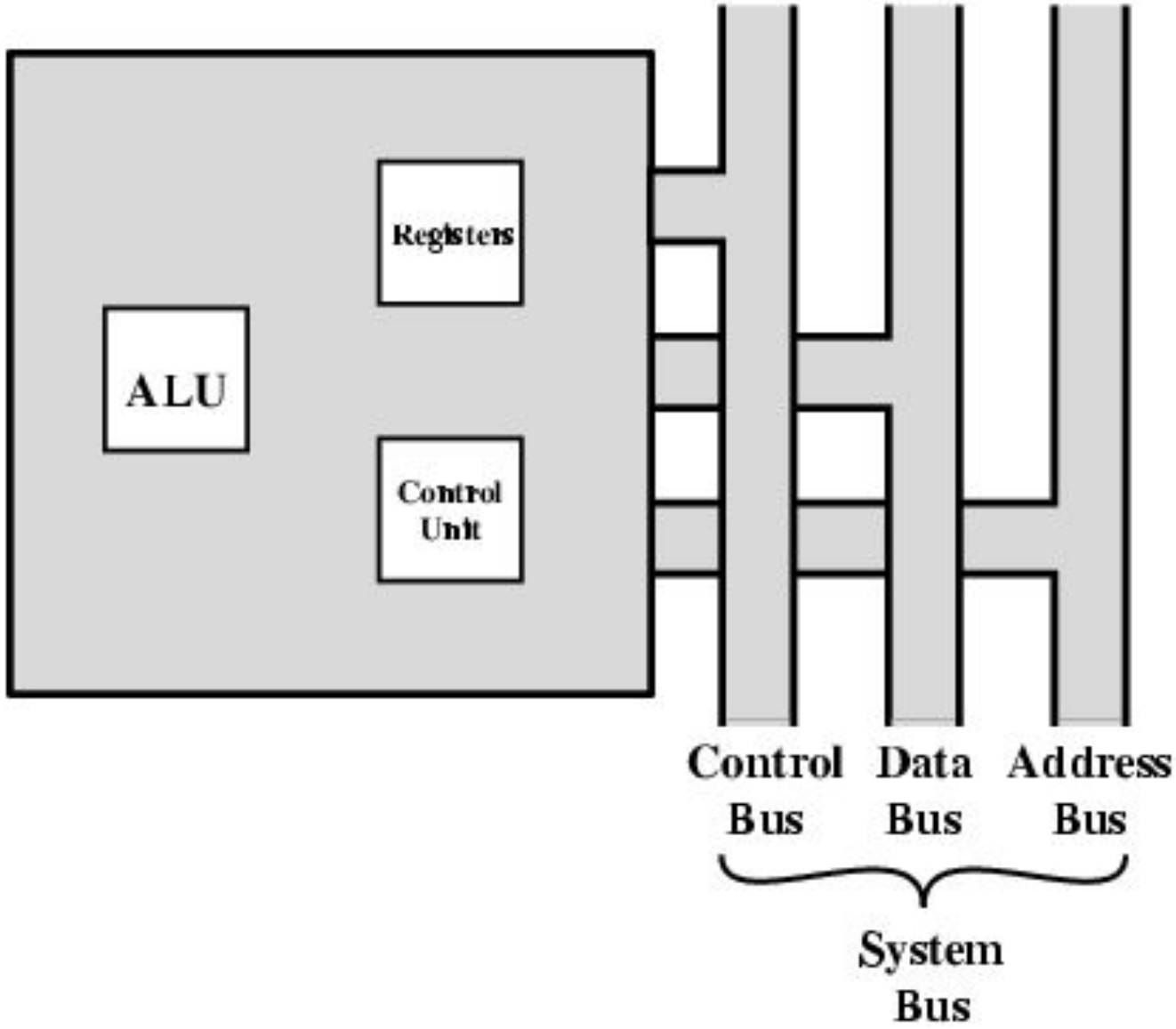
Pipelining: Datapath and Hazards

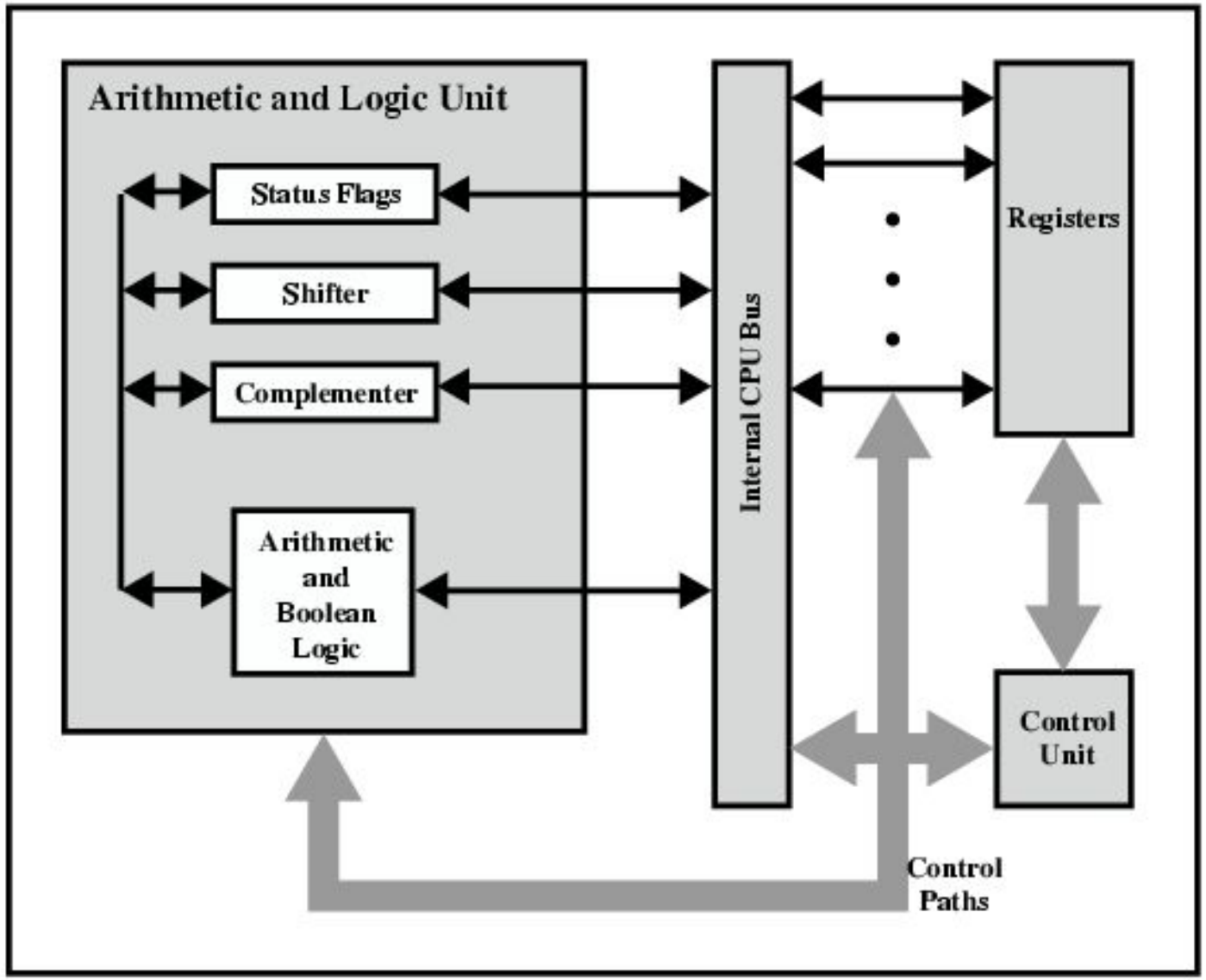
CPU Structure

- CPU must:
  - Fetch instructions
  - Interpret instructions
  - Fetch data
  - Process data
  - Write data

CPU With Systems Bus



CPU Internal Structure



Registers

- CPU must have some working space (temporary storage)
- Called registers
- Number and function vary between processor designs
- One of the major design decisions
- Top level of memory hierarchy

User’s Visibility

- General Purpose
- Data
- Address
- Condition Codes

General Purpose Registers (1)

- May be true general purpose
- May be restricted
  - (floating point and stack operation)
- May be used for data or addressing
- Data
  - Accumulator
- Addressing
  - Segment (ES,DS,SS,CS,SI/DI, SP)

Design Issues with GPR

- Make them general purpose
  - Increase flexibility and programmer options
  - Increase instruction size & complexity
- Make them specialized
  - Smaller (faster) instructions
  - Less flexibility

How Many GP Registers?

- Between 8 – 32 registers are optimum
- Fewer in count more memory references
- More does not reduce memory references and takes up processor real estate
- See also RISC showing advantage of using 100s of registers.

How big?

- Large enough to hold full address
- Large enough to hold full word
  - (almost all data types)
- Often possible to combine two continuous data registers
  - C programming
    - double int a;
    - long int a;

Condition Code Registers

- Sets of individual bits
  - e.g. result of last operation was zero
- Can be read (implicitly) by programs
  - e.g. Jump if zero
- Can not (usually) be set by programs

Control & Status Registers

- Program Counter
- Instruction Decoding Register
- Memory Address Register
- Memory Buffer Register

- Revision: what do these all do?

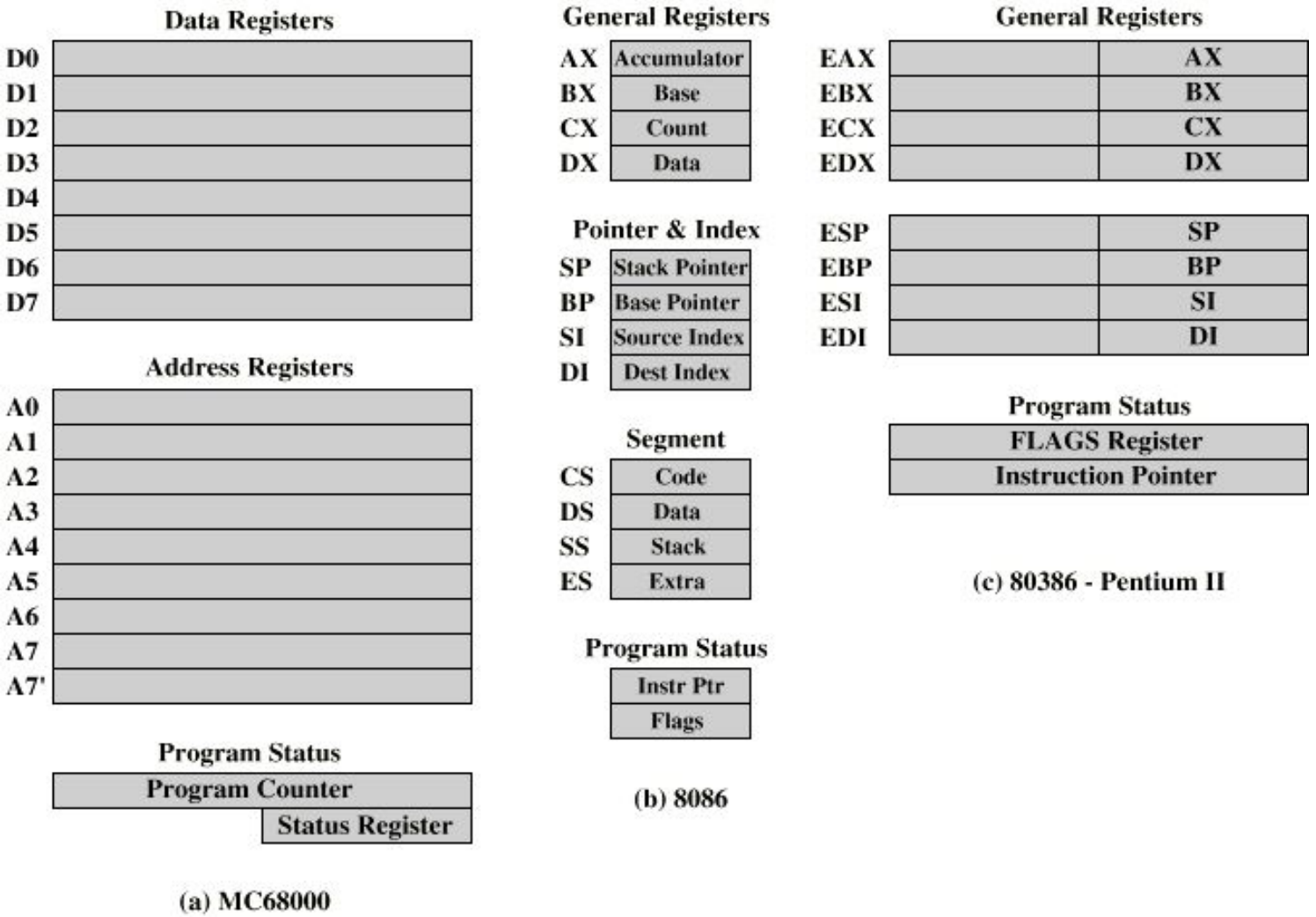
Program Status Word: PSW

- A set of bits
- Includes Condition Codes
- Sign of last result
- Zero
- Carry
- Equal
- Overflow
- Interrupt enable/disable
- Supervisor (Mode bits)- certain privileged instructions and memory location can be executed in supervised mode rather user mode.

Supervisor Mode

- Intel ring zero
- Kernel mode
- Allows privileged instructions to execute
- Used by operating system
- Memory locations not available to user programs

Example Register Organizations



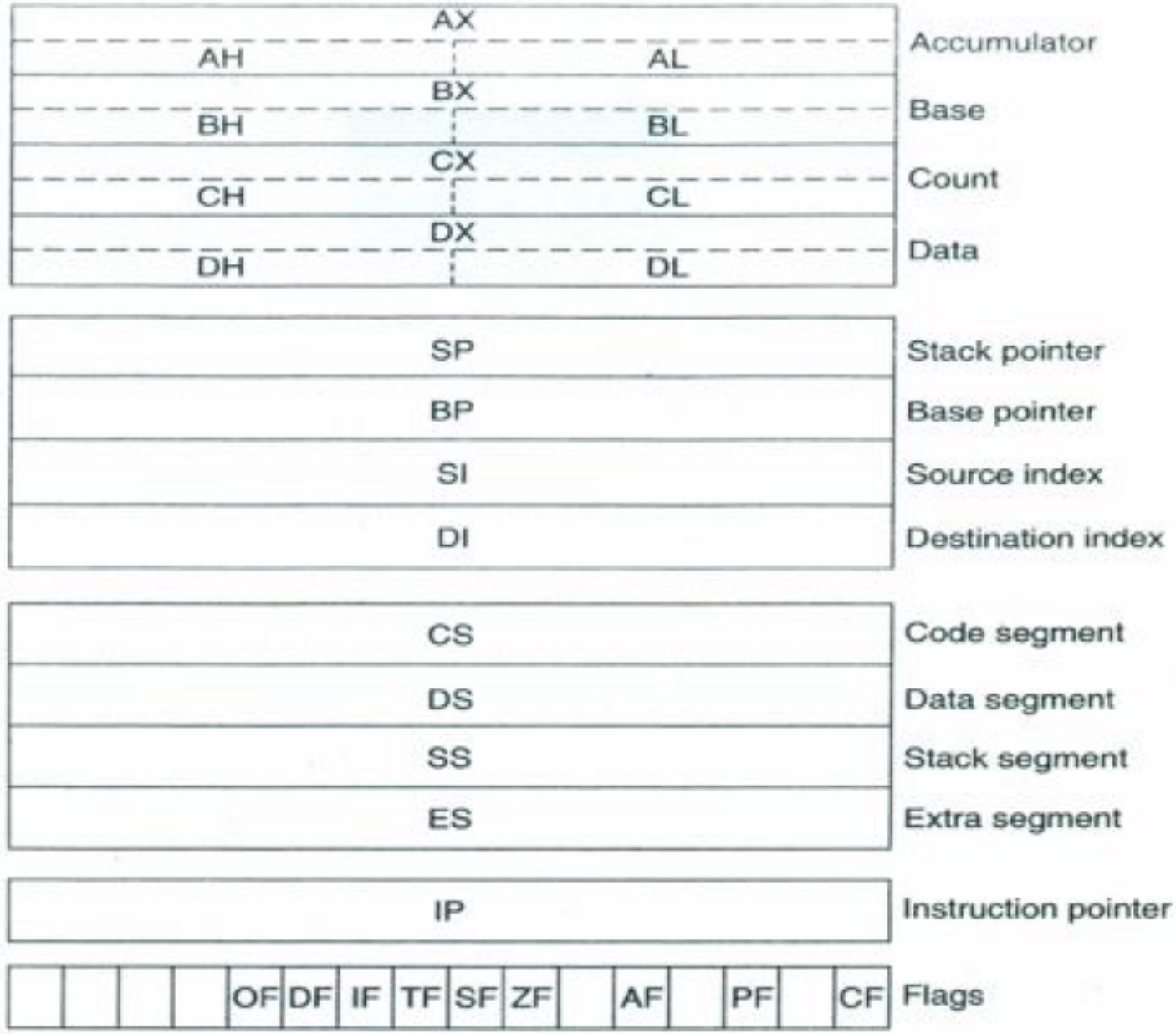
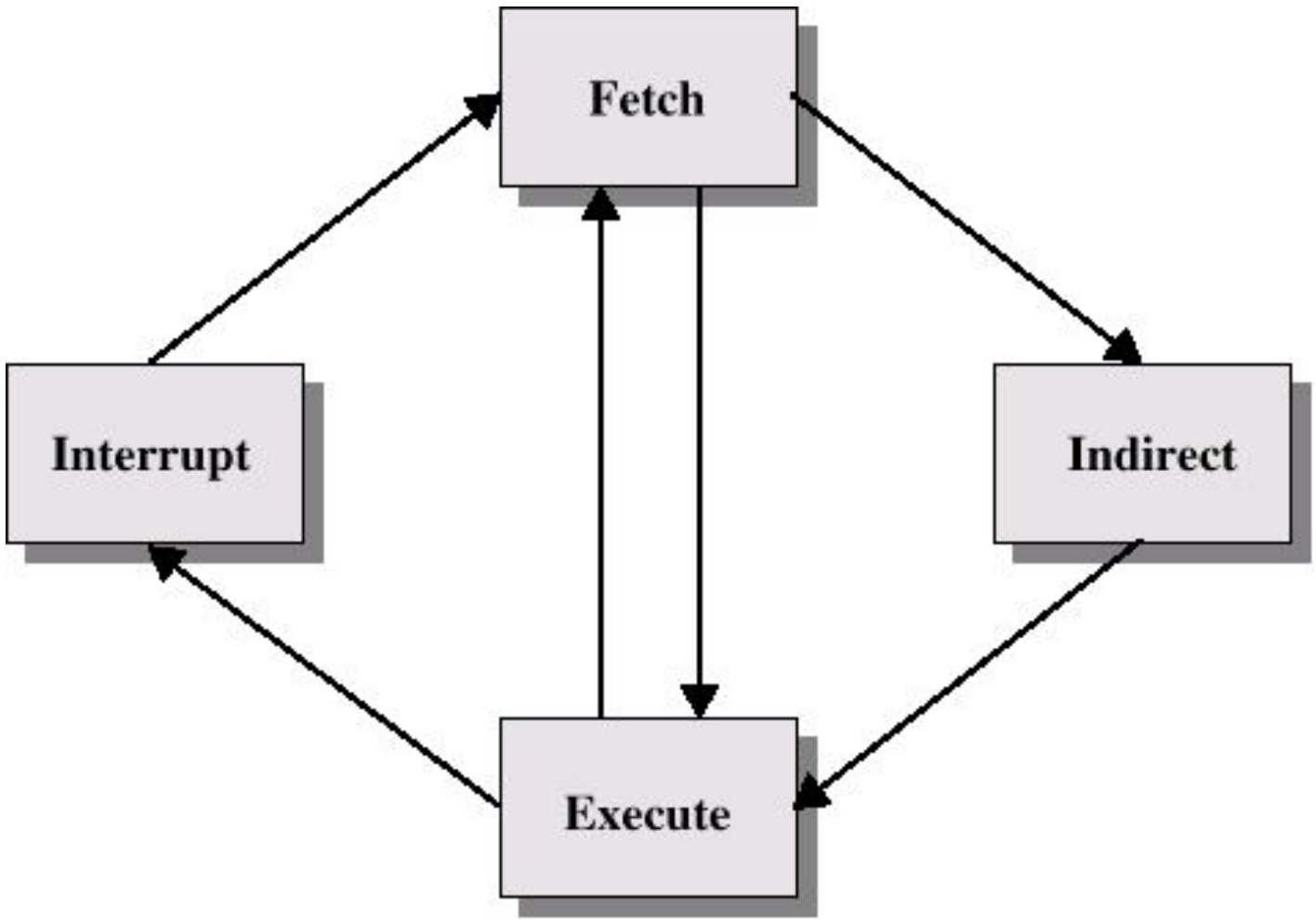


Figure 5.3 Intel 8086/8088 register set.

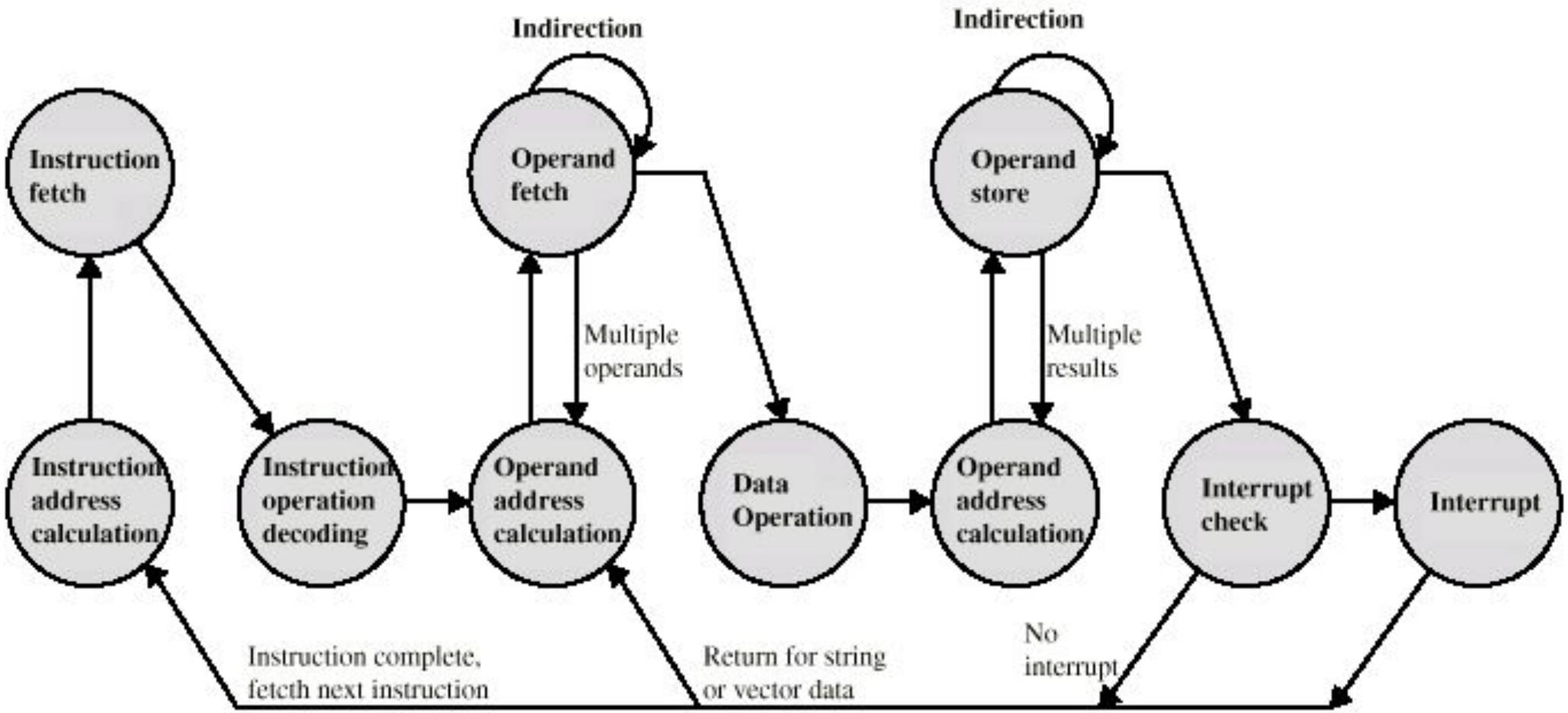
Indirect Cycle

- May require memory access to fetch operands
- Indirect addressing requires more memory accesses
- Can be thought of as additional instruction subcycle

Instruction Cycle with Indirect



Instruction Cycle State Diagram



Prefetch

- Fetch accessing main memory
- Execution usually does not access main memory
- Can fetch next instruction during execution of current instruction
- Called instruction prefetch

Improved Performance

- But not doubled:
  - Fetch usually shorter than execution
    - Prefetch more than one instruction?
  - Any jump or branch means that prefetched instructions are not the required instructions
- Add more stages to improve performance

Pipelining

- Fetch instruction
- Decode instruction
- Calculate operands address(i.e. EAs)
- Fetch operands
- Execute instructions
- Write result

- Overlap these operations

Introduction

The performance of a Single CPU can be increased by :

- Improving the hardware by introducing faster circuits.
- Arranging the hardware such that more than one operation can be performed at the same time.

Since, there is a limit on the speed of hardware and the cost of faster circuits is quite high, we have to adopt the 2nd option. This second approach is Instruction Level Parallelism.

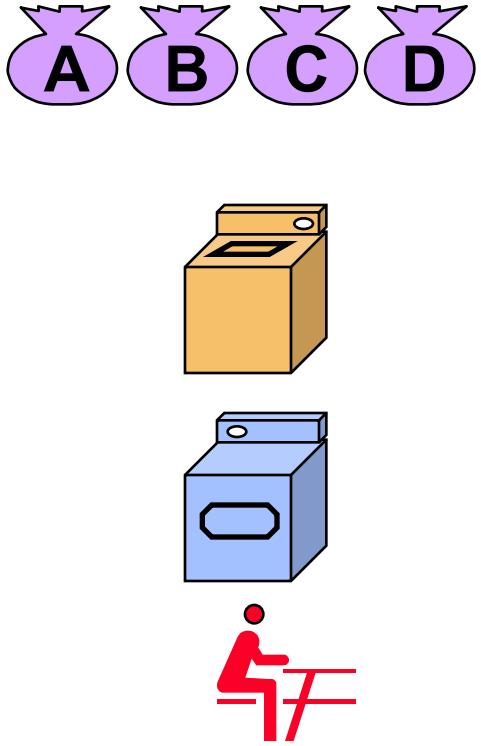
Pipelining : Pipelining is a technique for implementing instruction-level parallelism within a single processor.

It is a process of arrangement of hardware elements of the CPU such that its overall performance is increased with simultaneous execution of more than one instruction taking place in a pipelined processor.

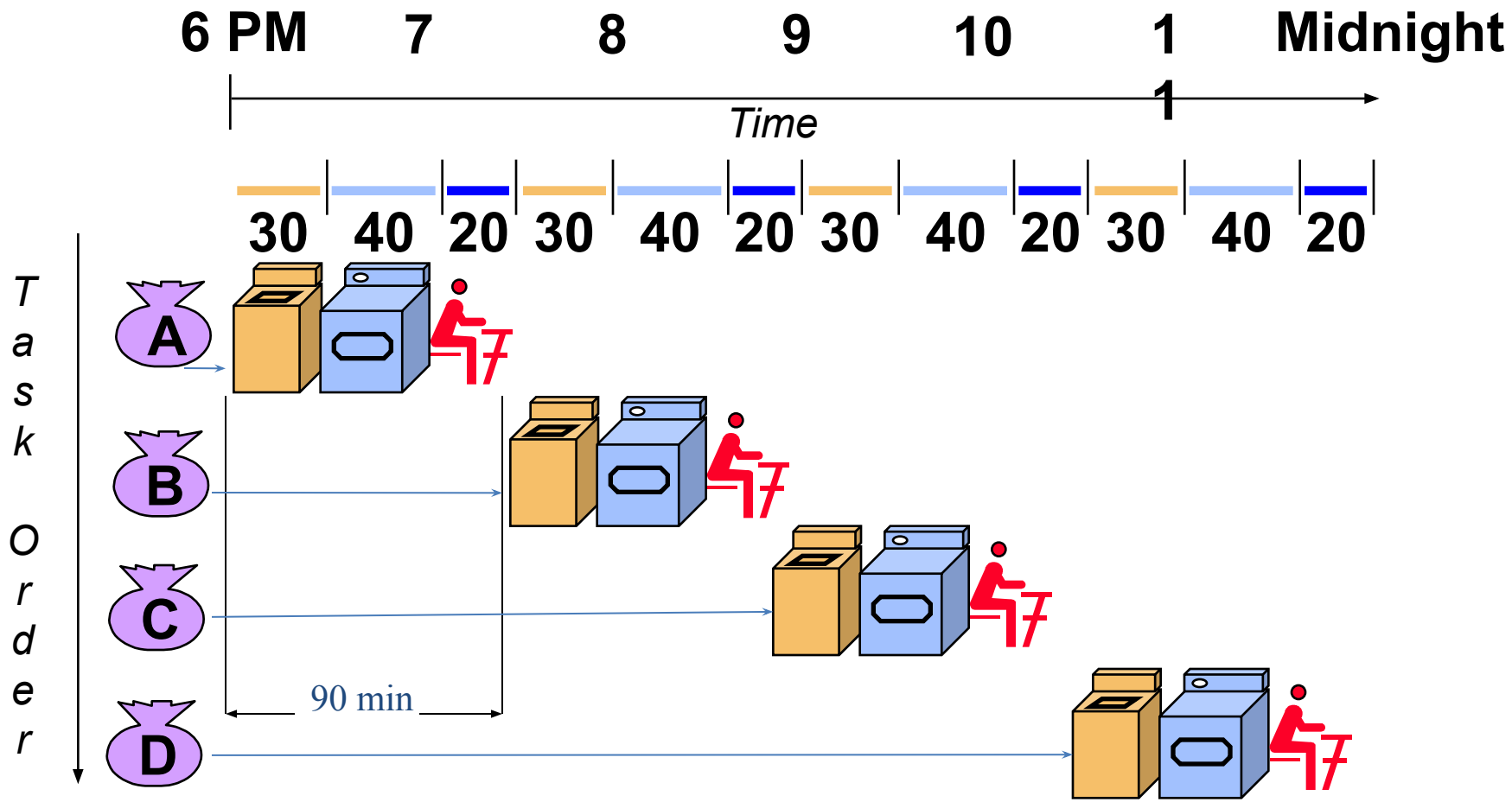
Pipelining attempts to keep every part of the processor busy with some instruction by dividing incoming instructions into a series of sequential steps performed by different processor units with different parts of instructions processed in parallel.

Pipelining: Laundry Example

- Small laundry has **one washer, one dryer and one operator**, it takes 90 minutes to finish one load:
  - Washer takes 30 minutes
  - Dryer takes 40 minutes
  - "operator folding" takes 20 minutes

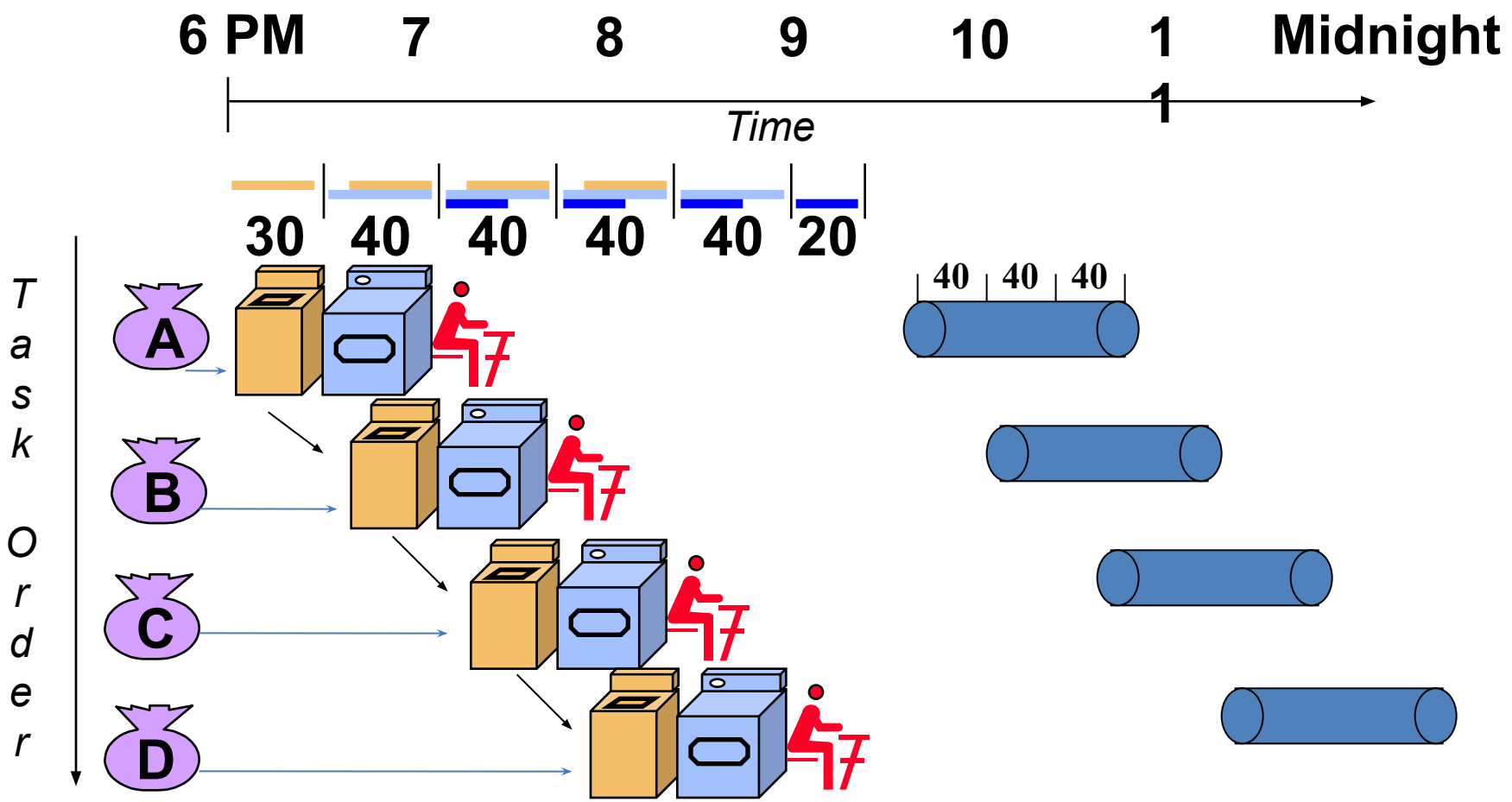


Sequential Laundry



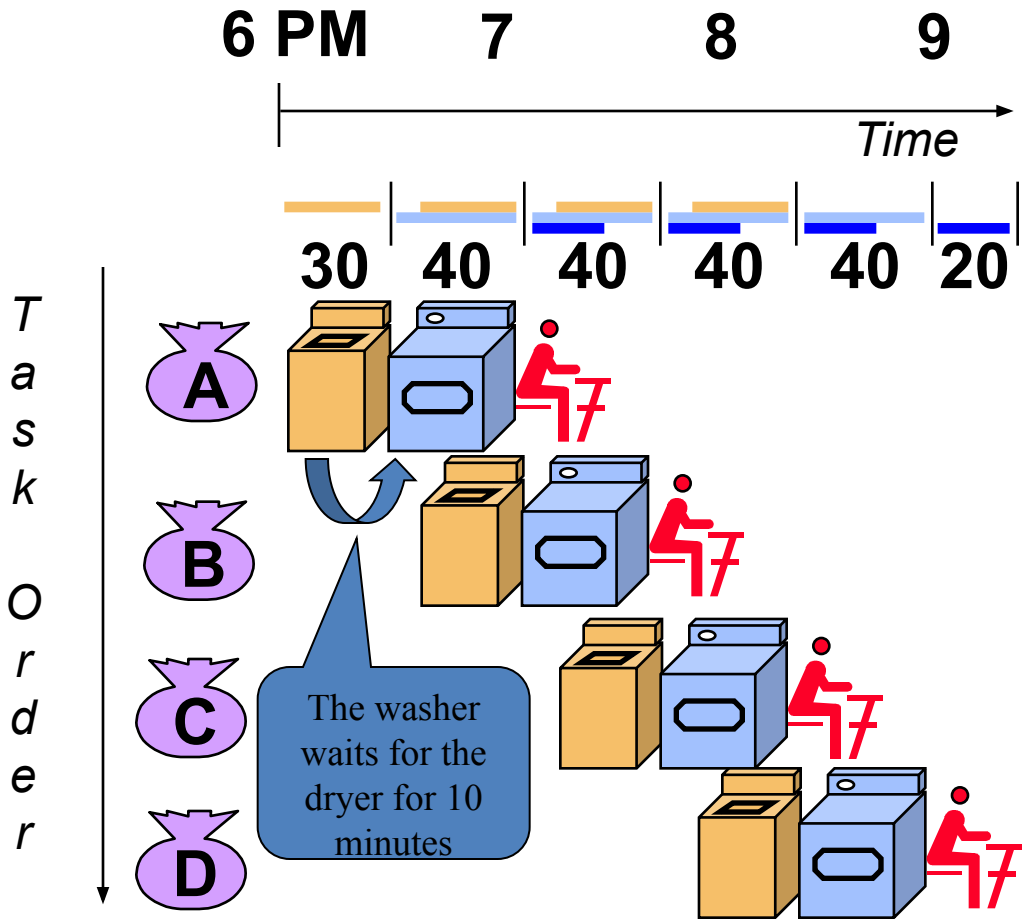
- This operator scheduled his loads to be delivered to the laundry every 90 minutes which is the time required to finish one load. In other words, he will not start a new task unless he is already done with the previous task
- The process is sequential. Sequential laundry takes 6 hours for 4 loads

Efficiently scheduled laundry: Pipelined Laundry



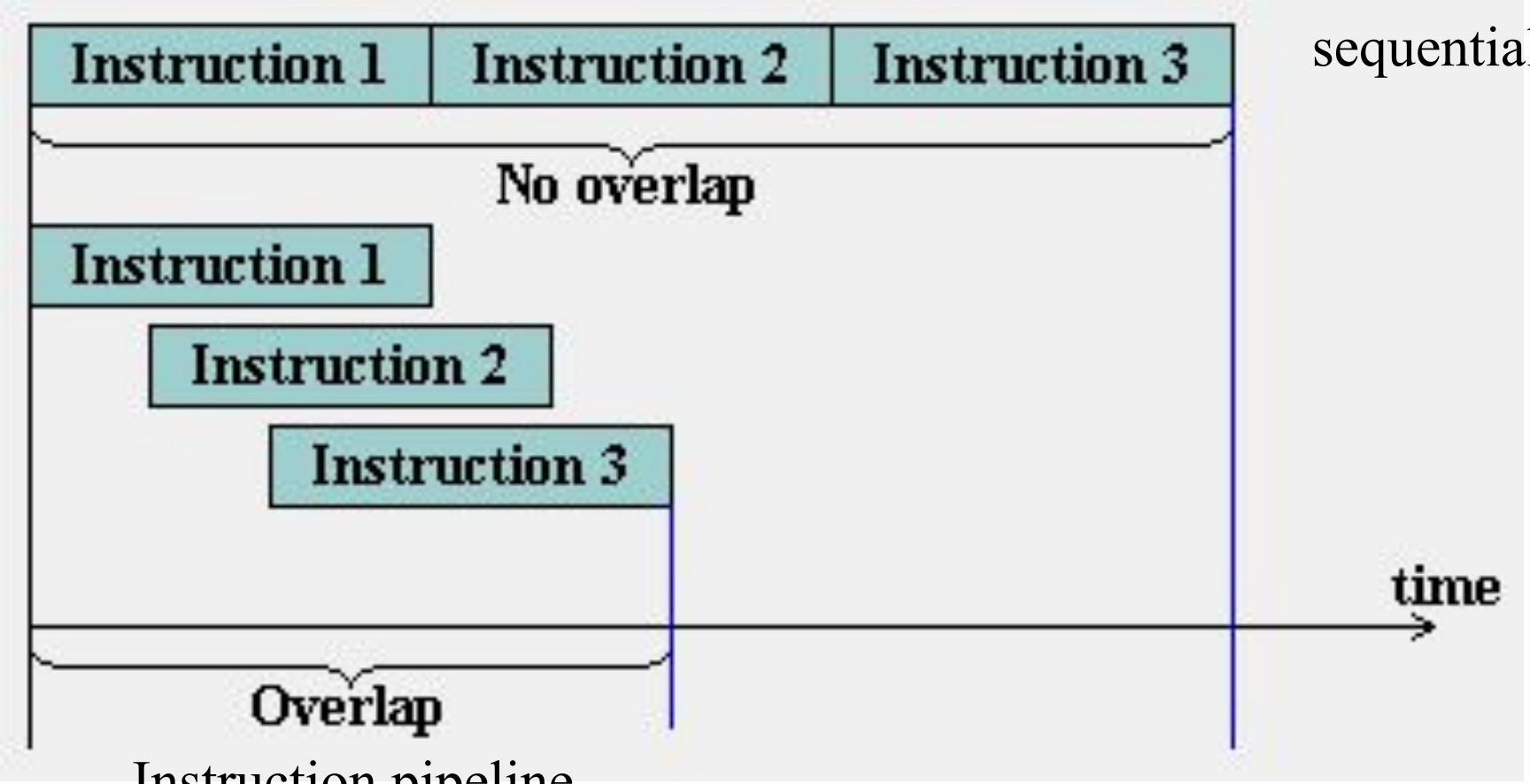
- Another operator asks for the delivery of loads to the laundry every 40 minutes!
- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Facts



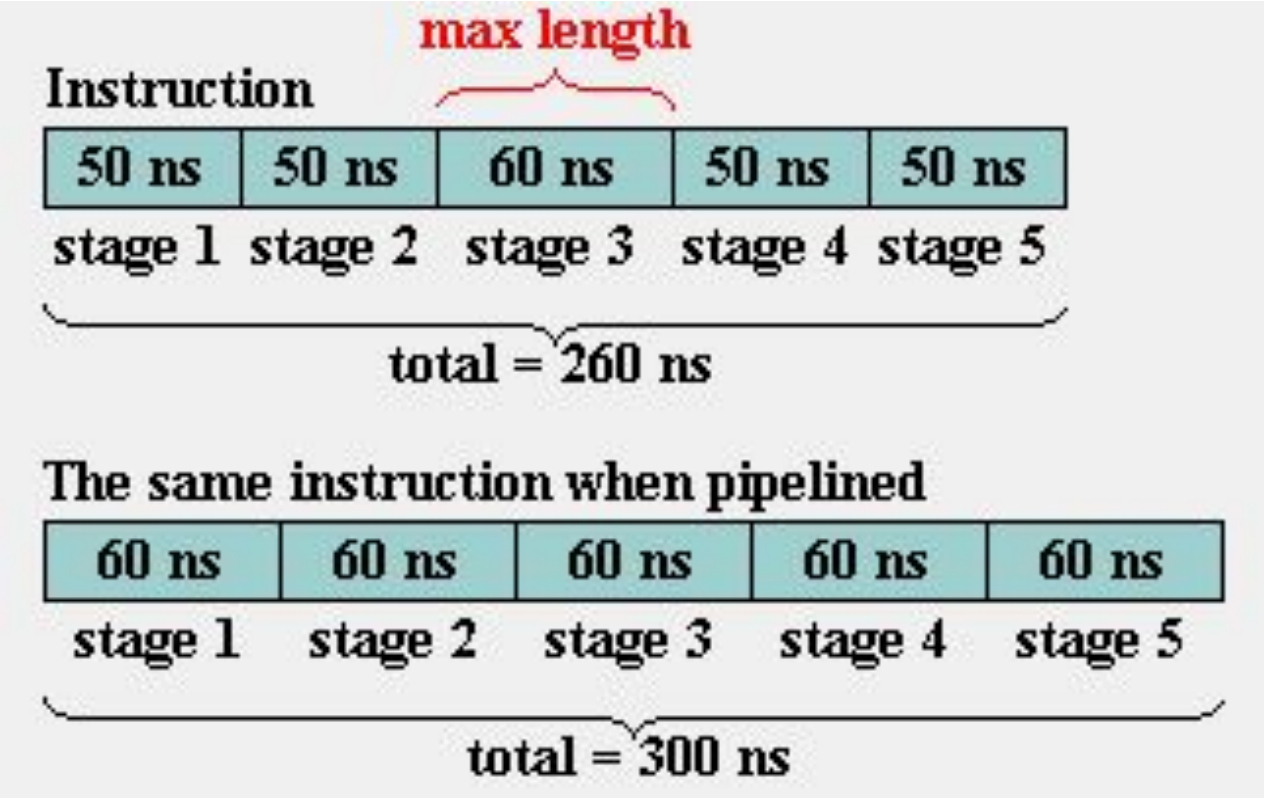
- Multiple tasks operating simultaneously
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Potential speedup = Number of pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

Instruction pipeline versus sequential processing



Instruction pipeline

Instruction pipeline (Contd.)



sequential processing is

faster for few instructions

Performance of Pipelining system

Throughput of the instruction pipeline is determined **by how often an instruction exits the pipeline**. Pipelining does not decrease the time for individual instruction execution. Instead, it increases instruction throughput.

Machine cycle . The time required to move an instruction one step further in the pipeline. The **length** of the machine cycle is determined by the time required for the slowest pipe stage.

Performance Measurement

- n : instructions
- k: stages in pipeline
- τ : clockcycle
- T_k: total time

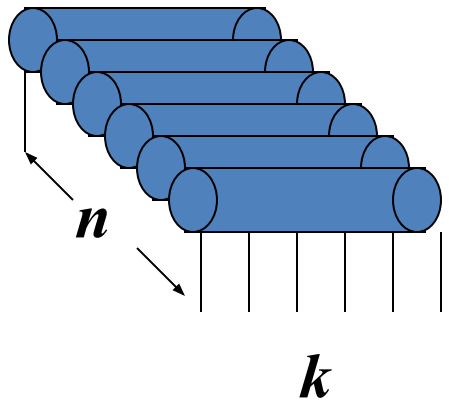
n is equivalent to number of loads in the laundry example

k is the stages (washing, drying and folding.

Clock cycle is the slowest task time

$$T_k = (k + (n-1))\tau$$

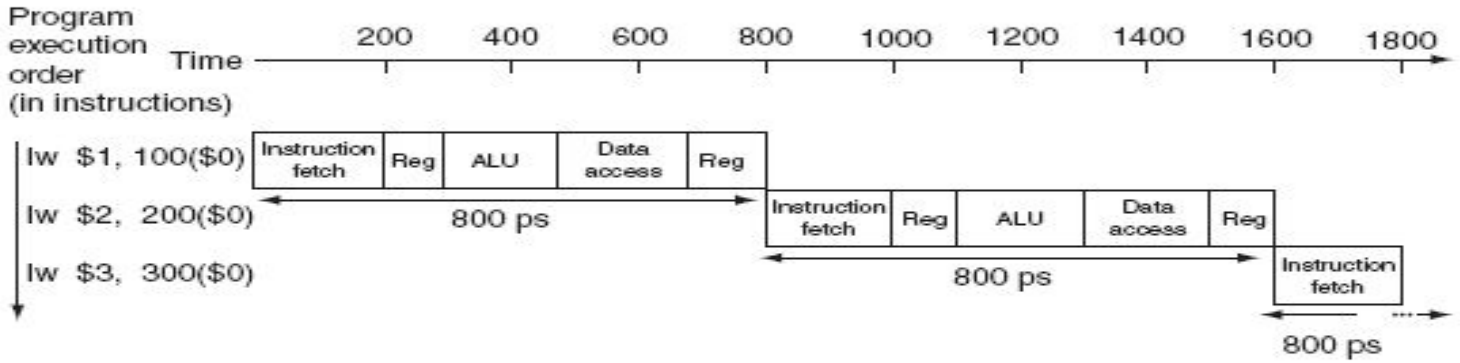
$$Speedup = \frac{T_1}{T_k} = \frac{nk}{k + (n-1)}$$

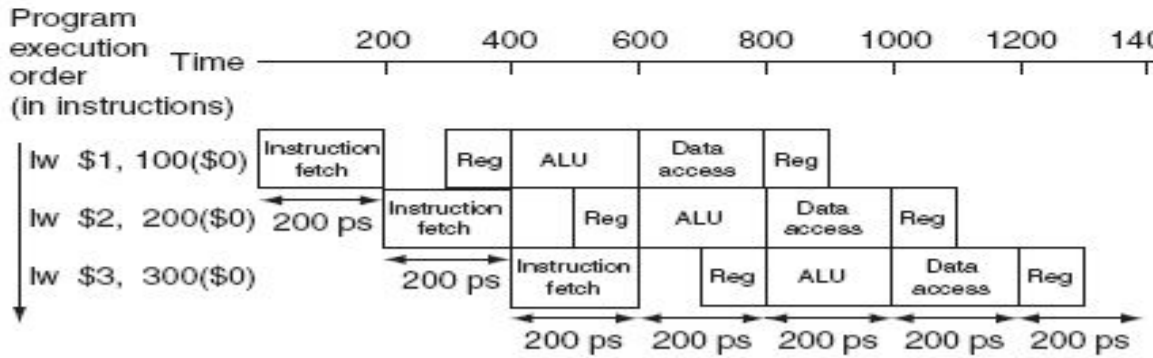


Pipeline Datapath for MIPS Instruction

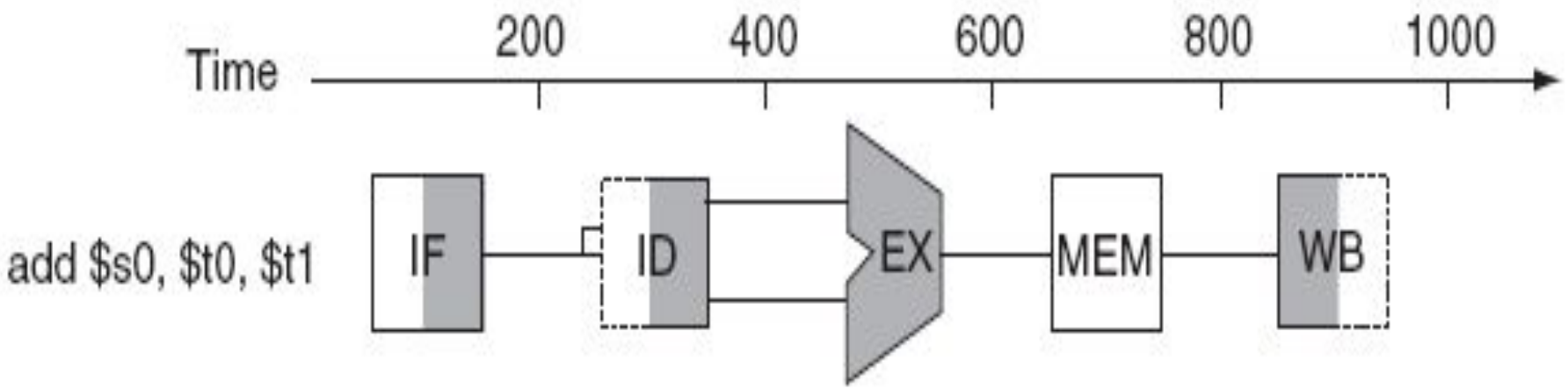
Time Taken by each MIPS Instruction: Sequential Vs Pipeline Execution

Instruction class	fetch	read	operation	access	write	time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

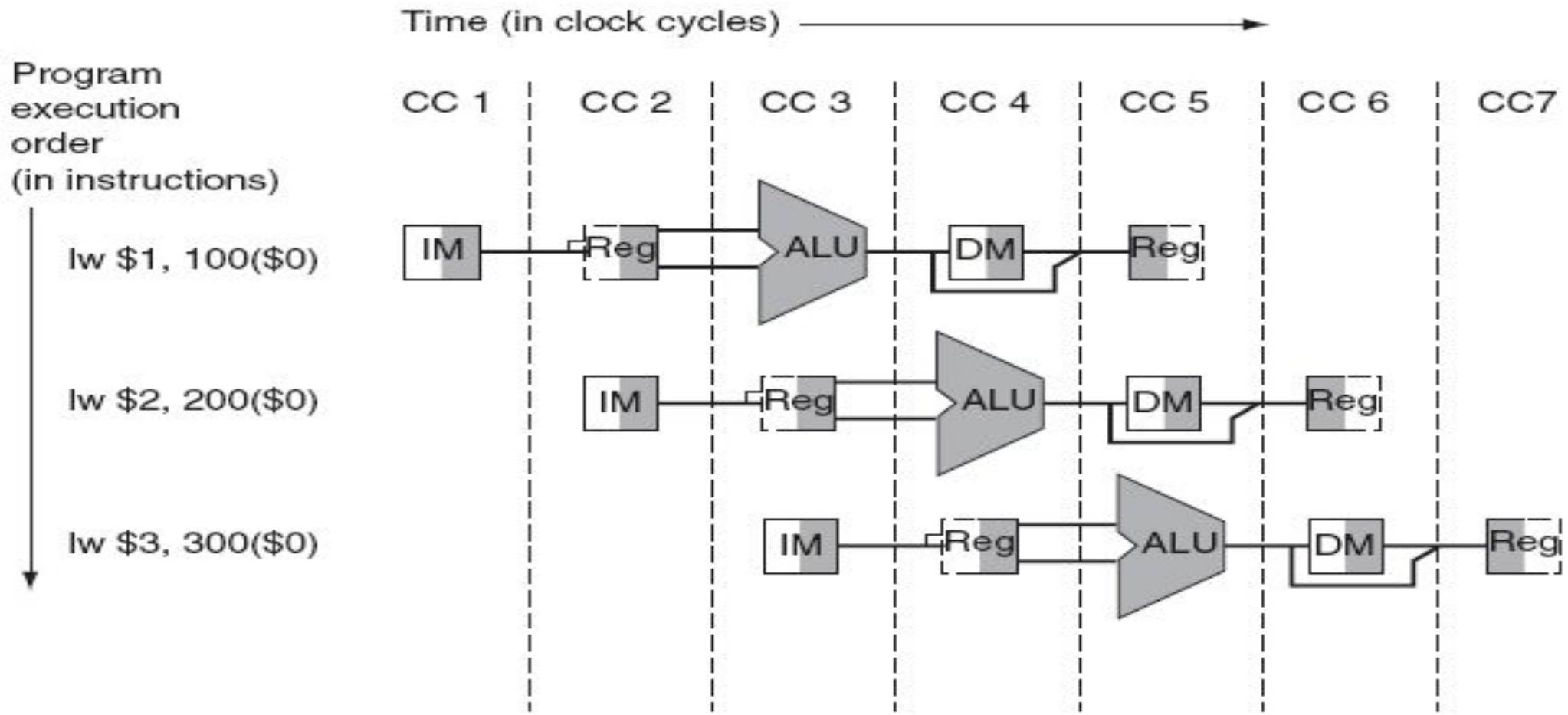




Graphical Representation of ILP



Instruction Execution in Single Cycle Data path Assuming Pipelining



Hazards in
Pipelining System: Data, Structure & Branch

Data Hazards

Data hazards occur when instructions that exhibit data dependence modify data in different stages of a pipeline. Ignoring potential data hazards can result in race conditions (also termed race hazards).

There are three situations in which a data hazard can occur:

- read after write (RAW), a *true dependency*
- write after read (WAR), an *anti-dependency*
- write after write (WAW), an *output dependency*
- Read after read (RAR) is not a hazard case.

Consider two instructions `i1` and `i2`, with `i1` occurring before `i2` in program order.

Read after write (RAW)

(i2 tries to read a source before i1 writes to it)

i1. R2 <- R5 + R8

i2. R4 <- R2 + R8

Write after read (WAR)

(i2 tries to write a destination before it is read by i1)

i1. R4 <- R1 + R5

i2. R5 <- R1 + R2

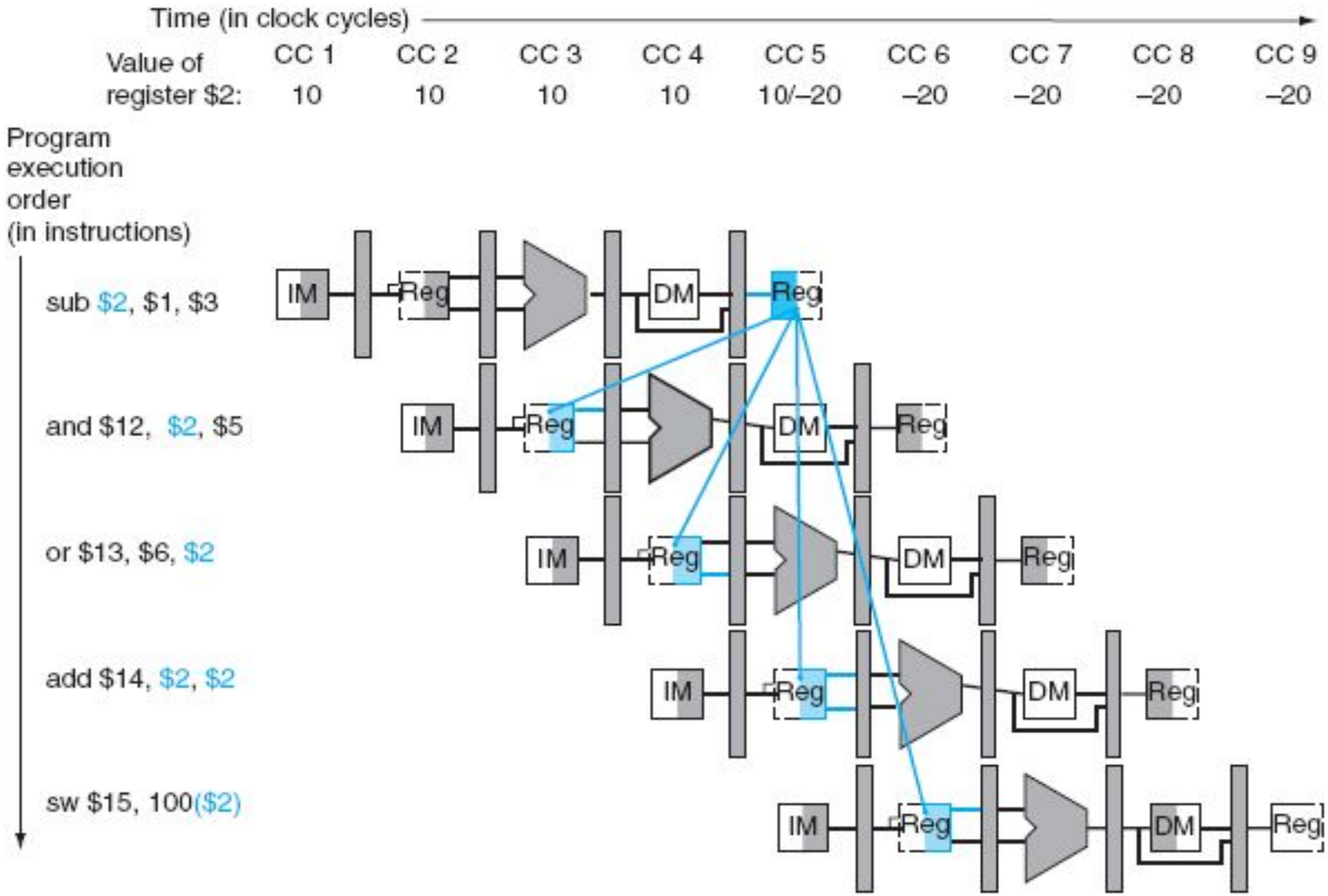
Write after write (WAW)

(i2 tries to write an operand before it is written by i1)

i1. R5 <- R4 + R7

i2. R5 <- R1 + R3

Data Hazard

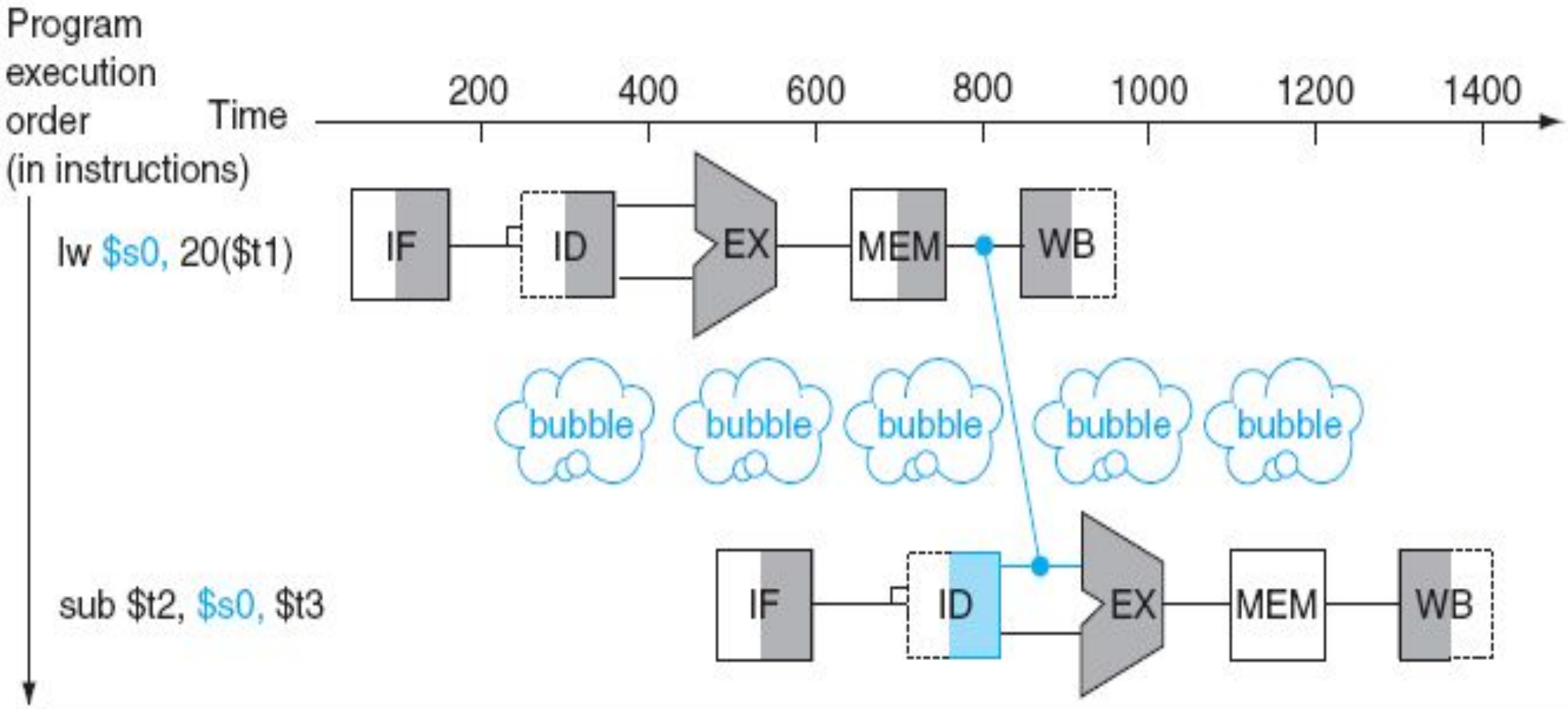

Time (in clock cycles)
Value of register $2:
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
10 10 10 10 10/–20 –20 –20 –20 –20
Program execution order (in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
IM
Reg
DM

Data Hazards and Stalls

In the design of pipelined computer processors, a **pipeline stall** is a delay in execution of an instruction in order to resolve a hazard.

In a Von Neumann architecture which uses the program counter (PC) register to determine the current instruction being fetched in the pipeline, **to prevent new instructions from being fetched when an instruction in the decoding stage has been stalled**, the value in the PC register and the instruction in the fetch stage are preserved to prevent changes. The values are preserved until the instruction causing the conflict has passed through the execution stage. Such an event **is often called a bubble.**

Stall when R-format dependent instruction follow a Load Instruction



Data Hazard Solution: Operand Forwarding

- **Operand forwarding** (or **data forwarding**) is an optimization in pipelined CPUs to limit performance deficits which occur due to pipeline stalls. A data hazard can lead to a pipeline stall when the current operation has to wait for the results of an earlier operation which has not yet finished.

Example

ADD A B C #A=B+C

SUB D C A #D=C-A

If these two assembly pseudocode instructions run in a pipeline, after fetching and decoding the second instruction, the pipeline stalls, waiting until the result of the addition is written and read.

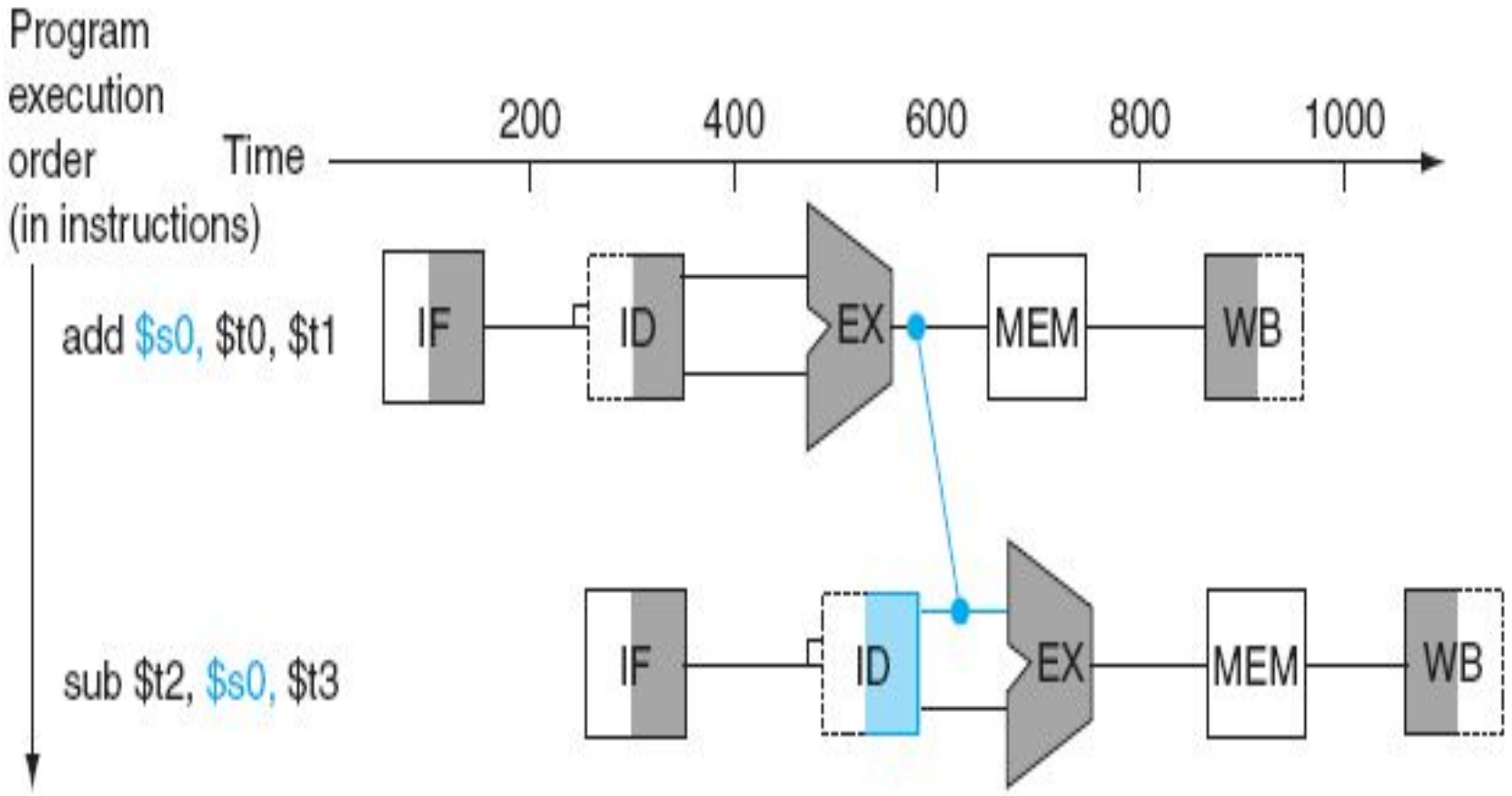
Without operand forwarding

1	2	3	4	5	6	7	8
Fetch ADD	Decode ADD	Read Operands ADD	Execute ADD	Write result			
	Fetch SUB	Decode SUB	*stall*	*stall*	Read Operands SUB	Execute SUB	Write result

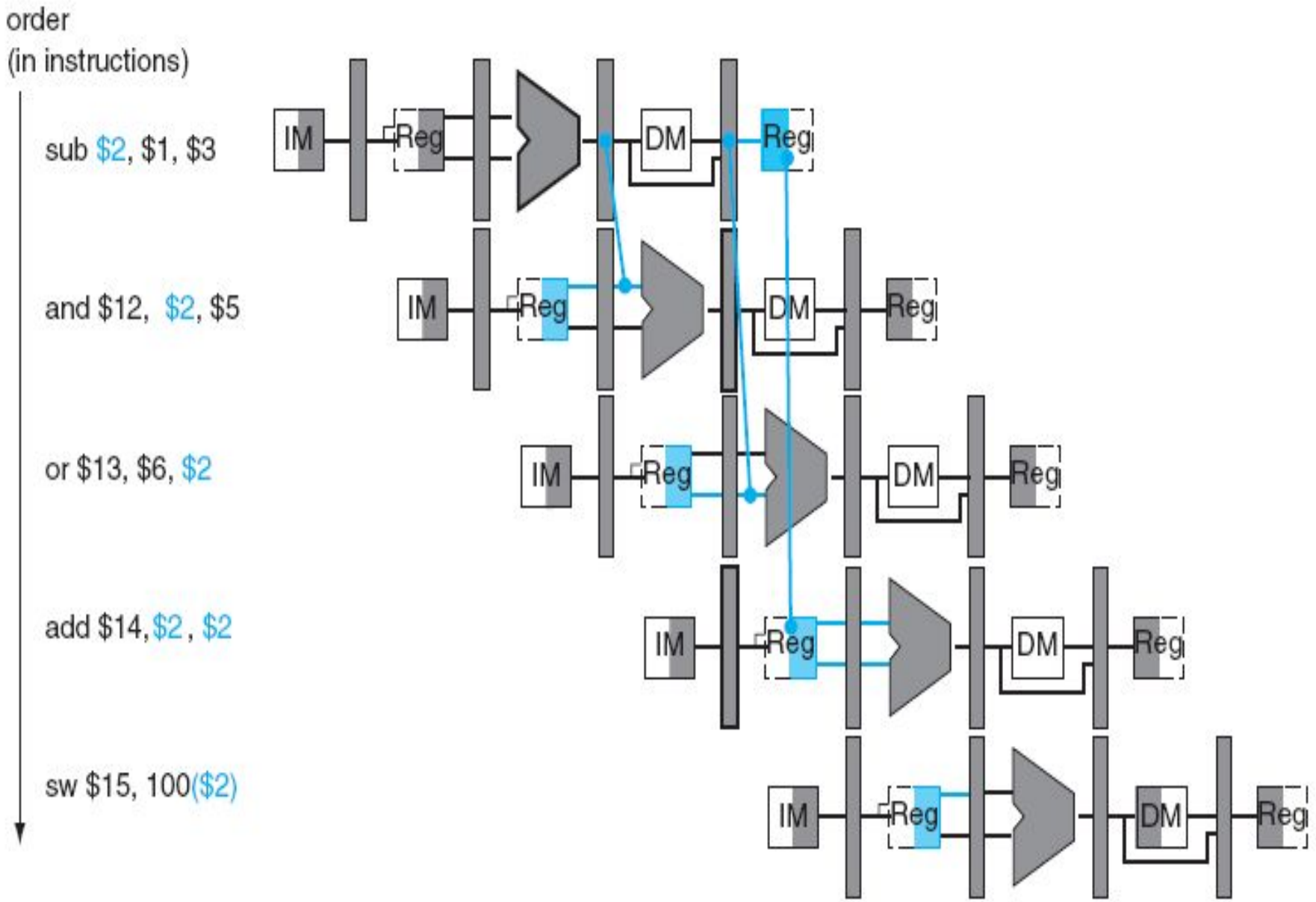
With operand forwarding

1	2	3	4	5	6	7
Fetch ADD	Decode ADD	Read Operands ADD	Execute ADD	Write result		
	Fetch SUB	Decode SUB	*stall*	Read Operands SUB: use result from previous operation	Execute SUB	Write result

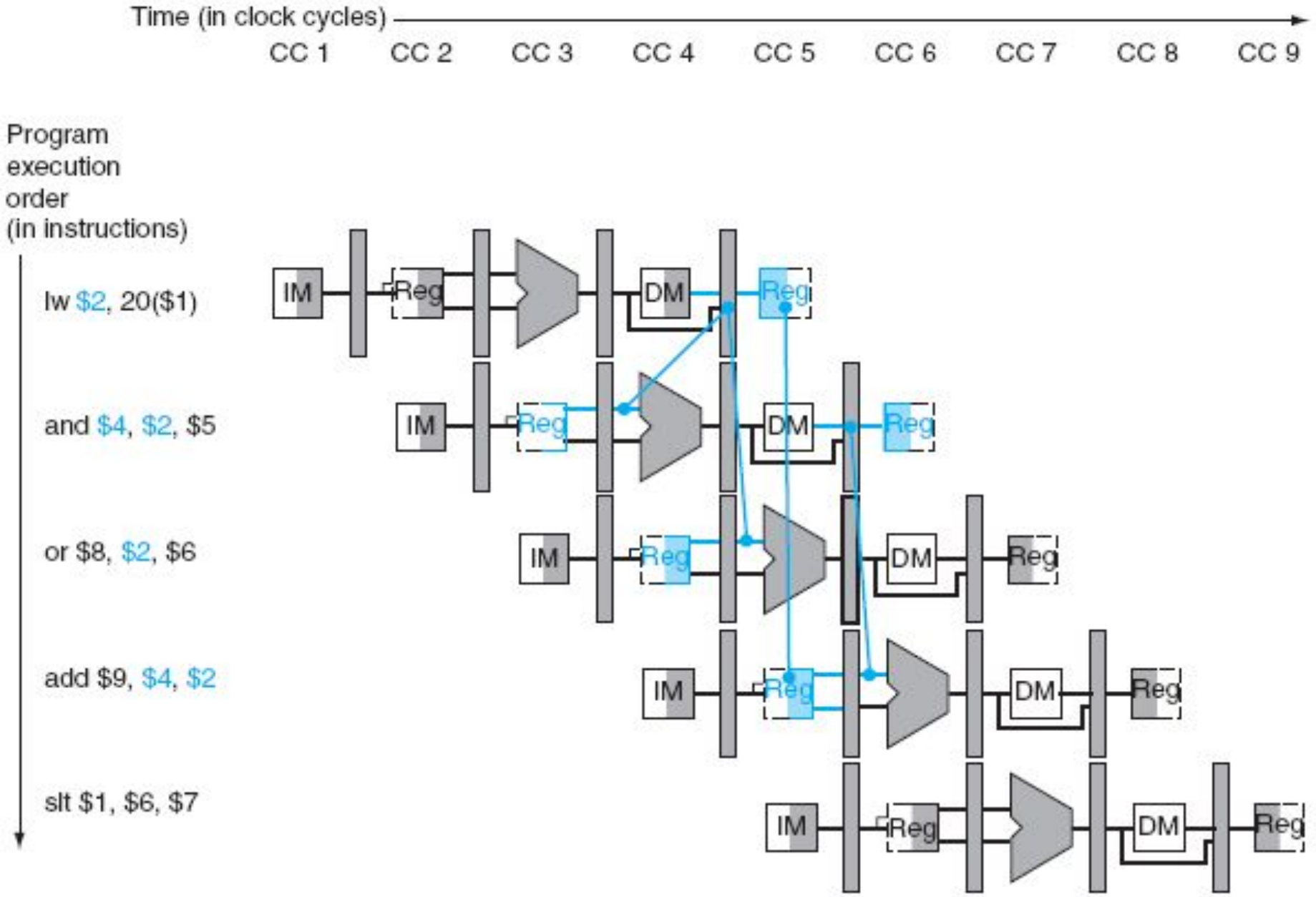
Data Hazard Solution: Operand Forwarding



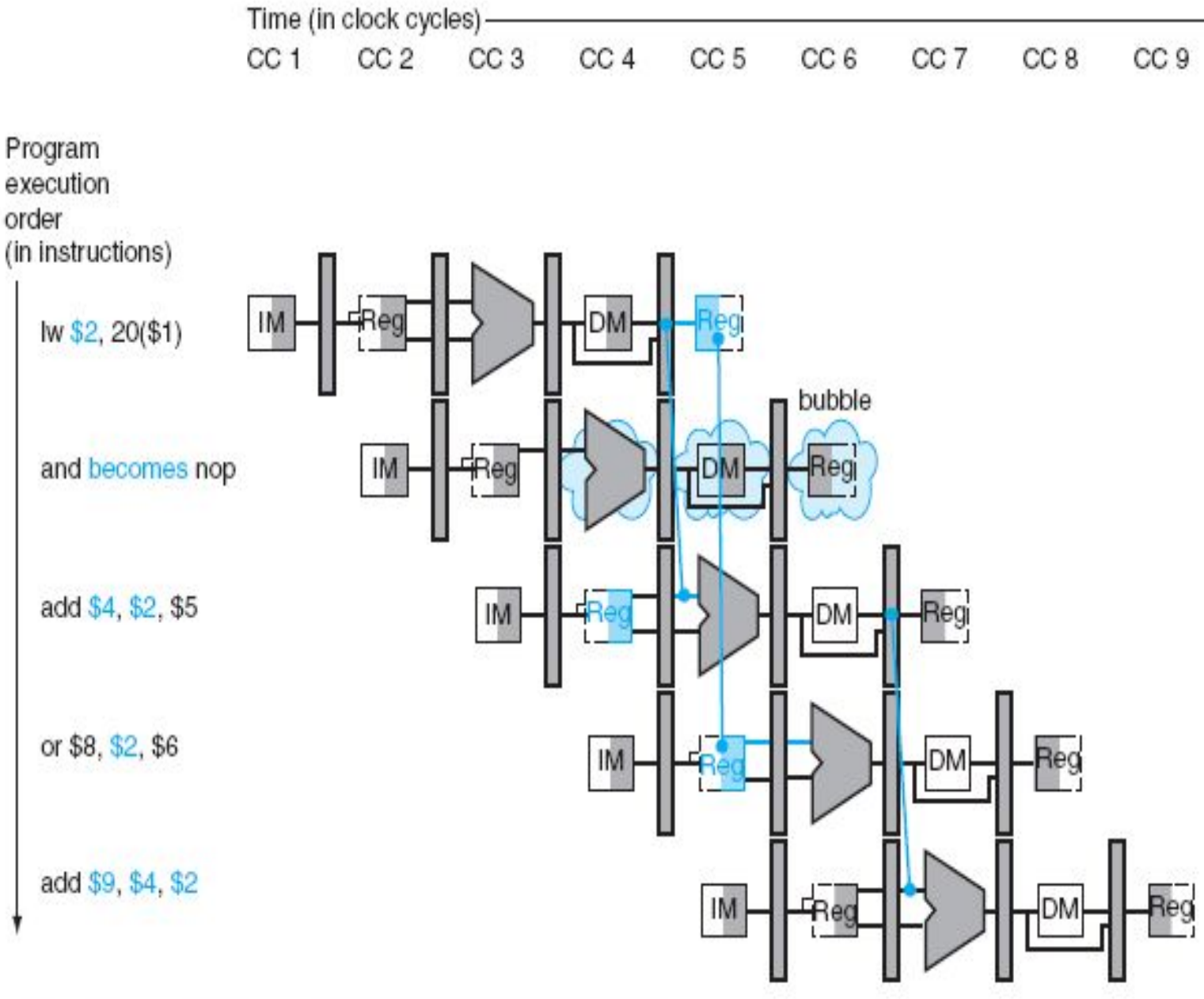
Operand Forwarding



Data Hazard Solution using Operand Forwarding and Stall



Data Hazard Solution using Operand Forwarding and Stall (Cont.)



Structural Hazards

- A structural hazard occurs when two (or more) instructions that are already in pipeline need the same resource. The result is that instruction must be executed in series rather than parallel for a portion of pipeline. Structural hazards are sometimes referred to as resource hazards.

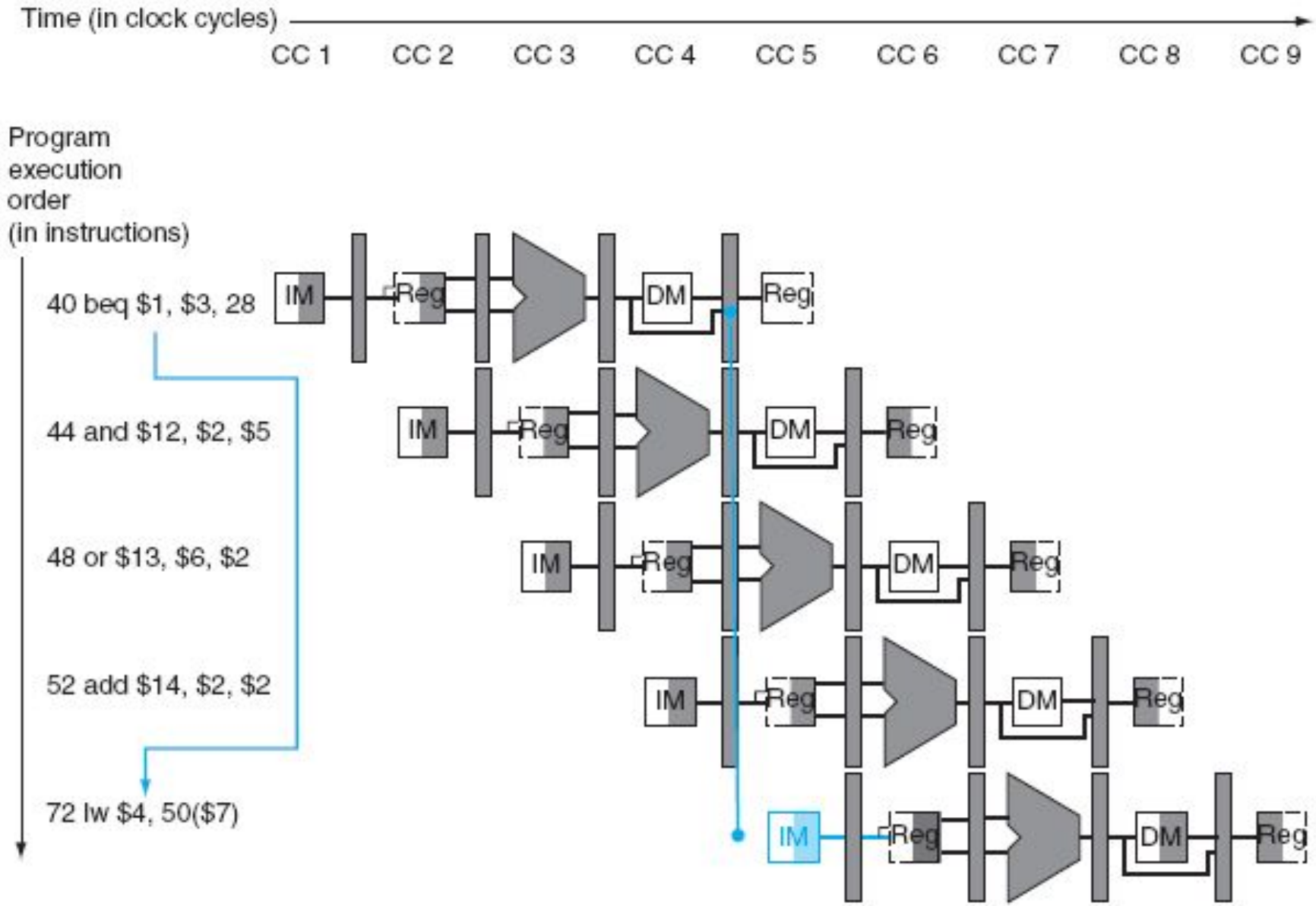
- A situation in which multiple instructions are ready to enter the execute instruction phase and there is a single ALU (Arithmetic Logic Unit). One solution to such resource hazard is to increase available resources, such as having multiple ports into main memory and multiple ALU (Arithmetic Logic Unit) units.

Branch/Conditional Hazards

To avoid control hazards microarchitectures can:

- insert a *pipeline bubble*, guaranteed to increase latency, or use branch prediction and essentially make educated guesses about which instructions to insert, in which case a *pipeline bubble* will only be needed in the case of an incorrect prediction
- In the event that a branch causes a pipeline bubble after incorrect instructions have entered the pipeline, care must be taken to prevent any of the wrongly-loaded instructions from having any effect on the processor state excluding energy wasted processing them before they were discovered to be loaded incorrectly.

Control or Branch Hazards



Two schemes for resolving control hazards.

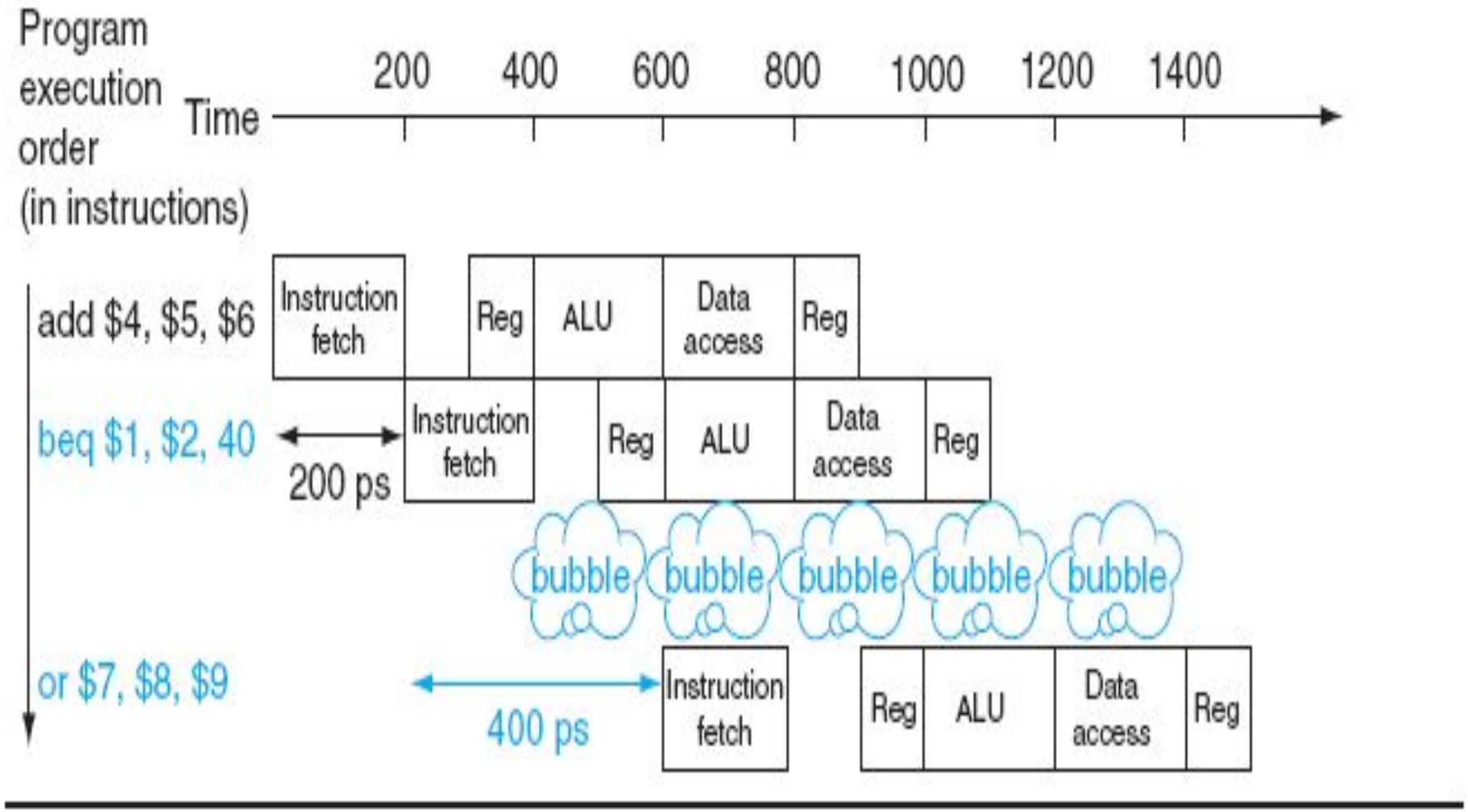
1. Assume Branch Not Taken: A common improvement over branch stalling is to assume that the branch will not be taken and thus continue execution down the sequential instruction stream. If the branch is taken, the instructions that are being fetched and decoded must be discarded. Execution continues at the branch target.

2. Reducing the Delay of Branches: One way to improve branch performance is to reduce the cost of the taken branch. Thus far we have assumed the next PC for a branch is selected in the MEM stage, but if we move the branch execution earlier in the pipeline, then fewer instructions need be flushed.

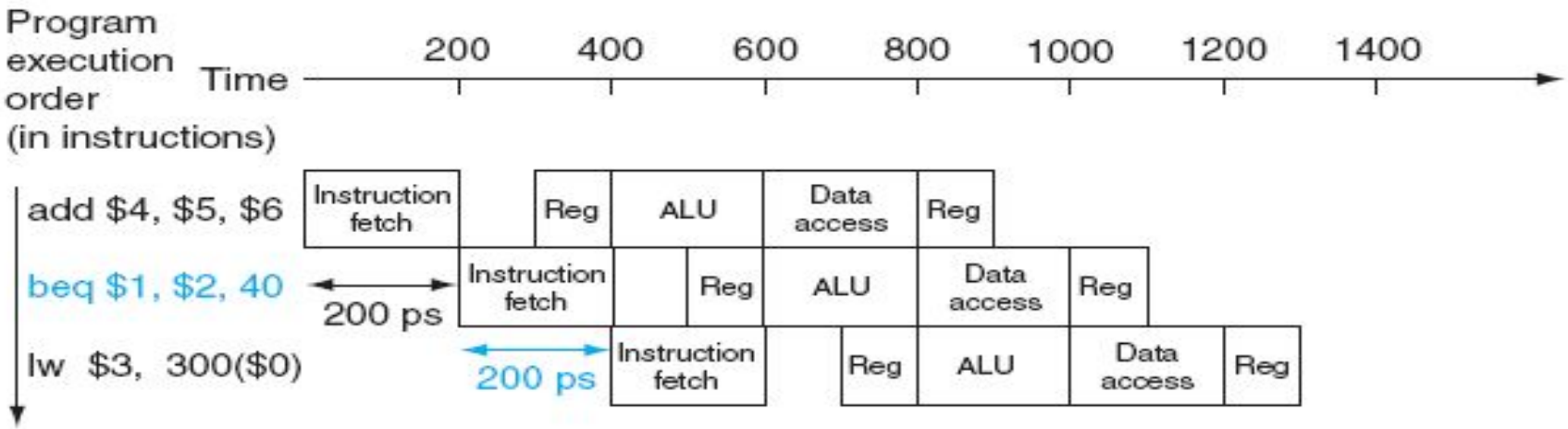
Moving the branch decision up requires two actions to occur earlier:

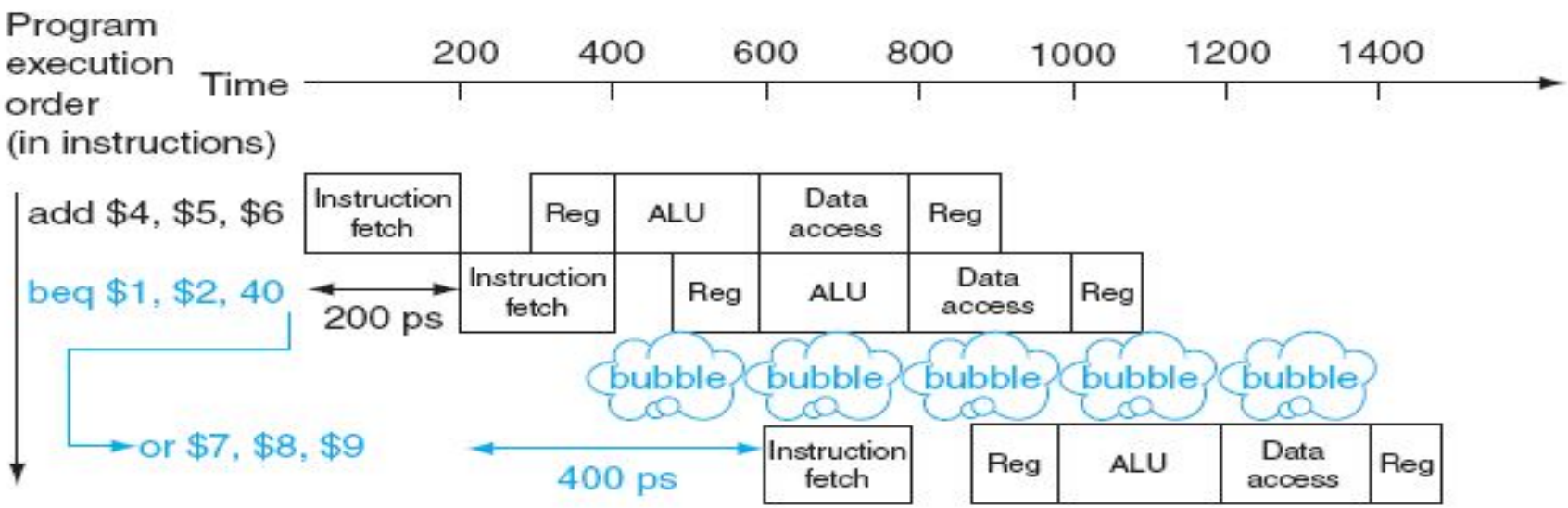
- Computing the branch target address
- Evaluating the branch decision.

Stalling in Branch Hazard



Prediction: Branches are Never Taken





GATE-2005

A 5 stage pipelined CPU has the following sequence of stages:

- IF – instruction fetch from instruction memory
- RD – Instruction decode and register read
- EX – Execute: ALU operation for data and address computation
- MA – Data memory access – for write access, the register read at RD state is used.
- WB – Register write back

Consider the following sequence of instructions:

- I_1: $L\ R0, loc\ 1; R0 \Leftarrow M[loc1]$
- I_2: $A\ R0, R0; R0 \Leftarrow R0 + R0$
- I_3: $S\ R2, R0; R2 \Leftarrow R2 - R0$

Let each stage take one clock cycle.

What is the number of clock cycles taken to complete the above sequence of instructions starting from the fetch of I_1?

A. 8
B. 10
C. 12
D. 15

GATE 2015

Consider the sequence of machine instruction given below:

```
MUL   R5, R0, R1
DIV   R6, R2, R3
ADD   R7, R5, R6
SUB   R8, R7, R4
```

In the above sequence, $R0$ to $R8$ are general purpose registers. In the instructions shown, the first register shows the result of the operation performed on the second and the third registers. This sequence of instructions is to be executed in a pipelined instruction processor with the following 4 stages: (1) Instruction Fetch and Decode (IF), (2) Operand Fetch (OF), (3) Perform Operation (PO) and (4) Write back the result (WB). The IF, OF and WB stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD and SUB instruction, 3 clock cycles for MUL instruction and 5 clock cycles for DIV instruction. The pipelined processor uses operand forwarding from the PO stage to the OF stage. The number of clock cycles taken for the execution of the above sequence of instruction is _____.

GATE 2010

A 5—stage pipelined processor has Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) and Write Operand (WO) stages. The IF, ID, OF and WO stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD and SUB instructions, 3 clock cycles for MUL instruction and 6 clock cycles for DIV instruction respectively. Operand forwarding is used in the pipeline. What is the number of clock cycles needed to execute the following sequence of instructions?

Instruction	Meaning of instruction
t_0: MUL R_2, R_0, R_1	$\mathrm{R_2} \leftarrow \mathrm{R_0} * \mathrm{R_1}$
t_1: DIV R_5, R_3, R_4	$\mathrm{R_5} \leftarrow \mathrm{R_3} / \mathrm{R_4}$
t_2: ADD R_2, R_5, R_2	$\mathrm{R_2} \leftarrow \mathrm{R_5} + \mathrm{R_2}$
t_3 : SUB $\mathrm{R_5}, \mathrm{R_2}, \mathrm{R_6}$	$\mathrm{R_5} \leftarrow \mathrm{R_2} - \mathrm{R_6}$

A. 13
B. 15
C. 17
D. 19

GATE 2016

An instruction pipeline consists of following 5 stages:

IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MA = Memory Access and WB = Register Write Back.

Now consider the following code:

```
1. LOAD    R8, 0(R5);     R8 = memory [R5]
2. LOAD    R9, 4(R5);     R9 = memory [R5 + 4]
3. ADD     R7, R8, R9;    R7 = R8 + R9
4. SUB     R6, R7, R8;    R6 = R7 – R8
```

Assume that each stage takes 1 clock cycle for all the instructions. How many cycles are required to execute the code, without operand forwarding over a bypass network?

A. 9
B. 10
C. 11
D. 14

GATE 2017

Question Number : 50 **Correct : 2 Wrong : 0**

Instruction execution in a processor is divided into 5 stages. *Instruction Fetch* (IF). *Instruction Decode* (ID). *Operand Fetch* (OF). *Execute* (EX). and *Write Back* (WB). These stages take **5, 4, 20, 10,** and **3 nanoseconds (ns)** respectively. A pipelined implementation of the processor requires buffering between each pair of consecutive stages with a delay of **2 ns**. Two pipelined implementations of the processor are contemplated:

(i) a naive pipeline implementation (NP) with 5 stages and
(ii) an efficient pipeline (EP) where the OF stage is divided into stages OF1 and OF2 with execution times of **12 ns** and **8 ns** respectively.

The speedup (correct to two decimal places) achieved by EP over NP in executing 20 independent instructions with no hazards is _____ .

GATE 2023

Consider a 3-stage pipelined processor having a delay of 10 ns (nanoseconds), 20 ns, and 14 ns, for the first, second, and the third stages, respectively. Assume that there is no other delay and the processor does not suffer from any pipeline hazards. Also assume that one instruction is fetched every cycle.

The total execution time for executing 100 instructions on this processor is _____ ns.

GATE 2012

Register renaming is done in pipelined processors

A. as an alternative to register allocation at compile time

B. for efficient access to function parameters and local variables

C. to handle certain kinds of hazards

D. as part of address translation

GATE 2003

For a pipelined CPU with a single ALU, consider the following situations

1. The j+1 instruction uses the result of the j-th instruction as an operand
2. The execution of a conditional jump instruction
3. The j-th and j+1 instruction require the ALU at the same time

Which of the above can cause a hazard?

A. 1 and 2 only

B. 2 and 3 only

C. 3 only

D. ALL the three

GATE 2021 set1

A five-stage pipeline has stage delays of 150, 120, 150, 160 and 140 nanoseconds. The registers that are used between the pipeline stages have a delay of 5 nanoseconds each.
The total time to execute 100 independent instructions on this pipeline, assuming there are no pipeline stalls, is _____ nanoseconds.

GATE 2021 set1

Consider the following instruction sequence where register R1, R2 and R3 are general purpose and MEMORY[X] denotes the content at the memory location X.

Instruction	Semantics	Instruction Size (bytes)
MOV R1, (5000)	R1 ← MEMORY[5000]	4
MOV R2, (R3)	R2 ← MEMORY[R3]	4
ADD R2, R1	R2 ← R1 + R2	2
MOV (R3), R2	MEMORY[R3] ← R2	4
INC R3	R3 ← R3 + 1	2
DEC R1	R1 ← R1 – 1	2
BNZ 1004	Branch if not zero to the given absolute address	2
HALT	Stop	1

Assume that the content of the memory location 5000 is 10, and the content of the register R3 is 3000. The content of each of the memory locations from 3000 to 3010 is 50. The instruction sequence starts from the memory location 1000. All the numbers are in decimal format. Assume that the memory is byte addressable.

After the execution of the program, the content of memory location 3010 is

_____