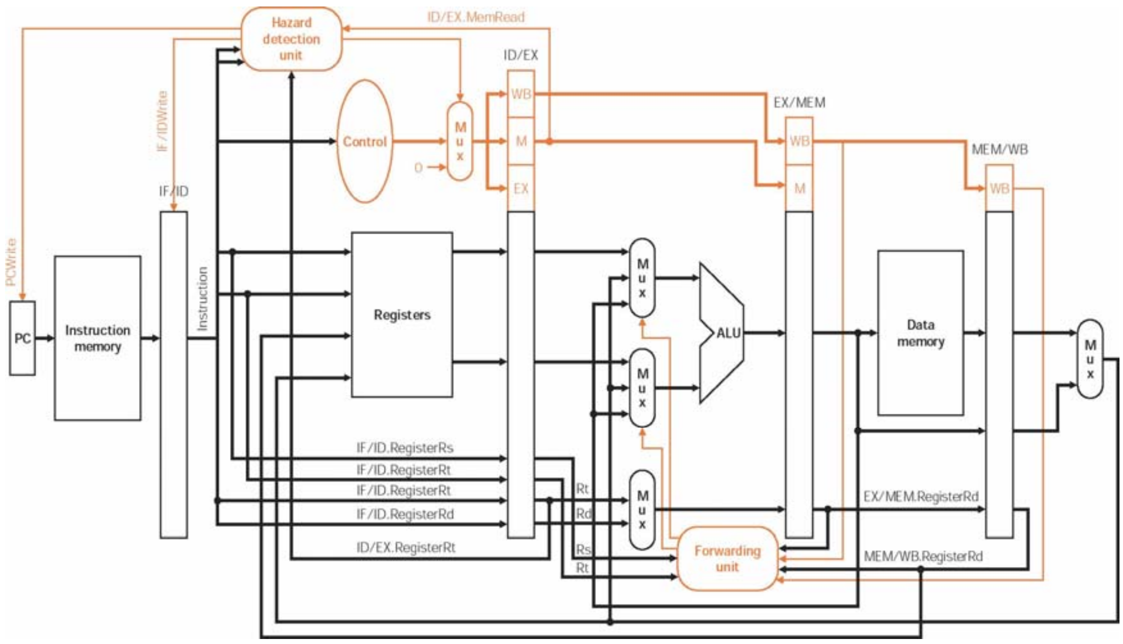
Chapter Four – II (4/5)

How to Stall the Pipeline

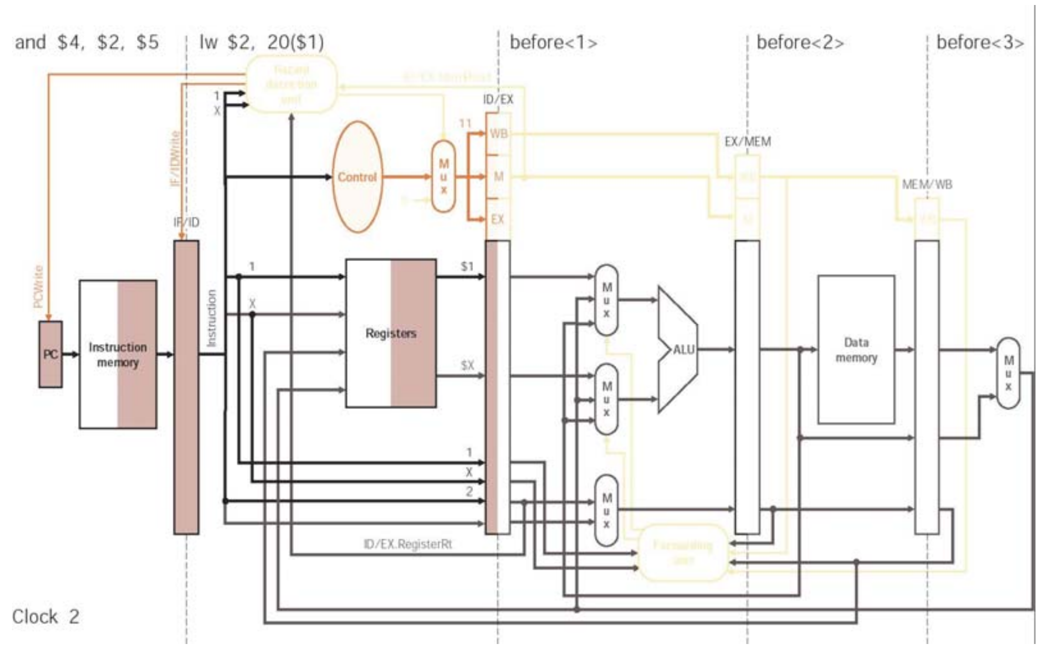
- **Force control values in ID/EX register to 0**
  - **EX, MEM and WB do** nop **(no-operation)**
- **Prevent update of PC and IF/ID register**
  - **Using instruction is decoded again**
  - **Following instruction is fetched again**
  - **1-cycle stall allows MEM to read data for `lw`**
    - Can subsequently forward to EX stage

Hazard Detection Unit

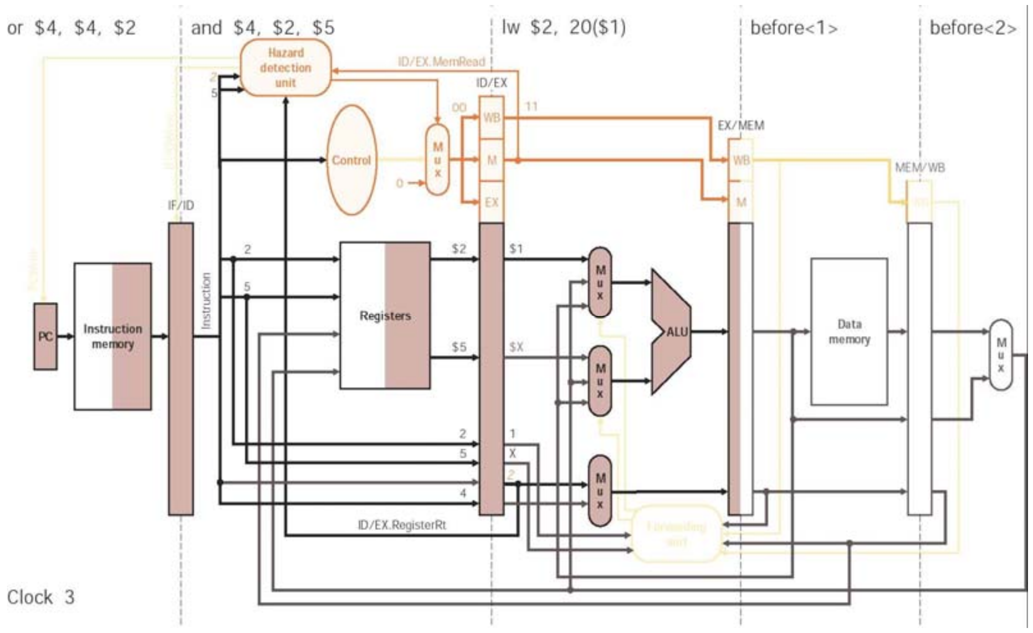
- **Stall by letting an instruction that won't write anything go forward**



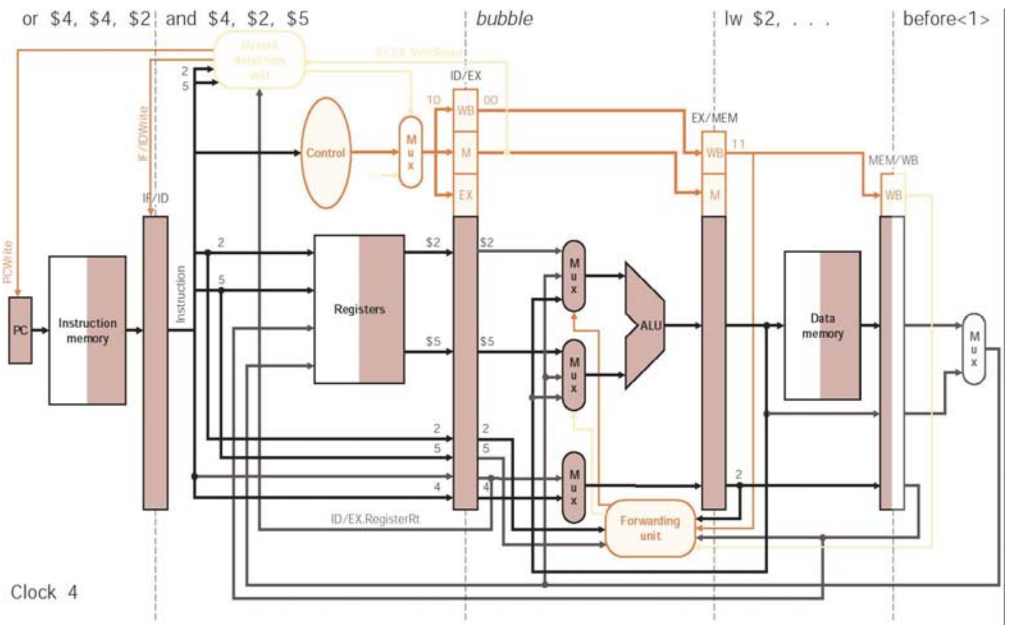
Modified Execution Example for Load replacing (C2)



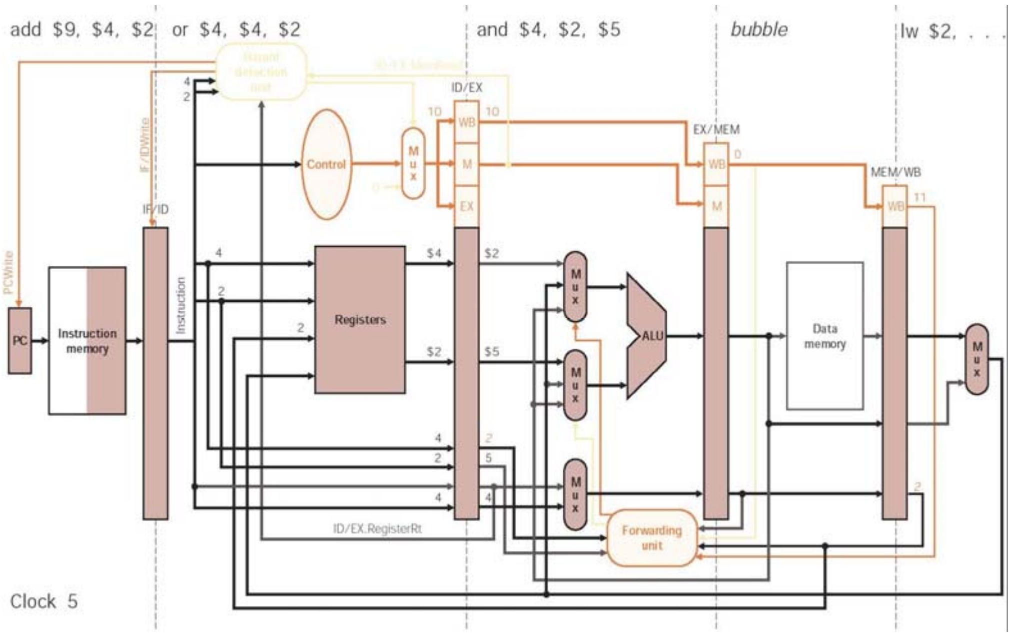
Modified Execution Example for Load replacing (C3)



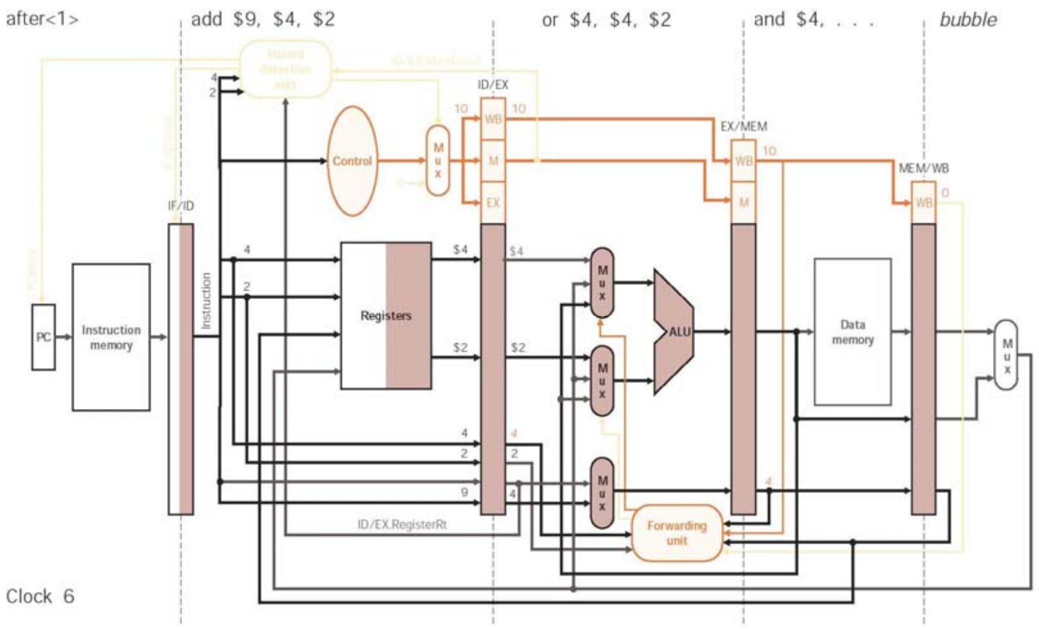
Modified Execution Example for Load replacing (C4)



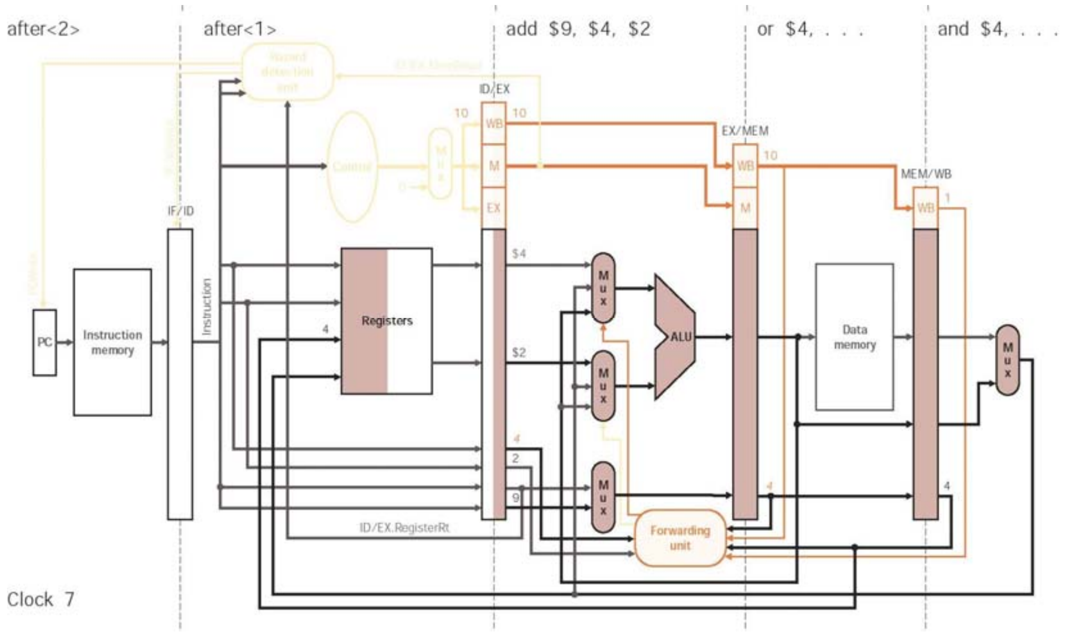
Modified Execution Example for Load replacing (C5)



Modified Execution Example for Load replacing (C6)

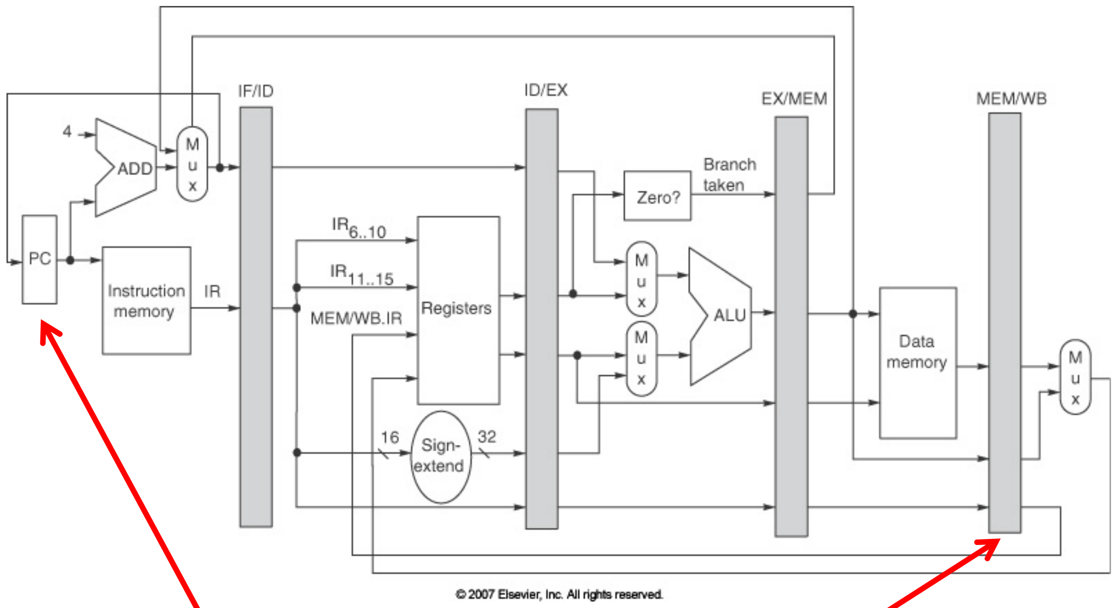


Modified Execution Example for Load replacing (C7)



CE, KWU Prof . S.W. LEE 9

Basic MIPS pipeline implementation

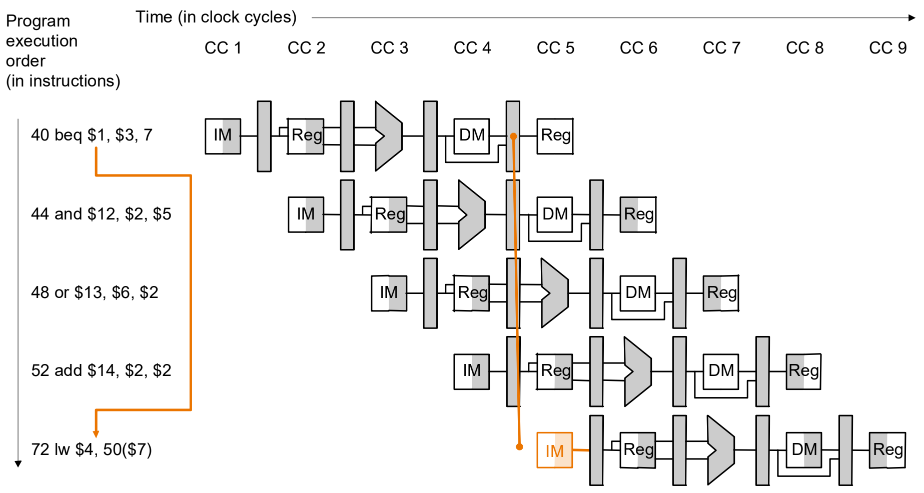


A branch instruction updates PC and MEM/WB at the same clock

CE, KWU Prof . S.W. LEE 10

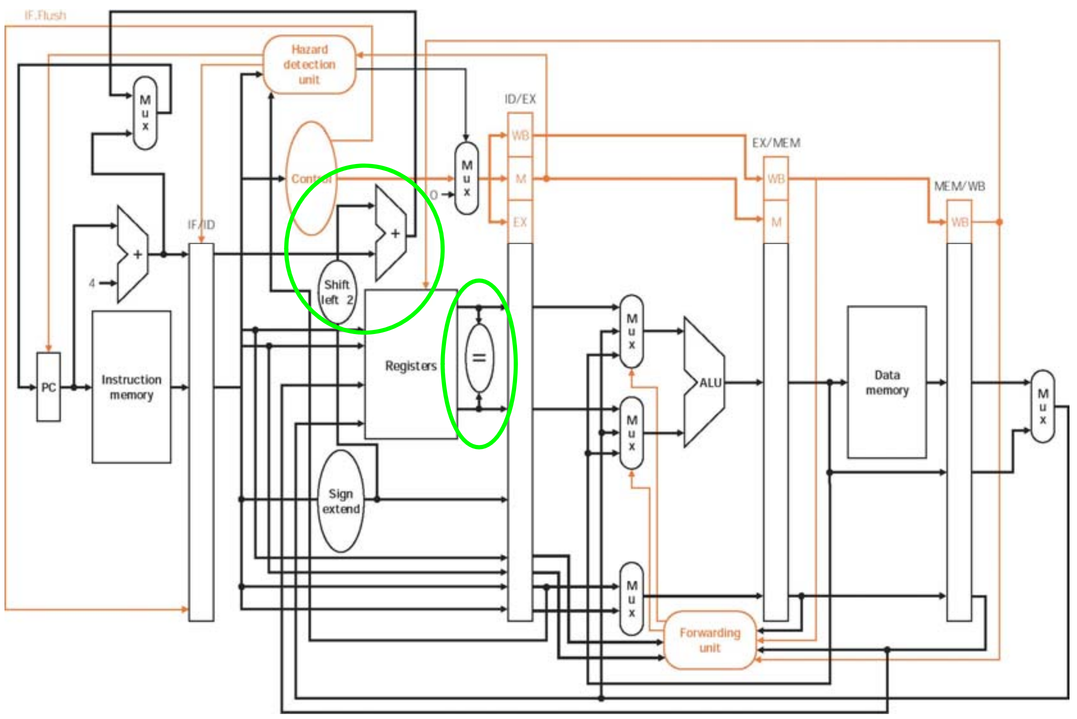
Branch Hazards

- **When we decide to branch, other instructions are in the pipeline!**



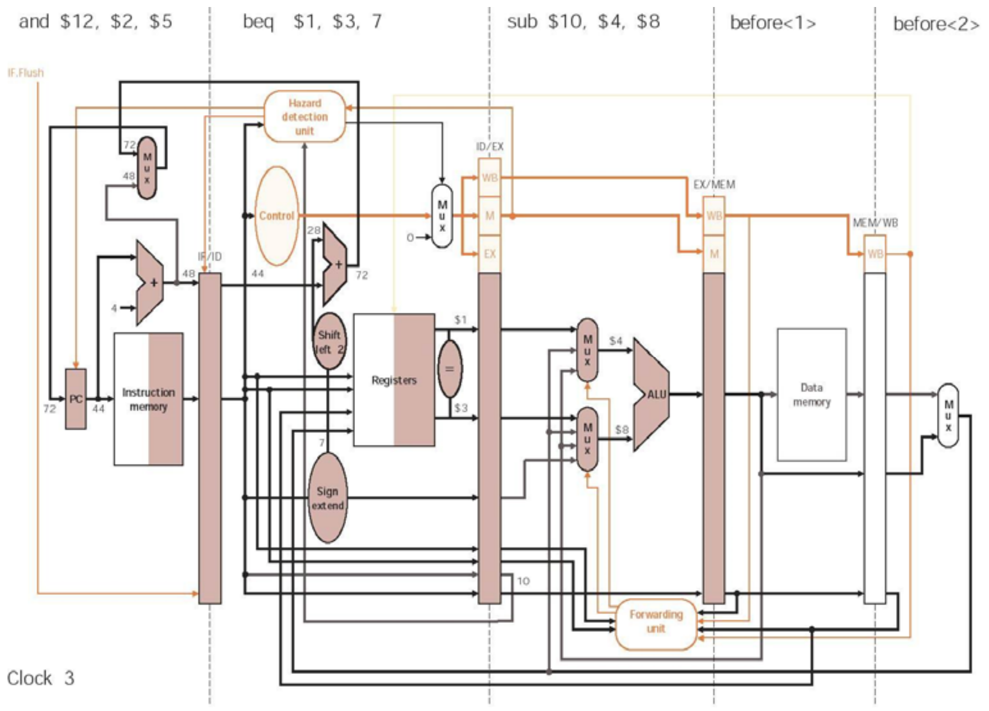
- **We are predicting "branch not taken"**
  - **need to add hardware for flushing instructions if we are wrong**

Datapath with Branch Handling

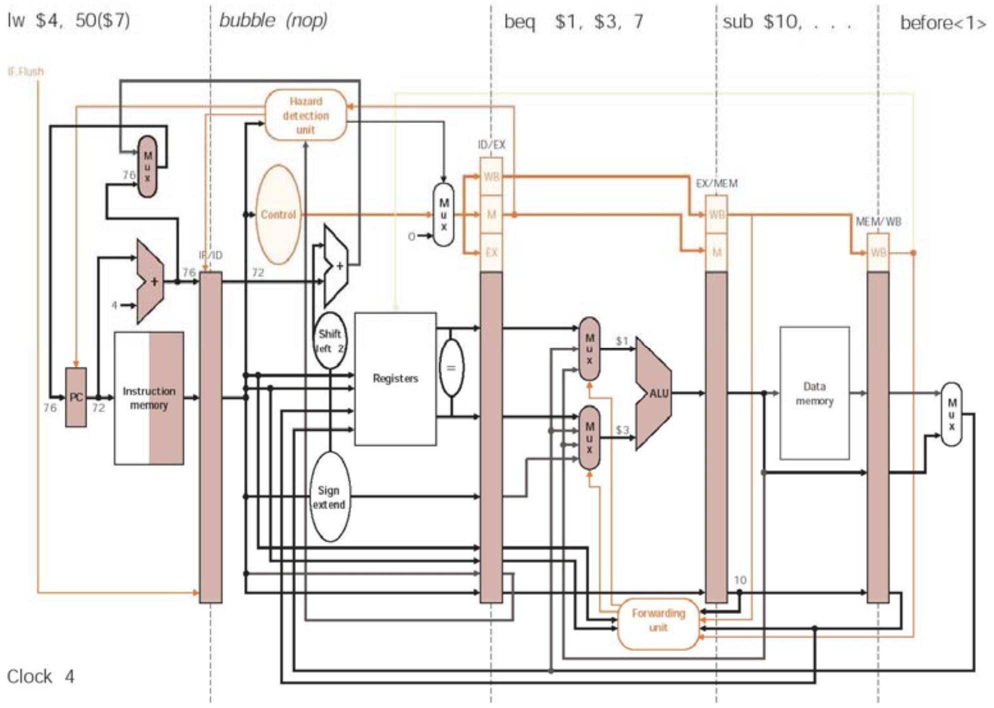


- **Note: we've also moved branch decision to ID stage**

Corrected Datapath (Cycle 3)



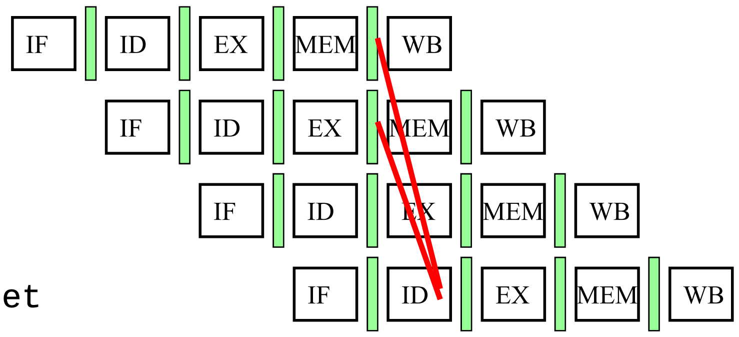
Corrected Datapath (Cycle 4)



Data Hazards for Branches

- **If a comparison register is a destination of 2nd or 3rd preceding ALU instruction**

```
add $1, $2, $3
add $4, $5, $6
…
beq $1, $4, target
```

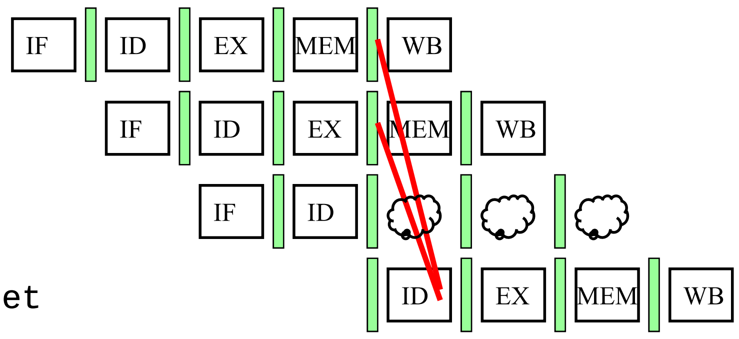


- Can resolve using forwarding

Data Hazards for Branches

- **If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction**
  - **Need 1 stall cycle**

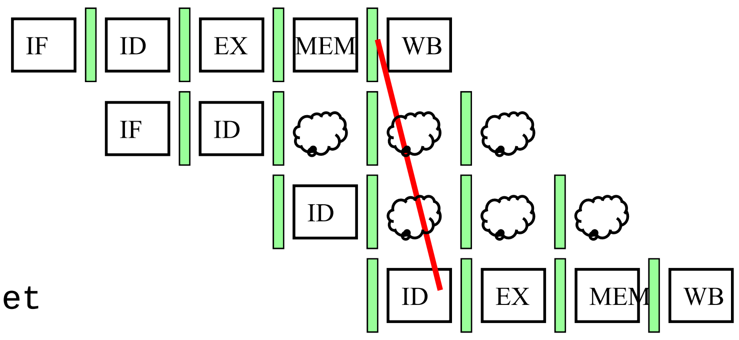
```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



Data Hazards for Branches

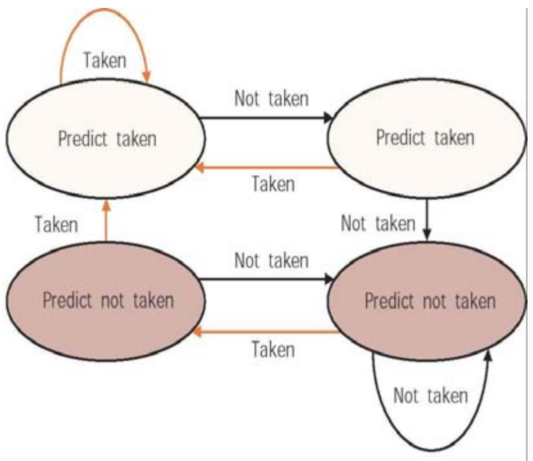
- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles

```
lw  $1, addr
beq stalled
beq stalled
beq $1, $0, target
```



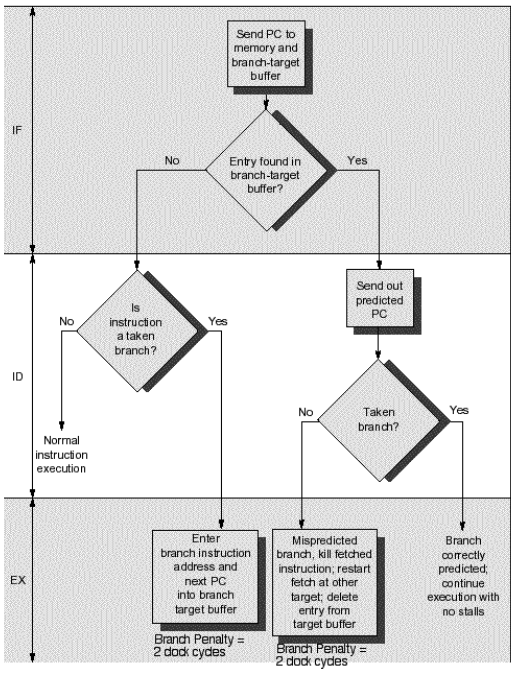
Branches

- If the branch is taken, we have a penalty of one cycle
- For our simple design, this is reasonable
- With deeper pipelines, penalty increases and static branch prediction drastically hurts performance
- Solution: dynamic branch prediction

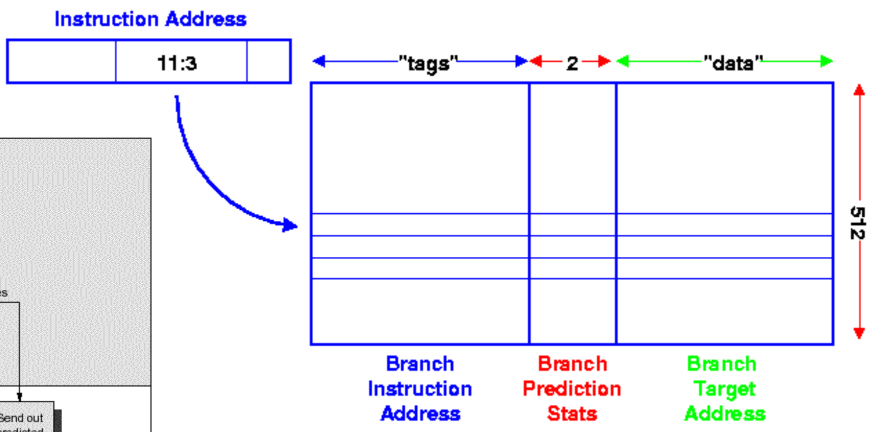


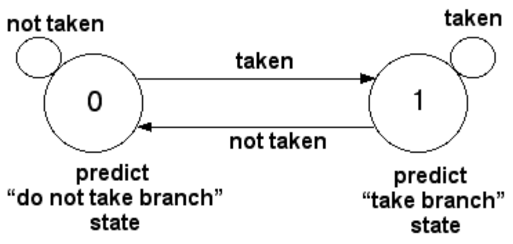
A 2-bit prediction scheme

Branch Target Buffer



IF
ID
EX
Send PC to memory and branch-target buffer
Entry found in branch-target buffer?
No
Yes
Is instruction a taken branch?
No
Yes
Normal instruction execution
Send out predicted PC
Taken branch?
No
Yes
Enter branch instruction address and next PC into branch target buffer
Branch Penalty = 2 clock cycles
Mispredicted branch, kill fetched instruction; restart fetch at other target; delete entry from target buffer
Branch Penalty = 2 clock cycles
Branch correctly predicted; continue execution with no stalls



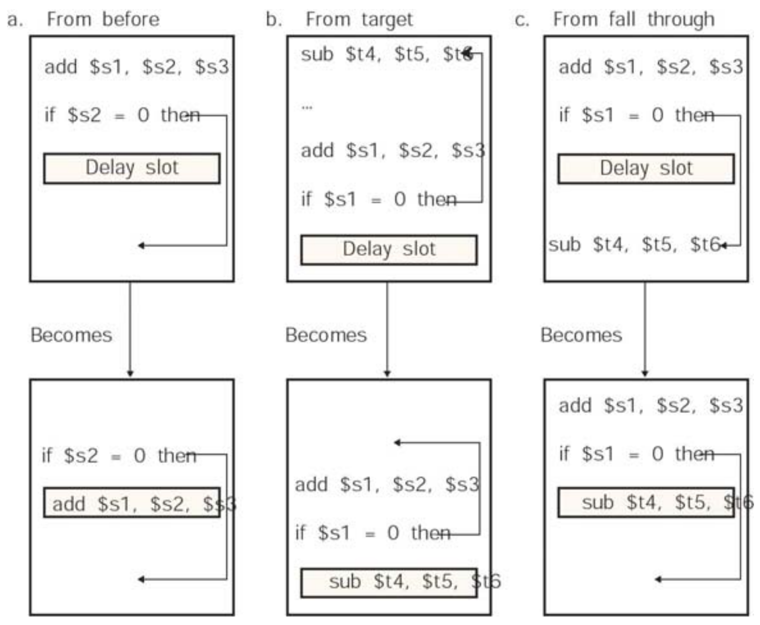


not taken
taken
taken
not taken
0
1
predict "do not take branch" state
predict "take branch" state

Branch Prediction

- Sophisticated Techniques:
  - A "branch target buffer" to help us look up the destination
  - Correlating predictors that base prediction on global behavior and recently executed branches (e.g., prediction for a specific branch instruction based on what happened in previous branches)
  - Tournament predictors that use different types of prediction strategies and keep track of which one is performing best.
  - A "branch delay slot" which the compiler tries to fill with a useful instruction (make the one cycle delay part of the ISA)
- Branch prediction is especially important because it enables other more advanced pipelining techniques to be effective!
- Modern processors predict correctly 95% of the time!

Branch Delay Slot



Scheduling Branch Delay Slot

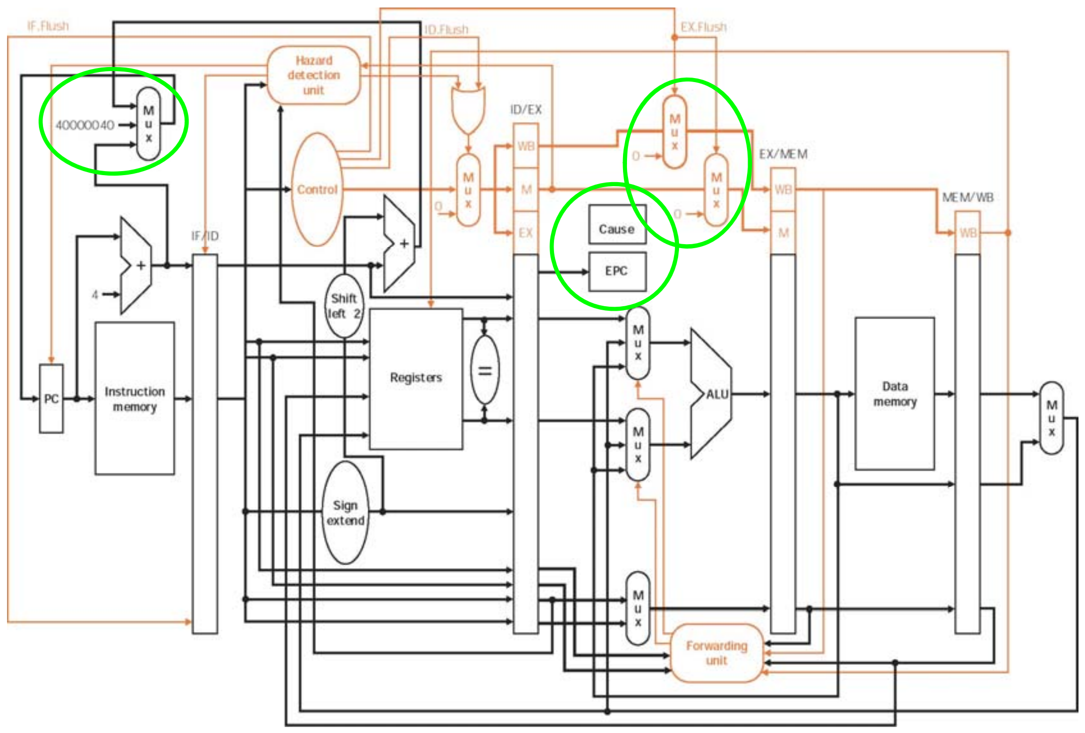
Exception Problem

- **Exceptions/Interrupts: 5 instructions executing in 5 stage pipeline**
  - **How to stop the pipeline?**
  - **Restart?**
  - **Who caused the interrupt?**

Stage	**Problem interrupts occurring**
IF	**Page fault on instruction fetch; misaligned memory access; memory-protection violation**
ID	**Undefined or illegal opcode**
EX	**Arithmetic exception**
MEM	**Page fault on data fetch; misaligned memory access; memory-protection violation; memory error**

- **Resolution: Freeze above & Bubble Below**

Datapath supporting Exception handling



Exception Properties

- **Restartable exceptions**
  - **Pipeline can flush the instruction**
  - **Handler executes, then returns to the instruction**
    - Refetched and executed from scratch
- **PC saved in EPC register**
  - **Identifies causing instruction**
  - **Actually PC + 4 is saved**
    - Handler must adjust

Exception Example

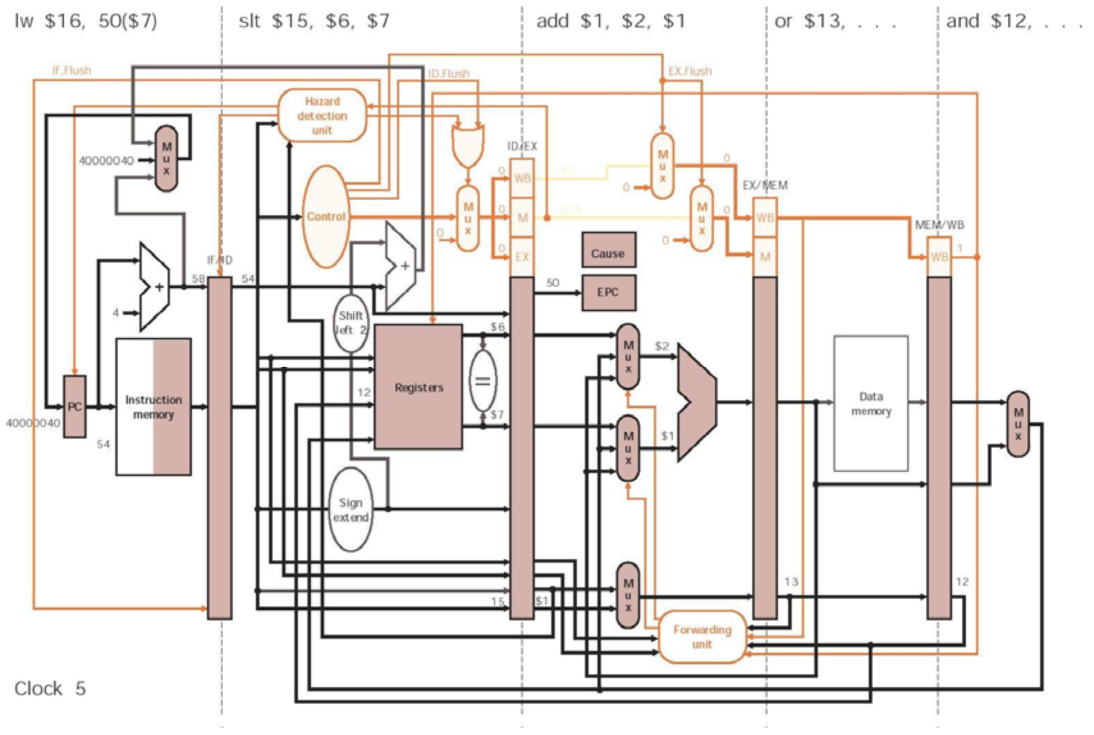
- Exception on add in

```
40          sub  $11, $2, $4
44          and  $12, $2, $5
48          or   $13, $2, $6
4C          add  $1,  $2, $1
50          slt  $15, $6, $7
54          lw   $16, 50($7)
…
```

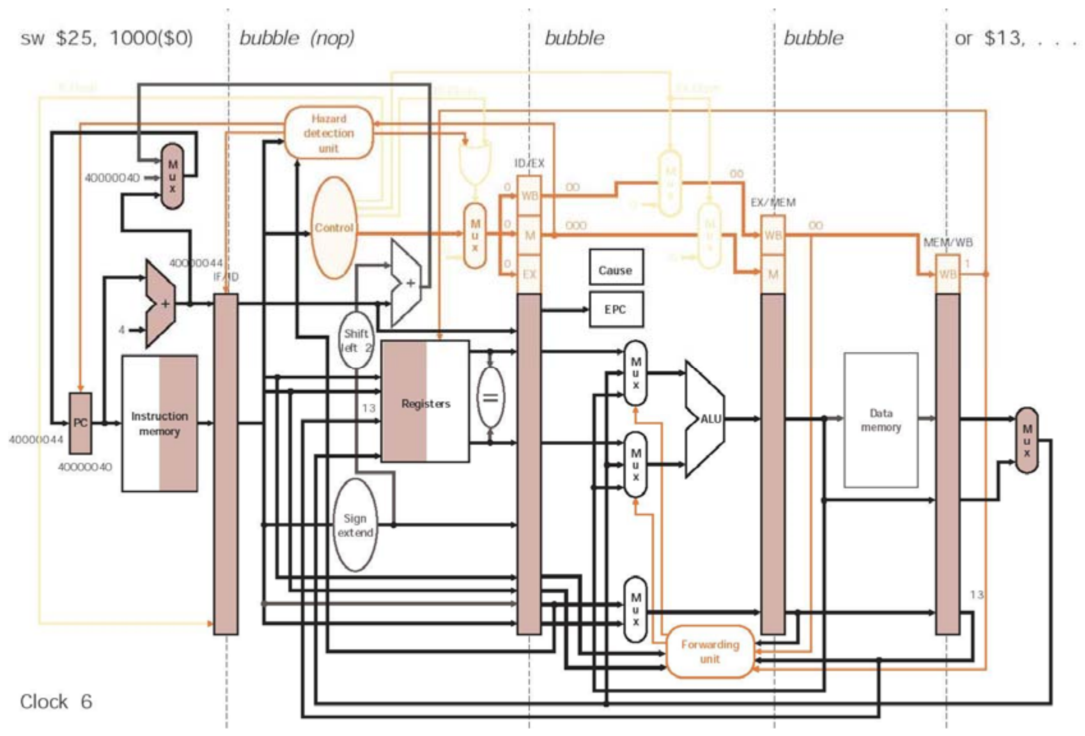
- Handler

```
80000180 sw   $25, 1000($0)
80000184 sw   $26, 1004($0)
…
```

Handling Exception (Arithmetic overflow)



Handling Exception (Arithmetic overflow)



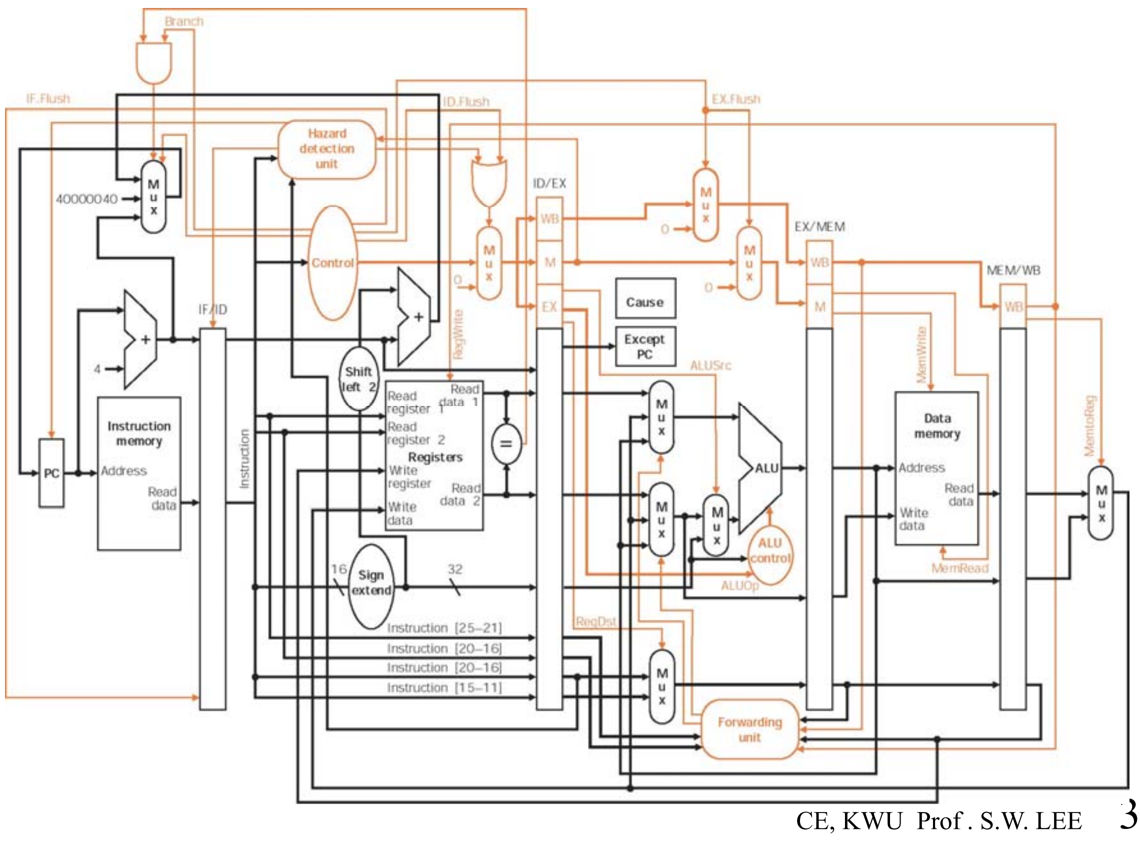
Multiple Exceptions

- **Pipelining overlaps multiple instructions**
  - **Could have multiple exceptions at once**
- **Simple approach: deal with exception from earliest instruction**
  - **Flush subsequent instructions**
  - **“Precise” exceptions**
- **In complex pipelines**
  - **Multiple instructions issued per cycle**
  - **Out-of-order completion**
  - **Maintaining precise exceptions is difficult!**

Imprecise Exceptions

- **Just stop pipeline and save state**
  - **Including exception cause(s)**
- **Let the handler work out**
  - **Which instruction(s) had exceptions**
  - **Which to complete or flush**
    - May require "manual" completion
- **Simplifies hardware, but more complex handler software**
- **Not feasible for complex multiple-issue out-of-order pipelines**

Final version of datapath and control


Branch
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
40000040
M u x
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
0
Cause
Except PC
IF/ID
4
Instruction memory
Address
Read data
PC
Instruction
Shift left 2
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
ALUSrc
ALU
ALU control
ALUOp
MemWrite
Data memory
Address
Read data
Write data
MemRead
MemtoReg
16
Sign extend
32
Instruction [25–21]
Instruction [20–16]
Instruction [20–16]
Instruction [15–11]
RegDst
Forwarding unit