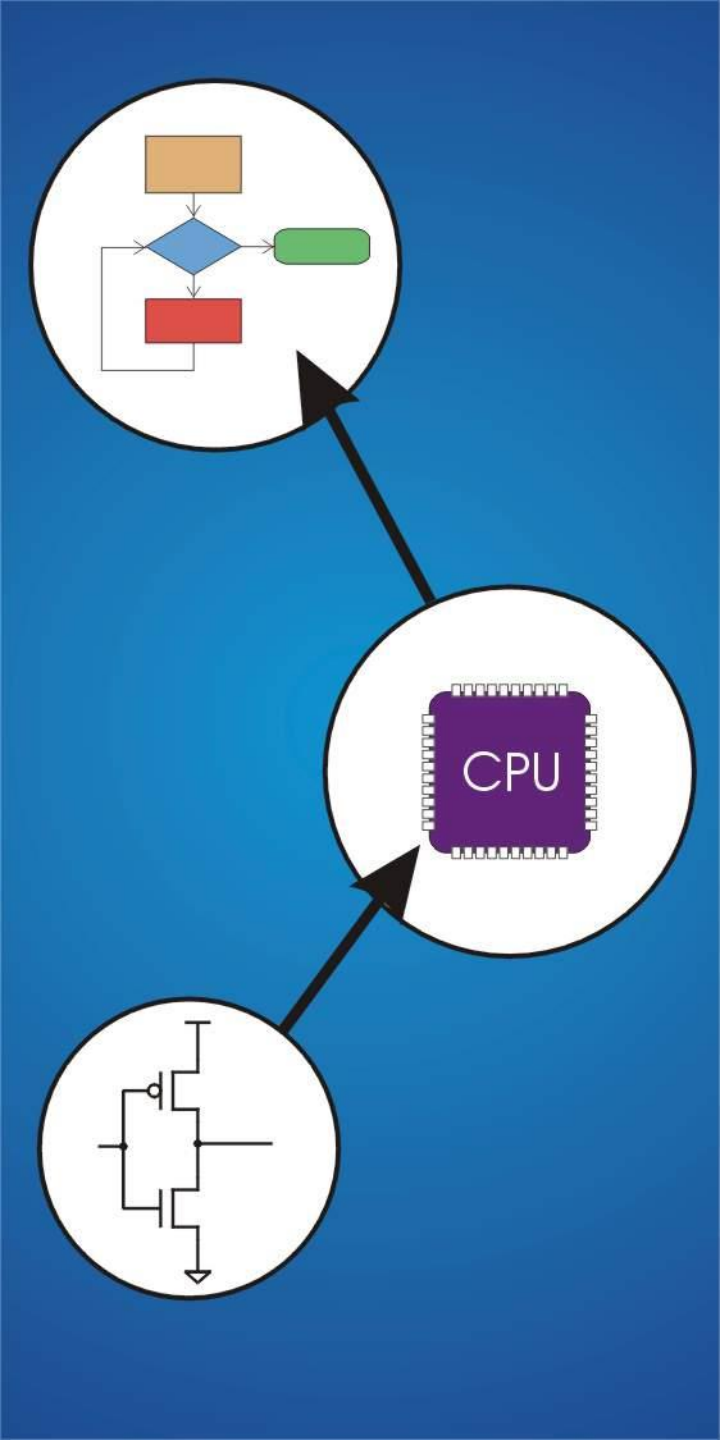


Chapter 10
And, Finally...
The Stack

Stacks: An Abstract Data Type

A LIFO (last-in first-out) storage structure.

- **The first thing you put in is the last thing you take out.**
- **The last thing you put in is the first thing you take out.**

This means of access is what defines a stack, not the specific implementation.

Two main operations:

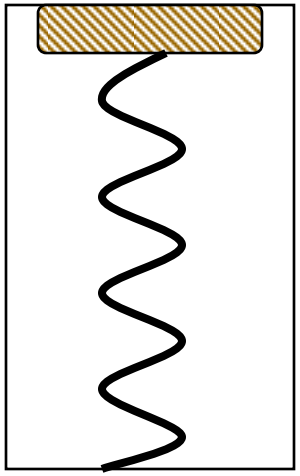
PUSH: add an item to the stack

POP: remove an item from the stack

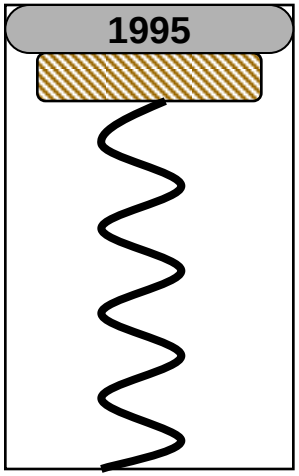
A Physical Stack

Coin rest in the arm of an automobile

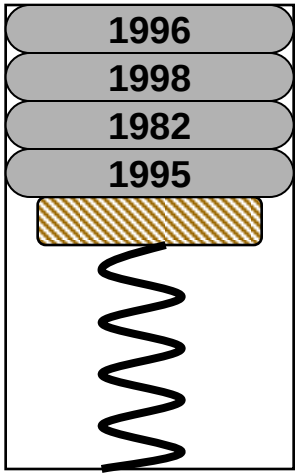
img src: edarts.net



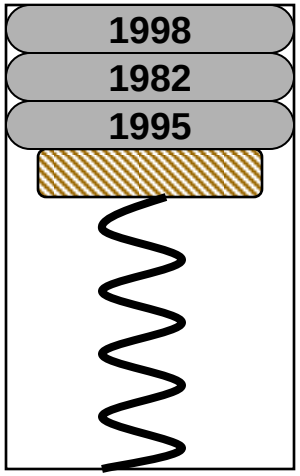
Initial State



After
One Push



After Three
More Pushes

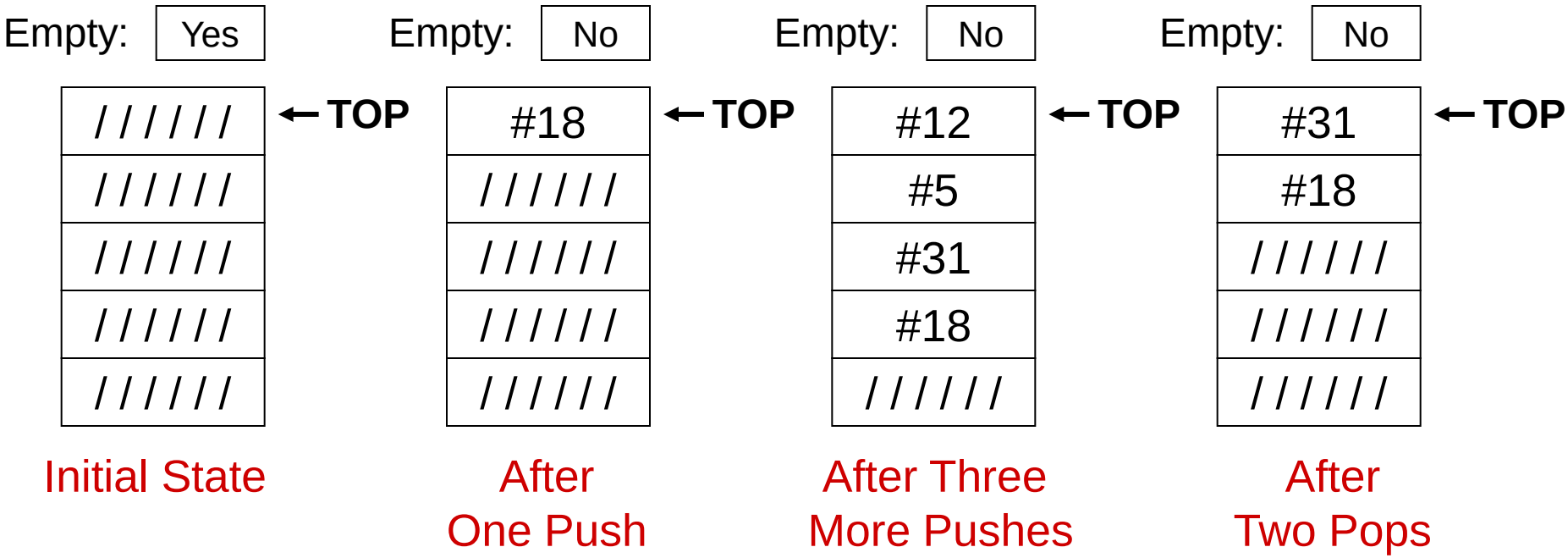


After
One Pop

First quarter out is the last quarter in.

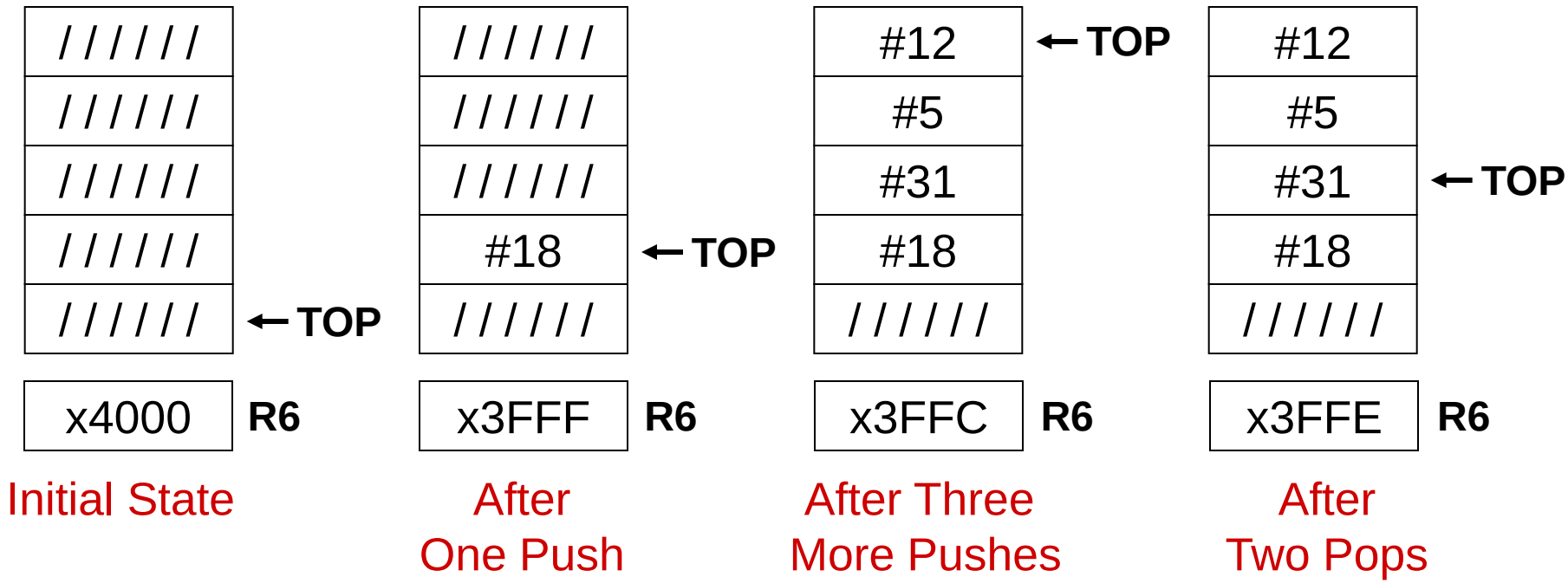
A Hardware Implementation

Data items move between registers



A Software Implementation

**Data items don't move in memory,
just our idea about where the TOP of the stack is.**



By convention, R6 holds the Top of Stack (TOS) pointer.

Basic Push and Pop Code

For our implementation, stack grows downward (when item added, TOS moves closer to 0)

Push

```
ADD   R6, R6, #-1 ; decrement stack ptr
STR   R0, R6, #0  ; store data (R0)
```

Pop

```
LDR   R0, R6, #0  ; load data from TOS
ADD   R6, R6, #1  ; increment stack ptr
```

Pop with Underflow Detection

If we try to pop too many items off the stack, an underflow condition occurs.

- Check for underflow by checking TOS before removing data.
- Return status code in R5 (0 for success, 1 for underflow)

```
POP    LD   R1, EMPTY   ; EMPTY = -x4000
       ADD  R2, R6, R1  ; Compare stack pointer
       BRz  FAIL        ; with xC000 (= -x4000)
       LDR  R0, R6, #0
       ADD  R6, R6, #1
       AND  R5, R5, #0  ; SUCCESS: R5 = 0
       RET
FAIL   AND  R5, R5, #0  ; FAIL: R5 = 1
       ADD  R5, R5, #1
       RET
EMPTY  .FILL xC000
```

Push with Overflow Detection

If we try to push too many items onto the stack, an overflow condition occurs.

- **Check for overflow by checking TOS before adding data.**
- **Return status code in R5 (0 for success, 1 for overflow)**

```
PUSH   LD   R1, MAX      ; MAX = -x3FFB
       ADD  R2, R6, R1   ; Compare stack pointer
       BRz  FAIL         ; with xC005 (=-x3FFB)
       ADD  R6, R6, #-1
       STR  R0, R6, #0
       AND  R5, R5, #0   ; SUCCESS: R5 = 0
       RET
FAIL   AND  R5, R5, #0   ; FAIL: R5 = 1
       ADD  R5, R5, #1
       RET
MAX    .FILL xC005
```

Interrupt-Driven I/O (Part 2)

Interrupts were introduced in Chapter 8.

1. **External device raises an interrupt when it needs attention.**
2. **Processor saves state and starts service routine.**
3. **When finished, processor restores state and resumes program.**

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

Chapter 8 didn't explain how (2) and (3) occur, because it involves a stack.

Now, we're ready…

Processor State

What state is needed to completely capture the state of a running process?

Memory

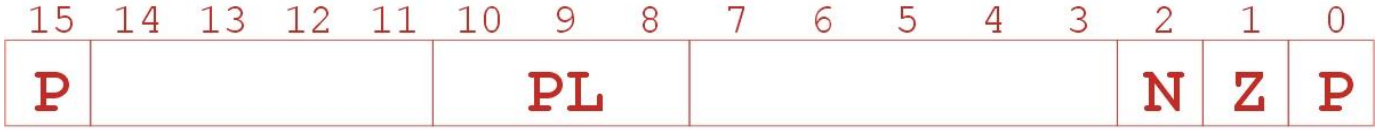
Program Counter

- **Pointer to next instruction to be executed.**

General-Purpose Registers (R0~R7)

Processor Status Register

- **Privilege [15], Priority Level [10:8], Condition Codes [2:0]**



P = 0: Supervisor Mode

P = 1: User Mode

Miscellaneous Registers: Saved.SSP, Saved.USP (will see next), etc

Supervisor Stack

A special region of memory is 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.

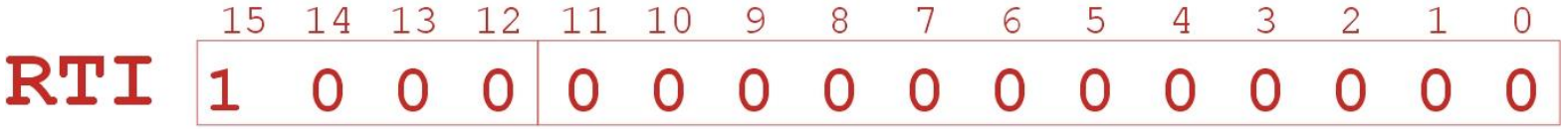
Invoking the Service Routine – The Details

1. If Priv = 1 (user),
   Saved.USP ← R6; 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 (i.e., atomic)

Returning from Interrupt

Special instruction – RTI – that restores state.



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** (if previous mode is USER mode)**, R6 = Saved.USP.**
   (If going back to user mode, need to restore User Stack Pointer.)

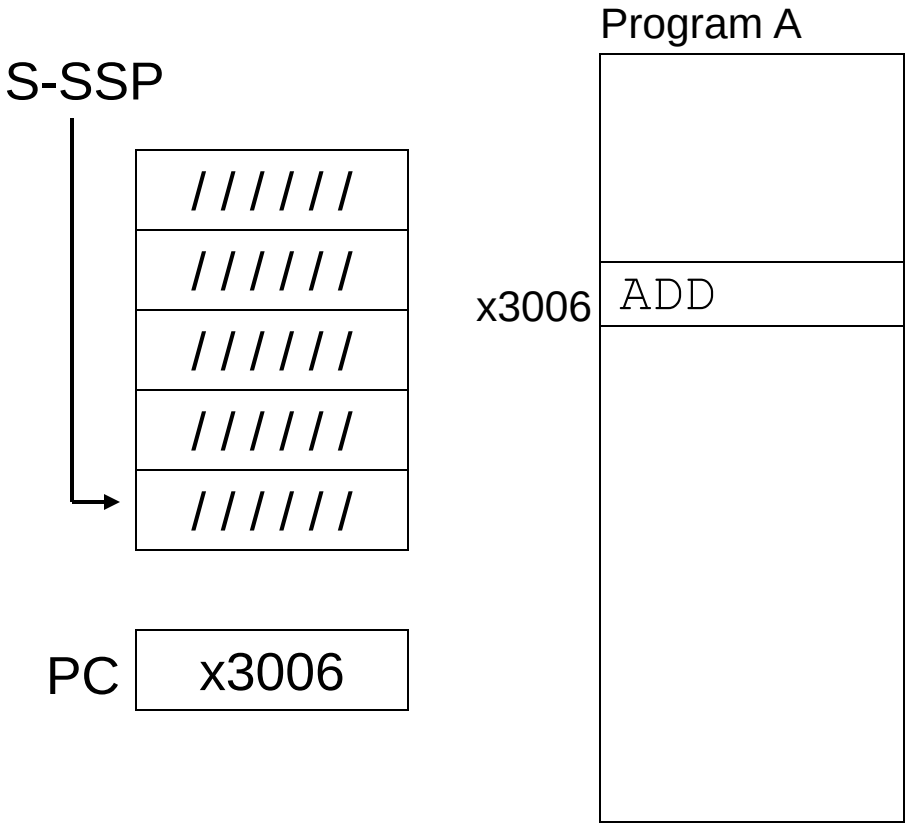
RTI is a privileged instruction.

- **Can only be executed in Supervisor Mode.**
- **If executed in User Mode, causes an exception.**
  (More about that later.)

Example (1)

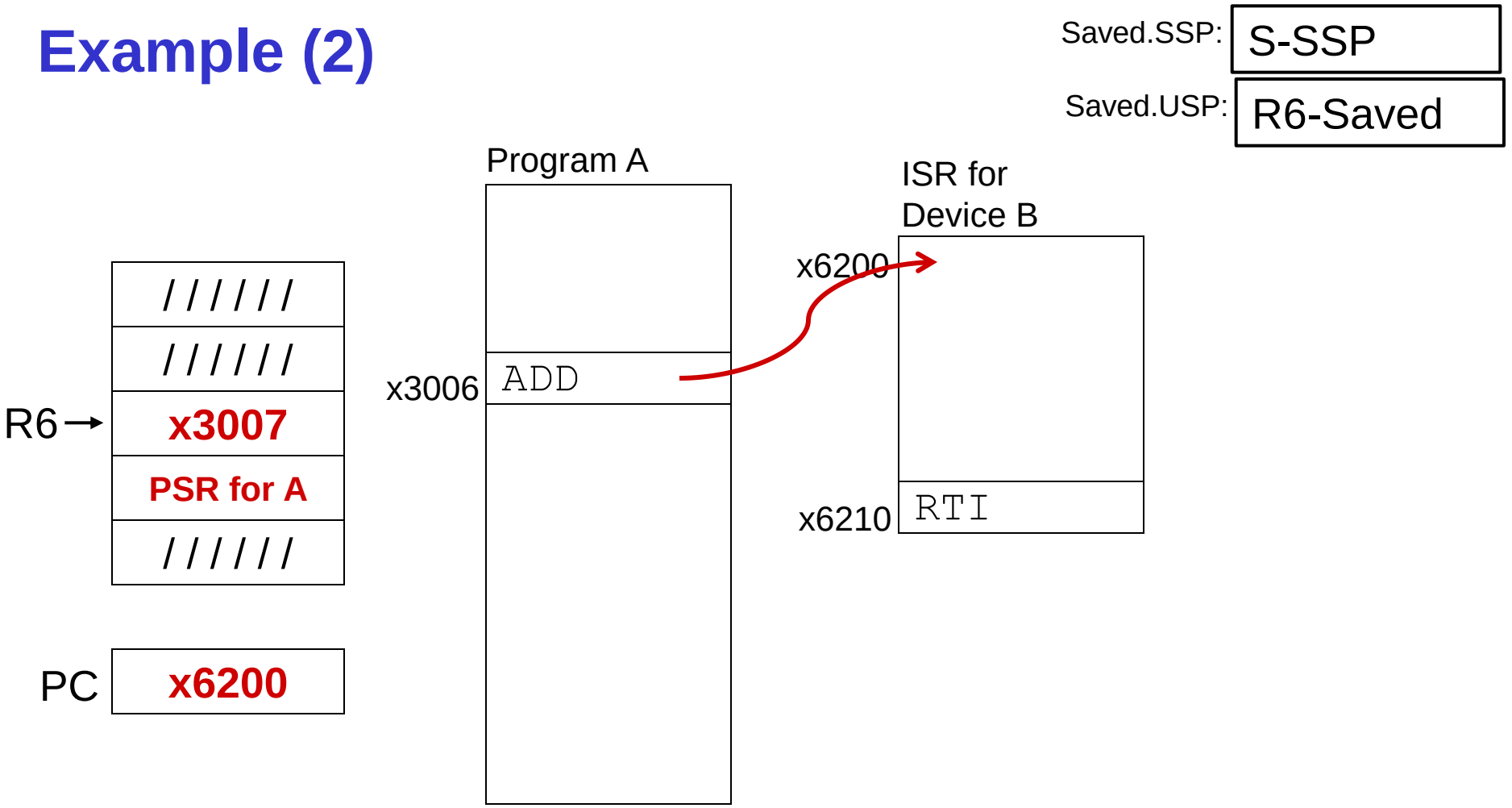
Saved.SSP: S-SSP

Saved.USP: ????



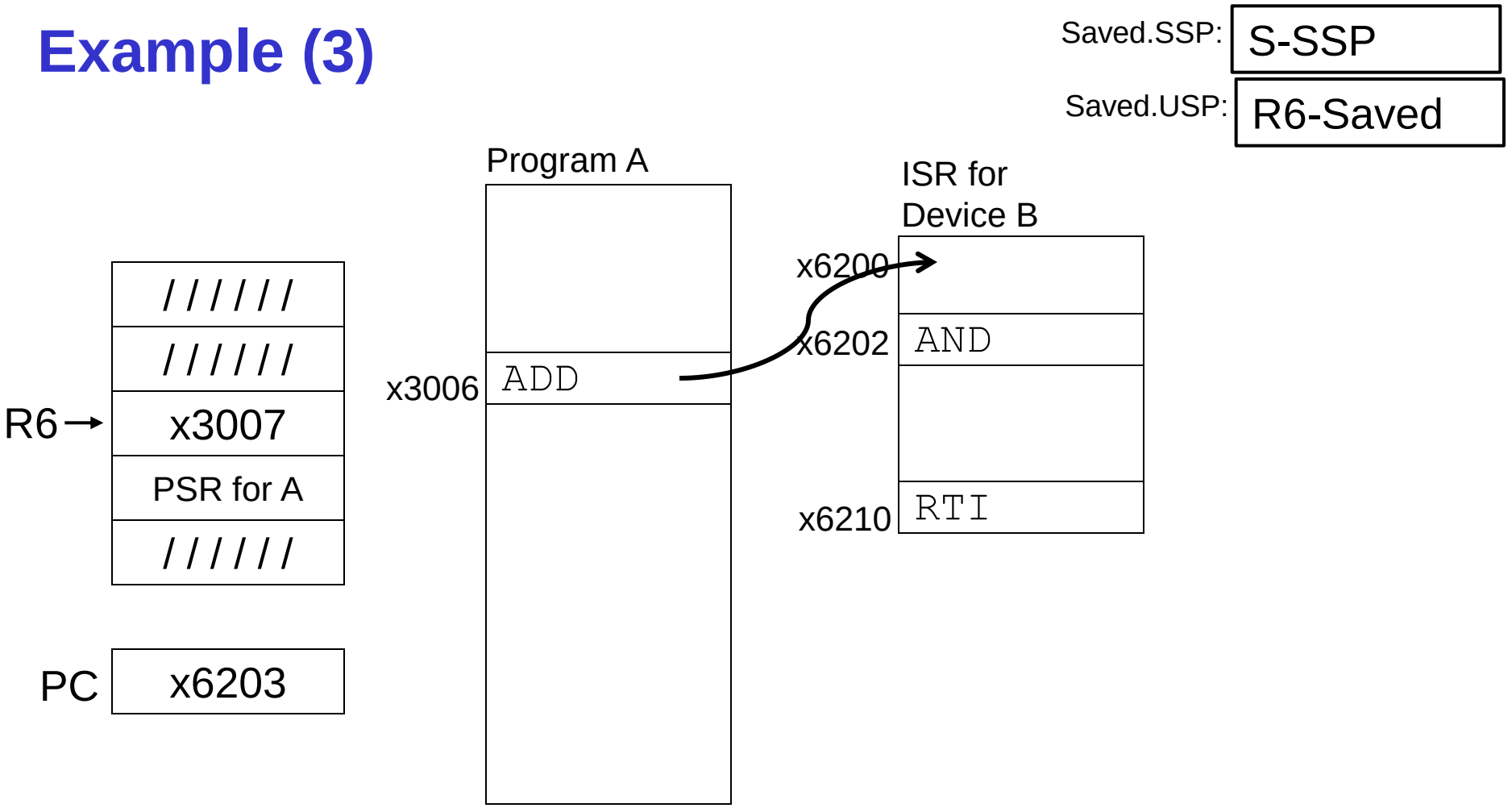
Executing ADD at location x3006 when Device B interrupts.

Example (2)



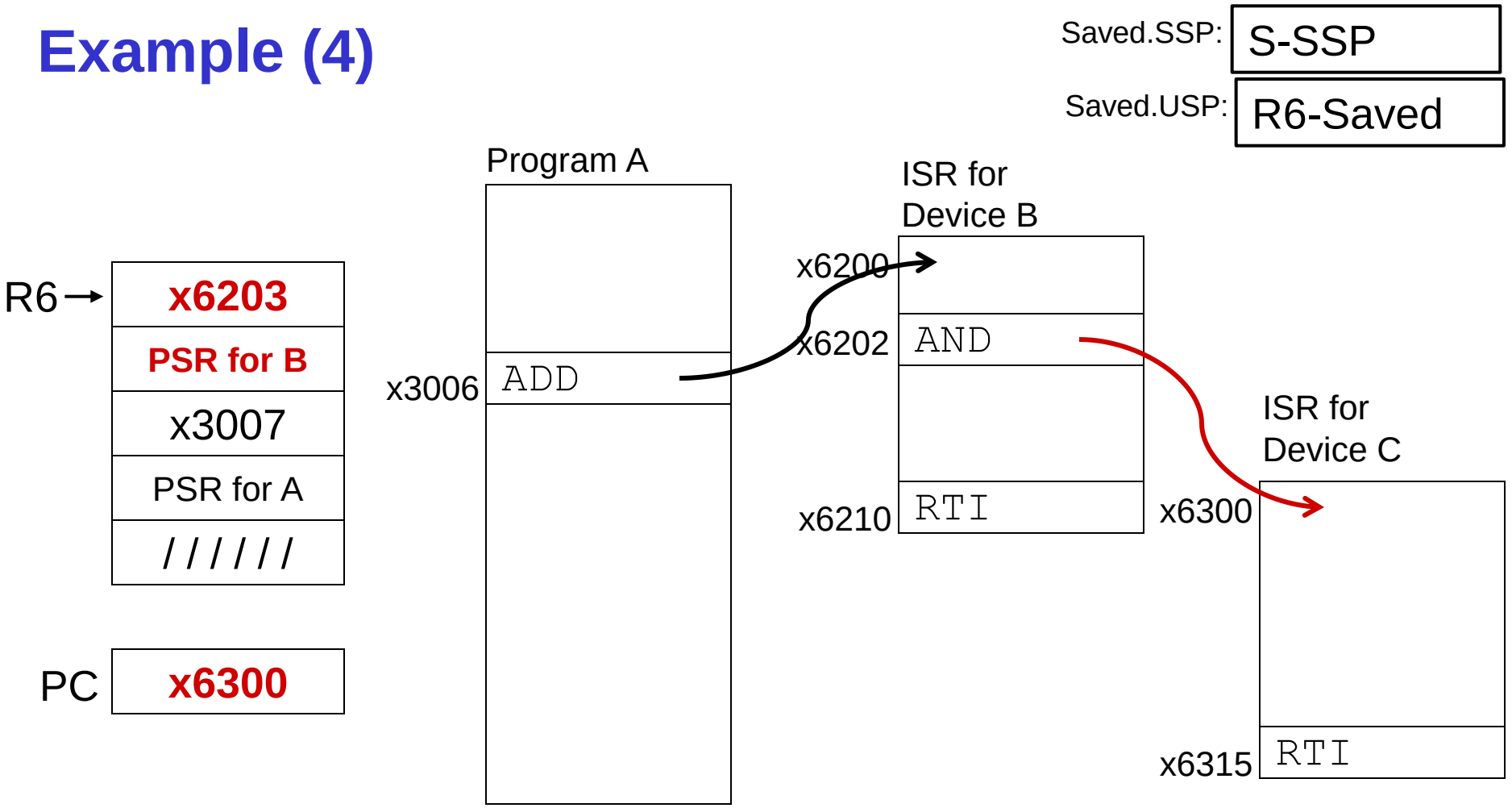
Saved.USP = R6. R6 = Saved.SSP.
Push PSR and PC onto stack, then transfer to Device B service routine (at x6200).

Example (3)



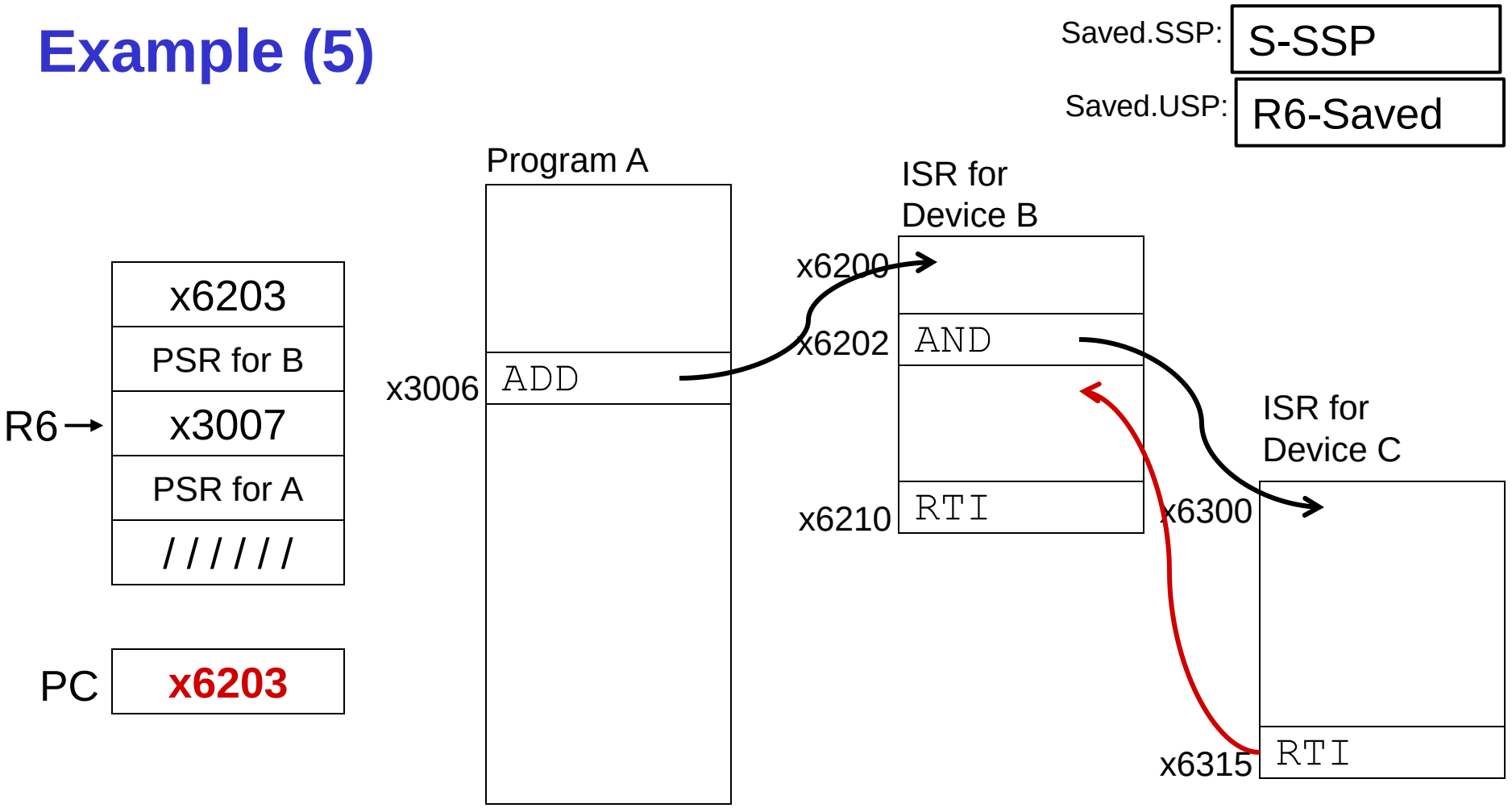
Executing AND at x6202 when Device C interrupts.

Example (4)



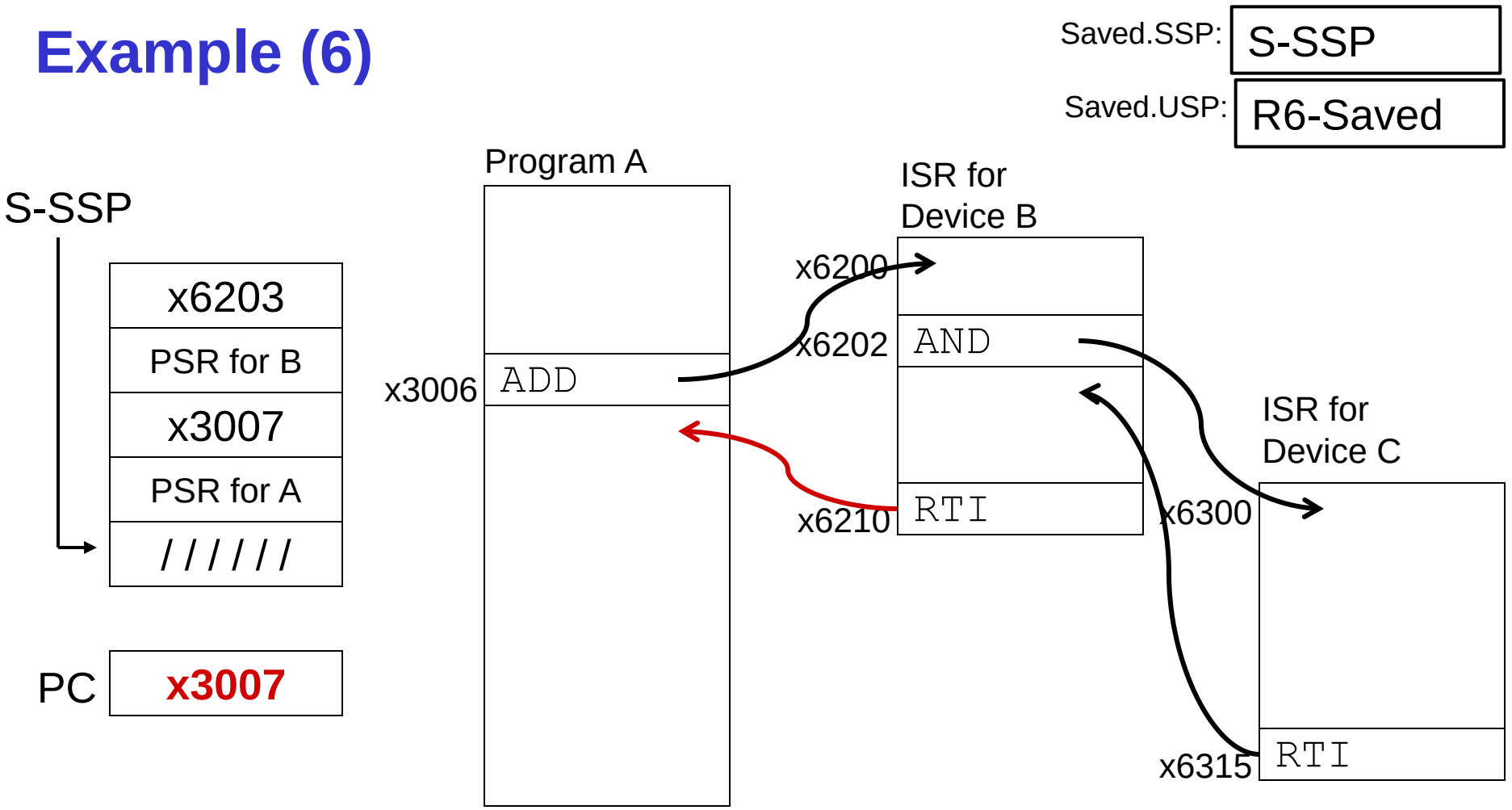
Push PSR and PC onto stack, then transfer to Device C service routine (at x6300).

Example (5)



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

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:

- **Executing an illegal opcode (e.g., 1101)**
- **Divide by zero**
- **Accessing an illegal address**
- **Executing RTI in the User mode**

Handled just like an interrupt

- **Vector is determined internally by type of exception**
- **Priority is the same as the program that caused the exception**

꼭 기억해야 할 것

- Stack: An Abstract Data Type
  - Push
  - Pop
  - IsEmpty
- Interrupt Service Routine
  - Invoked in response to an interrupt
  - States of the program being interrupted are saved to / restored from the Supervisor Mode Stack
  - We will see the use of the User Mode Stack soon
- Mechanisms to get attention from the operating system (OS)
  - System calls (via TRAP instruction)
  - Interrupts
  - Exceptions (processed in the same way as interrupts)