Microprocessor Systems

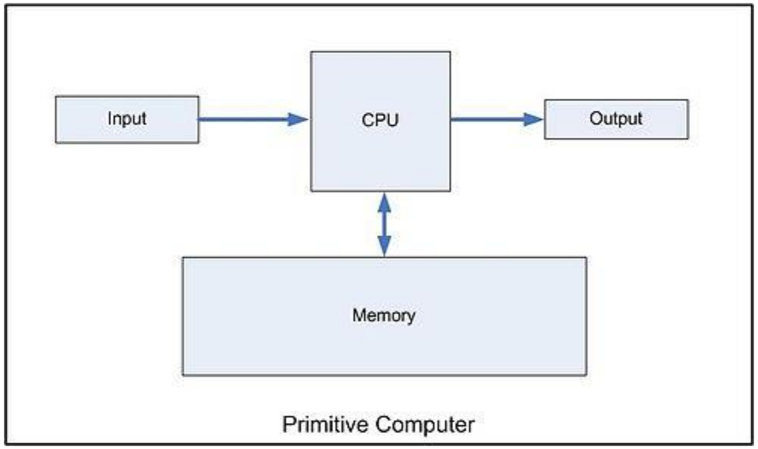
Dr. Gökhan İnce

Topics

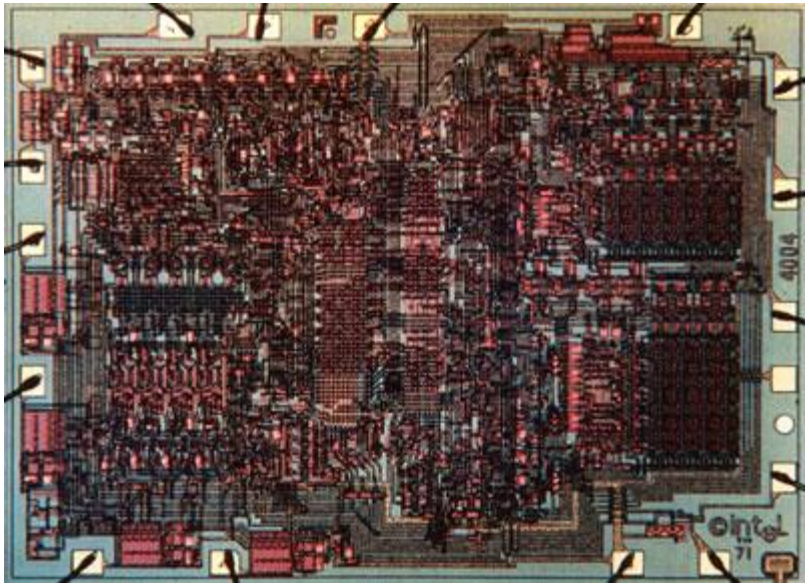
- Central Processing Unit Structure
- Instruction Format

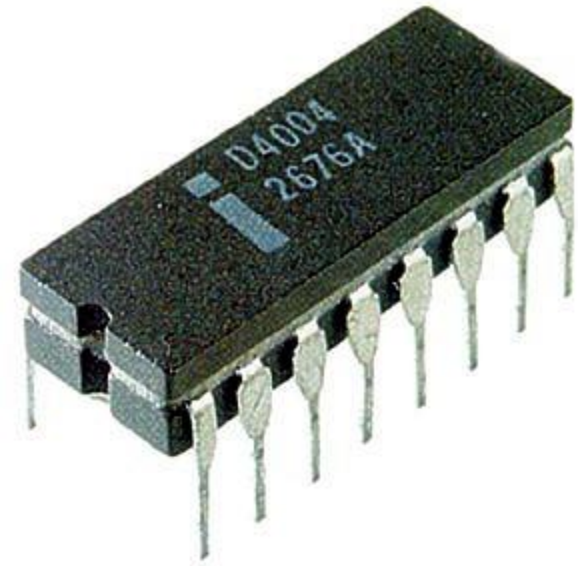
Central Processing Unit

- CPU is the fundamental execution/processing unit of the computer
- CPU consists of ALU, Control Unit, and Registers
- CPU is characterized by:
  - Clock frequency
  - Speed
  - Data bus width
  - Instruction set
  - Addressing capability
  - Addressing capacity



Internal Structure of CPU





- Intel 4004
  - in 1971, commercially available single-chip microprocessor
  - 12 bit address bus
  - 4 bit data bus

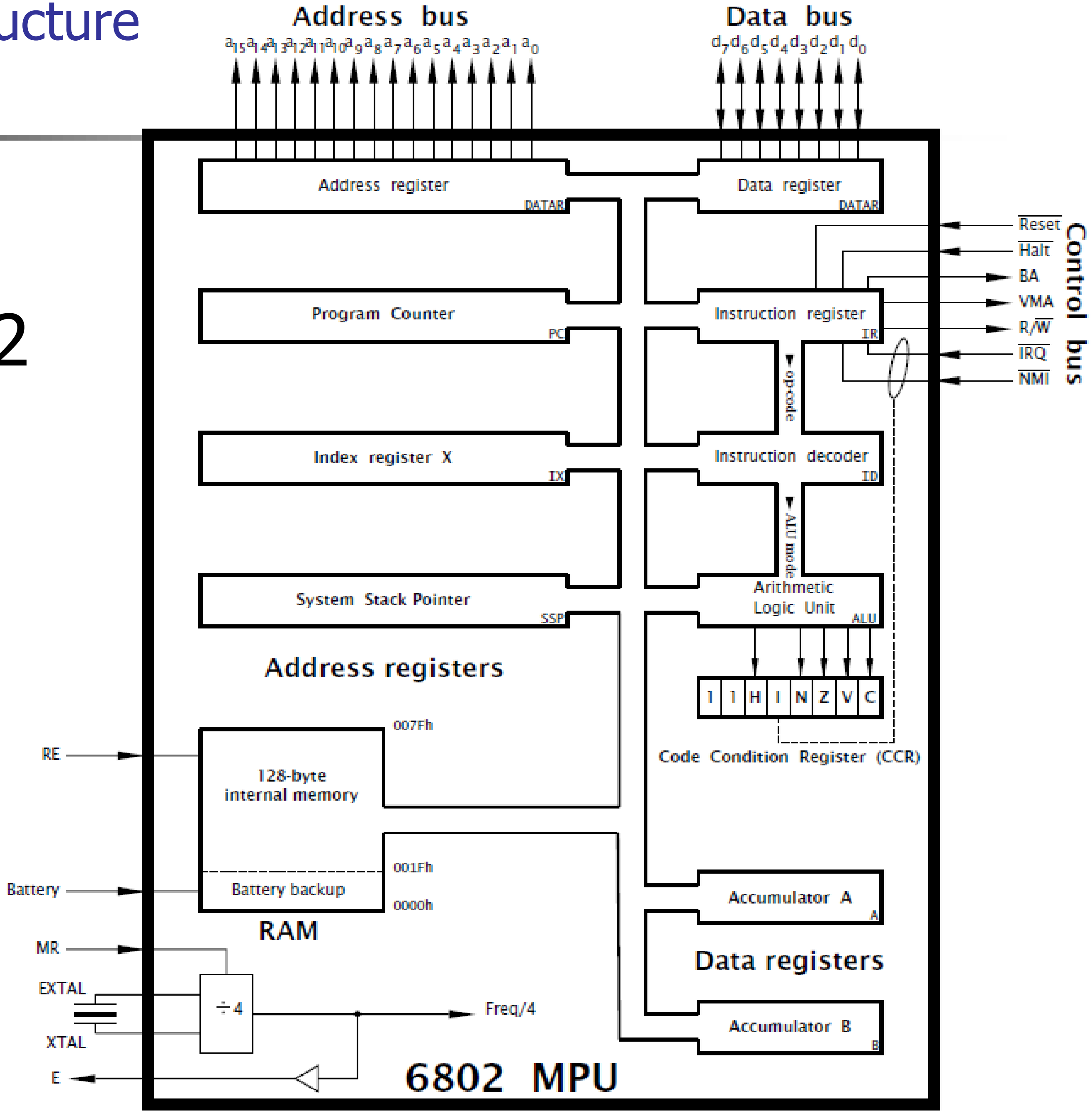
CPU Elements

- Memory Address Register (MAR)
- Memory Data Register (MDR)
- Arithmetic and Logic Unit (ALU)
- Accumulator (ACC)
- Condition Code Register (CCR)
- General Purpose Registers (C, D, H, L …)
- Program Counter (PC)
- Instruction Register (IR)
- Instruction Decoder
- Stack Register (Pointer) (SP)
- Index Register (IX)
- Control Unit (CU)

Internal Structure of CPU

Example:

Motorola 6802

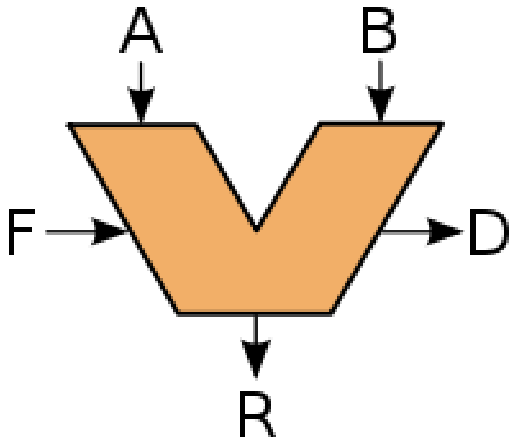


CPU Components

- Memory Address Register (MAR) stores the memory address from which data will be fetched to the CPU or the address to which data will be sent and stored.
- Memory Data Register (MDR) contains the data to be stored in the computer storage (e.g. RAM), or the data after a fetch from the computer storage.
- Accumulator (ACC) may contain data to be used in a mathematical or logical operation, or it may contain the result of an operation.
- General purpose registers are used to support the accumulator by holding data to be loaded to/from the accumulator.
- Arithmetic and Logic Unit (ALU) performs all arithmetic and logic operations in a microprocessor

Arithmetic Logic Unit

- ALU has two inputs (A, B) for the operands and one input for a control signal that selects the operation
- Operation and Shift control bits determine, which type of operation to perform (F)
- Output is the result of operation (R) and status information (D)
- Status information is used to indicate cases
  - Zero: if all result lines have value 0
  - Overflow: integer overflow of add and subtract functions
    - for unsigned integers, it does not provide any useful information



Condition Code Register

- Depending on the outcomes of Arithmetic or Logical operations, we can *branch* and *jump*
- The eight-bit Condition Code Register (CCR) provides a status report on the ALU's **activity**
  - **C**arry/Borrow
  - **H**alf carry from bit 3 to bit 4
  - o**V**erflow
- CCR also provides a status report **after loading** ACC
  - **Z**ero
  - **N**egative

			V	Z	N	H	C

Condition Code Register (Flag Register)

- They flag certain conditions resulting from the ALU outcomes
- Example:

A= 01001000 B= 01111001

A+B:

A 01001000 H=1→BCD:+0110

B +01111001

11000001 V=1 Z=0 N=1 H=1 C=0

- Depending on the outcomes of Arithmetic or Logical operations, we can branch and jump

Registers

- A register is a storage location in the CPU
- It is used to hold data or a memory address during the execution of an instruction
- Because the register file is small and close to the ALU, accessing data in registers is much faster than accessing data in memory outside the CPU
- The register file makes program execution more efficient
- The number of registers varies from computer to computer

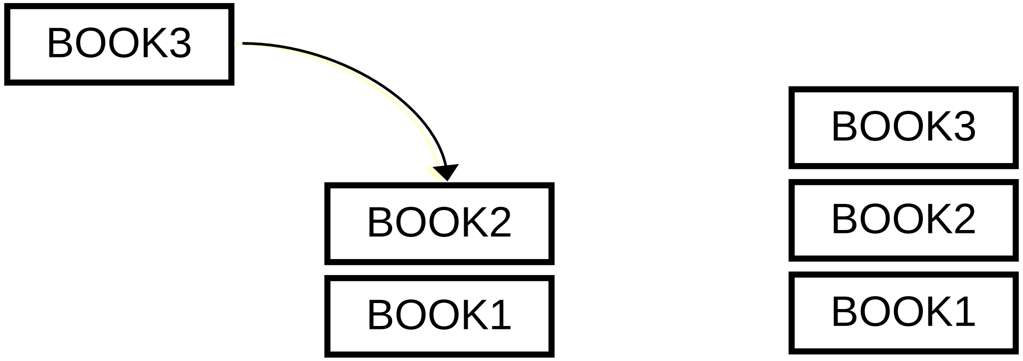
Additional Computer Registers

- **Program Counter:** holds the memory location of the next instruction.
- **Instruction Register:** holds the current instruction being executed

Instruction Decoder: It decodes the instructions and generates the control signals

The Stack

- A stack of a computer works just like a real stack, e.g., of books. If you have a stack of books, you can put another book on top:



BOOK3

BOOK2

BOOK1

BOOK3

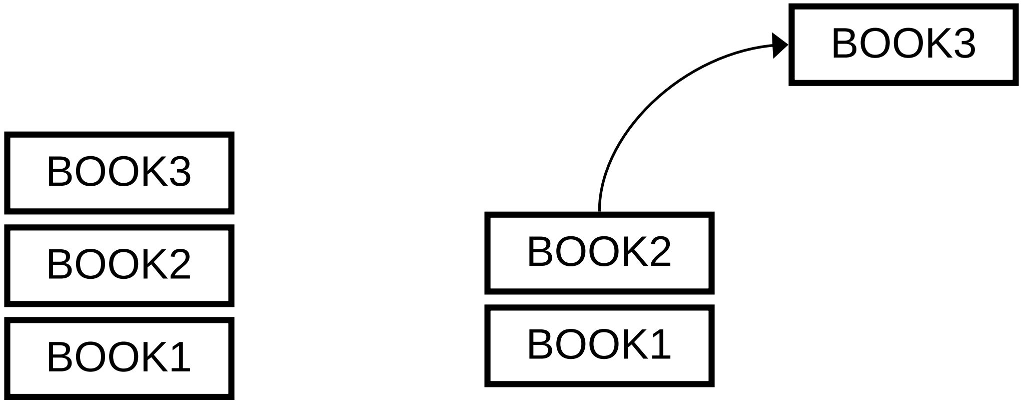
BOOK2

BOOK1

- This is called a **push**. All that happens is the stack gets one book deeper, and the last book you added is on top.

The Stack

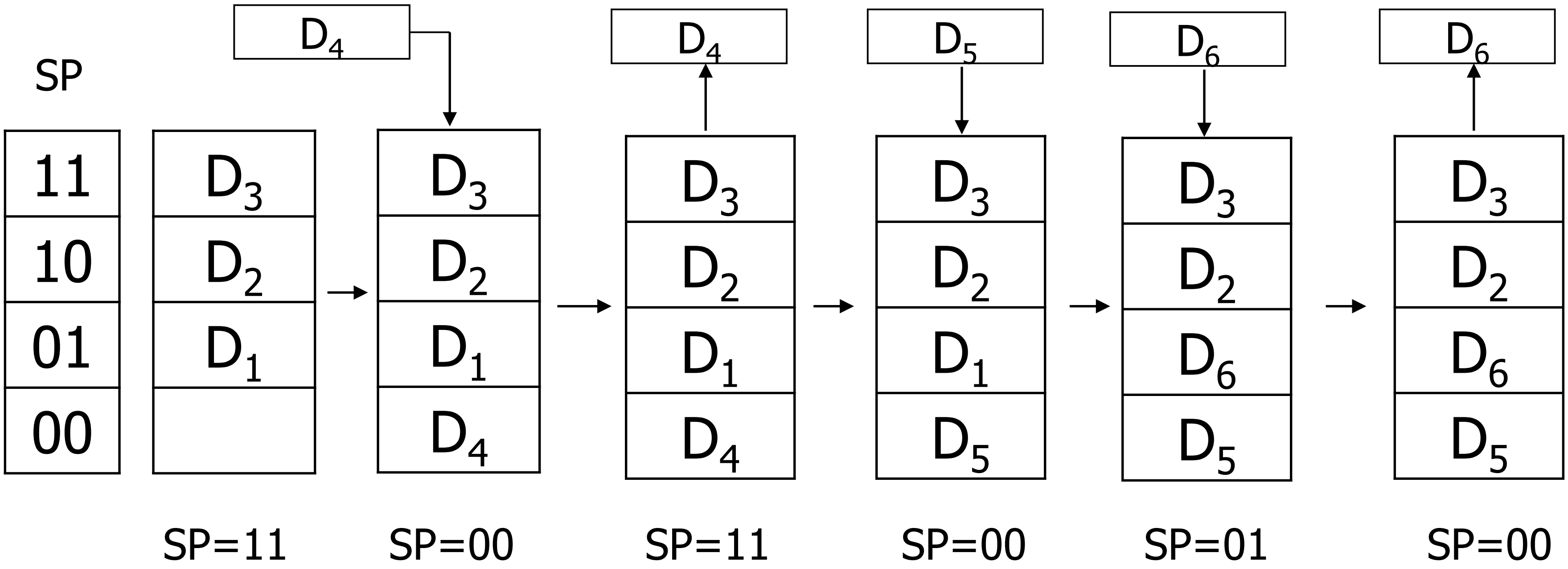
- You can also take a book off the top of the stack:



- This is called a **pop**. The stack gets one book shorter, and the book you get from the top is the one you added, or pushed, most recently.
- Because a pop gives you back the item you most recently pushed, a stack is called a **last-in-first-out**, or **LIFO**, structure.

The Stack

A stack is a last-in-first-out data structure.



Stack Pointer

- The stack is a way of using the memory. All that's needed is some **unused memory and an index register, called the Stack Pointer** (SP), that always points to the next available (empty) location above the current top of the stack. The stack grows toward lower addresses.

		SP
Address		$A000
$A000	D_0	$9FFF
$9FFF	D_1	$9FFE
$9FFE	D_2	$9FFD
$9FFD	D_3	$9FFC
$9FFC	D_4	$9FFB
$9FFB		
$9FFA		

Index Register

Index Register (IX) is used to calculate the address of a specified element within an array.

Control Unit

- The control unit is a **synchronous sequential** logic circuit that sends control signals to the data processing unit, memory and other parts of the system
- The signals from the control unit tells the data processing unit to manipulate data according to the algorithm built into the sequential logic circuit
- The control unit is **instruction controlled**; therefore it can do more than one algorithm based on its design.
- Typical control units recognize several hundred different instruction codes.

System Clock

- In order to regulate when the control unit issues its control signals, computers use a system clock

- System clock generates **regular pulses** to synchronize all system events and determine the speed at which processing can occur

- Each fetch-execute instruction cycle is divided into states, which are one clock pulse long. Most instructions require multiple steps, and so require **several clock pulses** to complete

- Some individual steps (e.g. a **memory access**) take longer & may require additional clock pulses to complete – these clock cycles spent waiting are called **wait states**

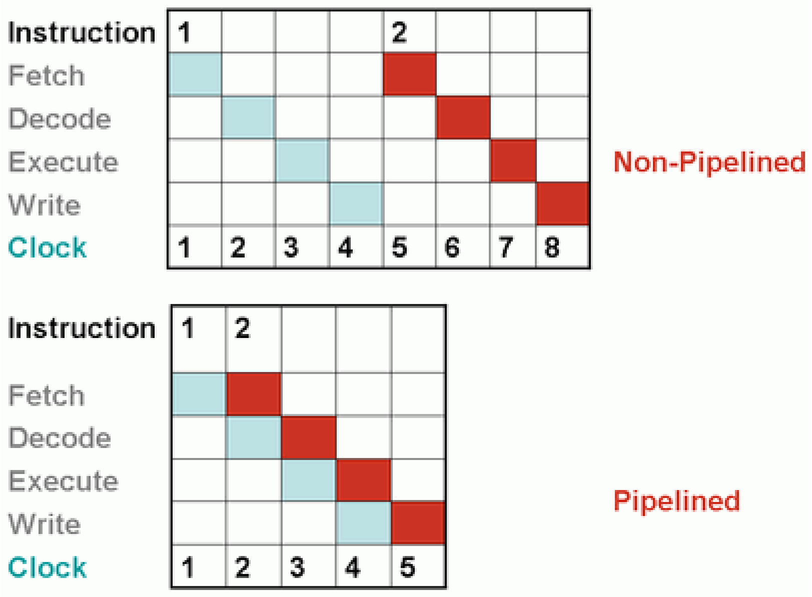
System Clock

- The clock speed of a CPU determines how often a new instruction is executed, and is measured in MHz or GHz:
  1.7GHz means that a computer executes 1,700,000,000 instructions per second!

System Clock

- However, all recent microprocessors overlap the fetching, decoding and execution of a number of instructions at same time. This is called **pipelining**.

- Therefore, clock speed is not necessarily an accurate measure of performance, and other measurements are required.



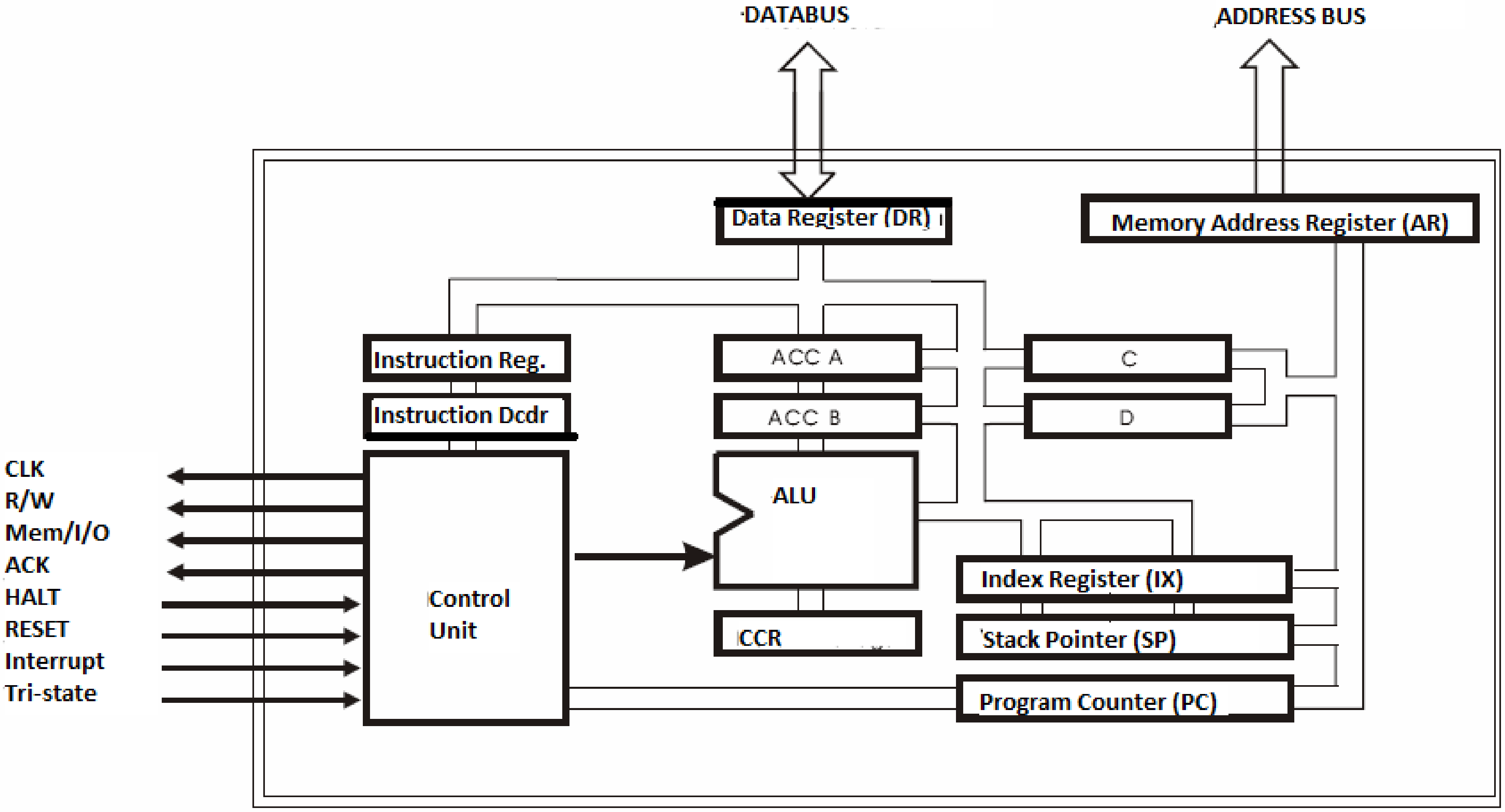
Instruction	1				2			
Fetch	🟦				🟥			
Decode		🟦				🟥		
Execute			🟦				🟥	
Write				🟦				🟥
Clock	1	2	3	4	5	6	7	8

Non-Pipelined

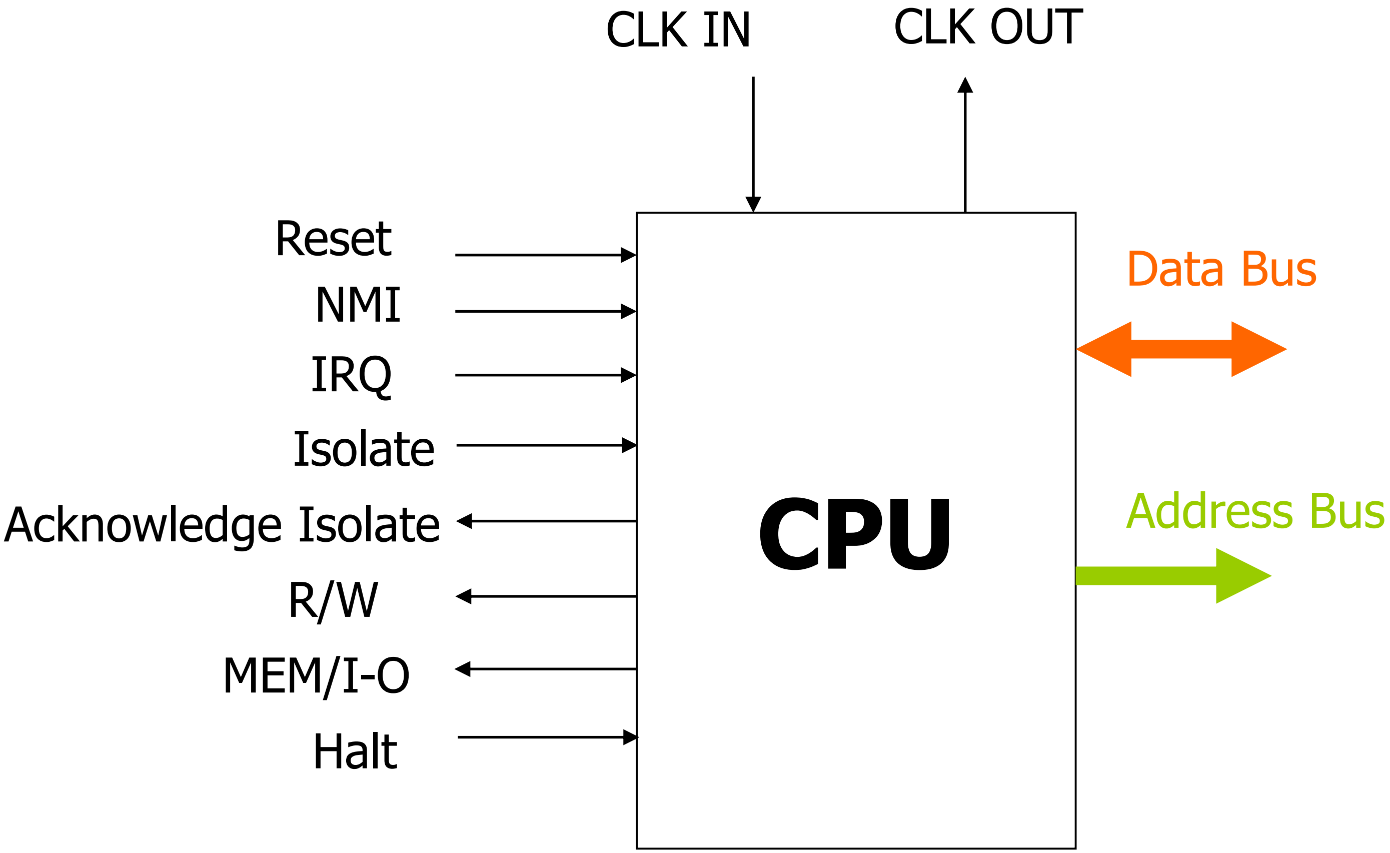
Instruction	1	2			
Fetch	🟦	🟥			
Decode		🟦	🟥		
Execute			🟦	🟥	
Write				🟦	🟥
Clock	1	2	3	4	5

Pipelined

Internal Structure of Educational CPU



External Pin Diagram of Educational CPU



Isolate: Isolation of the data and address busses

* NMI: Non-maskable interrupt
* IRQ: Interrupt request

Topics

- Central Processing Unit Structure
- Instruction Format

Instructions

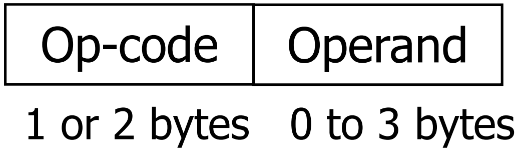
- Instructions are stored in the memory
- They are executed in a sequence
- They instruct the computer what to do
- The computer fetches the next instruction and decodes it (See Instruction Cycle).
- Each CPU has different set of instructions
- The instructions include an opcode and an address or multiple addresses

Memory

0	1	0	1	1	1	0	0
1	1	1	0	0	0	0	1
1	0	1	0	1	0	1	0
0	1	0	1	1	0	0	1
1	1	1	0	0	1	0	0
1	0	0	0	1	1	0	1

Instruction Format

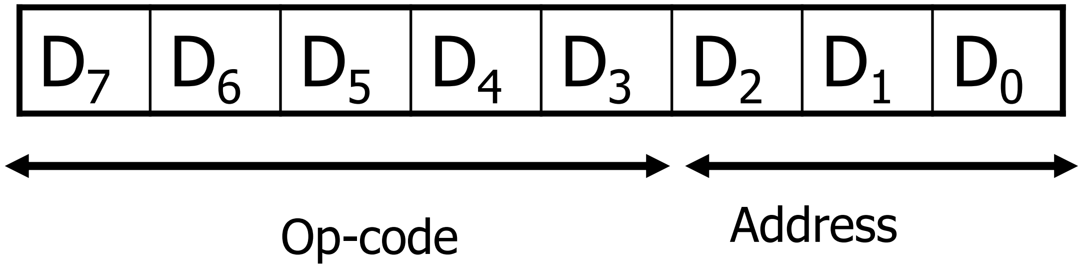
- A computer instruction is often divided into two parts
  - An opcode (Operation Code) that specifies the operation for that instruction
  - An address that specifies the registers and/or locations in memory to use for that operation



Op-code	Operand
1 or 2 bytes	0 to 3 bytes

Single Word, 1-Address Instruction Format

- Single word instruction example for 8-bit words

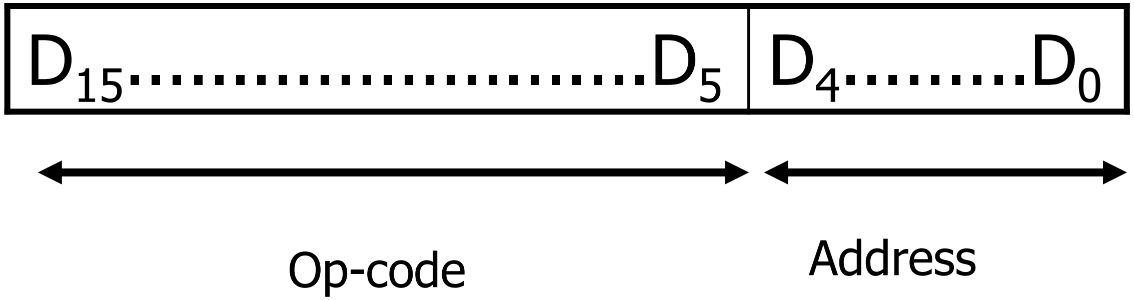


D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

32 possible op-codes
8 possible addresses

Plausible for register operations

- Example for 16-bit words



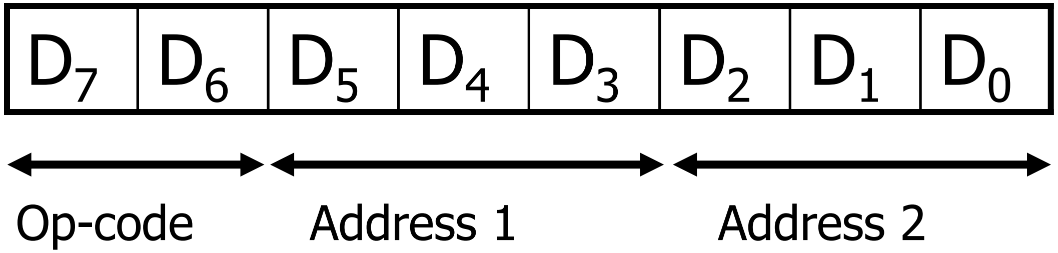
D_{15}.........................D_5	D_4.........D_0

2048 possible op-codes
32 possible adresses

More operation code and addressing possibilities for longer memory words

Single Word, 2-Address Instruction Format

- 2-Address instruction in an 8-bit word

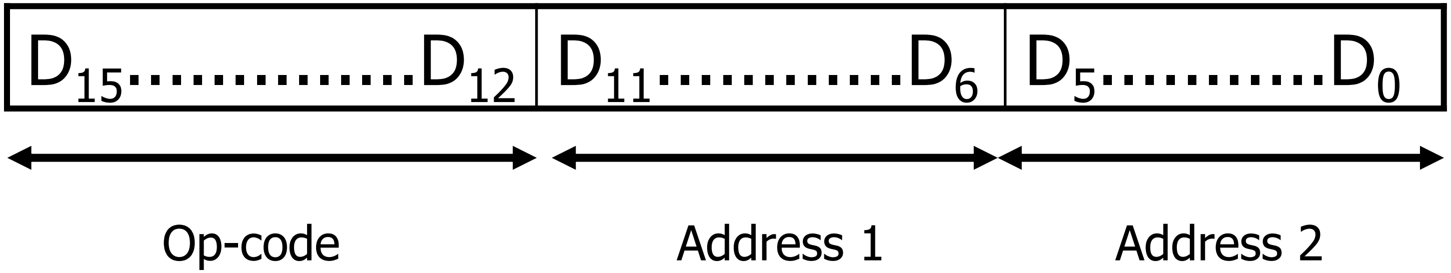


D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Op-code Address 1 Address 2

4 op-codes
8 Address 1
8 Address 2

- Example for 16-bit words



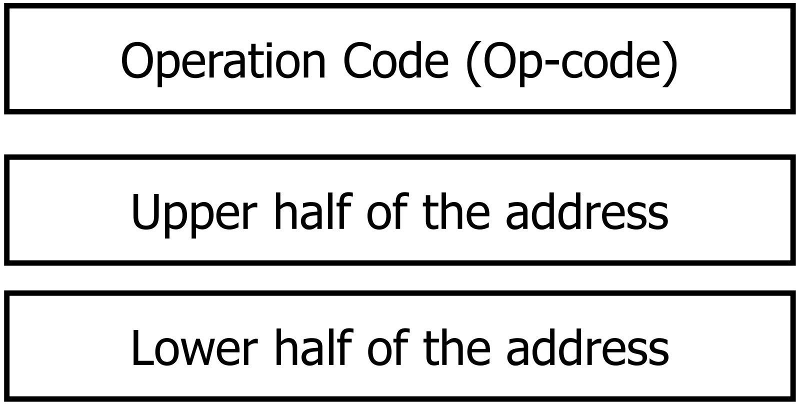
D_{15}.............D_{12}	D_{11}...........D_6	D_5..........D_0

Op-code Address 1 Address 2

16 op-codes
64 Address 1
64 Address 2

Multiple Words,
1-Address Instruction Format

- 1-Address instruction in multiple words



Operation Code (Op-code)

Upper half of the address

Lower half of the address

1. Octal

2. Octal

3. Octal

256 Instructions
65536 Address

Machine and Assembler Languages

- The binary format (1s and 0s) that codes the instructions for the computer is called the machine language

- The abbreviations (or mnemonics) designating the binary machine language are called assembler codes.

- The programming method with such codes form the assembler language

Instruction Sets

- Depending on the architecture, the instruction set is organized

- CISC (Complex Instruction Set Computer): Contains of a large number of instructions
  - More complex on hardware
  - Examples:
    - MC680x, MC68K, Intel40xx, Intel80xx,
    - Intel x86 (32bit and 64bit laptop, desktop, server systems),
    - IBM System-Z Mainframes and many other supercomputers

- RISC (Reduced Instruction Set Computer): Contains fewer but effective instructions
  - More complex on software
  - Examples:
    - ARM (iPad, iPhone, iPod, Blackberry, Android phones)
    - IBM Power PC (Wii, Xbox, Sony's PS)
    - Oracle (SUN) Sparc
    - Embedded applications
    - Single board computers

Instruction Set Differences

- Consider A = B + C in a high level language

- It might be translated into one instruction with a CISC architecture

```
add mem(B), mem(C), mem(A)
```

- Or four with a RISC architecture

```
load R1, B
load R2, C
add R3, R2, R1
store A, R3
```

Instruction Set Completeness

- A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable
- Computer design should have a sufficient number of four instruction categories
  - Transfer instructions: Data transfers among registers or registers and main memory
    - Load, Store, Transfer, Swap…
  - Arithmetic, logic, and shift instructions:
    - Add, Complement, Increment, circulate, shift, AND, Clear, Set…
  - Program control Instructions and instructions to check status conditions: Program sequencing and control
    - Compare, Branch, Jump, Go to and Return from Subroutine, Handle Interrupt service, Allow or Not-Allow interrupt requests
  - Input/Output Instructions:
    - Input data, Output data, Control peripherals, Status

Machine and Assembly Language Example:

Assembly language template

{Tag}   Operation,   Operand   : {Explanation}

```
START  LDAA, <$0080> : load ACCA the contents of memory address <$0080>
       ADDA, <$0081>  : Add ACCA the contents of memory address <$0081>
       STAA, <$0082>  : Store contents of ACCA to the memory address <$0082>
```

Address	Content (Machine language)
0010	00 20 00 80
0014	03 20 00 81
0018	01 20 00 82

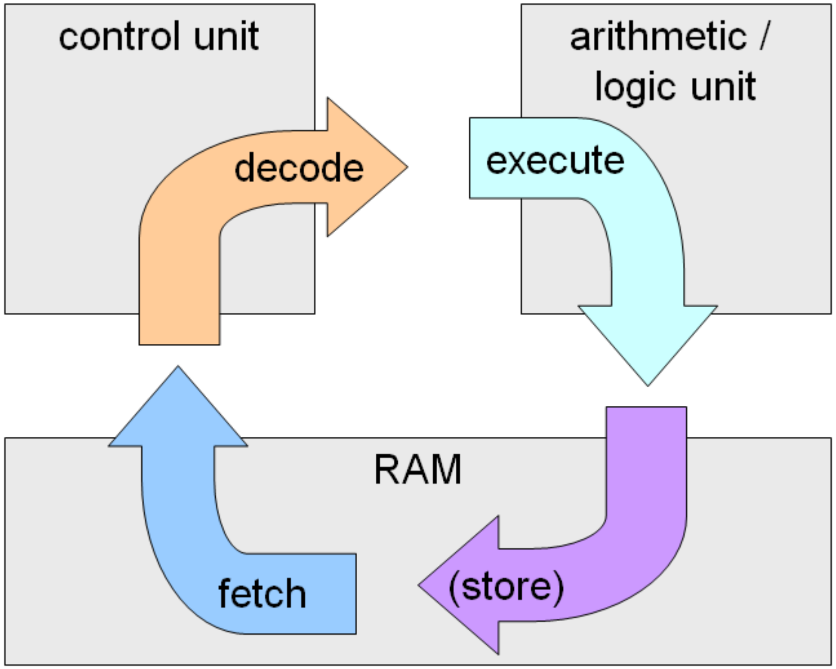
Execution of Machine Code

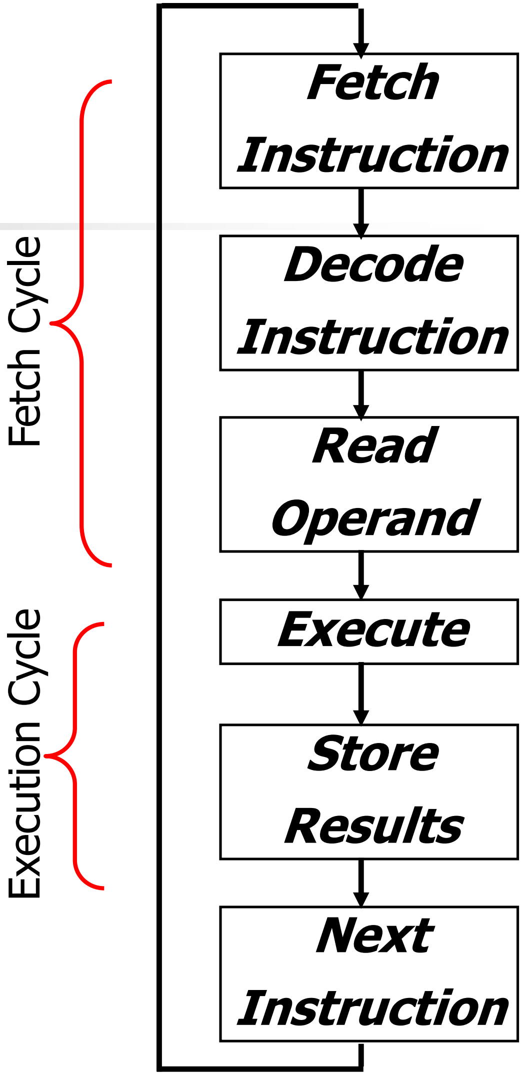
- Fetch-execute-cycle (on a von Neumann architecture)

```
initialize the program counter
repeat forever
  fetch the instruction pointed by the counter
  increment the counter
  decode the instruction
  execute the instruction
end repeat
```

Instruction Cycle

Fetch-Decode-Execute

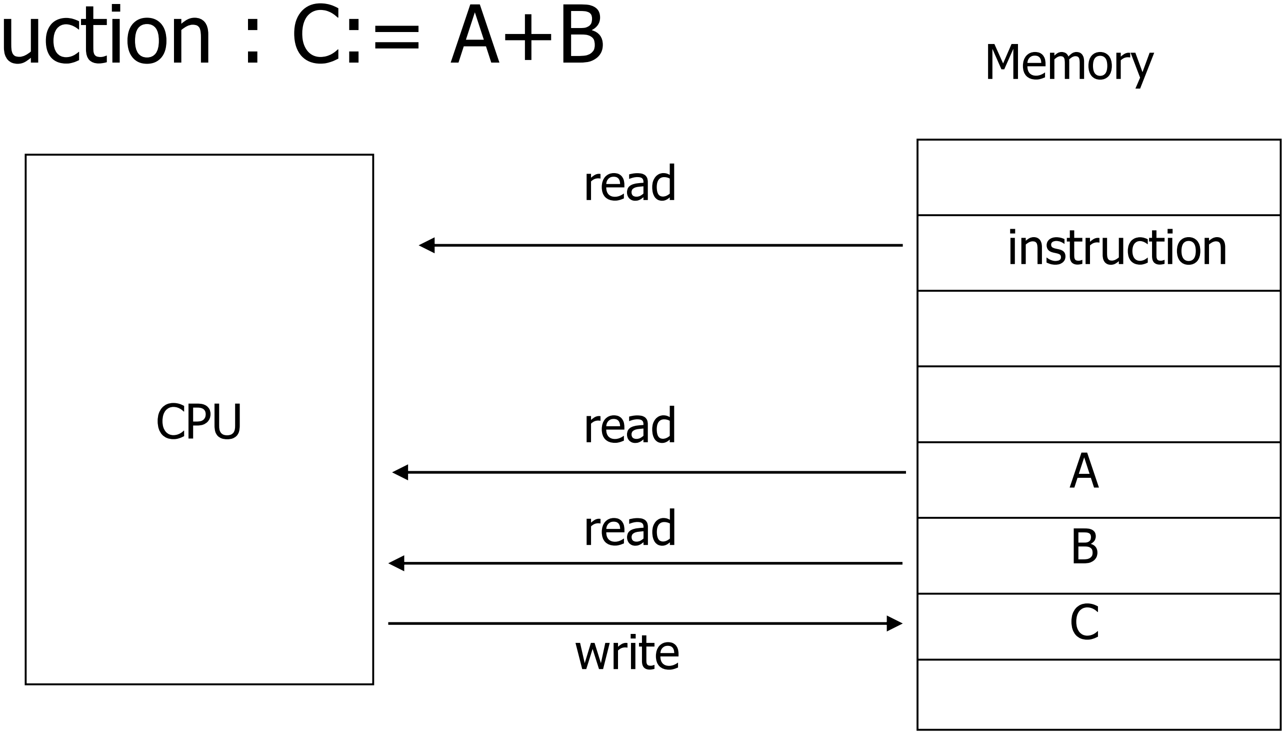




- Two-cycle process because both instructions and data are in memory
- Fetch
  - Decode or find instruction, load from memory into register and signal ALU
- Execute
  - Performs operation that instruction requires
  - Move/transform data

Example

Instruction : C:= A+B



Fetch-Decode-Execute

- An instruction at address 10 will be executed. The starting address of the program (10 in this example) must be written into the PC.
  - The 10 is transferred from the PC to the MAR and the memory is read at location 10
  - Since this is the first part of FETCH cycle, the data read is placed into the IR and the instruction decoder determines the operation
  - The CPU then increments the PC and MAR; and reads the content of 11 (**80**), which it places in the MAR.
  - The CPU is now ready to execute the LOAD instruction. It reads the content of 80 and places it in the ACC. The ACC contains now the contents of 80.

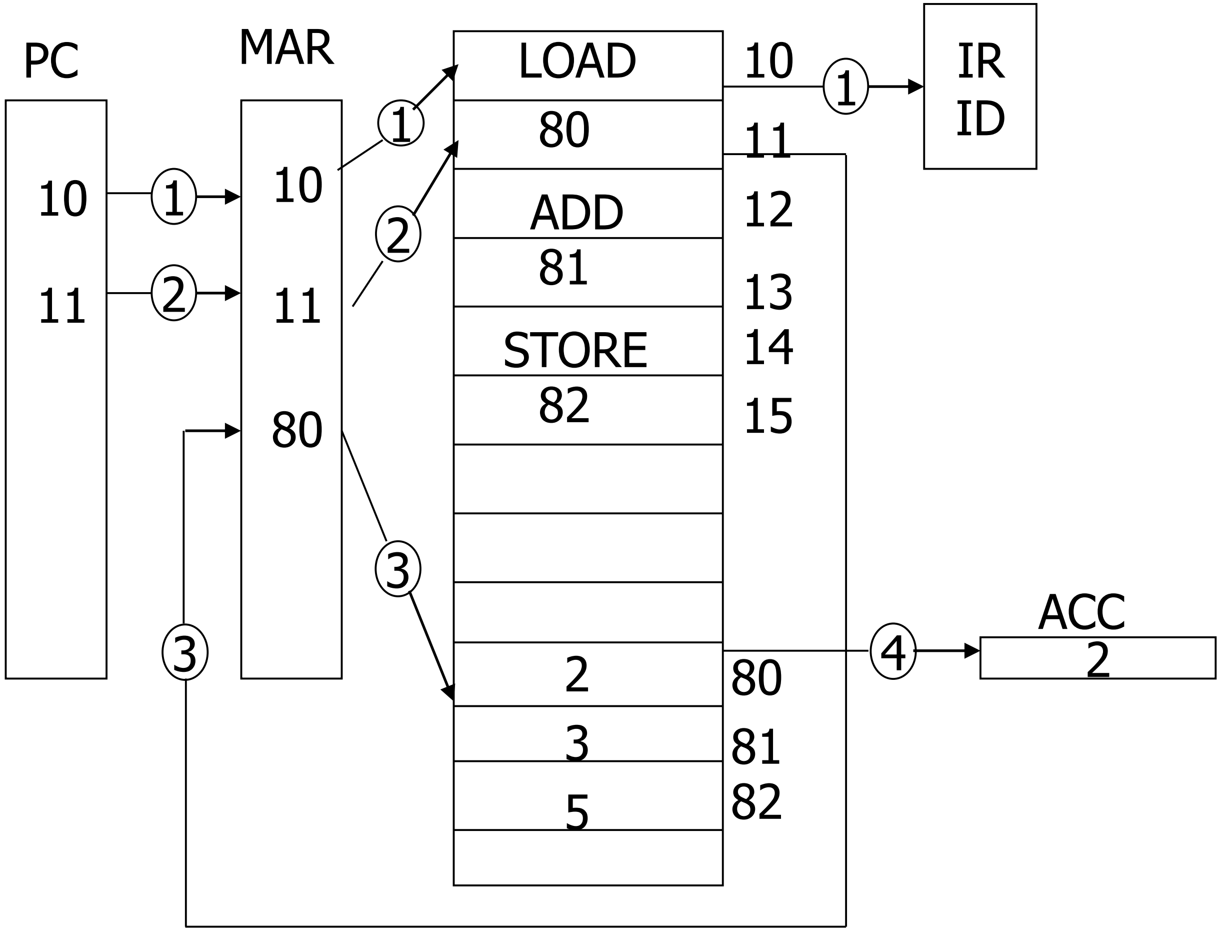
LOAD	10
80	11
ADD	12
81	13
STORE	14
82	15
2	80
3	81
5	82

Fetch-Decode-Execute

- The instruction execution is finished and the FETCH mode for the next instruction is entered.
  - The PC is again incremented to 12 and placed in the MAR.
  - The code for ADD is fetched from location 12 and decoded.
  - The CPU then increments the PC and 81 is read from memory and placed in the MAR.
  - The execute portion of the ADD instruction is entered. The contents of 81 (**3**) are read and added to the contents of the ACC using ALU. The results (**5**) are written to the ACC.
  - ...

LOAD	10
80	11
ADD	12
81	13
STORE	14
82	15
2	80
3	81
5	82

Example



PC	MAR	MDR	ACC
10	10	LOAD	X
11	11	80	X
11	80	2	2
12	12	ADD	2
13	13	81	2
13	81	3	5
14	14	STORE	5
15	15	82	5
15	82	5	5

8 memory reading
1 memory writing