Computer Architecture

Course Code: CSE360

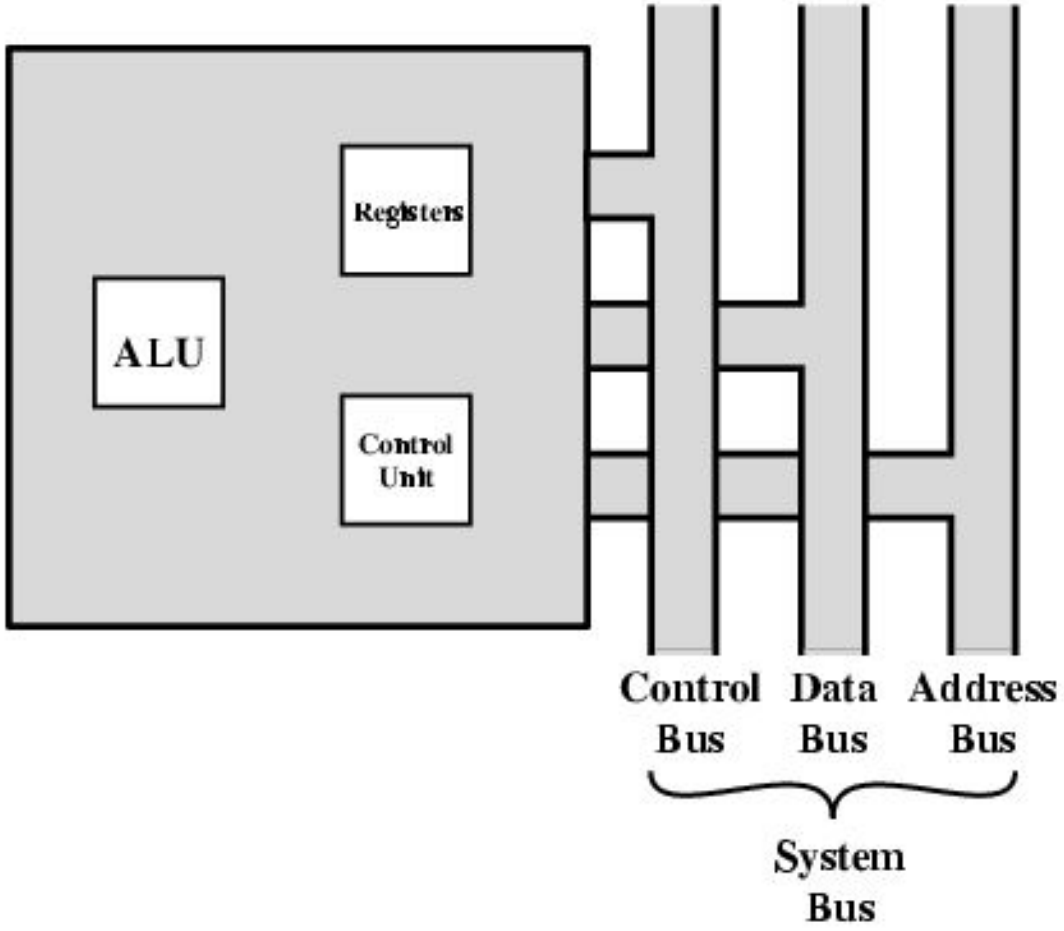
Lecture 11

CPU Structure and Function

CPU Structure

- CPU must:
  - Fetch instructions: The processor reads an instruction from memory (register, cache, main memory).
  - Interpret instructions: The instruction is decoded to determine what action is required.
  - Fetch data: The execution of an instruction may require reading data from memory or I/O module
  - Process data: The execution of an instruction may require performing some arithmetic or logical operation on data
  - Write data: The results of an execution may require writing data to memory or I/O module.

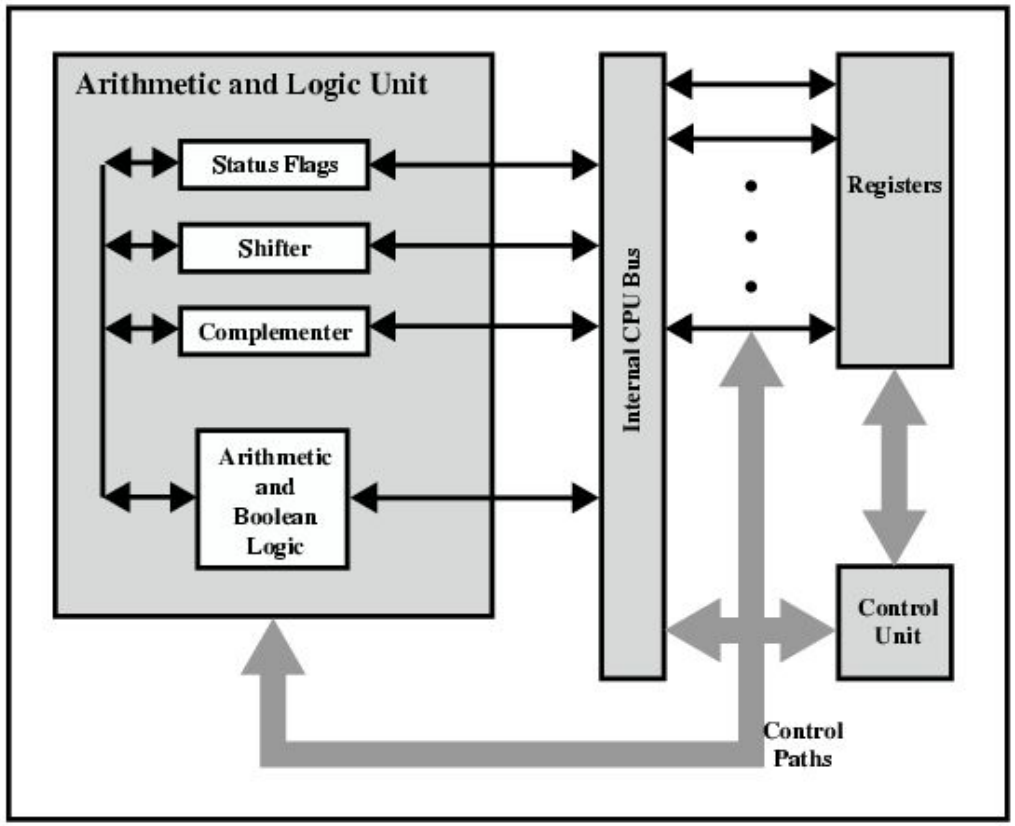
CPU With Systems Bus



- ALU performs actual computation or processing of data.
- The control unit controls the movement of data and instructions into and out of the processors, and controls the operation of ALU
- Registers, minimal internal memory consisting of a set of storage locations

  - processors need to store some data temporarily while execution of instructions as it needs to remember the location of last instruction to get the next instruction

CPU Internal Structure



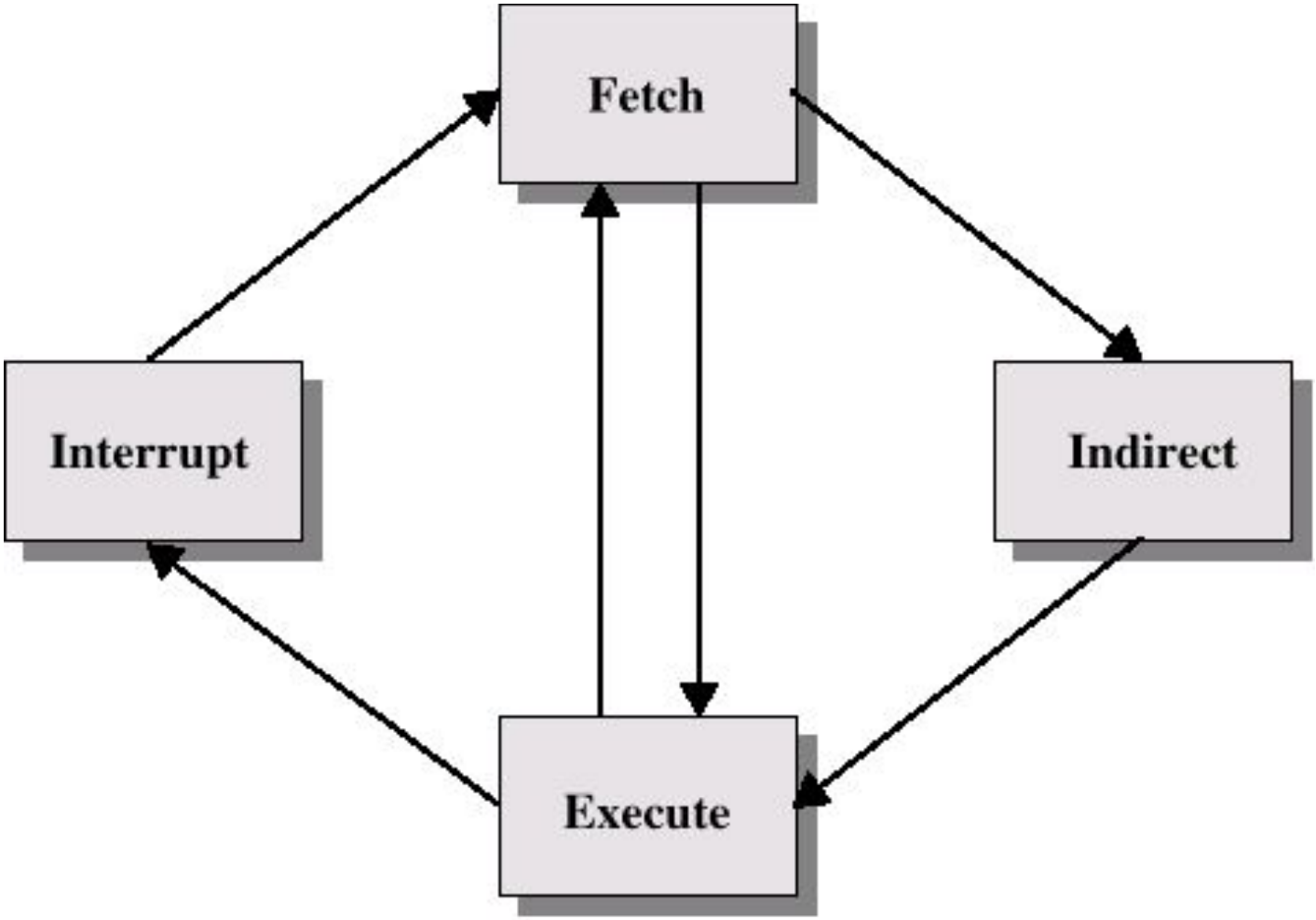
- Data transfer and logic control paths are indicated (bold arrow), including an element labeled *internal processor bus*.

- *Internal processor bus* is needed to transfer data between the various registers and the ALU. This is because, ALU operates only on data in the internal processor memory (e.g., registers)

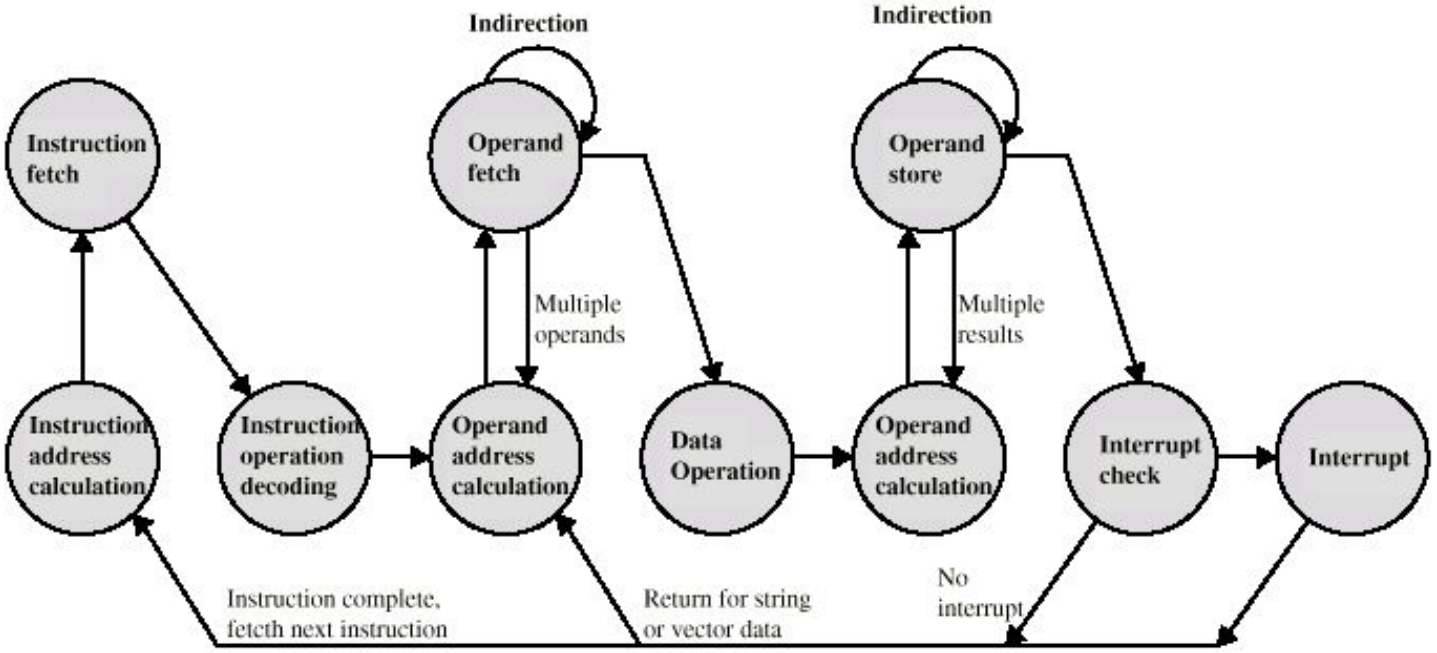
Control & Status Registers

- Program Counter (PC): Contains the address of an instruction to be fetched
- Instruction Register (IR): Contains the instruction most recently fetched
- Memory Address Register (MAR): Contains the address of a location in memory
- Memory Buffer Register (MBR): Contains a word of data to be written to memory or the word most recently read.

Instruction Cycle with Indirect



Instruction Cycle State Diagram



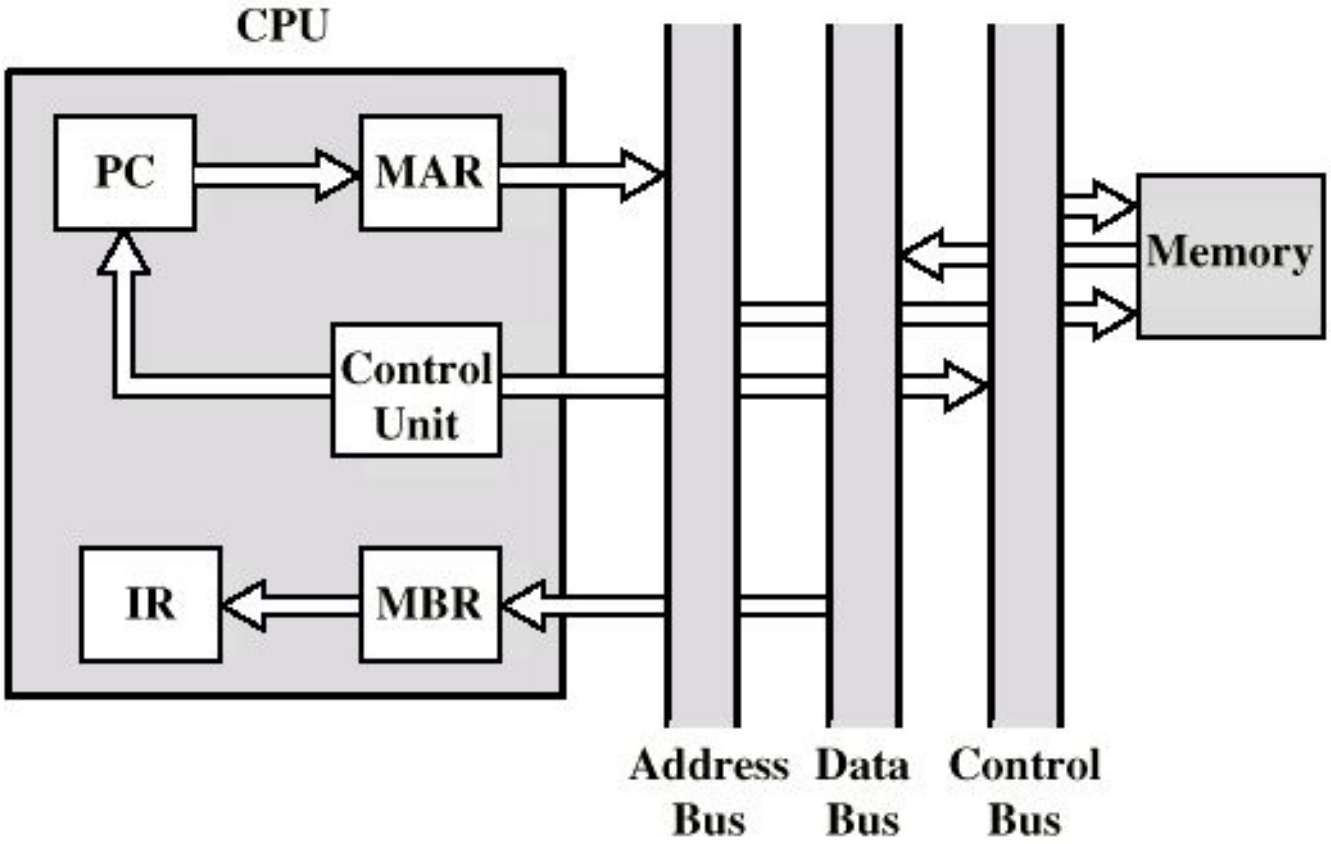
- Once an **instruction is fetched**, its **operand** specifiers **must be identified**.
- Each input **operand in memory is then fetched** and this process **may require indirect addressing**
- **Registers based operands need not be fetched**.
- **Once the operand is executed**, similar process may be needed to store the result in main memory

Data Flow (Instruction Fetch)

- Depends on CPU design
- In general:

- Fetch
  - PC contains address of next instruction
  - Address moved to MAR
  - Address placed on address bus
  - Control unit requests memory read
  - Result placed on data bus, copied to MBR, then to IR
  - Meanwhile PC incremented by 1

Data Flow (Fetch Diagram)

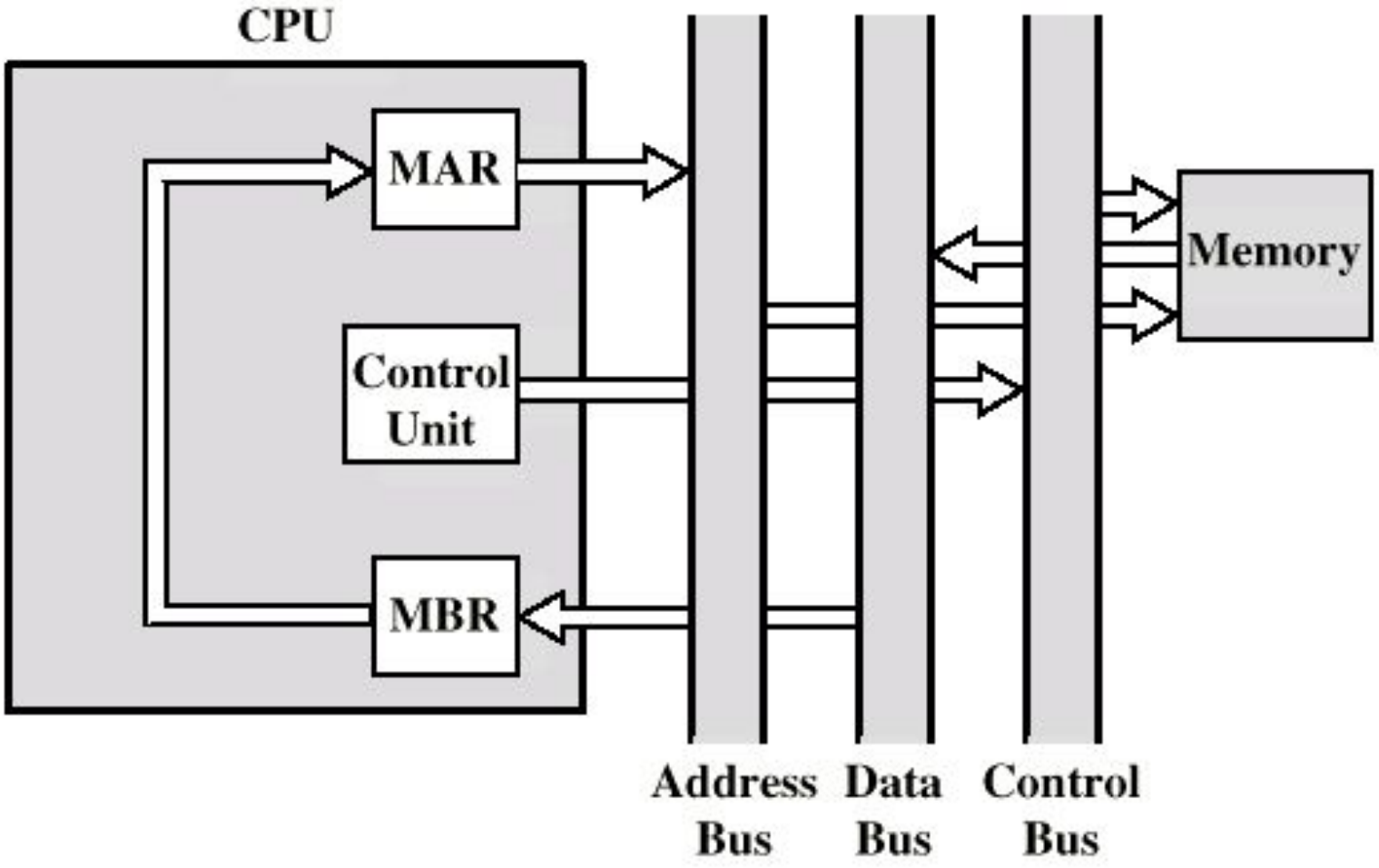


MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Data Flow (Data Fetch)

- IR is examined
- If indirect addressing, indirect cycle is performed
  - Right most N bits of MBR which contain the operand address, are transferred to MAR
  - Control unit requests memory read
  - Result (operand fetch) moved to MBR

Data Flow (Indirect Diagram)



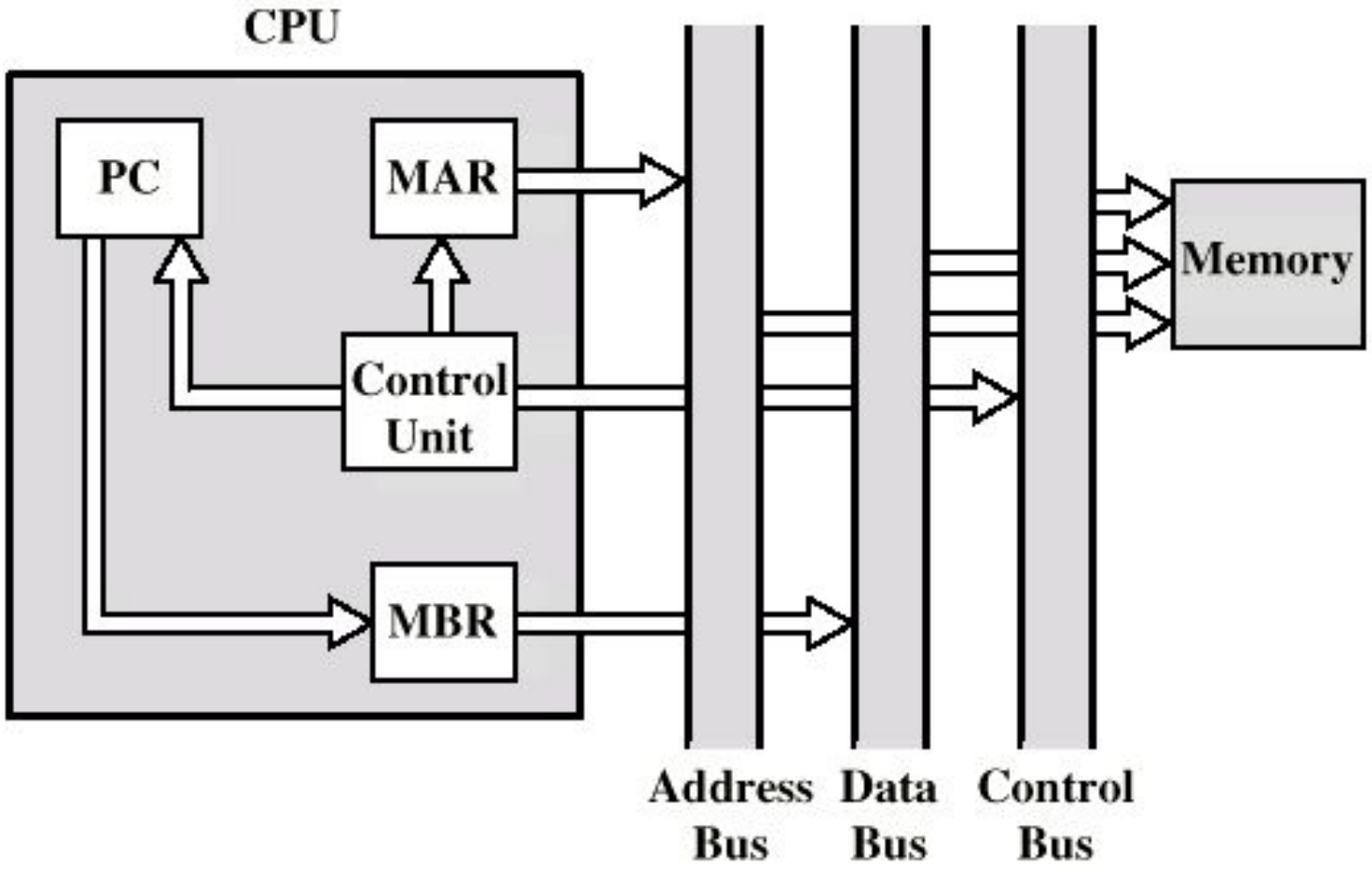
Data Flow (Execute)

- May take many forms
- Depends on instruction being executed
- May include
  - Read/write from memory or I/O
  - Transferring data among register
  - ALU operations

Data Flow (Interrupt)

- Simple
- Predictable
- Current PC saved to allow processor to resume activity after interrupt
- Contents of PC copied to MBR to be written into memory through data bus
- Special memory location reserved for this purpose is loaded into MAR from the control unit
- PC loaded with address of interrupt handling routine
- Next instruction (first of interrupt handler) can be fetched

Data Flow (Interrupt 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

- Processors make use of **instruction pipelining to speed up execution**.
- Pipeline involves **breaking up the instruction** cycle **into number of separate stages** that occur in sequence, such as **fetch instruction, decode instruction, calculate operands, fetch operands, execute instruction, and write result (6)**
- Overlap these operations
- Instructions move through these stages, as on an assembly line, so that , **each stage can work with different instructions at the same time**

Two-Stage Instruction Pipeline

- The pipeline has two independent stages
- The first stage fetches an instruction and buffers it
- When the second stage is free, the first stage passes it the buffered instruction
- While the second stage is executing the instruction, t**he first stage takes advantage of unused memory cycles to fetch and buffer the next instruction**
- **This is called instruction fetch/fetch overlap**

Two-Stage Instruction Pipeline-contd..

- It should be clear that this process will speed up instruction execution.
- If the fetch and execute stages were of equal duration, the instruction cycle time would be halved, but unlikely for two reasons:

(1) The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. **Thus, the fetch stage may have to wait for some time before it can empty its buffer.**

(2) A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until it receives the next instruction address from the execute stage. The execute stage may then have to wait while the next instruction is fetched.

Two Stage Instruction Pipeline

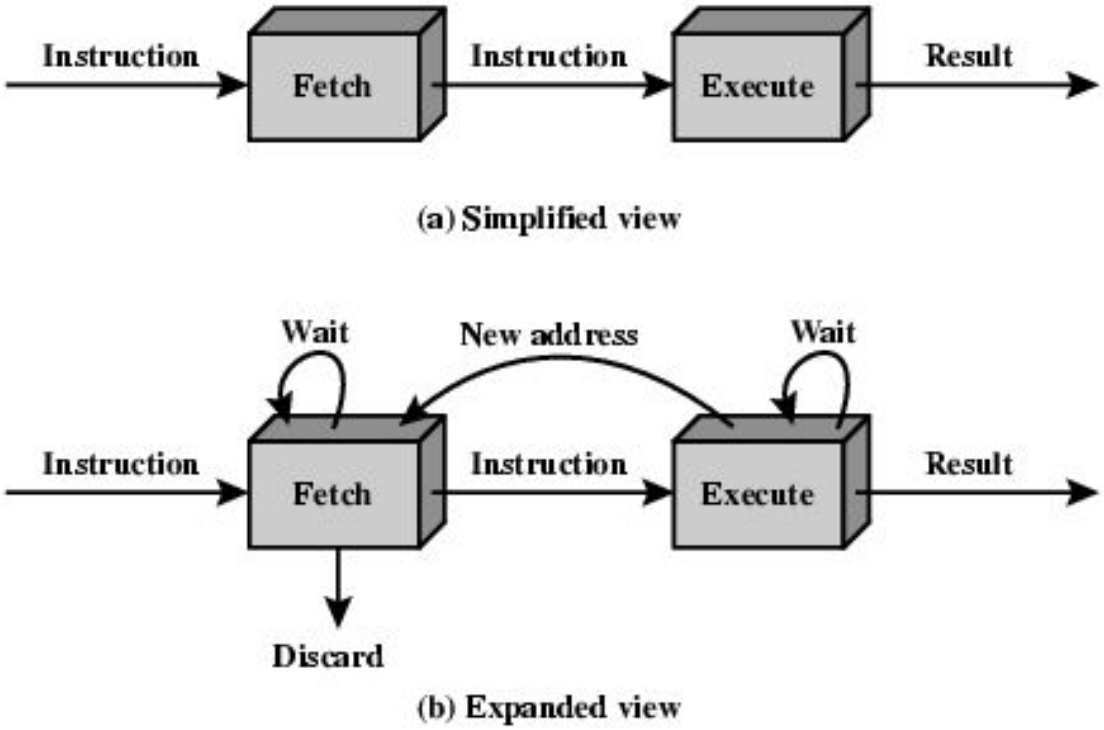


Figure: Two-stage Instruction Pipeline

- **Guessing can reduce the time loss from the second reason**
- When a conditional branch instruction is passed from fetch to the execute stage, the fetch stage fetches the next instruction in memory after the branch instruction
- Then, **if the branch is not taken, no time is lost**
- **If the branch is taken, the fetched instruction must be discarded and a new instruction fetched**

Decomposition of the Instruction Processing

While the factors reduce the potential effectiveness of the two-stage pipeline, some speed up occurs

To gain further speedup, the pipeline must have some more stages:

- Fetch instruction (FI)
- Decode instruction (DI)
- Calculate operands (CO)
- Fetch operands (FO)
- Execute instructions (EI)
- Write operand (WO)

Timing Diagram for Instruction Pipeline Operation

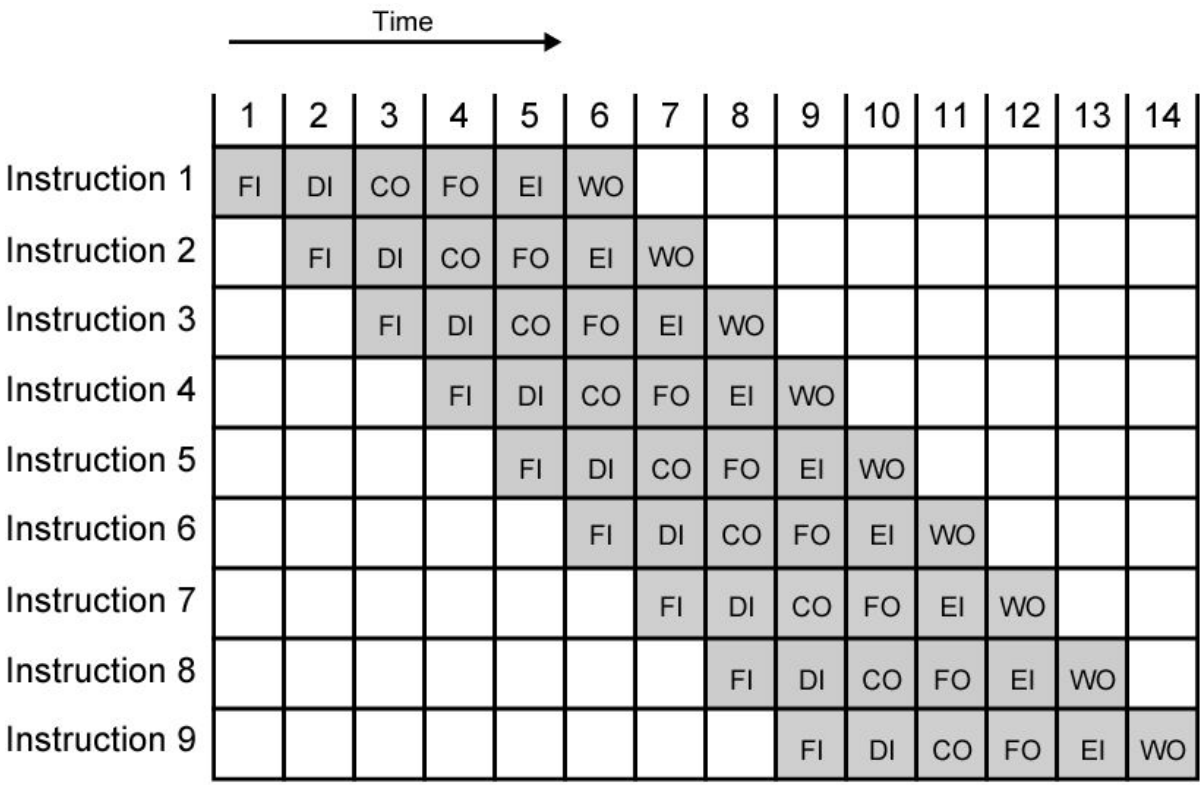


Figure: Timing diagram for Instruction Pipeline Operation

- A six-stage pipeline can reduce the execution for 9 instructions from 54-time units to 14-time units
- Time is set by assuming that each instruction requires all six stages, and all of the stages can be performed in parallel
- There are no memory conflicts (e.g. FI, FO, and WO involve memory access and occur simultaneously)
- Most memory system will not permit that. The desired value may be in cache, or the FO or WO stage may be null. So, memory conflicts will not slow down the pipeline

Factors to Limit the performance

- If six stages **are not of equal duration**, there will be **some waiting involved** at various pipeline stages
- Another difficulty is the **conditional branch instruction which can invalidate several instruction fetches**

The Effect of a Conditional Branch on Instruction Pipeline Operation

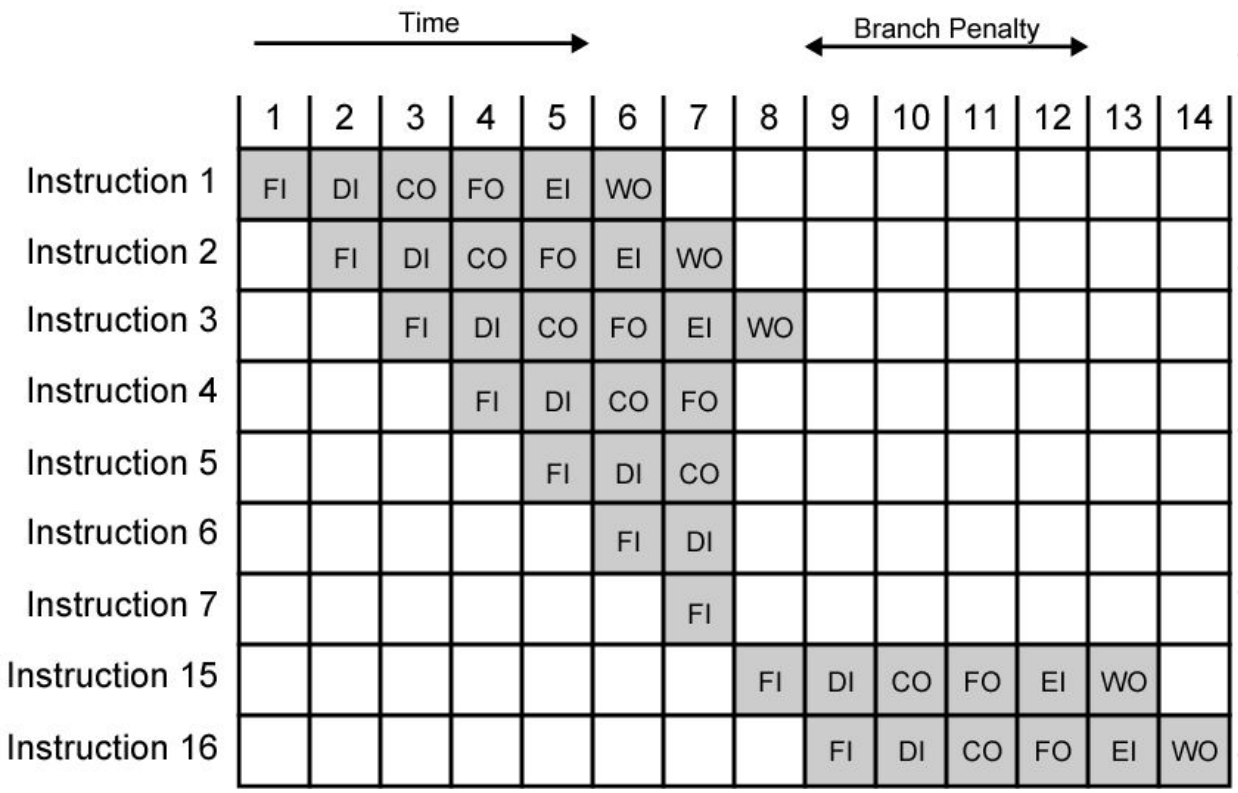


Figure: The effect of conditional branch on instruction pipeline operation

- Instruction 3 is a conditional branch to instruction 15.
- Until the instruction is executed, there is no way of knowing which instruction will come next (at which point)
- The pipeline simple loads the next instruction in sequence (instruction 4) and proceeds
- If branch is not taken, we will get full performance benefit of the enhancement (in pre. Dig)
- Here branch is taken and is not determined until the end of time unit 7
- At this point, the pipeline must be cleared of instructions that are not useful
- During time unit 8, instruction 15 enters pipeline
- No instructions complete during time units 9 through 13

Six Stage Instruction Pipeline

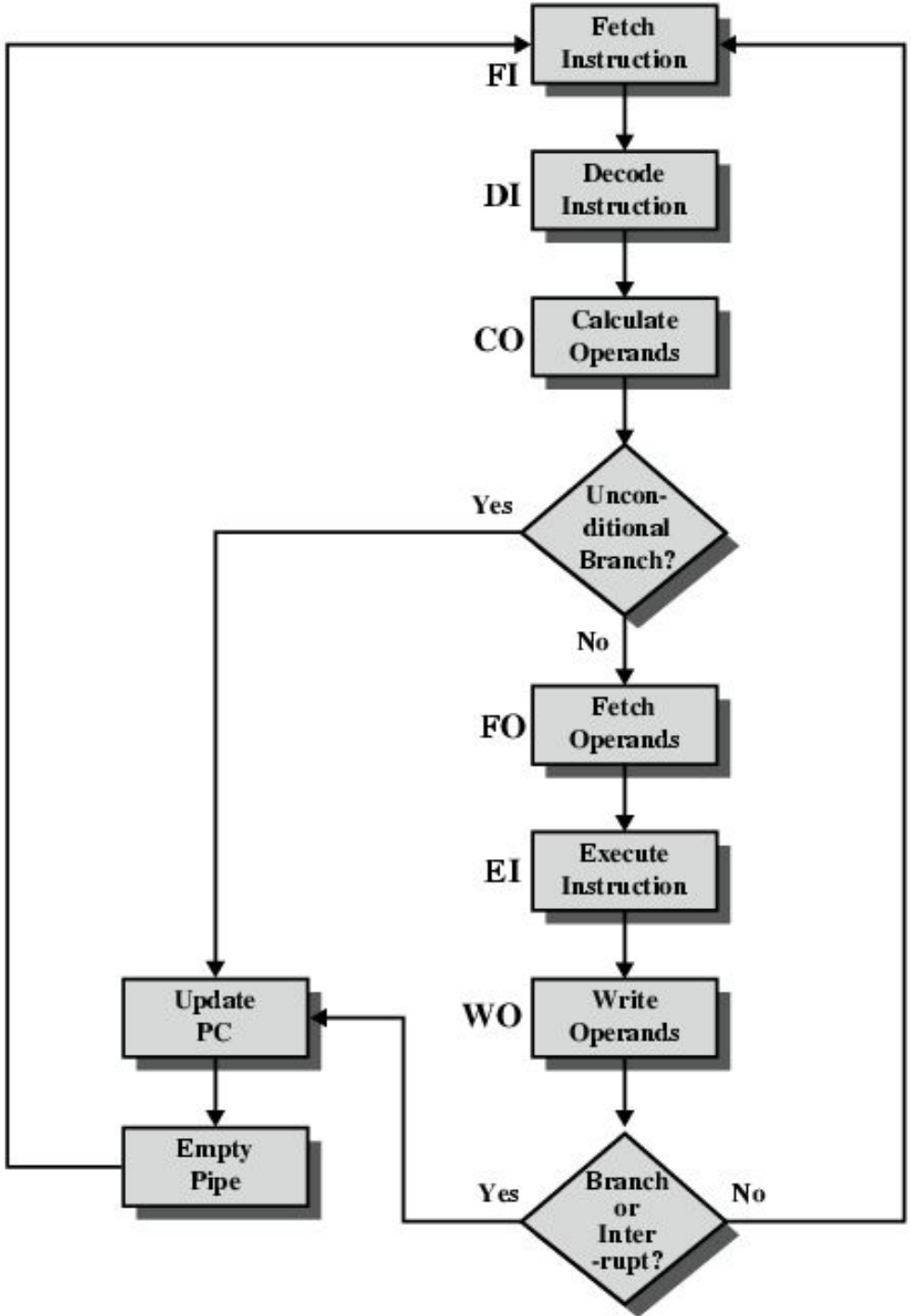
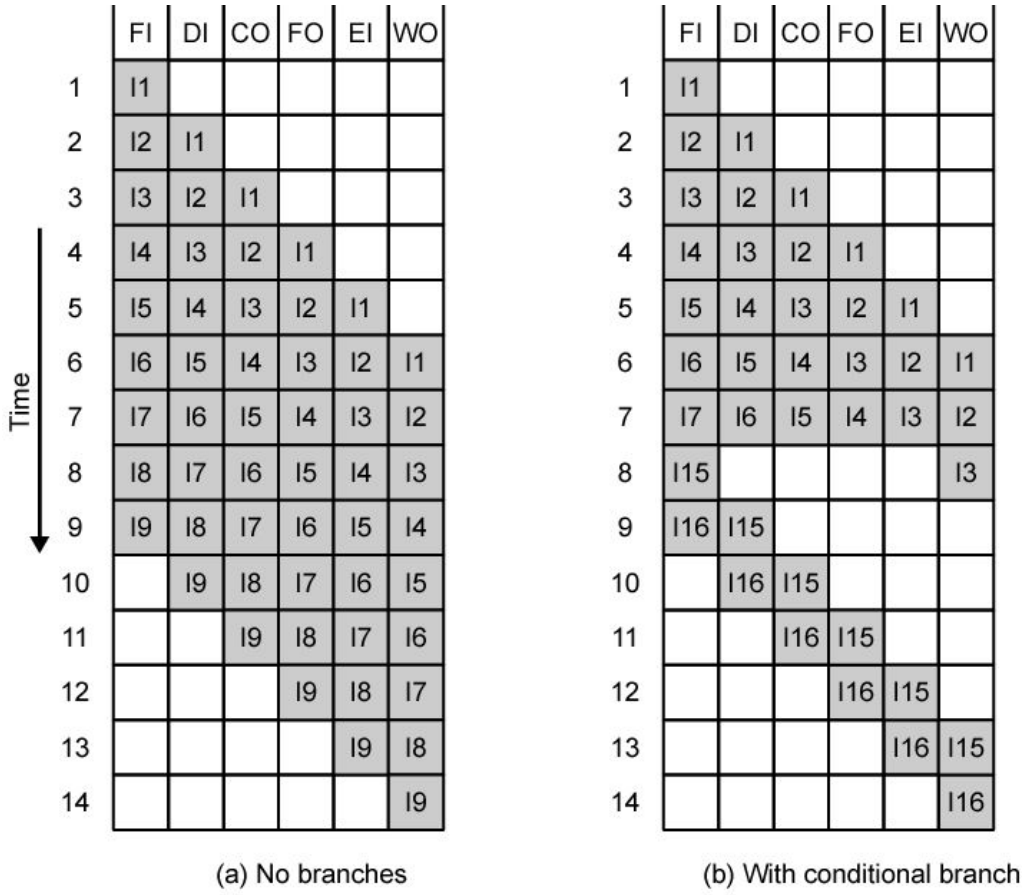


Figure: Six-stage CPU Instruction Pipeline

Alternative Pipeline Depiction



(a)

- The pipeline is full at time 6, with 6 different instructions in various stages of execution and remains full through time 9; we assume that I9 is the last instruction

(b)

- The pipeline is full at times 6 and 7.
- At time 7, instruction 3 is in execution stage and executes a branch to instruction 15
- At this point, instructions I4 through I7 are flushed from the pipeline, so that at time 8, only two instructions are in the pipeline, I3 and I15

Speedup Factors with Instruction Pipelining