Announcements

- HW5 extended to Tue night
- HW ever > HW never

CSE 141: Introduction to Computer Architecture

Advanced Pipelines

Part I: Exceptions

- This is the last piece of what's needed to make a "real" CPU useful

Exceptions

- There are two sources of non-sequential control flow in a processor
  - explicit branch and jump instructions
  - exceptions
- *Branches* are synchronous and deterministic
- *Exceptions* are typically asynchronous and non-deterministic
- Guess which is more difficult to handle?

(recall: *control flow* refers to the movement of the program counter through memory)

Exceptions and Interrupts

signed 8-bit math

-128 ... 127

100
+100
???

The terminology is not consistent, but we'll refer to

- *exceptions* as any unexpected change in control flow
- *interrupts* as any externally-caused exception

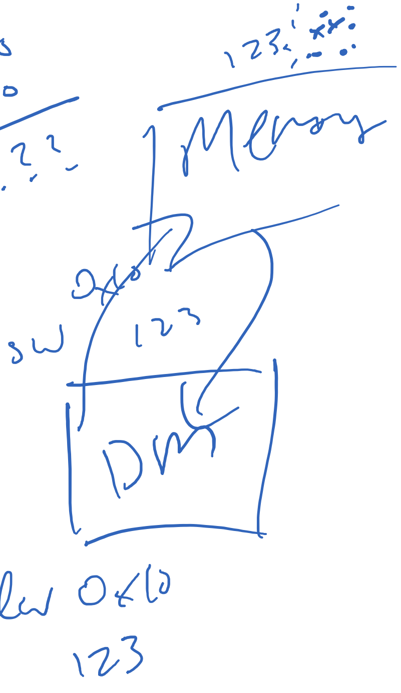
So then, what is:

- arithmetic overflow – exception
- divide by zero X
- I/O device signals completion to CPU i
- user program invokes the OS – ex
- memory parity error – i
- illegal instruction – ex
- timer signal – i

appl

add
sub
addi
arm svc

x86 int



ECC

For now…

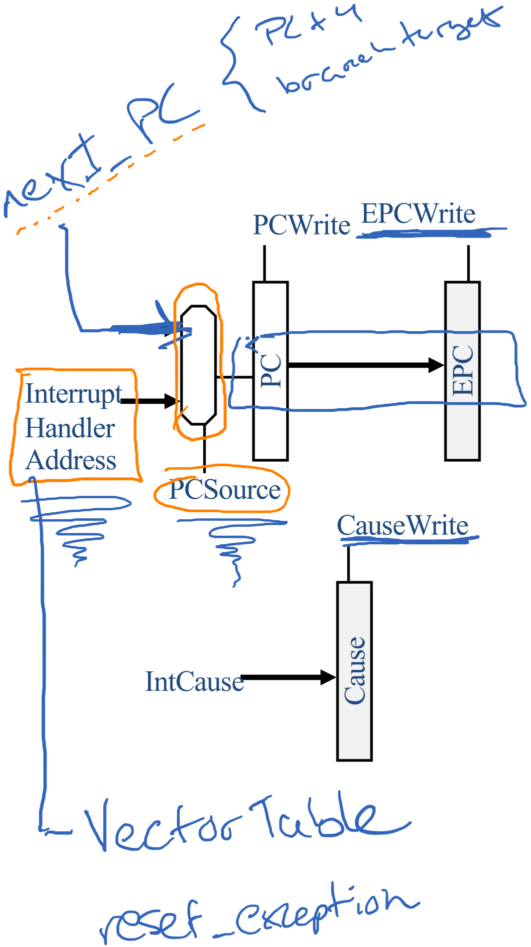
- The machine we've been designing in class can generate two types of exceptions.
  - overflow
  - illegel inst

For now…

- The machine we've been designing in class can generate two types of exceptions.
  - arithmetic overflow
  - illegal instruction
- On an exception, we need to
  - save the PC (invisible to user code)
  - record the nature of the exception/interrupt
  - transfer control to OS

First steps towards supporting exceptions

- For our MIPS-subset architecture, we will add two registers:
  - EPC: a 32-bit register to hold the user's PC
  - Cause: A register to record the cause of the exception
    - we'll assume undefined inst = 0, overflow = 1
- We will also add three control signals:
  - EPCWrite (will need to be able to subtract 4 from PC)
  - CauseWrite
  - IntCause
- We will extend PCSource multiplexor to be able to latch the interrupt handler address into the PC.



Pipelining and Exceptions

div

mul	F	D	X	M	W		
div		F	D	$\boxed{X}$	M	W	
~~add~~			~~F~~	~~D~~	X	M	W
~~sub~~				~~F~~	D	X	M	W
ex_addr: mov				F	D		

- Again, exceptions represent another form of control flow and therefore control dependence.
- Therefore, they create a potential branch hazard
- Exceptions must be recognized early enough in the pipeline that subsequent instructions can be flushed before they change any permanent state.
  - Q: What is the first stage that can change permanent state? ID X MEM
- We also have issues with handling exceptions in the correct order and "exceptions" on speculative instructions.
- Exception-handling that always correctly identifies the offending instruction is called *precise interrupts* ne exception
  - (different words, same idea: ARM has *asynchronous / synchronous exceptions*)

CS 148

Branch Prediction

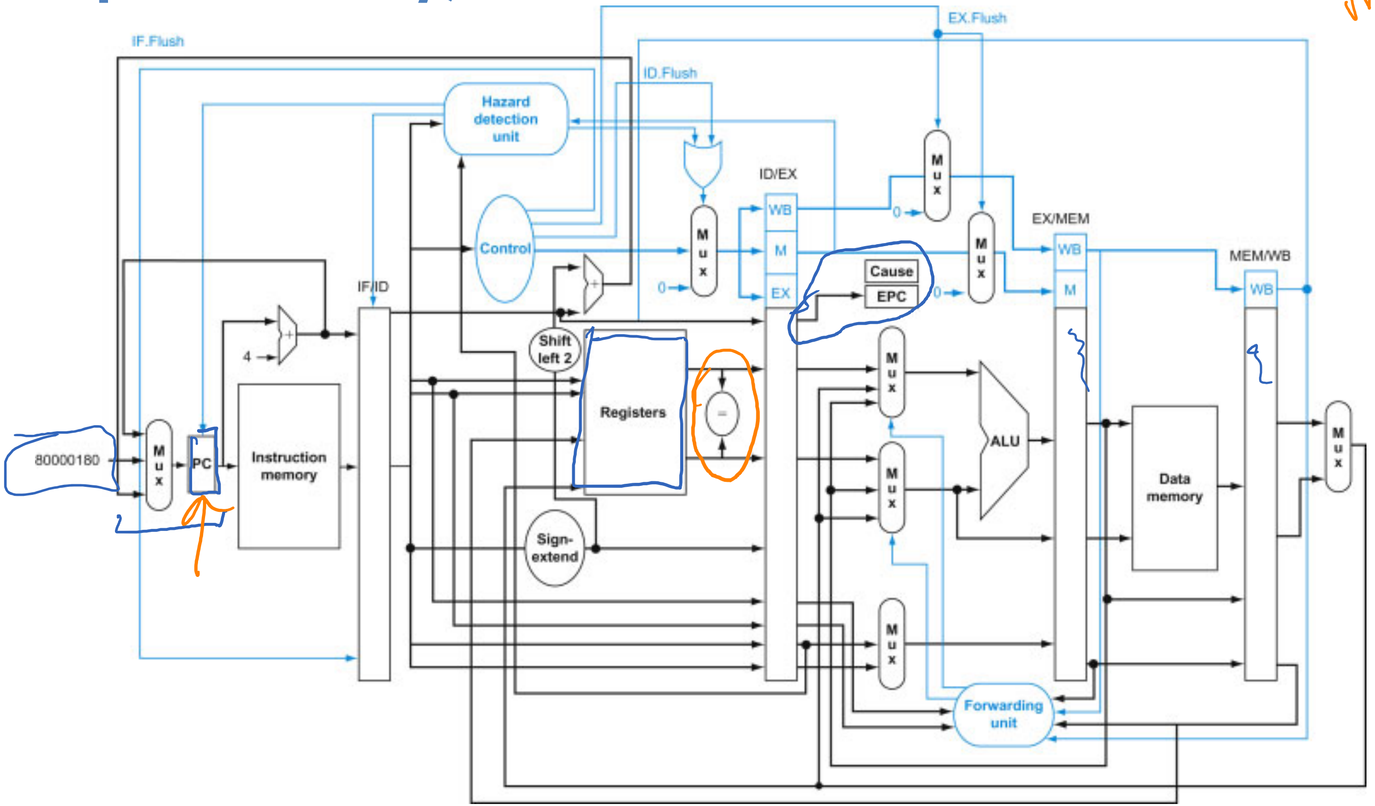
✓ add
✓ sub
✓ beq
and BDS
~~mov~~
~~sub~~
div
F D
F

ldm 0111001101

Pipelining and Exceptions – The Whole Picture (except not really)

Permanent State?

MIPS with early branch resolution


IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
IF/ID
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Cause
EPC
0
M u x
80000180
PC
4
Instruction memory
Shift left 2
Registers
=
Sign-extend
ALU
Data memory
Forwarding unit