CENTRAL PROCESSOR DESCRIPTION

H632 General Purpose Digital Computer System

Honeywell

COMPUTER CONTROL
DIVISION

Doc. No. 130071963A
M-383

H632
CENTRAL PROCESSOR
DESCRIPTION

November 1968

Honeywell
COMPUTER CONTROL
DIVISION

Printed in U.S.A.

Published by the Publications Department,
Honeywell Inc. Computer Control Division

CONTENTS

	Page
SECTION I INTRODUCTION	
Scope of Manual	1-1
Applicable Documents	1-1
Functional Description	1-2
SECTION II GENERAL DESCRIPTION	
Machine Structure	2-1
Registers and Logic Networks	2-1
Logic Domains	2-6
Control Gating and Data Flow/Cycle Phase	2-8
Sequencing and Control	2-8
Logic Elements	2-8
Normal CP Operating Sequence	2-15
CP Interfaces	2-20
MPC Interface	2-21
Memory Interface	2-21
Control Panel and Power Control Interconnections	2-25
Cable PACs	2-25
SECTION III DETAILED THEORY	
Start-Stop, Timing and Control Logic	3-1
System Initialize	3-1
CP Initialize	3-7
Start-Stop Logic	3-7
Timing Oscillator and Clock Distribution	3-13
Timing Level Generator	3-13
Interrupt Register and Control	3-16
Trap Control Logic	3-18
Memory Interface	3-20
Interface Signals and Cabling	3-23
CP/MAD/IOP/Memory Handshake	3-23
System Initialize and Startup	3-26
Cycle Initiate Logic	3-29
IOP Break-In	3-30
CP Inactive (Cycle Request Removed)	3-33
Panel Halts	3-33

CONTENTS (Cont)

	Page
Look-Ahead	3-34
Principles of Operation	3-34
Detailed Theory of Operation	3-43
Instruction Counter	3-49
Loading RP	3-50
Up-Counter Algorithm	3-50
Cycle Counter	3-52
Structure	3-52
Theory	3-52
Shift Network	3-53
Input Selection	3-53
One-Place Shift Gates	3-55
Four-Place Shift Gates	3-56
Sixteen-Place Shifting	3-56
Unshifted Data Transfers	3-58
Shifting Example	3-58
Adder	3-58
Summands and Summand Selection	3-58
Results	3-59
Method of Implementation	3-60
Skip-Carry Network	3-65
Adder Structure	3-66
Console Control and Display Panels	3-66
Address/Data Entry Switches	3-66
Address Halt Keys	3-73
Sense Keys	3-73
Execute and Maintenance Control Switches	3-73
Display Select Switches	3-73
Direct-Wired Status and Display Labeling	3-73
Lamp Drivers and Display Selection	3-73
Control Panel Power Switching	3-74
MPC Interconnections	3-74

ILLUSTRATIONS

		Page
1-1	Typical Series 32 General Purpose Computer System	1-3
2-1	Central Processor Block Diagram	2-3
2-2	Typical Control Logic Structure: Logic Domains	2-7
2-3	Control Gating Development and Data Flow in Each Logic Cycle Phase	2-9
2-4	Sequencing and Control Logic Block Diagram	2-11
2-5	Simplified Central Processor Timing Signals	2-14
2-6	Central Processor Sequence of Operations	2-17
2-7	Central Processor Interface Connection Summary	2-23
2-8	Cable PAC Applications (2 Parts)	2-26
3-1	Time Sharing of Execution Functions with Sequencing and Control Functions	3-3
3-2	System Initialize and Central Processor Initialize Flow Chart	3-5
3-3	Start-Stop Logic Flow Chart	3-9
3-4	Clock Oscillator Timing	3-14
3-5	Timing Level Generator Signals During Access (Example)	3-15
3-6	Interrupt Control Flow Chart	3-17
3-7	Trap Control Flow Chart	3-19
3-8	Memory Interface Signals	3-21
3-9	Overall Memory Interface Timing -- Read Cycle	3-24
3-10	CP Logic Functions Related to Memory Cycles	3-26
3-11	Memory Interface Logic Signals During System Initialize and Startup	3-27
3-12	Memory Interface Logic Signals (Write Cycle)	3-30
3-13	Memory Interface Logic Signals During IOP Break-In	3-31
3-14	Memory Interface Logic Signals During Central Processor Inactive Condition	3-35
3-15	Memory Interface Logic Signals During Panel Halt	3-37
3-16	Instruction Processing Cycle Alternation	3-39
3-17	Cycle Entry Decisions	3-41
3-18	Memory Address Switching Principles	3-44
3-19	Timing for TIC, TSP and TIP Look-Ahead Controls	3-46
3-20	Memory Address Switching Details	3-47
3-21	Instruction Counter Operation	3-51
3-22	Shift Network Block Diagram	3-54
3-23	One-Place Shift Gates (Typical)	3-55
3-24	Four-Place Shift Gates (Typical)	3-56
3-25	Sixteen-Place Shift Patterns	3-57
3-26	Shifting Example: 21-Place Right or 43-Place Left Rotation	3-58
3-27	Typical Adder Stage Block Diagram	3-61
3-28	Final Carry Signals Block Diagram	3-63
3-29	Adder Structure	3-67
3-30	Console Data Distribution	3-69

TABLES

		Page
1-1	Standard H632 Documentation	1-2
2-1	CP Start/Stop Condition Summary	2-21
3-1	Trap Condition Table Entries	3-20
3-2	Adder Inputs and Definitions	3-59
3-3	Truth Table for SSP_N and SCP_N	3-61
3-4	Truth Table for SPR_N, $SP0_N$, $SP1_N$ and $SP2_N$	3-62
3-5	Truth Table for Stage 25 Type - Final Carry Generation Where Carry = $(SCY_N) \vee (SP2_N)$	3-63
3-6	Truth Table for Stage 27 Type Carry Generation Where Carry - $(SCY_N) \wedge (\sim SP0_N)$	3-63
3-7	Truth Table for Stage 28 Type Final Sum Generation	3-64
3-8	Truth Table for Stage 27 Type Final Sum Generation	3-64
3-9	Truth Table for Stage 26 Type Final Sum Generation	3-65
3-10	Truth Table for Stage 25 Type Final Sum Generation	3-65
3-11	Display Lamp Driver Input Summary	3-71

SECTION I
INTRODUCTION

SCOPE OF MANUAL

Technical information on the H632 Central Processor appears in this manual and in the following two companion volumes:

a. H632 Central Processor Instructions (Flow charts and analyses of all instructions and control algorithm, and mnemonic glossary).

b. H632 Central Processor Diagrams (Logic and power distribution diagrams, PAC layout drawings, and mechanical coding diagrams).

This manual emphasizes the hardware implementation of H632 Central Processor functions described from a general operational viewpoint in the H632 Reference Manual. The main CP logic elements are discussed on a block-diagram basis, and unusual or complex logic structures, such as the adder, are described in detail. While all the H632 CP data-processing instructions and algorithms are analyzed by flow charts and analysis tables in the Central Processor Instructions manual, this manual provides additional detail on the sequencing and control functions that operate in addition to, and simultaneously with, the actual data-processing algorithms. These functions include power turn-on; system and CP initializing; panel start-stop and control of operating mode; interface synchronization of the CP, Memory Access Director (MAD), Multiprocess Controller (MPC), I/O Processor (IOP), and the memory system, and instruction overlap.

APPLICABLE DOCUMENTS

All manuals provided as standard documentation with each Series 32 Digital Computer System are listed in Table 1-1. Additional copies of any manual in Table 1-1 may be obtained by contacting a local Honeywell Inc., Computer Control Division representative or by writing directly to:

Honeywell Inc.
Computer Control Division
Old Connecticut Path
Framingham, Massachusetts 01701

Table 1-1.
Standard H632 Documentation

Title	Doc. No.
System Manuals	
H632 Reference Manual	130071960
H632 Operator's Manual	130071967
H632 Modular Products Manual Vol. I	130071974
H632 Modular Products Manual Vol. II	130072932
System Component Manuals	
H632 Central Processor Instructions	130071962
H632 Central Processor Diagrams	130071968
H632 Memory Access Director	130072037
H632 Multiprocess Controller	130072042

FUNCTIONAL DESCRIPTION (Figure 1-1)

The Central Processor (CP) is a 32-bit general-purpose, stored-program digital computer. It performs all of the Series 32 system computation (arithmetic and logic) and data-processing functions, exclusive of input/output transfers (direct CP input/output capability is optional).

The CP has the standard capability of processing eight levels of programming, each level having an assigned priority. The MPC specifies which level the CP is to be acting upon. The CP is equipped (as is the IOP) with a control process order which it delivers to the MPC. The control process order requests the MPC to affect a change of state in the CP process which issued the order and/or another CP or IOP process. Thus, the CP has the capability of affecting the activity state of any of the system processes.

Transfers between memory and input/output devices are, as a rule, performed by the IOP into and out of a processor port shared by the CP and IOP. Thus, the CP can be devoted to purely computation tasks.

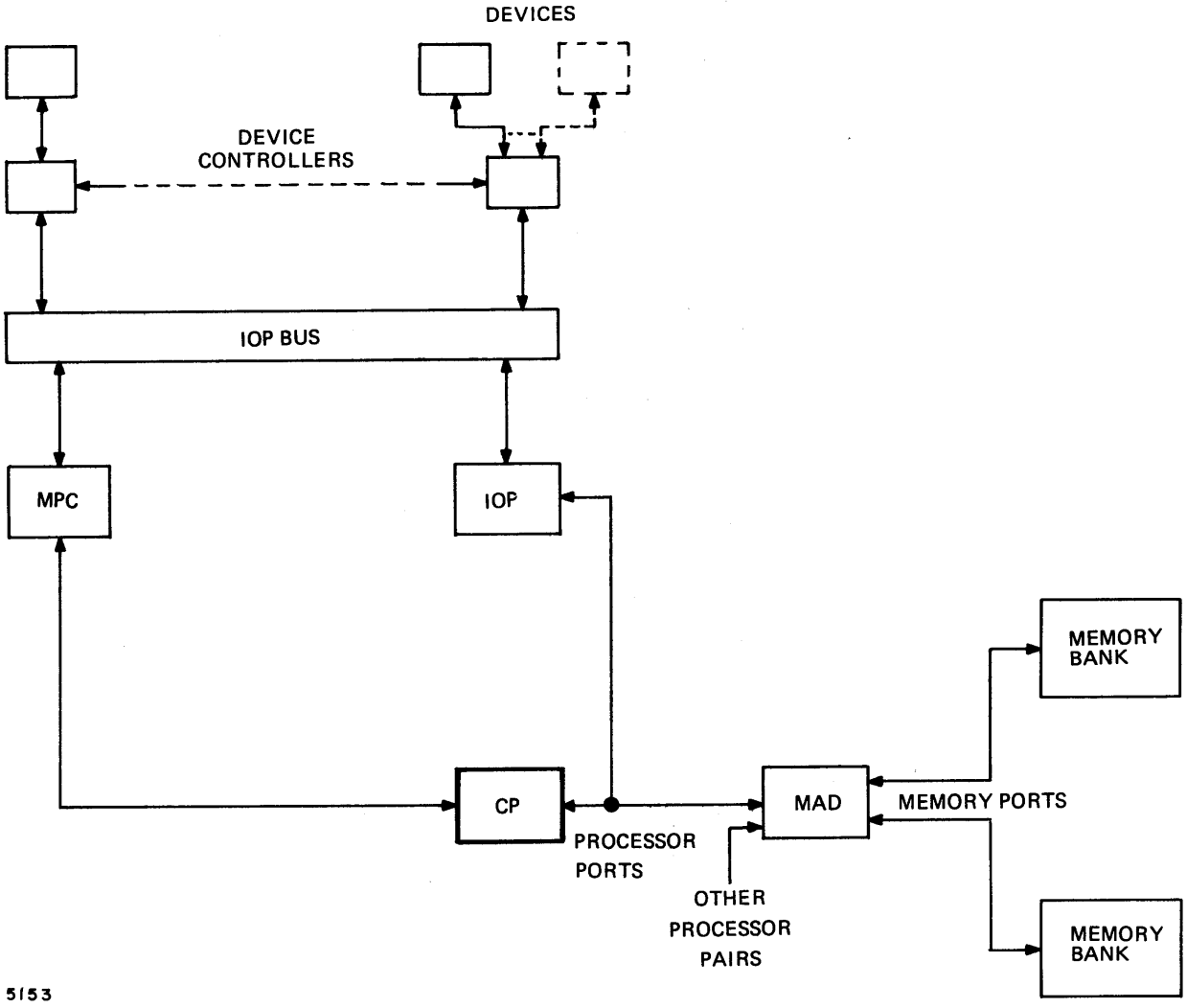


Figure 1-1. Typical Series 32 General Purpose Computer System

SECTION II
GENERAL DESCRIPTION

MACHINE STRUCTURE

Registers and Logic Networks (See Figure 2-1.)

Addressable Registers. -- There are 16 32-bit general registers, designated R0 through RF (second character is hexadecimal). The general registers can be accessed by the instruction word address field when the effective address is equal to or less than F_{16}, or by the instruction word R-field. The general registers can be used as operand and result locations. Registers R1 through R7 can also be specified by the instruction word X-field and serve as index registers.

Inputs to the general registers are from the adder or the RG or RT register. The contents of the general registers are gated out to the U and V shift network.

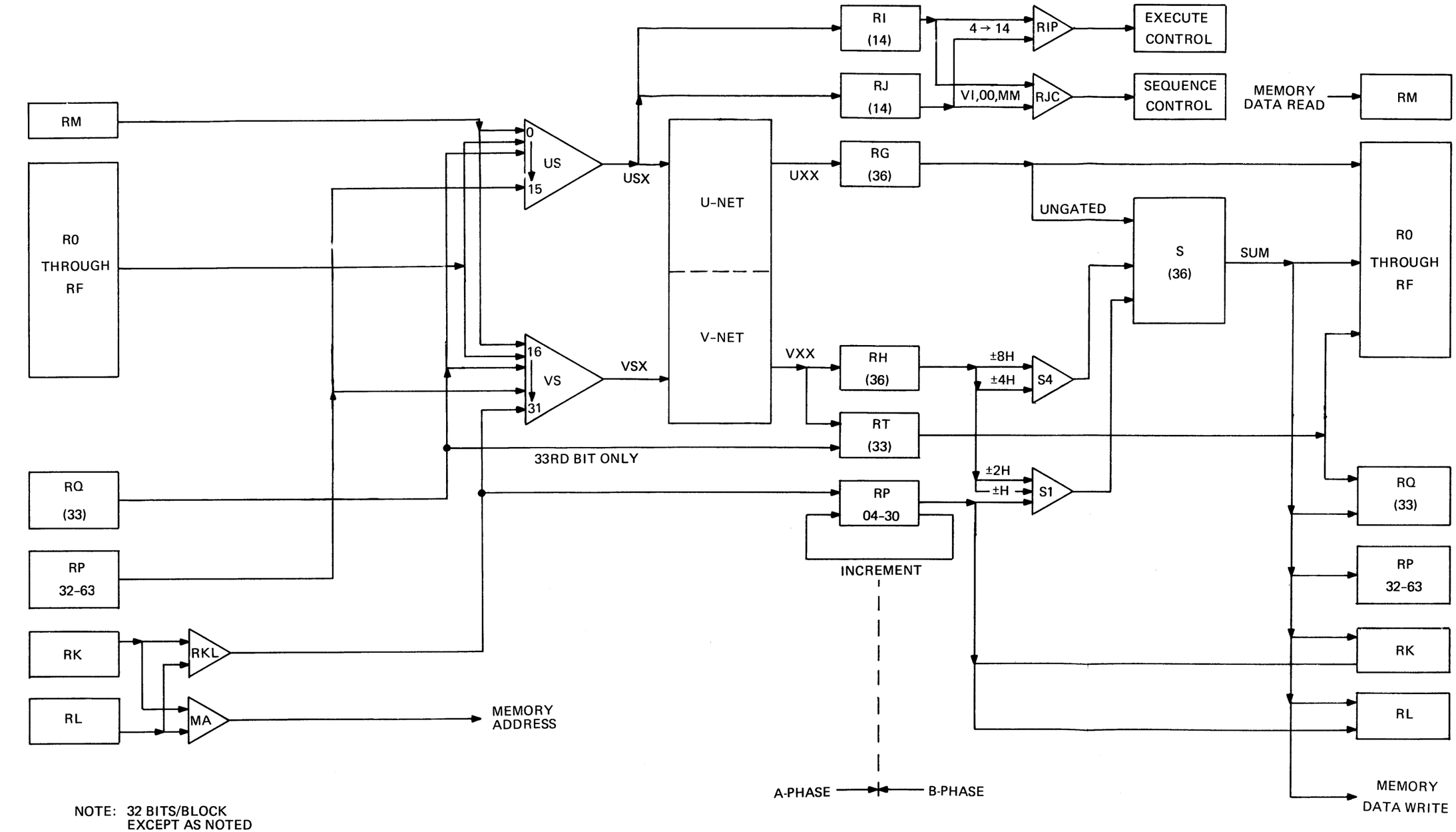
Auxiliary Registers. -- The auxiliary registers supply operands to the logic networks (the U and V shift network and the adder) and are used to retain partial results.

Register RG is a 36-bit register. Some of its functions are as follows. It retains augend, minuend, product, or dividend/remainder in arithmetic operations. It provides address storage during control panel operations and buffer storage during the direct read operation (optional). The main input path to RG is via the U and V shift network. The RG register output is hardwired (ungated) to the adder. The contents of RG can be gated to any one of the general registers.

Register RH is a 36-bit register. Some of its functions are as follows. It retains addend, subtrahend, multiplicand, or divisor in arithmetic operations. It also provides data storage during control panel operation. The input path to RH is via the U and V shift network. The contents of RH can be gated only to the adder.

Register RQ is a 33-bit register. RQ holds the multiplier or divisor during multiplication or division. RQ is used at various times to inject zeros into the U and V shift network. The input path to RQ is from the RT register. The contents of RQ can be gated to the U and V shift network.

Register RT is a 33-bit register. Generally, RT serves as temporary storage for intermediate results. It also is used as a buffer during the optional direct write operation. The input path to RT is via the shift network. The contents of RT can be gated to any general register or the RQ register.



NOTE: 32 BITS/BLOCK
EXCEPT AS NOTED

5152

Figure 2-1. Central Processor Block Diagram

Memory Interface Registers. -- There are three memory interface registers: RM, RK, and RL. Register RM is a 32-bit register for holding data received from memory. The input to RM is from memory. The contents of RM are gated to the U and V shift network.

The RK and RL registers are 32 bits each. The RK register holds the addresses for the I-sequence, and RL holds the addresses for the J-sequence (the I and J sequences are explained in the paragraph on Look-Ahead). The input path to either RK or RL is via the $RP_{04\rightarrow30}$ register or the adder. The contents of either RK or RL are gated to the memory address lines and to the V shift network.

Instruction Registers. -- There are two 14-bit instruction registers, RI and RJ, belonging to the I and J sequences, respectively. Bits 4-7 hold the instruction R-field, which is used to specify one of the general registers. Bits 8 through 14 hold the 2-digit hexadecimal operation code. Bit 0 holds the flag indicator. Bit VI holds the validity indicator. Bit MM is a specialized indicator set by a certain group of instructions. Bits 0, VI, and MM are used as look-ahead controls and are explained in the paragraph on Look-Ahead. The input path to RI or RJ is via the input level of the U and V shift network. The contents of RI or RJ are delivered to various control areas under control of the look-ahead logic.

Program Status Registers. -- There are two program status registers which dynamically retain the status of the program currently being processed by the CP. These registers are designated $RP_{04\rightarrow30}$ and $RP_{32\rightarrow63}$. The $RP_{04\rightarrow30}$ register is a double-rank register containing the memory address of the next instruction. The $RP_{32\rightarrow63}$ register retains the following data:

a. Bits 32 and 33 - Condition code, an indication of the result of a comparison operation.

b. Bit 35 - Floating-point normalize mask. When set, inhibits normalization of addition, subtraction, and difference results.

c. Bit 36 - Floating-point underflow mask. When set, enables entry to trap algorithm upon sensing underflow condition during floating-point operations.

d. Bit 37 - Floating-point overflow mask. When set, enables entry to trap algorithm upon sensing overflow condition during floating-point operations.

e. Bit 38 - Fixed-point halfword overflow mask. When set, enables entry to trap algorithm upon sensing a halfword overflow condition.

f. Bit 39 - Fixed-point word overflow mask. When set, enables entry to trap algorithm upon sensing a word overflow condition.

g. Bits 42-45 - Sense switch flip-flops. Used to suppress sense switch contact bounce.

Logic Networks. -- The U and V shift network and the adder are the two logic networks for performing arithmetic, shift, and logic operations on the CP data. Most data transfers from one register to another are via the shift network or adder.

The U and V shift network is 64 bits long and is made up of several layers, or levels. Each level is capable of performing a shift of a certain number of places. The shift network always shifts right and is capable of rotating a doubleword operand 0 through 63 places. The high-order 32 bits of the shift network are designated U, and the low-order 32 bits are designated V.

The adder has 36 stages numbered in order of significance as follows: 60 through 63 and 00 through 31. The adder has a skip-carry or carry-anticipation feature. The RG register is hardwired to the adder. The other data paths to the adder are designated S1 and S4. The adder performs addition and subtraction. The exclusive-OR function is performed by suppressing adder carries.

NOTE

The inclusive-OR and AND logical functions are performed by algorithm manipulations which are illustrated in the algorithm flow charts.

Logic Domains

Figure 2-2 shows the structure of the basic CP control logic domains and their relationship to each other. Note that the letter assigned to each domain is the first character of all mnemonics in that domain.

The K-domain consists of variable gating structures used to develop the various keying conditions (such as overflow) required by the A-domain.

The A-domain consists of AND gates used for developing control minterms. The inputs to the A-domain are primarily from the decoders (operation code, R-field, X-field), the timing level generator logic, the K-domain, and the R-domain. Basically, an A-domain gate is selected by an operation code and a specific time (or cycle phase).

The C-domain develops the actual control signals used to gate data into the R-domain registers. Generally, the C-domain structure is as follows. The first level of C-domain gates function as OR gates receiving inputs from the A-domain. The OR-gate output enables a particular AND gate which has a timing requirement; i.e., a set or clear pulse of phase A or B. If the AND gate has a clearing function, its output is sent directly to the R-domain. If the AND gate has a setting function, it is inverted to an assertion signal and then sent to the R-domain.

The R-domain consists of register stages and input gates. Generally, data is ANDed with a C-domain control to set a register stage.

NOTE

The flow charts in the H632 Central Processor Instruction Manual illustrate events in the first level of C-domain gates (OR gates). The flow chart analysis tables specify which A-domain signal and function (gate) enabled the C-domain OR gate.

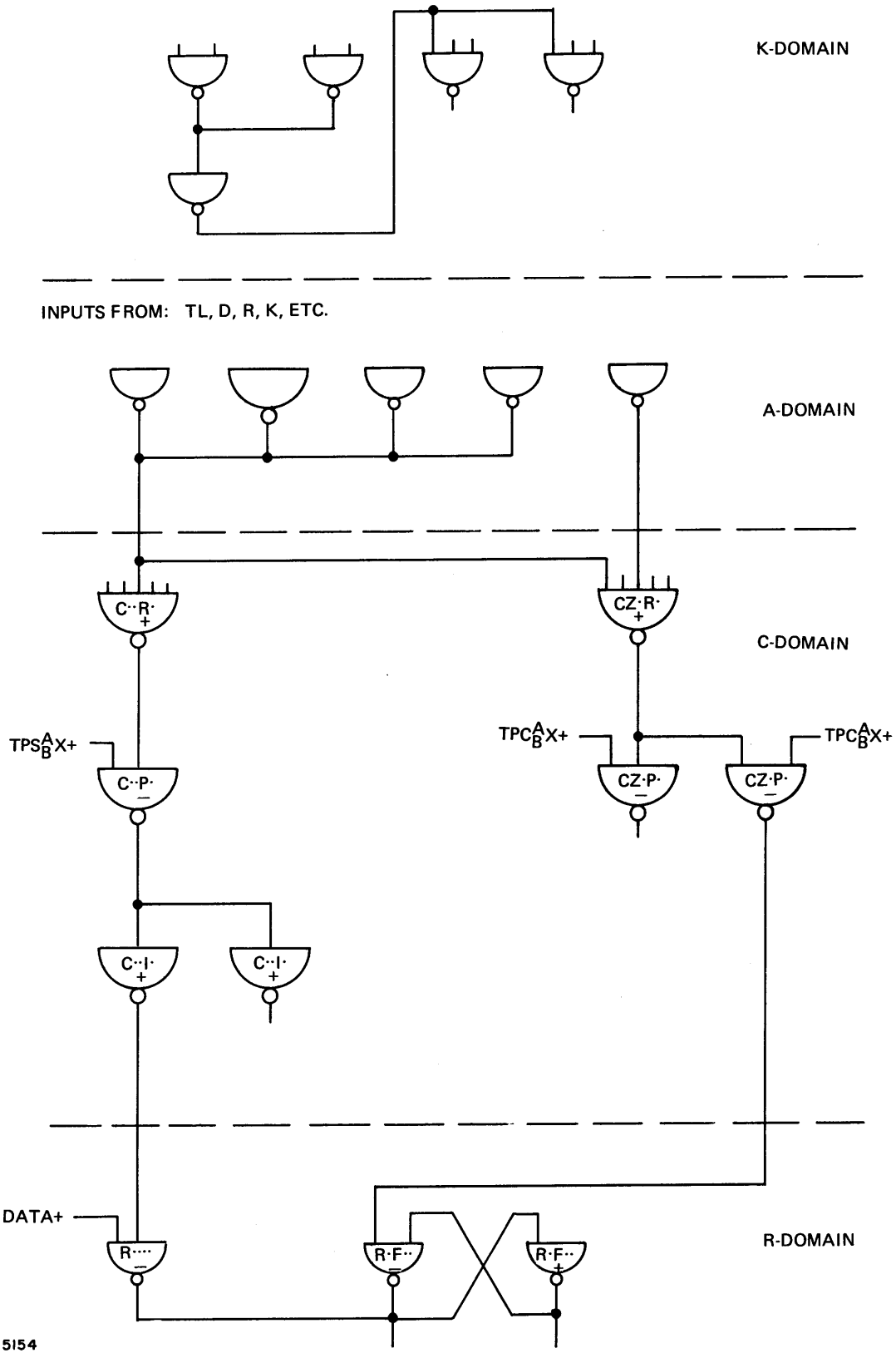


Figure 2-2. Typical Control Logic Structure: Logic Domains

Control Gating and Data Flow/Cycle Phase

Figures 2-1 and 2-3 show the data flow during the two phases of each logic cycle. During phase A, data is transferred through (and possibly manipulated by) the U and V shift network and then gated into one of the auxiliary registers. If the cycle is a fetch cycle, the instruction is transferred into the proper instruction register at this time.

In phase B, the contents of the appropriate auxiliary register (or registers) are gated into the adder, and the adder output, or result, is gated into an appropriate register or presented to the memory data lines.

SEQUENCING AND CONTROL

Logic Elements

The logic domains described previously perform the data transfer operations that execute all CP algorithms. In addition, sequencing and control logic provides the timing signals and conditional control signals that determine the CP operating sequence from algorithm to algorithm. Figure 2-4 is a simplified block diagram showing the sequencing and control logic structures and the main logic signals used for communication between them.

Panel Functions. -- The panel functions control initializing, startup, panel-directed halts, and the algorithms associated with the panel (access, fill, transfer).

IAF Register and Interrupt Control. -- The interrupt (IAF) register stores the current process level code and compares it to the code on the MPCs IDL lines. When the codes differ, an interrupt algorithm is set up. During the interrupt, the new code is transferred from IDL to IAF. The code in the IAF register, both before and after updating, is applied to RKL, to specify dedicated program stateword locations. When the interrupt is to an inactive process level, the memory request is cleared, and the CP enters the inactive condition. When the MPC supplies a new, valid code, the memory request is set; another interrupt restores active operation.

Memory Interface Control. -- When the CP memory request is set and MAD is granting the CP the next memory cycle, MAD responds with address select, and the CP generates cycle initiate, starting the memory cycle. Memory busy synchronizes events in MAD. During consecutive memory cycles, the CP waits for the trailing edge of read data select before enabling the next set of CP clock pulses. The A-phase clock pulse triggers cycle initiate for the next memory cycle.

When an interrupt brings the inactive process level code into the IAF register, the memory request is cleared. Since MAD does not return an address select, the CP cannot generate cycle initiate. MAD also withholds read data select, so the CP clock is not started. This suspension of operation is the "CP inactive" condition. In this state the CP requests memory cycles only when a fetch or store operation is initiated at the panel.

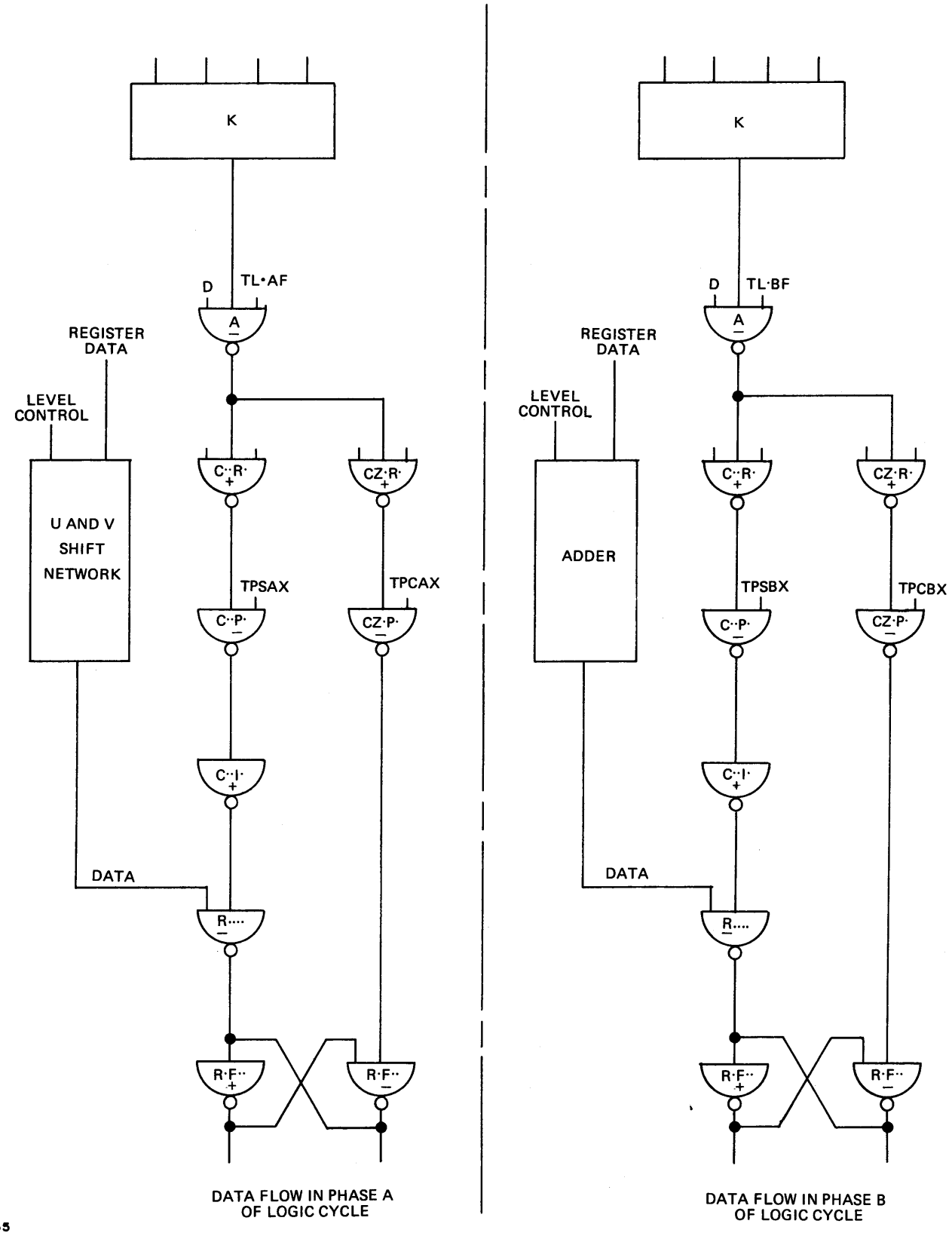


Figure 2-3. Control Gating Development and Data Flow in Each Logic Cycle Phase

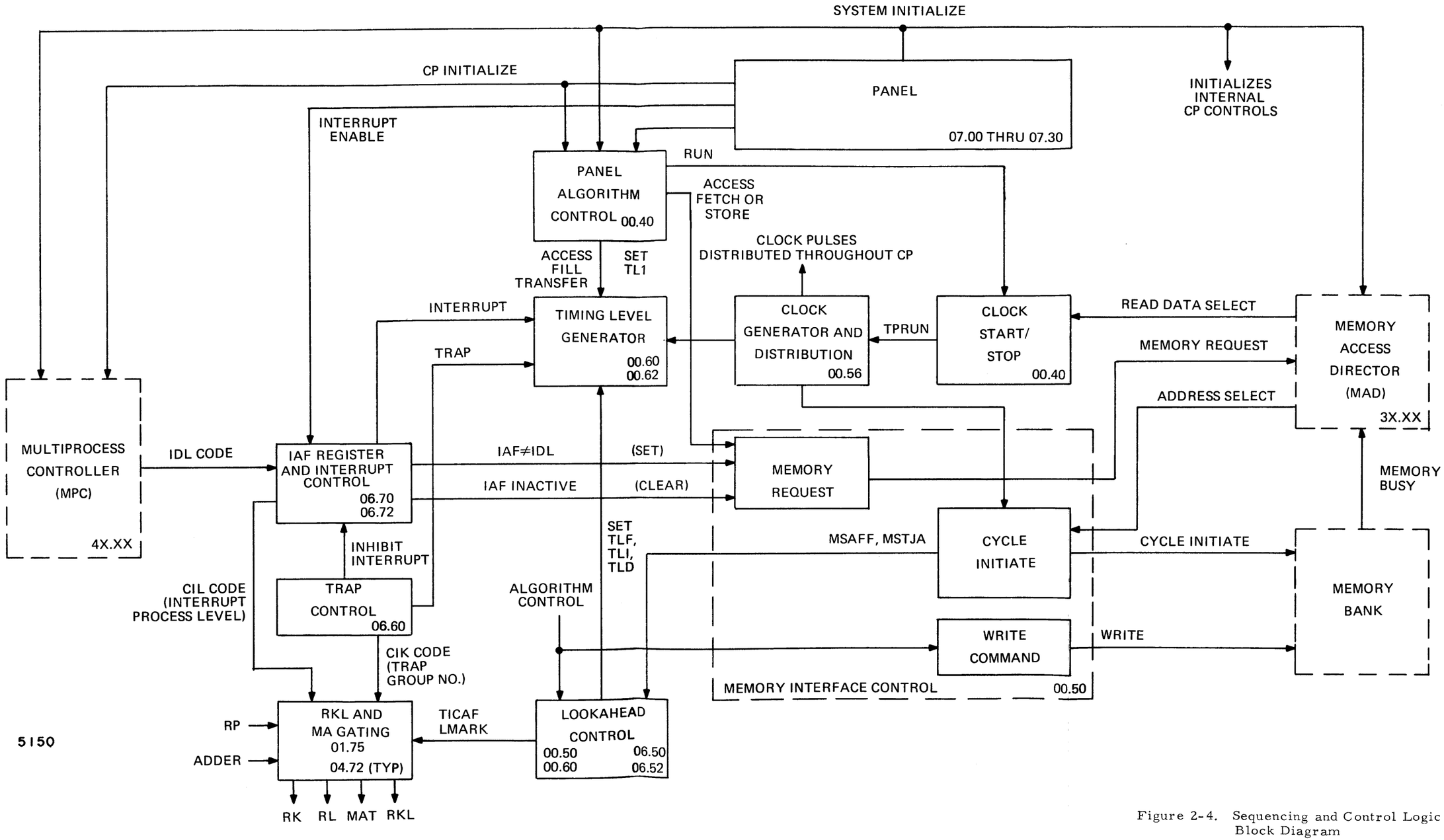


Figure 2-4. Sequencing and Control Logic Block Diagram

When MAD is giving memory cycles to the other processor, the CP does not receive address select, and so cannot generate cycle initiate. These brief suspensions of operation (one to five consecutive cycles) are the "memory wait" condition.

Clock and Timing Level Generator. -- The basic system timing signals are illustrated in Figure 2-5. After each memory cycle, a set of timing pulses (TPCA, TPSA, etc.) is produced by the timing oscillator. The timing pulses gate signals to and from the CP registers and also control the timing level generator.

During each set of timing pulses, one timing level is in effect in the timing level generator. Timing levels are generated in A and B pairs: TLFA-TLFB, TL1A-TL1B, etc. The timing levels enable data gating paths in the A-domain according to the requirements of the algorithm in effect. The timing pulses are applied in the C-domain to actually gate the data to the destination. Phase A timing pulses (TPCA, TPSA) gate data through paths enabled by phase A levels (TLFA, TL1A). Similarly, phase B pulses gate data transfers enabled by phase B levels. In general, TPS pulses perform register setting actions, while the shorter TPC pulses are used for clearing. During each TPSA pulse, the next B-phase timing level is set. During TPSB, the next A-phase timing level is set.

Look-Ahead Control. -- When the CP is in the normal run condition, the sequencing of timing levels is under control of the look-ahead logic, except during the execution phase of an algorithm, when the algorithm itself specifies which execute cycle to enter.

The look-ahead feature permits the concurrent fetch and effective address formation of two sequential instructions. Figure 2-5 illustrates a typical example of a cycle sequence under look-ahead control. The two instructions are each assigned to a sequence (I or J), and each sequence has its own address and instruction register. Figure 2-5 shows a fetch cycle for the I-sequence instruction, followed by a fetch cycle for the J-sequence. Assuming there is no indirect addressing required, the I-sequence enters its execute phase of one or more execute cycles. After the I-sequence instruction is executed, the J-sequence is executed. Generally, the look-ahead feature provides for maximum use of memory. It includes control for cycle entry, gating to and from the instruction and address registers, and guarding against possible conflicts between the two instructions being processed.

Trap Control. -- The trap control logic conditions the timing level generator to set up the trap algorithm when errors are detected after certain arithmetic instructions, or when an unimplemented instruction is fetched. A code is generated which specifies a group number appropriate to the type of trap in effect. The trap group number is used to address a dedicated location during the trap algorithm. Interrupts are inhibited until an arithmetic trap condition has been processed.

RKL and MA Gating. -- This logic selects and distributes memory addresses for the I and J sequences, under control of the look-ahead logic. The CIL and CIK codes specify dedicated memory locations during traps and interrupts.

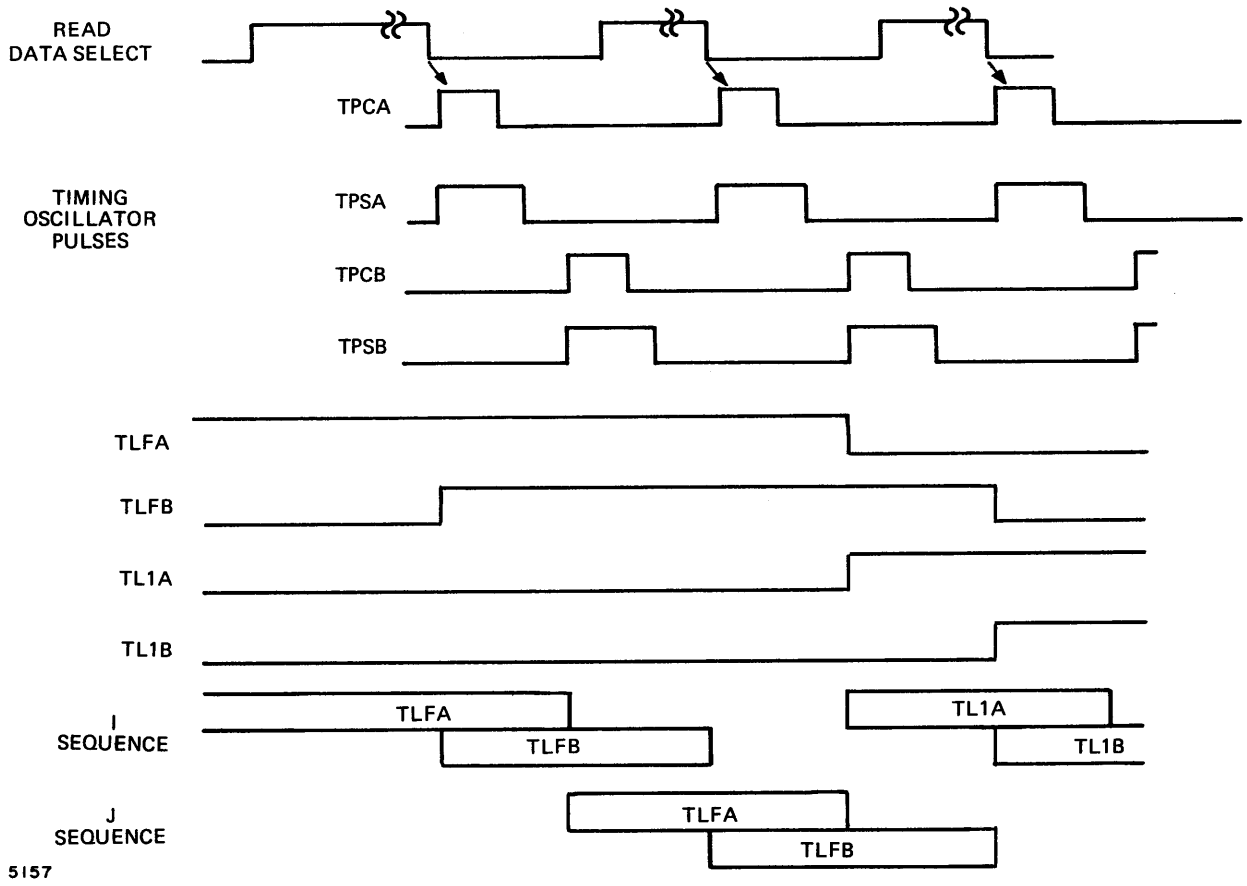


Figure 2-5. Simplified Central Processor Timing Signals

Normal CP Operating Sequence

The normal sequence of events from power-on through continuous operation, and the entry points to the CP inactive, memory wait, and panel halt conditions are illustrated in Figure 2-6.

System Initialize. -- Turning the POWER control switch to ON starts the primary power sequence which energizes the system dc power supplies. During the power-on sequence, a time delay relay simulates action of the SYSTEM INITIALIZE button for about a second. In the system initialized state, the CP is prepared to enter the fill mode and is in both a panel halt and the CP inactive condition. (This condition can be reinstated at any time by pressing the SYSTEM INITIALIZE button.) System initialize also transfers a code from the DEVICE SELECT switches on the panel to the RH register. The code specifies the input device that will supply the load program.

Fill Start. -- The CP is prepared to enter the fill (LOAD) algorithm as soon as the START button is pressed. The fill algorithm writes fill order commands in memory locations dedicated to IOP channel 6, and generates the MPC fill strobe which causes the MPC to start IOP channel 6 sequencing.

Immediately after the fill algorithm, because the MPC IDL code is the inactive code and the IAF register is cleared, the CP enters an interrupt. The only significant effect of this interrupt is to transfer the inactive code to the IAF register and clear the memory request. The CP then waits in the inactive condition. If the IOP bootstrap program so specifies, the MPC assigns a new and active CP process level by changing the IDL code. The resulting interrupt starts the CP running at the new process level.

CP Initialize Start. -- If the fill operation is not required, the CP INITIALIZE button is pressed. This clears the fill mode and causes the MPC to place the code for CP process level 0 on the IDL lines. When the START button is pressed, the CP interrupts to process level 0 and starts running.

Continuous Running. -- After a start, in the RUN mode, the CP runs continuously, fetching and executing instructions at the process level specified by the IAF register. The memory request remains set as long as the code in IAF is valid. If MAD is granting memory cycles to the CP, the CP initiates the next memory cycle, executes the current timing level of the algorithm in effect, and sets up the next timing level. Instruction execution continues until there is a trap or interrupt, or until a panel operation causes a halt. After an interrupt to an invalid process level, the CP enters the inactive condition.

Memory Wait. -- Even though the IAF code is valid and the CP is generating cycle request, operation may be suspended because MAD is giving memory cycles to a processor of higher priority. This appears to the CP as a withholding of address select and read data select. Once the other processor has released MAD, the waiting CP memory request

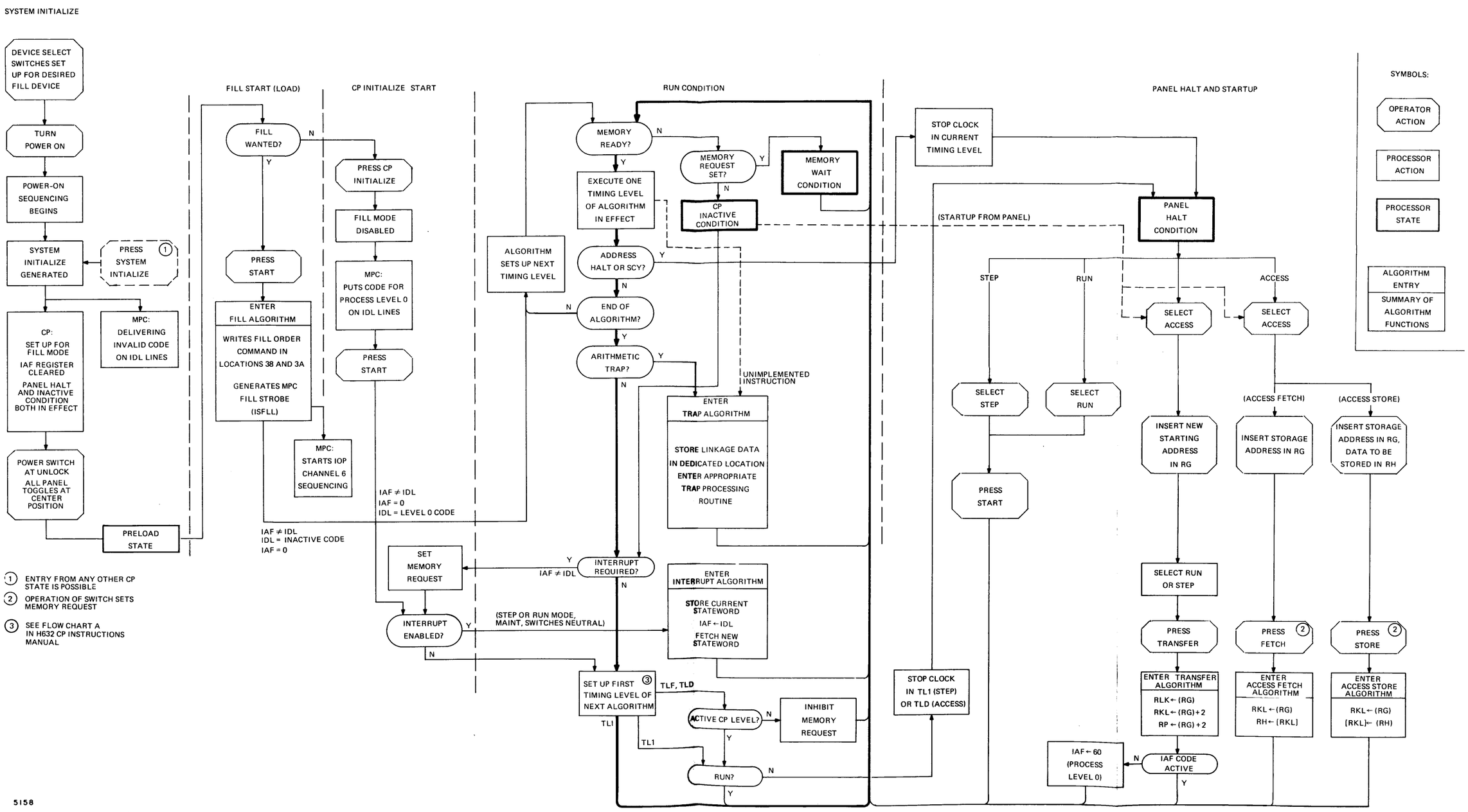


Figure 2-6. Central Processor Sequence of Operations

is recognized and both address select and read data select are restored to normal operation. The CP resumes continuous running. A memory wait condition is possible before each clock cycle of the trap, interrupt, and panel algorithms, as well as the CP execution algorithms.

When the IOP breaks in, the CP is held up in a memory wait for from three to six memory cycles; the IOP then releases control to the CP for at least one cycle.

When a memory wait occurs because the CP has removed its cycle request (CP inactive condition), operation stops until a change in the IDL code from the MPC causes an interrupt or the operator causes a panel start.

Address Halt and SCY Halt. -- During every timing level, if an address halt is present or the panel SCY/RPT switch is at SCY (single cycle), the CP clock stops and the CP enters the panel halt condition. The address halt results when the address for the next memory cycle matches an address manually set into the ADDRESS HALT keys on the panel. A switch at the left of the address keys permits disabling the address halt function. (This feature is an option.)

Traps. -- After a fetch cycle, if the fetched operation code is for an instruction that is not implemented in the system, the CP proceeds automatically to the trap algorithm. A trap can also be entered at the end of the execution of certain arithmetic instructions. After the trap algorithm, instruction fetching and execution resume.

Interrupts. -- The MPC may, asynchronously, change the code on the IDL lines at any time. At the end of each algorithm, when the resulting inequality between the IDL and IAF codes is detected, the CP enters the interrupt algorithm (provided an interrupt is enabled). Statewords of the current process are stored, the new process code is brought into IAF, and the statewords corresponding to that process are fetched. After the interrupt, processing continues at the location specified by the new stateword, provided the MPC code is valid.

CP Inactive Condition. -- After an interrupt, if the inactive code is in IAF, the memory request is cleared. Since no memory cycle occurs, the CP clock stops, and the CP inactive condition is in effect. This condition holds until the MPC delivers an active IDL code, which sets the memory request and causes an interrupt. After the interrupt, processing resumes at the new process level. During the CP inactive condition, the CP can be started from the panel for the duration of the panel access fetch or store algorithm.

Panel Halt. -- The CP may be halted at any time without destroying data by placing the execute control key at STEP or ACCESS. The CP completes the algorithm in effect, performs any waiting traps or interrupts, and then stops the clock. Startup must be controlled from the panel.

Access Mode. -- When the CP is stopped in the access mode, data and addresses may be inserted in RG and RH through the panel ADDRESS/DATA keys, as required for access store, access fetch, and transfer operations. For access fetch or store, startup is initiated by the FETCH or STORE buttons. After completing the selected algorithm, the CP stops in another panel halt.

Transfer. -- For a transfer, the step or run mode must be selected after setting up the starting address in the access mode. When the TRANSFER button is pressed, the transfer algorithm is performed. If the step mode is in effect, the CP enters another panel halt. If the run mode is selected, the CP fetches the instruction at the transfer location and continues running. If a transfer is attempted while the code in IAF is invalid, the transfer algorithm forces the code for process level 0 into the IAF register. After the transfer, the CP enters an interrupt and stores the transfer address in the dedicated location for process level 0. Because the interrupt is to an invalid level, operation then ceases in the CP inactive condition. When the MPC next commands the CP to operate at process level 0, another interrupt fetches the former transfer address, and the CP proceeds from that point in process level 0.

Resuming Continuous Instruction Processing. -- Continuous operation can be resumed from a panel halt by selecting the run mode and pressing START.

Single Instruction Operation. -- When the step mode is selected, the CP performs one full instruction algorithm, plus any waiting traps or interrupts, and then returns to the panel halt condition.

Identifying CP Status at Panel. -- The current status of the CP can be determined by inspecting a few indicators on the console. See Table 2-1.

CP INTERFACES

The CP logic frame is interfaced directly to the MPC and to MAD and the memory system. In addition, there is a connection to the IOP, actually a tandem connection to the main data/address bus to the memory port. (Either the CP or the MPC controls the bus through the bidirectional cable PACs.)

The control panel, indicator panel, and power control chassis (considered a part of the CP) interconnect to the CP logic frame through connectors. Interconnections to the system power distribution equipment are brought in through the power control chassis.

Figure 2-7 summarizes the methods of interconnection (cable PACs, connectors, or direct wiring). Wire-by-wire connection details and signal mnemonics appear in the logic diagrams specified in the illustration.

Table 2-1.
CP Start/Stop Condition Summary

CP Condition	Panel Indicators			
	CP ACTIVE	RUN	TLG	Other
System Initialize	OFF	OFF	TL1	LOAD
CP Initialize	ON	OFF	No change	
Normal Run	ON	ON	Changing	
Panel Halt:				
Step	--	OFF	TL1	
Access	--	OFF	None (TLD)	
SCY or Address Halt	--	OFF	Any	
CP Inactive	OFF	ON	TLF	
	Unusual Conditions			
Memory Wait	ON	ON	Any	(See Note)
Wait for Control Process Acknowledge	ON	ON	TL6	Op code = $30 (See Note)
Indirect Hangup	ON	ON	TLF & TLI or TLI alone	R-field and Op code not changing (Program Error)
Indefinite Execute Loop	ON	ON	Changing	Op code = $4A (Program Error)

NOTE

Normally, these waits are a few microseconds or less. If the CP pauses long enough for the condition to be seen at the panel, there is a hardware fault.

MPC Interface

The MPC has two interface connections to the CP. One exchanges process level codes and delivers an order code word and strobe during control process instructions. Sampling of the IDL code and gating from the IAF register to the MPC are discussed under Interrupt in Section III. See the Control Process Instruction Flow Chart and Analysis for detail on the order code word, order strobe, and acknowledgment.

The other interface supplies IOP and CP activity state data for display at the control panel; see the Console Control and Display Panel description in Section III.

Memory Interface

Interconnections to the Memory Access Director and to the memory port (shared with the IOP) are discussed in the Memory Interface description in Section III.

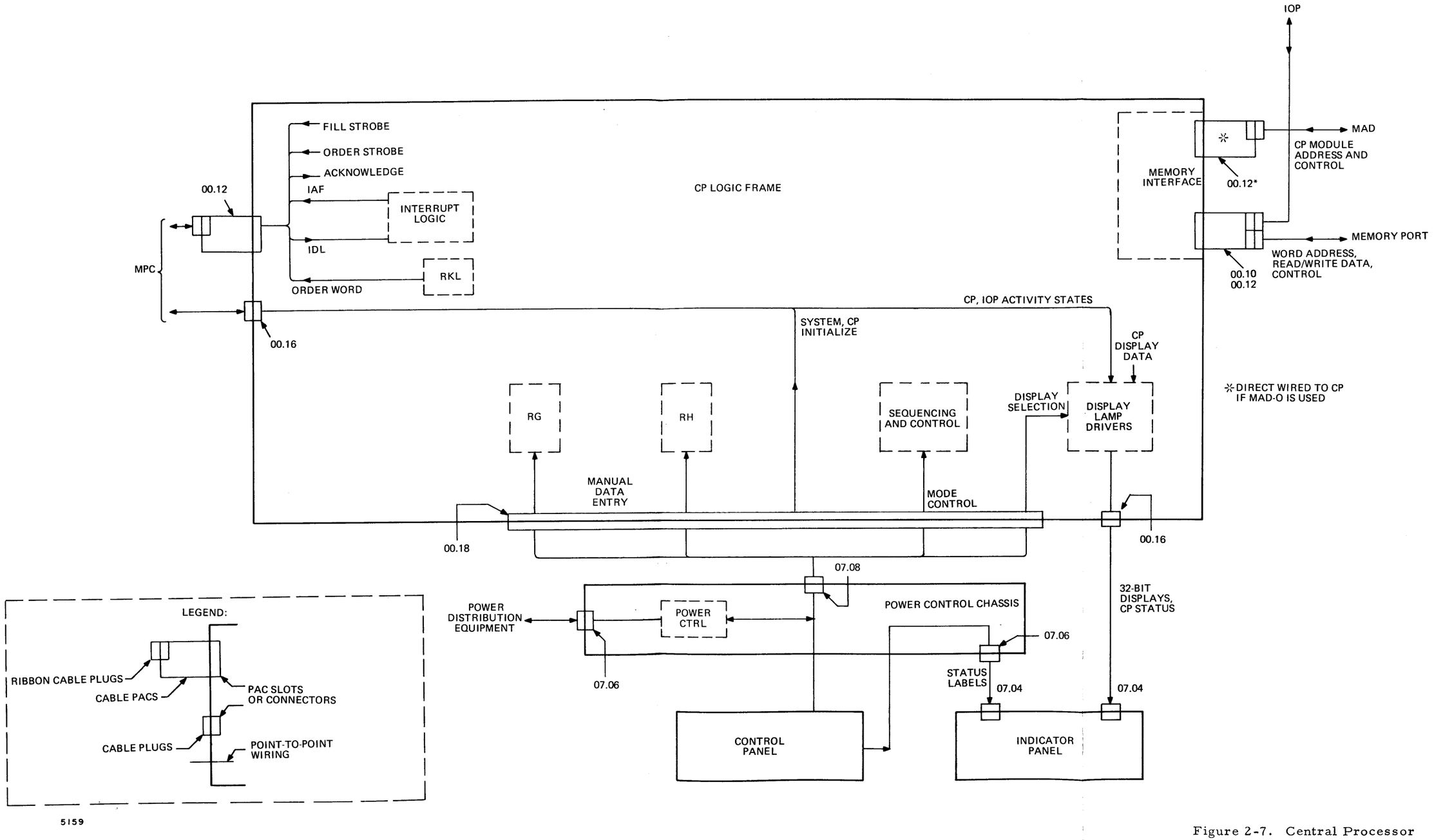


Figure 2-7. Central Processor Interface Connection Summary

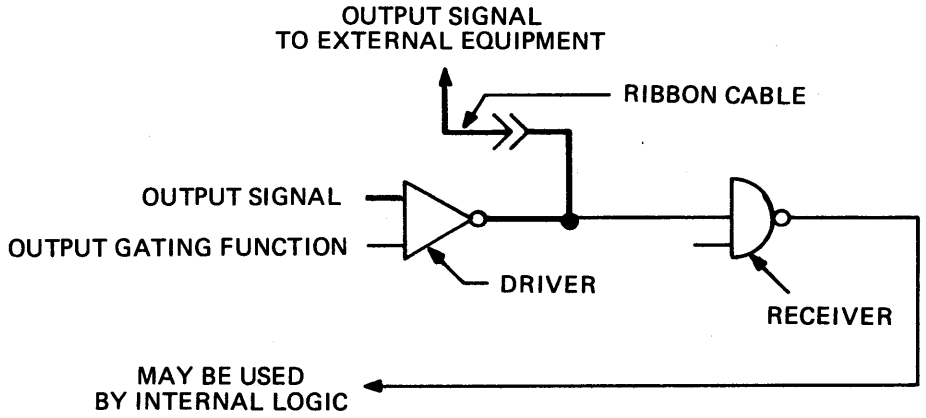
Control Panel and Power Control Interconnections

See the Console Control and Display Panel description in Section III for details.

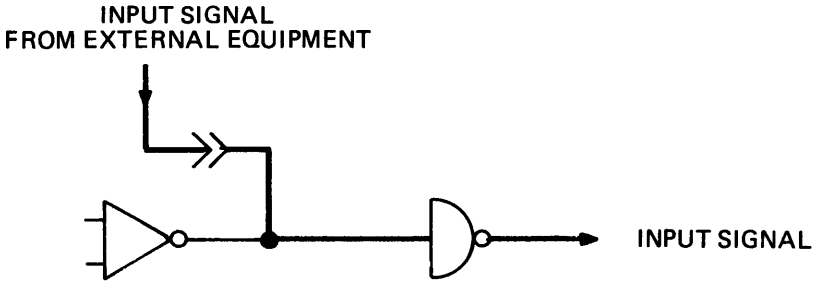
Cable PACs

Special cable PACs are used for one-way and bidirectional information transfer at the CP, at MAD, at the IOP, at the memory port interface and at all IO controllers. Each PAC contains ten driver-receiver circuits of the type illustrated in Figure 2-8. The circuits permit gated data transfer in either direction along a single interconnecting cable. On some PACs (type CC-214) each circuit is wired to the same pin on two identical ribbon cable connectors. This permits data to be placed on, or read from, a given line by different devices. (See Figure 2-8C.) Memory address, read/write data, and control lines from both the IOP and CP are applied to common lines in this way.

Type CC-215 cable PACs have a single connector only; each receiver circuit contains resistors which terminate a cable or chain of cables.



A. OUTPUT FUNCTIONS



5160

B. INPUT FUNCTIONS

Figure 2-8. Cable PAC Applications (Sheet 1 of 2)

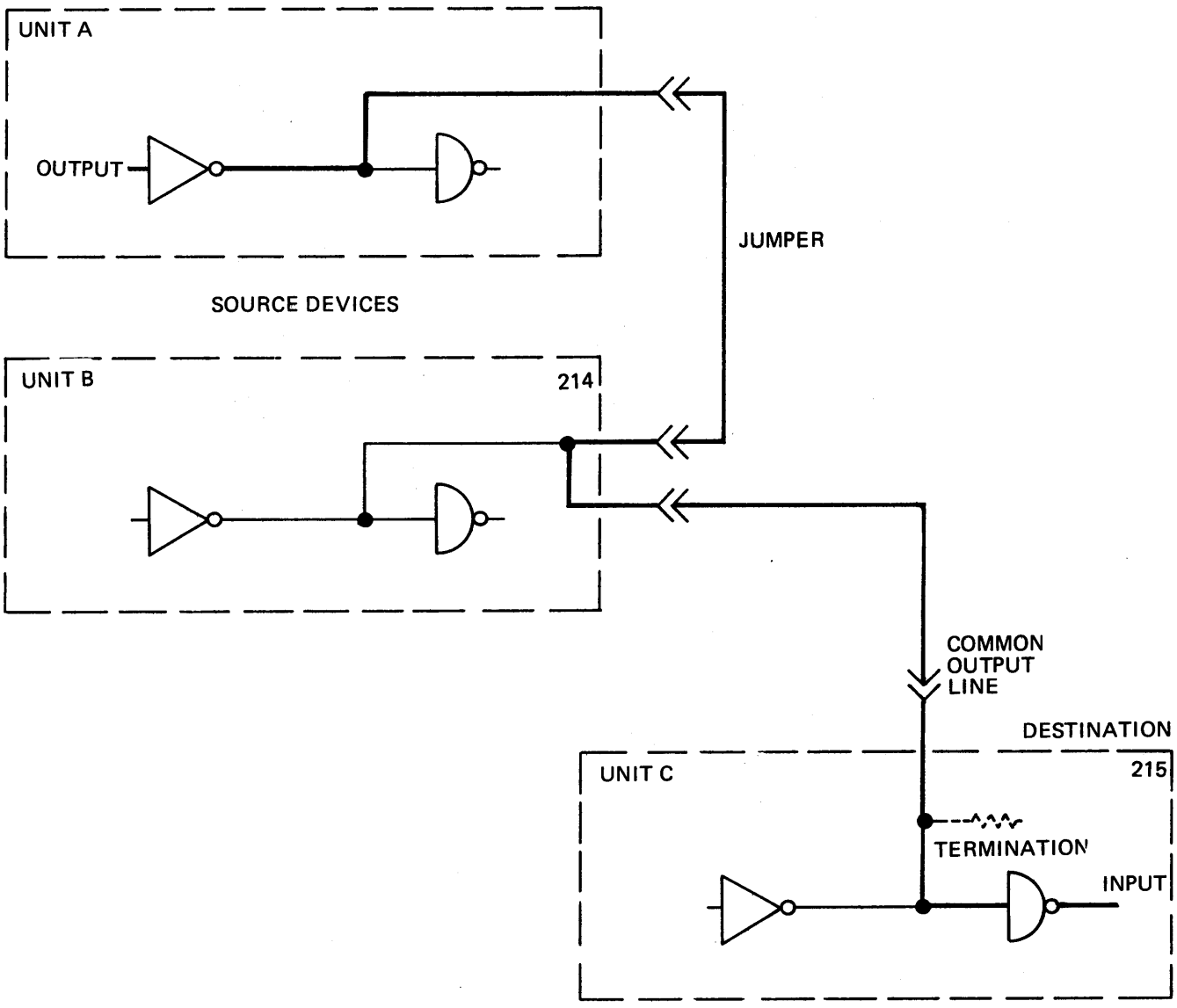


Figure 2-8. Cable PAC Applications (Sheet 2 of 2)

SECTION III
DETAILED THEORY

Figure 3-1 maps the functions that are active in the CP during execution of an algorithm. The primary functions (bold outlines) are the operational CP algorithms (fetch, indirect cycle, instructions, panel functions, trap, and interrupt) which execute the actual program. These primary functions are described in detail in the flow charts and instruction analyses of the H632 Central Processor Instructions manual.

Secondary functions (single-weight lines) operate during the same timing levels as the primary algorithm to anticipate interrupts, detect trap conditions, control the memory interface, permit operator control at the proper time, and control the cycle look-ahead function. These sequencing and control functions are described in this section with the use of detailed flow charts or timing diagrams.

START-STOP, TIMING, AND CONTROL LOGIC

System Initialize (See Figure 3-2.)

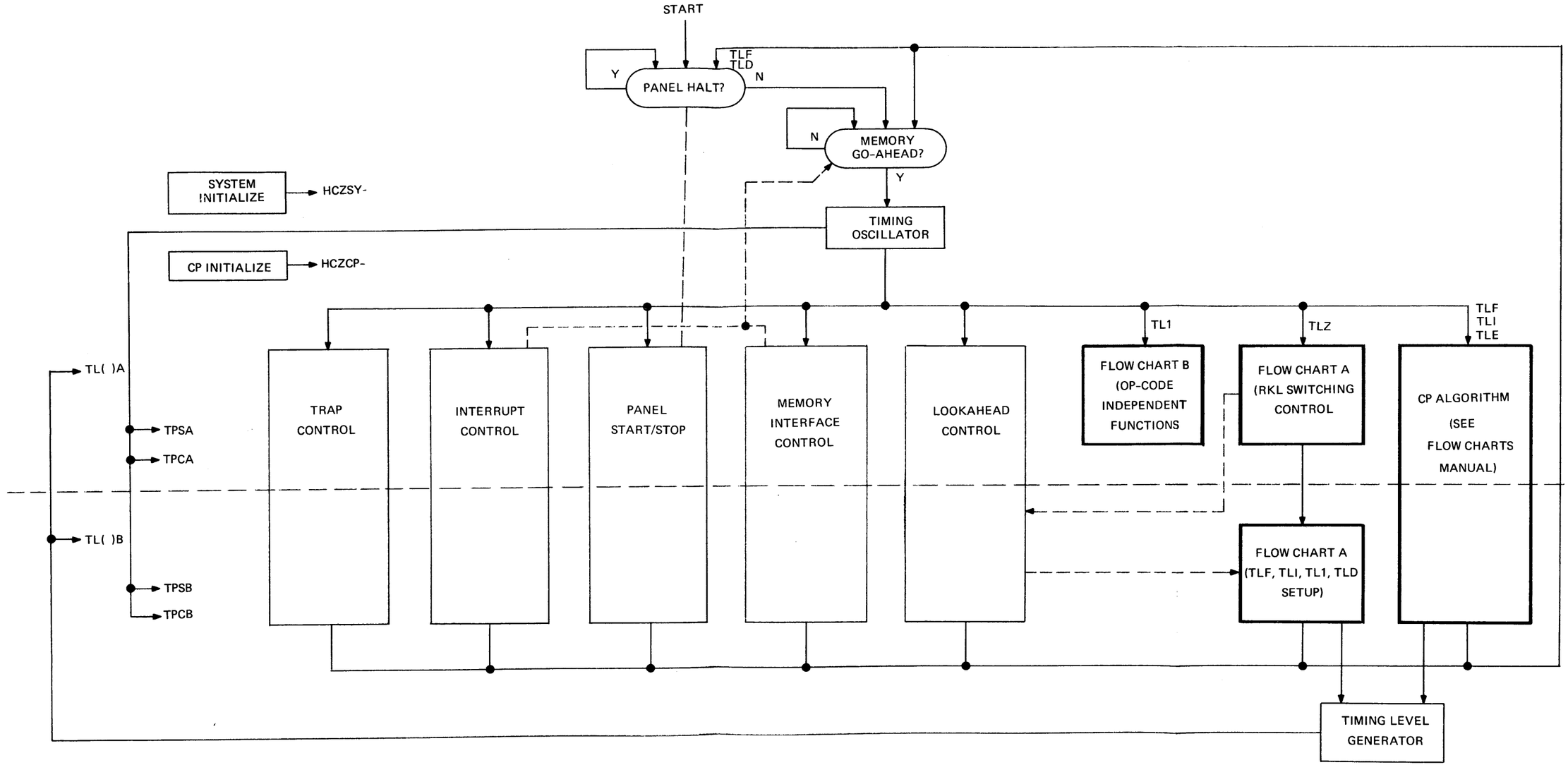
Turning power on or pressing the SYSTEM INITIALIZE pushbutton (with the power switch at UNLOCK) presets a known starting condition in the CP sequencing and control logic, and also in MAD, the MPC, and the IOP.

Start-Stop Logic. -- The fill flip-flop is set and conditions are set up for a panel-halt condition (the clock will not run until a manual start from the panel is executed). Once started from the panel, the CP enters the fill algorithm.

Timing Level Generator. -- The TLXAZ signal clears all timing level flip-flops except TL1A, in preparation for the fill algorithm (or an interrupt, after CP initialize).

Memory Interface. -- System initialize clears TPCAF, disabling cycle initiate. The memory busy line is cleared in memory during power turn-on.

Memory Access Director (MAD). -- Pressing the SYSTEM INITIALIZE pushbutton clears the main CP interface data and address select signals. The memory busy line also initializes MAD internal timing circuits.



5162

Figure 3-1. Time Sharing of Execution Functions with Sequencing and Control Functions

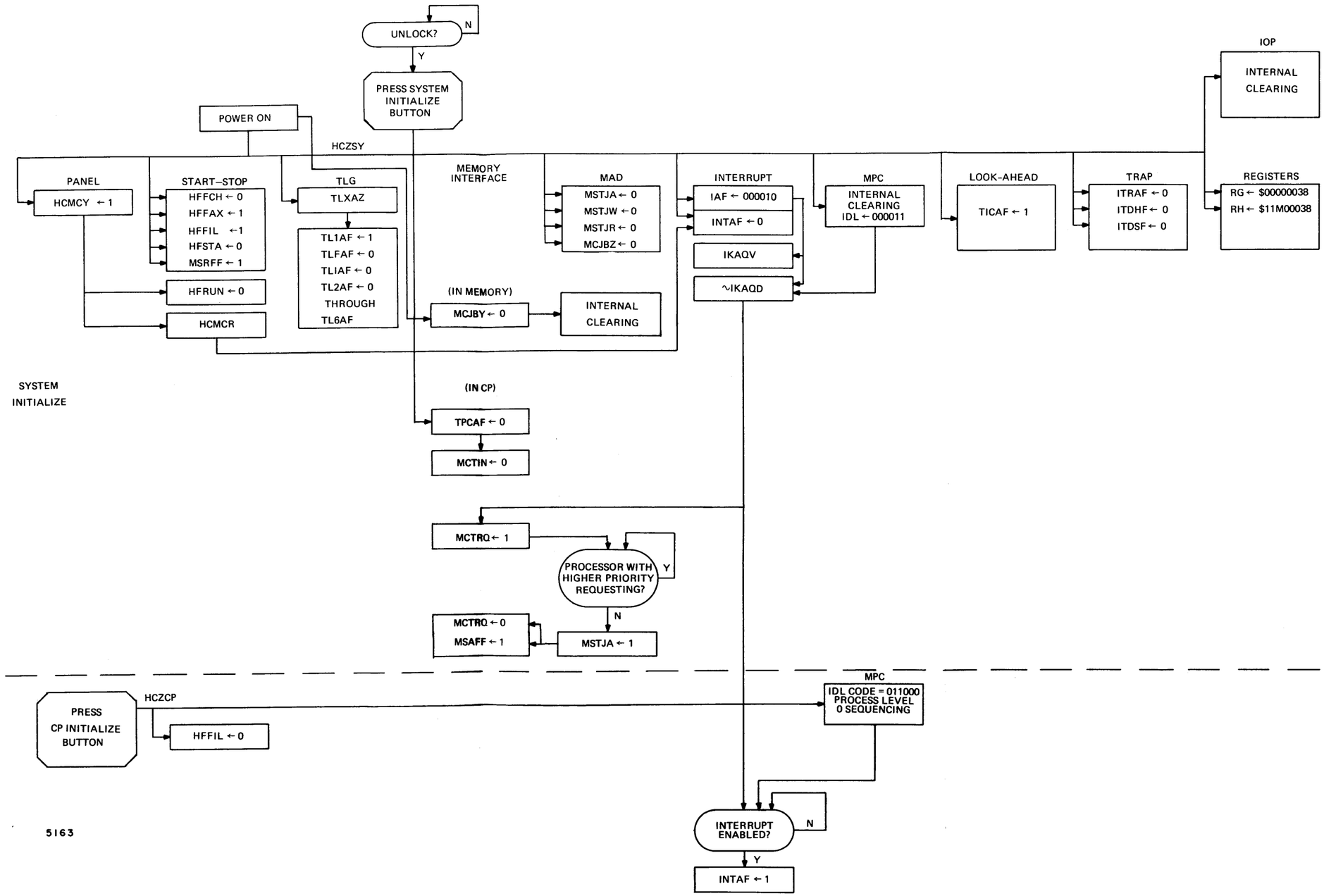


Figure 3-2. System Initialize and Central Processor Initialize Flow Chart

Multiprocess Controller (MPC). -- The system initialize condition forces code 000011 (the inactive code) on the IDL lines from MPC to the CP.

Interrupt Logic. -- The system initialize condition sets an inactive level code in the IAF register and clears the interrupt flip-flop. The difference between the IDL code from MPC and the IAF register code is detected by IKAQD, and the memory request flip-flop is set. The memory request is received by MAD; if no higher priority processor is requesting a memory cycle, MAD responds with address select. The CPs memory request is cleared by address select.

Trap Logic. -- The three trap-condition flip-flops are cleared.

Registers. -- In preparation for a fill operation, register RG is loaded with $00000038 (the CS1 location for IOP process level 6), and register RH is loaded with $11M00038. The latter is a fill order transmission command addressed to device 000 of controller 1M (M is the code manually set into the panel device select keys).

Look-Ahead Logic. -- The system initialize condition sets the TICAF flip-flop.

Input/Output Processor (IOP). -- Internal logic is preset to a known starting condition (no effect on CP).

CP Initialize (See Figure 3-2.)

The CP INITIALIZE button clears the fill flip-flop in the start-stop logic and also forces the code for process level 0 on the IDL lines from the MPC. The interrupt flip-flop is set (unless interrupt is disabled from the panel).

Start-Stop Logic (See Figure 3-3.)

The start-stop logic (LBD 00.40) controls an orderly computer startup after system or CP initialize and controls stopping of the clock oscillator in the step or access mode, during an address halt (optional), or when the single-cycle (SCY) maintenance mode is selected. Continuous running and startup depend on a go-ahead from the memory interface logic.

System Initialize. -- Power turn-on or pressing the SYSTEM INITIALIZE button sets the fill flip-flop (HFFIL) and initializes the start-stop controls for a halt condition. (HFSTA and HFRUN are cleared.) The timing level generator is set up in TL1A. The mode key is assumed to be in RUN, with the power switch at UNLOCK.

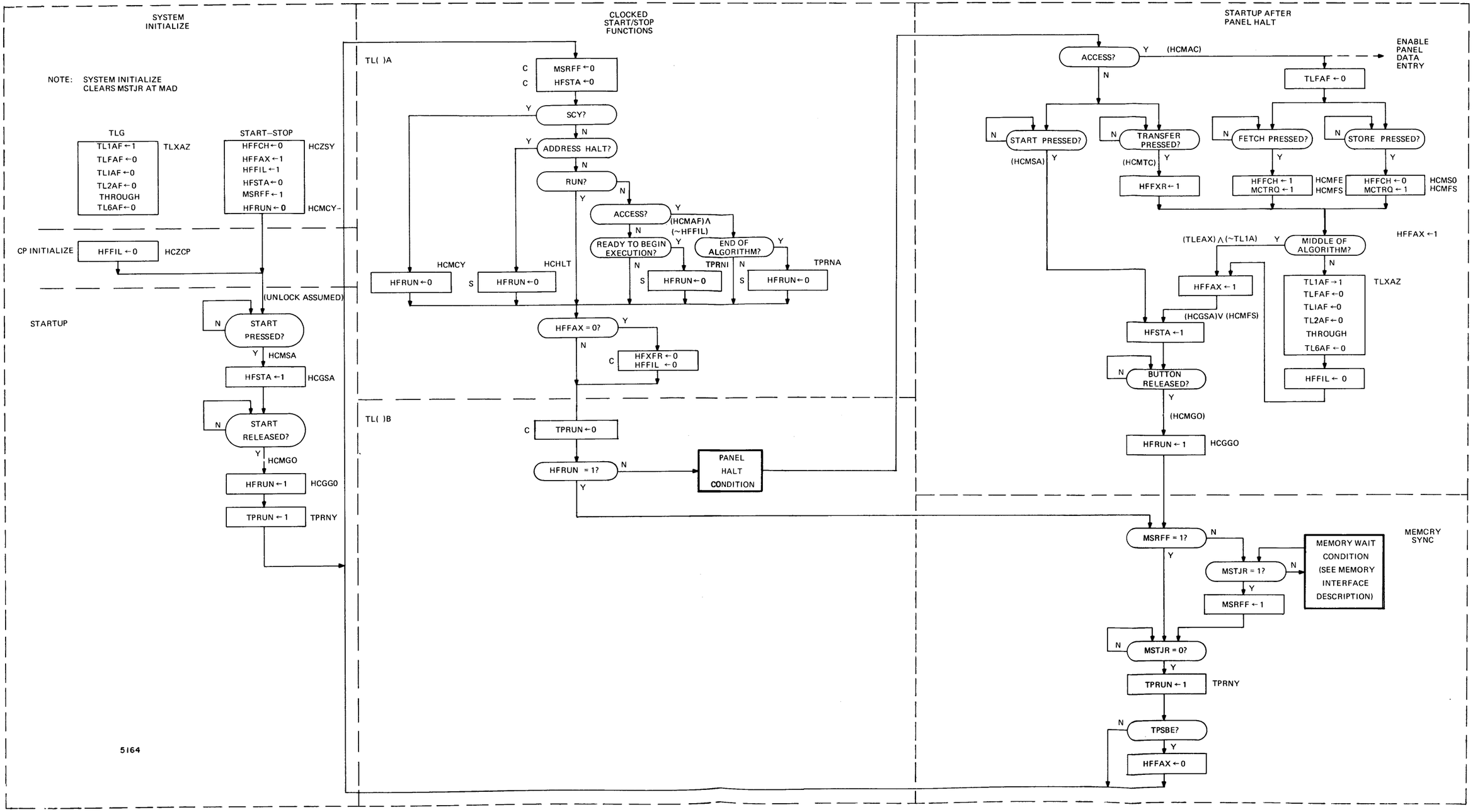


Figure 3-3. Start-Stop Logic Flow Chart

Startup from System of CP Initialize. -- In a normal operating sequence, if a fill operation is desired, the START button is pressed and released. Because MAD and the memory are also initialized, the TPRUN flip-flop is set as soon as the RUN flip-flop (HFRUN) is set. The CP is then running in the fill algorithm.

After a CP initialize, the fill flip-flop is cleared. When START is pressed and released, the CP is running in an interrupt algorithm. The interrupt is followed by normal continuous running and execution of the program level brought in by the interrupt.

Continuous Running. -- Program execution of the current process level continues until the step mode is selected or an interrupt brings a different MPC code to the IAF register.

While the run mode is in effect and the process level in IAF is valid, the start-stop logic maintains clock synchronization with the memory interface. The memory-wait flip-flop (MSRFF) is cleared during TPCA. Both the leading and trailing edges of read data select (MSTJR) must be received in order for the clock to proceed. The leading edge is stored by MSRFF. (After a system or CP initialize, MSRFF is waiting-set.) Ordinarily, MSRFF is set during timing level B, but the trailing edge of MSTJR does not occur until after TPCB. As a result, TPRUN is cleared by TPCB. As soon as MSTJR ends (the memory go-ahead), TPRUN is set, and another cycle of clock pulses is generated.

Memory Wait. -- When MAD has given another processor access to the memory bank (or an inactive MPC code has removed the CP memory request), the CP waits with the clock stopped. MSRFF and TPRUN are cleared and MSTJR is zero. When MAD returns control to the CP (or when an active MPC code is received), read data select is received, MSRFF and TPRUN are set, and normal running resumes. (See the Memory Interface description.)

Panel Halt. -- Continuous running can be stopped from the panel by placing the mode key in STEP or ACCESS, by selecting the SCY maintenance mode, or by using the ADDRESS HALT option.

In the step mode, when the CP has fetched an instruction and formed its effective address, the run flip-flop (HFRUN) is cleared during TPSA. During TPSB, TPRUN is cleared, stopping the clock in the panel halt condition with TL1A set.

In the access mode, during the last execution cycle of the algorithm, the run flip-flop is cleared by TPSA. HCMAC prevents setup of the next fetch cycle. During TPCB, TPRUN is cleared and the CP stops in dummy timing level TLD (no timing level set).

In any timing level, if SCY is on or an address halt is present, HFRUN is cleared. During the following TPCB, TPRUN is cleared and the panel halt is in effect.

Step Startup. -- When the START button is pressed and released, HCGGO sets the run flip-flop. If MSRFF and MSTJR are in a memory go-ahead condition, the TPRUN flip-flop is set and one full algorithm is executed. A normal step halt occurs with TL1A of the next algorithm set up and waiting to be executed.

Transfer Startup. -- If TRANSFER is pressed instead of START, HFFXR and HFFAX are set as the effective op code for the transfer algorithm, and TL1A is set up in the timing level generator. TPRUN is set as in START (if there is a memory go-ahead), and the CP performs the transfer algorithm.

After a transfer from the step halt condition, a normal step halt occurs with TL1A of the next algorithm waiting to be executed. If transfer is initiated in the run mode, continuous operation resumes at the new instruction location.

Access Fetch and Store. -- With the mode key in ACCESS, startup is controlled by the FETCH or STORE button. When the FETCH button is pressed, the HFFCH and HFFAX flip-flops are set as the effective op code for the access fetch algorithm and TL1A is set up in the TLG. When the FETCH button is released, HCGGO starts the clock (assuming a memory go-ahead), and the access fetch algorithm is performed. During the last execution cycle of the algorithm (TLZAX), HFRUN is cleared, and the clock is stopped in the panel halt condition.

When the STORE button is pressed and released, the start-stop logic operates as in access fetch, except that the HFFCH flip-flop is cleared to provide the effective op code for the access store algorithm.

If the panel RPT (repeat) maintenance switch is on, the access fetch and store algorithms do not generate TLZAX; access continues until the RPT switch is turned off.

Clearing the Console Algorithms. -- Any operation of the transfer, fetch, or store buttons sets HFFAX, which serves as a part of the effective op code for panel algorithms. One other start-stop control flip-flop (such as HFXFR, for transfer) contributes to the op code. System initialize also sets HFFAX to enable the fill algorithm.

At the end of any panel algorithm, TPSBE clears HFFAX. During the following TPCA, HFXFR and HFFIL are cleared so that no console algorithm is in effect.

Returning to Continuous Run. -- When the run mode is selected after a step or an access operation, startup can be controlled either by the START or TRANSFER button. If the START button is used, the CP fetches and begins execution of the current instruction. If the TRANSFER button is used, the transfer algorithm is executed, and continuous instruction processing begins at the address manually inserted in RG. (See Normal CP Operating Sequence in Section II for information on attempted transfer in the CP inactive condition.)

Single-Cycle Operation. -- If SCY (single-cycle) operation is selected by use of the maintenance panel switch, the HFRUN flip-flop is cleared unconditionally. After every operation of the START, TRANSFER, FETCH, or STORE button, the HFRUN flip-flop is set, then cleared every TPCA time, when the HFSTA flip-flop is cleared. This mode permits the operator to step through the current algorithm one clock cycle at a time.

The first time the TRANSFER, FETCH or STORE button is pressed, TLXAZ sets the timing level generator to TL1A. Thereafter, each operation of the button steps the CP through one clock cycle of the selected algorithm. The (TLEAX) $\wedge$ ($\sim$TL1A) input to TLXAZ prevents setup of TL1A again until the algorithm is complete.

Address Halt (Optional). -- This optional logic (shown on LBD 00.42) compares the current memory address on the MAT lines with the settings of the ADDRESS HALT keys on the panel. When the addresses match, HCHLL and HCHLH cause entry to the panel halt condition by clearing HFRUN.

Timing Oscillator and Clock Distribution (LBD 00.56)

Once it is triggered by TPRUN from the start-stop logic, the timing oscillator PAC generates one set of clock pulses: clear A, set A, clear B and set B. The basic signals are produced by the TPCAX, TPSAX, TPCBX and TPSBX gates, controlled by the three delay lines. Idealized timing is shown in Figure 3-4. TPCBX clears the TPRUN flip-flop preventing more clock pulses until the "panel run" and "memory go-ahead" conditions are in effect.

The four basic signals are fanned out through load-balancing resistors to distributing gate PACs, one in each quadrant of the central processor frame. The active signals (TPSA1, etc.) are wired from the distributing gates to logic circuits nearby. The delay lines can be adjusted so that the active signals approach the timing shown in Figure 3-4 in spite of gate and propagation delays.

The TPXXX signal clears MSAFF and also provides for an early clear of the write flip-flop (MCTWR) in cases where a memory write cycle is followed by a read cycle. (See the Memory Interface description.)

Timing Level Generator (LBD 00.60)

The timing level generator consists of the flip-flops that store the current CP timing state, and the logic gates that determine the next timing state. Timing levels are divided into two phases, A-phase and B-phase. At any given time, no more than one A-phase and one B-phase flip-flop are set.

Sequencing from state to state is triggered by the timing oscillator pulses. A fetch cycle, for example, starts with TLFA set. The next TPSA clock pulse executes the TLFA functions and sets up the TLFB flip-flop. The following TPSB clock pulse executes the TLFB functions and sets up the next A-phase timing level.

Entry into the TLFA, TLIA, and TL1A states is controlled by instruction look-ahead, interrupts, and trap conditions. More details appear under those headings and in flow chart A of the Central Processor Instructions manual.

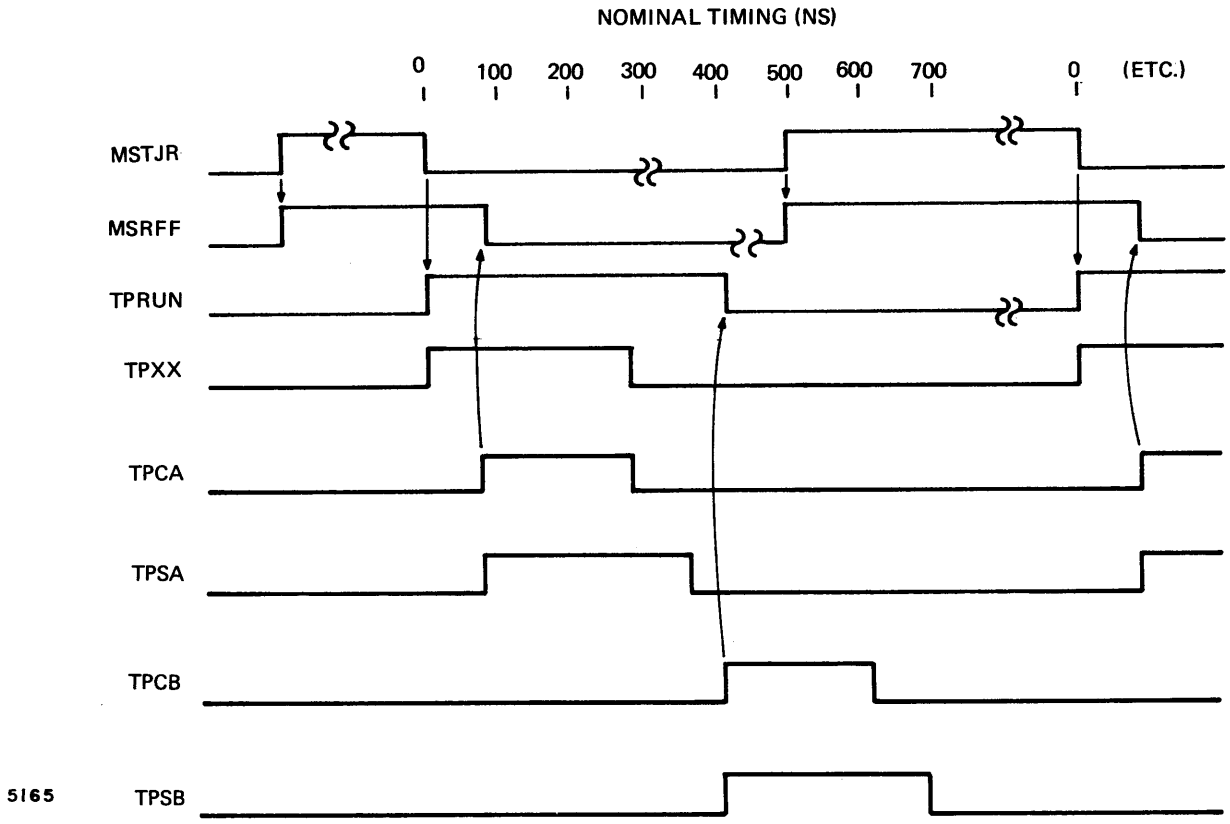


Figure 3-4. Clock Oscillator Timing

Execution of every algorithm starts in TL1A. Some instructions complete execution during a single cycle. Algorithms that require more execution cycles arbitrarily select a sequence of other timing states (TL2A-TL2B, etc.).

Figure 3-5 illustrates a series of timing levels during execution of a console access fetch or store algorithm. Timing level TL1A is set up when the console FETCH or STORE button is pressed (or also by a system initialize). When the switch is released, HCGGO starts the timing oscillator and the timing pulses advance the timing levels. The access fetch/store algorithms arbitrarily proceed from TL1A to TL6, TL2, and TL3. If the console RPT (repeat) button is off, the algorithm ends in a panel halt after TL3, as illustrated. (With the RPT button on, TL2 and TL3 repeat indefinitely.)

In the access mode, TLFAF is inhibited by HCMAC, so the algorithm ends with no timing level flip-flop set. This is the "dummy" timing level called "TLD" elsewhere in this manual and in the Central Processor Instructions manual. Dummy timing levels are also set up during overlapped instruction execution, to resolve precedence conflicts between the I and J sequences. (See description of Look-Ahead.)

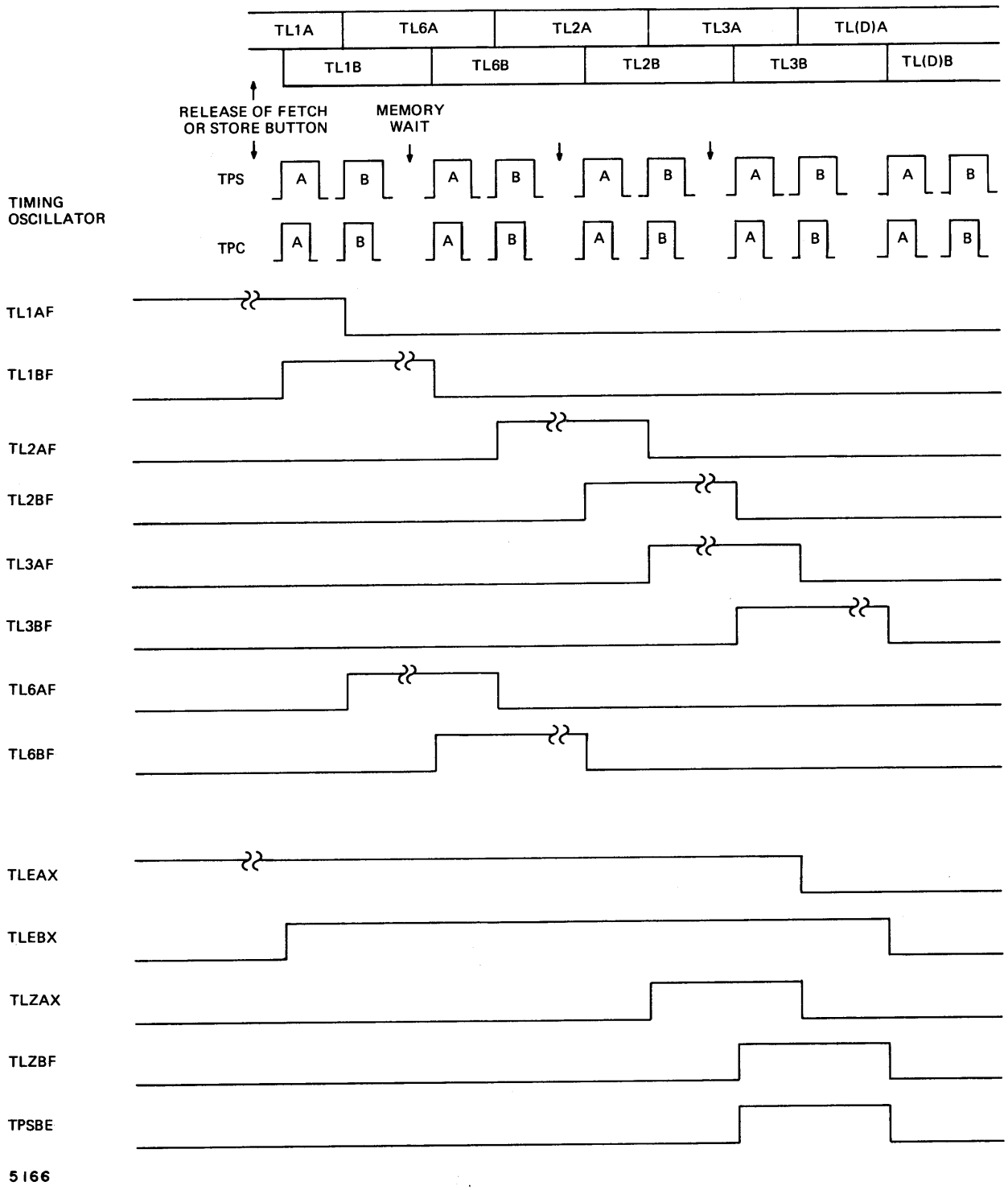


Figure 3-5. Timing Level Generator Signals During Access (Example)

Figure 3-5 illustrates signals that keep track of the execution portion of an algorithm. The TLEAX flip-flop is set during TL1A of each algorithm and kept set through the last execution A-phase. The TLEBX flip-flop is kept set during all B-phases of the algorithm. The last execution cycle of each algorithm is identified by the TLZAX signal, which permits the TLZBF flip-flop to be set by the next TPSA pulse. The TLZBJ and TPSBE signals contribute to the TL1A, TLIA, and TLFA setup logic to be discussed under "Look-Ahead."

The notation at the top of Figure 3-5 represents the current timing state of the CP in relation to other internal or external events.

Interrupt Register and Control (LBD 06.70 and 06.72)

IAF Register. -- The interrupt register stores the currently active CP process level and compares it with the process level code from the MPC. When the codes are unequal, IKAQD goes false and causes an interrupt during the next timing level (if interrupt is enabled). During TL3 of the interrupt, the MPC code on the IDL lines is transferred into the IAF register. (Thereafter, the MPC and IAF codes are equal; IKAQD is true until the MPC changes its code again.)

Interrupt Controls (See Figure 3-6). -- When a change in the MPC code is detected by the interrupt register during any timing level except TL1A, IKAQD sets INTAF (provided interrupt is enabled and no arithmetic-exception trap condition is present). INTAF enables the timing level generator to enter TL1A during the last execution cycle of the instruction in progress, or during a dummy cycle, F-cycle, or I-cycle. During TL1A of the interrupt algorithm, INTBF is set. INTBF serves as the equivalent of an interrupt op code, enabling the minterms which execute the interrupt algorithm.

Enabling Interrupt. -- In normal run or step operation, with all maintenance switches at the neutral center position, interrupt is enabled from the panel. In the access mode, interrupt is inhibited; since addresses are set up manually, the active process level is unimportant. The maintenance switches affect interrupt as follows:

ENI/INI at INI	Permits single-instruction stepping without interruption to another process level
SCY/RPT at SCY	Permits single-clock-cycle stepping without interrupt to another process level
SCY/RPT at RPT	Prevents interrupt during repetition of algorithms
ENI/INI at ENI	Enables interrupt in spite of SCY to permit single-cycle checkout of the interrupt algorithm itself

Even when interrupt is enabled from the panel, an arithmetic-exception trap condition postpones the interrupt until the trap operation is complete.

CIL Gates. -- Under control of the interrupt algorithm, the code in IAF is transferred through the CIL 24 through 29 gates to RKL, to identify the location of the program state-words for the interrupted and interrupting process levels.

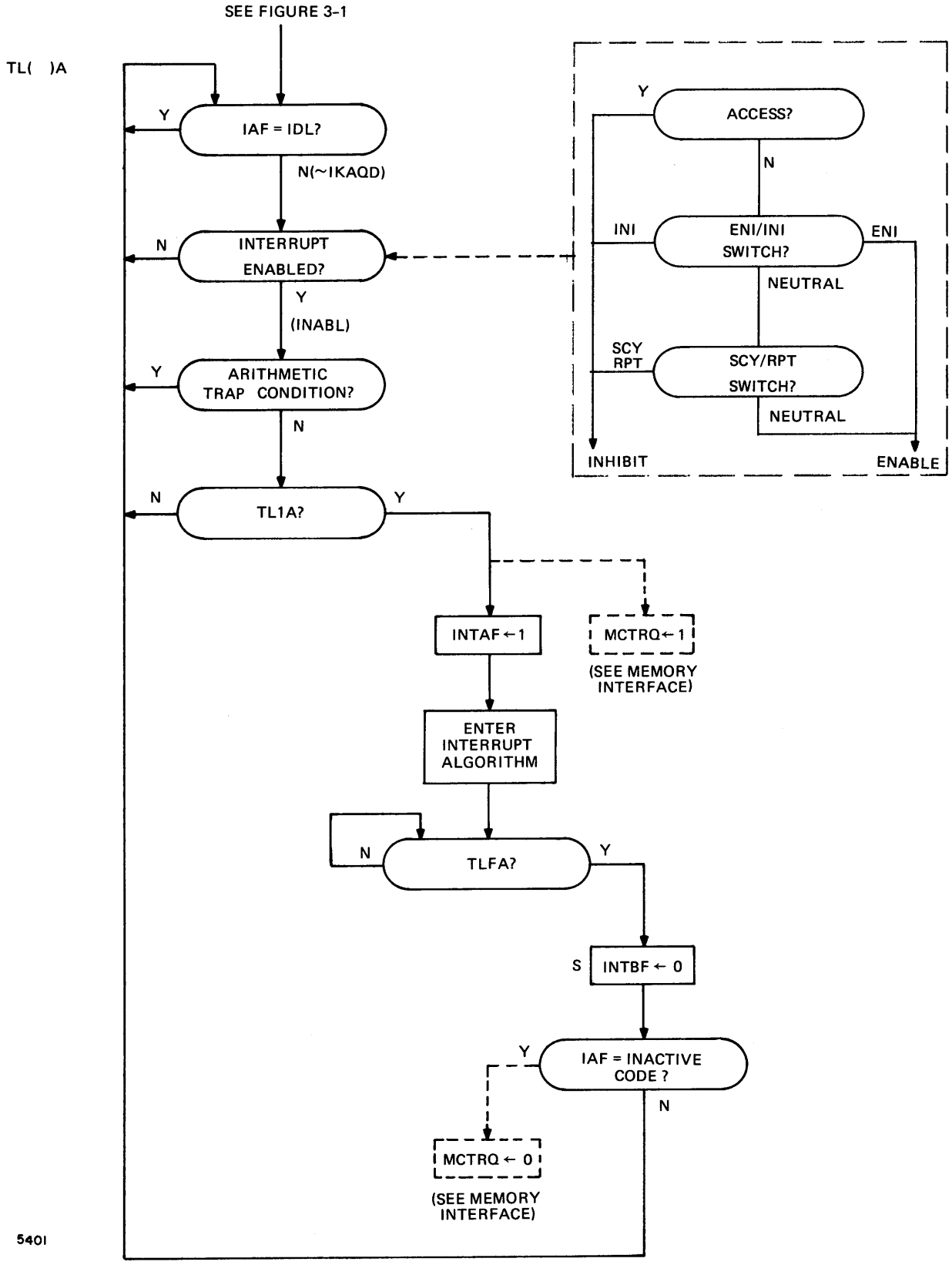


Figure 3-6. Interrupt Control Flow Chart

Inactive Code Detector. -- The IKAQV decoder monitors IAF 24, 25 and 28 for an inactive code of the form 00XX1X. When the MPC inactive code (000011) has been stored in IAF by an interrupt, IKAQV becomes false. When the interrupt is over and the TLG enters TLF, the memory request flip-flop (MCTRQ) (LBD 00.50) is cleared. The CP waits in the CP inactive condition until a new MPC code is received. IKAQD then becomes false, sets the memory request flip-flop, and starts another interrupt by setting INTAF (assuming interrupt is enabled from the panel).

IKAQV also modifies the transfer algorithm when IAF contains an inactive code.

Trap Control Logic (LBD 06.60) (See Figure 3-7.)

Arithmetic Exception. -- When an arithmetic exception occurs, the ITRAF flip-flop is set during TPSB. At the end of the instruction, flow chart A detects ITRAF and sets up TL1A of the trap algorithm. ITRAF takes precedence over a waiting interrupt by preventing INTAF (LBD 06.72) from setting. The ITDHF and ITDSF flip-flops are set at the same time as ITRAF to further subdivide the type of arithmetic trap.

During TL1A of the trap algorithm, ITRBF is set. ITRAF and ITRBF generate DITRA and DITRB, which serve as effective op codes during the trap algorithm. ITRAF is cleared during TL6B of the trap algorithm. ITRBF is cleared during the following TL1A.

Unimplemented and Undefined Operations. -- Also contributing to DITRA and DITRB are terms generated when unimplemented or undefined operation codes are detected:

DIUNA, B	Undefined Operation
DIFLA, B	Unimplemented Floating Point Operation
DIEXA, B	Unimplemented Extended Operation
DIBBA, B	Unimplemented Byte Operation

These terms do not set ITRAF and so do not take precedence over waiting interrupts. When an undefined or unimplemented instruction is fetched, the CP proceeds with the trap algorithm rather than an instruction algorithm.

Trap Condition Class Code Generation. -- Early in the trap algorithm, the CP accesses a dedicated location to obtain a pointer to one of four class entry tables. The address of the dedicated location is encoded when CIKRC enables ITCOV, ITCUN, and ITCEB, which control CYK24 through CYK30 on LBD 01.75. The four class codes are generated as follows:

		CYK Code								
	Trap Class	24	25	26	27	28	29	30	(31)	Hex Equiv.
I	Arithmetic Exception (ITCOV)	0	0	0	1	0	0	0	0	10
II	Unimplemented Floating Point (ITCUN)	0	0	0	1	0	0	1	0	12
III	Unimplemented Extended or Byte Operation (ITCEB)	0	0	0	1	0	1	0	0	14
IV	Undefined Operation (ITCUN, ITCEB)	0	0	0	1	0	1	1	0	16

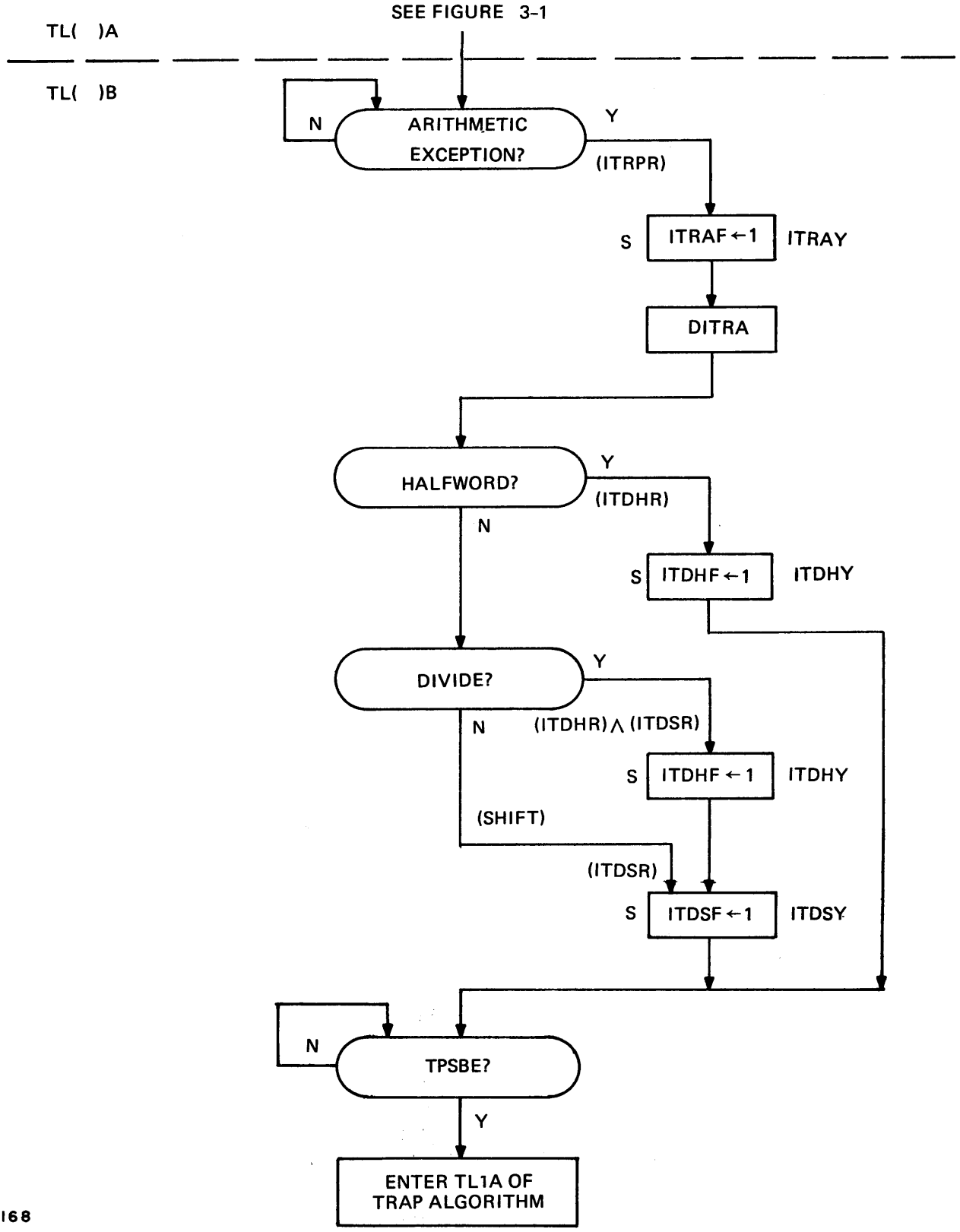


Figure 3-7. Trap Control Flow Chart

Addressing Trap Condition Table Locations. -- Later in the trap algorithm, one of the locations in the selected trap class entry table is accessed to obtain to the trap processing routine. The offset from the initial table address is produced when CIKRO enables a code on CIK27 through CIK30. These signals drive CYK27 through CYK30 on LBD 01.75. The code is related to trap class subdivisions in Table 3-1.

Table 3-1.
Trap Condition Table Entries

Bits 27-31 of Table Address	Trap Condition Class			
	I Arithmetic Exception	II Op Code	III Op Code	IV Op Code
00000	X	X	X	X
00010	X	X	X	X
00100	-	E2	E0	-
00110	-	C2	C0	-
01000	Overflow on fullword	82	B0	A0
01010	Overflow on shift	92	90	80
01100	Overflow on halfword	F2	F0	A2
01110	Zero division	D2	D0	82
10000	-	A6	24	-
10010	-	86	-	-
10100	-	E6	26	-
10110	-	C6	06	-
11000	-	B6	2C	20
11010	-	96	28	00
11100	-	F6	2E	22
11110	-	D6	2A	02

Note: Locations marked "X" are loaded by trap; locations marked "-" are reserved.

MEMORY INTERFACE

The CP memory interface consists of the control logic on LBD 00.50 and the control lines exchanged with MAD and a processor port. Logic elements in the CP which are related to the handshake between the CP start-stop and timing logic, MAD, the processor port, and IOP are illustrated in Figure 3-8.

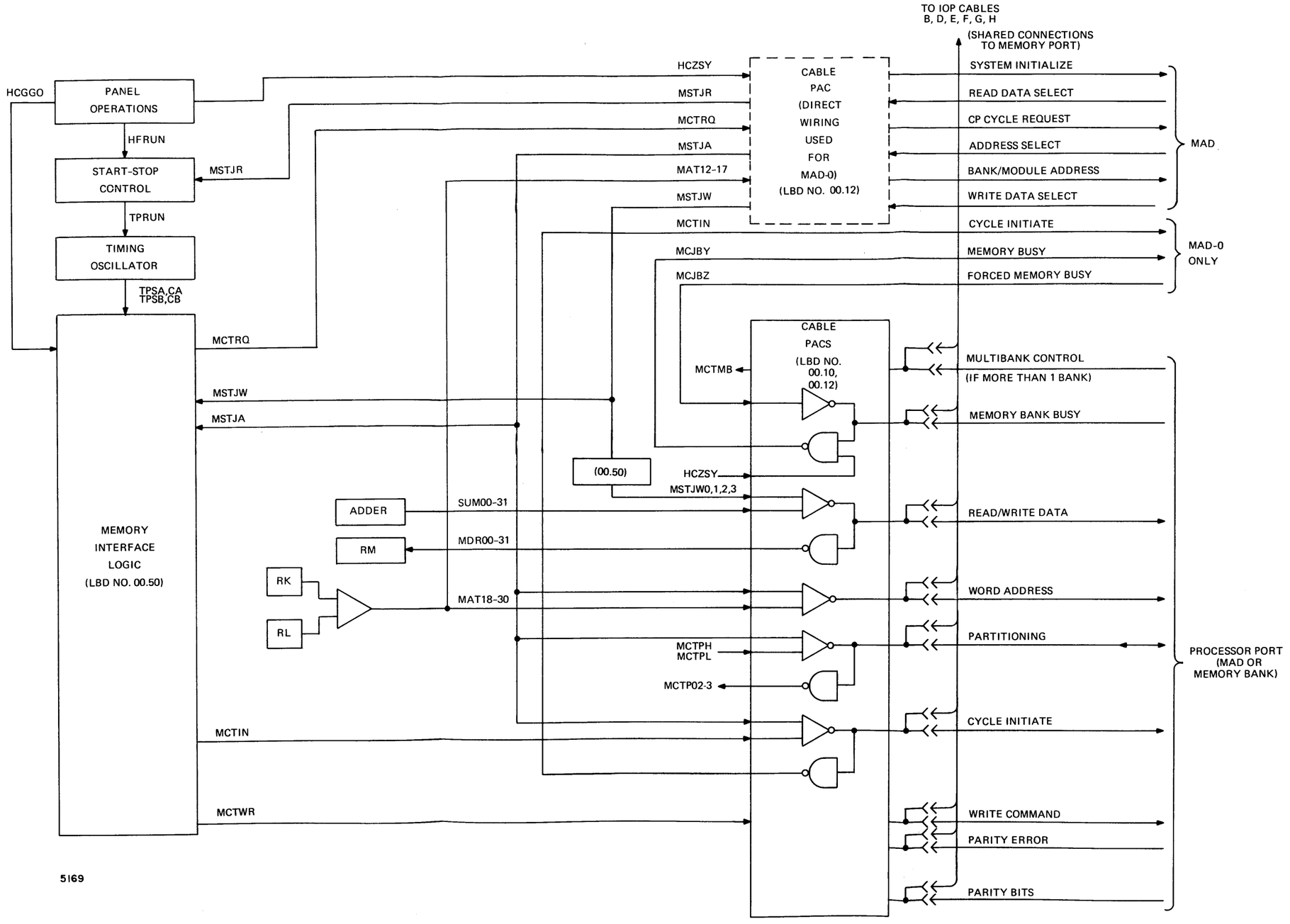


Figure 3-8. Memory Interface Signals

Interface Signals and Cabling

The CP supplies the bank/module portion of the memory address and a cycle request to MAD, which responds with address and data select signals when the CP is selected for a memory cycle. MAD decodes the module portion of the address and sends the memory bank individual module select lines. MAD also checks the address and generates a "forced memory busy" when an illegal bank or nonexistent module address is present. The forced busy allows the memory/MAD/processor cycle to proceed until a legal bank and existing module are specified.

In a system containing MAD-0, the CP/MAD-0 interface connections are direct-wired; no cables are required. Systems using a larger version of MAD (capable of handling more than two processors or more than one memory bank) use cabled connections from CP to MAD.

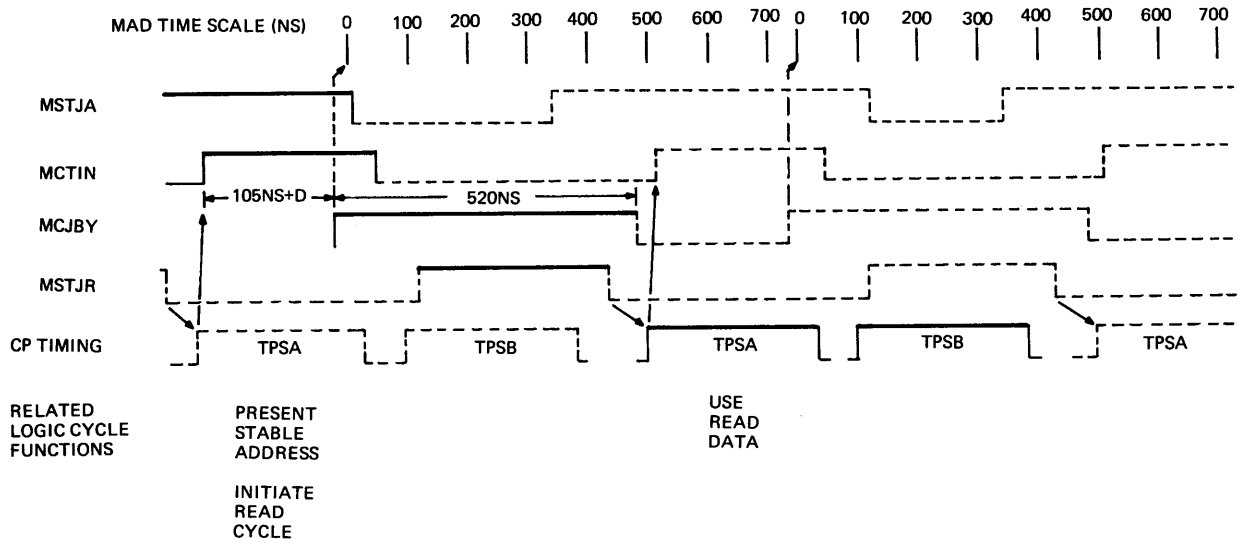
Two processors (usually a CP and an IOP) share a processor port through common cables. Both processors are capable of placing data, addresses, and control signals on the lines, or reading data from memory. MAD referees conflicts between the two processors and allows only one to use a memory port during each cycle. In systems using MAD-0, the processor port cables go directly to the memory bank; in larger systems with more than one memory bank, the shared processor port cables go to MAD for switching from bank to bank. Signals on the processor port connections include the word address, read/write data, write command, cycle initiate, partitioning control, memory busy, and optional parity signals.

CP/MAD/IOP/Memory Handshake

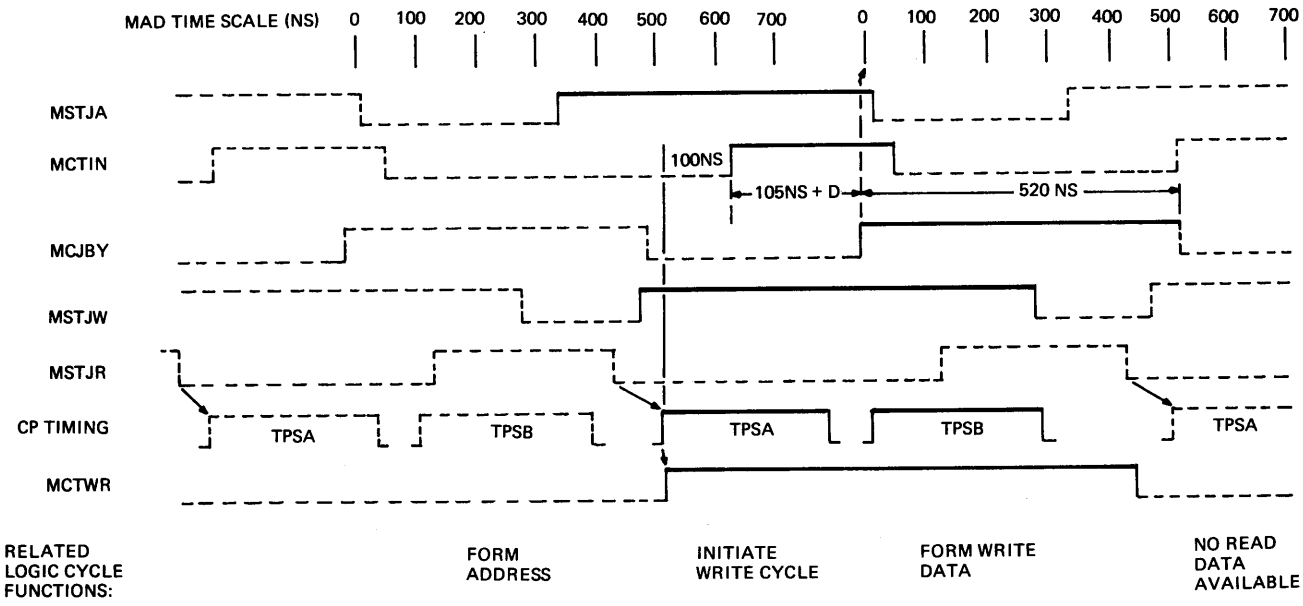
In a typical system the CP competes with the IOP for the use of a processor port. The MAD (Memory Access Director), oversees the process and makes sure that only one processor at a time is using a given port or a given memory bank.

When the CP has access to a processor port, there is a three-way "handshake" among the CP, MAD, and the memory. After a CP inactive period or a system initialize, the cycle starts when the MPC sends the CP an active process code that differs from the current CP process code in the IAF register. The CP responds by turning on its cycle request to MAD. MAD returns an address select, which prepares the CP to generate cycle initiate (after a system initialize, the first cycle initiate is postponed until the clock is started from the panel). Then, both MAD and the CP wait for the memory to respond with memory busy, which indicates that a memory cycle is underway.

Memory and Logic Cycle Timing -- Relative timing of the interface signals during two consecutive cycles is illustrated in Figure 3-9. Since the memory specifications guarantee that read data will be available at a specified time after memory busy, MAD generates a read data select at that time, to gate read data to the CPs RM register. At the end of read data select, the CP (conditional on receipt of a second address select) generates another cycle initiate and enables one set of pulses from the timing oscillator. The current timing level of the algorithm in effect is executed. MAD independently generates another address select, for the next cycle, and both MAD and the CP wait for memory busy.



A. READ CYCLE



NOTE: D = GATE AND CABLE DELAYS FOR MCTIN FROM CP TO MEMORY AND FOR MCJBY FROM MEMORY TO MAD.

B. WRITE CYCLE

Figure 3-9. Overall Memory Interface Timing

MAD also generates a write data select shortly after address select (Figure 3-9). Under control of the algorithm the CP may send memory a write command, which causes the memory to store the data gated to the memory by write data select. During a write operation, cycle initiate is delayed for about 100 ns in order to allow write data, gated through the adder, to stabilize.

CP Logic Functions Related to Memory Cycles. -- Figure 3-10 illustrates a typical series of overlapped memory and logic cycles. Memory busy (MCJBY) represents the memory cycle, and the CP timing pulses (TPSA, B) represent the CP logic cycle. Memory and logic cycles are correlated in terms of read data; logic cycle A can use the read data from memory cycle A, etc.

Read data is gated into the CPs RM register by MSTJR toward the end of each read cycle. The data cannot be used until the next logic cycle. Thus, the data at the address formed during logic cycle A and applied to memory during logic cycle B is available for use in TM during logic cycle C, etc.

Write commands are formed early in the logic cycle that initiates a write memory cycle (see logic cycle E). Write data is gated through the adder during the same logic cycle and is stable during timing level B; for this reason, a write cycle initiate is delayed for about 100 ns to allow the write data to stabilize.

During each cycle the CP supplies an address from the RKL network either from the I-sequence address register (RK) or the J-sequence address register (RL). Usually the RKL network selects RK and RL alternately, but a number of special conditions govern the alternation. A detailed discussion of these conditions and the switching logic (LMAHL) appears in the paragraph on Look-Ahead. From the standpoint of the overall memory/MAD/CP cycle, it is unimportant whether the address is obtained from RK or RL. Because the address is required early in each logic cycle it must be formed and stable in RKL before a memory cycle is initiated. As a result, the address for memory cycle C is formed during logic cycle A, etc.

IOP Break-In, CP Inactive Condition, and Panel Halts. -- If the IOP requests a memory cycle while the CP is running, MAD prevents the CP from generating cycle initiate (by withholding address select), stops the CP clock (by withholding read data select) and generates address and data select signals for IOP. The latter communicates with memory over the interface lines shared with the CP through the cables.

When the MPC sends the CP the inactive process level code, the CP removes its cycle request to MAD after interrupting to the inactive process level, and enters the inactive condition. The IOP retains free access to MAD and memory. If no processor is requesting, MAD idles as well.

If the CP enters a panel halt the clock is stopped and the CP does not generate cycle initiate. If the halt occurs while a valid process code is in IAF, the CP address select is set and memory is not available to the IOP until the CP is started. However, if the CP is in the inactive condition during the halt, IOP can receive memory cycles.

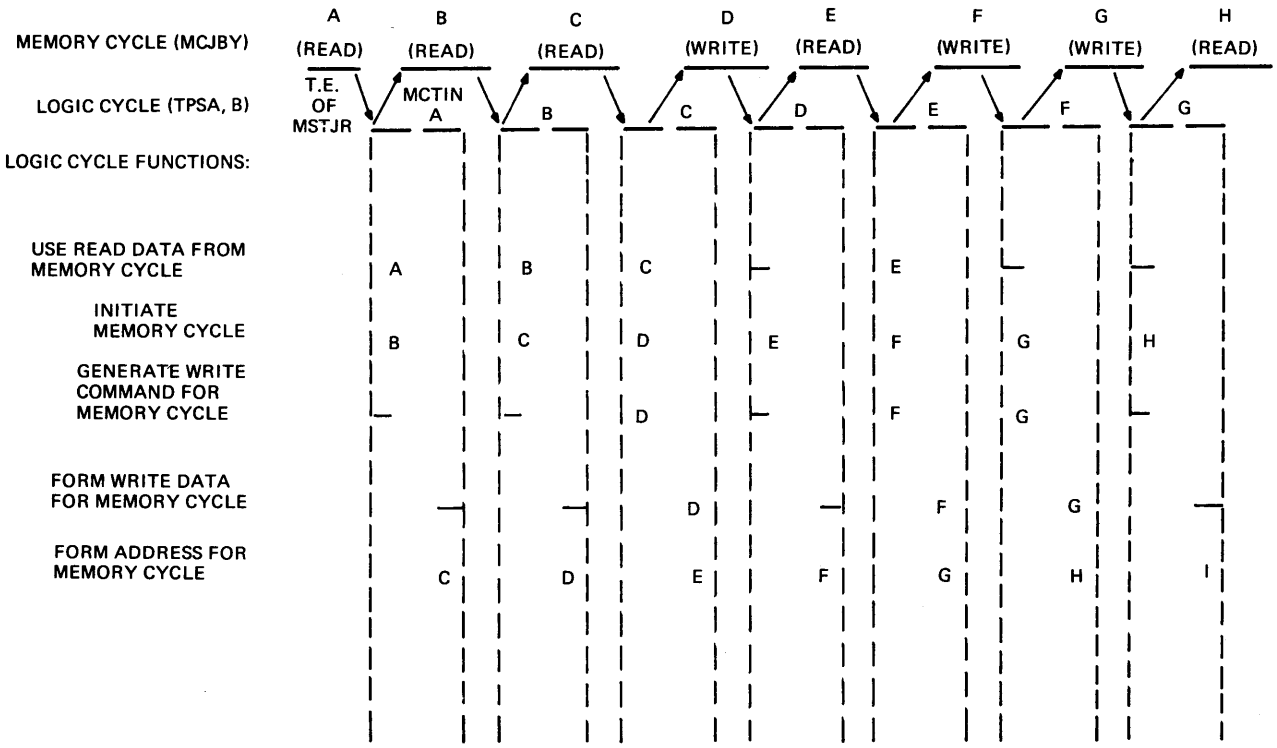
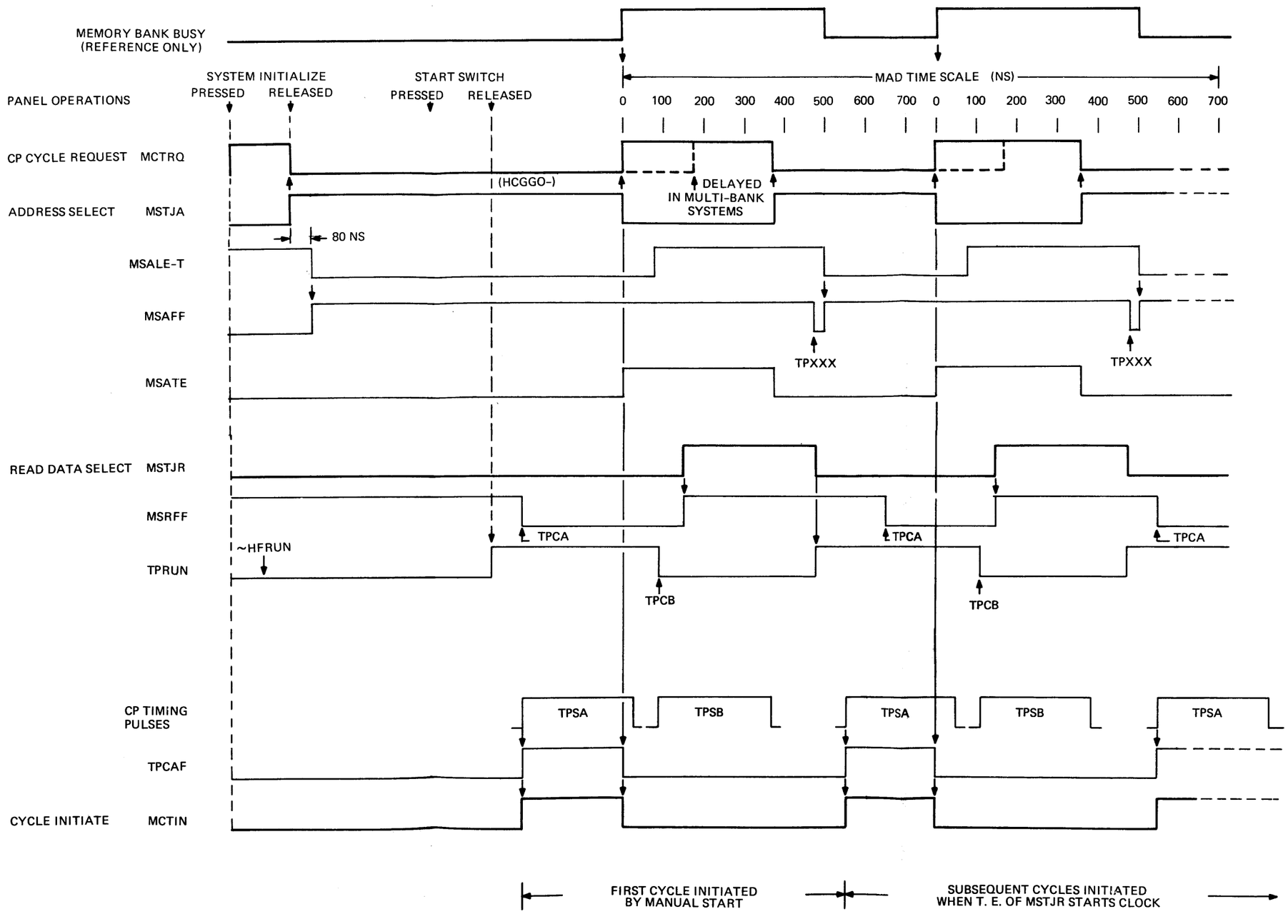


Figure 3-10. CP Logic Functions Related to Memory Cycles

System Initialize and Startup

Startup of the CP from a system initialize and a series of memory cycles are shown in Figure 3-11. Timing oscillator output pulses are shown in relation to the memory interface signals. Relative timing is based on nominal MAD and memory cycle times. The waveforms are idealized; events which are shown as simultaneous are actually staggered in a practical system because of gate and cabling delays. Read cycles are assumed. Write cycles are discussed later.

System Initialize. -- A system initialize (HCZSY) clears MSTJR and MSTJA in MAD and sets MSRFF in the clock control (LBD 00.40). Because TPCAF is cleared, cycle initiate is held off. The CP is waiting only for a manual start to set TPRUN and start the timing oscillator.



5172

Figure 3-11. Memory Interface Logic Signals During System Initialize and Startup

Startup and Normal Consecutive Cycles. -- Since the system initialize forces code 000011 on the MPC IDL lines and also sets an inactive code (000010) in the IAF register, ~IKAQD sets the CP memory request (MCTRQ). As soon as MAD receives the request it sets address select, which clears the request. When the console START, TRANSFER, FETCH, or STORE button is activated and released, TPRUN is set, enabling the timing oscillator to develop one cycle of timing signals. TPCA then sets TPCAF, which enables cycle initiate.

From the time TPCA enables cycle initiate, about 200 ns elapse before the memory busy signal from the memory bank reaches the CP. (The actual time can vary because of cable delays.) Memory busy is, effectively, "time 0" for MAD; address select is cleared and a fixed timing sequence begins. In the CP, ~MSTJA generates MSATE, which clears TPCAF, turning off cycle initiate.

A memory request for the next cycle is set when address select goes false. In multi-bank systems, the request is delayed for about 180 ns to give MAD time to do the necessary bank or processor switching.

During the MAD cycle, read data select for the memory cycle in progress is turned on and off. The leading edge sets MSRFF in the start-stop logic; the trailing edge permits the TPRUN flip-flop to set, provided the panel run flip-flop is still on. In turn, TPRUN enables the next cycle of clock pulses and another cycle initiate.

Cycle Inititate Logic

Logic shown on LBD 00.50 generates the cycle initiate (MCTIN) signal in the correct synchronous relationship with MSTJA and MSTJR from MAD and allows an additional delay during write cycles.

Read Cycles. -- During a read cycle the logic contains provisions to make sure that cycle initiate is held false until both the trailing edge of the previous MSTJR and the delayed leading edge of the new cycle's MSTJA. Normally, MAD generates MSTJA for the new cycle first, so that the effective control for MCTIN is the MSTJR trailing edge. The MSALE-T delay ensures address stability when MSTJA enables the first cycle initiate after an IOP break-in, by withholding cycle initiate until the address gated through the CP/memory cable PACs is stable.

The basic control for MCTIN is the TPCAF flip-flop; the flip-flop is set by TPCAX, which is initiated by the end of MSTJR. (TPCAX is also applied directly to the MCINR gate to produce MCTIN without the flip-flop turn-on delay.) The MCINW gate is disabled during read cycles (MCTWR is false) and has no effect on MCTIN. If MSALE-T has returned to ground before TPCAX, the MSALE+T gate also has no effect on MCTIN timing. However, if MSTJA for the next cycle is late, the MSAXX+T gate holds MCTIN false until the end of the MSALE-T delay.

The MSATE signal, controlled by the MSAFF flip-flop, occurs on the trailing edge of address select. MSATE clears TPCAF (which, in turn, inhibits cycle initiate) in preparation for the next cycle.

Write Cycles (See Figure 3-12). -- During TPSA, if specified by the algorithm, the MCTWR (write command) flip-flop is set. Because write data is applied to memory through the adder, the cycle initiate is delayed for about 100 ns after TPCA to ensure that the adder output is stable. To do this the MCINW gate holds MCTIN false until the end of the TPCAT-T delay. Cycle-initiate is turned off as usual when MSATE clears TPCAF. The MCTWR flip-flop is cleared by the special TPXXX output of the clock oscillator PAC in order to clear the write command sufficiently ahead of the next cycle initiate.

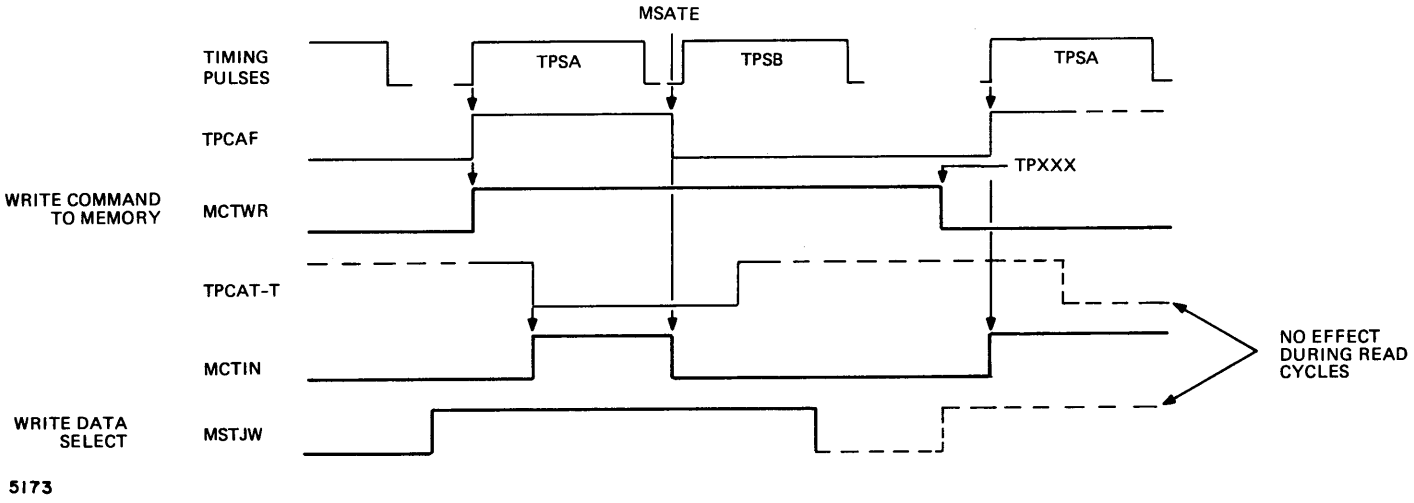


Figure 3-12. Memory Interface Logic Signals (Write Cycle)

IOP Break-In

When the IOP requests a memory cycle, either for a data transfer or to execute a command, it breaks into normal CP running as shown in Figure 3-13. An IOP cycle-request conditions the MAD logic to withhold address-select from the CP and give it to the IOP instead. Because the CP does not receive an address-select, MSALE-T is held high and MCTIN is inhibited even after TPCAF is set. The IOP responds to the MAD address- and read-data-select by controlling the cycle-initiate line to memory through the bus shared by the CP. The memory busy line pulses as usual but under IOP control. The MSAFF flip-flop is held on by the absence of delayed CP address-select (MSALE-T) during IOP cycles. As a result, MSATE is prevented and the TPCAF flip-flop stays set for the duration of IOP control.

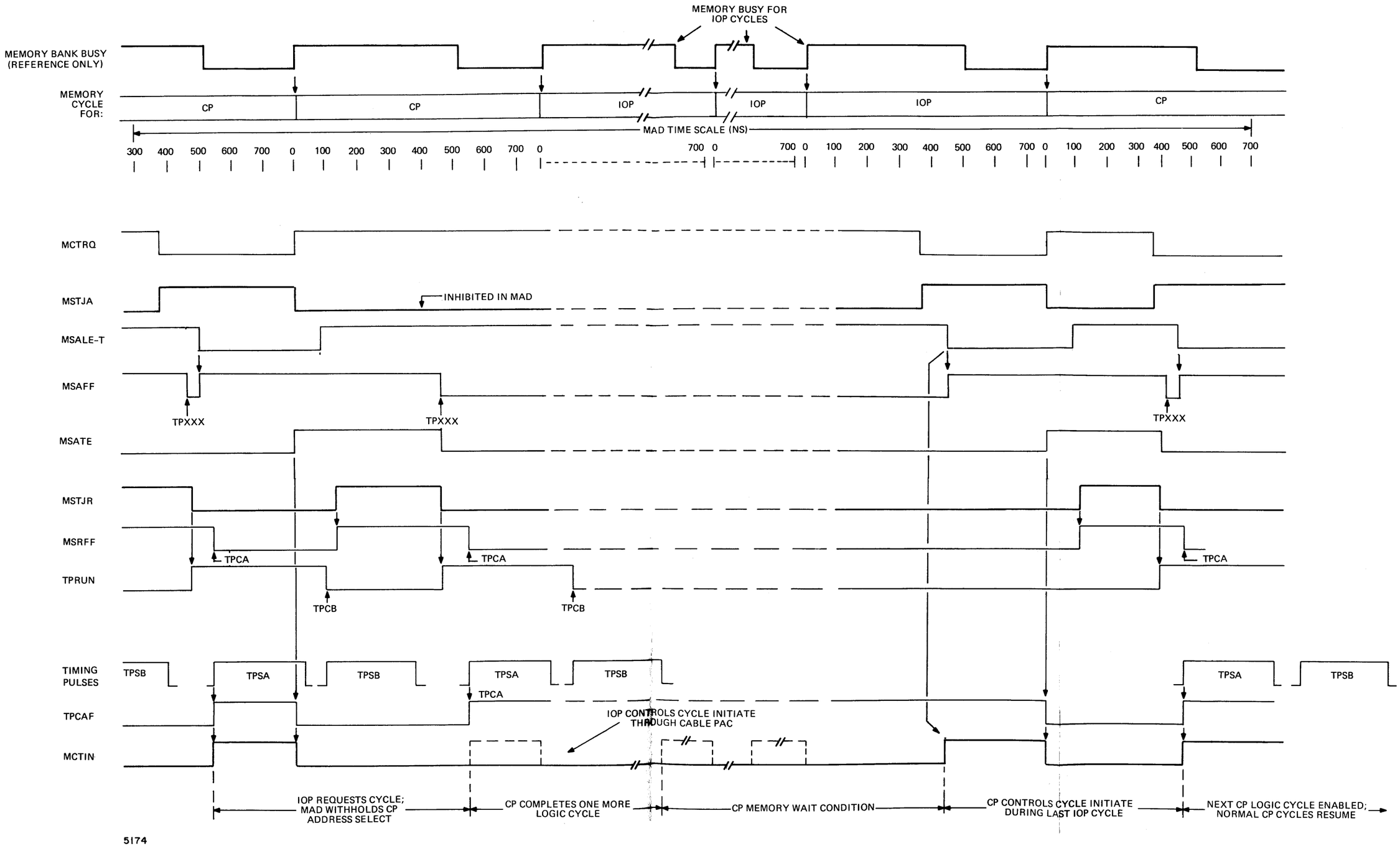


Figure 3-13. Memory Interface Logic Signals During IOP Break-In

The IOP typically requires several consecutive cycles. When it is through with memory, it removes its cycle-request to MAD, which responds with the usual address select to the CP (MSTJA). Because TPCAF is still set, a CP cycle-initiate is generated as soon as MSTJA passes through the MSALE-T delay. Thereafter, the CP resumes normal control and the MSAFF flip-flop and MSATE signal are cycled as usual. While the IOP has control, the CP is in the memory-wait condition.

CP Inactive (Cycle Request Removed) (Figure 3-14)

When the MPC delivers the inactive process-level code on IDL 24-29, the CP enters the interrupt algorithm as usual, storing the statewords of the previously active process and entering the inactive code in the IAF register, negating IKAQV (LBD 06.72). After the interrupt, the CP enters TLF and performs a meaningless fetch; no programs are associated with the inactive process level. During the fetch cycle, the absence of IKAQV permits MCTRQ to be cleared, and inhibits TSOAX (LBD 06.52) to prevent setting the validity bit of the sequence that initiated the fetch. (See paragraph on Look-Ahead.) Because address-select and cycle-initiate are already present, MAD and the CP complete the current cycle except that MAD withholds the next address-select. The end of MSTJR triggers the CP clock for one more logic cycle (also an inactive fetch) but the absence of address-select prevents cycle-initiate. The CP enters the CP inactive condition.

The wait is ended when a program is advanced to the active condition by the MPC. The interrupt register logic detects the inequality between IAF and IDL, negating IKAQD, which sets MCTRQ. MAD responds with address select as soon as a memory cycle is available. (The IOP may be using memory at the time.) Because TPCAF is waiting set, cycle-initiate is generated as soon as address-select passes through the MSALE-T delay.

When the memory responds with memory-busy, MAD begins a CP cycle; read-data-select starts the CP clock and normal consecutive cycles resume. The first algorithm performed after a CP inactive condition is, of course, an interrupt.

The CP can also be started from the panel during CP inactive periods. The HCMFS signal, generated when the FETCH or STORE button is activated, sets the cycle request and MAD responds as shown in Figure 3-14. (After completing the panel algorithm, the CP enters the panel-halt condition.)

Panel Halts

When a halt condition is set up at the panel during normal CP operation, the HFRUN flip-flop in the start-stop logic is cleared during the appropriate timing level (prior to TL1 for step, TLD for access, etc). The effect on the memory signals is shown in Figure 3-15.

Because HFRUN is cleared, the trailing edge of read-data-select does not set TPRUN and start the clock oscillator as usual. No cycle-initiate is generated. The memory completes the cycle in progress, but because there is no cycle-initiate, MAD does not receive memory-busy and so cannot proceed. Since the CPs address select line is on, the IOP cannot break in. This condition (panel halt) lasts until a manual start is performed.

When the panel START, TRANSFER, FETCH, or STORE button is activated and released, the HCGGO signal sets HFRUN. Because MSRFF is set and MSTJR has ended, TPRUN is set immediately. The TPCA clock pulse sets TPCAF, generating cycle-initiate. After the usual wait for memory-busy, normal operation continues.

LOOK-AHEAD

Principles of Operation

To obtain maximum use of memory, the CP processes two instructions at a time, interleaving the fetch, indirect addressing, and execute operations of the instruction pair. The primary object is to present an effective address and access memory for one instruction while a logic cycle is simultaneously preparing the effective address for the other instruction or executing the other instruction.

Assuming for the moment that instructions use only one execute cycle, it can be said that at the beginning of the logic cycle for one instruction a memory cycle for the next instruction is initiated (given certain conditions; refer to paragraphs on Start/Stop and Memory Interface). The result is that each logic cycle is entered with its data (if required) fetched and waiting. It is thus necessary to provide an effective address approximately two logic cycles before the logic cycle which uses the contents of the addressed location.

The alternation of logic cycles between the instruction pair is prohibited during execute operations of more than one cycle. An instruction proceeds through its execute cycles without interruption under control of the particular instruction algorithm. The interleaving of fetch cycles, indirect address cycles and entry to the first execute cycle is under control of the look-ahead logic. The following is a specific example of look-ahead instruction processing. (See Figure 3-16.)

While a CP fetch cycle is forming the effective operand address for instruction 1, instruction 2 is being fetched from memory. The next logic cycle is a fetch cycle which forms the effective operand address for instruction 2 while the operand for instruction 1 is being read from memory. Assuming that the instruction 1 operand is being read from memory, the next logic cycle or cycles are execution cycles for instruction 1. During the last execute cycle of instruction 1, the operand for instruction 2 is read from memory. The next logic cycle is then the first execute cycle for instruction 2, and so on. Note that when an instruction is in its execution phase, there is no alternation of logic cycles between the two instructions.

As instruction 3 (Figure 3-16) requires an indirect address cycle, instruction 4 is ready for execution before instruction 3. This is not permitted as instruction 3 was fetched first and has precedence. Therefore, a dummy cycle (in which no timing levels are set) is entered while the instruction 3 operand is fetched. Instruction 3 then enters its execution phase.

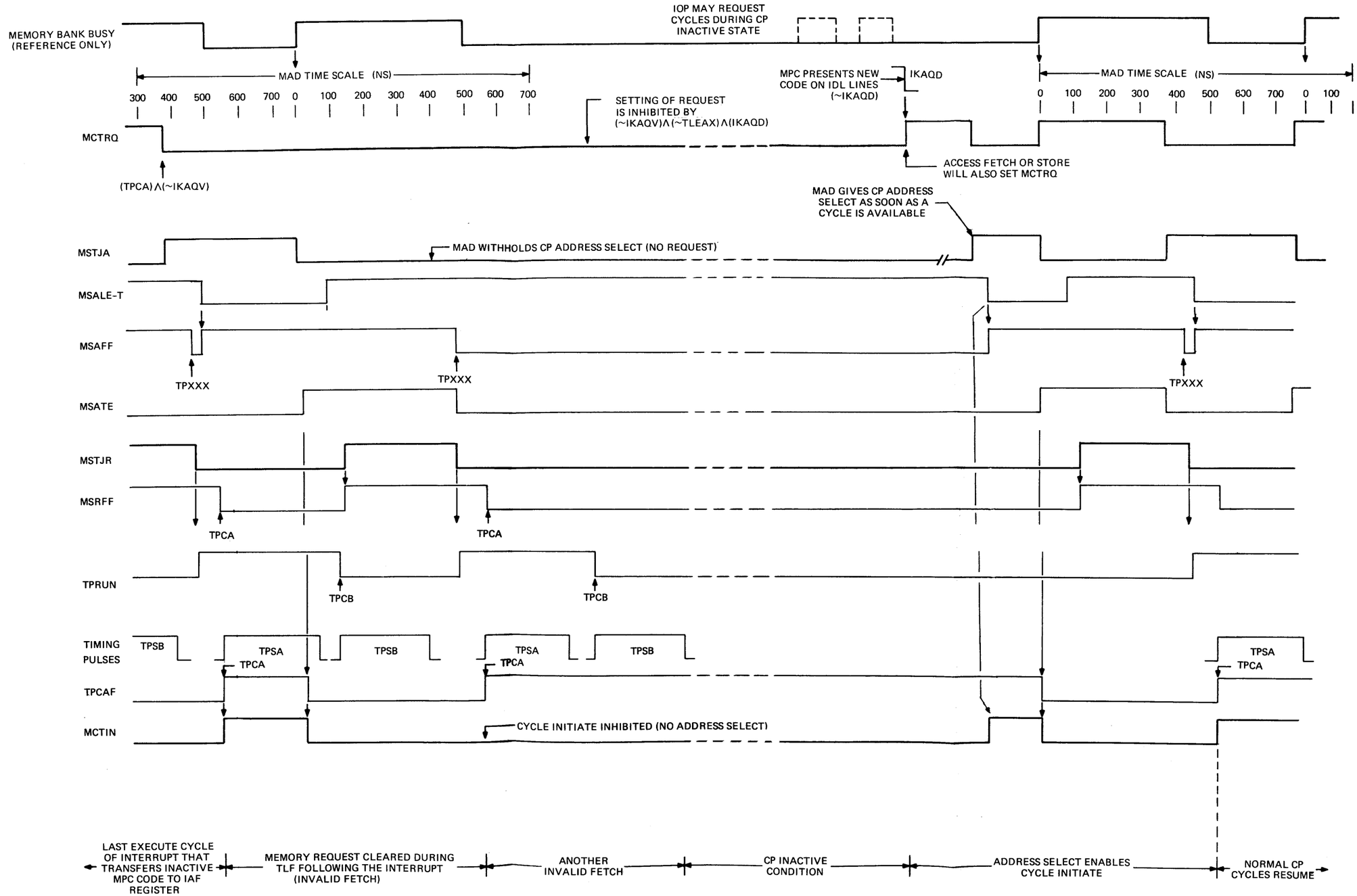


Figure 3-14. Memory Interface Logic Signals During Central Processor Inactive Condition

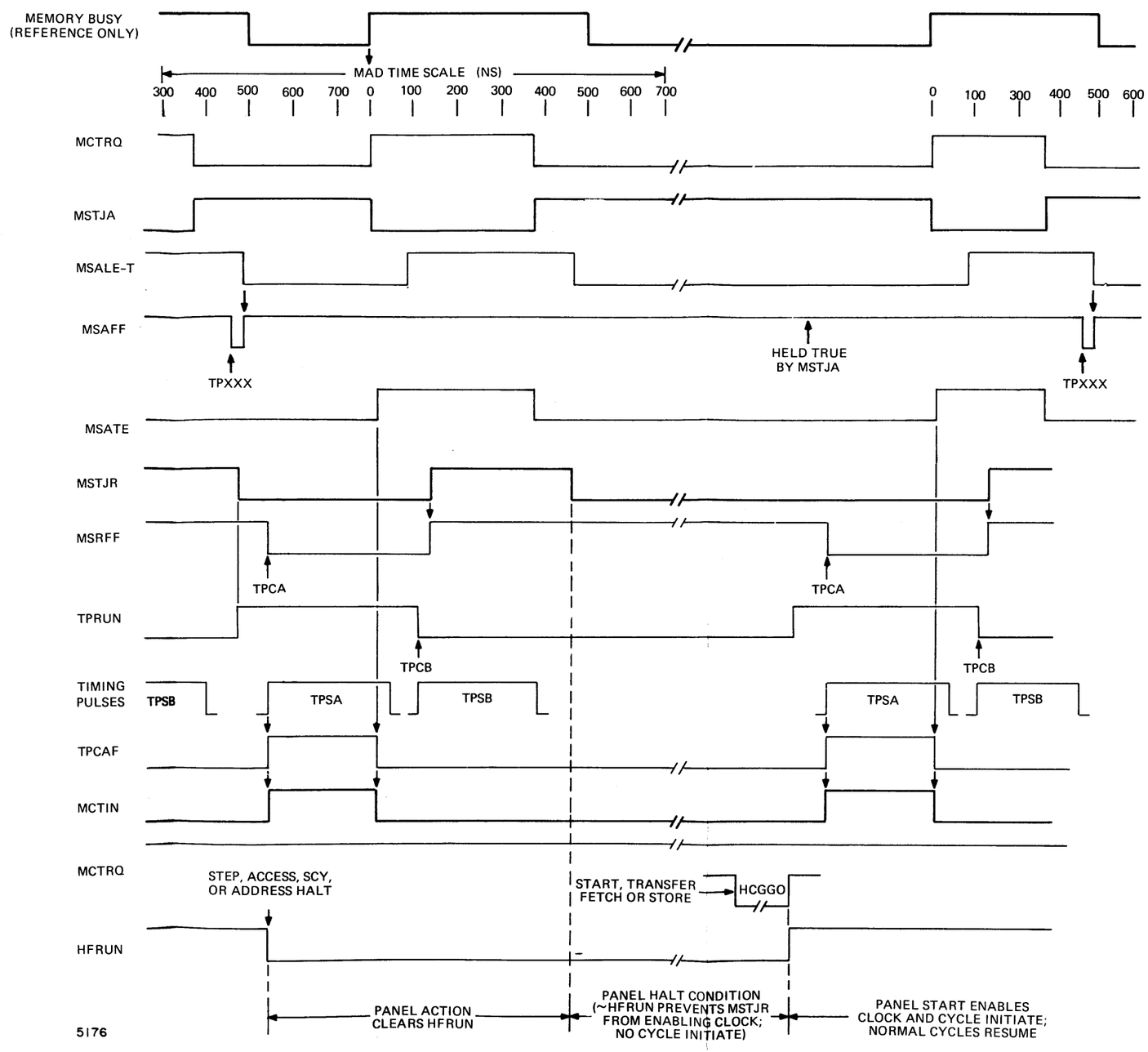
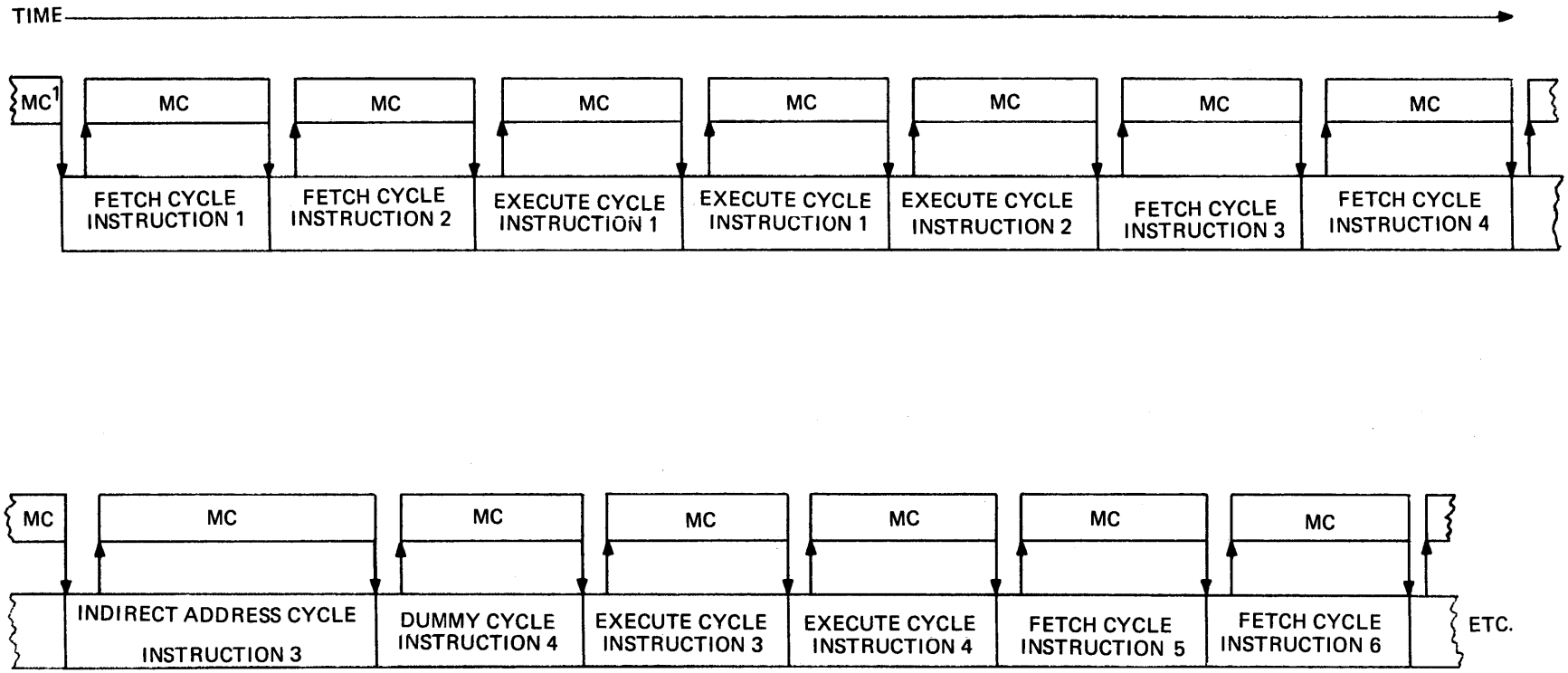


Figure 3-15. Memory Interface Logic Signals During Panel Halt



[1]MC = MEMORY CYCLE

NOTE:

THIS ILLUSTRATION DOES NOT SHOW THE EXACT TIMING RELATIONSHIP BETWEEN MEMORY CYCLES AND LOGIC CYCLES.

5177

Figure 3-16. Instruction Processing Cycle Alternation

I and J Sequences. -- There are two instruction registers, one to hold the current instruction and the other to store the next sequential instruction. The two instruction registers are designated RI and RJ. There are also two address registers, RK and RL. The RI register and RK register are always used together to process an instruction and the instruction processing defined by RI and RK is designated the I-sequence. The RJ and RL registers are similarly used together and the instruction processing defined by RJ and RL is designated the J-sequence.

Sequence Status Controls: TIP, TIC, and TSP. -- Since the CP can hold two instructions at a time and since the processing of the two instructions is interleaved, the CP maintains three controls to reflect the status of the I and J sequences. These controls are TIP, TIC, and TSP.

TIP is true when the I-sequence has precedence, that is, when its instruction is fetched before the J-sequence instruction. TIP is false when the J-sequence has precedence.

TIC is true when the I-sequence instruction "has the clock," that is, when the current logic cycle is processing the I-sequence. TIC is false when the J-sequence has the clock.

TSP is true when the sequence which has precedence also has the clock. It is false when the sequence which has precedence does not have the clock.

Sequence Conflicts: TSO and Validity Indicators. -- Since instruction 2 (for example) can be fetched and have its effective address formed before instruction 1 is executed, the CP checks for the possibility that instruction 1 will modify either the instruction 2 word itself or the instruction 2 indirect-address word or the instruction 2 index value. This possible conflict between the two sequences is checked every time the sequence without precedence fetches its instruction or indirect-address word (i.e., every fetch or indirect-address cycle).

If there is no conflict, the TSO (timing sequence OK) control is true, and the fetch or indirect cycle is permitted to be effective, and RI or RJ validity indicator (RIFVI or RJFVI) is set. If there is a conflict, the sequence without precedence goes through the fetch or indirect-address cycle but fails to set the validity indicator in the first case and does not change the bit in the second case. Furthermore, the CP will not accept the instruction or indirect address word. When a conflict exists, the sequence without precedence is not permitted to accept its instruction or indirect address word (whichever caused the conflict) until the preceding instruction has been executed. By this time any intended modification to the next instruction's instruction word, indirect address word, or index value has been accomplished.

Cycle Entry Decisions. -- As can be seen in Figure 3-16 the CP must decide what type of logic cycle it must enter, and it makes this decision when in the last phase (phase B) of the preceding cycle. Flow chart A in the H632 Central Processor Instructions manual shows this decision-making in detail. Figure 3-17 in this manual is a simplified flow chart of these decisions. The cycle entry decisions illustrated in Figure 3-17 and the sequence conflict description give the general restrictions imposed upon the CP look-ahead feature.

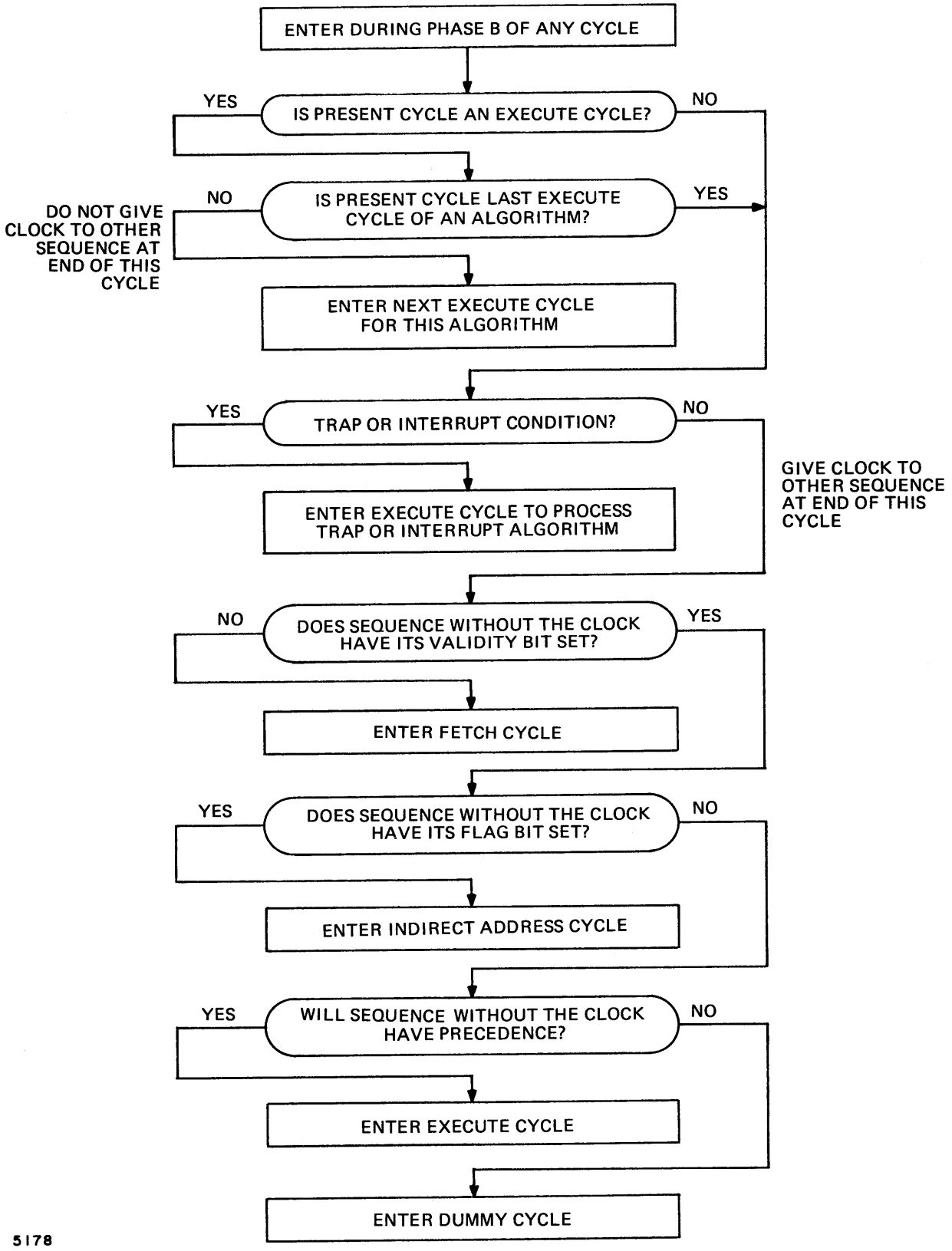


Figure 3-17. Cycle Entry Decisions

Address to Memory Switching. -- The LMARK flip-flop controls the gating of either the contents of the RK or the RL register to the memory address lines. (See Figure 3-18.) Note that it is necessary to supply a memory address two logic cycles before the logic cycle which uses the contents of that address. Unless inhibited, LMARK is complemented each cycle, thus alternately switching the contents of RK and RL to the memory address lines.

Certain algorithms inhibit switching. (This is shown in the H632 Central Processor Instruction manual.) An algorithm inhibits switching, for example, to ensure that the correct address register is on the memory lines for a write cycle used by the algorithm. An algorithm also will inhibit switching to ensure that, in its second-to-last execute cycle, the address register of the sequence without the clock is gated to the memory address lines.

If an instruction uses the memory cycle which the instruction initiates in its first execute cycle (TL1), the address switching must be inhibited before the instruction algorithm itself can possibly do so. This inhibition is controlled by the RIFMM or RJFMM flip-flop (associated with the I and J sequence instruction registers, respectively). These flip-flops are op-code sensitive, and a certain group of op codes will enable the setting of RIFMM or RJFMM when such an instruction is fetched. Although all of the instructions which set the IRFMM or RJFMM flip-flop do not of necessity require switching inhibition, this is of no consequence, for the instruction algorithm can then effect switching as required. The set condition of RIFMM or RJFMM will cause switching to be inhibited in the logic cycle prior to the first execute cycle of the instruction.

Figure 3-18 shows the address switching during the processing of several instructions. The STW instruction (Store Word, op code 1E) stores data in its first logic cycle. Address switching is inhibited during the preceding J-sequence fetch cycle (caused by RIFMM being set), so that RK is specifying the memory address during TL1 of the STW instruction when this instruction is storing a word in memory.

Instruction Register Input and Output Gating. -- The input gating to the RI and RJ instruction registers is controlled by TIC. TIC ensures that when an instruction is fetched the instruction is gated into the instruction register of the sequence which has the clock.

The output gating of instruction register bits 4 through 14 is controlled by TIP. TIP ensures that only the register and op code fields of the instruction with precedence are gated out into the CP logic. The output gating of the instruction register validity bit, flag bit, and MM bit is controlled by TIC. These bits are used in look-ahead control. TIC controls the output gating of these bits so that the sequence which has the clock, when it is preparing to give the clock to the other sequence, can observe these bits of the sequence without the clock and set the correct type of logic cycle for the other sequence.

Address Register Input and Output Gating. -- The input gating to the RK and RL address registers is controlled by TIC. Normally, TIC steers the address into the address register of the sequence which has the clock. Exceptions are those algorithms which load a new address into the address register of the sequence without the clock, for example, the interrupt, trap, and jump algorithms.

The output gating of the address registers to memory is controlled by LMARK and has been previously discussed.

For various reasons, a sequence must at times examine its own address register contents; for example, when forming an effective address in a fetch or indirect address cycle, when the effective address is a general register, and during a shift instruction. TIC therefore controls the gating of the address register contents (RKL) into the CP logic.

Detailed Theory of Operation

TIP, TIC, and TSP Controls. -- These controls specify which sequence an instruction is assigned to, control the cycle entry decision, and effect sundry other functions. The operation of these controls, their setting and clearing, is shown in Figure 3-19. (See LBD 06.50.)

TSO Controls. -- The TSO controls (TSOAX, TSOAY, and TSOBF) must be true to allow an effective fetch or indirect address cycle to occur. There are seven conditions which, if not met, disable the TSO control signals. (See LBD 06.52.) These conditions are represented by seven gates whose function names are: TSOWR, TSOXR, TSOWN, TSOXN, TSOHI, TSOIV, and TSOOP. If the output signal of any one of these gates is false TSOAX or TSOAY will be false, which condition will make A++FC (LBD 01.04) false. With A++FC false, CUIRW (LBD 06.50) is disabled and an effective or valid fetch or indirect address cycle will not occur.

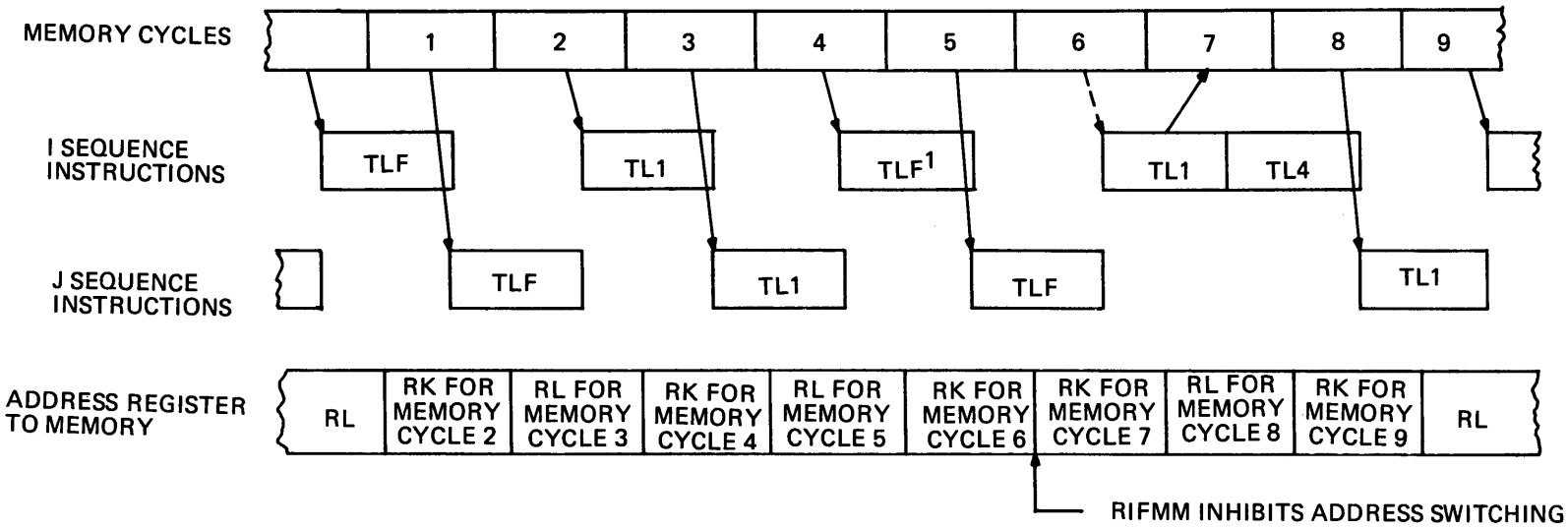
a. TSOWR is false if the address from which the next instruction or indirect address word is being fetched is the same general register specified by the R-field of the instruction with precedence.

b. TSOXR is false if the X-field of the nonprecedent instruction with the clock specifies the same general register as the R-field of the instruction with precedence. DXRQA prevents the indication of a conflict if the X-field content is zero.

c. TSOWN is false if the next instruction or indirect address word is being fetched from the same general register specified by the precedent instruction's R-field plus one, when the precedent instruction uses the register specified by R-field plus one as an operand (DIRNA).

d. TSOXN is false if the X-field of the nonprecedent instruction with the clock specifies the same general register as the precedent instruction's R-field plus one and if the precedent instruction uses the register specified by the R-field plus one as an operand (DIRNA). DXRQA prevents the indication of a conflict if the X-field content is zero.

e. TSOHI is false if the look-ahead switch is in the inhibit position, (~HCMEL) $\wedge$ (HCMIL). If the look-ahead switch is in mid-position, the single-cycle or step mode (HCMCS) or single-cycle, repeat or memory access modes (HCMCR) will make TSOHI false. When in the enable position, (HCMEL) $\wedge$ (~HCMIL), TSOHI is false only in the memory access mode, (HCMAC) $\wedge$ (HCMCR).



ARROWS INDICATE DIRECTION OF MEMORY DATA TRANSFERS.

[1]THE INSTRUCTION FETCHED IS AN STW INSTRUCTION (OP CODE 1E) WHICH IN THIS EXAMPLE WOULD SET THE RIFMM FLIP-FLOP IN ITS FETCH CYCLE. TL1 OF THE STW INSTRUCTION REQUIRES A MEMORY WRITE CYCLE.

5179

Figure 3-18. Memory Address Switching Principles

f. TSOIV is false if an inactive process code is detected in a fetch cycle.

NOTE

This function is not a look-ahead function; it is used here as a convenient means for aborting the fetching of instructions when the CP is inactive (idling).

g. TSOOP is false if the instruction with precedence is a store instruction. No attempt is made to anticipate whether the effective address of the store instruction (which may not yet have been formed) will cause modification of the nonprecedent instruction.

Cycle Entry Decisions (See LBD 00.60). -- The progression of execute cycles is under particular control of each algorithm. Entry to a fetch cycle (TLF), indirect address cycle (TLI), the first executive cycle (TL1), or a dummy cycle (no timing level set) is under control of the look-ahead logic.

The TPSBE signal is generated by TPSBX when the sequence with the clock is going to give the clock to the other sequence. The rules for giving the clock away can be deduced from the input to TPSBE. Gate TCYEP enables the giving away of the clock at the end of an algorithm if a trap condition does not exist. Gate TLEBP enables the giving away of the clock whenever the sequence with the clock is not in an execute cycle.

TPSBE is a TPSB timing pulse which is enabled at the end of each algorithm. Depending upon the requirements of the sequence without the clock and whether it has precedence, TPSBE will appropriately set either TLFAF, TLIAF, TL1AF, or no timing level.

If an interrupt condition does not exist and the sequence without the clock does not have its validity indicator set (RJBVI), the sequence without the clock has not yet successfully fetched its instruction; therefore, TLFAF is set.

If an interrupt condition does not exist and if the sequence without the clock has its validity indicator set and has its flag bit set, an indirect address cycle is required; therefore, TLIAF is set.

The first execute cycle flip-flop (TL1AF) can be set by normal processing requirements (gates TL1AY and TL1AE), a trap condition (gates ITRPF and ITRPG), an interrupt condition (gate TL1AI), or human intervention (gate TLXAZ).

Gate TL1AY sets TL1AF if all of the following conditions prevail: if the sequence with the clock is not in an execute cycle, if the sequence without the clock is ready to execute (its validity indicator is set and its flag bit is not set), and if the sequence without the clock has precedence. (Note that if the sequence without the clock has precedence (TSPBF), the other sequence is not allowed to perform an execute.)

Gate TL1AE sets TL1AF if both of the following conditions prevail: if the sequence with the clock is in its last execute cycle and the sequence without the clock will be ready to execute.

Gate ITRPG sets TL1AF at the end of an instruction if a trap condition has set the ITRAF trap flip-flop in a previous cycle. Gate ITRPF sets TL1AF at the end of an instruction if a trap condition (ITRPR) is setting the trap flip-flop in the present cycle.

Gate TL1AI sets the TL1AF flip-flop on TPSBE if an interrupt condition has occurred.

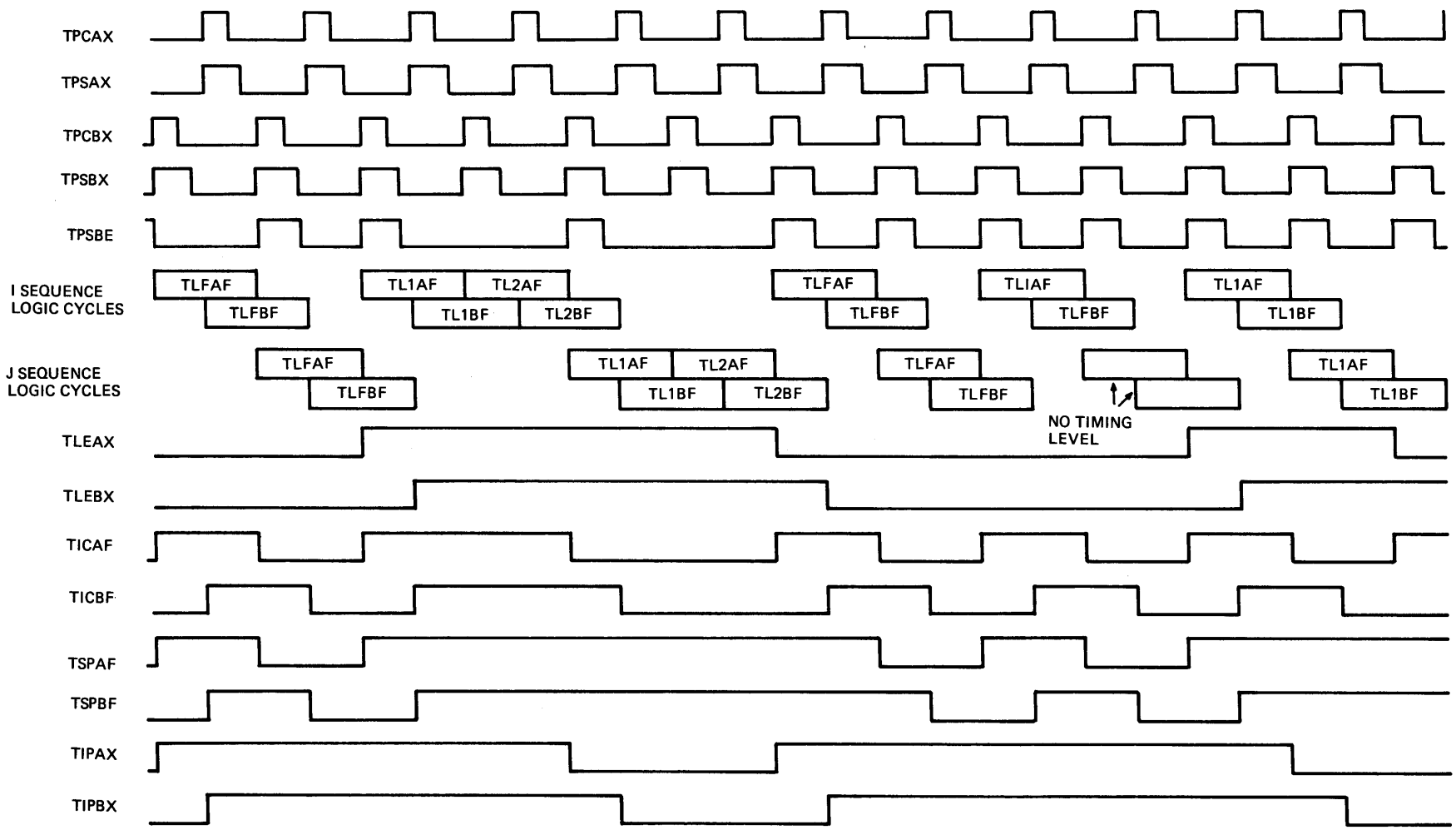
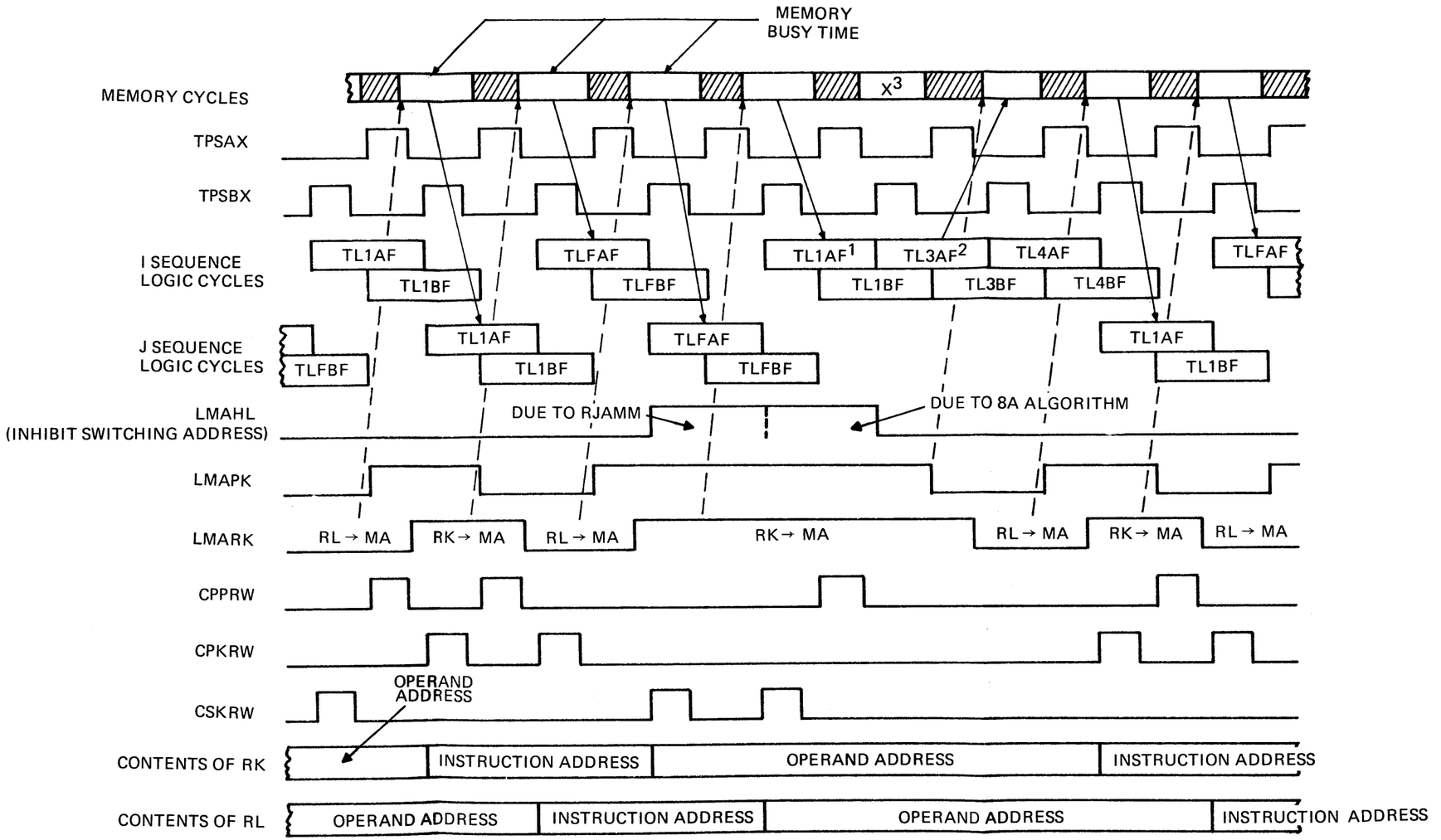


Figure 3-19. Timing for TIC, TSP and TIP Look-Ahead Controls



1. EXCLUSIVE-OR TO STORAGE INSTRUCTION, CASE H (OP CODE 8A)
2. WRITE CYCLE
3. MEMORY CYCLE X IS USED TO ENABLE OSCILLATOR PULSES AND SYNCHRONIZE WRITE CYCLE; IT IS NOT USED FOR DATA.

5181

Figure 3-20. Memory Address Switching Details

Address Switching (See LBD 00.50). -- The normal complementation of LMARK is implemented to accomplish address switching as follows. (See Figure 3-20). The complement of LMARK is stored in LMAPK. LMAPK is then copied into LMARK. Normally, LMAPK is complemented every TPCAX. Then LMAPK is normally copied into LMARK on the trailing edge of TPSAX when memory goes busy and no longer requires the old address [(MSAFF) ∧ (~MSTJA)].

Initially and during the DIJAM algorithm, the LMAPI and LMAPJ gates ensure that LMAPK is set (if the I-sequence has the clock) or cleared (if the J-sequence has the clock). Initially it is these two gates which ensure that LMARK is correctly associated with the address registers.

The inhibition of the normal complementation of LMARK is accomplished by LMAHL (LMARK hold) which inhibits the copying of the complement of LMARK into LMAPK. LMAHL is generated by certain control panel functions, by certain instruction algorithms, or when the set condition RIFMM or RJFMM is sensed at certain times. RIFMM or RJFMM (LBD 03.44) is set when a valid fetch cycle is underway and one of the MM group of op codes is sensed. The op codes in this group are 10, 12, 16, 30, 32, and 36, and all op codes ending in A or E (mostly storage operations).

As the switching inhibition caused by RIFMM or RJFMM is accomplished by the sequence with the clock on behalf of the sequence without the clock, the set condition of one of these flip-flops is steered to RJAMM by TIC. RJAMM represents the MM bit of the sequence without the clock.

At certain times, RJAMM generates ABT1M (LBD 01.02) which generates LMAHL. Note that RJAMM, in addition to indicating an MM instruction, also indicates that a valid fetch was accomplished.

Gate A+++F generates LMAHL if the following conditions exist: the sequence without the clock has precedence (~TSPAF), has fetched an MM instruction (RJAMM), and has satisfied any indirect addressing requirements (~RJA00); the sequence with the clock is in its last execution cycle (TLZAX), and is not going to clear the instruction register of the sequence without the clock (~CZJR0).

Gate ACJ3B generates LMAHL during jump instructions (DICJA) and ensures that if the jump condition is not met the address switching is properly set up for the MM instructions. If the jump condition is met, LMAHL is inhibited in TL4AF, thus ensuring that the address register of the sequence with the clock is supplying the address for the logic cycle after the next. Note that at the end of a jump the next cycle is a fetch, after which the clock will be given back to the sequence which processed the jump.

INSTRUCTION COUNTER

RP register stages 04 through 30 (LBD 05.30 through 05.42) also called the "instruction counter," address the storage location of the next instruction to be fetched, interpreted and executed. The contents are incremented during the TL1A execution phase of all algorithms except interrupt, trap, console access fetch/store, transfer, LNJ, JMP, XEC, and LNX. The following algorithms provide for incrementing at times other than

TL1A: transfer, interrupt, trap, bit operations (skips), LNJ, conditional jumps, and load and swap program stateword.

The instruction counter consists of master rank flip-flops (RPF04-30), slave rank flip-flops (RPA04-30); master-to-slave and slave-to-master copy gates, a carry speedup network, and input gates that permit the contents of address register RK or RL (RKL) to be copied into the master rank.

Output of the instruction counter is gated to one of the address registers (controlled by CPK, and selected by the gating signals LRKL) or to the adder by the control level LSRPI.

Loading RP

The RP register is loaded from RKL during TPSA of certain algorithms. CZPPI clears the register, and CKP loads ones from RKL. (During TPSB of the same timing level, the CPPIA signal transfers a function of RPF into the RPA slave register.)

Up-Counter Algorithm

The instruction counter operates as a binary up-counter in a two-step process based on simple rules for incrementing binary numbers. To increment a number, set the least significant stage which contained zero, and clear any less significant stages. See examples below:

	Example 1	Example 2
Initial value →	XXXXX0	XXX011
Set least significant zero →	1	1
Clear less significant ones→		00
Leave more significant stages unchanged →	XXXXX1	XXX100

Slave Rank Setup. -- The following example illustrates a case where the three low-order stages of RPF contain 011 as shown in the starting condition of Figure 3-21. Slave rank setup takes place during every TPSB. (See LBD 05.42.)

a. CZPPA attempts to clear all RPA stages. Since RPF28 contains the first zero, RPA30 and RPA29 are cleared. (If PRF contained more low-order ones, all of them would cause corresponding RPA stages to be cleared.)

b. RPF28 contains the least significant zero. This enables CPPIA to set RPA28 (via gate RPP28).

c. The RPG27 carry-speedup gate detects the fact that the low-order zero has appeared in this group of four gates. The RPG27 signal is applied to RPA stages 23 through 26, setting them all. Other carry-speedup gates detect ones in this group and rapidly propagate ones into all higher stages of the RPA register. Setup of the slave rank is now complete.

Incrementing RPF. -- The instruction counter itself remains unchanged during and after the slave-rank setup. When the counter is to be incremented (usually during TL1A of

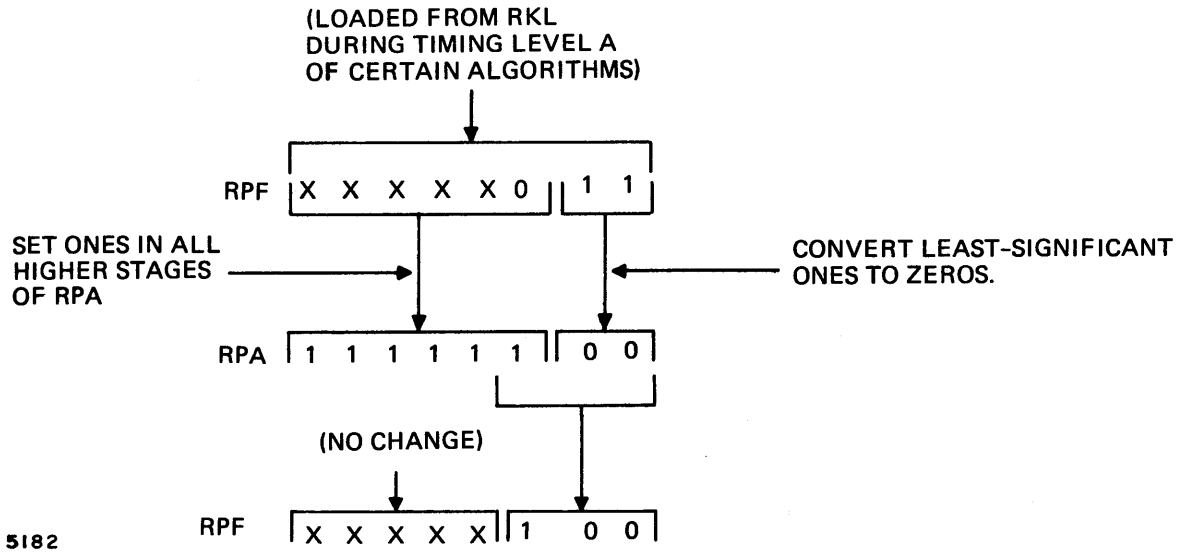


Figure 3-21. Instruction Counter Operation

each algorithm), the CPPIW signal operates as follows (example 2 in Figure 3-21). (See LBD 05.42.)

a. Since RPA29 and 30 contain zeros, RPF29 and 30 are cleared.

b. Since RPA28 contains the first one, RPF28 is set. (Gate RPY28 is enabled by the fact that RPA29 contains a zero.)

c. RPA27 contains a one, but because RPA28 does also, the RPY27 gate is inhibited and RPF27 is unchanged. Since all higher stages of RPA contain ones, corresponding stages of RPF are unchanged also. Incrementing is complete.

In each group of four stages the low-order RPYXX gate is inhibited by the carry-speedup input (RPG) from the preceding group rather than RPA of the preceding stage. (RPG27 inhibits RPY26 on LBD 05.40, for example.)

CYCLE COUNTER (See LBD 00.66.)

The cycle counter is used to count cycle iterations for the following algorithms:

a. Interval compares (op codes 12, 16, 32, 36)
b. Multiplies (op codes D4, D8, DC)
c. Divides (op codes F4, F8, FC)
d. Load and store registers (op codes 6A, 7A)
e. Load and store statewords (op codes 10, 30 with R=0)
f. Trap
g. Interrupt
h. Fill

Structure

The cycle counter is a five-stage double-rank down-counter. It consists of the following components:

a. Master rank flip-flops ($TCT_{27\rightarrow31}$) whose contents indicate the number of remaining cycles.

b. Slave rank flip-flops ($TCA_{27\rightarrow31}$)

c. Master and slave flip-flop set and clear gates used in implementing the decrement function.

d. Four gating structures to preset the counter to a desired count. They are defined as follows:

1. TCYY is pulsed by TPCA and attempts to preset the master rank to a value of 31_{10}. The remaining preset gates (whichever is selected) clear the appropriate master rank stages on TPSA to preset the counter to a value less than 31_{10}.

2. TC01 is used by the interval compare, load and store program stateword, interrupt and trap algorithms. TC01 presets the counter to 1.

3. TCML is used by the multiply and fill algorithms to preset the master rank to 7.

4. TCIR is used by the load and store registers algorithms. TCIR presets the counter to one less than the number of registers to be loaded or whose contents are to be stored.

e. Gating structure for developing the decrementing function, TCDC. The algorithms which use the cycle counter control decrementation by enabling TCDC. As long as the counter contents are not equal to zero and while TCDCR is true, each TPSA pulse generates TCDCI, which decrements the counter by one.

Theory

Initially, the cycle counter is preset by TCYY alone or TCYY and one of the other preset signals. Depending on the algorithm, the counter is initially preset to 31_{10} (divide algorithm only), 7, 1, or one less than the number of registers to be transferred.

Note that TCYY occurs on TPCA and functions to clear all master stages to ones. The other preset signals are used with TPSA to set selected stages to zero.

In phase B the slave rank is set up for the decrementing of the master rank in the next A-phase. Pulse TPCB attempts to clear the slave rank stages to ones, and TPSB sets to zero the first slave rank stage whose associated master was set. All slave stages to the left (more significant stages) of the first stage to be set will also be set. This sets up the slave rank.

When the algorithm has completed an iteration, the decrement pulse is generated to copy the slave rank into the master rank. The slave rank is copied into the master rank up to and including the first slave stage which contains a zero. The changing of all more significant master stages is inhibited. See the following example.

27	28	29	30	31	← Bits
0	1	1	0	0	← Master rank after preset
0	0	0	1	1	← Slave rank after TPSB and TPCB
0	1	0	1	1	← Master rank after TCDC

SHIFT NETWORK

The shift network (Figure 3-22) is one of the primary data transfer structures in the CP. In addition to performing logical and arithmetic shifts and rotations, the network is used during almost every algorithm for the unshifted transfer of data into the G, H, and T registers. Two nearly identical 32-bit structures (the U-net and the V-net) make up the shift network. Each section includes input gating plus three layers of shifting (1-place, 4-place, and 16-place).

The input selection gates and shift network of stages 14 and 15 are representative. (See LBDs 03.40 and 03.42.)

Input Selection

Inputs to the U-net come from the memory register (RM), the 16 general-purpose registers (R0 through RF), the quotient register (RQ), and the PS_2 stateword (RP_{32-63}). The V-net receives the same inputs, but only stages 0, 1, and 3-7 of the stateword are used. In addition, the V-net accepts the address register of the sequence with the clock (RKL). In systems containing the optional byte-manipulation instructions, special inputs are brought into V-net stages 16-31. These inputs, controlled by certain byte-control instructions, are a function of the current content of RKL. The gating terms that select U-net and V-net inputs are shown in Figure 3-22. Some of the notation in the flow charts and analyses for input selection is symbolic.

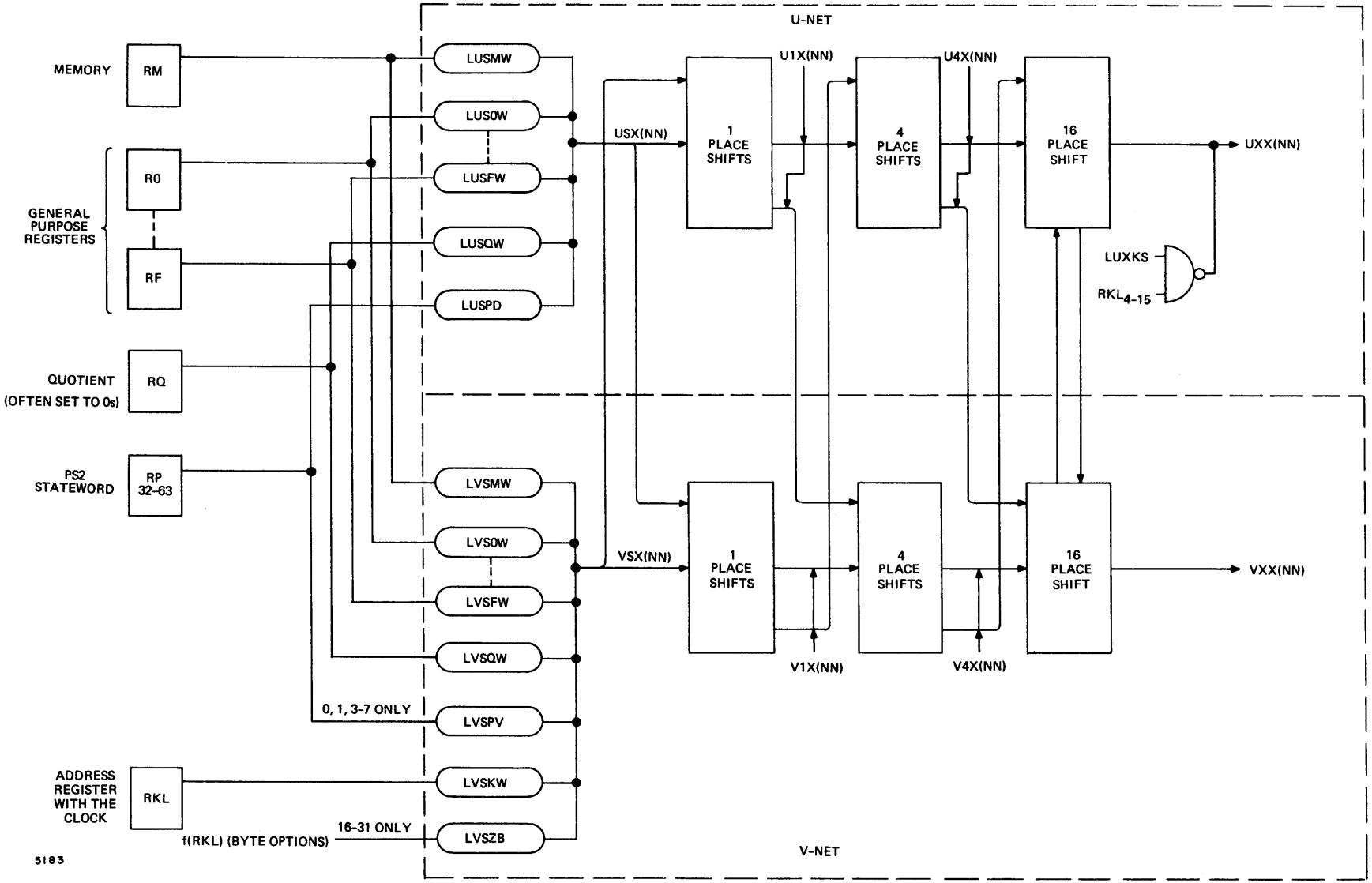


Figure 3-22. Shift Network Block Diagram

Flow Chart Notation	Actual Gating Term
LUSRW	LUS(R)W where R = (RIA04-07)
LUSNW	LUS(N)W where N = 1+R
LUSXW	LUS(X)W where X = (USX01-03)

LUSWW	If (RKL28-31) > 15: LUSMW
	If (RKL28-31) < 15: LUS(W)W
	where W = ($\overline{R}$KL28-31)

(The preceding examples are for the U-net only; V-net input gating is similar.)

When two or more inputs are gated to one of the networks at the same time, the logical product function (A∧B) is performed.

The selected input (or inputs) appear on the USX(NN) and VSX(NN) lines, unshifted. If no input is selected, the network delivers an output of all ones.

In many algorithms, RQ is gated to a shift net input after it is cleared; this effectively applies all zeros to the shift net.

One-Place Shift Gates (See Figure 3-23.)

The one-place shift gates receive inputs from the input selection gates (USX-- and VSX--) and rotate the input zero, one, two, or three places to the right. Vacated stages of the U-net are filled by selected V-net inputs, and vacated stages of the V-net receive selected U-net inputs.

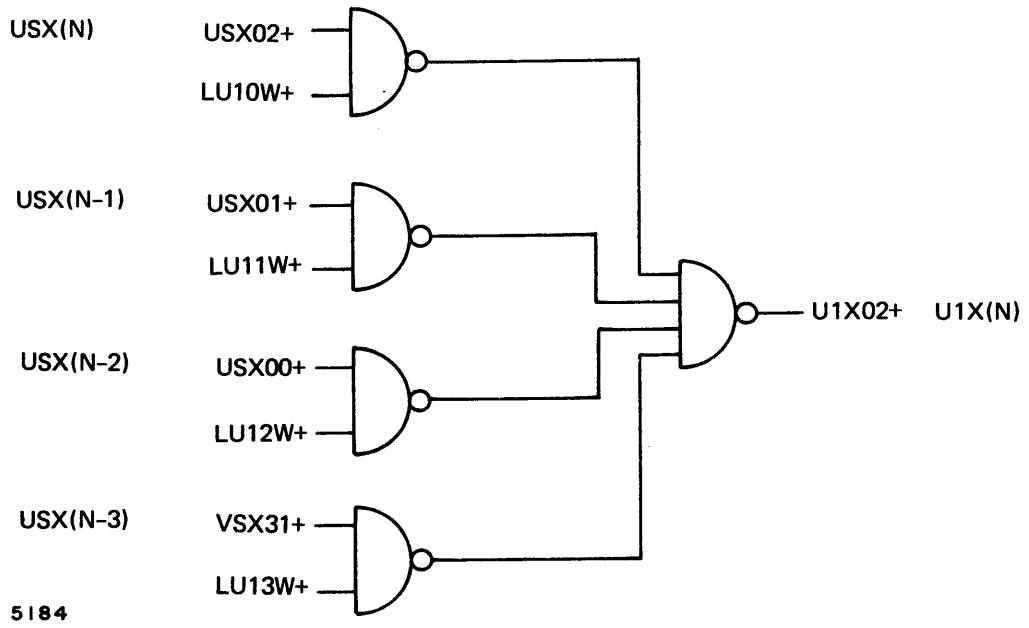


Figure 3-23. One-Place Shift Gates (Typical)

Four-Place Shift Gates (See Figure 3-24.)

The 4-place shift gates receive inputs from the 1-place gates (U1X-- and V1X--) and rotate the input 0, 4, 8, or 12 places to the right. Vacated stages of the U-net are filled from the V-net one-place shift outputs, and vacated stages of the V-net are filled from the U-net one-place outputs.

Sixteen-Place Shifting (See Figure 3-25.)

The 16-place shift gates operate much like the 1- and 4-place gates. By selection of gating terms, the 16-place shift gates can perform straight-through transfer or 16-place right rotations of full words or halfwords. Selection of gating terms is controlled by the algorithms.

Normally, unless shifting is specified, the LUH0H, LUH0L, LVH0H, and LVH0L gating terms are true, and the full 32-bit outputs of the U and V 4-place shift gates are transferred straight through to the UXX and VXX lines. The LUHYH and LUHYL signals inhibit LUH0H and LUH0L, respectively, to force ones in either halfword of the U-net output. LVHYH and LVHYL similarly force ones in either half of the V-net output.

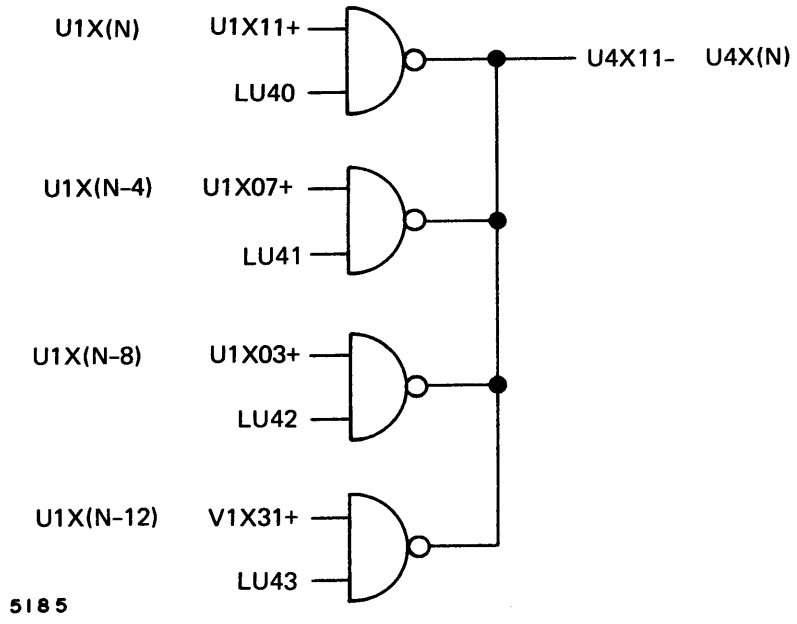


Figure 3-24. Four-Place Shift Gates (Typical)

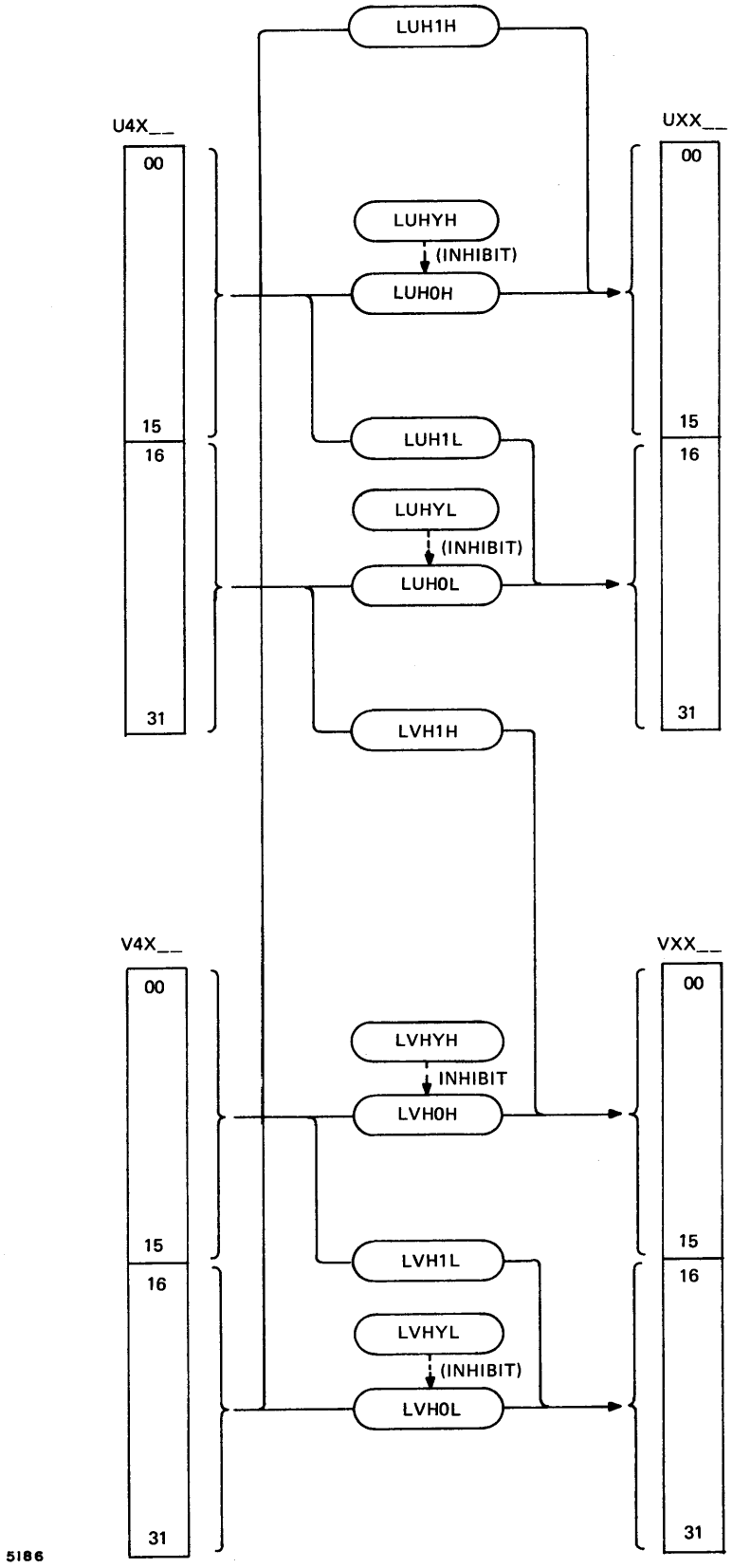


Figure 3-25. Sixteen-Place Shift Patterns

Unshifted Data Transfers

All unshifted data transfers to RG from the U-selection gates and to RH or RT from the V-selection gates use the direct path through the 1-place, 4-place, and 16-place shift gates. Unshifted data transfers are identified in the flow charts in the Central Processor Instructions manual by the symbolic notation "LU*0W," meaning that LU10W, LU40W, LUH0H, and LUH0L are active in the U-net, or "LV*0W," meaning that LV10W, LV40W, LVH0H, and LVH0L are active in the V-net.

Shifting Example

Figure 3-26 illustrates a 21-place right rotation or a 43-place left rotation. Note that bit 11 of the selected U-net input ends in bit 00 of the V-net output; bit 27 of the selected V-net input ends in bit 16 of the U-net output, etc.

ADDER

Summands and Summand Selection

There are three summands: RGF, S1H, and S4H. The RG register is a hard-wired adder input. Whenever the contents of RG are not required to contribute to the adder result, the current algorithm clears RG.

Several inputs can contribute to summands S1H and S4H and are selected under algorithm control. These inputs and their selection control are listed in Table 3-2.

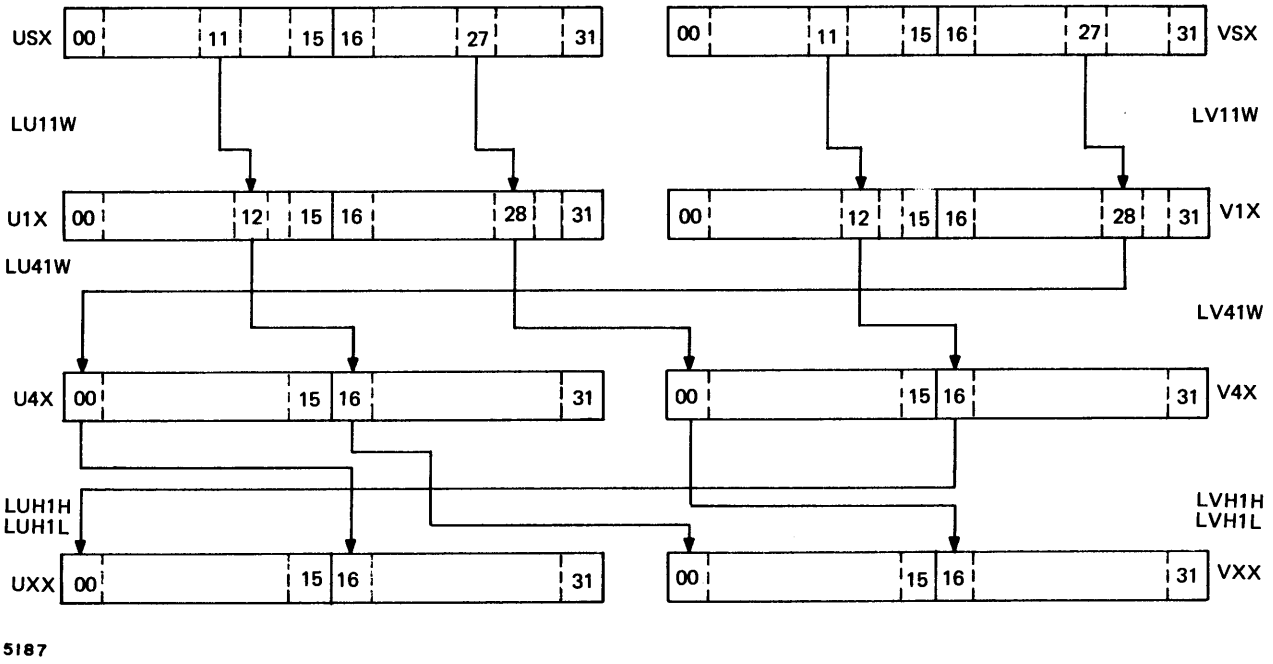


Figure 3-26. Shifting Example: 21-Place Right or 43-Place Left Rotation

Table 3-2.
Adder Inputs and Definitions

Selection Control	Result	Remarks
LSRPI	(SIH) ← $(RP_{04 \rightarrow 30})$	Used for transferring (RP) to one of the address registers. Certain algorithms increment (RP) during this transfer.
LSP1W	(S1H) ← (RH)	Gates (RH) x 1 to summand S1H.
LSN1W	(S1H) ← (∼RH)	Gates one's complement of (RH) to summand S1H.
LSP2W	(S1H) ← (RH) x 2	Gates (RH) x 2 (shifted left one place) to summand S1H.
LSN2W	(S1H) ← (∼RH) x 2	Gates one's complement of (RH) x 2 to summand S1H.
Multiple Selection of S1H Inputs	(S1H) ← Inclusive-OR of selections	
No S1H Inputs Selected	(S1H) ← 0	
LSP4W	(S4H) ← (RH) x 4	Gates (RH) x 4 (shifted left two places) to summand S4H.
LSN4W	(S4H) ← (∼RH) x 4	Gates one's complement of (RH) x 4 to summand S4H.
LSP8W	(S4H) ← (RH) x 8	Gates (RH) x 8 (shifted left three places) to summand S4H.
LSN8W	(S4H) ← (∼RH) x 8	Gates one's complement of (RH) x 8 to summand S4H.
Multiple Selection of S4H Inputs	(S4H) ← Inclusive-OR of selections	
No S4H Inputs Selected	(S4H) ← 0	
LSCIC	(Stage 31 Preliminary Adder) ← 1	Injects carry
LSCIW	(Stage 31 Final Adder) ← 1	Injects carry
LSZCW	(All Carries) ← 0	Suppresses all carries

Results

The adder produces the sum:

$$SUM = (RG) + (S1H) + (S4H) + k$$

where $k = (0 \text{ or } 1 \text{ or } 2) \times 2^{-31}$ and is a function of LSCIC and LSCIW (carry injections).

The CP is two's-complement oriented and the sign is therefore considered only to detect overflow or underflow conditions. For subtraction the subtrahend is obtained from the RH register and is formed by gating (∼RH) to S1H and injecting a carry. Subtraction is then performed by adding.

If LSZCW is true and LSCIC and LSCIW are false, carries are suppressed throughout the adder and an exclusive-OR result is produced.

$$SUM = (RG) \not\vee (S1H) \not\vee (S4H)$$

The RG and RH registers and the sum are 36 bits long and are numbered 60 through 63, 00 through 31. Carries are numbered for the stage from which they emanate.

Method of Implementation

Typical Adder Stage Block Diagram. -- As it is possible to add three summands simultaneously (used by the multiply algorithm only), each adder stage is constructed of two full adders. This provides the capability for receiving and generating two carries.

```
      1  1                  } Carries
   1  1  1  1               }
   0  0  1  1  1            }
   0  0  1  1  1  } Summands
+  0  0  1  1  1            }
   1  0  1  0  1  Sum
```

Figure 3-27 is a block diagram of a typical adder stage showing the two full adders (preliminary and final). The preliminary adder of stage N produces a preliminary sum (SSP_N) and a preliminary carry (SCP_N) from the summands RGF_N, $S1H_N$, and $S4H_N$. The preliminary carry is presented to the final adder of the next most-significant stage (N-1). The preliminary sum is presented to the final adder of this stage (N). The inputs to the final adder are SSP_N, SCP_{N+1}, and the final carry of the next least-significant stage, FCY_{N+1}. The final adder produces a final sum (SUM_N) and a final carry (FCY_N).

NOTE

FCY is a pseudo-mnemonic used for convenience, as the final carry is carried on two wires and has no single mnemonic.

Preliminary Adder. -- See LBD 04.70 for a logic drawing of a specific adder stage. The summands are S1H27, S4H27, and RGF27. The preliminary sum (SSP27) and preliminary carry (SCP27) are produced as shown in the following truth table (Table 3-3) which is valid for all adder stages.

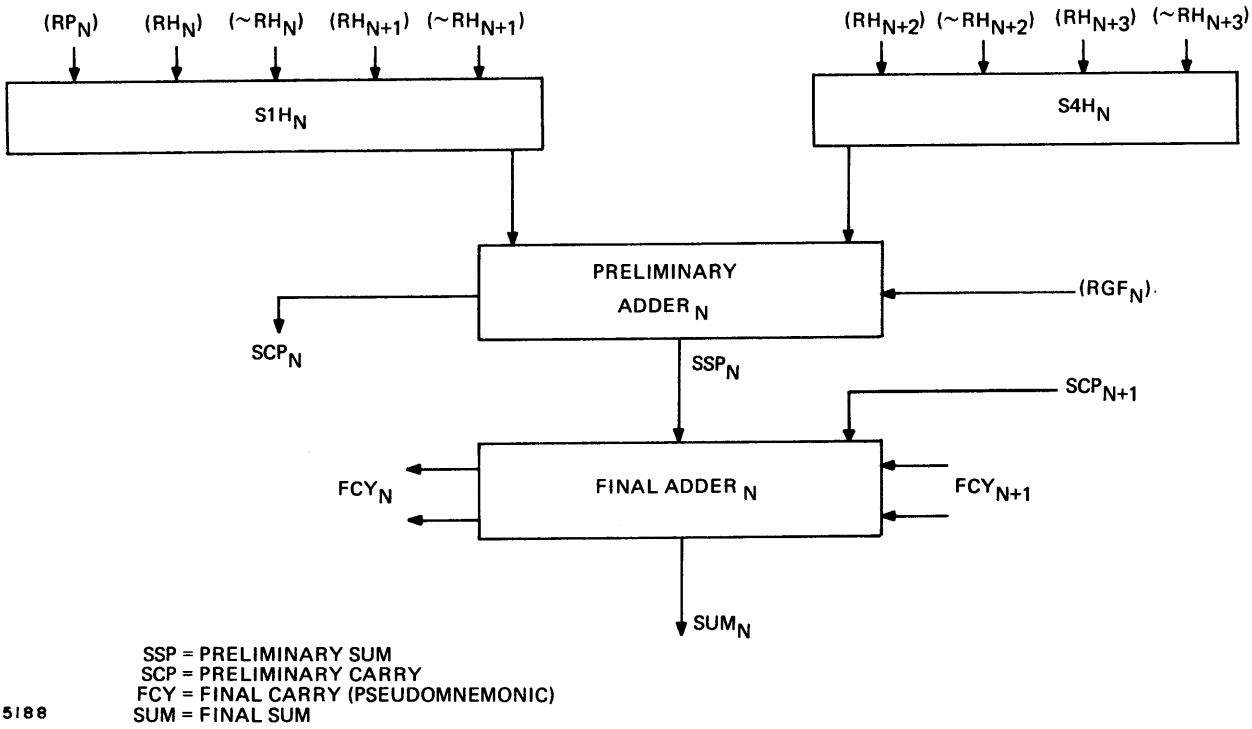


Figure 3-27. Typical Adder Stage Block Diagram

Table 3-3.
Truth Table for SSP_N and SCP_N

RGF_N	0	1	0	0	1	1	0	1
$S1H_N$	0	0	1	0	1	0	1	1
$S4H_N$	0	0	0	1	0	1	1	1
SSP_N	0	1	1	1	0	0	0	1
SCP_N*	0	0	0	0	1	1	1	1

*If LSZCW is true, $SCP_{60 \rightarrow 31} = 0$

Intermediate Functions. -- There are several intermediate functions between the preliminary adder and final adder. These functions are developed to indicate the number of ones being delivered to the final adder by SSP_N and SCP_{N+1}. The functions are SPR_N (SSP_N is a one and/or SCP_{N+1} is a zero), $SP0_N$ (both SSP_N and SCP_N are zeros), $SP1_N$ (either SSP_N or SCP_N is a one but not both), and $SP2_N$ (both SSP_N and SCP_N are ones). Note that $SP0_N$, $SP1_N$ and $SP2_N$ are mutually exclusive. Table 3-4 is a truth table for these functions.

Table 3-4.
Truth Table for SPR_N, $SP0_N$, $SP1_N$ and $SP2_N$

SSP_N	0	1	0	1
SCP_{N+1}	0	0	1	1
SPR_N	1	1	0	1
$SP0_N$	1	0	0	0
$SP1_N$	0	1	1	0
$SP2_N$	0	0	0	1

Final Adder. -- The development of the final carry and the final sum is discussed in the following. To reduce gate delays and speed up carry propagation, even-numbered adder stages do not produce final carries. Instead, the even-numbered stages deliver the raw data necessary to form their carries to the next most significant odd-numbered stage to be used in forming its final sum and carry.

An additional step in eliminating gate delays results in two wires being used to represent the final carry out of each odd-numbered stage. The signal over each wire represents only a partial carry or part of the total carry equation. Furthermore, there are two methods used to develop final carries, the methods being alternated every odd-numbered stage (ignoring insertion of skip carries).

One method uses SCY_N and $SP2_N$. These two signals are inclusively-ORed by stage N to develop its final sum, also by stage N-1 to develop its final sum, and also by stage N-2 to develop its final carry and sum.

The second method uses SCY_N and $SP0_N$. SCY_N is ANDed with $\sim SP0_N$ by stage N to develop its final sum, also by stage N-1 to develop its final sum, and also by stage N-2 to develop its final carry and sum.

Tables 3-5 and 3-6 are truth tables for final carry generation. The Full Carry row indicates the logical operation used to develop the actual final carry from SCY_N and either $SP2_N$ or $SP0_N$.

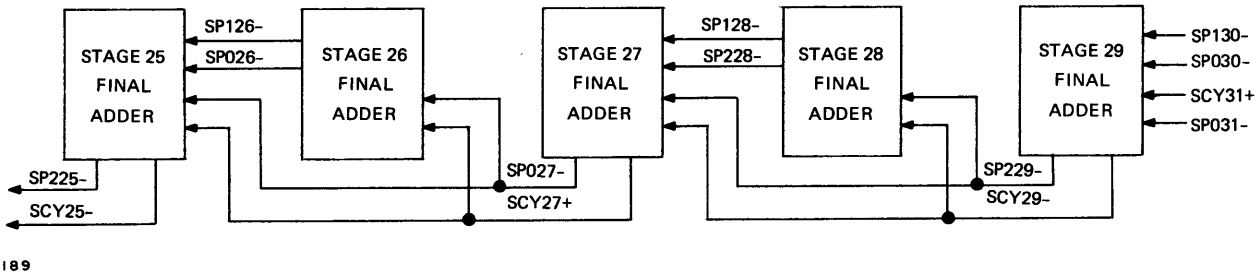
Table 3-5.
Truth Table for Stage 25 Type
Final Carry Generation Where Carry = $(SCY_N) \vee (SP2_N)$

Inputs	SPX_N, where X =	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2
	SPX_{N+1}, where X =	0	0	1	1	2	2	0	0	1	1	2	2	0	0	1	1	2	2
	$Carry_{N+2}$ =	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Partial Carries	SCY_N =	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	$SP2_N$ =	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
Full Carry	$(SCY_N) \vee (SP2_N)$ =	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Table 3-6.
Truth Table for Stage 27 Type
Carry Generation Where Carry = $(SCY_N) \wedge (\sim SP0_N)$

Inputs	SPX_N, where X =	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2
	SPX_{N+1}, where X =	0	0	1	1	2	2	0	0	1	1	2	2	0	0	1	1	2	2
	Carry N+2	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Partial Carries	SCY_N =	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1
	$SP0_N$ =	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
Final Carry	$(SCY_N) \wedge (\sim SP0_N)$ =	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Figure 3-28 shows a segment of the adder with the signals necessary to produce final carries passing from stage to stage. Note that the preceding discussion has ignored the skip carry logic which is discussed later.



5189

Figure 3-28. Final Carry Signals Block Diagram

Four methods are used to develop the final sum, the reasons being: (a) the two methods for generating final carries (and the consequent alternation of signal polarities), and (b) the odd-numbered stages receive raw data for the carries out of the even-numbered stages. Therefore, there are two methods for sum generation in the odd-numbered stages and two methods in the even-numbered stages.

Tables 3-7 through 3-10 are truth tables for the four methods of sum generation. The tables show the truth values for partial sums SMQ_N, SME_N, and SMD_N for all possible input combinations. The truth values for the final sum are shown as logical functions of SMQ_N, SME_N, and SMD_N.

Ignoring skip-carries, Table 3-7 is for those even-numbered stages receiving a carry on lines SCY_{N+1} and $SP2_{N+1}$. Table 3-8 is for those odd-numbered stages receiving a carry on lines SCY_{N+2} and $SP2_{N+2}$. Table 3-9 is for those even-numbered stages receiving a carry on lines SCY_{N+1} and $SP0_{N+1}$. Table 3-10 is for those odd-numbered stages receiving a carry on lines SCY_{N+2} and $SP0_{N+2}$.

Table 3-7.
Truth Table for Stage 28 Type
Final Sum Generation

Inputs	SPX_N, where X =	0	0	1	1	2	2
	C_{N+1}* =	0	1	0	1	0	1
Partial Sums	SMD_N =	0	0	1	0	0	0
	SMQ_N =	0	1	0	0	0	1
Final Sum	(SMD_N) V (SMQ_N) =	0	1	1	0	0	1
*(C_{N+1}) = (SCY_{N+1}) V $(SP2_{N+1})$							

Table 3-8.
Truth Table for Stage 27 Type
Final Sum Generation

Inputs	SPX_N, where X =	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2
	SPX_{N+1}, where X =	0	0	1	1	2	2	0	0	1	1	2	2	0	0	1	1	2	2
	C_{N+2}* =	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Partial Sums	SMQ_N =	0	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1
	SME_N =	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1
	SMD_N =	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
Final Sum	$(SMQ_N) \wedge (SME_N) \wedge (SMD_N)$ =	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1
*(C_{N+2}) = (SCY_{N+2}) V $(SP2_{N+2})$																			

Table 3-9.
Truth Table for Stage 26 Type
Final Sum Generation

Inputs	SPX_N, where X =	0	0	1	1	2	2
	C_{N+1}* =	0	1	0	1	0	1
Partial Sums	SMQ_N =	0	1	1	1	0	1
	SMD_N =	1	1	1	0	1	1
Final Sum	$(SMQ_N) \wedge (SMD_N)$	0	1	1	0	0	1
*$(C_{N+1}) = (SCY_{N+1}) \wedge (\sim SP0_{N+1})$							

Table 3-10.
Truth Table for Stage 25 Type
Final Sum Generation

Inputs	SPX_N, where X =	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2
	SPX_{N+1}, where X =	0	0	1	1	2	2	0	0	1	1	2	2	0	0	1	1	2	2
	C_{N+2}* =	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Partial Sums	SMD_N =	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
	SME_N =	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1
	SMQ_N =	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1
Final Sum	$(SMD_N) \vee (SME_N) \vee (SMQ_N)$ =	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1
*$(C_{N+2}) = (SCY_{N+2}) \wedge (\sim SP0_{N+2})$																			

Skip-Carry Network

The skip-carry network (LBD 05.20) functions to speed up carry propagation. The network inserts four signals into various stages: SKP23-, SKP13+, SKP05-, and SKP63+. The generation of one of these signals indicates a carry generation in some less significant stage and a continuous propagation of that carry through the stage indicated in the SKP mnemonic. For example, if the first carry is generated by stage 29 and propagated through stage 23, gate SKP28 generates SKP23. (The gates are named for the first stage through which the carry is propagated, not for the stage which generates the carry.) The conditions for gate SKP28 are SP229, i.e., $(\sim SP029) \wedge (\sim SP129)$, and at least an input of one ($\sim$SP0) to the final adders of stages 28 through 23.

Note that the signal SKP23- indicates carry propagation through stage 23 but not necessarily a carry generation by stage 23. Therefore, the skip carry signal SKP23-, SKP13+, SKP05-, and SKP62+ are partial carries and are inserted into their next respective stages. The SKP assertion signals are inserted instead of the SCY assertion signals and to form the complete carry, are ORed (by the stages which use the carries) with SP2, just as would be the SCY assertion carries. Similarly, the SKP negation signals are ANDed with ~SP0 just as the SCY negation signals.

Adder Structure

The adder structure is depicted in Figure 3-29, illustrating general signal routing and polarity. For example, it shows the inputs to adder stage 11 as RHF11 or RHF12 via S1 and RHF13 or RHF14 via S4. The carry and sum polarities for the different stages are indicated. The insertion of skip carries and their polarities is shown.

Figure 3-29 also indicates the PAC boundaries of the RG, RH, and adder stages. For example, stages 10 and 11 of the RG and RH registers and of the preliminary adder are on one PAC with the intermediate functions and the final adder of stages 11 and 12. The skip-carry network is contained on one separate PAC.

CONSOLE CONTROL AND DISPLAY PANELS

The Series 32 control and display console consists of two electrically separate assemblies cabled to each other, to the CP logic frame, and to the power control chassis. Mounted on the control panel are the pushbuttons, toggle switches, and keys that control power turn-on, CP operation mode and start-stop control, manual data entry, display selection, maintenance mode control, address halt (optional), fill device selection, and sense switching. The display panel features two 32-bit indicator displays that can be supplied with data from a variety of sources, controlled by the control panel display selection switches. Back-lighted labels display CP status indications and identify the contents of the selected 32-bit displays. Full information on display labeling appears in the Series 32 Operator's Manual.

Figure 3-30 shows the routing of data and control to and from the control and display panels. Logic diagrams that provide detail on interconnections and logic functions are specified on the block diagram.

All console signals are defined in the mnemonic glossary in the Central Processor Instructions manual.

Address/Data Entry Switches (LBD 07.26 and 07.28)

These switches are used for direct insertion of addresses and data in RG and RH prior to a transfer, access fetch, or access store operation. The access mode must be selected for the switches to be effective.

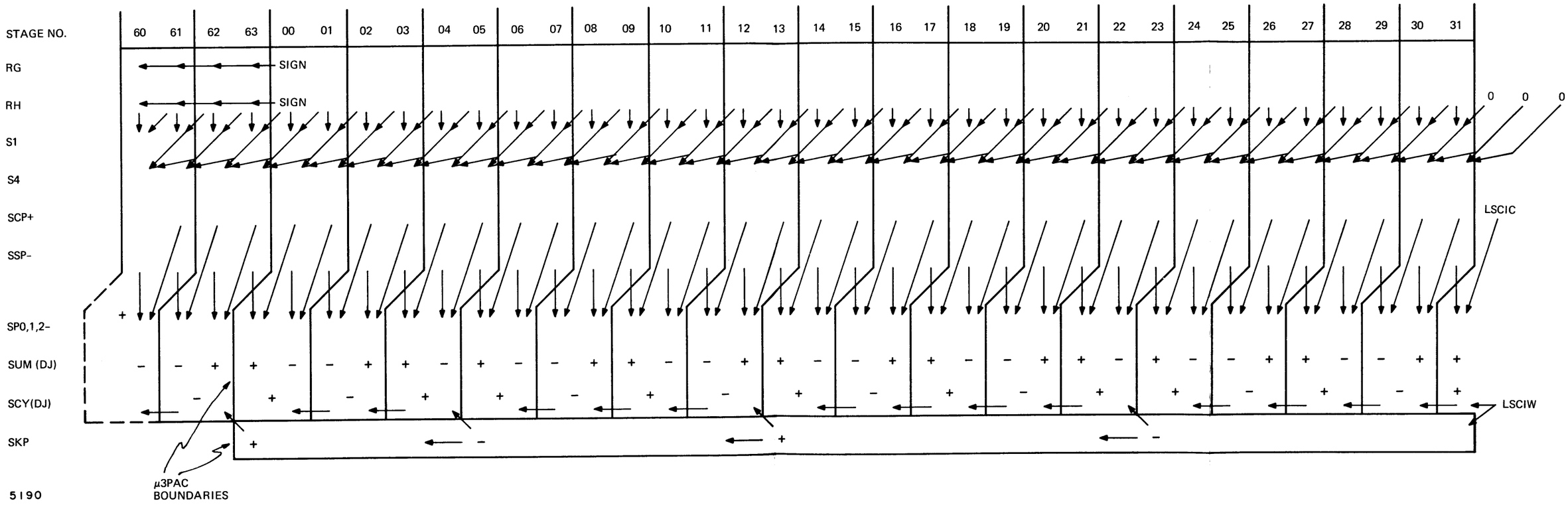


Figure 3-29. Adder Structure

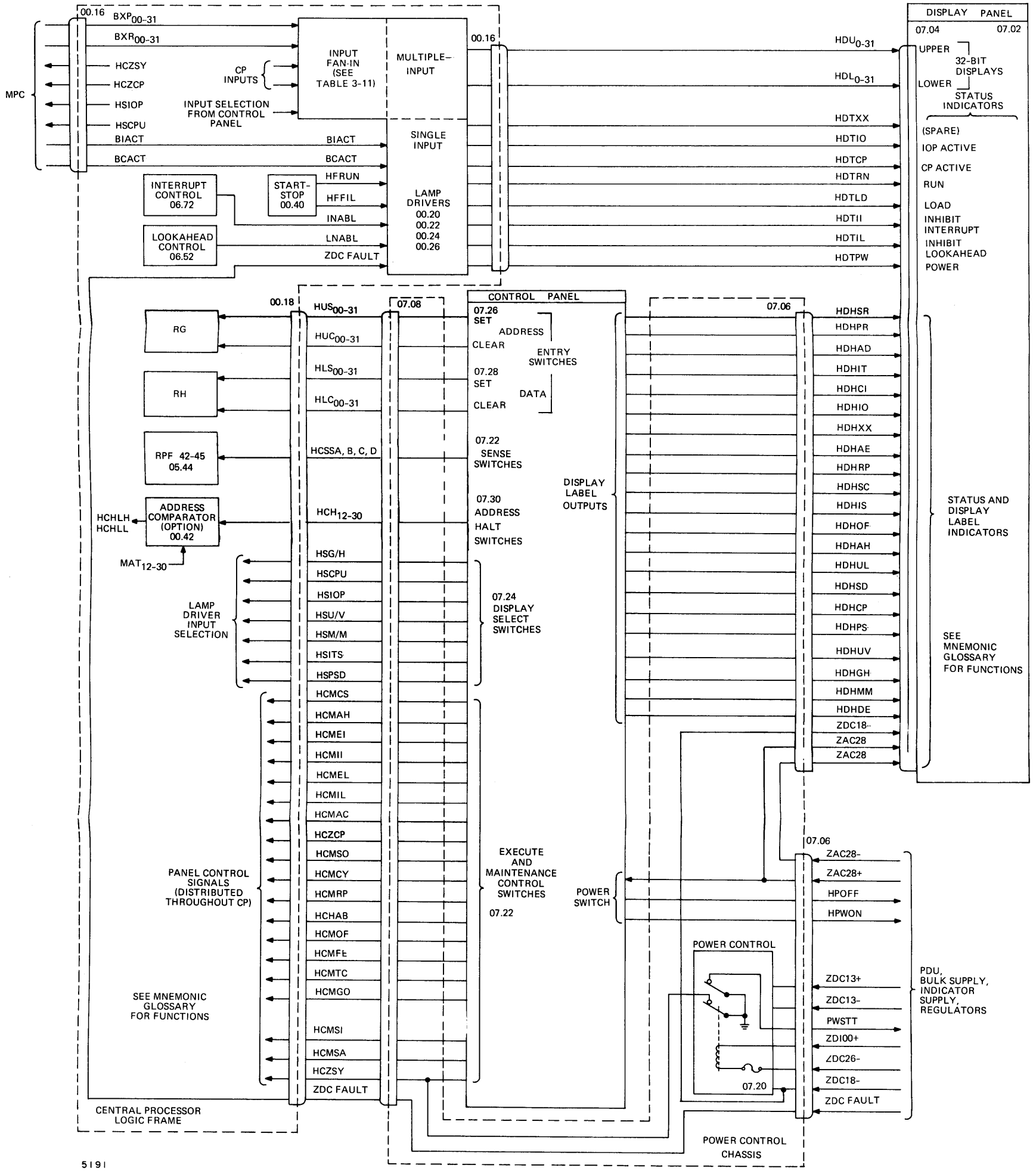


Figure 3-30. Console Data Distribution

Bit	PS1/PS2 Display*		IOP Display		CP Display		G/H Display or Access Display**		Inst. Display		M Display		U/V Display		Bit
	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper	
0	RPF32+		BXR00+	BXP00+	BXR00+	BXP00+	RHF00+	RGF00+		TSOBF-	RMF00+	RKL00+	VXX00+	UXX00+	0
1	RPF33+		BXR01+	BXP01+	BXR01+	BXP01+	RHF01+	RGF01+		TSPAF+	RMF01+	RKL01+	VXX01+	UXX01+	1
2			BXR02+	BXP02+	BXR02+	BXP02+	RHF02+	RGF02+		TIPAX+	RMF02+	RKL02+	VXX02+	UXX02+	2
3	RPF35+		BXR03+	BXP03+	BXR03+	BXP03+	RHF03+	RGF03+		TICAF+B	RMF03+	RKL03+	VXX03+	UXX03+	3
4	RPF36+	RPF04+	BXR04+	BXP04+	BXR04+	BXP04+	RHF04+	RGF04+		RIA04+	RMF04+	RKL04+	VXX04+	UXX04+	4
5	RPF37+	RPF05+	BXR05+	BXP05+	BXR05+	BXP05+	RHF05+	RGF05+		RIA05+	RMF05+	RKL05+	VXX05+	UXX05+	5
6	RPF38+	RPF06+	BXR06+	BXP06+	BXR06+	BXP06+	RHF06+	RGF06+		RIA06+	RMF06+	RKL06+	VXX06+	UXX06+	6
7	RPF39+	RPF07+	BXR07+	BXP07+	BXR07+	BXP07+	RHF07+	RGF07+		RIA07+	RMF07+	RKL07+	VXX07+	UXX07+	7
8		RPF08+	BXR08+	BXP08+	BXR08+	BXP08+	RHF08+	RGF08+		RIA08+	RMF08+	RKL08+	VXX08+	UXX08+	8
9		RPF09+	BXR09+	BXP09+	BXR09+	BXP09+	RHF09+	RGF09+		RIA09+	RMF09+	RKL09+	VXX09+	UXX09+	9
10	RPF42+	RPF10+	BXR10+	BXP10+	BXR10+	BXP10+	RHF10+	RGF10+		RIA10+	RMF10+	RKL10+	VXX10+	UXX10+	10
11	RPF43+	RPF11+	BXR11+	BXP11+	BXR11+	BXP11+	RHF11+	RGF11+		RIA11+	RMF11+	RKL11+	VXX11+	UXX11+	11
12	RPF44+	RPF12+	BXR12+	BXP12+	BXR12+	BXP12+	RHF12+	RGF12+		RIA12+	RMF12+	RKL12+	VXX12+	UXX12+	12
13	RPF45+	RPF13+	BXR13+	BXP13+	BXR13+	BXP13+	RHF13+	RGF13+		RIA13+	RMF13+	RKL13+	VXX13+	UXX13+	13
14		RPF14+	BXR14+	BXP14+	BXR14+	BXP14+	RHF14+	RGF14+		RIA14+	RMF14+	RKL14+	VXX14+	UXX14+	14
15		RPF15+	BXR15+	BXP15+	BXR15+	BXP15+	RHF15+	RGF15+			RMF15+	RKL15+	VXX15+	UXX15+	15
16		RPF16+	BXR16+	BXP16+	BXR16+	BXP16+	RHF16+	RGF16+		TLFAF+	RMF16+	RKL16+	VXX16+	UXX16+	16
17		RPF17+	BXR17+	BXP17+	BXR17+	BXP17+	RHF17+	RGF17+		TLIAF+	RMF17+	RKL17+	VXX17+	UXX17+	17
18		RPF18+	BXR18+	BXP18+	BXR18+	BXP18+	RHF18+	RGF18+		TL1AF+D	RMF18+	RKL18+	VXX18+	UXX18+	18
19		RPF19+	BXR19+	BXP19+	BXR19+	BXP19+	RHF19+	RGF19+		TL2AF+	RMF19+	RKL19+	VXX19+	UXX19+	19
20		RPF20+	BXR20+	BXP20+	BXR20+	BXP20+	RHF20+	RGF20+		TL3AF+B	RMF20+	RKL20+	VXX20+	UXX20+	20
21		RPF21+	BXR21+	BXP21+	BXR21+	BXP21+	RHF21+	RGF21+		TL4AF+	RMF21+	RKL21+	VXX21+	UXX21+	21
22		RPF22+	BXR22+	BXP22+	BXR22+	BXP22+	RHF22+	RGF22+		TL5AF+	RMF22+	RKL22+	VXX22+	UXX22+	22
23		RPF23+	BXR23+	BXP23+	BXR23+	BXP23+	RHF23+	RGF23+		TL6AF+	RMF23+	RKL23+	VXX23+	UXX23+	23
24		RPF24+	BXR24+	BXP24+			RHF24+	RGF24+			RMF24+	RKL24+	VXX24+	UXX24+	24
25		RPF25+	BXR25+	BXP25+			RHF25+	RGF25+			RMF25+	RKL25+	VXX25+	UXX25+	25
26		RPF26+	BXR26+	BXP26+	IAF24+	IDL24+	RHF26+	RGF26+			RMF26+	RKL26+	VXX26+	UXX26+	26
27		RPF27+	BXR27+	BXP27+	IAF25+	IDL25+	RHF27+	RGF27+		TCT27+	RMF27+	RKL27+	VXX27+	UXX27+	27
28		RPF28+	BXR28+	BXP28+	IAF26+	IDL26+	RHF28+	RGF28+		TCT28+	RMF28+	RKL28+	VXX28+	UXX28+	28
29		RPF29+	BXR29+	BXP29+	IAF27+	IDL27+	RHF29+	RGF29+		TCT29+	RMF29+	RKL29+	VXX29+	UXX29+	29
30		RPF30+	BXR30+	BXP30+	IAF28+	IDL28+	RHF30+	RGF30+		TCT30+	RMF30+	RKL30+	VXX30+	UXX30+	30
31			BXR31+	BXP31+	IAF29+	IDL29+	RHF31+	RGF31+		TCT31+	RMF31+	RKL31+	VXX31+	UXX31+	31
	Lamp Driver Gating Term: HSPSD		Lamp Driver Gating Term: HSIOP		Lamp Driver Gating Term: HSCPU		Lamp Driver Gating Term: HSG/H		Lamp Driver Gating Term: HSITS		Lamp Driver Gating Term: HSM/M		Lamp Driver Gating Term: HSU/V		

*Selected when ⇄ switch is centered in run mode **or** step mode.

**Selected when ⇄ switch is centered in access mode.

Table 3-11.
Display Lamp Driver Input Summary

Address Halt Keys (LBD 07.30)

These keys are used in connection with the optional address halt feature. An address is manually set into the keys and the control key to the left is set at HALT. When a memory address matches the key setting, the CP clock is stopped in a panel halt condition. (See start-stop logic description.) LBD 00.42 illustrates the comparison logic.

Sense Keys (LBD 07.22)

The four sense keys directly control bounce-suppression flip-flops RPF42 through 45, where they are available for testing by bit-addressing instructions.

Execute and Maintenance Control Switches (LBD 07.22)

These switches control the CP execution mode. Many of the signals (such as HCZSY and HCMSA) have been discussed in connection with start-stop sequencing and control. Others take effect during the execution of a particular algorithm. Refer to the mnemonic glossary and flow charts in the Central Processor Instructions manual for details.

Display Select Switches (LBD 07.24)

These switches generate the signals that gate inputs through the lamp drivers to the 32-bit indicator registers. The gating signals correspond directly to selection switch positions except for HSPSD and HSG/H. HSPSD is enabled when the left-right selector switch is centered and the CP is not in the access mode, to provide a normal PS1/PS2 display. HSG/H is enabled in the access mode (assuming the left-right switch is centered) to provide the G/H display of memory address and data.

Direct-Wired Status and Display Labeling

Many of the control panel switches produce outputs that are wired directly to the display panel to light status or labeling indicators. For example, HDHGH lights the G-REG and H-REG labels that appear when the G/H display is selected. See the mnemonic glossary in the Central Processor Instructions manual for definition of these terms.

Lamp Drivers and Display Selection (LBD 00.20 through 00.26)

One lamp driver PAC is provided for each 8 stages of the upper or lower 32-bit display indicator banks. Each PAC contains a seven-input data selection fan-in and an output switch that controls the display indicator.

Input selection is controlled by the gating terms from the display select switches on the control panel. Data selected by the lamp drivers for each display is summarized in Table 3-11.

The lamp driver output stage provides a low impedance to ground when the selected data input is true, and an open circuit when the input is false. The output stage thus serves as a switch in series with the 18-volt indicator supply and the associated indicator lamp.

Lamp drivers with ungated inputs drive the status indicators (POWER, LOAD, etc.) that are controlled by signals from CP logic. The other status indicators are controlled directly from control panel switches, and so do not require lamp drivers.

Control Panel Power Switching (LBD 07.20)

When the control panel POWER switch is held ON, 28 Vac is applied across the power distribution unit's main power contactor. If the bulk supply and the indicator supply reach full voltage and there is no ac fault in the system, a relay in the PDU holds the contactor on; the panel POWER switch can be released. (The HPOFF contact holds the main contactor on.) When the POWER switch is turned OFF, the HPOFF contact opens the contactor holding circuit and turns off prime power in the PDU. (Loss of the indicator supply or presence of an ac fault at the input to the bulk supply also drops out the main power contactor.)

A time delay relay on the power control chassis generates system initialize for about 1 second during and after power-on. The same relay provides the PWSTT contact closure which informs external devices that CP ac power is off.

The ZDCFAULT signal, brought into the power control chassis from the system power supply regulators, controls the POWER lamp on the panel through a lamp driver. The ZDCFAULT signal is ORed from all regulators in the system; if any regulator develops a dc fault, the signal turns off the POWER indicator on the panel. (No automatic ac or dc power shutdown occurs, however, outside of the faulty regulator.)

MPC Interconnections

The BXP and BXR lines from the MPC contain activity state codes used in the CP and IOP display modes. The HSIOP and HSCPU gating terms from the panel specify which group of data is required. The MPC also provides control signals for the CP ACTIVE and IOP ACTIVE indicators, driven by lamp drivers. Over this same cable the system and CP initialize signals are supplied from the CP to the MPC. (See paragraph on Sequencing and Control in Section II.)

HONEYWELL, COMPUTER CONTROL DIVISION, FRAMINGHAM, MASS. 01701

Printed in U.S.A.