

CPU – Central Processing Unit

The component(s) responsible for :

- Fetching instructions from memory
- Decoding fetched instructions
- Executing decoded instructions
- Transferring data to and from memory or peripherals

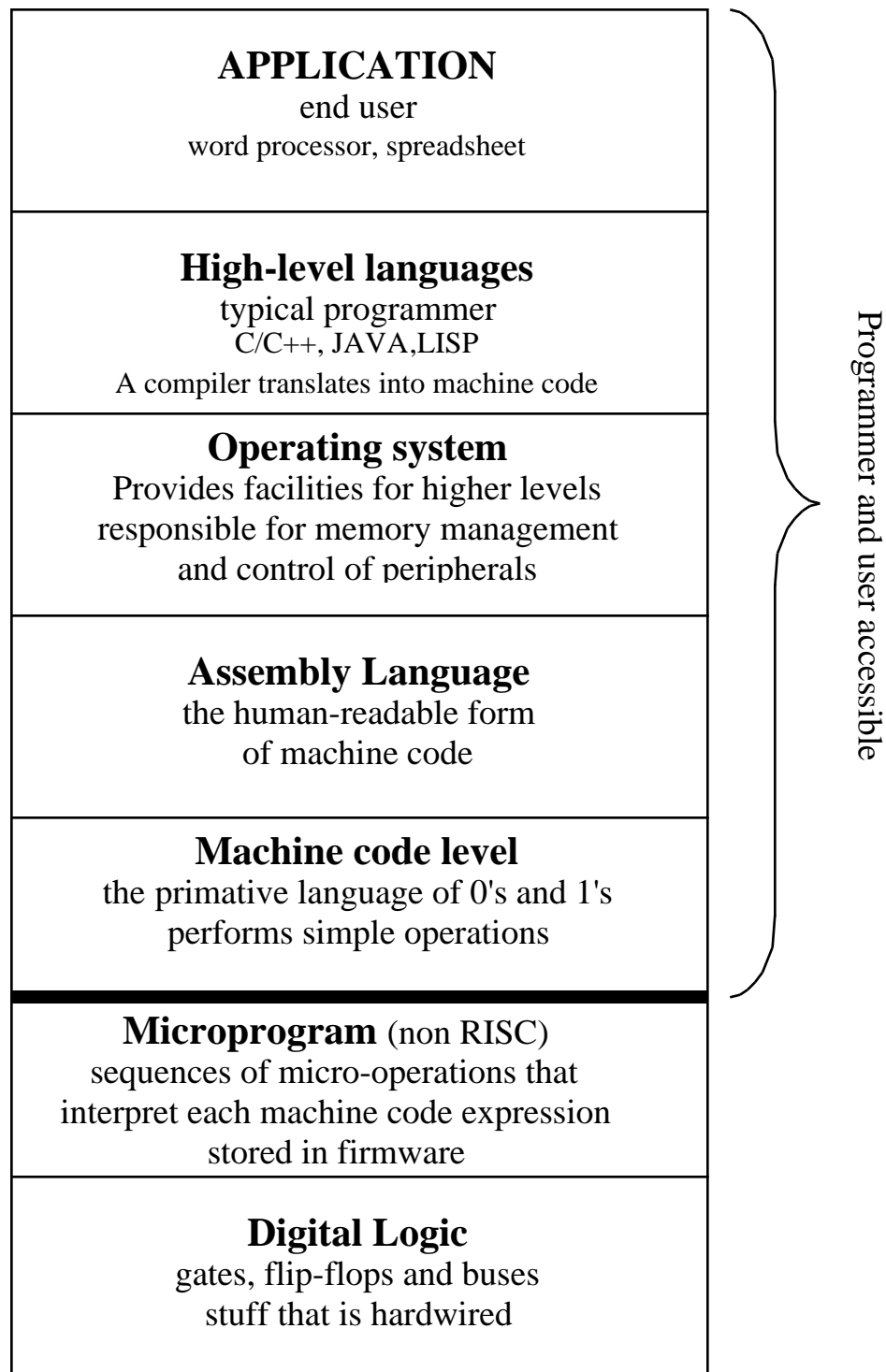
Microprocessor –

A CPU fabricated on a single chip of silicon.

A computer constructed around a microprocessor will be referred to as a microcomputer

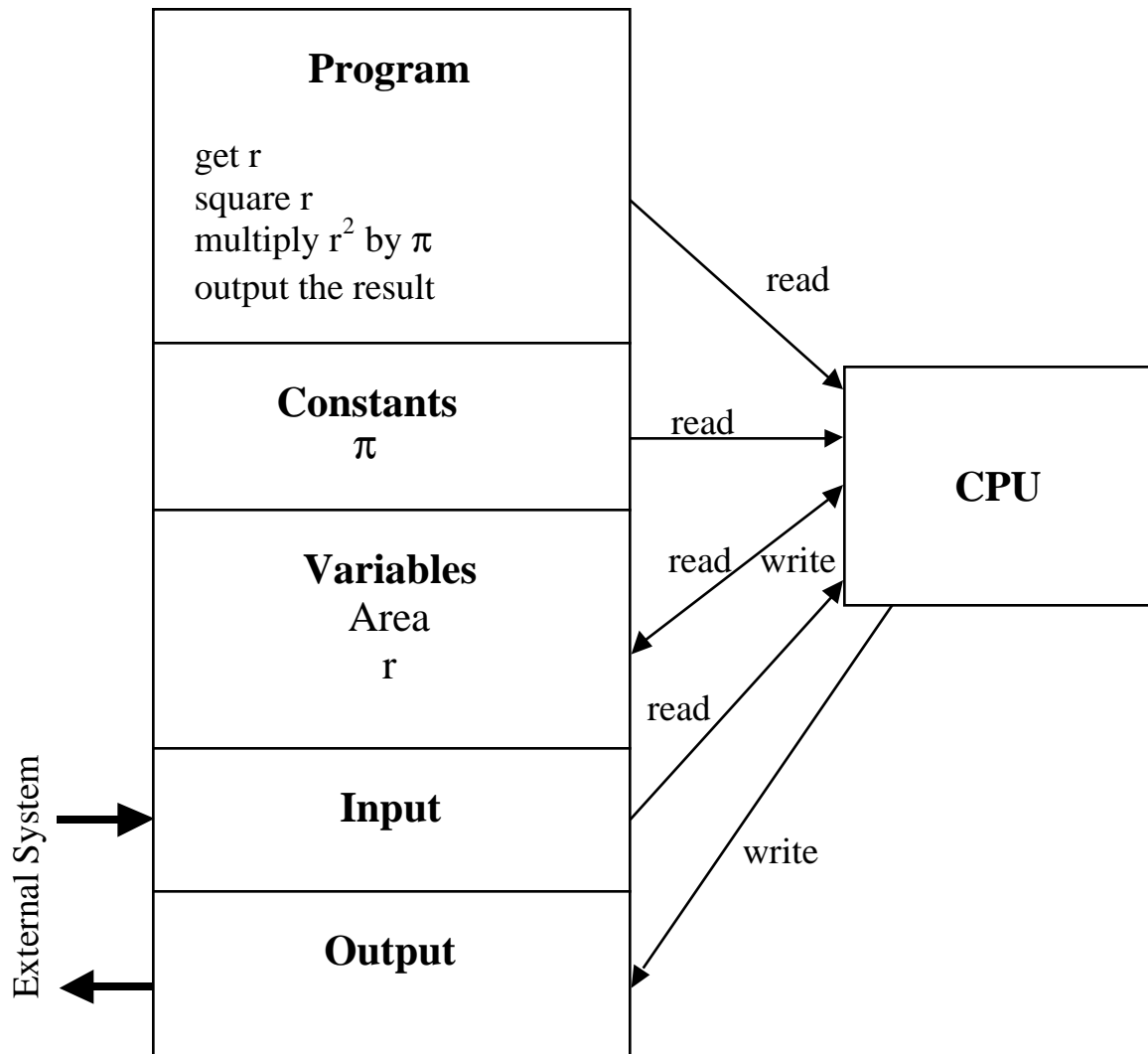
The CPU can be constructed from flip-flops and buses using gates with tri-state inputs.

The Different Levels at which individuals view a computer:



Introduction to the CPU

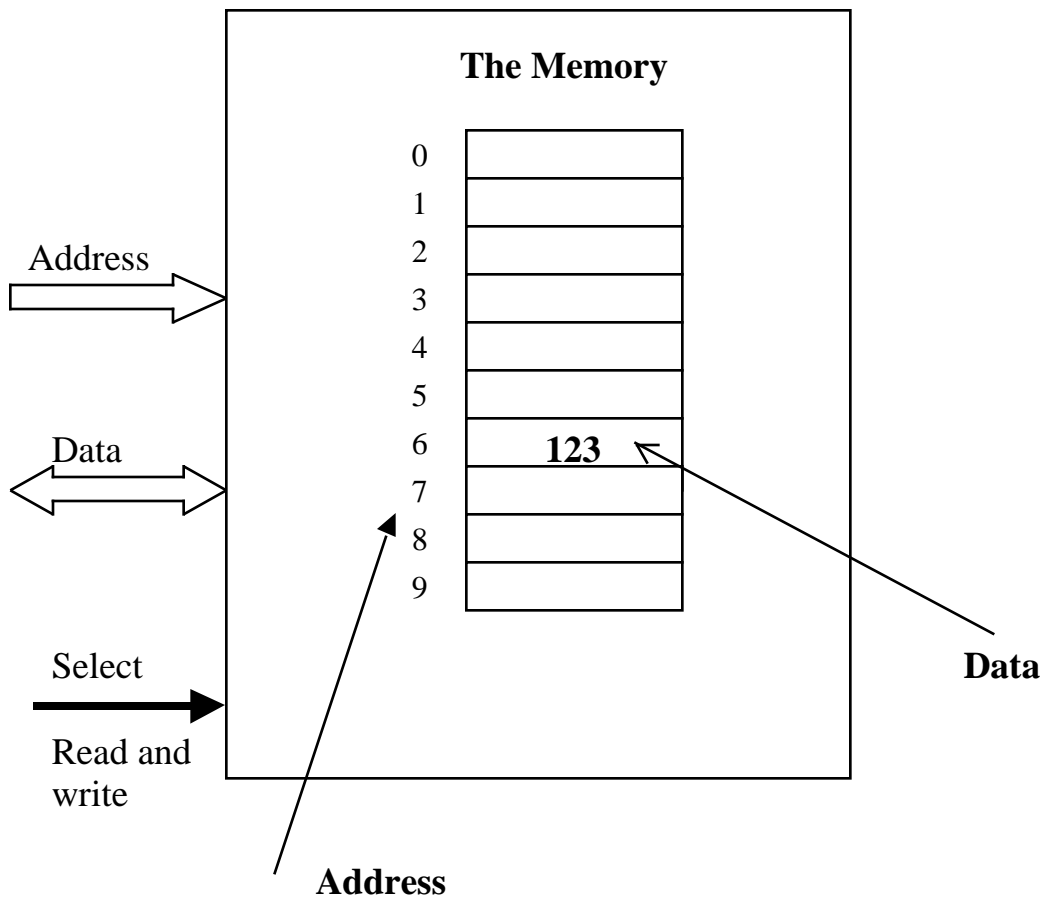
We start our introduction to the CPU by looking at the relationship between a hypothetical machine and how it could handle a simple program, that of calculating the area of a circle.



Memory –

RAM – random access memory

We view RAM as a block of sequential locations, each of which has a unique address determining the location and each of which contains a data element.



Note both memory address and data are binary quantities and we need to make sure we don't confuse address and contents.

Register Transfer Language (RTL) / Register Transfer Notation (RTN)

- a short hand algebraic notation we use to describe the workings of the CPU
- we use variables to denote registers. Ex . PC
- we use square brackets to indicate contents of
[PC] means the contents of PC
- a backwards arrow \leftarrow indicates transfer of data
[MAR] \leftarrow [PC] transfers a copy of the contents of PC to MR
- PC stands for the *program counter* -(used in relative addressing for locating operands)

To increment the program counter by 1: [PC] \leftarrow [PC]+1

- The computers memory will be referred to as M and to indicate the contents of x we will write [M(x)]

So [M(x)] is the data stored at address x in memory.

In our previous diagram we had 123 stored at location 6. We can now express this as [M(6)] =123 -- Note this is not assignment.

- We (the class & text) will adopt the following conventions:

Numbers with a % prefix are binary

Numbers with a \$ prefix are hexadecimal

The following are thus equivalent:

[PC] \leftarrow 120

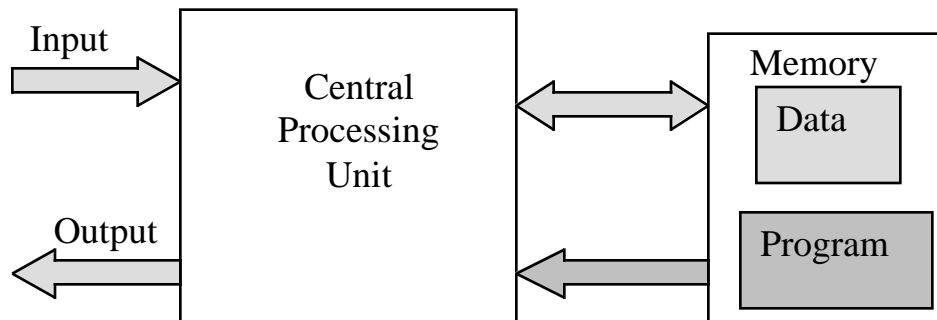
[PC] \leftarrow %1111000

[PC] \leftarrow &78

How do we interpret the following RTL expressions?

- (a) $[M(16)] = 320$
- (b) $[M(16)] \leftarrow 320$
- (c) $[M(16)] \leftarrow [M(5)]$
- (d) $[M(5)] \leftarrow [M(5)] + 2$

The Structure of the CPU



General-purpose digital computer

Note : Same memory contains both data and program – von Neumann machine.

The above machine shows two data paths between memory and CPU . In reality only one data path is actually used, thus data and instructions share the path. This often creates congestion on the bus between CPU and memory, resulting in reduced performance. This effect is called the **von Neumann bottleneck**.

The lower path depicts the sequence of commands that make up a program being read one-by-one from memory. These commands control the flow of data along the bi-directional data path.


During a write cycle data flows from the CPU to the memory. During a read cycle the CPU retrieves data from memory.

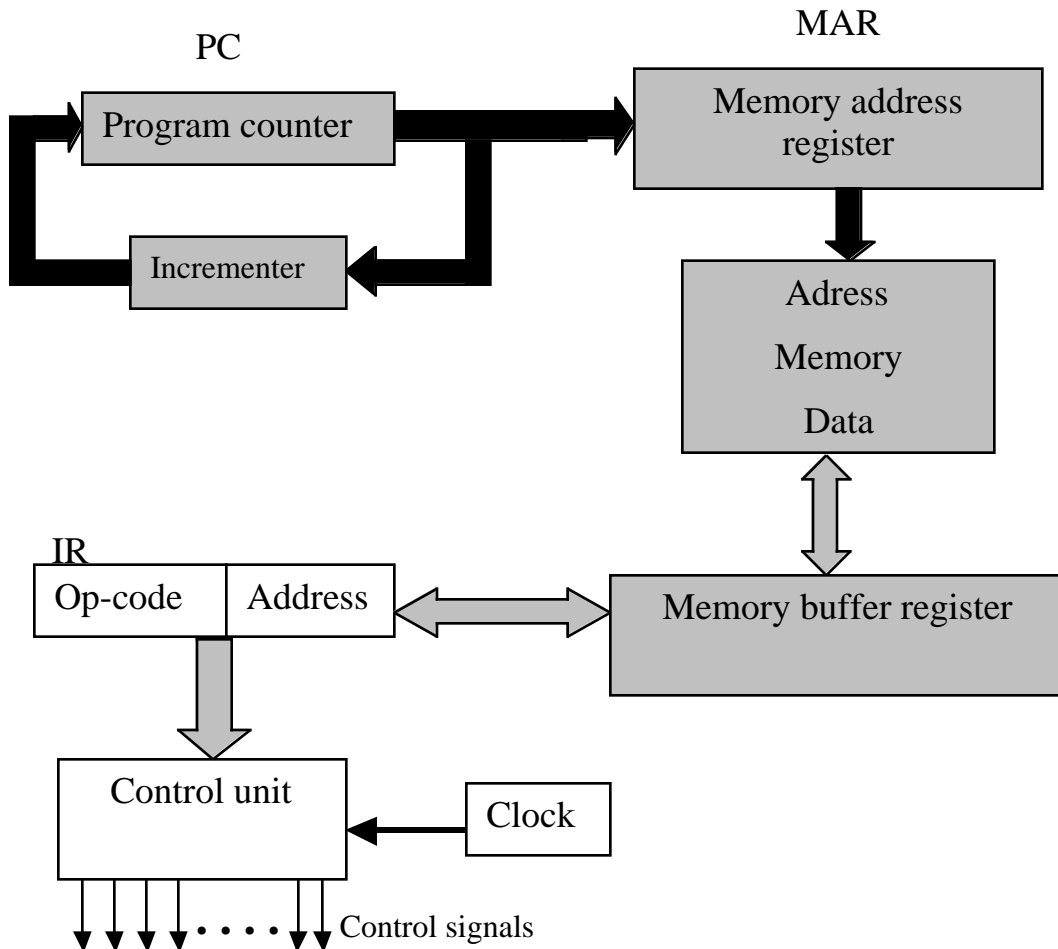
Consider the instruction

ADD X, Y, Z (meaning $z=x+y$)

- First CPU fetches the instruction from memory
- CPU decodes the instruction
- CPU retrieves X and Y
- CPU adds X and Y
- CPU sends result back to Z

The step by step construction of the CPU

Address Paths denoted by 



In order to execute an instruction it must first be brought from memory to the CPU. The program counter (PC) contains the address of the next instruction in memory to be executed (it points to it)

The contents of the program counter get moved to the Memory Address Register (MAR). In RTL this is denoted by $[MAR] \leftarrow [PC]$

Then the program counter gets incremented. $[PC] \leftarrow [PC] + 1$

The MAR holds the address of the memory location into which data is written (write cycle) or from which data is to be read (read cycle).

Note: data can refer to instructions, constants or variables.

During a read cycle the contents of the memory location specified by the MAR are read from memory and transferred into the memory buffer register (MBR some call the memory data register - MDR).

In RTL $[MBR] \leftarrow [M([MAR])]$

Next the instruction contained in the MBR is moved into the Instruction Register (IR) where it is divided into two fields **. They are

- the operation code (op-code) which tells the CPU the operation to be carried out
- the other is the operand field which contains the address of the data to be used by the instruction.

The control unit (CU) takes the op-code from the IR along with a stream of clock pulses and generates signals controlling all parts of the CPU. Typical time between clock pulses: 2ns – 0.8 ns. The clock rate is $1/(\text{time between clock pulses})$. Thus 2ns = 500Mhz

** here we made an assumption of one-address instructions.

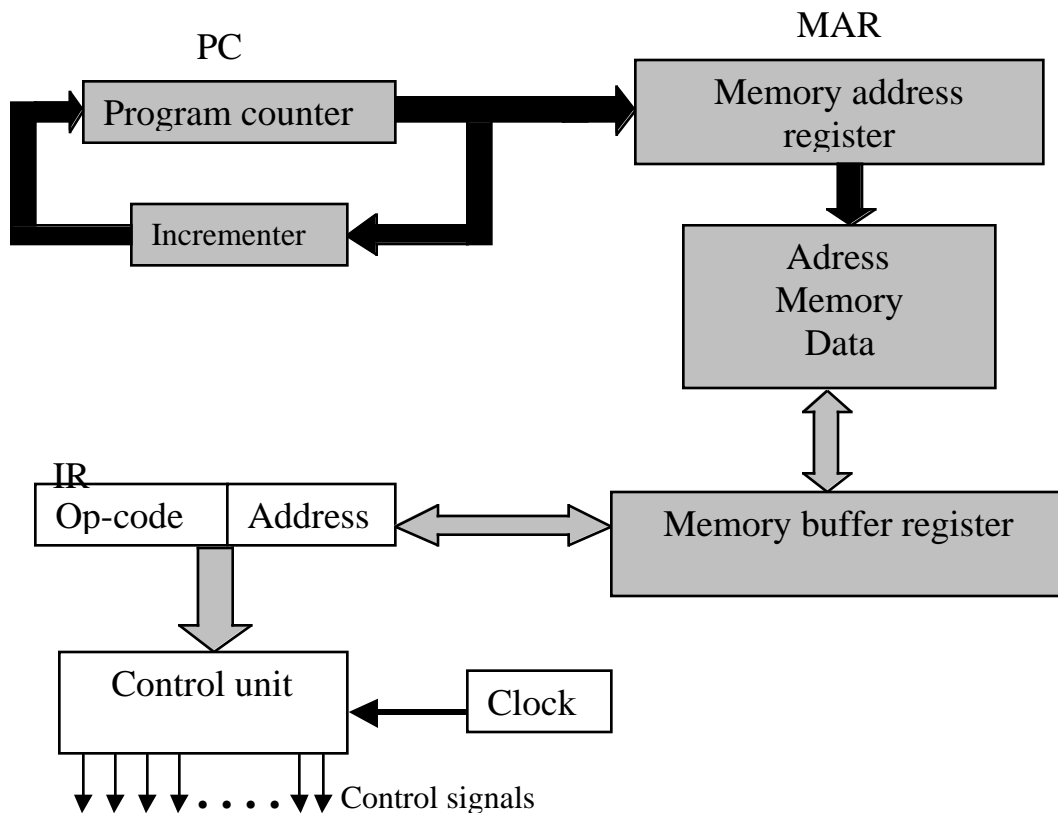
The control unit is responsible for moving the contents of the PC into the MAR, executing a read cycle and moving the contents of the MBR to the IR.

Instruction **fetch phase** – the sequence of operations to move the next instruction from memory to the IR.

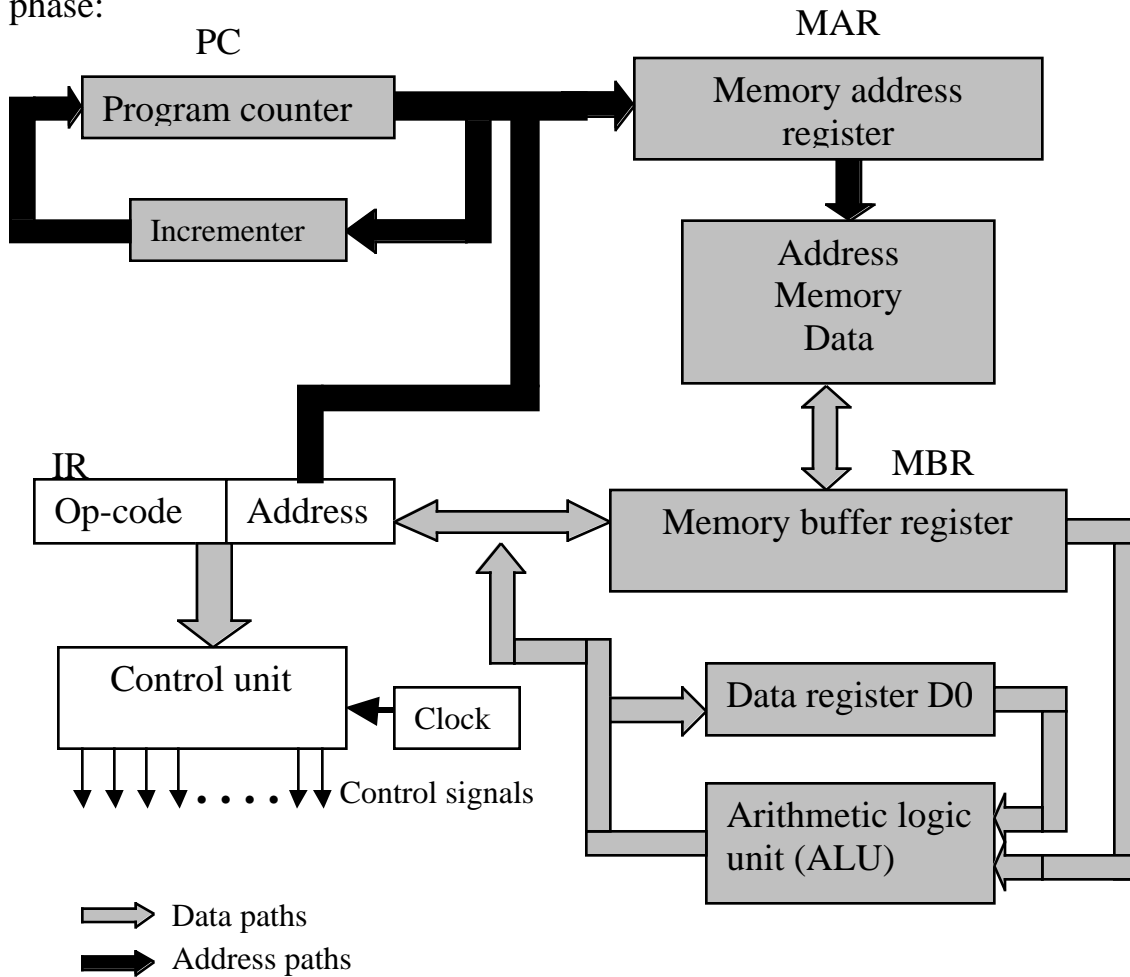
Execution takes place in a two-phase operation called the **fetch-execute cycle**.

We can summarize the discussion so far as:

Fetch $[MAR] \leftarrow [PC]$
 $[PC] \leftarrow [PC] + 1$
 $[MBR] \leftarrow [M([MAR])]$
 $[IR] \leftarrow [MBR]$
 $CU \leftarrow [IR(\text{op-code})]$



We now add some additional components that will aid in the execution phase:



The data register D0 holds temporary intermediate results during a calculation (it is not unusual to have multiple such registers). Operations such as +, *, -, / etc require two operands. One is supplied by the IR the other is supplied by D0. The result is placed in D0.

ADD x, D0

$[D0] \leftarrow [D0] + [M(x)]$

The Arithmetic and Logic Unit (ALU) performs all the arithmetic and logic calculations. The ALU operates on D0 or MBR alone or both D0 and MBR.

Typical ALU operations:

Operation	Class	Typical mnemonic
Addition	Arithmetic	ADD
Subtraction	Arithmetic	SUB
Negation	Arithmetic	NEG
Multiplication	Arithmetic	MULU
Division	Arithmetic	DIVU
Divide by 2	Arithmetic	ASR
Multiply by 2	Arithmetic	ASL
AND	Logical	AND
OR	Logical	OR
NOT	Logical	NOT
EOR	Logical	EOR
Shift Left	Logical	LSL
Shift Right	Logical	LSR

Lets consider what happens when we do a high level operation such as

$$P = Q + R$$

Here is what the assembly language program might look like:

MOVE	Q, D0	Loads D0 with the value stored at Q
ADD	R, D0	Add the contents of R to D0 and store in D0
MOVE	D0, P	Store the contents of D0 in P

Let us now describe how our one-address machine would carry out the
ADD R D0 statement.

FETCH	[MAR] ← [PC]	
	[PC] ← [PC] + 1	
	[MBR] ← [M([MAR])]	
	[IR] ← [MBR]	
	CU ← [IR(op-code)]	Signals ALU to do arithmetic
ADD	[MAR] ← [IR(address)]	To get operand
	[MBR] ← [M([MAR])]	
	ALU ← [MBR]; ALU ← [D0]	
	[D0] ← ALU	