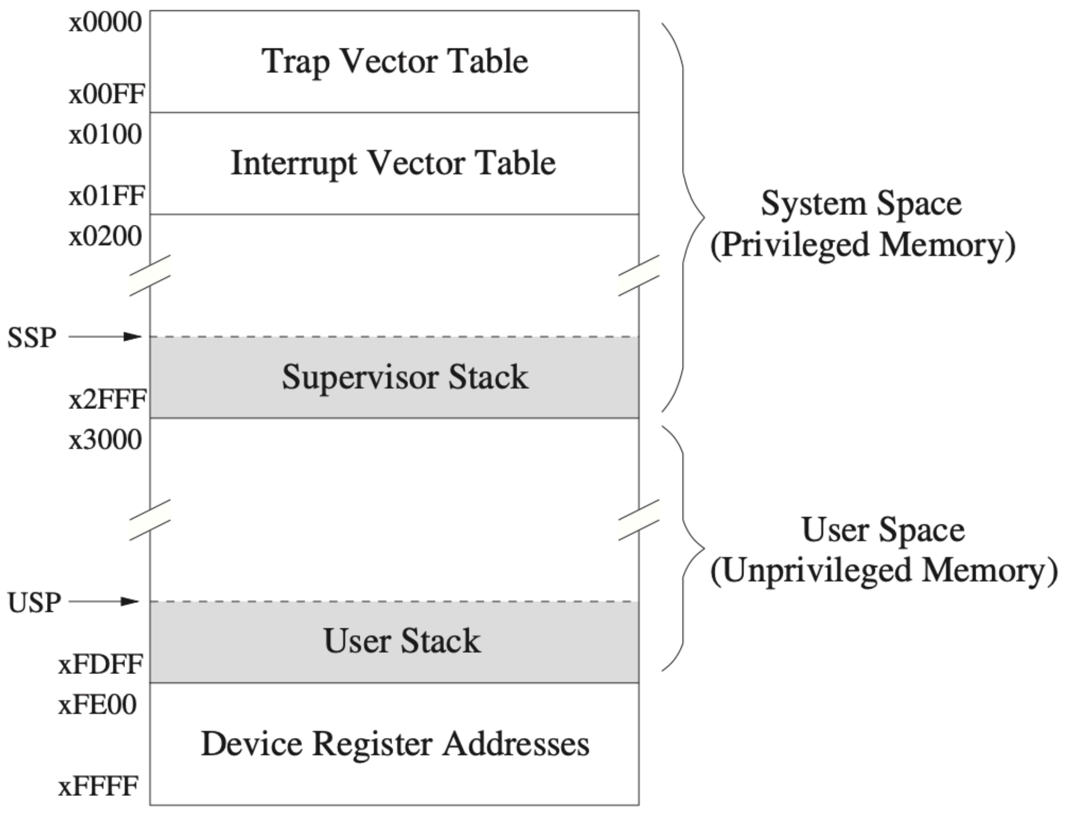
Patt-Ch9 I/O

9.7 Exceptions

9.1 Overview: Memory Organization



9.1.1 System Space

- Range: [x0000, x2FFF]
- Privileged
- Contents
  - Trap Vector Table, [x0000, x00FF]
  - Interrupt Vector Table, [x0100, x01FF]
  - System Service Routines
  - Exception Service Routines
  - Interrupt Service Routines
  - etc.

Trap Vector Table

A table of the starting address of System service routines

- Locations: [x0000, x00FF]

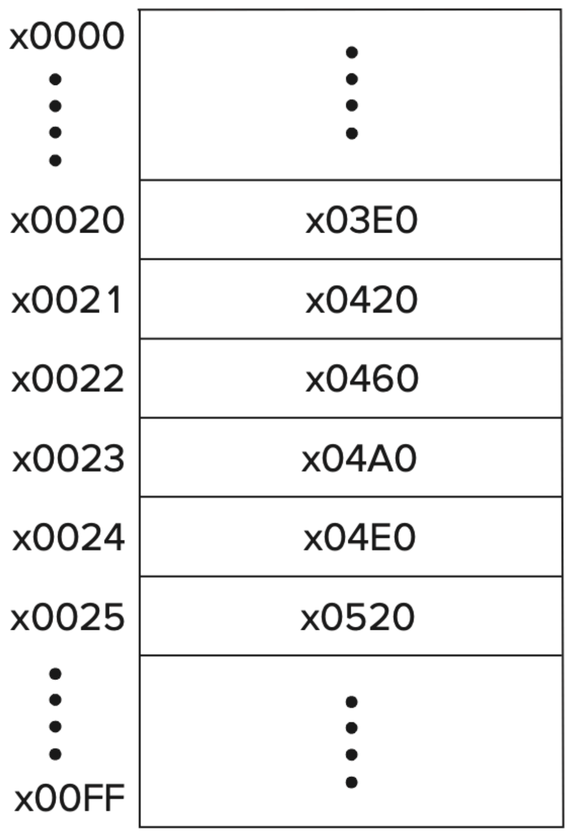


Figure 9.10 **The Trap Vector Table.**

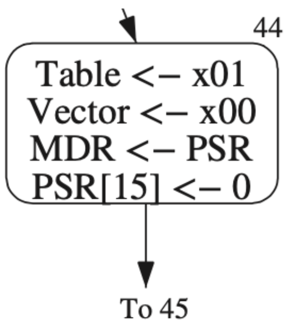
Interrupt Vector(INTV) Table

> See Page 675 A.3

exception(异常) service routines

- amount to 128 entries
- Locations x0100 to x017F
- Contents: the starting addresses of routines which are called because they handle exceptional events, that is, events that prevent the program from executing normally.

e.g. RTI state 44, use RTI instruction but PSR[15]=1



interrupt service routines

- amount to 128 entries

- Locations x0180 to x01FF
- Contents: provide the starting addresses of routines that service events that are **external to the program** that is running
    - requests from I/O devices
    - *timer interrupt, machine check, power failure, and so on*

9.1.2 User Space

- Range: [x3000, xFDFF]
- Unprevileged

Attention: xFDFF is the last memory location of LC-3

9.1.3 Device Register Address

- Range: [xFE00, xFFFF]
- Privileged
- Contents
    - Not correpsond to memory locations at all
    - identify registers taking part in I/O functions
    - some special registers associated with the processor

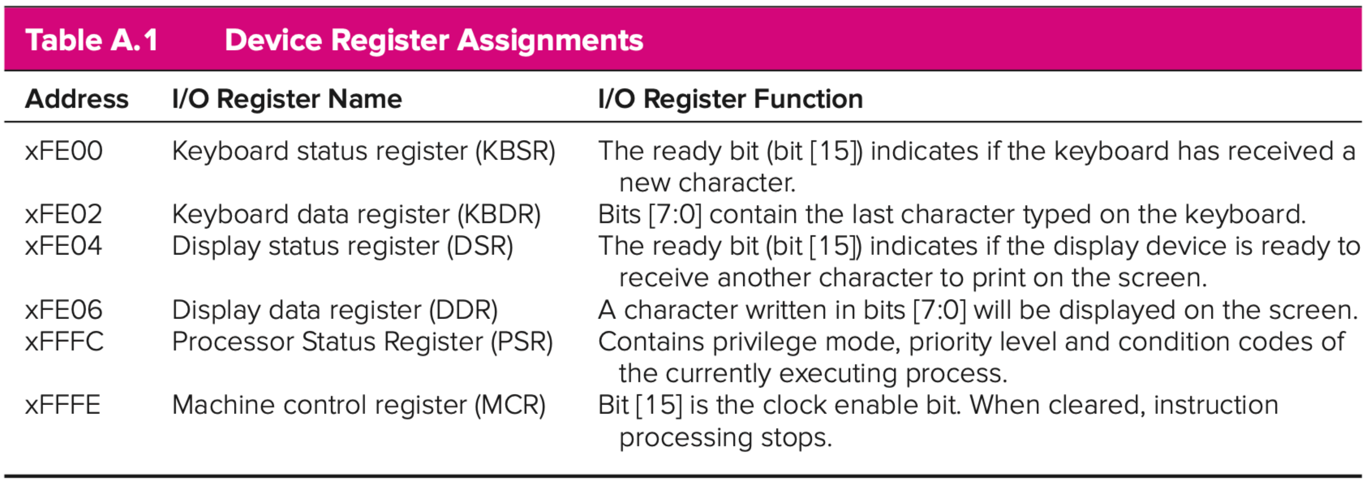
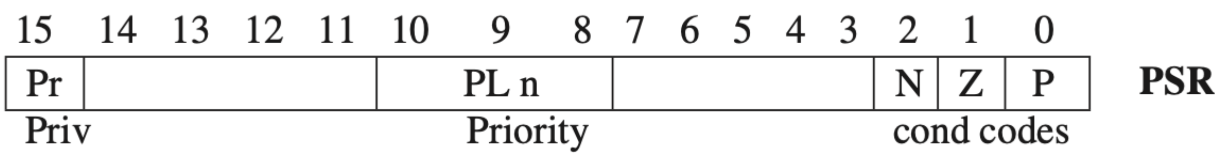


Table A.1 Device Register Assignments

Address	I/O Register Name	I/O Register Function
xFE00	Keyboard status register (KBSR)	The ready bit (bit [15]) indicates if the keyboard has received a new character.
xFE02	Keyboard data register (KBDR)	Bits [7:0] contain the last character typed on the keyboard.
xFE04	Display status register (DSR)	The ready bit (bit [15]) indicates if the display device is ready to receive another character to print on the screen.
xFE06	Display data register (DDR)	A character written in bits [7:0] will be displayed on the screen.
xFFFC	Processor Status Register (PSR)	Contains privilege mode, priority level and condition codes of the currently executing process.
xFFFE	Machine control register (MCR)	Bit [15] is the clock enable bit. When cleared, instruction processing stops.

PSR



- **Privilege**
    - PSR[15]=0 means ***supervisor privilege*** (kernal mode)
    - PSR[15]=1 means ***unprivileged*** (user mode)

> **Right(特权)** to do something
>
> such as execute a particular instruction or access a particular memory location.

- **Priority**
  - Bits [10:8] specify the priority level (PL) of the program.
  - The highest priority level is 7 (PL7), the lowest is 0(PL0)

> **Urgency(紧急度)** of a program to execute

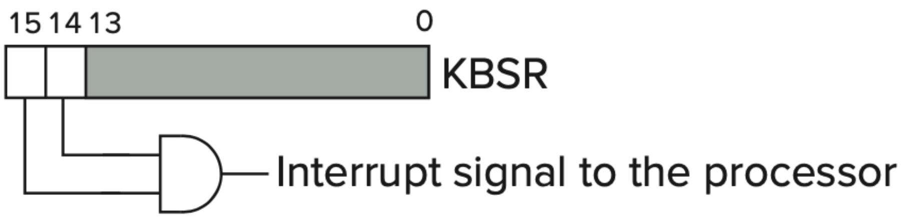
*Attention: Privilege and Priority have **nothing to do** with each other*

- ***Why CC is set in PSR?***

  Due to the machanism of Interrupt-Driven I/O, when we come back from the interrupt routine, we want CC not changed.

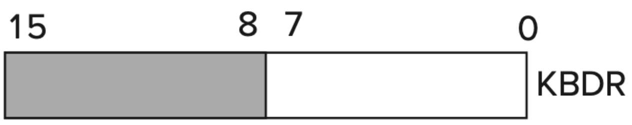
  Popping PSR from Supervisor Stack can handle this issue.

KBSR



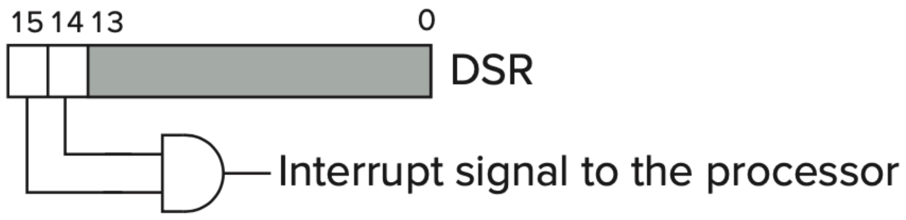
- bit [15] contains the ready bit
  - When keyboard puts a char, R = 1
  - When processor gets the char, R = 0
- bit [14] contains the interrupt enable bit

KBDR



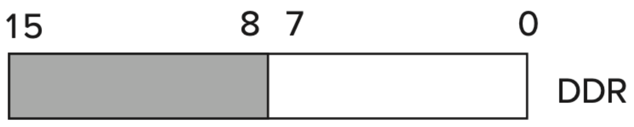
- bits [7:0] are used for the data, and bits [15:8] contain x00

DSR



- bit [15] contains the ready bit
    - When monitor has written a char (idle), R = 1
    - When monitor is writing(busy), R = 0
- Bit [14] contains the interrupt enable bit

DDR



- bits [7:0] are used for data, and bits [15:8] contain x00

MCR

9.1.4 Supervisor Stack/User Stack

- Supervisor mode or User mode at any one time, so only one of the two stacks is active at any one time.
- R6 is generally used as the stack pointer (SP) for the active stack.
- **Saved_SSP** and **Saved_USP** are provided to save the SP not in use.
    - for example, from Supervisor mode to User mode, the SP is stored in Saved_SSP, and the SP is loaded from Saved_USP.

9.2 Input/Output

9.2.1 Three Basic Characteristics of I/O

(1) Memory-Mapped I/O vs. Special I/O Instructions

> 内存映射 vs. 特殊指令
>
> *Noting: Most computers nowadays use Memory-Mapped I/O rather than special instructions*

Memory-Mapped I/O

Input and output are handled by load/store (LD/ST, LDI/STI, LDR/STR) instructions using memory addresses from xFE00 to xFFFF to designate each device register.

Above are the input and output device registers and internal processor registers that have been specified for the LC-3 thus far, along with their corresponding assigned addresses from the memory address space.

refer to 9.1.3 Device Register Address

(2) Asynchronous vs. Synchronous

> 同步 vs. 异步
>
> Most interaction between a processor and I/O is asynchronous

asynchronous: I/O devices usually operate at speeds very different from that of a microprocessor, and not in lockstep.

synchronization mechanism: ready bit (KBSR[15] and DSR[15])

- In the case of the keyboard, we will need a one-bit status register to indicate if someone has or has not typed a character.
  - Each time the typist types a character, the ready bit is set to 1.
  - Each time the computer reads a character, it clears the ready bit.
  - When the computer detects that the ready bit is set, it could only have been caused by a **new** character being typed, so the computer would know to again read a character.
  - If not detected Ready Bit, we will input the same char many times重复读
- In the case of the monitor, we will need a one-bit status register to indicate whether or not the most recent character sent to the monitor has been displayed, and so the monitor can be given another character to display.
  - If not detected Ready Bit, we will ignore or lost some chars stored in DDR丢输出

keyboard & monitor 有差异，原因在于processor和I/O设备读写速度的差异

(3) Interrupt-Driven vs. Polling

> 中断 vs. 轮询
>
> the issue of who **controls** the interaction between processor and I/O devices
>
> Both are used in today's computers

interrupt-driven I/O: the **keyboard** controls the interaction.

polling(查询): the **processor** is in charge. The ready bit is polled by the processor, asking if any key has been struck.

Time Efficiency of Interrupt-Driven I/O and Polling

With interrupt-driven I/O, none of that testing and branching has to go on.

Interrupt-Driven I/O improve time efficiency

Polling Input

```
START   LDI   R1, KBSR      ; must use LDI, not LD
        BRzp  START
        LDI   R0, KBDR
        BRnzp NEXT_TASK
KBSR    .FILL xFE00
KBDR    .FILL xFE02
```

ready bit = 1 means it's OK to load a character to the register

> Difference between LD and LDI
>
> LD R1, A A contains data
>
> LDI R1, A A contains an address. Go to the address to get data indirectly
>
> x3000 -> xFE00 must use LDI, because out of offset9

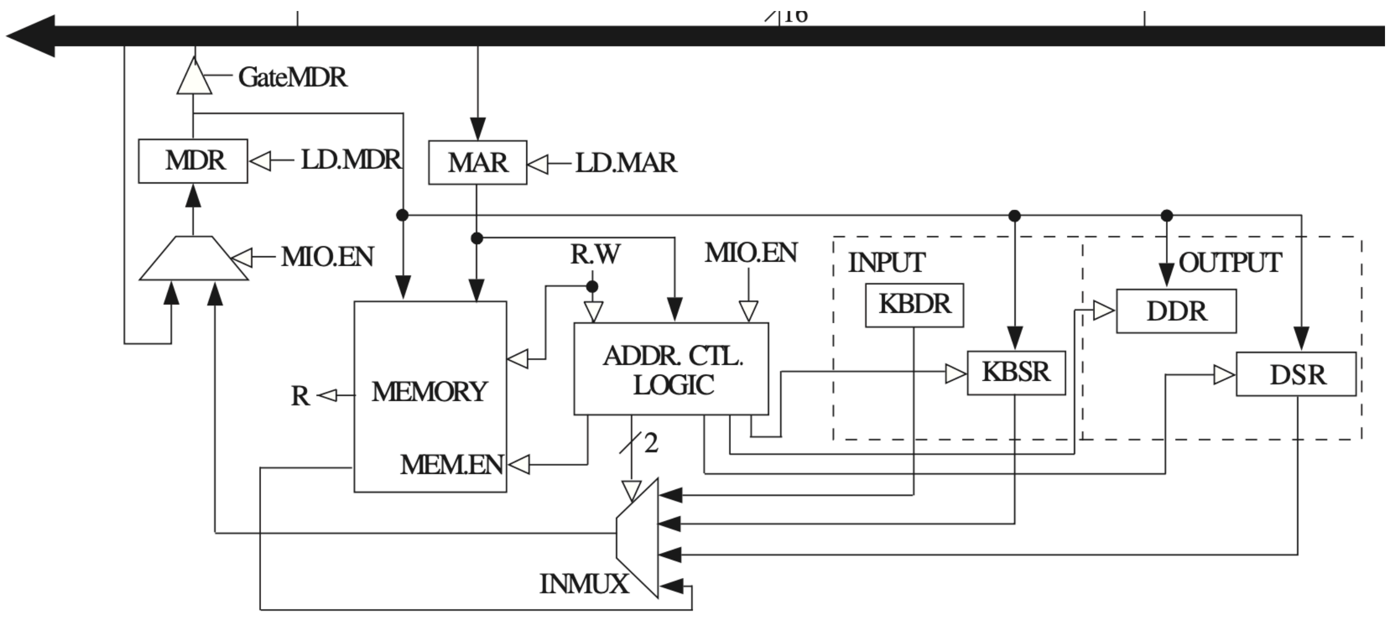
Polling Output

```
START   LDI   R1, DSR       ; must use LDI, not LD
        BRzp  START
        STI   R0, KBDR
        BRnzp NEXT_TASK
DSR     .FILL xFE04
DDR     .FILL xFE06
```

ready bit = 1 means it's OK to write a character to the screen

Drawback for polling: waste time

9.2.2 Memory-Mapped I/O Details



(1) Data Path - Elements and Functions

- **Address Control Logic** controls the input or output operation

3 Inputs

- **MIO.EN** (Memory IO Enable) indicates whether a data movement from/to memory or I/O is to take place this clock cycle
    - MIO.EN = 0, ACL do nothing
    - MIO.EN = 1, ACL provides control signal depending on the other two inputs
- **R.W** indicates whether a load or a store is to take place
    - R/W depends on instruction opcode
    - Read(load, or input), transfer from memory or I/O device to MDR
        - KBDR, KBSR, DSR, MEM -> INMUX (selected by MAR)
    - Write(output), transfer from MDR to memory or I/O device
        - MDR -> KBSR, DDR, DSR, MEM
- **MAR** contains the address of the memory location or *the memory-mapped address of an I/O device register*

5 Outputs

- **MEM.EN**(Memory Enable) indicates whether memory can be accessed
- **IN.MUX** select MEM, KBDR, KBSR, DSR depending on MAR
- **LD.KBSR**
- **LD.DSR**
- **LD.DDR**

(2) Truth Table for ACL

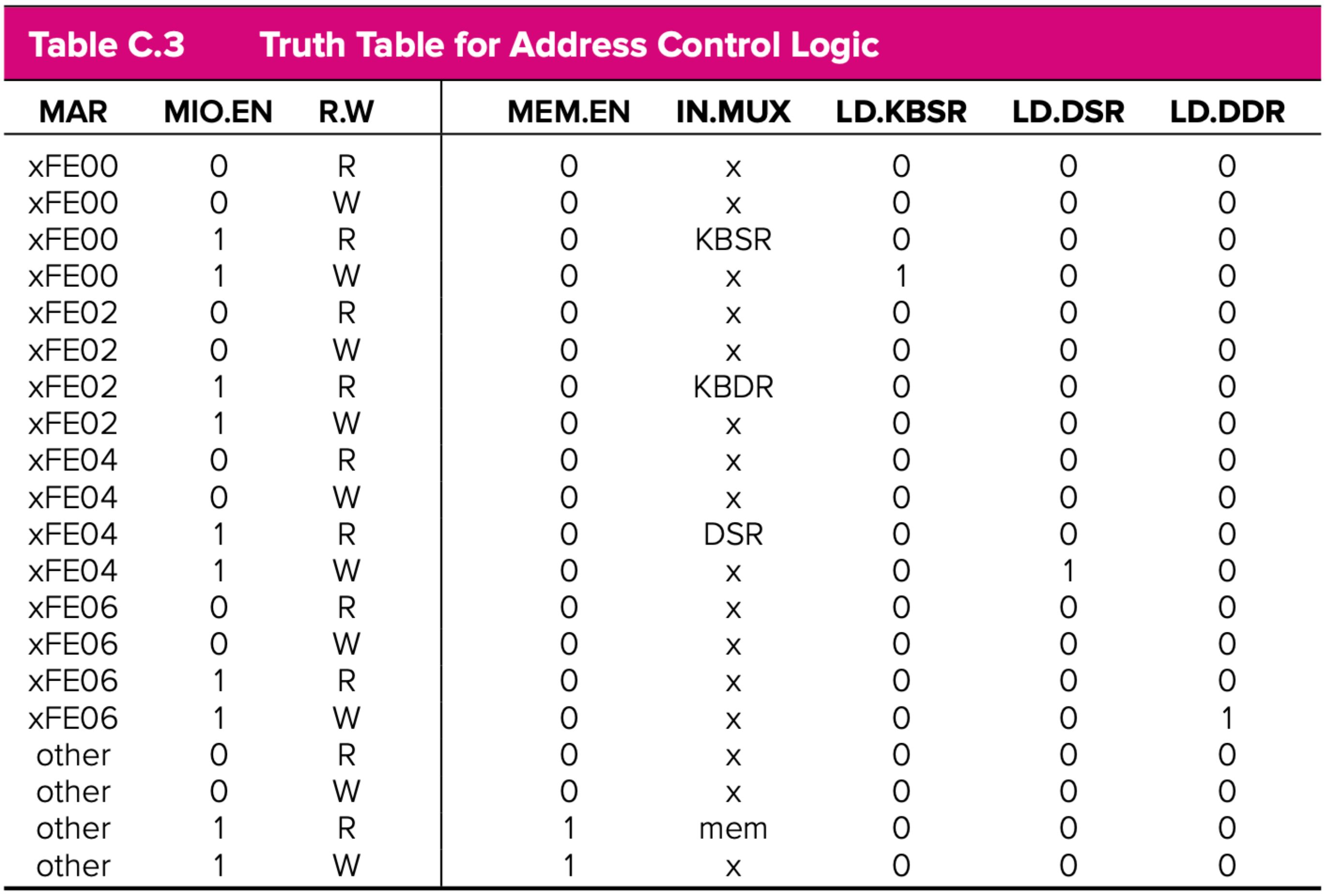


Table C.3 Truth Table for Address Control Logic

MAR	MIO.EN	R.W	MEM.EN	IN.MUX	LD.KBSR	LD.DSR	LD.DDR
xFE00	0	R	0	x	0	0	0
xFE00	0	W	0	x	0	0	0
xFE00	1	R	0	KBSR	0	0	0
xFE00	1	W	0	x	1	0	0
xFE02	0	R	0	x	0	0	0
xFE02	0	W	0	x	0	0	0
xFE02	1	R	0	KBDR	0	0	0
xFE02	1	W	0	x	0	0	0
xFE04	0	R	0	x	0	0	0
xFE04	0	W	0	x	0	0	0
xFE04	1	R	0	DSR	0	0	0
xFE04	1	W	0	x	0	1	0
xFE06	0	R	0	x	0	0	0
xFE06	0	W	0	x	0	0	0
xFE06	1	R	0	x	0	0	0
xFE06	1	W	0	x	0	0	1
other	0	R	0	x	0	0	0
other	0	W	0	x	0	0	0
other	1	R	1	mem	0	0	0
other	1	W	1	x	0	0	0

> Noting:
>
> x represents either of the inputs is irrelevant to the output
>
> Every address in MAR has 4 statuses of input, depending on MIO.EN and R.W
>
> other represents memory locations

9.3 Interrupt Routines

> Interrupt, but **as if nothing had happened.**

9.3.1 Three Situations for Interruption

(1) Interrupt-Driven I/O

Want, Right, Higher Priority

- Want: Bit[15], ready bit
- Right: Bit[14], interrupt enable bit
- Higher Priority: Bit[10:8]

All of them depend on PSR

INT signal Generation

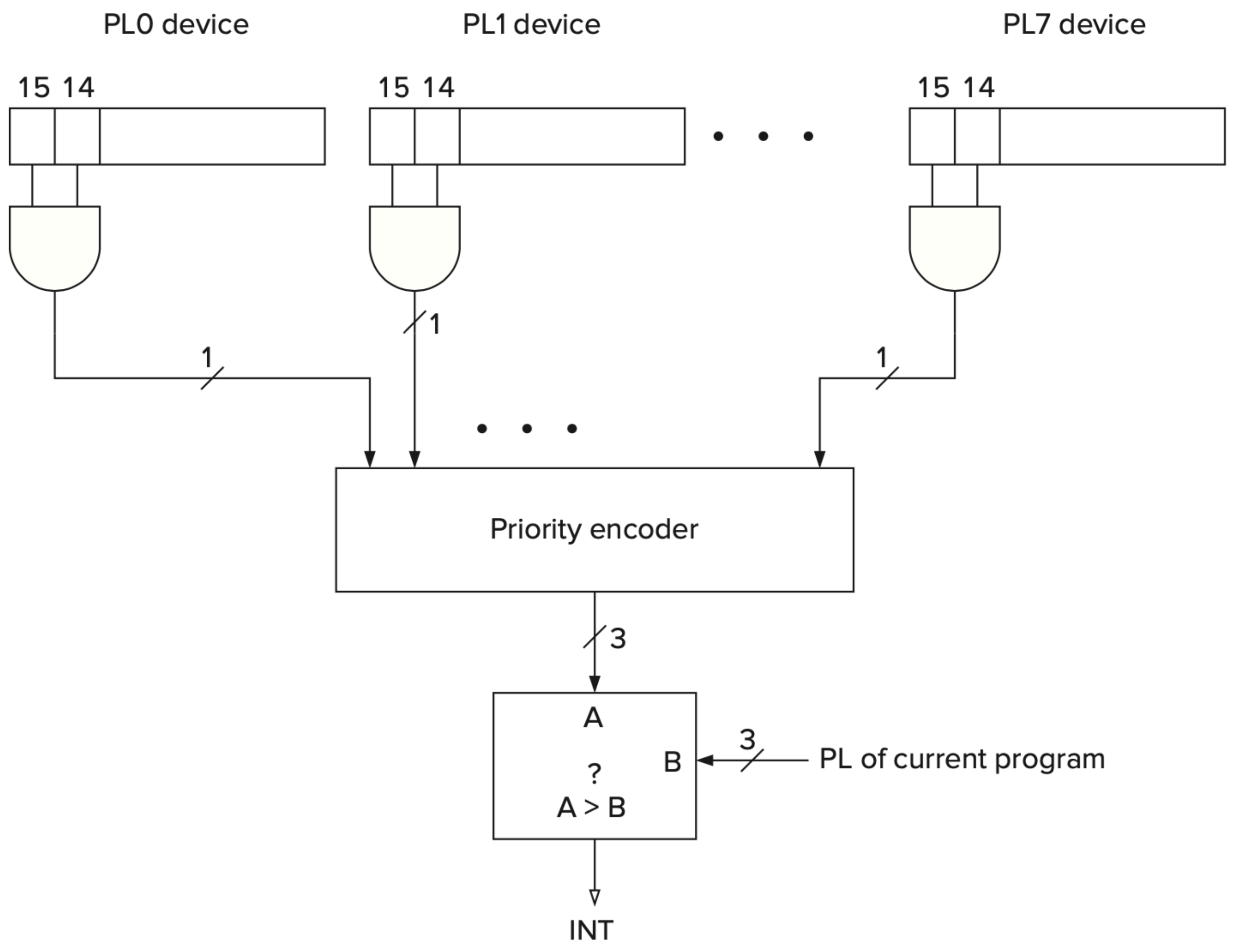


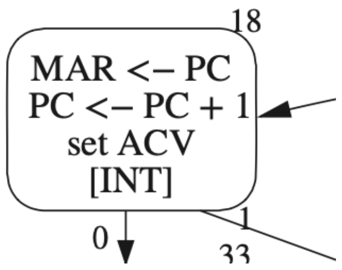
Figure 9.19 **Generation of the INT signal.**

priority encoder is a combinational logic structure that selects the highest priority request from all those asserted.

If the PL of that request is higher than the PL of the currently executing program, the INT signal is asserted.

When to test for INT

Always test at the start of FETCH phase, otherwise it's too complicated



(2) TRAP - System Call

Difference:

- Table, `x00` for Trap, `x01` for INTV
- Interrupt need `PSR[10:8] <- Priority`, Trap doesn't

(3) Exception

Example: use RTI in user mode

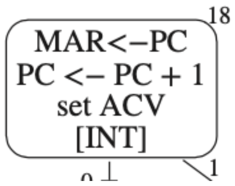
9.3.2 The Interrupt Process

(1) 4 Procedures

Not the precise sequence in state machine

- Save the state of the interrupted program
    - PUSH PSR to SS
    - PUSH PC to SS
- Load the state of the higher priority interrupting program
    - update PSR priority
    - PSR[15] <- 0
    - PC <- mem[Table'Vector]
    - (switch between US and SS)
- Service
- Restore and return -- RTI
    - Check PSR[15] - normal or exception
    - POP PC from SS
    - POP PSR from SS
    - (switch between US and SS)

(2) Finite State Machine

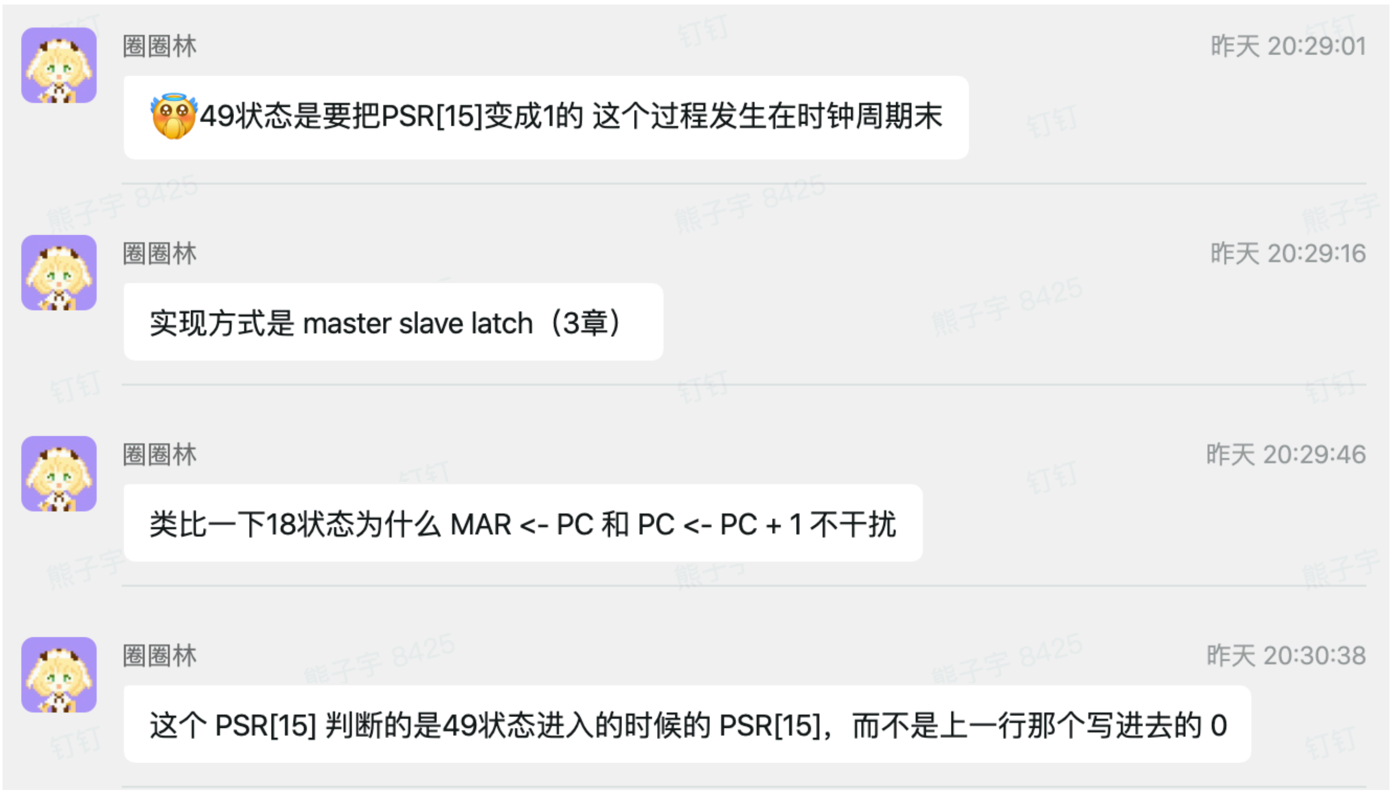


- ⚠️**Review MS flipflop to figure out when the value updates - not change in the whole state, until the next!**

`A <- B`

the master latch changes during CLOCK bar, and the slave latch changes at the next start of CLOCK

That is to say, the value containing (e.g. PSR[15], or PC, MAR) doesn't change during the whole CLOCK cycle until the next state



圈圈林 昨天 20:29:01

49状态是要把PSR[15]变成1的 这个过程发生在时钟周期末

圈圈林 昨天 20:29:16

实现方式是 master slave latch（3章）

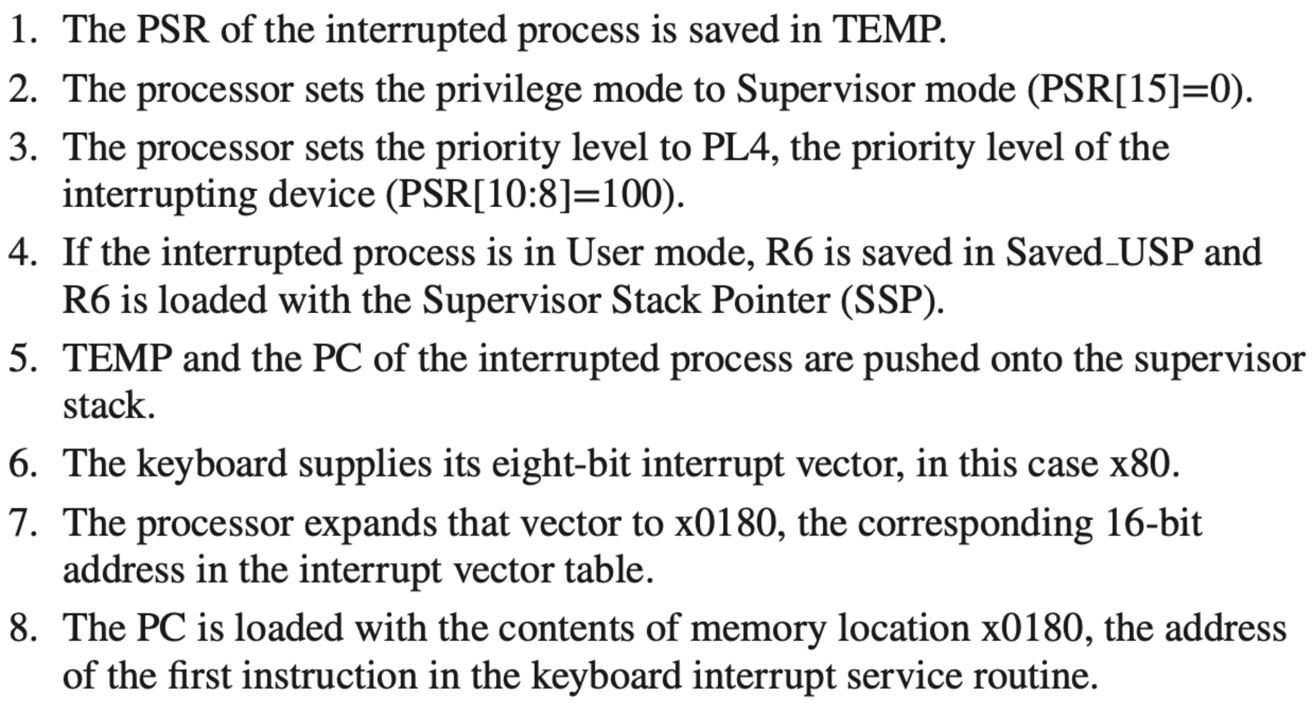
圈圈林 昨天 20:29:46

类比一下18状态为什么 MAR <- PC 和 PC <- PC + 1 不干扰

圈圈林 昨天 20:30:38

这个 PSR[15] 判断的是49状态进入的时候的 PSR[15]，而不是上一行那个写进去的 0

(3) Detailed Description



1. The PSR of the interrupted process is saved in TEMP.
2. The processor sets the privilege mode to Supervisor mode (PSR[15]=0).
3. The processor sets the priority level to PL4, the priority level of the interrupting device (PSR[10:8]=100).
4. If the interrupted process is in User mode, R6 is saved in Saved_USP and R6 is loaded with the Supervisor Stack Pointer (SSP).
5. TEMP and the PC of the interrupted process are pushed onto the supervisor stack.
6. The keyboard supplies its eight-bit interrupt vector, in this case x80.
7. The processor expands that vector to x0180, the corresponding 16-bit address in the interrupt vector table.
8. The PC is loaded with the contents of memory location x0180, the address of the first instruction in the keyboard interrupt service routine.

(4) Return from the Interrupt - RTI

RTI Instructions

RTI

Return from Trap or Interrupt

Assembler Format

RTI

Encoding

15–12	11–0
1000	000000000000

Operation

```
if (PSR[15] == 1)
   Initiate a privilege mode exception;
else
   PC=mem[R6]; R6 is the SSP, PC is restored
   R6=R6+1;
   TEMP=mem[R6];
   R6=R6+1; system stack completes POP before saved PSR is restored
   PSR=TEMP; PSR is restored
   if (PSR[15] == 1)
      Saved_SSP=R6 and R6=Saved_USP;
```

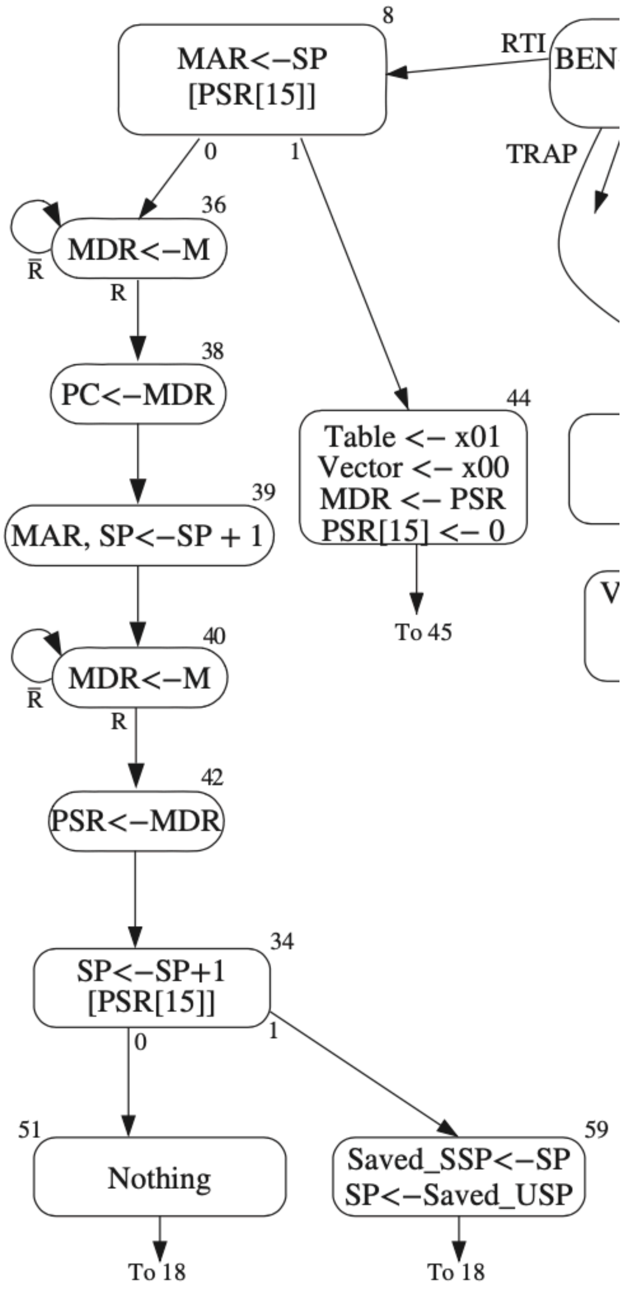
Description

If the processor is running in User mode, a privilege mode exception occurs. If in Supervisor mode, the top two elements on the system stack are popped and loaded into PC, PSR. After PSR is restored, if the processor is running in User mode, the SSP is saved in Saved_SSP, and R6 is loaded with Saved_USP.

Example

RTI ; PC, PSR ← top two values popped off stack.

RTI - Finite State Machine



POP PC: state 8, 36, 38

POP PSR: state 39, 40, 42

If go back to user mode: 59

Exception(ACV, that is access privileged state in user mode): state 44

9.3.3 Nested Interruption

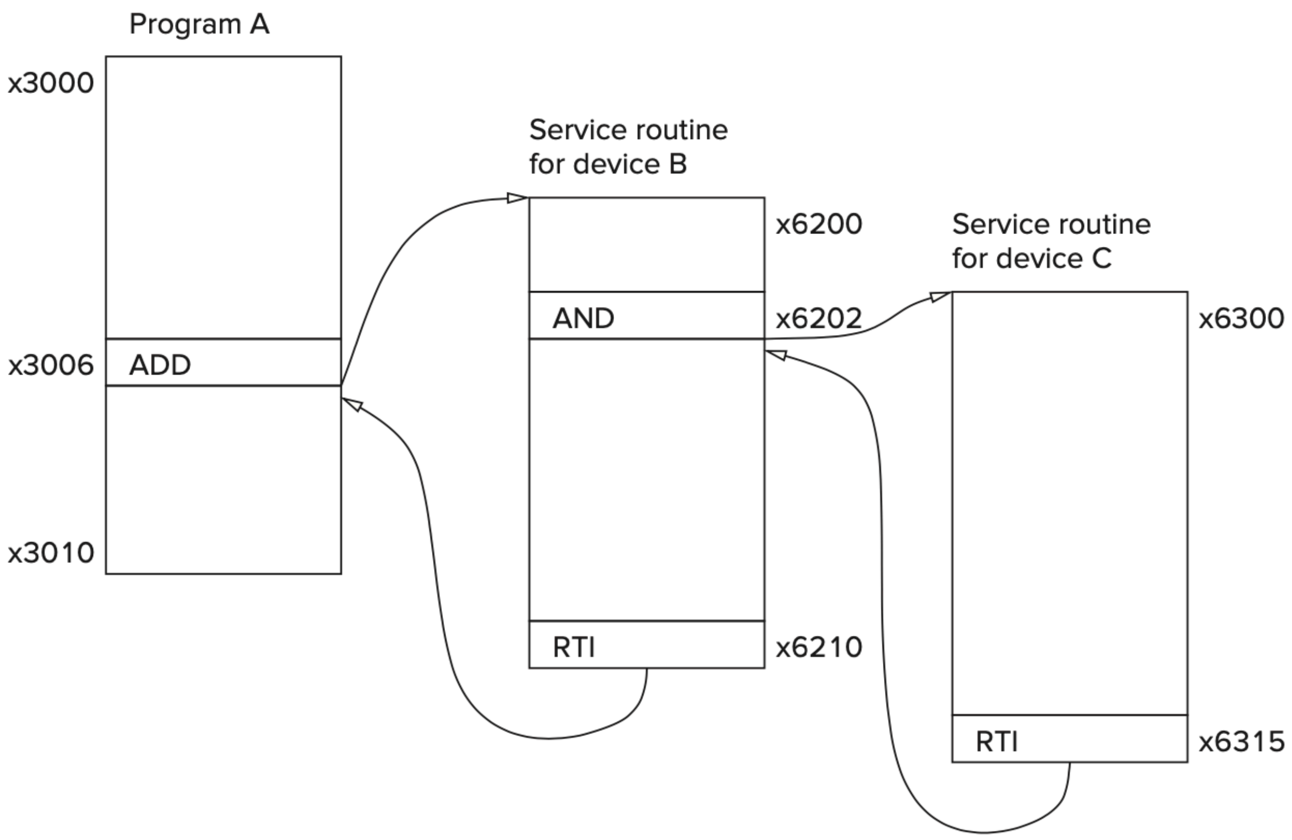


Figure 9.20 **Execution flow for interrupt-driven I/O.**

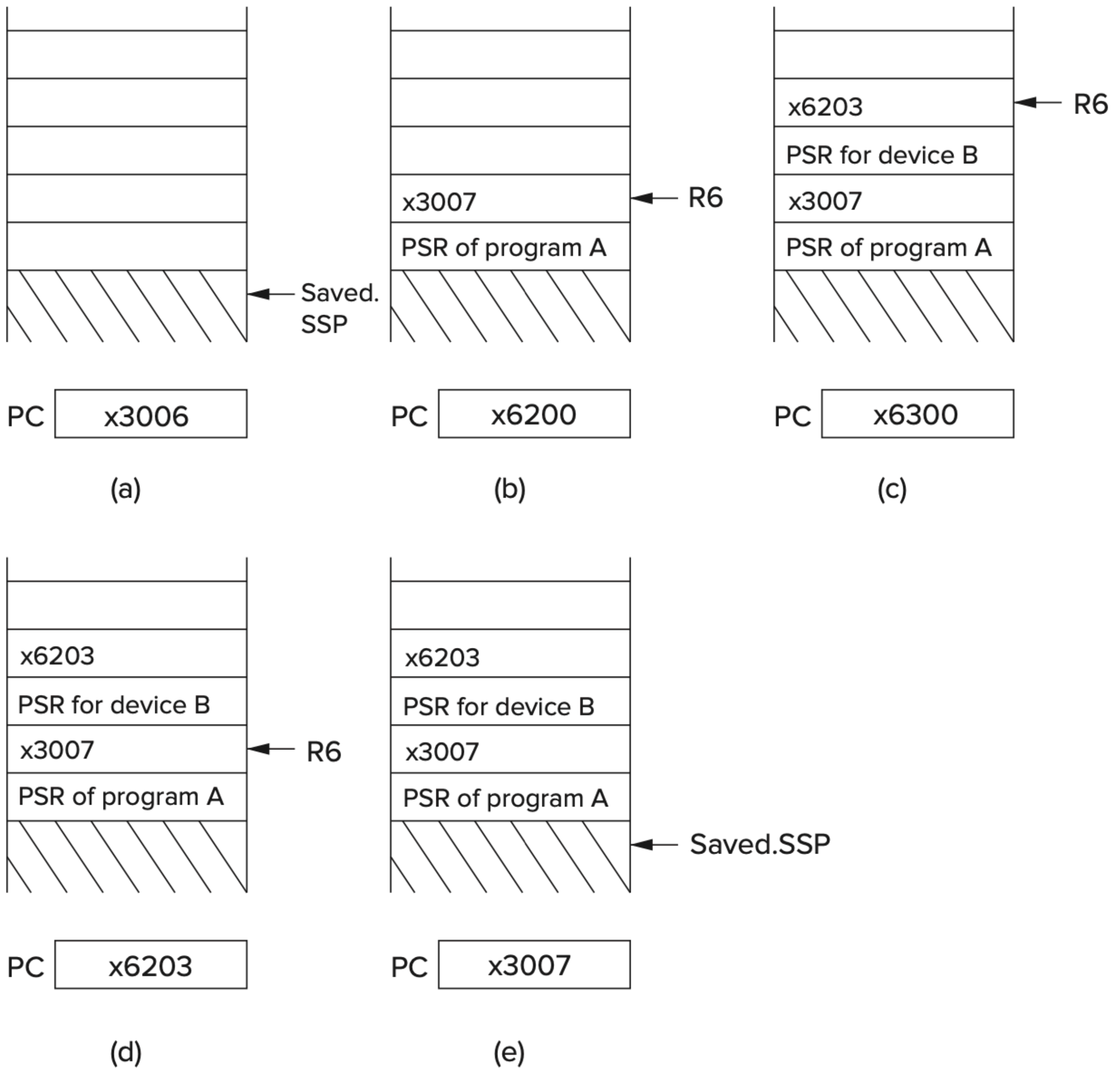


Figure 9.21 **Snapshots of the contents of the supervisor stack and the PC during interrupt-driven I/O.**

9.4 Polling Revisited: Together with Interrupt

9.4.1 The Problem

> Example:
>
> Polling Output as follows
>
> ```
> START LDI R1, DSR
>       BRnp START
>       STI  R0, DDR
> ```
>
> if we complete BR, and are to do STI. Suddenly the keyboard interrupt routine occurs, what will happen?

The keyboard interrupt occurs, the character typed is echoed, i.e., written to the DDR, and the keyboard interrupt service routine completes.

The interrupted loop body then takes over and **“knows” the monitor is ready,** so it executes the STI. … except the monitor is not ready because it has not completed the write of the keyboard service routine! **The STI of the loop body writes, but since DDR is not ready, the write does not occur.**

9.4.2 The Solution

Simple but Silly Solution

Disable all interrupts while polling was going on

A Wise Solution

An interrupt are only permitted to happen before LDI. It would have to wait for the three-instruction sequence LDI, BRzp, STI to execute, rather than for the entire polling process to complete.

```
               .ORIG x0420
             ADD   R6,R6,#-1
             STR   R1,R6,#0
             ADD   R6,R6,#-1
             STR   R2,R6,#0
             ADD   R6,R6,#-1
             STR   R3,R6,#0    ; Save R1,R2,R3 on the stack
   ;
             LDI   R1, PSR
             LD    R2,INTMASK
             AND   R2,R1,R2    ; R1=original PSR, R2=PSR with interrupts disabled

     POLL    STI   R1,PSR    ; enable interrupts (if they were enabled to begin)
             STI   R2,PSR    ; disable interrupts
             LDI   R3,DSR
             BRzp  POLL      ; Poll the DSR
             STI   R0,DDR   ; Store the character into the DDR
             STI   R1,PSR   ; Restore original PSR

             LDR   R3,R6,#0
             ADD   R6,R6,#1
             LDR   R2,R6,#0
             ADD   R6,R6,#1
             LDR   R1,R6,#0
             ADD   R6,R6,#1  ; Restore R3,R2,and R1 from the stack

             RTI

   INTMASK .FILL   xBFFF
   PSR     .FILL   xFFFC
   DSR     .FILL   xFE04
   DDR     .FILL   xFE06

           .END
```

Figure 9.22 **Polling AND allowing interrupts.**

9.5 Operating System Service Routines (TRAP)

> OSH, Operating System Help
>
> System Call = Service Call

9.5.1 Why use TRAP?

- Ignore detailed implement
- Security

9.5.2 TRAP Mechanism

Several Elements

- **Service Routines**

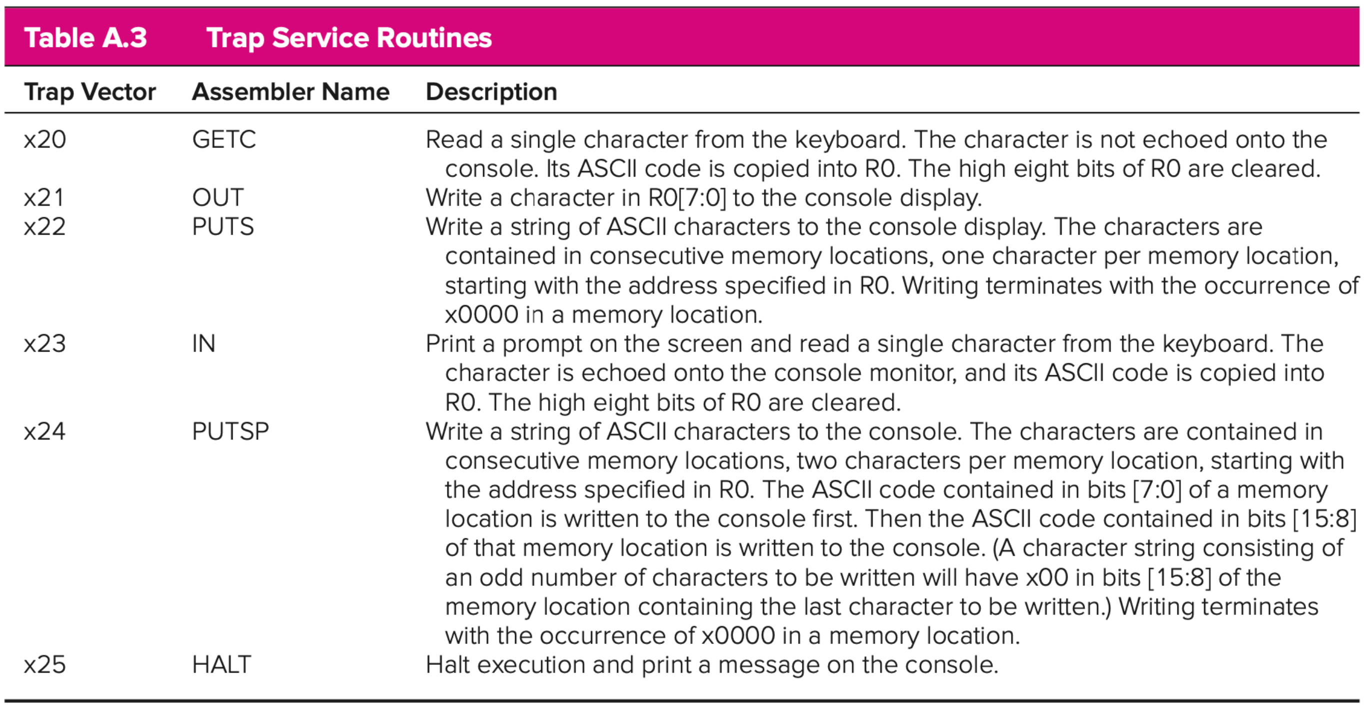


Table A.3 Trap Service Routines

Trap Vector	Assembler Name	Description
x20	GETC	Read a single character from the keyboard. The character is not echoed onto the console. Its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x21	OUT	Write a character in R0[7:0] to the console display.
x22	PUTS	Write a string of ASCII characters to the console display. The characters are contained in consecutive memory locations, one character per memory location, starting with the address specified in R0. Writing terminates with the occurrence of x0000 in a memory location.
x23	IN	Print a prompt on the screen and read a single character from the keyboard. The character is echoed onto the console monitor, and its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x24	PUTSP	Write a string of ASCII characters to the console. The characters are contained in consecutive memory locations, two characters per memory location, starting with the address specified in R0. The ASCII code contained in bits [7:0] of a memory location is written to the console first. Then the ASCII code contained in bits [15:8] of that memory location is written to the console. (A character string consisting of an odd number of characters to be written will have x00 in bits [15:8] of the memory location containing the last character to be written.) Writing terminates with the occurrence of x0000 in a memory location.
x25	HALT	Halt execution and print a message on the console.

- **The Trap Vector Table**: [x0000, x00FF], A table of the starting address of these service routines
- **TRAP instructions**
- **Linkage back** to the user program - RTI

9.5.3 TRAP Instruction

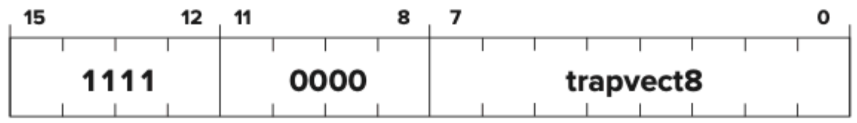
TRAP

System Call

Assembler Format

TRAP   trapvector8

Encoding



Operation

```
TEMP=PSR;
if (PSR[15] == 1)
   Saved_USP=R6 and R6=Saved_SSP;
   PSR[15]=0;
Push TEMP,PC† on the system stack
PC=mem[ZEXT(trapvect8)];
```

Description

If the the program is executing in User mode, the User Stack Pointer must be saved and the System Stack Pointer loaded. Then the PSR and PC are pushed on the system stack. (This enables a return to the instruction physically following the TRAP instruction in the original program after the last instruction in the service routine (RTI) has completed execution.) Then the PC is loaded with the starting address of the system call specified by trapvector8. The starting address is contained in the memory location whose address is obtained by zero-extending trapvector8 to 16 bits.

Example

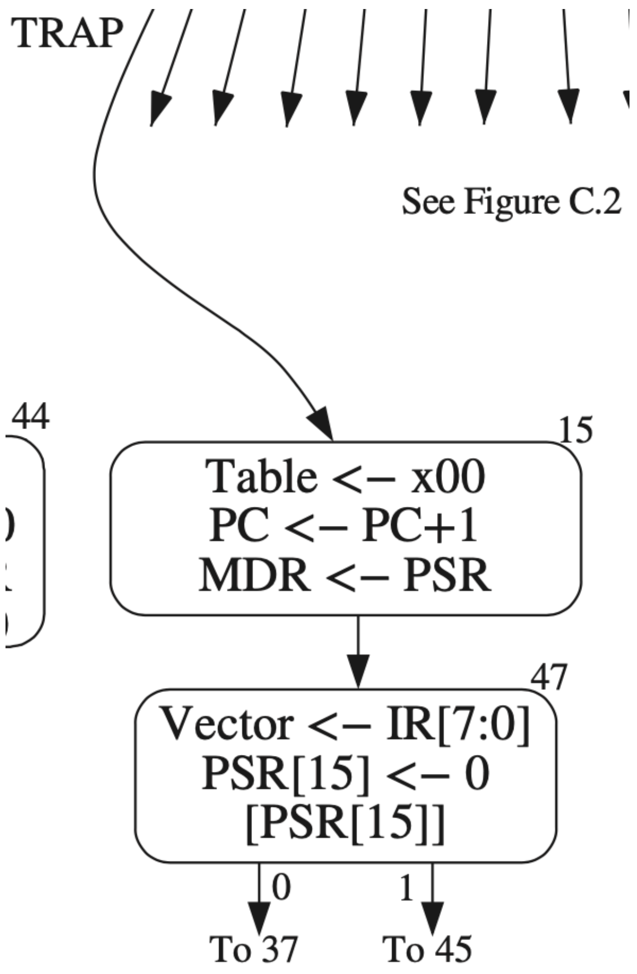
```
TRAP   x23   ; Directs the operating system to execute the IN system call.
             ; The starting address of this system call is contained in
             ; memory location x0023.
```

Note:

Memory locations x0000 through x00FF, 256 in all, are available to contain starting addresses for system calls specified by their corresponding trap vectors. This region of memory is called the Trap Vector Table. Table A.3 describes the functions performed by the service routines corresponding to trap vectors x20 to x25.

† This is the incremented PC.

Finite State Machine



state 37 and 45: Go to Interrupt

9.5.4 Example: Trap x23, IN

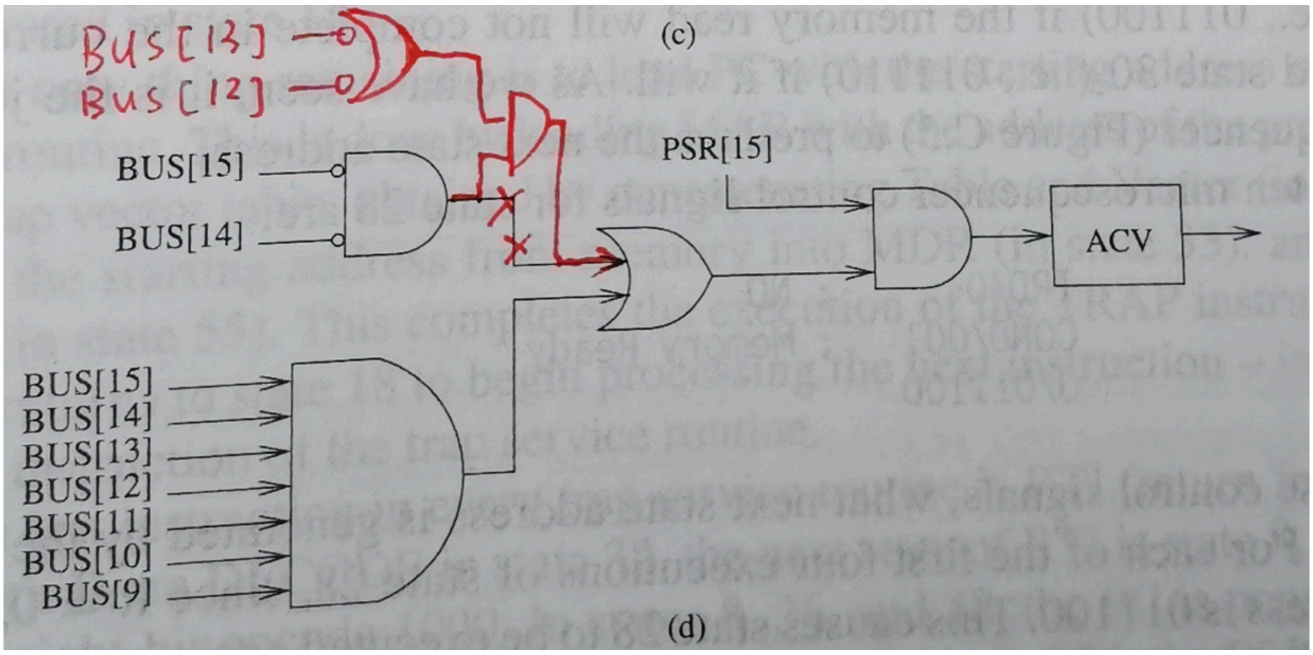
```
;  Service Routine for Keyboard Input
;
        .ORIG   x04A0
START   ST      R1,SaveR1       ; Save the values in the registers
        ST      R2,SaveR2       ; that are used so that they
        ST      R3,SaveR3       ; can be restored before RET
;
        LD      R2,Newline
L1      LDI     R3,DSR          ; Check DDR --  is it free?
        BRzp    L1
        STI     R2,DDR          ; Move cursor to new clean line
;
        LEA     R1,Prompt       ; Prompt is starting address
                                ; of prompt string
Loop    LDR     R0,R1,#0        ; Get next prompt character
        BRz     Input           ; Check for end of prompt string
L2      LDI     R3,DSR
        BRzp    L2
        STI     R0,DDR          ; Write next character of
                                ; prompt string
        ADD     R1,R1,#1        ; Increment prompt pointer
        BRnzp   Loop
;
Input   LDI     R3,KBSR         ; Has a character been typed?
        BRzp    Input
        LDI     R0,KBDR         ; Load it into R0
L3      LDI     R3,DSR
        BRzp    L3
        STI     R0,DDR          ; Echo input character
                                ; to the monitor
;
L4      LDI     R3,DSR
        BRzp    L4
        STI     R2,DDR          ; Move cursor to new clean line
        LD      R1,SaveR1       ; Service routine done, restore
        LD      R2,SaveR2       ; original values in registers.
        LD      R3,SaveR3
        RTI                     ; Return from Trap
;
SaveR1  .BLKW   1
SaveR2  .BLKW   1
SaveR3  .BLKW   1
DSR     .FILL   xFE04
DDR     .FILL   xFE06
KBSR    .FILL   xFE00
KBDR    .FILL   xFE02
Newline .FILL   x000A           ; ASCII code for newline
Prompt  .STRINGZ "Input a character>"
        .END
```

More Examples see p333-339

9.7 Exceptions

Access Control Violation(ACV)

若总线上的地址，也就是下一条指令的地址 (after MAR <- PC, PC <- PC + 1) <x3000 或者 ≥xFE00，且 当前为 user mode，则set ACV


(c)
BUS[13]
BUS[12]
BUS[15]
BUS[14]
PSR[15]
ACV
BUS[15]
BUS[14]
BUS[13]
BUS[12]
BUS[11]
BUS[10]
BUS[9]
(d)