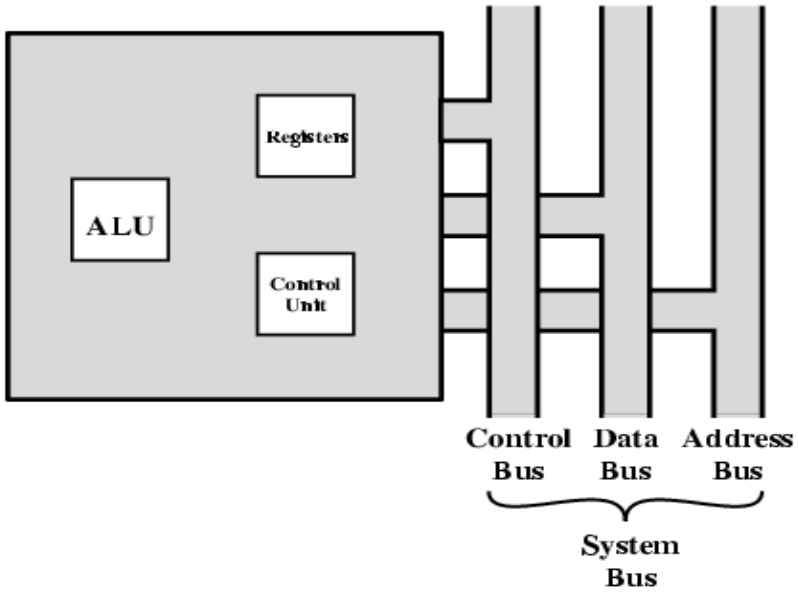
CA - Module – III : Processor Structure

1. CPU Structure

1.1. The steps involved in Instruction execution:

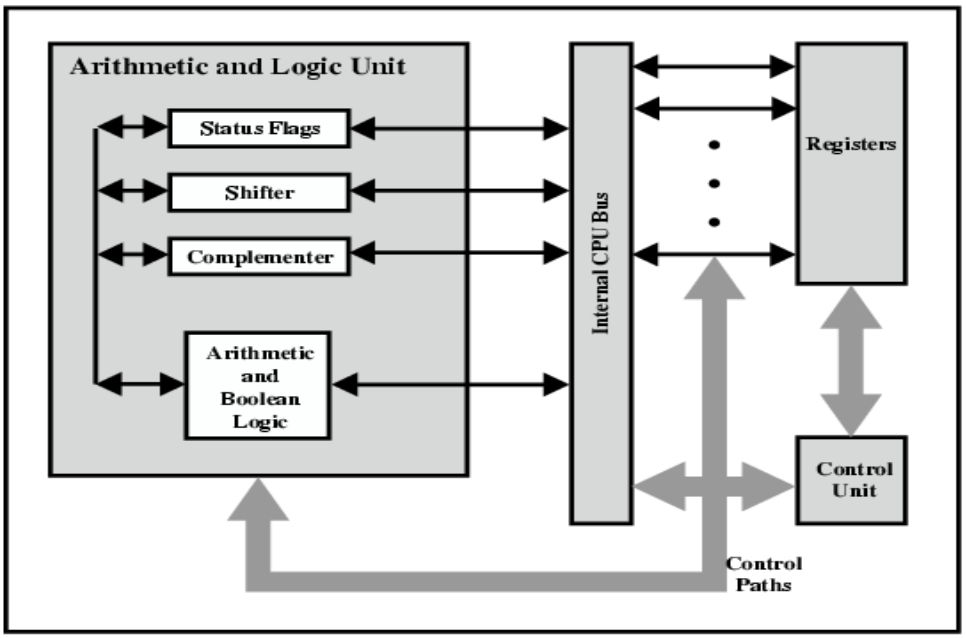
- **Fetch instruction**: The processor reads an instruction from memory.
- **Interpret instruction**: The instruction is decoded to determine what action is required.
- **Fetch data**: The execution of an instruction may require reading data from memory or an I/O module.
- **Process data**: The execution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

1.2. CPU With Systems Bus



- It is a simplified view of a processor, indicating its connection to the rest of the system via the system bus.
- The ALU does the actual computation or processing of data.
- The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU.
- The registers is used for internal storage.

2. CPU Internal Structure



- The ALU does the actual computation or processing of data.
- The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU.
- The registers is used for internal storage.
- The data transfer and logic control paths includes an element labeled internal processor bus.
- This element is needed to transfer data between the various registers and the ALU.
- CPU must have some working space (temporary storage) Called registers

3. REGISTER ORGANIZATION

- The registers in the processor perform two roles:
  - **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 and by privileged, operating system programs to control the execution of programs.

3.1. User-Visible Registers

- A user-visible register is one that is referenced by machine language that the processor executes.
- We can characterize these in the following categories:
  - General purpose Registers
  - Data Registers
  - Address Registers
  - Condition Code
- **General purpose Registers**

- General-purpose registers can be assigned to a variety of functions by the programmer.
- Any general-purpose register can contain the operand for any opcode.

- **Data Registers**
  - Data registers are used to hold data
- **Address Registers**
  - Used to hold addresses
- **Condition codes**
  - Condition codes are bits set by the processor hardware as the result of operations
  - An arithmetic operation may produce a positive, negative, zero, or overflow result.
  - In addition to the result itself being stored in a register or memory, a condition code is also set.

3.2. Control and Status Registers

- Four registers are 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
- These four registers are used for the movement of data between the processor and memory.
- Within the processor, data must be presented to the ALU for processing.
- Many processor contains a register or set of registers the *program status word (PSW), that contain status information.*
- *The PSW typically* contains condition codes plus other status information.
- Common fields or flags include the following:
  - **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.
  - **Equal :** Set if a logical compare result is equality.
  - **Overflow:** Used to indicate arithmetic overflow.
  - **Interrupt Enable/Disable:** Used to enable or disable interrupts.
  - **Supervisor:** Indicates whether the processor is executing in supervisor or user mode.

4. The Indirect Cycle

- The execution of an instruction may involve one or more operands in memory, each of which requires a memory access.
- Further, if indirect addressing is used, then additional memory accesses are required.
- The execution of an instruction consists of alternating instruction fetch and instruction execution activities.

- After an instruction is fetched, it is examined to determine if any indirect addressing is involved.
- If so, the required operands are fetched using indirect addressing.
- Following execution, an interrupt may be processed before the next instruction fetch.

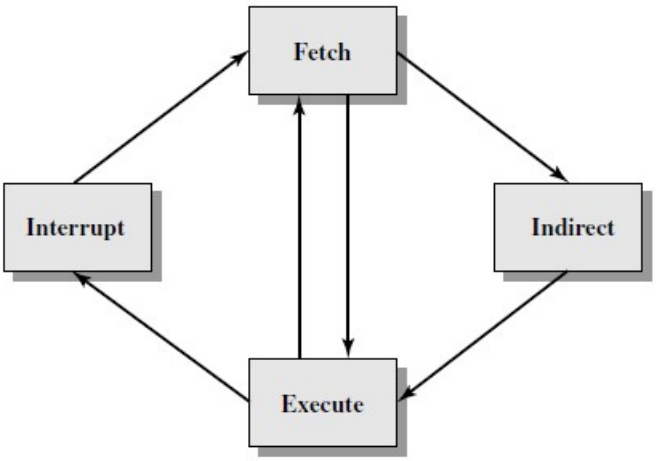
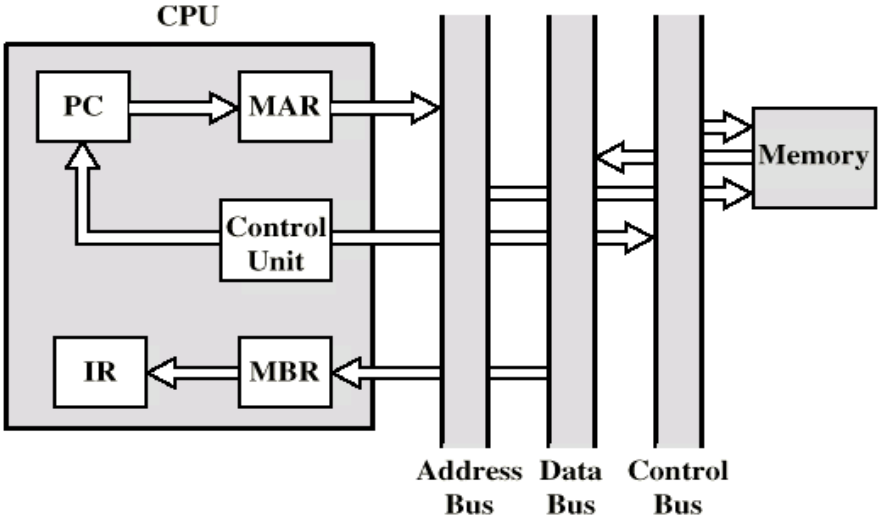


Figure 12.4 The Instruction Cycle

5. Data Flow (Instruction Fetch)

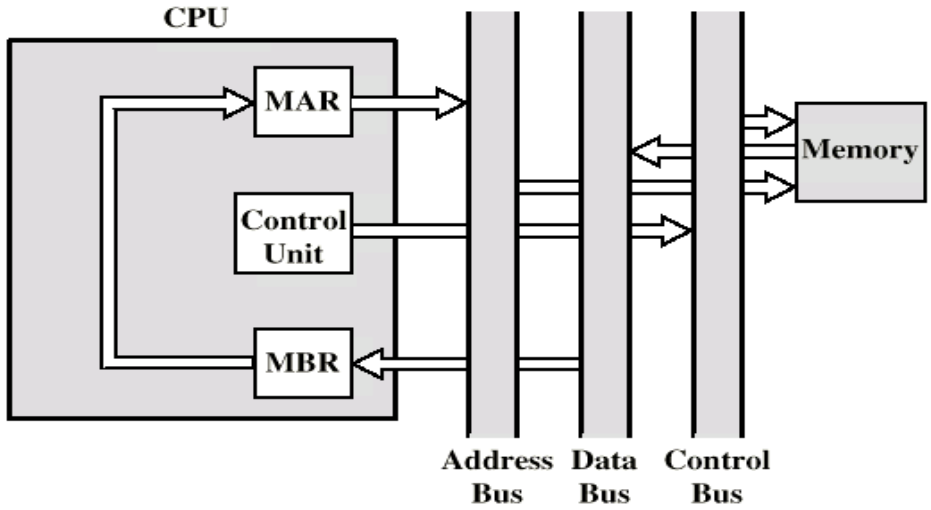
- During the *fetch cycle, an instruction is read from memory.*
- The PC contains the address of the next instruction to be fetched.
- This address is moved to the MAR and placed on the address bus.
- The control unit requests a memory read, and the result is placed on the data bus and copied into the MBR and then moved to the IR.
- Meanwhile, the PC is incremented by 1, preparatory for the next fetch.



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

6. Data Flow (Data Fetch)

- Once the fetch cycle is over, the control unit examines the contents of the IR to determine if it contains an operand specifier using indirect addressing.
- If so, an *indirect cycle is performed.*
- The rightmost N bits of the MBR, 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.

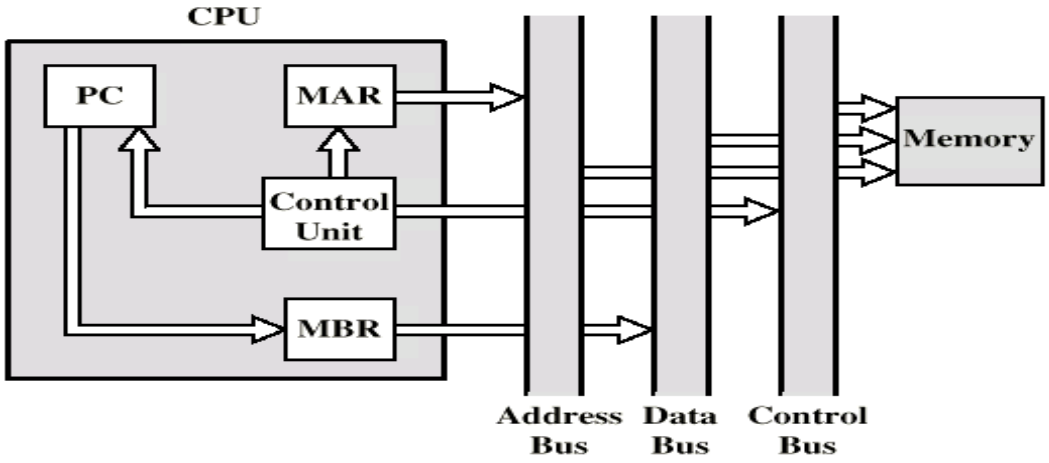


7. Data Flow (Execute Cycle)

- The *execute cycle* takes many forms; the form depends on which of the various machine instructions is in the IR.
- This cycle may involve transferring data among registers, read or write from memory or I/O, and/or the invocation of the ALU.

8. Data Flow (Interrupt)

- The current contents of the PC must be saved so that the processor can resume normal activity after the interrupt.
- Thus, the 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.
- The PC is loaded with the address of the interrupt routine.



9. Instruction Pipelining

1. Processors make use of instruction pipelining to speed up execution.
2. Pipelining involves breaking up the instruction cycle into a number of separate stages that occur in sequence, such as fetch instruction, decode instruction, determine operand addresses, fetch operands, execute instruction, and write operand result.
3. Instructions move through these stages, as on an assembly line, so that each stage can be working on a different instruction at the same time.
4. Instruction pipelining improves the overall system performance.
5. Instruction pipelining is similar to the use of an assembly line in a manufacturing plant.
6. This process is also referred to as *pipelining, because, as in a pipeline, new inputs are accepted at one end* before previously accepted inputs appear as outputs at the other end.

9.1. Two Stage Instruction Pipeline

- Instruction processing is subdivided into two stages: **fetch instruction** and **execute instruction**.
- There are times during the execution of an instruction when main memory is not being accessed.
- This time could be used to fetch the next instruction in parallel with the execution of the current one.
- 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*** *or* ***fetch overlap****.*

9.2. Six-stage Instruction Pipeline

- Instruction Execution is decomposed into the following stages
  - **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 execution consists of 6 stages.
- All these stages can be executed in parallel.

Timing Diagram for Instruction Pipeline Operation