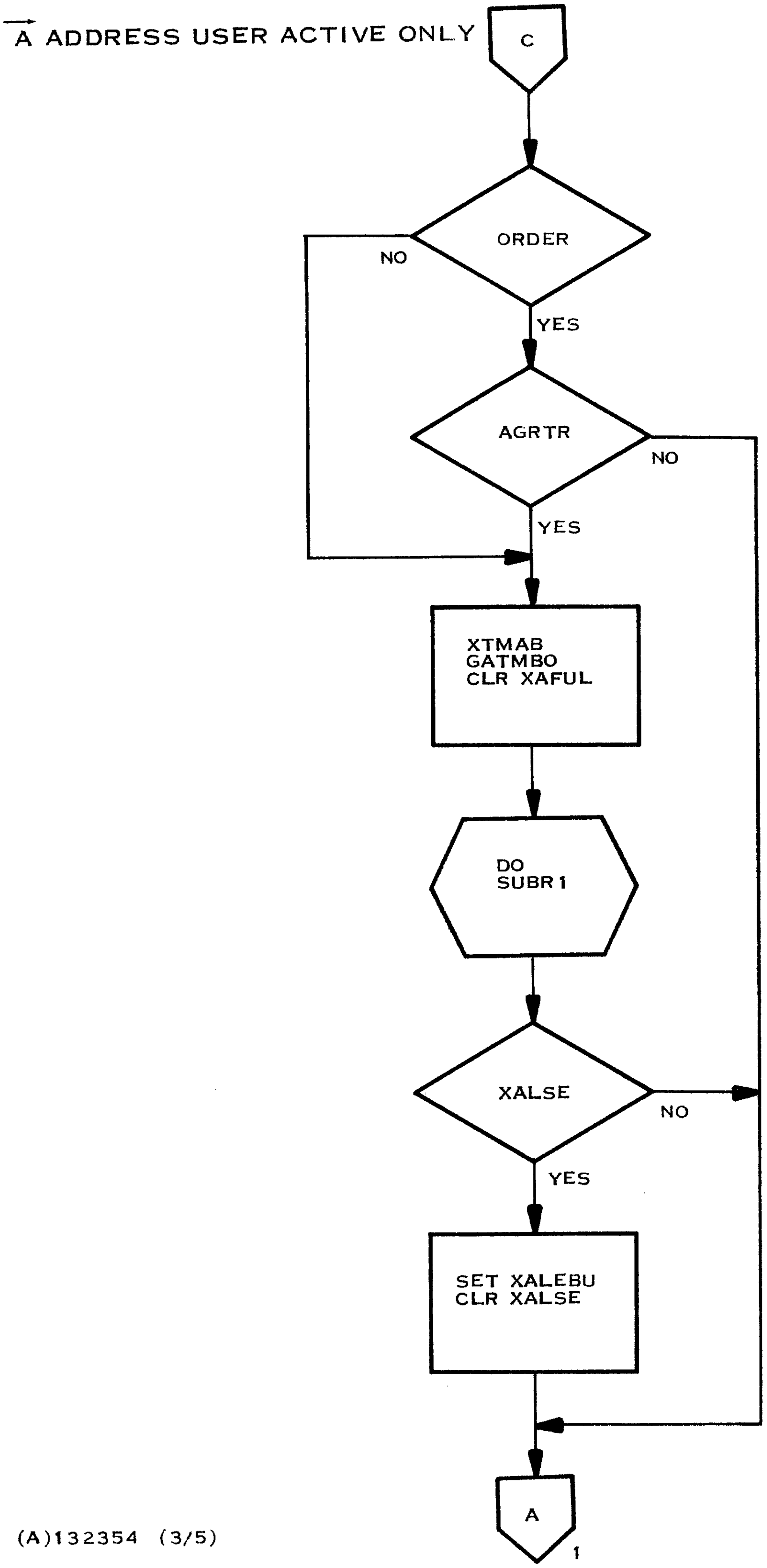
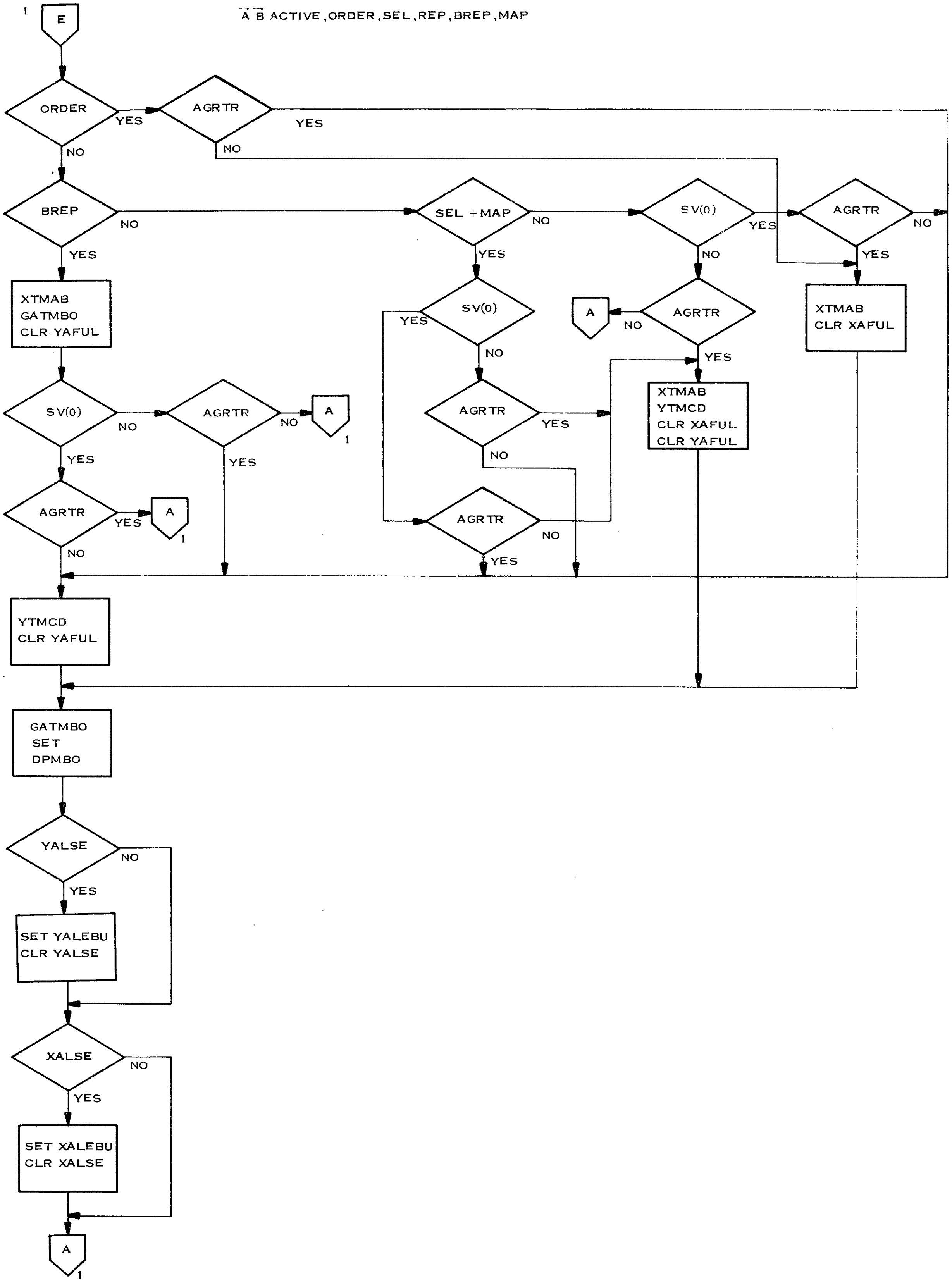
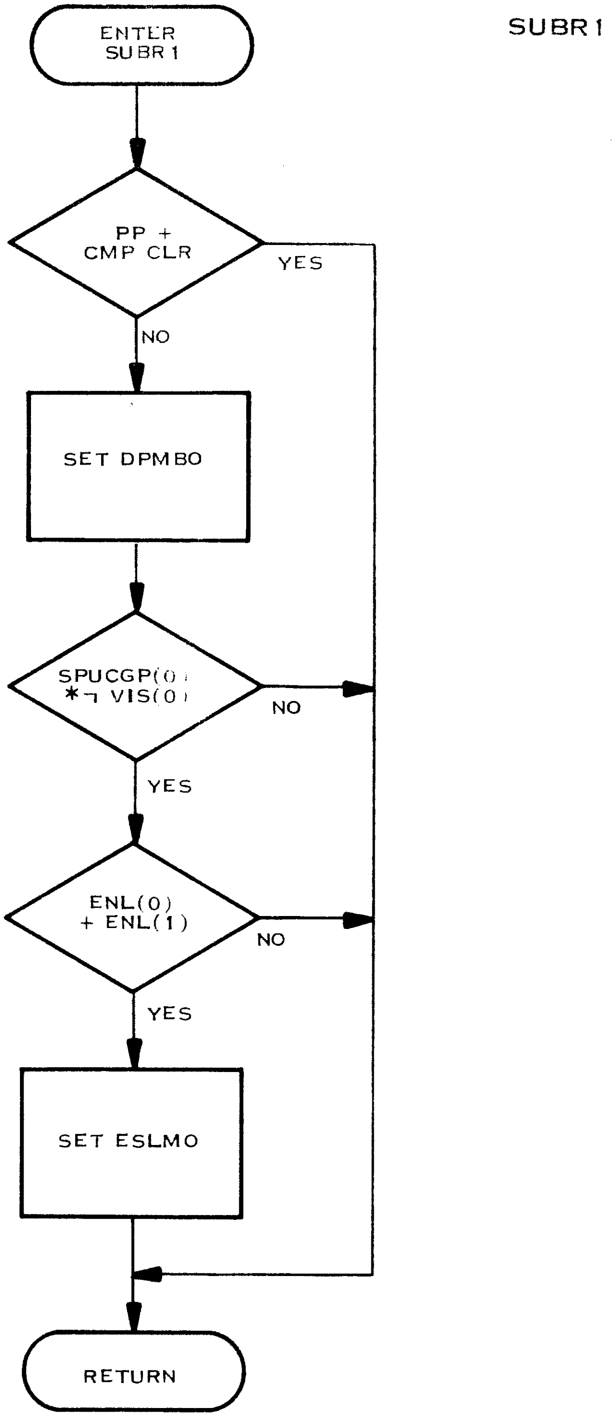


Figure 4-43. Level 6 MBU Output Control Flowchart - Vector (Sheet 3 of 5)



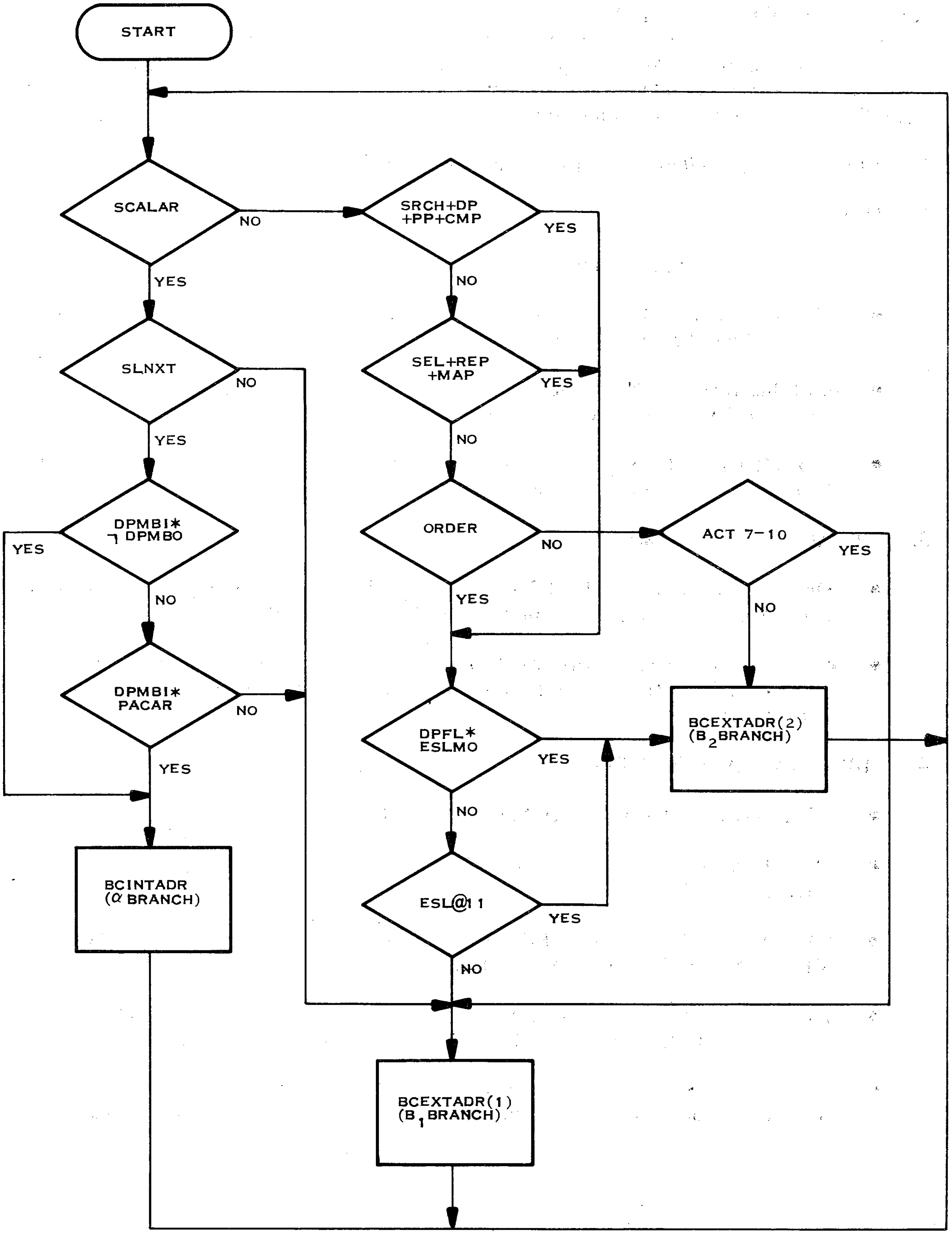
(C)132354 (4/5)

Figure 4-43. Level 6 MBU Output Control Flowchart - Vector (Sheet 4 of 5)



(A)132354 (5/5)

Figure 4-43. Level 6 MBU Output Control Flowchart - Vector (Sheet 5 of 5)



(A) 132355

Figure 4-44. Level 6 ROM Address Selection Flowchart (Scalar and Vector)

4.26 SELECT NEXT CONTROLLER

The Select Next controller (figure 4-45) enables transfer of instructions into the AU, operation codes into the AU control ROM, and operands from level 5 to level 6 by setting the SLNXT control bit. Except for vector initiation, the controller is used for scalar operations only. Select Next (SLNXT) is set under the following conditions:

1. VECTOR INITIATION loads first vector address into the AU control ROM.

2. MBI INACTIVE

   - MBI Inactive and AU empty - enables level 5 to level 6.

   - MBI Inactive and AU active with a one clock operation - enables level 5 to level 6 since AU will be empty on next clock.

   - MBI Inactive, PACMBO and AU active during last clock of operation - enables level 5 to level 6 since AU will be empty on next clock.

   - Next ROM Code NO OP - No Op will have no effect on operation in AU; therefore level 5 to level 6 is enabled.

3. MBI ACTIVE and PACMBI

   - One clock operation at level 5 - enables level 5 to level 6 to prepare to insert one clock operation into pipe at completion of current operation.

   - Same group at level 4, first clock of same group and not short circuit (wait) at level 4 - allows transfer of instruction to prepare for interleave of operation in same group time.

4. MBI ACTIVE AND NOT PACMBI

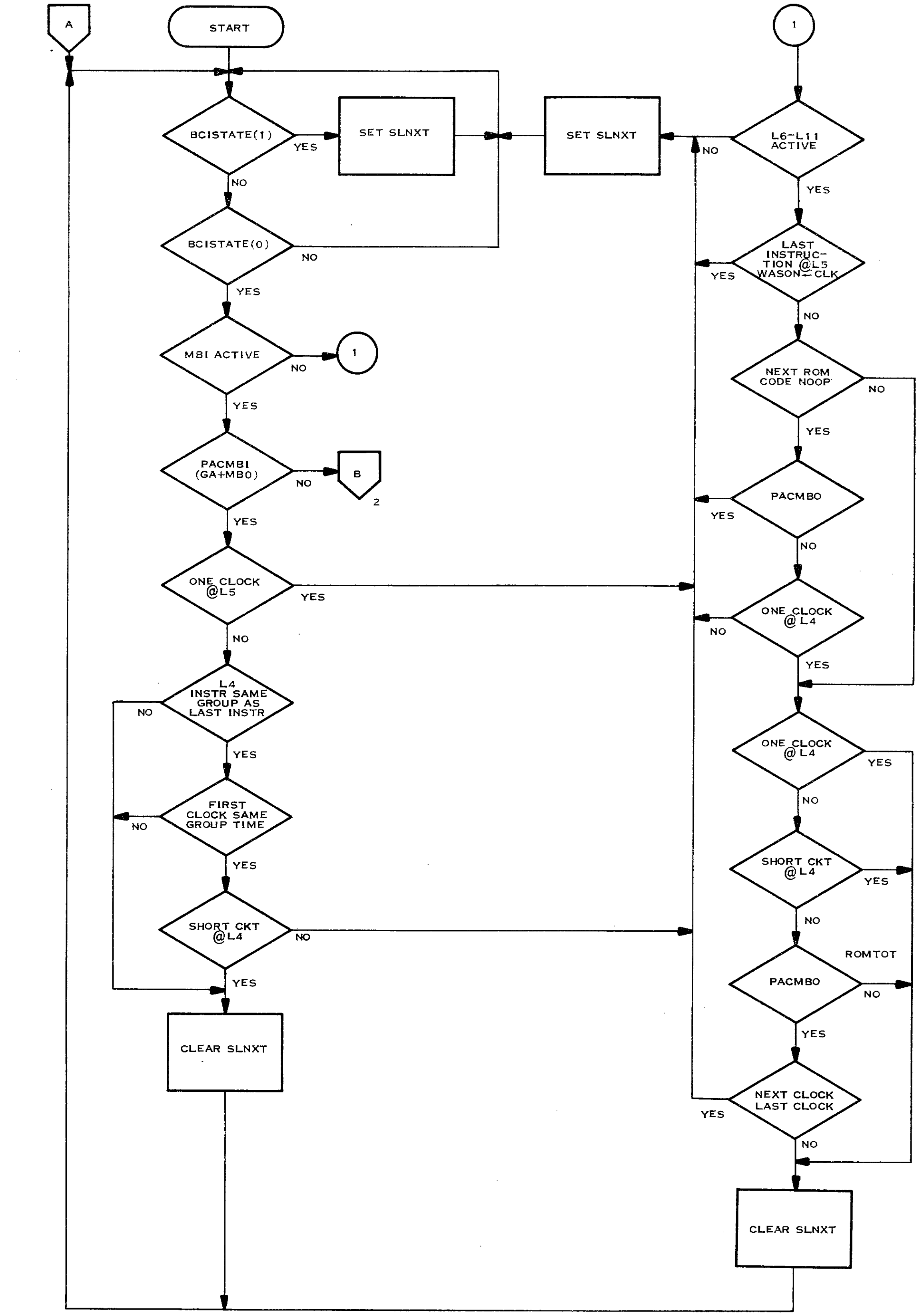
   - AU Inactive - enter next operation into pipe for processing.

   - One clock at level 5 and in AU - Enter next operation since AU will be vacant after next clock.

   - AU active, PACMBO, Next ROM Code No Op - No Op will not affect AU, load MBO in preparation for next operation.

   - AU active, PACMBO, Next clock last clock of current ROM instruction - AU operation is completing, load MBO in preparation for next operation.

   - Same group, Same group time, and if single-length multiply, the result in the AU will be the same as the result of the new instruction at level 5 (see explanation below).



(B)132356 (1/2)

Figure 4-45. Select Next Controller Flowchart (Sheet 1 of 2)

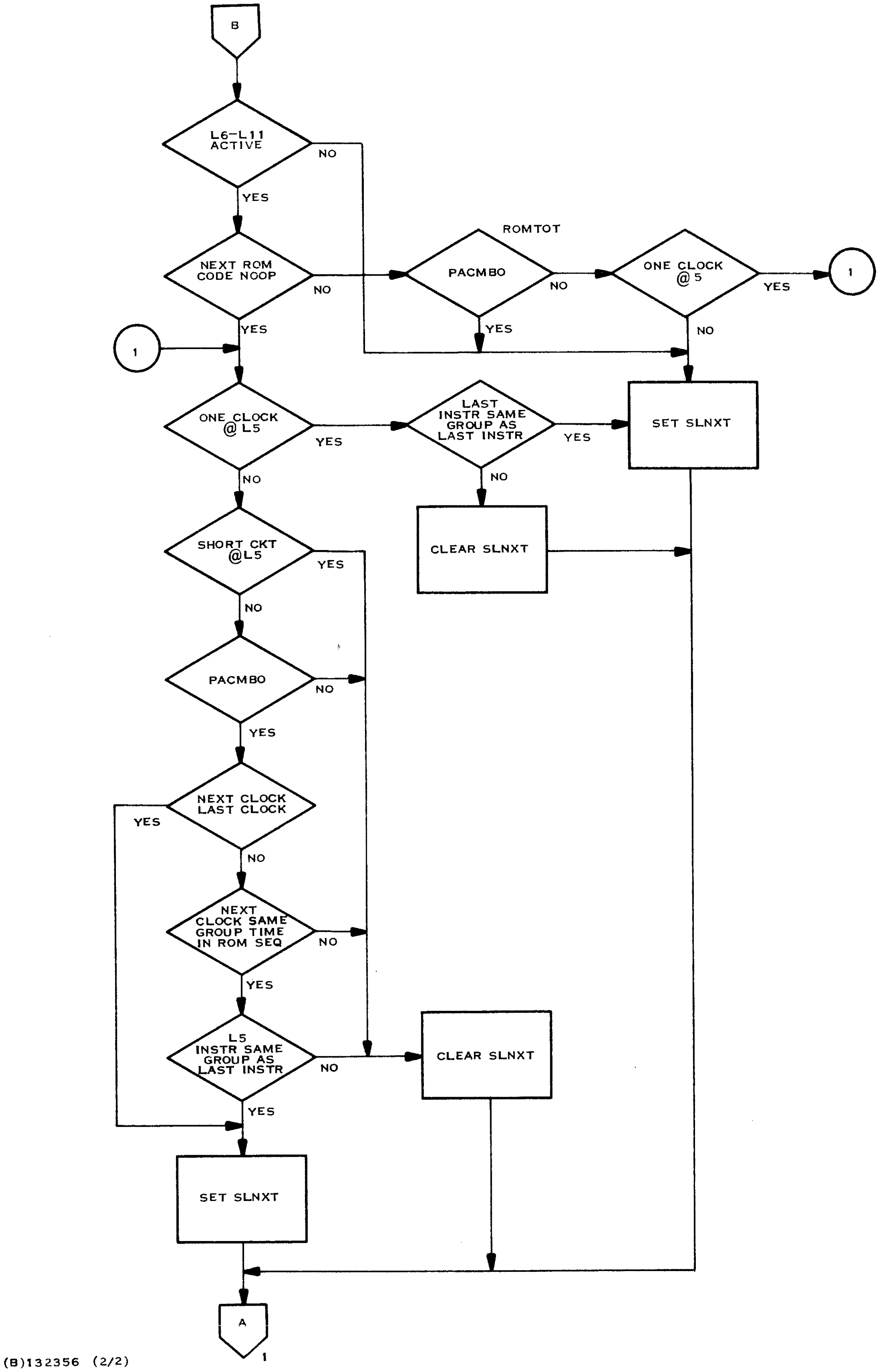


Figure 4-45. Select Next Controller Flowchart (Sheet 2 of 2)

RODD indicates that the result of a single-length multiply at level 5 will be stored into an odd memory location, and therefore cannot be a doubleword result. R OPTION FALSE and R OPTION TRUE are AU ROM signals that indicate the word length of the result of a single-length multiply currently in the pipe. R OPTION FALSE indicates that the result will be single-length; R OPTION TRUE indicates that the result will be double-length. The Select Next controller; however, examines the complement of these two ROM signals so that non-multiply instructions need not be coded with ones. Therefore, if RODD is false (indicating an even storage address) and R OPTION FALSE is also false (indicating a double-length result), both operations specify the same length result. Similarly, if RODD is true and R OPTION TRUE is false (indicating a single-length result), both operations specify the same length result. The controller can then enable SLNXT so that the operands in level 5 can be inserted into the AU at the same group time to be overlapped with the current instruction in the AU. One of these two paths will always be true for same group operations other than single-length multiply.

4.27 CENTRAL MEMORY REQUESTER (CMR)

The Central Memory Requester (CMR) in the MBU (figure 4-46) receives requests for octets from either the level 5 controller or from the Z storage control circuit. CMR then decides priority for the requests, issues the requests to memory, and routes the returning octets from memory, through the SC Memory Interface File, into the buffer file (X, Y or Z) that requested the data. Data requested for the Z buffer is for a Z Fill-in (ZFILN) operation. That is, the data in the Z buffer contains some incomplete halfwords and cannot be stored into memory without filling the empty halfword positions with valid data from memory. The halfword modified flags in the ZBM register designate which halfwords in the Z buffer contain valid data, and prevent the incoming memory words from changing those halfword positions in ZB. CMR also issues storage requests to memory to write the contents of the ZB file into memory after a fill-in operation, or when the next word to be stored into the Z buffer will be in a new octet. The control circuits contain an examination state for each number of allowable outstanding requests to memory. Each cycle performs the same basic functions.

If a hardcore operation must be performed, the controller checks to see if an outstanding request has returned from memory. If SCFUL is set (new octet in SC) and the queue indicate that the octet is for a Z buffer fill-in operation (CUE = 10), the controller allows the next clock to transfer the contents of SC into the ZB file, clears the ZFILN flag, and enables the halfword modified flags to transfer the changed halfwords from the Z buffer into ZB to update the octet for storage into memory. The controller relinquishes control to MBU Unit Hard Core for the next clock cycle.

If the OA register is available to transfer another address to central memory, the controller also checks for a Z Fill-in operation and performs the transfers described above. The controller then checks for a request that originates in the Z buffer stream since Z requests must be made immediately due to lack of a buffer stage for the Z data. If there is no ZFILN operation, the controller transfers the address in ZBA to OA to designate the storage area in memory to be used, clears ZWAIT and ZBREQ, and sets the protect mode bits to a "01", indicating that the request is to be governed by the write protection parameters in the MCU. The controller then sets the zone control bits in ZCB that correspond to any of the halfword-modified flags in ZBM, and sets OAFUL to indicate that a request is being made to memory.

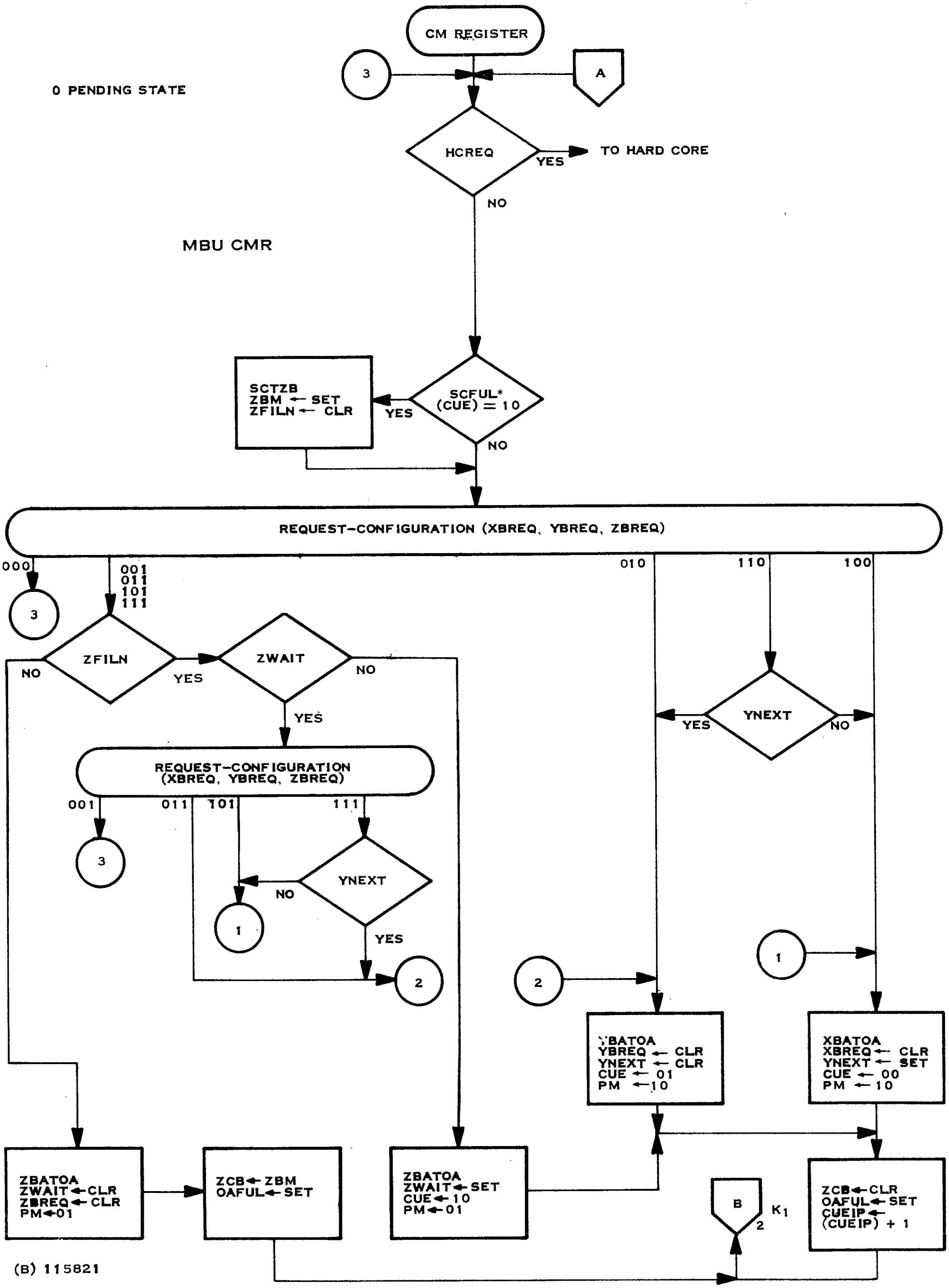
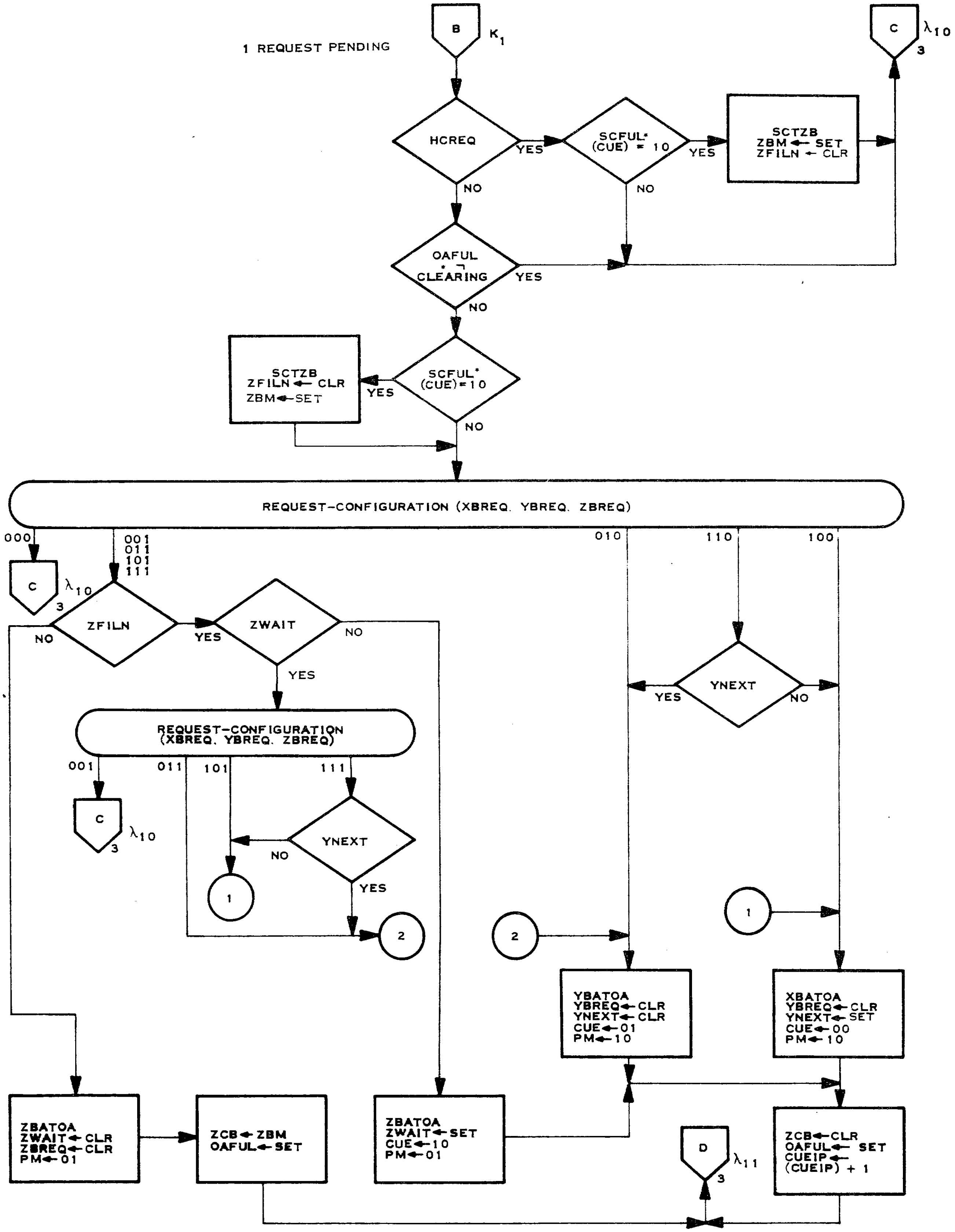
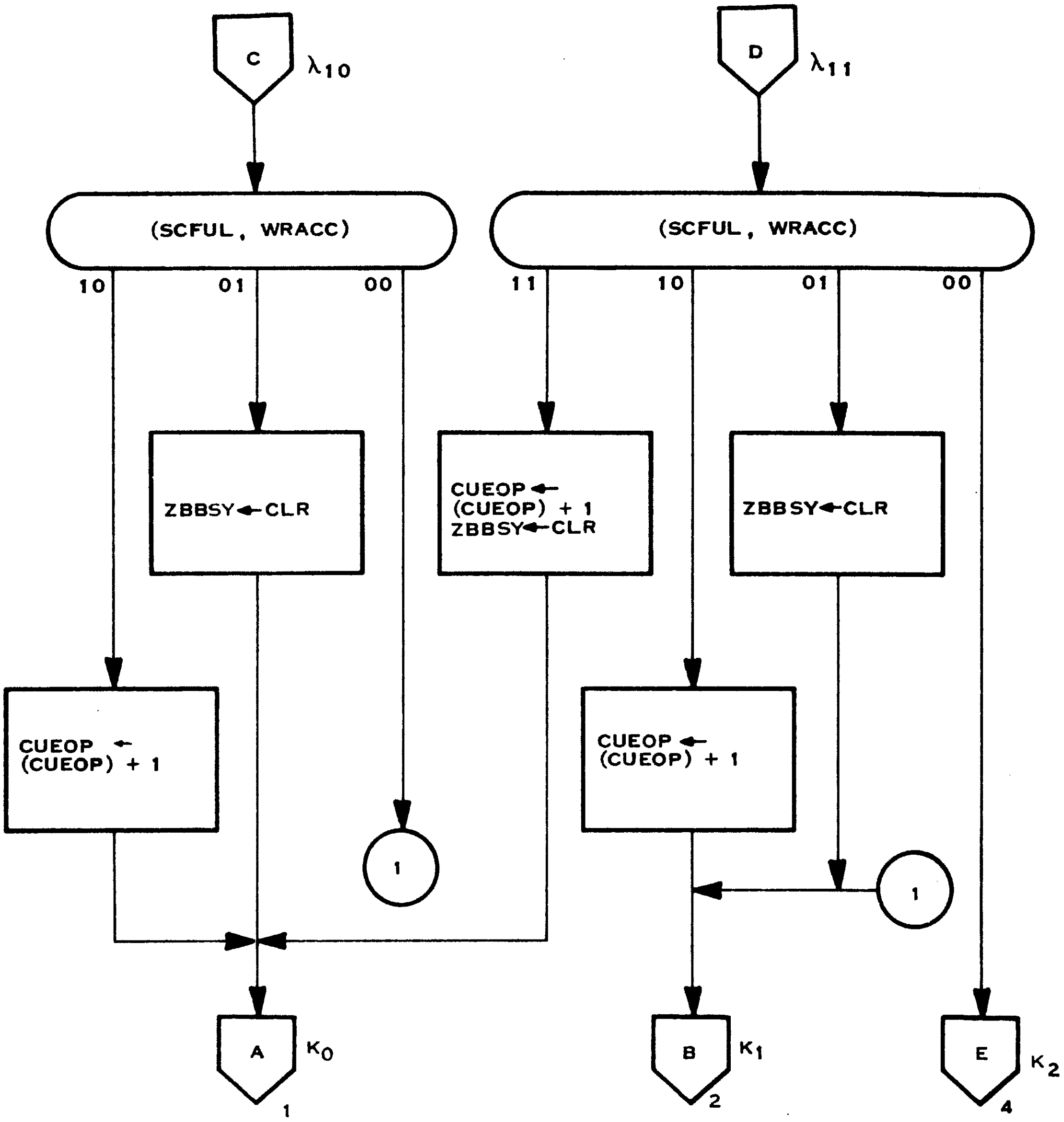


Figure 4-46. CM Requester Flowchart (Sheet 1 of 15)



(B) 115822

Figure 4-46. CM Requester Flowchart (Sheet 2 of 15)



(A) 115823

Figure 4-46. CM Requester Flowchart (Sheet 3 of 15)

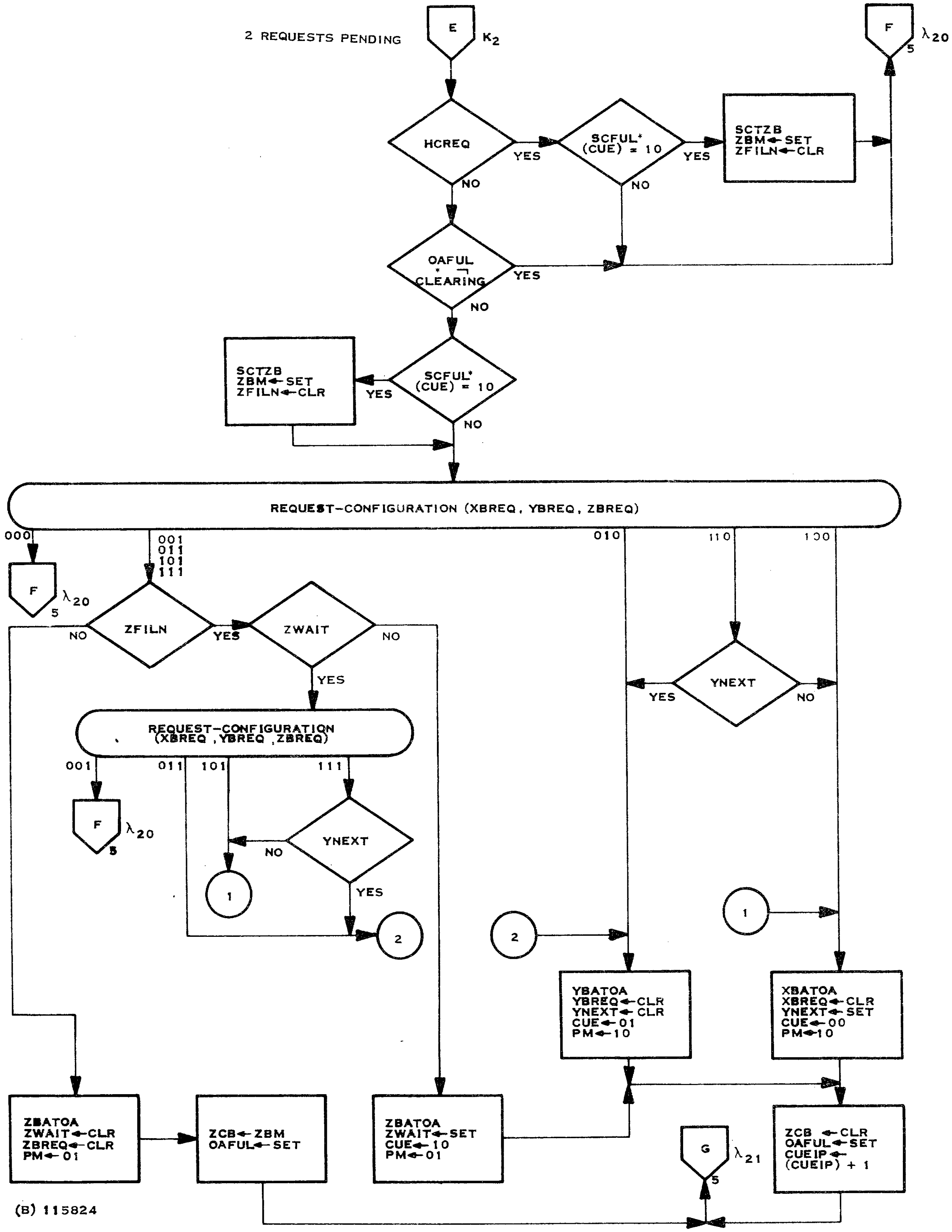
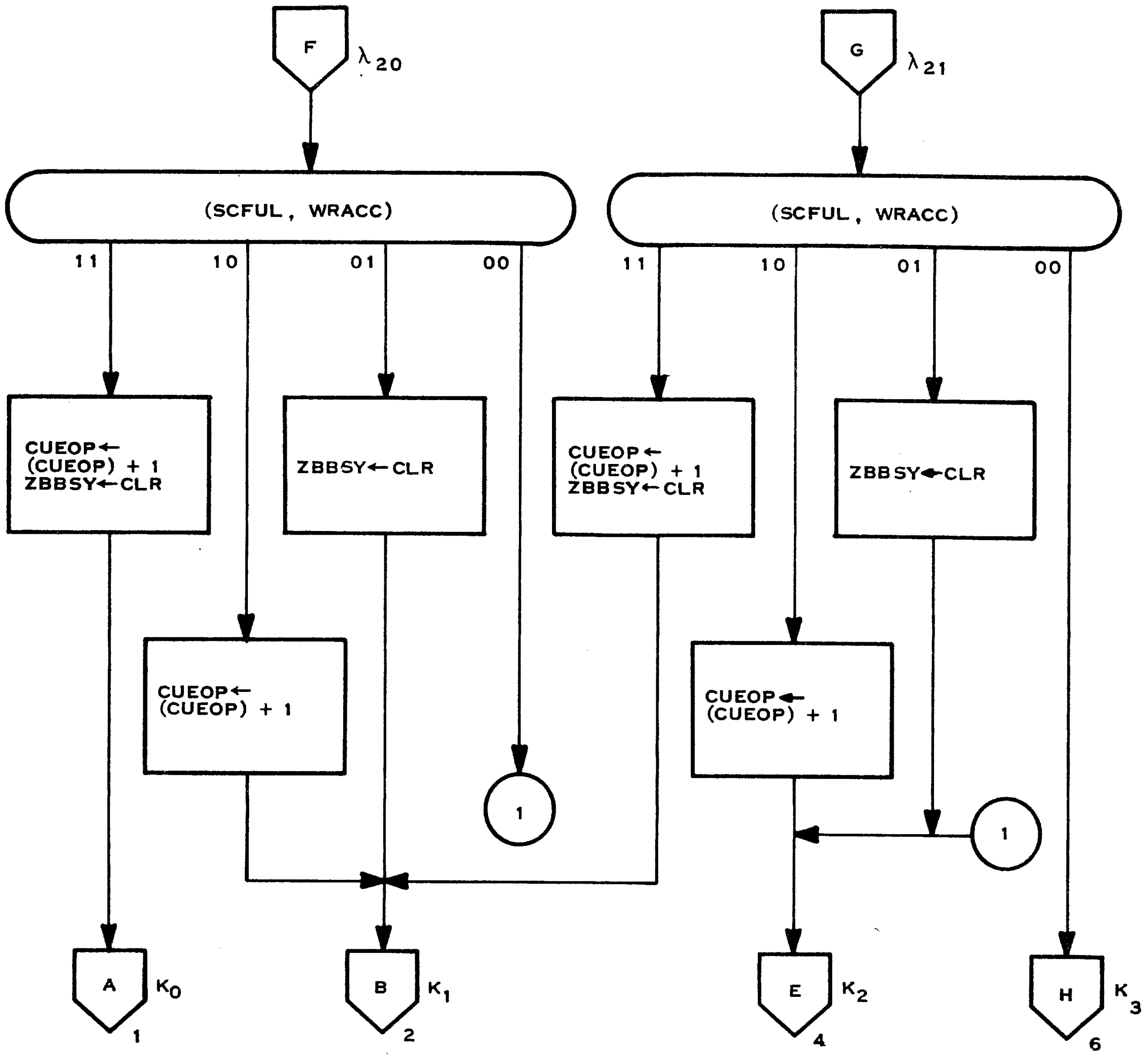
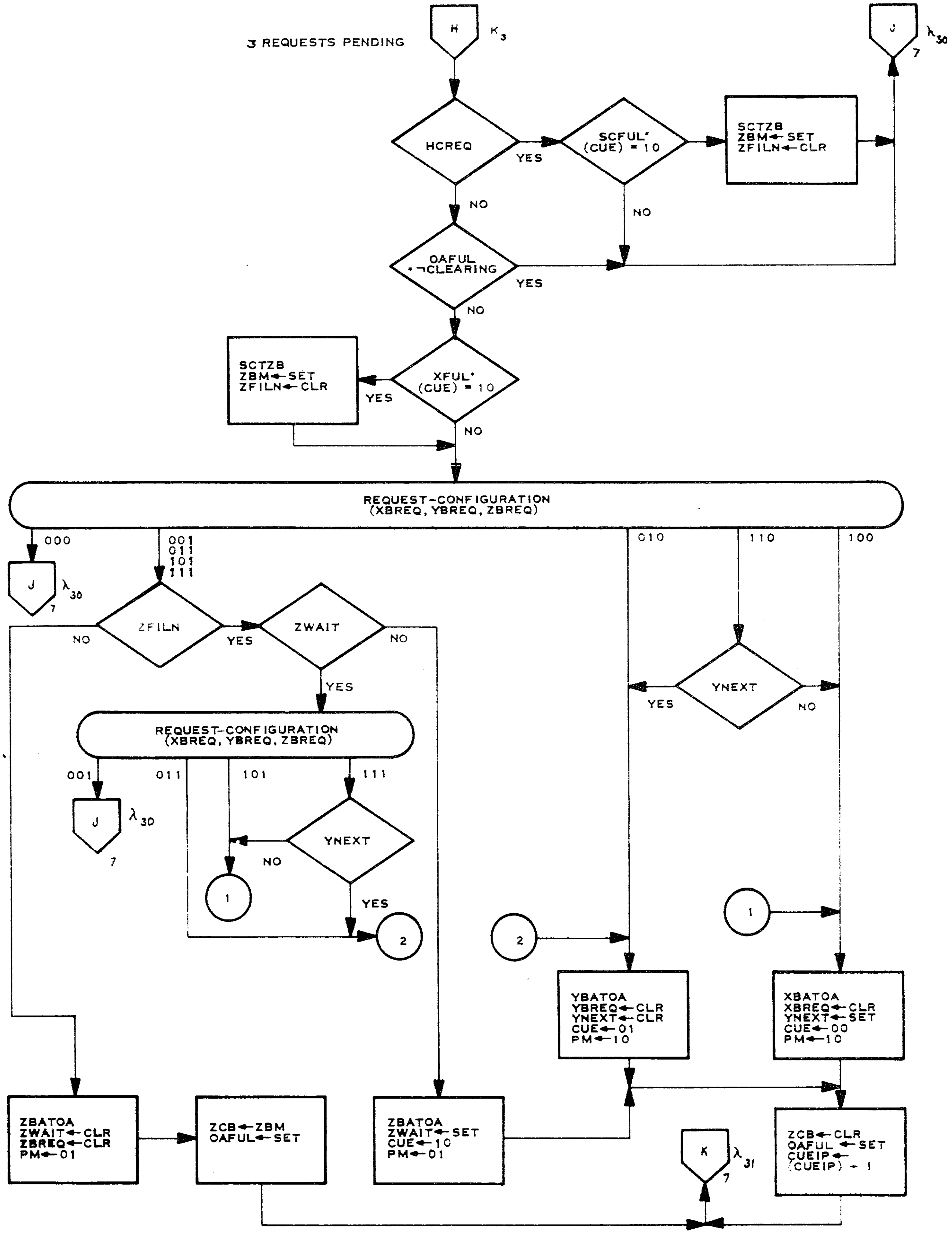


Figure 4-46. CM Requester Flowchart (Sheet 4 of 15)



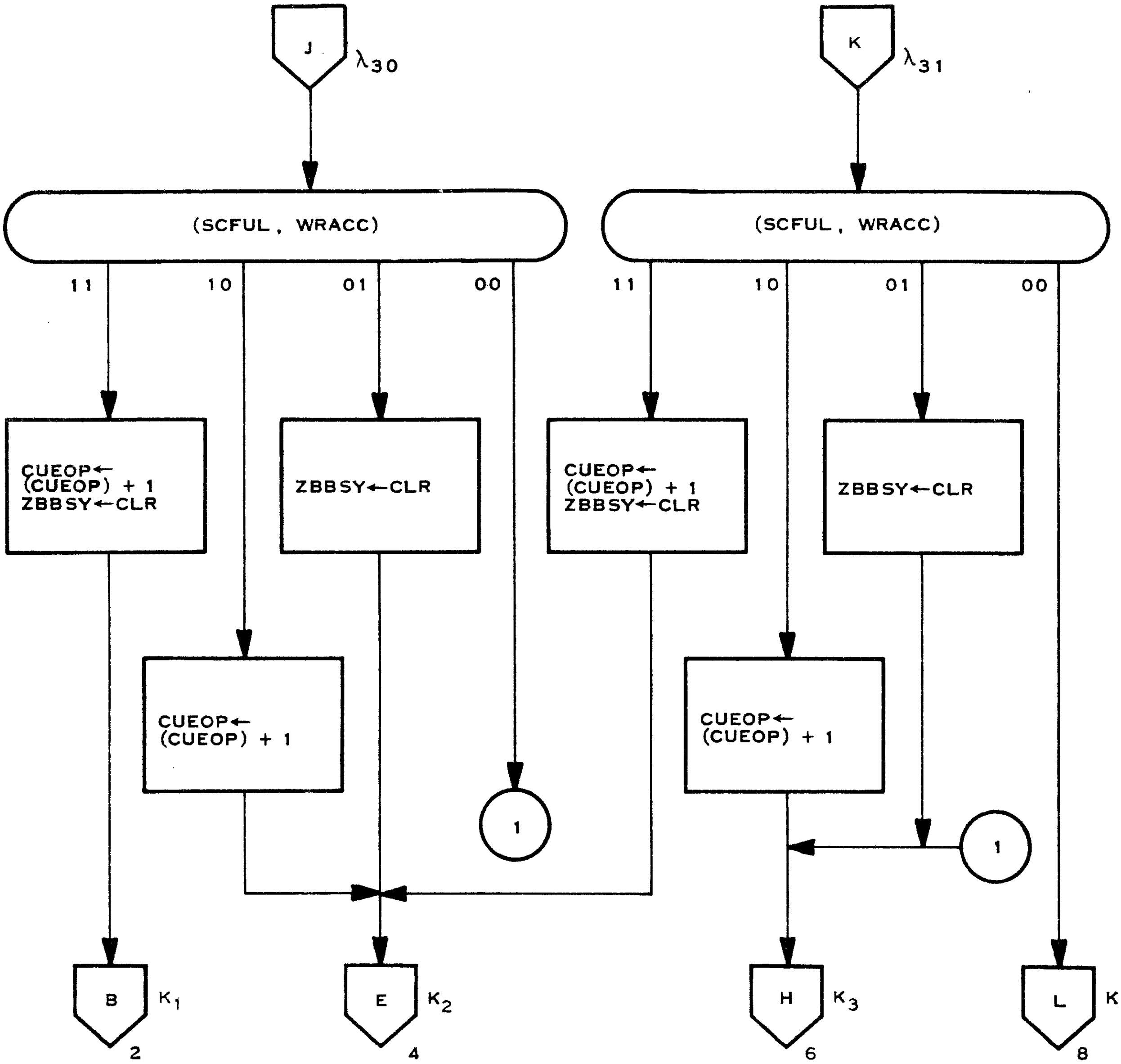
(A) 115825

Figure 4-46. CM Requester Flowchart (Sheet 5 of 15)



(B) 115826

Figure 4-46. CM Requester Flowchart (Sheet 6 of 15)



(A) 115827

Figure 4-46. CM Requester Flowchart (Sheet 7 of 15)

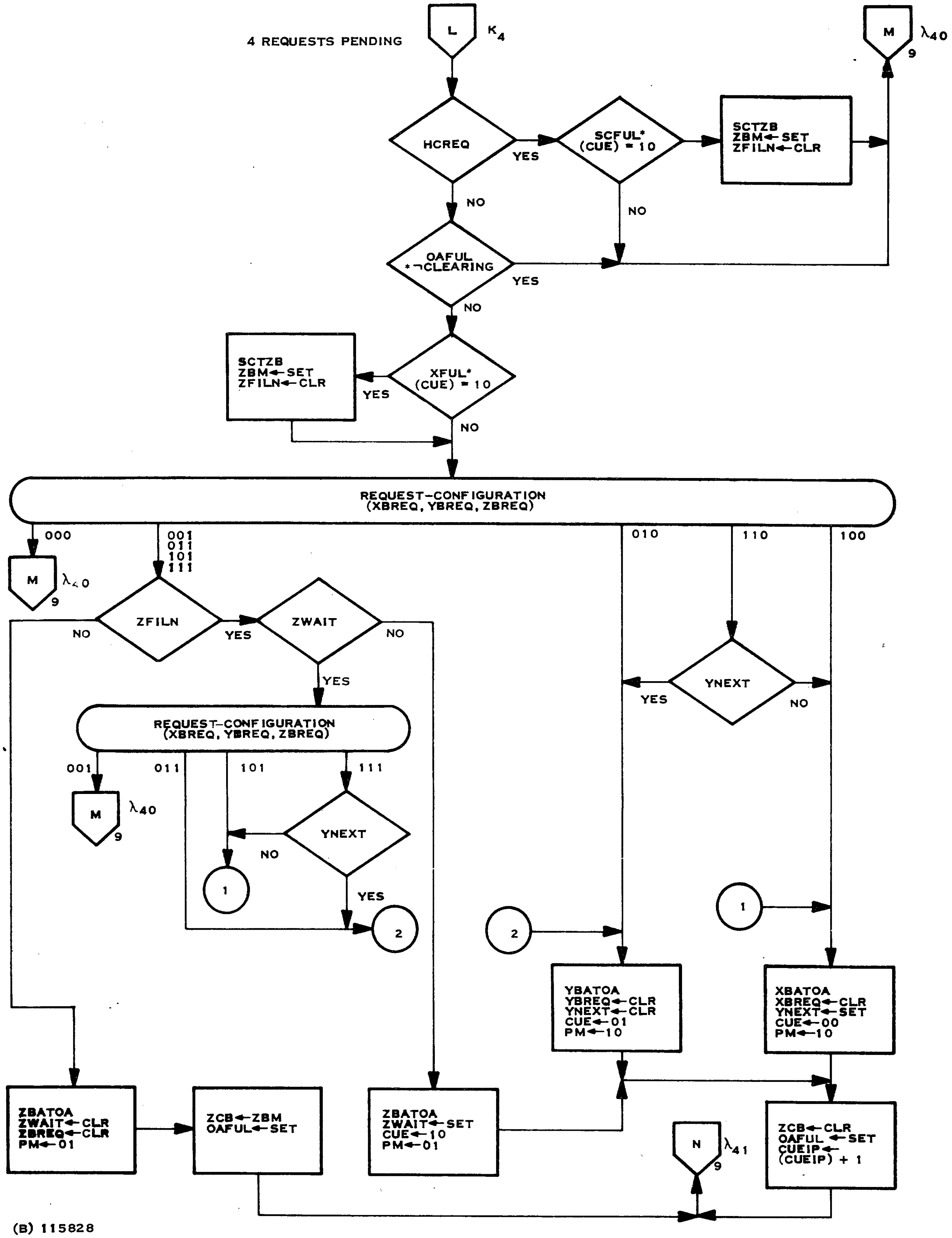
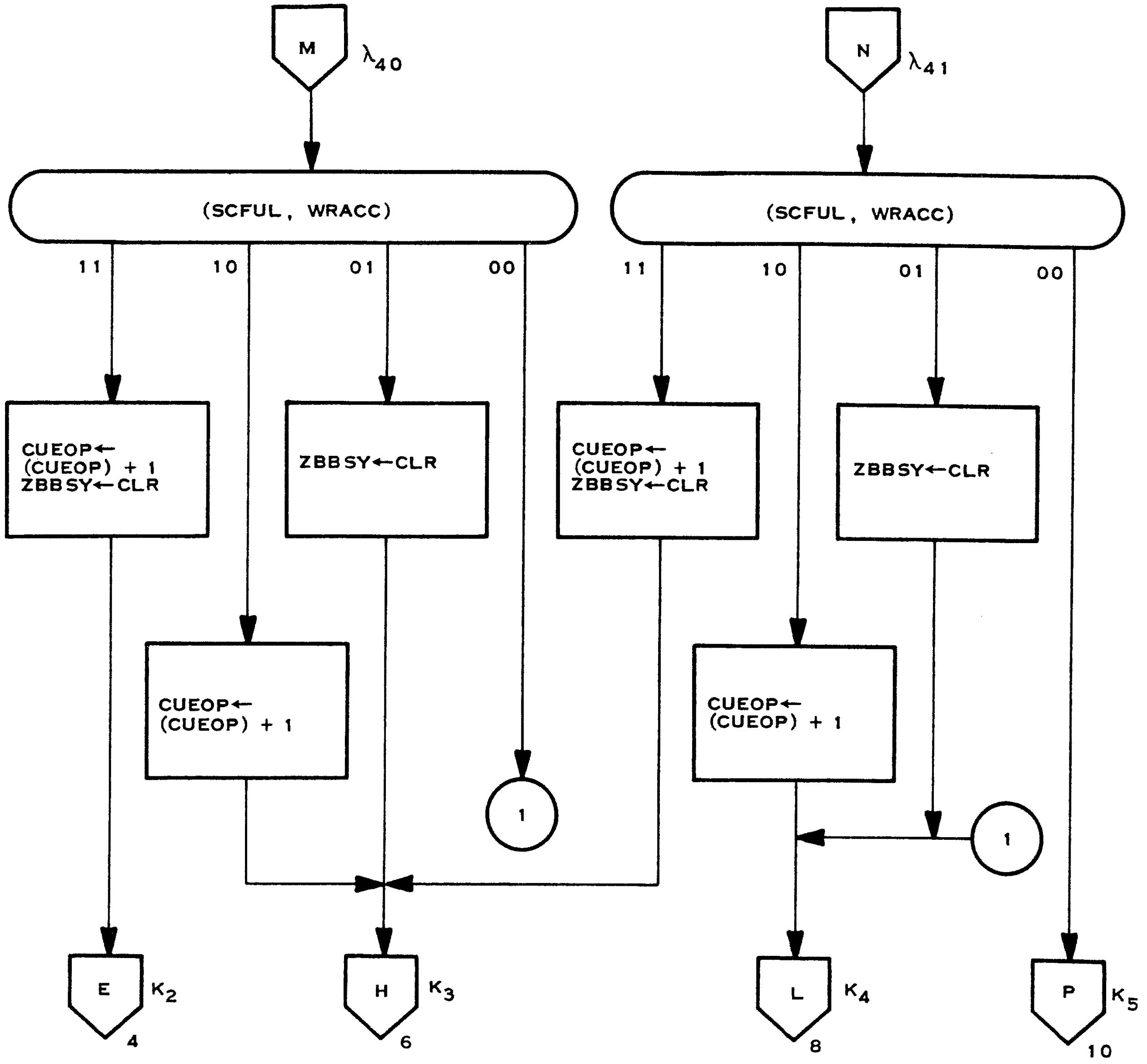
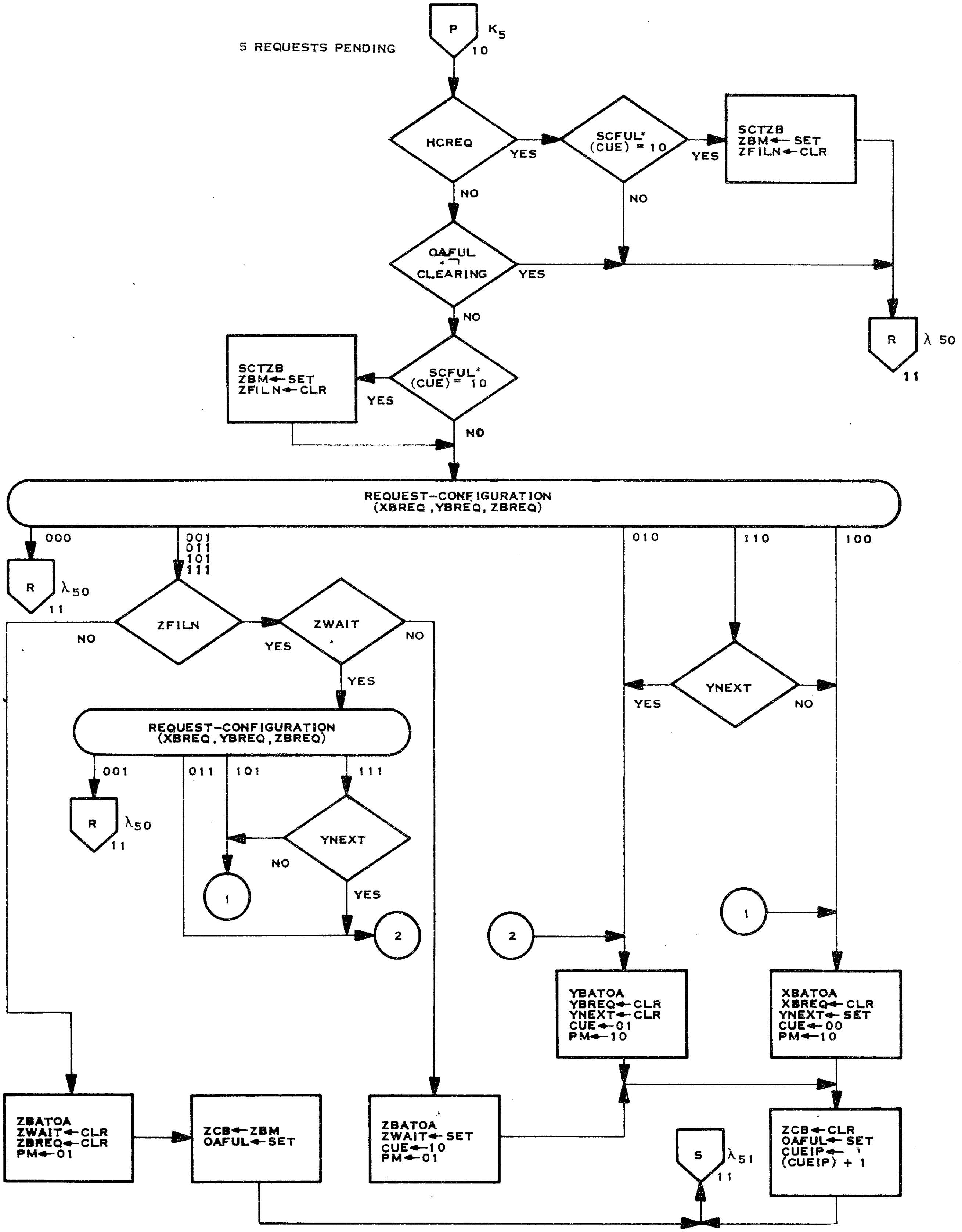


Figure 4-46. CM Requester Flowchart (Sheet 8 of 15)



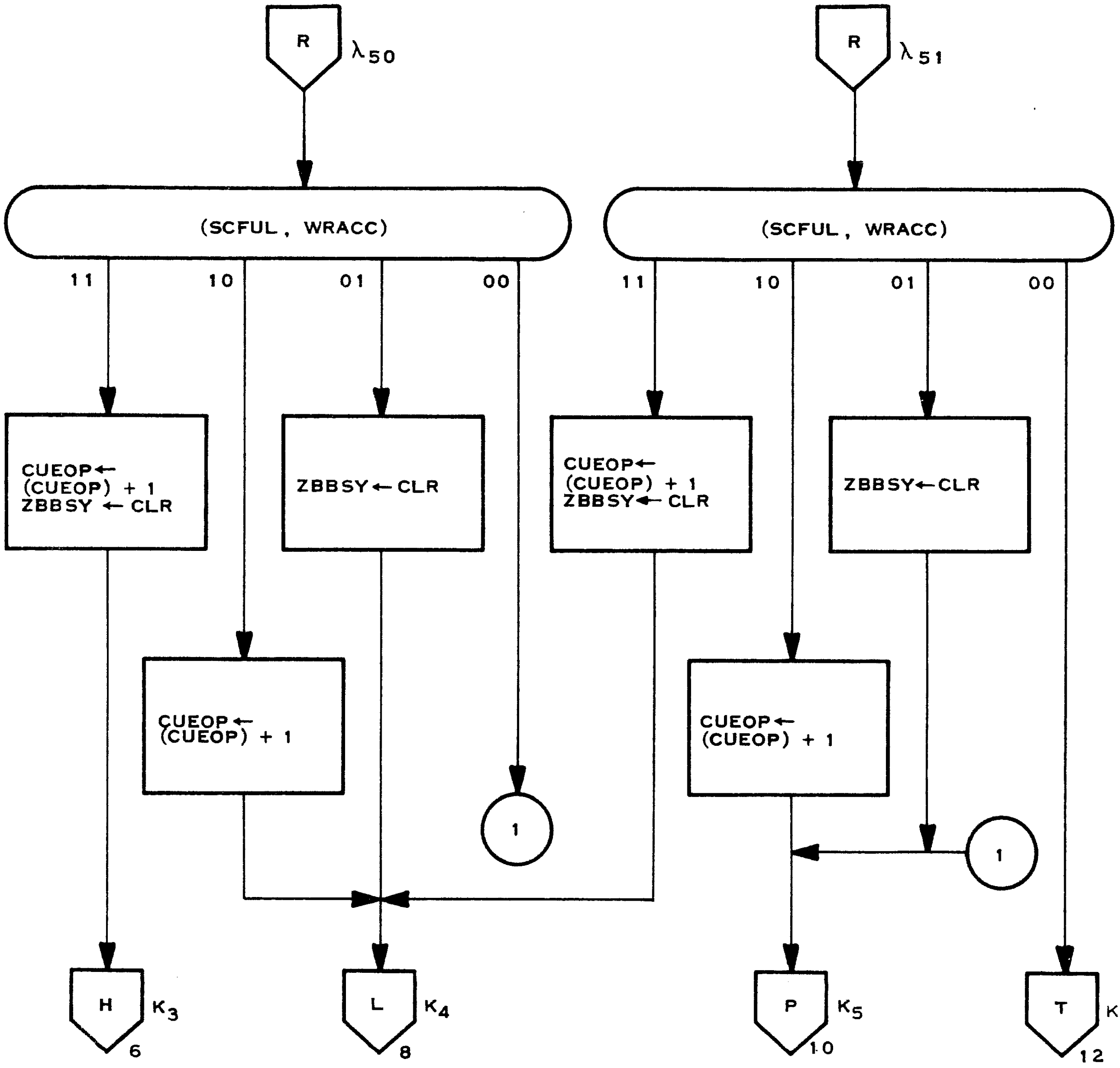
(A) 115829

Figure 4-46. CM Requester Flowchart (Sheet 9 of 15)



(B) 115830

Figure 4-46. CM Requester Flowchart (Sheet 10 of 15)



(A) 115831

Figure 4-46. CM Requester Flowchart (Sheet 11 of 15)

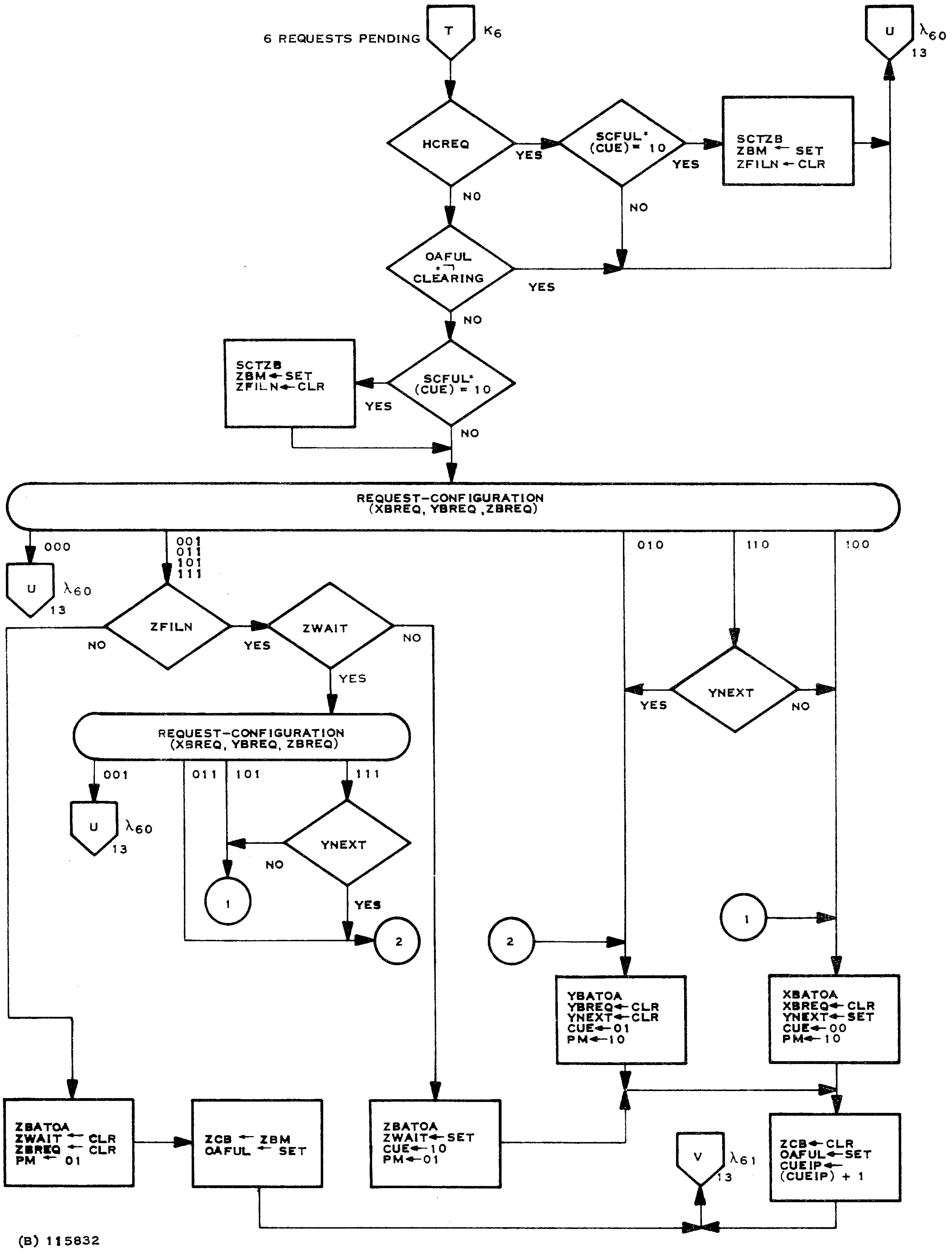
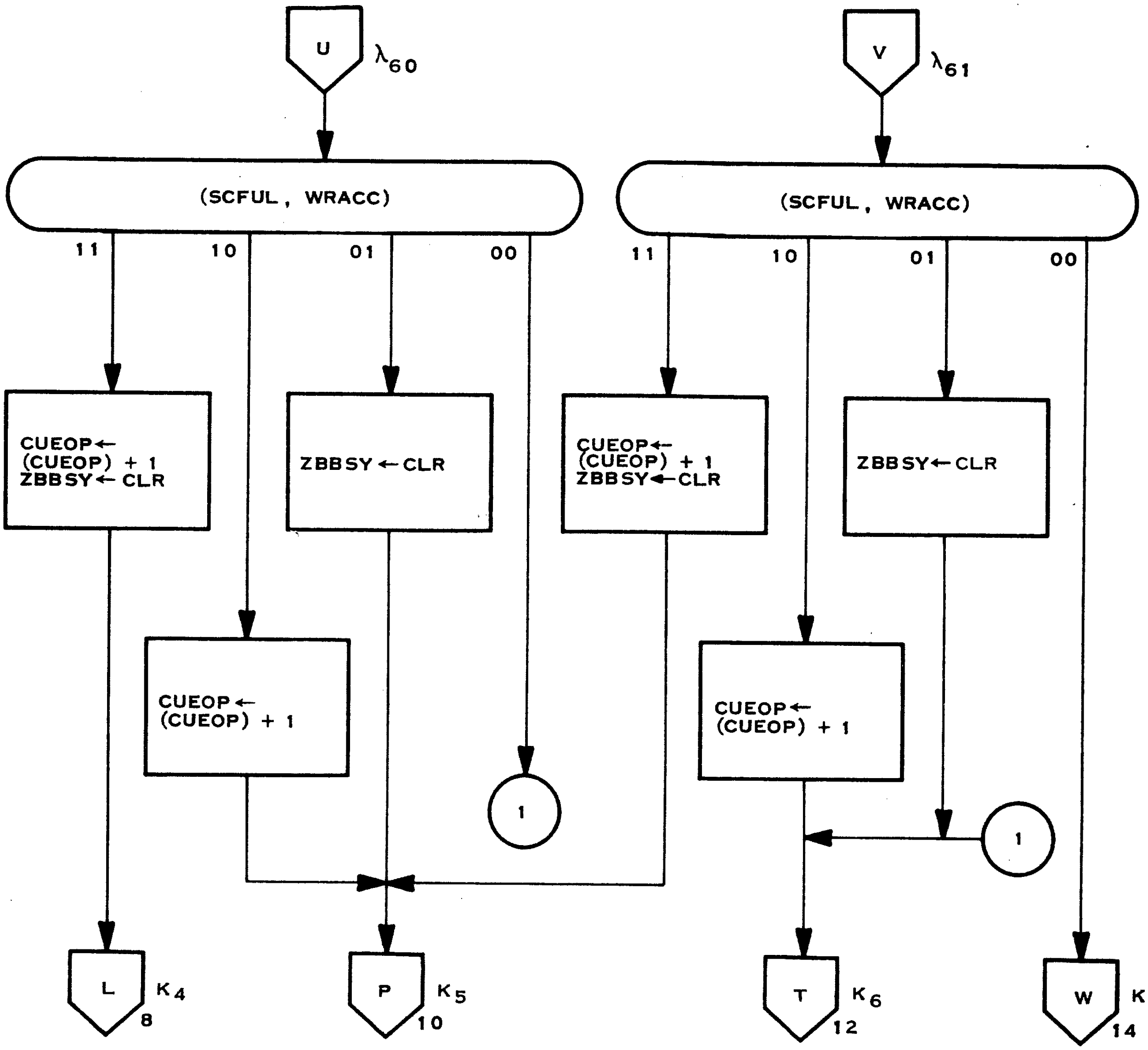
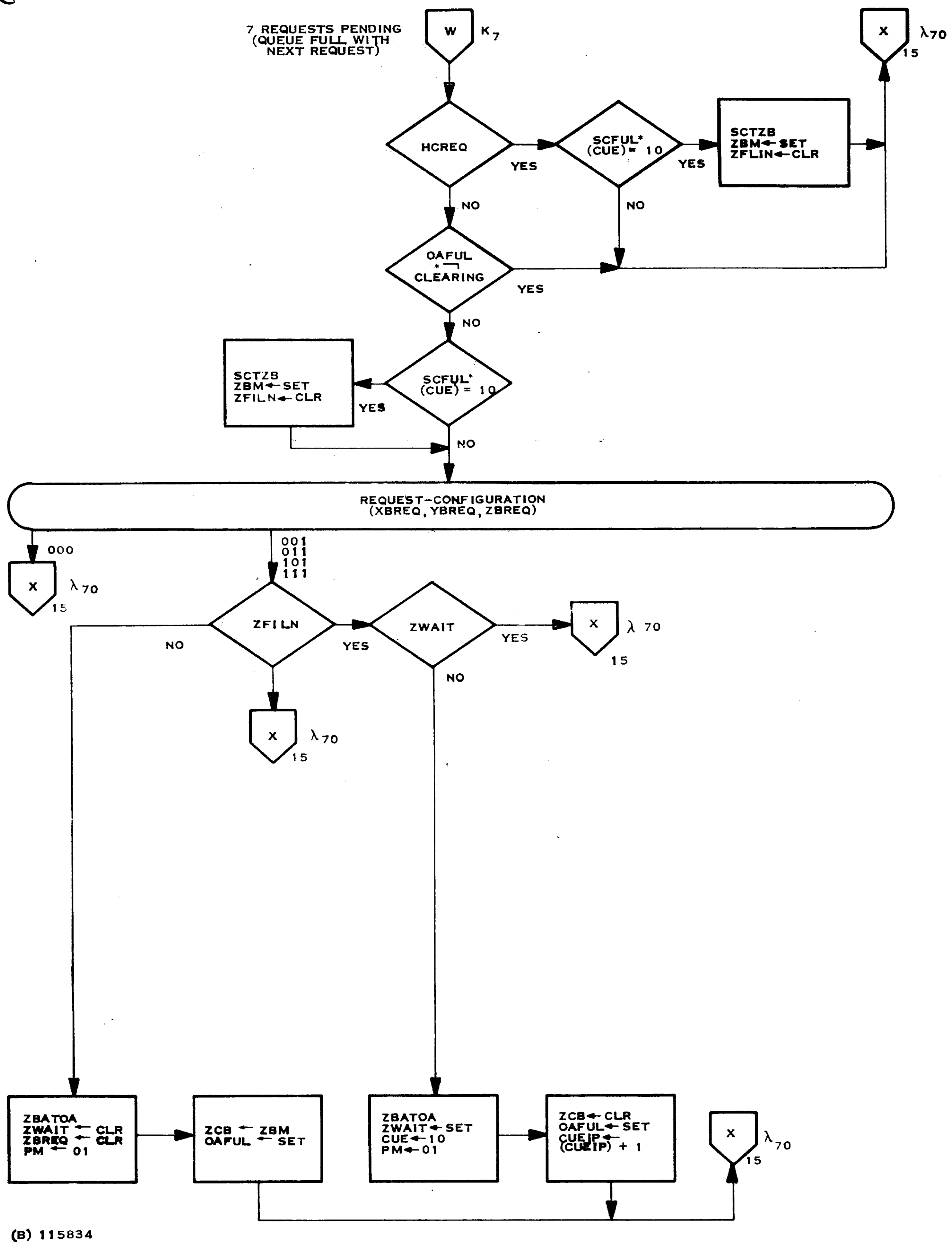


Figure 4-46. CM Requester Flowchart (Sheet 12 of 15)



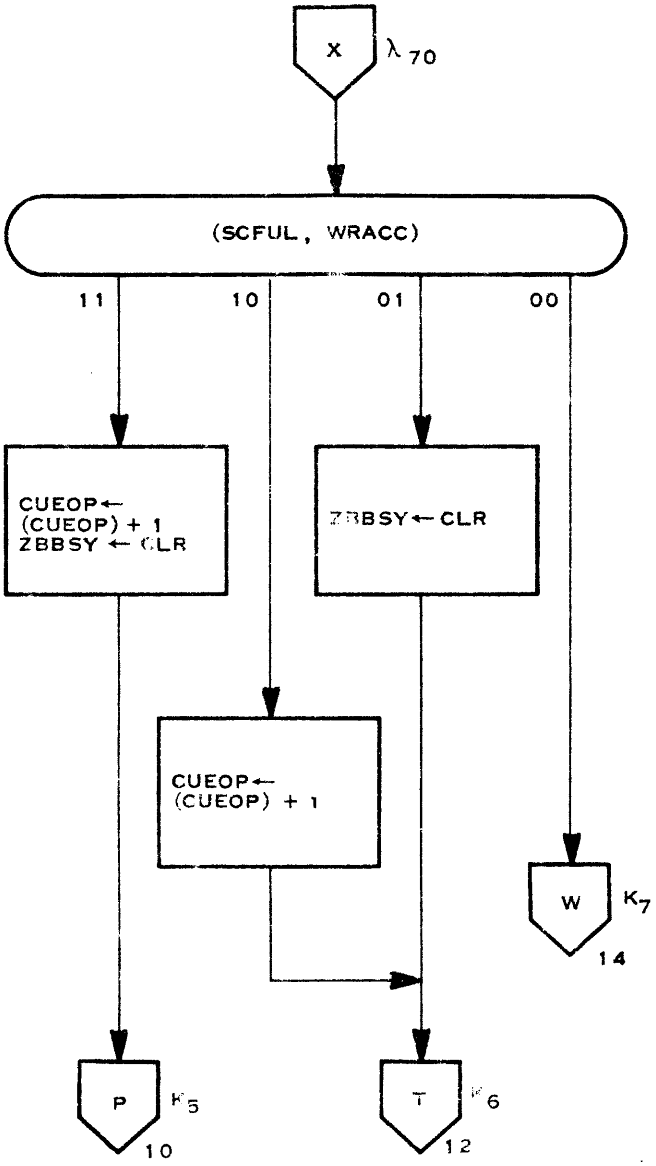
(A) 115833

Figure 4-46. CM Requester Flowchart (Sheet 13 of 15)



(B) 115834

Figure 4-46. CM Requester Flowchart (Sheet 14 of 15)



(A) 115835

Figure 4-46. CM Requester Flowchart (Sheet 15 of 15)

If ZBREQ is set and a ZFILN operation is required, the controller examines ZWAIT to determine if the Z buffer data will be stored or held. If ZWAIT is set, the controller continues examining the requests from the X and Y buffers. If ZWAIT is not set, the controller sets ZWAIT to indicate that a ZFILN fetch is in progress, enables the next clock to transfer the address in ZBA to the OA register for transfer to memory, sets the queue code to "10", indicating a destination of the Z buffer for the returning octet, and sets the protect mode bits to "01" to indicate to the MCU that this particular read operation is to be governed by the write protect bits (since the final result will be a write into that location). The controller then clears ZCB indicating a read operation to the MCU, sets OAFUL to indicate that a memory requests is in progress, and increments the queue input pointer for the next operation.

If no Z buffer operation is to be performed during this clock period, the controller examines the XBREQ, YBREQ and YNEXT signals. YNEXT resolves conflicts between X and Y data streams when simultaneous requests appear from both (YNEXT selects YBREQ; not YNEXT selects XBREQ). For either request, the controller transfers the memory address from the corresponding address register (XBA or YBA) into the OA register for transfer to memory, clears the request flag (XBREQ or YBREQ), toggles the YNEXT indicator so that the other buffer request will be selected during the next conflict, sets the protect mode bits to "10" to designate the read protect parameters to the MCU, and loads the corresponding code into the queue to indicate the origin of the request and ultimate destination of the octet from memory (X buffer = 00, Y buffer = 01). CMR then clears the zone control bits for the read operation, sets OAFUL to indicate that a memory operation is in progress, and increments the queue input pointer (CUEIP) to select the queue position for the next operation.

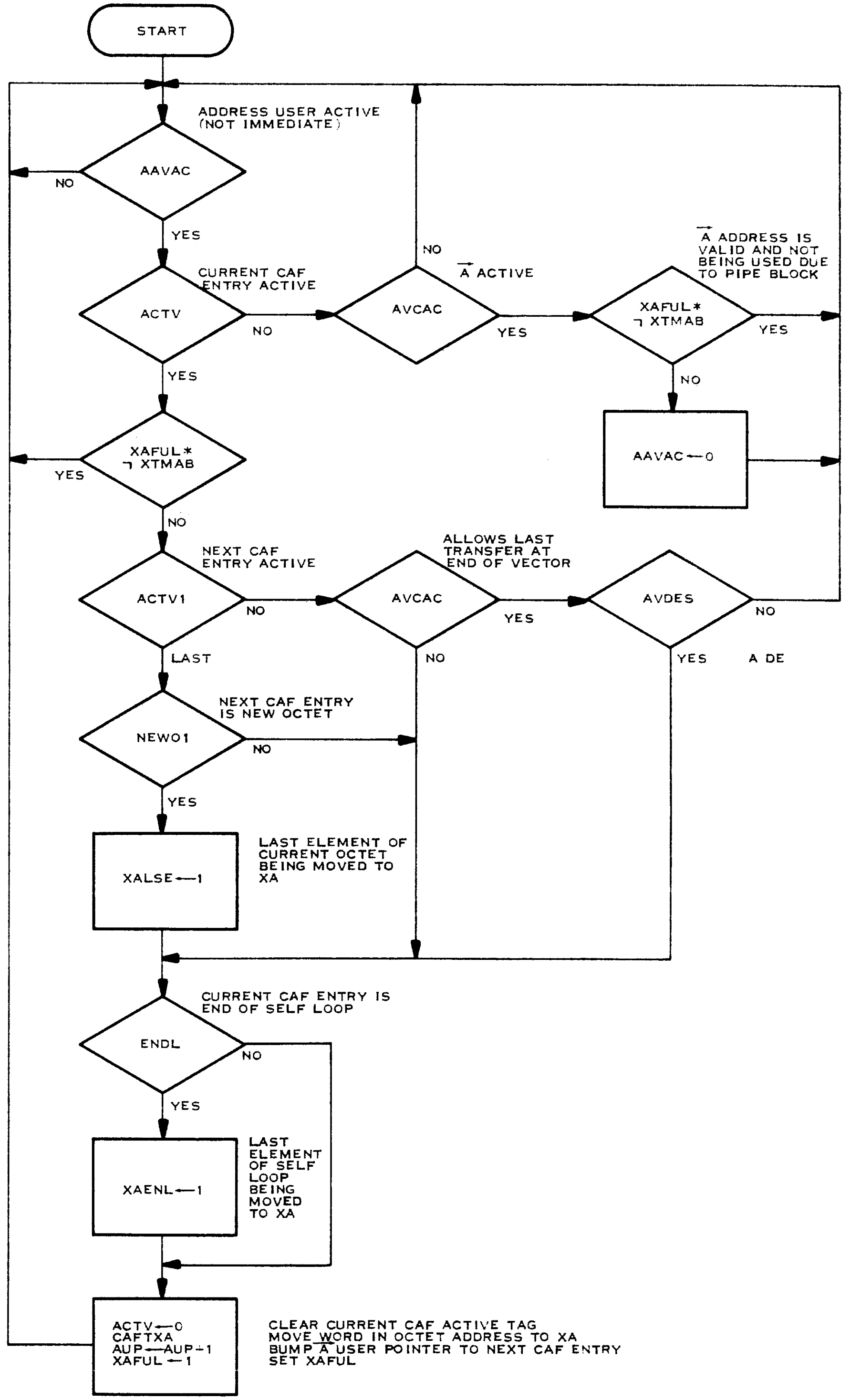
Regardless of the path through the decode cycle, the controller completes the control cycle by inspecting the SCFUL and WRACC flags. If SCFUL is set, the controller increments the queue output pointer (CUEOP) since the code has been used to select the destination of the new octet. If WRACC is present (write acknowledgement from the MCU), the last write operation is complete. The controller, therefore, clears ZBBSY indicating that the ZB buffer no longer contains needed data and that a new octet may be transferred into ZB for storage into memory.

4.28 OTHER CONTROL CIRCUITS

Flowcharts for the remaining controllers and control circuits in this volume are provided in figures 4-47 through 4-57. These circuits include:

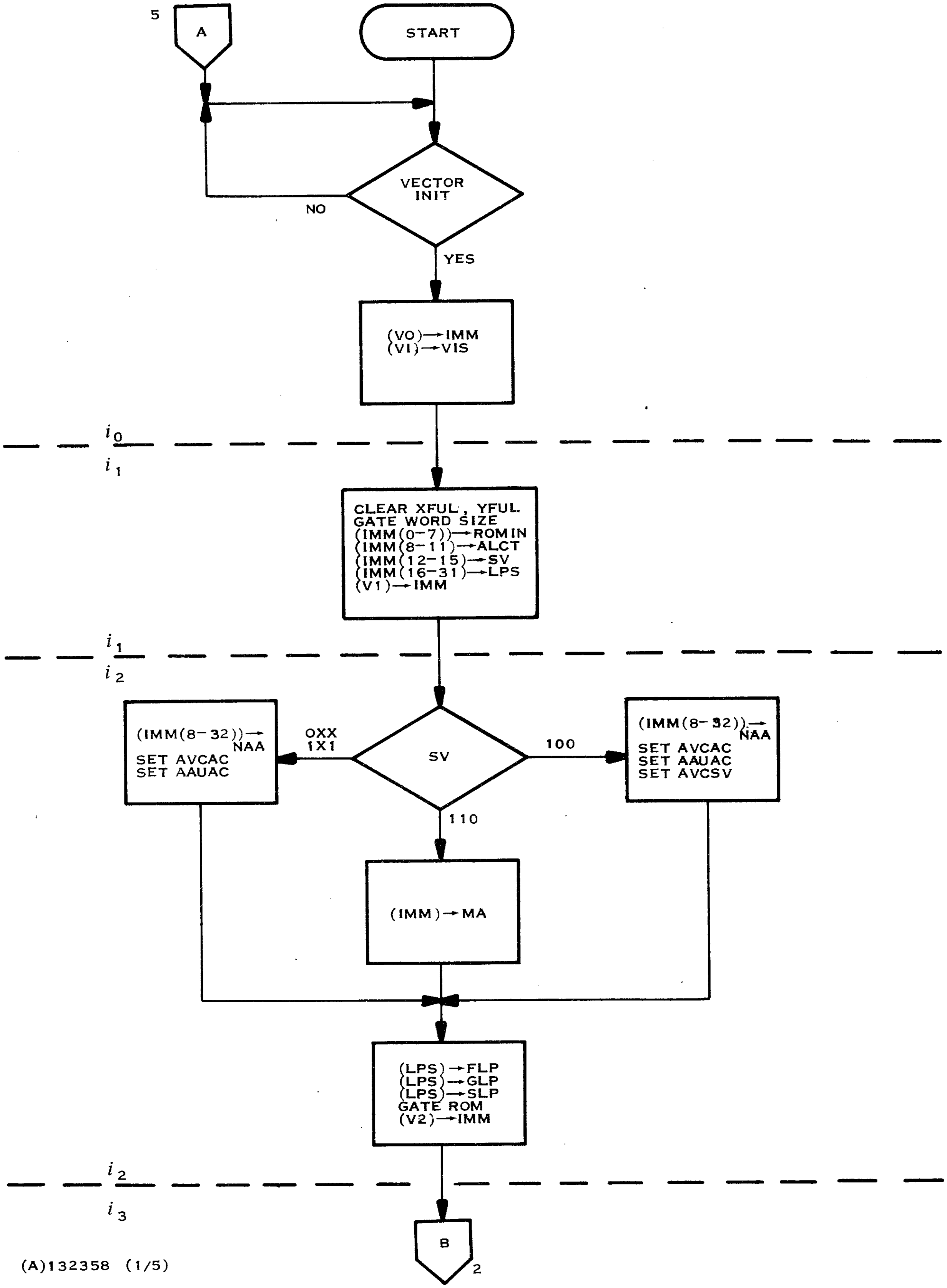
- CAF output control logic (figure 4-47)
- Vector initialization control (figure 4-48)
- AB vector address generation control (figure 4-49)
- C vector address generation (figure 4-50)
- AU control (figure 4-51 through 4-55)
- Z address flow and $\vec{C}$ control (figure 4-56)
- Z data flow (figure 4-57).

See Appendix B for the IPU's CMR and Look-ahead Controllers, and Appendix D for the Hard Core Controllers.


START
ADDRESS USER ACTIVE (NOT IMMEDIATE)
AAVAC
NO
YES
CURRENT CAF ENTRY ACTIVE
ACTV
NO
YES
NO
A ACTIVE
AVCAC
YES
A ADDRESS IS VALID AND NOT BEING USED DUE TO PIPE BLOCK
XAFUL* ¬ XTMAB
YES
NO
AAVAC←0
XAFUL* ¬ XTMAB
YES
NO
NEXT CAF ENTRY ACTIVE
ACTV1
NO
ALLOWS LAST TRANSFER AT END OF VECTOR
AVCAC
YES
AVDES
NO
LAST
NO
YES
A DE
NEXT CAF ENTRY IS NEW OCTET
NEWO1
NO
YES
XALSE←1
LAST ELEMENT OF CURRENT OCTET BEING MOVED TO XA
CURRENT CAF ENTRY IS END OF SELF LOOP
ENDL
NO
YES
XAENL←1
LAST ELEMENT OF SELF LOOP BEING MOVED TO XA
ACTV←0
CAFTXA
AUP←AUP+1
XAFUL←1
CLEAR CURRENT CAF ACTIVE TAG
MOVE WORD IN OCTET ADDRESS TO XA
BUMP A USER POINTER TO NEXT CAF ENTRY
SET XAFUL

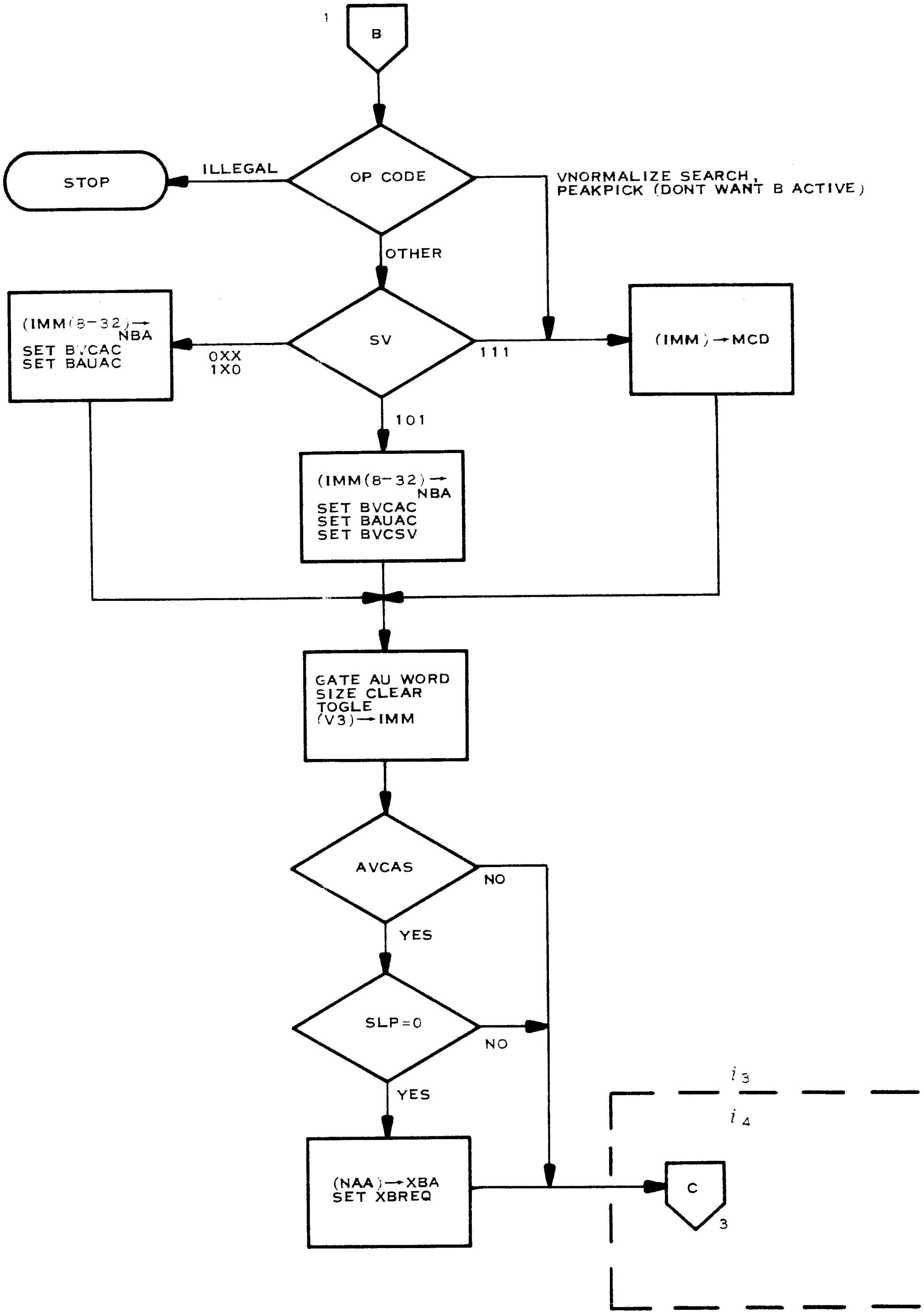
(B)132357

Figure 4-47. CAF Output Control Flowchart



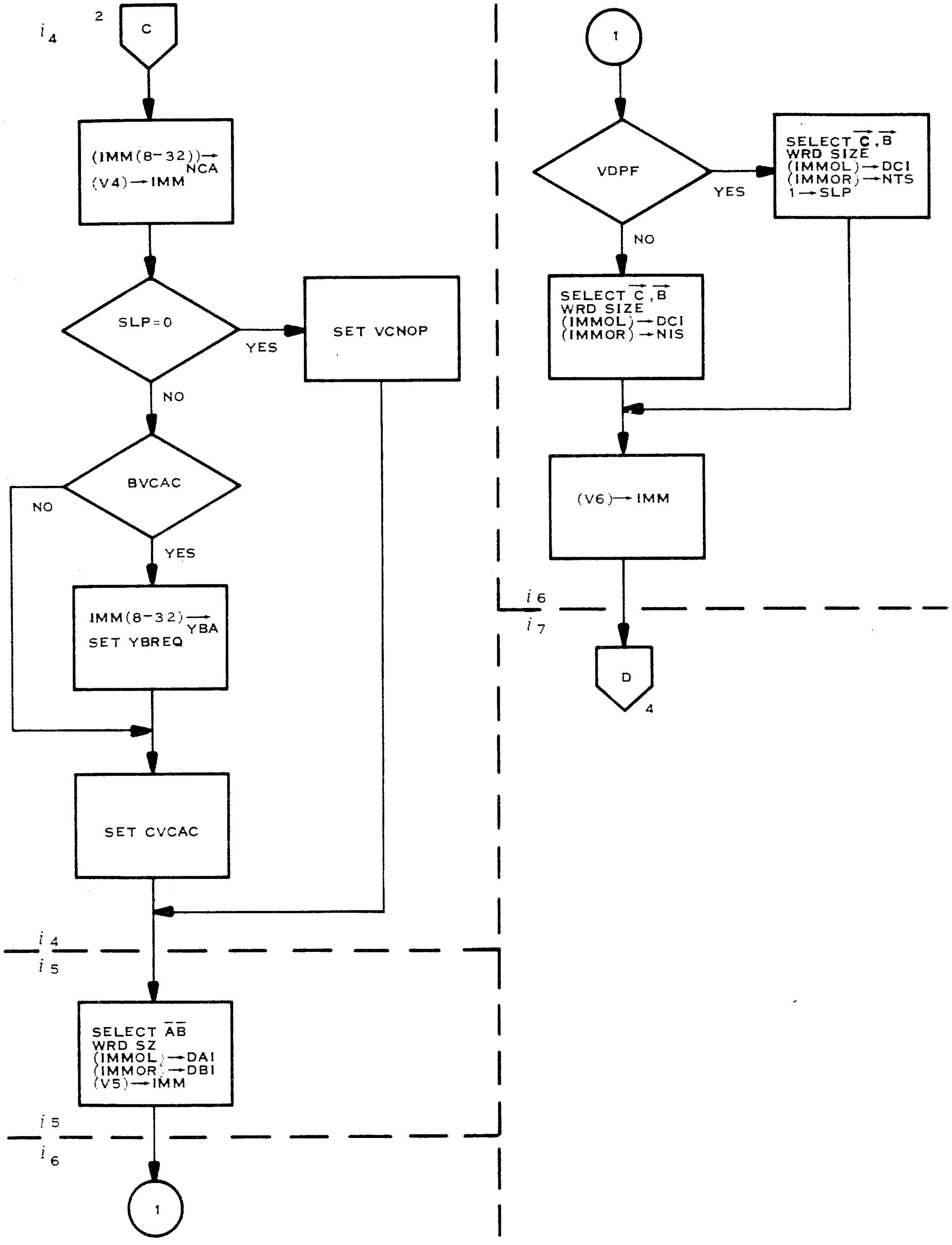
(A)132358 (1/5)

Figure 4-48. Vector Initialization Control Flowchart (Sheet 1 of 5)



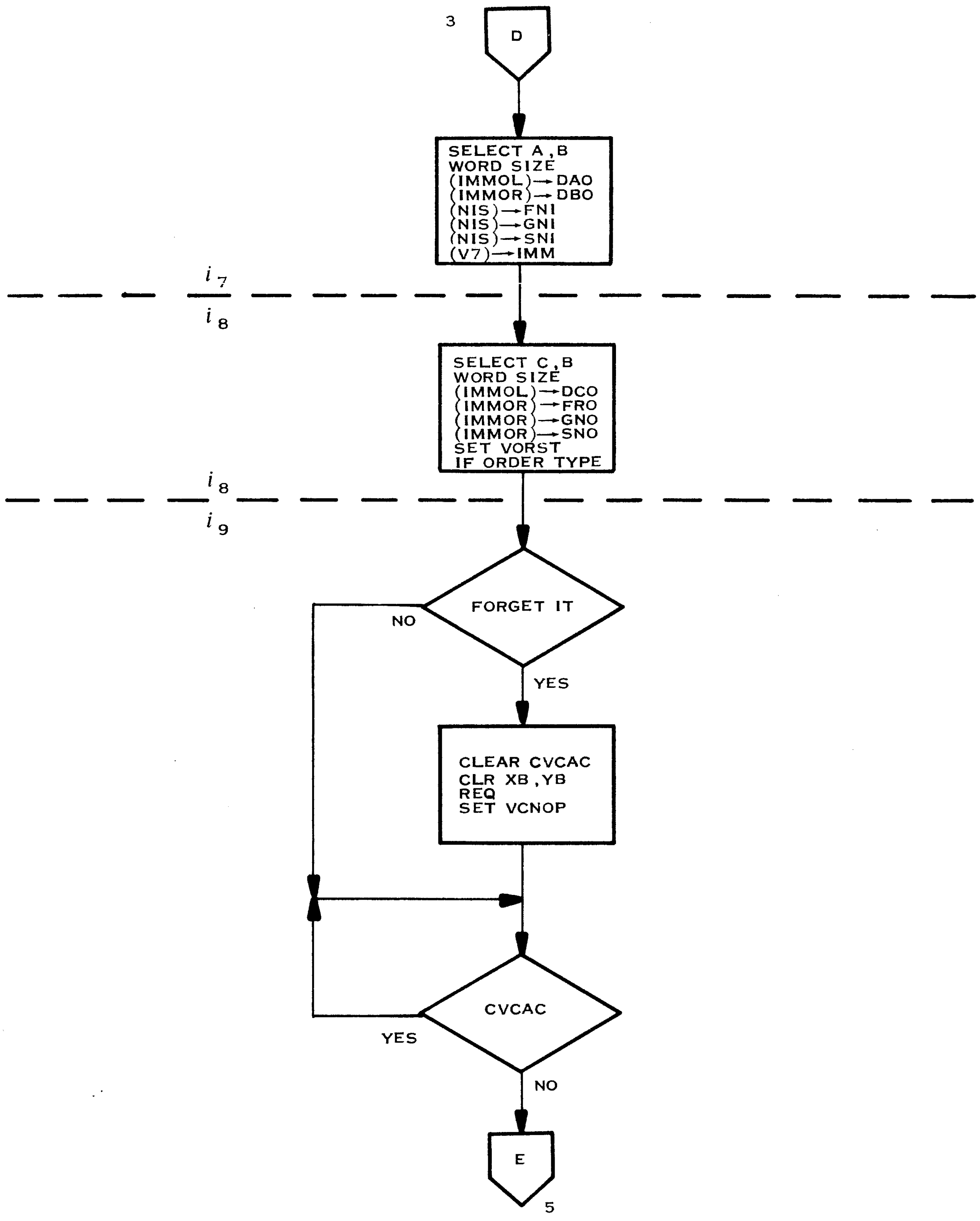
(A) 132358 (2 5)

Figure 4-48. Vector Initialization Control Flowchart (Sheet 2 of 5)



(A)132358 (3/5)

Figure 4-48. Vector Initialization Control Flowchart (Sheet 3 of 5)



NOTE: FORGET IT CLEARS CVCAC IN ANY STATE, NOT JUST STATE 9

(A)132358 (4/5)

Figure 4-48. Vector Initialization Control Flowchart (Sheet 4 of 5)

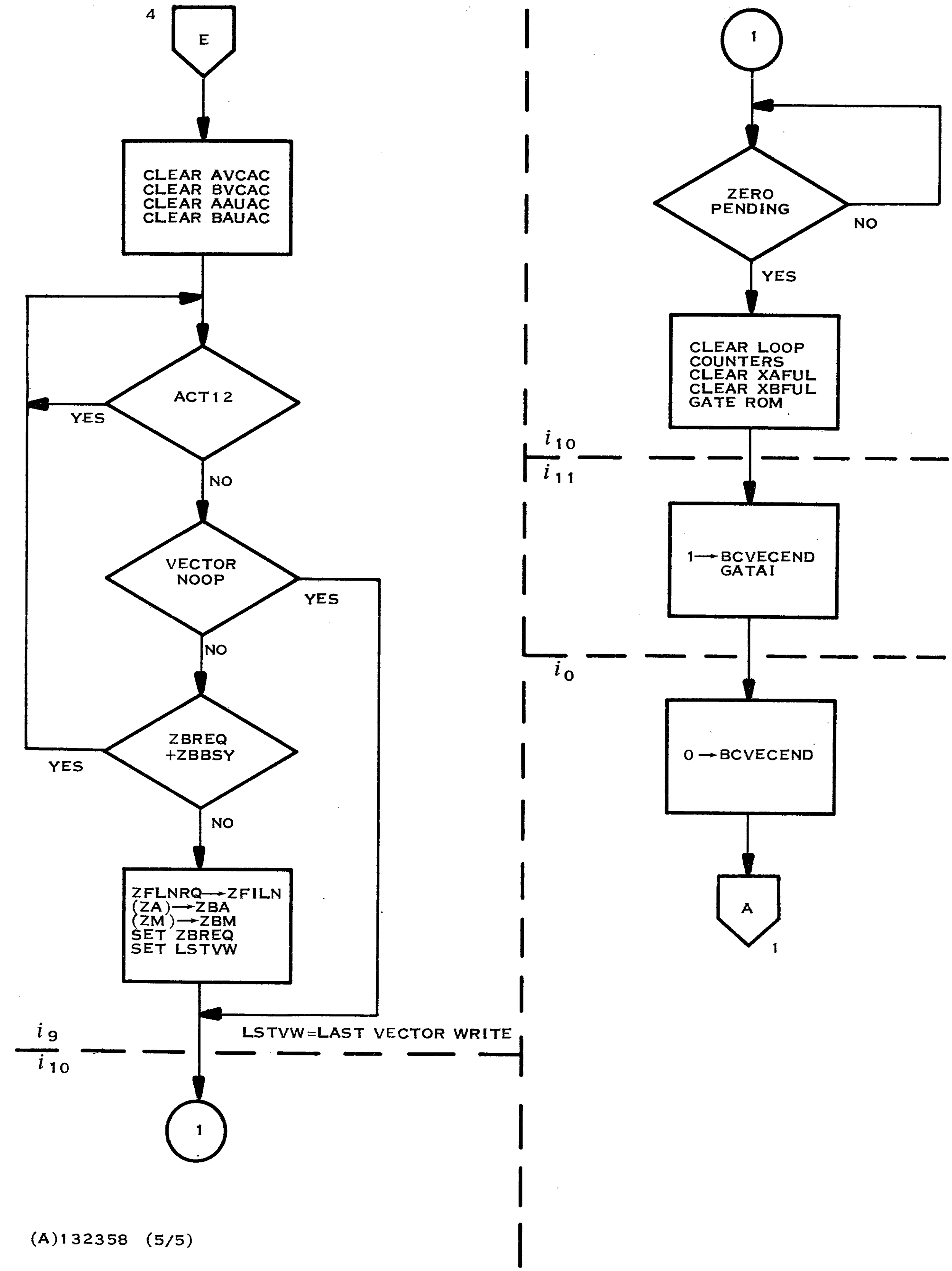
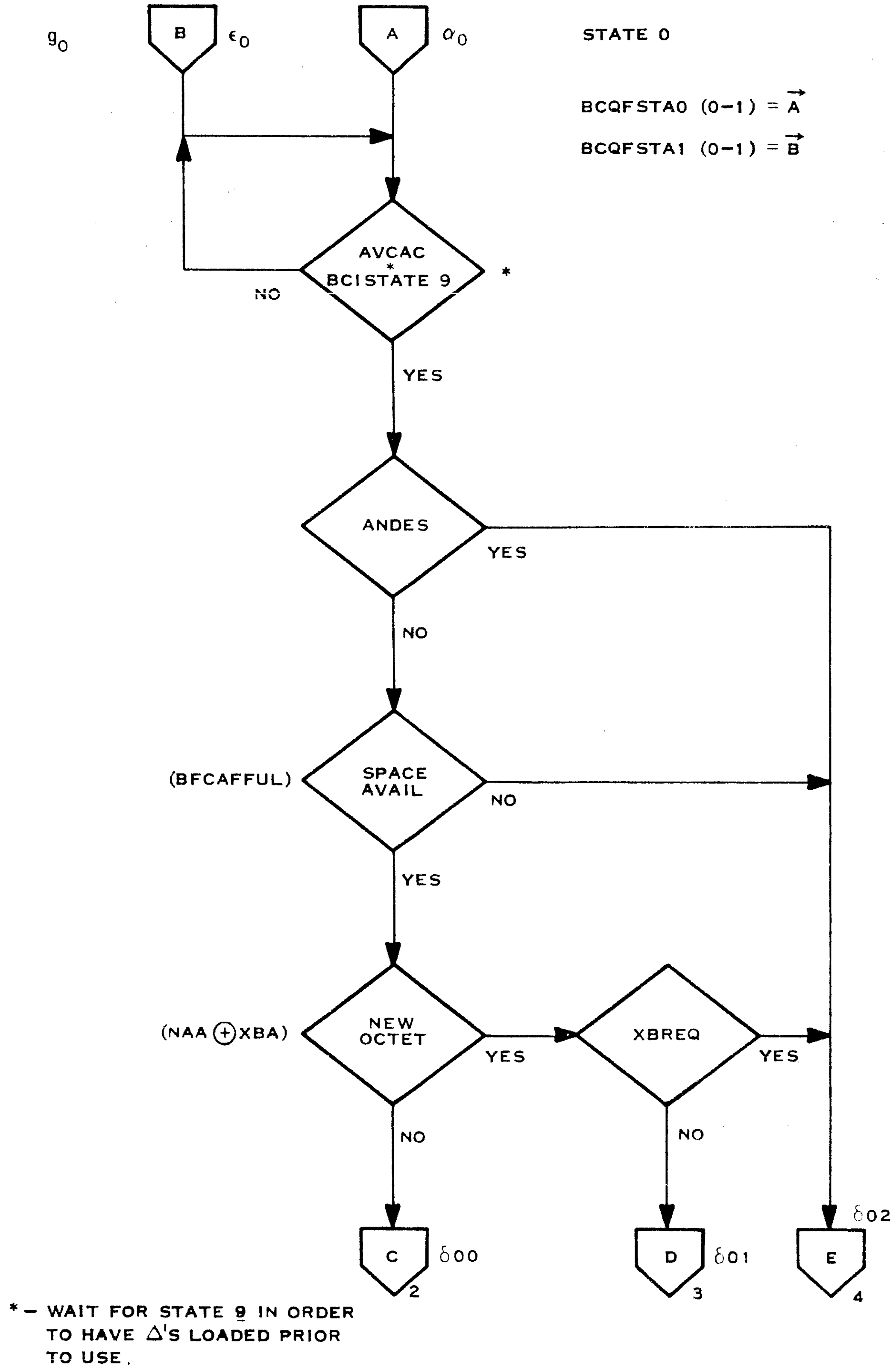
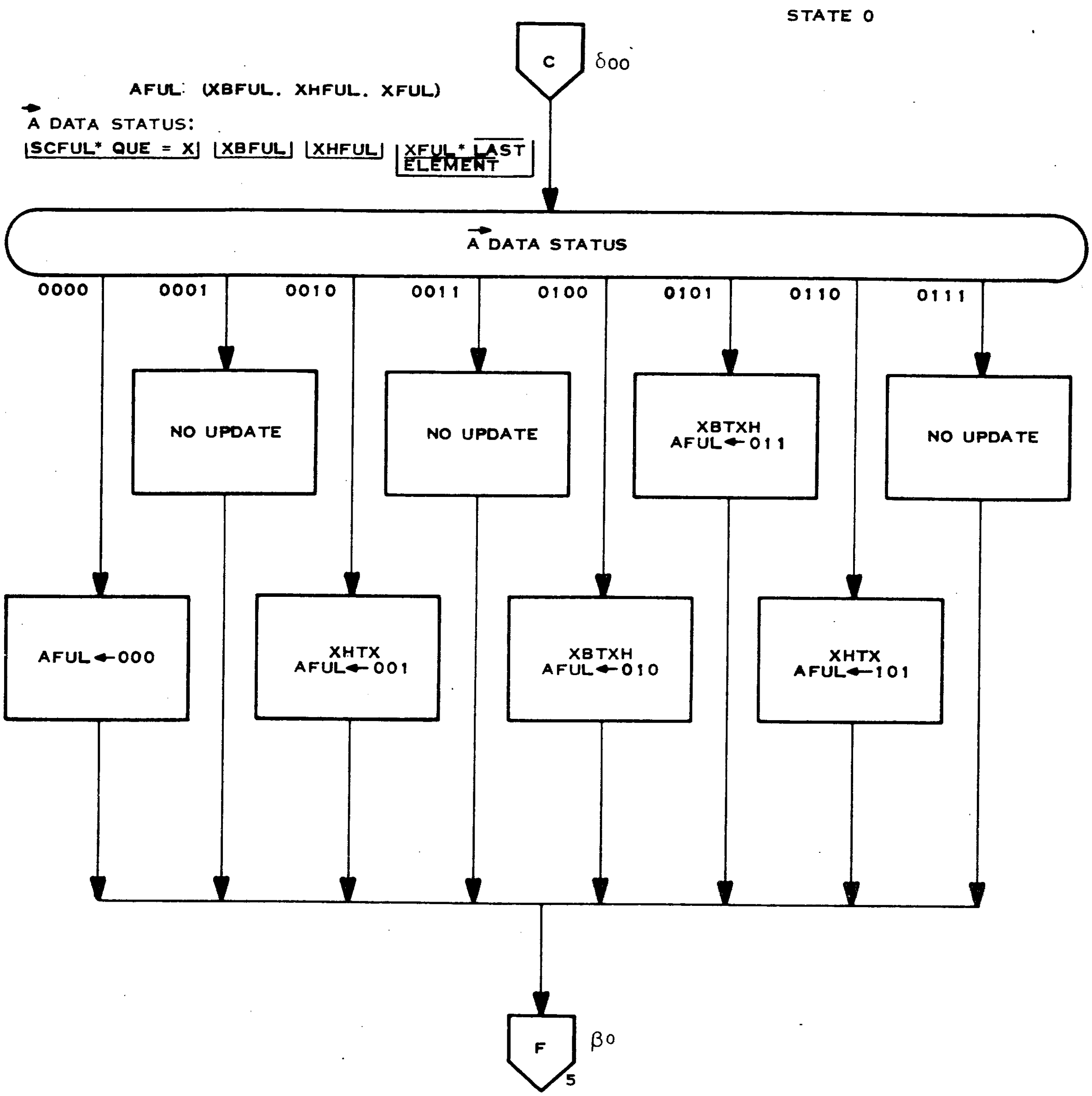


Figure 4-48. Vector Initialization Control Flowchart (Sheet 5 of 5)



(A) 115800

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 1 of 17)



(A)115801

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 2 of 17)

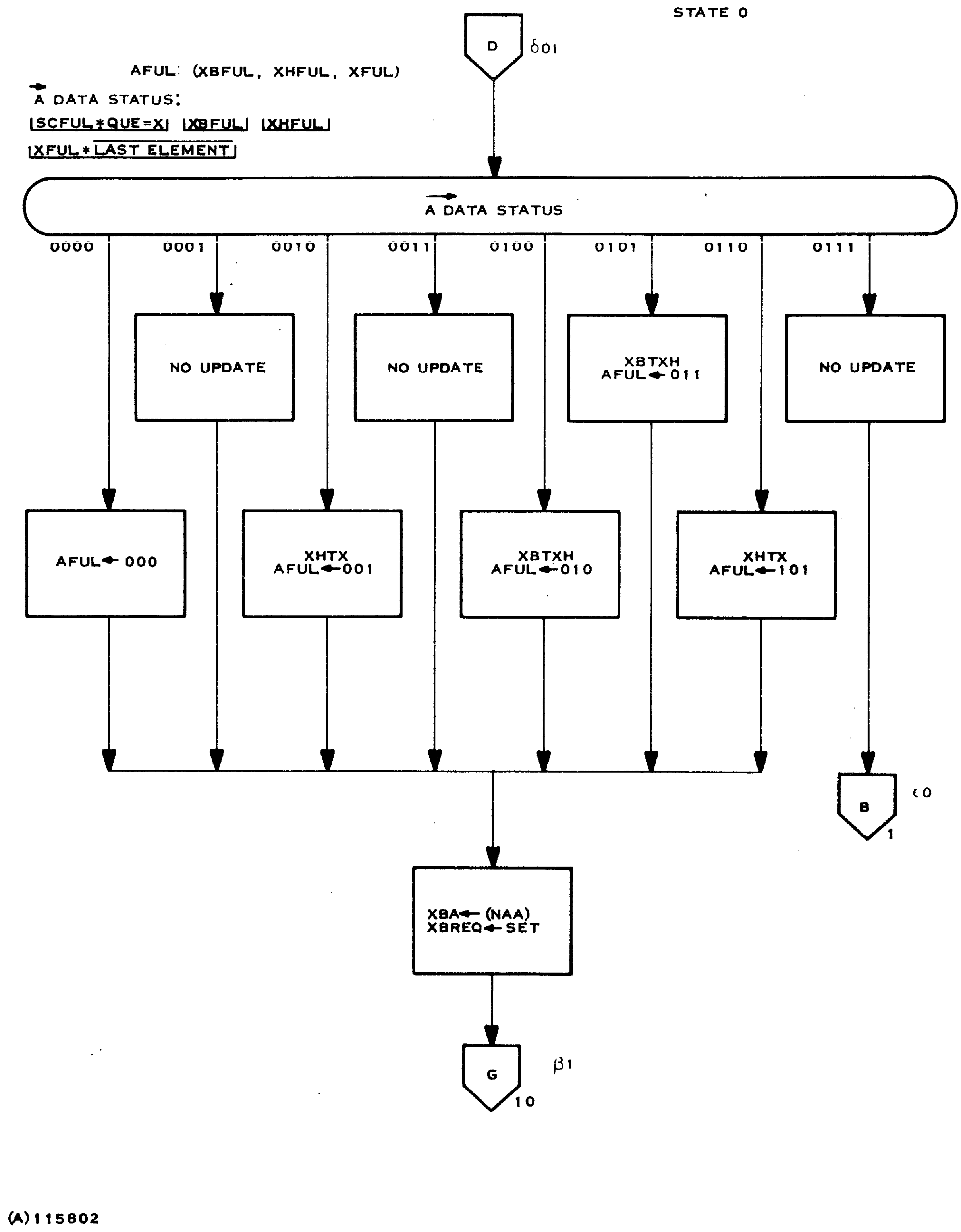
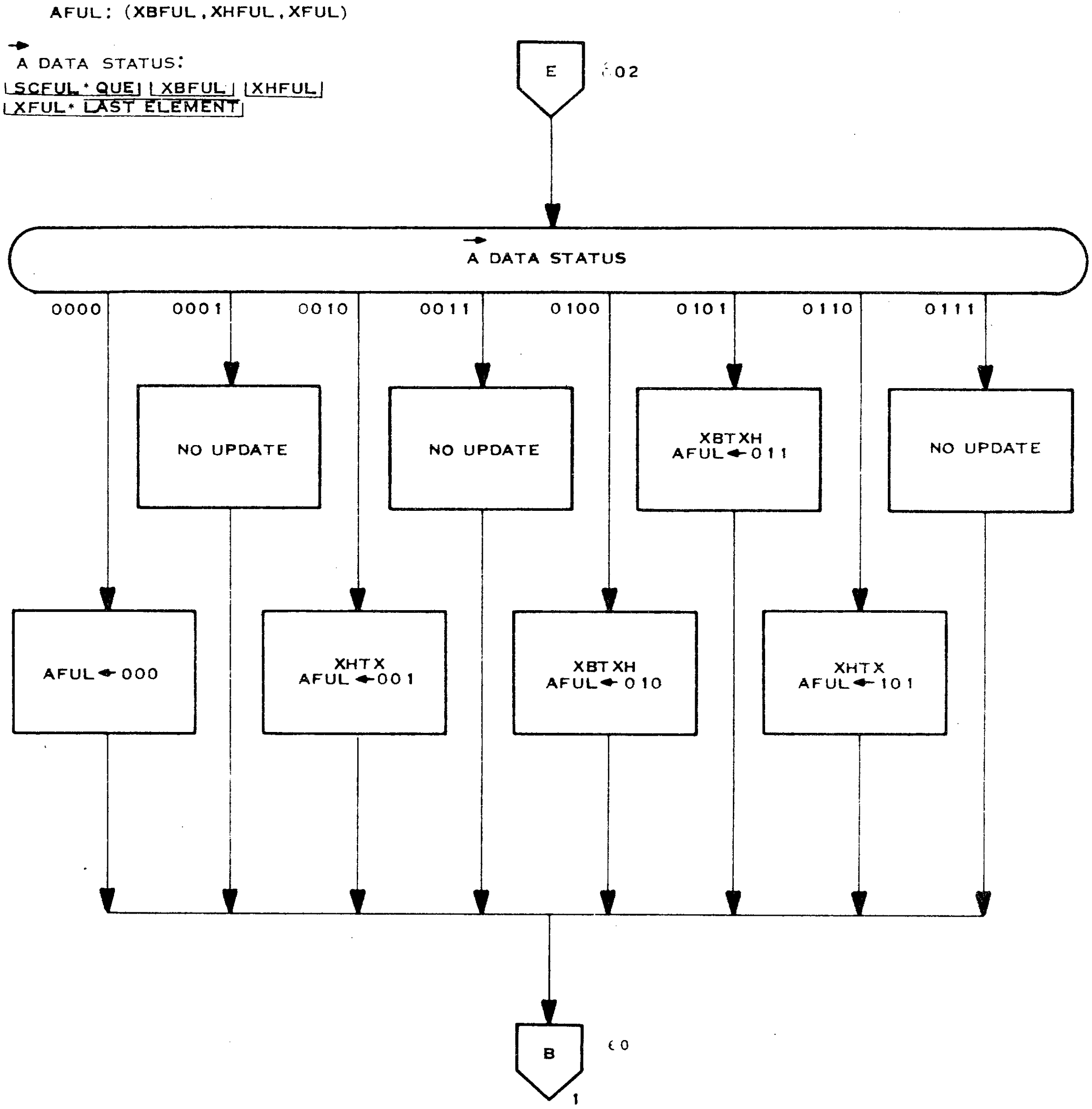


Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 3 of 17)

STATE 0



(A) 115803

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 4 of 17)

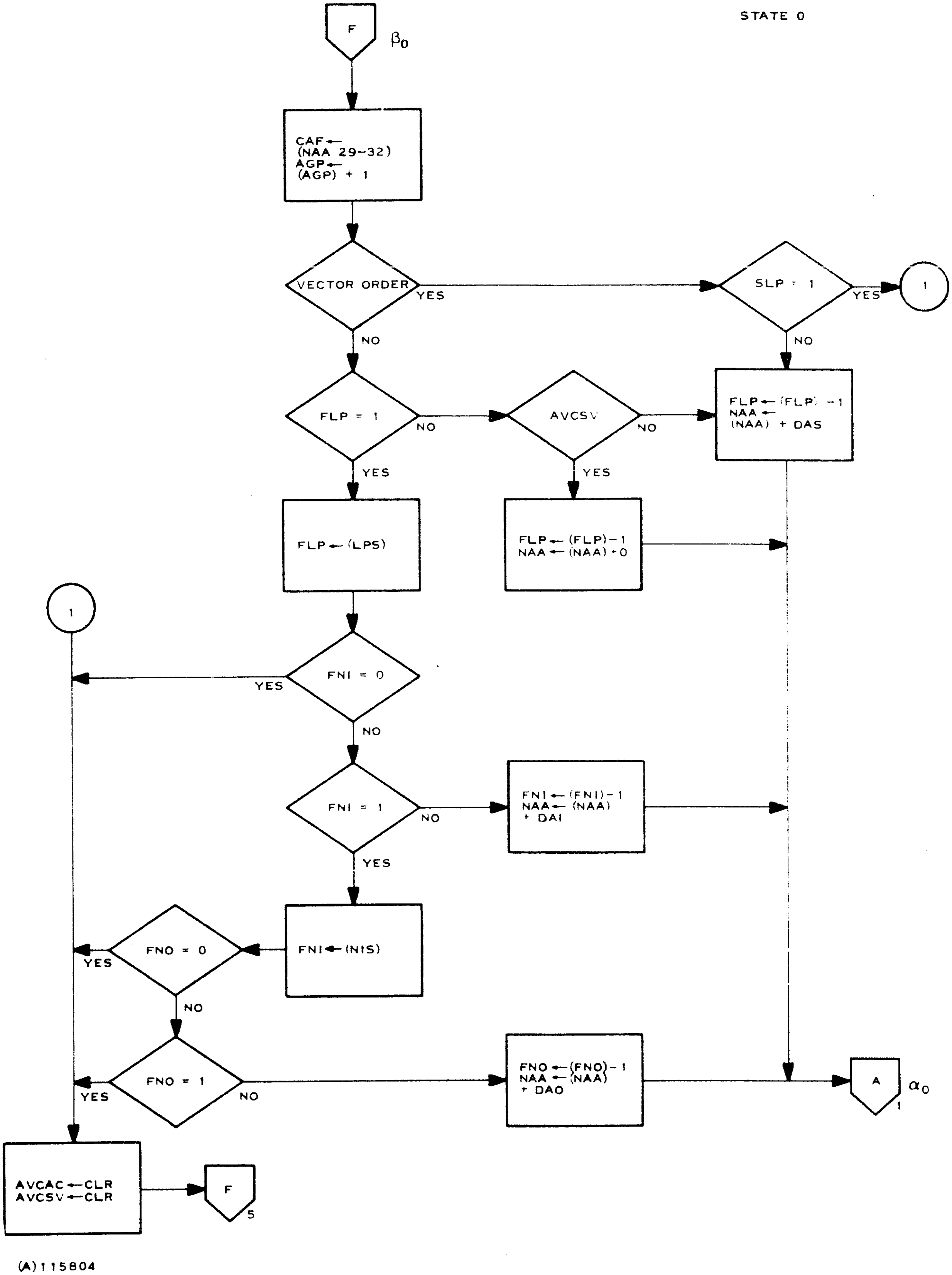
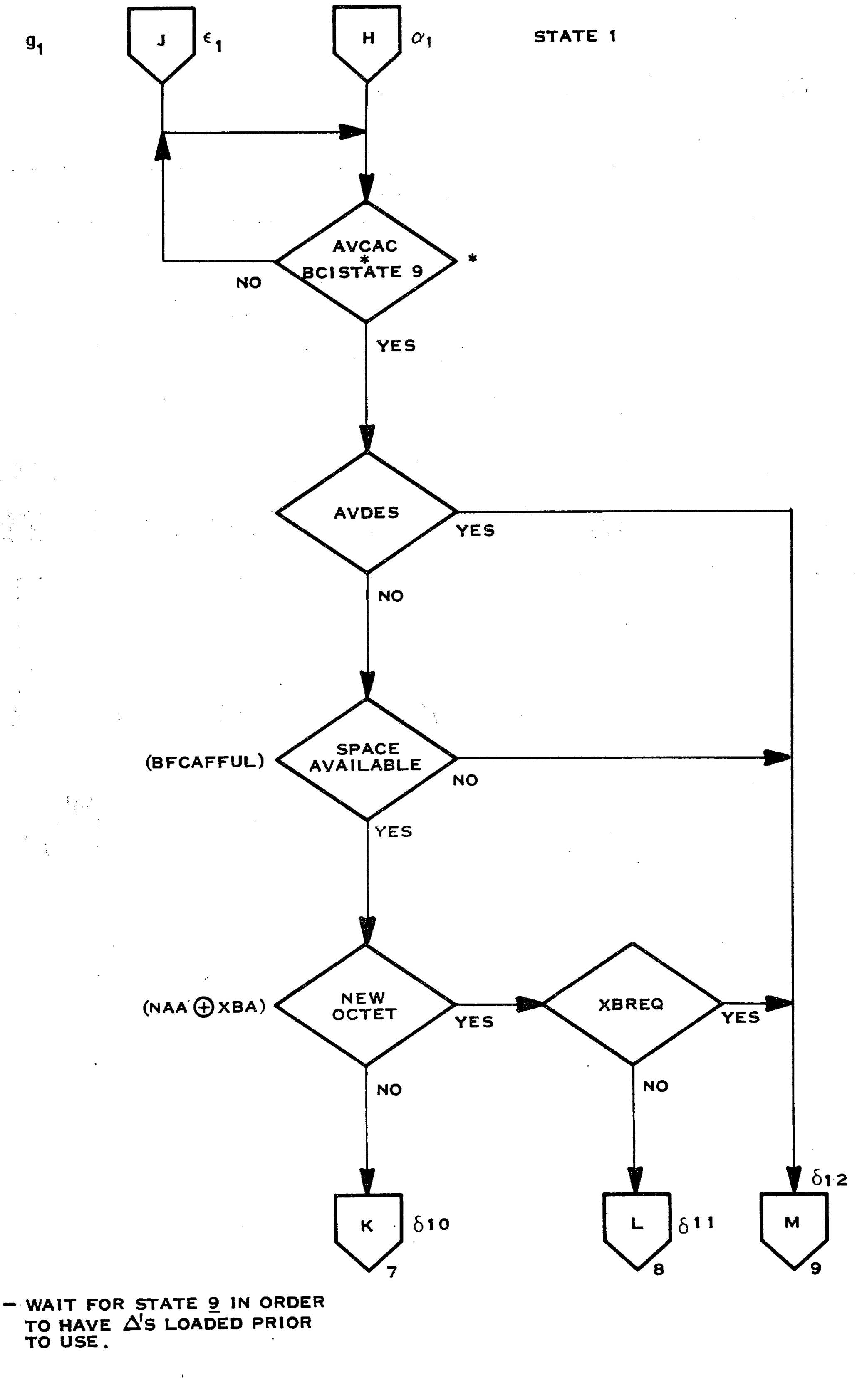
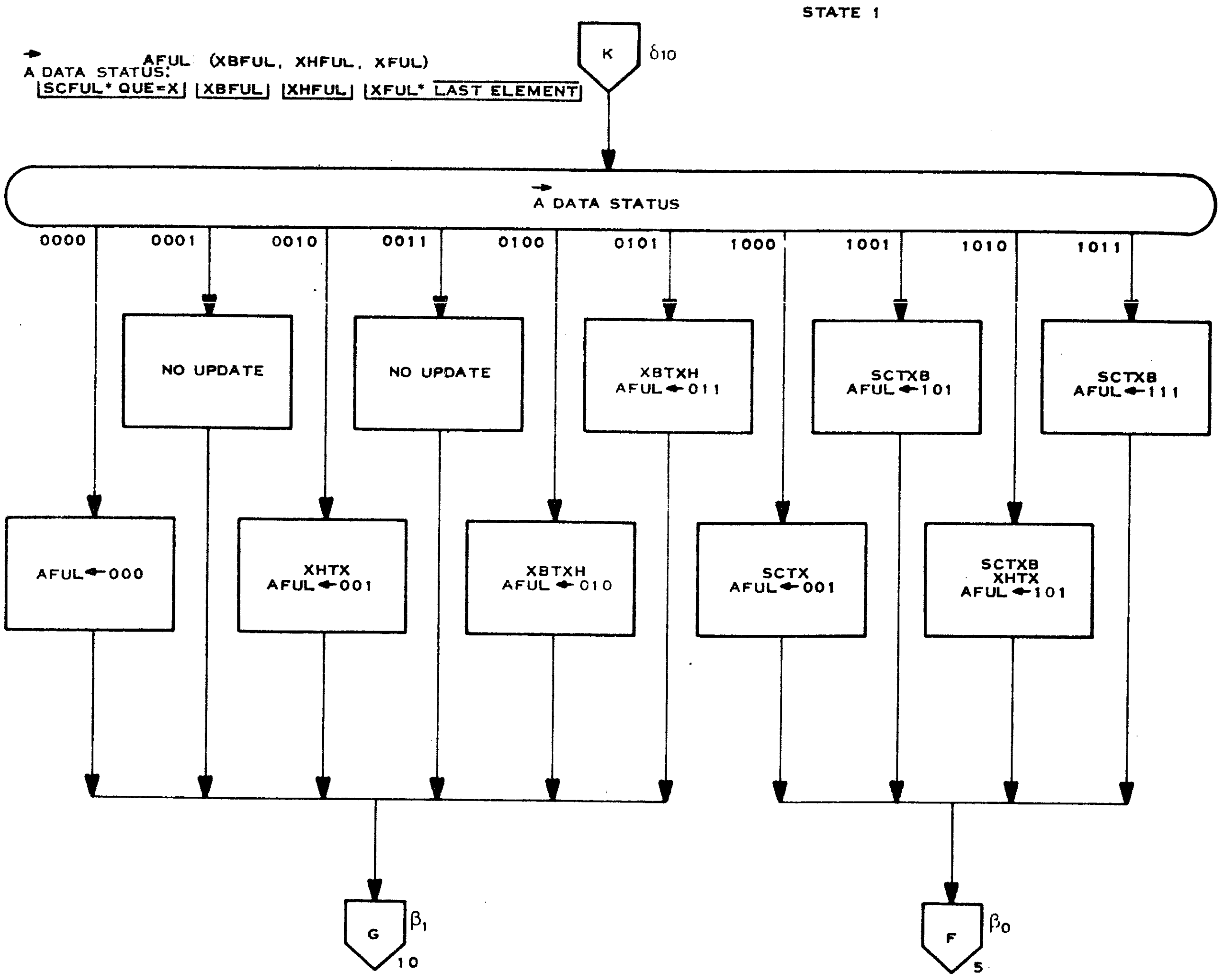


Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 5 of 17)



(A) 115805

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 6 of 17)



(A)115806

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 7 of 17)

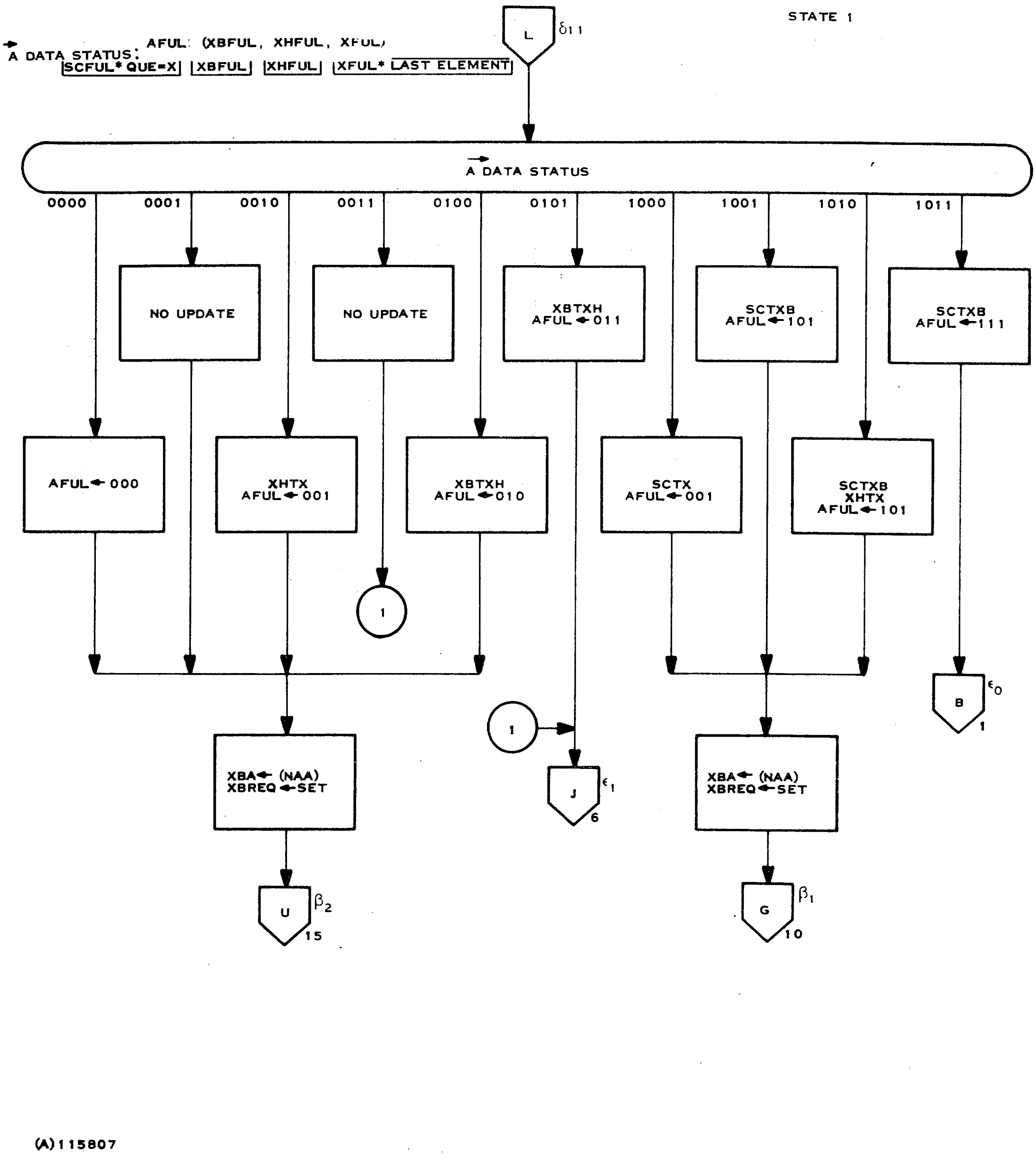
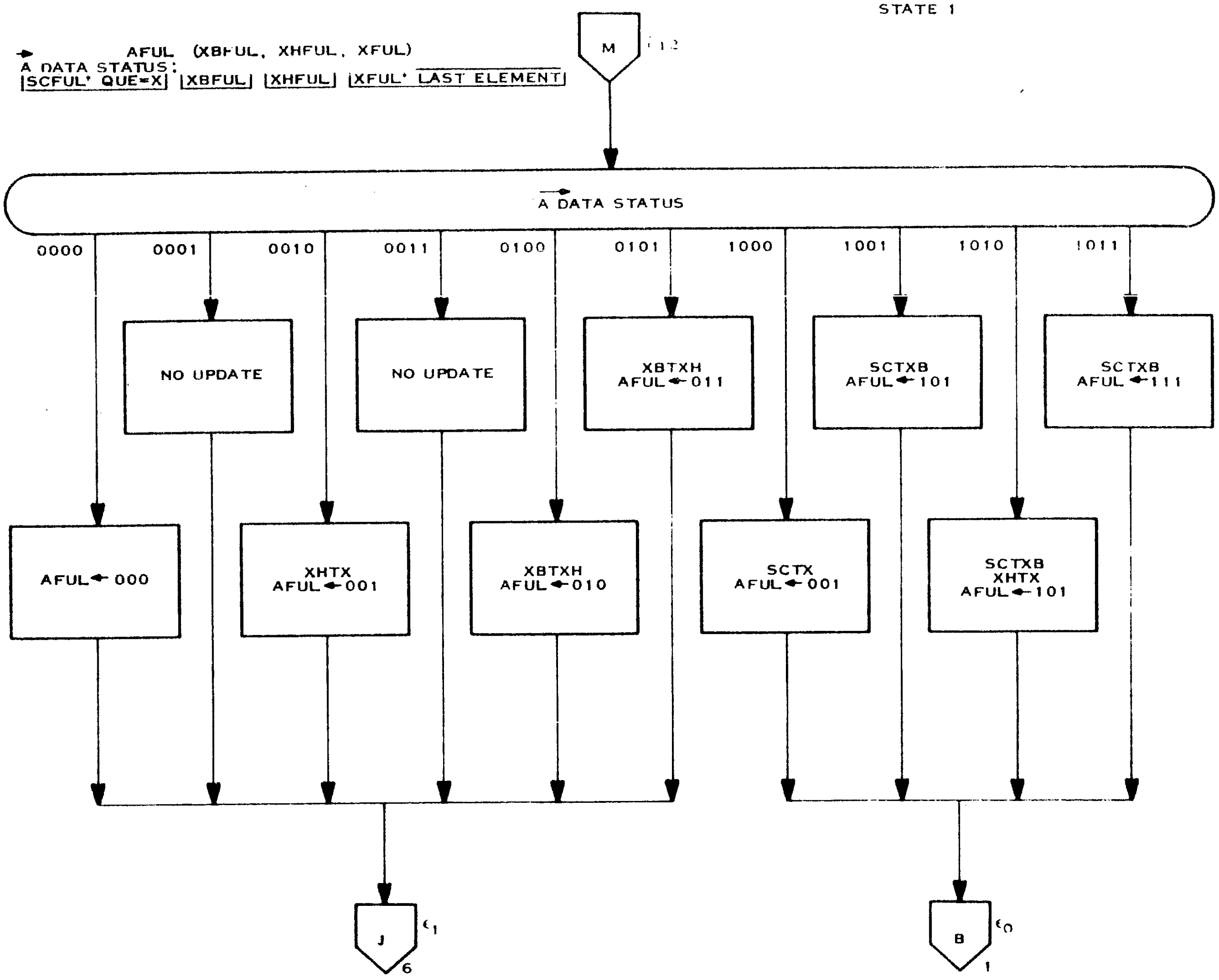
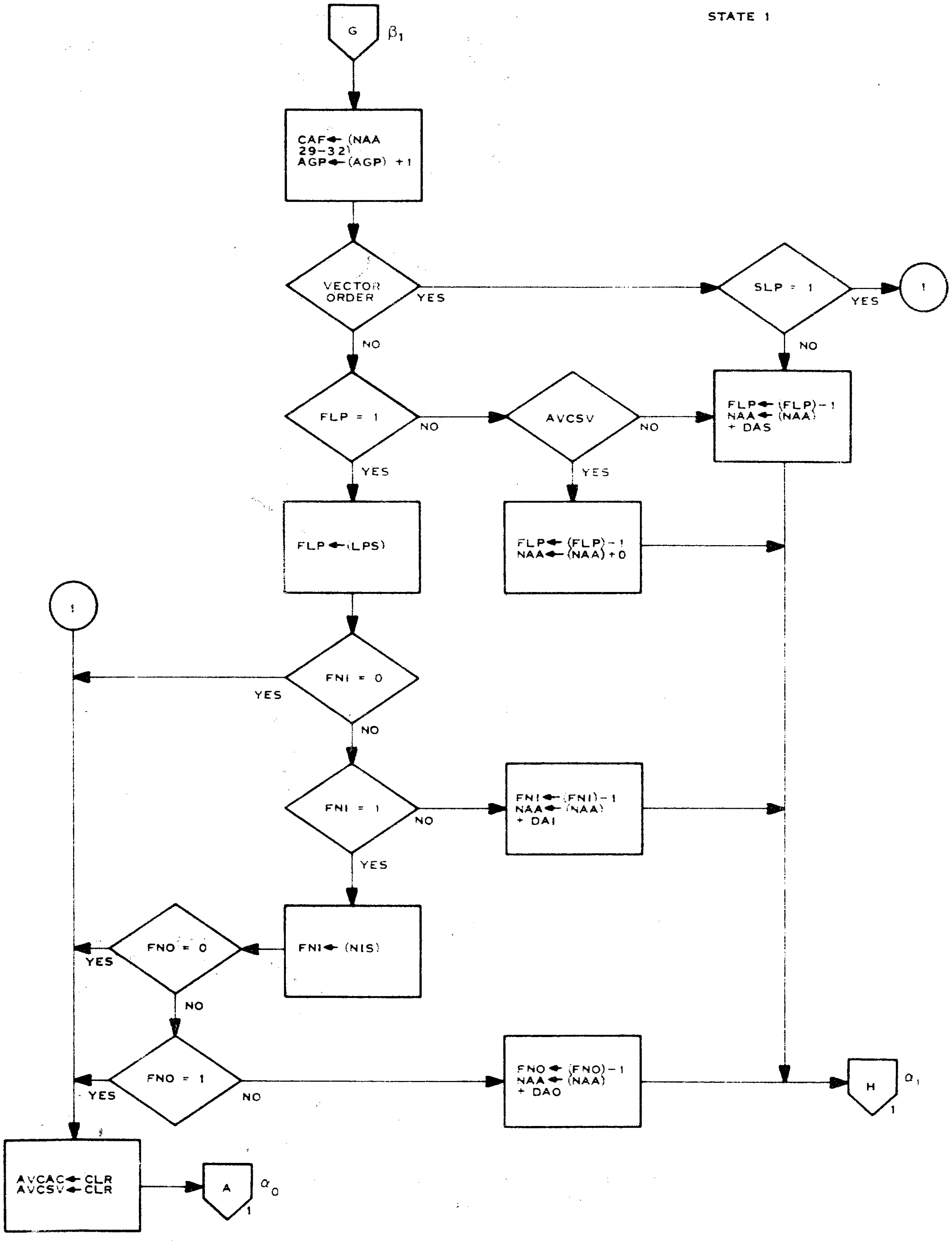


Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 8 of 17)



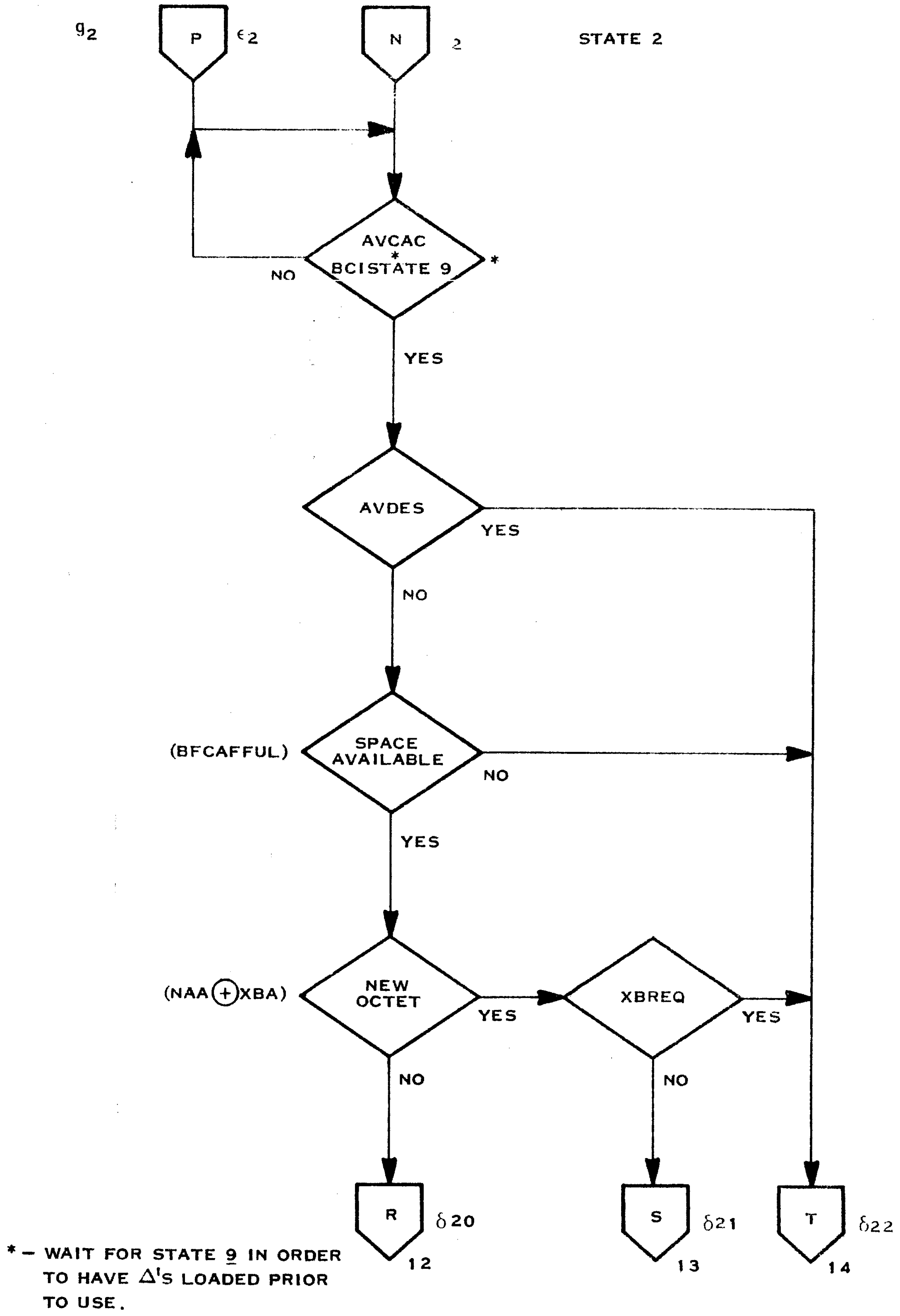
(A)115808

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 9 of 17)



(A)115809

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 10 of 17)



(A) 115810

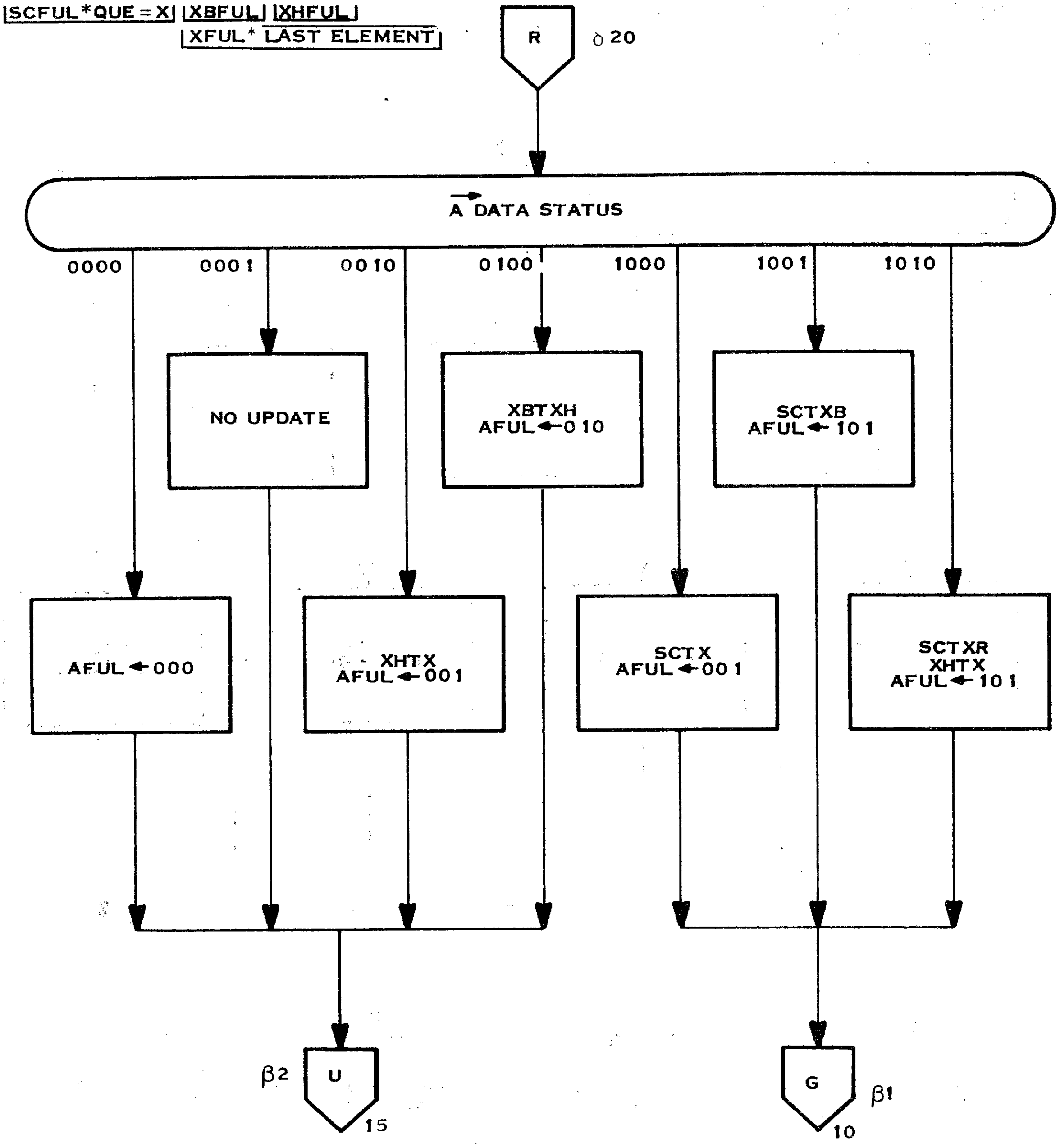
Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 11 of 17)

STATE 2

AFUL: (XBFUL, XHFUL, XFUL)

$\vec{A}$ DATA STATUS:

|SCFUL*QUE = X| |XBFUL| |XHFUL|
|XFUL* $\overline{\text{LAST ELEMENT}}$|



(A) 115811

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 12 of 17)

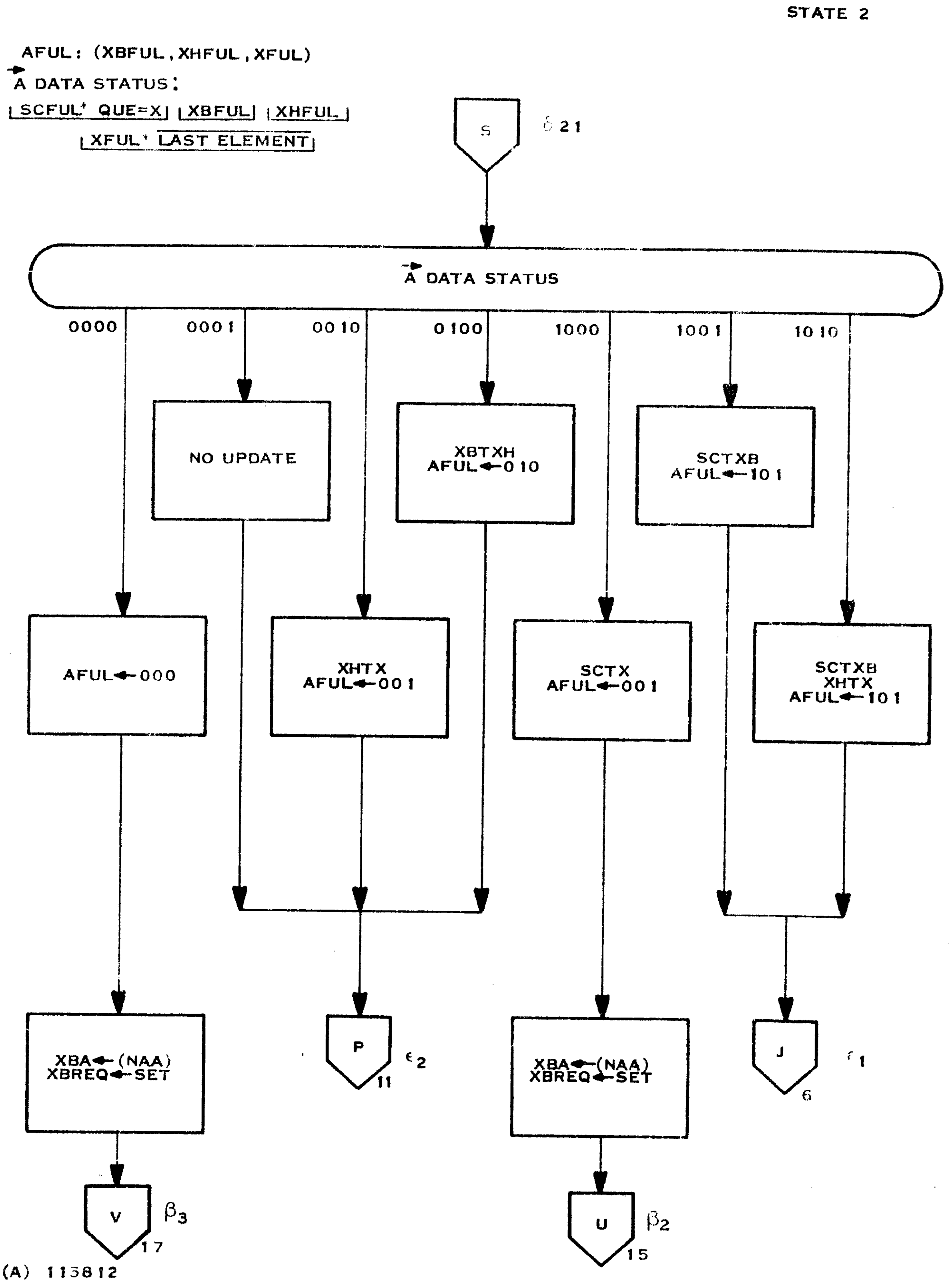


Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 13 of 17)

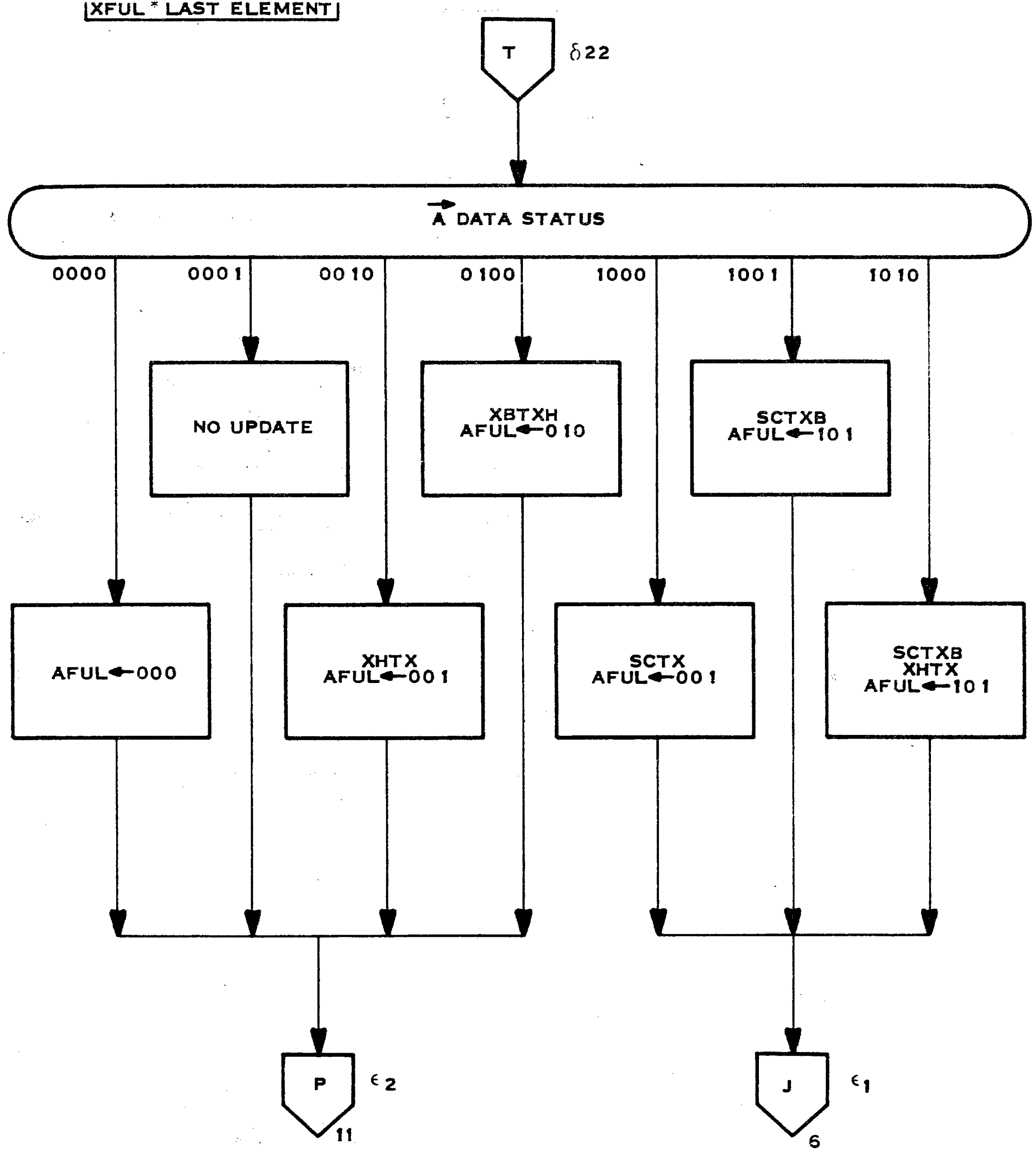
STATE 2

AFUL: (XBFUL, XHFUL, XFUL)

$\overrightarrow{A}$ DATA STATUS:

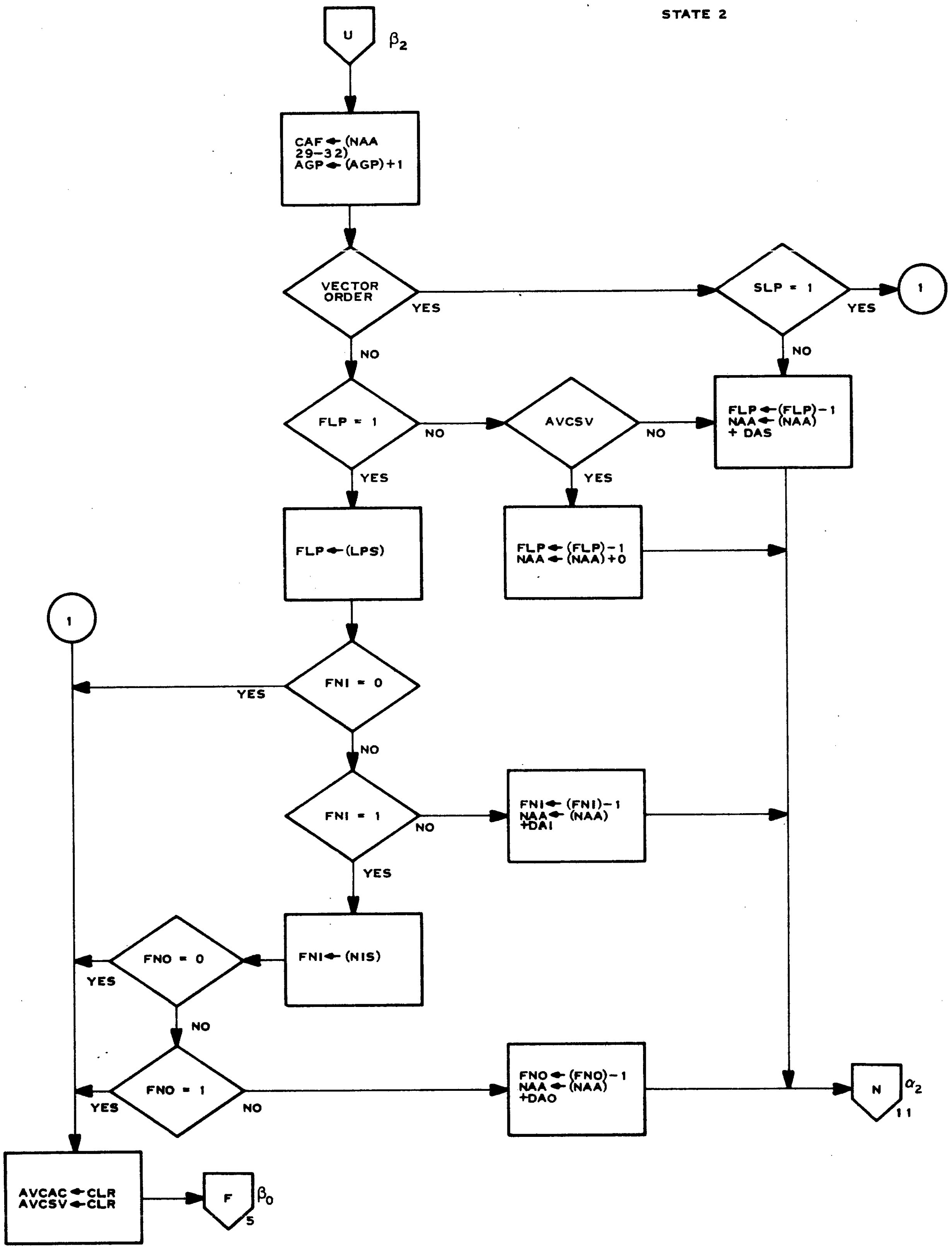
|SCFUL * QUE = X| |XBFUL| |XHFUL|

|XFUL * $\overline{\text{LAST ELEMENT}}$|



(A) 115813

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 14 of 17)



(A)115814

Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 15 of 17)

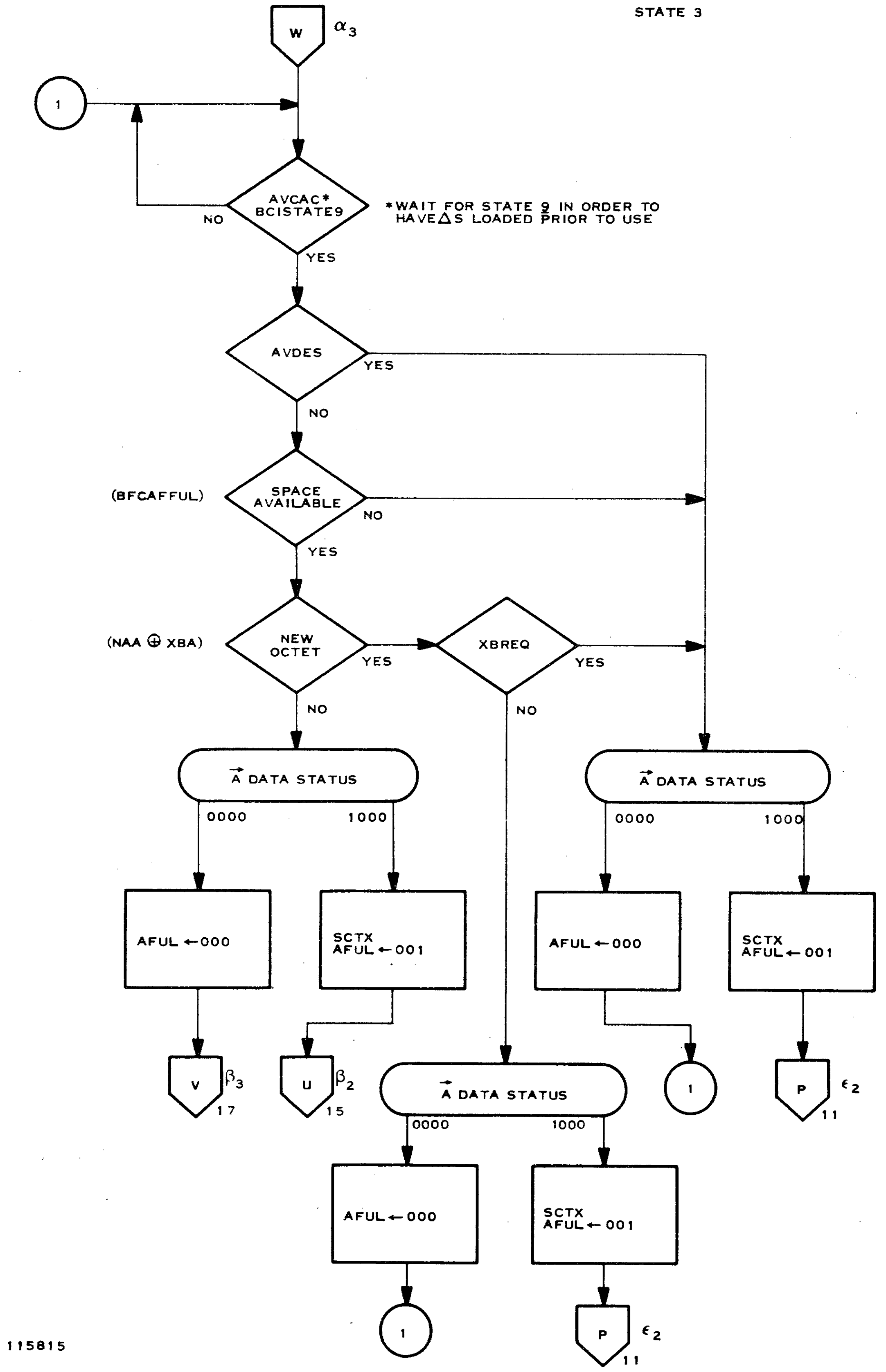


Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 16 of 17)

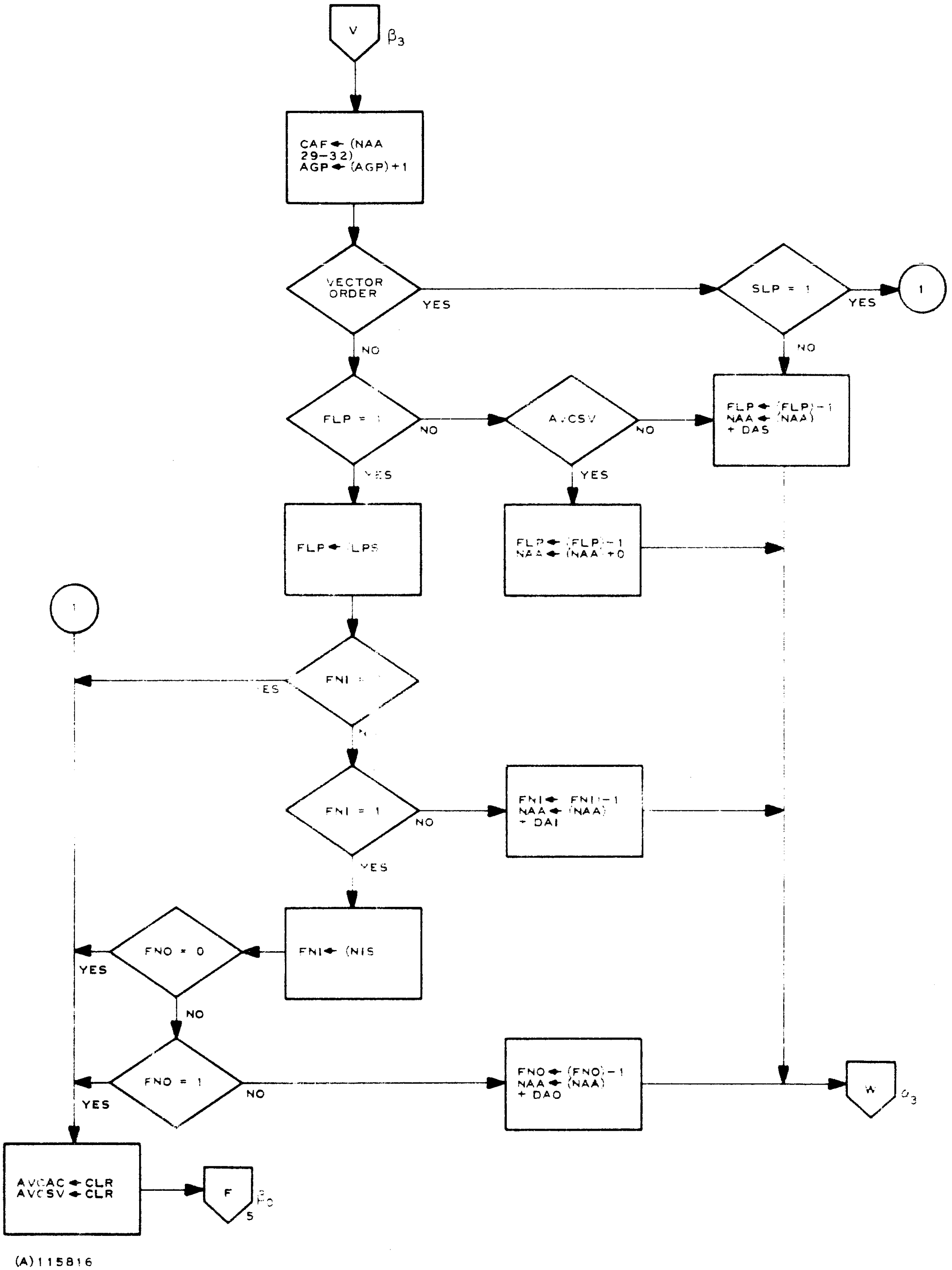


Figure 4-49. A/B Vector Address Generation Flowchart (Sheet 17 of 17)

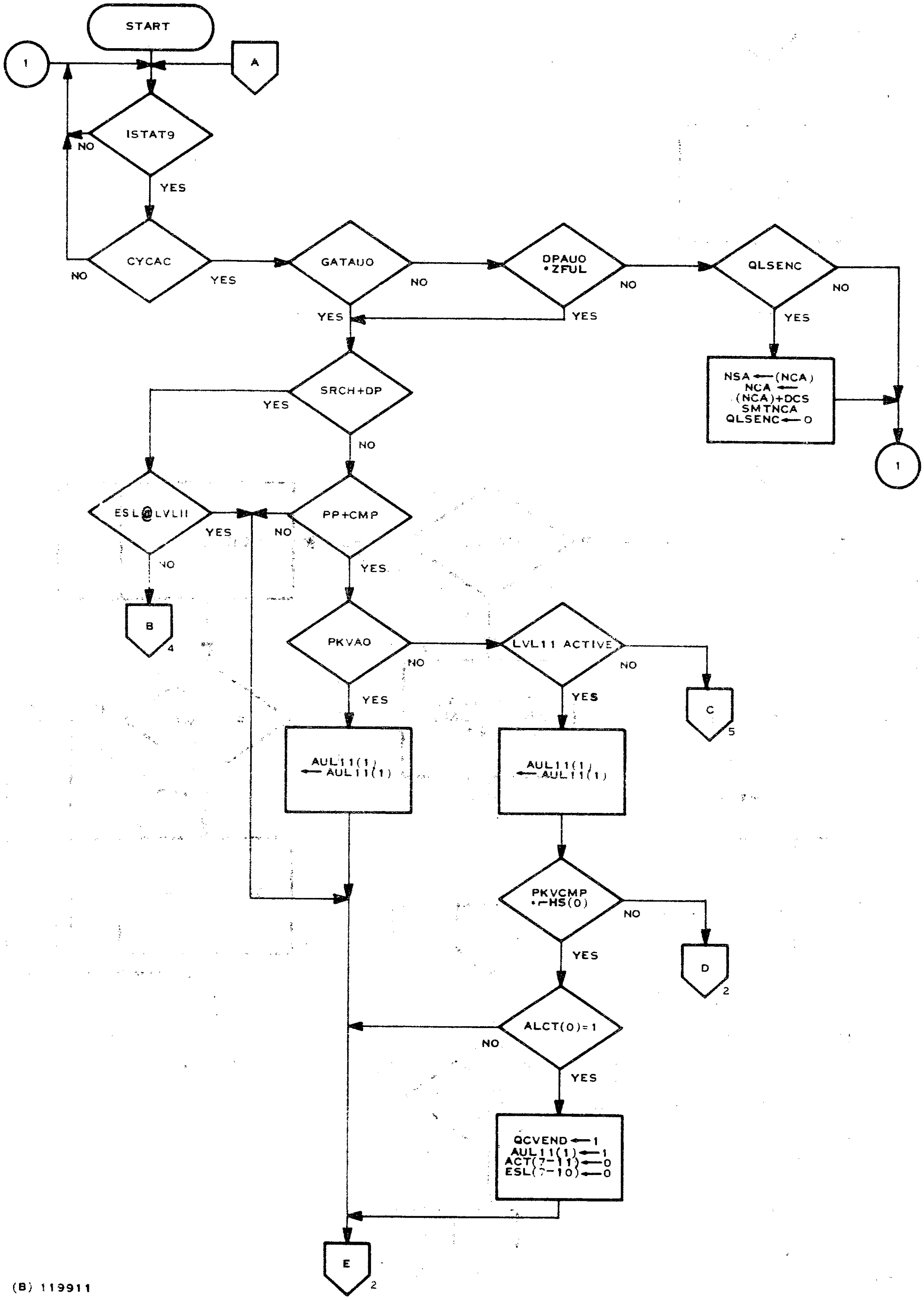


Figure 4-50. C Vector Address Generation Flowchart (Sheet 1 of 6)

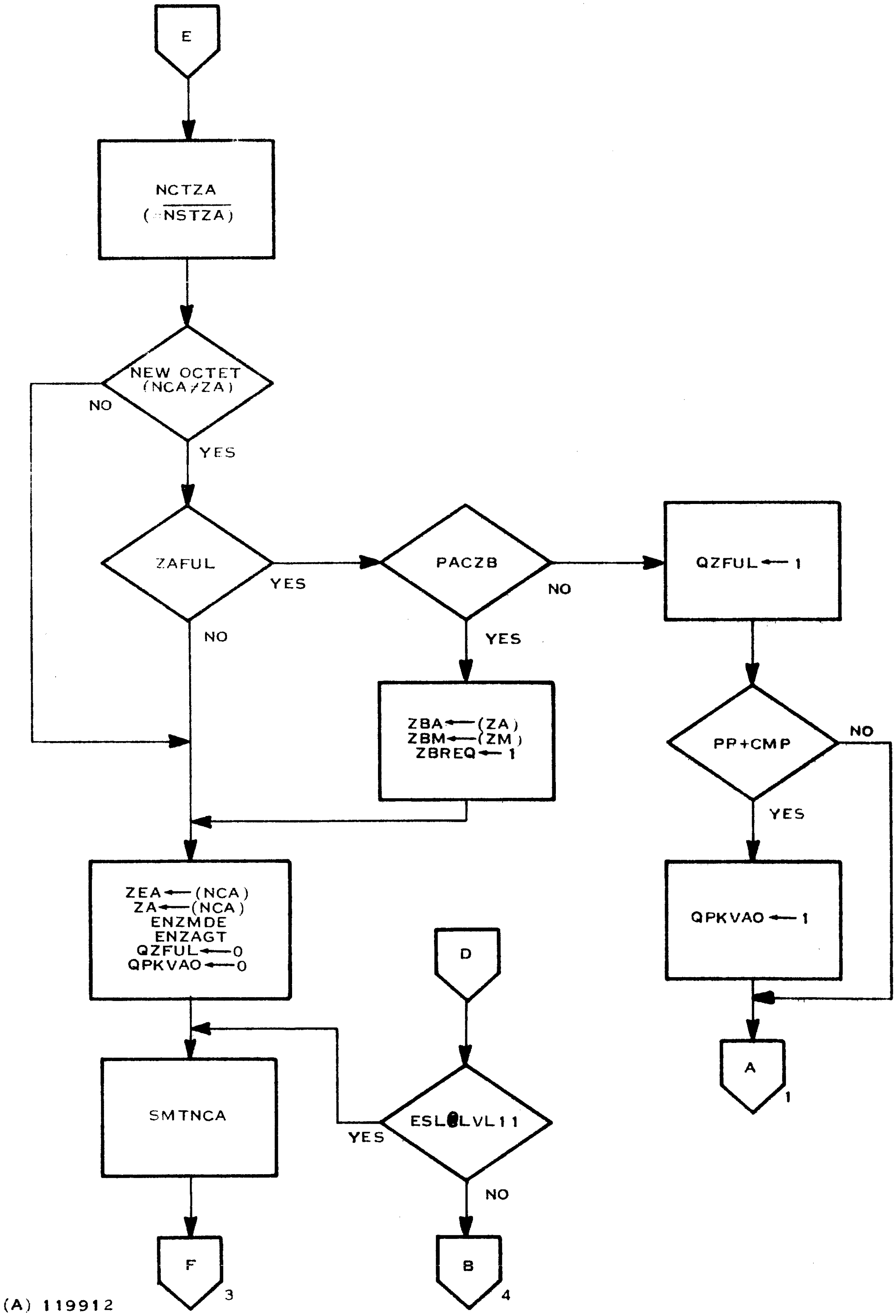
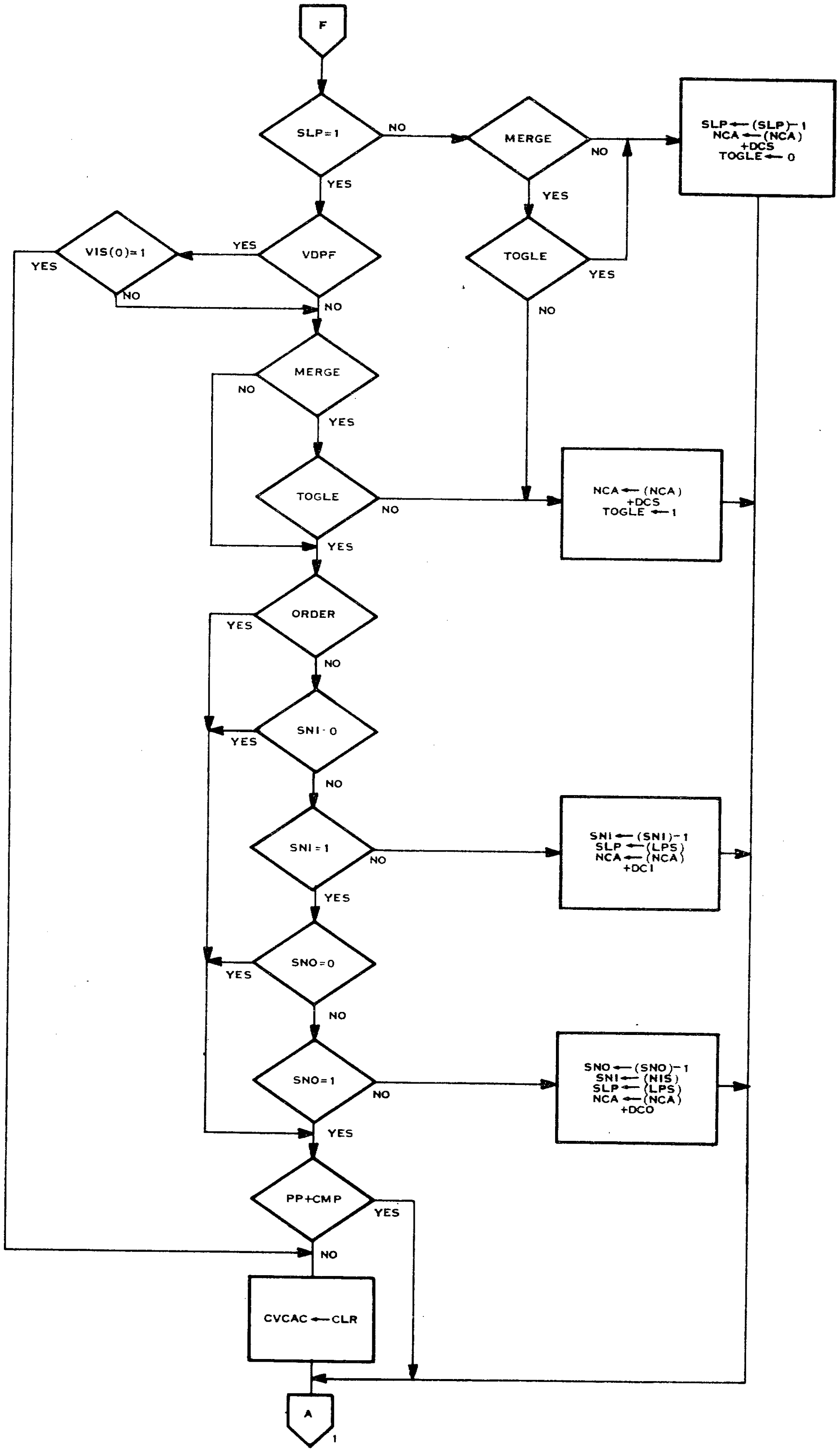
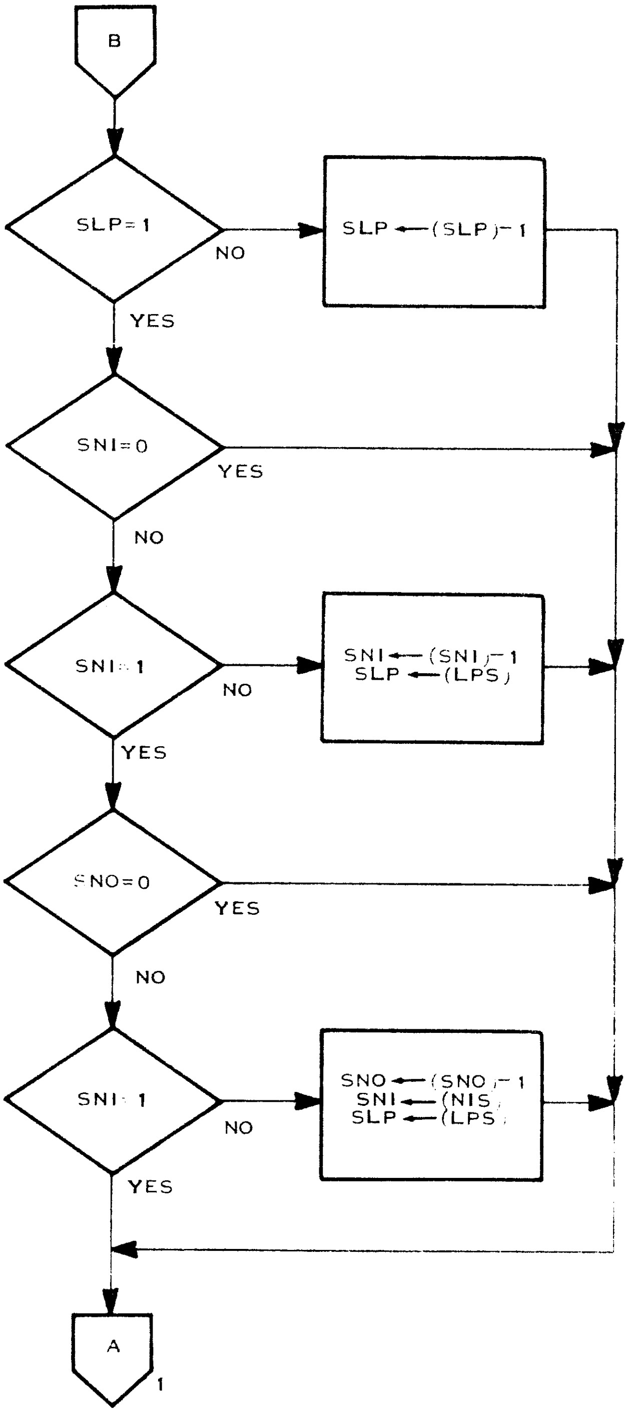


Figure 4-50. C Vector Address Generation Flowchart (Sheet 2 of 6)



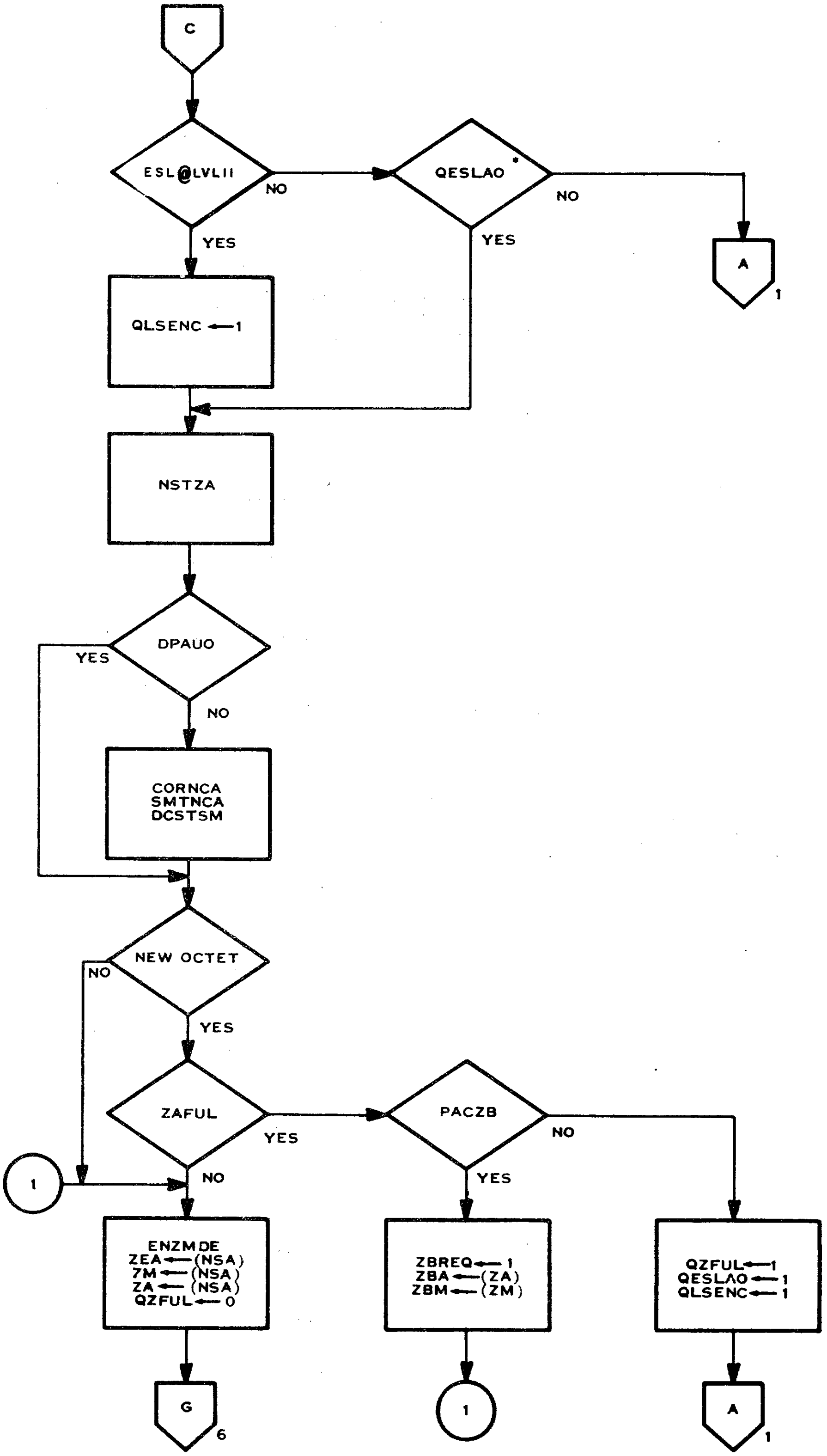
(B) 119913

Figure 4-50. C Vector Address Generation Flowchart (Sheet 3 of 6)



(A) 119914

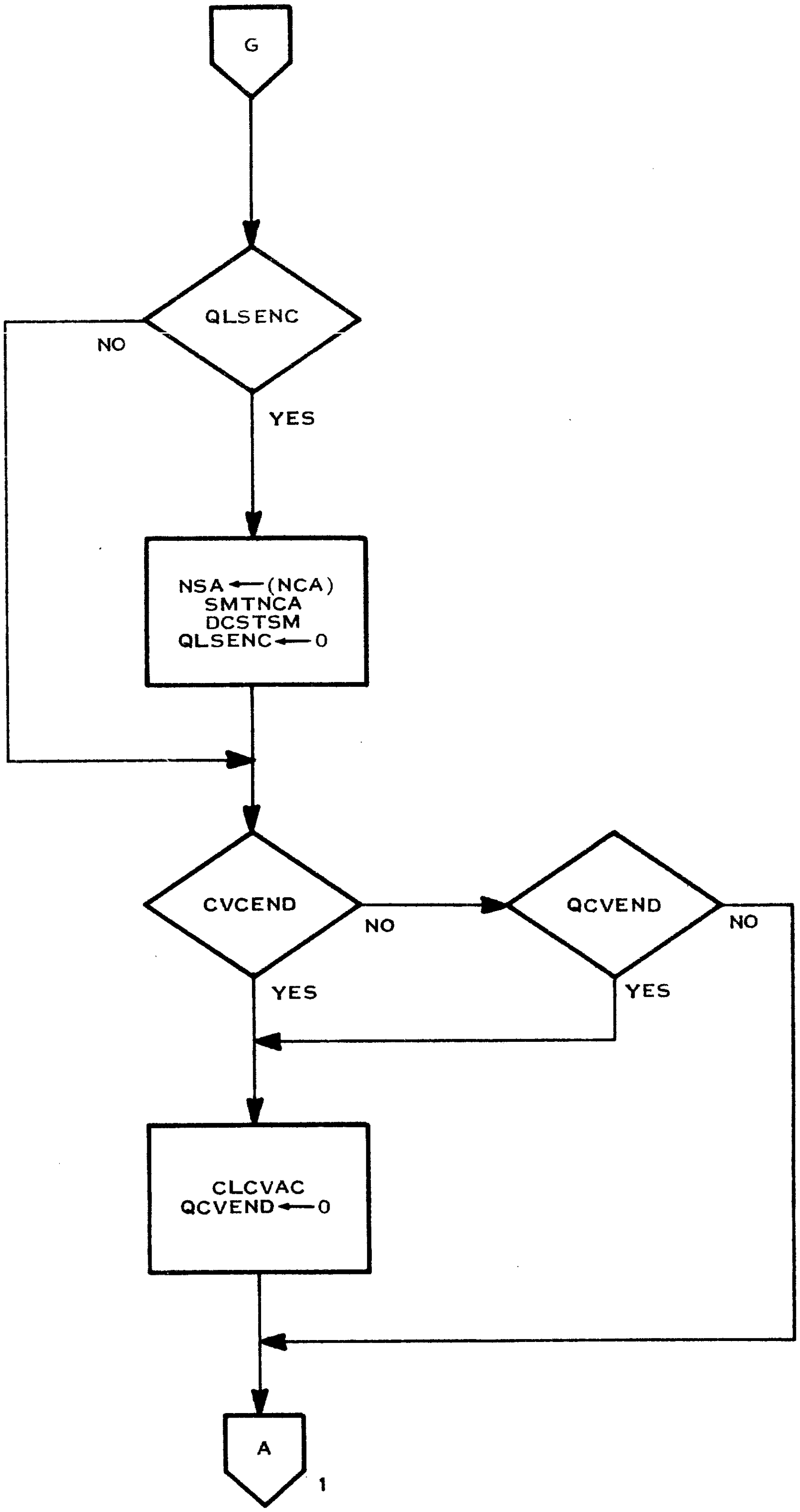
Figure 4-50. C Vector Address Generation Flowchart (Sheet 4 of 6)



* CLEARED BY BCAUTZ

(B) 119915

Figure 4-50. C Vector Address Generation Flowchart (Sheet 5 of 6)



(A) 119916

Figure 4-50. C Vector Address Generation Flowchart (Sheet 6 of 6)

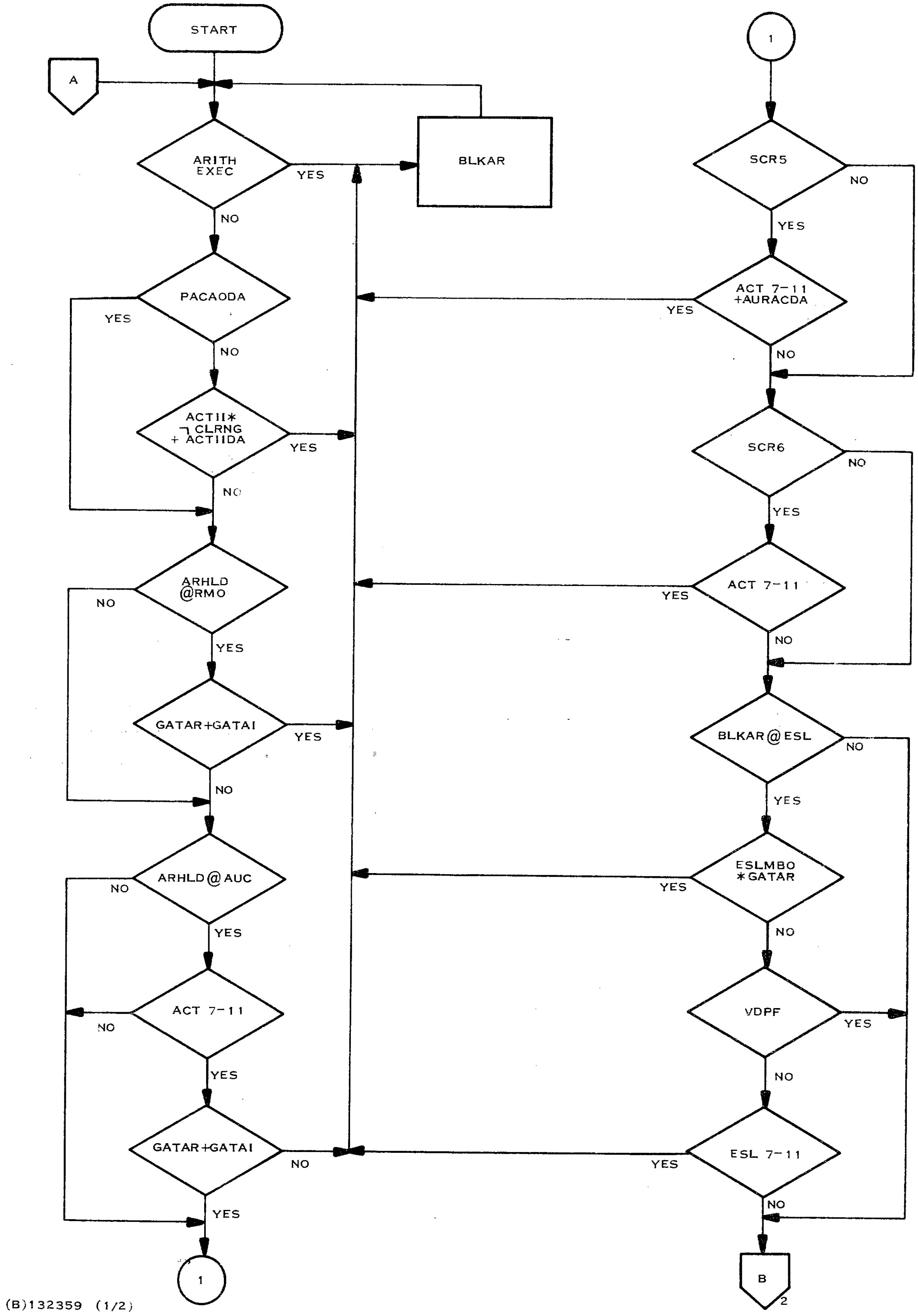
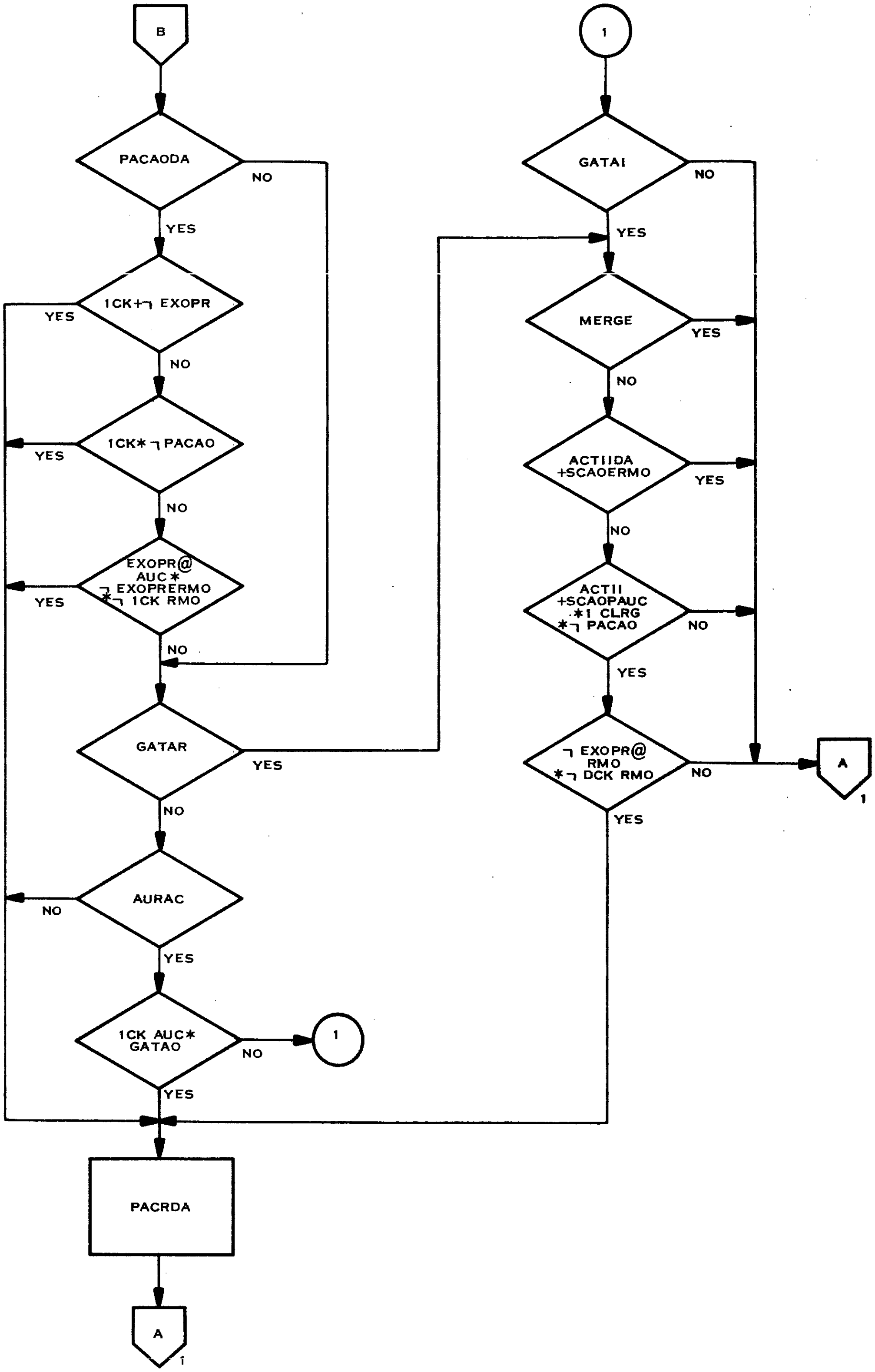
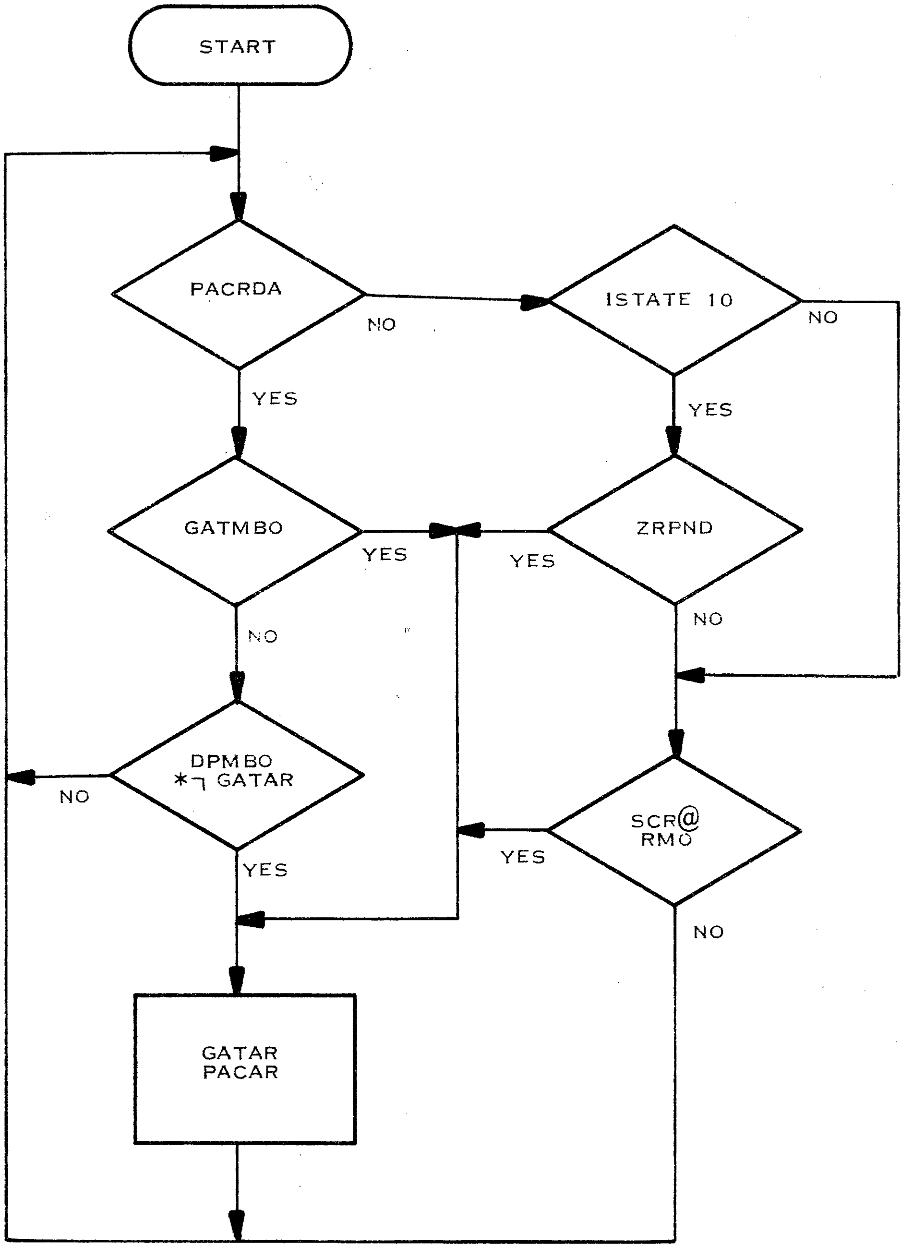


Figure 4-51. AU Control - PACR Flowchart (Sheet 1 of 2)



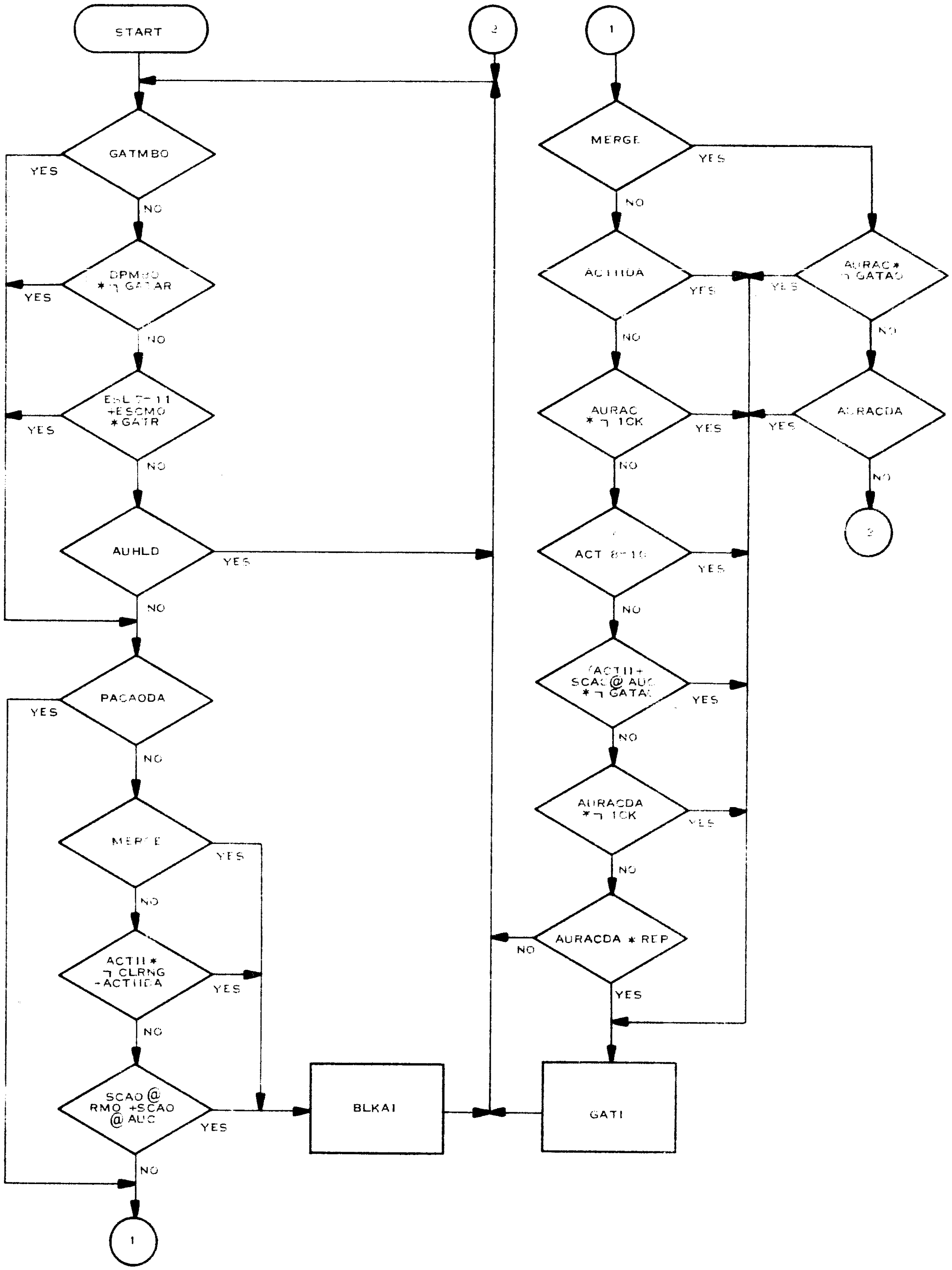
(B)132359 (2/2)

Figure 4-51. AU Control - PACR Flowchart (Sheet 2 of 2)



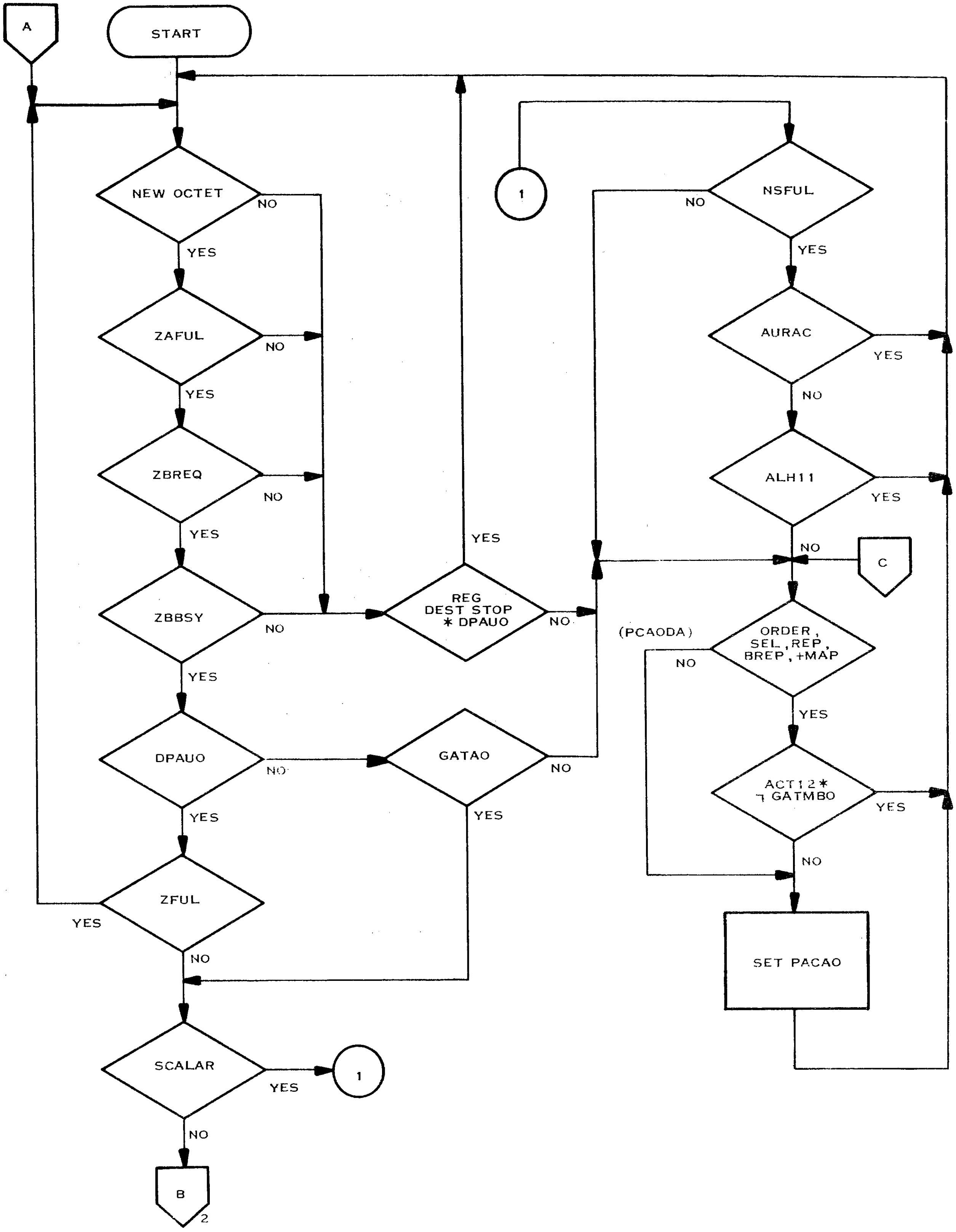
(A)132360

Figure 4-52. AU Control - GATAR Flowchart



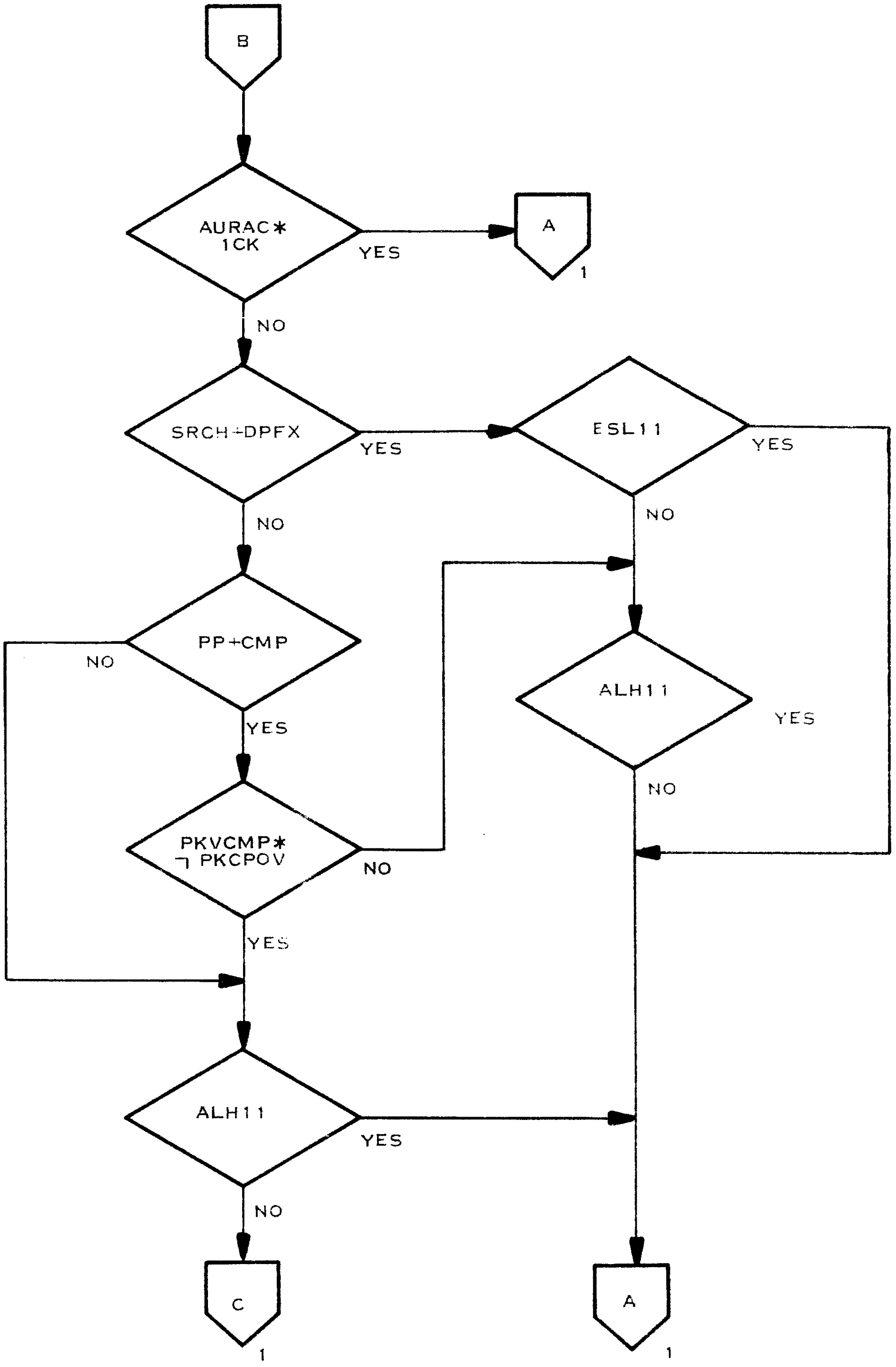
(B)134361

Figure 4-53. AU Control - GATAI Flowchart



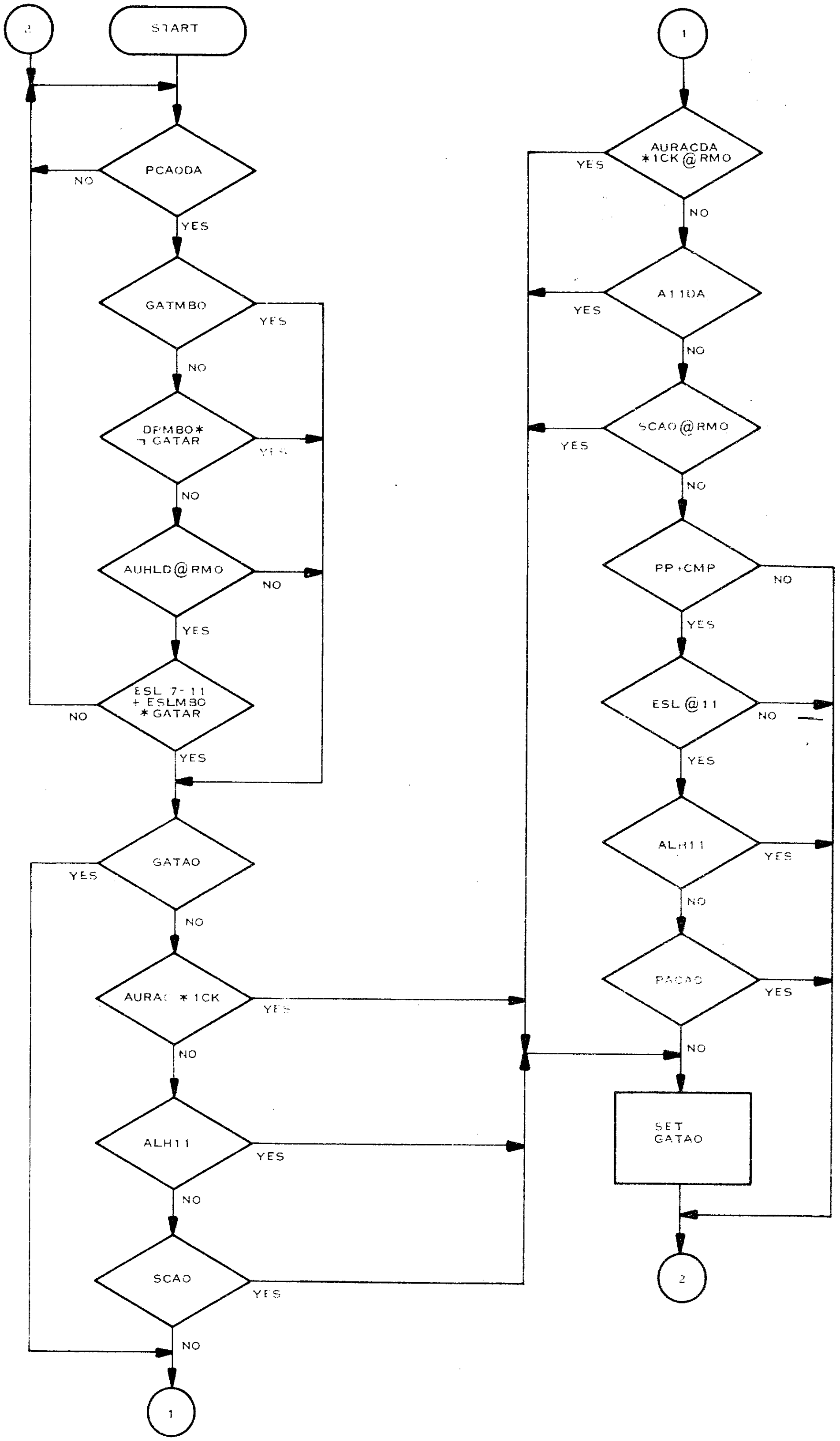
(B)132362 (1 2)

Figure 4-54. AU Control - PCAUO Flowchart (Sheet 1 of 2)



(A)132362 (2 2)

Figure 4-54. AU Control - PCAUO Flowchart (Sheet 2 of 2)



(B)132363

Figure 4-55. AU Control - GATAO Flowchart

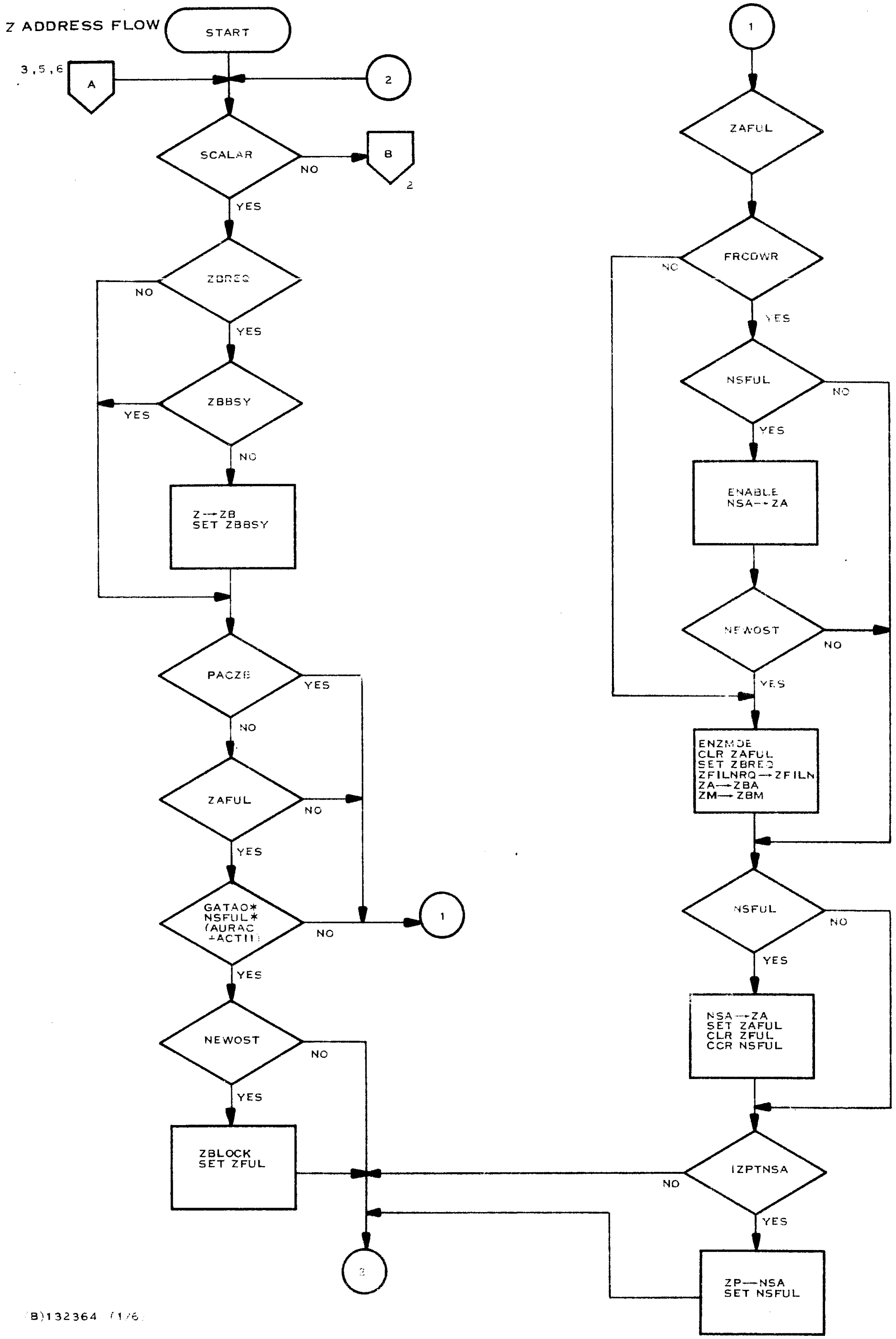
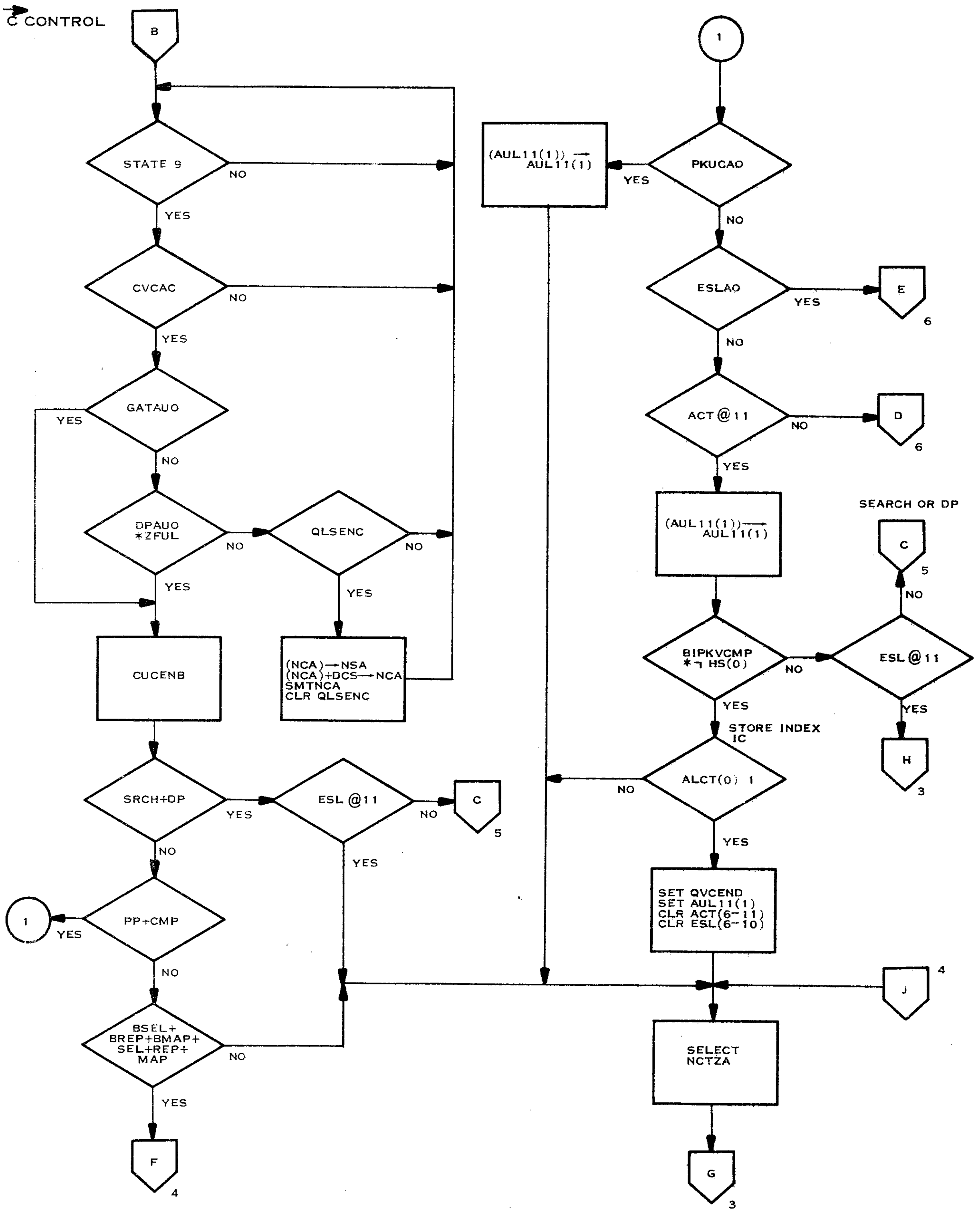
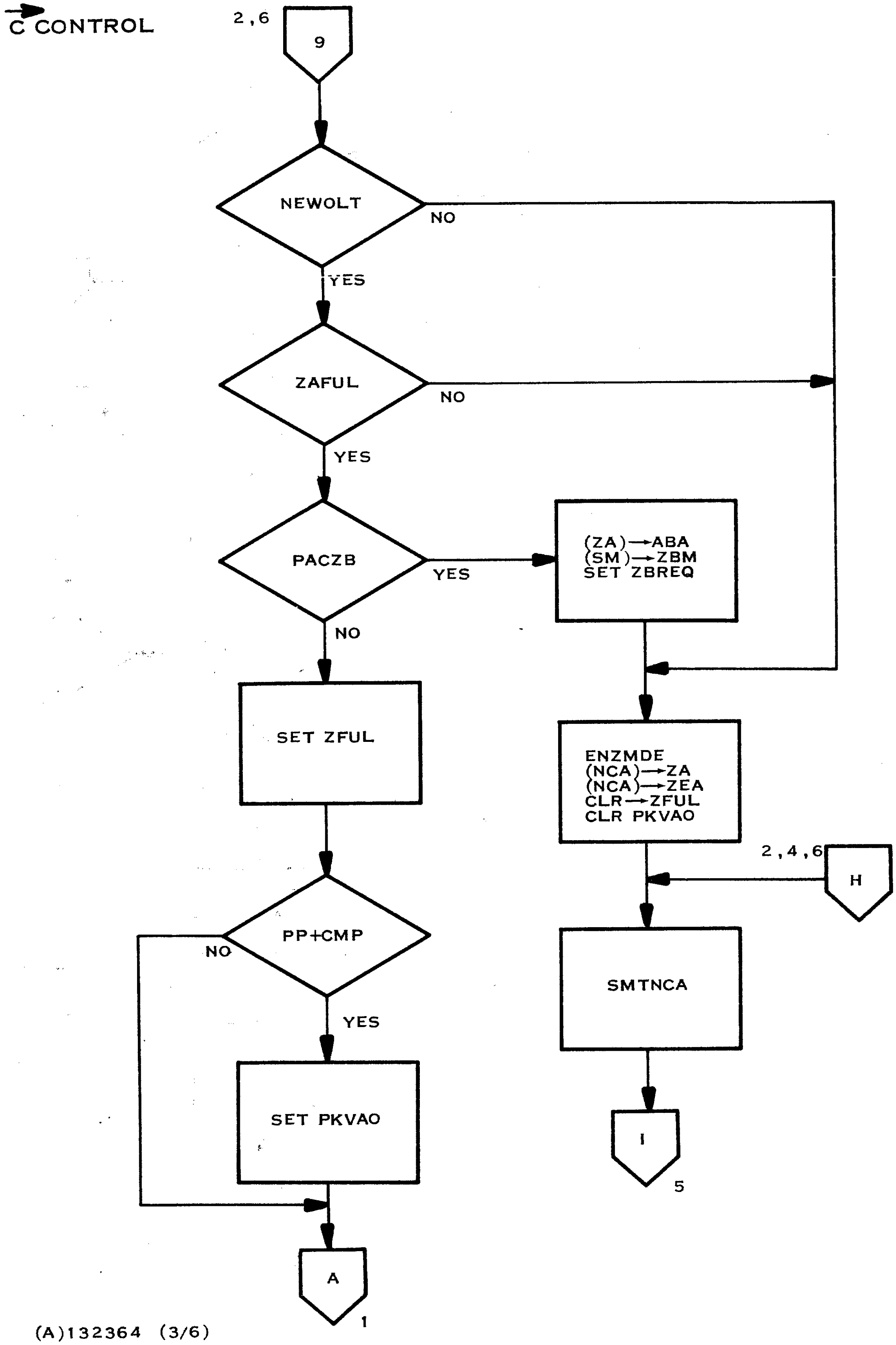


Figure 4-56. Z Address Flow and $\vec{C}$ Control Flowchart (Sheet 1 of 6)



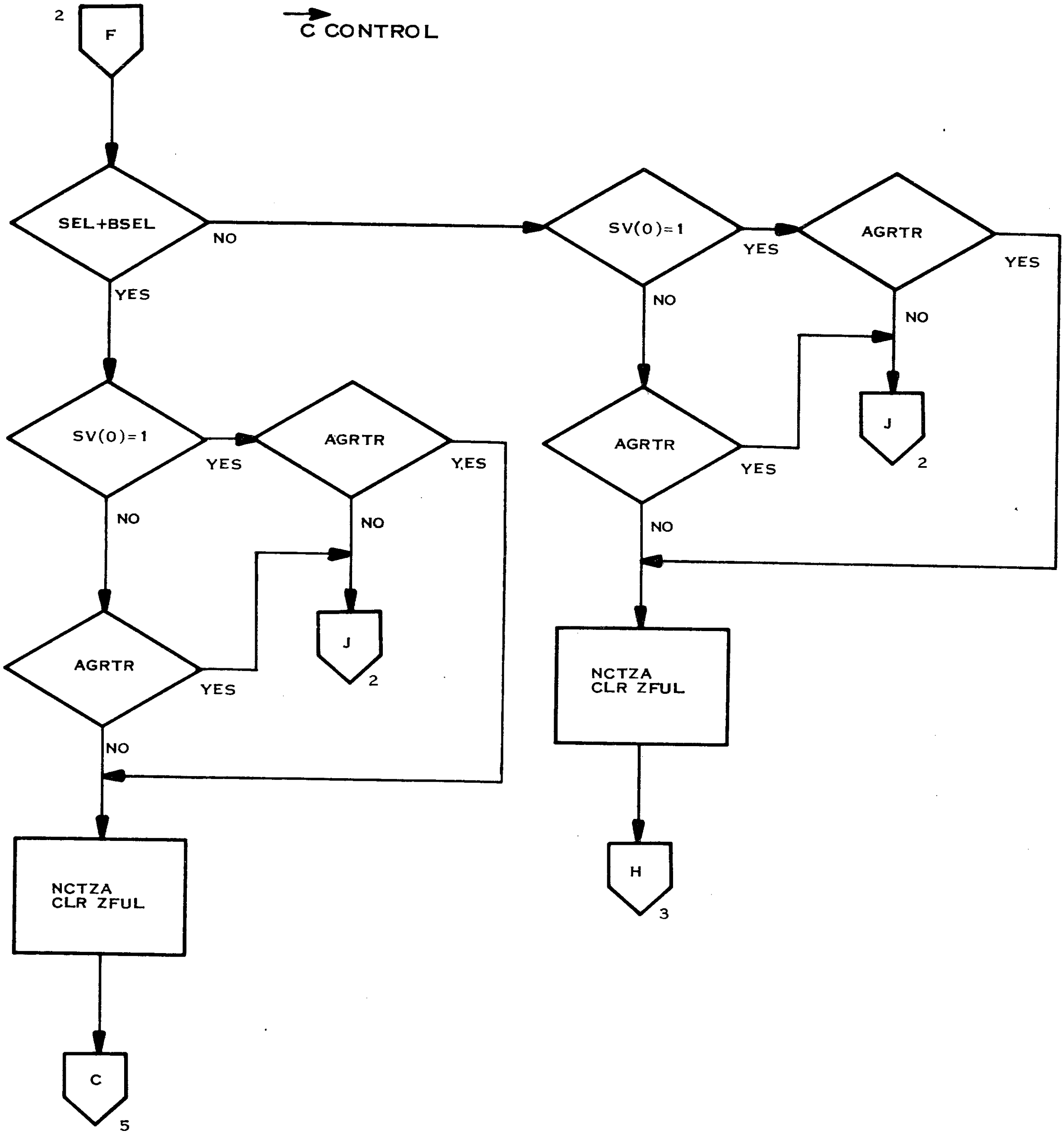
(B)132364 (2/6)

Figure 4-56. Z Address Flow and $\vec{C}$ Control Flowchart (Sheet 2 of 6)



(A)132364 (3/6)

Figure 4-56. Z Address Flow and $\vec{C}$ Control Flowchart (Sheet 3 of 6)



(A)132364 (4/6)

Figure 4-56. Z Address Flow and $\vec{C}$ Control Flowchart (Sheet 4 of 6)

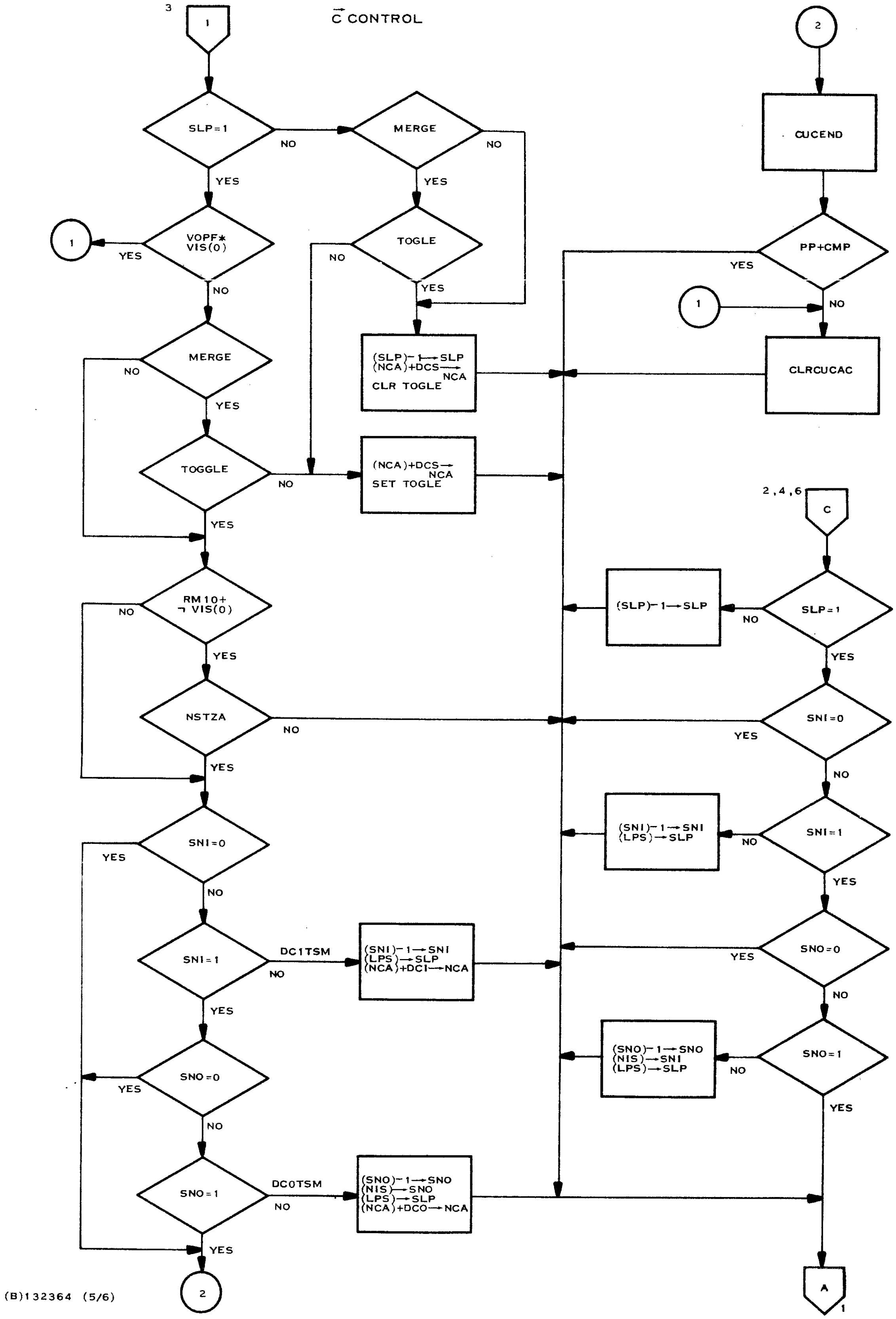
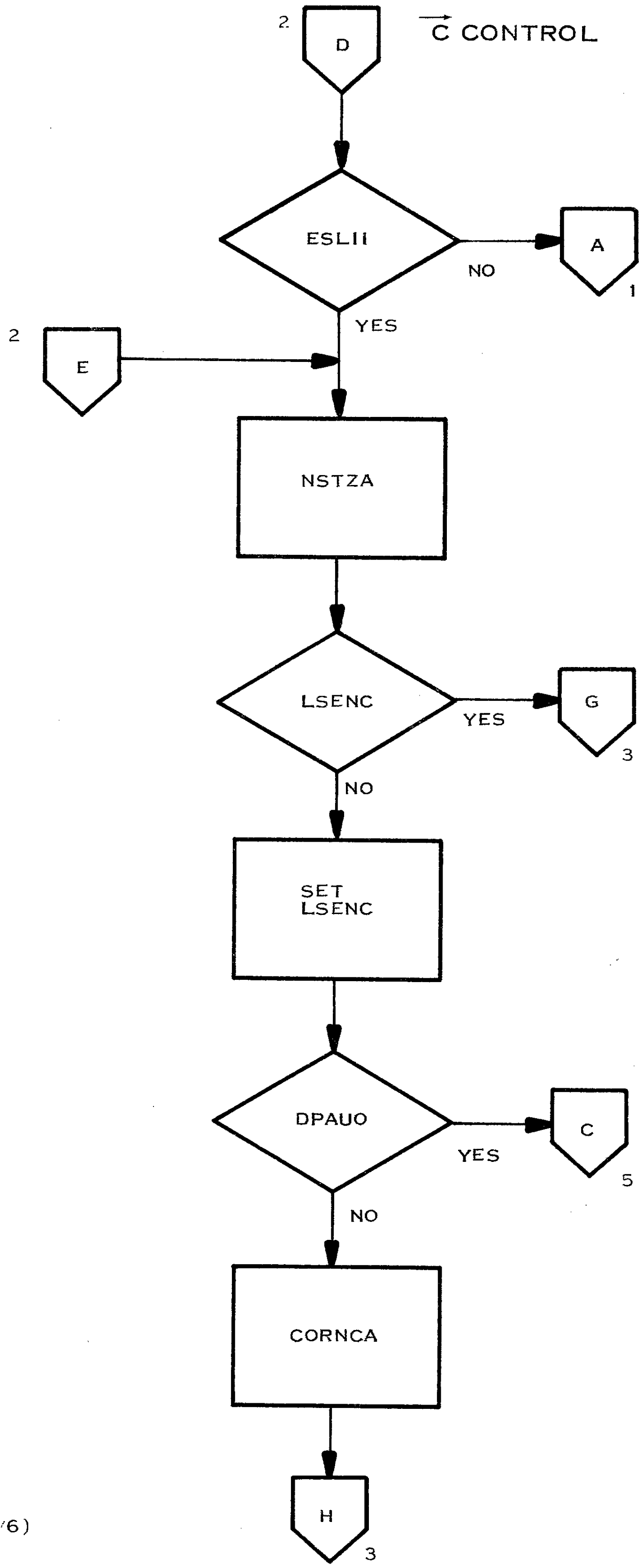


Figure 4-56. Z Address Flow and $\vec{C}$ Control Flowchart (Sheet 5 of 6)



(A)132364 (6/6)

Figure 4-56. Z Address Flow and $\vec{C}$ Control Flowchart (Sheet 6 of 6)

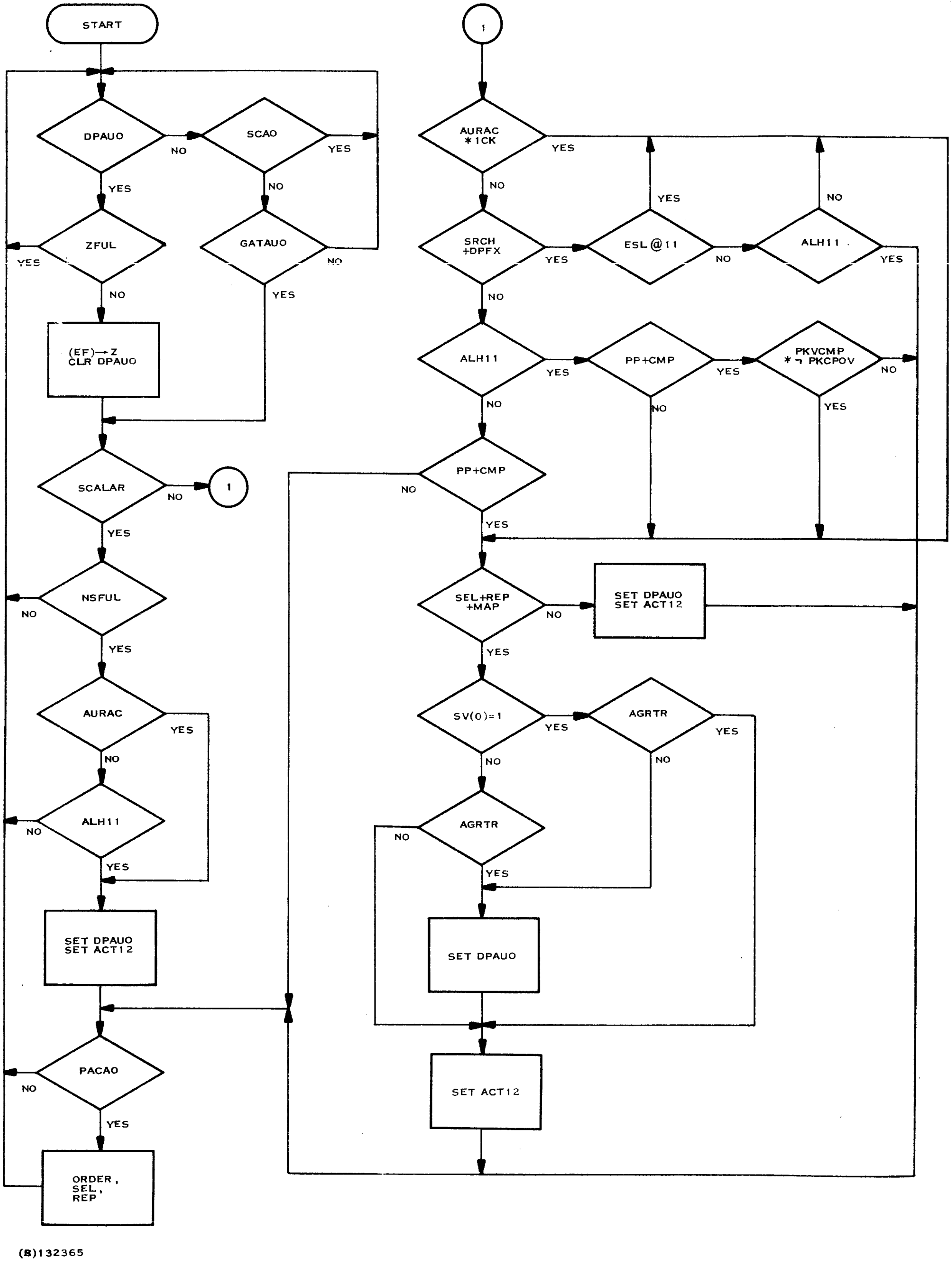


Figure 4-57. Z Data Flow Flowchart

TI-22006