ECE 2300
Digital Logic & Computer Organization

Fall 2016

Exceptions

Cornell University

Memory Protection

- Address space of each *memory resident* program must be protected from undesired access

- Special *User/Supervisor* bit is implemented within the processor

- U/S bit must be 1 to unlock additional privileges
  - Access to the PTR within the processor
  - Access to the page tables and TLB
  - Access to the U/S bit
  - Special instructions to manipulate this information

Memory Protection

- The U/S bit gets set when a program executes a *system call* (*syscall*) instruction
  - Open a file, read a keystroke, write the screen, ...
- A syscall creates an *exception* that kicks out the user program and transfers control to an OS *service routine*

Exception

- A method for signaling the CPU that some event has occurred that requires action

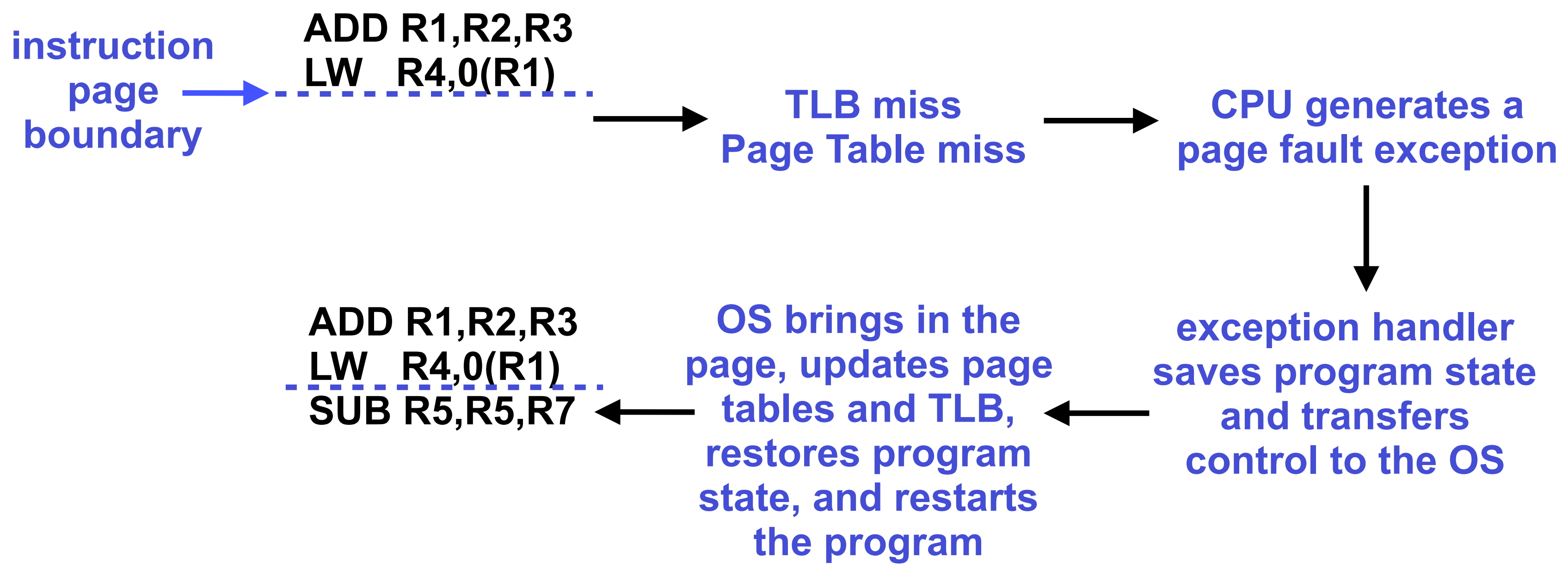
- In response, the CPU may suspend the running program in order to handle the exception

- <u>Interrupt</u>: An exception from an external source
  - I/O device request
  - External error or malfunction

Why are Exceptions Useful?

- Allow user programs to get service from the OS
- Allow I/O devices to signal the CPU
- Handle unexpected events
  - Memory protection violation, parity error, power supply failure, etc
- Handle page faults

Handling a Page Fault



```
ADD R1,R2,R3
LW  R4,0(R1)
```

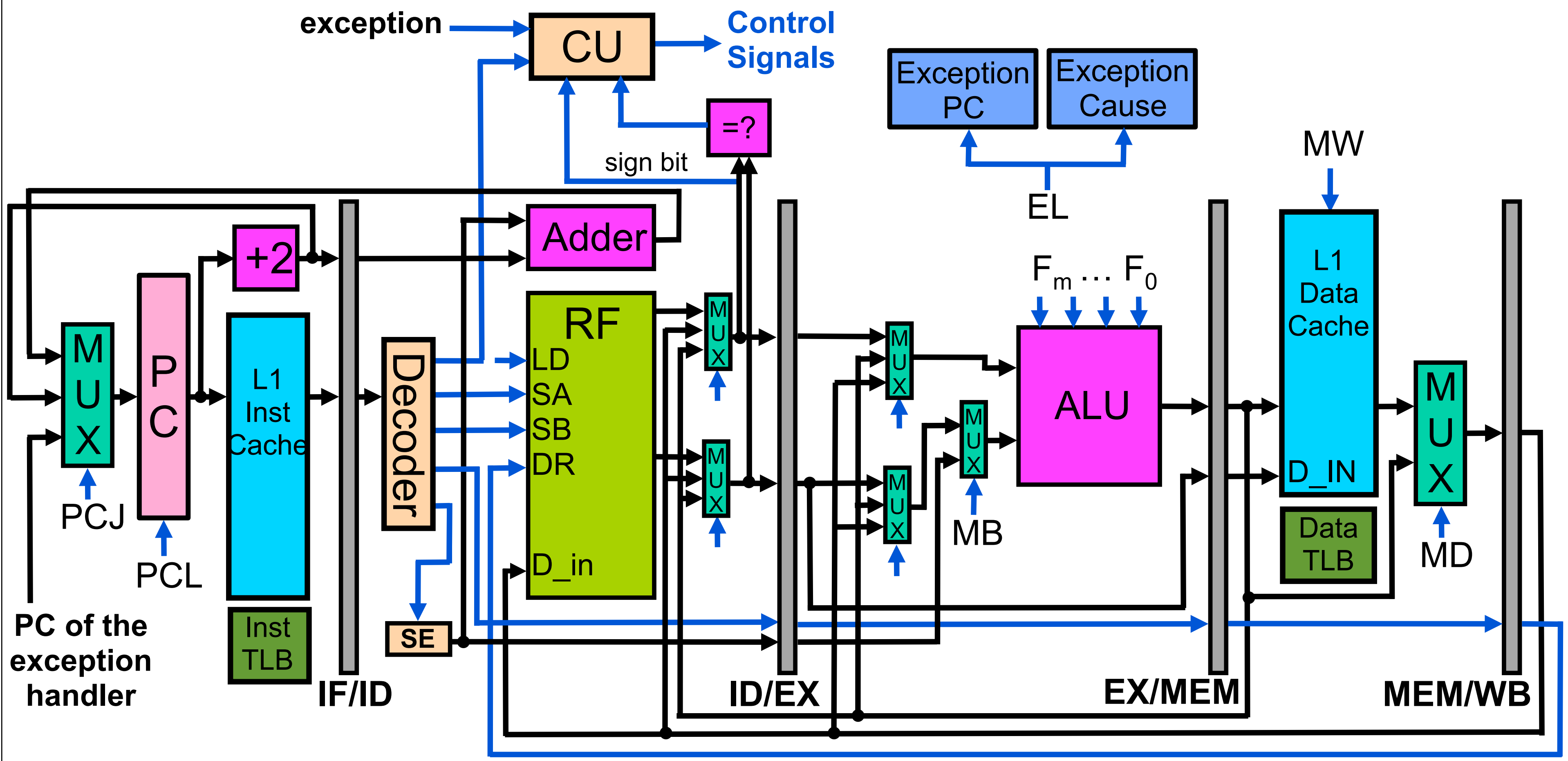
```
ADD R1,R2,R3
LW  R4,0(R1)
SUB R5,R5,R7
```

instruction page boundary
TLB miss
Page Table miss
CPU generates a page fault exception
exception handler saves program state and transfers control to the OS
OS brings in the page, updates page tables and TLB, restores program state, and restarts the program

Key Steps and Requirements

- Generate an exception, enter supervisor mode
- Save program state (registers) to memory
- Identify the type of exception
- Load the software routine for handling this exception type
- Handle the exception
- Restore program state, switch to user mode, and restart at the faulting instruction

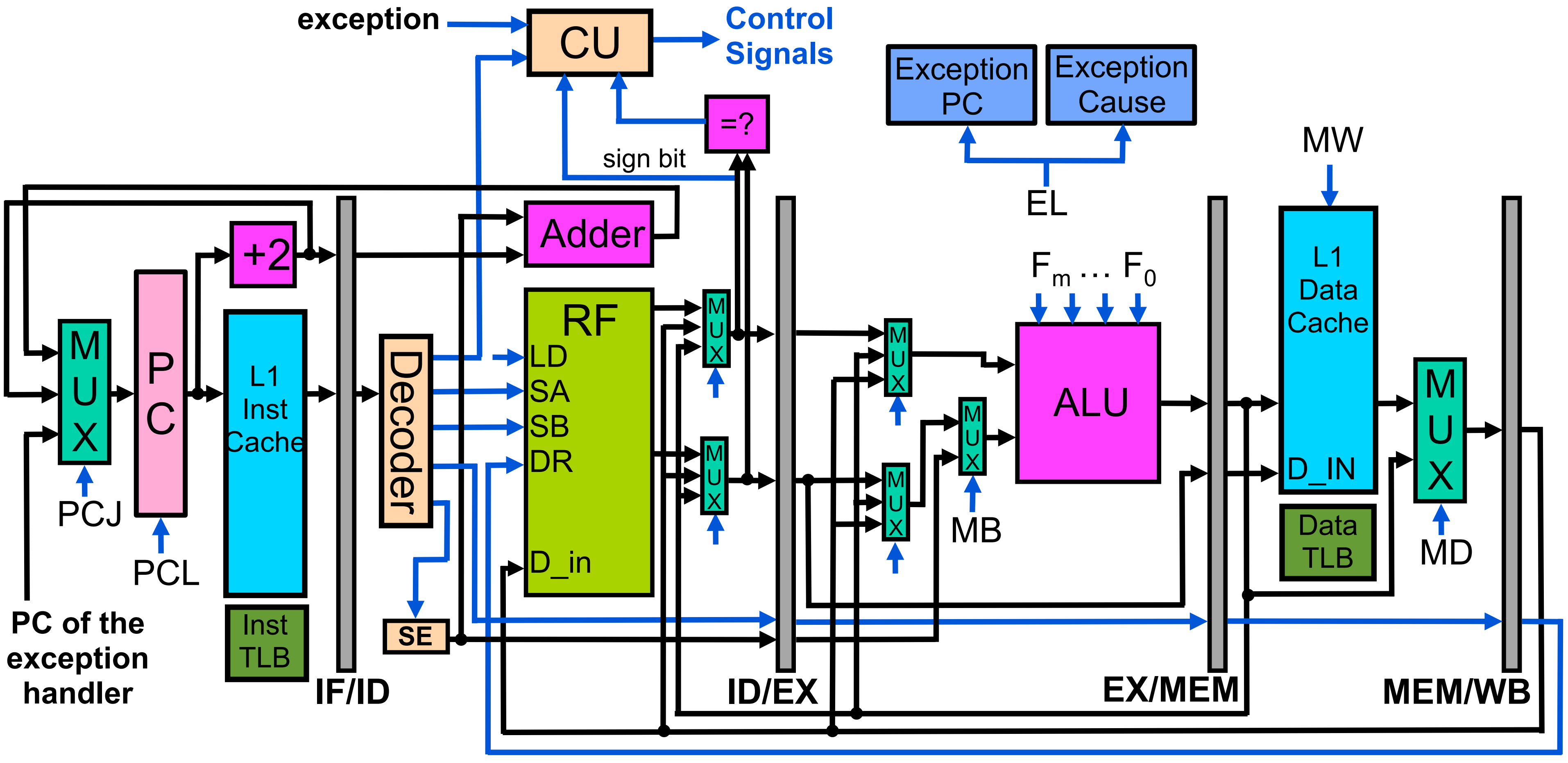
Pipeline with Exception Handling



Pipeline with Exception Handling

- An Exception signal is sent to the CU
- The Control Unit loads the Exception PC and Cause
- All instructions before the exception complete
- The faulting instruction, and any behind it in the pipeline, are turned into NOPs
- The PC of the first instruction in the exception handler code is loaded into the PC register

Instruction Page Fault

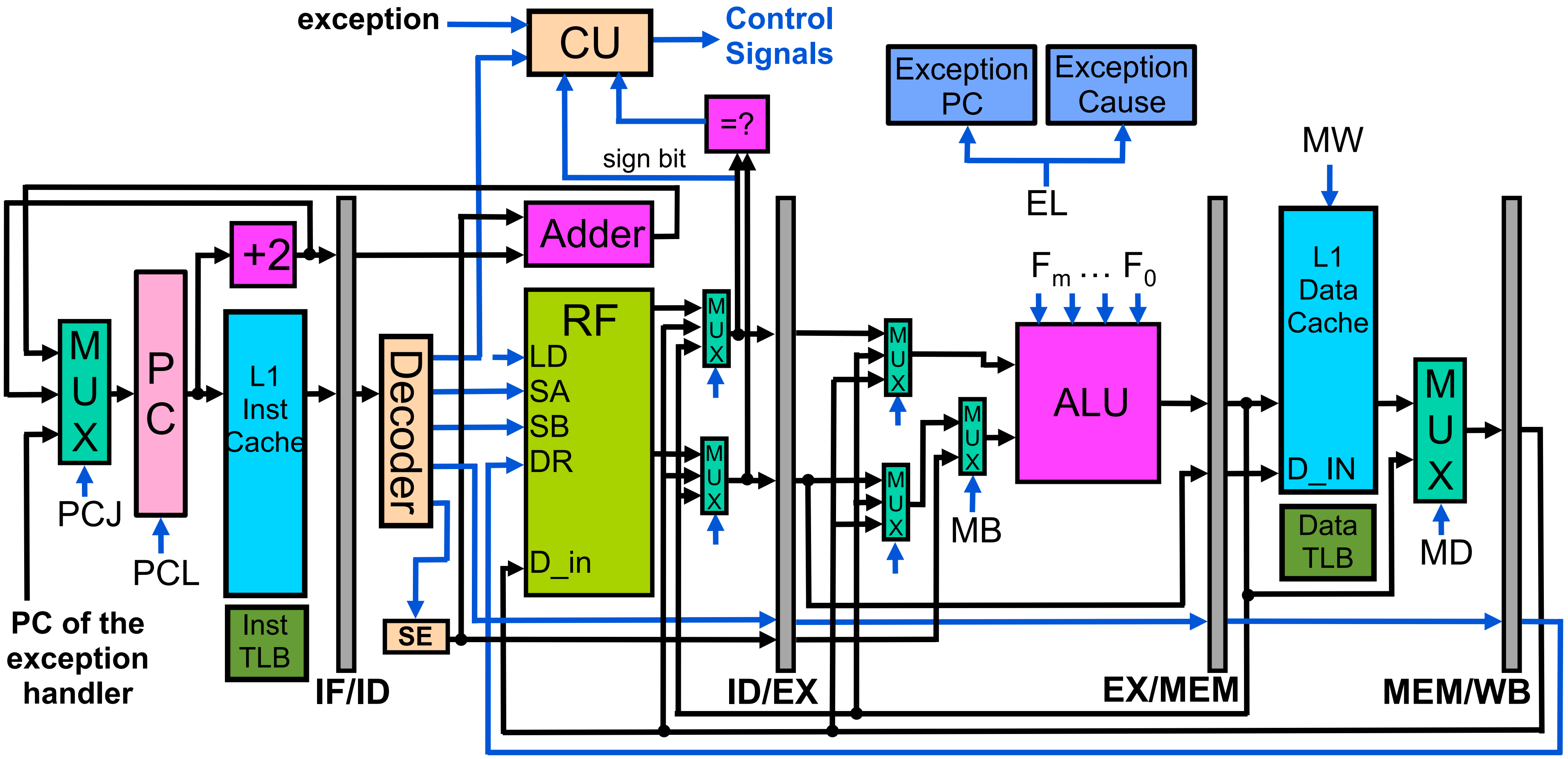


[SUB R5,R5,R7]
<not in memory>

LW R4,0(R1)

ADD R1,R2,R3

Instruction Page Fault

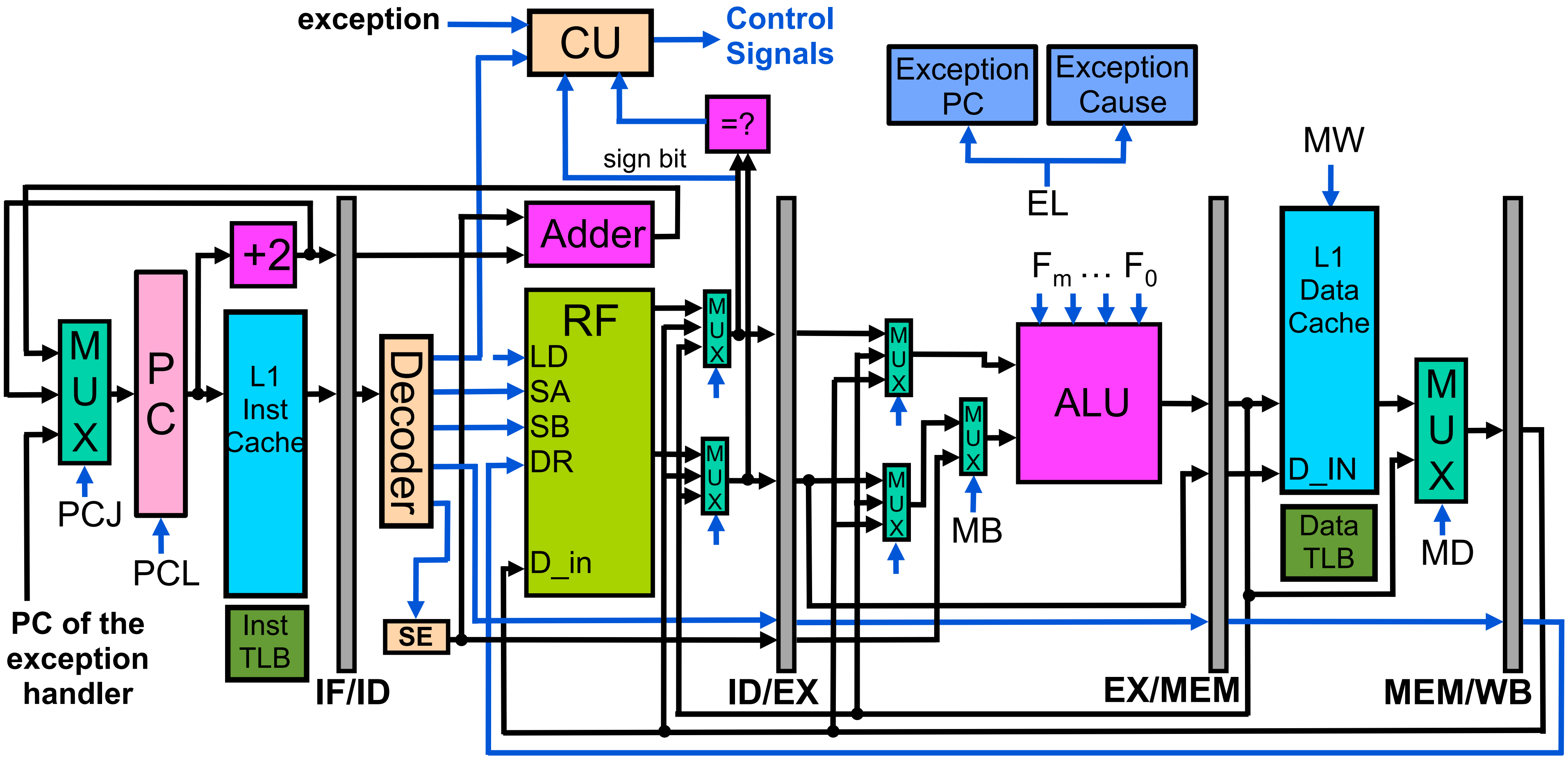


[SUB R5,R5,R7]
<TLB miss>

LW R4,0(R1)

ADD R1,R2,R3

Instruction Page Fault



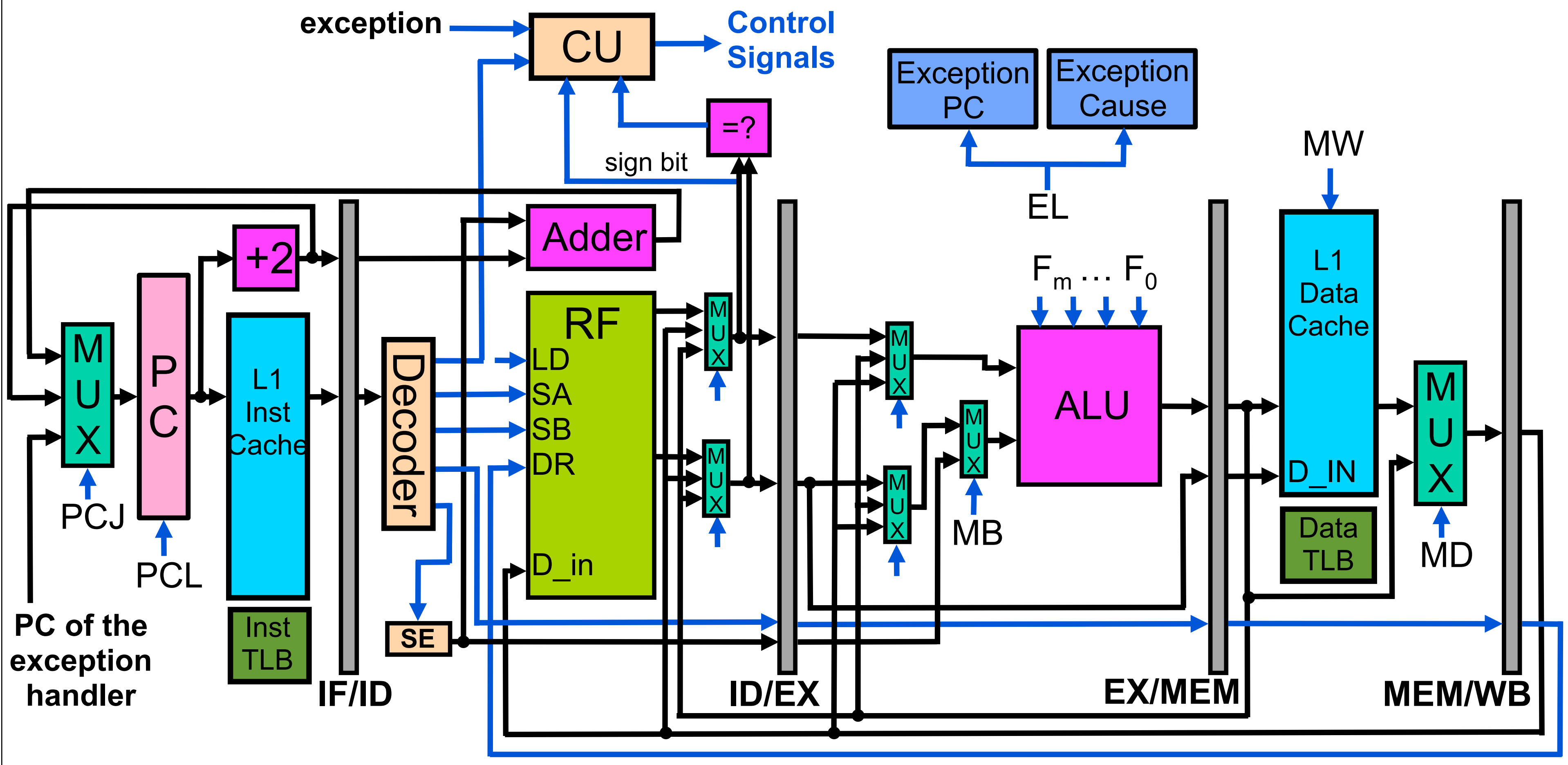
[SUB R5,R5,R7]

<stall>

<access page table>

<LW and ADD have completed>

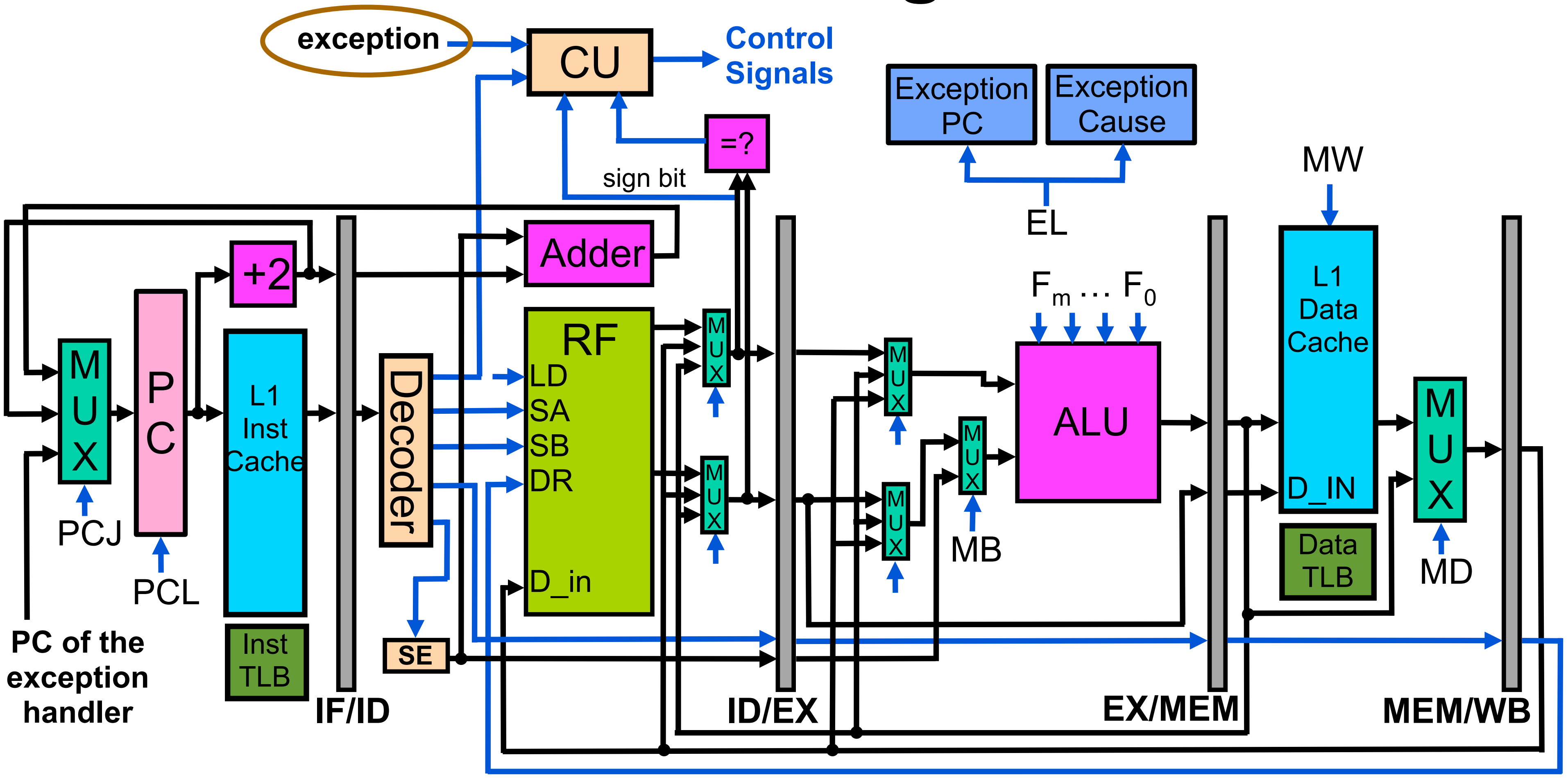
Instruction Page Fault



[SUB R5,R5,R7]

<page fault>

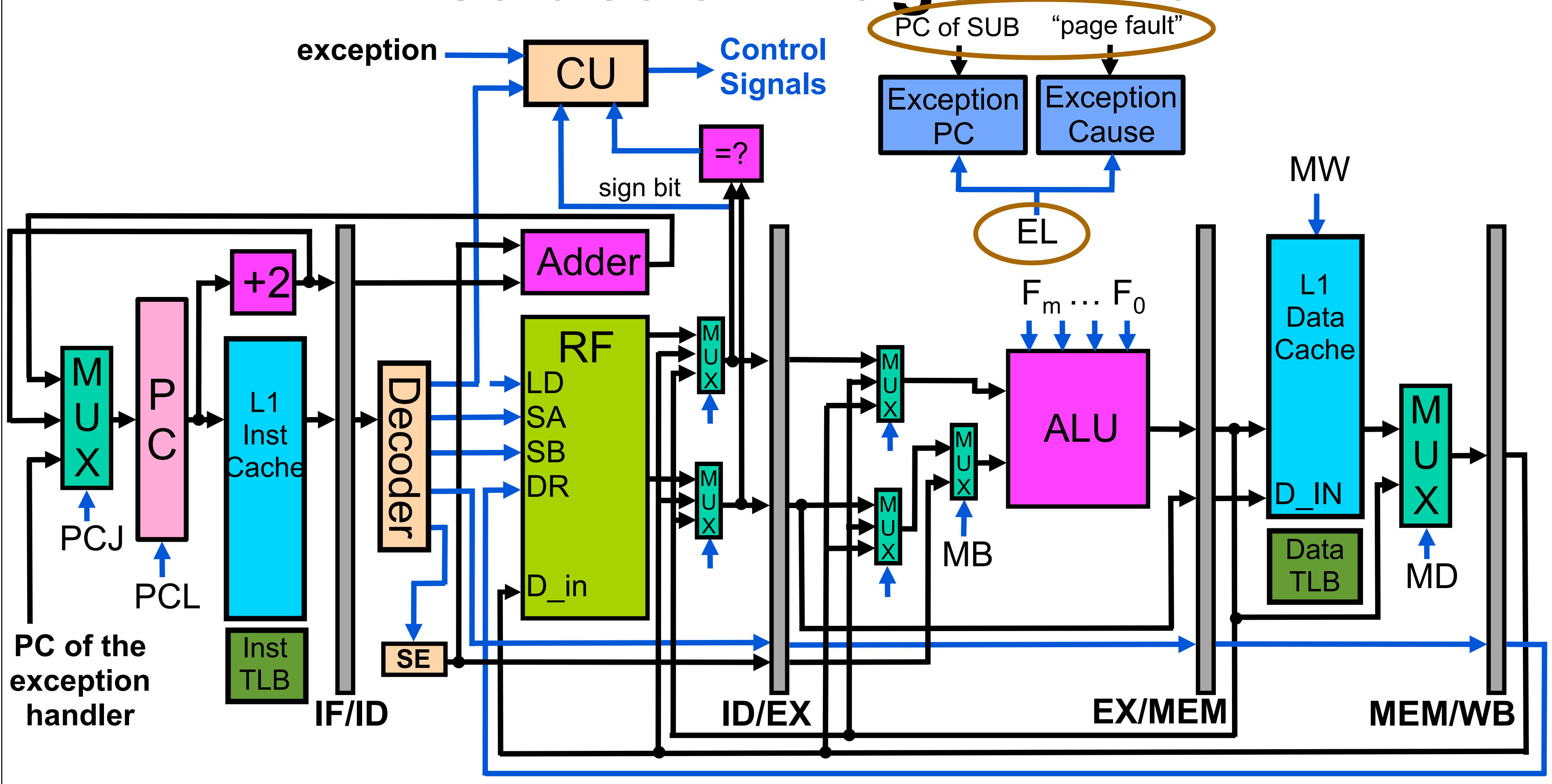
Instruction Page Fault



[SUB R5,R5,R7]

<page fault>

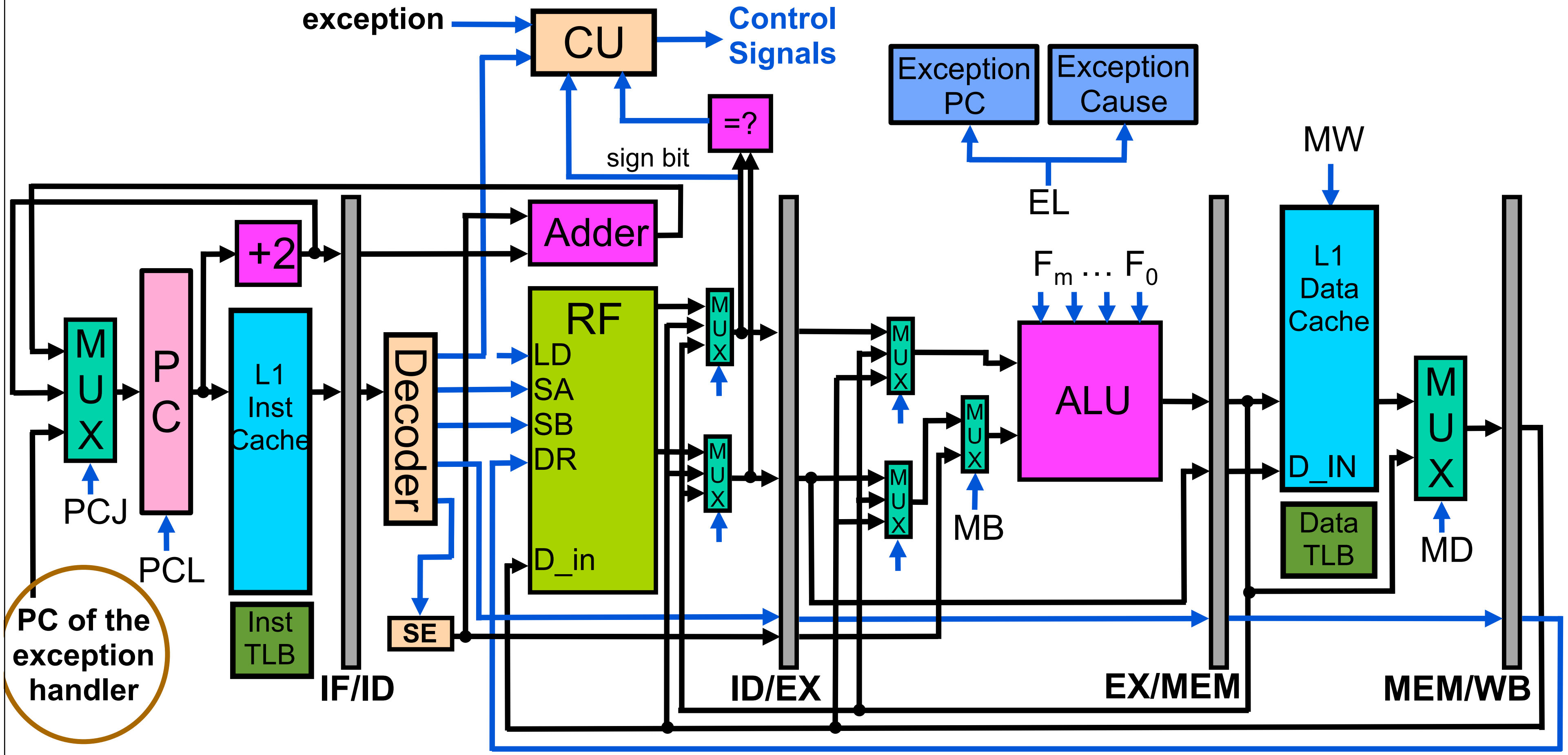
Instruction Page Fault



[SUB R5,R5,R7]

<page fault>

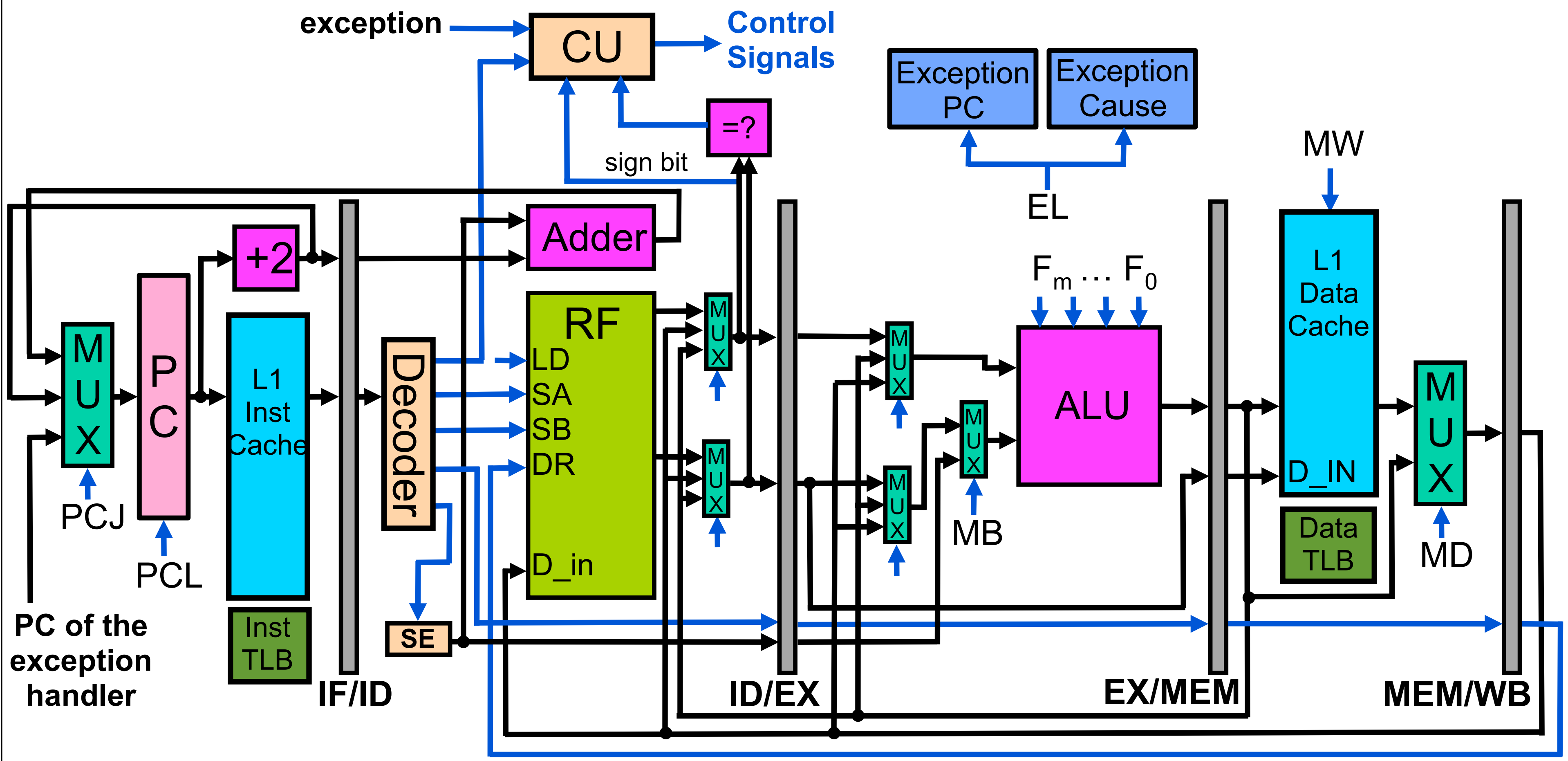
Instruction Page Fault



[SUB R5,R5,R7]

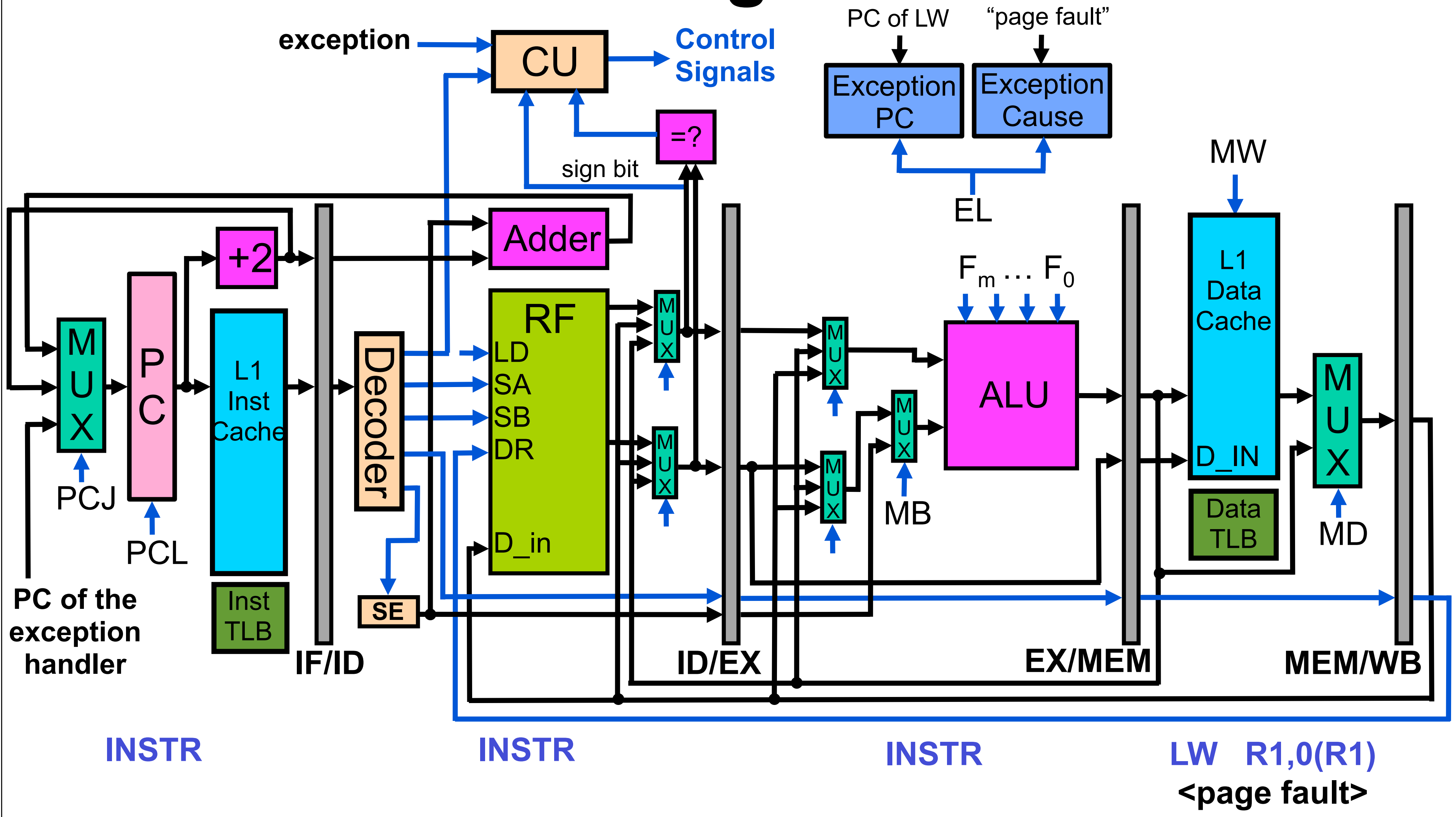
<page fault>

Instruction Page Fault

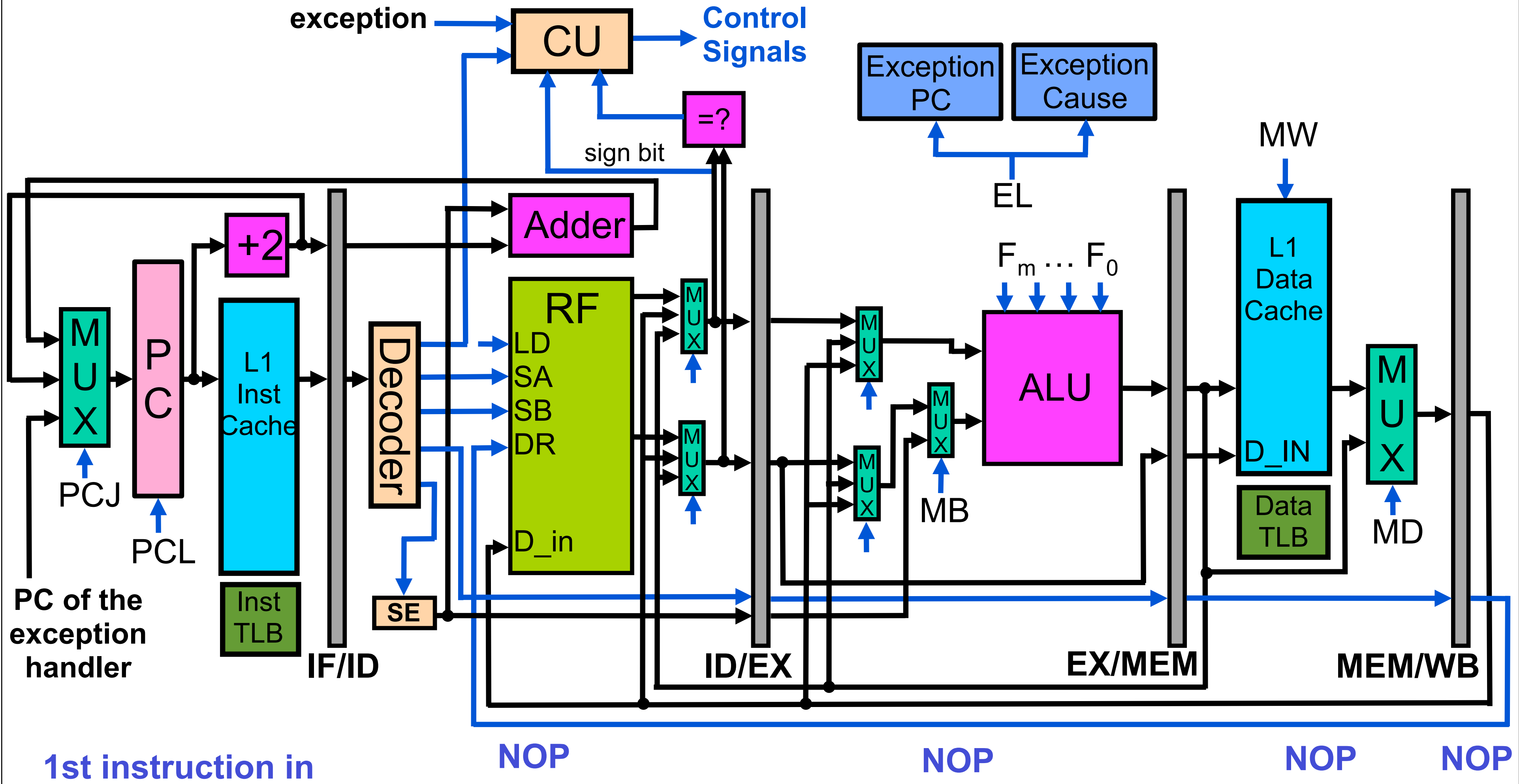


1st instruction in exception handler

Data Page Fault



Data Page Fault

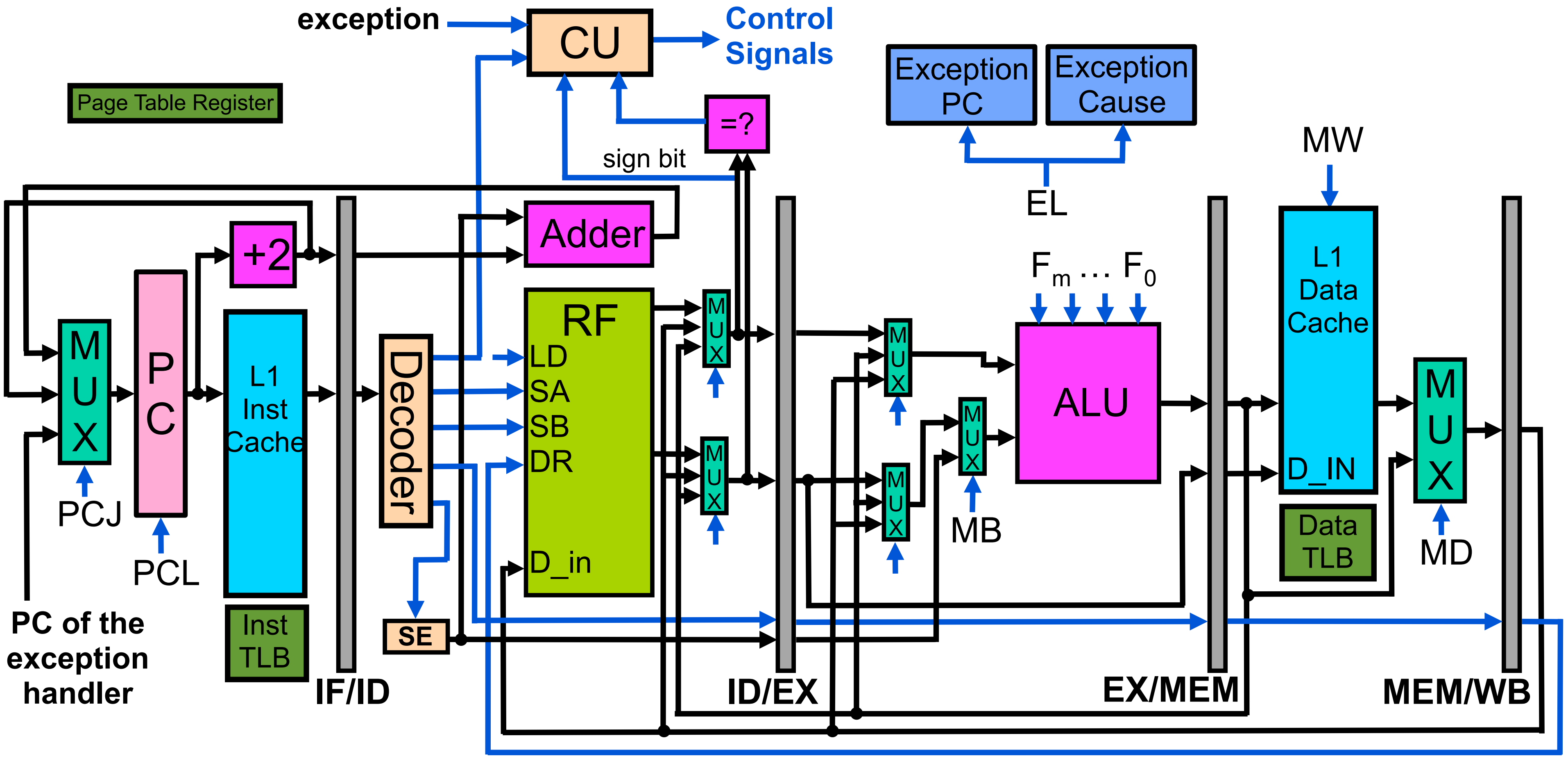


1st instruction in exception handler

Enabling Program Restart

- After the exception is handled, want to restart the program starting at the faulting instruction
- The program state (register values) is saved by the exception handler into memory, and restored when the exception has been handled
- All instructions before the faulting one complete (write their results) before the exception is handled
- The faulting instruction, and those behind it in the pipeline, are turned into NOPs

OS Switches from Task A to Task B



OS Switches from Task A to Task B

- What do we do about the registers?
- What do we do about the PC?
- What do we do about the Page Table Register?
- What do we do about the TLBs?
- What do we do about the caches?

Next Time

Input/Output