ECE 220: Computer Systems & Programming

Lecture 25: Interrupts and Exceptions

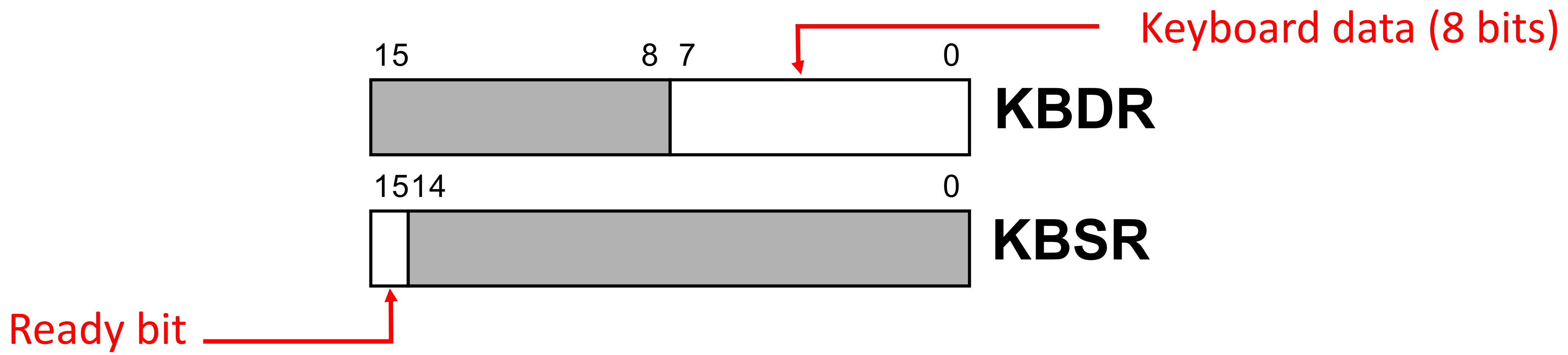
Thomas Moon

April 23, 2024

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

- ICES form:
  If 70% of students complete ICES, the whole class get 1% credit

From Lec2 : Input from keyboard



When a character is typed in …

1. Its ASCII code is placed in bits [7:0] of KBDR (bits [15:8] are always zero)
2. The “ready bit” (KBSR[15]) is set to one
3. Keyboard is disabled -- any typed characters will be ignored

When KBDR is read …

1. KBSR[15] is set to zero
2. Keyboard is enabled

From Lec2 : I/O Types

3. Who *controls* the interaction?

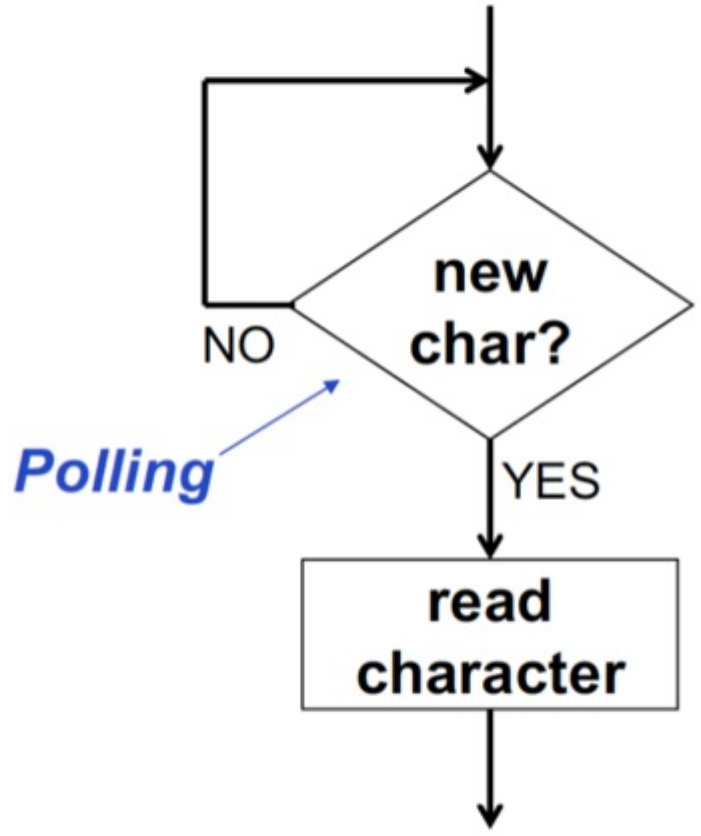
Polling

- Processor controls the interaction
- Keep asking whether the I/O data is ready
- *"Are you ready? Are you ready? …"*

Interrupt-Driven

- I/O controls the interaction
- Processor is interrupted by announcement from I/O
- *"Wake me up when you are ready."*

Basic Input Routine (by polling)



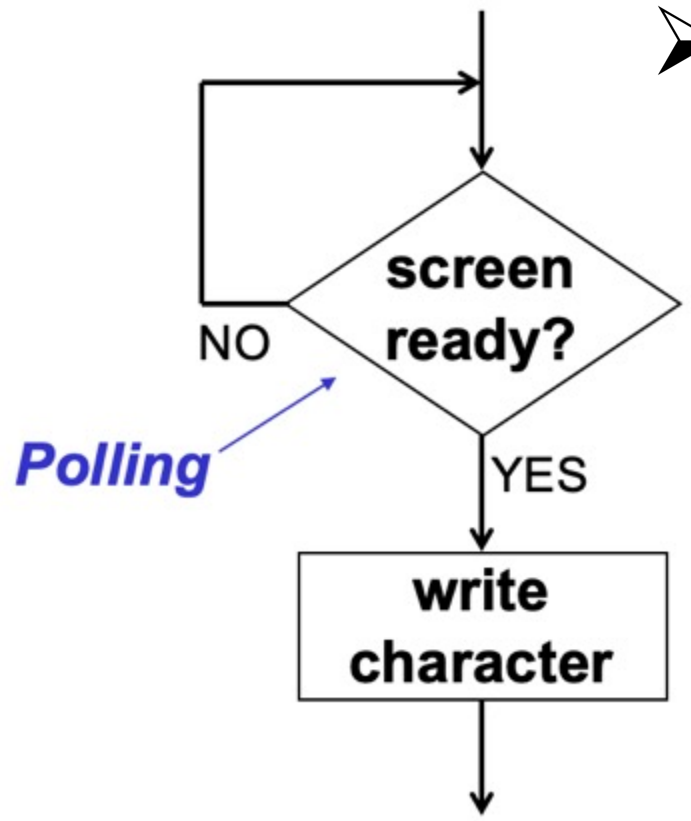
- Read a single character from keyboard.

  1. Load KBSR to a register (R0-R7)
  2. Check its MSB by sign

     → *Z or P: repeat 1*

     → *N: Load KBDR to a register*

Basic Output Routine (by polling)



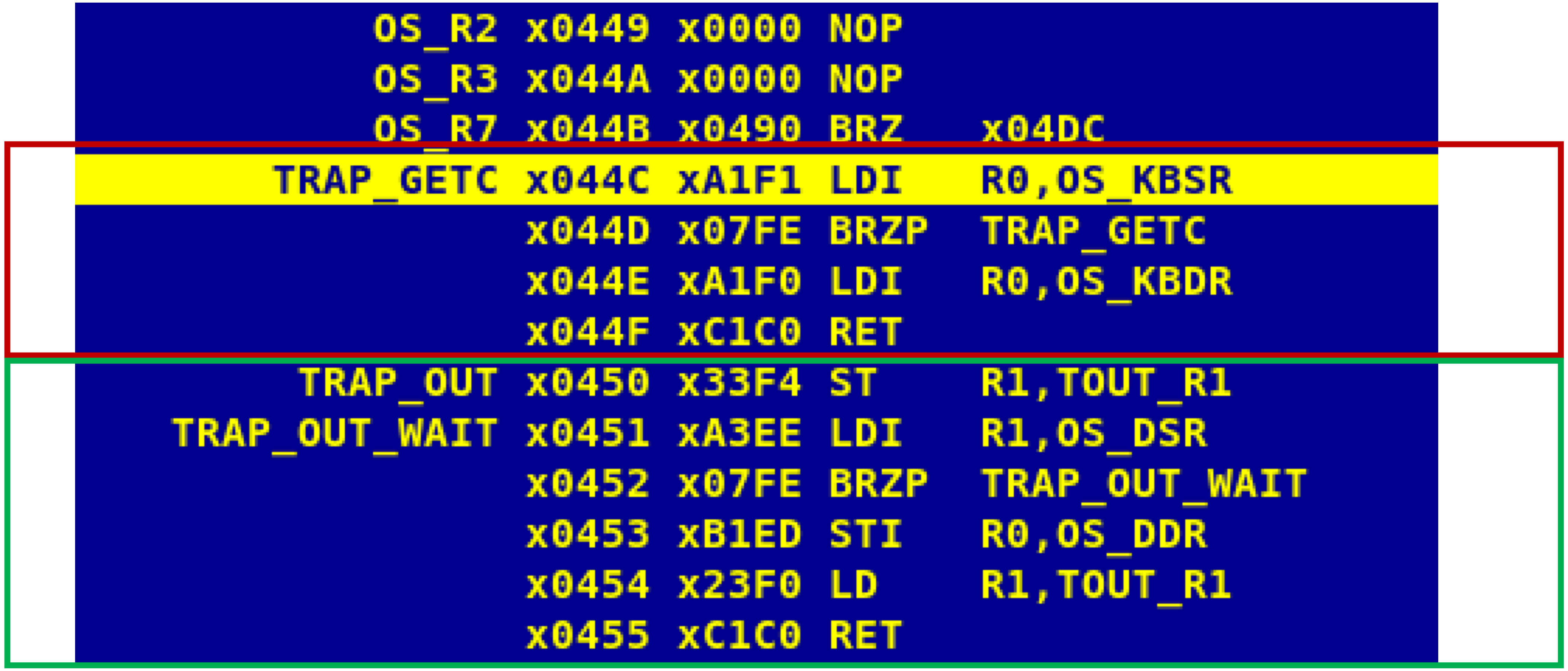
- Display a single character to monitor.

  1. Load DSR to a register (R0-R7)
  2. Check its MSB sign

     → *Z or P: repeat 1*

     → *N: Store a character to DDR*

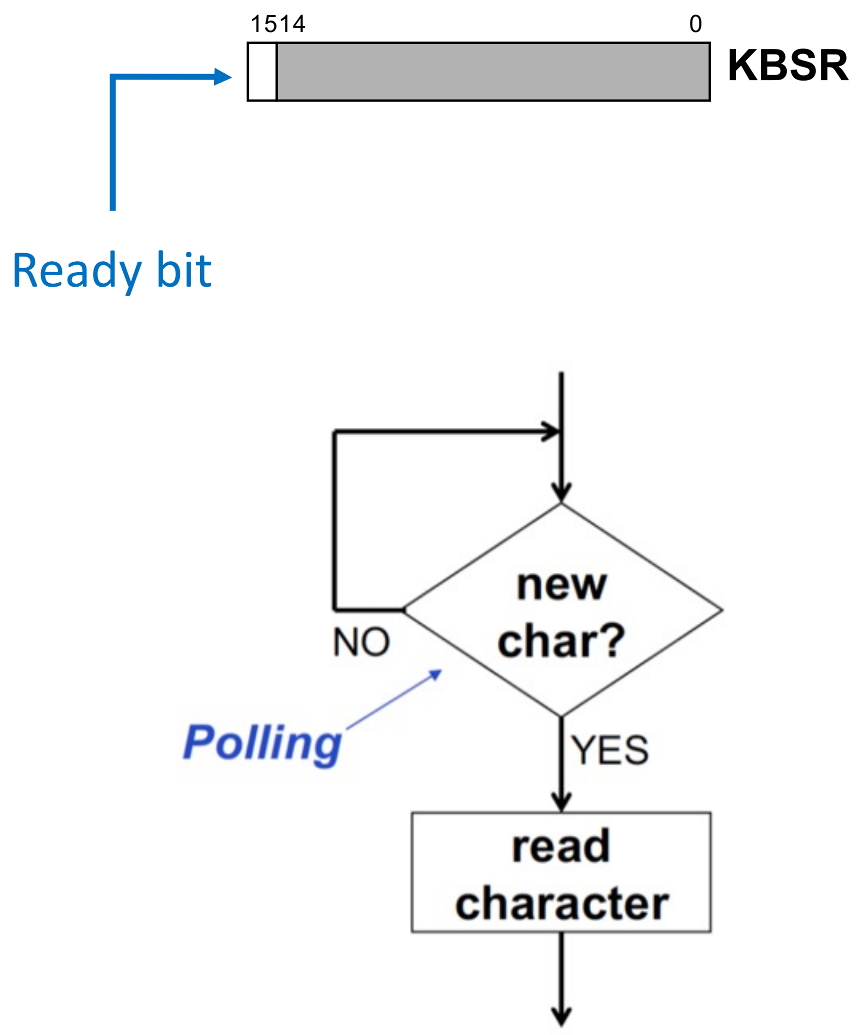
GETC/OUT vs input.asm & output.asm

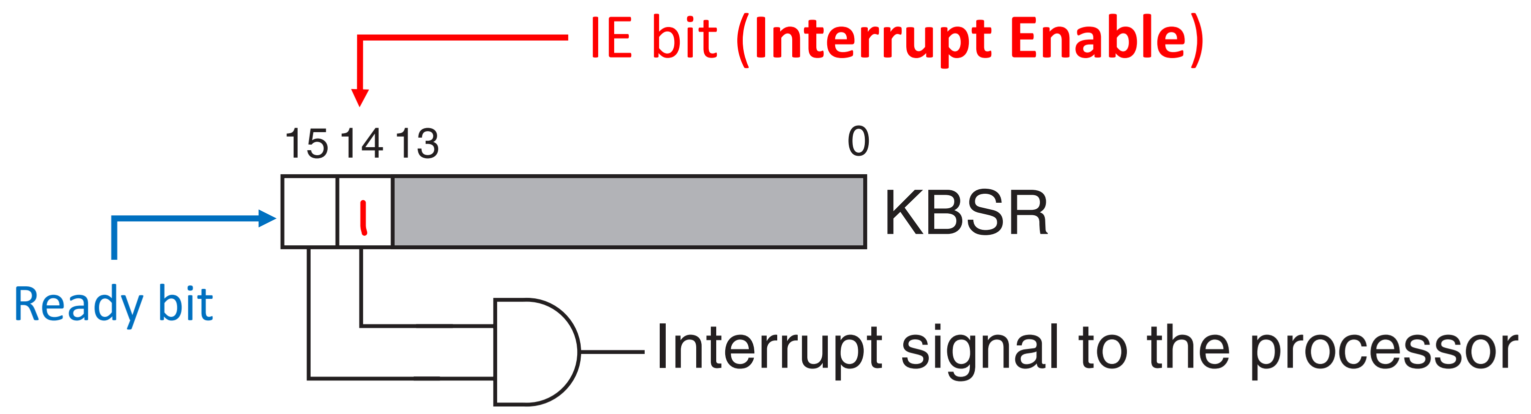


```
          OS_R2 x0449 x0000 NOP
          OS_R3 x044A x0000 NOP
          OS_R7 x044B x0490 BRZ    x04DC
      TRAP_GETC x044C xA1F1 LDI    R0,OS_KBSR
                x044D x07FE BRZP   TRAP_GETC
                x044E xA1F0 LDI    R0,OS_KBDR
                x044F xC1C0 RET
       TRAP_OUT x0450 x33F4 ST     R1,TOUT_R1
  TRAP_OUT_WAIT x0451 xA3EE LDI    R1,OS_DSR
                x0452 x07FE BRZP   TRAP_OUT_WAIT
                x0453 xB1ED STI    R0,OS_DDR
                x0454 x23F0 LD     R1,TOUT_R1
                x0455 xC1C0 RET
```

Polling vs Interrupt-driven I/O

Ready bit: set by keyboard
IE bit: set by software





- **IE = 0**
  - I/O device will NOT be able to interrupt
  - Polling
- **IE = 1**
  - Interrupt-driven I/O enabled
  - Interrupt request generated as soon as Ready bit sets (a key typed)

Flow of Interrupt-driven I/O

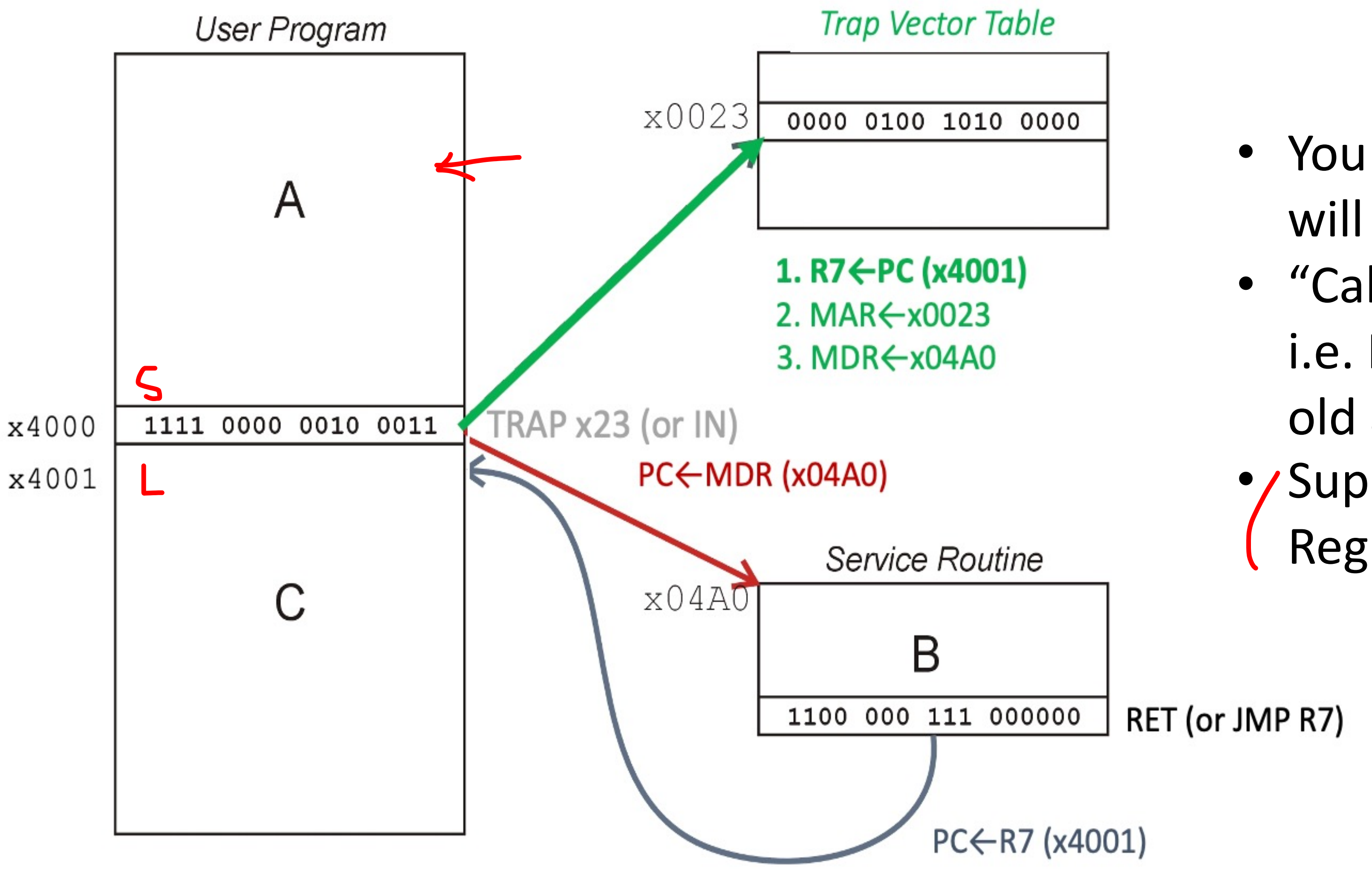
Stage1: **Initiate** the interrupt

☆Stage1.5: Prepare/Transfer

Stage2: **Service** the interrupt

Stage3: **Return** from the interrupt

TRAP/Subroutine vs Interrupt



- You don't know "WHEN" an interrupt will happen.
- "Caller-save" is not possible
  i.e. R7 cannot be recovered using the old approach.
- Supervisor Stack + Special Internal Register

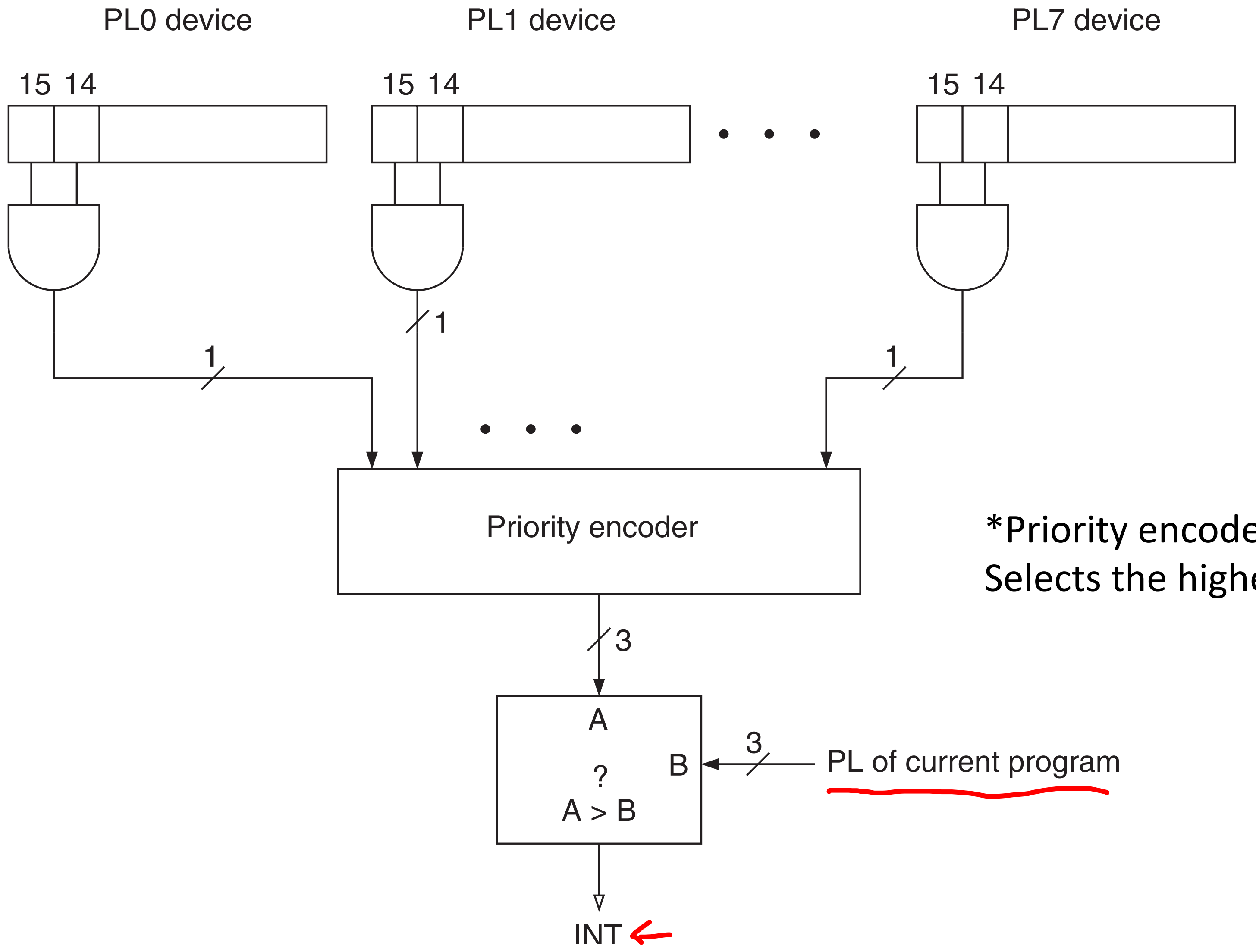
Stage1: Initiate

- 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)

- 3 conditions to interrupt the processor
  1. The device must want to interrupt (**Ready bit** in KBSR)
  2. The device must have the right to request (**IE bit** in KBSR)
  3. The request must be more urgent than the processor's current task (**priority level**: PL0-PL7 (higher is more urgent), e.g. keyboard is PL4)

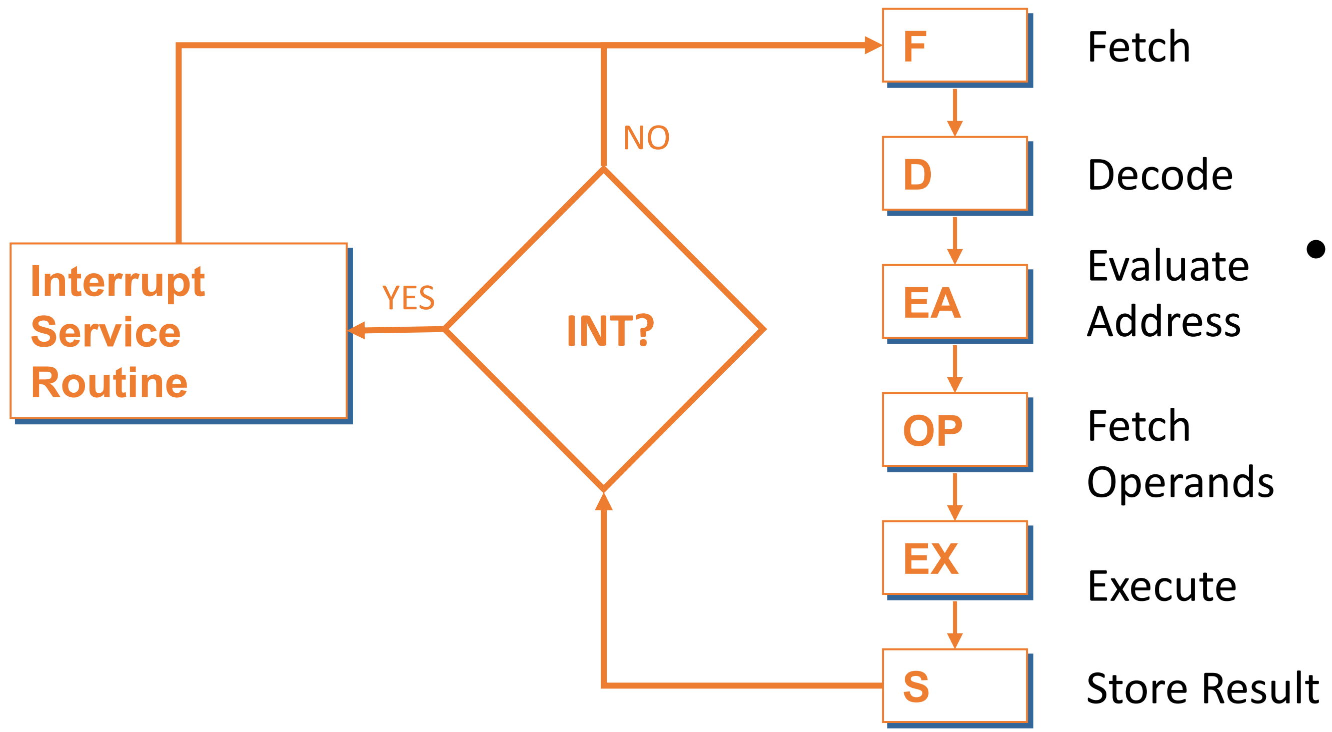
Stage1: Initiate

Generate INT Signal



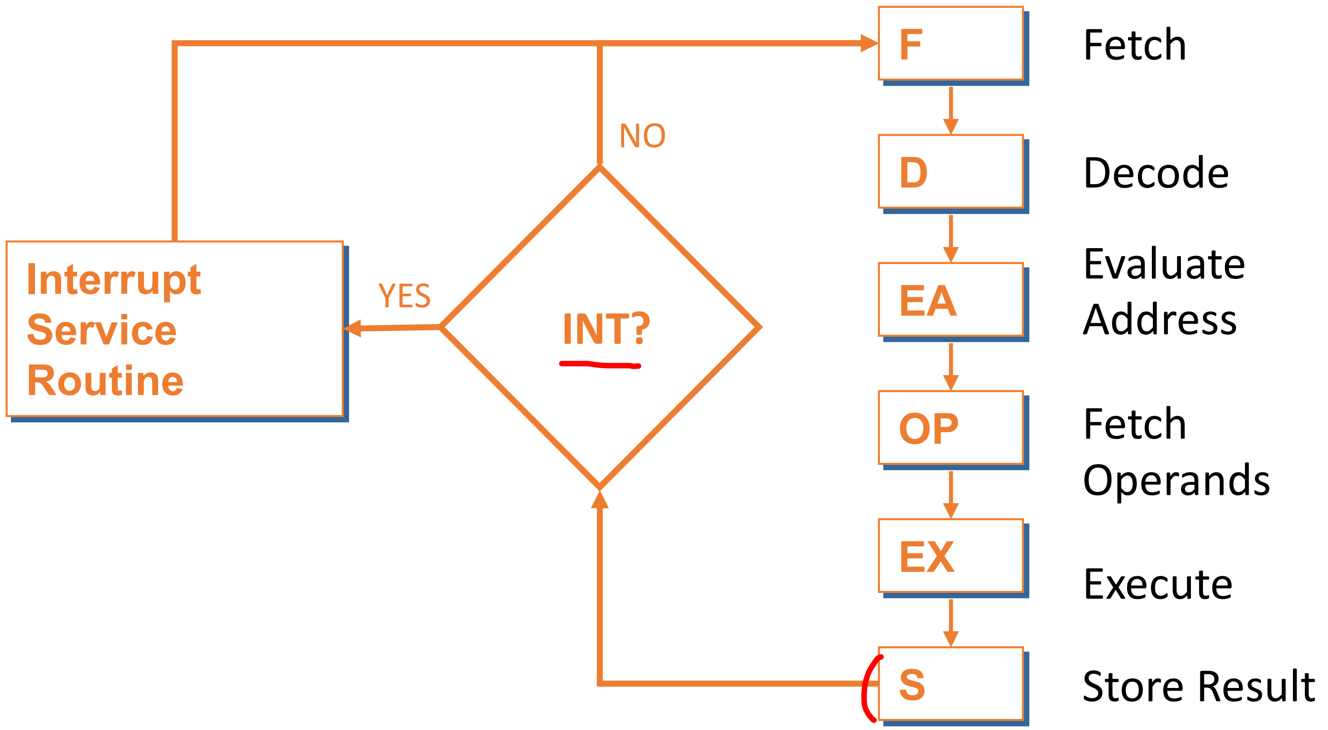
*Priority encoder:
Selects the highest priority request from all devices

When INT is checked?



- If INT = 0
  1. Return to FETCH phase to start the next instruction
- If INT = 1
  1. Prepare the interrupt
  2. Transfer to Interrupt Service Routine (ISR)

When INT is checked?



Q. If INT is issued in the middle of the instruction cycle, i.e. 'ADD R0, R1, R2,' will the result be updated before serving ISR?

[✓ A. Yes]

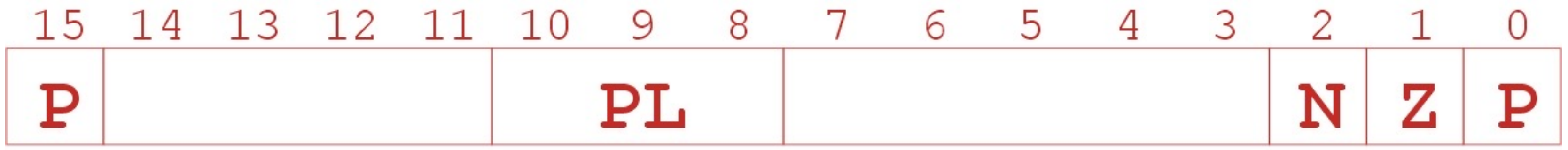
B. No

Stage1.5: Prepare/Transfer

1. (Prepare) The state of the interrupted program must be saved to resume later.
2. (Transfer) The state of the ISR must be loaded to begin the service.

State of a program

- Contents of all the general purpose registers (R0-R7) *<- If needed, they will be saved by ISR (callee-save)*
- **PC**
- **PSR** (Processor Status Register)



Privileged Mode
0: Privileged (Supervised)
1: Unprivileged (User)

Priority Level

Condition Code

Where to save the state? – Supervisor Stack

Stage1.5: Prepare/Transfer

Supervisor Stack

A special region of memory used as the stack for ISR

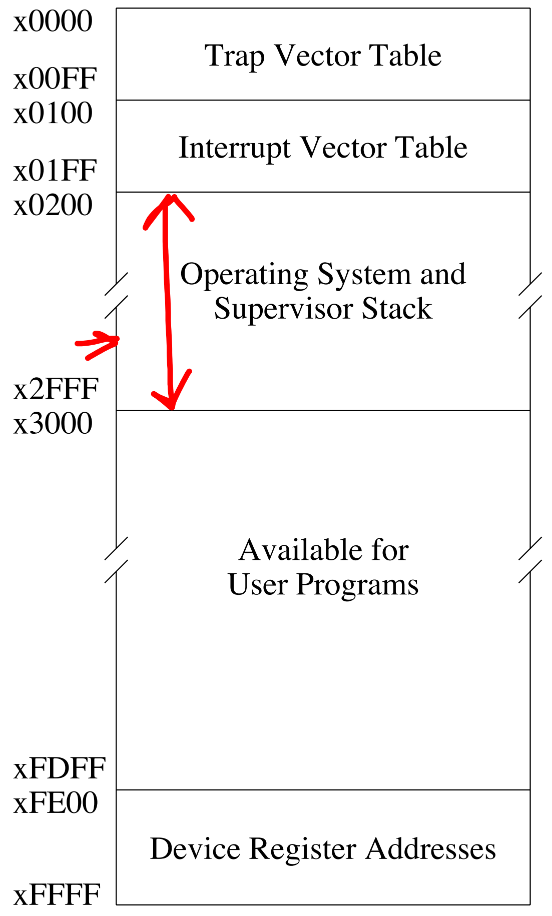
- Supervisor Stack Pointer (SSP)
- Saved.SSP: **Internal register** to store SSP

TOS

User Stack

A stack accessed by user programs

- User Stack Pointer (USP)
- Saved.USP: **Internal register** to store USP


x0000
x00FF
Trap Vector Table
x0100
x01FF
Interrupt Vector Table
x0200
Operating System and Supervisor Stack
x2FFF
x3000
Available for User Programs
xFDFF
xFE00
Device Register Addresses
xFFFF

- Access both stacks using R6 as the stack pointer.
- When switching **from User mode to Supervisor mode**, save R6 to Saved.USP *(because R6 gets automatically corrupted)* .

INT = 1 → **Stage1.5: Prepare/Transfer** → ISR

Detail Steps

1. If PSR[15]=1 (user),
   Saved.USP = R6, then R6 = Saved.SSP.
   (transfer from User Stack to Supervisor Stack)
2. Push PSR and PC to Supervisor Stack.
3. Set PSR[15] = 0 (Supervisor mode)
   PSR[10:8] = PL of interrupt being served (e.g. keyboard = PL4)
   PSR[2:0] = 0
4. Set MAR = x01vv, where vv = 8-bit interrupt vector (INVT) from interrupting device (e.g. keyboard = x80 -> MAR=x0180) →
5. Load memory, MDR = MEM[x01vv] .
6. Set PC = MDR. Now the first instruction of ISR will be fetched.

**TRAP vector table: x0000 - x00FF*

Interrupt vector table: x0100 – x01FF

Stage1.5: Prepare/Transfer

Summary

Save R6 in Saved.USP

– only when it was in user mode

Update R6 = Saved.SSP

Push old PSR and PC in Supervised Stack

Update new PSR and PC

Stage2: Service

- PC contains the starting address of the ISR.
- Callee-save for general purpose registers.
- The ISR will execute, and the requirements of the I/O device will be served.
- For example, copy KBDR into some memory location.

Stage3: Return

- RTI (Return from Interrupt) $\longleftrightarrow$ RET

$\longrightarrow$

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTI	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Pop PC from Supervisor Stack (PC = M[R6], R6 = R6 +1)
2. Pop PSR from Supervisor Stack (PSR = M[R6], R6 = R6 +1)
3. If PSR[15] = 1, Saved.SSP = R6 and then R6 = Saved.USP
   (If transferring back to user mode, save SSP and restore USP)

- RTI is a privileged instruction.
  - Can only be executed in Supervisor mode.
  - If executed in User mode, causes an exception.

Interrupt vector table

Addr	Data
x01F1	x6200
x01F2	x6300

INTV

Device B = xF1

Device C = xF2

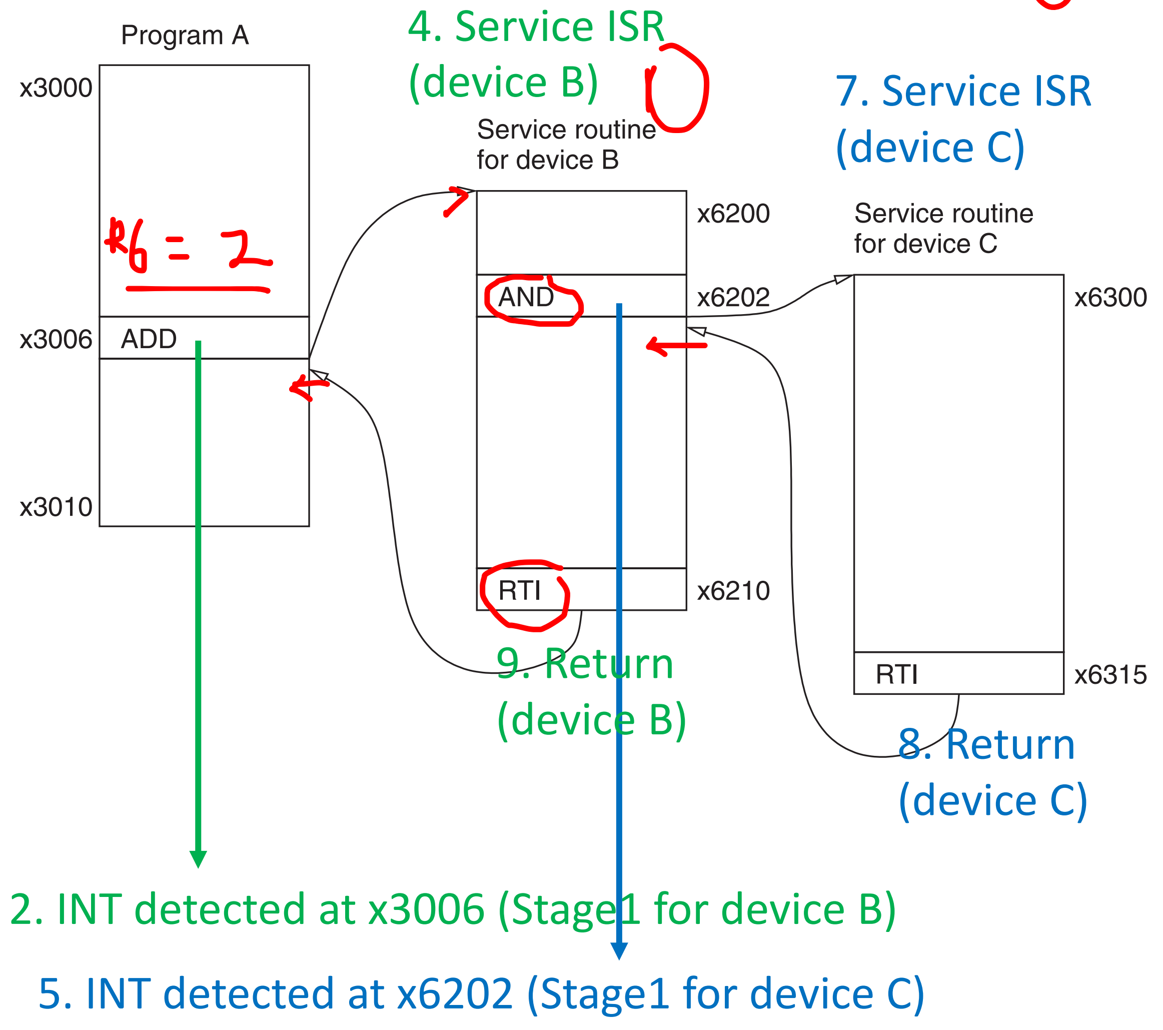
PL

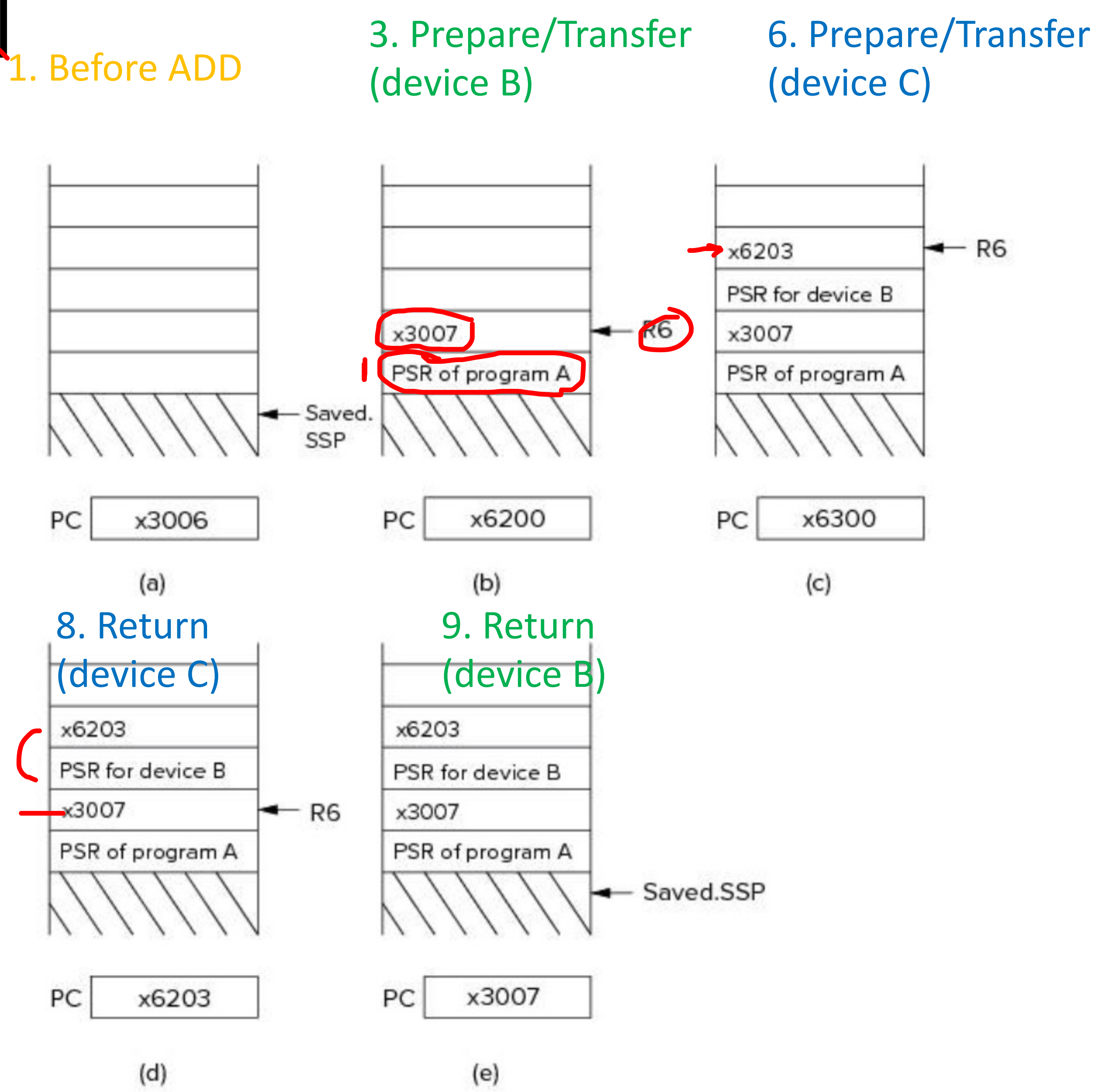
A<B<C

Saved.USP

R6 = 2

R6 = 2


Program A
x3000
R6 = 2
x3006
ADD
x3010
4. Service ISR (device B)
Service routine for device B
x6200
AND
x6202
RTI
x6210
7. Service ISR (device C)
Service routine for device C
x6300
RTI
x6315
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)



Example Interrupt Code – Lc3web

Interrupt vector table

Addr	**Data**
x01F1	x6200
x01F2	x6300
x0180	**MyISR**

INTV

Device B = xF1
Device C = xF2
keyboard = x80

https://github.com/tmoon-illinois/ece220_sp24/blob/main/lec25/interrupt_simple.asm

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.

Exceptions: Internal Interrupt

- When something unexpected happens inside the processor, it may cause an exception.

- In LC3,
    - Privilege mode violation (RTI in user mode)
    - Executing an illegal opcode (bits[15:12] = x1101)

- Other examples:
    - Divide by zero
    - Accessing an illegal memory address

- Handled just like an interrupt
    - Vector is determined internally by type of exception
    - Priority level does not change