Exceptions: Example LC3

Jeremy Bolton, PhD

Assistant Teaching Professor

Constructed using materials:

- Patt and Patel *Introduction to Computing Systems (2nd)*
- Patterson and Hennessy *Computer Organization and Design (4th)*

**A special thanks to Eric Roberts and Mary Jane Irwin

GEORGETOWN UNIVERSITY

Interrupt-Driven I/O

- External device can:

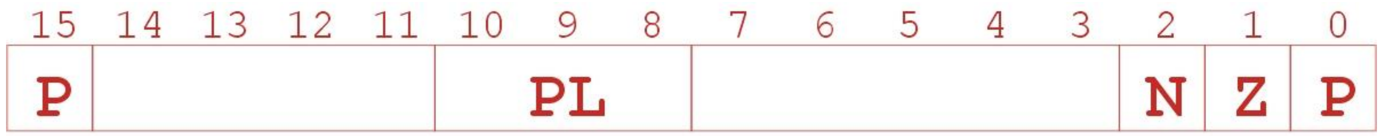
(1) Force currently executing program to stop;
(2) Have the processor satisfy the device's needs; and
(3) Resume the stopped program as if nothing happened.

Interrupt is an ***unscripted subroutine call,*** *triggered by an external event.*

- Why?
  - Polling consumes a lot of cycles, especially for rare events – these cycles can be used for more computation.
  - Example: Process previous input while collecting current input. (See Example 8.1 in text.)

Processor State: Example LC3

- What state is needed to completely capture the state of a running process?
- Processor Status Register
  - Privilege [15], Priority Level [10:8], Condition Codes [2:0]



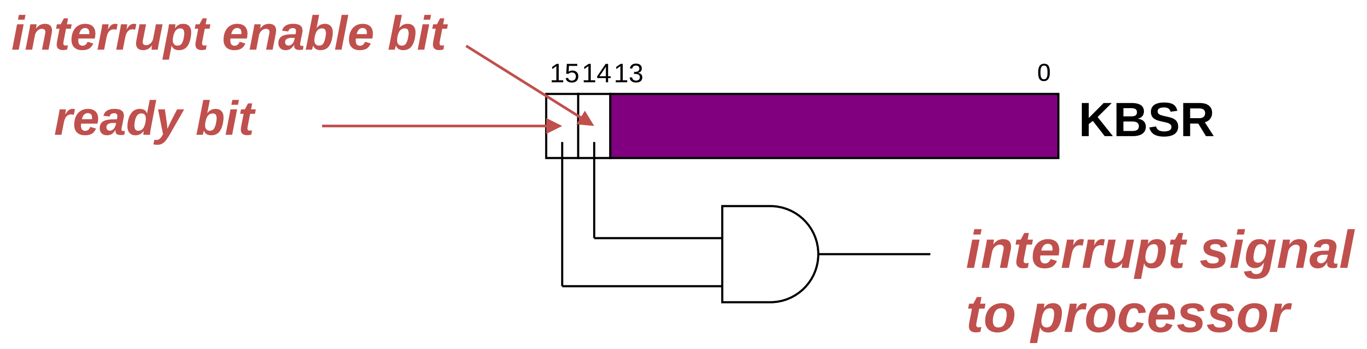
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P						PL							N	Z	P

- Program Counter
  - Pointer to next instruction to be executed.
- Registers
  - All temporary state of the process that's not stored in memory.

Interrupt-Driven I/O

- To implement an interrupt mechanism, we need:
  - A way for the I/O device to signal the CPU that an interesting event has occurred.
  - A way for the CPU to test whether the interrupt signal is set and whether its priority is higher than the current program.

- Generating Signal
  - Software sets "interrupt enable" bit in device register.
  - When ready bit is set and IE bit is set, interrupt is signaled.

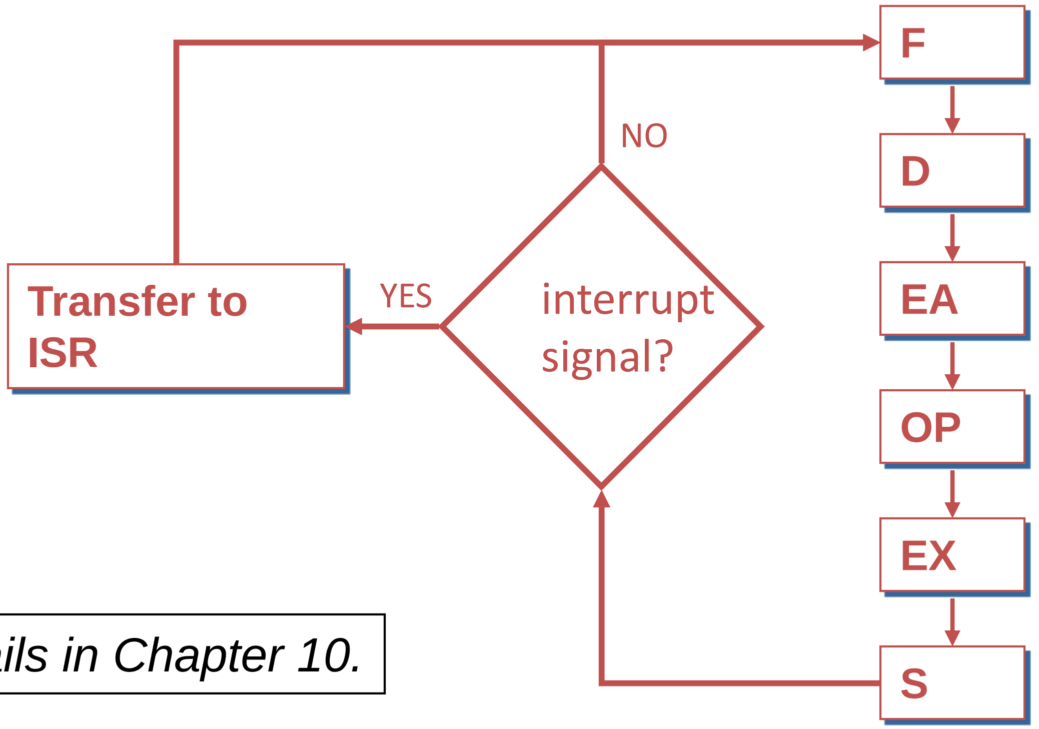


Priority

- Every instruction executes at a stated level of urgency.
- LC-3: 8 priority levels (PL0-PL7)
  - Example:
    - Payroll program runs at PL0.
    - Nuclear power correction program runs at PL6.
  - It's OK for PL6 device to interrupt PL0 program, but not the other way around.
- Priority encoder selects highest-priority device, compares to current processor priority level, and generates interrupt signal if appropriate.

Testing for Interrupt Signal

- CPU looks at signal between STORE and FETCH phases.
- If not set, continues with next instruction.
- If set, transfers control to interrupt service routine.



More details in Chapter 10.

GEORGETOWN UNIVERSITY

Where to Save Processor State?

- Can’t use registers.
  - Programmer doesn’t know when interrupt might occur, so she can’t prepare by saving critical registers.
  - When resuming, need to restore state exactly as it was.

- Memory allocated by service routine?
  - Must save state before invoking routine, so we wouldn’t know where.
  - Also, interrupts may be nested – that is, an interrupt service routine might also get interrupted!

- Use a stack!
  - Location of stack “hard-wired”.
  - Push state to save, pop to restore.

Supervisor Stack

- A special region of memory used as the stack for interrupt service routines.
  - Initial Supervisor Stack Pointer (SSP) stored in Saved.SSP.
  - Another register for storing User Stack Pointer (USP): Saved.USP.
- Want to use R6 as stack pointer.
  - So that our PUSH/POP routines still work.
- When switching from User mode to Supervisor mode (as result of interrupt), save R6 to Saved.USP.

Invoking the Service Routine – The Details

1. If Priv = 1 (user),
   Saved.USP = R6, then R6 = Saved.SSP.
2. Push PSR and PC to Supervisor Stack.
3. Set PSR[15] = 0 (supervisor mode).
4. Set PSR[10:8] = priority of interrupt being serviced.
5. Set PSR[2:0] = 0.
6. Set MAR = x01vv, where vv = 8-bit interrupt vector
   provided by interrupting device (e.g., keyboard = x80).
7. Load memory location (M[x01vv]) into MDR.
8. Set PC = MDR; now first instruction of ISR will be fetched.

Note: This all happens between
the STORE RESULT of the last user instruction and
the FETCH of the first ISR instruction.

Returning from Interrupt

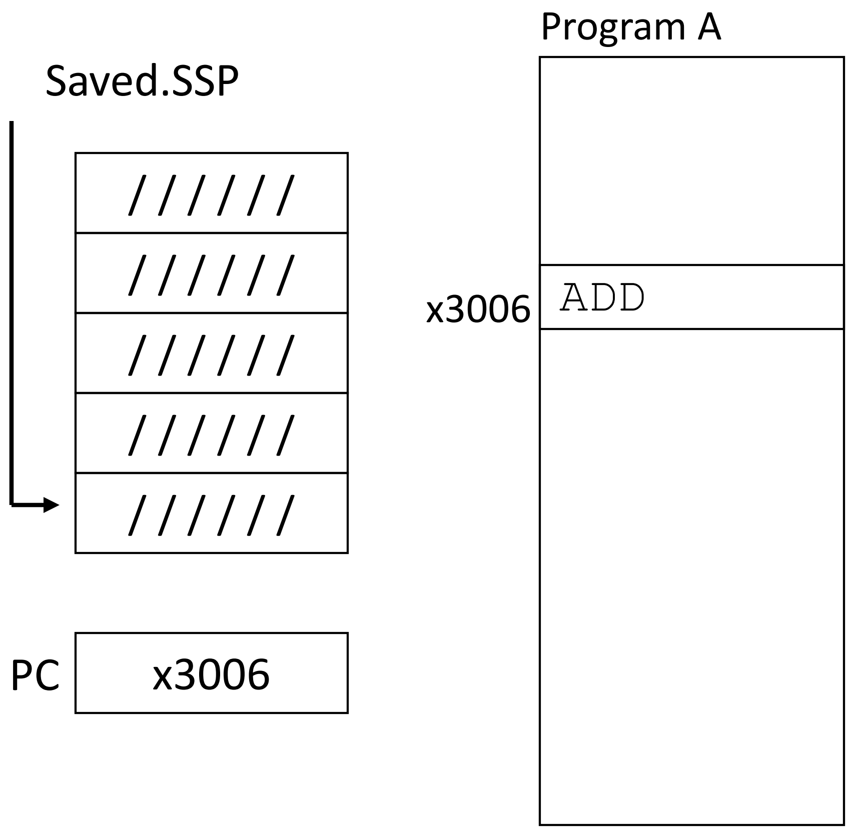
- Special instruction – RTI – that restores state.

	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, R6 = Saved.USP.
   (If going back to user mode, need to restore User Stack Pointer.)

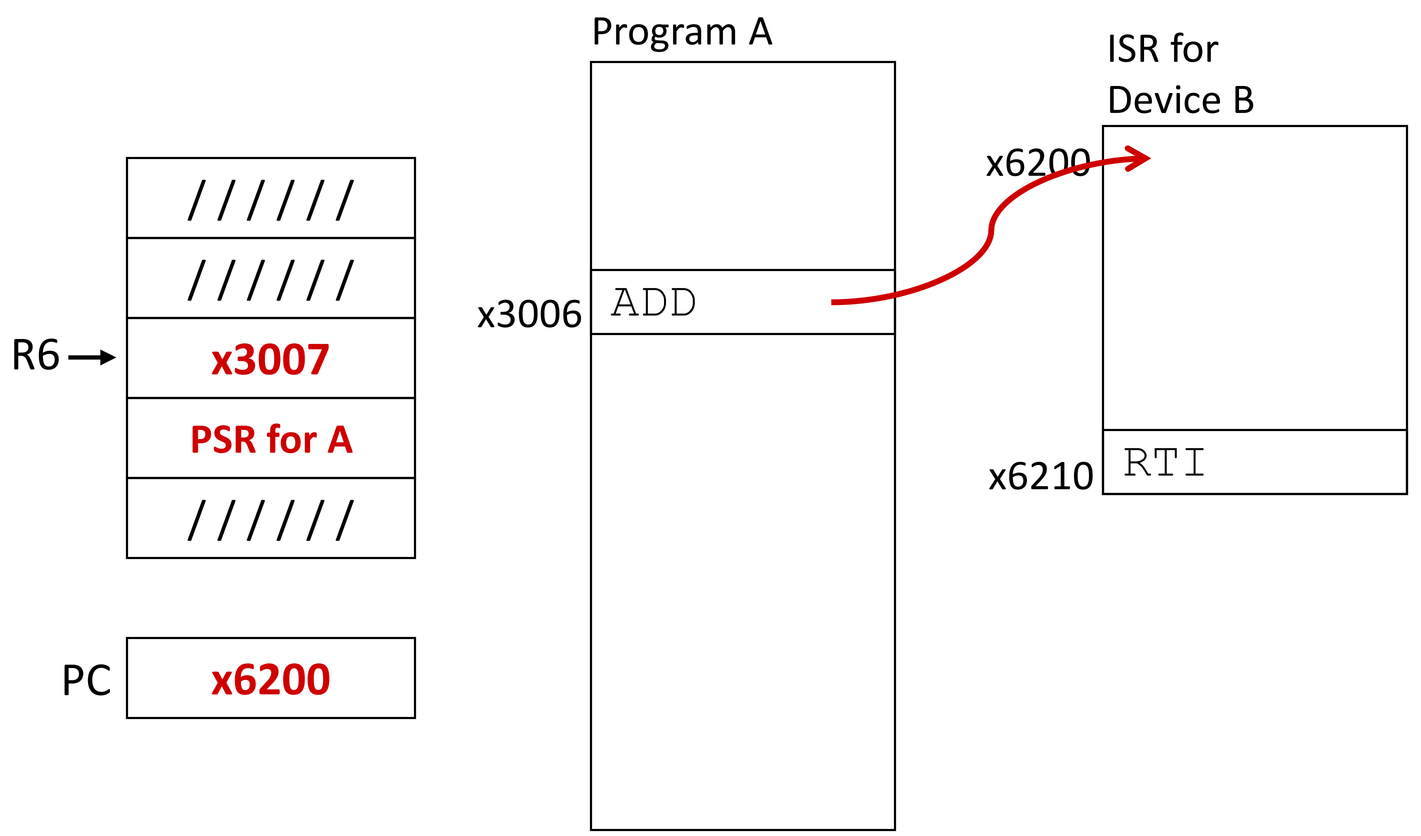
- RTI is a privileged instruction.
  - Can only be executed in Supervisor Mode.
  - If executed in User Mode, causes an <u>exception</u>.
    (More about that later.)

Example (1)



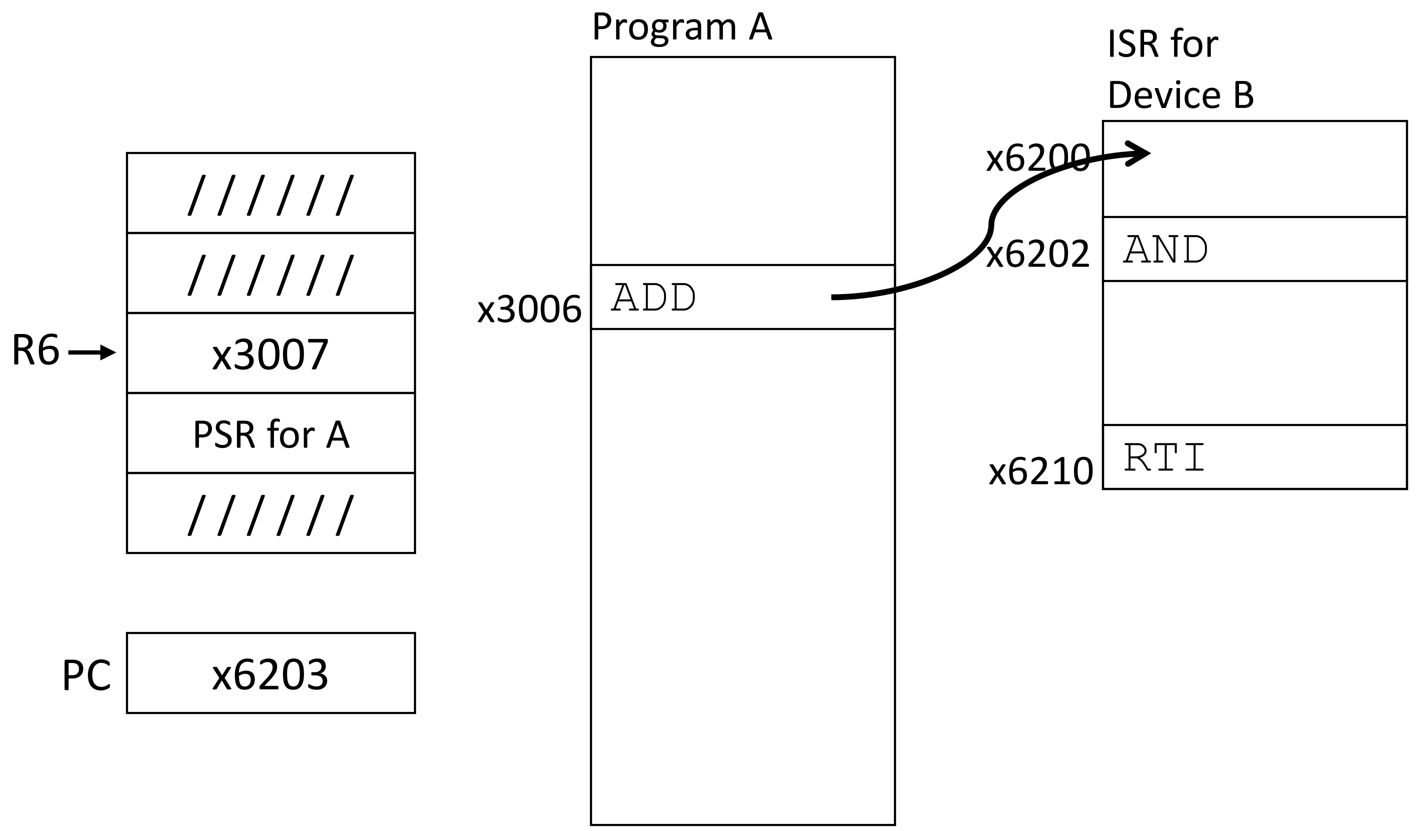
Executing ADD at location x3006 when Device B interrupts.

Example (2)



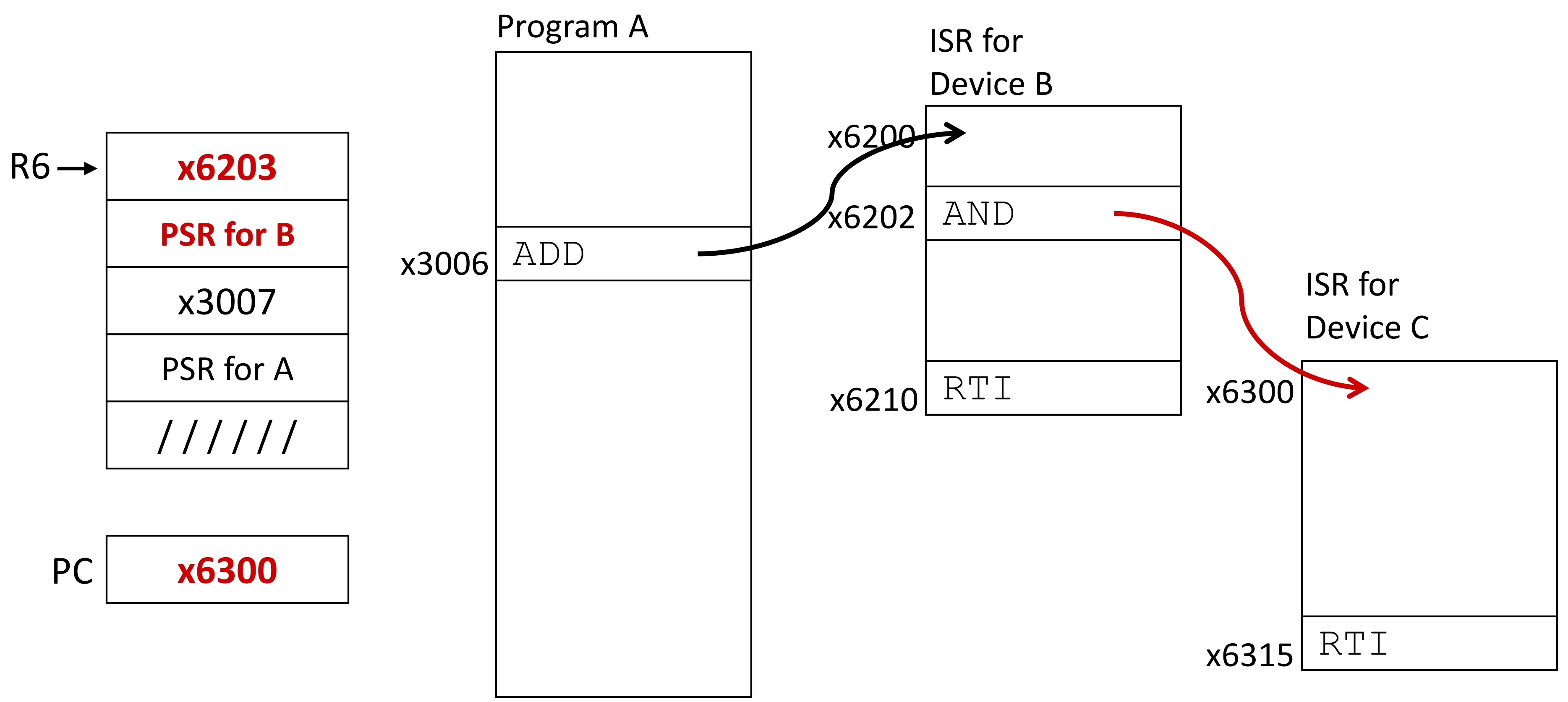
Saved.USP = R6. R6 = Saved.SSP.
Push PSR and PC onto stack, then transfer to
Device B service routine (at x6200).

Example (3)



Executing AND at x6202 when Device C interrupts.

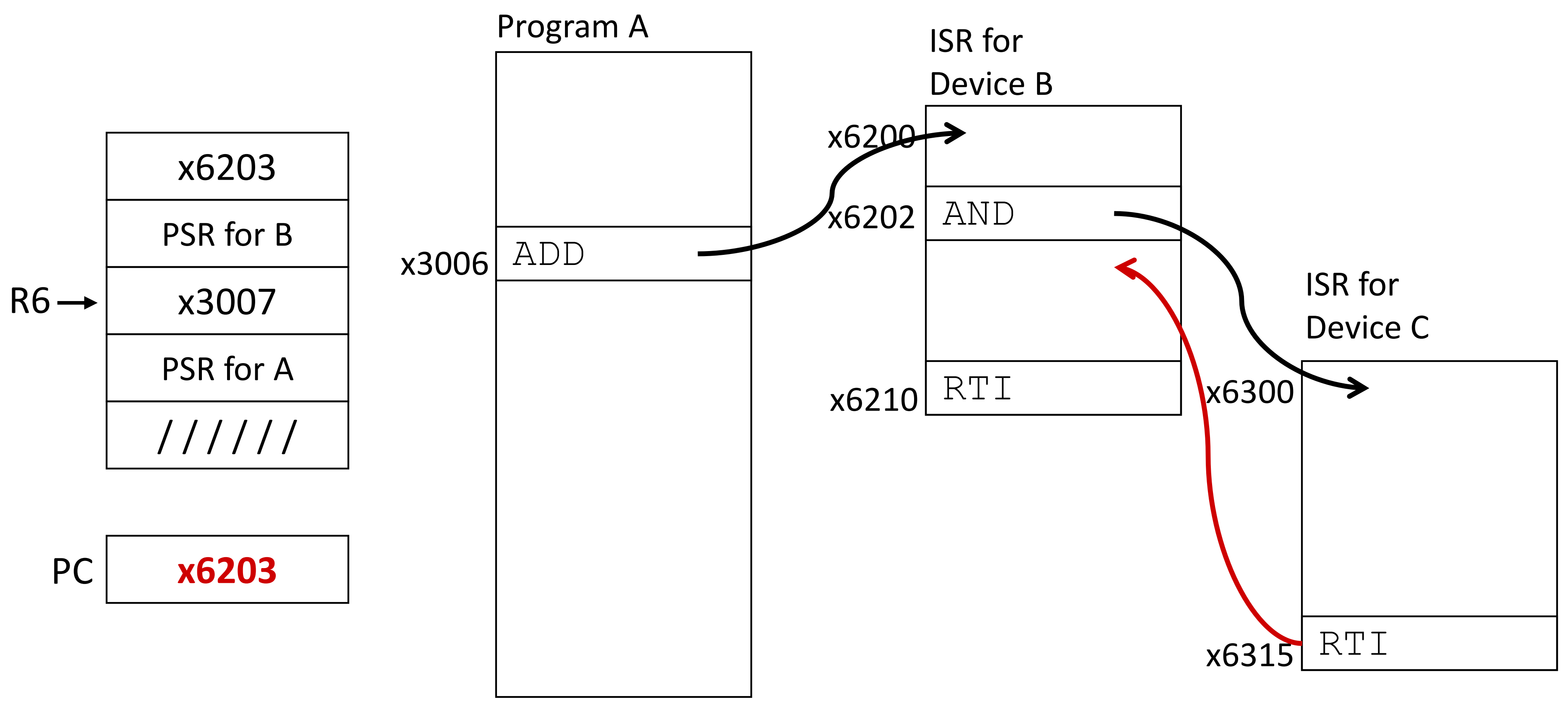
Example (4)



Push PSR and PC onto stack, then transfer to Device C service routine (at x6300).

GEORGETOWN UNIVERSITY

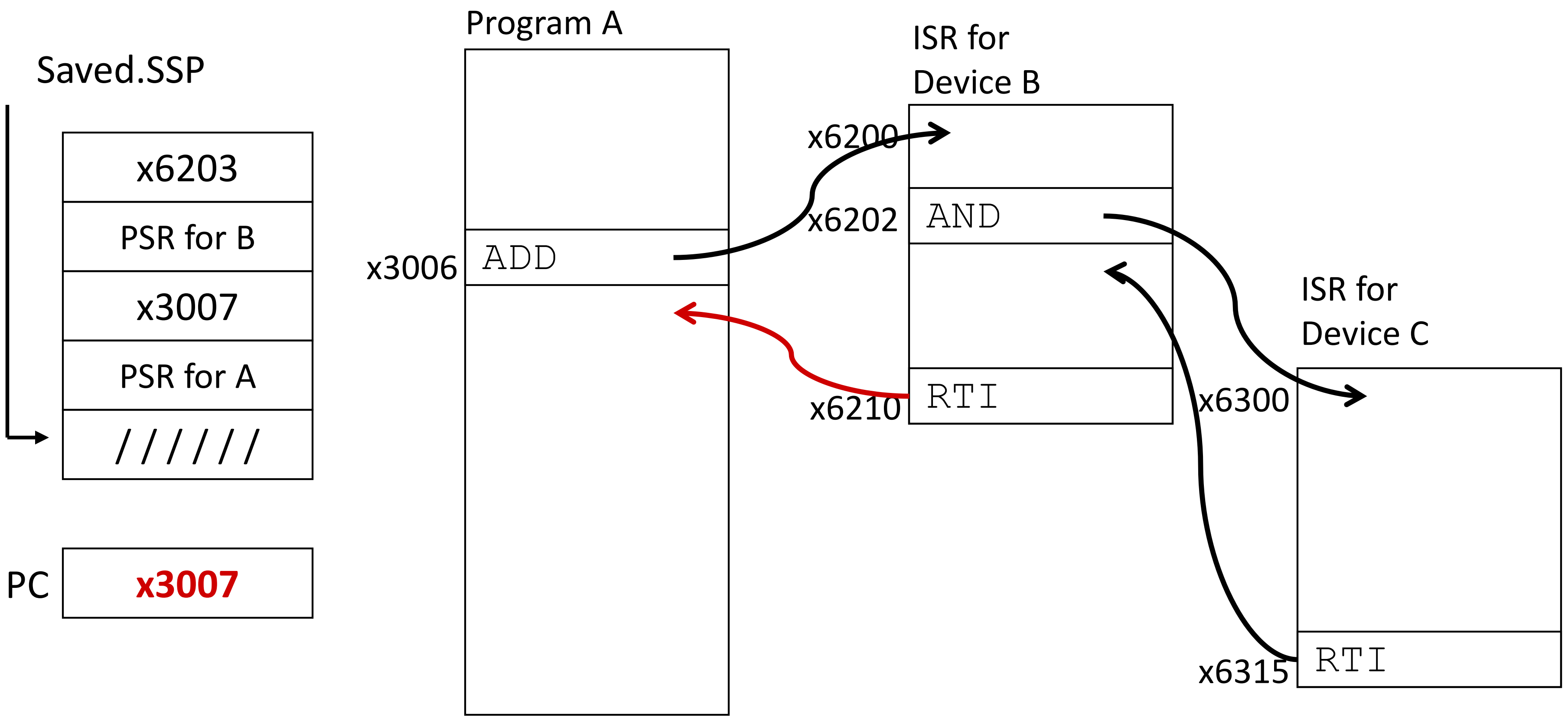
Example (5)



Execute RTI at x6315; pop PC and PSR from stack.

GEORGETOWN UNIVERSITY

Example (6)



Execute RTI at x6210; pop PSR and PC from stack.
Restore R6. Continue Program A as if nothing happened.

Exception: Internal Interrupt

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

- Examples:
  - Privileged operation (e.g., RTI in user mode)
  - Executing an illegal opcode
  - Divide by zero
  - Accessing an illegal address (e.g., protected system memory)

- Handled just like an interrupt
  - Vector is determined internally by type of exception
  - Priority is the same as running program

Interrupt-driven I/O (MORE DETAILS!)

- We can now finish servicing the interrupt!
  - Remember that an INT signal “hijacks” the CPU, diverting it without warning from processing the program.
  - We left two questions hanging in our earlier treatment of interrupt handling:
    - How do we save enough information about the current program to be able to pick up where we left off after servicing the interrupt? And
    - How do we reset the CPU to deal with the Interrupt Service Routine?
  - In both cases, the answer has to do with *state* (remember the Finite State Machine?), and the use of stacks.

GEORGETOWN UNIVERSITY

The State of a Program

- State is a “snapshot” of the system
  - The complete state specification would include the contents of relevant memory locations, the general purpose registers, and some special registers, including the PC and …
- Introducing … the PSR
  - or Processor Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pr					---PL---							N	Z	P	
Privilege					Priority Level								Condition codes		

  - which stores the most important pieces of information about the current program

The PSR

- Privilege
  - PSR[15] indicates whether the current program is running in Supervisor (0) or User (1) mode
    - This allows some resources to be kept “off limits” to non-OS programs
- Priority
  - PSR[10:8] stores the priority level of the current program
    - There are 8 levels of priority, from PL0 (lowest) to PL7 (highest).
- Condition Codes
  - PSR[2:0] stores the current NZP condition codes

Saving the state - 1

- We only need to save the PC and the PSR
  - We assume that the ISRs will be smart enough to save relevant Registers (remember “callee save”?)
  - The PC and the PSR between them hold enough state information to be able to reconstitute the program when needed.
- Where do we save them?
  - On a stack!
    - Remember, there might be nested interrupts, so simply saving them to a register or reserved memory location would not work.

The Supervisor Stack

- Only Supervisors can use the Supervisor Stack!
  - The User stack & the Supervisor stack are in separate regions of memory
    - The stack pointer for the current stack is always R6.
    - If the current program is in privileged mode, R6 points to the Supervisor stack, otherwise it points to the user stack.
    - Two special purpose registers, Saved.SSP and Saved.USP, are used to store the pointer currently not in use, so the two stacks can share push/pop subroutines without interfering with each other.

Saving the state - 2

- When the CPU receives an INT signal …
  - The privilege mode changes from User to Supervisor mode
    - PSR[15] <= 0
  - The User stack pointer is saved & the Supervisor stack pointer is loaded
    - Saved.USP <= (R6)
    - R6 <= (Saved.SSP)
  - PC and PSR are pushed onto the Supervisor Stack
    - Now the CPU is free to handle the interrupting device!

Loading the state of the ISR

- Vectored interrupts
  - Along with the INT signal, the I/O device transmits its priority level and an 8-bit vector (INTV).
  - If the interrupt is accepted, INTV is expanded to a 16-bit address:
    - The Interrupt Vector Table resides in locations x0100 to x01FF and holds the starting addresses of the various Interrupt Service Routines. (similar to the Trap Vector Table and the Trap Service Routines)
    - INTV is an index into the Interrupt Vector Table, i.e. the address of the relevant ISR is (x0100 + Zext(INTV))
  - The address of the ISR is loaded into the PC
  - The PSR is set as follows:
    - PSR[15] <= 0 (Supervisor mode)
    - PSR[10:8] is set to the priority level of the interrupting device
    - PSR[2:0] <= 000 (no condition codes set)

Returning from the Interrupt

- The last instruction of an ISR is RTI
  - Return from Interrupt (opcode 1000)
  - Pops PSR and PC from the Supervisor stack
  - Restores the condition codes from PSR
  - If necessary (i.e. if the current privilege mode is User) restores the user stack pointer to R6 from Saved.USP
  - Continues running the program as if nothing had happened!