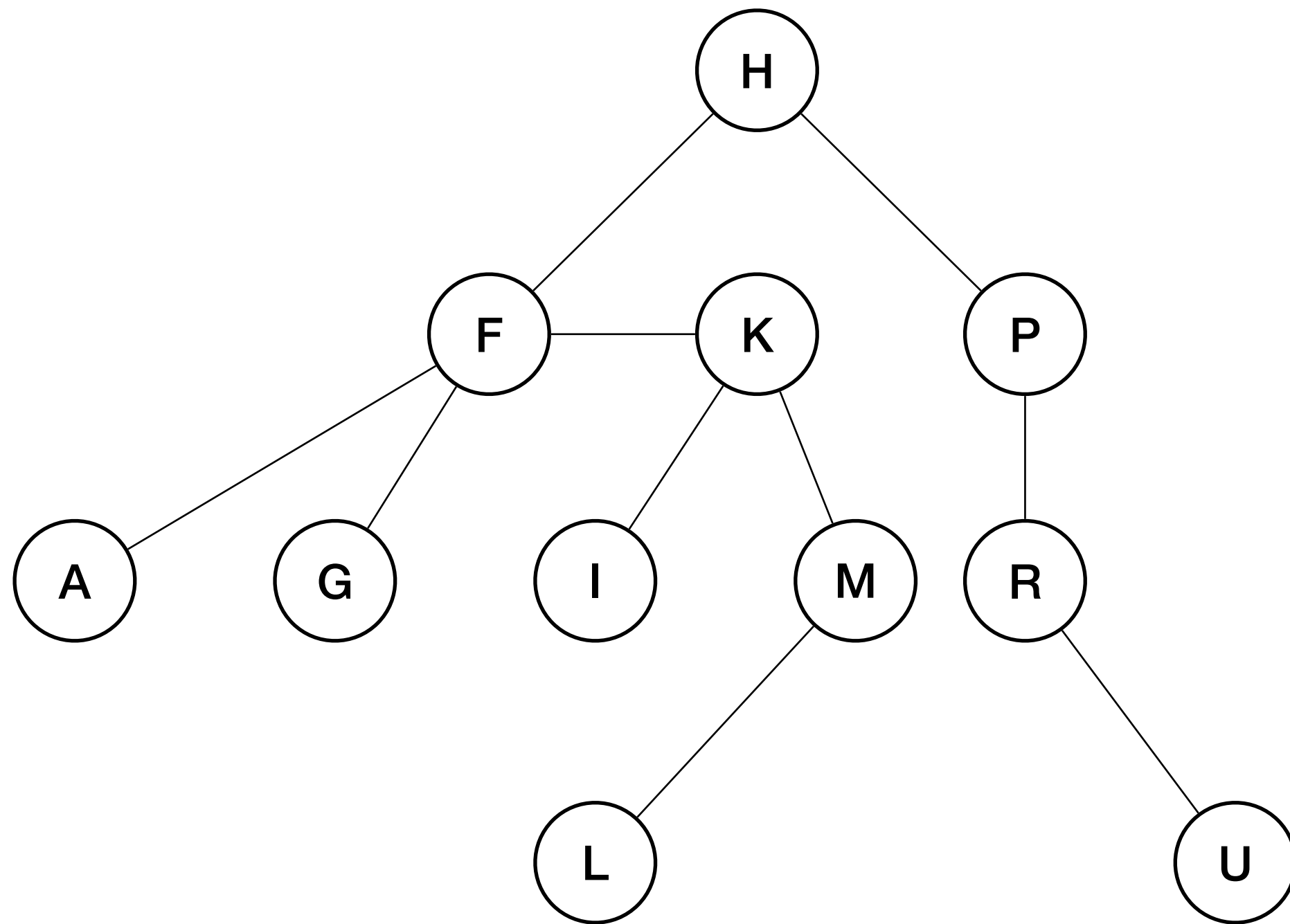


00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
1C3015C0 01010100 30011100 00002020 20202E4F 52494720 20207833 3030300A E0001300 00002020 20204C45 41202052
302C206D 794C696E 6509E200 13000000 20202020 4C454120 2052312C 206D794C 696E6540 60001600 00004C4F 4F502020
20204C44 52205230 2C205231 2C202330 21F00010 00000020 20202020 20202054 52415020 78323105 24001400 00002020
20202020 20204C44 20205232 2C207465 726D8014 00160000 00202020 20202020 20414444 2052322C 2052322C 20523002
04001000 00002020 20202020 20204252 7A205B54 F50612 00150000 0202020 20202020 20414444 2052312C 2052312C
2031F90F 00120000 00202020 20202020 20425B54 F50612 00150000 0202020 00005354 4F502020 20204841 4C54D0FF
00150000 00746572 6D202020 202E4649 4C4C2020 20784646 44306900 00010000 00697400 00010000 00746100 00010000
00616200 00010000 00627200 00010000 00725200 00510000 00515800 00010000 00683200 00010000 00324000 00010000
00406600 00010000 00666100 00010000 00403320 00010000 00332D00 00010000 002D6500 00010000
00656300 00010000 00636500 00010000 00653200 00010000 00323200 00010000 00323000 00010000 00300000 002A0000
006D794C 696E6520 202E5354 52494E47 5A202020 20226974 61627261 68324066 6132332D 65636532 32302200 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

ECE 220

Lecture x0019 - 12/03
Recap & Interrupts in LC3

Post-break recall



Recap + reminders

Recap + reminders

- Trees

Recap + reminders

- Trees
 - Concepts & properties

Recap + reminders

- Trees
 - Concepts & properties
 - Traversals: preorder/
inorder/postorder

Recap + reminders

- Trees
 - Concepts & properties
 - Traversals: preorder/
inorder/postorder
 - Binary search trees

Recap + reminders

- Trees
 - Concepts & properties
 - Traversals: preorder/
inorder/postorder
 - Binary search trees
 - C to LC3 (structs, trees,
linked lists)

Recap + reminders

- Trees
 - Concepts & properties
 - Traversals: preorder/
inorder/postorder
 - Binary search trees
 - C to LC3 (structs, trees,
linked lists)
 - Code posted

Recap + reminders

- Trees
 - Concepts & properties
 - Traversals: preorder/inorder/postorder
 - Binary search trees
 - C to LC3 (structs, trees, linked lists)
 - Code posted
- Reminders

Recap + reminders

- Trees
 - Concepts & properties
 - Traversals: preorder/inorder/postorder
 - Binary search trees
 - C to LC3 (structs, trees, linked lists)
 - Code posted
- Reminders
 - Extra credit quiz

Recap + reminders

- Trees
 - Concepts & properties
 - Traversals: preorder/inorder/postorder
 - Binary search trees
 - C to LC3 (structs, trees, linked lists)
 - Code posted
- Reminders
 - Extra credit quiz
 - Programming competition

Recap + reminders

- Trees
 - Concepts & properties
 - Traversals: preorder/inorder/postorder
 - Binary search trees
 - C to LC3 (structs, trees, linked lists)
 - Code posted
- Reminders
 - Extra credit quiz
 - Programming competition
 - ICES forms available

Today

Today

- Change gears & talk about *interrupts* in LC3

Today

- Change gears & talk about *interrupts* in LC3
- Recall: KBSR/KBDR & DSR/DDR?

Today

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

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?

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 KBSR into R0
;Other instructions
; --
; --
KBSR .FILL xFE00
KBDR .FILL xFE02
.END
```

```
.ORIG x1000
KFPOLL LDI R1, KBSR
BZP KFPOLL
LDI R0, KBDR
; --
; --
RALT
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

Previously - I/O using Polling or TRAPs

Previously - I/O using Polling or TRAPs

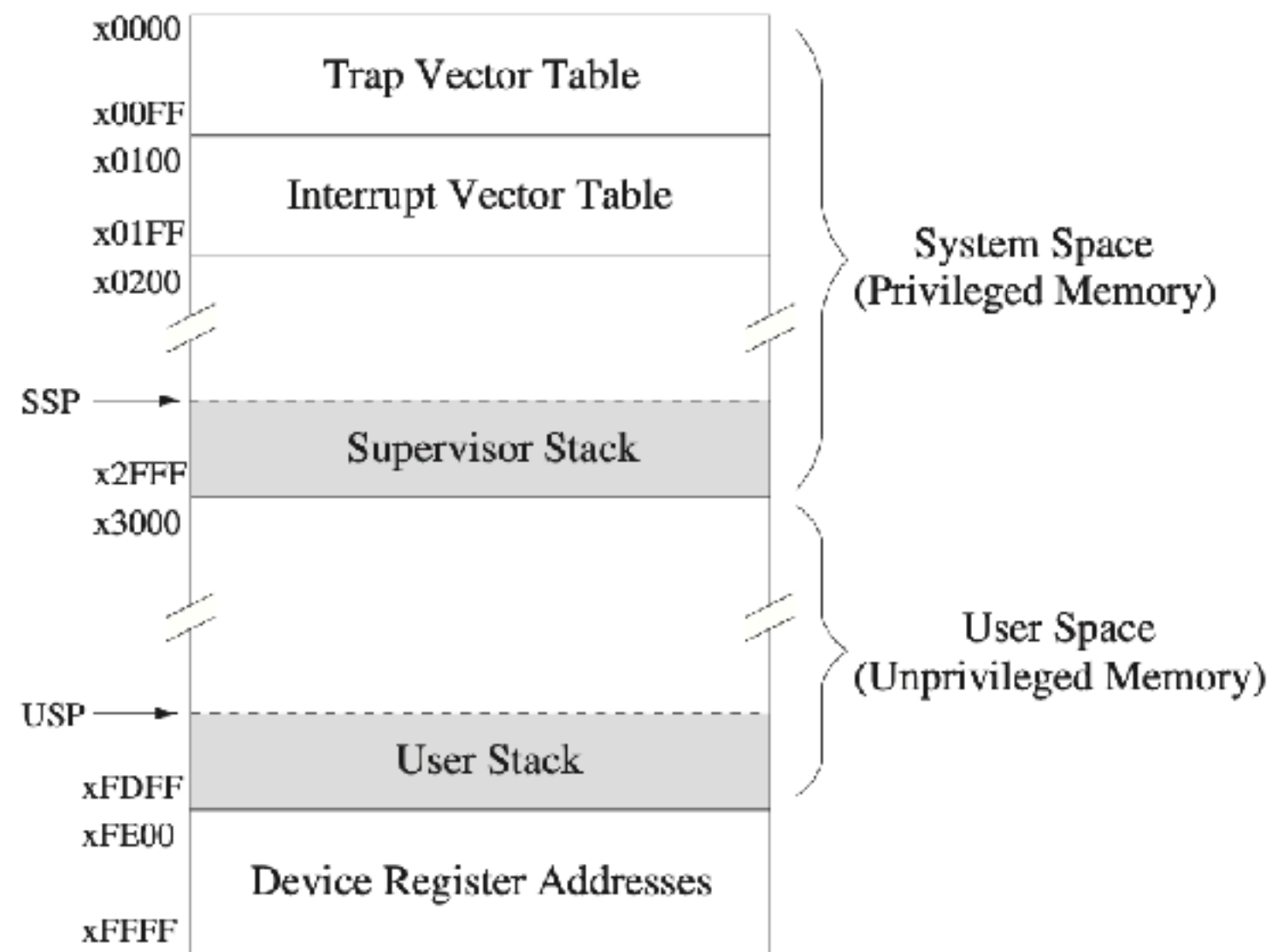


Figure A.1 - P&P 3rd Ed.

Previously - I/O using Polling or TRAPs

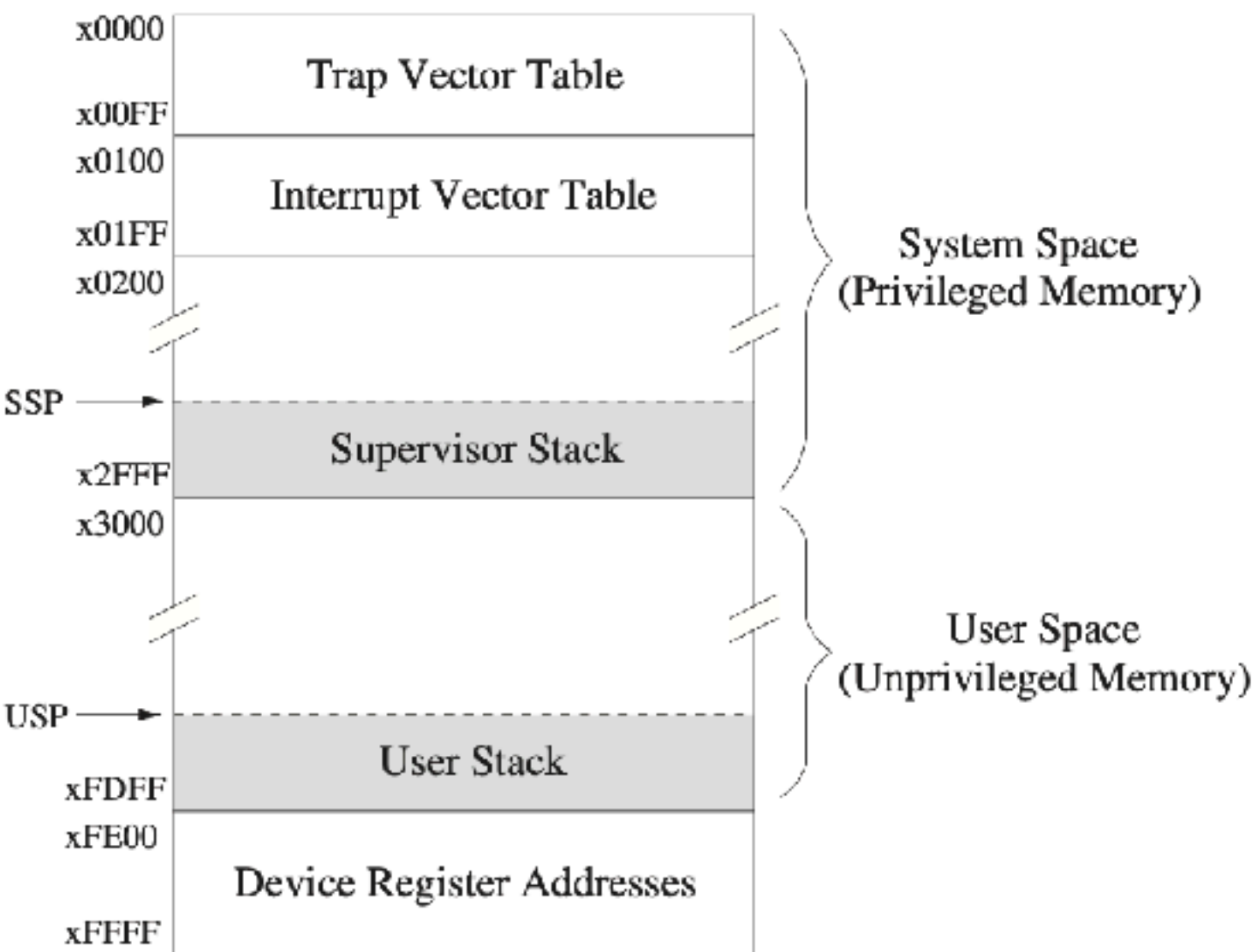


Figure A.1 - P&P 3rd Ed.

Address	I/O Register Name	I/O Register Function
---------	-------------------	-----------------------

Previously - I/O using Polling or TRAPs

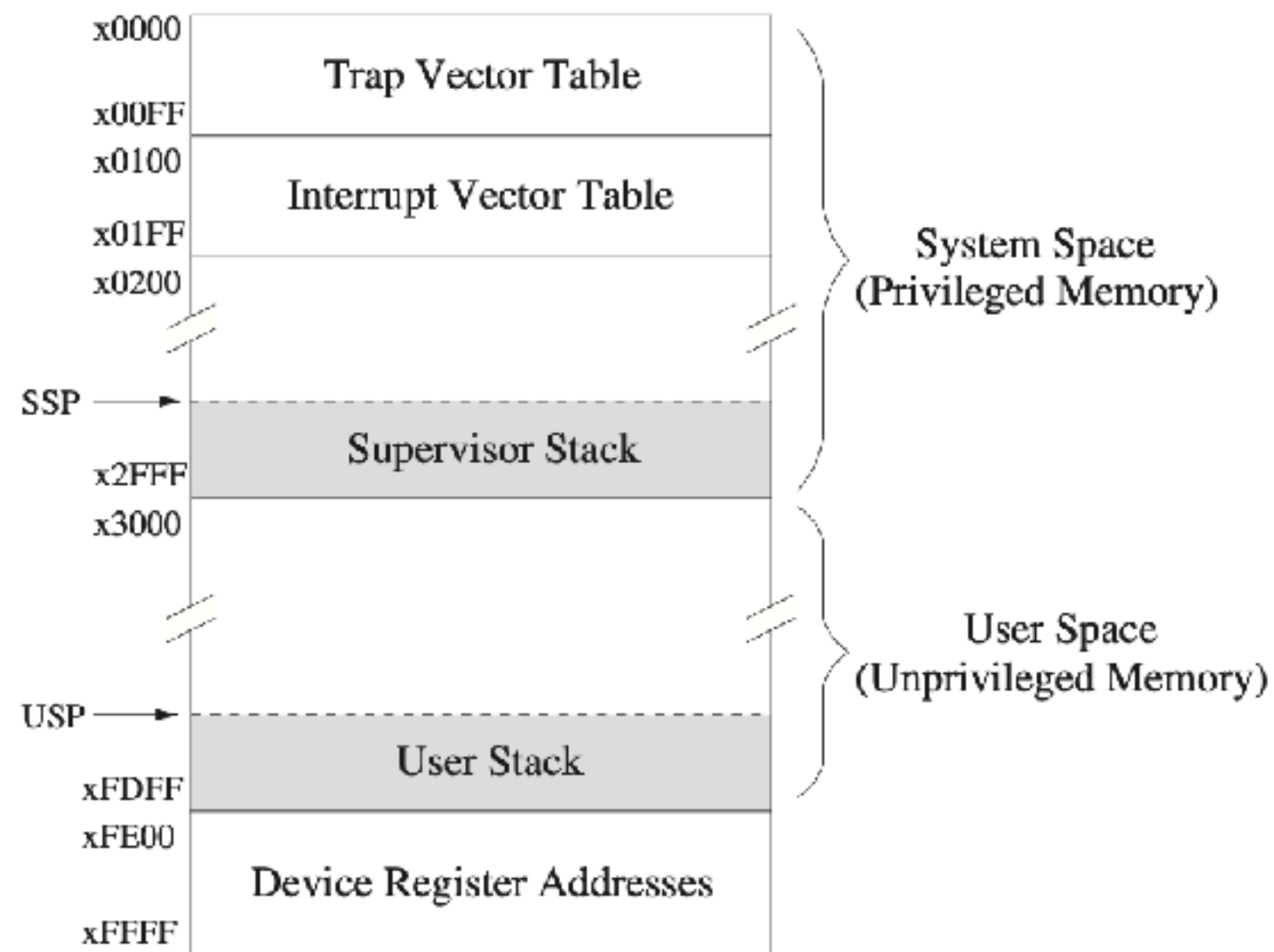


Figure A.1 - P&P 3rd Ed.

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

Previously - I/O using Polling or TRAPs

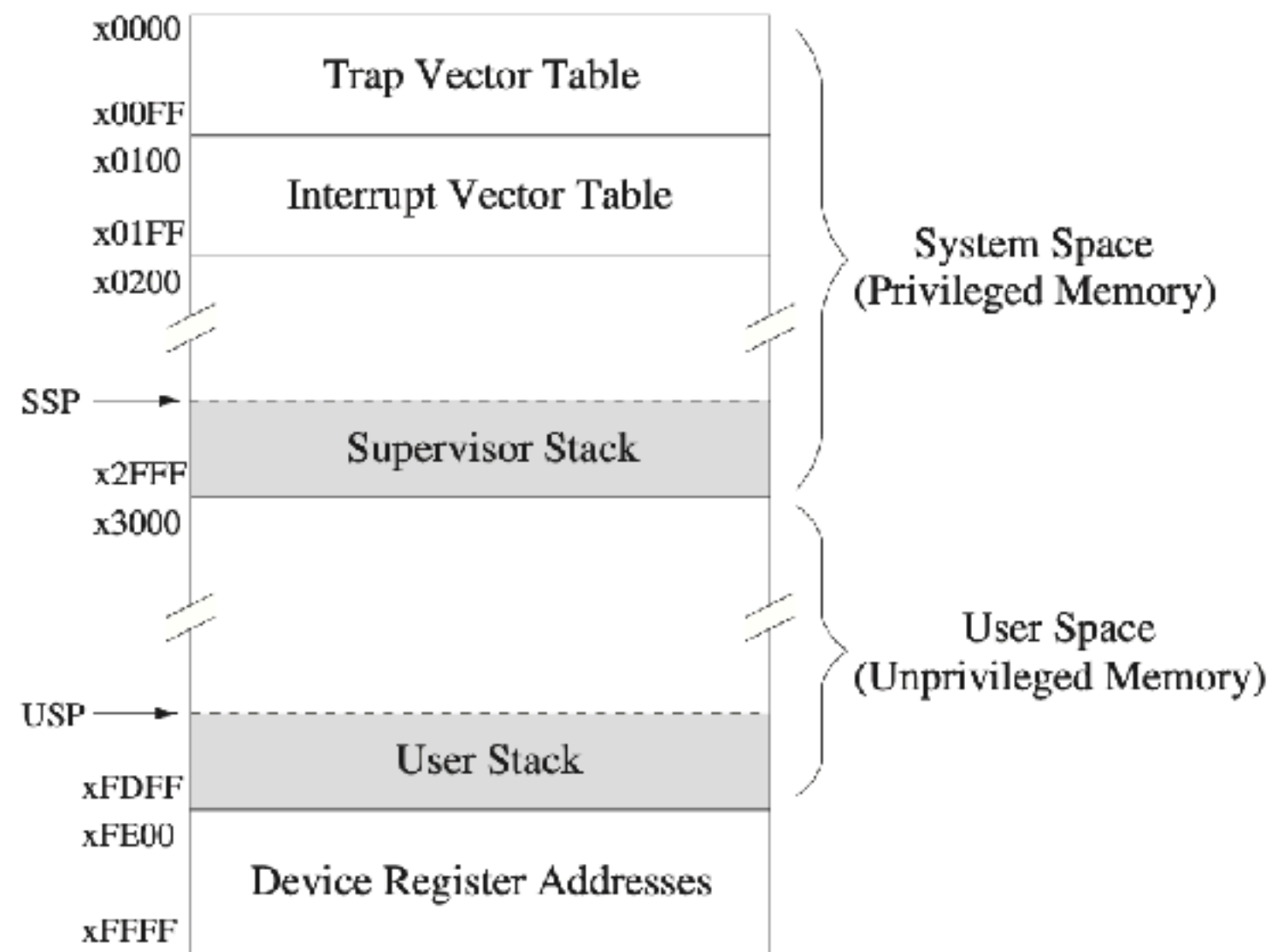


Figure A.1 - P&P 3rd Ed.

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

Previously - I/O using Polling or TRAPs

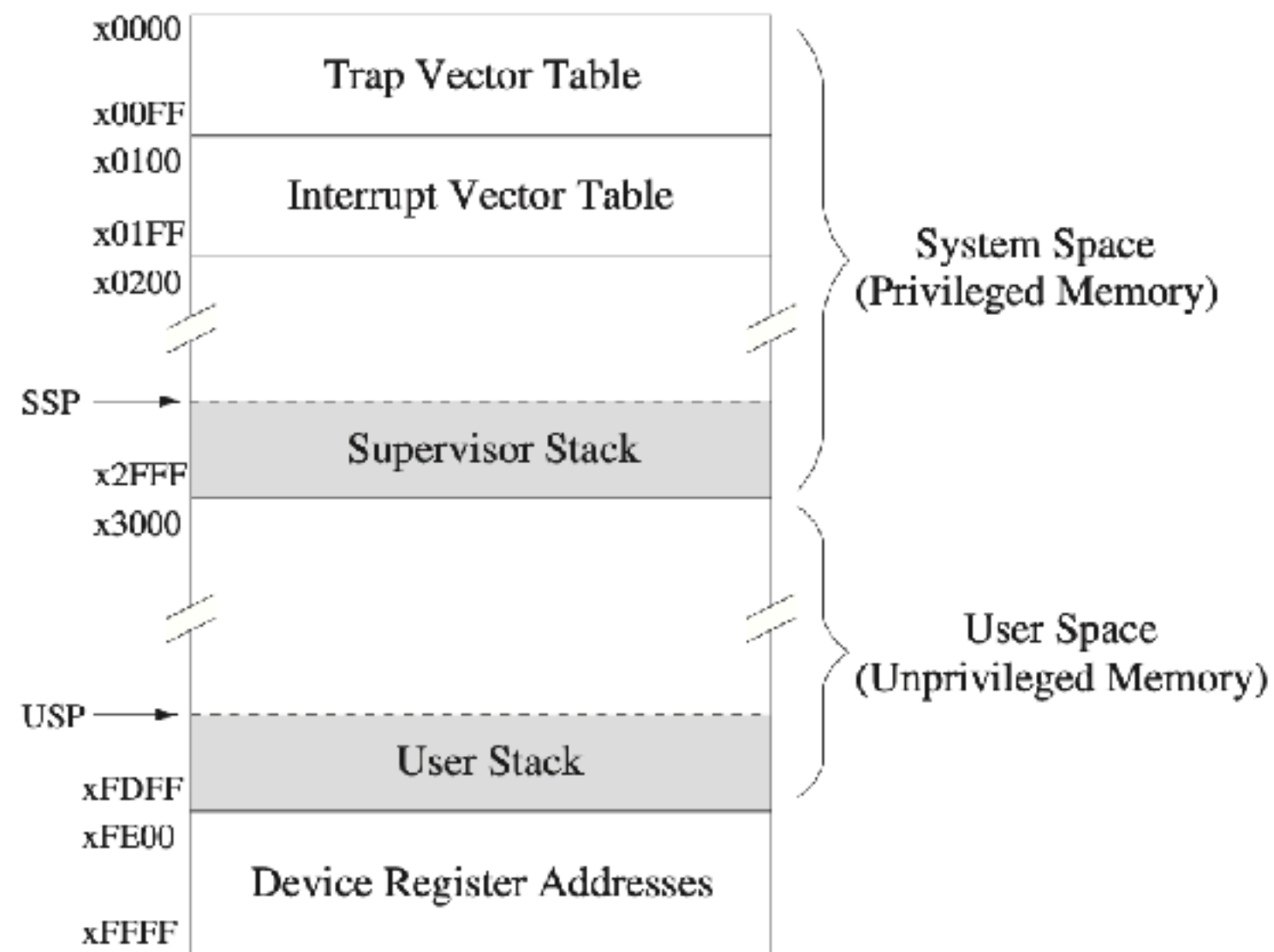
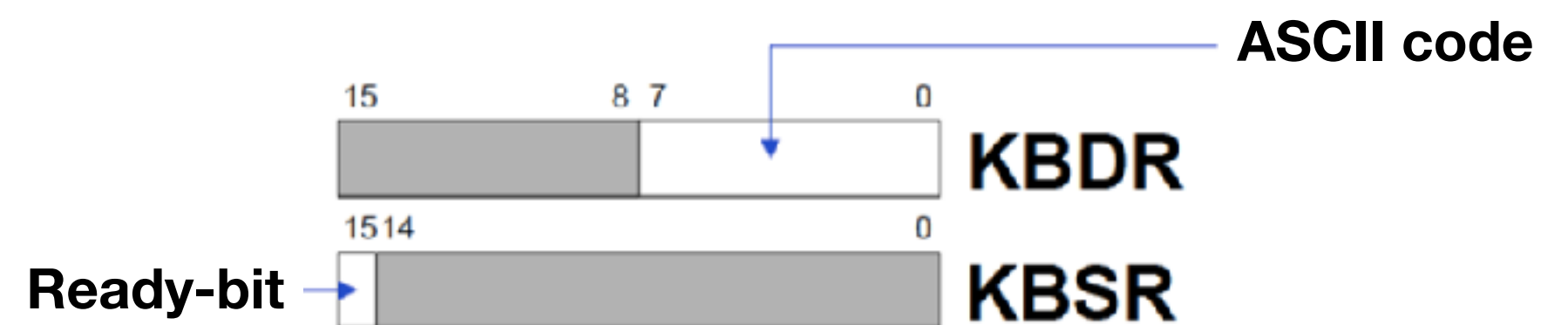


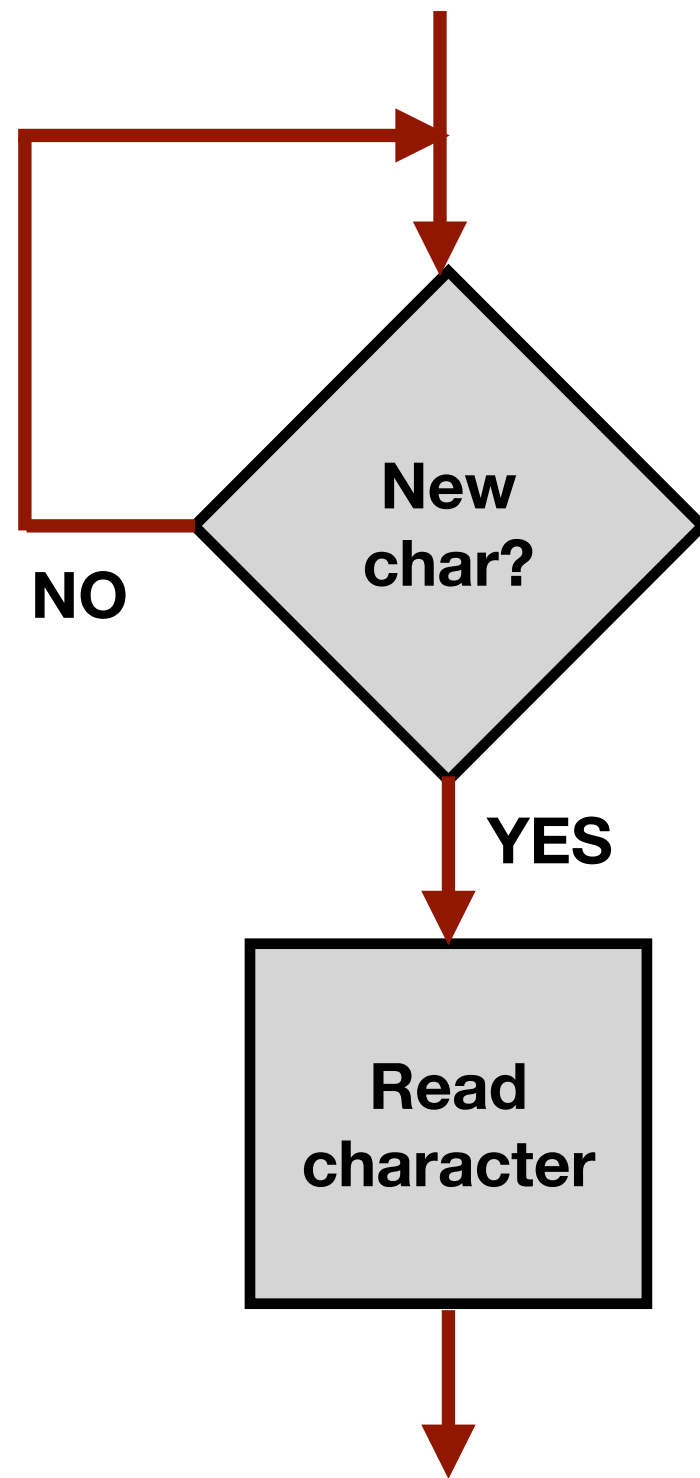
Figure A.1 - P&P 3rd Ed.

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



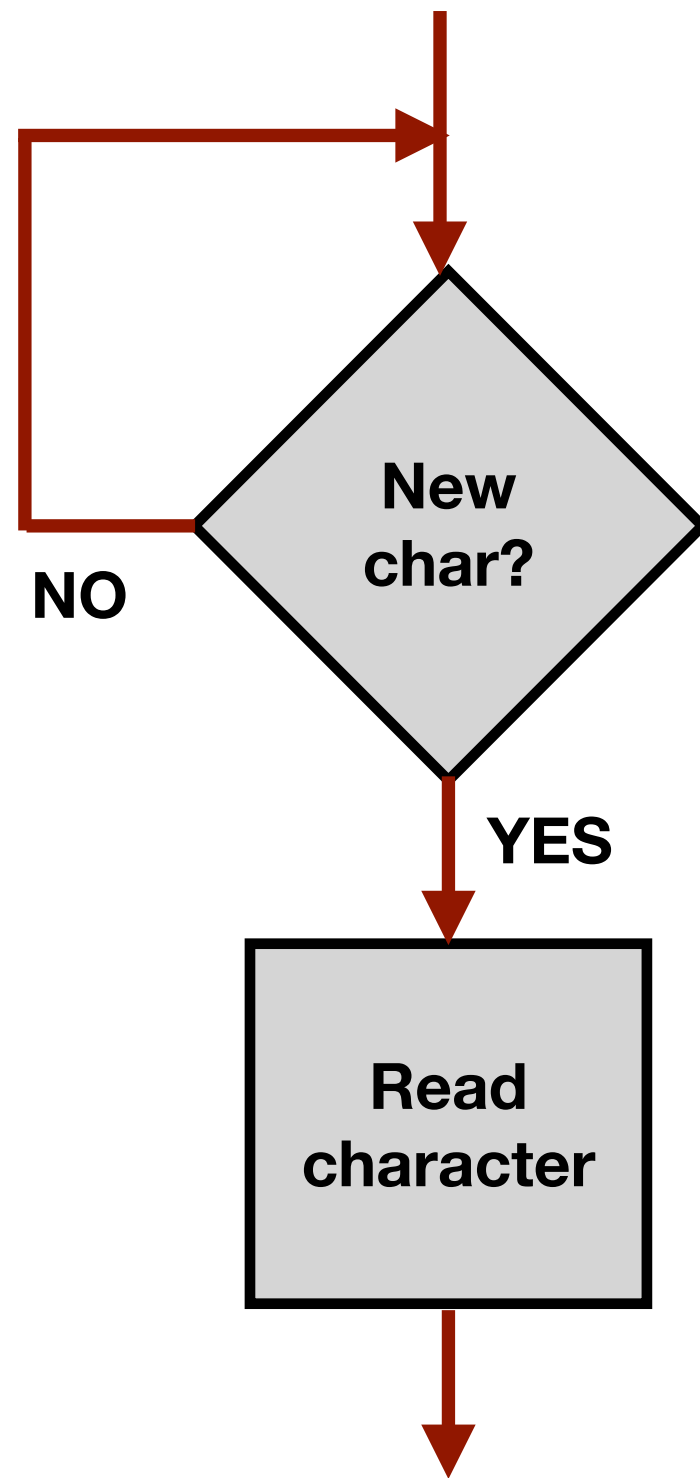
Any problems with polled I/O?

Any problems with polled I/O?

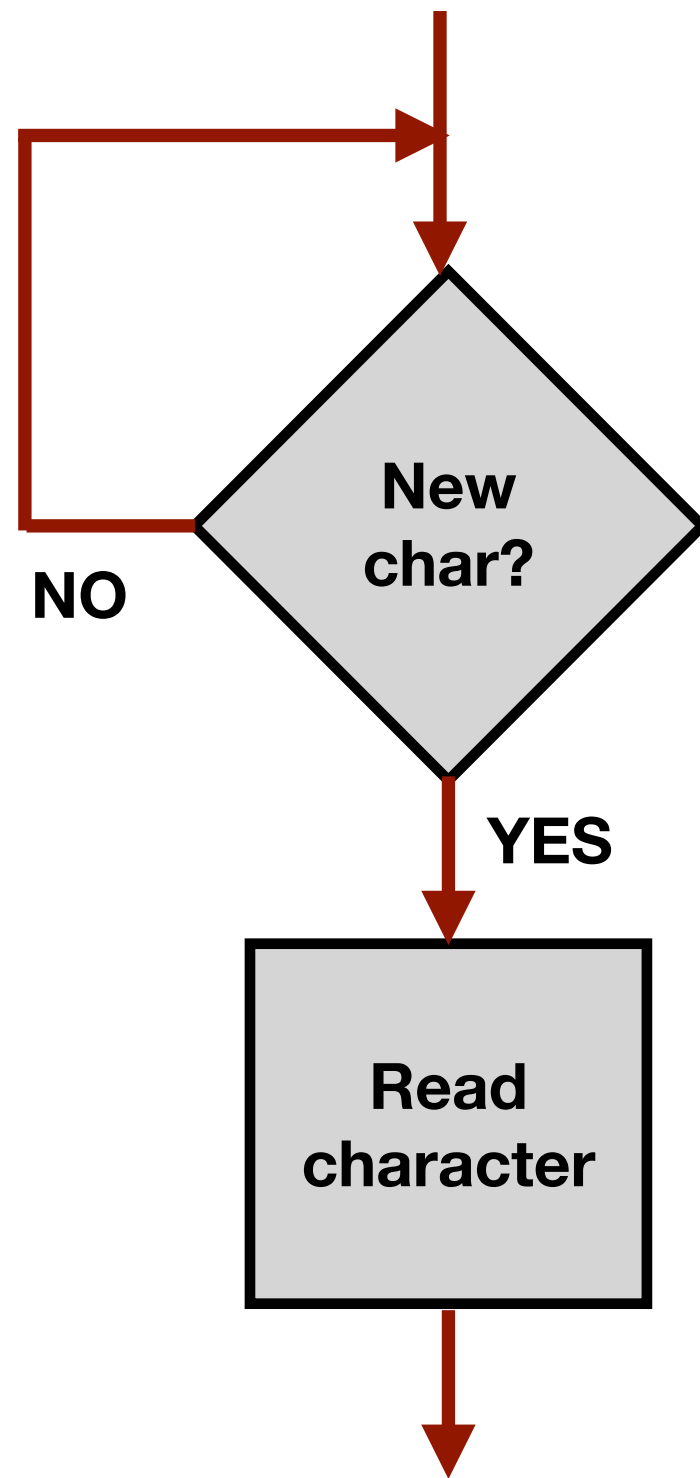


Any problems with polled I/O?

- Suppose we want to type 100 characters:

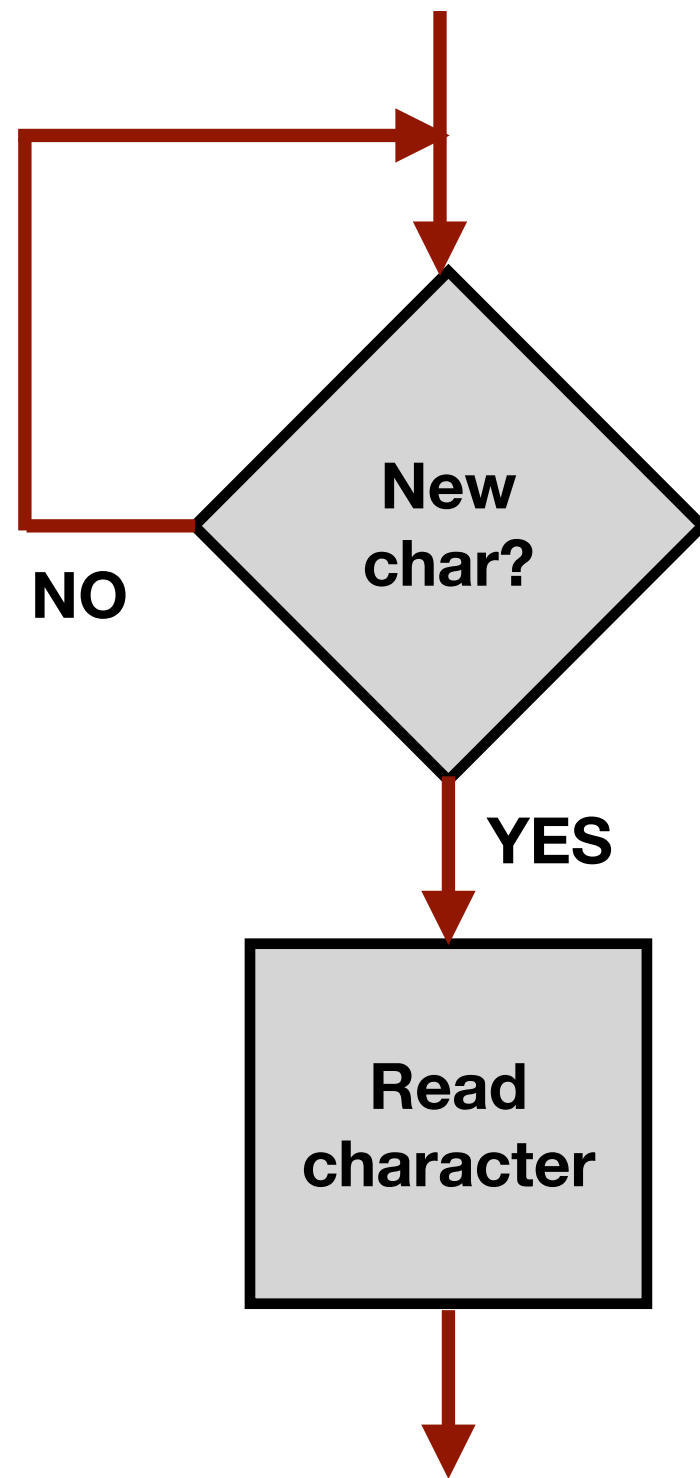


Any problems with polled I/O?



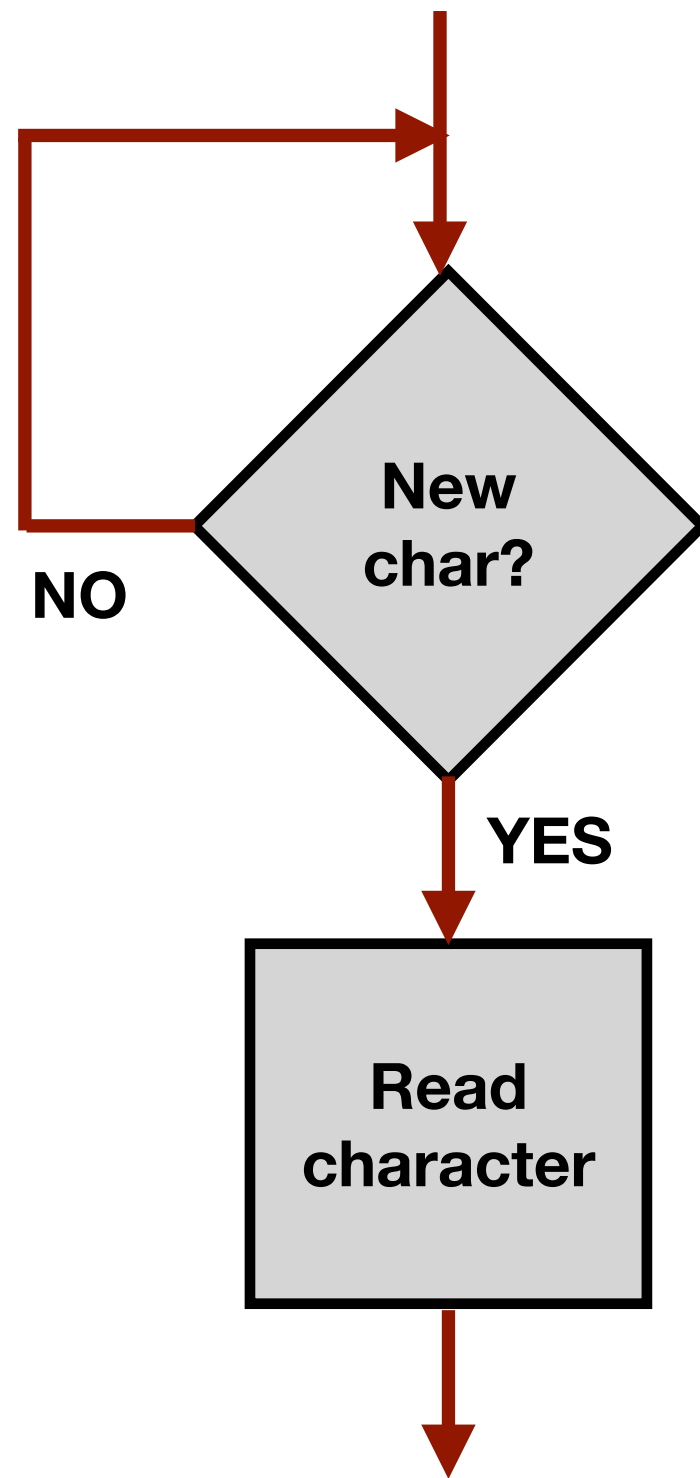
- Suppose we want to type 100 characters:
 - User can input at speed of 60 WPM, assume, eight characters/word:

Any problems with polled I/O?



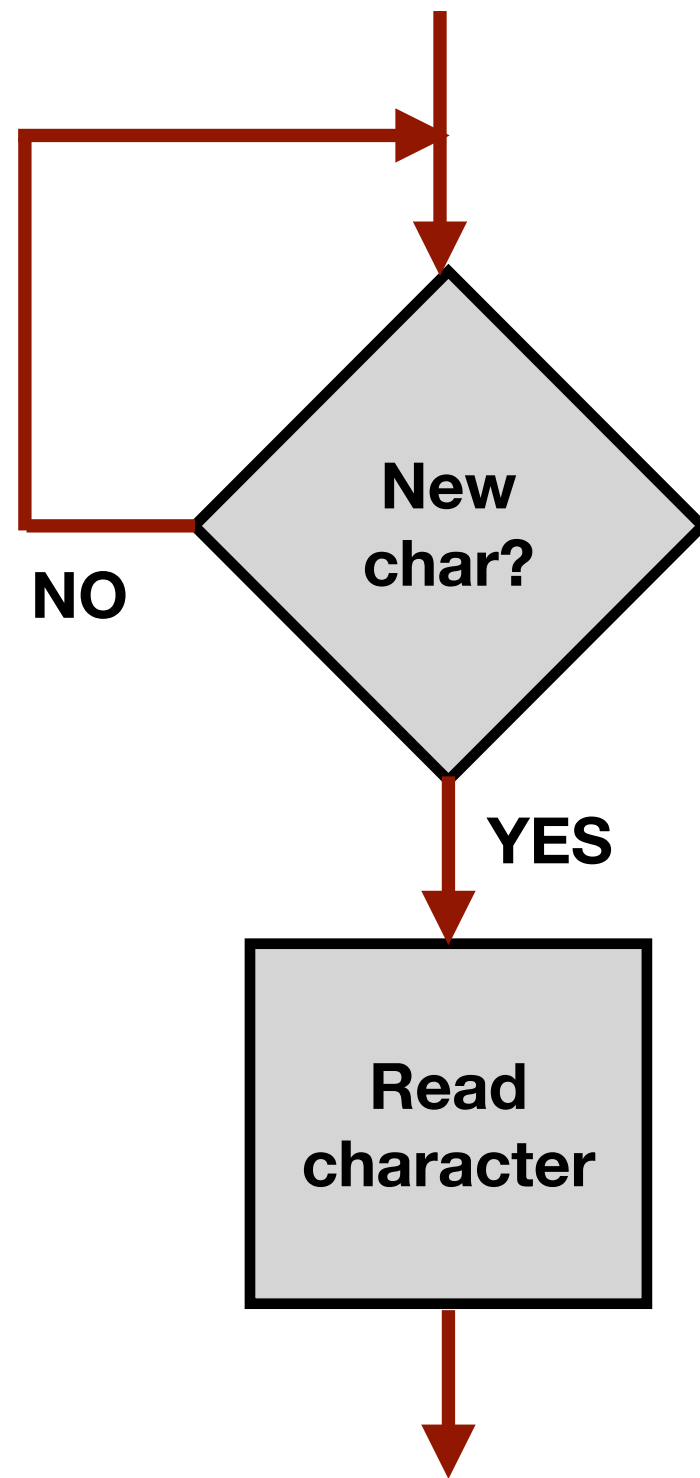
- Suppose we want to type 100 characters:
 - User can input at speed of 60 WPM, assume, eight characters/word:
 - How long to type 100 characters?

Any problems with polled I/O?



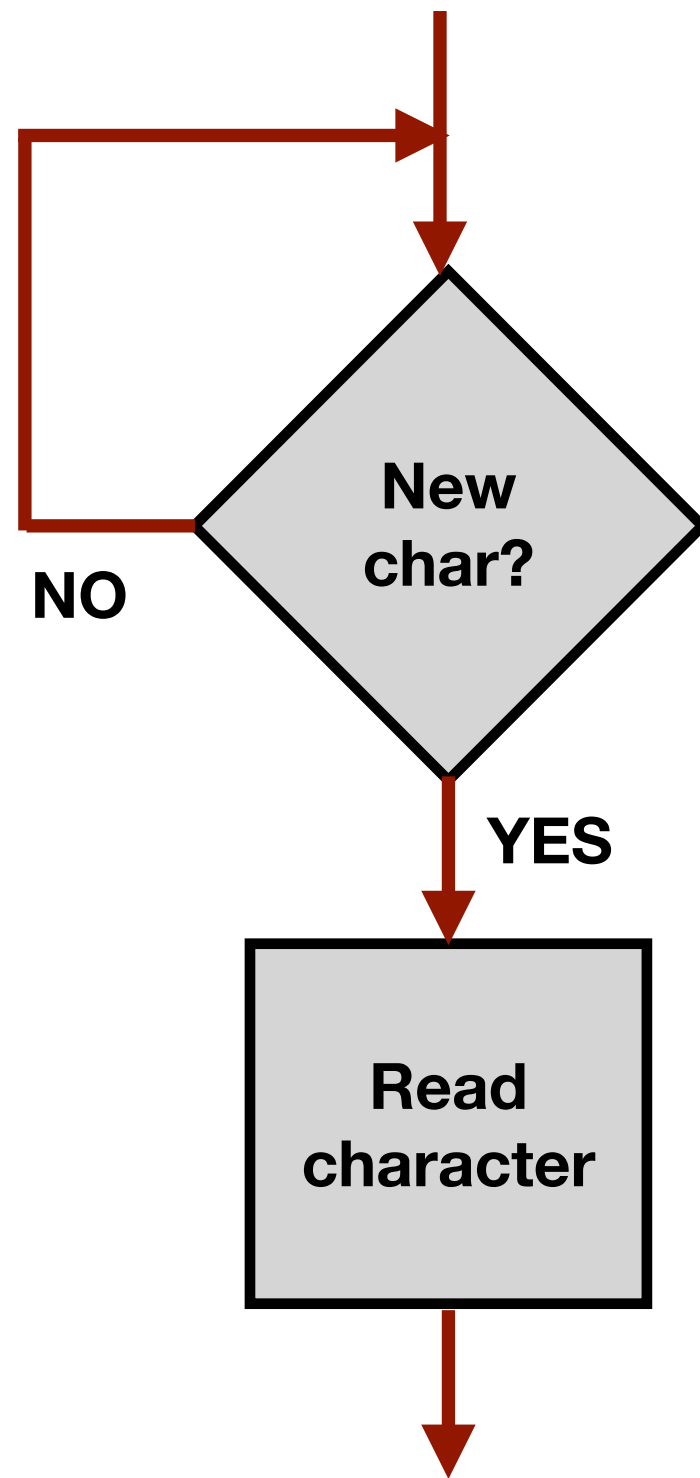
- Suppose we want to type 100 characters:
 - User can input at speed of 60 WPM, assume, eight characters/word:
 - How long to type 100 characters?
- Processor cannot do anything else while it waits on *next* character!

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!

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

Interrupt driven I/O

- Key idea: An I/O device can

Interrupt driven I/O

- Key idea: An I/O device can
 - force running program to stop

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*

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

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

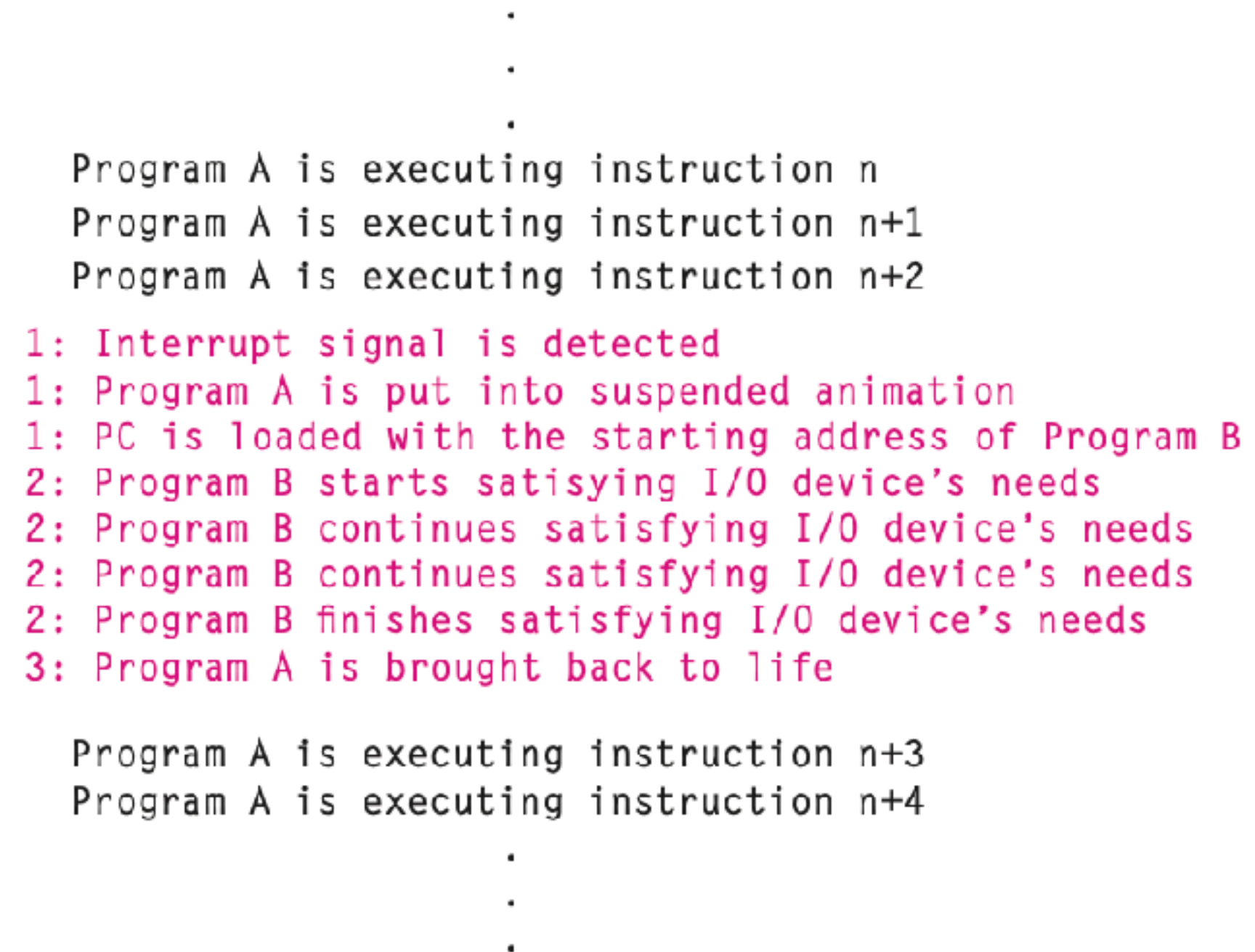


Figure 9.17 Instruction execution flow for interrupt-driven I/O.

Now - I/O using interrupts

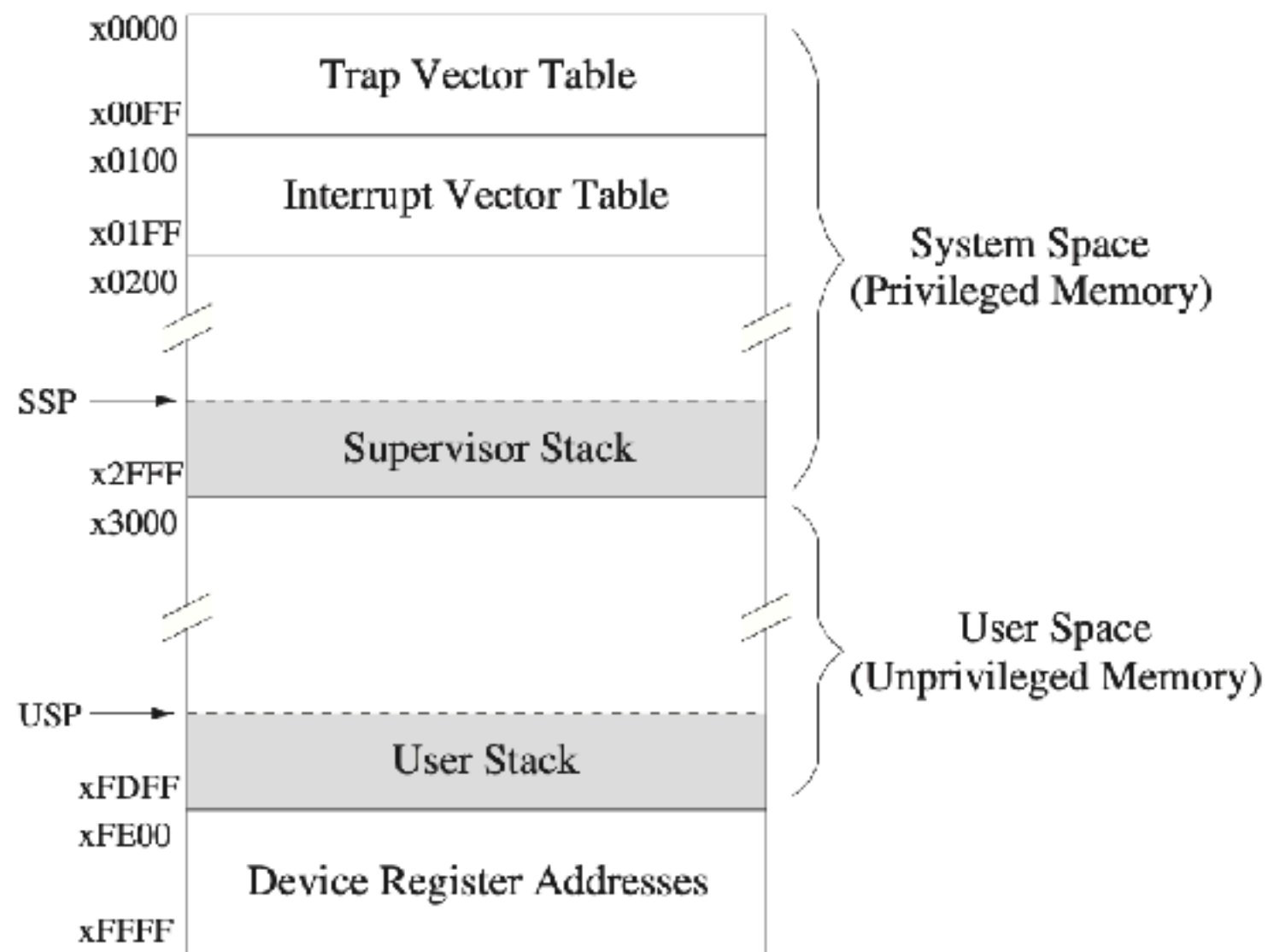
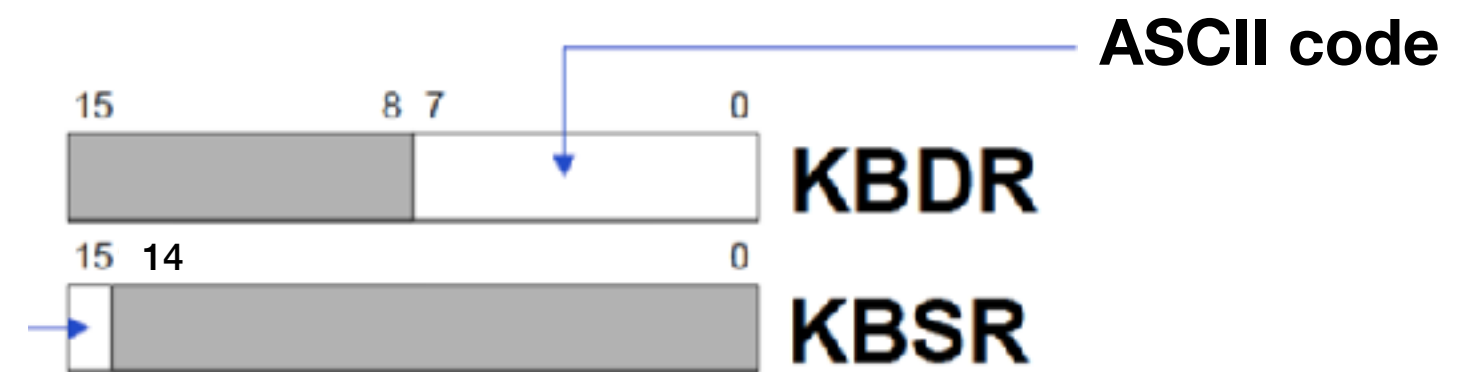


Figure A.1 - P&P 3rd Ed.



Now - I/O using interrupts

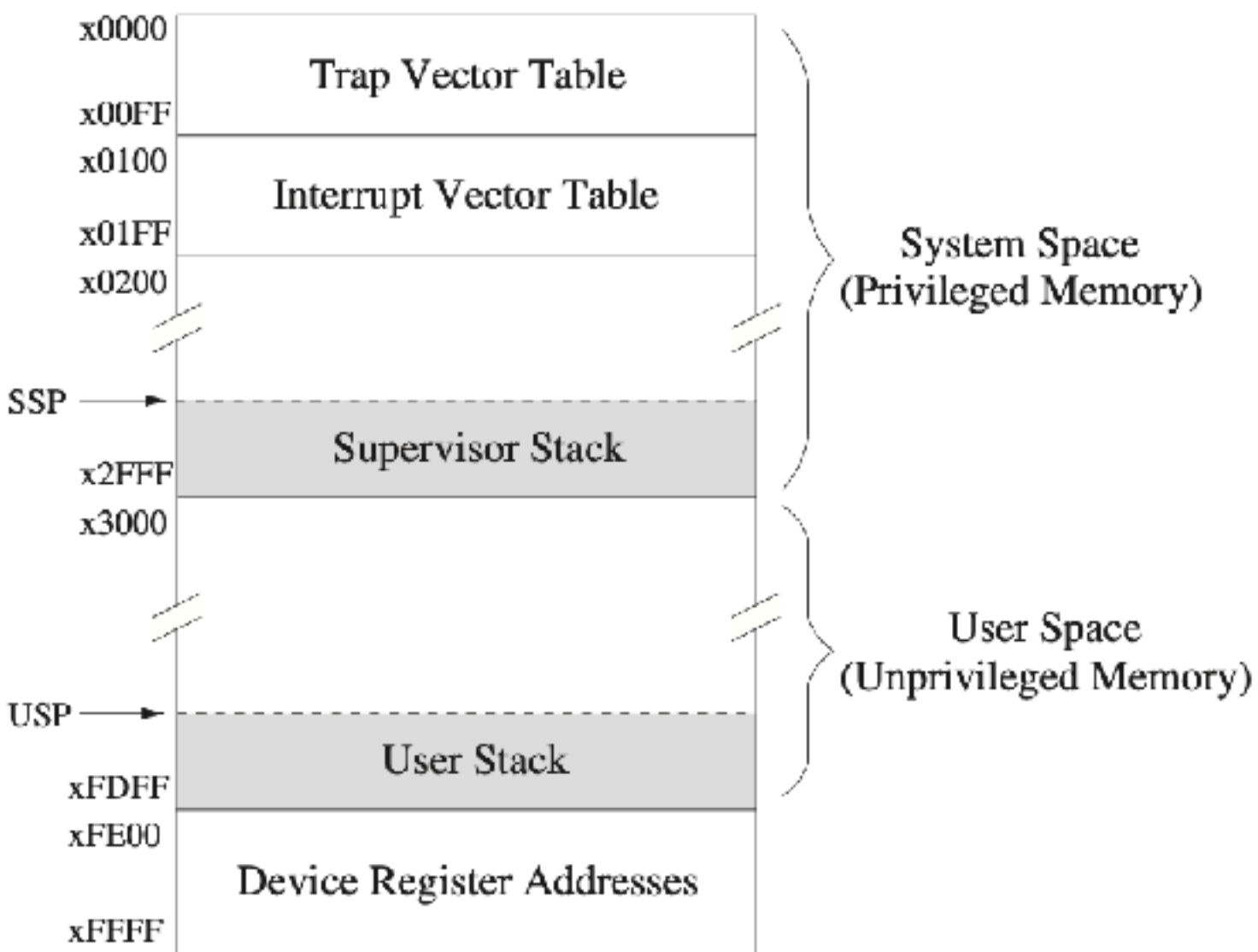
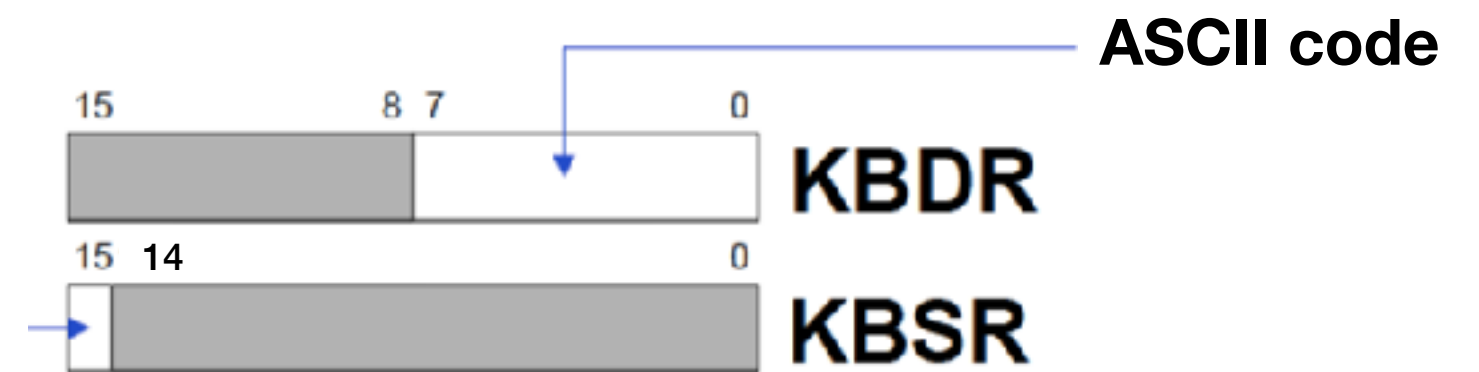


Figure A.1 - P&P 3rd Ed.

Address	I/O Register Name	I/O Register Function
---------	-------------------	-----------------------



Now - I/O using interrupts

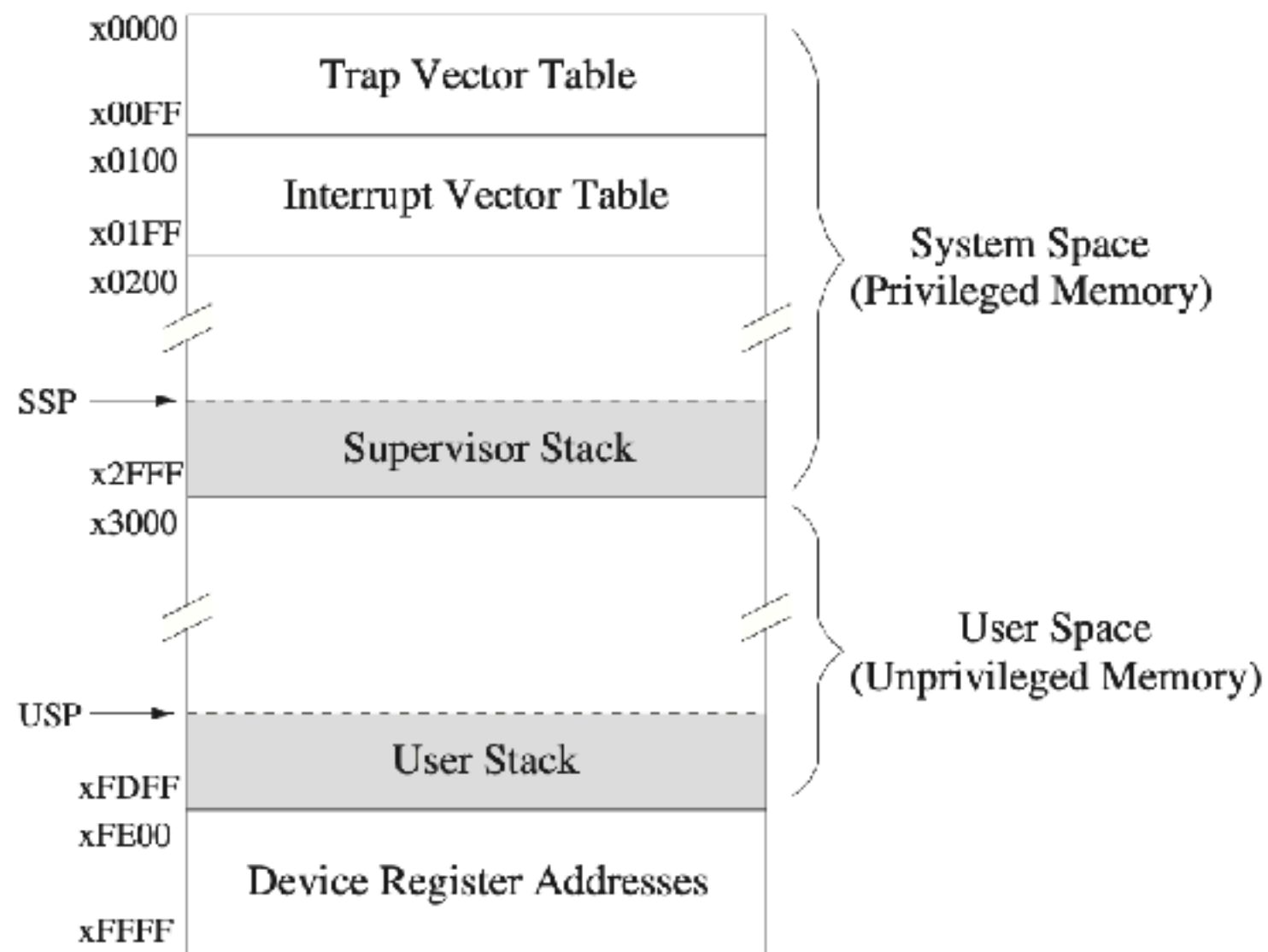
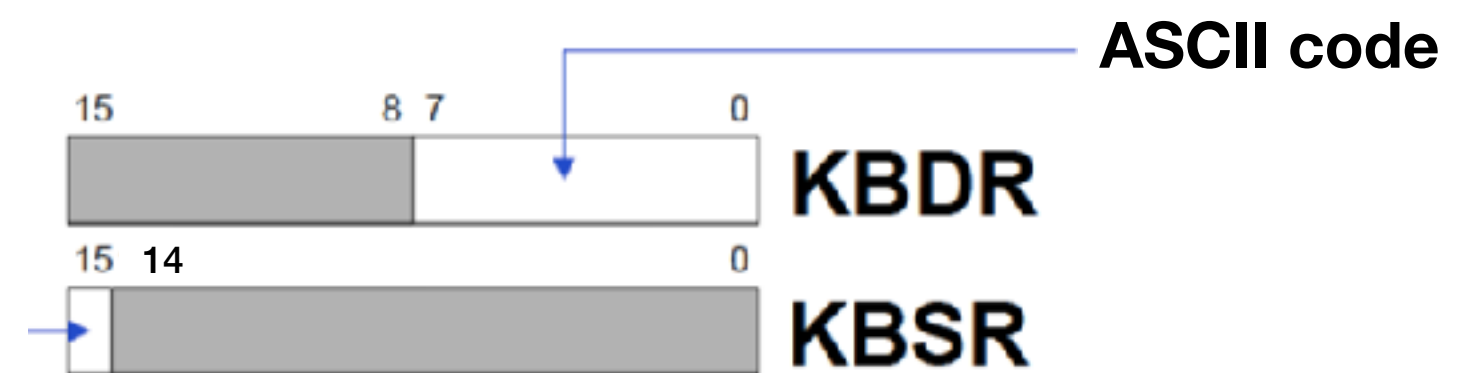


Figure A.1 - P&P 3rd Ed.

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



Now - I/O using interrupts

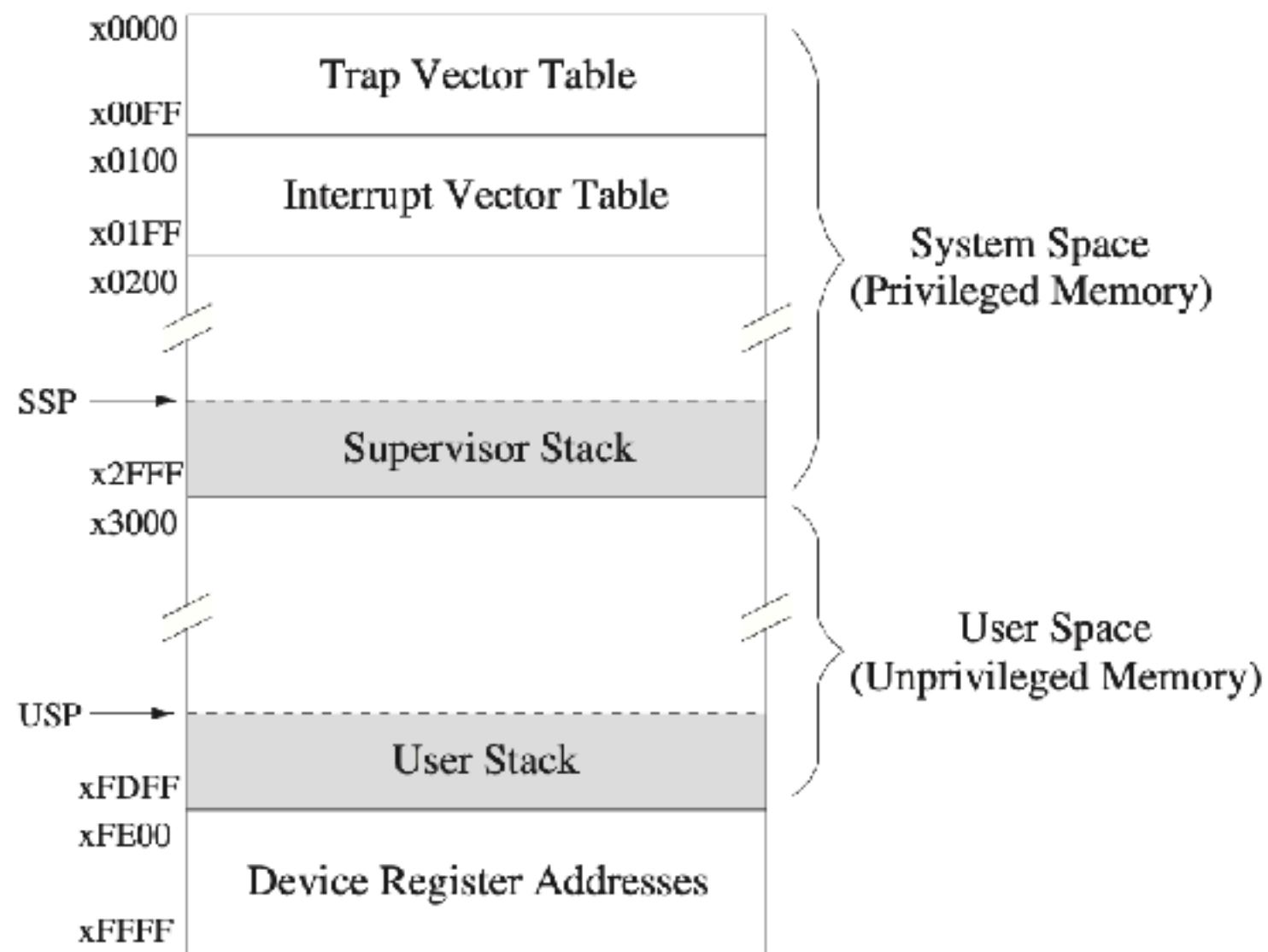
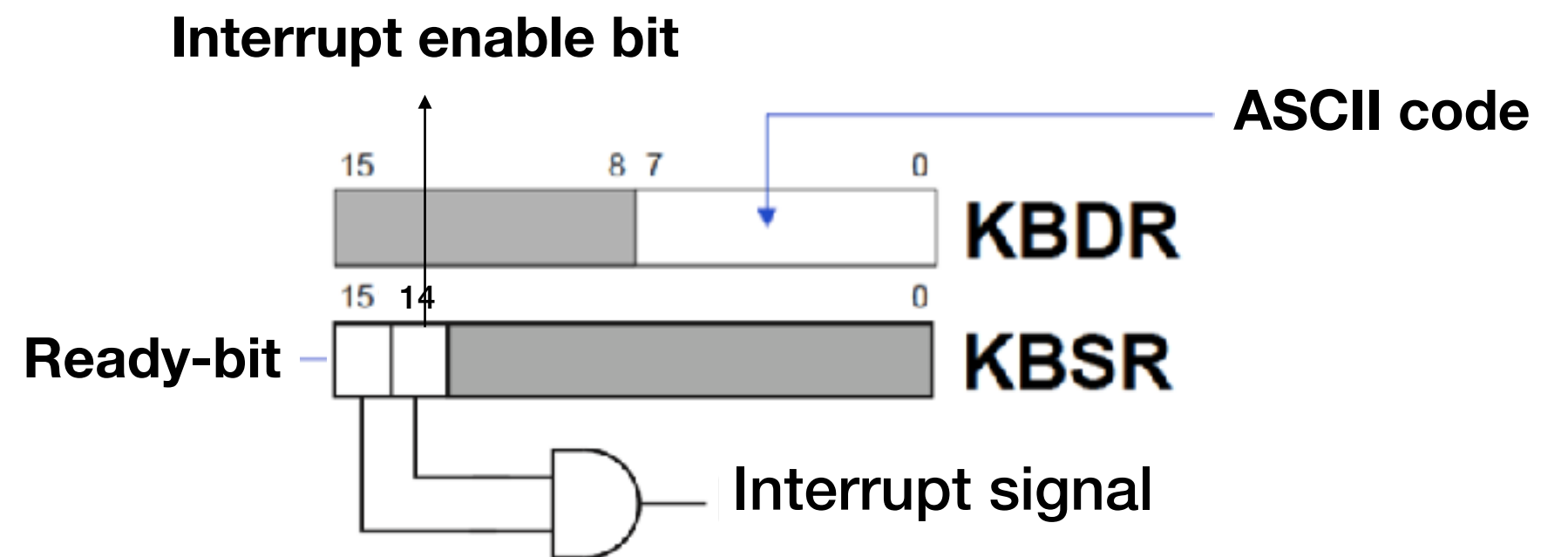


Figure A.1 - P&P 3rd Ed.

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



Now - I/O using interrupts

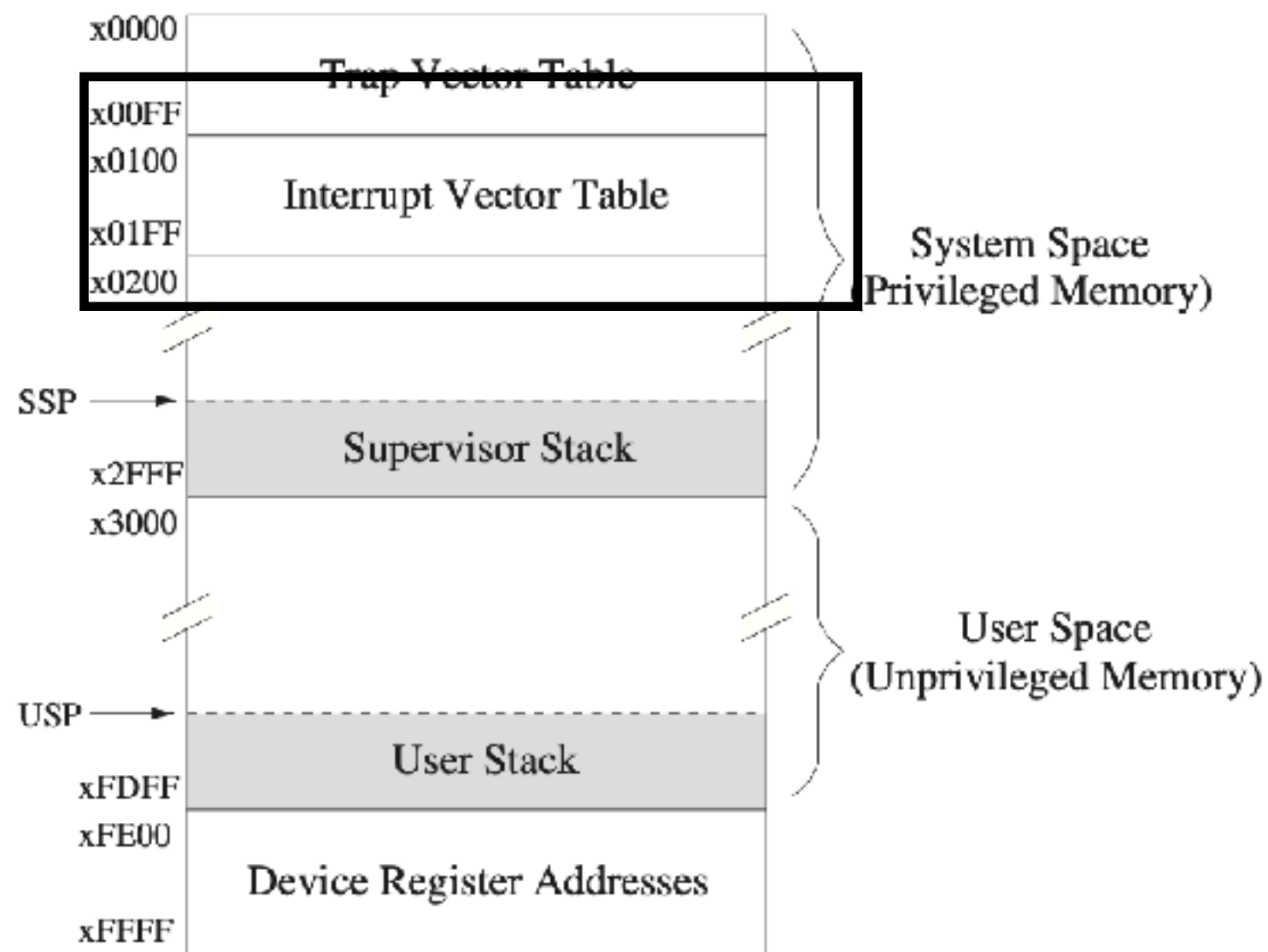
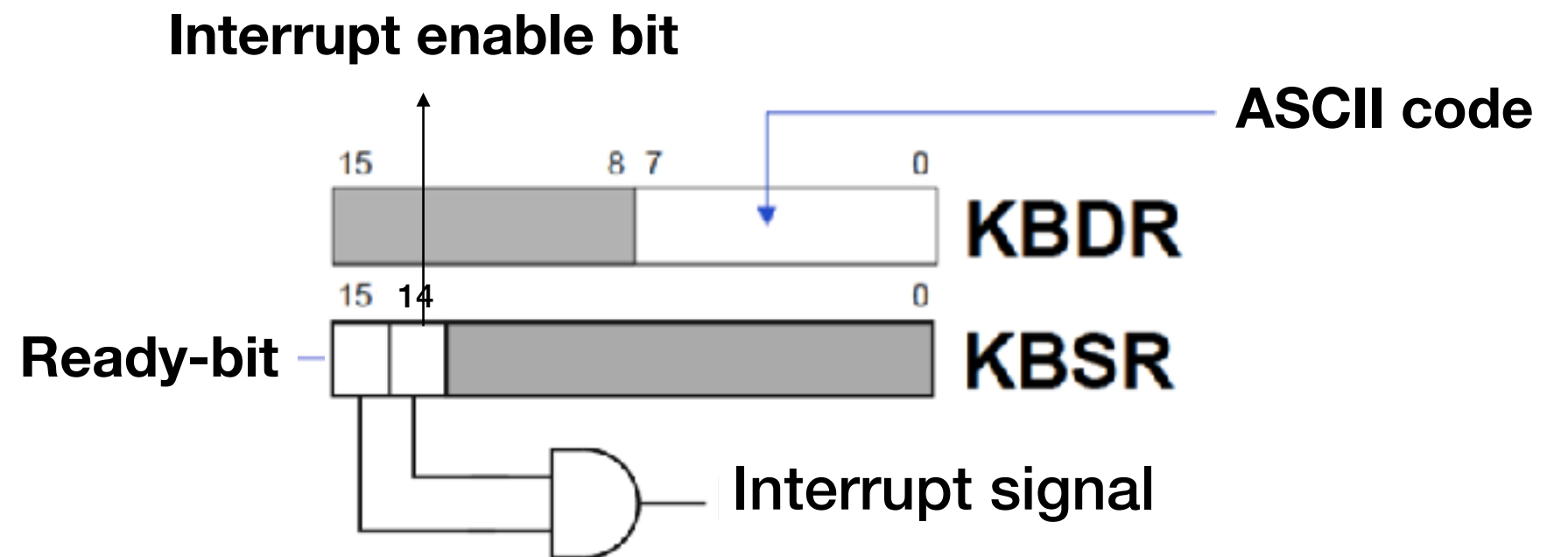


Figure A.1 - P&P 3rd Ed.

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?

What needs to happen?

For interrupt driven I/O to work:

What needs to happen?

For interrupt driven I/O to work:

1. The I/O device **must want service.**

What needs to happen?

For interrupt driven I/O to work:

1. The I/O device **must want service.**
 - Ready bit

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.

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

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.

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:** $PL0 < PL1 < \dots < PL6$

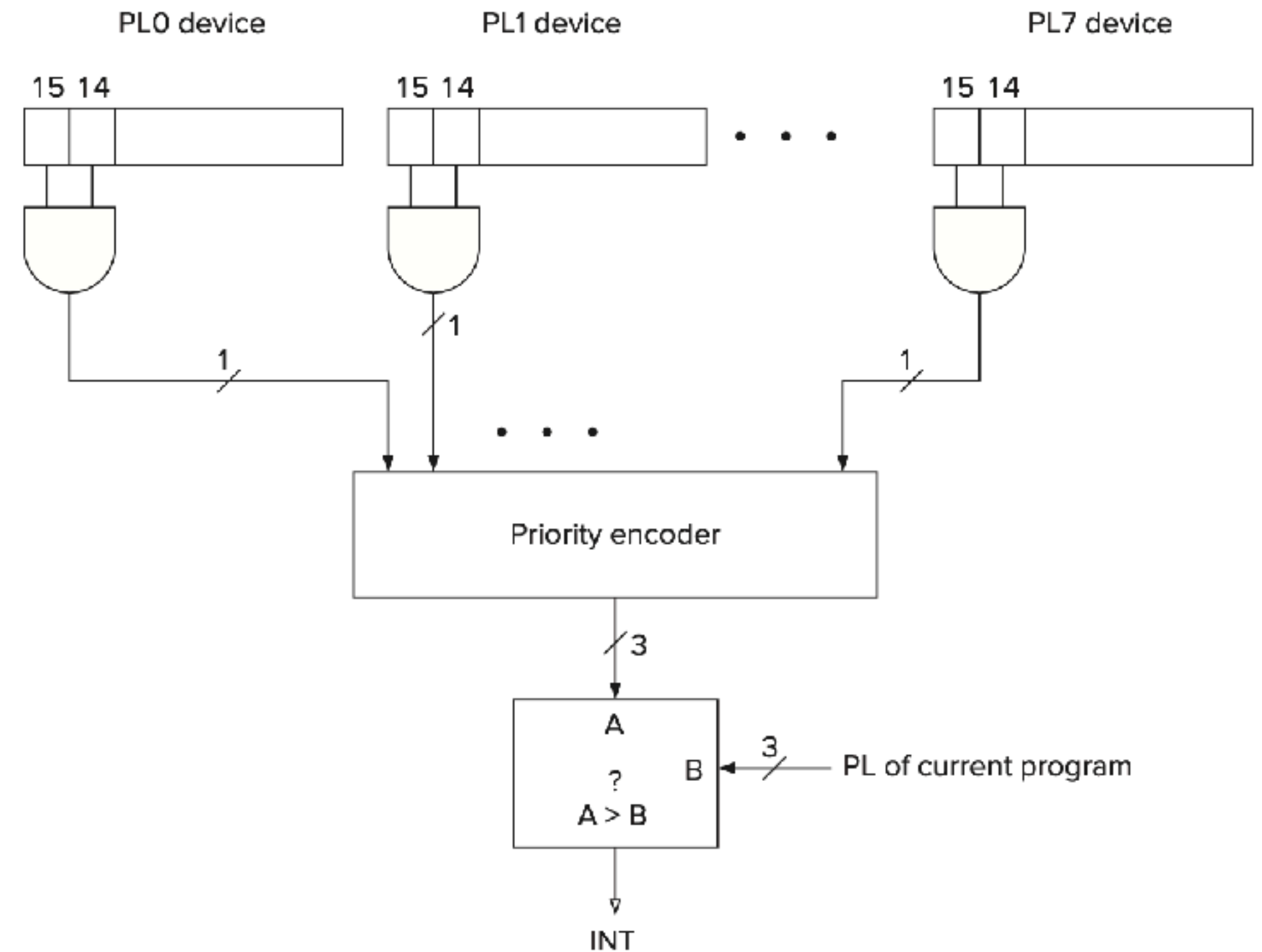
Generating an INTerrupt

Generating an INTerrupt

- All devices asserting an interrupt signal are compared

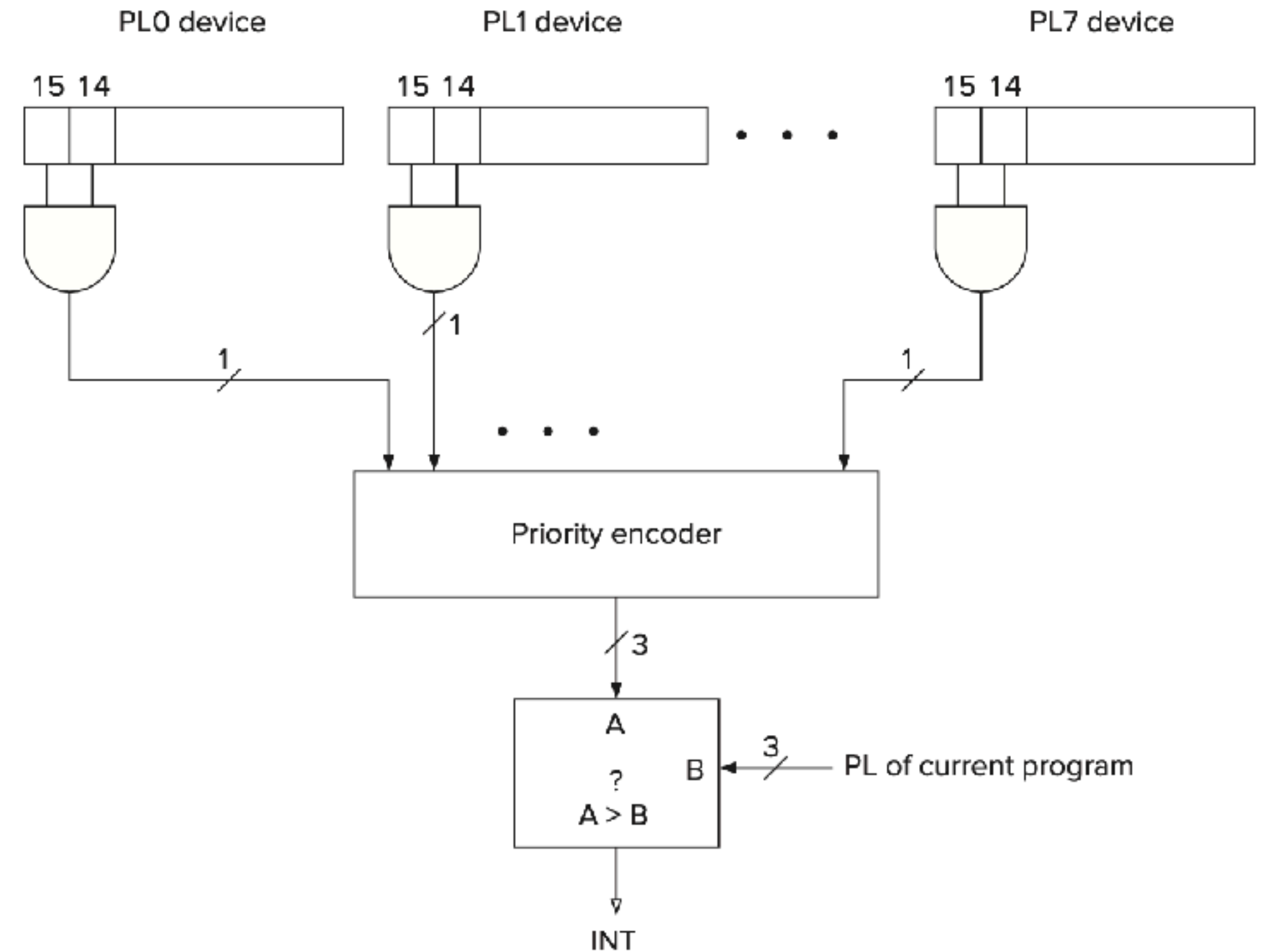
Generating an INTerrupt

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



Generating an INTerrupt

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



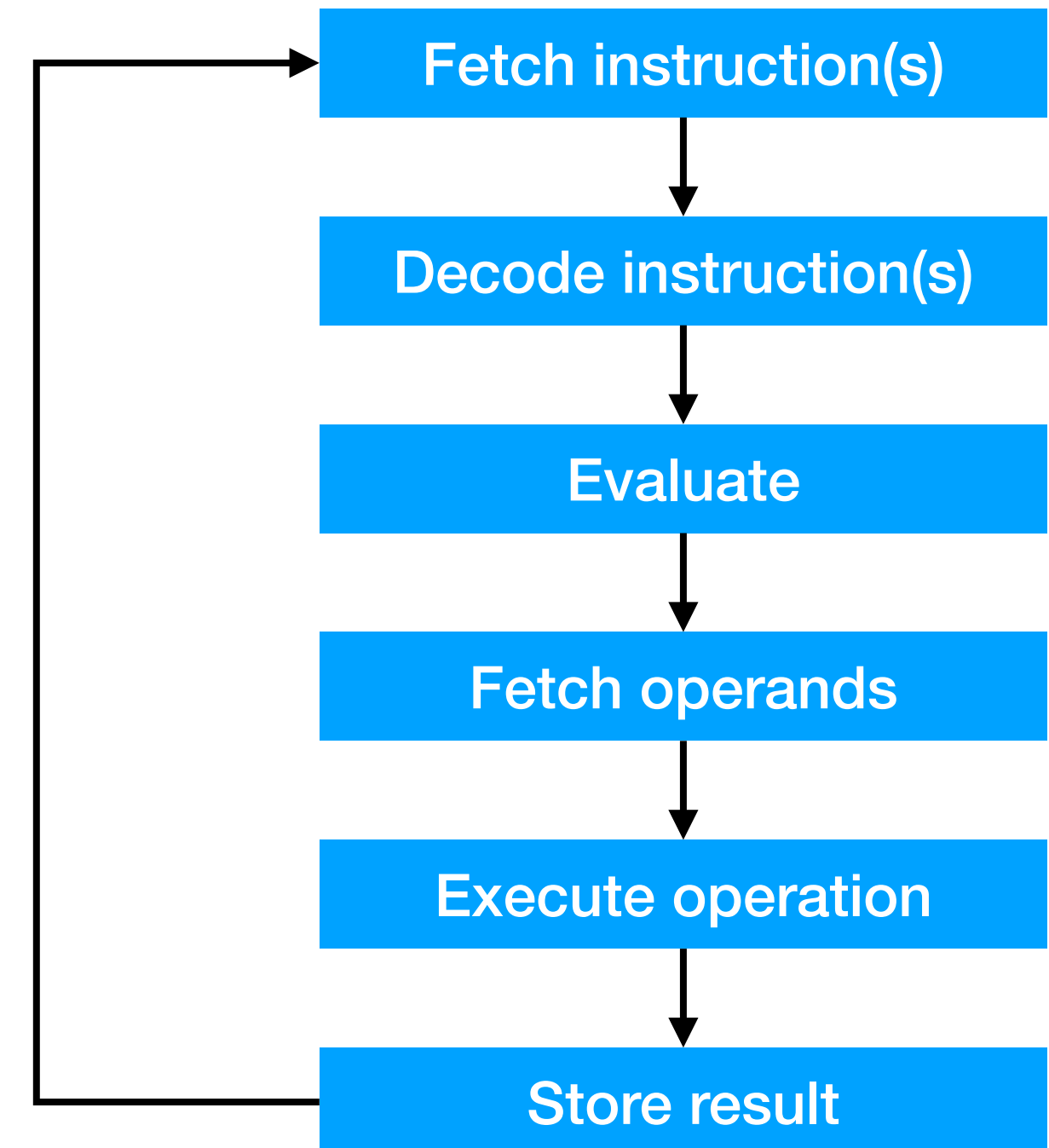
How processor detects INTerrupts

How processor detects INTerrupts

- Recall instruction cycle.

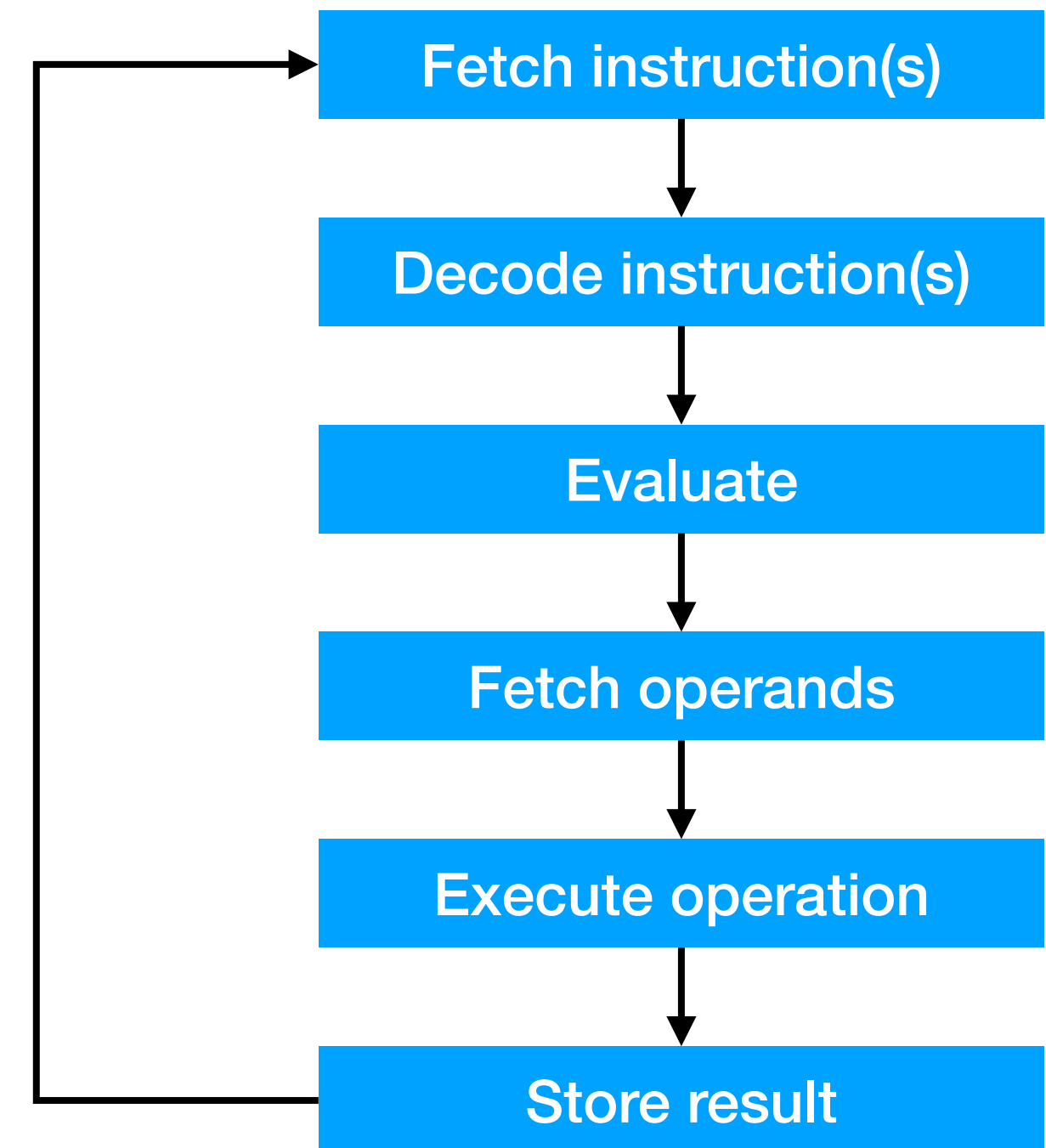
How processor detects INTerrupts

- Recall instruction cycle.

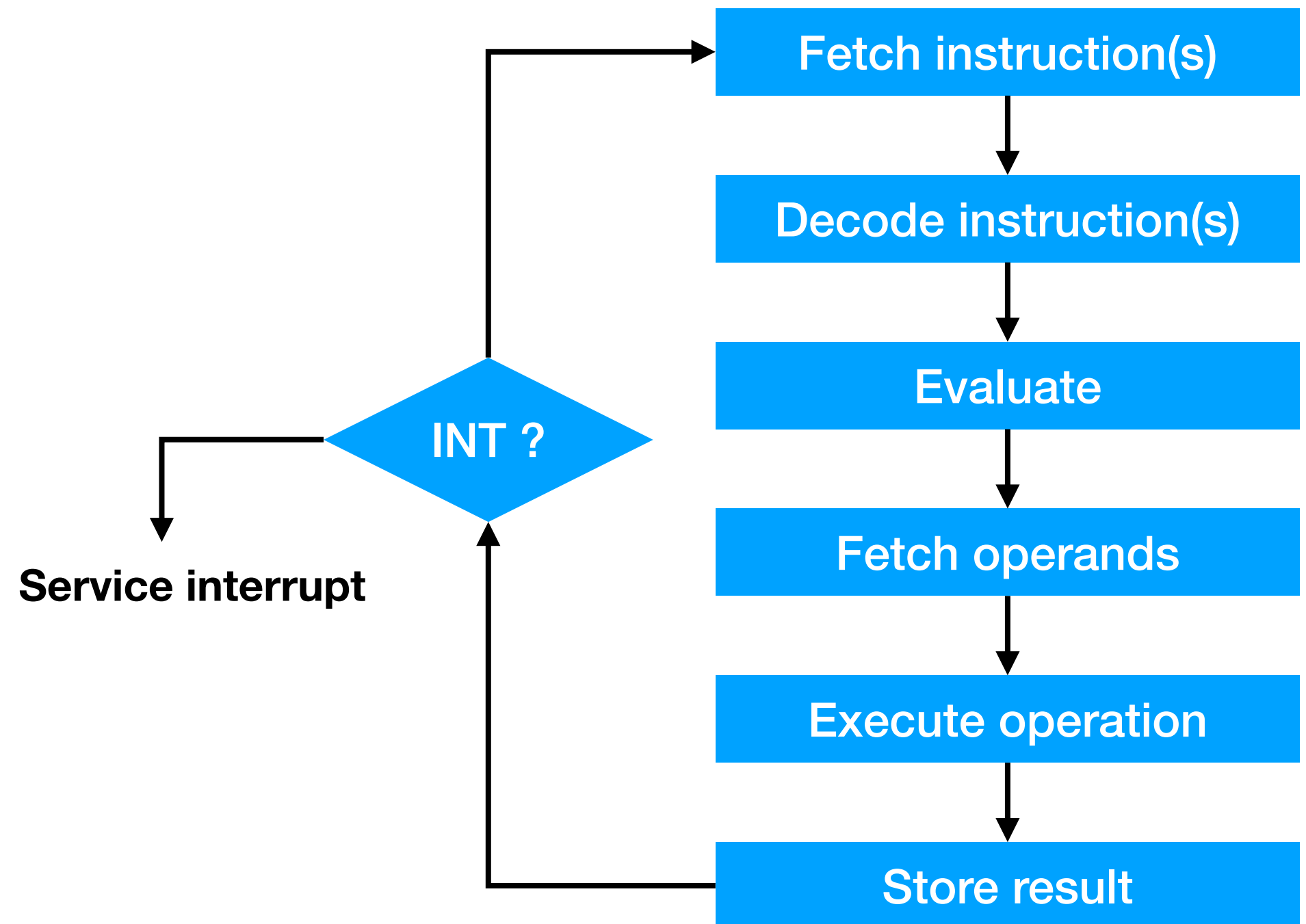


How processor detects INTerrupts

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

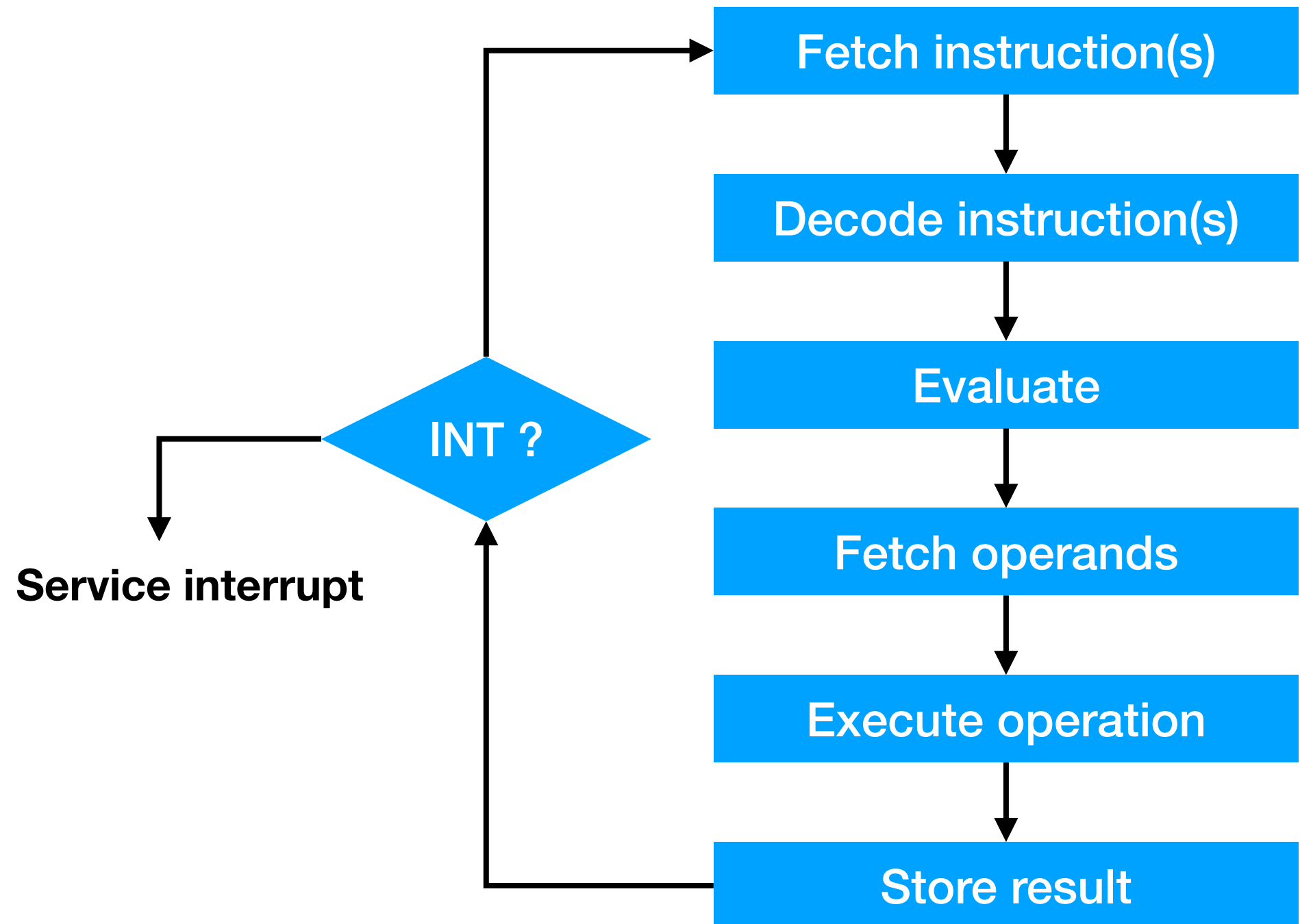


How processor detects INTerrupts



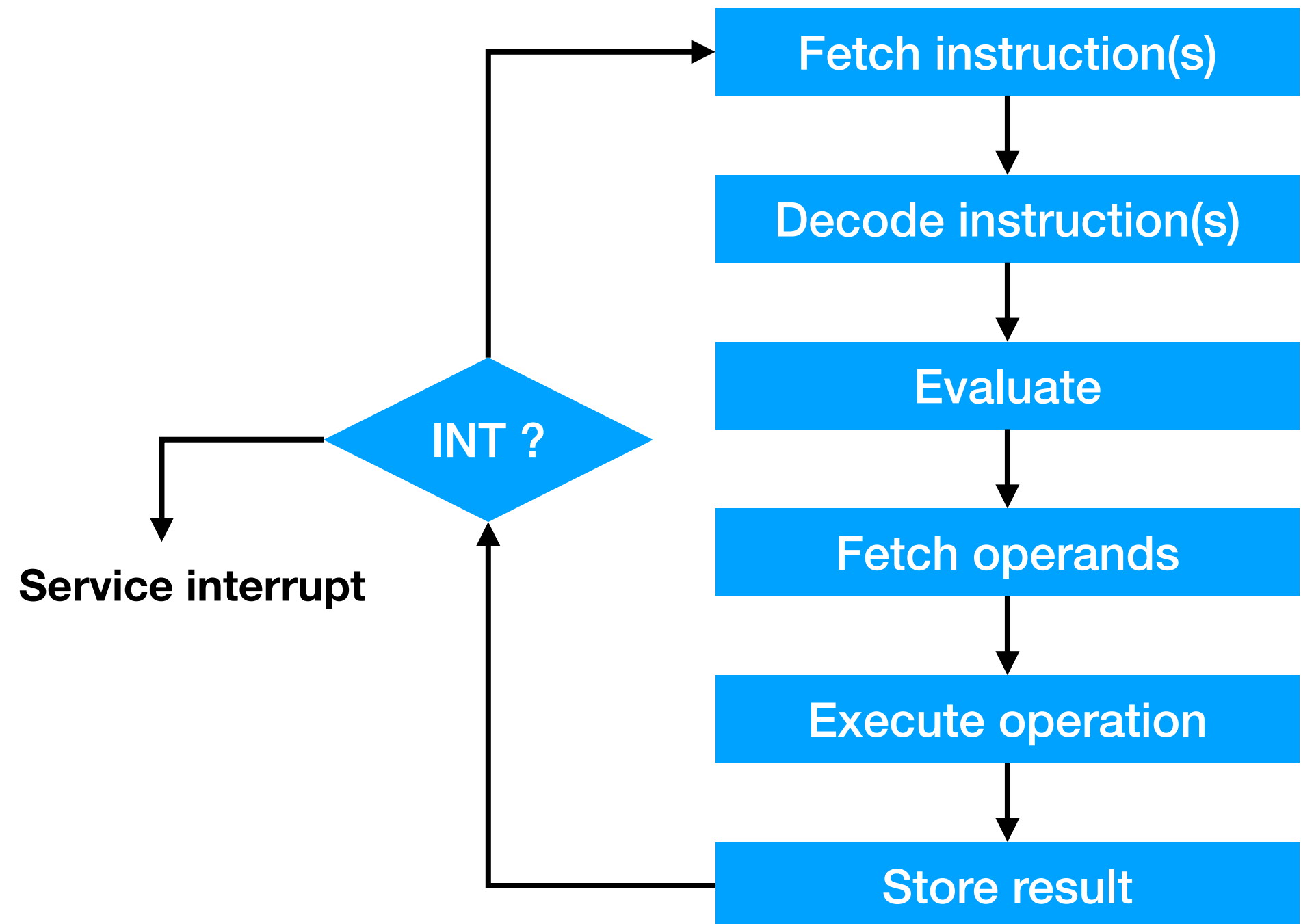
How processor detects INTerrupts

- Control unit needs to:



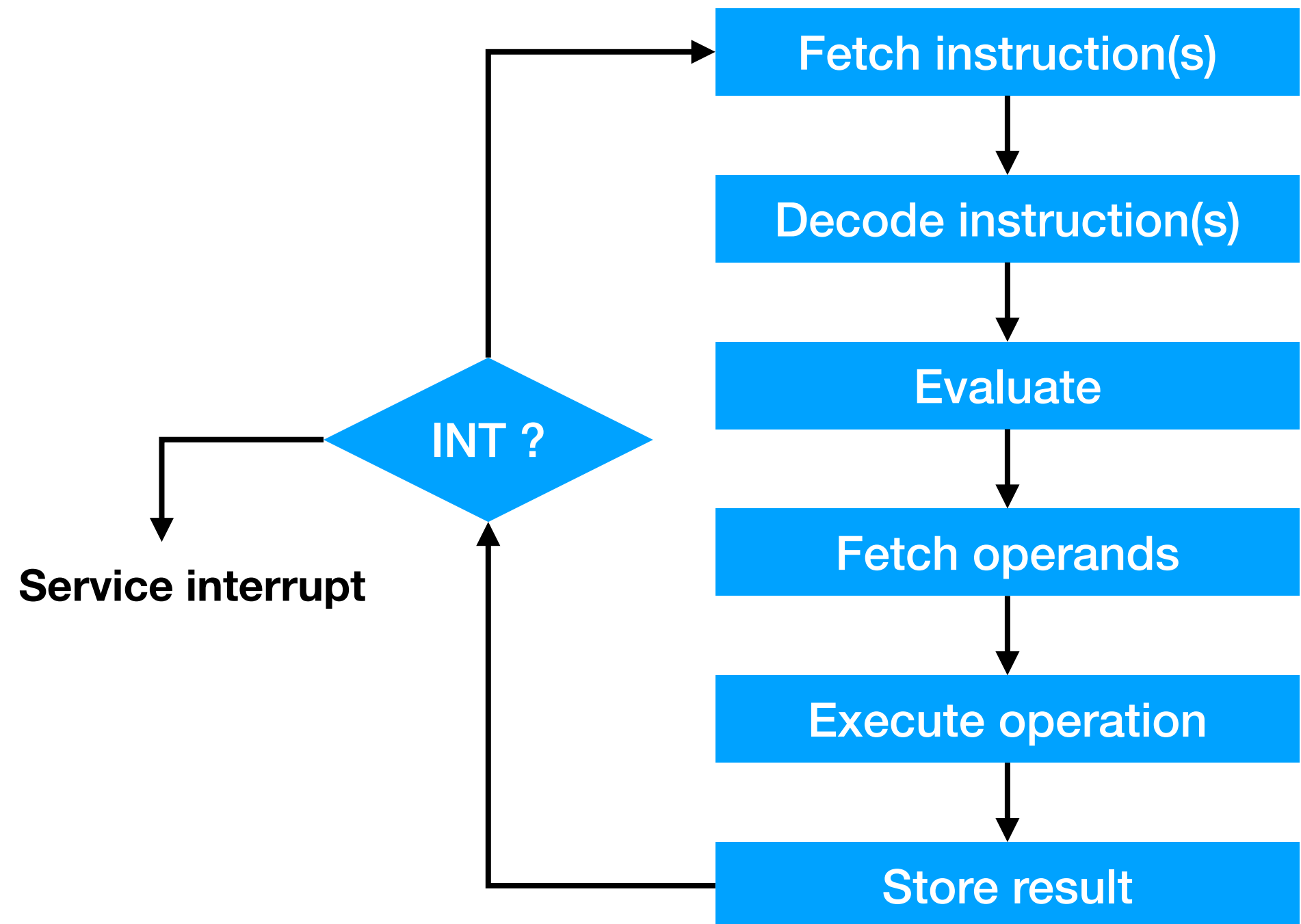
How processor detects INTerrupts

- Control unit needs to:
 - save state information to be able to resume interrupted program



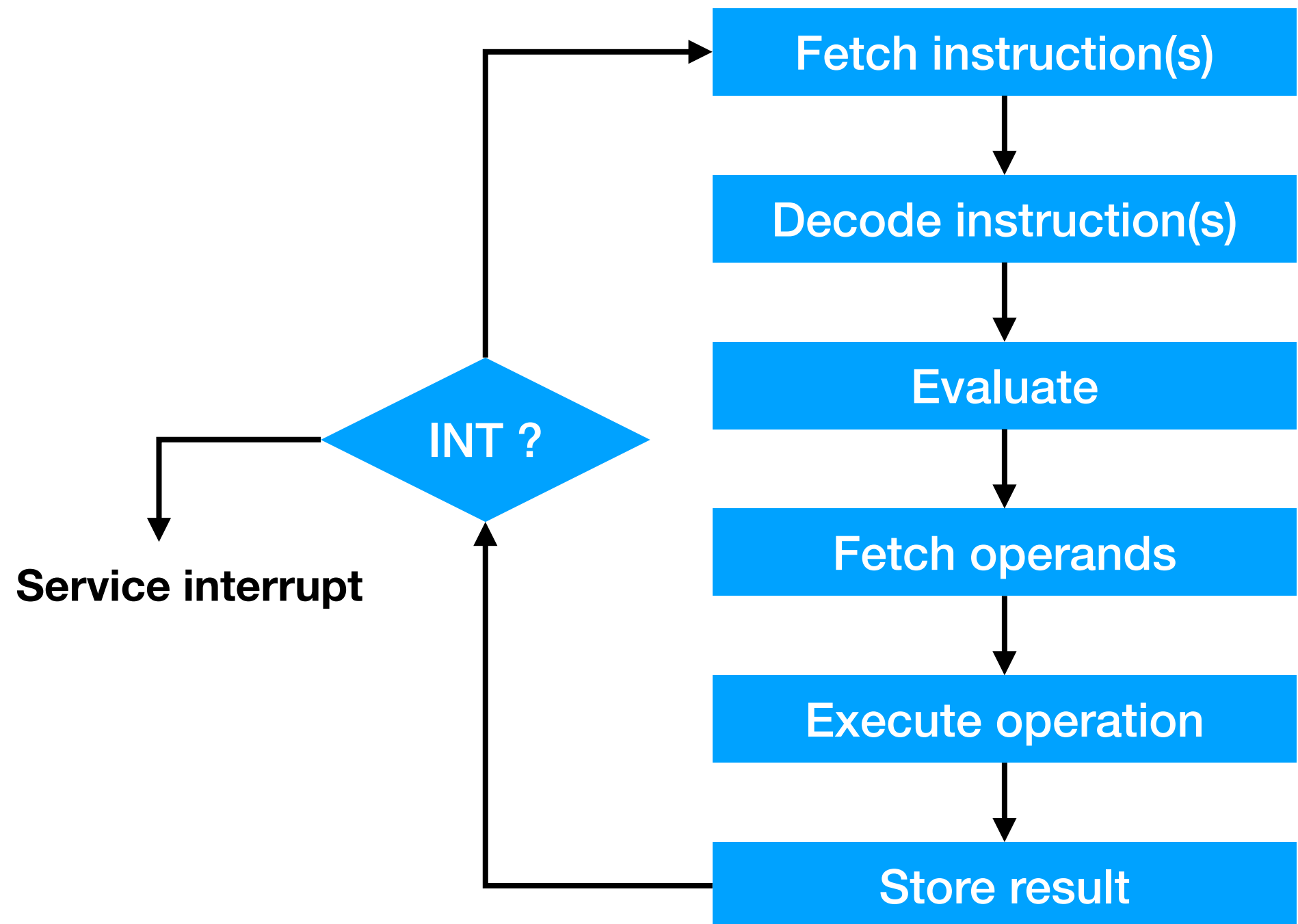
How processor detects INTerrupts

- 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)



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

Handling the INTerrupt

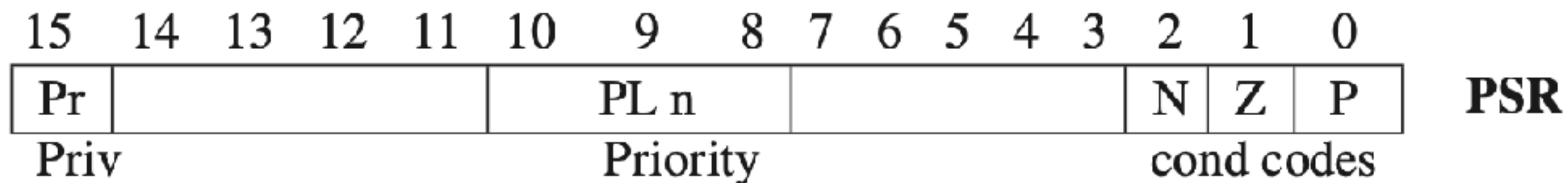
- What should be saved for program to resume?

Handling the INTerrupt

- What should be saved for program to resume?
 - PC (Program Counter)

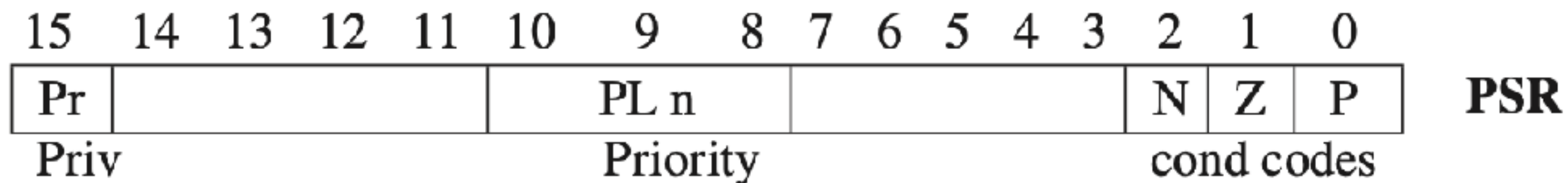
Handling the INTerrupt

- What should be saved for program to resume?
 - PC (Program Counter)
 - Processor Status Register (PSR)



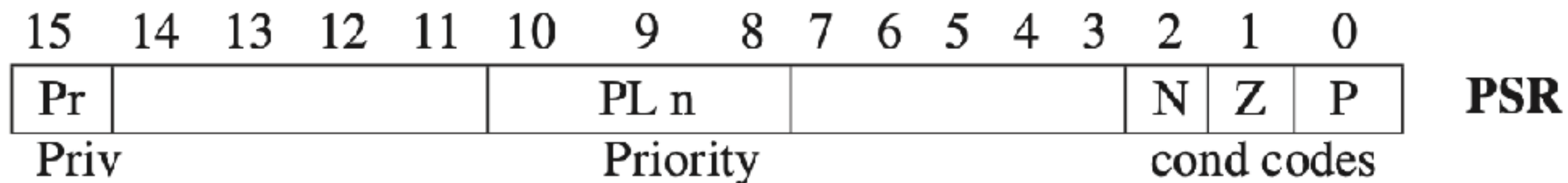
Handling the INTerrupt

- What should be saved for program to resume?
 - PC (Program Counter)
 - Processor Status Register (PSR)
 - What about GPRs?



Handling the INTerrupt

- What should be saved for program to resume?
 - PC (Program Counter)
 - Processor Status Register (PSR)
 - What about GPRs?
 - Callee saved



Privilege vs. priority

Privilege vs. priority

- Privilege

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.

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

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.

User stack vs. supervisor stack

User stack vs. supervisor stack

- Portion of privileged memory reserved for stack in supervisor mode (why?)

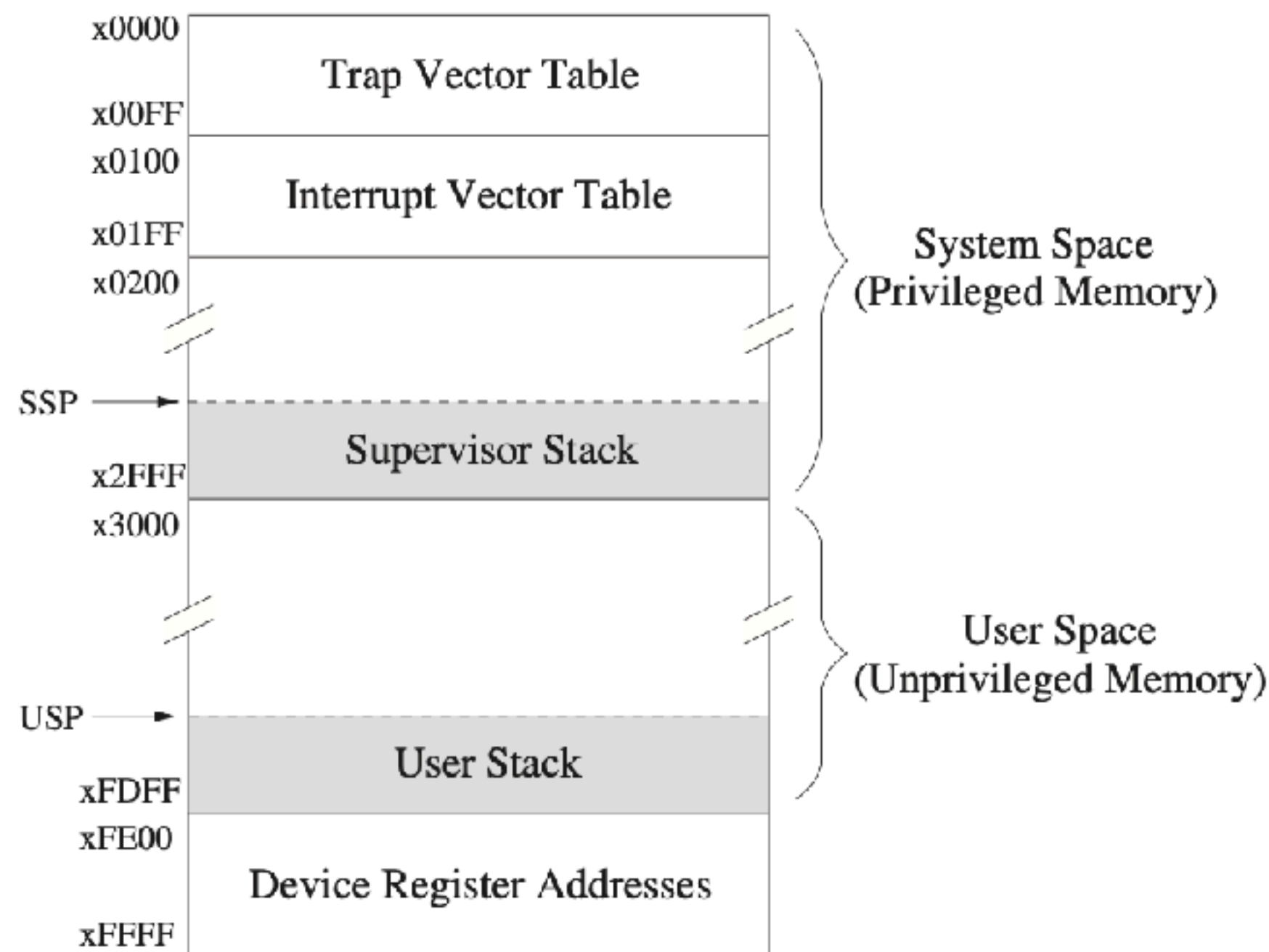


Figure A.1 - P&P 3rd Ed.

User stack vs. supervisor stack

- Portion of privileged memory reserved for stack in supervisor mode (why?)
- R6 is used to denote active TOS always.

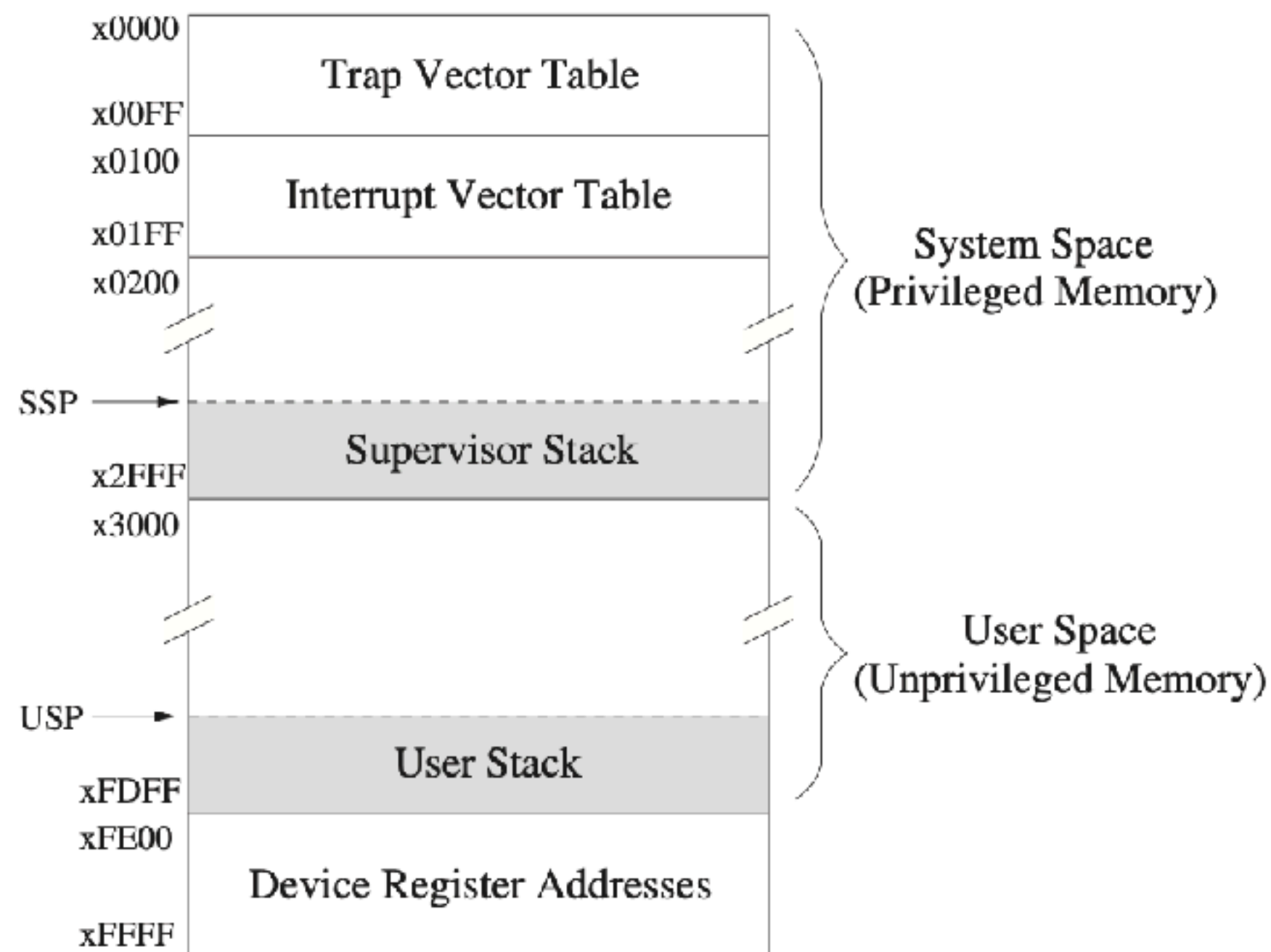


Figure A.1 - P&P 3rd Ed.

User stack vs. supervisor stack

- Portion of privileged memory reserved for stack in supervisor mode (why?)
- R6 is used to denote active TOS always.
 - Can only run in user or supervisor mode at a time.

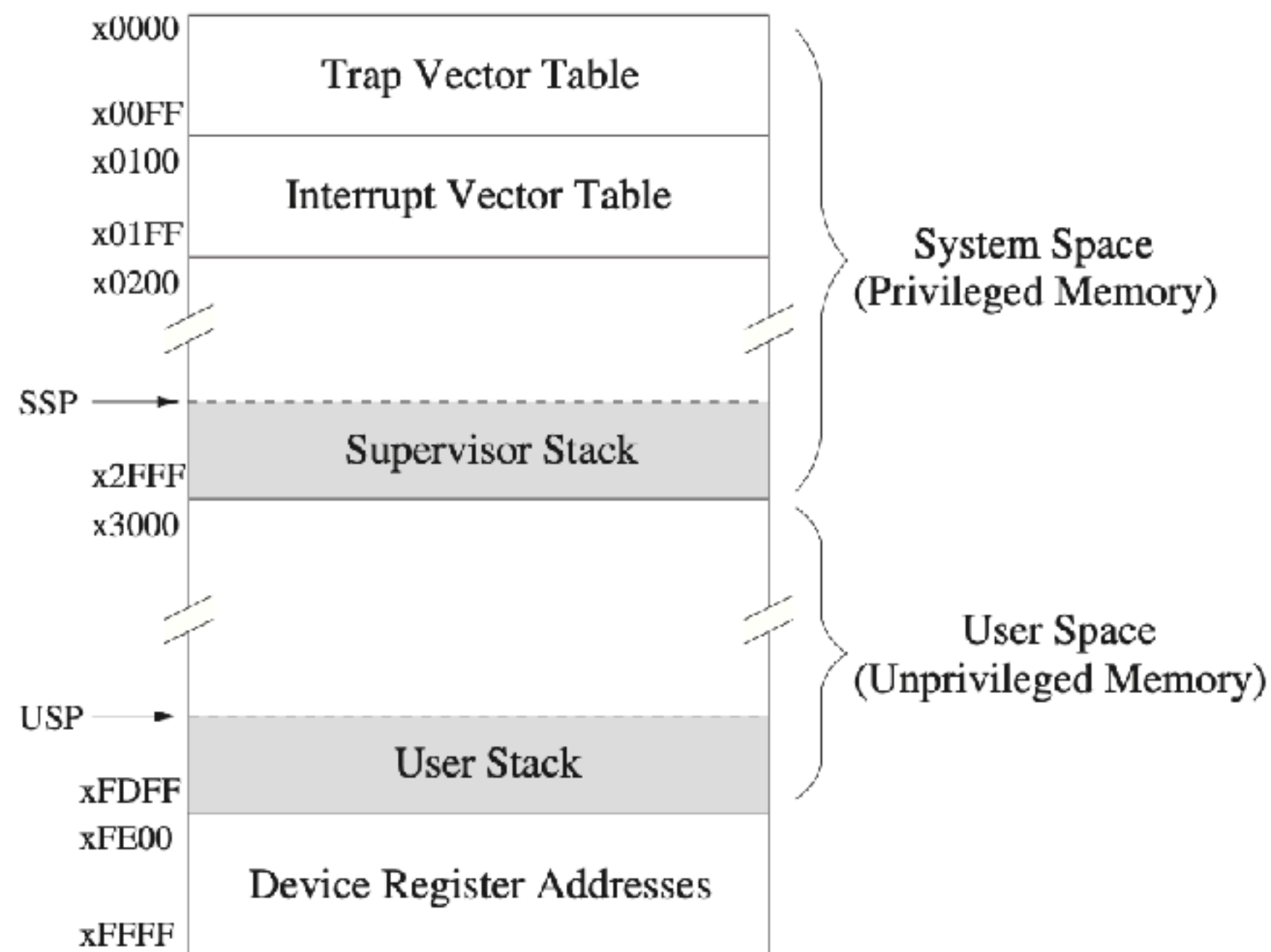


Figure A.1 - P&P 3rd Ed.

User stack vs. supervisor stack

- Portion of privileged memory reserved for stack in supervisor mode (why?)
- 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

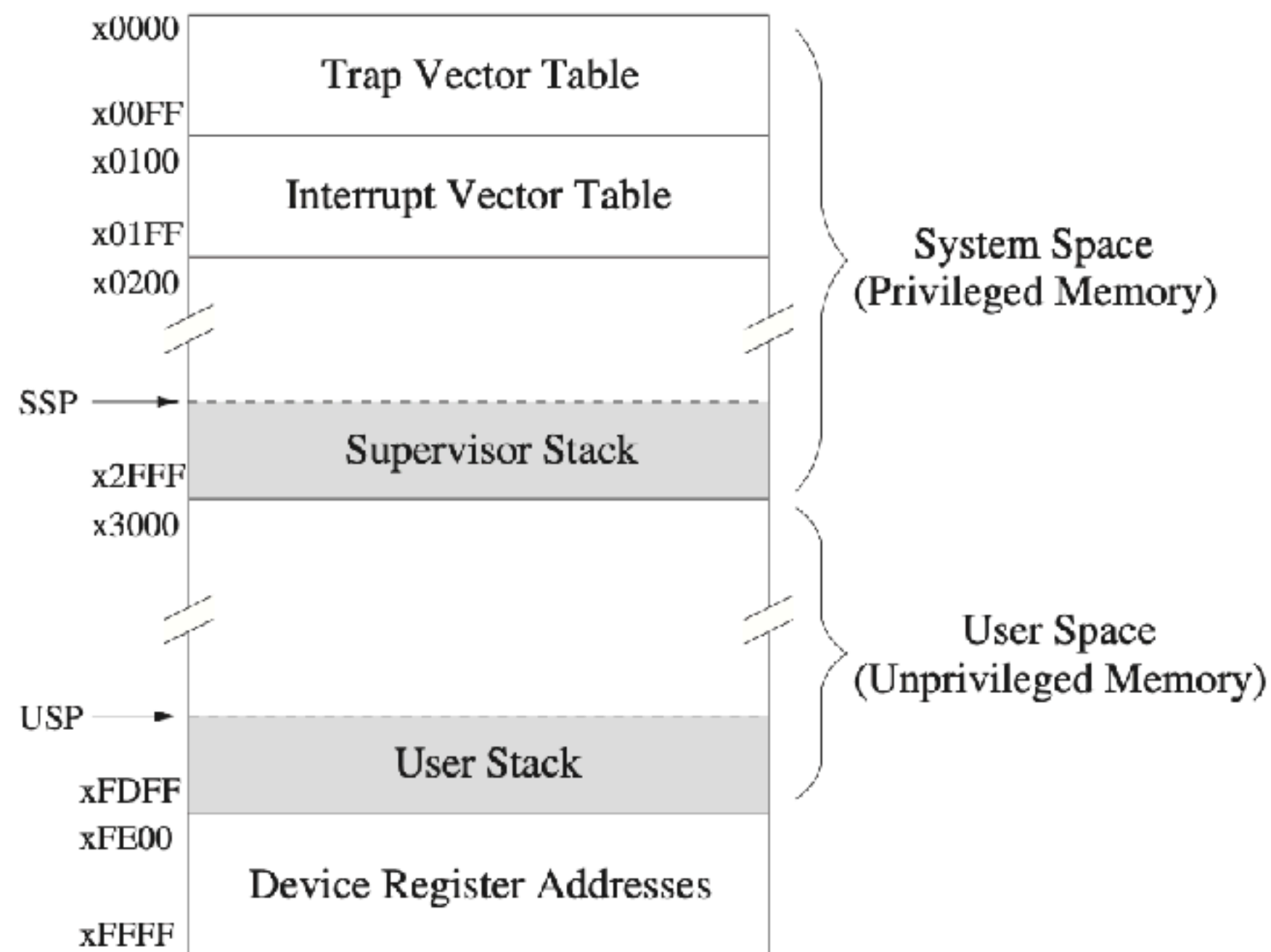


Figure A.1 - P&P 3rd Ed.

Handling the INTerrupt

Handling the INTerrupt

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

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

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

Summary of steps

1. Processor detects asserted `INT` along with `INTV`

Summary of steps

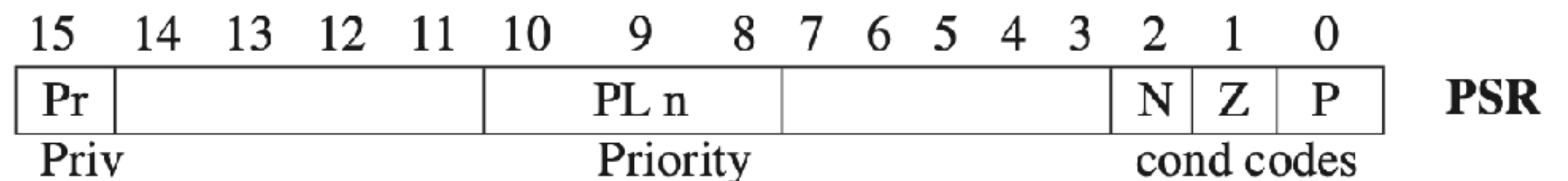
1. Processor detects asserted `INT` along with `INTV`
2. Save `R6` to `Saved_USP`. Set `R6` to `Saved_SSP`.

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

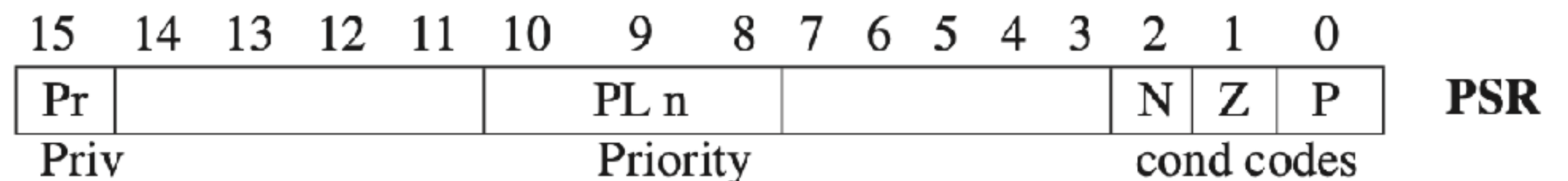
Summary of steps

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
4. Set **PSR[15]=0** and set **PSR[10:8]** = priority of ISR



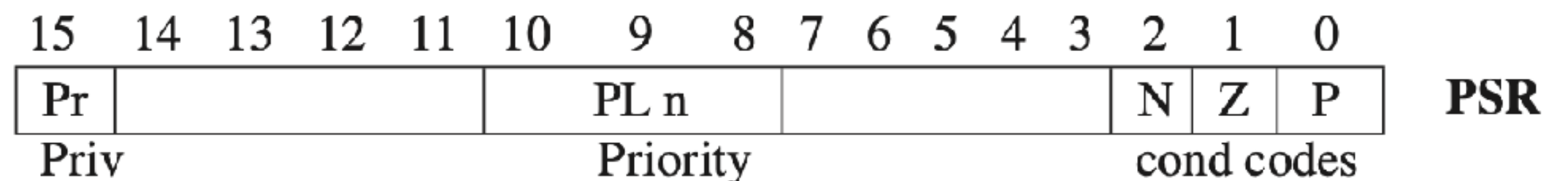
Summary of steps

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
4. Set **PSR[15]=0** and set **PSR[10:8]** = priority of ISR
 1. Set **PSR[2:0]=010** (arbitrarily sets condition codes) \longrightarrow 3rd Ed.



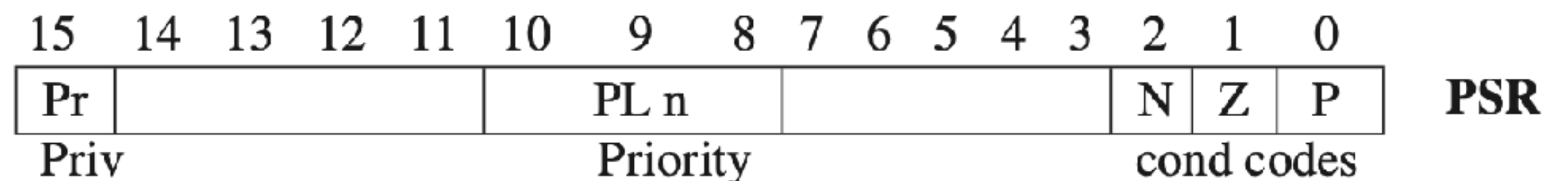
Summary of steps

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
4. Set **PSR[15]=0** and set **PSR[10:8]** = priority of ISR
 1. Set **PSR[2:0]=010** (arbitrarily sets condition codes) \longrightarrow 3rd Ed.
5. Load **MAR** with **x01vv**, where **vv**= 8-bit interrupt vector, then load **MDR**



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
4. Set **PSR[15]=0** and set **PSR[10:8]** = priority of ISR
 1. Set **PSR[2:0]=010** (arbitrarily sets condition codes) \longrightarrow 3rd Ed.
5. Load **MAR** with **x01vv**, where **vv**= 8-bit interrupt vector, then load **MDR**
6. Set **PC=MDR**



Returning from (TRAP or) INTerrupt: RTI

Returning from (TRAP or) INTerrupt: RTI

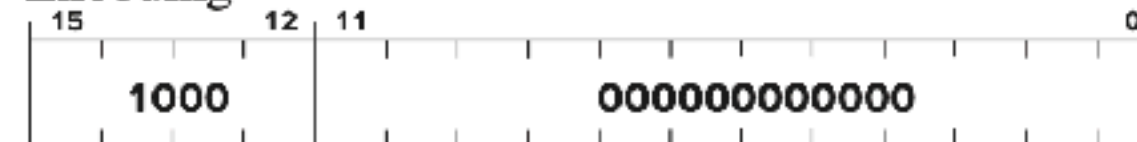
- Both **TRAP** routines and **INT**errupt routines end with the **RTI** instruction.

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.

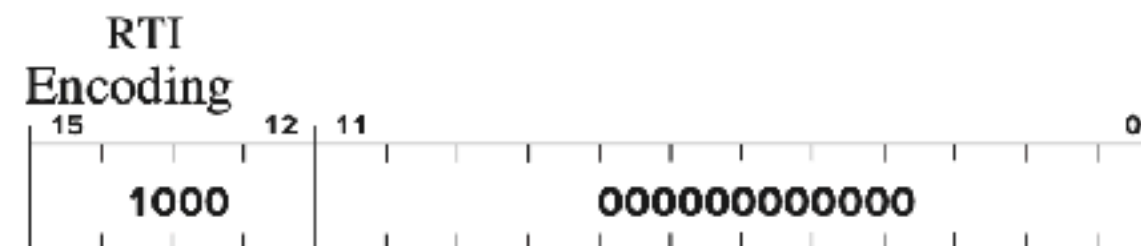
Returning from (TRAP or) INTerrupt: RTI

- Both **TRAP** routines and **INT**errupt routines end with the **RTI** instruction.
- Pop **PC** & **PSR** from supervisor stack

RTI

Return from Trap or Interrupt

Assembler Format



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.

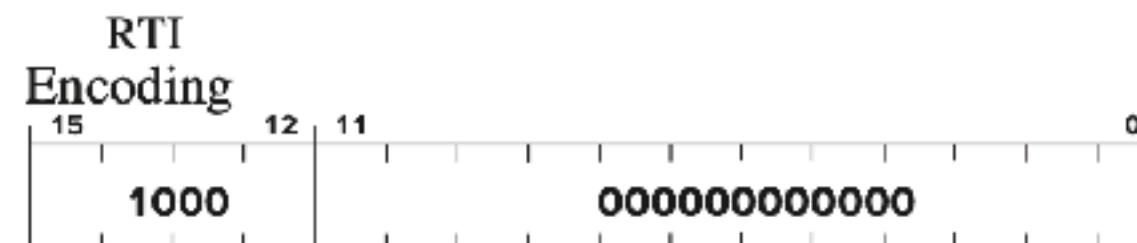
Returning from (TRAP or) INTerrupt: RTI

- 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

Return from Trap or Interrupt

Assembler Format



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.
```

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



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;
```

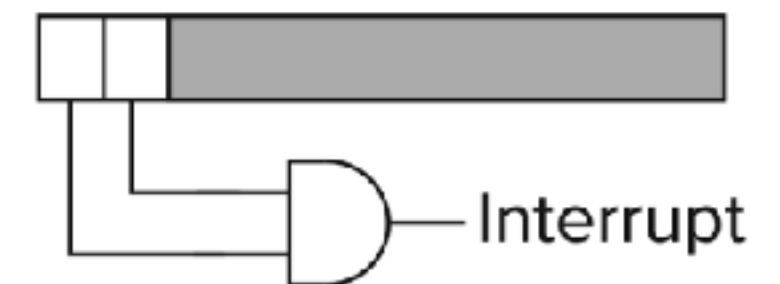
Description

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.

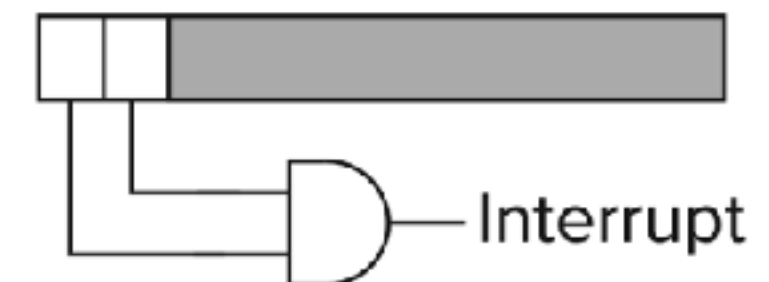
Interrupt example



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
```

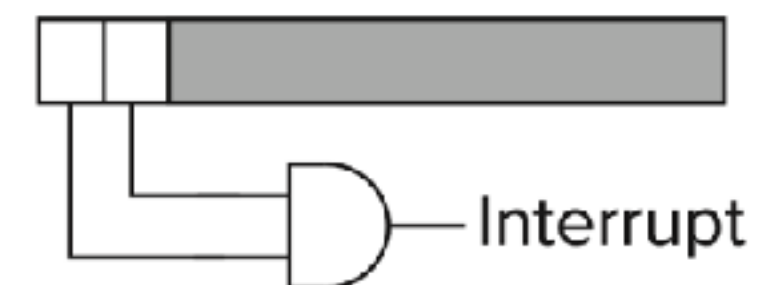


Interrupt example

```
.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

- Exceptions happen when something ... well *unexpected* happens within the processor.

Exceptions

- Exceptions happen when something ... well *unexpected* happens within the processor.
- **Examples:** Privilege mode violation (PMV), illegal/invalid opcodes, accessing privileged memory (ACV)

Exceptions

- Exceptions happen when something ... well *unexpected* happens within the processor.
 - **Examples:** Privilege mode violation (PMV), illegal/invalid opcodes, accessing privileged memory (ACV)
 - Exception services routines occupy memory locations `x0100` to `x017F`, mechanism exactly like INTerrupts except ...

Exceptions

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

Next lecture(s)

- Examples
- Course review: <https://surveys.illinois.edu/sec/1742038613>
 - So far only C++ and recursion
- Exam preparation tips
- ICES forms