University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Interrupts and exceptions

Memory-mapped I/O

In memory-mapped I/O, interaction with the I/O devices is controlled by our program

- Our program polls ready bits of I/O registers to see if the I/O devices are ready for interaction
- This leads to inefficiencies since our program effectively stalls until an I/O operation is complete

Interrupt-driven I/O

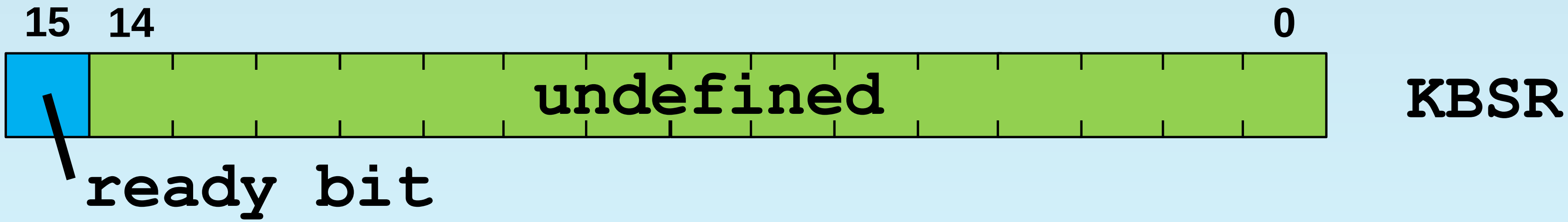
In interrupt-driven I/O, interaction with the I/O device is controlled by the I/O device itself

- An I/O device generates an interrupt signal to indicate that I/O device is ready with new I/O operation
- In response to this interrupt, the currently executed program stops its execution and the control is passed to some subroutine designed to handle the interrupt
- Once the subroutine processes the interrupt, the control is passed back to the program that was previously executed

Several things must be true for an I/O device to actually interrupt the processor

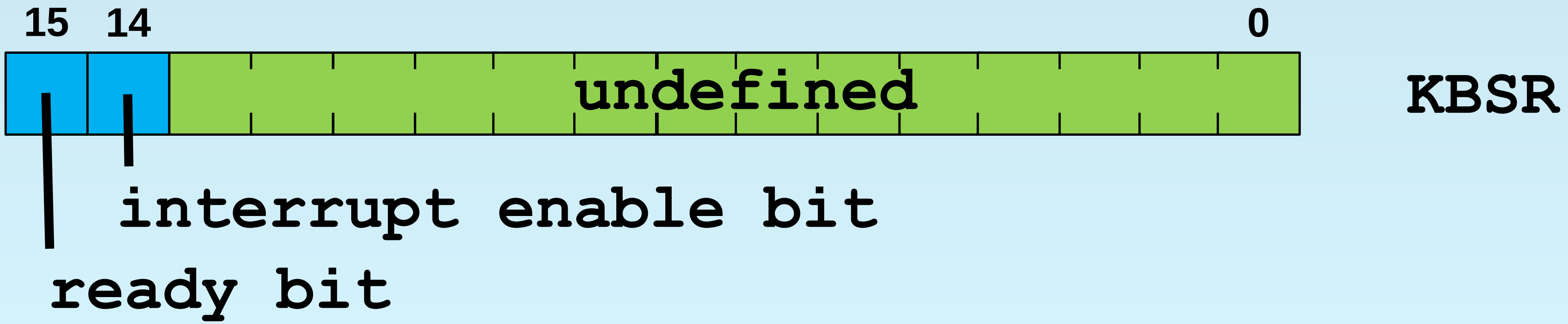
1. The device must request the service.

   This is indicated by ready bit (KBSR[15] and DSR[15]). If these bits are set, there is a new I/O request ready to be served



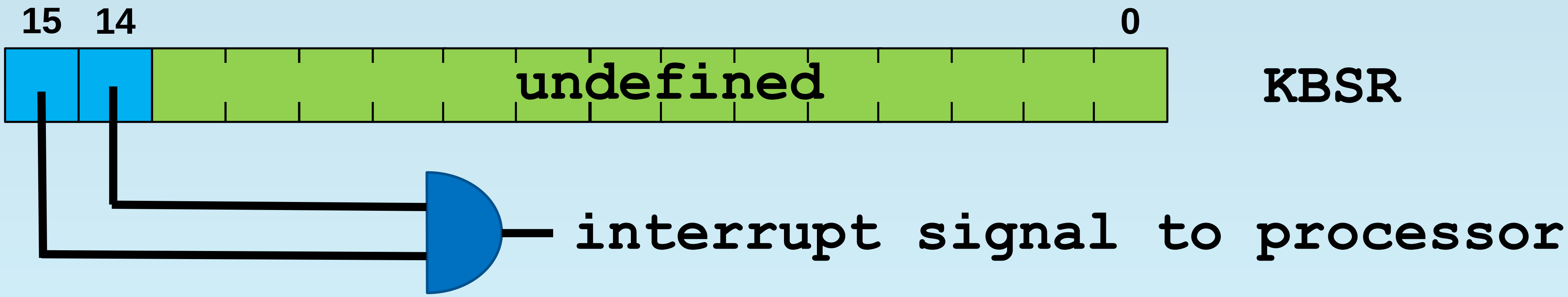
2. The device must have the right to request service.

This is indicated by an interrupt enable bit (KBSR[14] and DSR[14]). If such bit is set by the processor, the processor wants to give the I/O device the right to request the interrupt service



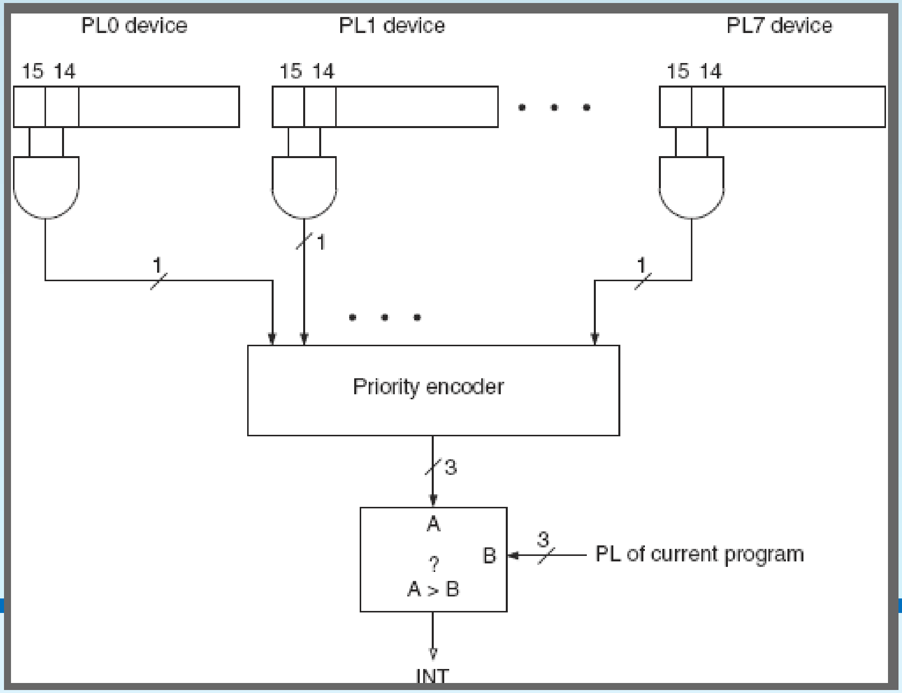
Interrupt Signal (INT)

Ready bit and interrupt enable bit together are used to generate an interrupt

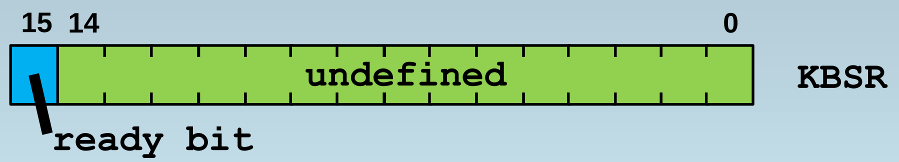


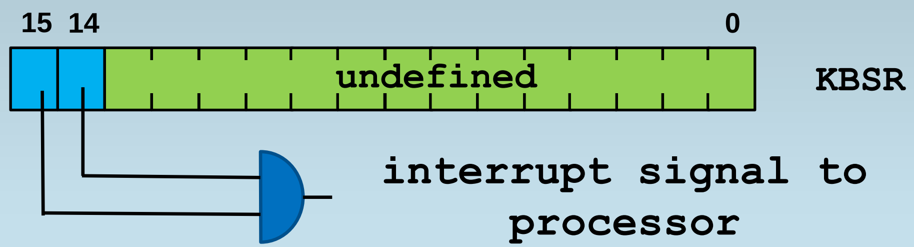
3. This request must be more urgent than the processor's current task.

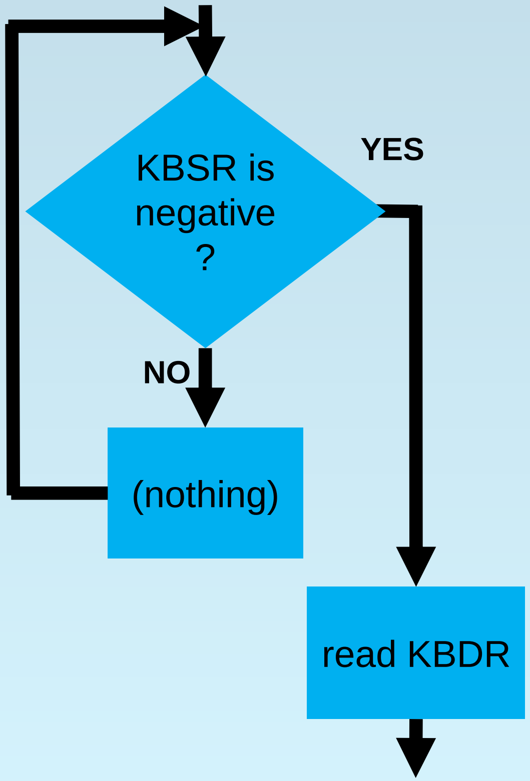
A program is executed with some specified priority level, LC-3 has 8 such priority levels PL0..PL7.



Polling vs Interrupt-driven I/O







IE = 0

- I/O device will NOT be able to interrupt
- Have to use polling

IE = 1

- Interrupt-driven I/O enabled
- Interrupt request generated as soon as Ready bit sets (e.g., a key is typed)

Flow of Interrupt-driven I/O

Stage 1: Initiate the interrupt

- 1.1 Stop the running program on any instruction
- 1.2 Save the state of the running program
- 1.3 Generate address of the interrupt servicing subroutine

Stage 2: Service the interrupt

- 2.1 Transfer control to the interrupt subroutine
- 2.2 Execute the interrupt subroutine

Stage 3: Return from the interrupt

- 3.1 Resume right where we left off

Stage 1: Initiating the Interrupt

An I/O device generates an interrupt signal (INT) to indicate that I/O device is ready with a new I/O operation (e.g., a new character has been entered on the keyboard)

I/O device presents an 8-bit interrupt vector (INTV) which is used to construct a memory address that contains the location of the interrupt handler in a interrupt table

Interrupt Priority

For an interrupt to be served, the request must be more urgent than the processor's current task

LC-3 priority levels are PL0-PL7

- Higher is more urgent, e.g., keyboard is PL4

LC-3 maintains an interrupt priority in PSR[10:8]

Devices wanting to interrupt also have a 3-bit priority

The interrupt will be served only when program is running at priority < PL4

LC-3 Interrupt Table

Each device is associated with an 8-bit vector INTV to index an interrupt vector table

Interrupt vector table is in memory between x0100 and x01FF

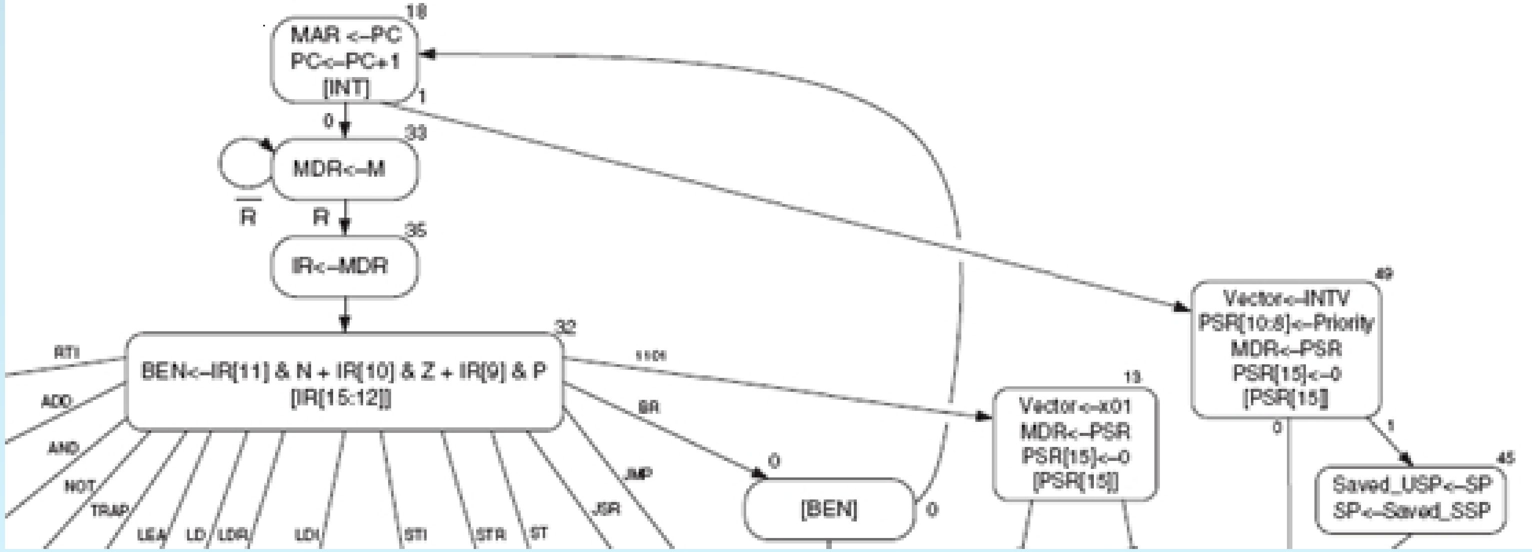
Each record in the interrupt vector table contains beginning address of service routine for handling interrupt

- Exception service routines (x0100-x017F)
- Interrupt service routines (x0180-x01FF)

1.1 Stopping the Execution of the Program

State 18 in LC-3 FSM is the only state in which the processor checks for interrupts

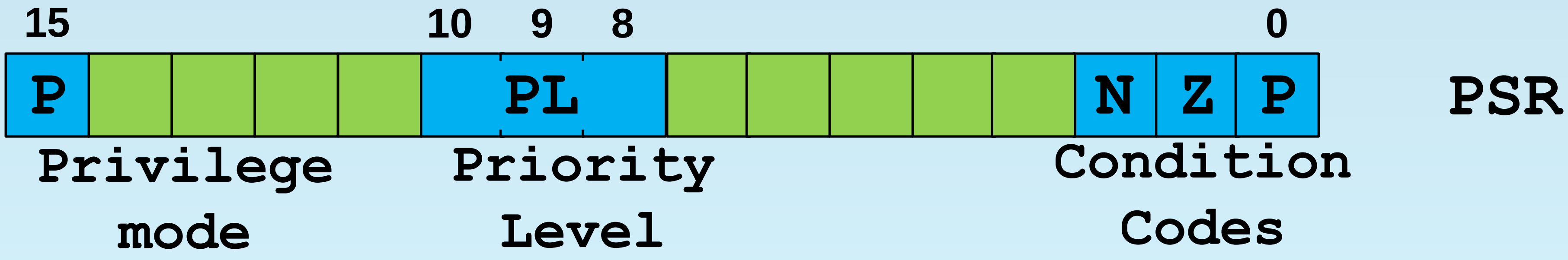
- If INT=0 (no interrupt) go to State 33
- If INT=1 go to State 49 (110001)



1.2 Saving the State - What

PC so that we can return to execute the next instruction after the interrupt has been served

Processor Status Register (PSR)



15 10 9 8 0

P PL N Z P

PSR

Privilege mode

Priority Level

Condition Codes

1.2 Saving the State - Where

Supervisor Stack - a special region of memory used as the stack for serving interrupts

Supervisor Stack Pointer (SSP)

- Saved.SSP: Internal register to store SSP

User Stack - a stack accessed by user programs

User Stack Pointer (USP)

- Saved.USP: Internal register to store USP

Access both stacks using R6 as the stack pointer.

When switching from User mode to Supervisor mode, save R6 to Saved.USP

1.3 Generating ISR address

Set MAR to x01vv, where vv is 8-bit interrupt vector (INVT) from interrupting device

- e.g., for keyboard INTV=x80, MAR ← x0180

Load from memory: MDR ← MEM[x01vv]

Set PC to MDR

LC-3 FSM for Handling an Interrupt

Load PSR to MDR in preparation for pushing into Supervisory Stack

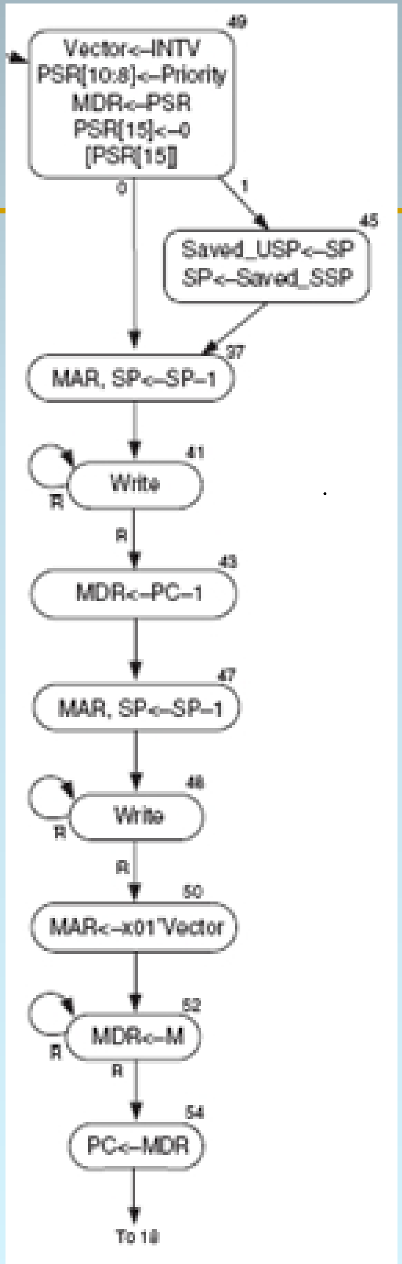
Record Priority Level and INTV provided by interrupting device

Test old PSR[15]

- If old PSR[15] = 1 then system was in User mode and hence save USP (R6) in Saved.USP, load R6 with Saved.SSP, go to state 37
- If old PSR[15] = 0 then system was in supervisory mode already

Save PSR, old PC to Supervisory Stack

Load PC with address of interrupt service routine



Stage 2: Service the interrupt

PC contains the starting address of the ISR

The ISR will execute, and the requirements of the I/O device will be served

- For example, copy KBDR into some memory location
- Callee-save for general purpose registers

Only Input from the keyboard interrupt is implemented on LC-3

Return from the Interrupt (RTI) Instruction

To return from ISR, we need special instruction, RTI

RTI is a privileged instruction

- Can only be executed in Supervisor mode
- If executed in User mode, causes an exception

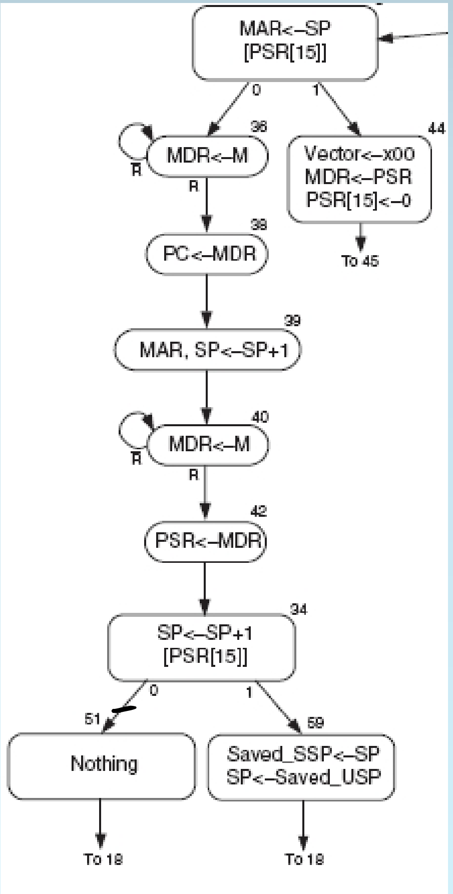
Stage 3: Return from the interrupt (RTI)

If PSR[15] = 0

- Restore PC and PSR
- Test old PSR[15]
  - If old PSR[15] = 1 then the system returns to User mode and hence restore USP (R6) and store SSP
  - If old PSR[15] = 0 then system continues to be in the Supervisory mode

If PSR[15] = 1 (Privilege Mode Exception)

- Handle condition as a privileged mode violation
- Load Interrupt Vector with starting address of Privilege mode violation
- Go to State 45 to handle interrupt as if by INT



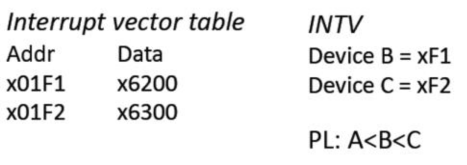
Exception Handling

Only exceptions from

- Privilege mode violation
  - If processor encounter RTI when in User mode
- Illegal opcode
  - If IR[15:12] = 1101 is true (unused opcode)

Exception handling is similar to interrupt handling

Nested Interrupts



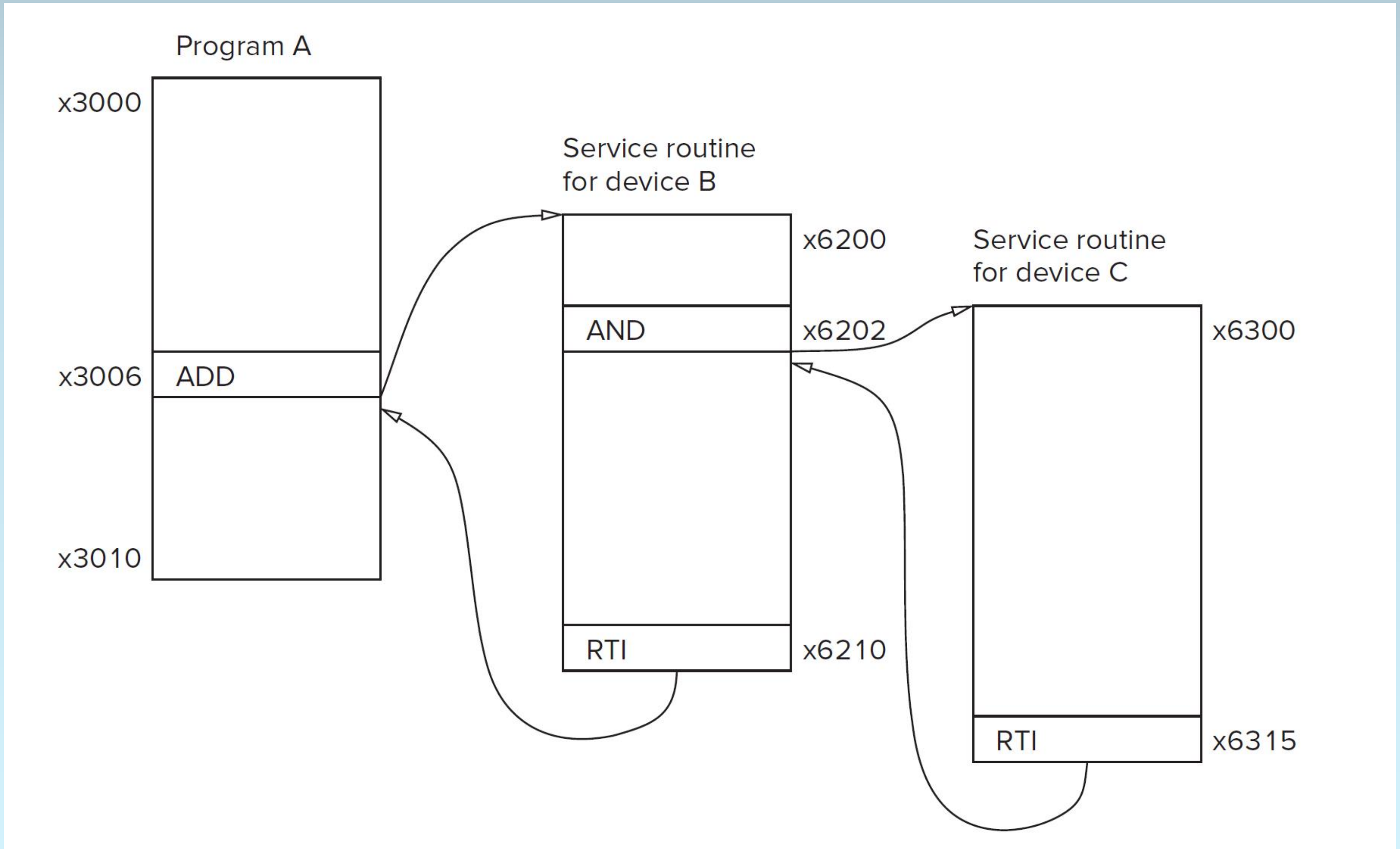
Interrupt vector table

Addr	Data
x01F1	x6200
x01F2	x6300

INTV

Device B = xF1
Device C = xF2

PL: A<B<C



Interrupt vector table

Addr	Data
x01F1	x6200
x01F2	x6300

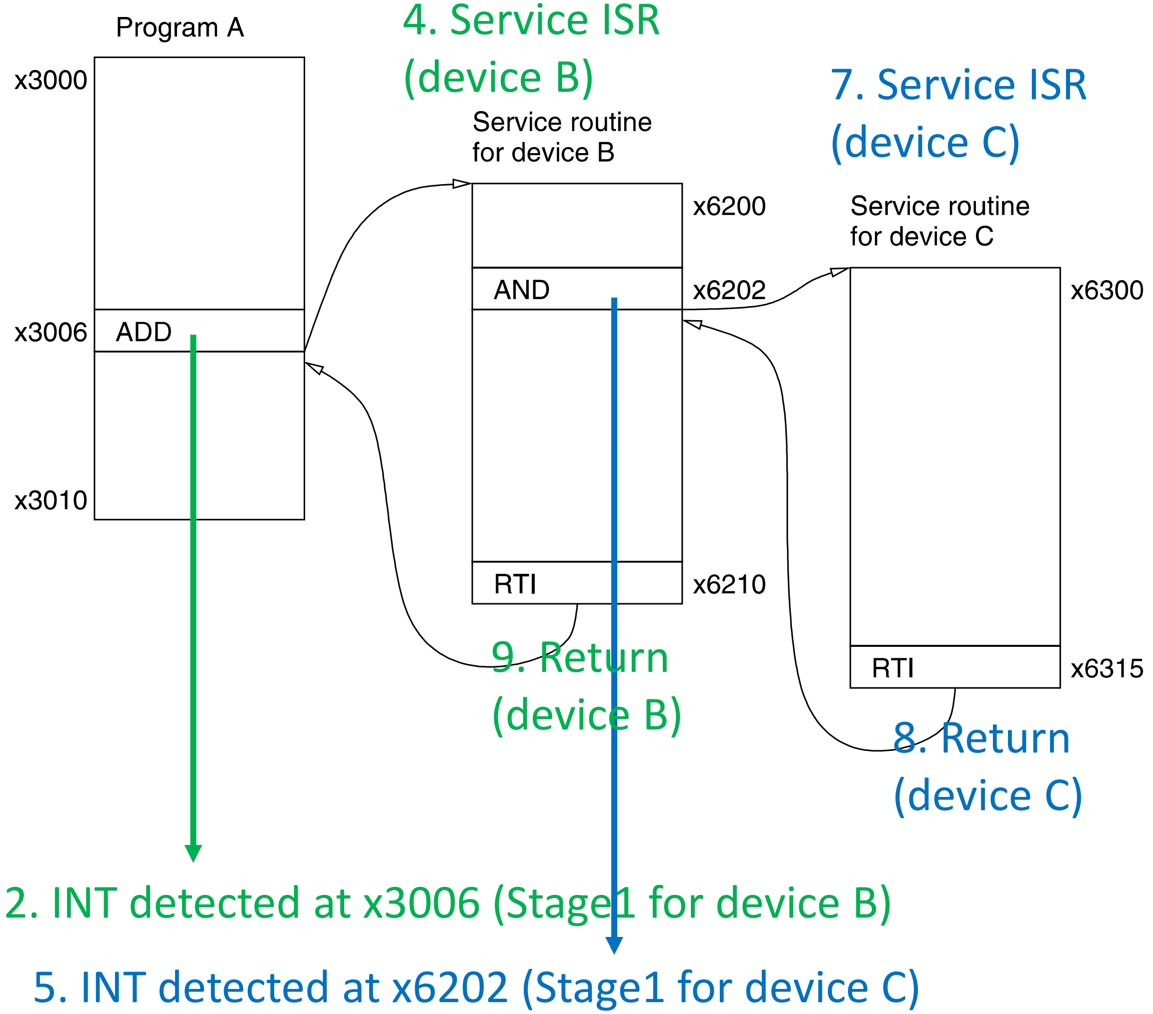
INTV

Device B = xF1

Device C = xF2

PL

A<B<C



4. Service ISR (device B)

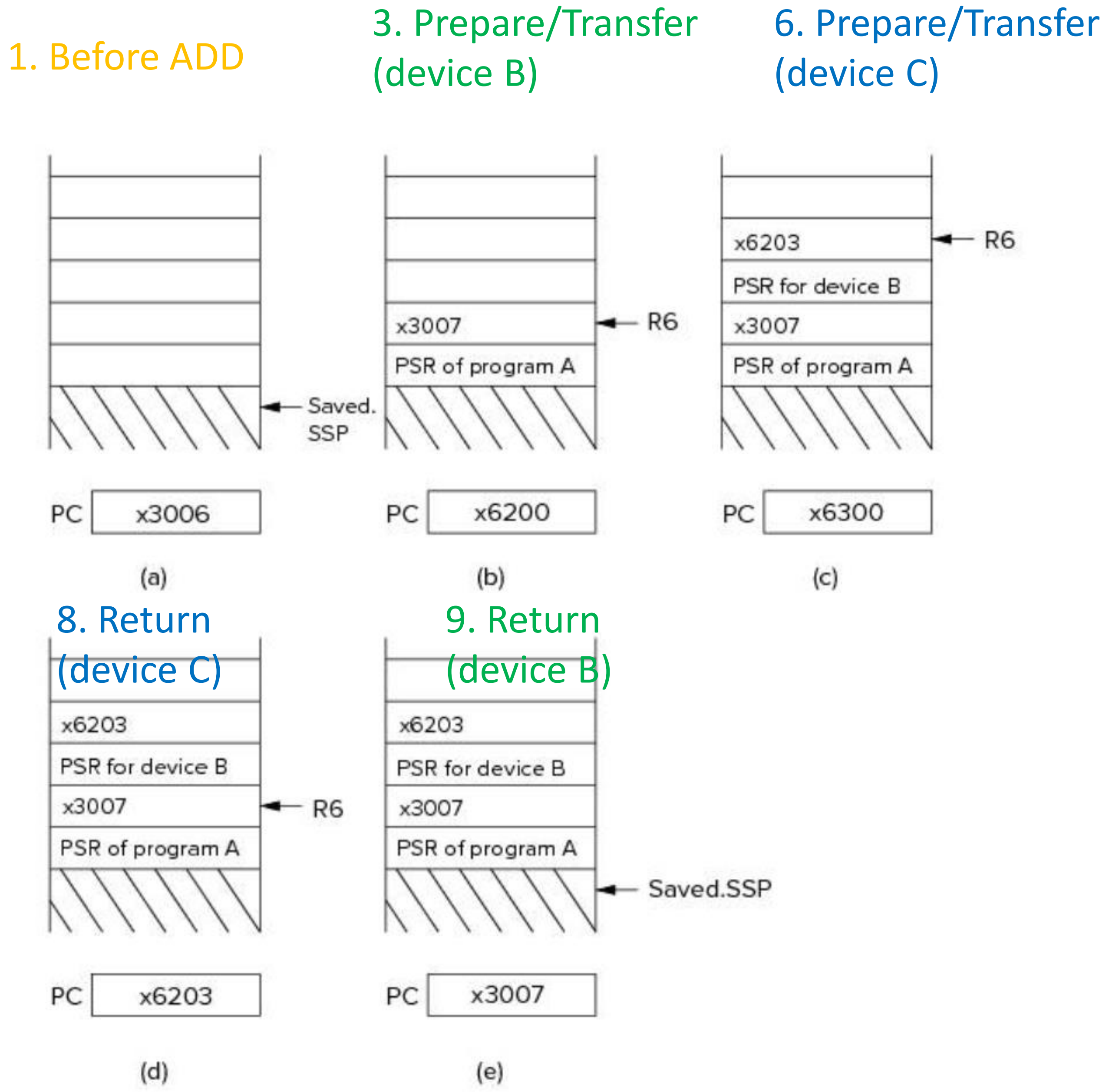
7. Service ISR (device C)

9. Return (device B)

8. Return (device C)

2. INT detected at x3006 (Stage1 for device B)

5. INT detected at x6202 (Stage1 for device C)



1. Before ADD

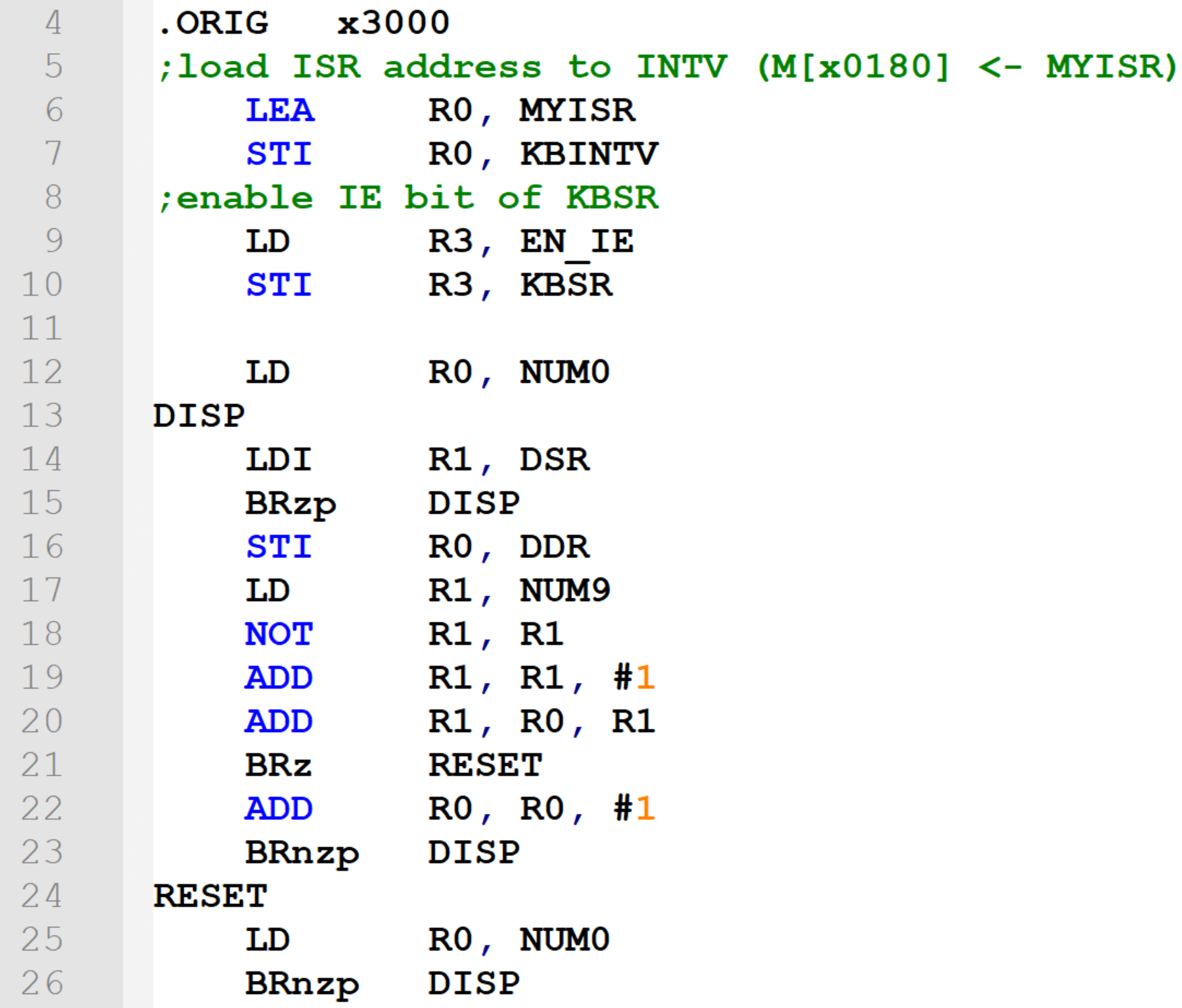
3. Prepare/Transfer (device B)

6. Prepare/Transfer (device C)

8. Return (device C)

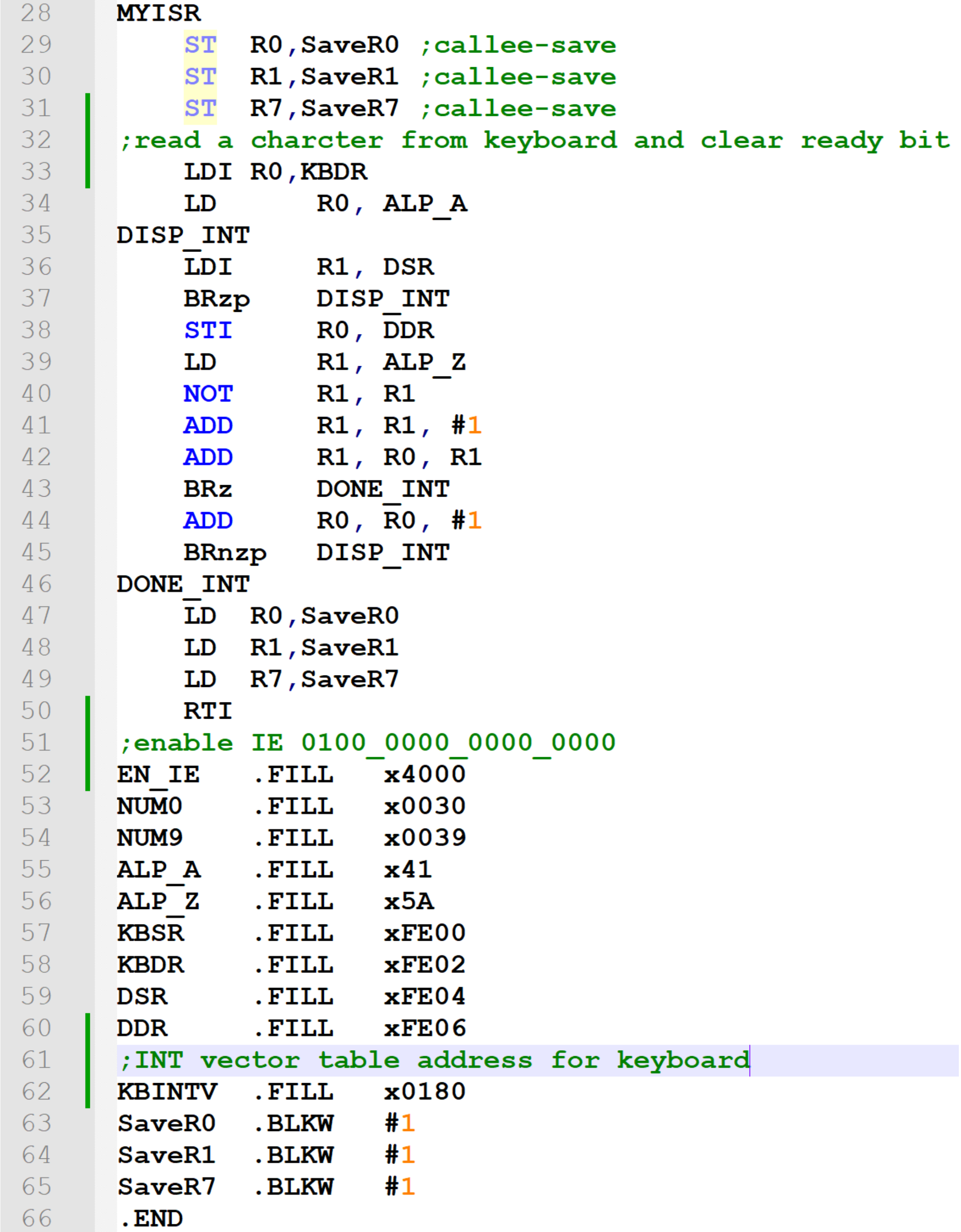
9. Return (device B)

Execute the Interrupt Code



```
.ORIG   x3000
;load ISR address to INTV (M[x0180] <- MYISR)
    LEA     R0, MYISR
    STI     R0, KBINTV
;enable IE bit of KBSR
    LD      R3, EN_IE
    STI     R3, KBSR

    LD      R0, NUM0
DISP
    LDI     R1, DSR
    BRzp    DISP
    STI     R0, DDR
    LD      R1, NUM9
    NOT     R1, R1
    ADD     R1, R1, #1
    ADD     R1, R0, R1
    BRz     RESET
    ADD     R0, R0, #1
    BRnzp   DISP
RESET
    LD      R0, NUM0
    BRnzp   DISP
```

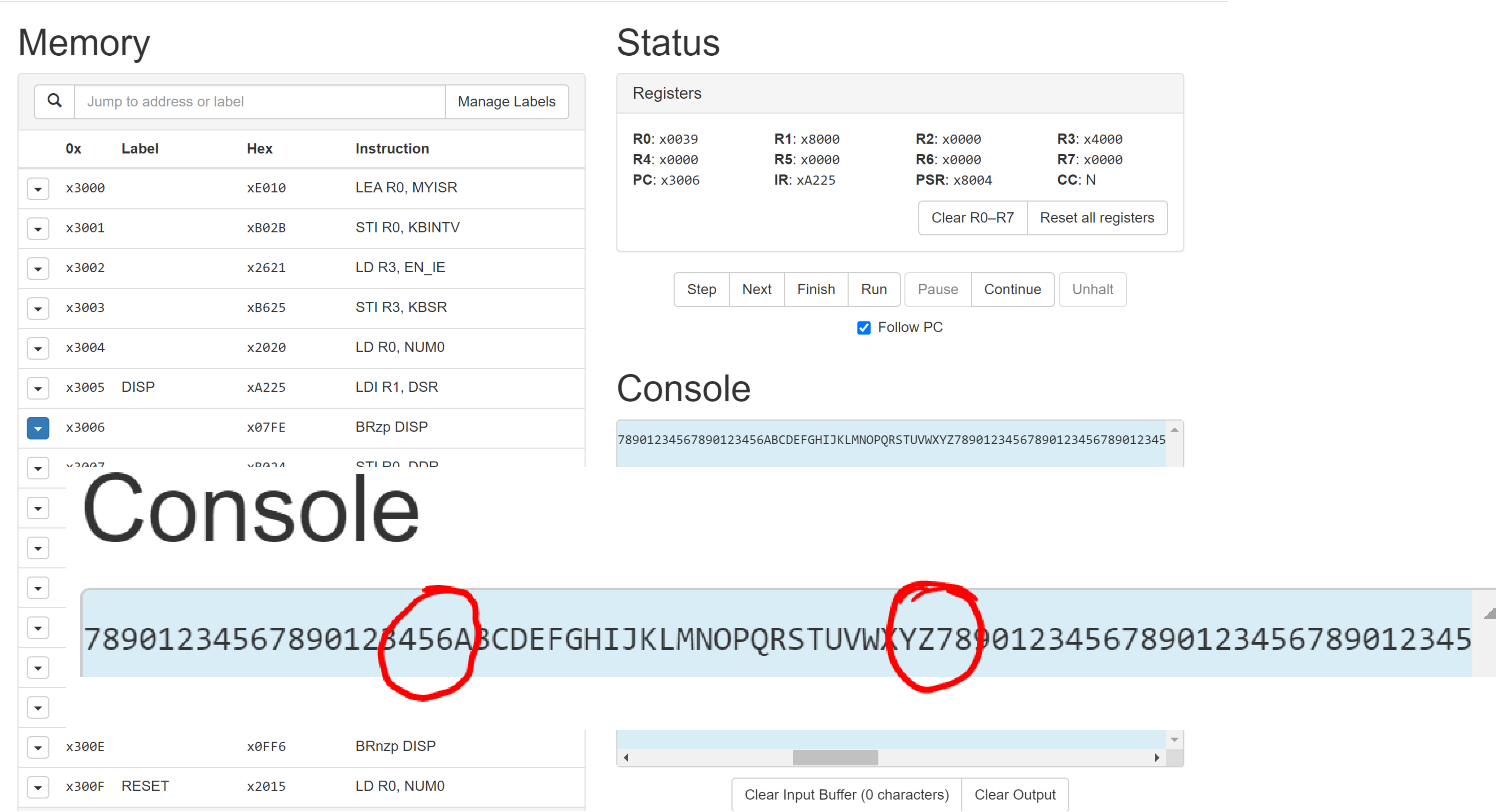


```
MYISR
    ST  R0,SaveR0 ;callee-save
    ST  R1,SaveR1 ;callee-save
    ST  R7,SaveR7 ;callee-save
;read a charcter from keyboard and clear ready bit
    LDI R0,KBDR
    LD      R0, ALP_A
DISP_INT
    LDI     R1, DSR
    BRzp    DISP_INT
    STI     R0, DDR
    LD      R1, ALP_Z
    NOT     R1, R1
    ADD     R1, R1, #1
    ADD     R1, R0, R1
    BRz     DONE_INT
    ADD     R0, R0, #1
    BRnzp   DISP_INT
DONE_INT
    LD  R0,SaveR0
    LD  R1,SaveR1
    LD  R7,SaveR7
    RTI
;enable IE 0100_0000_0000_0000
EN_IE   .FILL   x4000
NUM0    .FILL   x0030
NUM9    .FILL   x0039
ALP_A   .FILL   x41
ALP_Z   .FILL   x5A
KBSR    .FILL   xFE00
KBDR    .FILL   xFE02
DSR     .FILL   xFE04
DDR     .FILL   xFE06
;INT vector table address for keyboard
KBINTV  .FILL   x0180
SaveR0  .BLKW   #1
SaveR1  .BLKW   #1
SaveR7  .BLKW   #1
.END
```

****Assemble and Run the Code on the LC3 Web Simulator****

https://courses.grainger.illinois.edu/ece220/sp2020/lc3web/index.html

LC3 Web Simulator



Memory

	0x	Label	Hex	Instruction
▾	x3000		xE010	LEA R0, MYISR
▾	x3001		xB02B	STI R0, KBINTV
▾	x3002		x2621	LD R3, EN_IE
▾	x3003		xB625	STI R3, KBSR
▾	x3004		x2020	LD R0, NUM0
▾	x3005	DISP	xA225	LDI R1, DSR
▾	x3006		x07FE	BRzp DISP
▾	x3007		xB024	STI R0, DDR
▾				
▾				
▾				
▾				
▾				
▾				
▾	x300E		x0FF6	BRnzp DISP
▾	x300F	RESET	x2015	LD R0, NUM0

Status

Registers

R0: x0039 **R1**: x8000 **R2**: x0000 **R3**: x4000
R4: x0000 **R5**: x0000 **R6**: x0000 **R7**: x0000
PC: x3006 **IR**: xA225 **PSR**: x8004 **CC**: N

Console

```
78901234567890123456ABCDEFGHIJKLMNOPQRSTUVWXYZ78901234567890123456789012345
```

Console

78901234567890123456ABCDEFGHIJKLMNOPQRSTUVWXYZ78901234567890123456789012345

Q. Suppose a device A initiates an interrupt. The interrupt vector of device A is x30 and its ISR starts at x1200. What can you tell about the contents of any memory location?

A. The content of address x0030 is x1200.

B. The content of address x0130 is x1200.

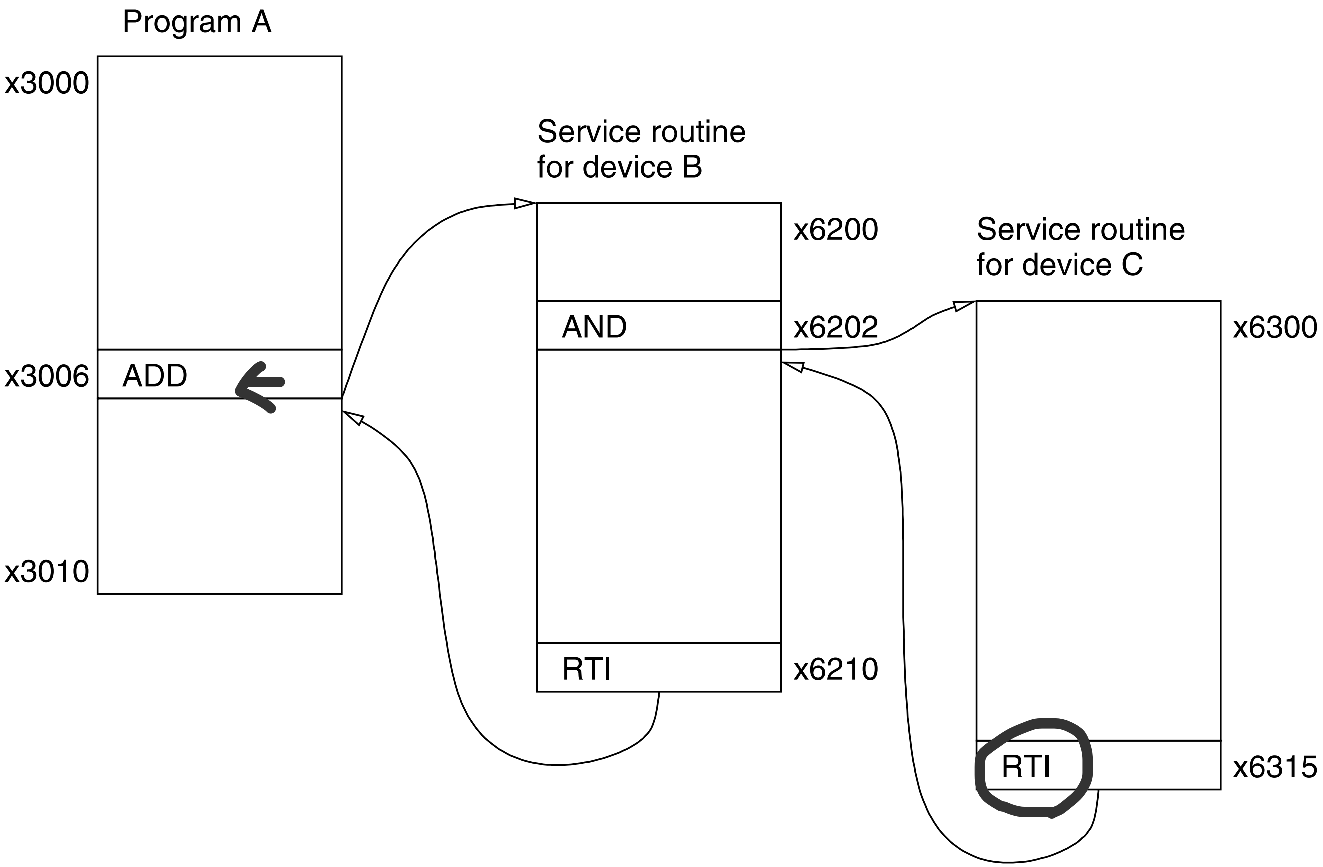
C. The content of address x1200 is x0030.

D. The content of address x1200 is x0130.

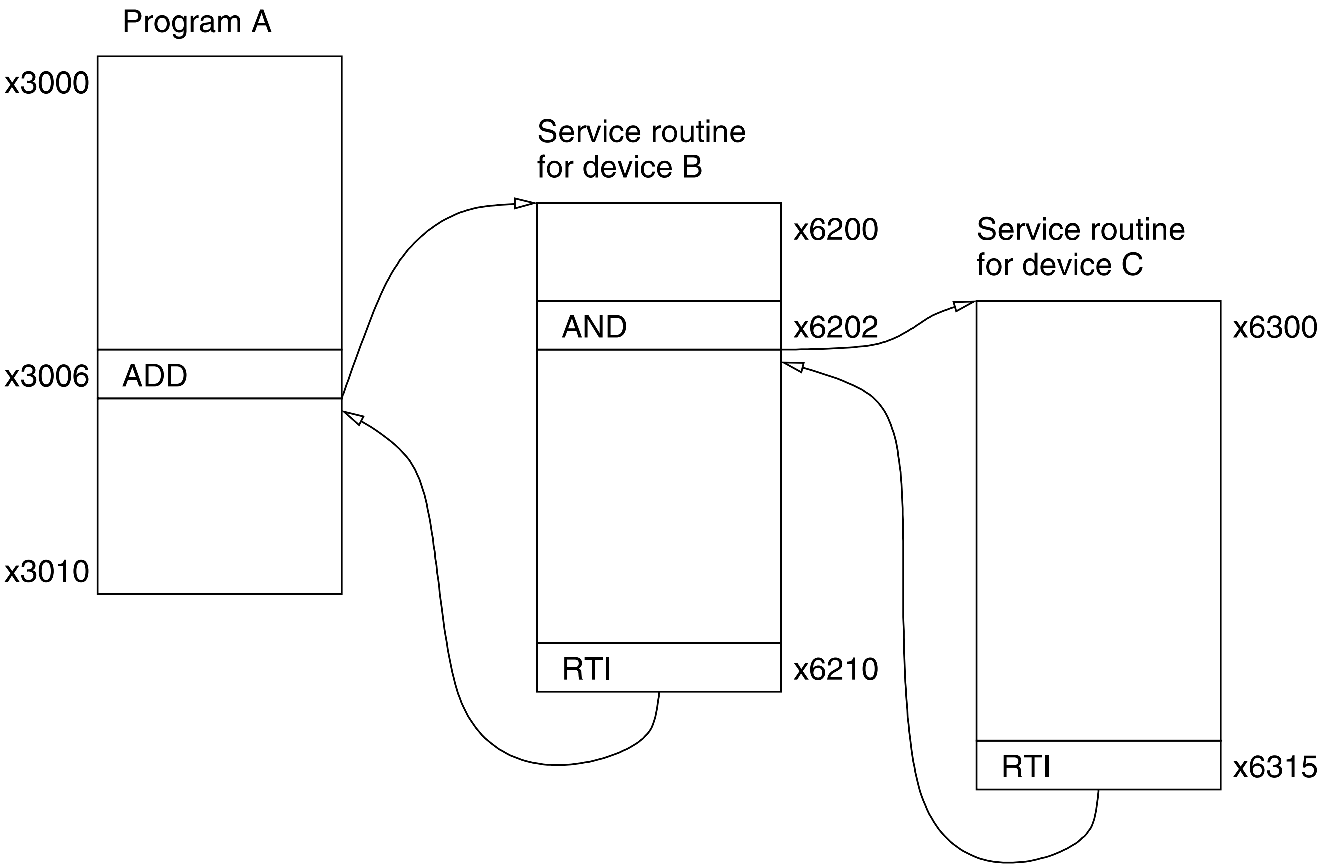
E. You cannot determine anything about the memory by the above information.

Q. After the RTI of device C is executed, what is the value of PC? Draw the Supervisor Stack and R6 (assume textbook-style stack).

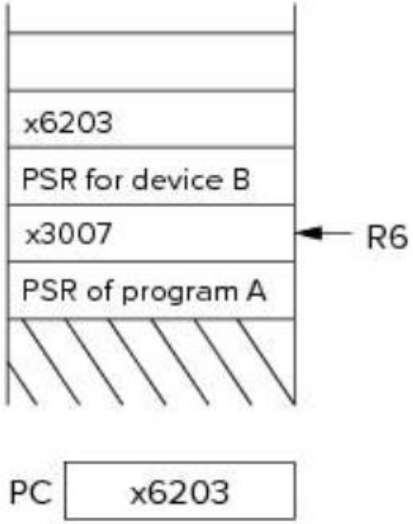
PC = x6203



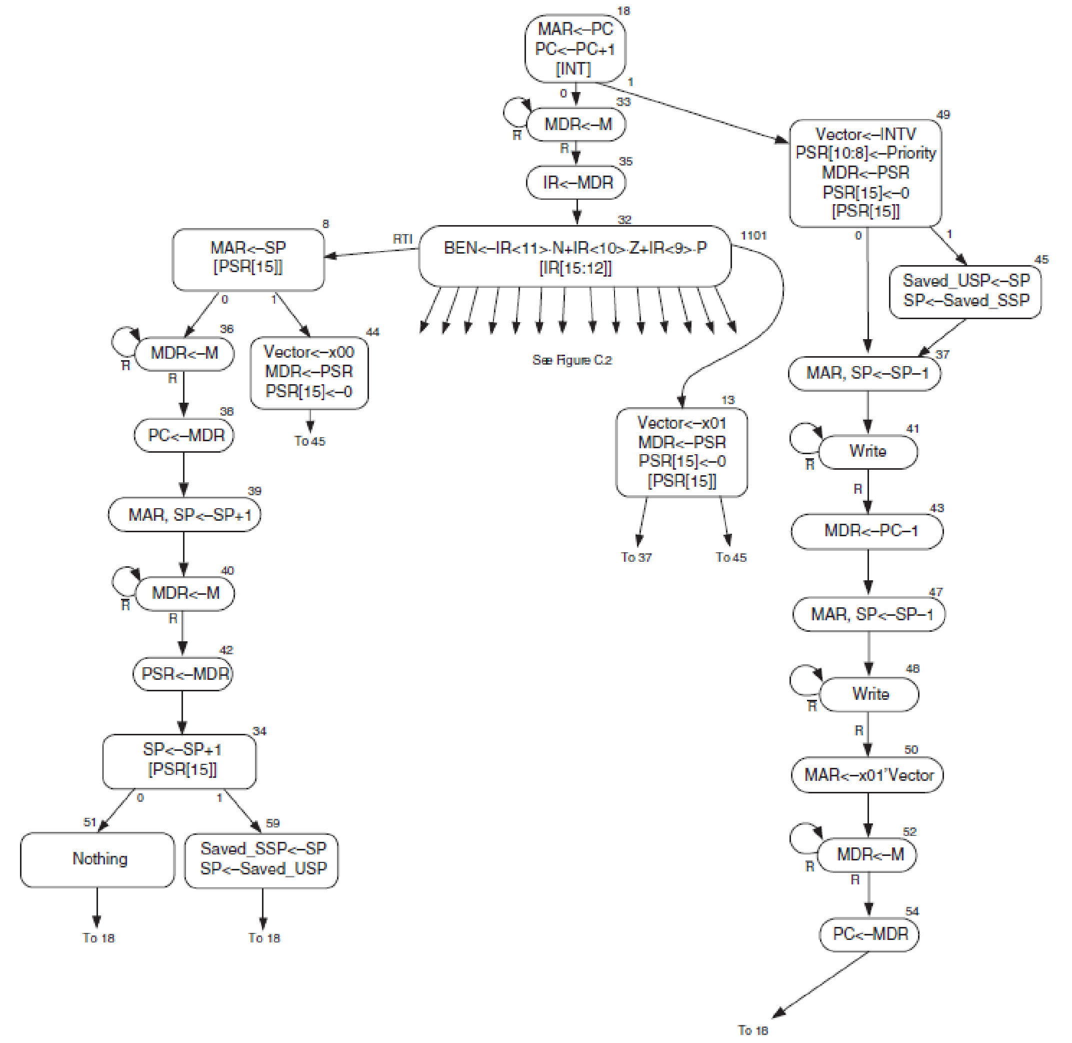
Q. After the RTI of device C is executed, what is the value of PC? Draw the Supervisor Stack and R6 (assume textbook-style stack).

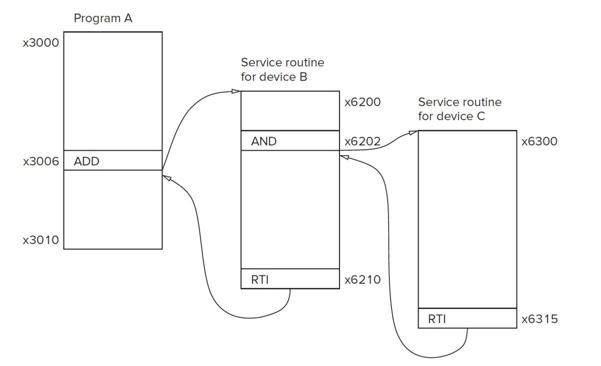


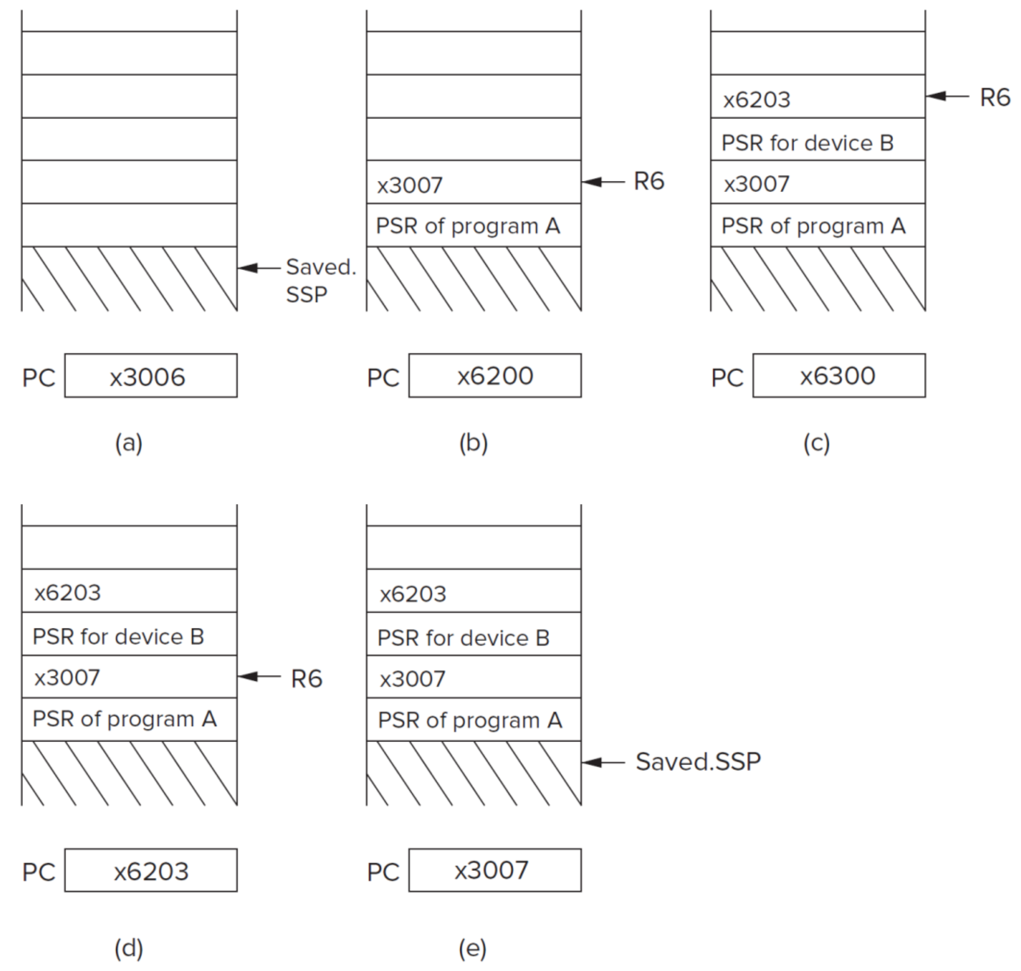
A.



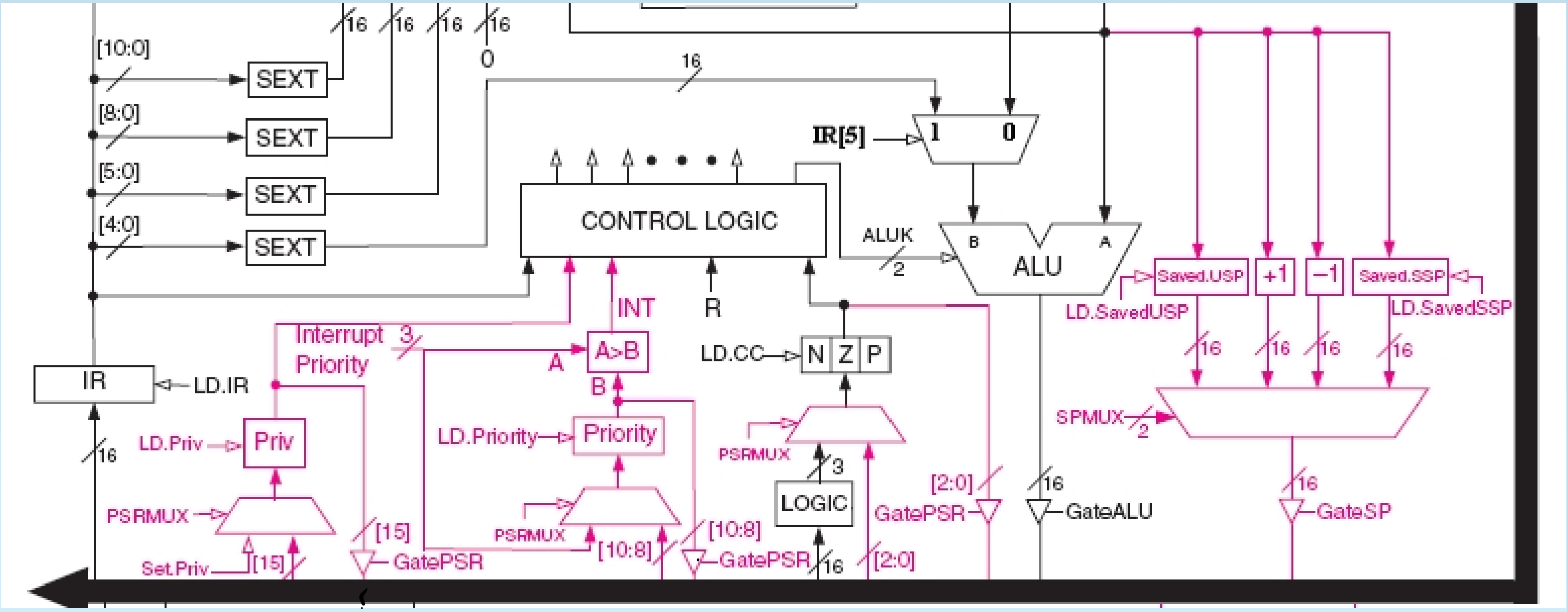
Content of Supervisory stack and PC
during interrupt-driven I/O



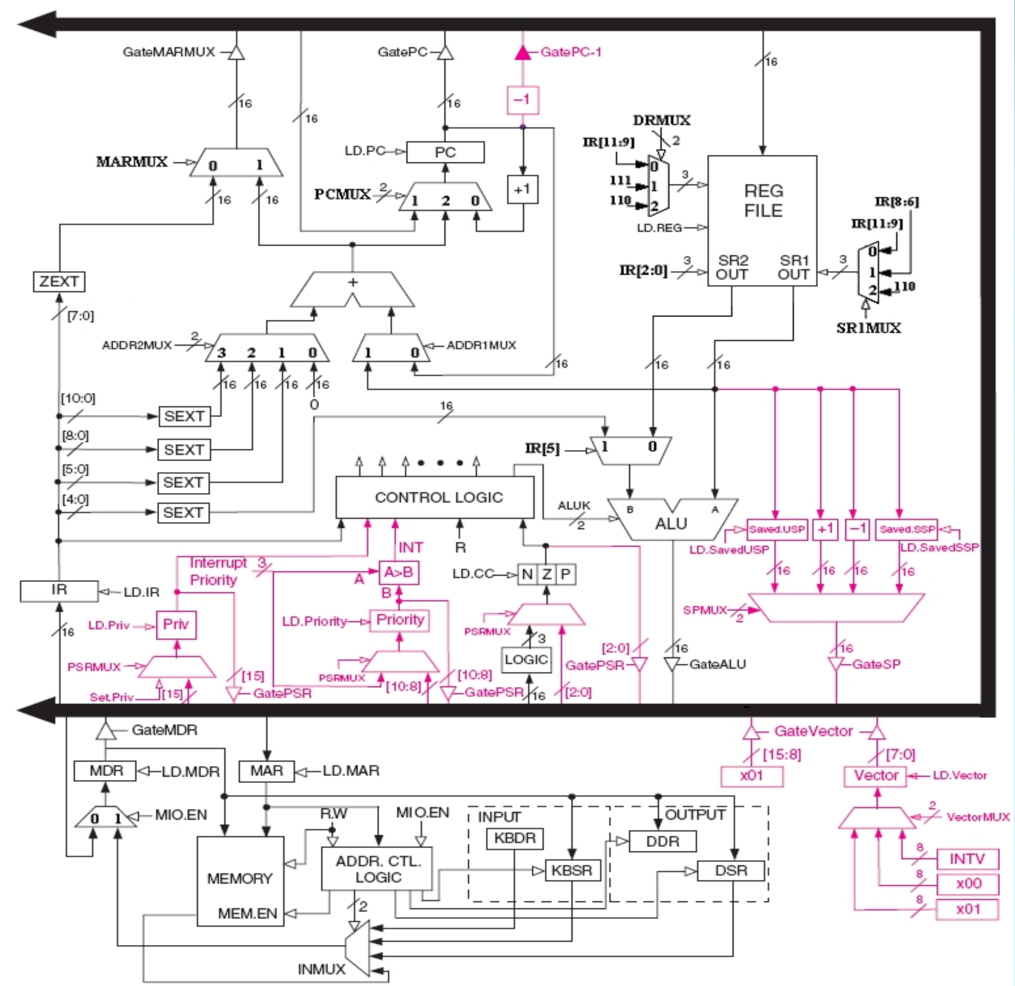


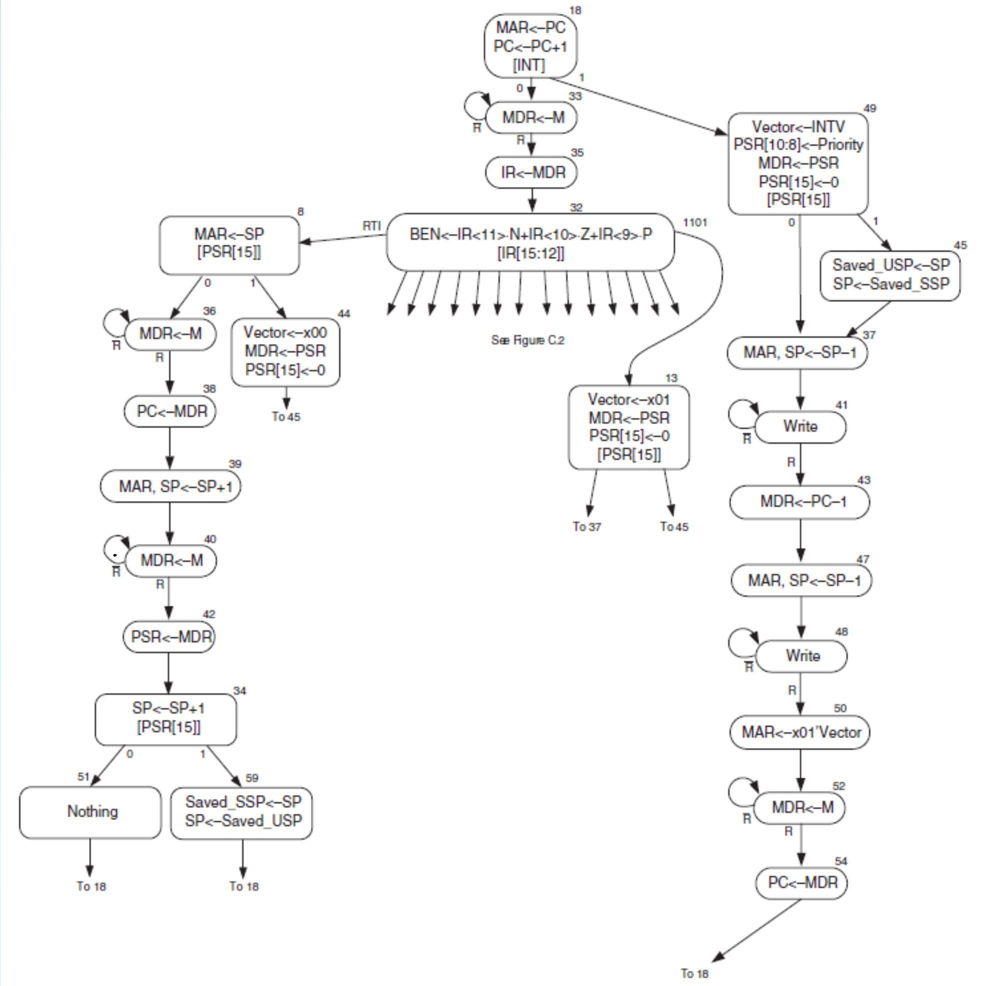


LC-3 Hardware to Support Interrupts



Extended LC-3 Datapath and FSM


GateMARMUX
GatePC
GatePC-1
−1
16
MARMUX
0 1
LD.PC
PC
PCMUX
2
1 2 0
+1
DRMUX
IR[11:9]
111
110
0
1
2
3
LD.REG
REG FILE
IR[8:6]
IR[11:9]
SR2 OUT
SR1 OUT
IR[2:0]
110
SR1MUX
ZEXT
[7:0]
+
ADDR2MUX
3 2 1 0
ADDR1MUX
1 0
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
[4:0]
SEXT
IR[5]
CONTROL LOGIC
ALUK
ALU
B
A
INT
R
Interrupt Priority
A>B
LD.CC
N Z P
IR
LD.IR
LD.Priv
Priv
LD.Priority
Priority
PSRMUX
LOGIC
[2:0]
GatePSR
[15]
[10:8]
Set.Priv
Saved.USP
+1
−1
Saved.SSP
LD.SavedUSP
LD.SavedSSP
SPMUX
GateALU
GateSP
GateMDR
MDR
LD.MDR
MAR
LD.MAR
MIO.EN
R.W
MIO.EN
INPUT
OUTPUT
KBDR
KBSR
DDR
DSR
MEMORY
MEM.EN
ADDR. CTL. LOGIC
INMUX
GateVector
[15:8]
[7:0]
x01
Vector
LD.Vector
VectorMUX
INTV
x00
x01
8


18
MAR<–PC
PC<–PC+1
[INT]
0
1
33
MDR<–M
R
35
IR<–MDR
32
BEN<–IR<11>·N+IR<10>·Z+IR<9>·P
[IR[15:12]]
RTI
1101
See Figure C.2
8
MAR<–SP
[PSR[15]]
36
MDR<–M
38
PC<–MDR
39
MAR, SP<–SP+1
40
MDR<–M
42
PSR<–MDR
34
SP<–SP+1
[PSR[15]]
51
Nothing
To 18
59
Saved_SSP<–SP
SP<–Saved_USP
To 18
44
Vector<–x00
MDR<–PSR
PSR[15]<–0
To 45
13
Vector<–x01
MDR<–PSR
PSR[15]<–0
[PSR[15]]
To 37
To 45
49
Vector<–INTV
PSR[10:8]<–Priority
MDR<–PSR
PSR[15]<–0
[PSR[15]]
45
Saved_USP<–SP
SP<–Saved_SSP
37
MAR, SP<–SP–1
41
Write
43
MDR<–PC–1
47
MAR, SP<–SP–1
48
Write
50
MAR<–x01'Vector
52
MDR<–M
54
PC<–MDR
To 18