BBM 105

Binary Representations And Von Neumann Architecture

Binary vs. Decimal

- Binary is a base two **system** which works just like our **decimal system**.
- Considering the **decimal** number **system**, it has a set of values which range from 0 to 9.
- The binary number system is base 2 and therefore requires only two digits, 0 and 1.

Why binary?

- These questions can be answered by a series of relevant questions!
  - How to store the values in hardware?
  - How to automatically perform arithmetic operations on numbers?
  - …
- The fundamental question is can we find out a physical material to stably maintain in different status?

How to store?

- Advancement in material science guarantees that binary status can be represented with no ambiguity.
- Silicon and many other semiconductor materials can present one of two status at any given time, and can retain a status for a long time.
- Positive or negative, +5 volt or -5 volt.
- Think about 2 status in electronic world, if not One then Zero, very simple to implement in electronic world.
- One the other hand, it is difficult, if not impossible, to find out a material to be able to maintain 10 different status stably.
- Generally speaking, the more status to maintain, the more difficult to find out such a material.

The simplest answer is

- Basically speaking, binary system simplifies information representation and information processing in electronic world.
- Binary number system is the easiest one to implement from the hardware point of view.
- The binary number system suits a computer extremely well, because it allows simple CPU and memory designs.
- So computers use binary numbers.

decimal to binary

- **Keep dividing by 2**

- **Ex 2 : 237_{10}**

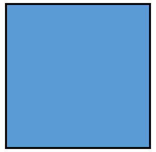
```
237 / 2 = 118    Remainder 1--------------------------------------------------|
118 / 2 = 59     Remainder 0-----------------------------------------------|   |
59 / 2  = 29     Remainder 1--------------------------------------------|   |   |
29 / 2 = 14      Remainder 1--------------------------------------------|   |   |
14 / 2 = 7       Remainder 0-----------------------------------------|   |   |   |
7 /  2 =3        Remainder 1--------------------------------------|   |   |   |   |
3 /  2 = 1       Remainder 1--------------------------------|    |   |  |   |   |   |
1 /  2 = 0       Remainder 1-----------------------------|   |    |  |   |   |   |   |

                                                  v  v  v  v  v  v  v  v
                                                   1  1  1  0  1  1  0  1
```

Hexadecimal <->binary

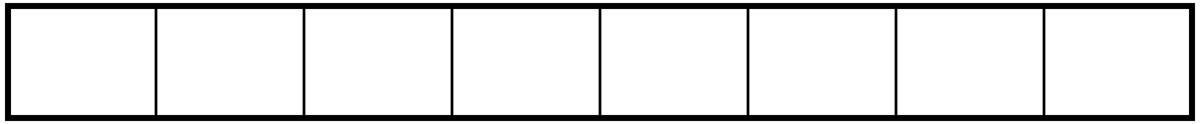
binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Hex	decimal	binary
10	16	10000
F0	240	11110000
FF	255	11111111

A ‘bit’ (from binary + digit) is the **smallest unit** of memory, also the unit of measurement of data information.



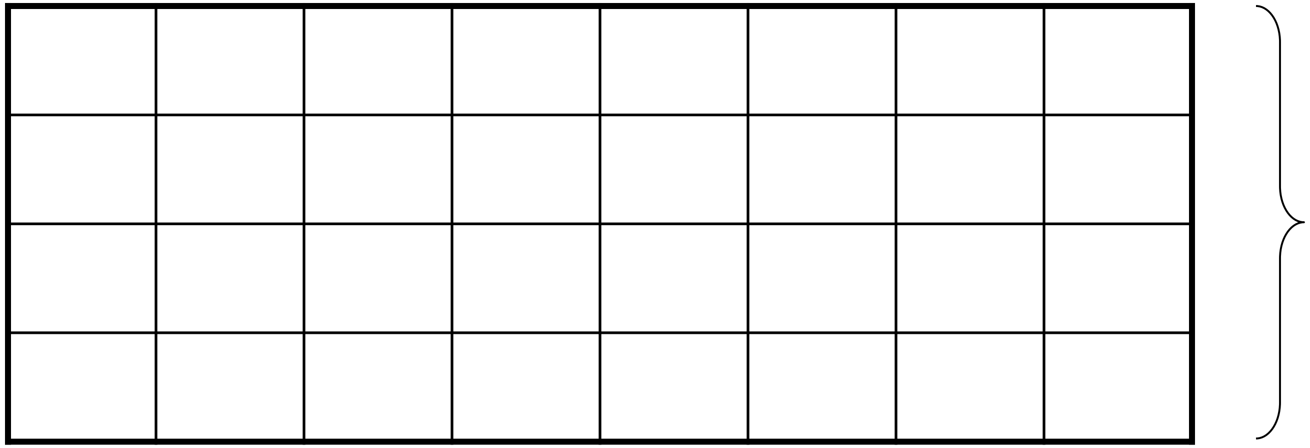
1 bit



1 byte

WHY 8 bit?

- To some extend, 8-bit is enough to represent all English characters and Arabic numbers. A byte used to be the basic unit to hold an individual character in a text document.

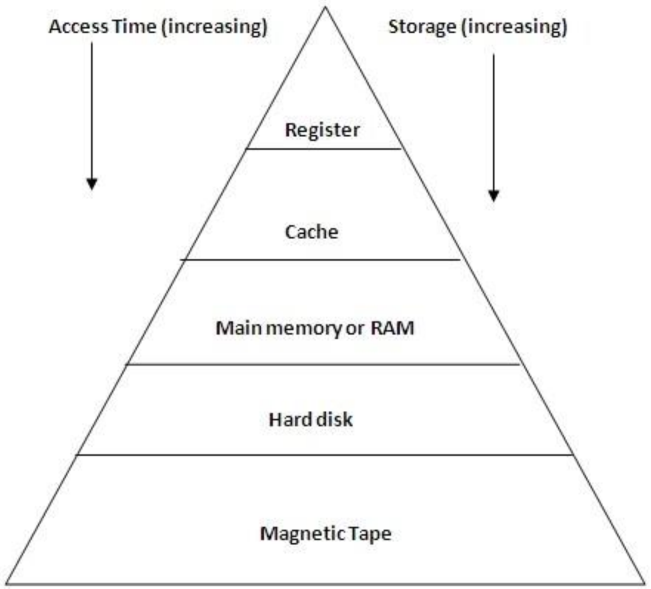


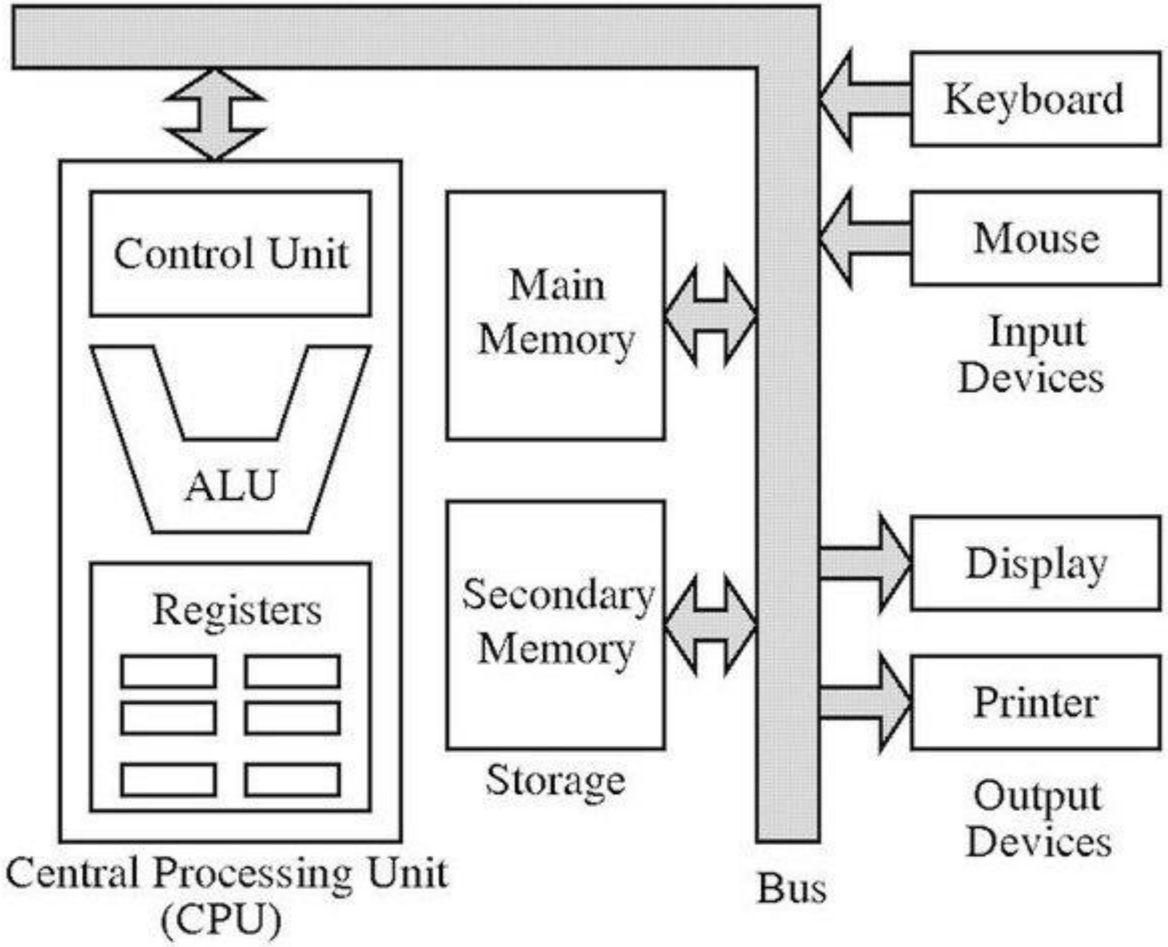
4 bytes = 1 word
System dependent.

Generally register sizes:
 in 32 bit system -> 4 bytes
 in 64 bir system -> 8 bytes
In windows legacy 2 bytes

- 1 bit
- 1 byte = 8 bits
- 1 KB = 2^{10} bytes = 1024 bytes (!=1000)
- 1 MB = 1 k k bytes = $2^{10} * 2^{10}$ bytes
- 1 GB = $2^{10} * 2^{10} * 2^{10}$ bytes
- 1 Terabyte = $2^{10} * 2^{10} * 2^{10} * 2^{10}$ bytes
- 1 petabyte = $2^{10} * 2^{10} * 2^{10} * 2^{10} * 2^{10}$ bytes (2 to the 50th power)
- 1 exabyte= 2^{60}
- 1 zettabyte = 2^{70}
- 1 yottabyte = 2^{80}

Von Neumann architecture (simplified)





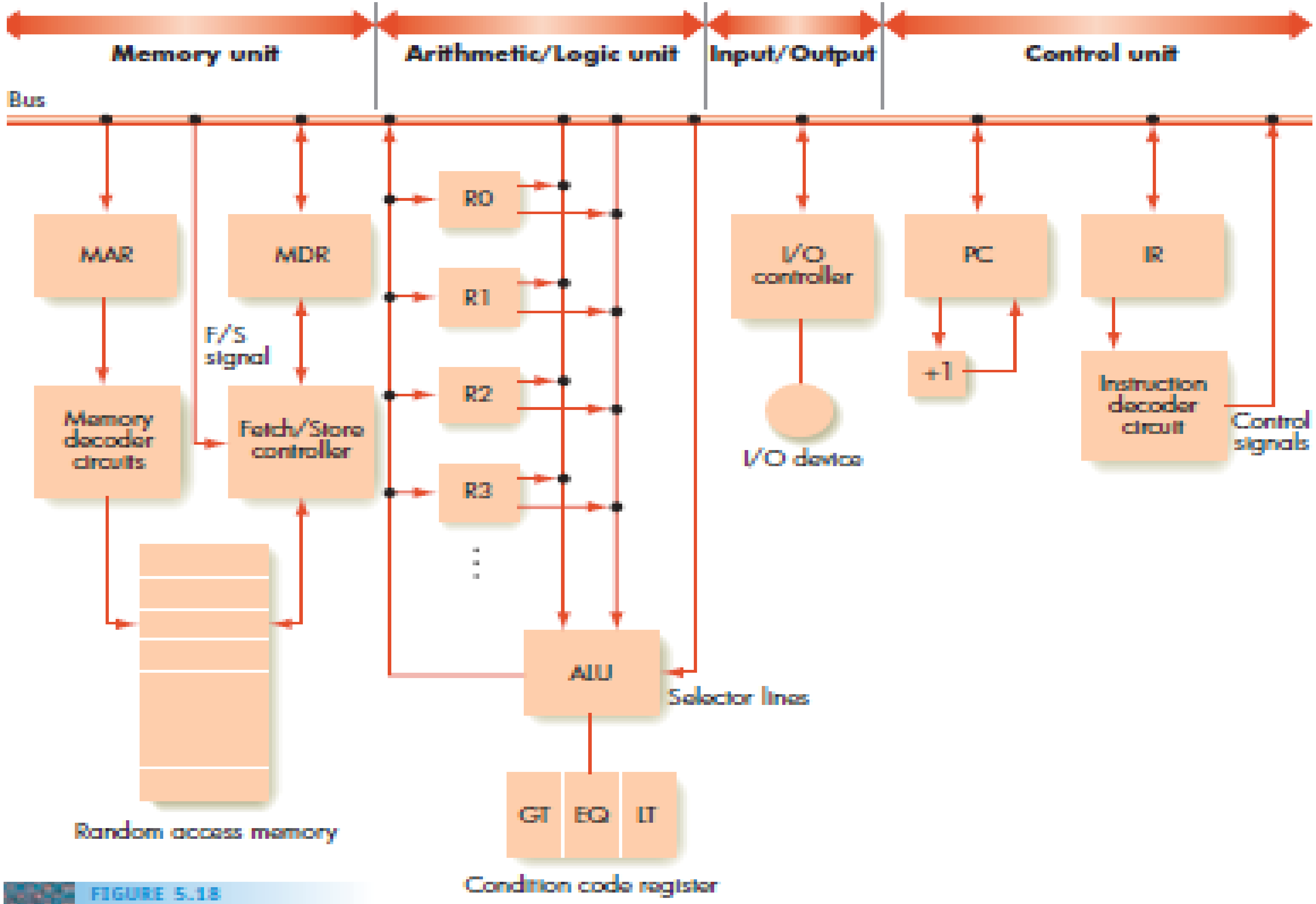
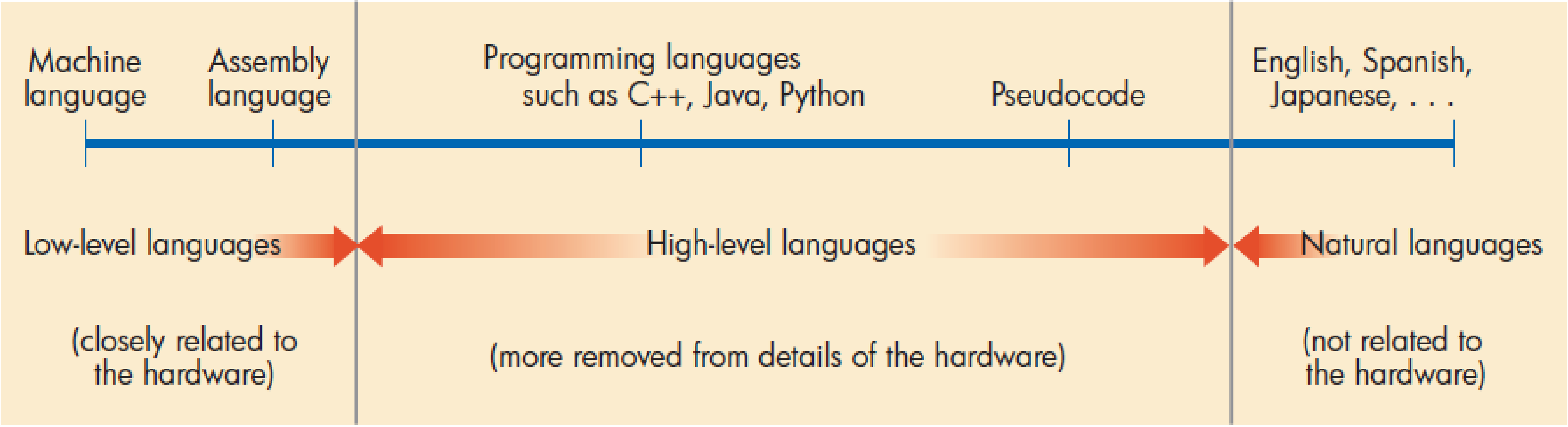


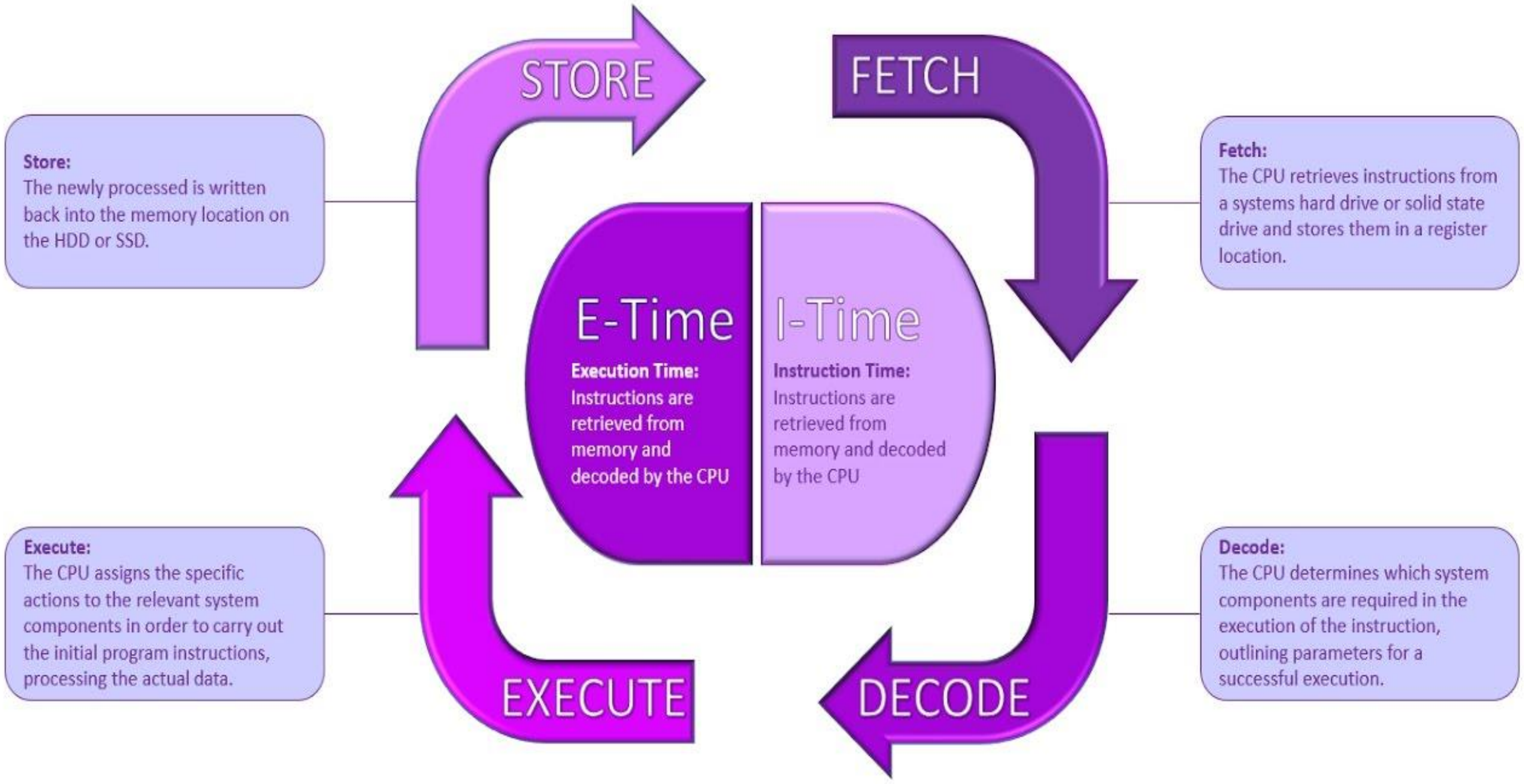
FIGURE 5.18

The Organization of a Von Neumann Computer

Continuum of Programming Languages



The Fetch-Execute Cycle



While we do not have a HALT instruction or a fatal erro

  Fetch phase

  Decode phase

  Execute phase

End of the loop

Basic computer commands

- Extremely limited set
  (nothing like the range of commands that we give to computers)
- Designed to be simple, easy to execute
  but can be built up to do things that we want computers to do (like spell-checks)

The power of a computer does not arise from complexity. Instead, the computer has the ability to perform simple operations at an extremely high rate of speed. These operations can be combined to provide the computer capabilities that you are familiar with.

The Architecture of Computer Hardware and Systems Software. By Irv Englander

Binary Op Code	Operation	Meaning
0000	LOAD X	CON(X) → R
0001	STORE X	R → CON(X)
0010	CLEAR X	0 → CON(X)
0011	ADD X	R + CON(X) → R
0100	INCREMENT X	CON(X) + 1 → CON(X)
0101	SUBTRACT X	R – CON(X) → R
0110	DECREMENT X	CON(X) – 1 → CON(X)
0111	COMPARE X	if CON(X) > R then GT = 1 else 0 if CON(X) = R then EQ = 1 else 0 if CON(X) < R then LT = 1 else 0
1000	JUMP X	Get the next instruction from memory location X.
1001	JUMPGT X	Get the next instruction from memory location X if GT = 1.
1010	JUMPEQ X	Get the next instruction from memory location X if EQ = 1.
1011	JUMPLT X	Get the next instruction from memory location X if LT = 1.
1100	JUMPNEQ X	Get the next instruction from memory location X if EQ = 0.
1101	IN X	Input an integer value from the standard input device and store into memory cell X.
1110	OUT X	Output, in decimal notation, the value stored in memory cell X.
1111	HALT	Stop program execution.

Machine Language Instructions

- A machine language instruction consists of:
  - Operation code, telling which operation to perform
  - Address field(s), telling the memory addresses of the values on which the operation works.
- Example: ADD X, Y

Opcode (8 bits)	Address 1 (16 bits)	Address 2 (16 bits)
00001001	0000000001100011	0000000001100100

Example

- Pseudo-code:
- Assuming variable:
  - A stored in memory cell 100, B stored in memory cell 150, C stored in memory cell 151
- Machine language (really in binary)
  - LOAD 150
  - ADD 151
  - STORE 100
  - or
  - (ADD 150, 151, 100)

How does this all work together?

- Program Execution:
  - PC is set to the address where the first program instruction is stored in memory.
  - Repeat until HALT instruction or fatal error

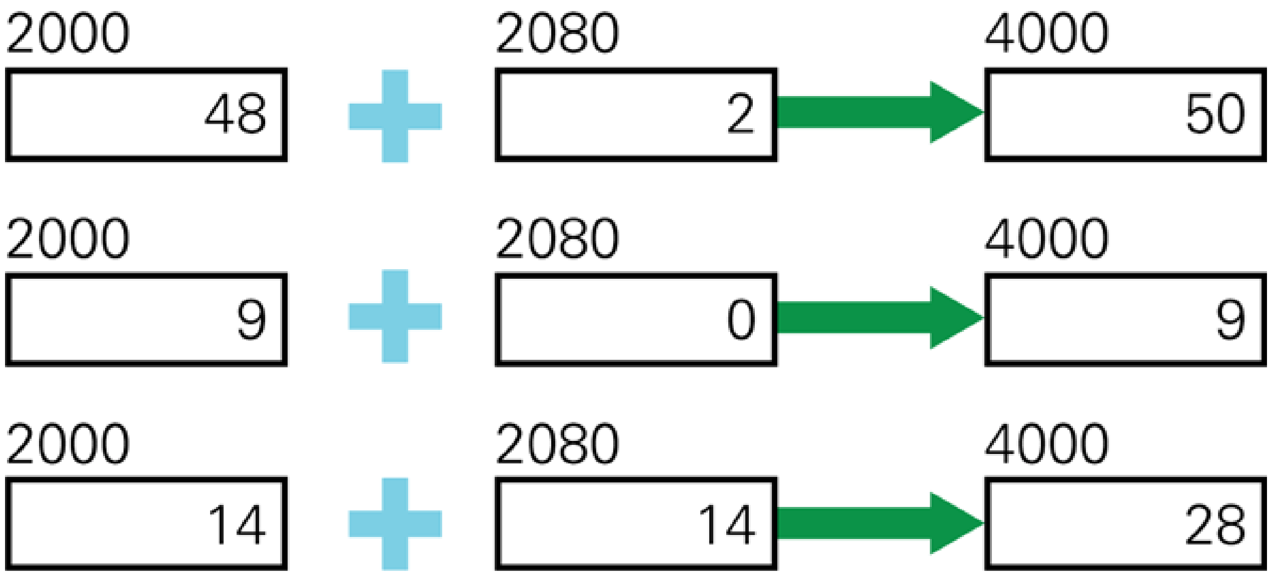
    Fetch instruction

    Decode instruction

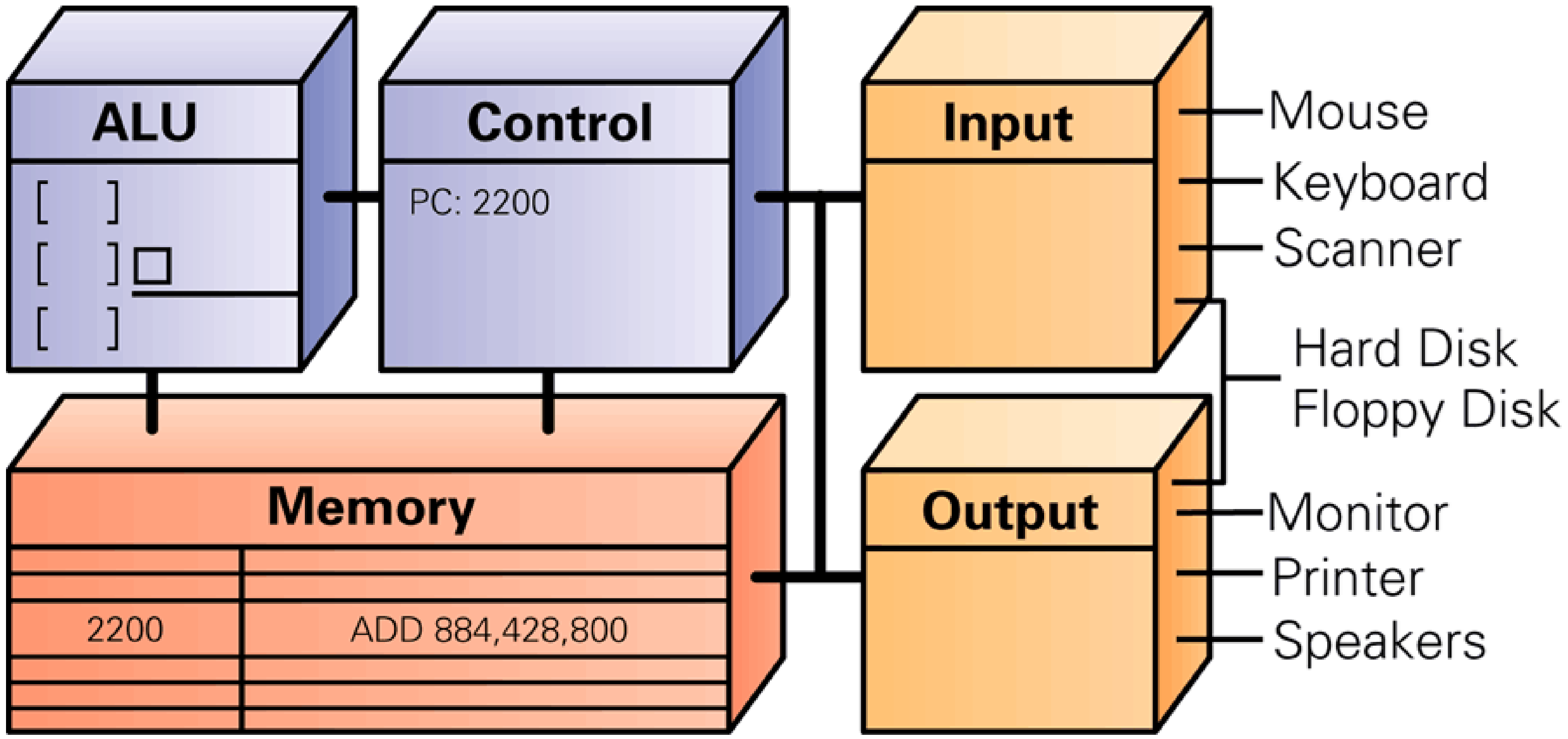
    Execute instruction

  End of loop

Illustration of a single `ADD 2000 2080 4000` instruction



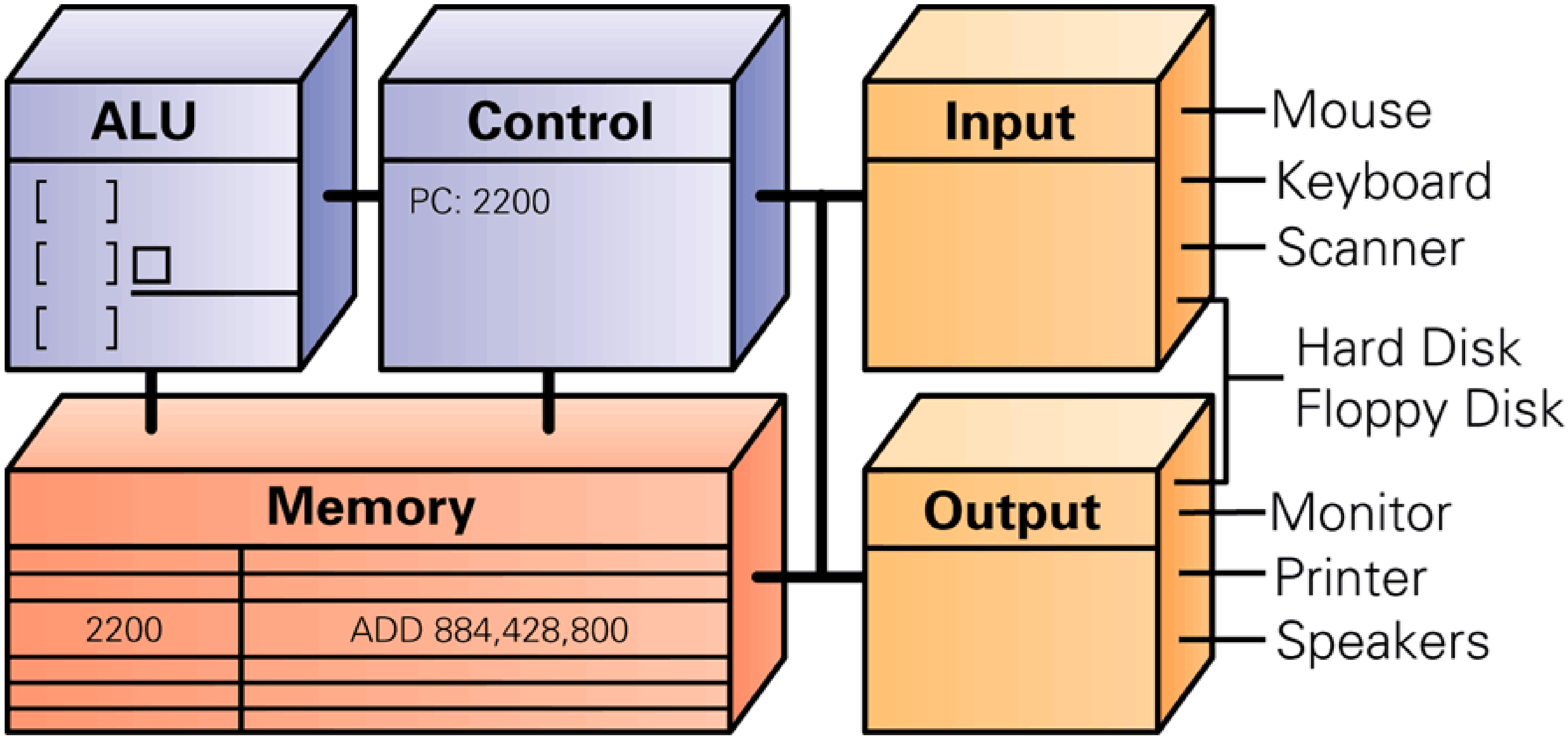
Computer before executing an ADD instruction.



The Program counter

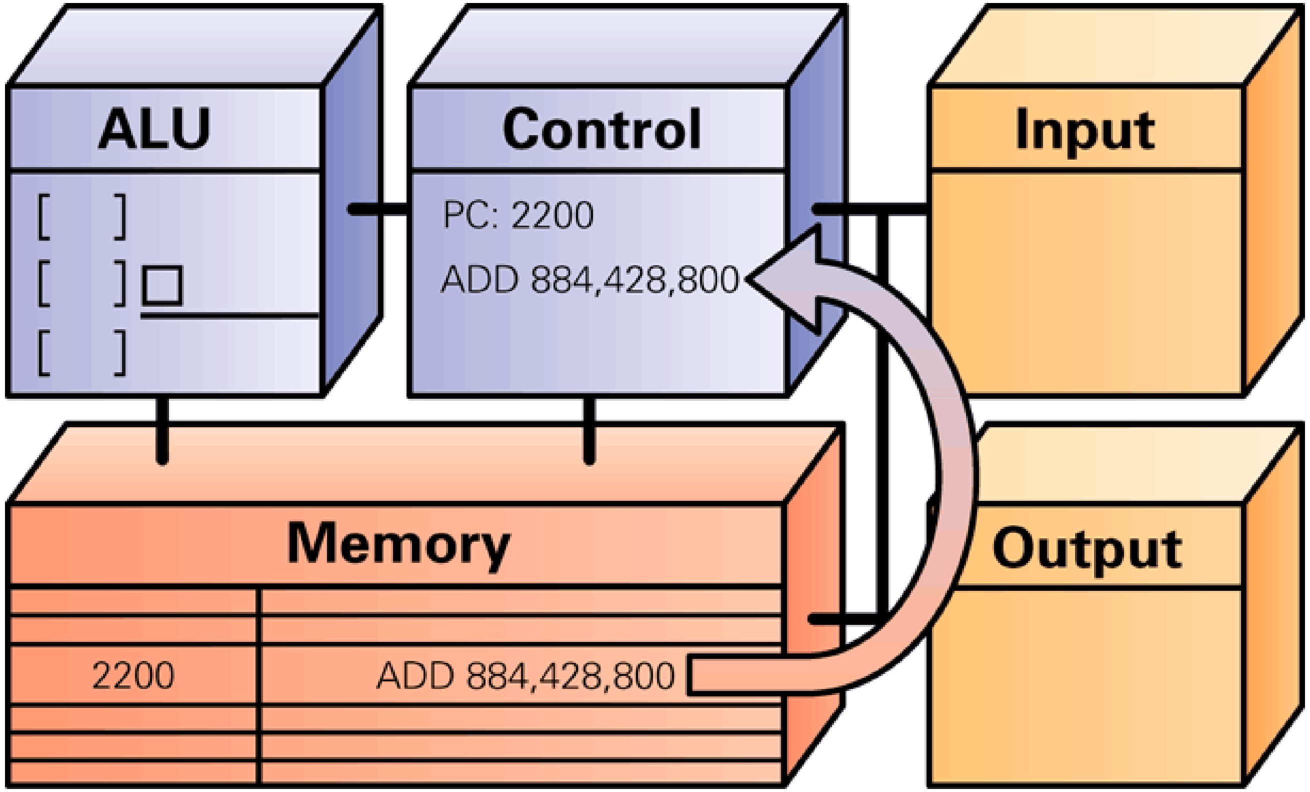
- This keeps track of what step needs to be executed next. It uses the address of the next instruction as its way of keeping track. This is called the program counter. (Here 'PC')
- Some instructions change the Program counter (branch and jump instructions)

Computer before executing an ADD instruction - again



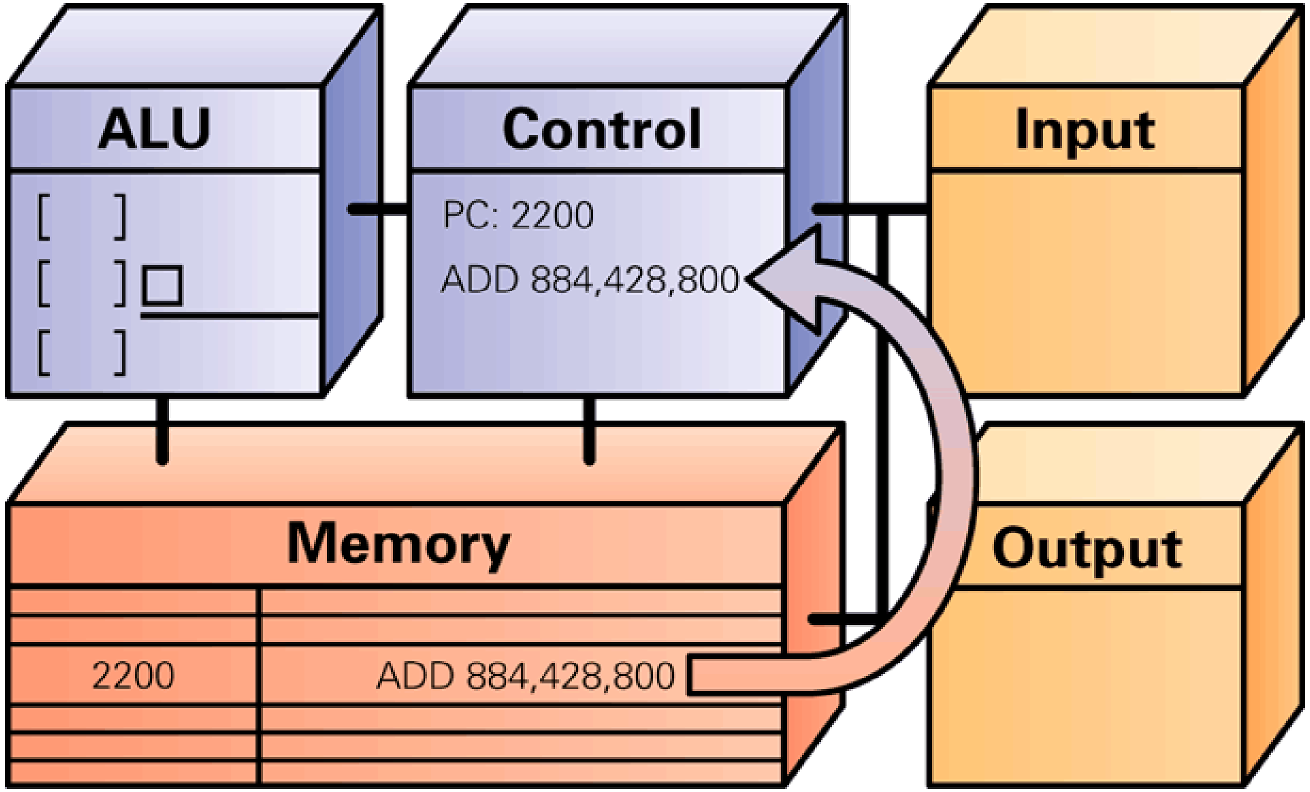
Instruction Fetch:

Move <u>instruction</u> from memory to the control unit.



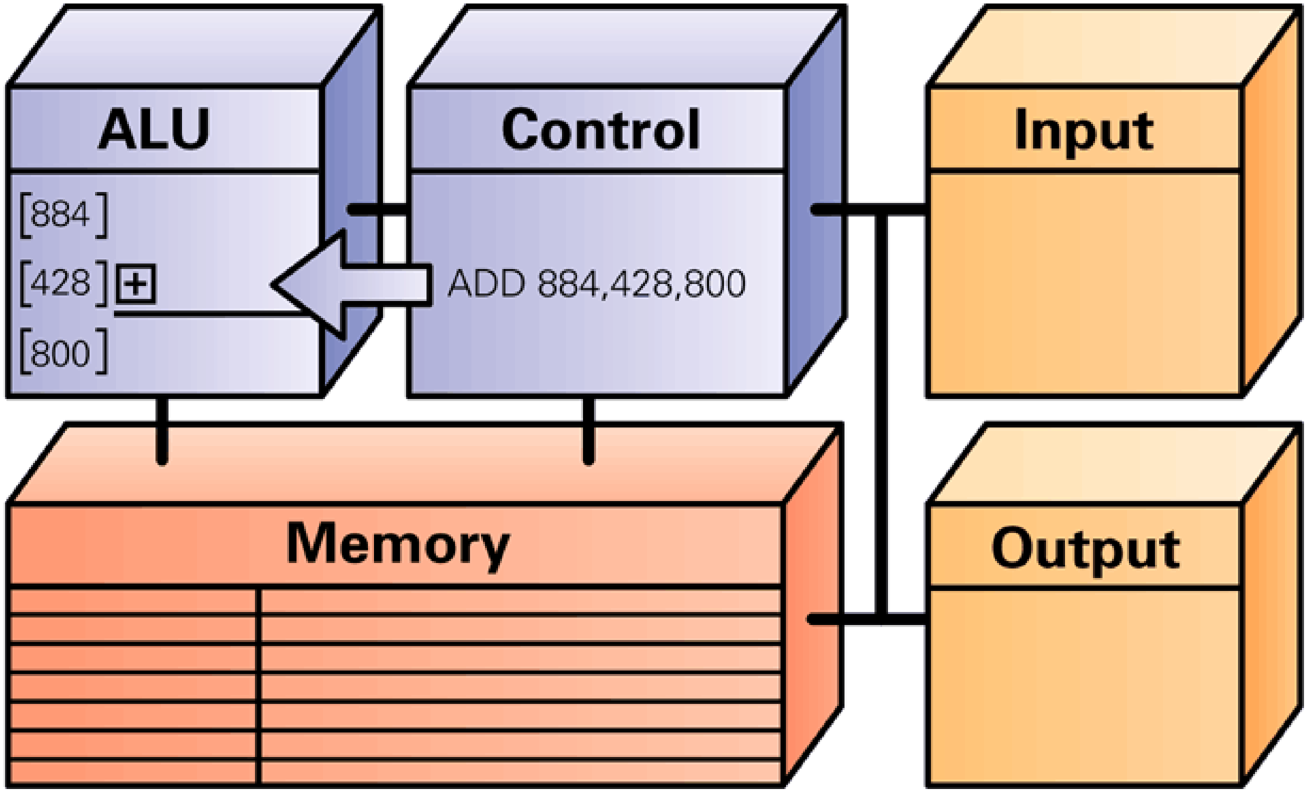
Instruction Fetch:

Move instruction from memory to the control unit.



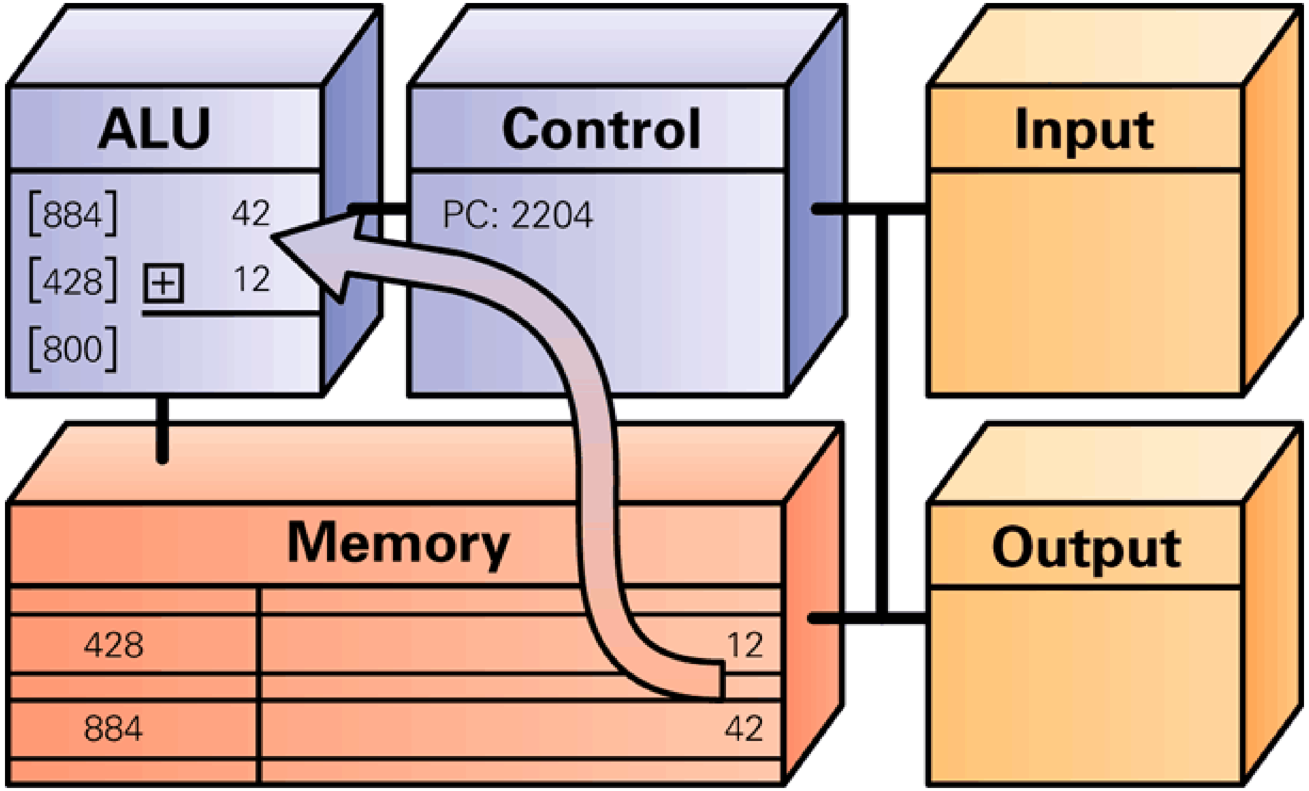
Instruction Decode:

Pull apart the instruction, set up the operation in the ALU, and compute the source and destination operand addresses.



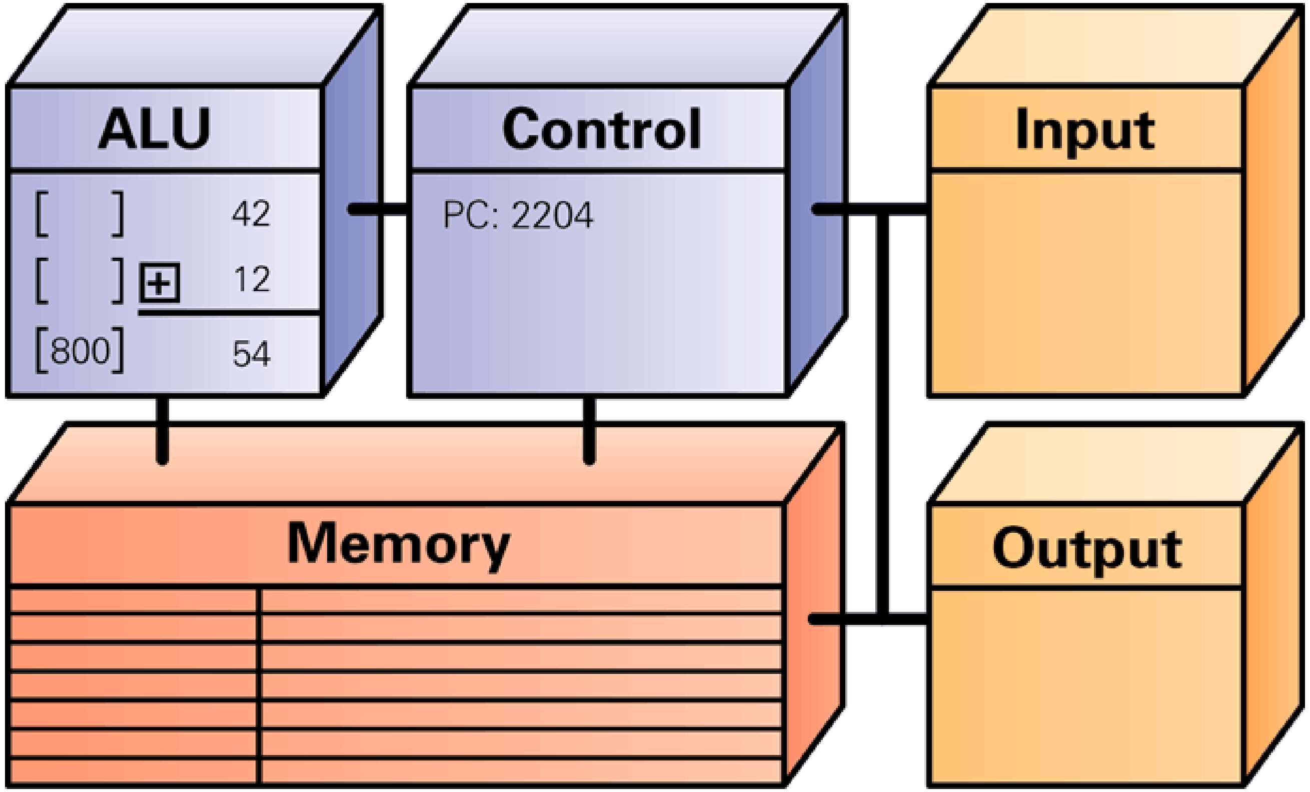
Data Fetch:

Move the operands from memory to the ALU.



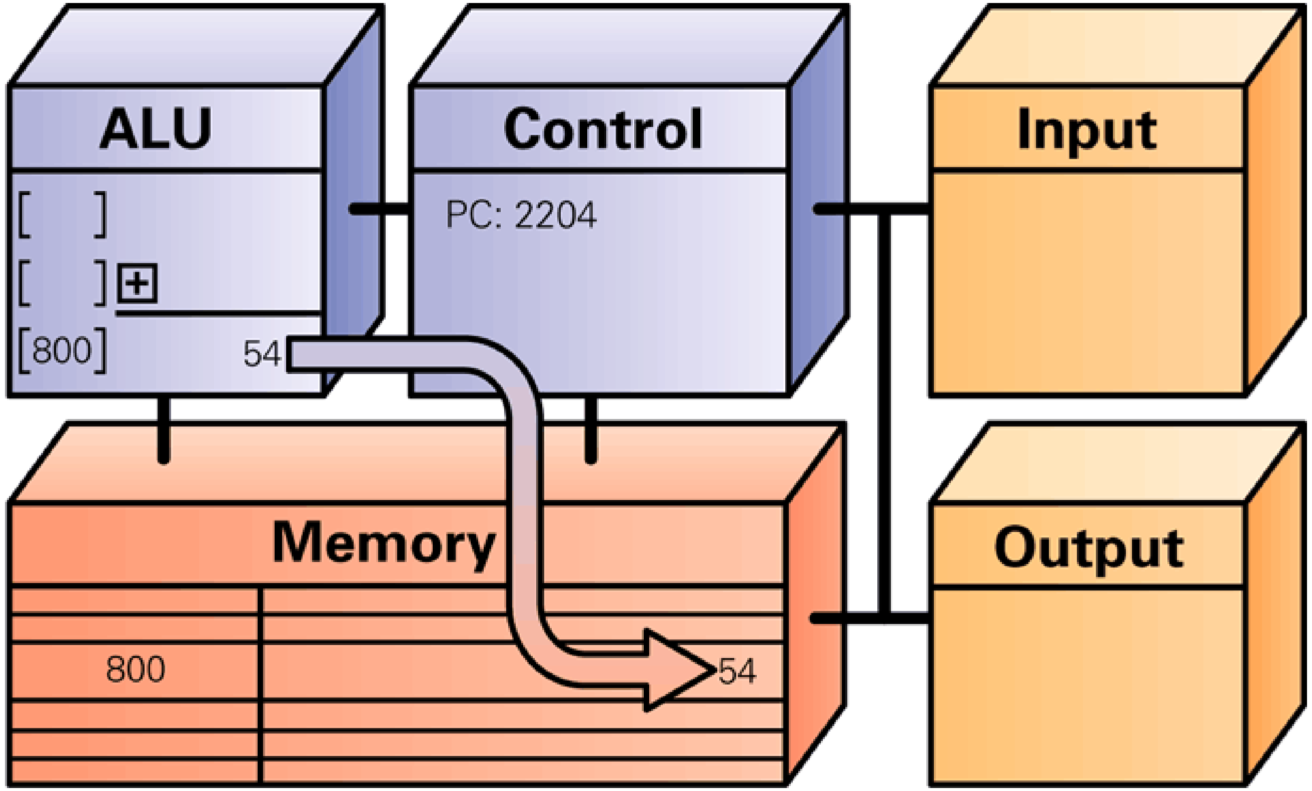
Execute:

Compute the result of the operation in the ALU.



Result Return:

Store the result from the ALU into the memory at the destination address.



That’s all folks.