CMPN3010: Computer Architecture

Pipelining Hazards

Control Hazards

Mayada Hadhoud

Computer Engineering Department

Cairo University

Pipeline Hazards

- **Hazards**: Situations that prevent an instruction from being executed in its designated clock cycle

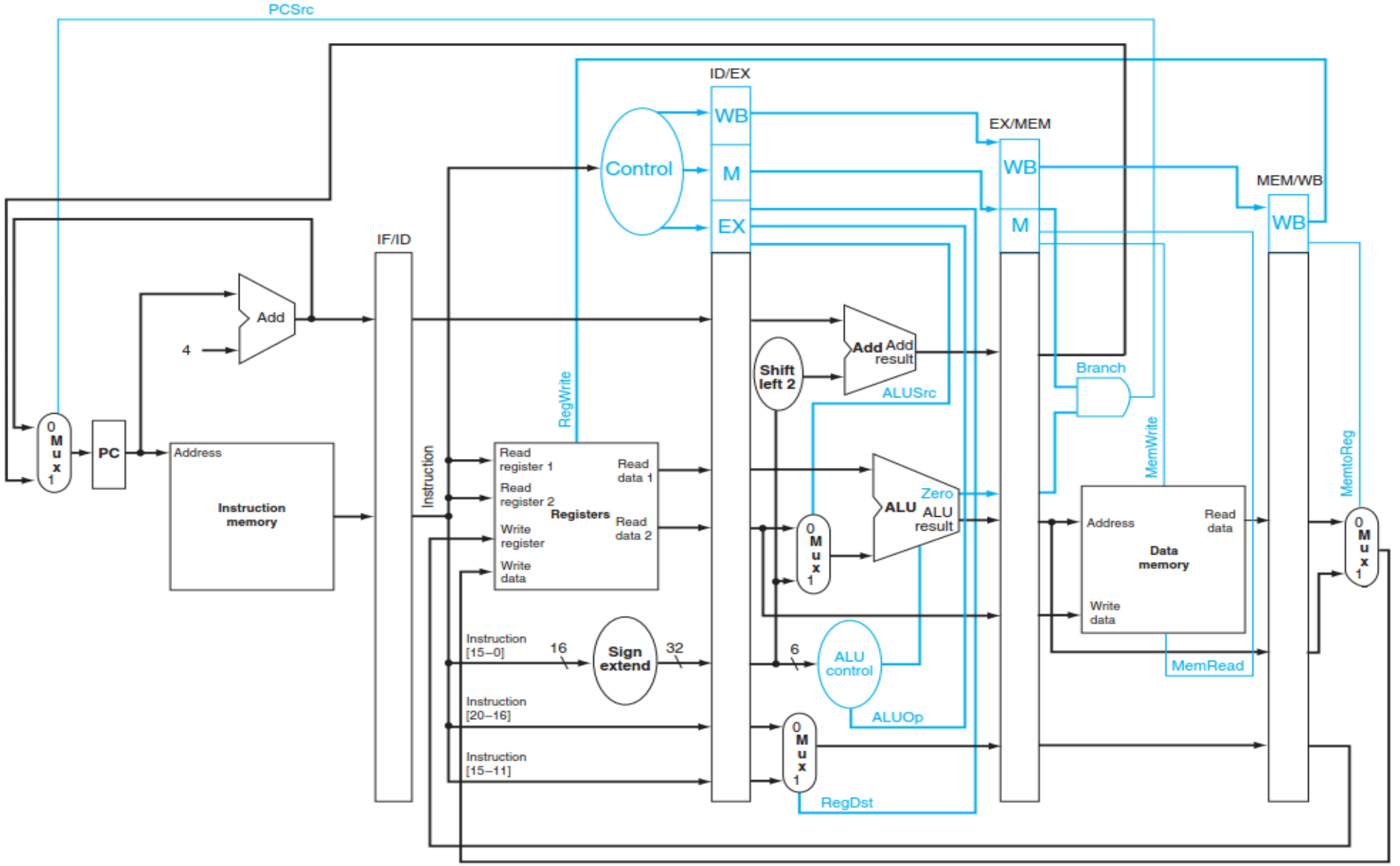
- **Types of Hazards**
  1. **Structural Hazards**: two different instructions use same h/w in same cycle
  2. **Data Hazards**: An instruction depends on the results of previous instruction
  3. **Control Hazards**: When an instruction changes PC, like branches

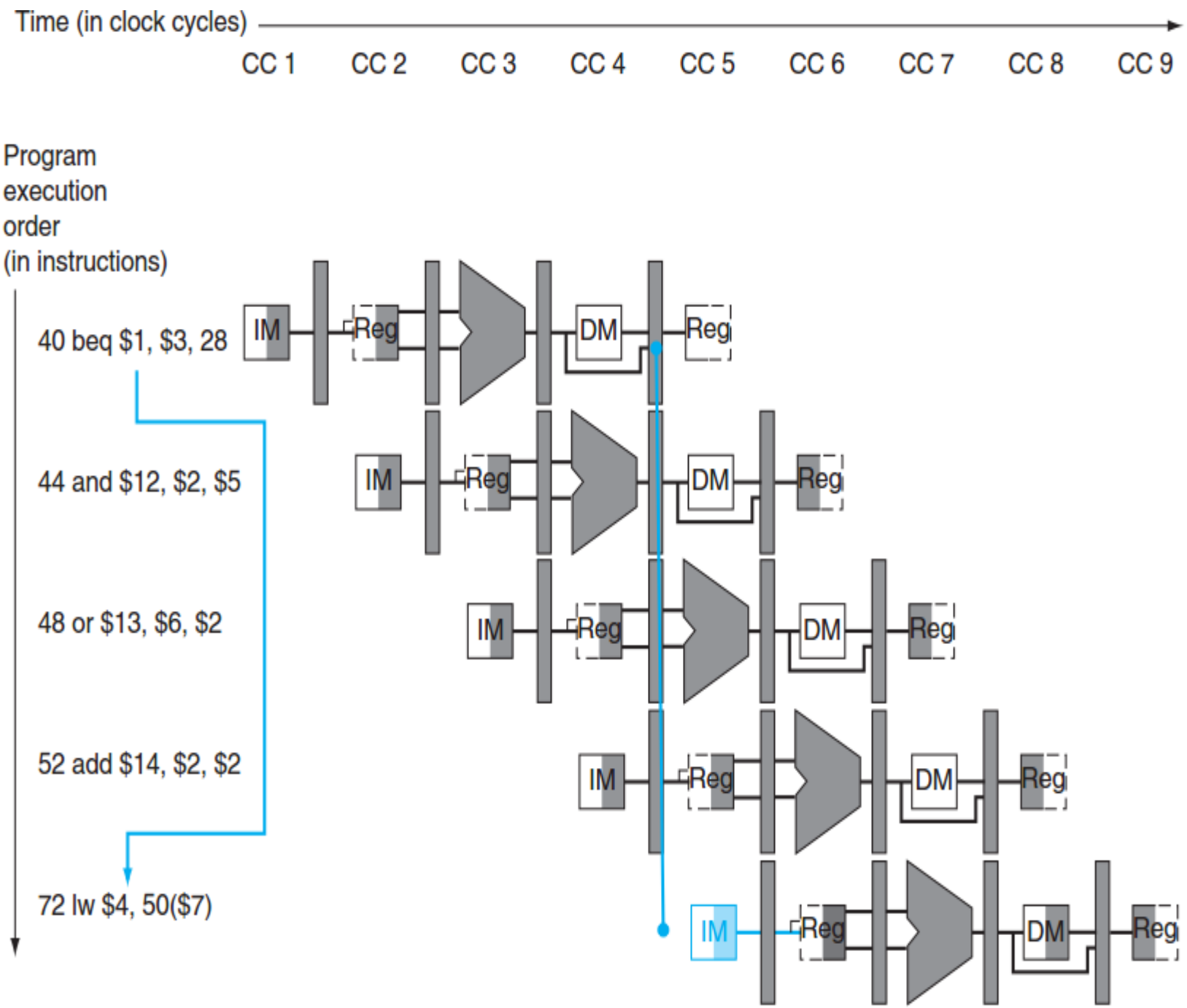
- The simplest hazards solution is to **stall the pipeline** (some instructions are allowed to proceed, while other are delayed)

Control Hazards/Branch Hazards

- **An occurrence in which the proper instruction cannot execute in the proper clock cycle because the instruction that was fetched is not the one that is needed; that is, the flow of instruction addresses is not what the pipeline expected.**

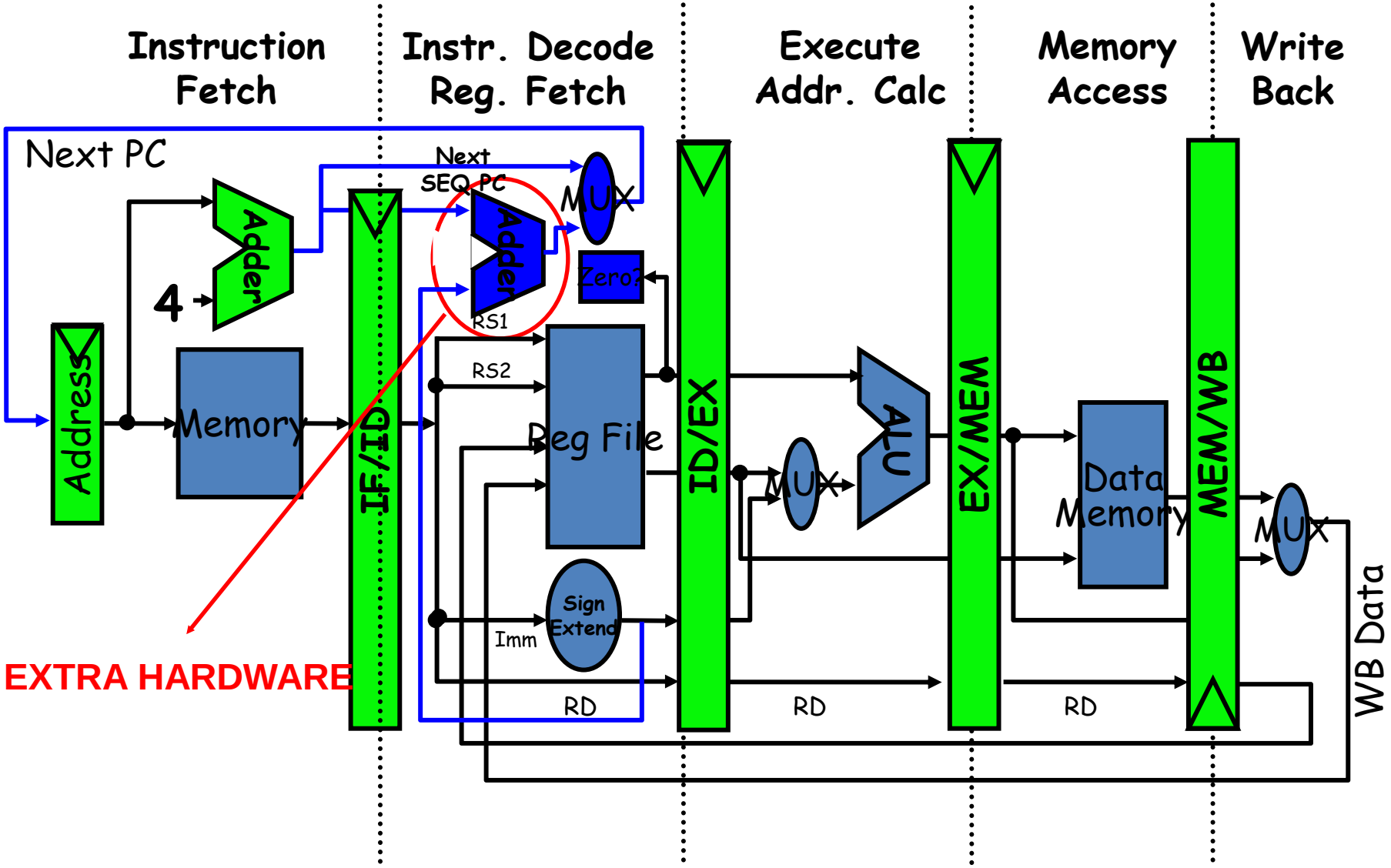
The pipelined data path with control signals




Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
40 beq $1, $3, 28
44 and $12, $2, $5
48 or $13, $6, $2
52 add $14, $2, $2
72 lw $4, 50($7)
IM
Reg
DM
Reg

- **If branch is taken**
  - What are the number of instructions that will be flushed from the pipe ???

Reducing Branch Penalty



Control Hazards Solutions

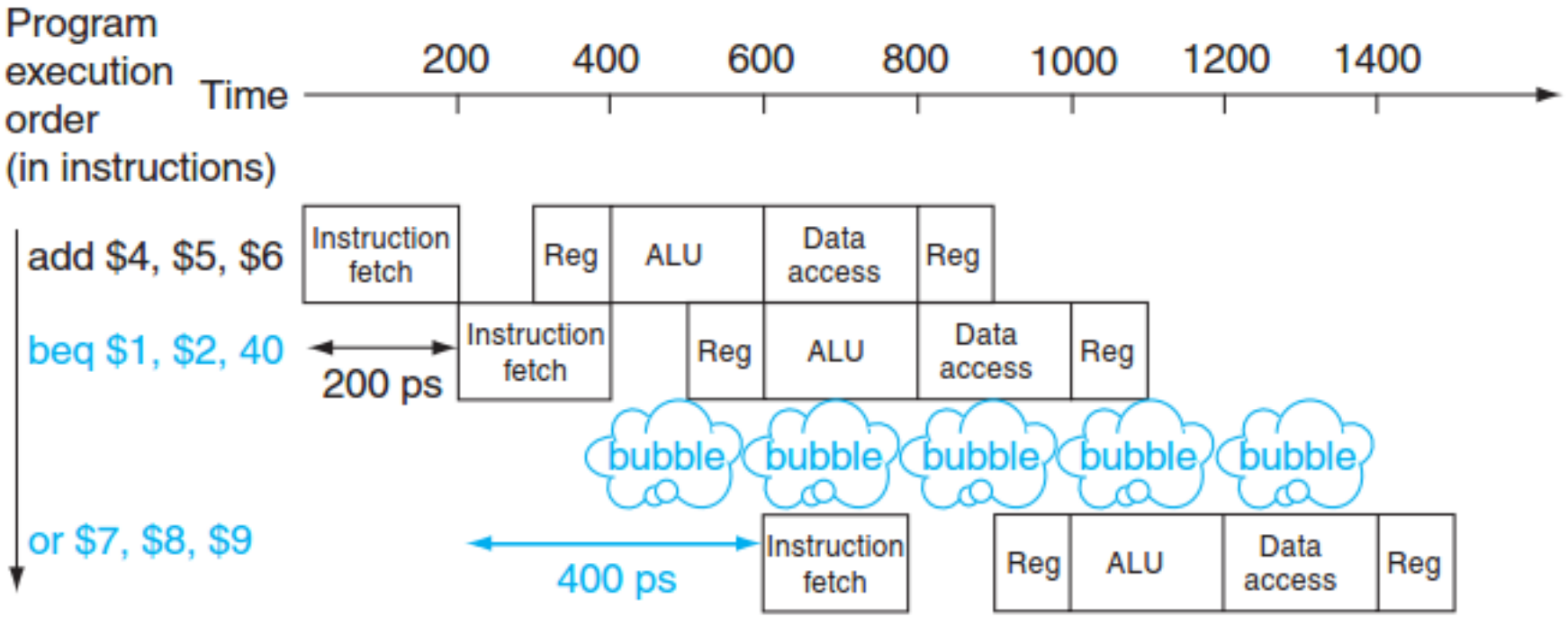
- **Solution 1: Stall the pipe**
- Stall “freeze” the pipeline till the branch direction is identified
- Simple, but not efficient

Control Hazards Solutions

- **Exercise:**
  - Estimate the impact on the clock cycles per instruction (CPI) of stalling on branches. Assume all other instructions have a CPI of 1. given branch percentage is 17%
  - **Solution**: branches took one extra clock cycle for the stall, then we would see a CPI of 1.17 and hence a slowdown of 1.17 versus the ideal case.

Control Hazards Solutions

- **Stalling effect**



The cost of stalling is too high to be used

Control Hazards Solutions

- **Solution 2: Predict**
- **Prediction approaches:**
  - Always predict that branches will be **untaken.**
  - Execute successor instructions in sequence
  - **Discard** instructions in pipeline if branch actually taken
  - 47% MIPS branches not taken on average
  - **How to Flush (Discard) instructions?**

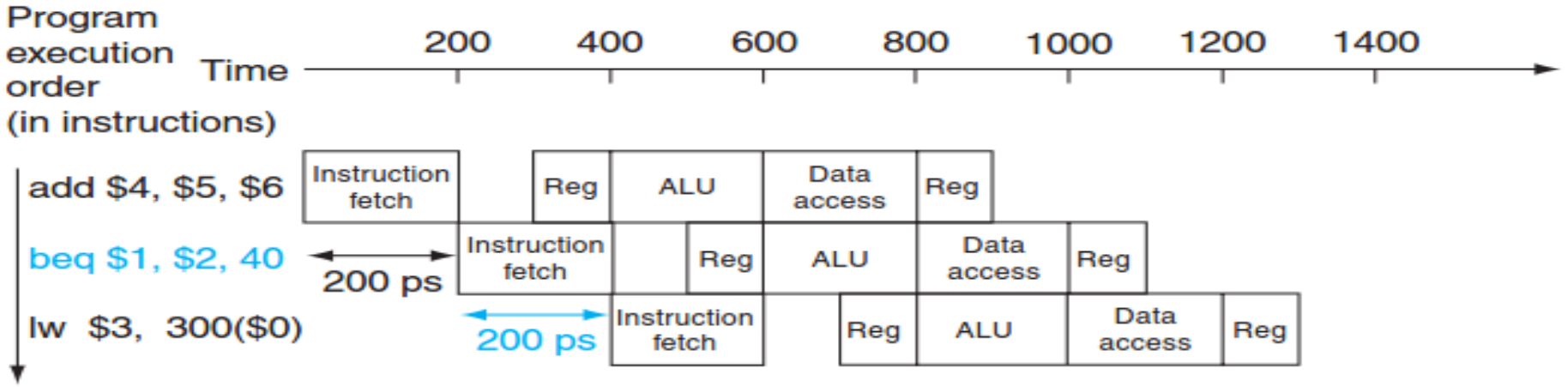
Example

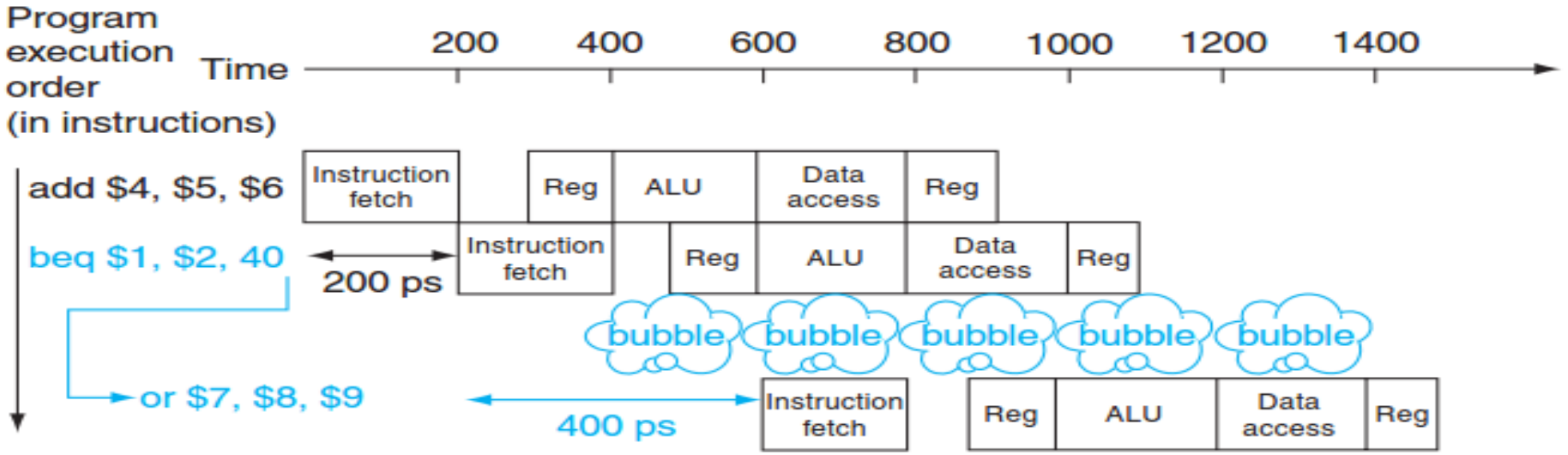
Pipelined Branch

Show what happens when the branch is taken in this instruction sequence, assuming the pipeline is optimized for branches that are not taken and that we moved the branch execution to the ID stage:

```
36 sub $10, $4, $8
40 beq  $1, $3, 7 # PC-relative branch to 40 + 4
+ 7 * 4 = 72
44 and $12, $2, $5
48 or  $13, $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7
. . .
72 lw  $4,  50($7)
```

Taken Branch





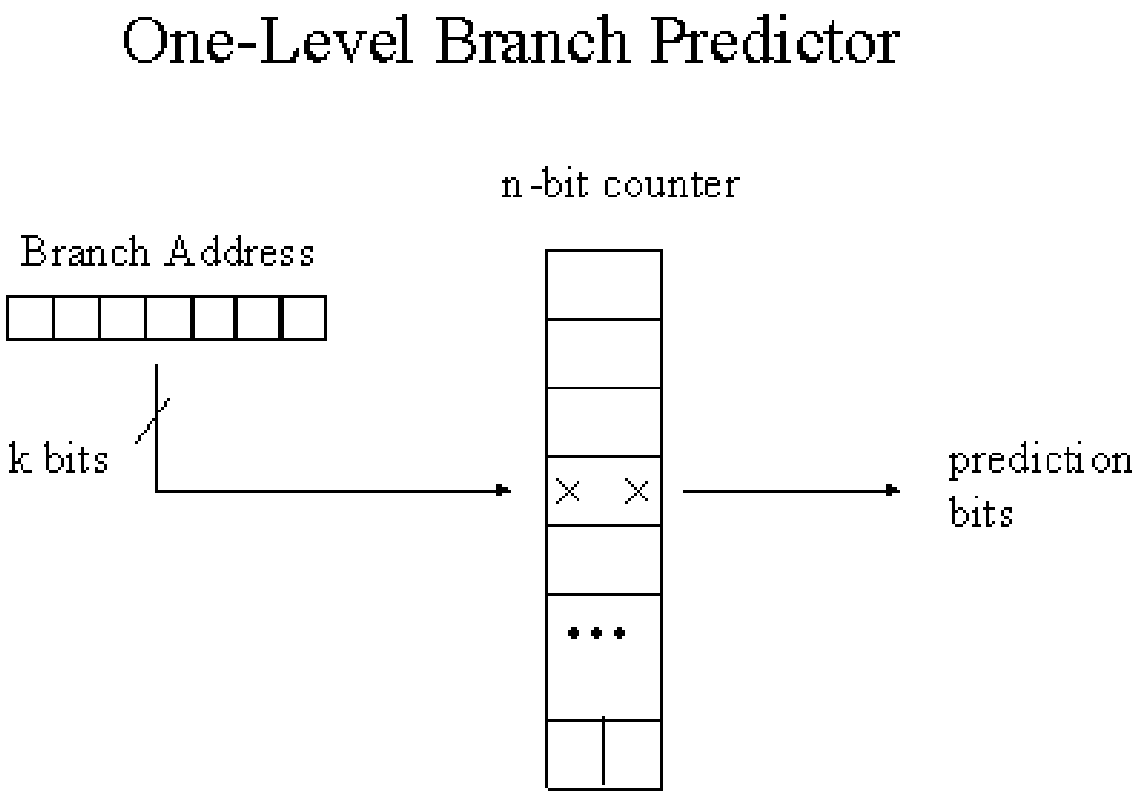
Control Hazards Solutions

- **Solution 2: Predict**
- **Prediction approaches:**
  - More sophisticated version of branch prediction would have **some branches predicted as taken and some as untaken.**
  - **Example:** at the bottom of loops are branches that jump back to the top of the loop. Since they are likely to be taken and they branch backwards, we could always predict **taken for branches that jump to an earlier address**

Dynamic Branch Prediction

- **Dynamic Branch Prediction** : Prediction of branches at runtime using run time information.
- **Branch Prediction Buffer**: A small memory that is indexed by the lower portion of the address of the branch instruction and that contains one or more bits indicating whether the branch was recently taken or not.

Dynamic Branch Prediction



Dynamic Branch Prediction

- The prediction changes according to the history of the individual branch instruction
- **1-bit predictor**: use the last outcome to predict whether the branch is taken or not taken
- **2-bit predictor**: Prediction must miss twice before it is changed

Example

Loops and Prediction

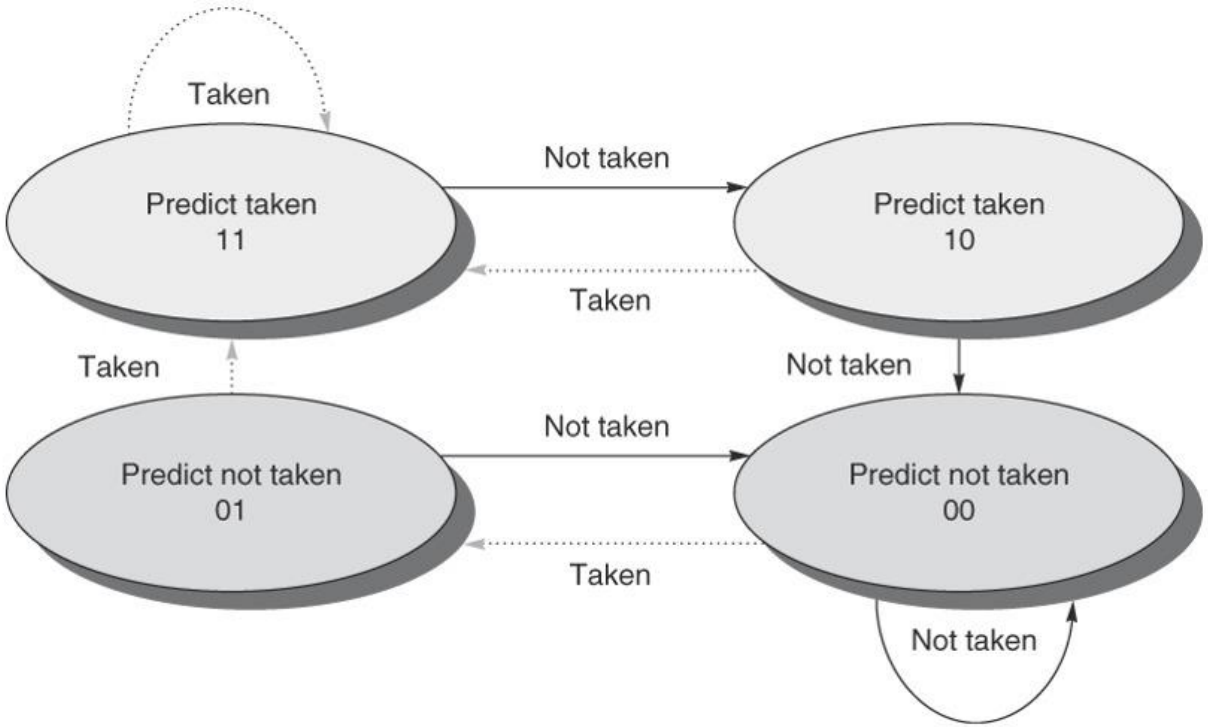
Consider a loop branch that branches nine times in a row, then is not taken once. What is the prediction accuracy for this branch, assuming the prediction bit for this branch remains in the prediction buffer?

ANSWER

The steady-state prediction behavior will mispredict on the first and last loop iterations. Mispredicting the last iteration is inevitable since the prediction bit will say taken: the branch has been taken nine times in a row at that point. The misprediction on the first iteration happens because the bit is flipped on prior execution of the last iteration of the loop, since the branch was not taken on that exiting iteration. Thus, the prediction accuracy for this branch that is taken 90% of the time is only 80% (two incorrect predictions and eight correct ones).

Dynamic Branch Prediction

- The prediction changes according to the history of the individual branch instruction
- 2-bit predictor



Control Hazards Solutions

- Solution 3: **Delayed Branch**
  - Compiler Solution
  - Compiler inserts instruction after the branch equal to the branch delay slots “usually one”
  - These instructions will be executed whether the branch is taken or not

Delayed Branch

LOOP	Shift_left	R1
	Decrement	R2
	Branch=0	LOOP
NEXT	Add	R1,R3

(a) Original program loop

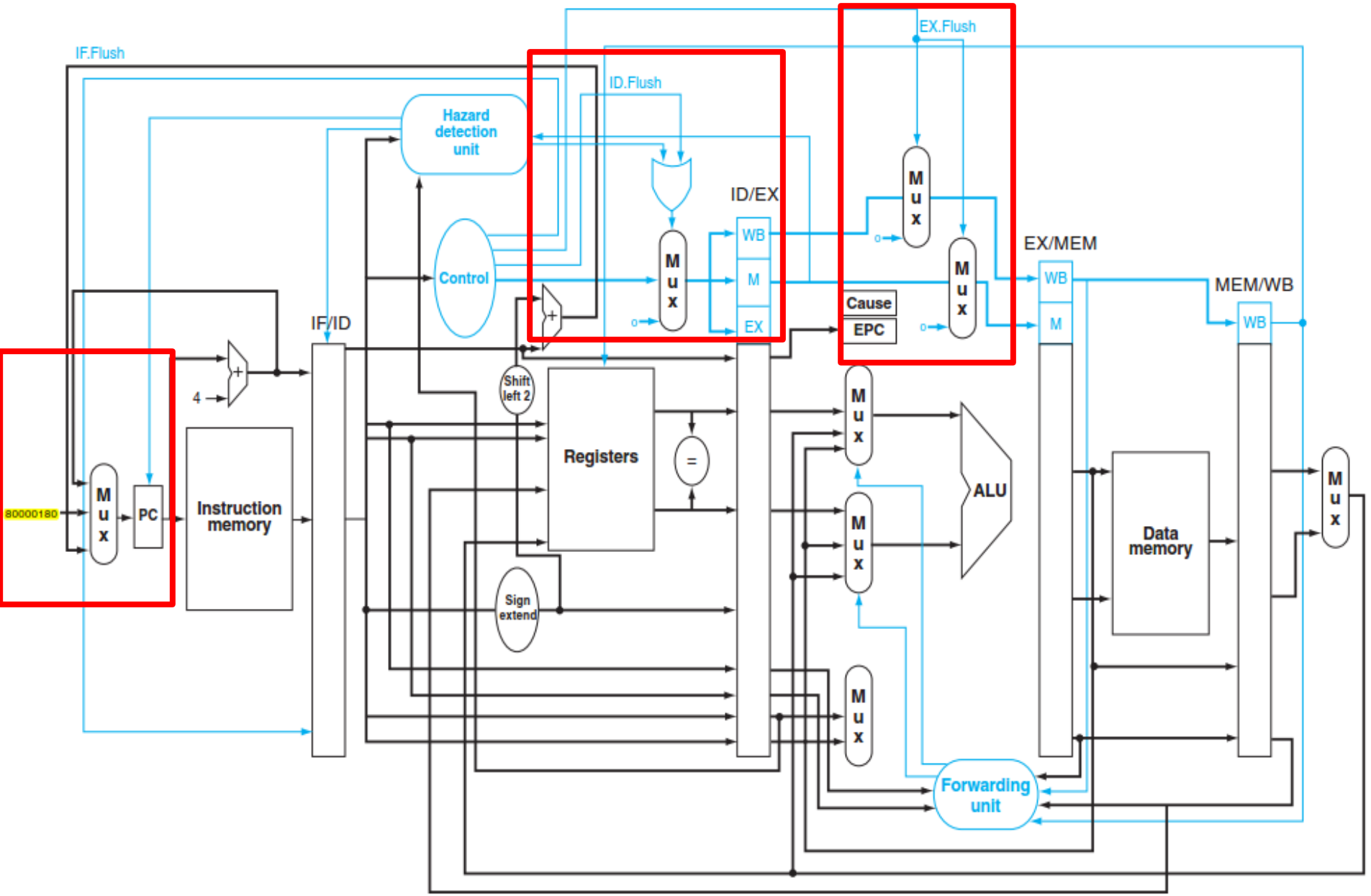
LOOP	Decrement	R2
	Branch=0	LOOP
	Shift_left	R1
NEXT	Add	R1,R3

(b) Reordered instructions

Exceptions

- **Another form of control hazards**
- **In exceptions we need to :**
  - Transfer control to the exception routine
  - Flush the instructions that follow the instruction that caused the exception from the pipeline and begin fetching instructions from the new address.
  - Finally, save the address of the offending instruction in the Exception Program Counter (EPC).

H/W changes


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

Example

Exception in a Pipelined Computer

Given this instruction sequence,

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

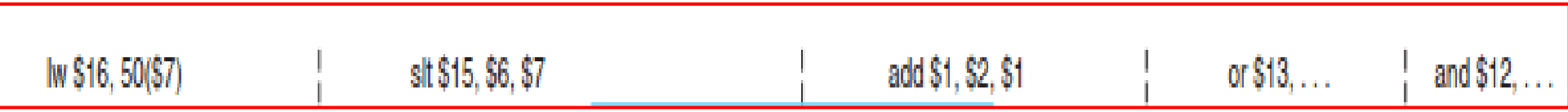
assume the instructions to be invoked on an exception begin like this:

```
40000040hex      sw       $25, 1000($0)
40000044hex      sw       $26, 1004($0)
...
```

Show what happens in the pipeline if an overflow exception occurs in the `add` instruction.

Example- Solution

- **After 5 clock cycles the pipe will look like**



- **At clock cycle 7**

Multiple Exceptions

- Multiple exceptions can occur simultaneously in a single clock cycle. **The normal solution** is to **prioritize** the exceptions so that it is easy to determine which is serviced first.

Recap

- Pipelining Hazards
- Structural Hazards
  - Structural Hazards Solutions
- Data Hazards
  - Data Hazards Solutions
- Control Hazards
  - Control Hazards Solutions
- Exceptions