ECE 220

Lecture x0019
Privilege & Interrupts in LC3

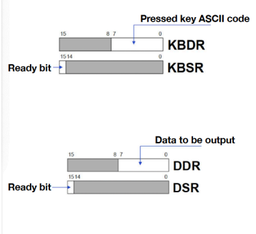
Recap + reminders

- Trees
  - Concepts & properties
  - Traversals: preorder/ inorder/postorder
  - Binary search trees
  - C to LC3 (structs, trees, linked lists)
- Reminders
  - Extra credit quiz
  - Programming competition
    - May 6th
  - Must read Chapter 9

Today

- Change gears & talk about *interrupts* in LC3
- Recall: `KBSR/KBDR` & `DSR/DDR`?
  - When did we see these acronyms?
  - How did we implement I/O in LC3 before?

LC3 - Input/Output (IO)



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 displayed on the screen

LC3 - Input from keyboard

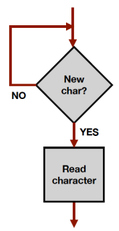
Basic routine

Handshaking is performed using KBSR & KBDR

- When user presses a key
  - Its ASCII code is placed in KBDR[0:7]
  - KBSR[15] is set to 1 *(ready bit)*
  - Keyboard is disabled, i.e., any further keypress is ignored
- When KBDR is read by CPU
  - KBSR[15] is set to 0
  - Keyboard is enabled

LC3 - Input from keyboard

Basic routine




```
.ORIG x3000
;Create a loop to
check KBSR

;If ready bit unset
loop again

;If ready bit set,
read KBDR into R0

KBSR .FILL xFE00
KBDR .FILL xFE02
```

```
.ORIG x3000
KPOLL LDI R1, KBSR
      BRzp KPOLL
      LDI R0, KBDR
; Other instructions
; …
; …
HALT
KBSR .FILL xFE00
KBDR .FILL xFE02
.END
```

LC3 - Display to console

Basic routine

Handshaking is performed using DSR & DDR

- When display is ready to present a character
  - DSR[15] is set to 1 *(ready bit)*
- When a new character is written to DDR
  - DSR[15] is set to 0
  - Any other chars written to DDR are ignored
  - DDR[7:0] is displayed

Slides from Lecture #1

Lesson objectives

- Understand and be able to articulate why interrupt driven I/O is preferable to polling-based I/O.

- Understand the mechanism(s) by which execution can be suspended and resumed.

- Understand the role of supervisor and user stack in enabling interrupts

- Understand concepts of privilege and priority as it relates to processes in a modern computer processor

Previously - I/O using Polling or TRAPs

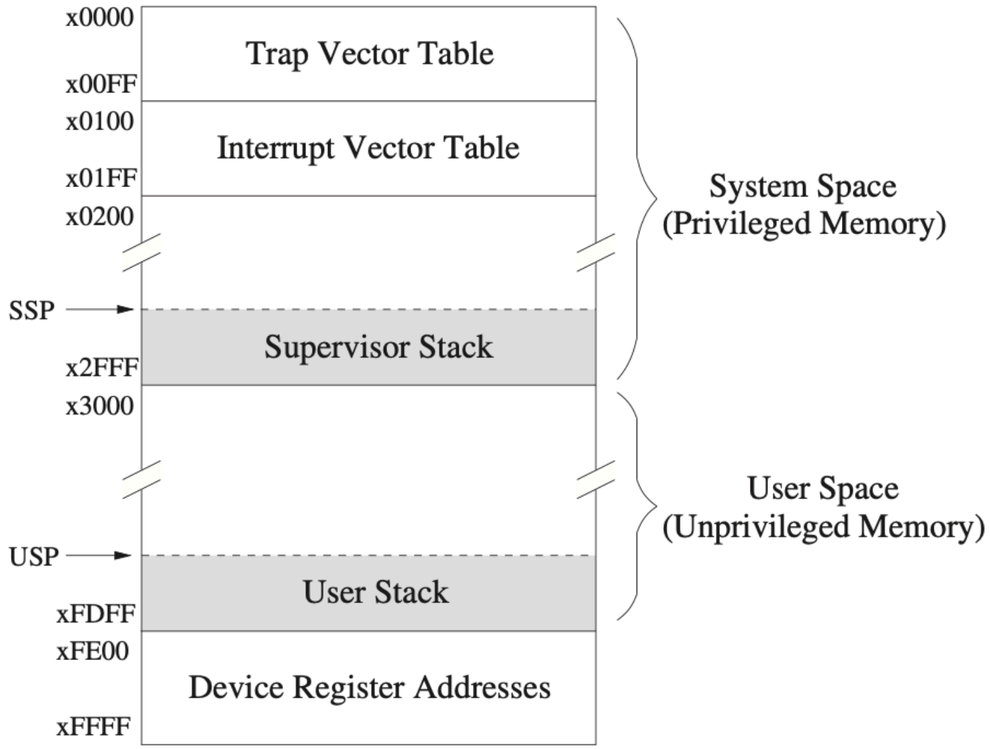
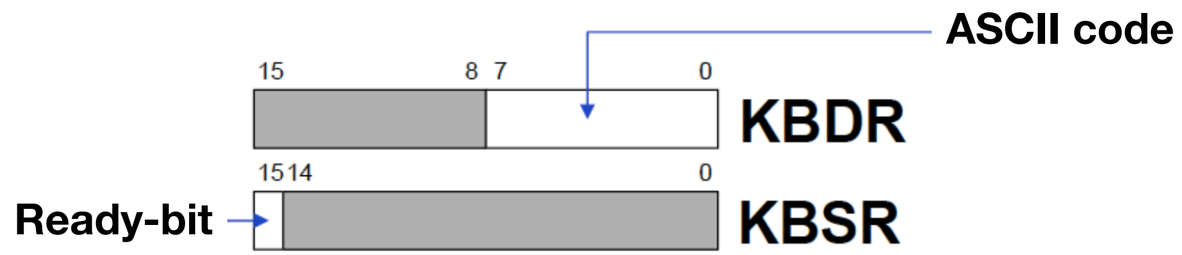
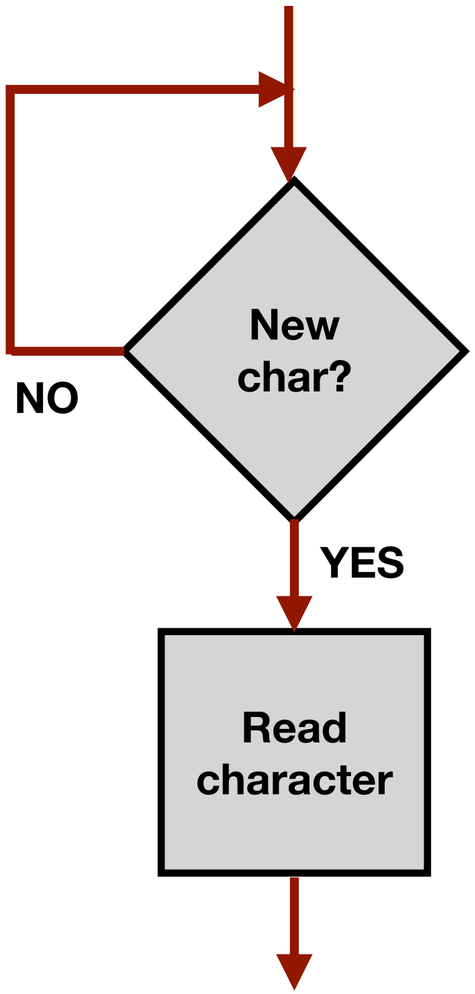


Figure A.1 - P&P 3rd Ed.

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



Any problems with polled I/O?



- Suppose we want to type 100 characters:
  - User can input at speed of 60 WPM, assume, eight characters/word:
    - How long to type 100 characters?

- Processor cannot do anything else while it waits on *next* character!
  - 12.5 seconds spend just polling!

- How can we free up the processor to do other tasks?

Interrupt driven I/O

- Key idea: An I/O device can
  - force running program to stop
  - have processor execute different program that carries out needs of I/O device, and *then*
  - resume execution of stopped program as if nothing had happened

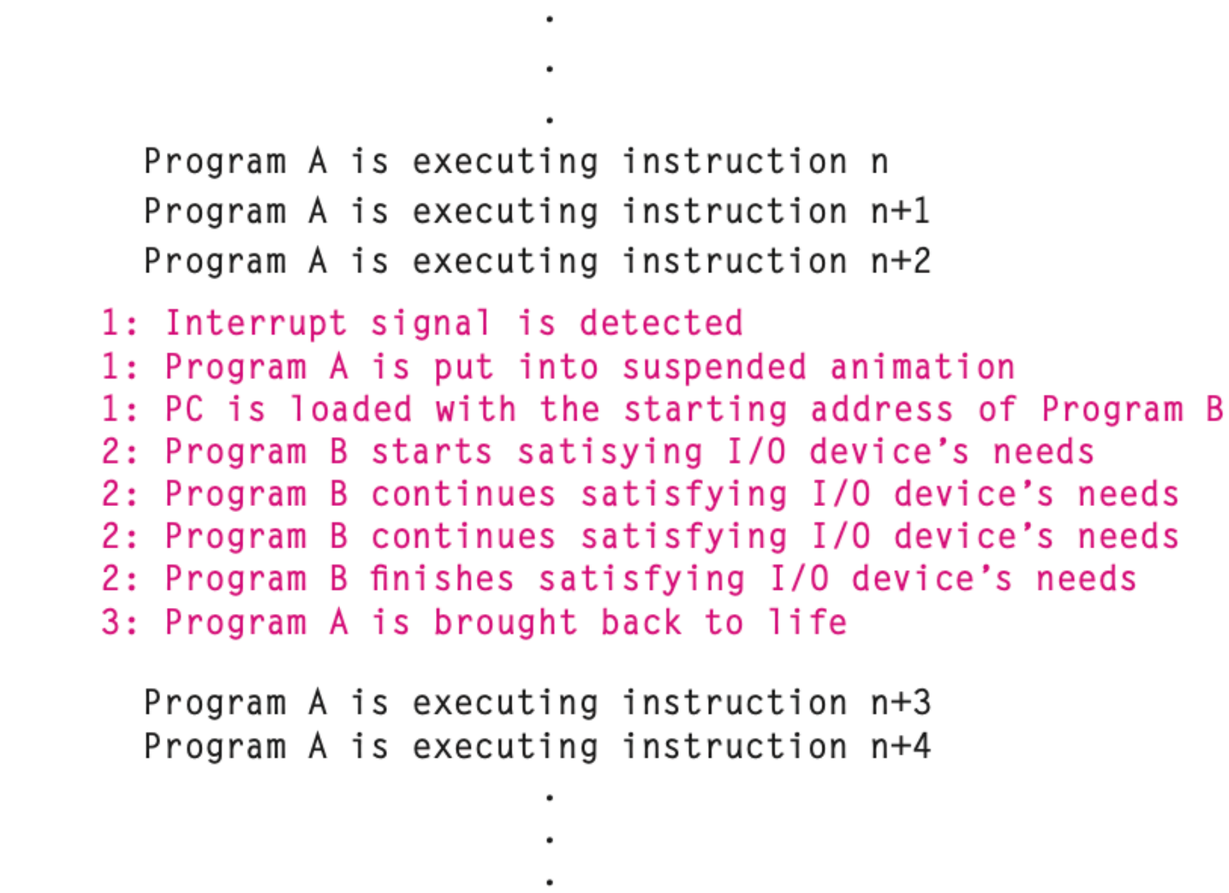


Figure 9.17 Instruction execution flow for interrupt-driven I/O.

Now - I/O using interrupts

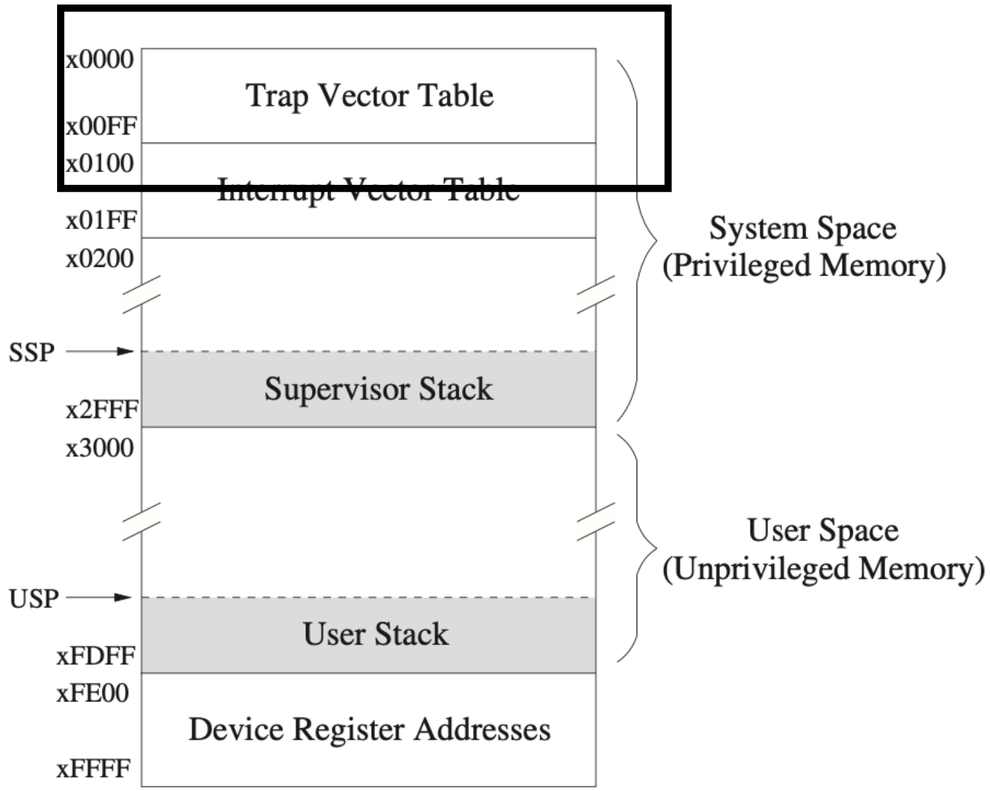
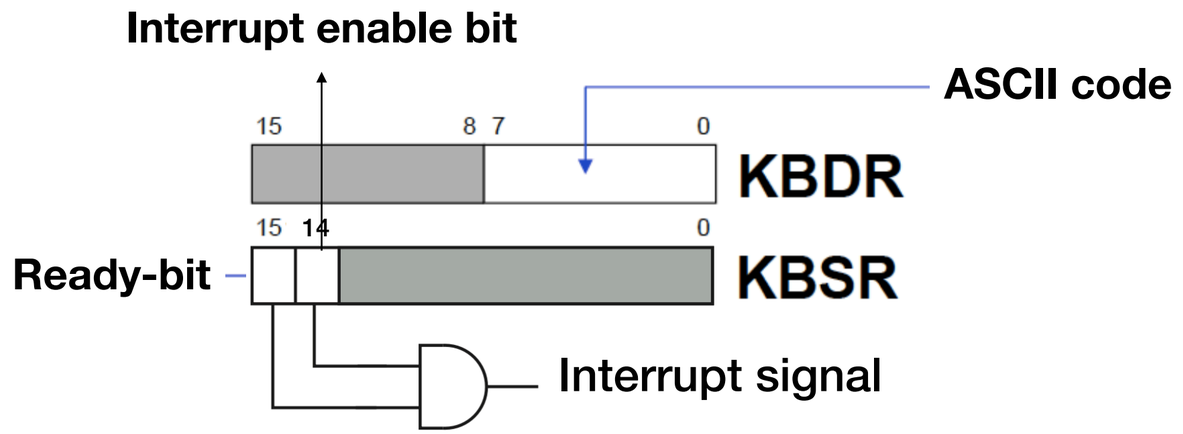


Figure A.1 - P&P 3rd Ed.

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
		The IE bit (bit[14]) specifies if the I/O device has permission to generate interrupt signal



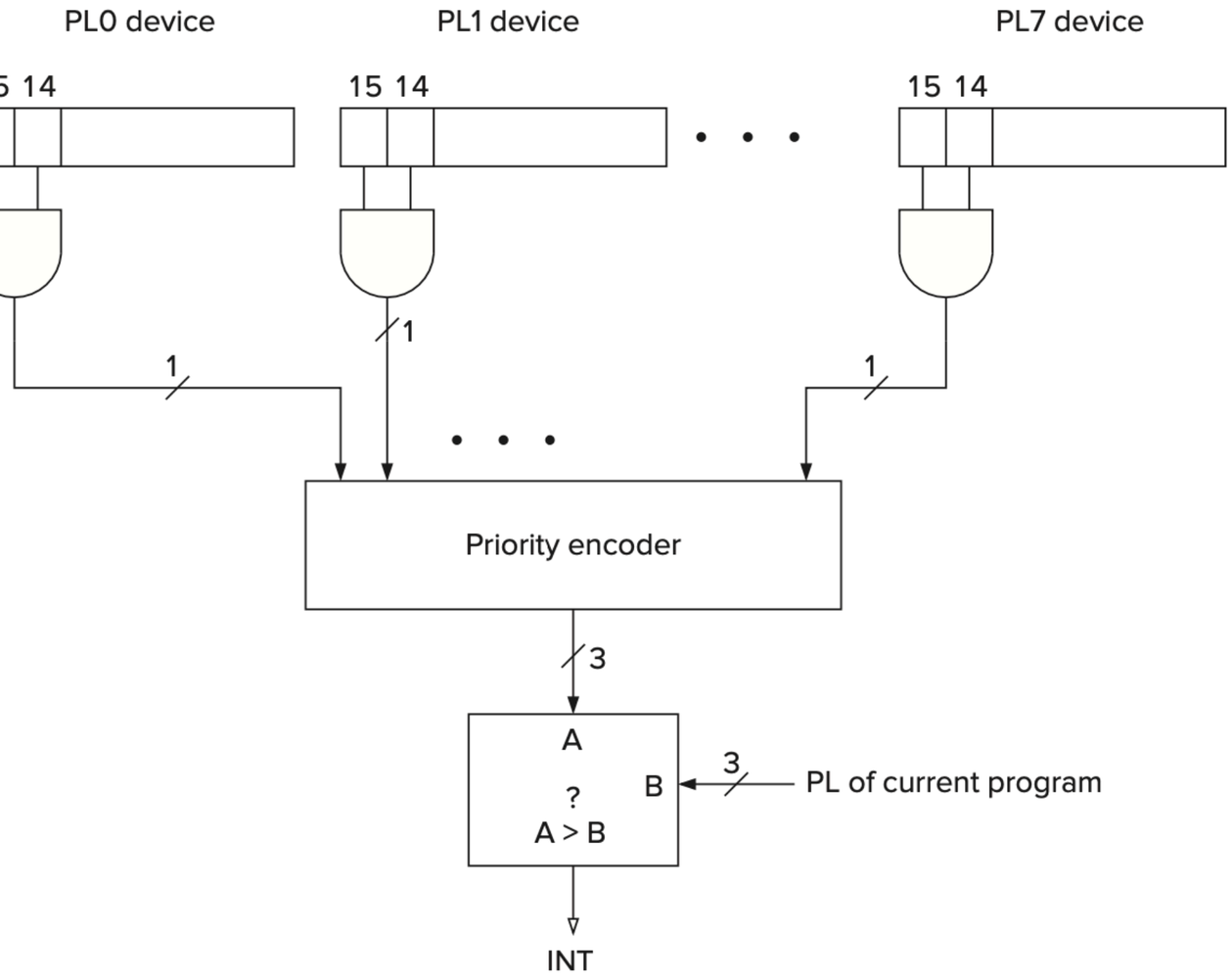
What needs to happen?

For interrupt driven I/O to work:

1. The I/O device **must want service.**
   - Ready bit
2. The device must have the **right to request** the service.
   - Interrupt enable bit
3. The device request must be **more urgent** than what the processor is currently doing.
   - ***Priority levels*:** PL0 < PL1 < … < PL6

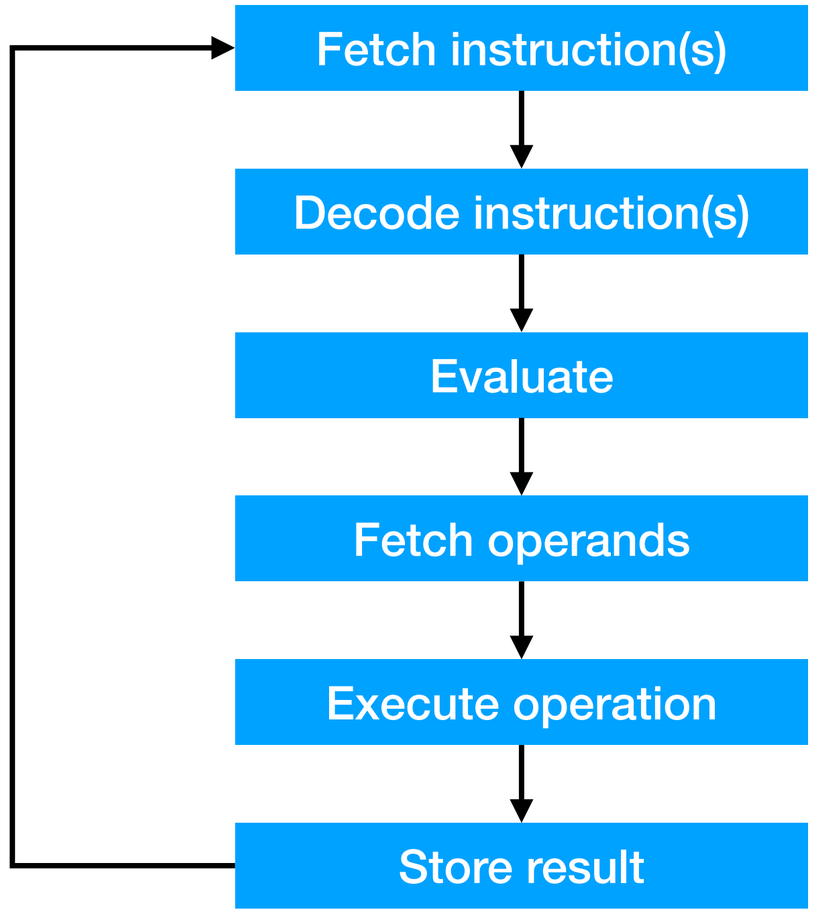
Generating an INTerrupt

- All devices asserting an interrupt signal are compared
- Highest PL request is compared to current program's PL
  - Interrupt is initiated.



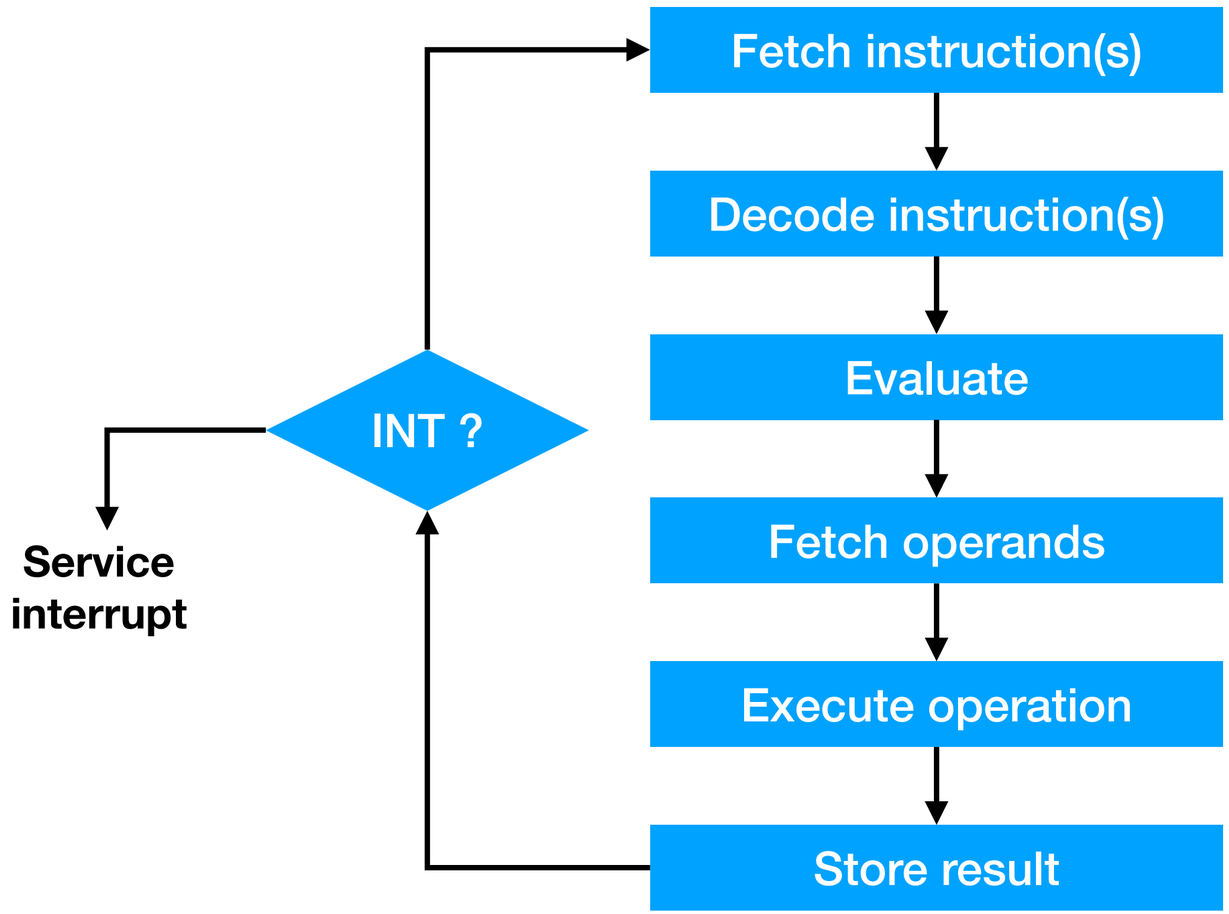
How processor detects INTerrupts

- Recall instruction cycle.
- Processor will check if INT is asserted at the end of each instruction cycle before starting a new FETCH



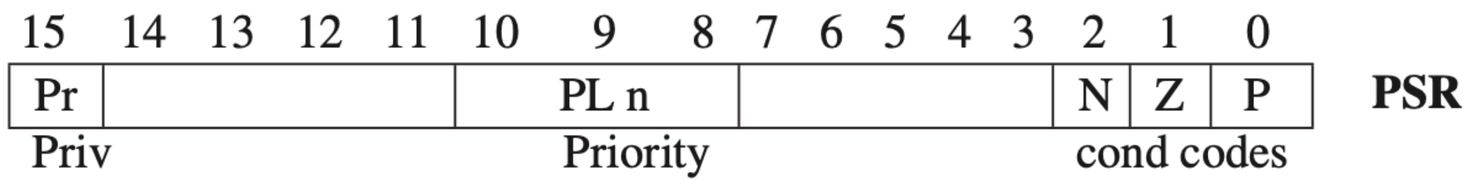
How processor detects INTerrupts

- Control unit needs to:
  - save state information to be able to resume interrupted program
  - load PC with the starting address of ISR (**Interrupt Service Routine**)
  - resume old process on RTI (coming later)



Handling the INTerrupt

- What should be saved for program to resume?
  - PC (Program Counter)
  - GPRs (General Purpose Registers)?
    - Callee saved
- Processor Status Register (PSR)
  - PSR[15]=0 (supervisor), PL higher is more urgent



Privilege vs. priority

- Privilege
  - Privilege is all about the right to do something, such as execute a particular instruction or access a particular memory location.
- Priority
  - Priority is all about the urgency of a program to execute. Every program is assigned a priority, specifying its urgency as compared to all other programs.

See Section 9.1.1.3

User stack vs. supervisor stack

- Portion of privileged memory reserved for stack in supervisor mode
- `R6` is used to denote active TOS **always**.
  - Can only run in user or supervisor mode at a time.
  - Save `R6` on switching modes using (built-in) registers, `Saved_SSP` and `Saved_USP`

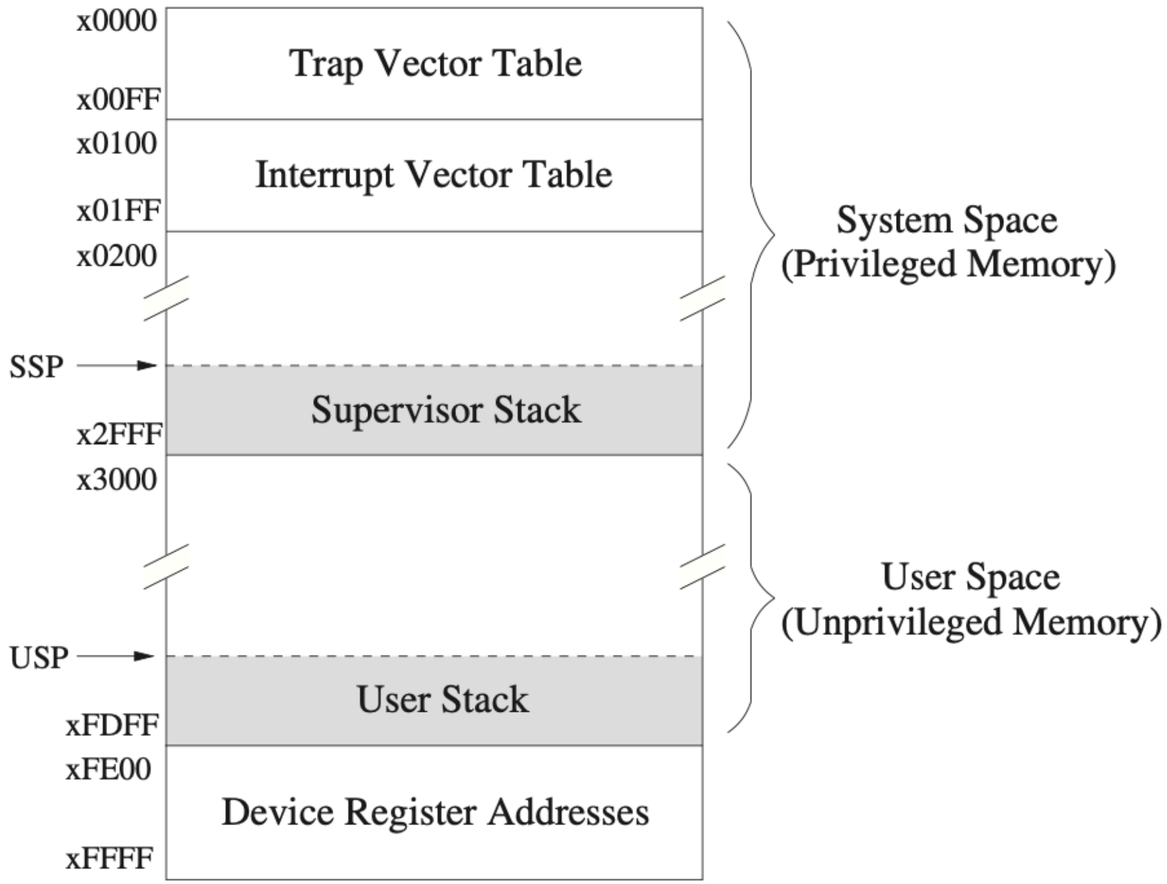


Figure A.1 - P&P 3rd Ed.

Handling the INTerrupt

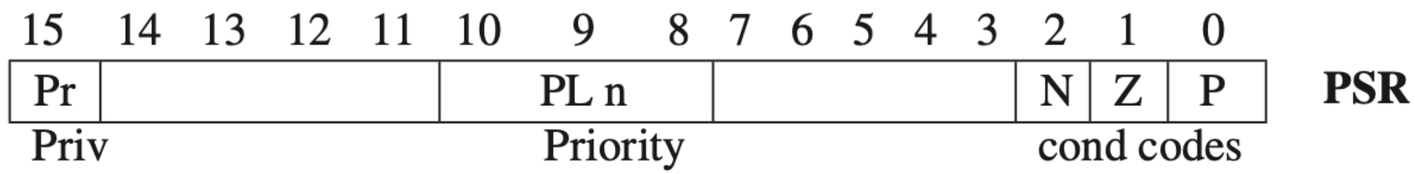
- The I/O device generating the `INT` signal also provides an 8-bit *interrupt vector* `INTV` which identifies the device

- `INTV` will determine which `ISR` to be executed to service the interrupt

  - Similar to `TRAP` vector table we have seen previously

Summary of steps

1. Processor detects asserted `INT` along with `INTV`
2. Save `R6` to `Saved_USP`. Set `R6` to `Saved_SSP`.
3. Push `PSR` & `PC` to supervisor stack in that order
4. Set `PSR[15]=0` and set `PSR[10:8]` = priority of `ISR`
    1. Set `PSR[2:0]=010` (arbitrarily sets condition codes) ⟶ **3rd Ed.**
5. Load `MAR` with `x01vv`, where `vv`= 8-bit interrupt vector, then load `MDR`
6. Set `PC=MDR`



See Section 9.4.5.1

Returning from (TRAP or) INTerrupt: RTI

- Both `TRAP` routines and `INT`errupt routines end with the `RTI` instruction.
- Pop `PC` & `PSR` from supervisor stack
- If `PSR[15]=1`, `R6=Saved.USP`
- `RTI` is *privileged*

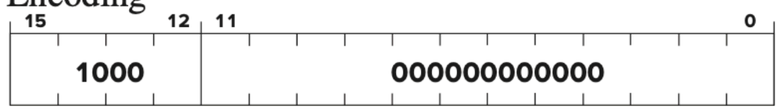
RTI

Return from Trap or Interrupt

Assembler Format

RTI

Encoding



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.

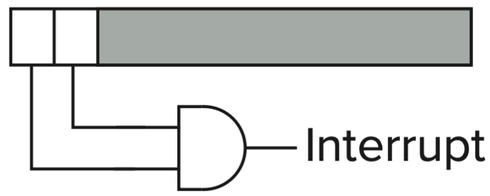
Example

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

Interrupt example

```
.ORIG   x3000
        LEA     R0, ISR_KB
        STI     R0, KBINTV  ; load ISR address to INTV
        LD      R3, EN_IE
        STI     R3, KBSR    ; set IE bit of KBSR
AGAIN   LD      R0, NUM2
        OUT
        BRnzp   AGAIN
;
ISR_KB  ST      R0, SaveR0  ; callee-save R0
        LDI     R0, KBDR    ; read a char from KB and clear ready bit
        OUT
        LD      R0, SaveR0  ; callee-restore R0
        HALT
;
EN_IE   .FILL   x4000 ; To enable the IE bit
NUM2    .FILL   x0032 ; ASCII Code for '2'
KBSR    .FILL   xFE00
KBDR    .FILL   xFE02
KBINTV  .FILL   x0180 ; INT vector table address for keyboard
SaveR0  .BLKW   #1
.END
```

What does this program do?



Exceptions

- Exceptions happen when something … well un*expected* happens within the processor.
  - **Examples**: Privilege mode violation (PMV), illegal/invalid opcodes, accessing privileged memory (ACV: Access Control Violation)
- Exception services routines occupy memory locations `x0100` to `x017F`, mechanism exactly like INTerrupts except …
  - Priority level is **not changed.**

See Appendix C7 for more details.

Next lecture(s)

- Examples & course review

- Problem solving & exam preparation tips