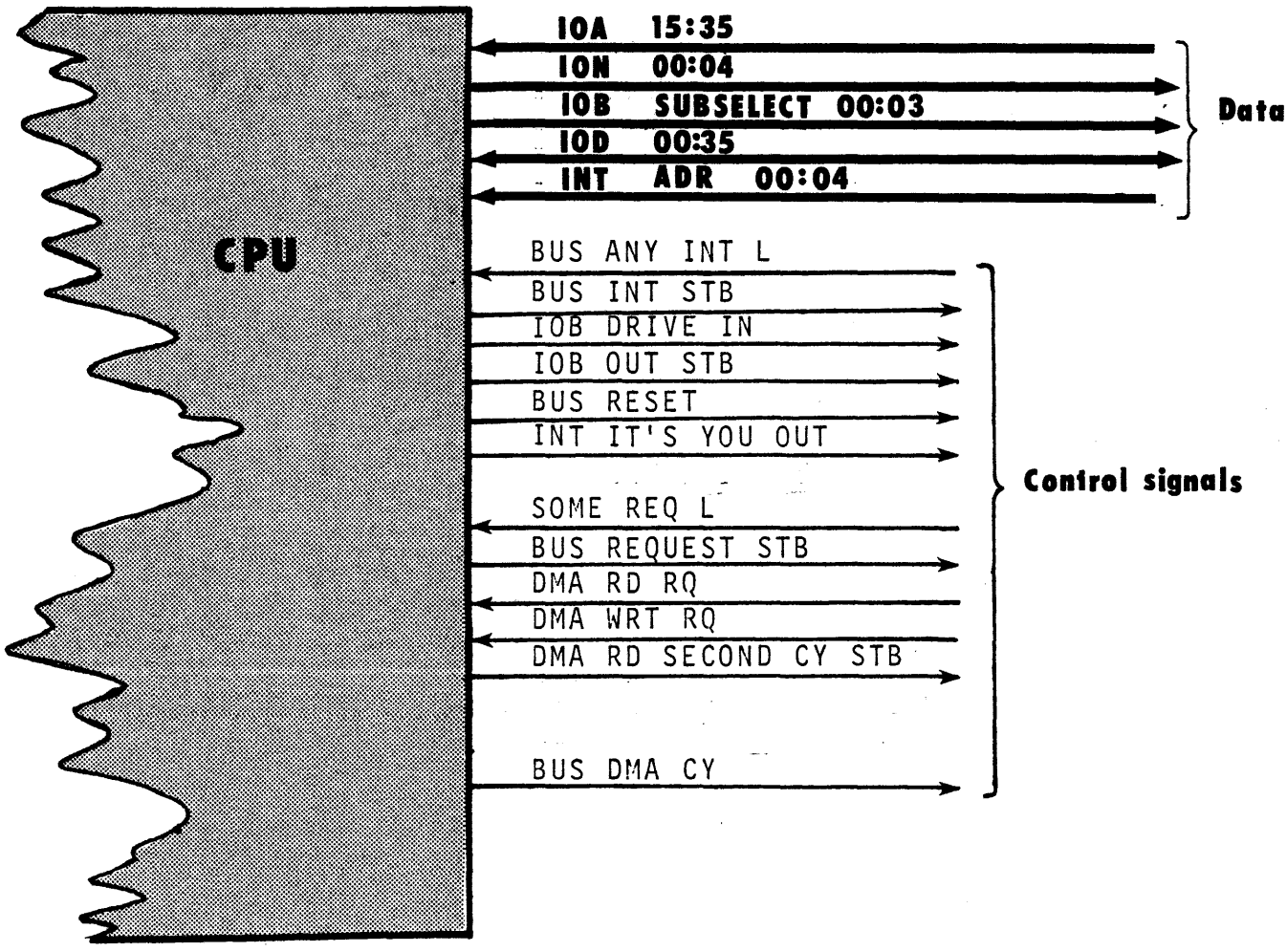
Foonly I/O
CHAPTER 6

6.1 OVERVIEW

The bus designated for I/O operations is called the F bus. The F bus is a 36 bit bidirectional data bus, and in addition contains control and address signals as shown in Figure 6-1 Three types of operations can occur on the F bus: (IOT) for Data transfers to and from the CPU, (DMA) for channeling data from a device directly to and from memory, and INTERRUPT for getting the CPU's attention.



[F-BUS SIGNALS]

Figure 6-1

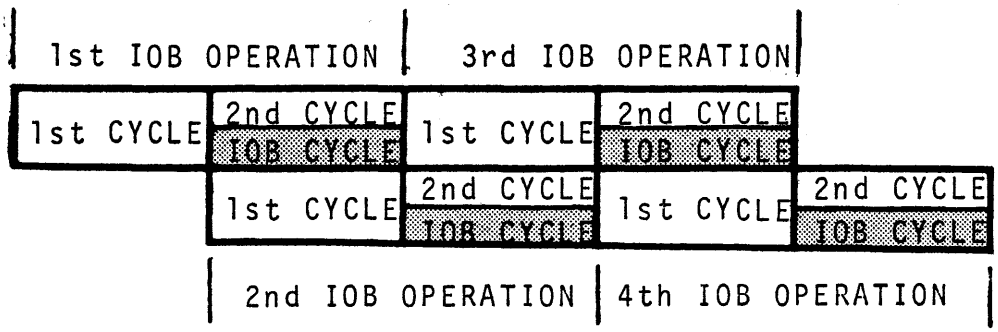


Figure 6-2
[IOT CYCLE OVERLAPPING]

Each type of IOB operation involves two machine cycles. However, by overlapping opposite cycles of contiguous IOB operations to the same device, each cycle may act as an IOB cycle. Figure 6-2 illustrates this further.

As shown in Figure 6-2, the second cycle of each operation is referred to as an IOB cycle. It is during this cycle that data is actually transferred on the F bus. During the first cycle of the first operation the device code is latched for eternity into the ION BUF. Because of inherent restrictions on uCode field usage, setting up the device code requires a separate cycle, during the first operation.

6.1 IOT TRANSFERS

An IOT transfer originates after the CPU clocks a micro instruction with IOB REQ IN, or OUT coded in its SPEC RUN Field. IOB REQ IN or OUT is decoded to set IOB DRIVE IN or IOB OUT at the first BUS CLK during the NEXT cycle. BUS CLK is a delayed version of CPU CLK, which determines the IOB cycles. Figure 6-3 illustrates this timing.

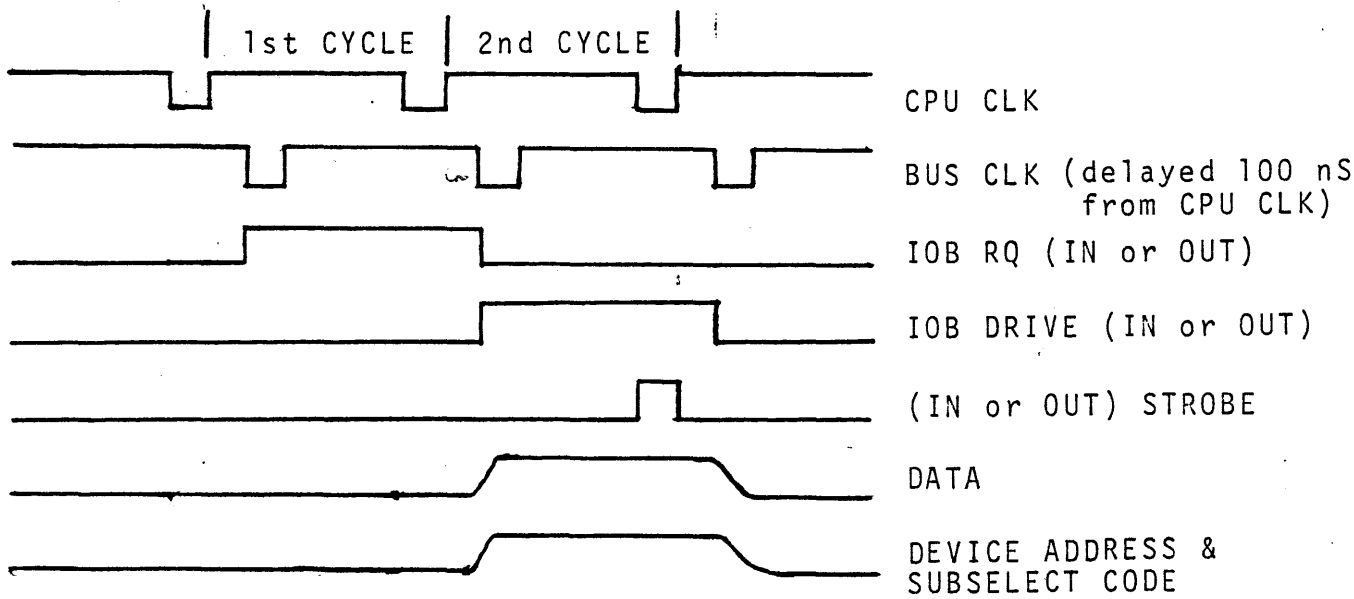


Figure 6-3
[IOT IN & OUT TIMING]

6.2.1 IOT out operations

Each CPU CLK addresses and decodes another micro instruction. Two or more micro instructions are required for any IOT operation, as shown in Figure 6-4. IOB OUT is decoded from the SPEC RUN field of the first micro instruction as previously mentioned. The first micro instruction may also decode the EXT D field as = 1_8 (mem). This enables the Main Memory Data which may represent the contents of E from the Macro I/O instruction into the IOD register via the O BUS, as a result of decoding an 11_8 from the DEST field. The second micro instruction asserts the device subselect code as an image of the MAP F field. Also this instruction must specify a 7_8 (constant) in its DEST field in order to load the DEVICE ADDRESS register as an image of the Constant field. The CONSTANT field, physically the low order bits of the MASK field, will specify a physical device number only when the EXT D field is coded as 3_8. It is important to realize that different micro instructions may be addressed from the macro op-codes, and that the op-codes are particular to the device code. Thus, for each type of macro I/O instruction the microcode will be addressed in a manner such that the correct device address will be sent to the F BUS, during the execution of the second and subsequent micro instructions.

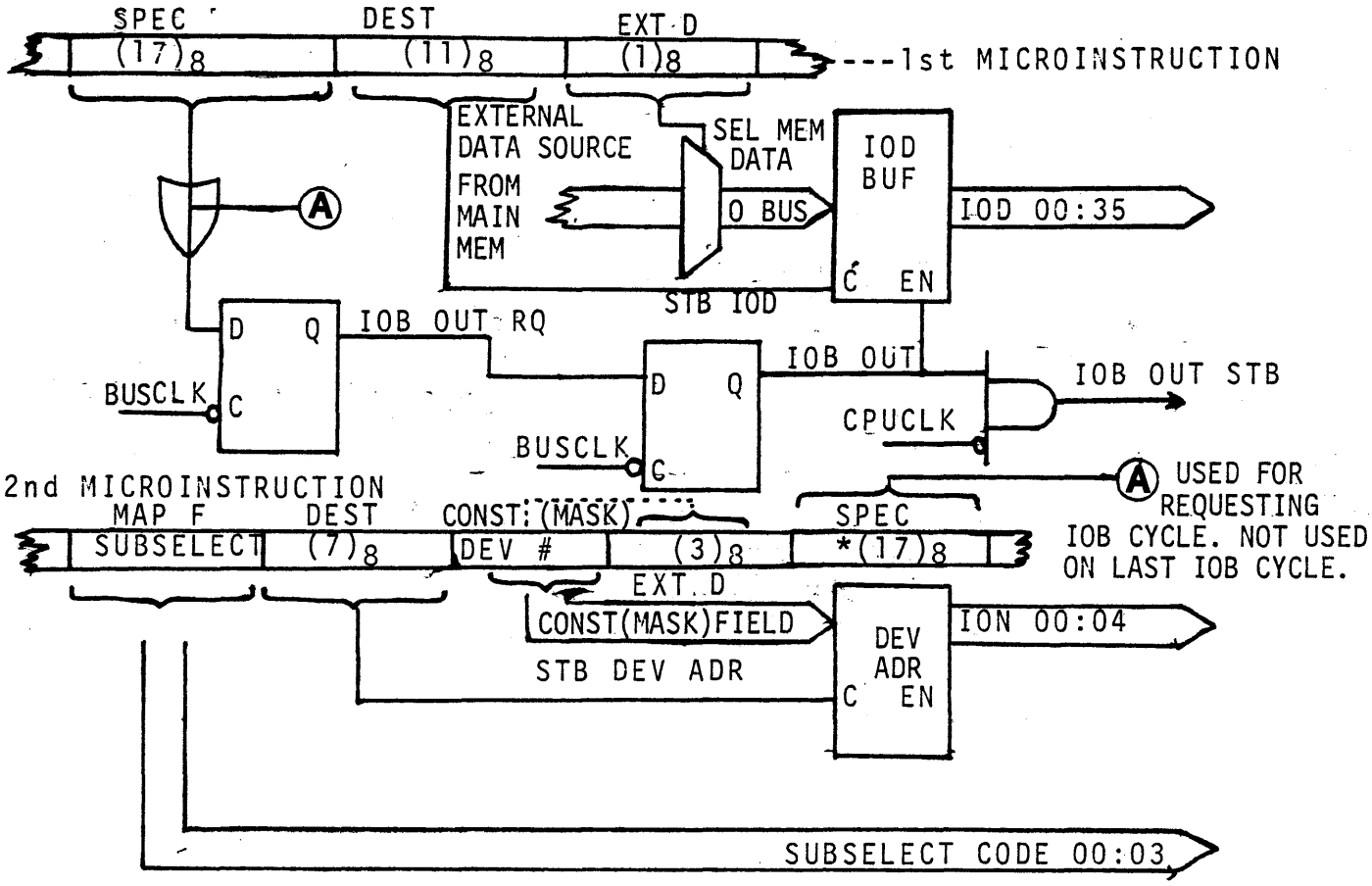


Figure 6-4
[IOT OUT OPERATION]

The BUS CLK, that occurs 100 nS after the CPU CLK, which initiated the second cycle, will cause IOB OUT to set from IOB OUT RQ. IOB OUT causes the OUTPUT DATA BUFFER (IOD BUF) to be enabled onto the F BUS. At this point in time the F BUS has all the necessary information in terms of DATA, DEVICE ADDRESS, and SUBSELECT CODE to perform the IOT operation.

The CPU CLK which signifies the end of the second machine cycle is ANDED with IOB OUT to produce IOB OUT STB. This signal is sent to the F BUS and strobes the addressed device to latch the SUBSELECT CODE, and the DATA into its internal registers. The SUBSELECT CODE, specifies what type of DATA was sent, and is interpreted by the device as a command. The second micro instruction may involve yet another IOT operation on the next cycle by setting the appropriate code for IOT OUT or IOT IN, in the SPEC field. The device code which is still latched and on the F BUS allows any subsequent IOT operations to use only the format of the second micro instruction thus permitting every MACHINE CYCLE to be an IOB CYCLE. The last cycle to be performed does not set IOT IN or OUT in its SPEC field, hence terminating the REQUEST for subsequent IOB operations.

IOT type transfers are NOT used to transfer DATA to and from MASS STORAGE DEVICES, but instead their DATA field is interrupted as CONTROL DATA by a controller. Actual mass storage DATA transfers are done directly to memory by DMA operations. In this case, the CPU must specify to the controller a starting address in main memory, and a location address of where the DATA can be found on mass storage. It is the purpose of IOT transfers to provide this type of information.

Since there is no handshake involved in an IOT OUT, an operation to a non-existent device does not cause the F BUS to hang, nor will the CPU ever know whether the device received the information without doing an IOT IN. (Setting Interrupts are an obvious exception to this).

6.2.2 IOT in operations

Inputting control information from an F BUS controller, shares much of the same timing and features as IOT OUT, with major differences occurring in the destination of DATA, and some of the microcode fields. As with an IOT OUT this type of operation is not used for obtaining actual DATA from MASS STORAGE, but rather is used for inquiring about the status of a controller by the CPU.

A macro I/O input instruction will cause the execution of two or more micro instructions. At the end of the second micro-instruction the requested information in terms of a 36 bit DATA word will be available in the CPU. Destinations for input data may be either a location in main memory, or an accumulator in the CPU as determined by the effective address of the MACRO I/O Input instruction.

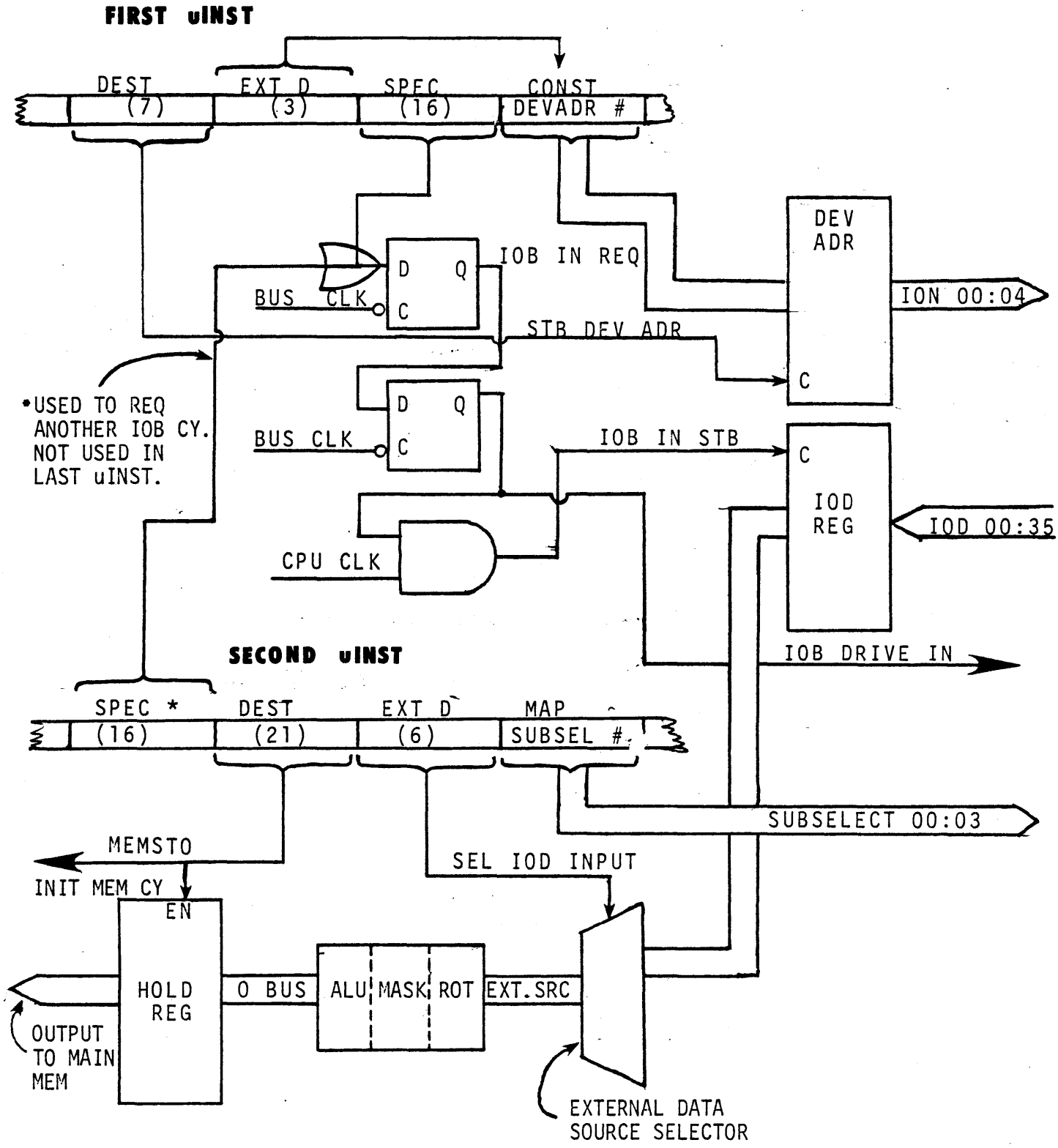


Figure 6-5

[IOT IN OPERATION]

When the first micro instruction is addressed (referring to Figure 6-5), it will have coded in its SPEC RUN field an octal 16 (IOD IN). This will allow the signal IOB DRIVE IN to set during the second cycle. Also in the first micro instruction period we must load the DEVICE ADDRESS register since the field which does this also specifies the code for gating the input DATA which must be done during the second cycle. This is done with the DEST field = 7_8, and the EXT D field = 3_8. The EXT D field code 3_8 specifies that the constant field will contain the correct DEVICE CODE to be loaded into the DEVICE ADDRESS register, and a DEST field of 7_8 causes the contents of the constant field to be strobed into the ION BUF or Device Address Register.

The first micro instruction, having specified the type of operation (IOB IN), and the device number, as well as setting up IOB REQ IN, leaves the second micro instruction with the task of specifying the SUBSELECT CODE, gating the appropriate DATA into the CPU and writing it into main memory. The SUBSELECT field is taken from the MAP F field of the micro code as a direct image. This field will be coded accordingly in the second micro instruction to specify the source of the input data from within the controller on the F BUS. The second micro instruction will also have its EXT D field = 6_8. The purpose of this field is to specify the source of the external DATA mixer located in the CPU to be selected as IOD, or in other words, the F BUS DATA lines. The second instruction will also have its DEST FIELD = 1 (MEMSTO). The F BUS DATA will be enabled through the EXTERNAL DATA SELECTOR through the ROTATOR, the MASKER, the ALU, and onto the O BUS where it will be clocked into the HOLD REGISTER and during the next machine cycle sent to a main memory location specified in the E of the MACRO INSTRUCTION which originated the operation. An important point to remember is that the last micro instruction of the I/O routine must strobe the ION BUF by DEST = 7_8 with the EXT D field = 3_8 and CONSTANT = 0. This will zero the F BUS ION lines which were set during the first micro instruction.

6.3 DMA TRANSFERS

DMA provides the means, by which I/O devices on the F BUS may transfer mass storage data to and from MAIN MEMORY. All DMA transfers are initiated by F BUS devices and are arbitrated in terms of priority in the event that more than one device asserts a DMA request at the same time.

MAIN MEMORY PHYSICAL ADDRESSES are supplied by, and are incremented by the device for each word which is sent or received. Figure 6-6 shows data and address paths in simplified form. Note data and address information utilize registers within the CPU. The reason for this type of architecture is that the MAIN MEMORY is not multi-controlled, and can thus handle only one device;the CPU.

DMA operations are handled in a totally autonomous manner

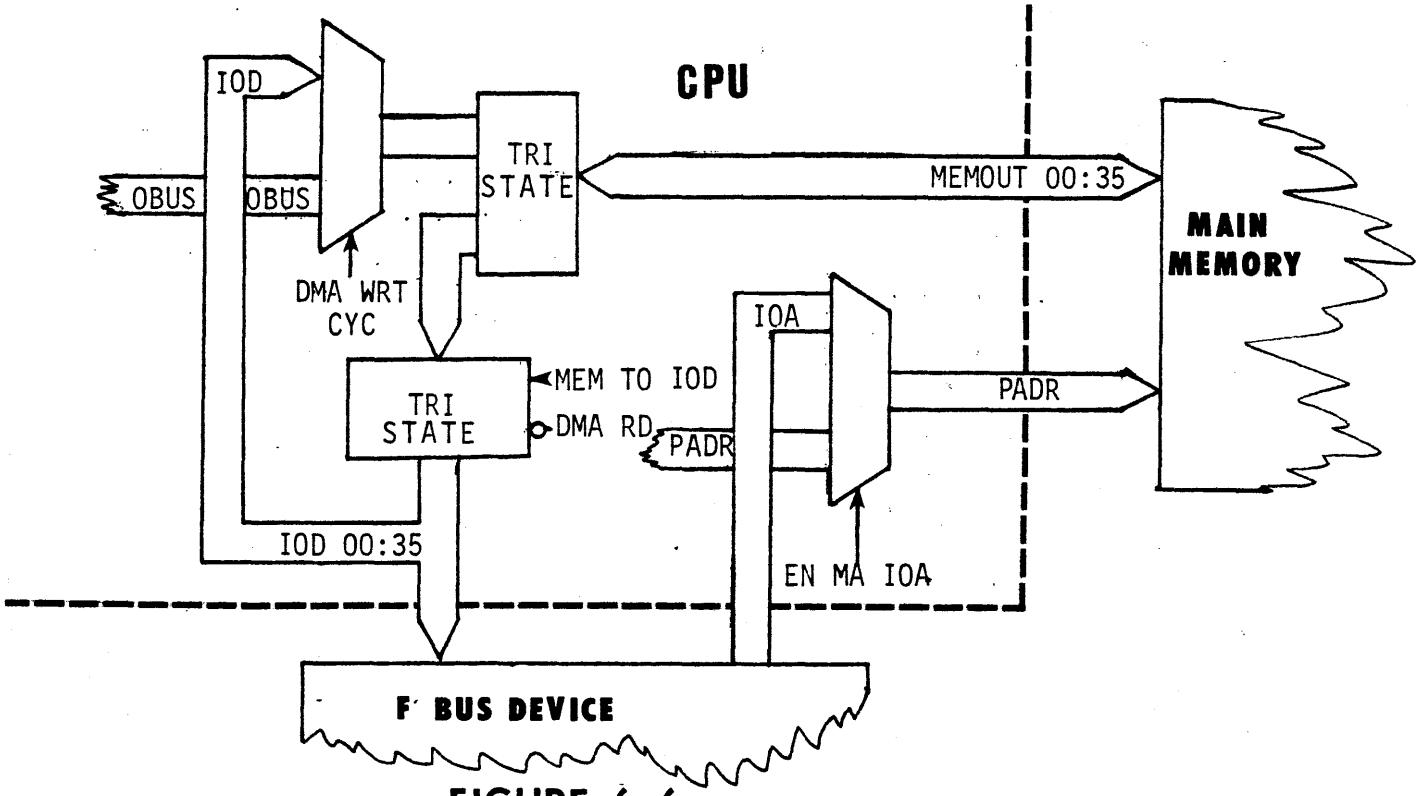


FIGURE 6-6
[SIMPLIFIED DATA AND ADDRESS PATHS FOR DMA OPERATIONS]

which requires no intervention on the part of the MICRO PROGRAM running with in the CPU Because of this independence it is a common circumstance to have a DMA REQUEST simultaneous with a micro program MAIN MEMORY REQUEST. In this case the CPU has higher priority and gets to transfer its data while the DMA REQUEST remains pending. This might seem confusing unless one realizes that first, there is no way of stopping the CPU MICRO INSTRUCTION PROCESS except by hard error, or manually, by switch control. Thus if a micro instruction was clocked which specified a memory operation and memory was busy doing DMA the CPU data would be lost. The second consideration to keep in mind, is that only a small percentage of micro instructions in the micro program flow actually require memory references; while most are crunching numbers and gating data internal to the CPU. Hence, the majority of machine cycles are available for DMA use.

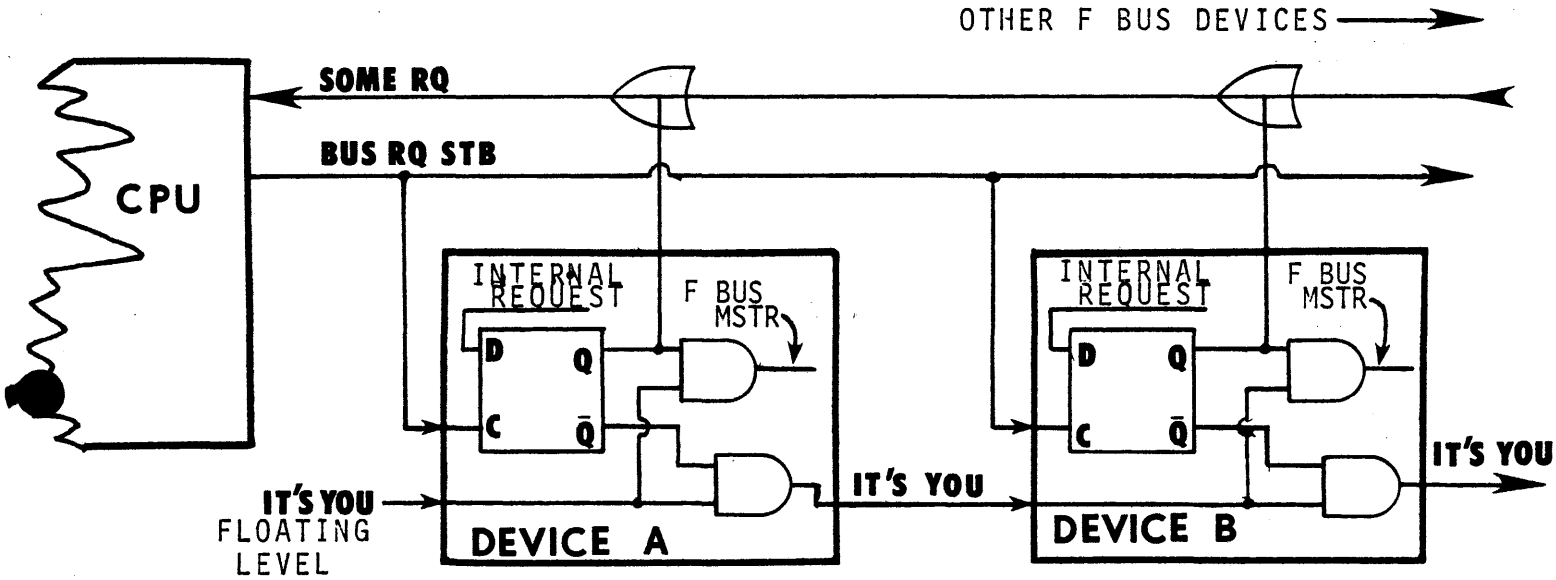
Another condition which will cause DMA operations to be temporarily suspended is when the CPU does an IOT operation. In this case any DMA requests will yield the F Bus to the CPU for the duration of the IOT while the request for DMA remains pending, to be acknowledged upon completion of the IOT.

Both F BUS and memory cycles are delayed 100nS from CPU machine cycles. By virtue of this convention DMA write cycles require only one cycle to gate information from a controller into main memory, while DMA reads are done in two cycles. The length of the cycles which are normally controlled by the CYLEN field of the micro code are forced to a length of 550nS in the event of a DMA operation. The

only effect this has on any number crunching which the uCode may be involved in at the time of the DMA is to allow more time than what otherwise may be required for propagation allowances relative to any operation invoked by the uCode.

6.3.1 DMA priority

In the event that more than one F BUS device makes a DMA request, a determination must be made based on each device's priority assignment as to whose request to honor first. This priority is determined soley by the particular F BUS slot into which devices are plugged. Figure 6-7 illustrates how this convention is implemented. A device will set its INTERNAL REQUEST when it is ready to read or write data.



[F BUS Priority Arbitration]

Figure 6-7

BUS REQUEST STROBE clocks a flop which sets SOME RQ when INTERNAL REQUEST is true. BUS RQ STB will occur every CPU CLK, unless SOME RQ already has been set. Therefore the BUS RQ STB which causes SOME RQ to set will inhibit further BUS RQ STROBES. At this point the first physical device on the bus with its SOME RQ flop set will become F BUS MASTER by virtue of the daisy chained signal named IT'S YOU. This signal is actually floating going into the first physical device slot which in Figure 6-7 is device A. Thereafter, provided that device A does not have its SOME RQ flop set it will be gated on down the F BUS to device B etc.

6.3.2 DMA write

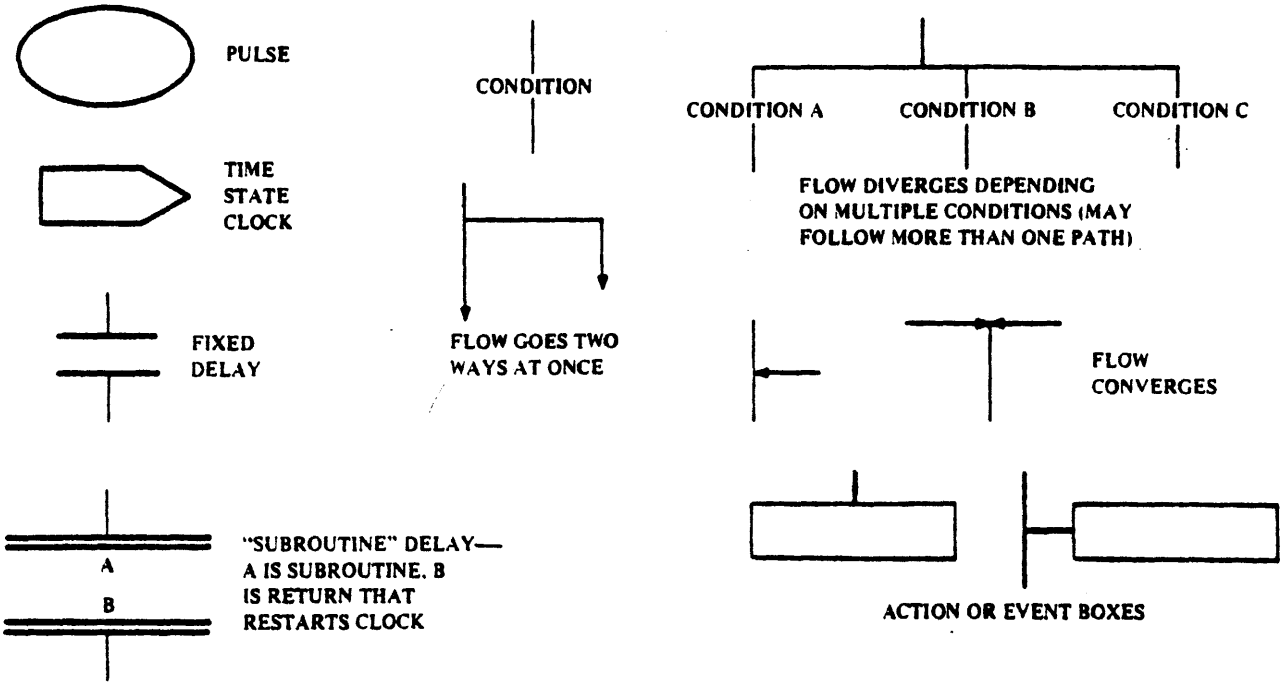
DMA write operations are initiated by an F BUS device whenever it has data to deposit into MAIN MEMORY. As a typical example the CPU under program control will issue several commands to the device in the form of IOT transfers. The commands will specify the location on mass storage of a block of data, the initial address in main memory, and that it is to be a DMA WRITE. Once these commands have been issued the CPU is free to go about other business, meanwhile the F BUS device (normally a periphial controller) will command the specific periphial to locate the beginning of the deisired data block and begin reading it into a FIFO buffer. Once the F BUS device has data ready to be sent to main memory it will raise its internal request and will begin arbitration according to priority as previously discussed. Assuming that no other requests are pending from other F BUS devices or that the particular device being discussed is the highest priority; it will then become F BUS MASTER. Refer to Figures 6-8 and 6-9 and observe that once F BUS MASTER, the device will drive the initial memory address;having been previously loaded into a register by an IOT OUT transfer,onto IOA lines 15:35. The address will be gated onto the PHYSICAL MEMORY ADDRESS lines (PADR) between the occurance of CPU CLK and BUS CLK. If the CPU was attempting a memory request or an IOT the DMA cycle will remain pending for one or more cycles, and the above operation and subsequent operations will be delayed accordingly. With the occurance of BUS CLK the memory cycle will begin and DATA will be transmitted from the F BUS device onto the F BUS IOD lines 00:35 and the CPU will gate the IOD lines onto the MEMORY OUT lines for storage. The next cycle will once again be available for a DMA cycle and the same process repeats itself.

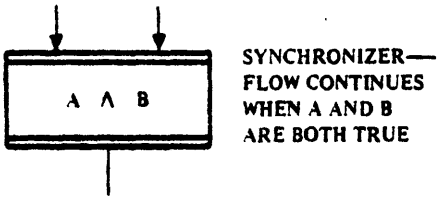
6.3.3 DMA read

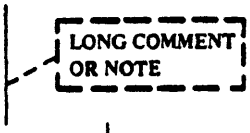
While it was possible to perform a DMA WRITE operation in only one machine cyle a DMA READ will requrire two machine cycles. During the first cycle the device supplies a main memory address to the CPU and the CPU uses this address to retrieve the DATA from memory and latch it internally. The second cycle will send the data onto the device. The major reason that two cycles are necessary comes from the fact that DATA is received by the CPU from MEMORY too late in the machine cycle to be sent onto the Controller on the F BUS, and hence another cycle is required for this task.

During the first cycle of a DMA READ a great deal of similarity exists with the exception that a MEMORY READ will be done. Priority arbitration remains unchanged, and provided that the CPU desires neither a MEMORY RQ or an IOT transfer the cycle of events will continue. Otherwise the DMA operation will be delayed until this condition is satisfied whereby the DMA OPERATION will continue.

FLOW CHART KEY



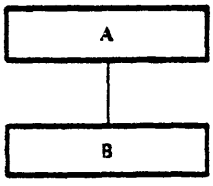




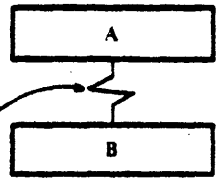
FLOW CONTINUES AT
ENTRY POINT D

FLOW CONTINUES FROM
EXIT POINT D

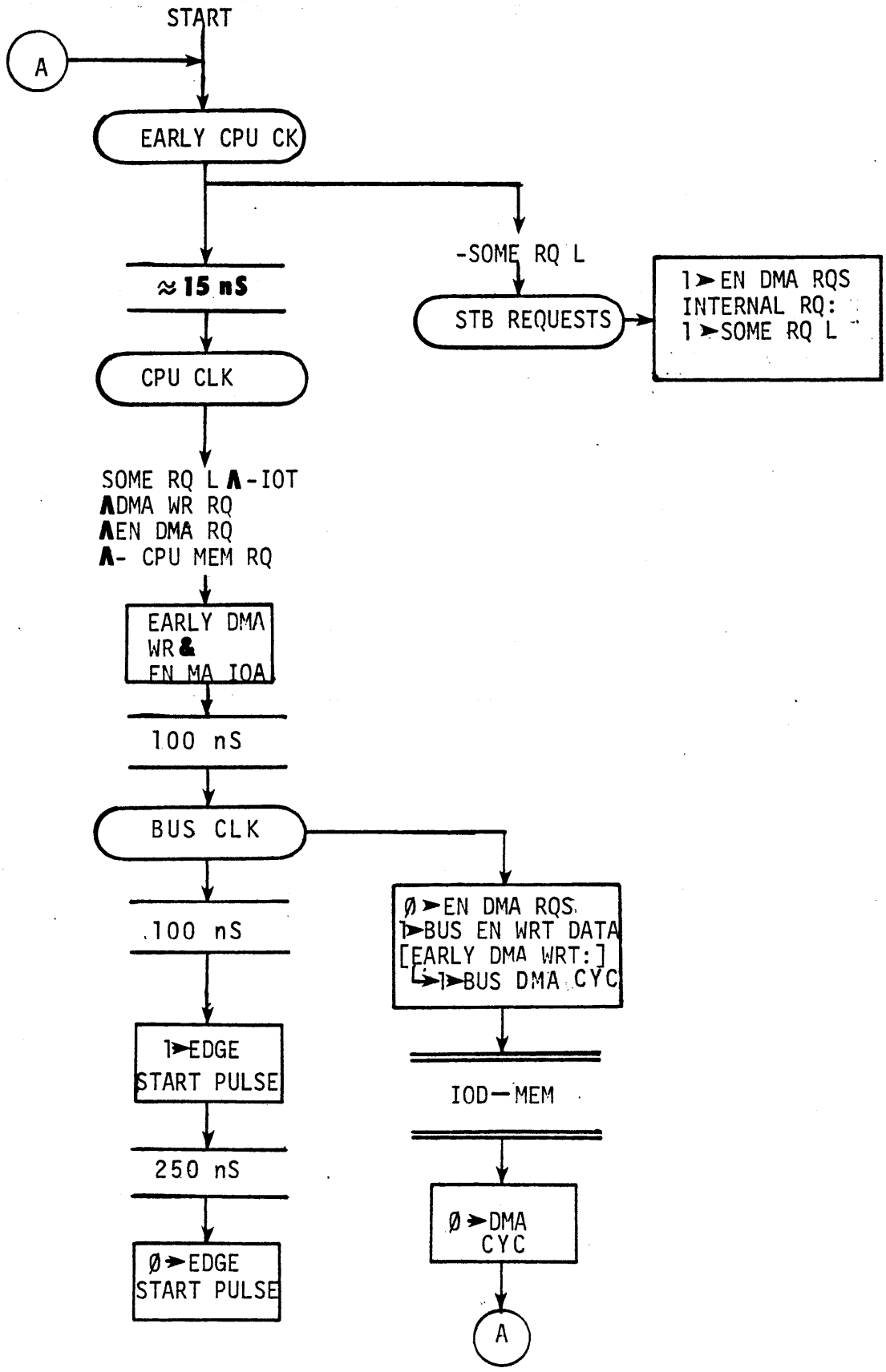
A:B	IF A IS TRUE, B OCCURS
A→B └→C	A IS TRUE AND HENCE B AND C ARE BOTH TRUE
[A:]	A IS A CONDITION GUARANTEED TO BE TRUE
A └→∧B:C	A IS TRUE, AND IF B IS ALSO TRUE, C OCCURS
[A]	A OCCURS BUT IS IRRELEVANT
: COMMENT	
( ) [ ] { }	USED FOR GROUPING IN FLOWS AND LOGIC DRAWINGS (IN LATTER, BRACKETS ALSO ENCLOSE EXPLANATORY SYMBOLS ACCOMPANYING A LOGIC SIGNAL)



EVENT A CAUSES EVENT B, OR
A BEING TRUE CAUSES B. MAY BE
DC LOGIC OR EVENT B TRIGGERED
BY TRANSITION IN A, PROVIDED
NEED FOR TRANSITION IS OBVIOUS
AS THROUGH A PA. IF TRANSITION
IS NOT OBVIOUS, THIS SYMBOL IS USED INSTEAD



10-0929



DMA WRITE CYCLE

Figure 6-8

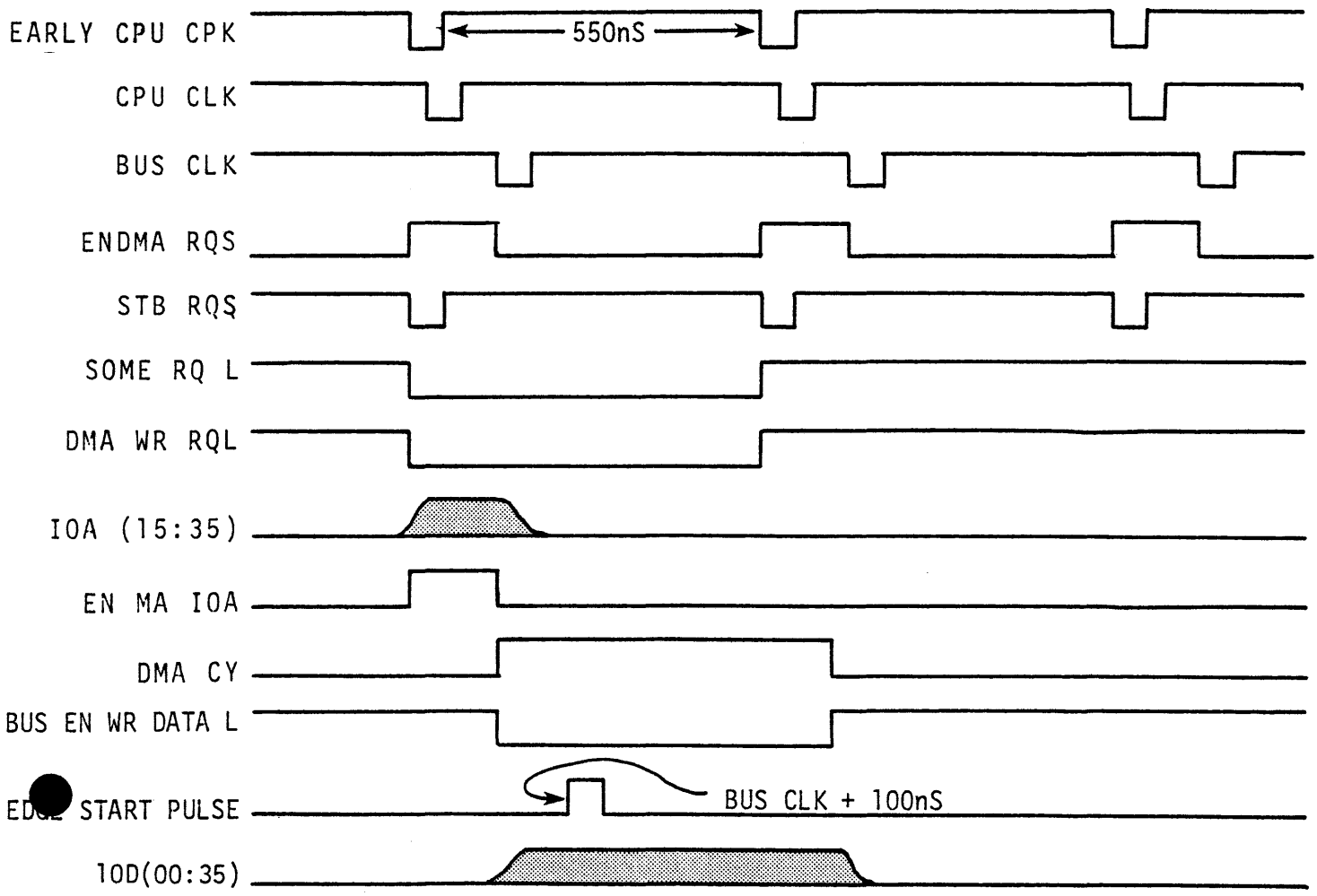


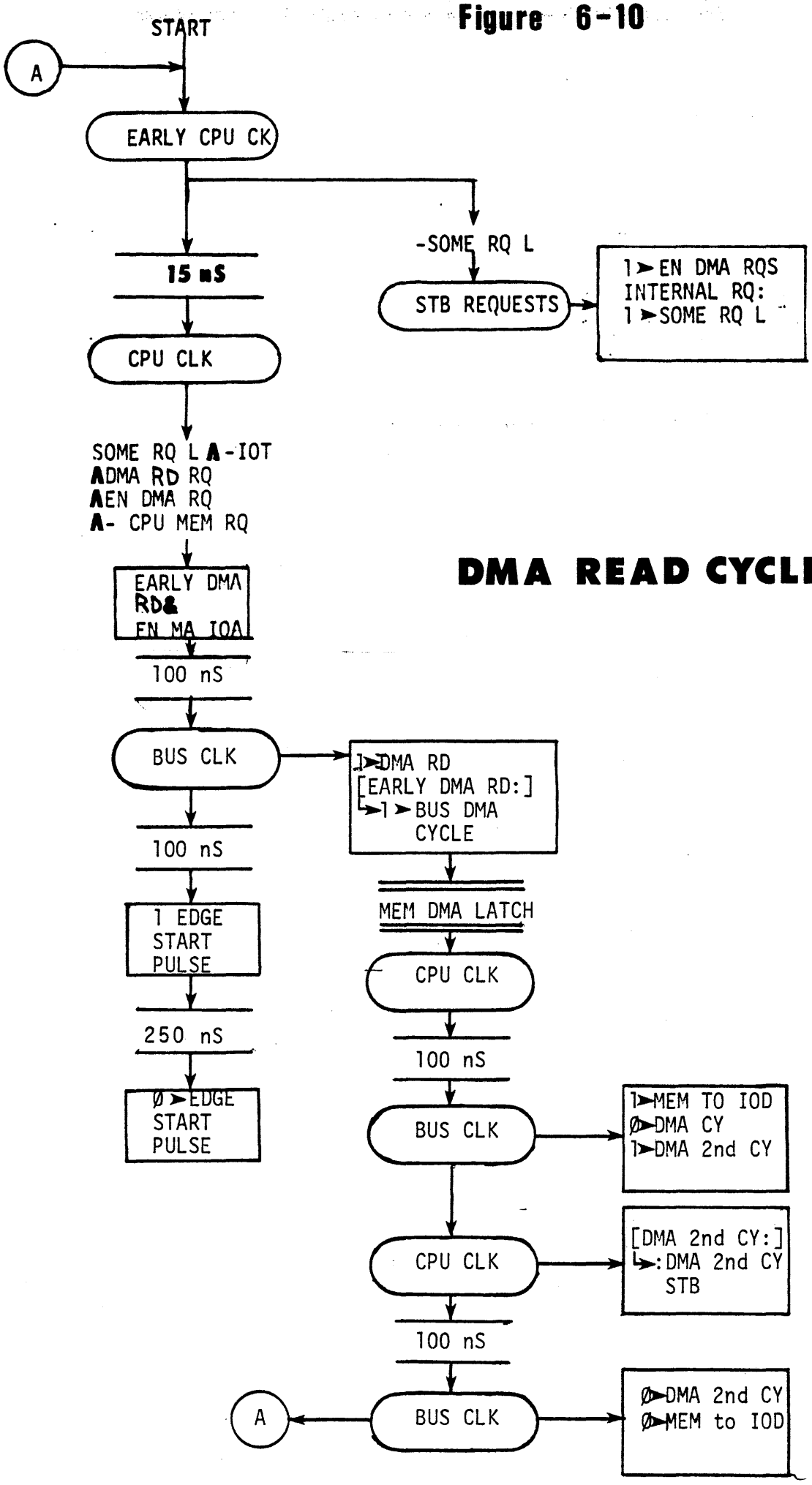
Figure 6-9
[DMA WRITE CYCLE TIMING DIAGRAM]

If the CPU sees that it is doing a DMA READ it will drop DMA CYCLE after the first cycle and bring up DMA SECOND CYCLE. With DMA SECOND CYCLE = 1, the next CPU CLK will cause DMA SECOND CYCLE STROBE whose purpose is to latch the DATA intoF BUS device from the IOD lines. Both figure 6-10 and 6-11 show timing and flow of events in detail for DMA READ operations.

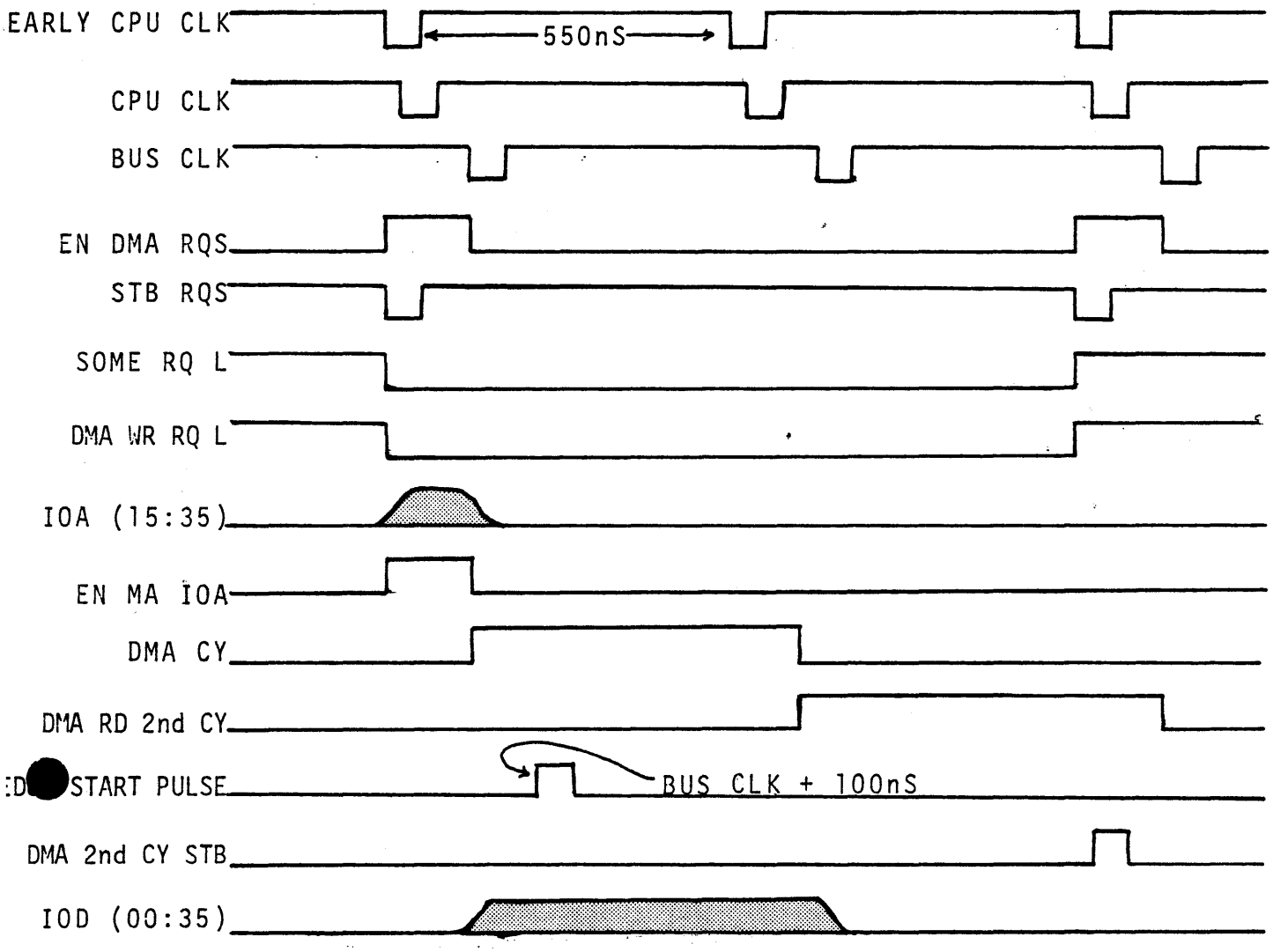
6.4 INTERRUPTS

Interrupts provide the means by which I/O devices may alert the CPU to various conditions which exist from within the device. These conditions may include errors, and completion of data transfers. For purposes of understanding the hardware involved with priority interrupts we shall segregate the logic into two subjects; that of arbitrating a request, and that of servicing and dismissing the request.

Figure 6-10



DMA READ CYCLE



[DMA READ CYCLE TIMING DIAGRAM]

Figure 6-11

The arbitration process will allow a priority scheme to determine which device will get the processors attention first in the event that more than one device makes a request at the same time. However once the processor honors a particular request, other requests are locked out (even if they are from a higher priority device). In other words further arbitration is inhibited until the dismissal of the interrupt being serviced. For this reason most interrupt service routines are kept shorter than 30 or 40 micro-seconds.

Servicing interrupts is controlled by the micro code. During a dispatch where by the micro code is going to branch to service the next instruction in the macro instruction sequence, the interrupt flop is tested and if found to be set the microcode will instead dispatch to a special location to begin handling the request. Upon completion of the service, or handler routine the microcode wil do yet another dispatch, at which time if there are no additional interrupts the dispatch will branch the microcode according to the op-code of the instruction previously being considered. It is important to be aware that dispatching occurs preceding the execution of each macro instruction.

6.4.1 Interrupt arbitration

Should more than one device request an interrupt at the same time it will be necessary to assign a priority such that faster devices such as disk will be reconized before slower ones like mag-tape. The convention used for this purpose is almost identical to that which was incorporated into the DMA priority scheme, however the hardware for doing PI arbitration is completly seperate.

The signal BUS INT STB is sent out on the F BUS as an image of CPU CLK. This is a bussed signal which is used to synchronize the internal requests of different devices. Before a device may become the F BUS MASTER it must also have INT IT'S YOU OUT as a true condition. INT YOU OUT is a daisy chained signal, which establishes the priority of a particular device according to the physical F BUS slot into which it is plugged. This signal is actually floting as it enters the first device slot on the F BUS. This allows the device plugged into that slot to always become F BUS MASTER if its INTERNAL REQUEST is set when it receives a BUS INT STB.

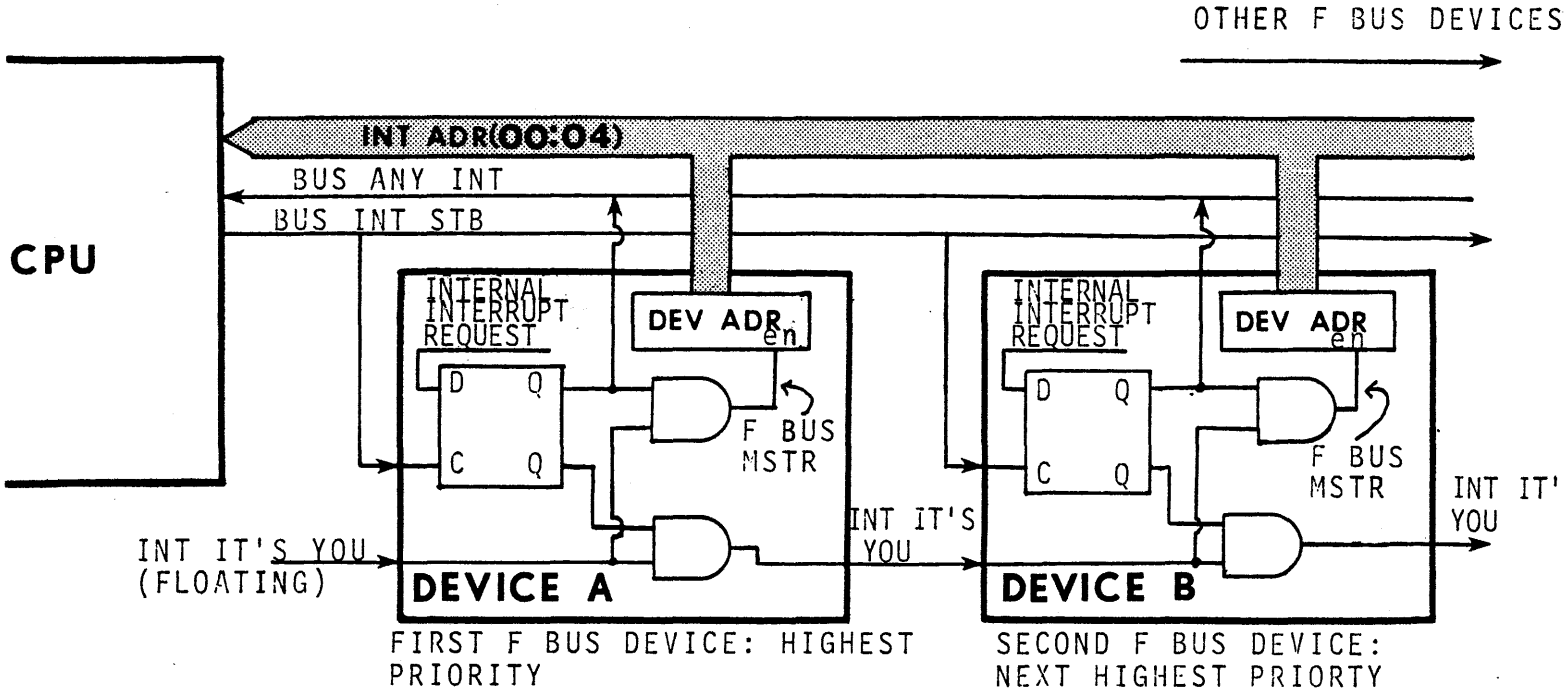


Figure 6-12

[INTERRUPT PRIORITY ARBITRATION]

Refering to figure 6-12, let us assume that device B has an internal request, and that device A has no request. At the occurance of BUS INT STB device B will become F BUS MASTER, by virtue of the presence of INT IT'S YOU OUT. If device A had an internal interrupt request the daisy chained signal INT IT'S YOU OUT would have been inhibited from being passed along to device B. Conversly if device B had not had an interrupt request set, then this would cause INT IT'S YOU OUT to be gated on to the next device and so on. Since in our example device B is interrupting, it will set its BUS ANY INT flop whose primary function is to alert the CPU and inhibit further BUS INR STB pulses from the CPU. The devices ANY INT signal is anded with INT IT'S YOU OUT to generate the internal signal F BUS MASTER, which gates the device address onto the INT ADR lines (00:04). The signal ANY INT will be turned off during the interrupt service routine which is tending to the interrupting device. If however more than one device had requested interrupts at the same time the ANY INT signal would remain on in the lower priority device untill it too was serviced. When the last device is finished being serviced and the ANY INT line is finally cleared BUS INT STB will be re-enabled for synchronizing future requests.

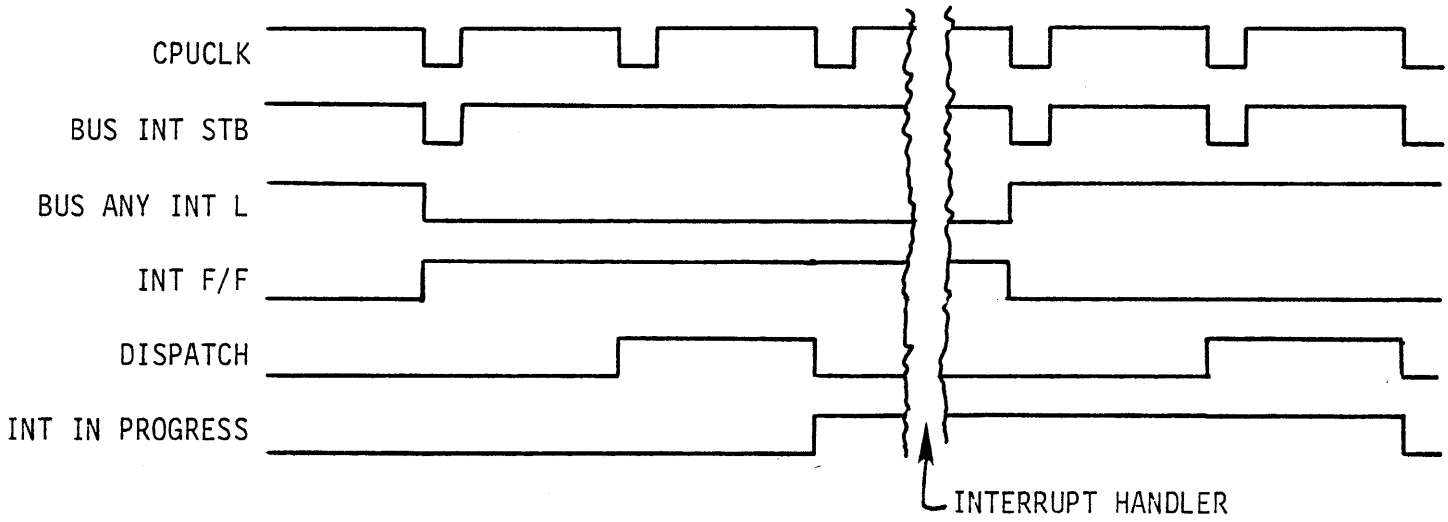


Figure 6-13

[SIMPLIFIED INTERRUPT TIMING]

6.4.2 Interrupt service

During each sequencial macro instruction, the micro-code will perform a conditional dispatch. This dispatch in its normal form, will branch the micro-code as a function of the op-code of the macro instruction contained in the IR (instruction registor). However, if when the dispatch occurs, the INT flop is set the micro code will instead be forced to location 2002.

After the arbitration process is completed the selected interrupring device will place its device address on to the INR ADR lines (00:04). This device address will be latched onto the DEV ADR lines (00:04), and will also address a block of 8 locations in A-MEM. The A-MEM contains 256 locations x 36 bits, and accordingly requires 8 bits of address. Of these 8 bits the high order 5 bits are derived directly from INT ADR (00:04) and are used to address one of 32 blocks. The remaining three low order address bits of the A-MEM are supplied from one of two fields within the micro instruction depending on whether data is being read from or written into the A-MEM it need only be concerned with selecting which word it wishes to reference.

As mentioned earlier the micro-code checks the interrupt flop during each macro instruction dispatch and if the interrupt condition is satisified the micro-code will be forced to location 2002. Once at this location the micro-code "still doesn't know" which device has caused the interrupt. In order that this be resolved the micro-code will do a SPECIAL DISPATCH (S DISP), unconditionally, vectoring from word Ø of the A-MEM block currently addressed by the interrupting device. The S DISP function is specified by a J CODE = $(15)_8$ and causes the next micro-code address to be loaded from the O BUS. In the instruction at 2002 we will also specify a function for getting word Ø onto the O BUS. This is accomplished by utilization of the 4 bit EXT D field whose high order bit being = 1 specifies the A-MEM as the selected source of EXTERNAL DATA SOURCE MIXER whose output feeds the O BUS via the ALU, the MASK, and the ROTATOR. The remaining low order 3 bits of the EXT D field are used to provide the three low order address bits to the A-MEM and thus select the word. Since word Ø is being used in the S DISP, the EXT D field = 1Ø). See figure 6-14.

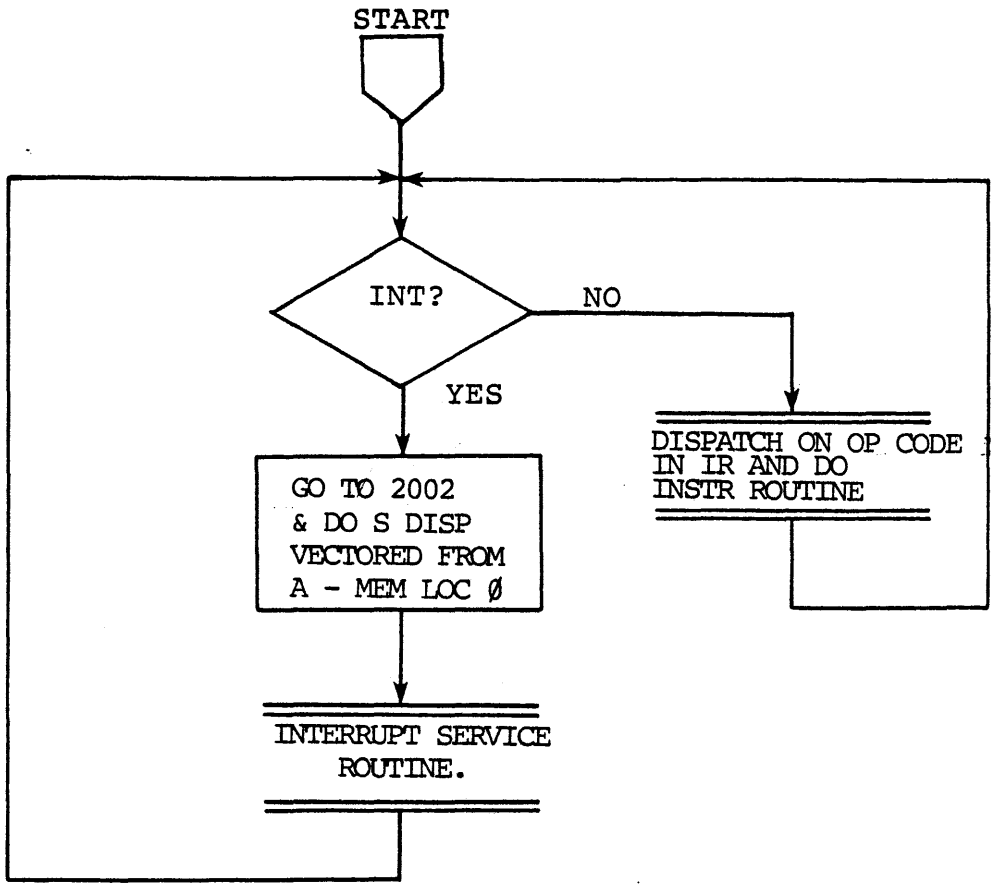


Figure 6-14

[SIMPLIFIED INTERRUPT DISPATCHING]

The Micro code will now be branched to a routine which is particular to the specific device. The other locations in the A-MEM Blocks may be utilized by the micro-code as a scratch pad in handling the devices interrupt. IOT's can also be performed to and from the device from A-MEM. Figure 6-15 and 6-16 show the logic for doing this. Note, that in performing the IOT's, the micro instruction need not be concerned with specifing the device address. This is made possible by having the device address latched for eternity from the INT ADR lines, which brings us to a very important point... how do we unlatch it at the end of the handler. This isn't shown in any of the figures however the micro-code must issue a CLEAR DEV ADR FORM INTERRUPT DEVICE command which is specified by DEST field = (12).

A simplified illustration of the micro-code and hardware implemented in performing an IOT OUT from A-MEM is shown in figure 6-15. This type of IOT is simpler than one from main memory in that we need not specify the device address. The SPEC field = (17) will cause an IOT OUT on the next machine cycle, while the EXT D

and DEST fields will select the A-MEM word and route the Data through the EXTERNAL DATA SOURCE onto the O BUS and strobe the IOD BUF to latch the data from the O BUS. The EXT D field selects the appropriate A-MEM word according to the three low order bits, while the high order bit causes the A-MEM to be selected as the input to the EXTERNAL DATA SOURCE. By setting the DEST field = (11) the IOD BUF will be strobed thus latching the data.

In a single IOT out the second micro instruction need only specify the device subselect code as an image of the MAP field. In the event that successive IOT outs are to be performed they may be over lapped by keeping the SPEC field = (17).

Figure 6-16 shows the IOT IN operation to A-MEM. IOT IN operations may <u>not</u> be over lapped. This is because the SPEC field must be used during the second micro instruction to write enable the A-MEM. Another apparent difference occurs in the A-MEM address which is derived from the low order 3 bits of the DEST field (all other bits in the DEST FIELD MUST BE ZERO).

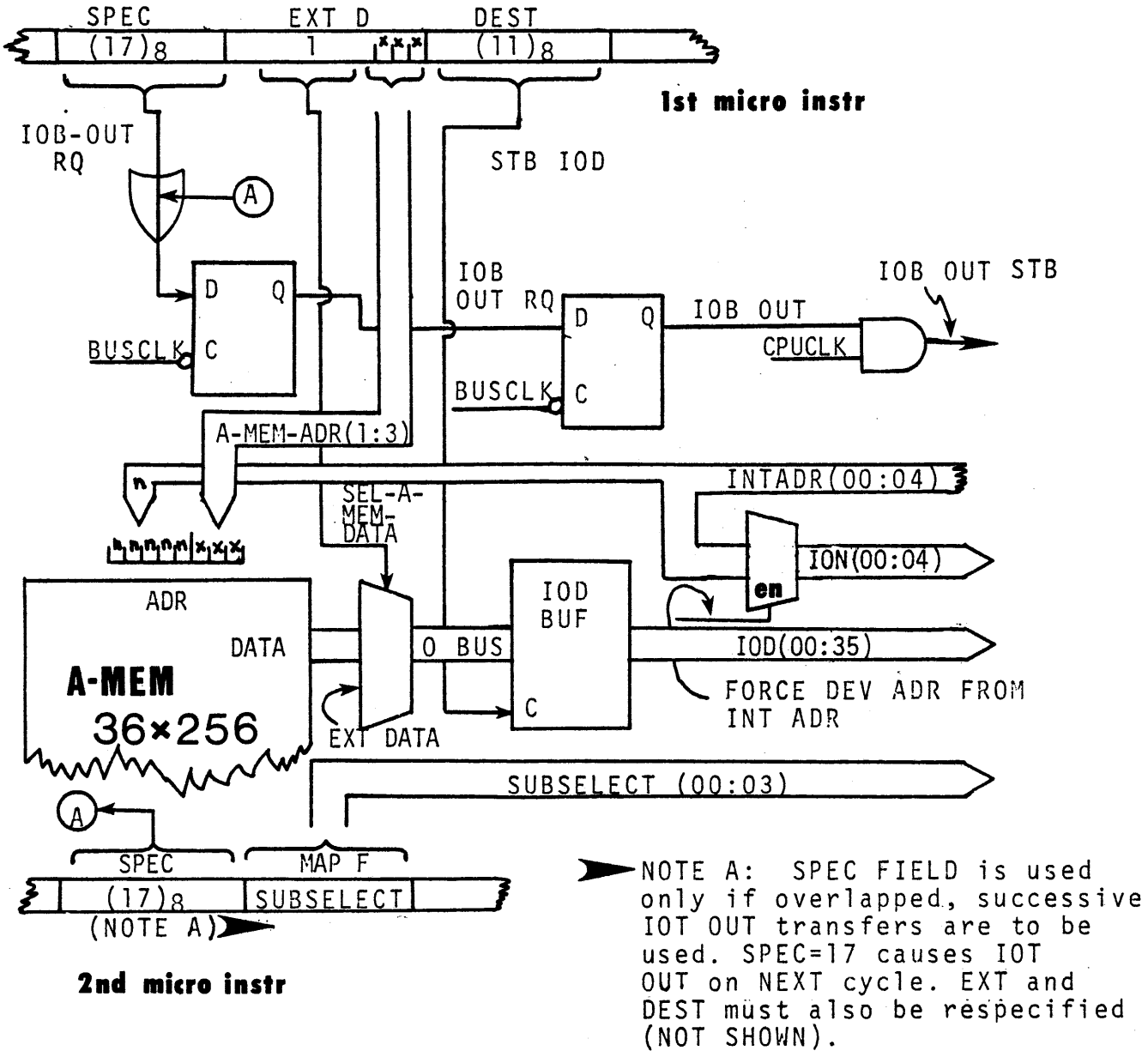


Figure 6-15

[IOT OUT DATA FROM A-MEMORY]

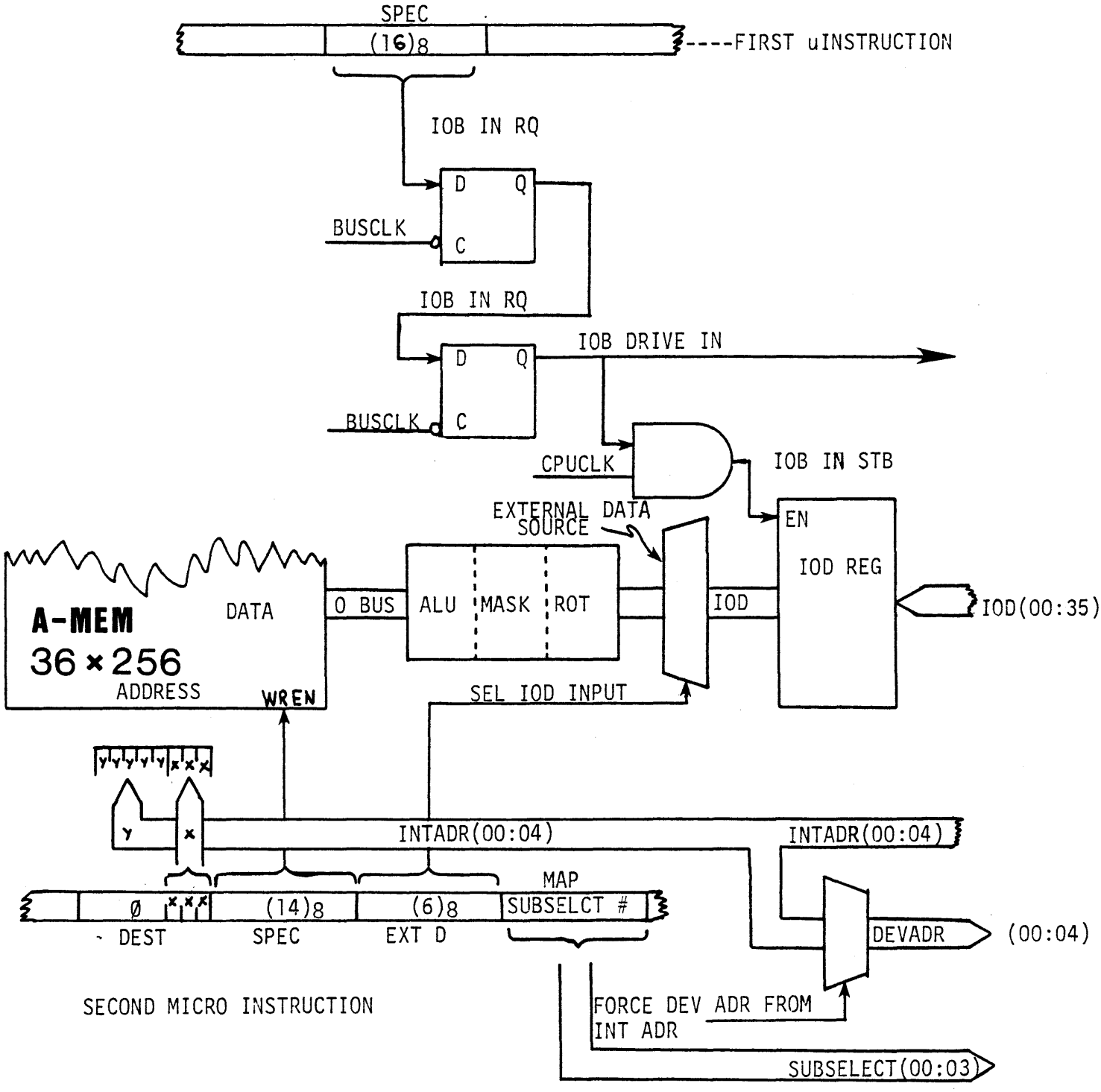
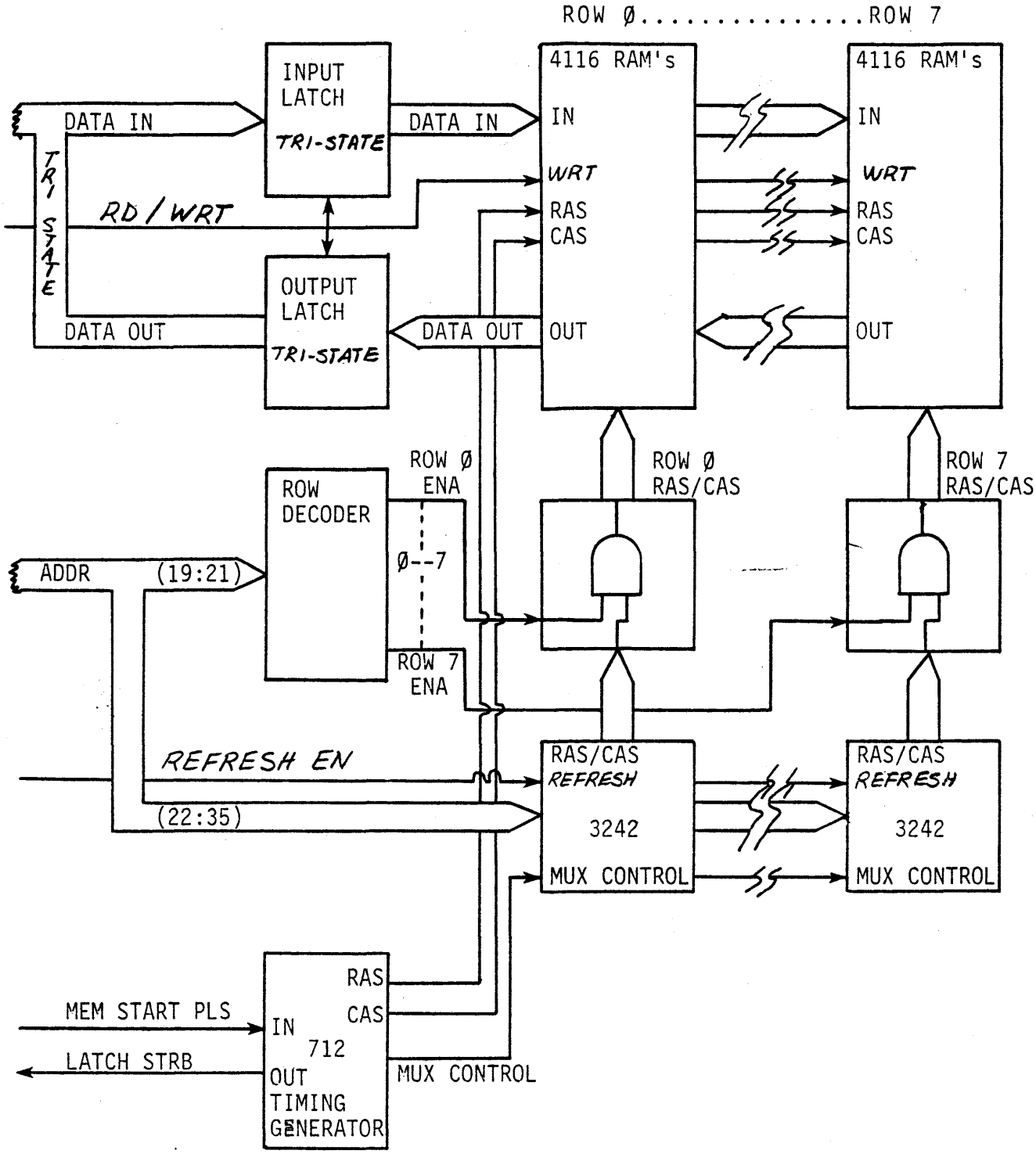


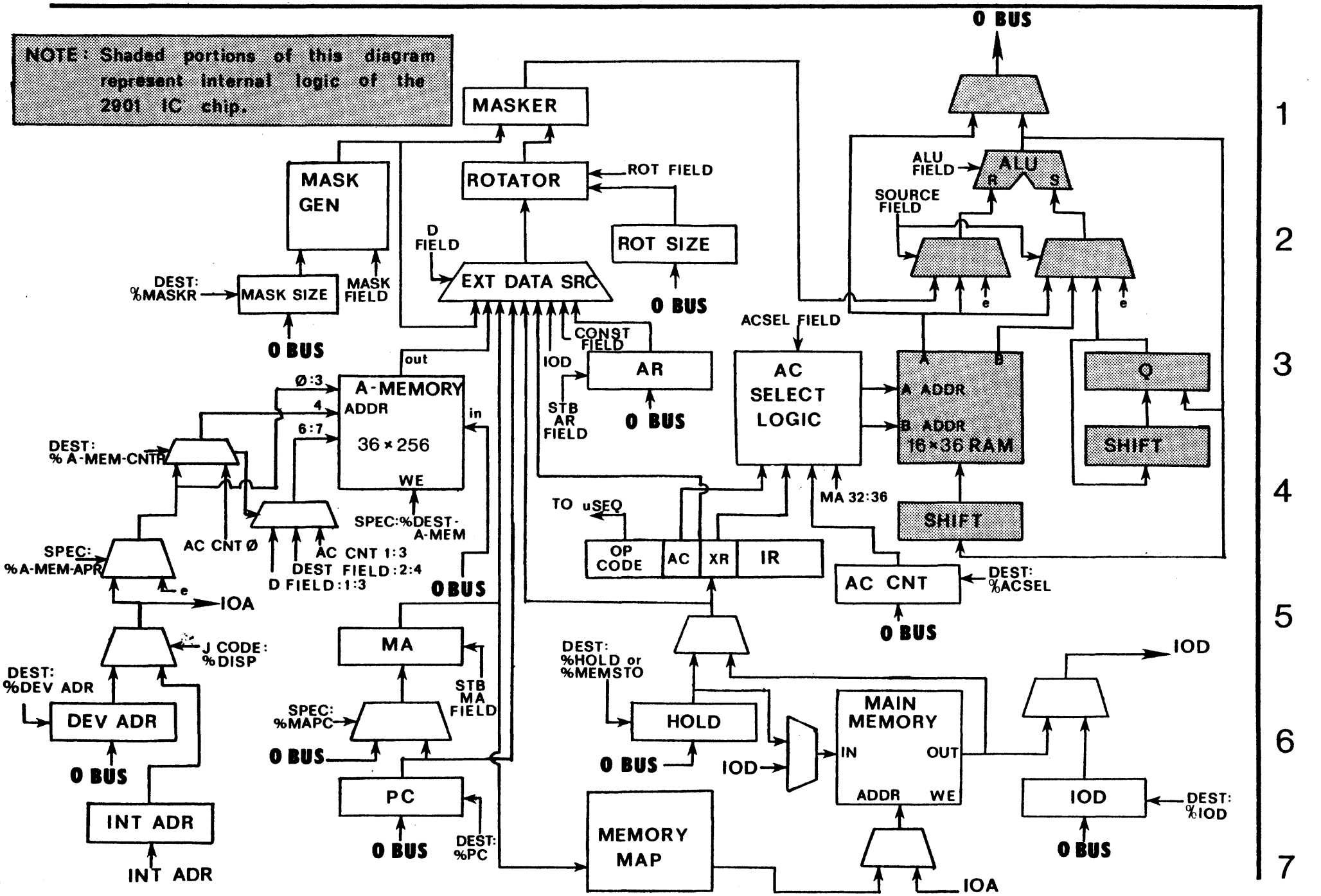
Figure 6-16

[IOT IN TO A-MEMORY]

XXV Main Memory

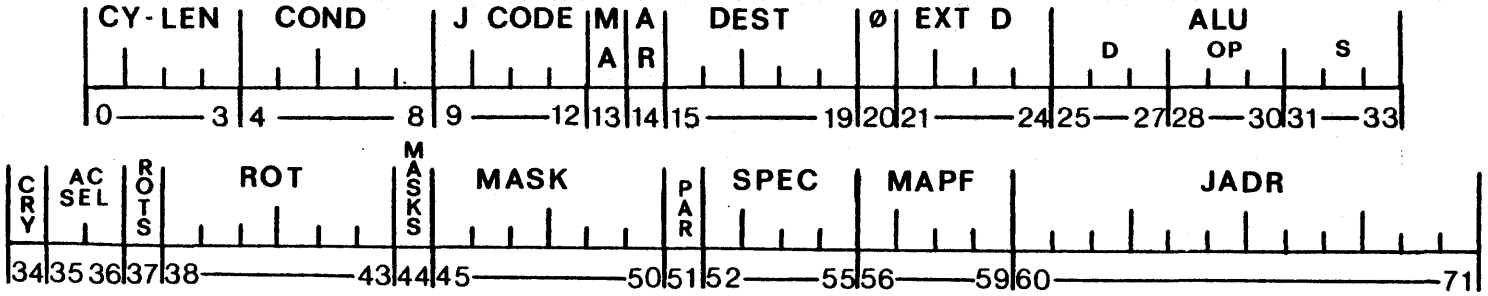


REVISED 1-21-81, F. HEBBEL



Foonly CPU BLOCK DIAGRAM

BY: FRED HEBBEL-TYMSHARE

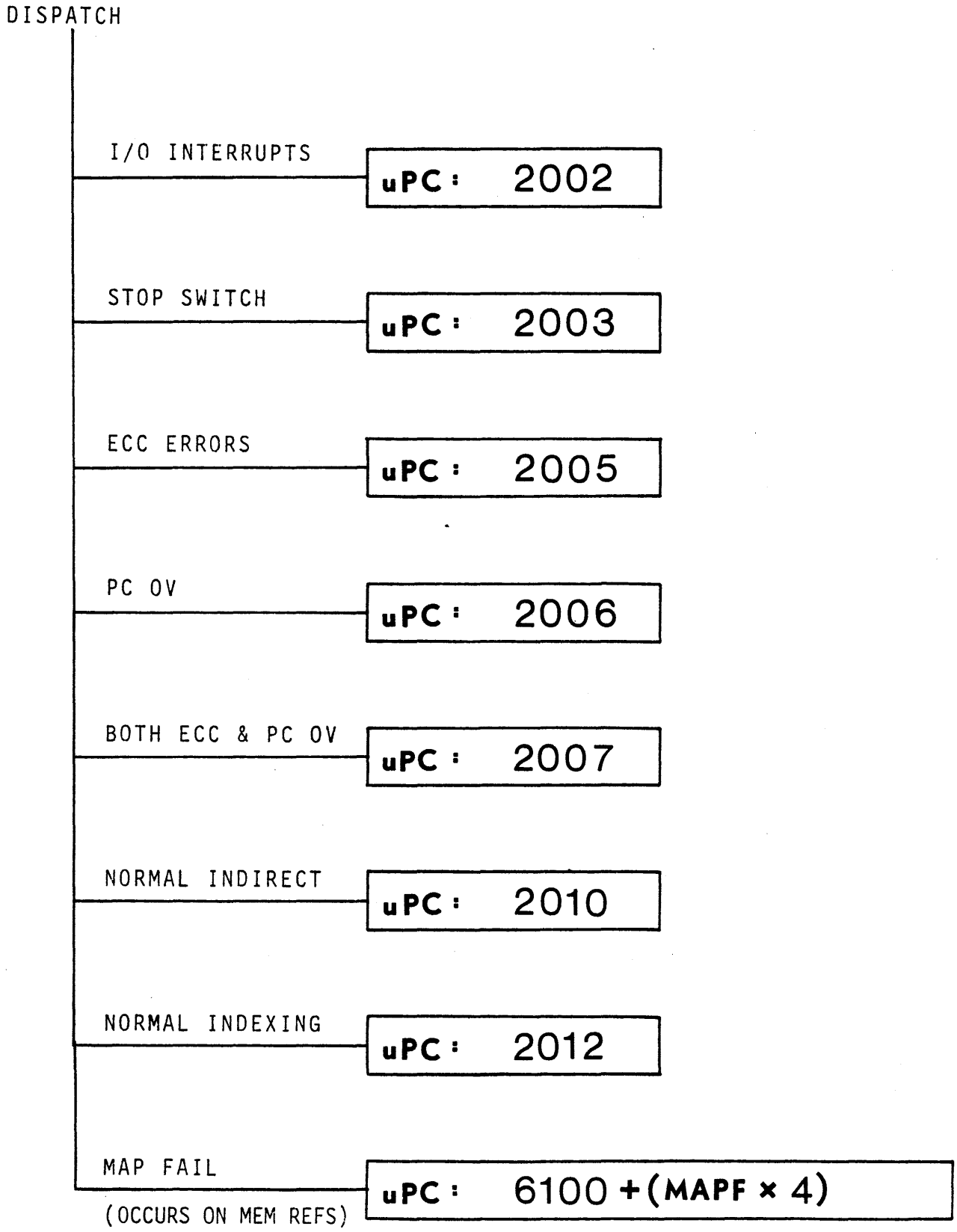


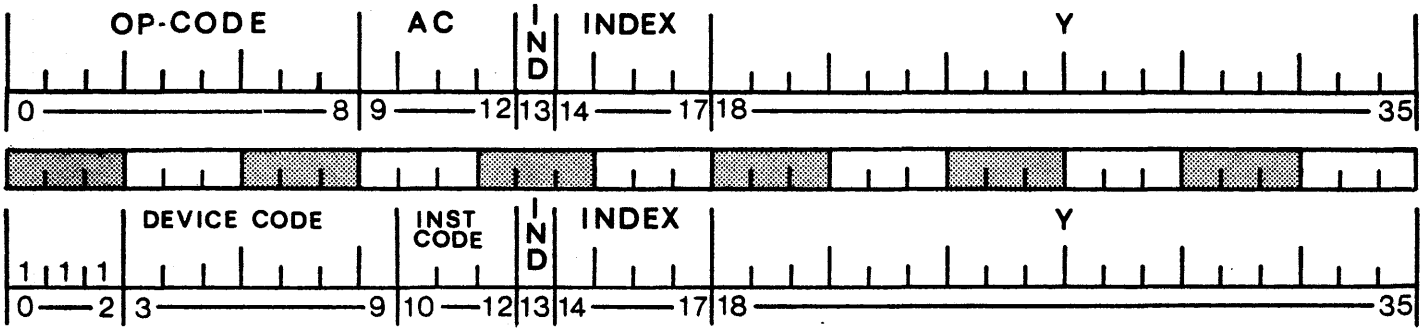
	COND	DEST	J CODE	True / False	EXT D	SPEC	CYLEN	AC SEL
0	TRUE	(NONE)	(NONE)	PC=Ø, RESET STACK	AR	(NONE)	LONG 1.05	A-MEM-CNTR
1	INTRPT	IR-ADR	PUSHJ	PUSH PC+1/PC:JADR	MEM	LEFT	1.00	IX MA
2	MA-AC	IR-23	LBJUMP	PC:JADR V 1/PC:JADR^-1	MASK	LEFT/MA:PC		AC
3	AC=Ø	IR-ALL	JUMP	PC:JADR/PC:PC+1	CONST	MA:PC	.90	AC+1
4	MEM-IDX-IND	AC-SEL			PC	CRYOV/MA:PC		
5	USER	ROTR	LOOP	note #1	MA	CRYOV	.80	
6		MASKR	POPJ	PC:STACK/PC:PC+1	IOD	PC+1 IF &		
7		DEV ADR	JPOP	PC:JADR/PC:PC+1	IR	PC+1 IF	C7ØØ .70	
10	0 BUS=Ø	MAP DISP		PC=Ø, RESET STACK	A-MEM	PC+1	C65Ø .65	
11	0 BUS Ø	IOD			A-MEM	CLR HALF	C6ØØ .60	
12	J COND	CLR-DEV:INT	DISP	note #2	A-MEM	A-MEM-APR	C55Ø .55	
13	0 BUS (18)	HI-ABS-MA	SDISP	PC:0 BUS/PC:PC+1	A-MEM	APR+DEST A	C500 .50	
14	Q(00:35)				A-MEM	DEST A MEM	C450 .45	
15	CRYØ		SLOOP	note #3	A-MEM		NORM .40	
16	HALF		CONT	PC:PC+1	A-MEM	IOB-IN	SHORT	
17	BYTE-OVF	CLR MI ERR	LLOAD	R:0 BUS, PC:PC+1	A-MEM	IOB-OUT	TOO SHORT	
20	FALSE	STRT WRT						
21	-INTRPT	MEMSTO						
22	-MA-AC	HOLD						
23	-AC=Ø	PC						
24	-IDX/IND	CLR MAP						
25	-USER	STO MAP						
26		CRYOV						
27		FIX EXC SR						
30	-0 BUS=Ø	MAPF RD						
31	-0 BUS Ø	MAPF WRT						
32	-J COND	FIXMAC						
33	-0 BUS(18)							
34	-Q(00:35)	AMEM CNTR +						
35	-CRYØ	A MEM CNTR						
36	-HALF	uCODE HI						
37	-BYTE OVF	uCODE LOW						

NOTE #1) if R=Ø then PC:PC+1
if R≠Ø, then PC:JADR,R=R-1
NOTE #2) if COND≠T, then
PC:(OPCODE x 2)+2000
NOTE #3) if R=Ø then PC:PC+1
if R≠Ø and cond=T, then
PC:JADR V 2, if R≠Ø and
cond≠T, then PC:JADR^-2

ALU			
	D	OP	S
0	Q	S+R	A,B/D,A
1	NONE	S-R	A,Q/Ø,A
2	Ø-AC	R-S	Ø,B/D,Ø
3	AC	RvS	Ø,Q/D,Q
4	B+Q=0/2	R^S	D,A/Ø,A
5	B=0/2	-R^S	Ø,A/D,A
6	B=0+2,Q=0+2	RvS	D,Ø/D,Q
7	B=0+1	R=s	D,Q/D,Ø

Foonly Microcode


DISPATCH
I/O INTERRUPTS
uPC: 2002
STOP SWITCH
uPC: 2003
ECC ERRORS
uPC: 2005
PC OV
uPC: 2006
BOTH ECC & PC OV
uPC: 2007
NORMAL INDIRECT
uPC: 2010
NORMAL INDEXING
uPC: 2012
MAP FAIL
(OCCURS ON MEM REFS)
uPC: 6100 + (MAPF × 4)


OP-CODE
AC
I
N
D
INDEX
Y
0
8
9
12
13
14
17
18
35
1 1 1
DEVICE CODE
INST
CODE
I
N
D
INDEX
Y
0
2
3
9
10
12
13
14
17
18
35