

EE5201- COMPUTER ARCHITECTURE

Processor Structure and Function

Miss. Awanthi Jayasundara

BScEng (Hons) (Ruhuna)

avanthi@eie.ruh.ac.lk

Processor Organization

- **Things done by a processor**
 - **Fetch instruction**
 - The processor reads an instruction from memory (register, cache, main memory)
 - **Interpret instruction**
 - The instruction is decoded to determine what action is required.
 - **Fetch data**
 - The execution of an instruction may require reading data from memory or an I/O module
 - **Process data**
 - The execution of an instruction may require performing some arithmetic or logical operation on data

Processor Organization

- **Things done by a processor**
 - **Write data**
 - The results of an execution may require writing data to memory or an I/O module
- Processor needs to store some data temporarily
- It must remember the location of the last instruction so that it can know where to get the next instruction
- It needs to store instructions and data temporarily while an instruction is being executed
- The processor needs a small internal memory

Processor Organization

- Major components of the processor
 - Arithmetic and Logic Unit (ALU)
 - Control Unit (CU)
 - Set of storage locations, called registers

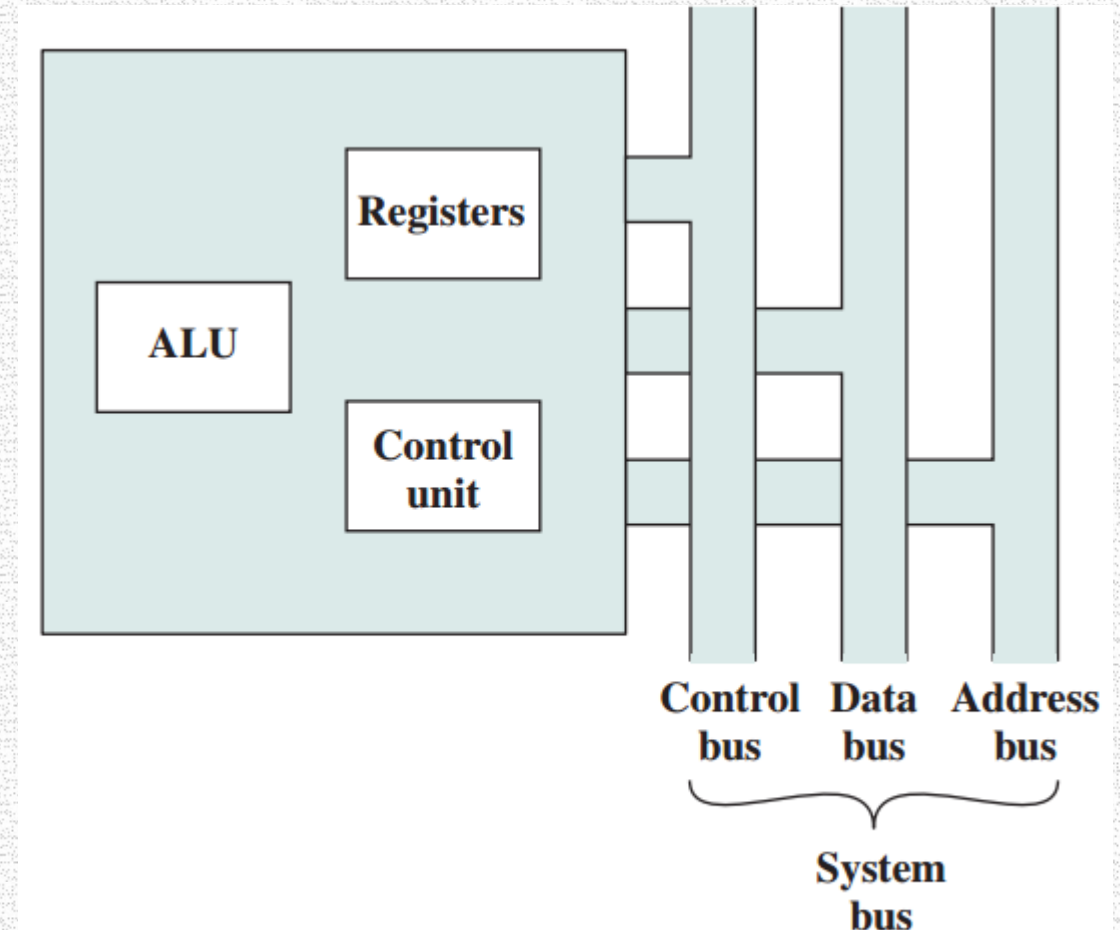
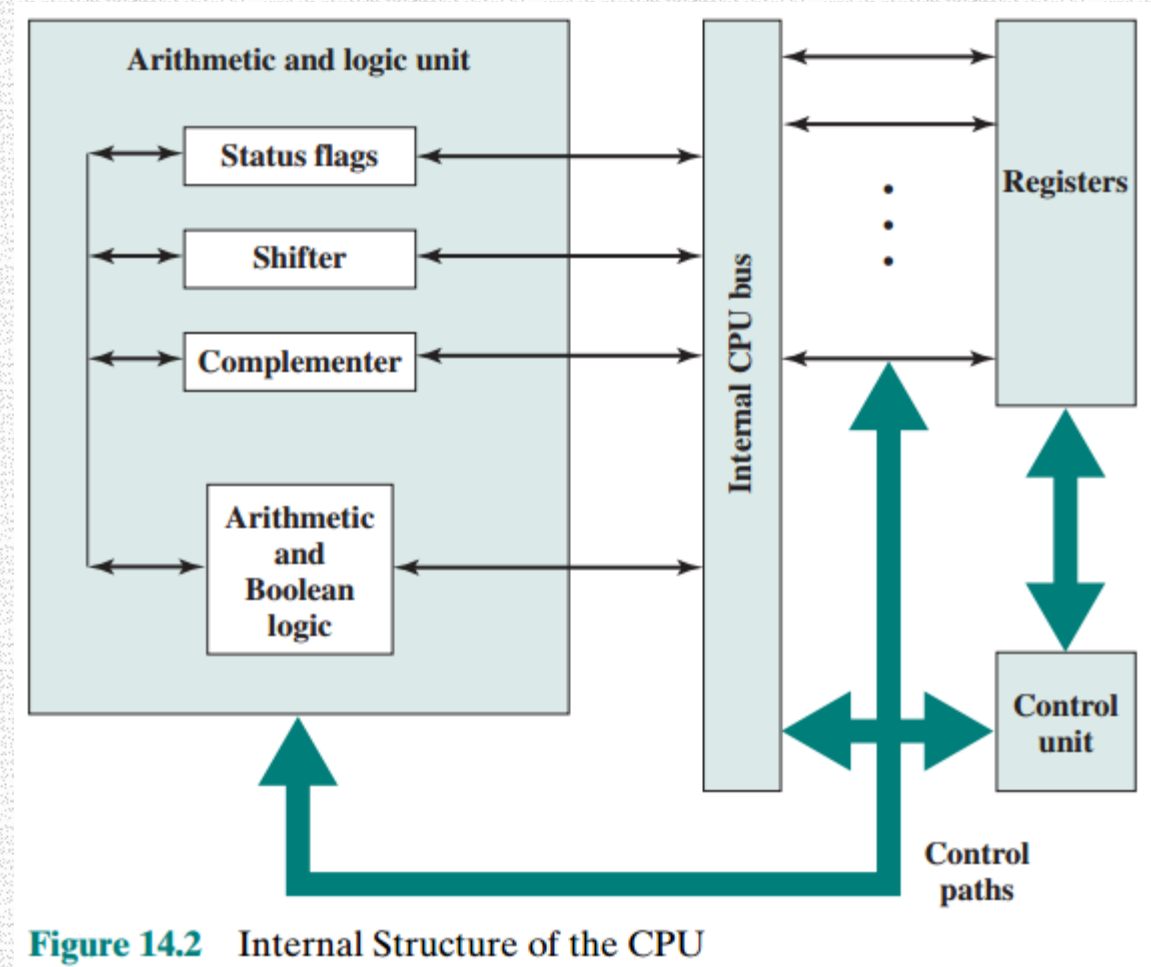


Figure 14.1 The CPU with the System Bus

Processor Organization

- Internal processor bus
 - This element is needed to transfer data between the various registers and the ALU
 - ALU in fact operates only on data in the internal processor memory



Register Organization

- User-Visible Registers
 - Enable the machine or assembly language programmer to minimize main memory references by optimizing use of registers
- Control and status registers
 - Used by the control unit to control the operation of the processor

Register Organization

- User-Visible Registers
 - Referenced by means of the machine language that the processor executes
 - General-purpose registers
 - Data registers
 - Address registers
 - Condition codes

User-Visible Registers

- General-purpose registers
 - Can be assigned to a variety of functions by the programmer
 - There may be dedicated registers for floating-point and stack operations
- Data registers
 - May be used only to hold data and cannot be employed in the calculation of an operand address

User-Visible Registers

- Address registers
 - Segment pointers
 - Segment register holds the address of the base of the segment
 - Index registers
 - These are used for indexed addressing and may be auto indexed
- Stack pointer
 - If there is user-visible stack addressing, then typically there is a dedicated register that points to the top of the stack

User-Visible Registers

- Condition codes (flags)
 - Bits set by the processor hardware as the result of operations
 - Example: An arithmetic operation may produce a positive, negative, zero, or overflow
 - In addition to the result itself being stored in a register or memory, a condition code is also set

Control and Status Registers

- Registers involved in control operation of the processor
- Not visible to the user
- Four registers essential to instruction execution
 - **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

Control and Status Registers

- Program Status Word (PSW)
 - Condition codes plus other status information
 - **Sign:** Contains the sign bit of the result of the last arithmetic operation.
 - **Zero:** Set when the result is 0
 - **Carry:** Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high-order bit. Used for multiword arithmetic operations
 - **Equal:** Set if a logical compare result is equality
 - **Overflow:** Used to indicate arithmetic overflow

Control and Status Registers

- Program Status Word (PSW)
 - **Interrupt Enable/Disable:** Used to enable or disable interrupts.
 - **Supervisor:** Indicates whether the processor is executing in supervisor or user mode

Register Organization

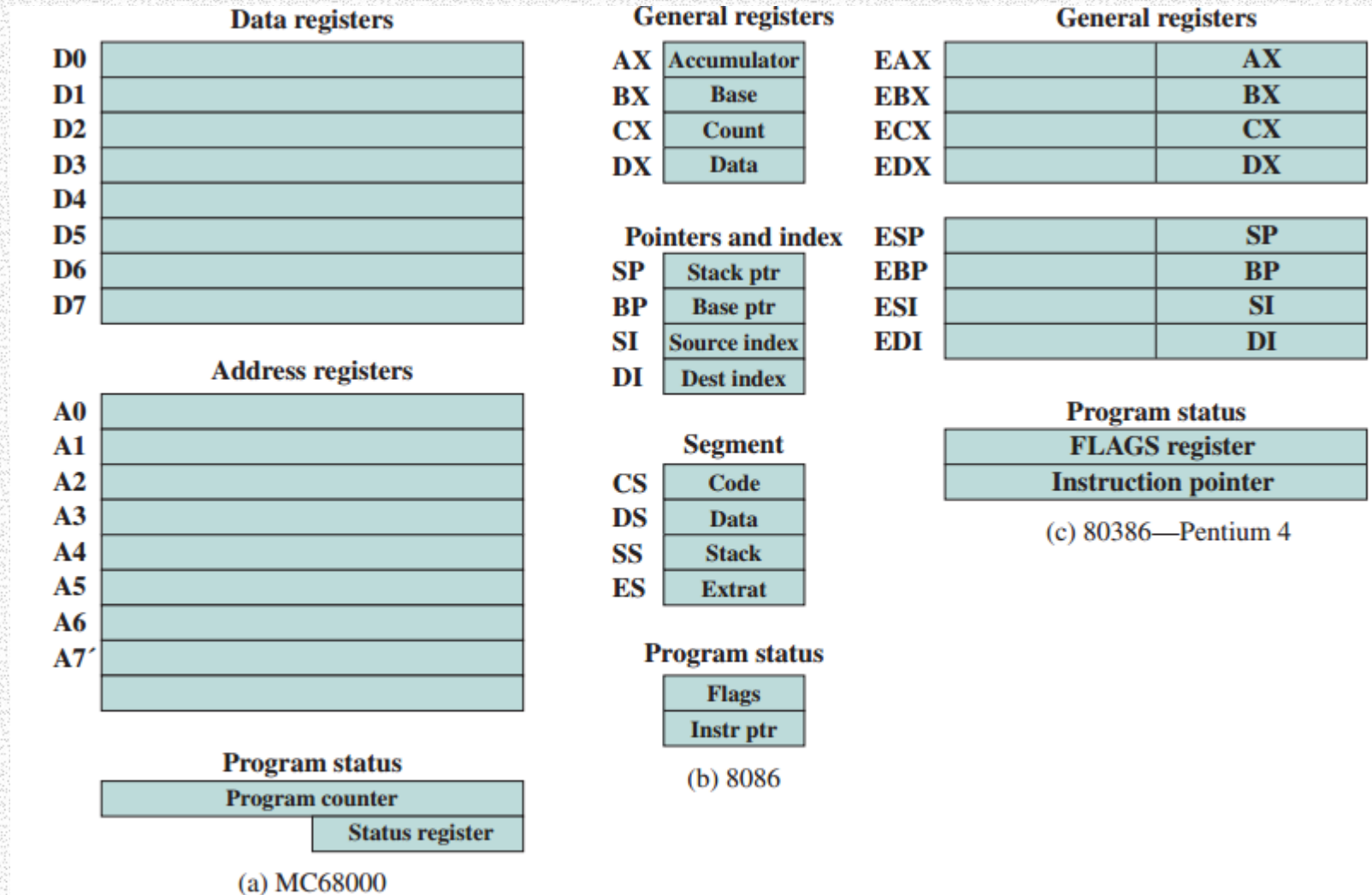


Figure 14.3 Example Microprocessor Register Organizations

Instruction cycle

- Instruction cycle includes the following stages
 - Fetch: Read the next instruction from memory into the processor
 - Execute: Interpret the opcode and perform the indicated operation.
 - Interrupt: If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.
 - Indirect addressing: Additional memory accesses are required

Instruction cycle

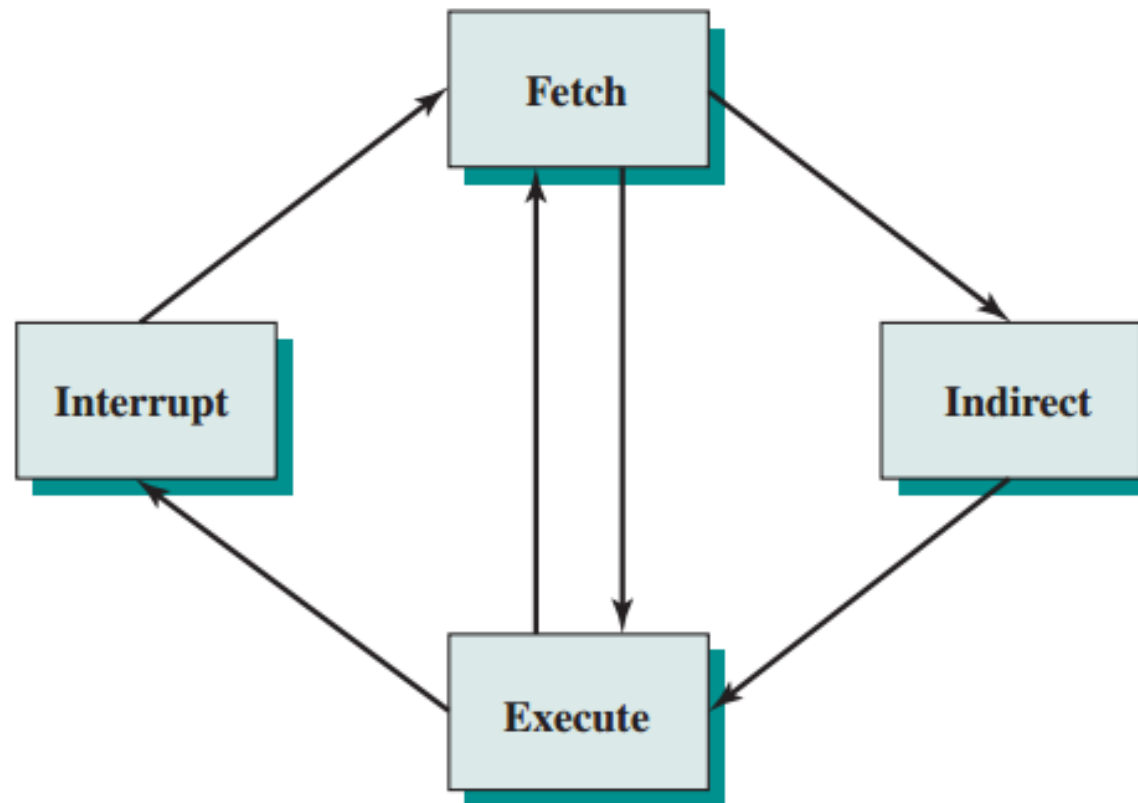


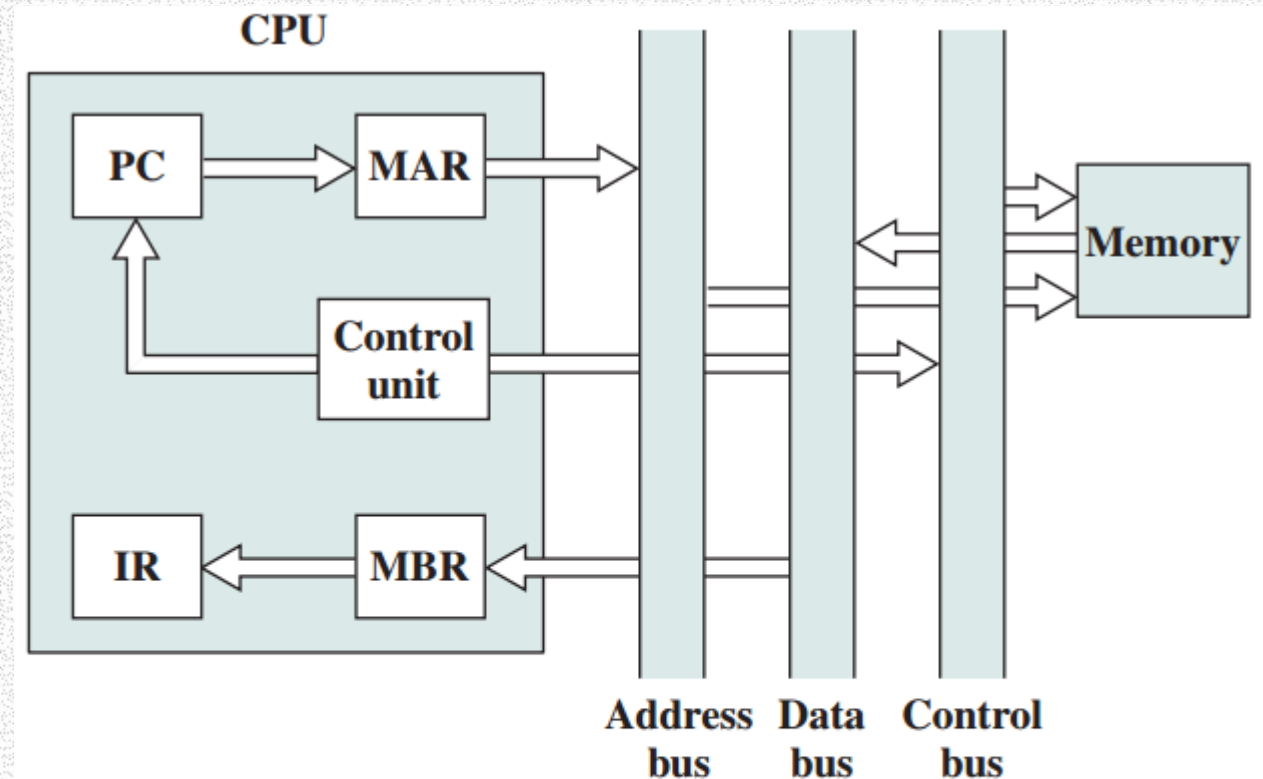
Figure 14.4 The Instruction Cycle

Data Flow

- Fetch cycle: An instruction is read from memory
- 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
- Result is placed on the data bus and copied into the MBR and then moved to the IR
- PC is incremented by 1, preparatory for the next fetch
- Control unit examines the contents of the IR to determine if it contains an operand specifier using indirect addressing

Data Flow

- Indirect cycle
 - The rightmost N bits of the MBR, which contain the address reference, are transferred to the MAR
 - Control unit requests a memory read, to get the desired address of the operand into the MBR



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

Figure 14.6 Data Flow, Fetch Cycle

Data Flow

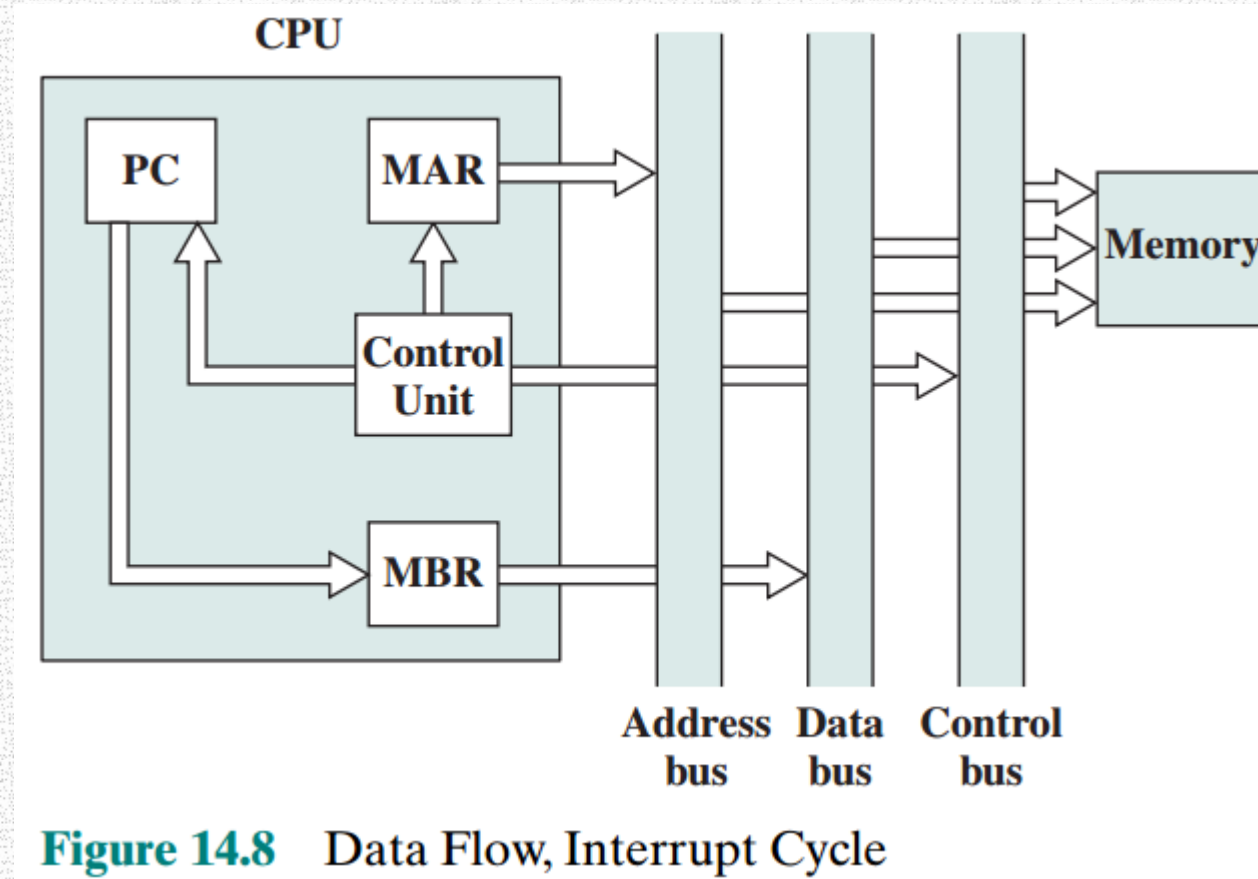
- Execute cycle takes many forms
 - Involve transferring data among registers
 - Read or write from memory or I/O
 - Invocation of the ALU

Data Flow

- Interrupt cycle
 - Current contents of the PC must be saved so that the processor can resume normal activity after the interrupt
 - Contents of the PC are transferred to the MBR to be written into memory
 - The special memory location reserved for this purpose is loaded into the MAR from the control unit (might be a stack pointer)
 - PC is loaded with the address of the interrupt routine
 - As a result, the next instruction cycle will begin by fetching the appropriate instruction

Data Flow

- Interrupt cycle



Instruction Pipelining

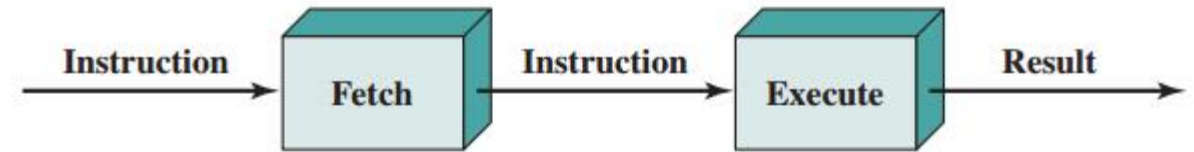
- Similar to the use of an assembly line in a manufacturing plant
- Products at various stages can be worked on simultaneously (pipelining)
- New inputs are accepted at one end before previously accepted inputs appear as outputs at the other end
- Instruction execution must have number of stages to apply this concept

Instruction Pipelining

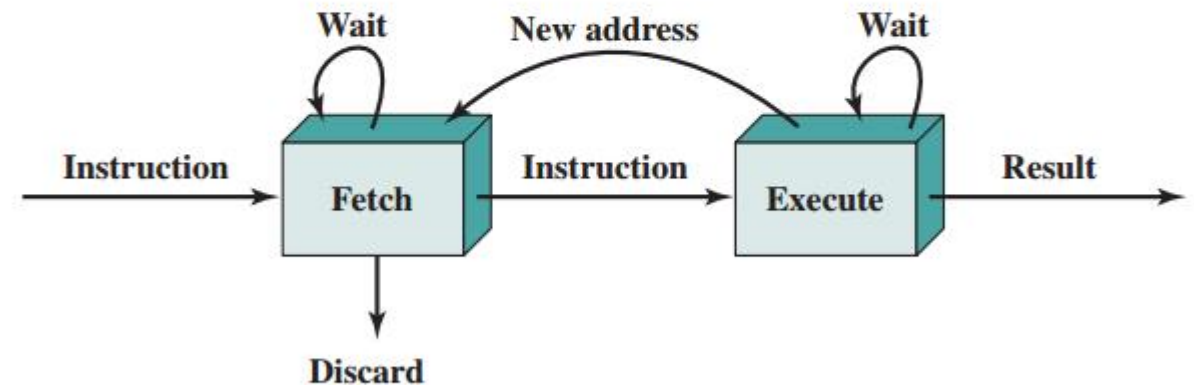
- Consider subdividing instruction processing into two stages
 - Fetch instruction
 - Execute instruction
- There are times during the execution of an instruction when main memory is not being accessed
- Fetch the next instruction in parallel with the execution of the current one.

Instruction Pipelining

- 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, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction
- This is called instruction prefetch



(a) Simplified view



(b) Expanded view

Figure 14.9 Two-Stage Instruction Pipeline

Instruction Pipelining

- If the fetch and execute stages were of equal duration, the instruction cycle time would be halved
 - The execution time will generally be longer than the fetch time
 - Fetch stage may have to wait for some time
 - A conditional branch instruction makes the address of the next instruction to be fetched unknown
 - Fetch stage must wait until it receives the next instruction address from the execute stage
 - Execute stage may then have to wait while the next instruction is fetched
 - Pipeline can gain further speedup by having more stages

Decomposition of the instruction processing

- **Fetch instruction (FI):** Read the next expected instruction into a buffer
- **Decode instruction (DI):** Determine the opcode and the operand specifiers.
- **Calculate operands (CO):** Calculate the effective address of each source operand. This may involve displacement, register indirect, indirect, or other forms of address calculation.
- **Fetch operands (FO):** Fetch each operand from memory. Operands in registers need not be fetched
- **Execute instruction (EI):** Perform the indicated operation and store the result, if any, in the specified destination operand location.
- **Write operand (WO):** Store the result in memory

Instruction Pipelining

- Assume the various stages will be of more nearly equal duration
 - Six-stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units
- ▶ If the six stages are not of equal duration, there will be some waiting involved at various pipeline stages
 - ▶ Conditional branch instruction will invalidate several instruction fetches
 - ▶ In an unpredictable event like interrupt

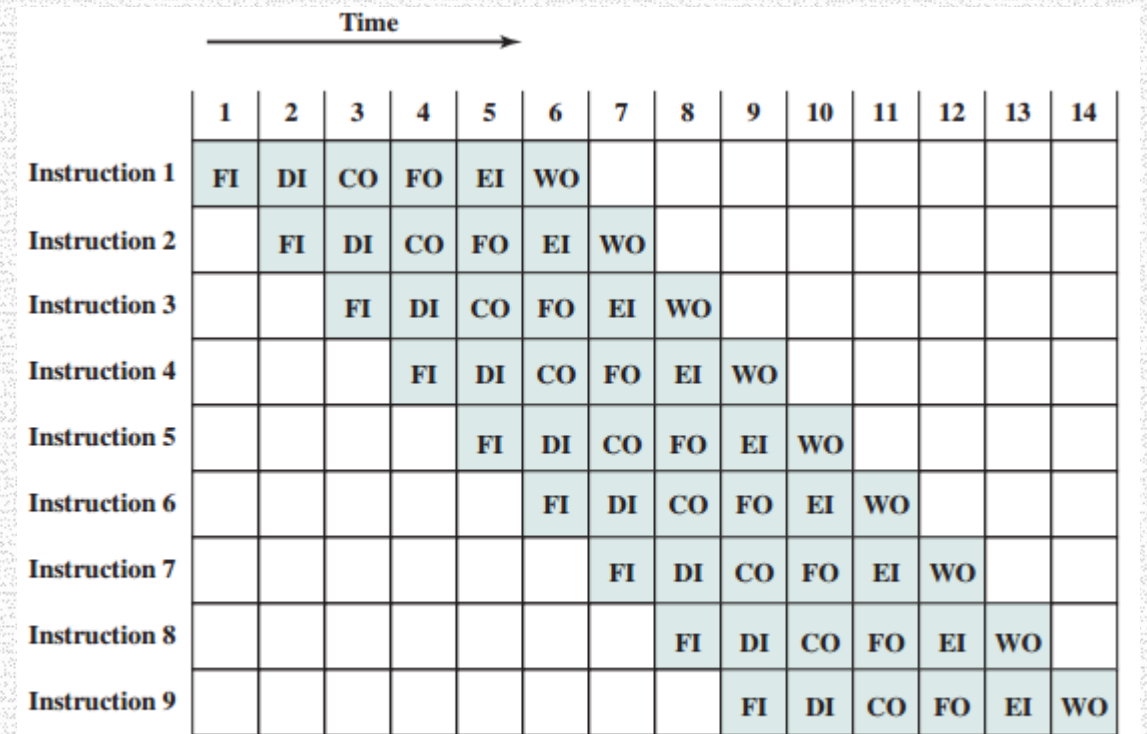


Figure 14.10 Timing Diagram for Instruction Pipeline Operation

Instruction Pipelining

- Assume that 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
- No instructions complete during time units 9 through 12
- This is the performance penalty

	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

Figure 14.11 The Effect of a Conditional Branch on Instruction Pipeline Operation

Pipeline Performance

- The cycle time of an instruction pipeline is the time needed to advance a set of instructions one stage through the pipeline

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

where

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

In general, the time delay d is equivalent to a clock pulse and $\tau_m \gg d$.

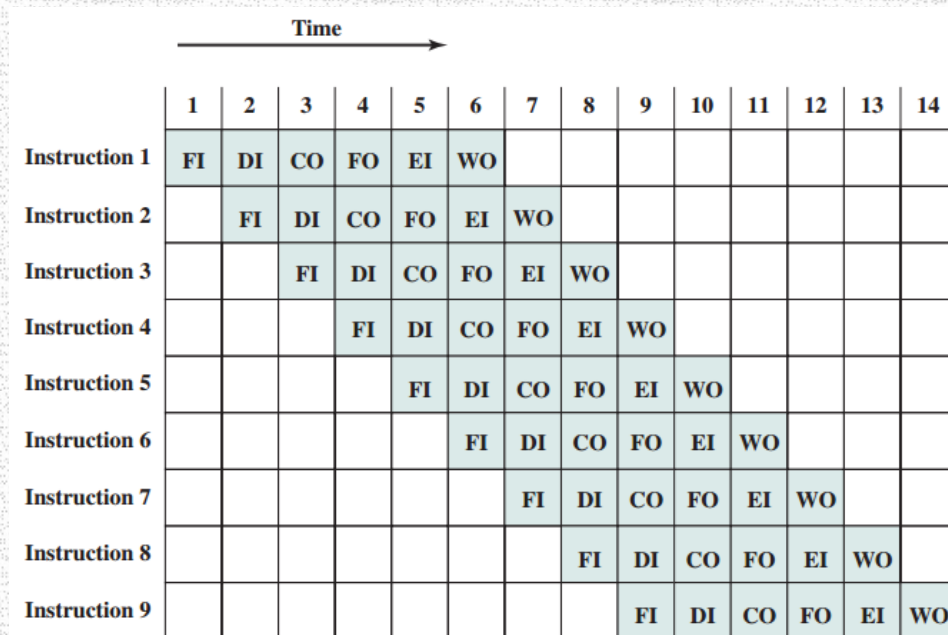
Pipeline Performance

- Let $T_{k,n}$ be the total time required for a pipeline with k stages to execute n instructions

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

- k cycles are required to complete the execution of the first instruction
- Remaining $(n - 1)$ instructions require $(n - 1)$ cycles

$$14 = [6 + (9 - 1)]$$



The diagram shows a 9x14 grid representing the execution of 9 instructions over 14 time cycles. The columns are labeled 1 through 14, and the rows are labeled Instruction 1 through Instruction 9. A horizontal arrow above the grid points to the right, labeled 'Time'. Each cell in the grid contains a two-letter code representing a pipeline stage: FI (Fetch Instruction), DI (Decode Instruction), CO (Calculate Operands), FO (Fetch Operands), EI (Execute Instruction), and WO (Write Back). The stages for each instruction are shifted one cycle to the right for each subsequent instruction, illustrating the pipeline's progression.

	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

Figure 14.10 Timing Diagram for Instruction Pipeline Operation

Pipeline Performance

- A processor with equivalent functions but no pipeline
- Assume that the instruction cycle time is $k\tau$
- The speedup factor for the instruction pipeline compared to execution without the pipeline is defined as

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

- When n tends to ∞ we get a k -fold speedup

Pipeline Performance

- In a CPU of 5 stage pipeline, first 2 stages take 30ns each while next 3 stages take 50ns each. Assuming no hazards occurring, calculate the speed-up of the CPU for a program of 20 instructions

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

Pipeline Hazards

- Pipeline hazard occurs when the pipeline, or some portion of the pipeline, must stall because conditions do not permit continued execution
 - Resource Hazards
 - Data Hazards
 - Control Hazards

Pipeline Hazards

- A resource hazard occurs when two (or more) instructions that are already in the pipeline need the same resource
- Instructions must be executed in serial rather than parallel for a portion of the pipeline

Resource Hazards

- Assume a simplified five-stage pipeline, in which each stage takes one clock cycle.
- In ideal case a new instruction enters the pipeline each clock cycle

	Clock cycle								
	1	2	3	4	5	6	7	8	9
	I1	FI	DI	FO	EI	WO			
	I2		FI	DI	FO	EI	WO		
	I3			FI	DI	FO	EI	WO	
I4				FI	DI	FO	EI	WO	

(a) Five-stage pipeline, ideal case

Resource Hazards

- Assume that main memory has a single port and that all instruction fetches and data reads and writes must be performed one at a time
- An operand read to or write from memory cannot be performed in parallel with an instruction fetch
- Fetch instruction stage of the pipeline must idle for one cycle

		Clock cycle								
		1	2	3	4	5	6	7	8	9
Instruction	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			Idle	FI	DI	FO	EI	WO	
	I4					FI	DI	FO	EI	WO

(b) I1 source operand in memory

Resource Hazards

- Multiple instructions are ready to enter the execute instruction phase and there is a single ALU
- Solutions to such resource
 - Having multiple ports into main memory
 - Multiple ALU units

Data Hazards

- A data hazard occurs when there is a conflict in the access of an operand location

As an example, consider the following x86 machine instruction sequence:

```
ADD EAX,    EBX /* EAX = EAX + EBX
```

```
SUB ECX,    EAX /* ECX = ECX - EAX
```

- The first instruction adds the contents of the 32-bit registers EAX and EBX and stores the result in EAX
- The second instruction subtracts the contents of EAX from ECX and stores the result in ECX
- Such a data hazard results in inefficient pipeline usage

Data Hazards

- The ADD instruction does not update register EAX until the end of stage 5
- SUB instruction needs that value at the beginning of its stage 2, which occurs at clock cycle 4
- The pipeline must stall for two clocks cycles

		Clock cycle									
		1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX		FI	DI	FO	EI	WO					
	SUB ECX, EAX		FI	DI	Idle		FO	EI	WO		
	I3			FI			DI	FO	EI	WO	
	I4						FI	DI	FO	EI	WO

Figure 14.16 Example of Data Hazard

Control Hazards

- Pipeline makes the wrong decision on a branch prediction and brings instructions into the pipeline

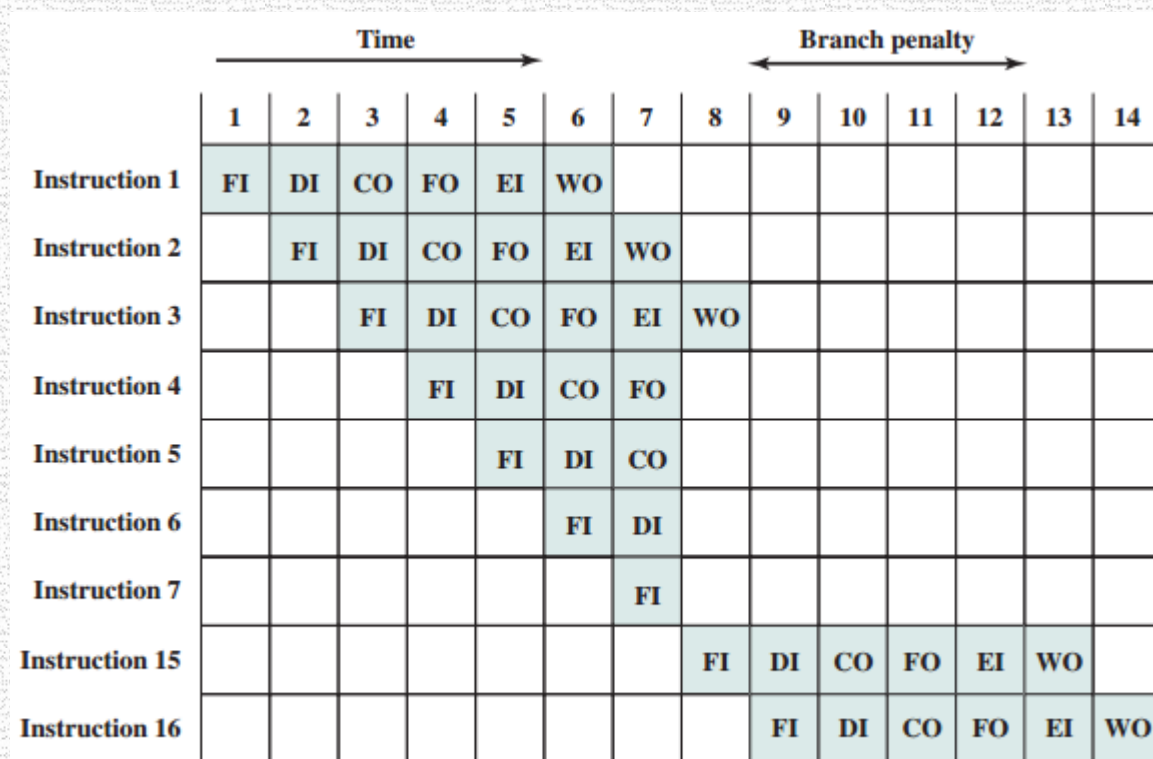


Figure 14.11 The Effect of a Conditional Branch on Instruction Pipeline Operation

Reference

- William Stallings - Computer Organization and Architecture Designing for Performance (9th Edition)
 - 14.1 Processor Organization
 - 14.2 Register Organization
 - 14.3 Instruction Cycle
 - 14.4 Instruction Pipelining

Questions

- List FIVE things done by a processor.
- Explain the stages of Instruction cycle using a diagram.
- Explain the data flow of the fetch cycle using a diagram.
- Derive a equation for Speed-up $S_{k,n}$ as a function of n-number executed instructions, and k- number of stages of the pipeline assuming ideal conditions
- Explain how data hazard occurs in Instruction Pipelining.