CSE 4305
Computer Organization and Architecture

Processor Structure & Function

Course Teacher: Md. Hamjajul Ashmafee

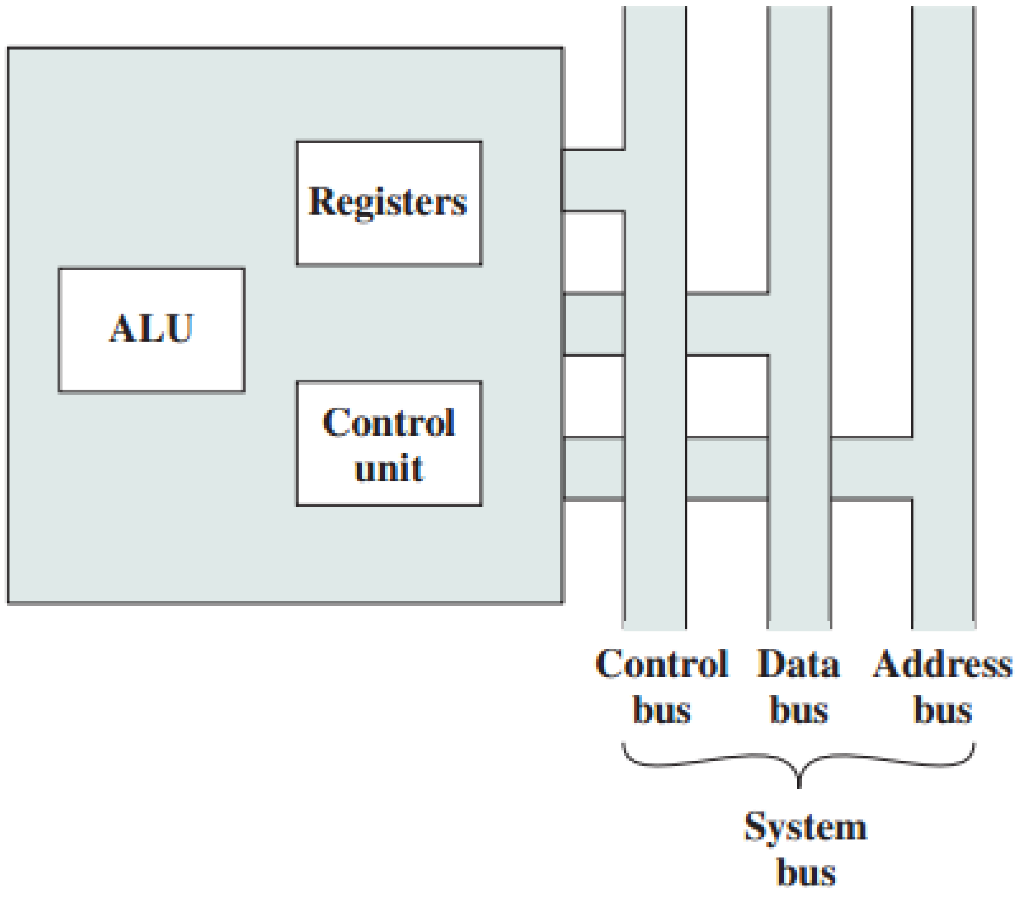
Lecturer, CSE, IUT

Email: ashmafee@iut-dhaka.edu

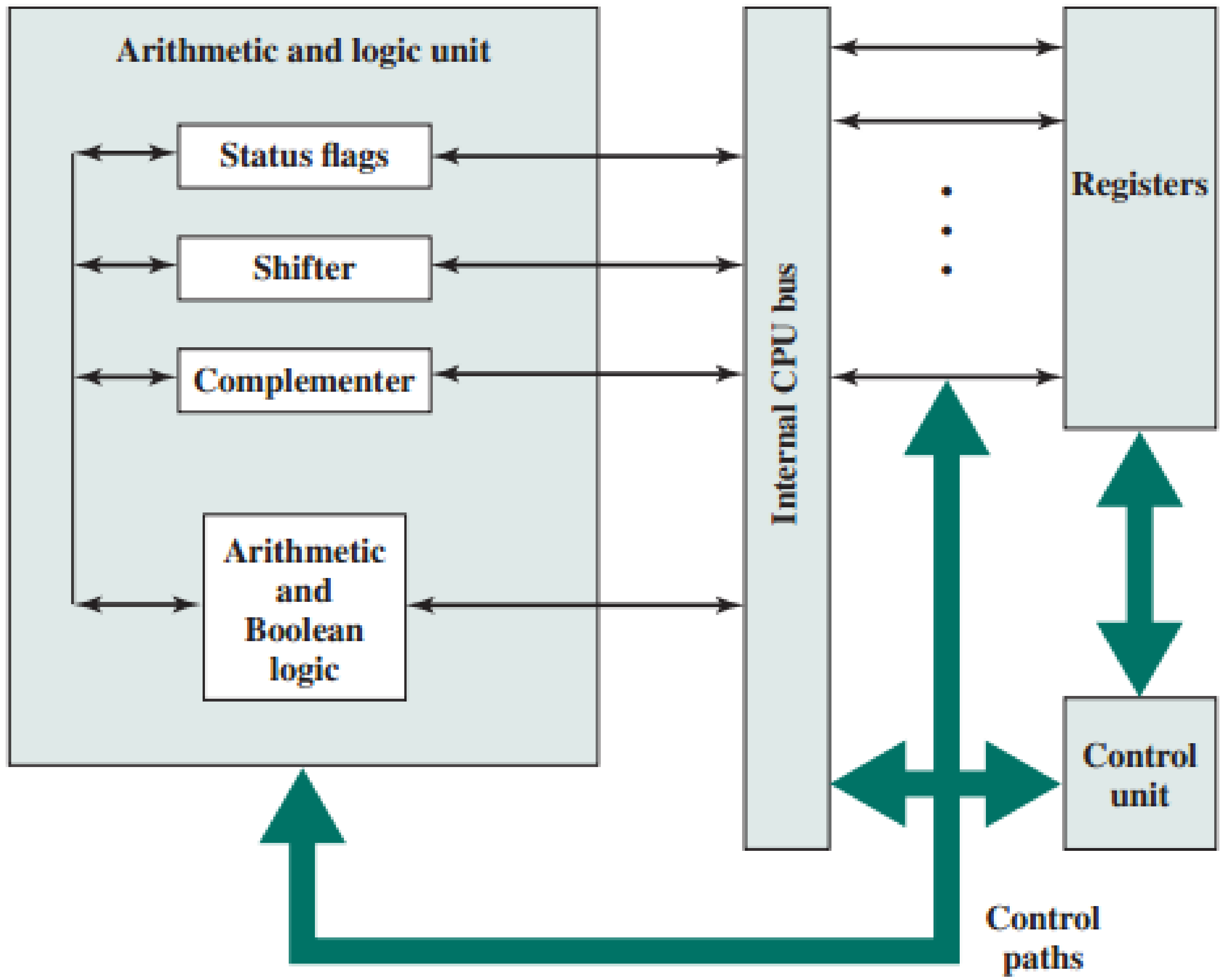
Processor's Functionalities

- Fetch Instruction
- Interpret Instruction
- Fetch Data
- Process Data
- Write Data

Simplified View of Processor



Detailed View of Processor



Registers in Processor

- **User Visible Register** – programmer can use them to reduce memory references during programming
- **Control and Status Register** – used by control unit to control the operation of the processor – also used by OS

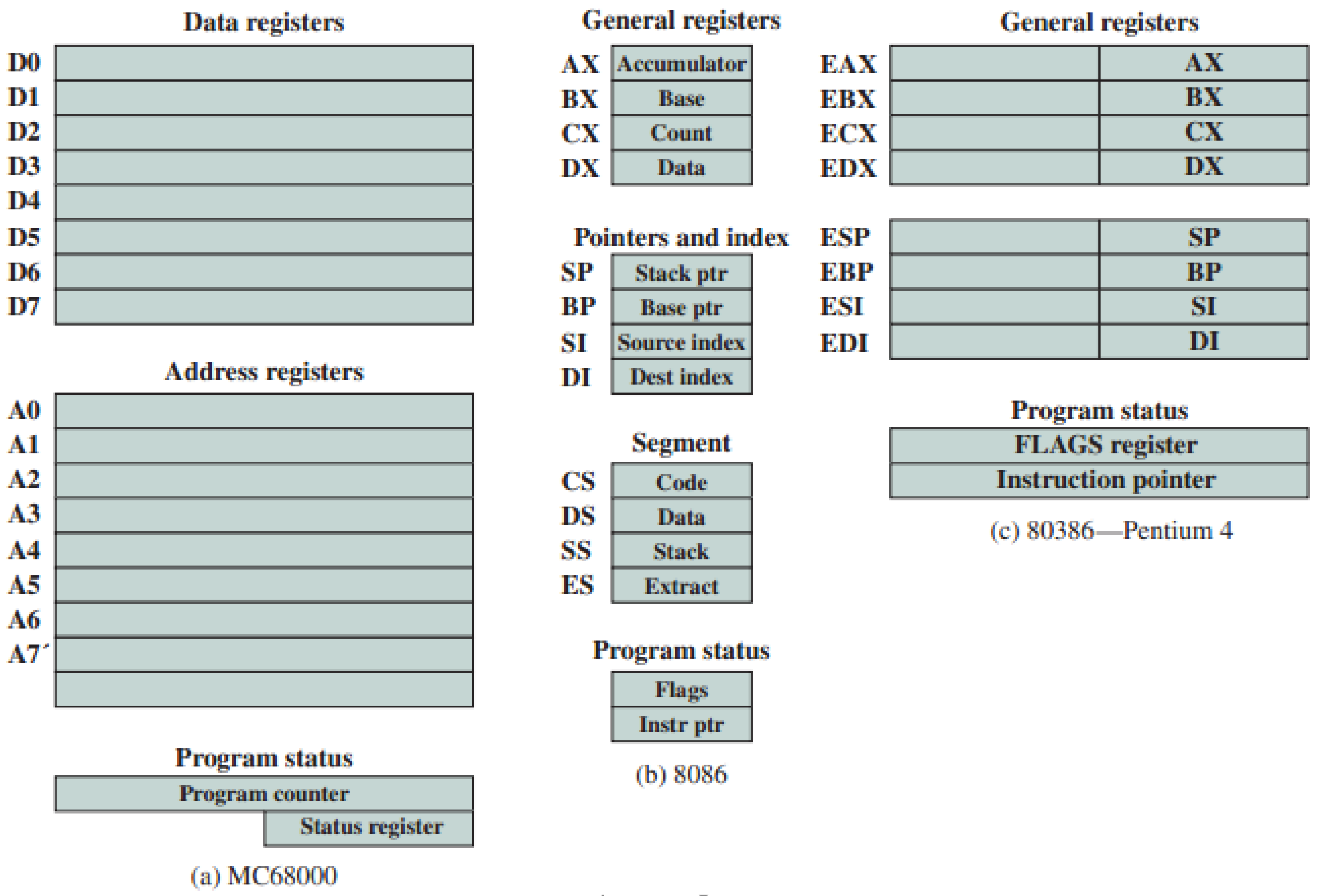
User Visible Register

- **General Purpose Register** – contains the operand for an opcode – some of them are dedicated for floating point and stack operations
- **Data Register** – hold data only without any further calculation on it
- **Address Register** – devoted for addressing mode
  - Segment pointer
  - Index register
  - Stack pointer
- **Condition codes** – also known as flags – bits set by the processor as the result of operations

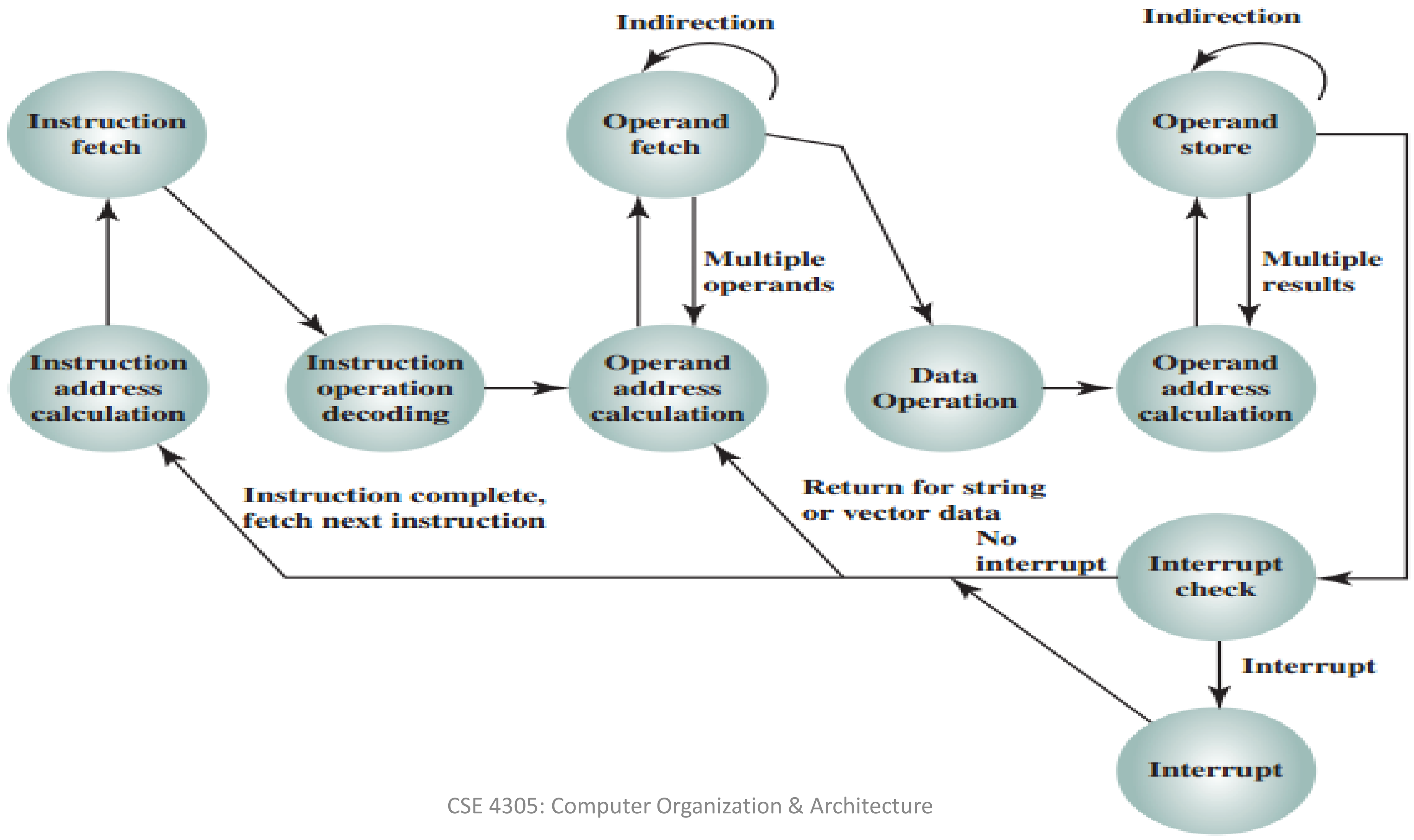
Control and Status Register

- **For instruction execution:**
  - Program Counter (PC)
  - Instruction Register (IR)
  - Memory Address Register (MAR)
  - Memory Buffer Register (MBR)
- **For status information**, processor includes ***Program Status Word (PSW)*** having following flags
  - Sign
  - Zero
  - Carry
  - Equal
  - Overflow
  - Interrupt Enable/ Disable
  - Supervisor

Examples of Register Organizations



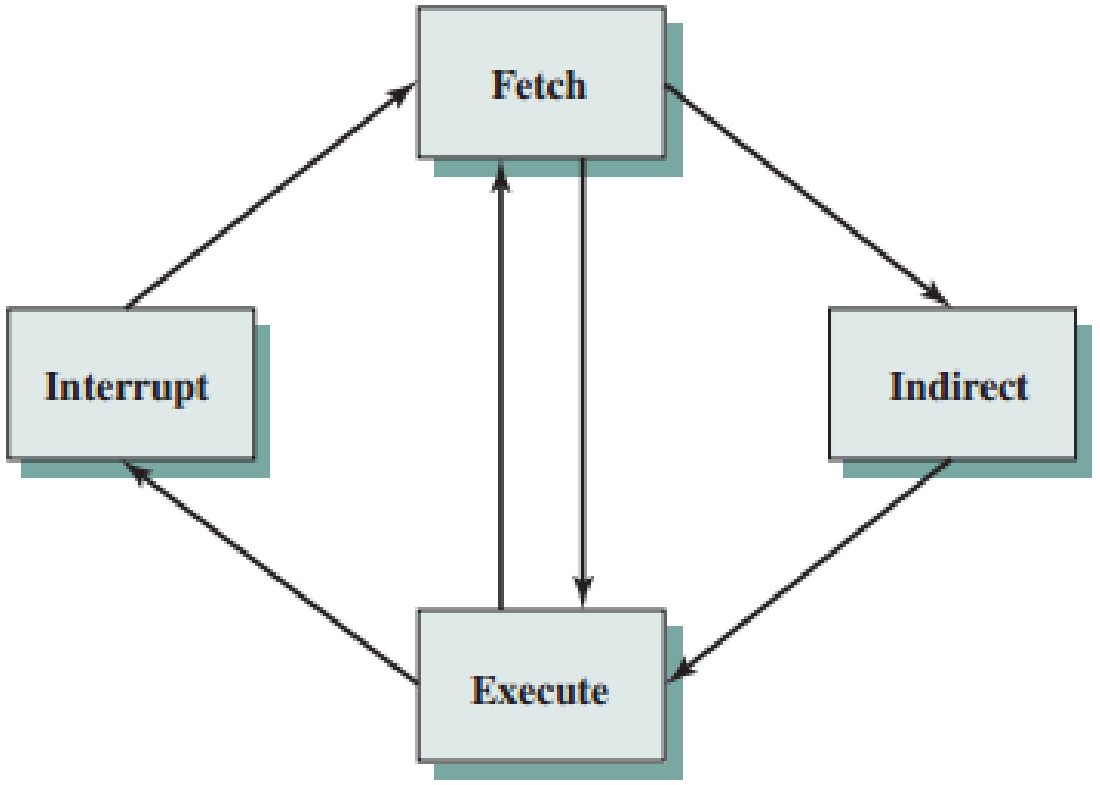
Instruction Cycle


Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt
Interrupt check
Interrupt
Interrupt

Instruction Cycle…

- It includes following stages:
    - Fetch
    - Execute
    - Interrupt
- **Another stage** must be introduced, **the indirect cycle** – to access memory to fetch operands using indirect addressing
- **Data Flow**
    - **Exact sequence of events** during an instruction cycle
    - Depends on **processor design** employing MAR, MBR, PC, IR

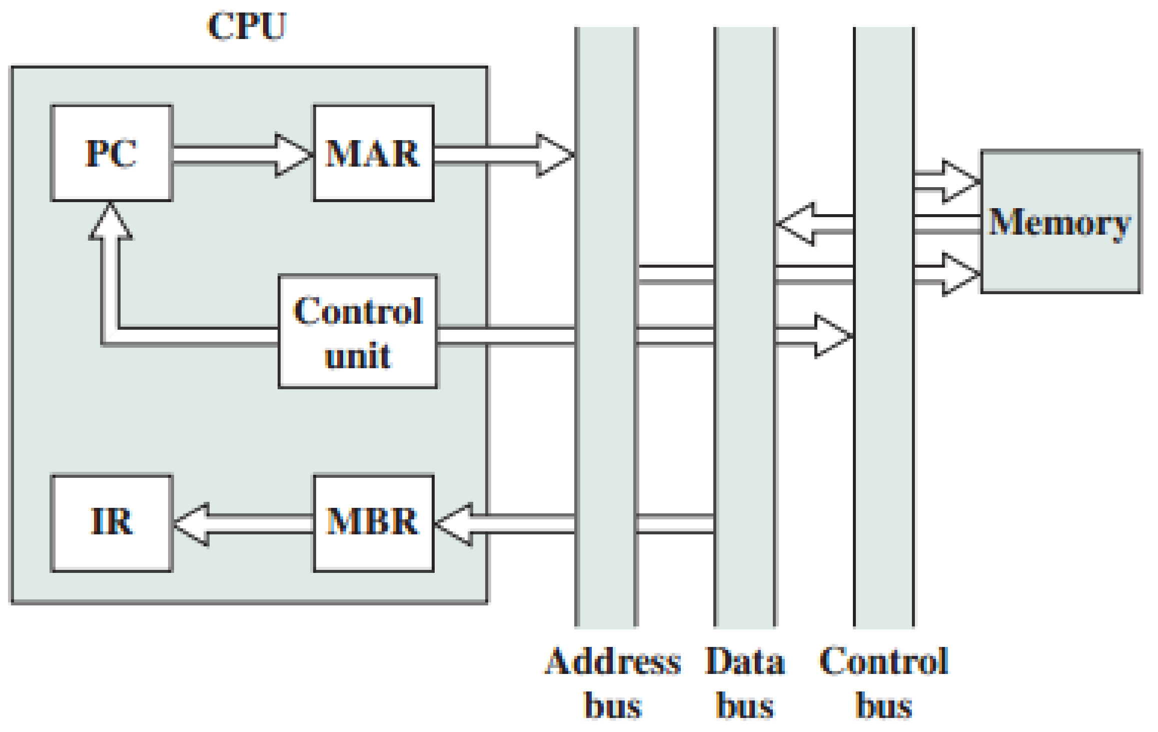
Instruction Cycle: Revised View



Fetch Cycle

- Fetch Cycle involves
  - An instruction read from memory
  - The **PC** contains the address of the next instruction to be fetched.
  - This address is moved to the **MAR** and placed on the **address bus**.
  - The control unit requests a **memory read**, and the result is placed on the **data bus**
  - That data from data bus copied into the **MBR** and then moved to the **IR**.
  - Meanwhile, the <u>PC is incremented by 1</u>, preparatory for the next fetch.
- Once the fetch cycle is completed, the control unit examines the IR to determine <u>if there any operand is required or not</u> and is that <u>operand requires any indirect cycle or not</u>

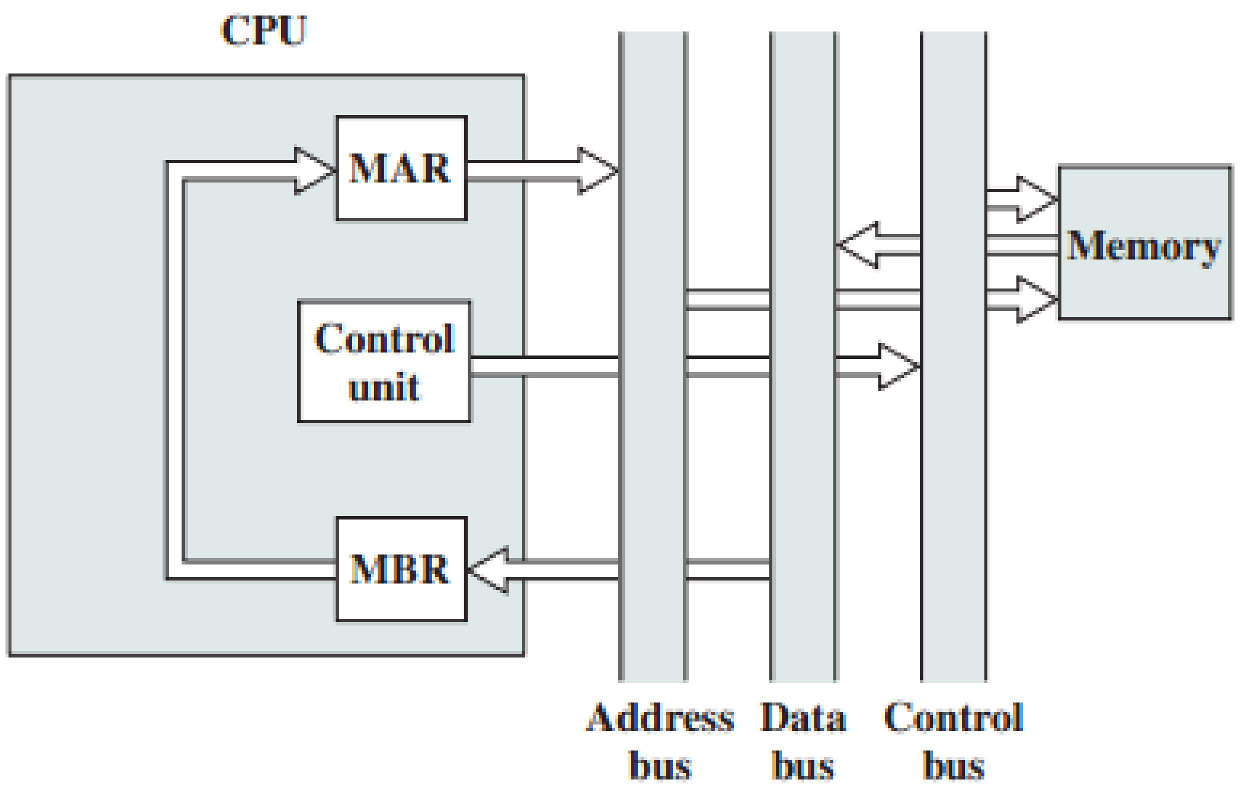
Fetch Cycle…


CPU
PC
MAR
Control unit
IR
MBR
Memory
Address bus
Data bus
Control bus

Indirect Cycle

- To access memory fetching one or more operands in memory **using indirect addressing mode** as it requires more memory access
- Can be thought as one more instruction stage
- It involves:
    - The rightmost N bits of the **MBR** (fetched instruction from memory), which contain the address reference, are transferred to the **MAR**.
    - Then the control unit requests **a memory read**, to get the desired address of the operand into the **MBR**.

Indirect Cycle…



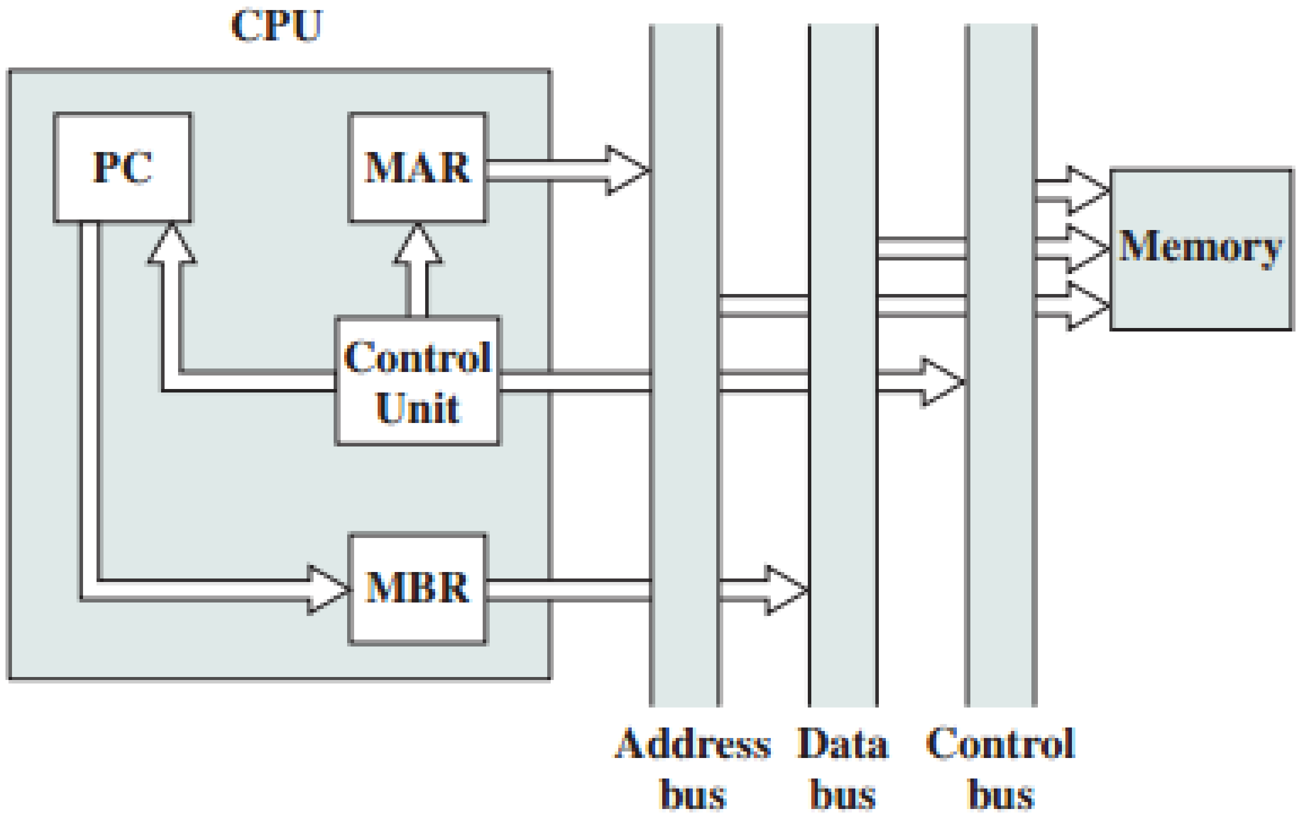
Execute Cycle

- It takes **many forms based on different machine instructions** in IR
- May involve:
  - **Transfer data** among registers
  - **Read or write** from memory or I/O
  - Invocation of the **ALU**

Interrupt Cycle

- It involves:
    - To resume the current program after ISR, the contents of the **PC** are transferred to the **MBR** to be written into memory. **(in stack)**
    - The special **memory location (stack pointer)** reserved for this purpose is loaded into the **MAR** from the control unit.
    - The **PC** is loaded with the address of the interrupt routine. As a result, the next instruction cycle will begin by fetching the appropriate instruction. (with the help of control unit)

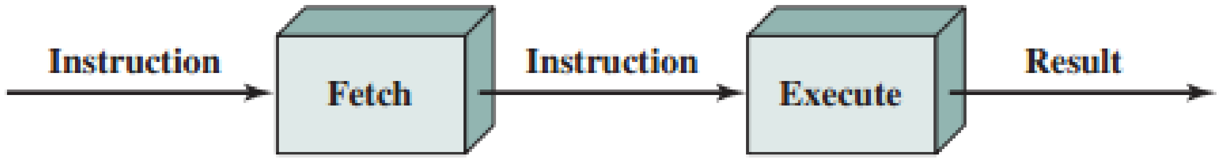
Interrupt Cycle…



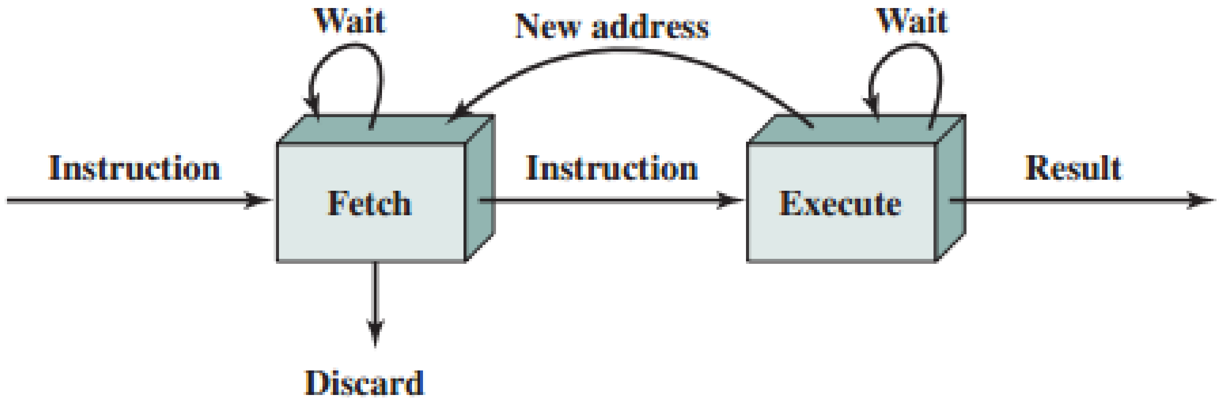
Pipeline Strategy

- Similar to the use of **an assembly line** in a manufacturing plant – products at various stages can be worked on simultaneously
- To apply this concept **in instruction execution**, we have to recognize a **number of stages** in an instruction cycle
- In **two-stage pipeline**, if the fetch and execute stages are of equal duration, the instruction **cycle time** will be halved, **execution rate** will be doubled

Two-Stage Instruction Pipeline



(a) Simplified view



(b) Expanded view

But Things Are Not Always Ideal!!!

- For Two-stage pipeline:
  - The execution time will take longer than fetch time as execution involves reading and storing operands, operation performance, storing back the results
  - A conditional branch instruction makes the address of the instruction unknown
- A simple rule to reduce time loss for second reason is to fetch next instruction of the branch instruction as there is 50% possibility that branch will not be taken. If happened otherwise, the next fetched instruction will be discarded.

More Stage Pipeline

- To increase the speedup (increase the execution rate), pipeline must have more stages like following:
    1. **Fetch Instruction** (FI)
    2. **Decode Instruction** (DI)
    3. **Calculate Operands** (CO)
    4. **Fetch Operands** (FO)
    5. **Execute Instruction** (EI)
    6. **Write Operand** (WO)
- As the number of stages are increased, they will be probably of **equal duration**

6-stage Instruction Pipeline

Time →

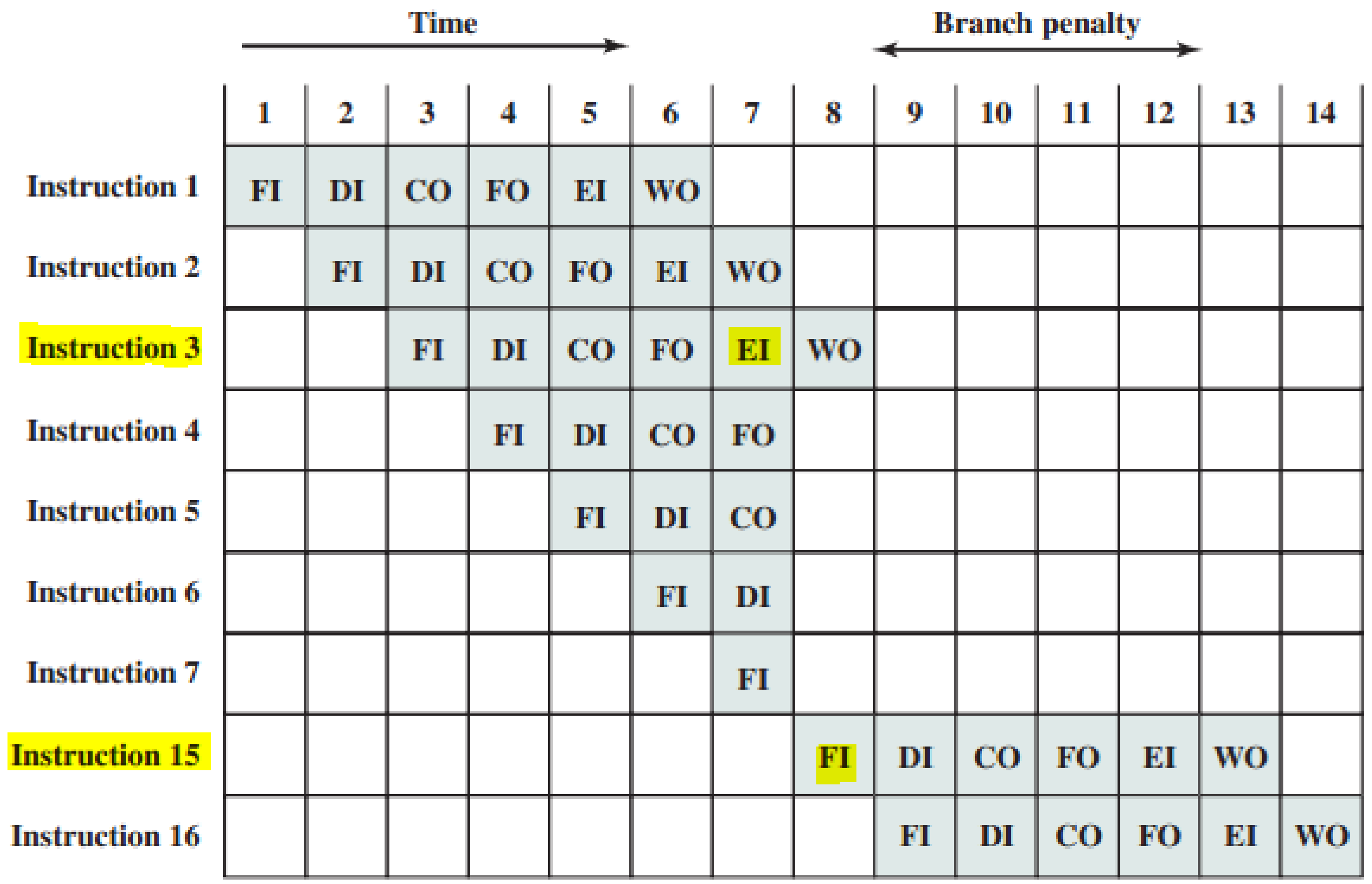
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Time for 9 instructions?

Limitations Again!!!

- Some instructions will not cover all stages (e.g. Load not have WO stage)
- It was assumed that all stages can be performed in parallel, but sometimes conflict are there among the instructions to perform simultaneously (FI, FO, WO may involve memory access at the same time)
- Conditional branch instruction invalidation
- Interrupt invalidation
- Some operand dependency are there to fetch next operands

Effect of Conditional Branch in Pipeline

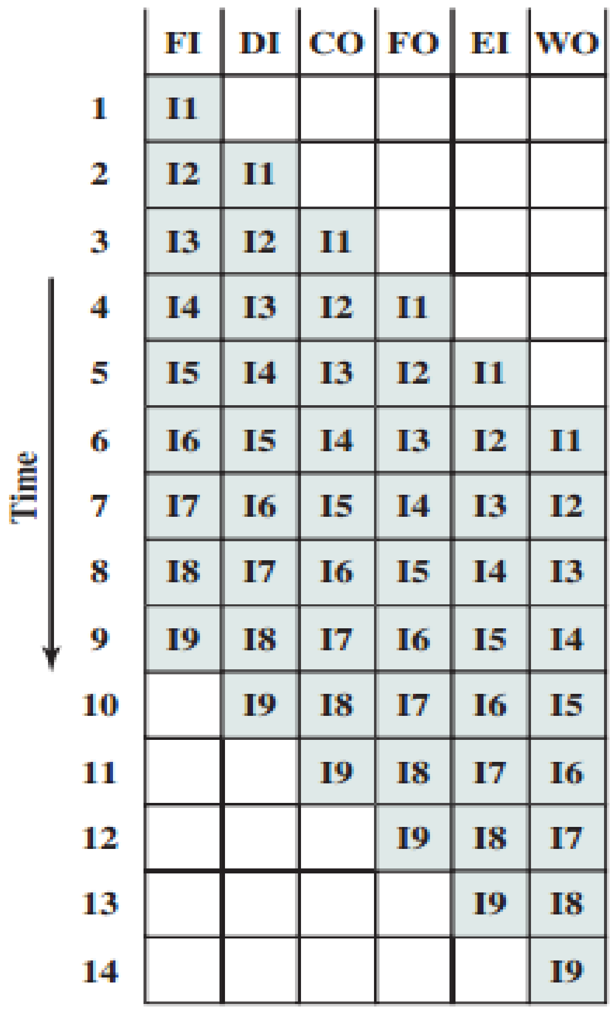


Time

Branch penalty

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

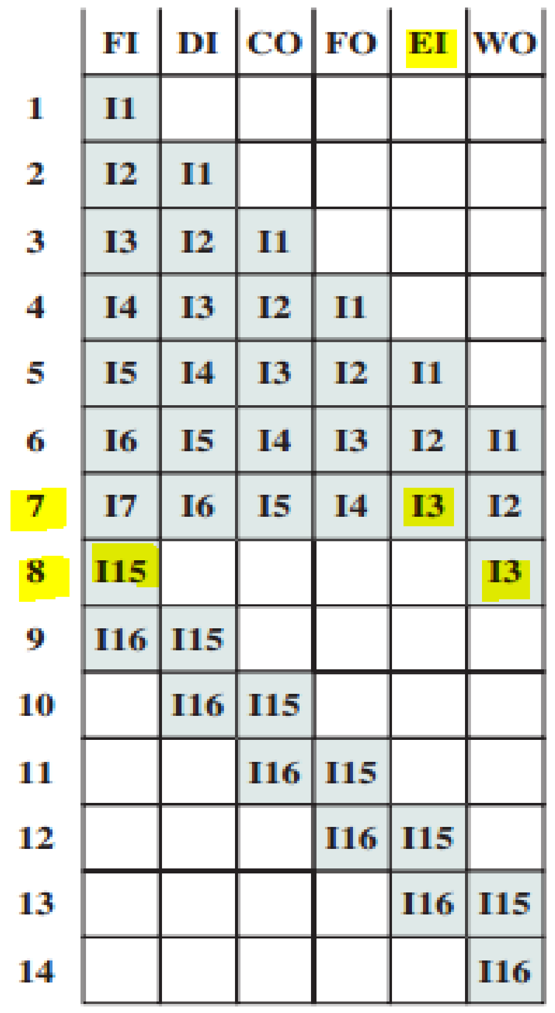
Alternative Pipeline Depiction



Time

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						I9

(a) No branches



	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

(b) With conditional branch

Pipeline Performance

- The **cycle time *τ*** of an instruction pipeline needed to advance **one stage**:

$$\tau = \max_i[\tau_i] + d = \tau_m + d \quad 1 \le i \le k$$

τ_i = time delay of the circuitry in the *i*th stage of the pipeline
τ_m = maximum stage delay (delay through stage which experiences the largest delay)
k = number of stages in the instruction pipeline
d = time delay of a latch, needed to advance signals and data from one stage to the next

Pipeline Performance…

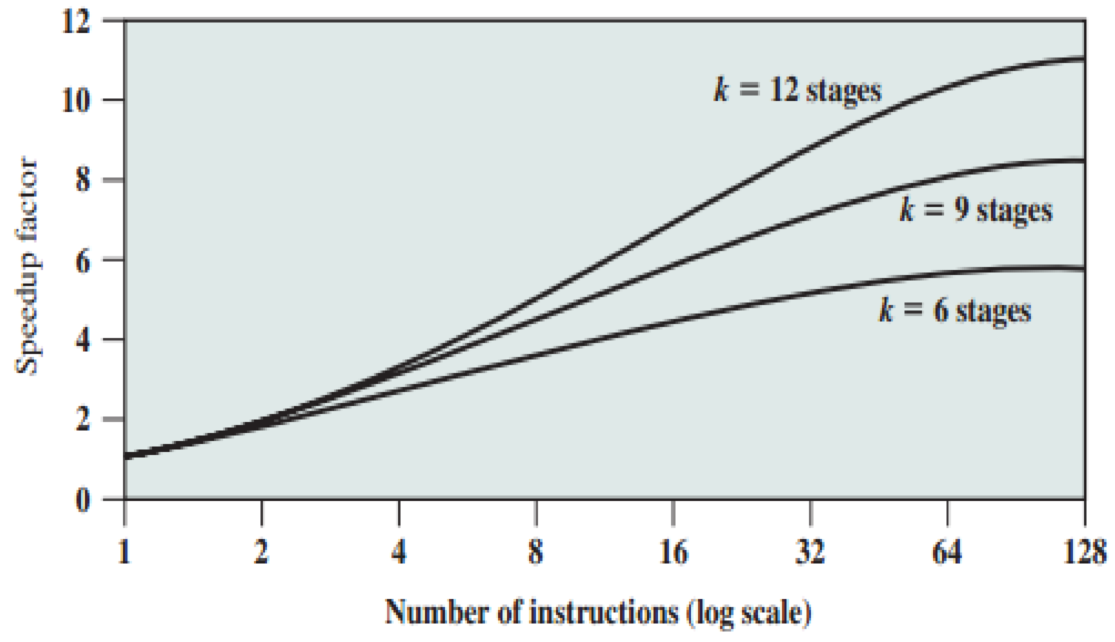
- ***$T_{k,n}$*** is the total time for a pipeline with k stages to execute n instructions :

$$T_{k,n} = [k + (n - 1)]\tau$$

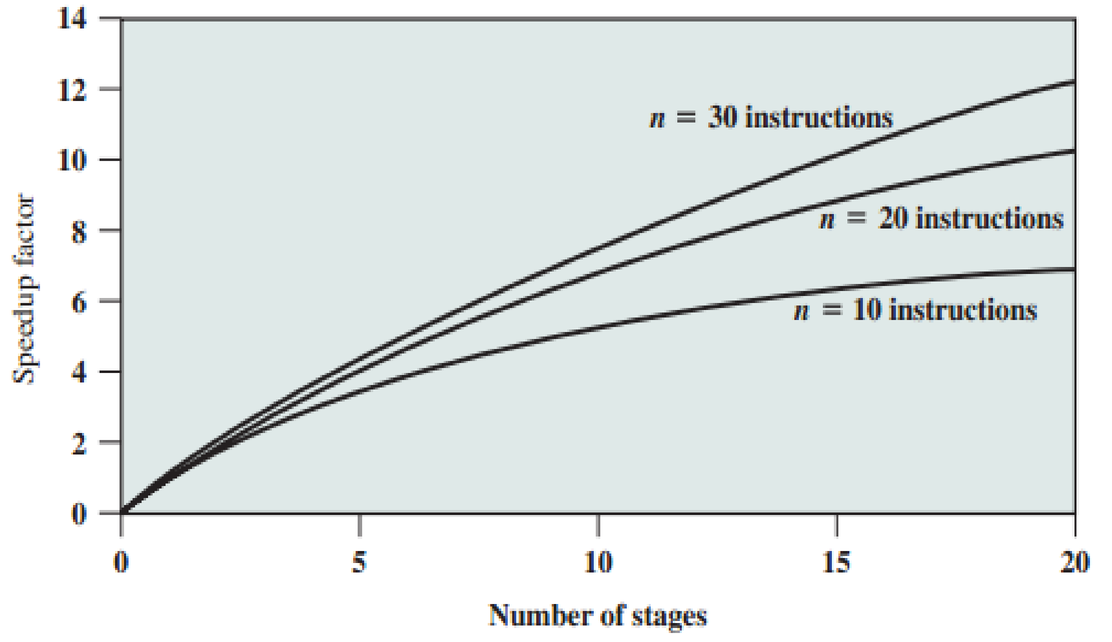
- Example: From the previous figure it is 14.
- **Speedup** with and without pipeline:

$$S_k = \frac{T_{1,n}}{T_{k,n}} = \frac{nk\tau}{[k + (n - 1)]\tau} = \frac{nk}{k + (n - 1)}$$

Pipeline Speedup



(a)



(b)

Pipeline Hazards

- **Pipeline must stall** as conditions don't permit to execute more
- **Three** types of Hazards:
  - **Resource Hazards**: need same resource for more than one instruction
  - **Data Hazards**: to access conflicting data
    - **Read after write** (true dependency)
    - **Write after read** (anti dependency)
    - **Write after write** (output dependency)
  - **Control Hazards**: branch to another location

Dealing with Branches

- **Multiple Stream** (replicate initial portion and allow the both branches)
- **Pre-fetch Branch Target** (only the target of the branch is pre-fetched)
- **Loop Buffer** (cache for instructions in branch)
- **Brach Prediction** (various techniques to decide)
- **Delayed Branch** (rearranging the instructions within a program)

Intel 80486 Pipelining

- Self Study