Computer Architecture and Operating System

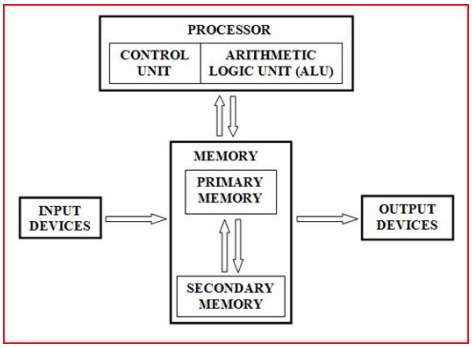
Prof. Indranil Sengupta

Department of Computer Science and Engineering

IIT Kharagpur

Basic Operation of a Computer

Introduction



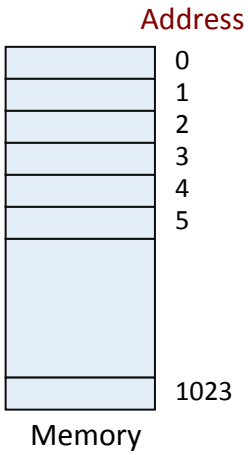
- The basic mechanism through which an instruction gets executed shall be discussed.
  - *Fetch-Decode-Execute* cycle.
- May be recalled:
  - ALU contains a set of registers, some general-purpose and some special-purpose.
  - First we briefly explain the functions of the special-purpose registers before we look into some examples.

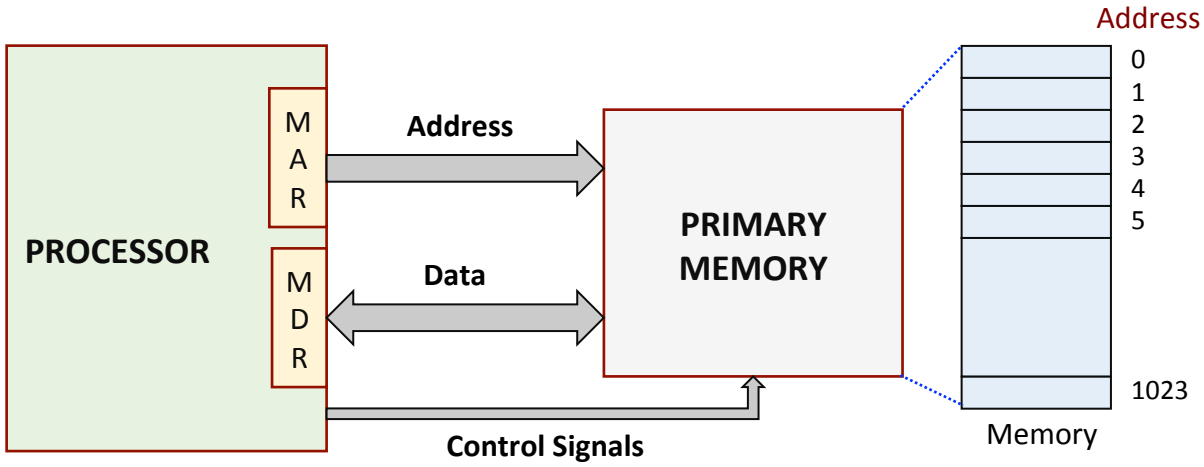
Special-purpose Registers: For Keeping Track of Program / Instructions

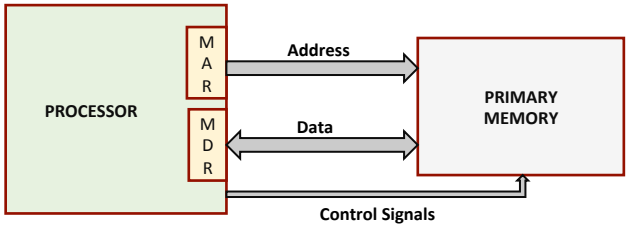
- Two special-purpose registers are used:
  - **Program Counter (PC)**: Holds the memory address of the next instruction to be executed.
    - Automatically incremented to point to the next instruction when an instruction is being executed.
  - **Instruction Register (IR)**: Temporarily holds an instruction that has been fetched from memory.
    - Need to be decoded to find out the instruction type.
    - Also contains information about the location of the data.

Special-purpose Registers: For Interfacing with the Primary Memory

- Two special-purpose registers are used:
  - **Memory Address Register (MAR)**: Holds the address of the memory location to be accessed.
  - **Memory Data Register (MDR)**: Holds the data that is being written into memory, or will receive the data being read out from memory.
- Memory considered as a linear array of storage locations (bytes or words) each with unique address.

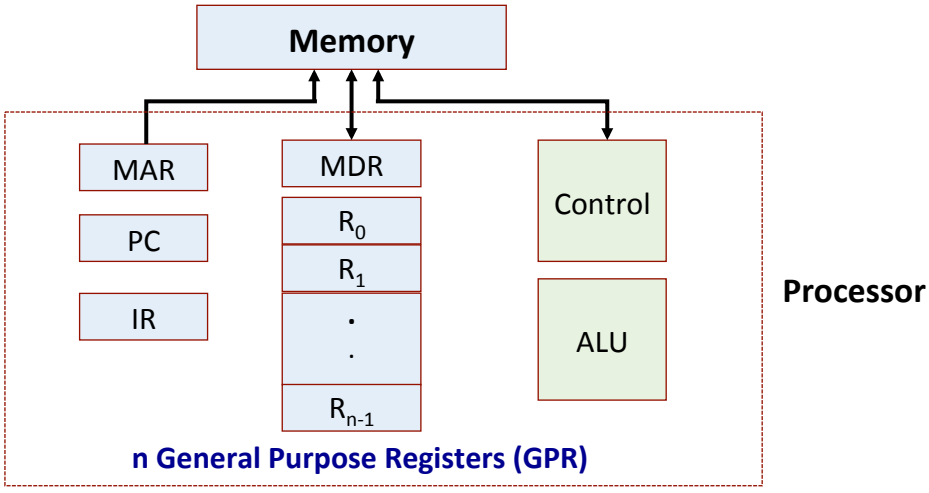






- To read data from memory
  - Load the memory address into MAR. — MAR ← address
  - Issue the control signal READ. — Read
    MDR ← Mem[MAR]
- To write data into memory
  - Load the memory address into MAR. — MAR ← address
  - Load the data to be written into MDR. — MDR ← data
  - Issue the control signal WRITE. — Write
    Mem[MAR] ← MDR

Architecture of an Example Processor



Example Instructions

- We shall illustrate the process of instruction execution with the help of the following two instructions:

a) ADD R1, LOCA

Add the contents of memory location LOCA (i.e. address of the memory location is LOCA) to the contents of register R1.

R1 ← R1 + Mem[LOCA]

b) ADD R1, R2

Add the contents of register R2 to the contents of register R1.

R1 ← R1 + R2

Execution of *ADD R1,LOCA*

- Assume that the instruction is stored in memory location 1000, the initial value of R1 is 50, and LOCA is 5000.
- Before the instruction is executed, PC contains 1000.
- Content of PC is transferred to MAR. MAR ← PC
- READ request is issued to memory unit. Read
- The instruction is fetched to MDR. MDR ← Mem[MAR]
- Content of MDR is transferred to IR. IR ← MDR
- PC is incremented to point to the next instruction. PC ← PC + 4
- The instruction is decoded by the control unit.

ADD R1	5000

- LOCA (i.e. 5000) is transferred (from IR) to MAR. MAR ← IR[Operand]
- READ request is issued to memory unit. Read
- The data is fetched to MDR. MDR ← Mem[MAR]
- The content of MDR is added to R1. R1 ← R1 + MDR

The steps being carried out are called micro-operations:

MAR ← PC
MDR ← Mem[MAR]
IR ← MDR
PC ← PC + 4
MAR ← IR[Operand]
MDR ← Mem[MAR]
R1 ← R1 + MDR

R1 125

Address	Content
1000	ADD R1, LOCA
1004	…

5000	75

LOCA

1. PC = 1000
2. MAR = 1000
3. PC = PC + 4 = 1004
4. MDR = ADD R1, LOCA
5. IR = ADD R1, LOCA
6. MAR = LOCA = 5000
7. MDR = 75
8. R1 = R1 + MDR = 50 + 75 = 125

Execution of *ADD R1,R2*

- Assume that the instruction is stored in memory location 1500, the initial value of R1 is 50, and R2 is 200.
- Before the instruction is executed, PC contains 1500.
- Content of PC is transferred to MAR. — MAR ← PC
- READ request is issued to memory unit. — Read
- The instruction is fetched to MDR. — MDR ← Mem[MAR]
- Content of MDR is transferred to IR. ADD R1, R2 — IR ← MDR
- PC is incremented to point to the next instruction. — PC ← PC + 4
- The instruction is decoded by the control unit.
- R2 is added to R1. — R1 ← R1 + R2

R1 250

R2 200

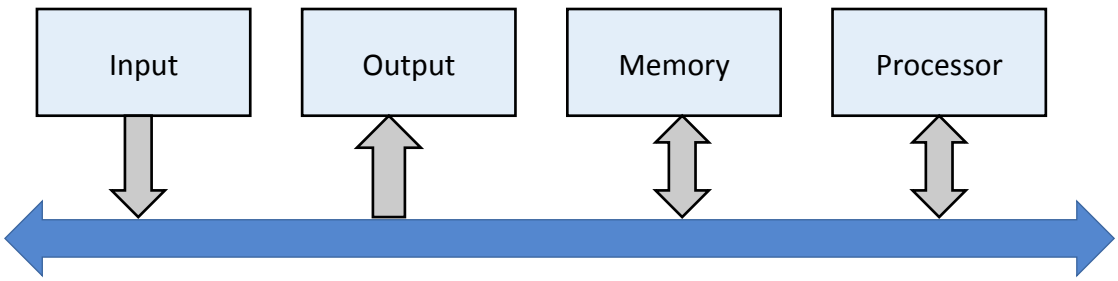
Address	Instruction
1500	ADD R1, R2
1504	…

1. PC = 1500
2. MAR = 1500
3. PC = PC + 4 = 1504
4. MDR = ADD R1, R2
5. IR = ADD R1, R2
6. R1 = R1 + R2 = 250

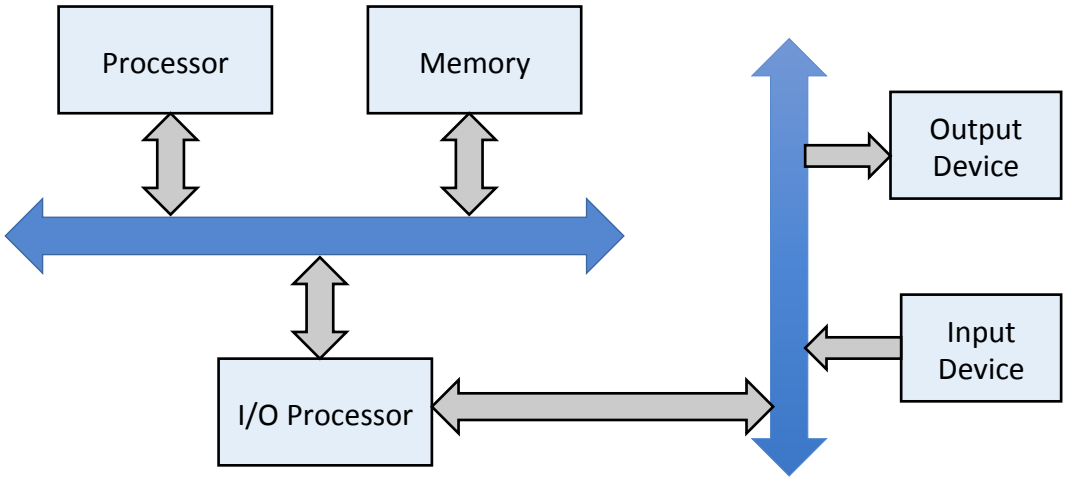
Bus Architecture

- The different functional modules within the processor must be connected in an organized manner to form an operational system.
- Bus refers to a group of lines that serves as a connecting path for several devices.
- The simplest way to connect the functional unit is to use the *single bus architecture*.
  - Only one data transfer allowed in one clock cycle.
  - For multi-bus architecture, parallelism in data transfer is allowed.

System-Level Single Bus Architecture

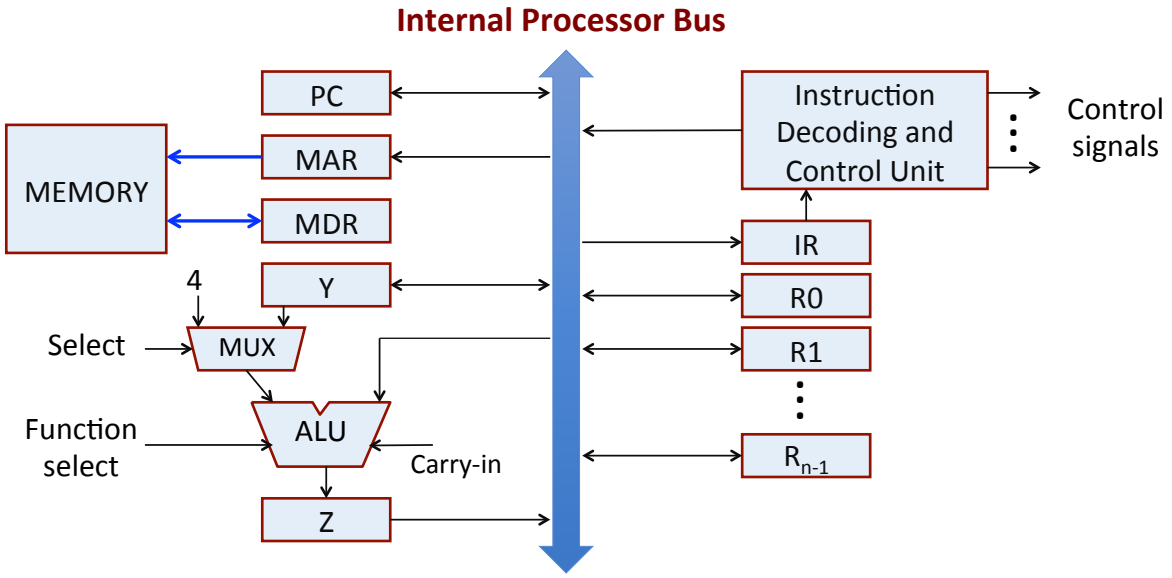


System-Level Two-Bus Architecture



Single-Bus Architecture Inside the Processor

- Suppose there is a single bus inside the processor.
  - ALU and the registers are all connected via the single bus.
  - An external bus can connect the processor to the memory and I/O devices.
- A typical single-bus processor architecture is shown on the next slide.
  - Two temporary registers *Y* and *Z* are also included.
  - Register *Y* temporarily holds one of the operands of the ALU.
  - Register *Z* temporarily holds the result of the ALU operation.
  - The multiplexer selects a constant operand 4 during execution of the micro-operation: $PC \leftarrow PC + 4$.



Multi-Bus Architectures

- Modern processors have multiple buses that connect the registers and other functional units.
  - Allows multiple data transfer micro-operations to be executed in the same clock cycle.
  - Results in overall faster instruction execution.
- Also advantageous to have multiple shorter buses rather than a single long bus.
  - Smaller parasitic capacitance, and hence smaller delay.