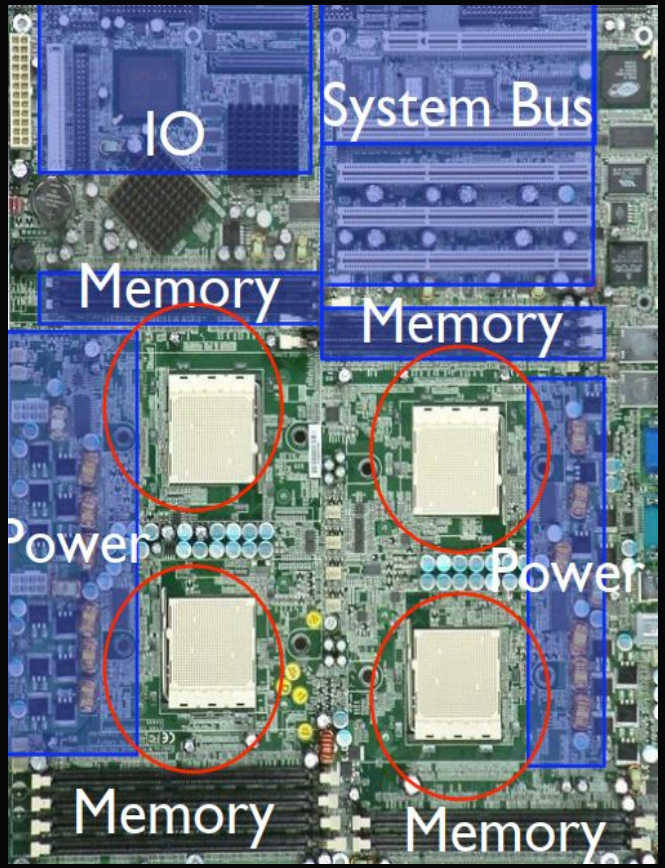
Computer Architecture (CSE-2207)

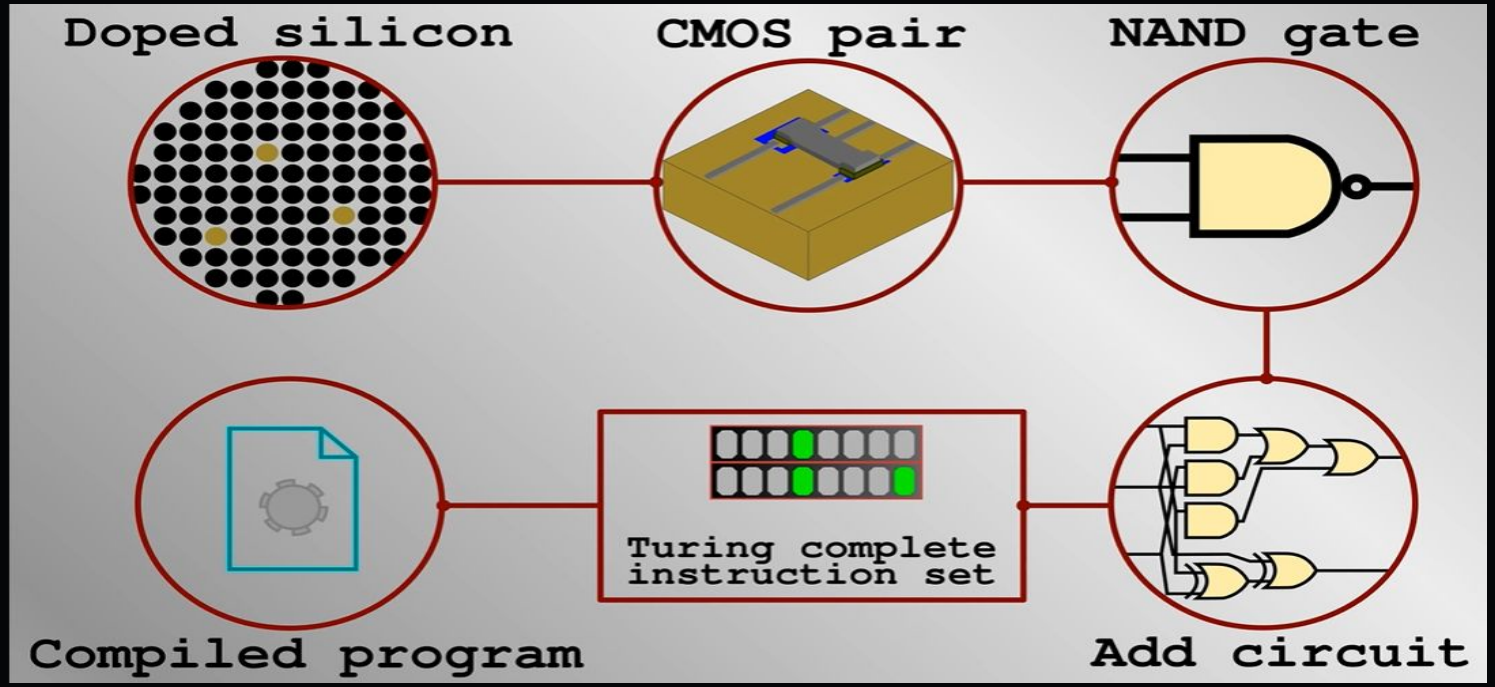
Top level view of computer Function and CPU Architecture

Agnik Saha
Department of Computer Science and Engineering
R. P. Shaha University

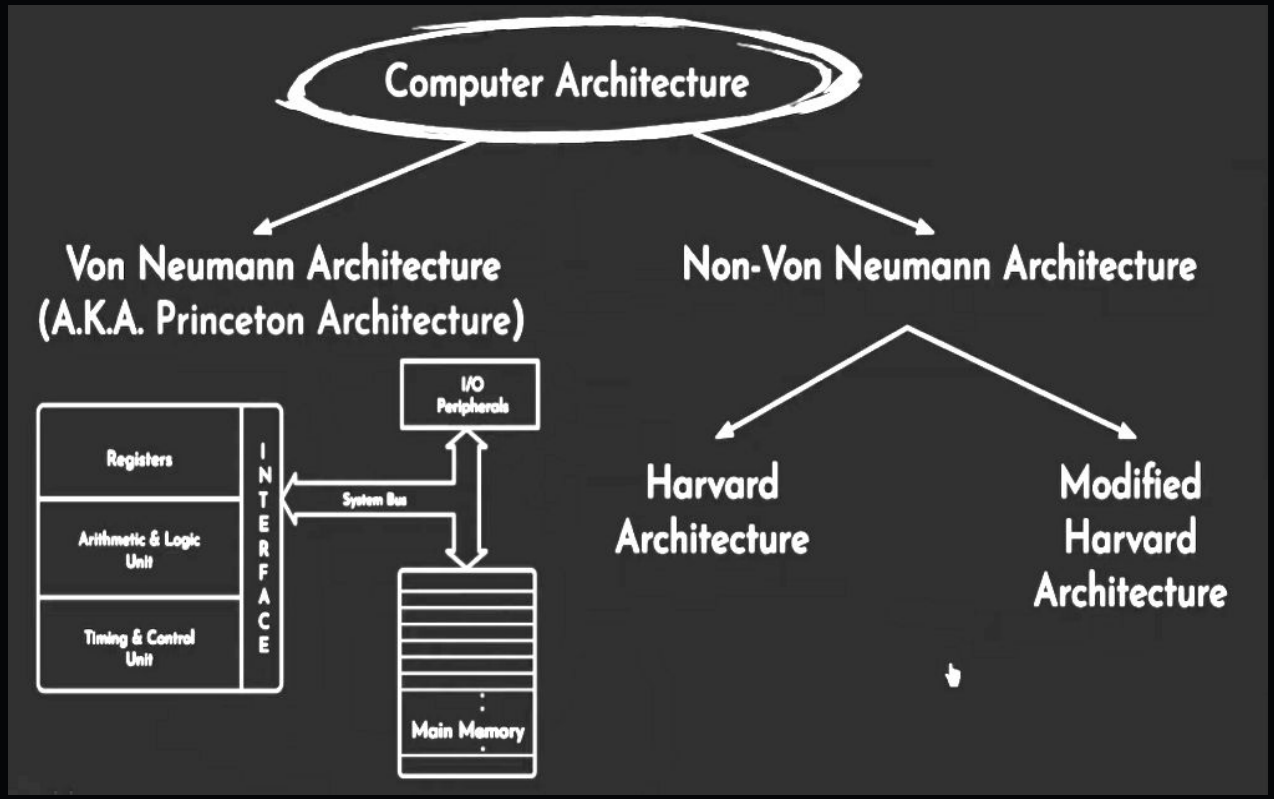
Ready...



Computer Map



Models of Computer Architecture



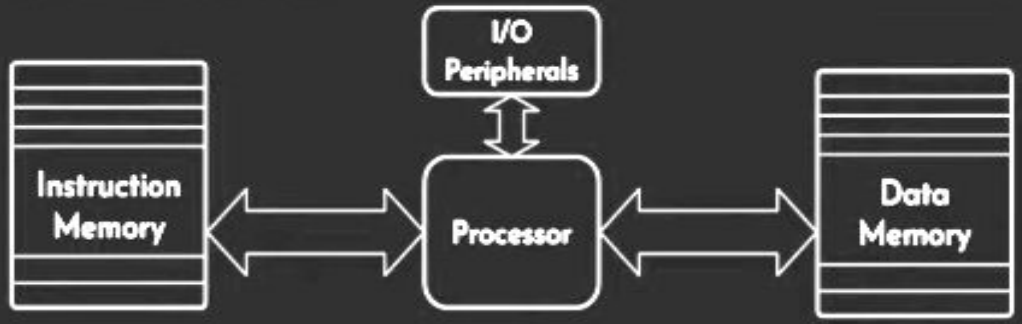
Software and IO components

- A sequence of codes or instructions
- Part of the hardware interprets each instruction and generates control signals
- Provide a new sequence of codes for each new program instead of rewiring the hardware

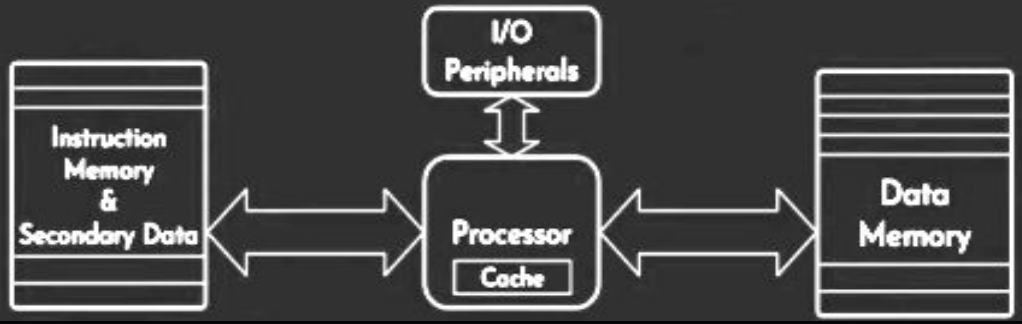
- CPU
  - Instruction interpreter
  - Module of general-purpose arithmetic and logic functions
- I/O Components
  - Input module
    - Contains basic components for accepting data and instructions and converting them into an internal form of signals usable by the system
  - Output module
    - Means of reporting results

Models of Computer Architecture

- **Harvard Architecture :**



- **Modified Harvard Architecture :**



Control Unit

Control unit CU is the part of CPU that helps orchestrate the execution of instructions. It tells what to do. According to the instruction, it helps activate the wires connecting CPU to different other parts of computer including the ALU. Control unit is the first component of CPU to receive the instruction for processing.

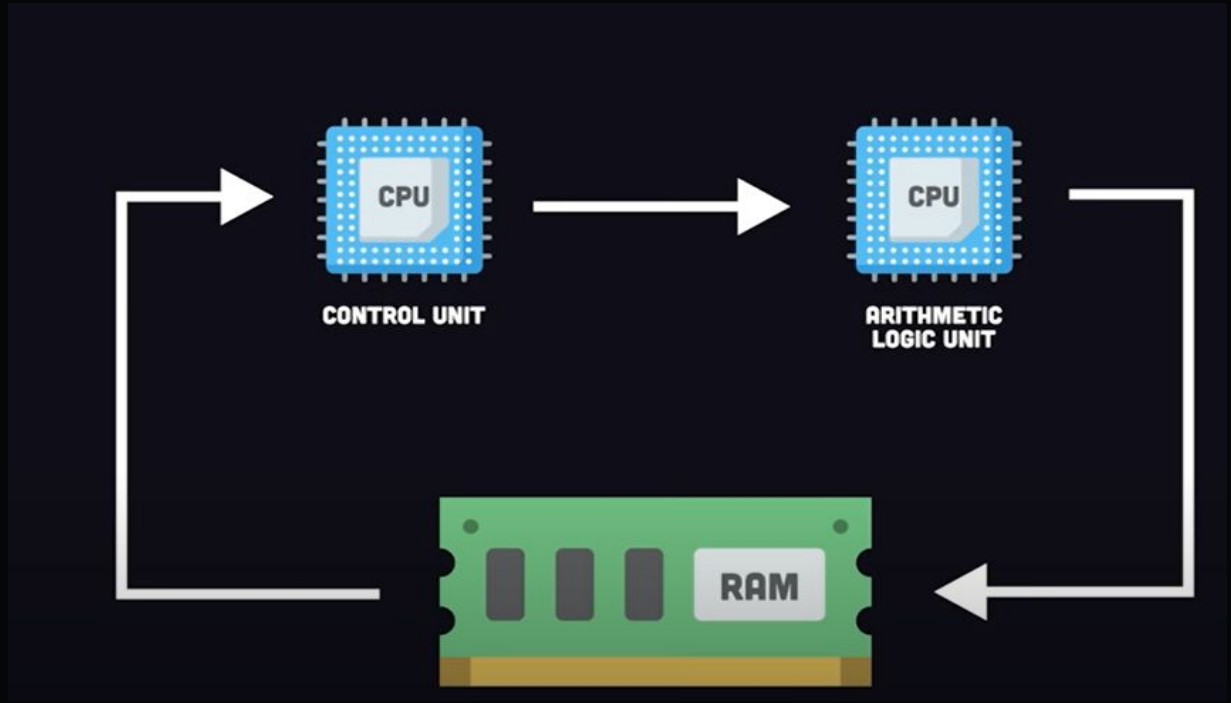
There are two types of control unit:

- ➔ hardwired control units.
- ➔ microprogrammable (microprogrammed) control units.

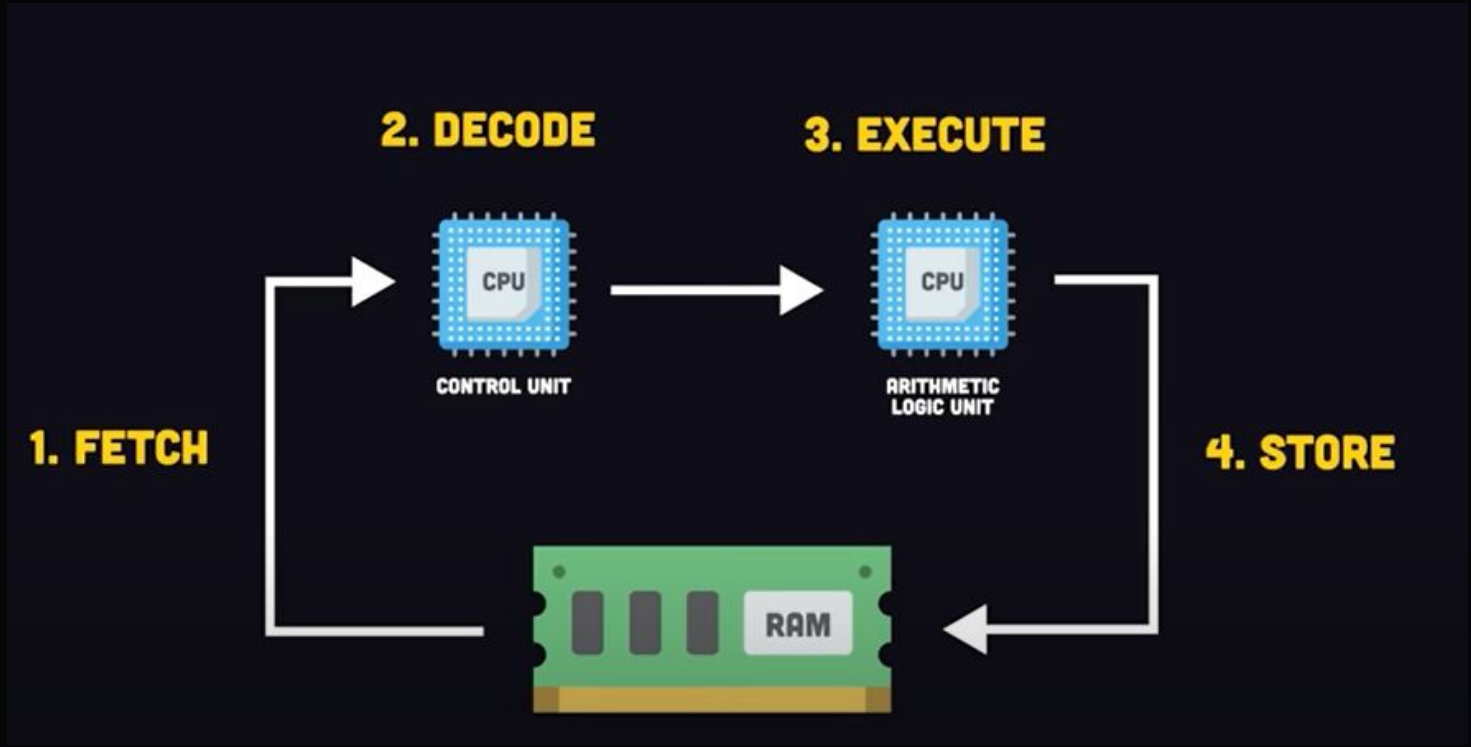
ALU

ALU consists of logic circuitry or logic gates which performs these operations.

Instruction Cycle



Instruction Cycle



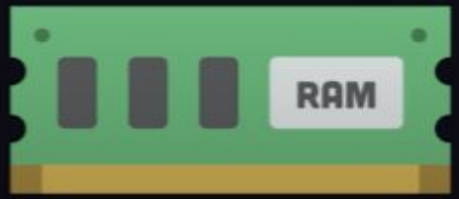
Instruction Cycle

INSTRUCTION REGISTER

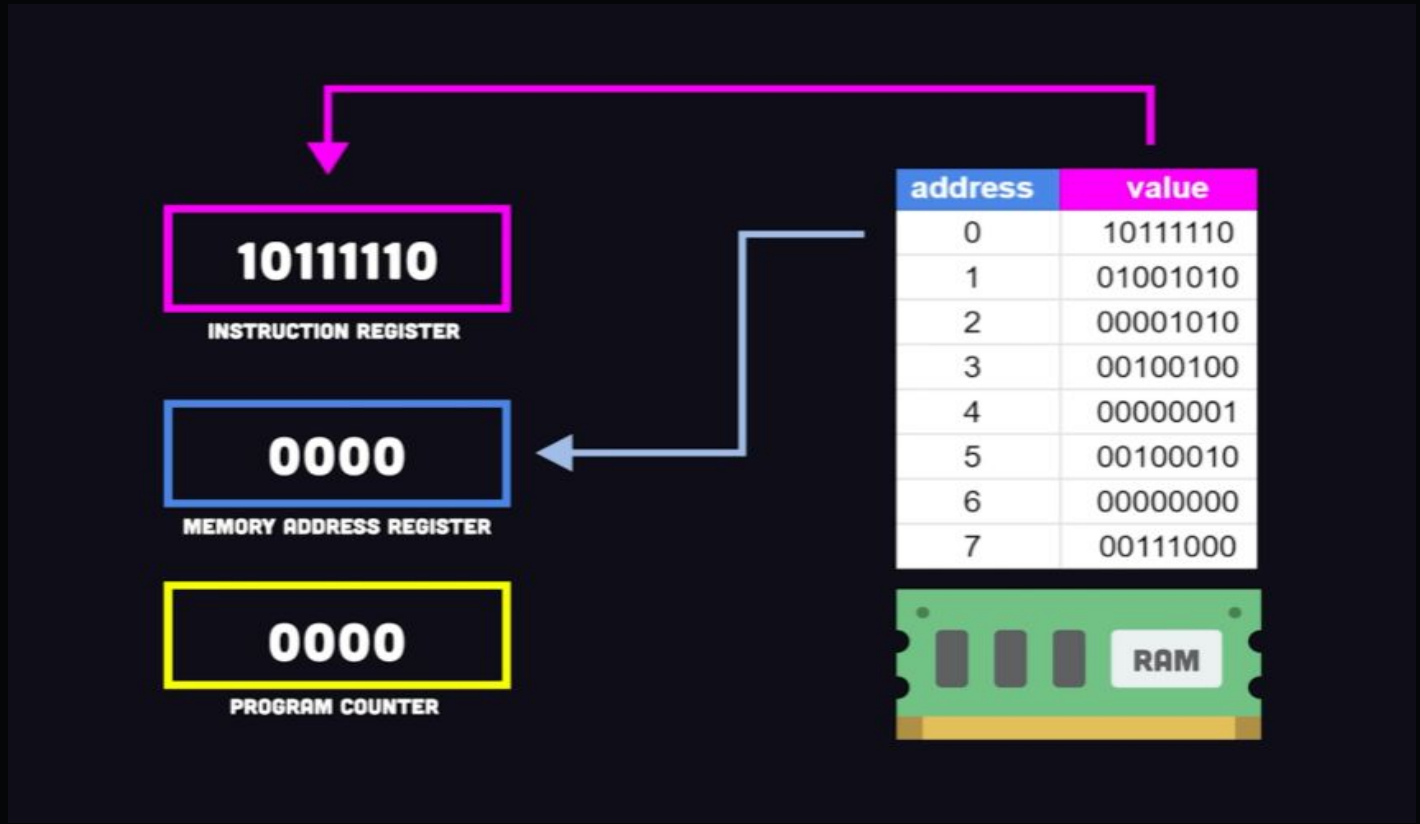
MEMORY ADDRESS REGISTER

PROGRAM COUNTER

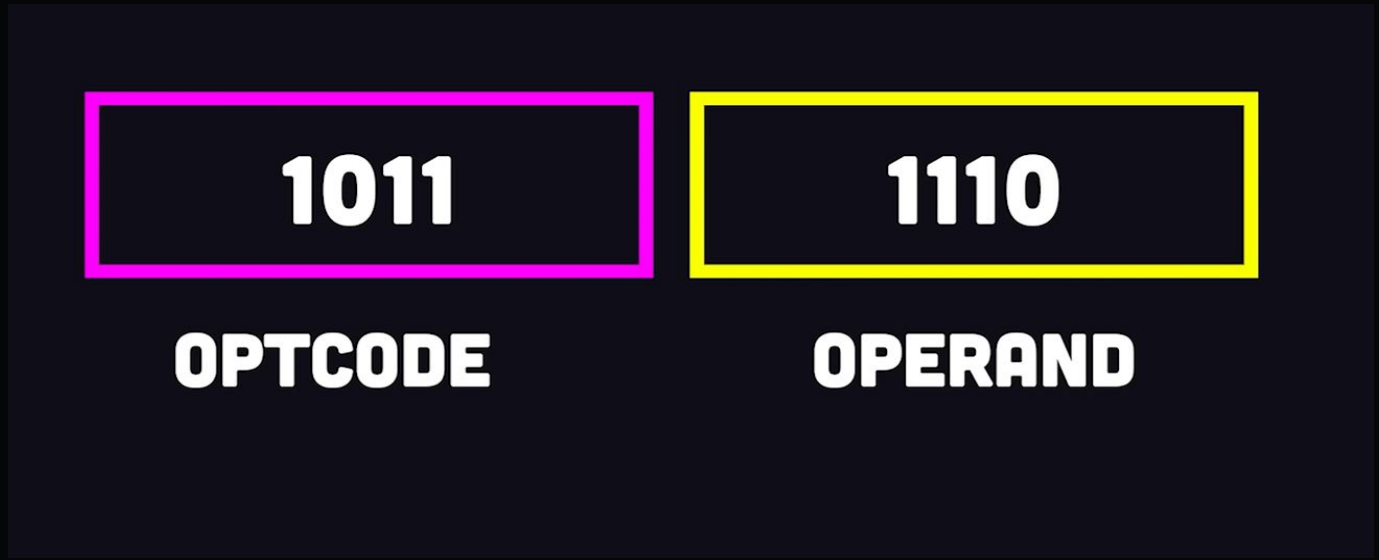
address	value
0	10111110
1	01001010
2	00001010
3	00100100
4	00000001
5	00100010
6	00000000
7	00111000



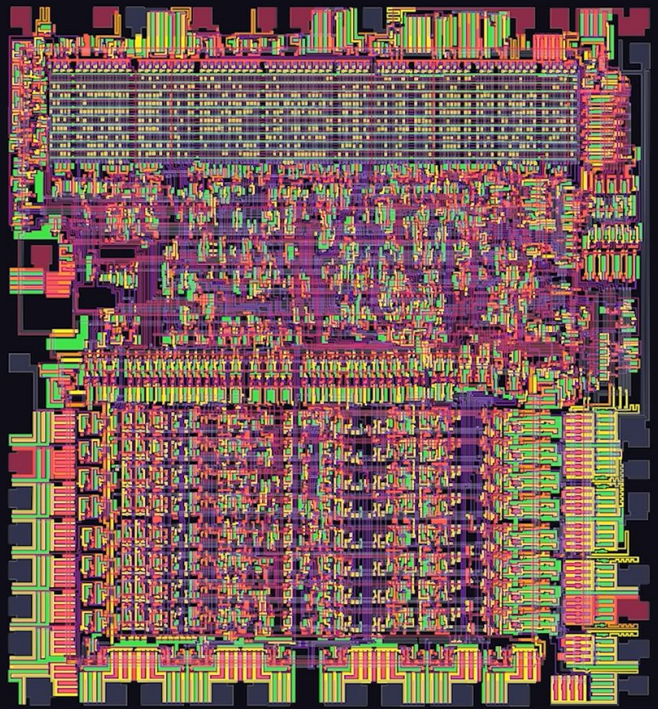
Fetch

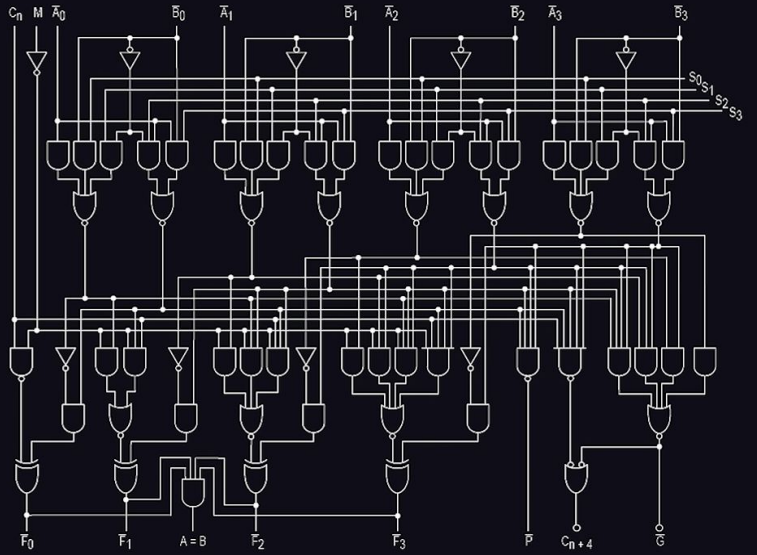

10111110
INSTRUCTION REGISTER
0000
MEMORY ADDRESS REGISTER
0000
PROGRAM COUNTER
address
value
0
10111110
1
01001010
2
00001010
3
00100100
4
00000001
5
00100010
6
00000000
7
00111000
RAM

Decode

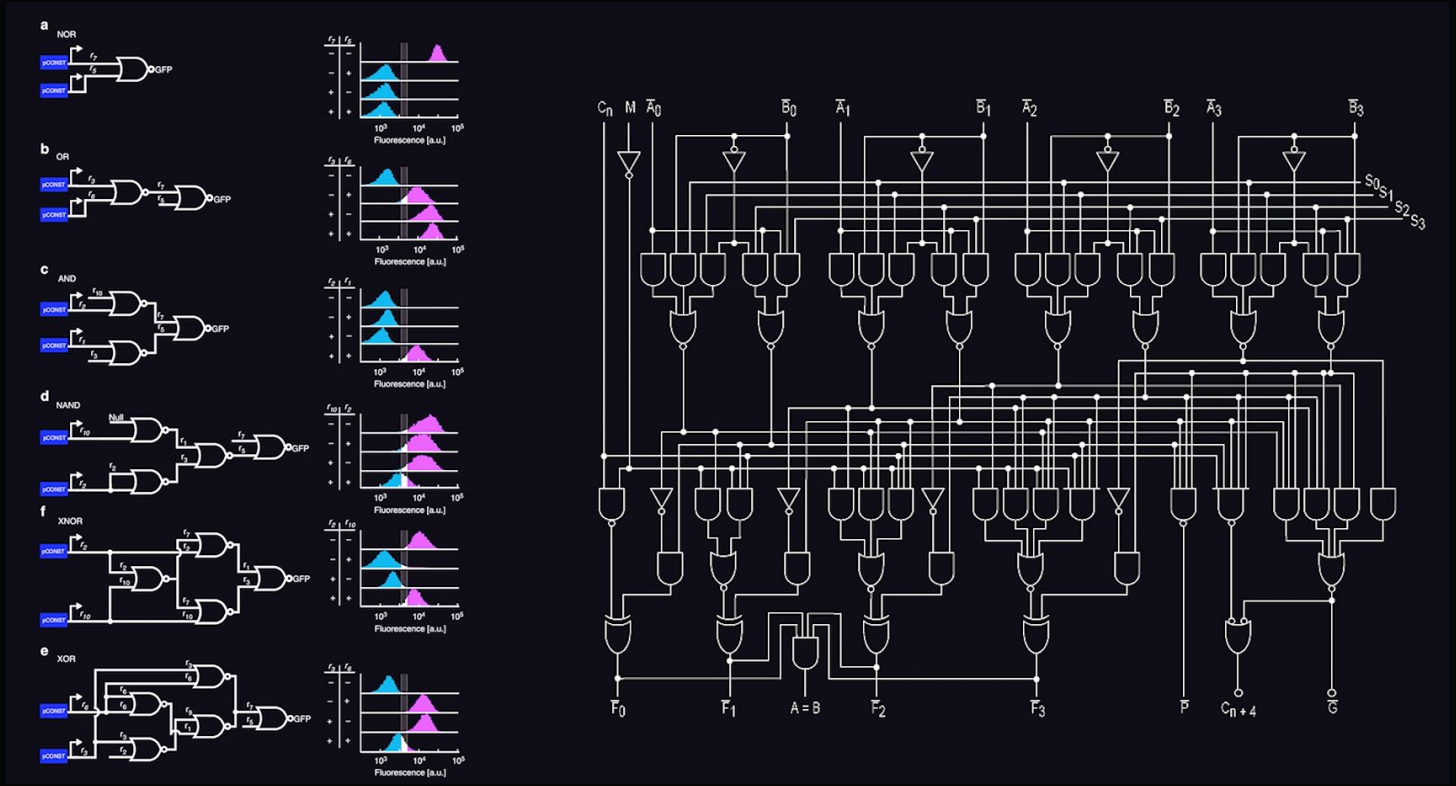


Execution

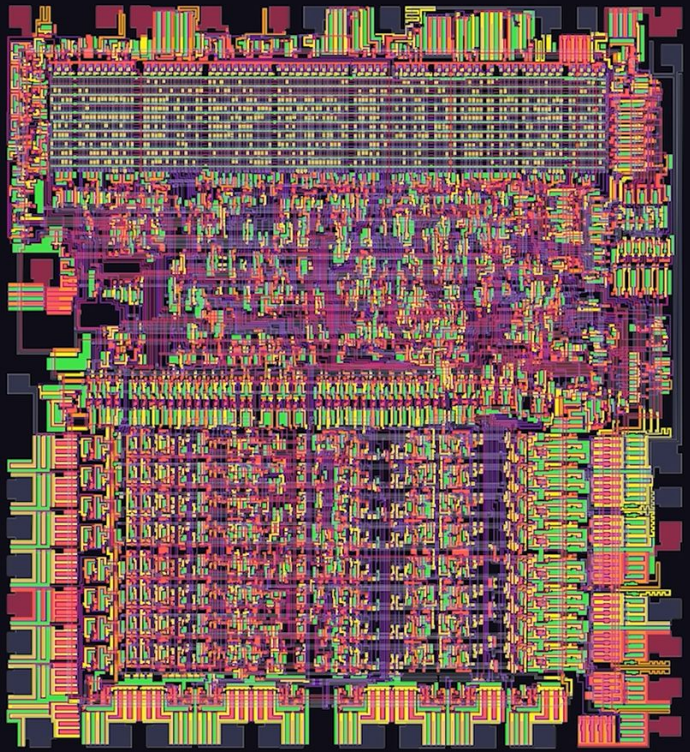




Execution


a NOR
b OR
c AND
d NAND
f XNOR
e XOR
pCONST
GFP
Null
Fluorescence [a.u.]
10^3 10^4 10^5
C_n M $\overline{A}_0$ $\overline{B}_0$ $\overline{A}_1$ $\overline{B}_1$ $\overline{A}_2$ $\overline{B}_2$ $\overline{A}_3$ $\overline{B}_3$
S_0 S_1 S_2 S_3
$\overline{F}_0$ $\overline{F}_1$ A = B $\overline{F}_2$ $\overline{F}_3$ $\overline{P}$ C_{n+4} $\overline{G}$

Clock Cycle

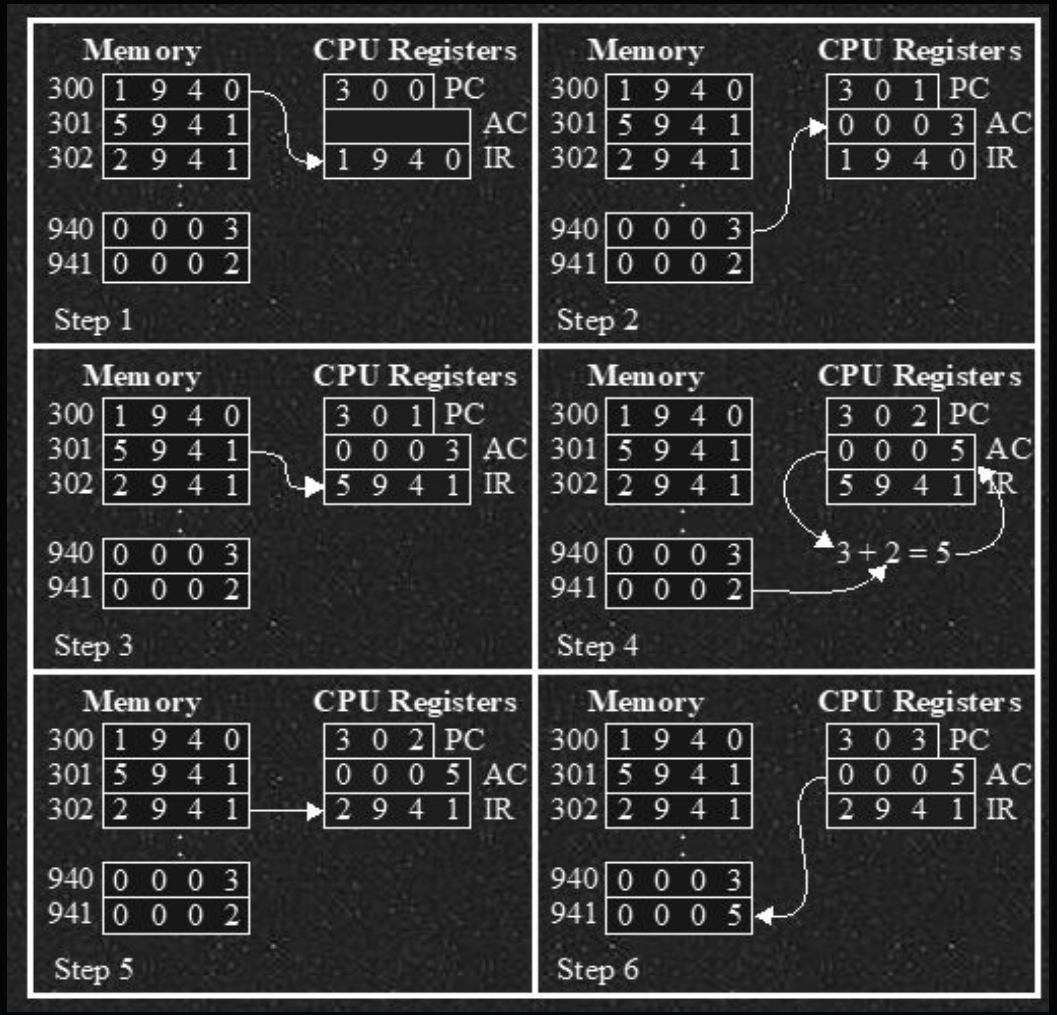


CLOCK GENERATOR

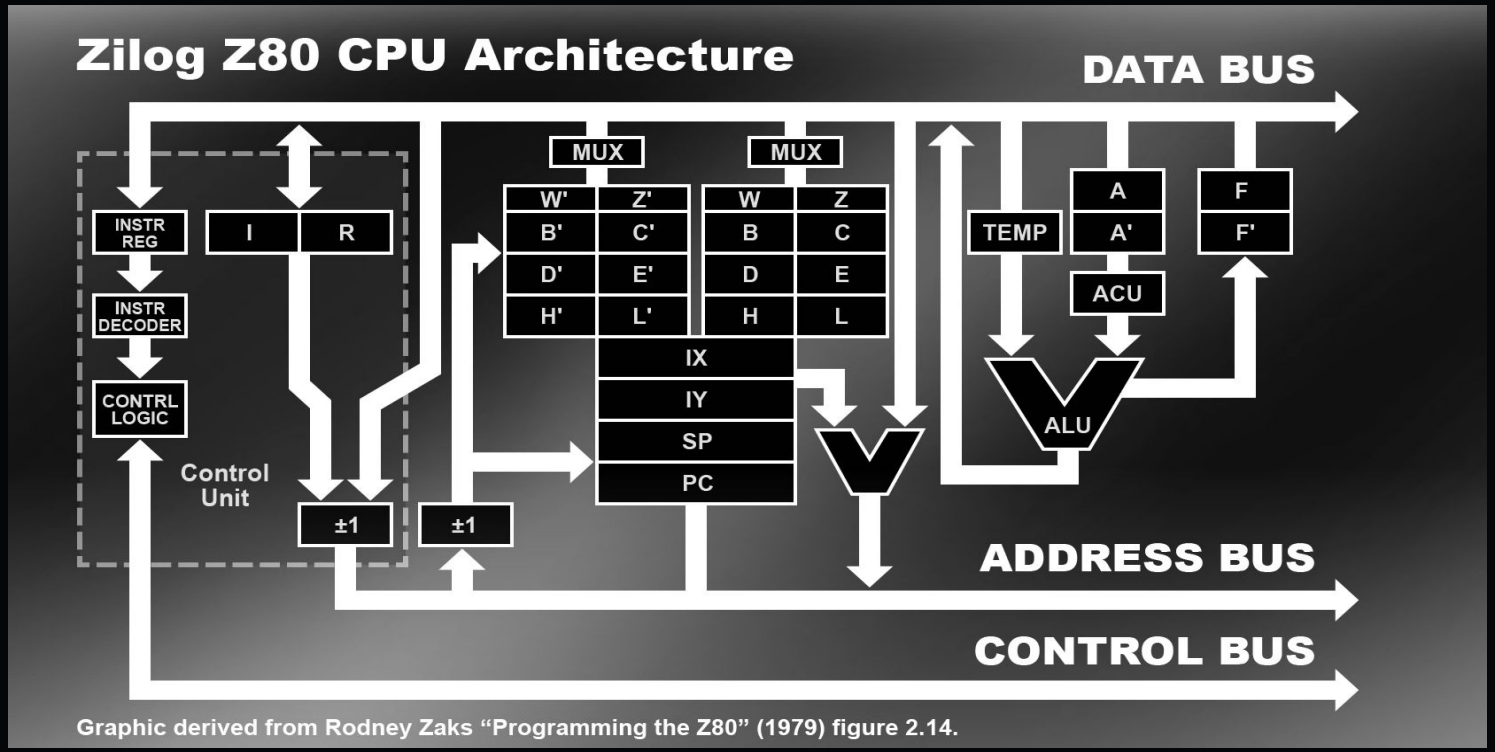
Clock Cycle

- 1 Gigahertz means 10^9 cycles per second.
- The faster the clock cycle, the more instructions the CPU can execute.Clock cycle = 1/clock rateCPU Time = number of clock cycle / clock rate
- This means to improve CPU time we can increase clock rate or decrease number of clock cycle by optimizing the instruction we provide to CPU.

Example of Program Execution



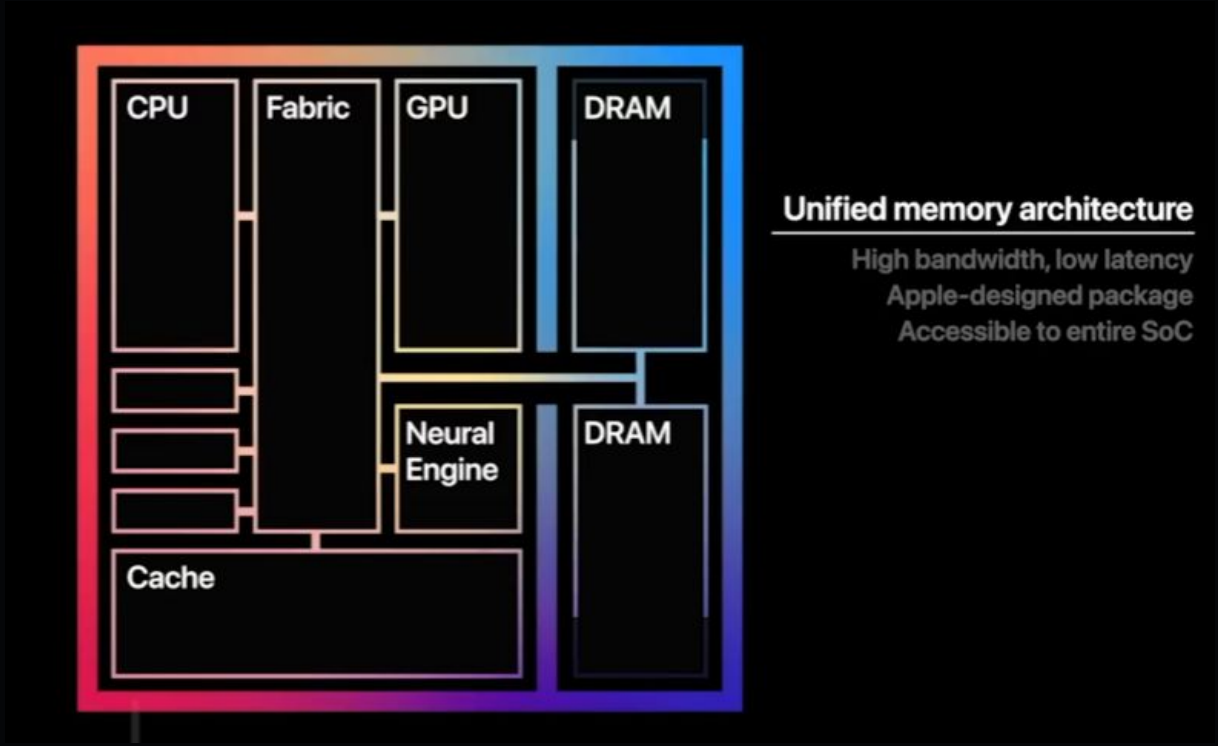
CPU Architecture



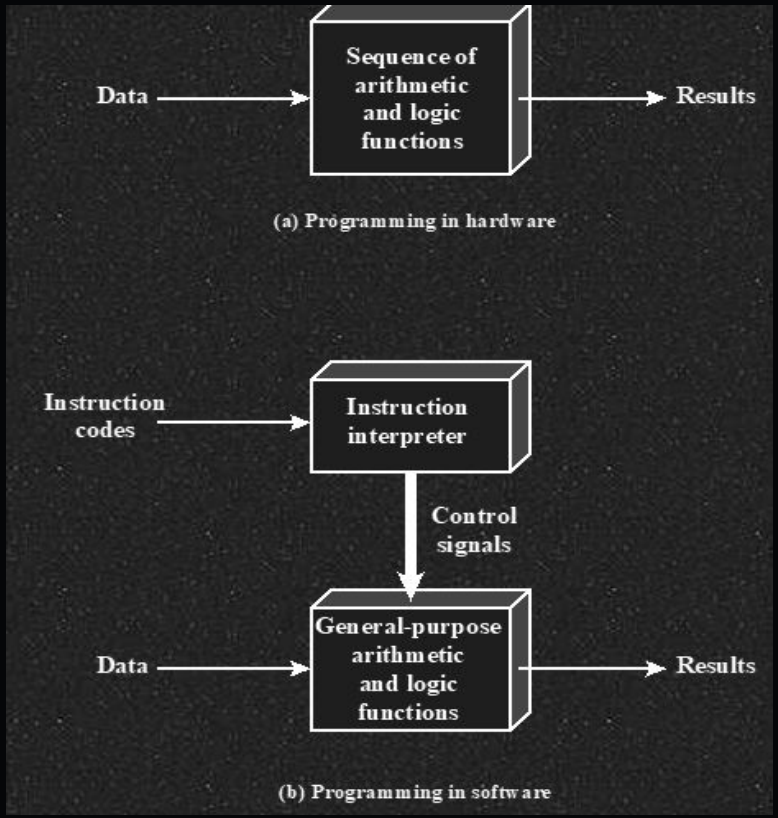
Graphic derived from Rodney Zaks “Programming the Z80” (1979) figure 2.14.

Modern Processors

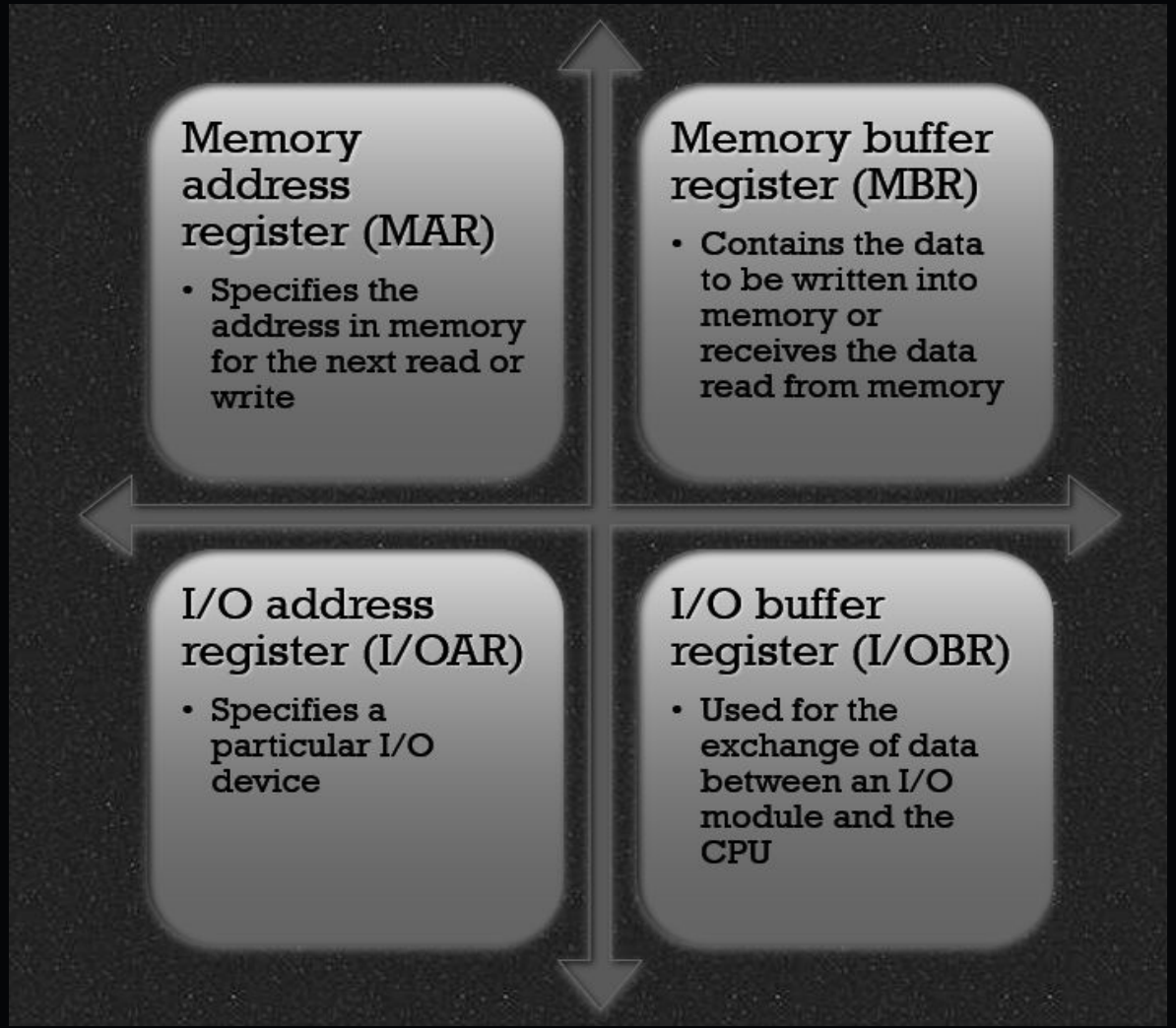
Apple M1 Chip



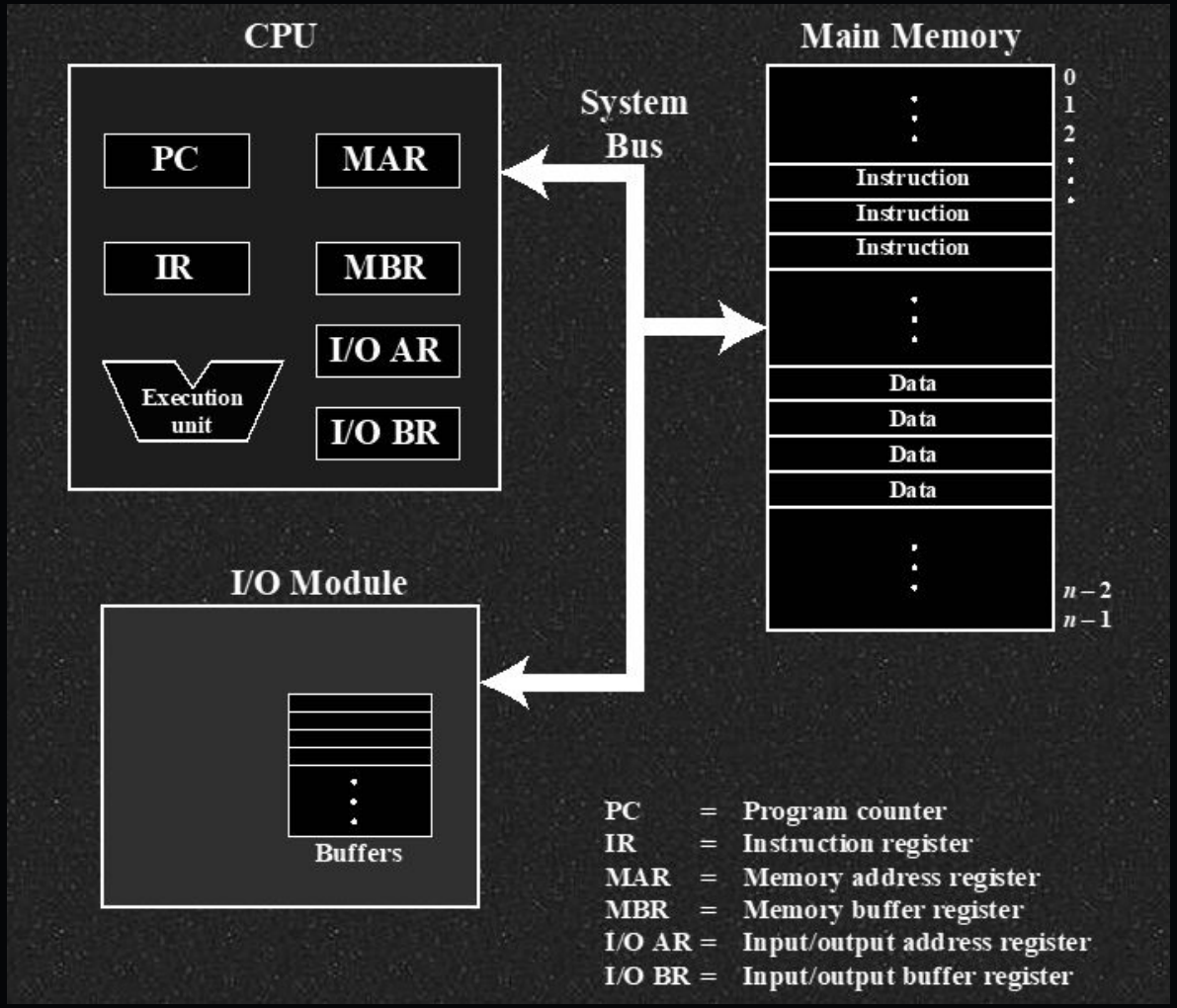
Hardware and Software Approaches



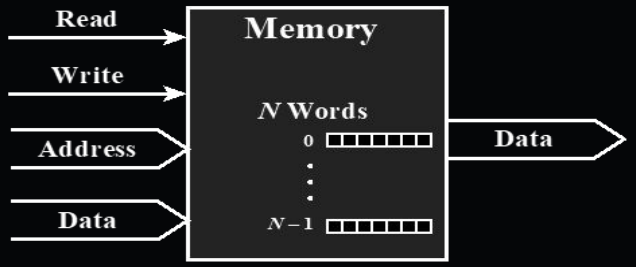
MAR & MBR

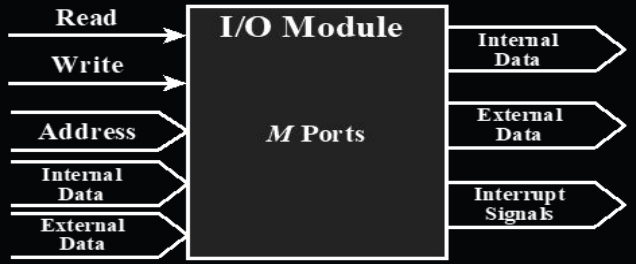

Memory address register (MAR)
• Specifies the address in memory for the next read or write
Memory buffer register (MBR)
• Contains the data to be written into memory or receives the data read from memory
I/O address register (I/OAR)
• Specifies a particular I/O device
I/O buffer register (I/OBR)
• Used for the exchange of data between an I/O module and the CPU

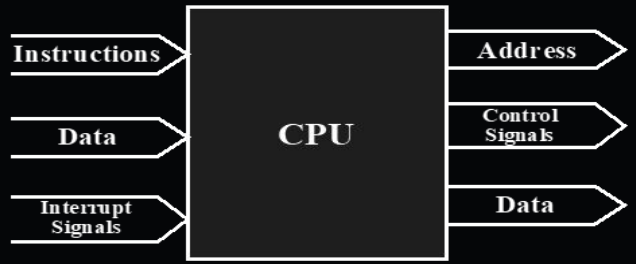
Computer Components



Computer Modules







Data Bus

- ➔ Data lines that provide a path for moving data among system modules
- ➔ May consist of 32, 64, 128, or more separate lines
- ➔ The number of lines is referred to as the width of the data bus
- ➔ The number of lines determines how many bits can be transferred at a time
- ➔ The width of the data bus is a key factor in determining overall system performance

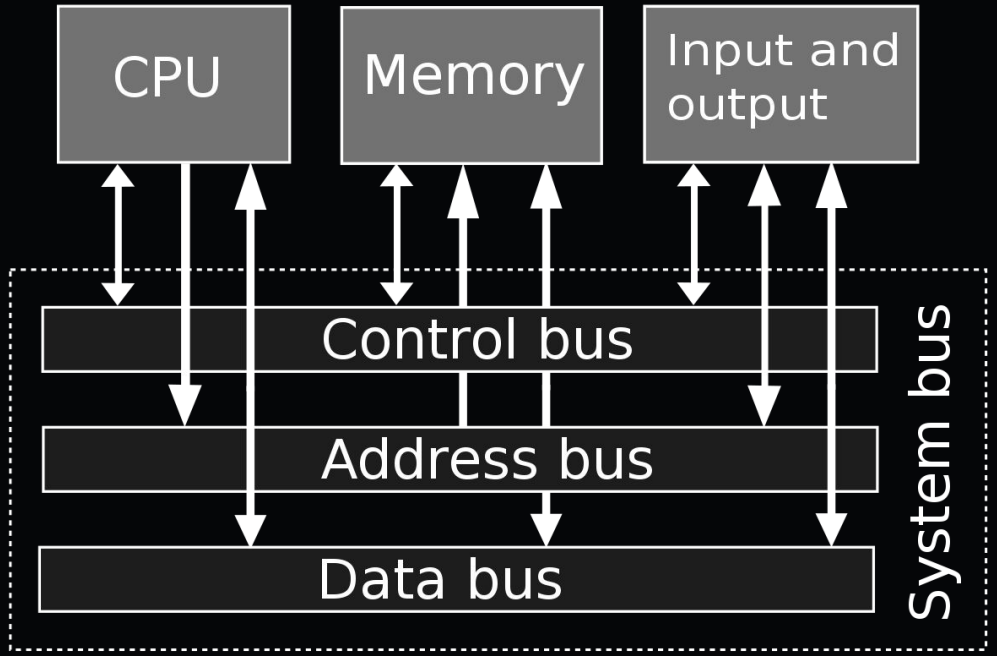
Address Bus

- Used to designate the source or destination of the data on the data bus
- If the processor wishes to read a word of data from memory it puts the address of the desired word on the address lines
- Width determines the maximum possible memory capacity of the system
- Also used to address I/O ports
- The higher order bits are used to select a particular module on the bus and the lower order bits select a memory location or I/O port within the module

Control Bus

- Used to control the access and the use of the data and address lines
- Because the data and address lines are shared by all components there must be a means of controlling their use
- Control signals transmit both command and timing information among system modules
- Timing signals indicate the validity of data and address information
- Command signals specify operations to be performed

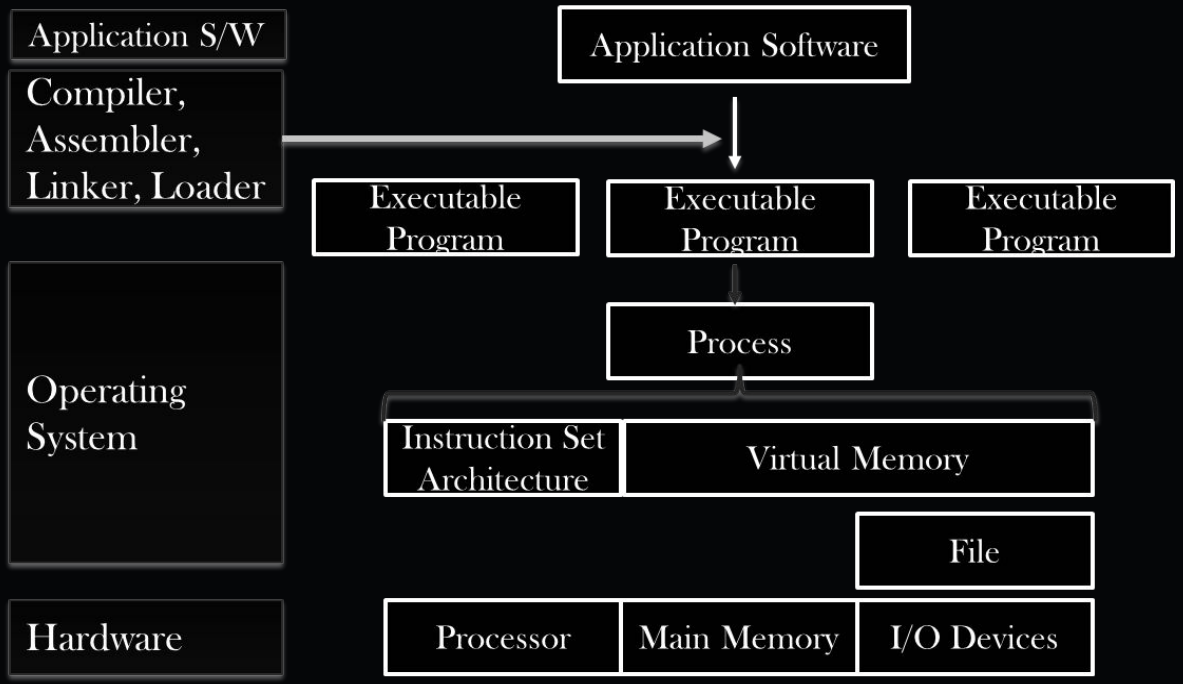
Bus Interconnection Scheme



Control Signals

- Memory Read
- Memory Write
- I/0 Read
- I/0 Write
- Bus Request
- Bus Grant
- Interrupt Request
- Interrupt Acknowledge
- Transfer Acknowledge
- Clock
- Reset

Simplified overall abstraction



Types of Computer Systems based on number of General Purpose Processors

1. Single Processor Systems
2. Multiprocessor Systems
3. Clustered Systems

Single Processor Systems

- One main CPU capable of executing a general purpose instruction set including instructions from user processes.
- Other special purpose processors are also present which perform device specific tasks

MultiProcessor Systems

- Also known as parallel systems or tightly coupled systems.
- Has two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices
- Advantages:
  - Increased throughput
  - Economy of scale
  - Increased reliability

Clustered Systems

- Like multiprocessor systems, clustered systems gather together multiple CPUs to accomplish computational work.
- They are composed of two or more individual systems coupled together.
- Provides high availability
- Can be structured asymmetrically or symmetrically

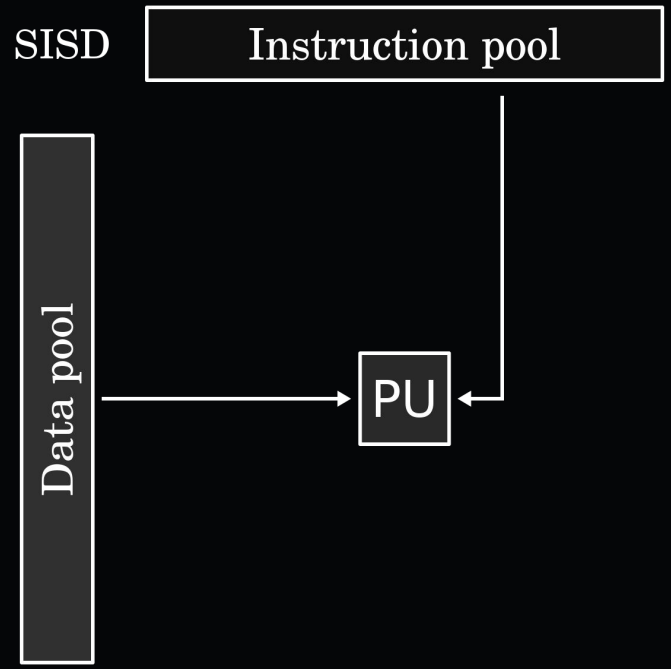
Flynn’s Classification

Categories of Computer systems

- ➔ SISD — Single Instruction Single Data stream
- ➔ SIMD — Single Instruction Multiple Data stream
- ➔ MISD - Multiple Instruction Single Data stream
- ➔ MIMD— Multiple Instruction Multiple Data stream

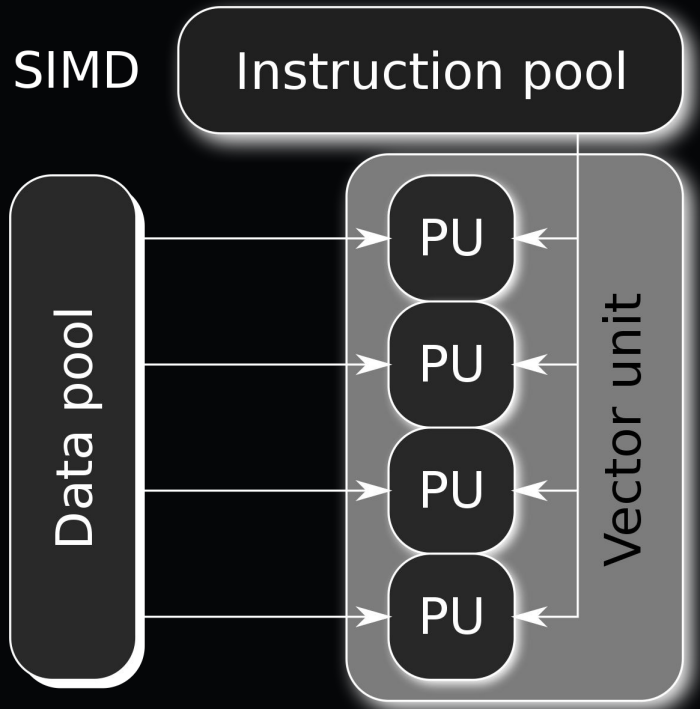
SISD

- In SISD, one instruction runs on one processor on the data stored on single memory.
- Uniprocessor comes in this category.
- Instruction And Data must be stored in primary memory
- Clock defines the speed of processing in SISD



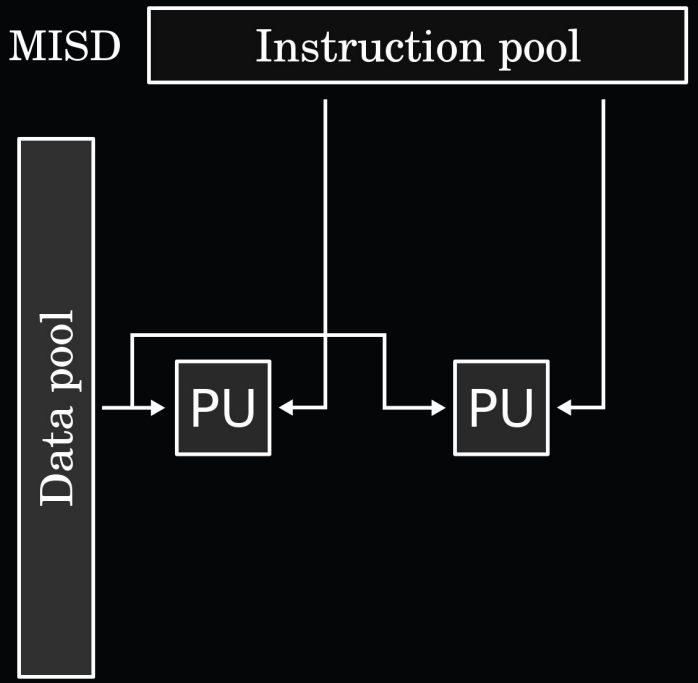
SIMD

- In SIMD, one instruction controls multiple simultaneous execution of number of processing elements on lock step basis.
- Each Processing unit has associate data memory for execution of instruction.



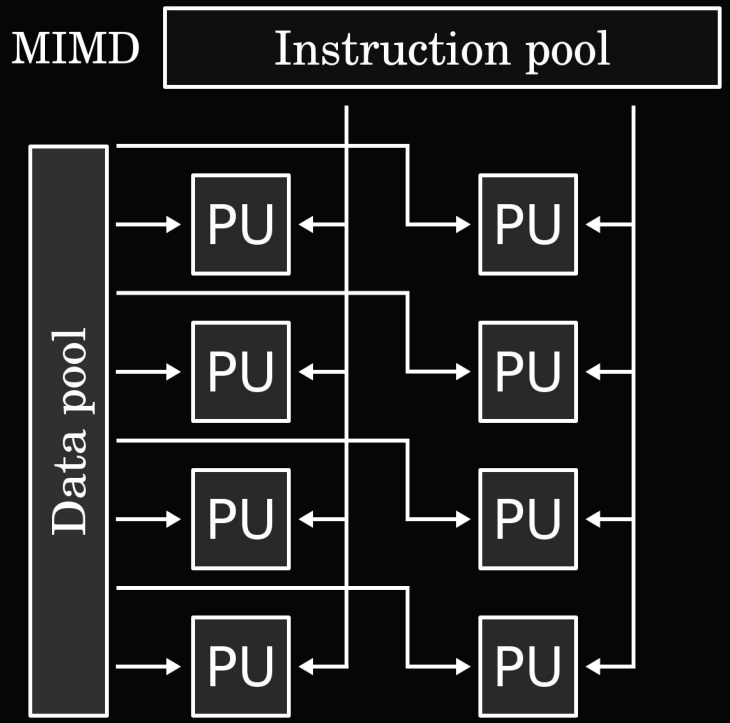
MISD

- In MISD, Here multiple processor executes different instructions with single data stream.

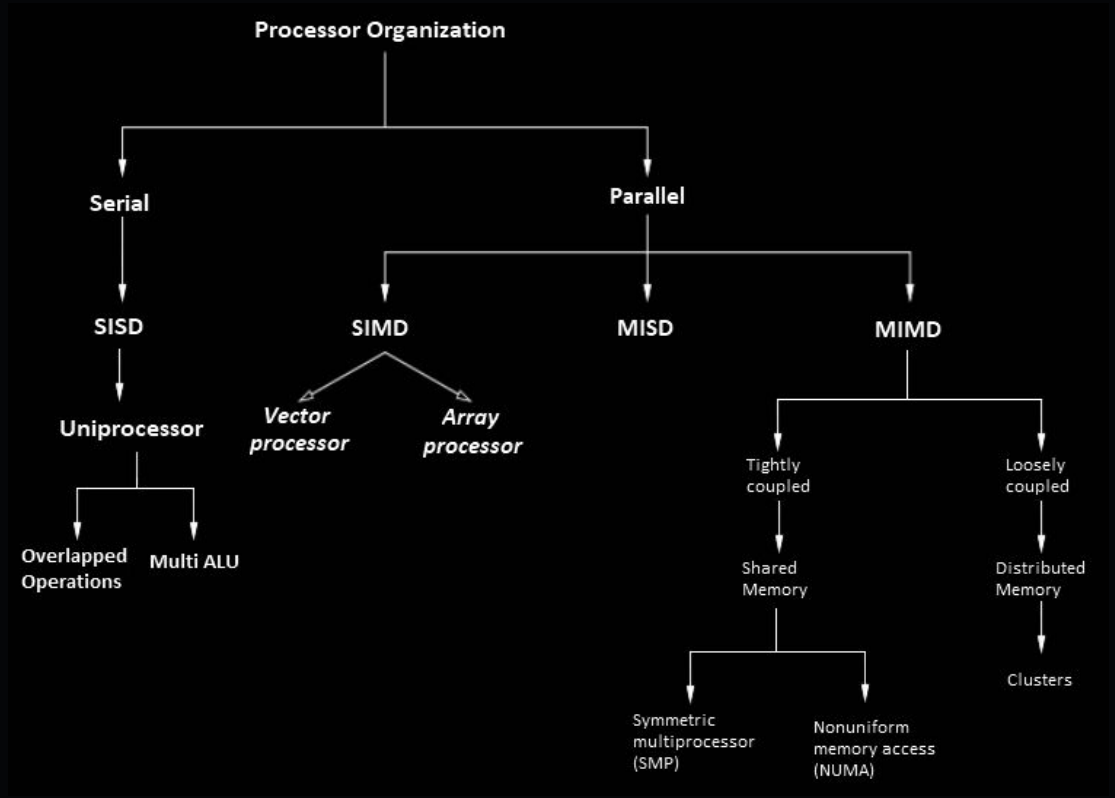


MIMD

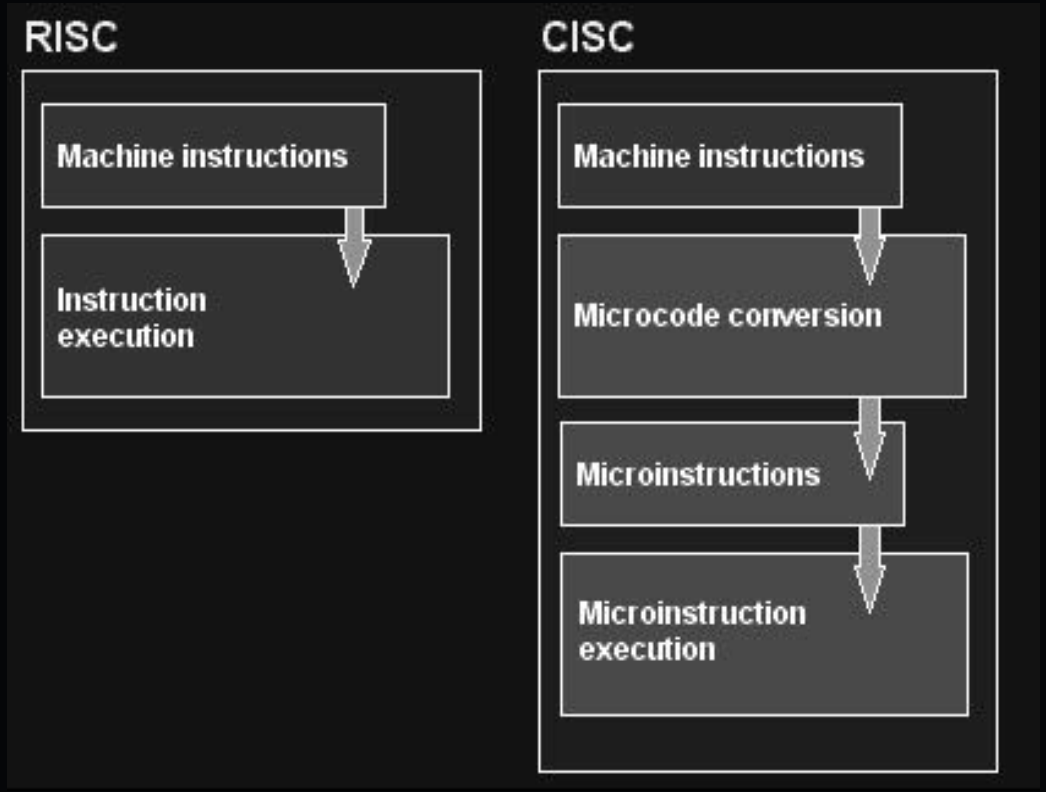
➔ In MIMD, Here multiple processor executes different instructions with multiple data stream.



Taxonomy of mono- mulitprocessor organizations



RISC VS CISC


RISC
Machine instructions
Instruction execution
CISC
Machine instructions
Microcode conversion
Microinstructions
Microinstruction execution

Amdahl's Law

$$S_p \leq \frac{1}{(1-f)+\frac{f}{p}}$$

Amdahl’s Law

1. Assume a microprocessor given which is widely used for scientific applications. It has both integer and floating point instructions. Now floating point instructions are enhanced and made 3 times faster than before and integer instructions are unenhanced. If there are 20% of floating point instructions in the program then find the overall speed up.

2. Suppose you have a machine used in an I/O intensive environment; the CPU is working 75% of the time and the rest is waiting for I/O operations to complete. You may consider an improvement of the CPU by a factor of 2 (it will run twice as fast as it runs now) for a fivefold increase in cost. The present cost of the CPU is 20% of the machine's cost. Is the suggested improvement cost effective?

Performance

- What is the metric of the performance of a computing system?
- Depends on the purpose
- Two commonly used metrics are:
  - Execution or Response Time
    - How long it takes to do a task
  - Throughput
    - Total work done per unit time
      - e.g., tasks/transactions/… per hour
- Are these two same?

Example

- Do the following changes to a computer system increase throughput, decrease response time, or both?
  - Replacing the processor in a computer with a faster version
    - Response time decreases or improves
    - Decreasing response time generally increases throughput
- Adding additional processors to a system that uses multiple processors for separate tasks—for example, searching the web
  - Throughput increases
  - Response time depends on scenario
    - Generally no impact on response time
    - But in case tasks were waiting in the queue previously, response time will decrease after the change

Comparing Performance

- We shall focus on Response or Execution time

$$Performance = \frac{1}{Execution\ Time}$$

- "X is n times faster than Y" means

$$\frac{Performance_X}{Performance_Y} = \frac{Execution\ Time_Y}{Execution\ Time_X} = n$$

- Example: Time taken to run a program
  - A: 10s, B=15s
  - $\frac{Performance_A}{Performance_B} = \frac{Execution\ Time_B}{Execution\ Time_A} = \frac{15s}{10s} = 1.5$
  - So, A is 1.5 times faster than B

Continue

Suppose we have to compare two machines A and B. The phrase *A is n% faster than* B means:

$$\frac{\text{Execution time of B}}{\text{Execution time of A}} = 1 + \frac{n}{100}$$

Because performance is reciprocal to execution time, the above formula can be written as:

$$\frac{\text{Performance A}}{\text{Performance B}} = 1 + \frac{n}{100}$$

Continue

If machine A runs a program in 5 seconds, and machine B runs the same program in 6 seconds, how can the execution times be compared?

Practice Problems

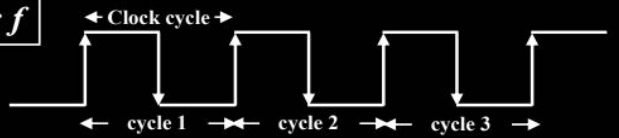
1. In hypothetical system, Floating point instructions are improved to run five times as fast, but only 40% of the time was spent on these instructions originally. How much speed up is achieved by new machine?

Cycles Per Instruction (CPI)

- **Most computers run synchronously utilizing a CPU clock running at a constant clock rate:** *Or clock frequency: f*

  where: Clock rate = 1 / clock cycle

  $f = 1/C$



- **The CPU clock rate depends on the specific CPU organization (design) and hardware implementation technology (VLSI) used.**
- **A computer machine (ISA) instruction is comprised of a number of elementary or micro operations which vary in number and complexity depending on the the instruction and the exact CPU organization (Design).**
  - **A micro operation is an elementary hardware operation that can be performed during one CPU clock cycle.**
  - **This corresponds to one micro-instruction in microprogrammed CPUs.**
  - **Examples: register operations: shift, load, clear, increment, ALU operations: add , subtract, etc.**
- **Thus: A single machine instruction may take one or more CPU cycles to complete termed as the Cycles Per Instruction (CPI).**

  Instructions Per Cycle = IPC = 1/CPI
- **Average (or effective) CPI of a program: The average CPI of all instructions executed in the program on a given CPU design.**

Clock Cycles

- Time is not continuous, rather discrete to a Computer's perspective
- Activities are performed during the discrete clock ticks

- cycle time = time between ticks = seconds per cycle
- clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)

  A 2 Ghz. clock has a $1 / 2 \times 10^9$ = 0.5 nano-second (*ns*) cycle time
- So, for a program

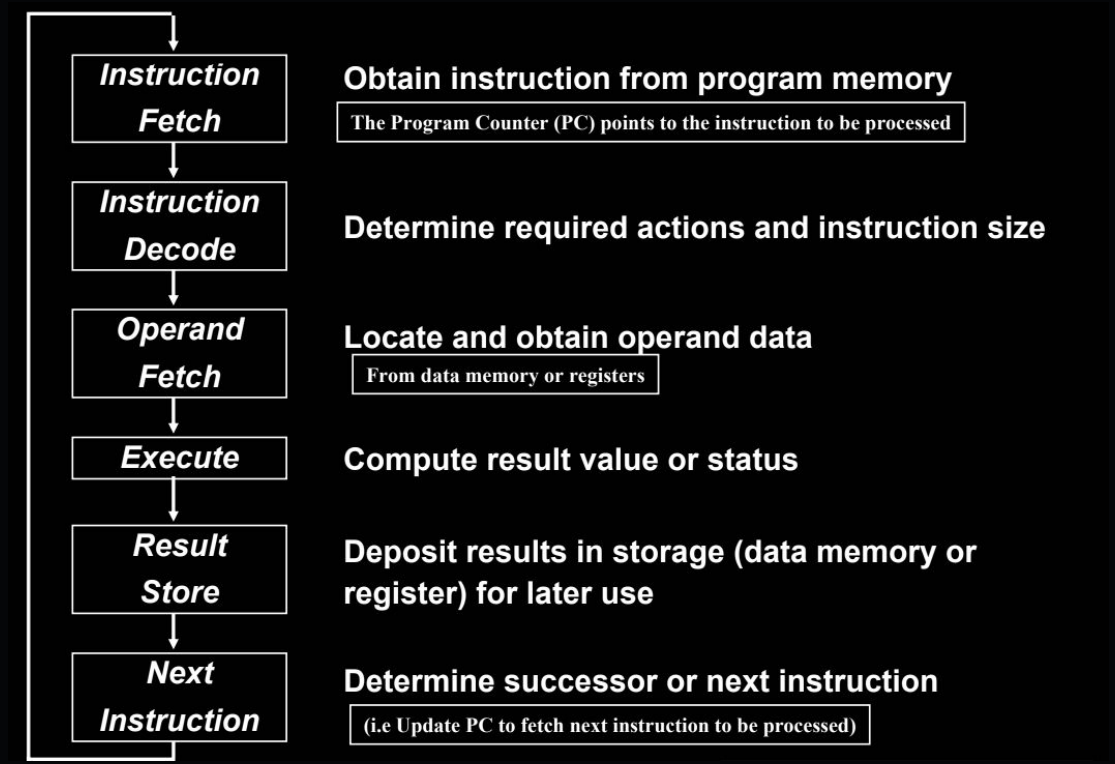
$$\text{CPU Time} = \text{CPU clock cyles} * \text{Clock cycle time}$$

$$\text{CPU Time} = \frac{\text{CPU clock cyles}}{\text{Clock rate}}$$

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

Generic CPU Machine Instruction Processing Steps



Computer Performance Measures: Program Execution Time

- For a specific program compiled to run on a specific machine (CPU) "A", has the following parameters:
  - The total executed instruction count of the program. I
  - The average number of cycles per instruction (average CPI). CPI
  - Clock cycle of machine "A" C

  Or effective CPI
- How can one measure the performance of this machine (CPU) running this program?
  - Intuitively the machine (or CPU) is said to be faster or has better performance running this program if the total execution time is shorter.
  - Thus the inverse of the total measured program execution time is a possible performance measure or metric:

Programs/second — $\text{Performance}_A = 1 / \text{Execution Time}_A$ — Seconds/program

How to compare performance of different machines?

What factors affect performance? How to improve performance?

CPU Execution Time: The CPU Equation

- A program is comprised of a number of instructions executed , I
  - Measured in: instructions/program

AKA Dynamic Instruction Count

- The average instruction executed takes a number of *cycles per instruction (CPI)* to be completed.
  - Measured in: cycles/instruction, CPI

Or Instructions Per Cycle (IPC):
IPC = 1/CPI

- CPU has a fixed clock cycle time C = 1/clock rate
  - Measured in: seconds/cycle

$C = 1/f$

- CPU execution time is the product of the above three parameters as follows:

Executed

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

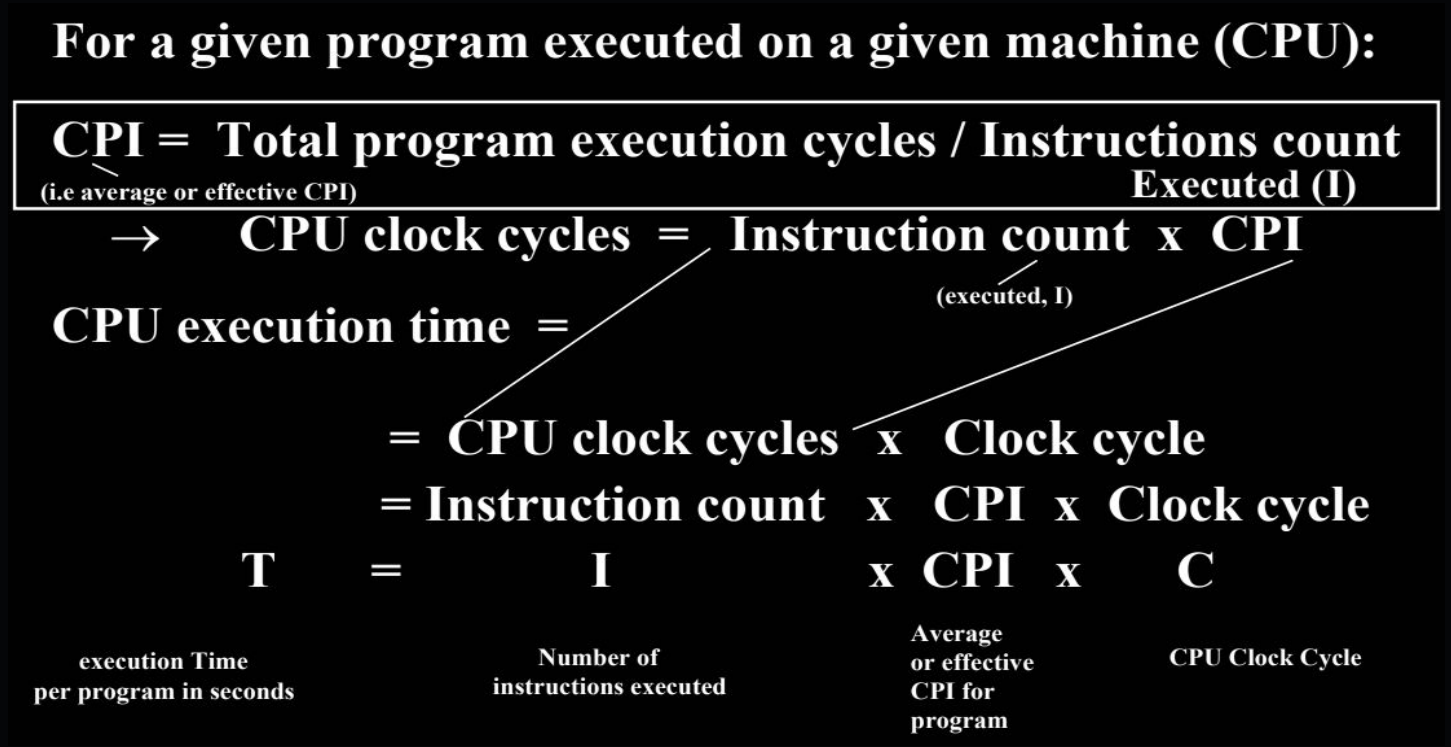
$$T = I \times CPI \times C$$

Instructions vs Cycles

Is the number of cycles identical with the number of instructions?

No!

CPI/Execution Time CPU Average CPI/Execution Time



CPU Execution Time: Example CPU Execution Time: Example

- **A Program is running on a specific machine (CPU) with the following parameters:**
  - **Total executed instruction count: 10,000,000 instructions** (I)
  - **Average CPI for the program: 2.5 cycles/instruction.**
  - **CPU clock rate: 200 MHz. (clock cycle = C = 5×10^{-9} seconds)** i.e 5 nanoseconds
- **What is the execution time for this program:**

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

CPU time = Instruction count x CPI x Clock cycle

= 10,000,000 x 2.5 x 1 / clock rate

= 10,000,000 x 2.5 x 5×10^{-9}

= 0.125 seconds

Factors Affecting CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$T = I \times \text{CPI}_{\text{Average}} \times C$$

	Instruction Count	Cycles per Instruction	Clock Rate (1/C)
Program			
Compiler			
Instruction Set Architecture (ISA)			
Organization (CPU Design)			
Technology (VLSI)			

Instruction count and CPI

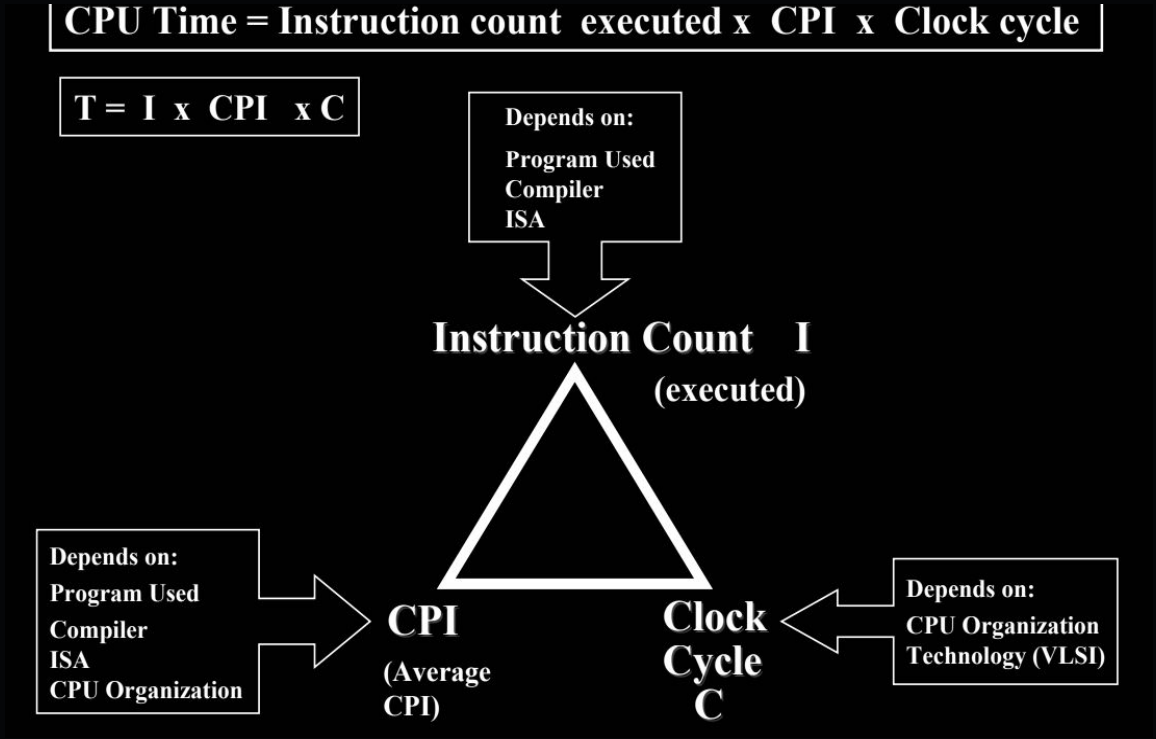
$$CPU\ Time = CPU\ clock\ cyles * Clock\ cycle\ time$$

$$CPU\ clock\ cyles = Instruction\ Count * CPI$$

$$CPU\ Time = Instruction\ Count * CPI * Clock\ cycle\ time$$

$$CPU\ Time = \frac{Instruction\ Count * CPI}{Clock\ Rate}$$

Aspects of CPU Execution Time



Average CPI

Instruction Category	Numbers of Instructions	CPI
ALU	40	2
Load & Store	15	3
Branch	35	4
Other	10	5

MIPS

$$\text{MIPS} = \frac{I_C}{T \times 10^6}$$

$$= \frac{I_C}{I_C \times \text{CPI} \times \tau \times 10^6}$$

$$= \frac{1}{\text{CPI} \times \frac{1}{f} \times 10^6}$$

$$= \frac{f}{\text{CPI} \times 10^6}$$

Practice Problems

Assume the frequency of CPU is given as 4.5GHz. Assume we have different type of instructions which have different numbers of instruction count and CPI. Calculate

1. Average instruction execution time
2. MIPS
3. Program execution time

Instruction Type	Instruction Count	CPI
Load	100	10
Store	200	8
Arithmetic	300	5
Logical	150	4
Shift	250	3
Branch	400	2

Practice Problems

In a certain processor, we have a program which takes 1 million instructions to execute with processor whose speed is given as 2GHz. Suppose 40% of the instruction takes 2 clock cycles, 30% of the instructions takes 3 clock cycles and remaining 30% takes 5 clock cycles for their respective execution. What is the total execution time for program?

Micro Operations of Instruction in COA

Basics of Micro-operations

- There can be multiple Micro Operations in single instruction execution.
- Micro Operations are executed on values stored in registers.
- To describe Micro Operations we use "Register Transfer Language RTL"

Instruction Set Architecture

The ISA can be classified as follows, based on where the operands are stored and whether they are named explicitly or implicitly:

Single accumulator organization, which names one of the general purpose registers as the accumulator and uses it to necessarily store one of the operands. This indicates that one of the operands is implied to be in the accumulator and it is enough if the other operand is specified along with the instruction.

General register organization, which specifies all the operands explicitly.

Ref: https://www.cs.umd.edu/~meesh/411/CA-online/chapter/instruction-set-architecture/index.html

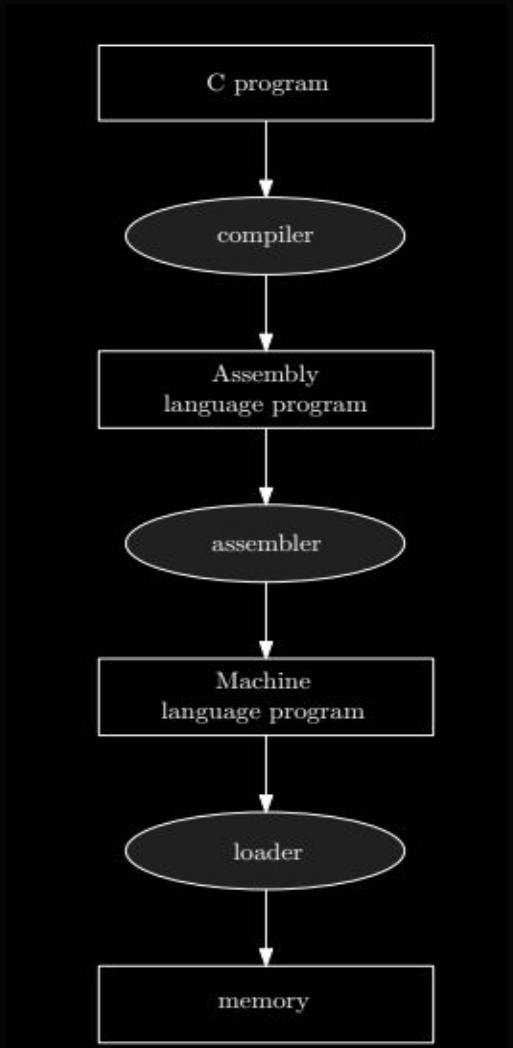
Contd..

Register – register, where registers are used for storing operands. Such architectures are in fact also called load – store architectures, as only load and store instructions can have memory operands.

– Register – memory, where one operand is in a register and the other one in memory.

– Memory – memory, where all the operands are specified as memory operands.

From C program to machine code



Instruction set design

The same program is compiled for a CISC machine (like a VAX) and for a pipelined RISC (like a SPARC based computer). The following data is

available:

IC_{CISC} = 500000, IC_{RISC} = 1100000, CPI_{CISC} = 6.8, CPI_{RISC}= 1.4, Tck_{CISC} = 25 ns

Tck_{RISC} = 30 ns

What is the relative performance of the two machines?

OPERATION SEQUENCE

What is the sequence of operations (single statement per operation) that evaluates the statement:

a = a + b + a * c;

Three Address Machine

The general format of an instruction is:

operation dest, op1, op2

What is the meaning of:

ADD r2, r1, r0

MEMORY ACCESSES

Addresses in an 8 bit machine are 16 bit wide. How many memory accesses are necessary to execute an instruction? Assume that the machine is memory-memory and operands are specified using absolute addresses

MEMORY ACCESSES

The instruction must specify three addresses, which means 3*2 = 6 bytes,

plus an opcode (this is the first to be read by the CPU) which, for an 8 bit

machine, will probably fit in one byte. Therefore the CPU has to read:

1 + 6 = 7 bytes only to get the whole instruction. The source operands have to be read from memory and the result has to be stored into memory: this means 3 memory accesses. The instruction takes: 7 + 3 = 10 memory accesses to complete.

Two-address machines

Implement the high level statement:

a = a + b + a * c

on a 3-address machine; assume that variables a, b, c are in registers r1, r2, and r3 respectively

Contd..

Show how to implement the high level statement

a = a + b + a * c

on a 3-address machine and then on a 2-address machine. Both machines are 8 bit register-register machines with 32 general purpose registers and a 16 bit addresses. The values of variables a, b, and c are stored in r1, r2 and r3 respectively. In any case calculate the number of clock cycles necessary if every memory access takes two clock cycles and the execution phase of an instruction takes one clock cycle.

Contd..

For the 3-address machine:

```
MUL r4, r1, r3      # 3 * 2 + 1 clock cycles
ADD r1, r1, r2      # 3 * 2 + 1
ADD r1, r1, r4      # 3 * 2 + 1
```

Contd..

For the 2-address machine:

MOV r4, r1 # 2 * 2 + 1 clock cycles

MUL r4, r3 # 2 * 2 + 1

ADD r1, r2 # 2 * 2 + 1

ADD r1, r4 # 2 * 2 + 1

One-Address

Show how to implement the statement

a = a + b + a * c

using an accumulator machine

Zero Address

Show how to implement the statement

a = a + b + a * c

Addressing Modes

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called Addressing.

- Immediate Addressing
- Direct Addressing
- Register Addressing
- Register Indirect Addressing
- Implied Addressing
- Relative Addressing
- Indexed Addressing

ADDRESSING MODES OF 8086

1. Immediate Addressing

In immediate addressing mode, an 8/16 bit immediate data/constant is specified in the instruction itself.

In this mode the operand is specified in the instruction itself. Instructions are longer but the operands are easily identified.

Eg:

MOV CL, 12H ;	Moves 12 immediately into CL register
MOV BX, 1234H ;	Moves 1234 immediately into BX register

2. REGISTER ADDRESSING MODE

In this mode operands are specified using registers.

Instructions are shorter but operands cannot be identified by looking at the instruction.

Eg:

MOV CL, DL ; Moves data of DL register into CL register

MOV AX, BX ; Moves data of BX register into AX register

3. DIRECT ADDRESSING MODE

In this mode address of the operand is directly specified in the instruction.

Eg:

```
MOV CL, [2000H]      ; CL Register gets data from memory location 2000H
                     ; CL <- [2000H]
```

Eg:

```
MOV [3000H], DL      ; Memory location 3000H gets data from DL Register
                     ; [3000H] <-  DL
```

4. INDIRECT ADDRESSING MODE

In Indirect Addressing modes, address is given by a register.

The register can be incremented in a loop to access a series of locations.

There are various sub-types of Indirect addressing mode

A. REGISTER INDIRECT ADDRESSING MODE

This is the most basic form of indirect addressing mode.

Here address is simply given by a register.

Eg: MOV CL, [BX] ; CL gets data from a memory location pointed by BX

; CL <- [BX]. If BX = 2000H, CL <- [2000H]

Eg: MOV [BX], CL ; CL is stored at a memory location pointed by BX

; [BX] <- CL. If BX = 2000H, [2000H] <- CL.

B. REGISTER RELATIVE ADDRESSING MODE

Here address is given by a register plus a numeric displacement.

Eg:

```
MOV CL, [BX + 03H]    ; CL gets data from a location BX + 03H
                      ; CL <-  [BX+03H]. If BX = 2000H, then CL <- [2003H]
```

Eg:

```
MOV [BX + 03H], CL    ; CL is stored at location BX + 03H
                      ; [BX+03H] <- CL. If BX = 2000H, then [2003H] <- CL
```

C. BASE INDEXED ADDRESSING MODE

Here address is given by a sum of two registers.

This is typically useful in accessing an array or a look up table.

One register acts as the base of the array holding its starting address and the other acts as

an index indicating the element to be accessed.

Eg: MOV CL, [BX + SI] ; CL gets data from a location BX + SI

; CL <- [BX+SI].

; If BX = 2000H, SI = 1000H, then CL <- [3000H

Contd..

Eg: MOV [BX + SI], CL ; CL is stored at location BX + SI

; [BX+SI] <- CL.

; If BX = 2000H, SI = 1000H, then [3000H] <- CL

D. BASE RELATIVE PLUS INDEX ADDRESSING MODE

Here address is given by a sum of base register plus index register plus a numeric displacement.

Eg: MOV CL, [BX+SI+03H] ; CL gets data from a location BX + SI + 03H

; CL <- [BX+SI+03H].

; If BX = 2000H, SI = 1000H, then CL <- [3003H]

5. IMPLIED ADDRESSING MODE

In this addressing mode, the operand is not specified at all, as it is an implied operand. Some instructions operate only on a particular register. In such cases, specifying the register becomes unnecessary as it becomes implied.

```
Eg:     STC     ; Sets the Carry flag.
                ; This instruction can only operate on the Carry Flag.
Eg:     CMC     ; Complements the Carry flag.
                ; This instruction can only operate on the Carry Flag.
```

ADDRESSING MODES OF 8085

Immediate Addressing Mode

In this mode, the Data is specified in the Instruction itself.

Eg:

MVI A, 35H ; Move immediately the value 35 into the Accumulator. i.e. A <- 35H

LXI B, 4000H ; Move immediately the value 4000 into BC register pair. i.e. BC <- 4000H

Contd..

Advantage:

- Programmer can easily identify the operands.

Disadvantage:

- Always more than one byte hence requires more space.
- The µP requires two or three machine cycles to fetch the instruction hence slow

REGISTER ADDRESSING MODE

In this mode, the Data is specified in Registers.

Eg:

```
MOV B, C        ; Move the Contents of C-Register into B-Register. i.e. B<- C
INR B           ; Increments the contents of B-Register. i.e. B <- B + 1
```

Contd...

Advantage:

Instructions are of one byte so only one cycle is required to fetch them.

Disadvantage:

Operands cannot be easily identified.

DIRECT ADDRESSING MODE

In this mode, the Address of the operand is specified in the Instruction itself.

Eg:

LDA 2000H ; Loads “A” register with Contents of Location 2000. i.e. A <- [2000]

STA 2000H ; Stores the Contents of “A” register at the Location 2000. I.e.

[2000] <- A

Contd...

Advantage:

The programmer can identify the address of the operand.

Disadvantage:

These are three byte instructions hence require three fetch cycles

INDIRECT ADDRESSING MODE

In this mode, the Address of the operand is specified in Registers.

Hence, the instruction indirectly points to the operands.

E.g.:

LDAX B ; Loads “A” register with the contents of the memory Location pointed by BC Pair

; So if BC pair = 4000 i.e. [BC] = 4000 then A <- [4000].

IMPLIED ADDRESSING MODE

In this mode, the Operand is implied in the instruction.

This instruction will work only on that implied operand, and not on any other operand

Eg:

STC ; Sets the Carry Flag in the Flag register, Cy <- 1.

CMC ; Complements the Carry Flag in the Flag register

Contd..

Advantage:

Instructions are generally only one byte.

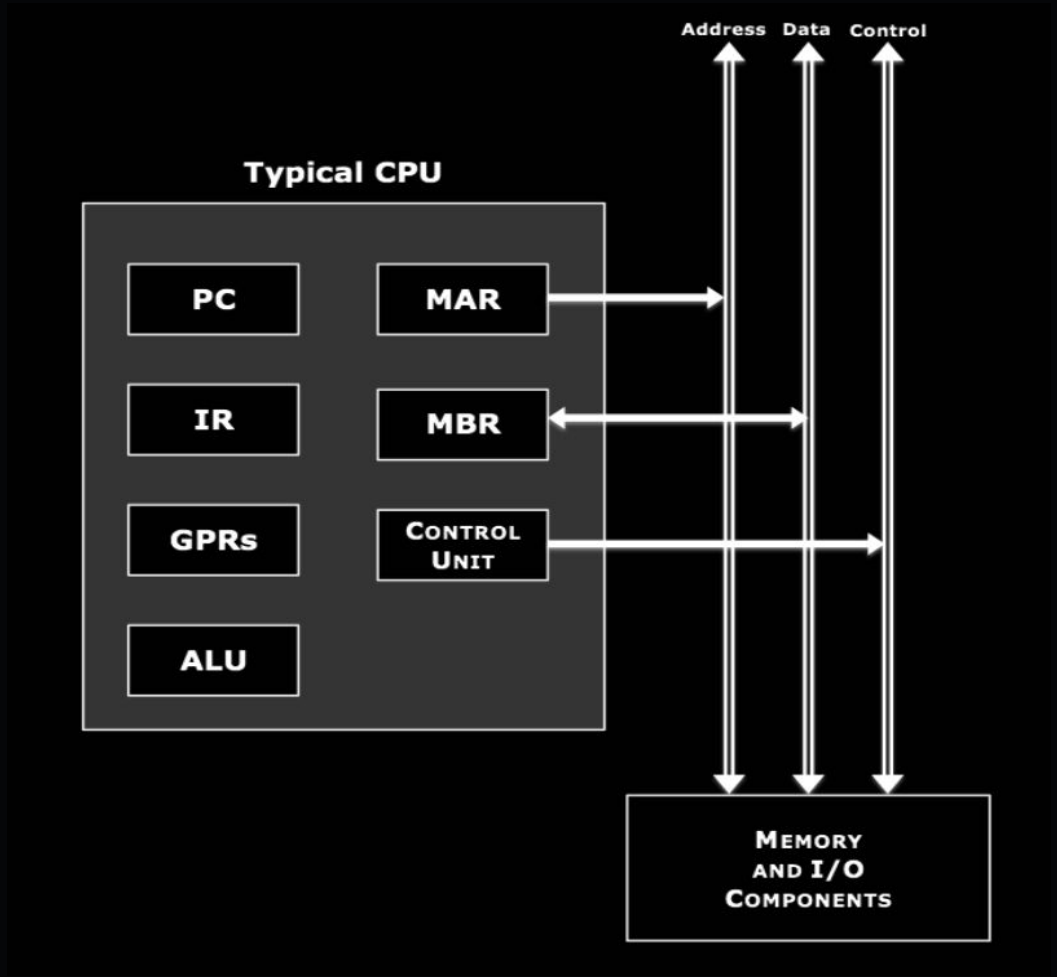
Disadvantage:

Rigid, as it works only on a fixed operand

MICRO-OPERATIONS & CONTROL SIGNALS

- A Program is a set of Instructions.
- An Instruction, requires a set of small operations called Micro-Operations.
- A Micro-Operation is a finite activity performed by the processor in one clock cycle. One clock cycle is also called one T-state (Transition state).
- One Micro-Operation requires One T-state.
- Several Independent Micro-Operations can be performed in the same T-state.
- Control unit generates control signals to perform these very Micro-Operations.
- To understand Control Units, we must first clearly understand Micro-Operations.

contd..


Address
Data
Control
Typical CPU
PC
MAR
IR
MBR
GPRs
Control Unit
ALU
Memory and I/O Components

MICRO-OPERATIONS FOR INSTRUCTION FETCHING

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction from MBR*
	PC ← PC + 1	*PC gets incremented*

Contd..

As we can see in the above table, two Micro-Operations can take place in the same T-state i.e.

IR <- MBR and PC <- PC + 1

This is because they are completely independent of each other.

In fact, PC becoming PC + 1 can also be performed in the 2nd T-state while the instruction is being fetched from the memory through the data bus into MBR.

MICRO-OPERATIONS FOR INSTRUCTION FETCHING (ALTERNATE METHOD)

State	Micro-Operation	Explanation
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
	PC ← PC + 1	*PC gets incremented*
T3:	IR ← MBR	*IR gets instruction from MBR*

Contd..

PC <- PC + 1 cannot take place in the 1st T-state.

That’s because, in the 1st T-state, PC is providing the address on the address bus, through MAR. If at the same time PC gets incremented, then the incremented address will be put on the address bus.

Now that we know how an instruction is fetched, we can proceed further and learn Micro-Operations for various instructions, of different Addressing Modes.

MICRO-OPERATIONS FOR IMMEDIATE ADDRESSING MODE

E.g.: MOV R1, 25H; R1 register gets the immediate value 25H

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, 25H" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	R1 ← 25H (IR)	*R1 register gets the value 25H from IR*

MICRO-OPERATIONS FOR REGISTER ADDRESSING MODE

E.g.: MOV R1, R2; R1 register gets the data from Register R2

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, R2" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	R1 ← R2	*R1 register gets the value from R2 Register*

MICRO-OPERATIONS FOR DIRECT ADDRESSING MODE

E.g.: MOV R1, [2000H]; R1 register gets the data from memory location 2000H

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, [2000H]" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← IR (2000H)	*MAR gets the address 2000H from IR*
T5:	MBR ← Memory ([2000H])	*MBR gets contents of location 2000H from Memory.*
T6:	R1 ← MBR ([2000H])	*Register R1 gets contents of memory location 2000H from MBR*

MICRO-OPERATIONS FOR INDIRECT ADDRESSING MODE

E.g.: MOV R1, [R2]; R1 register gets the data from memory location pointed by R2

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, [R2]" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← R2	*MAR gets the address R2 Register*
T5:	MBR ← Memory ([R2])	*MBR gets contents of location pointed by R2 from Memory.*
T6:	R1 ← MBR ([R2])	*Register R1 gets contents of memory location pointed by R2 from MBR*
		In the exam, once, Add R1, [R2] was asked. *Everything else will be same. Only change: T6: R1 ← R1 + MBR*

MICRO-OPERATIONS FOR INDIRECT ADDRESSING MODE

E.g.: MOV [R2], R1; R1 register stores data into memory location pointed by R2

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV [R2], R1" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← R2	*MAR gets the address R2 Register*
T5:	MBR ← R1	*R1 puts data into MBR to store it in the memory location pointed by R2.*

MICRO-OPERATIONS FOR IMPLIED ADDRESSING MODE

E.g.: STC; Set the Carry Flag; (CF <- 1)

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "STC" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	CF ← 1	*Carry Flag in the Flag Register becomes 1*

Control Unit

HARDWIRED CONTROL UNIT

Here control signals are produced by hardware.

There are three types of Hardwired Control Units

STATE TABLE METHOD

1) It is the most basic type of hardwired control unit.

2) Here the behavior of the control unit is represented in the form of a table called the state table.

3) The rows represent the T-states and the columns indicate the instructions.

4) Each intersection indicates the control signal to be produced, in the corresponding T-state of every

instruction.

5) A circuit is then constructed based on every column of this table, for each instruction.

Contd...

T-States	Instructions			
	I_1	I_2	...	I_N
T_1	$Z_{1,1}$	$Z_{1,2}$	...	$Z_{1,N}$
T_2	$Z_{2,1}$	$Z_{2,2}$	...	$Z_{2,N}$
...	...	...	...	...
T_M	$Z_{M,1}$	$Z_{M,2}$	...	$Z_{M,N}$

Contd..

ADVANTAGE:

It is the simplest method and is ideally suited for very small instruction sets.

DRAWBACK:

As the number of instructions increase, the circuit becomes bigger and hence more complicated.

As a tabular approach is used, instead of a logical approach (flowchart), there are duplications of many circuit elements in various instructions.

DELAY ELEMENT METHOD

1) Here the behavior of the control unit is represented in the form of a flowchart.

2) Each step in the flowchart represents a control signal to be produced.

3) Once all steps of a particular instruction, are performed, the complete instruction gets executed.

4) Control signals perform Micro-Operations, which require one T-states each.

5) Hence between every two steps of the flowchart, there must be a delay element

Cond..

6) The delay must be exactly of one T-state. This delay is achieved by D Flip-Flops.

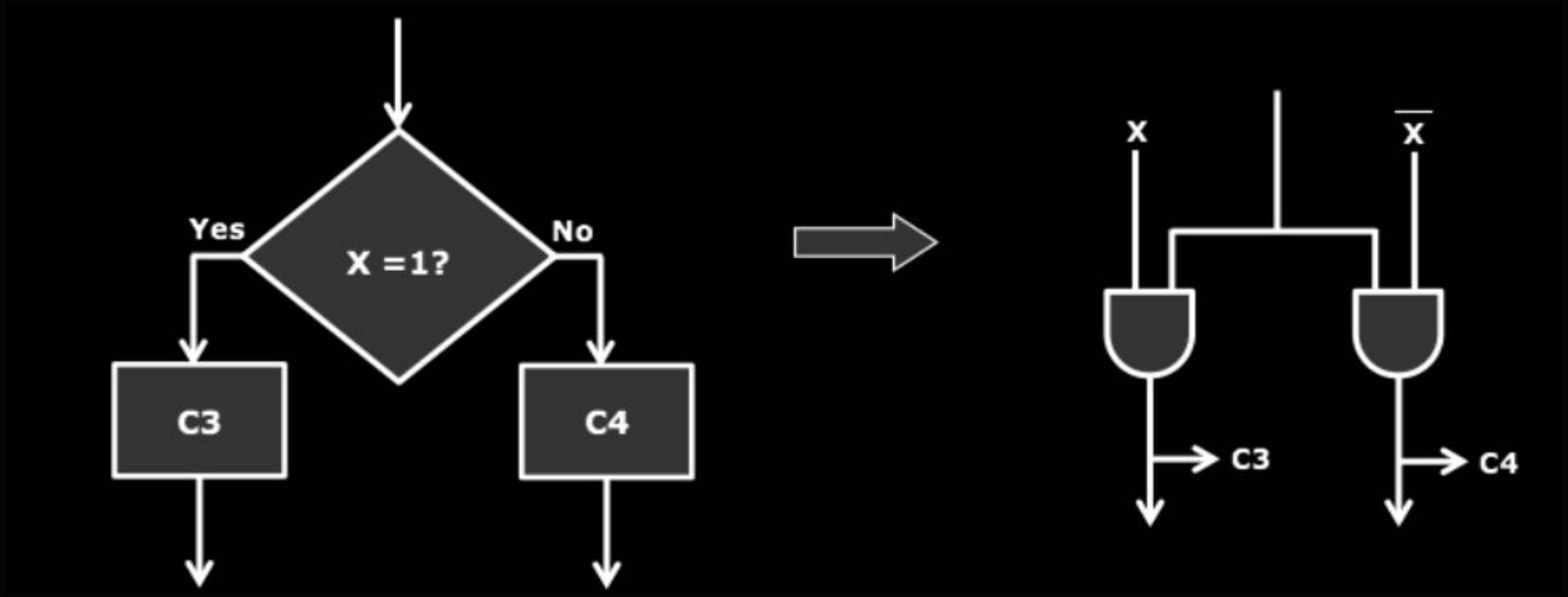
7) These D Flip-Flops are inserted between every two consecutive control signals.

8) Of all D Flip-Flops only one will be active at a time. So the method is also called “One Hot Method”.

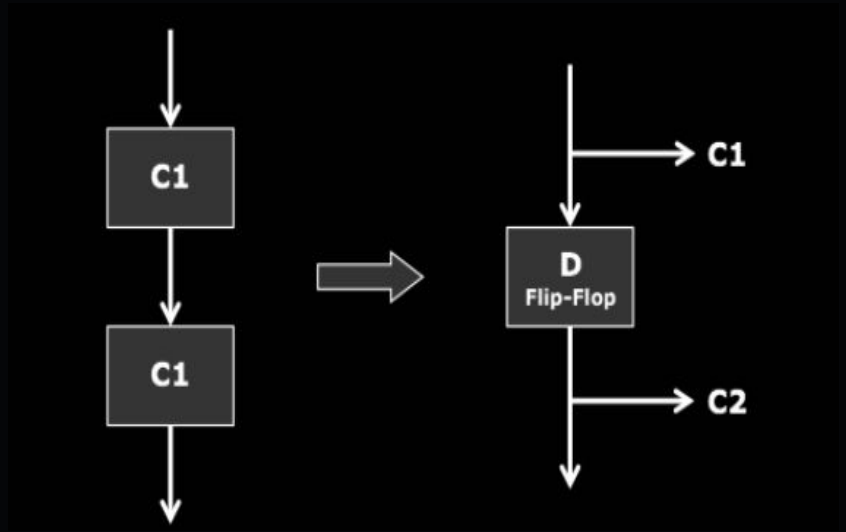
9) In a multiple entry point, to combine two or more paths, we use an OR gate.

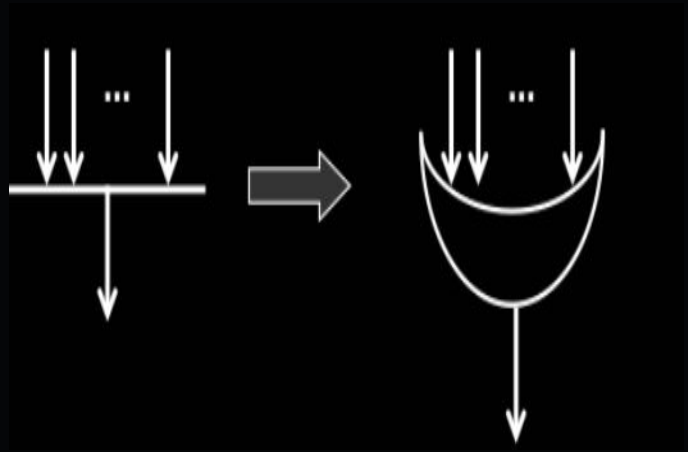
10)A decision box is replaced by a set of two complementing AND gates

Contd..



Contd..





Contd..

ADVANTAGE:

As the method has a logical approach, it can reduce the circuit complexity.

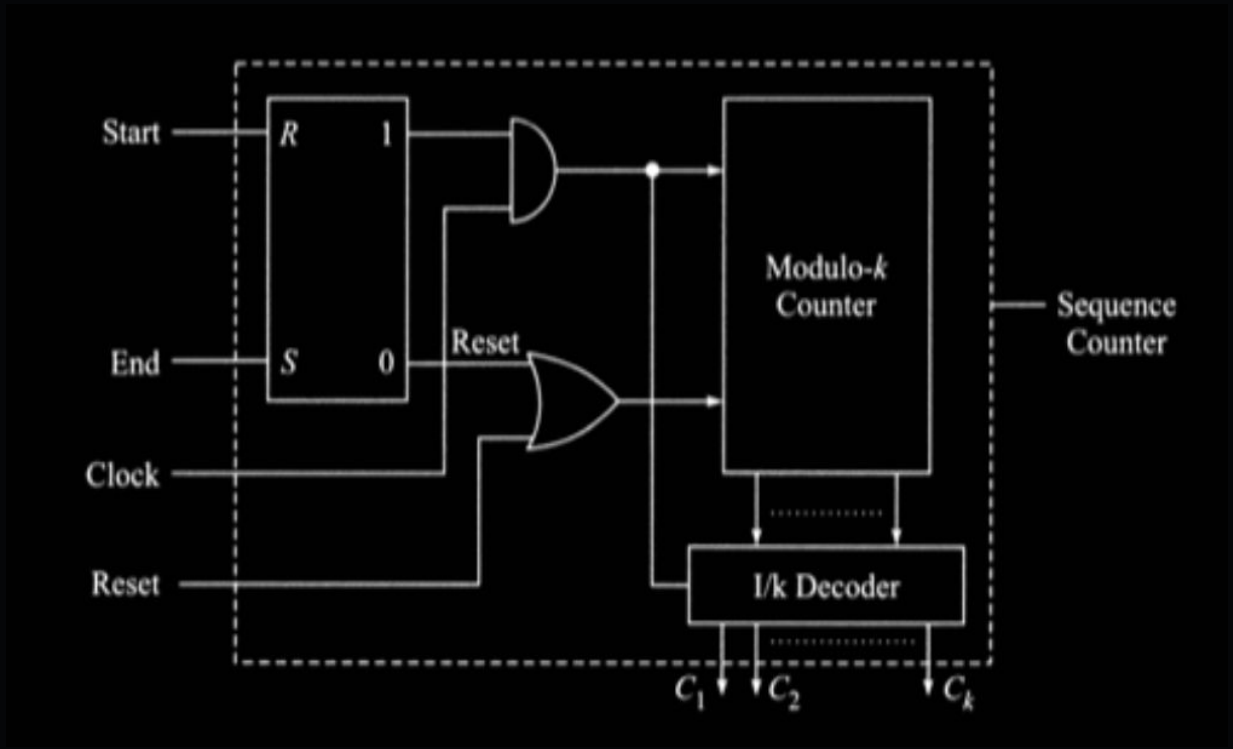
This is done by re-utilizing common elements between various instructions.

DRAWBACK:

As the no of instructions increase, the number of D Flip-Flops increase, so the cost increases.

Moreover, only one of those D Flip-Flops are actually active at a time.

SEQUENCE COUNTER METHOD



Contd...

1) This is the most popular form of hardwired control unit.

2) It follows the same logical approach of a flowchart, like the Delay element method, but does not

use all those unnecessary D Flip-Flops.

3) First a flowchart is made representing the behavior of a control unit.

4) It is then converted into a circuit using the same principle of AND & OR gates.

5) We need a delay of 1 T-state (one clock cycle) between every two consecutive control signals

Contd...

6) That is achieved by the above circuit.

7) If there are “k” number of distinct steps producing control signals, we employ a “mod k” and “k” output decoder.

8) The counter will start counting at the beginning of the instruction.

9) The “clock” input via an AND gate ensures each count will be generated after 1 T-state.

10)The count is given to the decoder which triggers the generation of “k” control signals, each after a delay of 1 T-state.

11)When the instruction ends, the counter is reset so that next time, it begins from the first count.

Contd...

ADVANTAGE: Avoids the use of too many D Flip-Flops.

GENERAL DRAWBACKS OF A HARDWIRED CONTROL UNIT

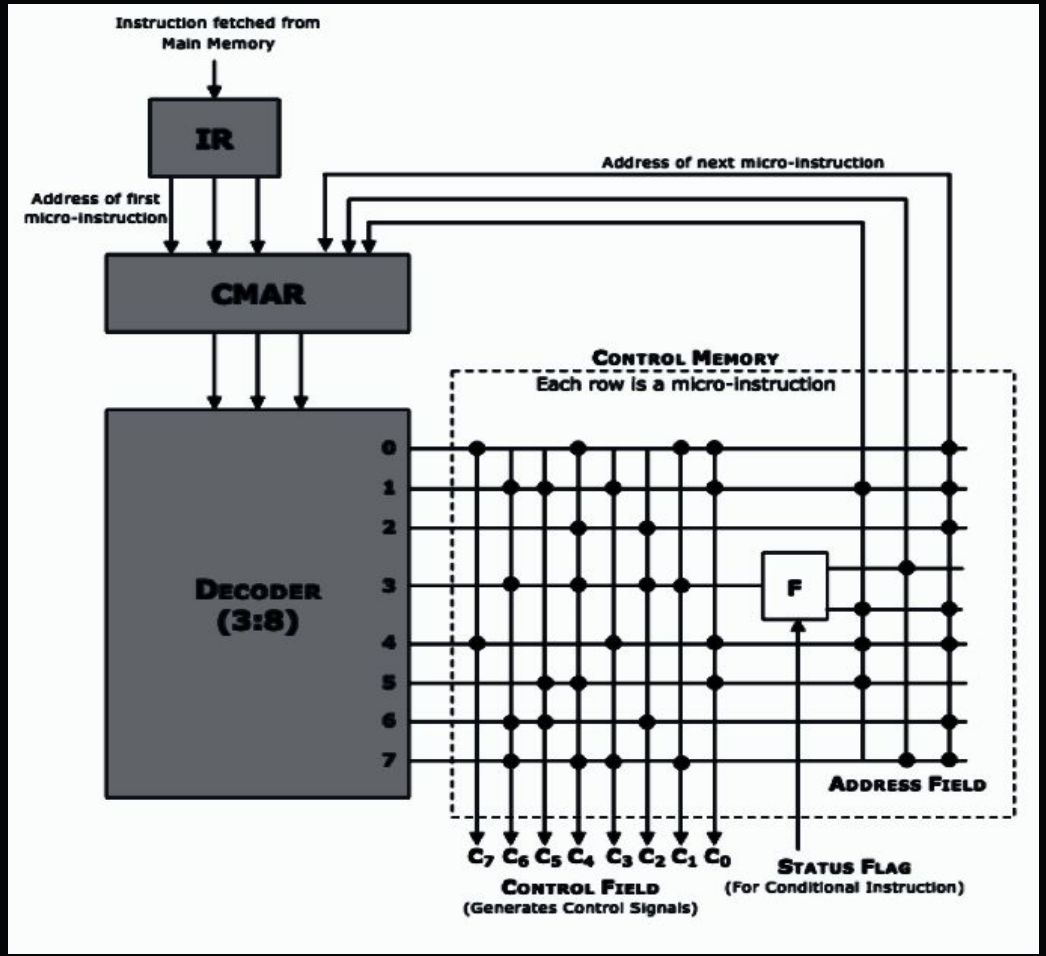
1) Since they are based on hardware, as the instruction set increases, the circuit becomes more and more complex. For modern processors having hundreds of instructions, it is virtually impossible to create Hardwired Control Units.

2) Such large circuits are very difficult to debug.

3) As the processor gets upgraded, the entire Control Unit has to be redesigned, due to the rigid nature of hardware design.

MICROPROGRAMMED CONTROL UNIT

WILKES' DESIGN FOR A MICRO-PROGRAMMED CONTROL UNIT



WILKES' DESIGN

1) Microprogrammed Control Unit produces control signals by software, using micro-instructions

2) A program is a set of instructions.

3) An instruction requires a set of Micro-Operations.

4) Micro-Operations are performed by control signals.

5) Instead of generating these control signals by hardware, we use micro-instructions.

WILKES' DESIGN

6) This means every instruction requires a set of micro-instructions

7) This is called its micro-program.

8) Microprograms for all instructions are stored in a small memory called "Control Memory".

9) The Control memory is present inside the processor.

10) Consider an Instruction that is fetched from the main memory into the Instruction Register (IR).

WILKES' DESIGN

11) The processor uses its unique "opcode" to identify the address of the first micro-instruction.

12) That address is loaded into CMAR (Control Memory Address Register).

13) CMAR passes the address to the decoder.

14) The decoder identifies the corresponding micro-instruction from the Control Memory.

15) A micro-instruction has two fields: a control field and an address field.

WILKES' DESIGN

16) Control field: Indicates the control signals to be generated.

17) Address field: Indicates the address of the next micro-instruction.

18) This address is further loaded into CMAR to fetch the next micro-instruction.

19) For a conditional micro-instruction, there are two address fields.

20) This is because, the address of the next micro-instruction depends on the condition.

WILKES' DESIGN

21) The condition (true or false) is decided by the appropriate control flag.

22) The control memory is usually implemented using FLASH ROM as it is writable yet non volatile.

ADVANTAGES

1) The biggest advantage is flexibility.

2) Any change in the control unit can be performed by simply changing the micro-instruction.

3) This makes modifications and up gradation of the Control Unit very easy.

4) Moreover, software can be much easily debugged as compared to a large Hardwired Control Unit.

DRAWBACKS

1) Control memory has to be present inside the processor, increasing its size.

2) This also increases the cost of the processor.

3) The address field in every micro-instruction adds more space to the control memory. This can be easily avoided by proper micro-instruction sequencing.