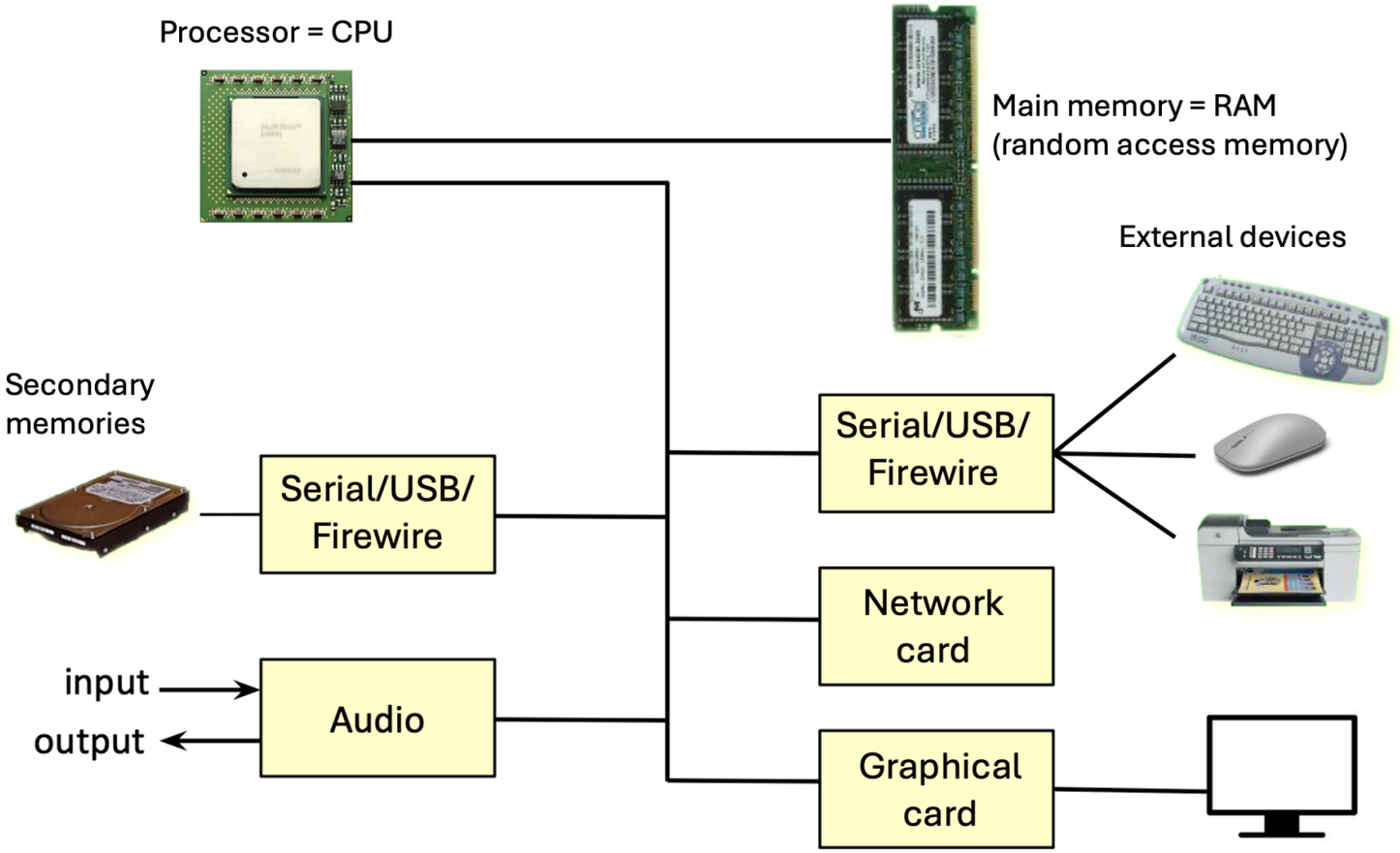
COMPUTER ARCHITECTURE AND SOFTWARE EXECUTION PROCESS

MICROPROCESSOR ARCHITECTURE

🎓 Bachelor in Artificial Intelligence, Data and Management Sciences
🏛 CentraleSupelec and ESSEC Business School - 2024/2025

Idir AIT SADOUNE
idir.aitsadoune@centralesupelec.fr

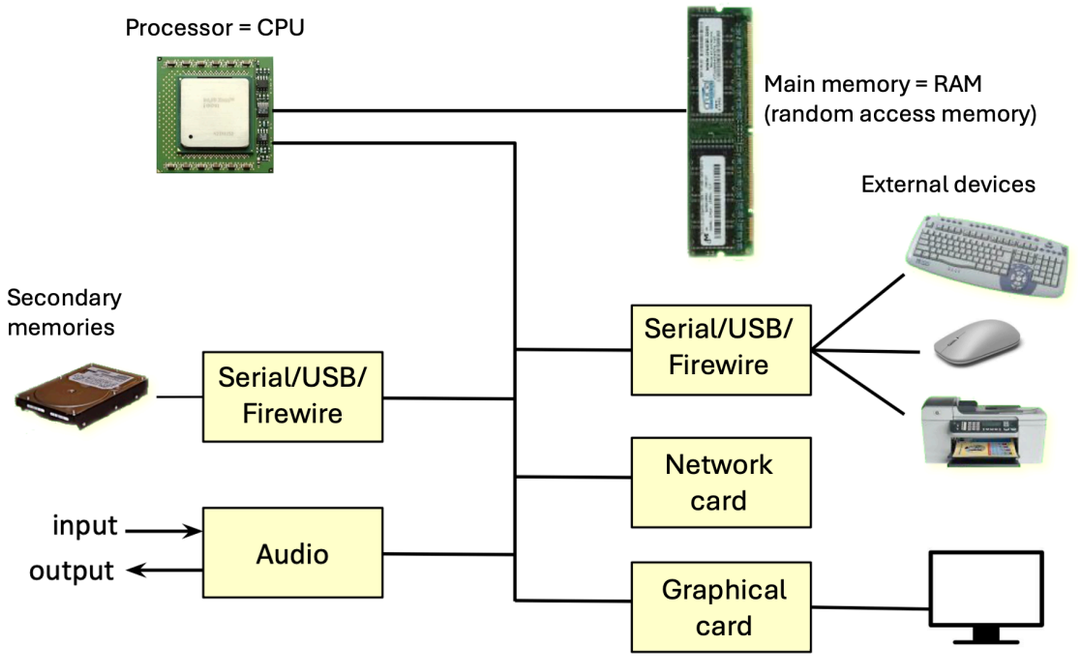
THE COMPUTER COMPONENTS



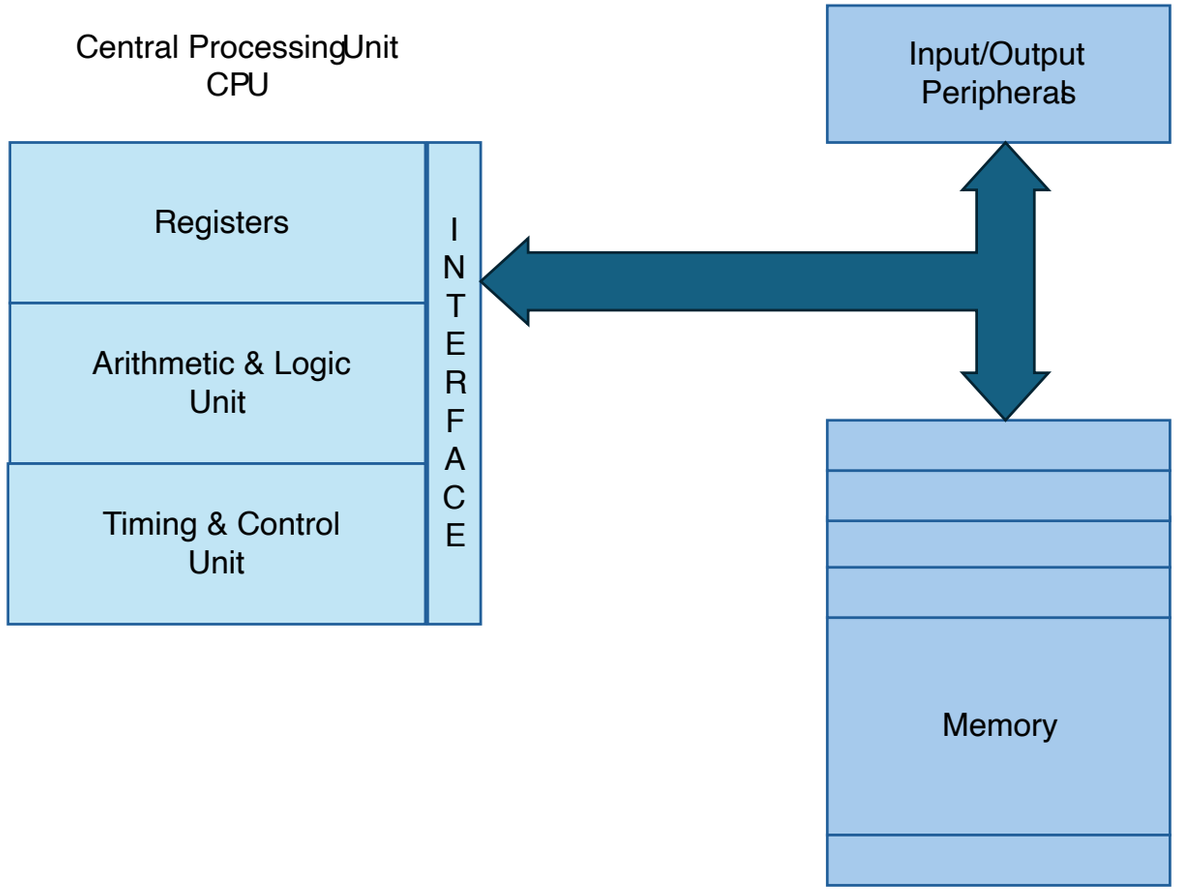
THE COMPUTER COMPONENTS

- What can a computer (processor) do?
  - **copy values** between storage units
  - **perform logical/arithmetic operations** between stored values
  - **move within the program**, possibly conditionally

- A **processor** executes a very low level language (**Machine Language**)
  - the instructions of this kind of language execute elementary operations.
  - machine instructions control electronic/logic circuits.



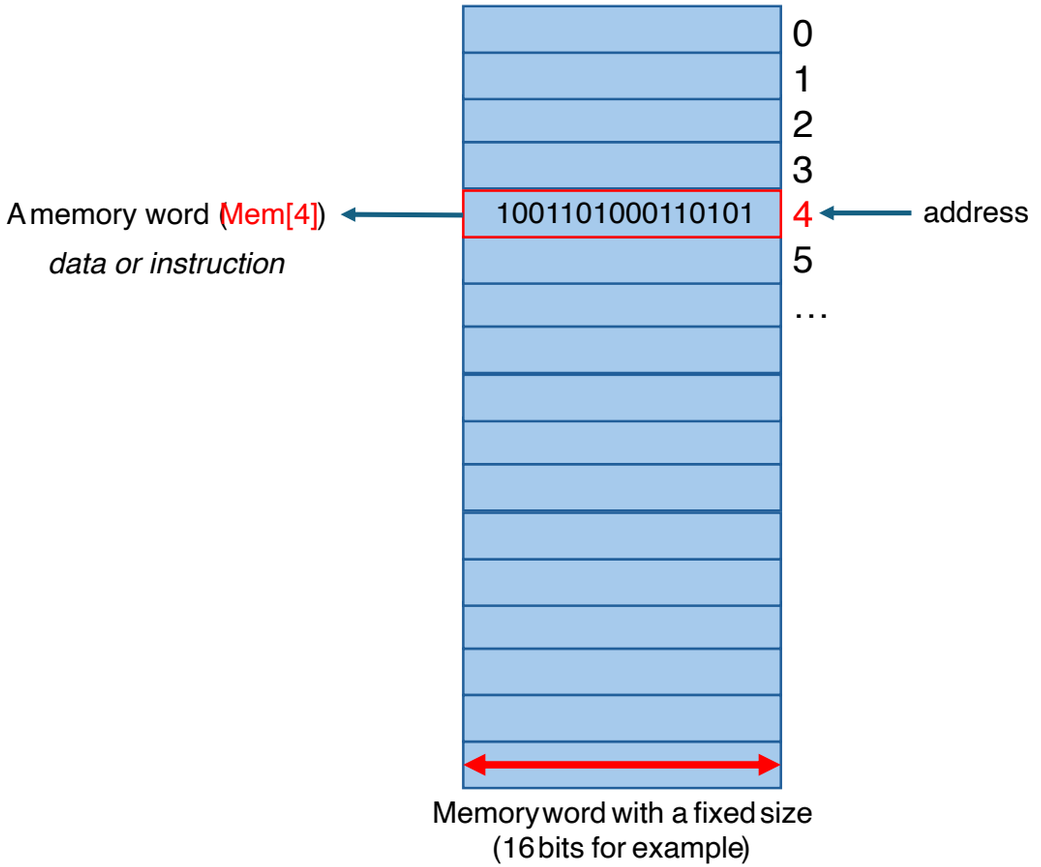
TYPICAL STRUCTURE OF CPU



TYPICAL STRUCTURE OF CPU

MEMORY UNIT

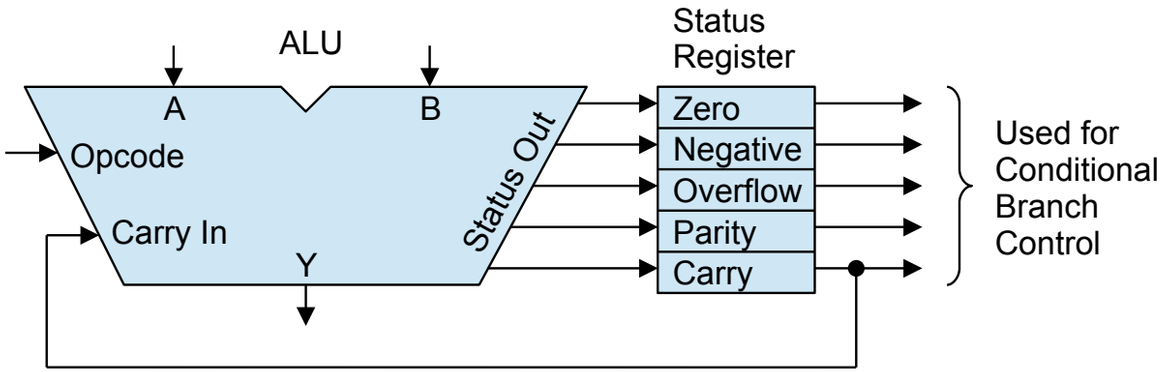
- **Computer memory** stores information, such as data and programs, for immediate use in the computer.
- The term memory is often synonymous with the terms **RAM**, **main memory**, or **primary storage**.



TYPICAL STRUCTURE OF CPU

ARITHMETIC AND LOGIC UNIT (ALU)

- The **Arithmetic and Logic Unit** (**ALU**) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers.

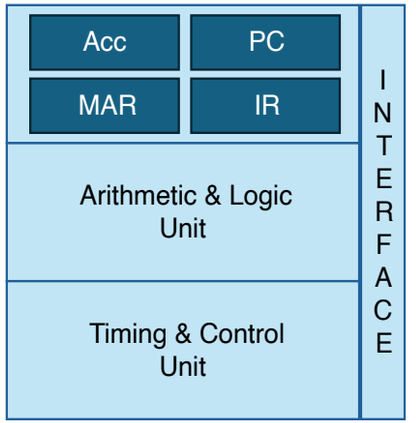


- The opcode input conveys to the **ALU** an operation selection code that specifies the desired arithmetic or logic operation to be performed by the **ALU**.
  - add, sub, and, or, ...
  - generally, an **ALU** opcode is not the same as a machine language instruction

TYPICAL STRUCTURE OF CPU

REGISTER SECTION

- **A processor register** is a quickly accessible memory available to a computer's processor.
- **Registers** are measured by the number of bits they can hold
  - for example, an 8-bit register, 16-bit register, or more.
- A processor often contains several kinds of registers:
  - Accumulator → stores intermediate arithmetic and logic unit results.
  - Program Counter → indicates where a processor is in its program sequence.
  - Istruction Register → holds the instruction currently being executed or decoded (each instruction to be executed is loaded into the IR register)
  - Memory Address Register → either stores the memory address from which data will be fetched or the address to which data will be sent and stored.

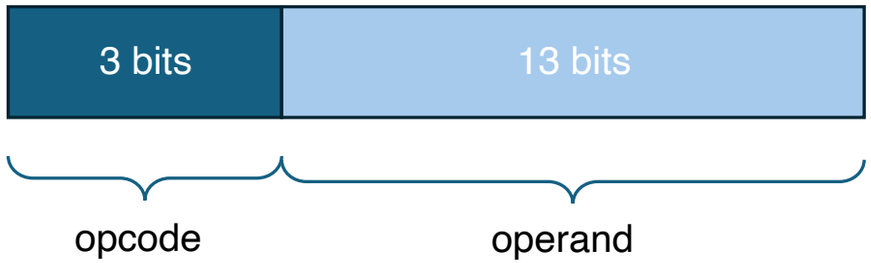


MACHINE LANGUAGE (INSTRUCTIONS)

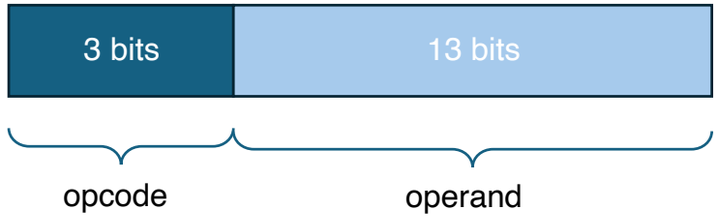
- **Machine language** is built up from **instructions**.
- **Instruction** → an elementary operation that a processor can perform.

- A given instruction may specify:
  - opcode → the instruction to be performed
  - any explicit operands → registers, constant values, addressing modes used to access memory

- **Example**:
  - `LOAD addr` → Acc := Mem[addr]
  - `LOAD val` → Acc := val
  - `STORE addr` → Mem[addr] := Acc
  - `ADD addr` → Acc := Acc + Mem[addr]
  - `ADD val` → Acc := Acc + val
  - `SUB addr` → Acc := Acc - Mem[addr]
  - `SUB val` → Acc := Acc - val



EXAMPLES



- OPCODE OPERAND
  $xxx \;\; xxxxxxxxxxxxx$
- LOAD 25 $\rightarrow$ Acc := Mem[25]
  $000 \;\; 0000000011001 = (0019)_{16}$
- LOAD 12 $\rightarrow$ Acc := 12
  $100 \;\; 0000000001100 = (800C)_{16}$
- STORE 54 $\rightarrow$ Mem[54] := Acc
- ADD 41 $\rightarrow$ Acc := Acc + Mem[41]
  $010 \;\; 0000000101001 = (4029)_{16}$
- ADD 10 $\rightarrow$ Acc := Acc + 10
  $110 \;\; 0000000001010 = (C00A)_{16}$
- SUB 14 $\rightarrow$ Acc := Acc - Mem[14]
  $011 \;\; 0000000001110 = (600E)_{16}$
- SUB 112 $\rightarrow$ Acc := Acc - 112

EXAMPLES

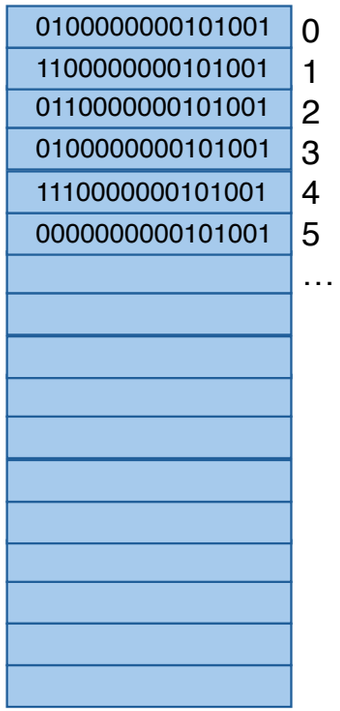
```
1 int a = 3;
2 int b = 0;
3
4 b = a + 10;
```

```
1      LOAD a
2      ADD 10
3      STORE b
4      ...
5  @a: memval 3
6      ...
7  @b: memval 0
```

```
1 0000: 0019
2 0001: C00A
3 0002: 2036
4       ...
5 0025: 0003
6       ...
7 0054: 0000
```

PROGRAM EXECUTION

- A program's instructions are stored at contiguous addresses
- The processor reads the program instructions from memory one by one.
- The address of the next instruction to be read is stored in the **Program Counter** (**PC**) register.

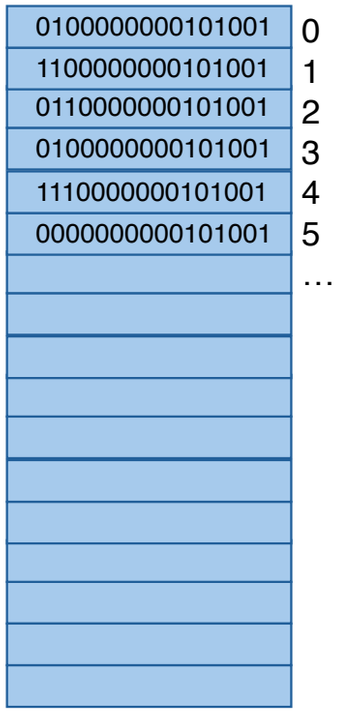


PROGRAM EXECUTION

1. **Fetch** $\rightarrow$ the instruction contained at the address indicated by the **PC** register is copied into the **Instruction Register** (**IR**)
   - The value contained in the **PC** register is incremented.

2. **Decode** $\rightarrow$ the processor "understands" the instruction to be executed.

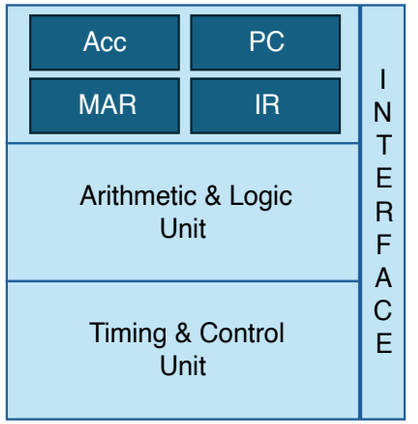
3. **Execute** $\rightarrow$ The instruction is executed by the processor.

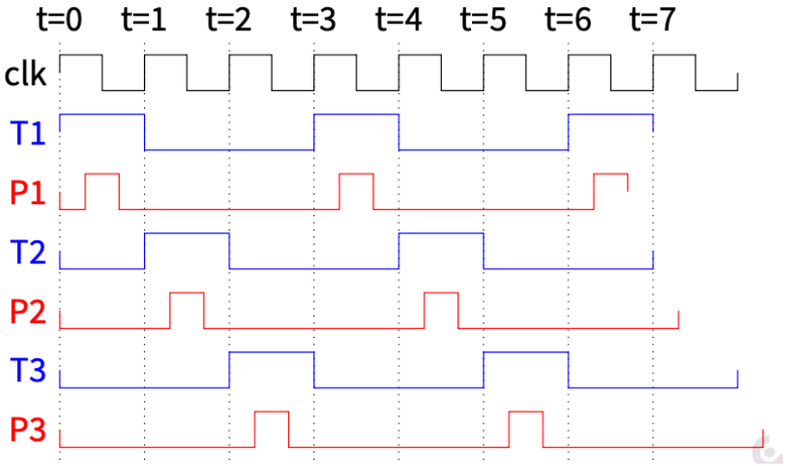


IMPLEMENTING THE CONTROL UNIT

- Creating a Time Table
  - One line for each instruction
  - One column for each clock time
  - For each time, the data path signals to be activated

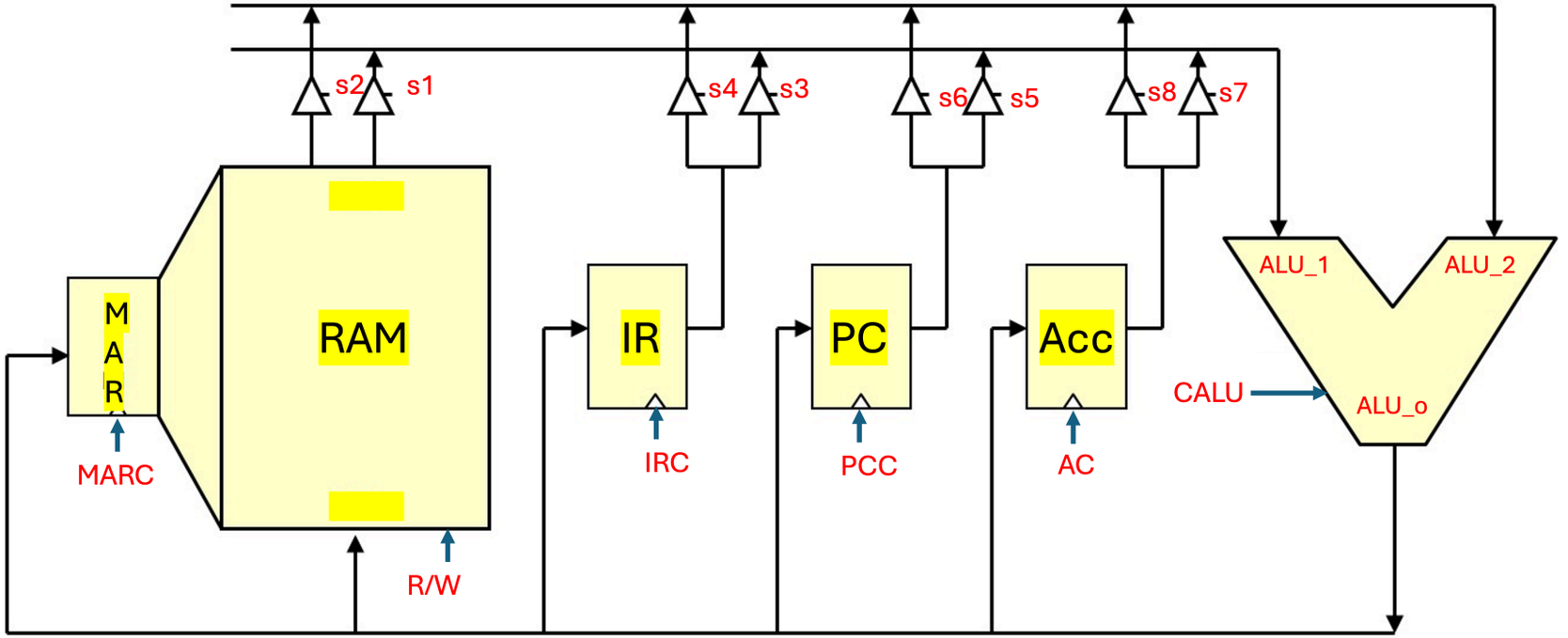
- Derivation of a set of logical equations
  - An equation for each signal in the data path
  - An equation indicates when a signal should be activated
  - An equation depends on the control unit inputs and the **T, P signals**





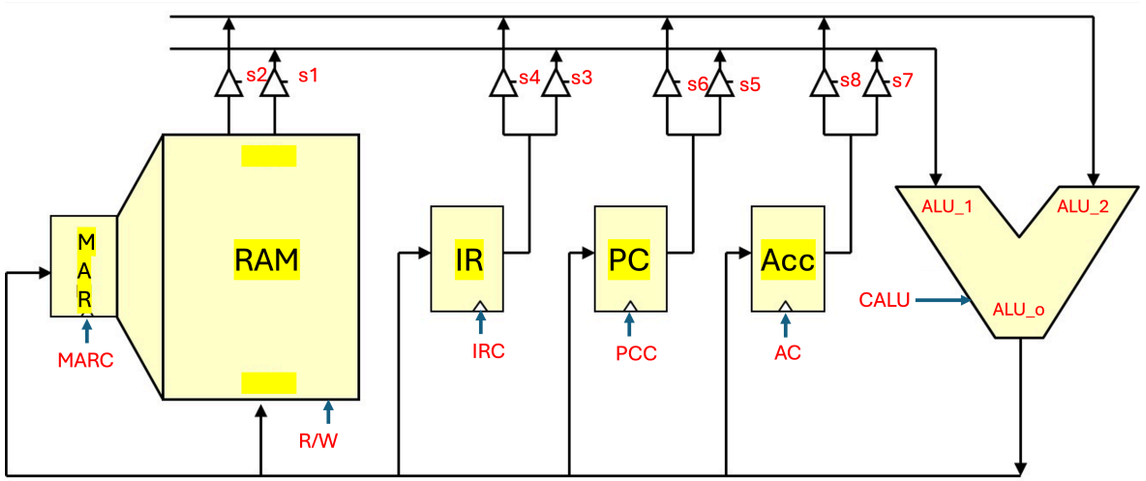
- $T_i \rightarrow$ setup time
- $P_i \rightarrow$ memorization time

DATA PATH



CALU=	00 ($ALU_o = ALU_1$),	01 ($ALU_o = ALU_1 + ALU_2$),	10 ($ALU_o = ALU_1 + 1$),	11 ($ALU_o = ALU_1 - ALU_2$)
R/W=	0 (READ),	1 (WRITE)		

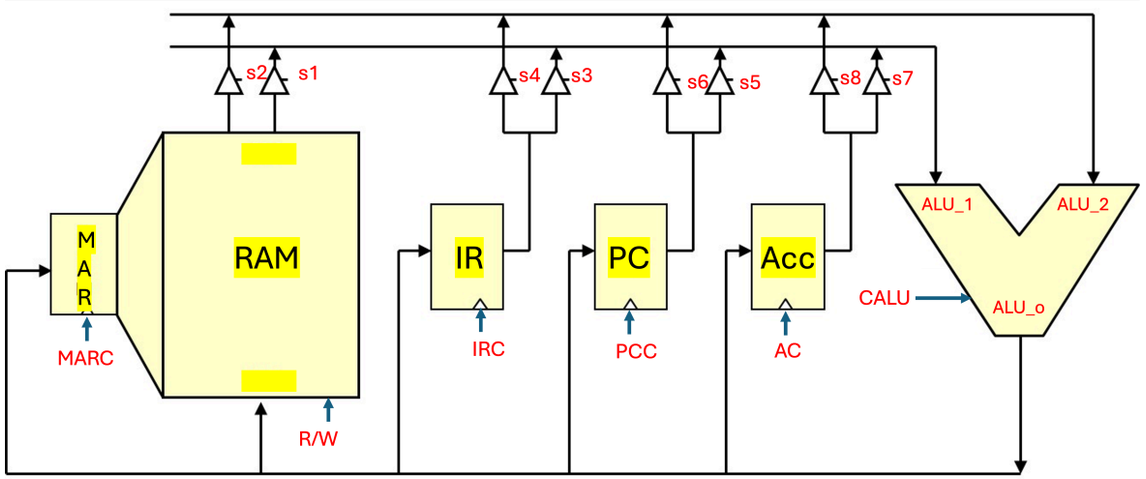
TIME TABLE - FETCH



xxx	time 1 (PC $\rightarrow$ RAM)	time 2 (RAM $\rightarrow$ IR)	time 3 (PC+1 $\rightarrow$ PC)
	PC $\rightarrow$ ALU$_1$, ALU$_O$ $\rightarrow$ MAR	RAM $\rightarrow$ ALU$_1$, ALU$_O$ $\rightarrow$ IR	PC $\rightarrow$ ALU$_1$, ALU$_O$ $\rightarrow$ PC
	s5=1, CALU=00 (T_1)	s1=1, CALU=00 (T_2)	s5=1, CALU=10 (T_3)
	MARC=1 (P_1)	IRC=1 (P_2)	PCC=1 (P_3)

The fetch steps are the same for all instructions.

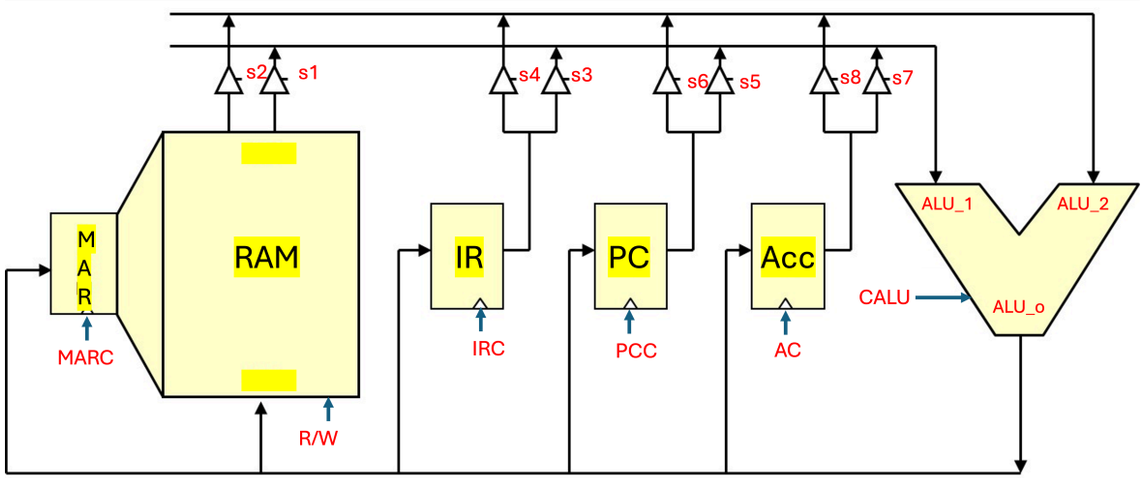
TIME TABLE - LOAD addr/value



$LOAD_1$ addr	time 4 (IR $\rightarrow$ RAM)	time 5 (RAM $\rightarrow$ ACC)
	IR $\rightarrow$ ALU_1, ALU_O $\rightarrow$ MAR s3=1, CALU=00 (T_4) MARC=1 (P_4)	RAM $\rightarrow$ ALU_1, ALU_O $\rightarrow$ ACC s1=1, CALU=00 (T_5) AC=1 (P_5)

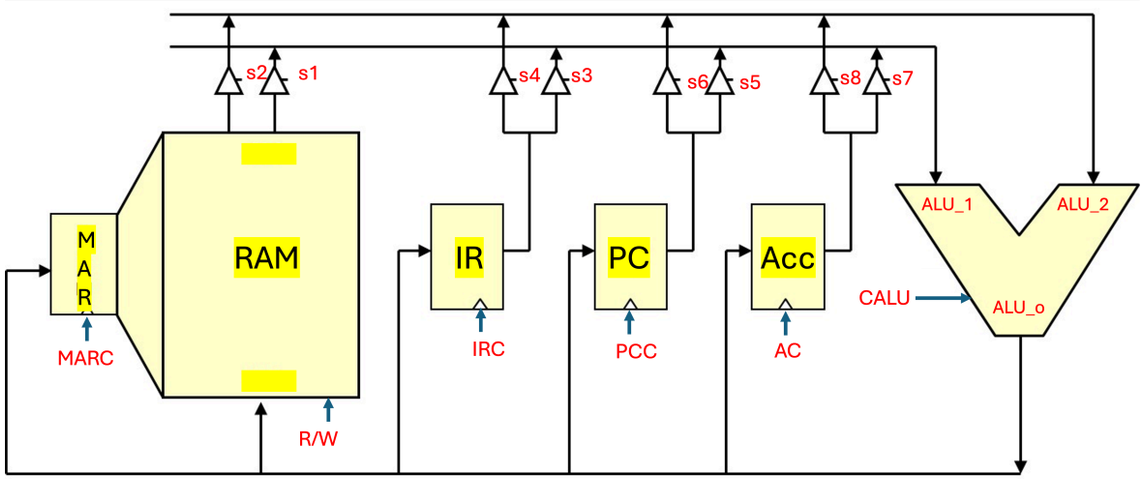
$LOAD_2$ value	time 4 (IR $\rightarrow$ ACC)
	IR $\rightarrow$ ALU_1, ALU_O $\rightarrow$ ACC s3=1, CALU=00 (T_4) AC=1 (P_4)

TIME TABLE - STORE addr



STORE addr	time 4 (IR $\rightarrow$ RAM)	time 5 (ACC $\rightarrow$ RAM)
	IR $\rightarrow$ ALU$_1$, ALU$_O$ $\rightarrow$ MAR	ACC $\rightarrow$ ALU$_1$, ALU$_O$ $\rightarrow$ RAM
	s3=1, CALU=00 (T_4)	s7=1, CALU=00 (T_5)
	MARC=1 (P_4)	R/W=1 (P_5)

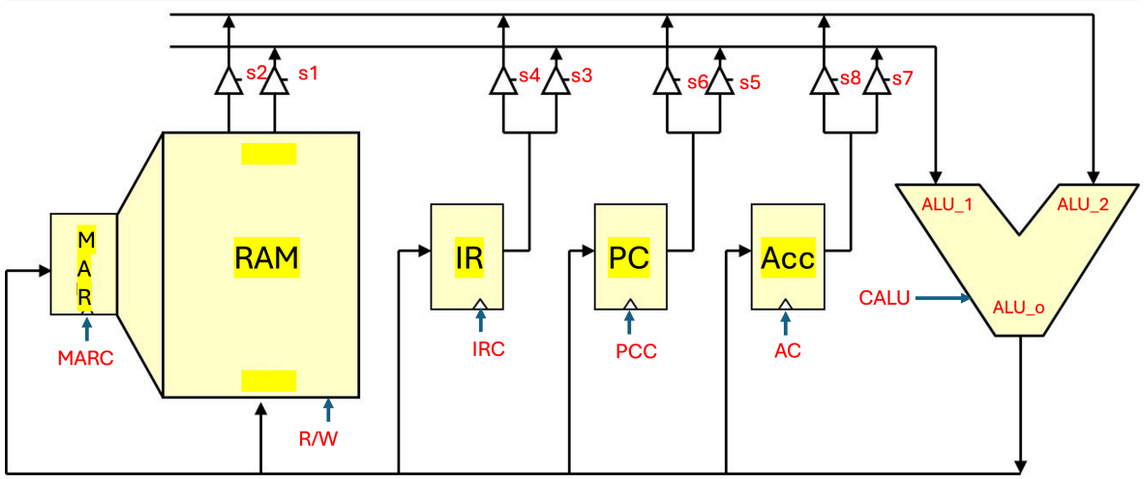
TIME TABLE - ADD addr/value



ADD_1 addr	time 4 (IR $\rightarrow$ RAM)	time 5 (ACC + RAM $\rightarrow$ ACC)
	IR $\rightarrow$ ALU_1, ALU_O $\rightarrow$ MAR s3=1, CALU=00 (T_4) MARC=1 (P_4)	ACC $\rightarrow$ ALU_1, RAM $\rightarrow$ ALU_2, , ALU_O $\rightarrow$ ACC s2=1, s7=1, CALU=10 (T_5) AC=1 (P_5)

ADD_2 value	time 4 (ACC + IR $\rightarrow$ ACC)
	ACC $\rightarrow$ ALU_1, IR $\rightarrow$ ALU_2, ALU_O $\rightarrow$ ACC s3=1, s7=1, CALU=10 (T_4) AC=1 (P_4)

TIME TABLE - SUB addr/value



SUB_1 addr	time 4 (IR $\rightarrow$ RAM)	time 5 (ACC - RAM $\rightarrow$ ACC)
	IR $\rightarrow$ ALU_1, ALU_O $\rightarrow$ MAR s3=1, CALU=00 (T_4) MARC=1 (P_4)	ACC $\rightarrow$ ALU_1, RAM $\rightarrow$ ALU_2, , ALU_O $\rightarrow$ ACC s2=1, s7=1, CALU=11 (T_5) AC=1 (P_5)

SUB_2 value	time 4 (ACC - IR $\rightarrow$ ACC)
	ACC $\rightarrow$ ALU_1, IR $\rightarrow$ ALU_2, ALU_O $\rightarrow$ ACC s3=1, s7=1, CALU=11 (T_4) AC=1 (P_4)

TIME TABLE

`fetch`	s5=1, CALU=00 (T_1) MARC=1 (P_1)	s1=1, CALU=00 (T_2) IRC=1 (P_2)	s5=1, CALU=10 (T_3) PCC=1 (P_3)
`LOAD`$_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s1=1, CALU=00 (T_5) AC=1 (P_5)	
`LOAD`$_2$ `value`	s3=1, CALU=00 (T_4) AC=1 (P_4)		
`STORE addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s7=1, CALU=00 (T_5) R/W=1 (P_5)	
`ADD`$_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s2=1, s7=1, CALU=10 (T_5) AC=1 (P_5)	
`ADD`$_2$ `value`	s3=1, s7=1, CALU=10 (T_4) AC=1 (P_4)		
`SUB`$_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s2=1, s7=1, CALU=11 (T_5) AC=1 (P_5)	
`SUB`$_2$ `value`	s3=1, s7=1, CALU=11 (T_4) AC=1 (P_4)		

EXAMPLE OF LOGICAL EQUATION

fetch	s5=1, CALU=00 (T_1) MARC=1 (P_1)	s1=1, CALU=00 (T_2) IRC=1 (P_2)	s5=1, CALU=10 (T_3) PCC=1 (P_3)
LOAD_1 addr	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s1=1, CALU=00 (T_5) AC=1 (P_5)	
LOAD_2 value	s3=1, CALU=00 (T_4) AC=1 (P_4)		
STORE addr	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s7=1, CALU=00 (T_5) R/W=1 (P_5)	
ADD_1 addr	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s2=1, s7=1, CALU=10 (T_5) AC=1 (P_5)	
ADD_2 value	s3=1, s7=1, CALU=10 (T_4) AC=1 (P_4)		
SUB_1 addr	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s2=1, s7=1, CALU=11 (T_5) AC=1 (P_5)	
SUB_2 value	s3=1, s7=1, CALU=11 (T_4) AC=1 (P_4)		

EXAMPLE OF LOGICAL EQUATION

$$\text{MARC} = P_1 + (\text{LOAD}_1 \cdot P_4) + (\text{STORE} \cdot P_4) + (\text{ADD}_1 \cdot P_4) + (\text{SUB}_1 \cdot P_4)$$

EXAMPLE OF LOGICAL EQUATION

`fetch`	s5=1, CALU=00 (T_1) MARC=1 (P_1)	s1=1, CALU=00 (T_2) IRC=1 (P_2)	s5=1, CALU=10 (T_3) PCC=1 (P_3)
$\texttt{LOAD}_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s1=1, CALU=00 (T_5) AC=1 (P_5)	
$\texttt{LOAD}_2$ `value`	s3=1, CALU=00 (T_4) AC=1 (P_4)		
`STORE addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s7=1, CALU=00 (T_5) R/W=1 (P_5)	
$\texttt{ADD}_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s2=1, s7=1, CALU=10 (T_5) AC=1 (P_5)	
$\texttt{ADD}_2$ `value`	s3=1, s7=1, CALU=10 (T_4) AC=1 (P_4)		
$\texttt{SUB}_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s2=1, s7=1, CALU=11 (T_5) AC=1 (P_5)	
$\texttt{SUB}_2$ `value`	s3=1, s7=1, CALU=11 (T_4) AC=1 (P_4)		

EXAMPLE OF LOGICAL EQUATION

$$s2 = (ADD_1 \cdot T_5) + (SUB_1 \cdot T_5)$$

EXAMPLE OF LOGICAL EQUATION

`fetch`	s5=1, CALU=00 (T_1) MARC=1 (P_1)	s1=1, CALU=00 (T_2) IRC=1 (P_2)	s5=1, CALU=10 (T_3) PCC=1 (P_3)
$\texttt{LOAD}_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s1=1, CALU=00 (T_5) AC=1 (P_5)	
$\texttt{LOAD}_2$ `value`	s3=1, CALU=00 (T_4) AC=1 (P_4)		
`STORE addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s7=1, CALU=00 (T_5) R/W=1 (P_5)	
$\texttt{ADD}_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s2=1, s7=1, CALU=10 (T_5) AC=1 (P_5)	
$\texttt{ADD}_2$ `value`	s3=1, s7=1, CALU=10 (T_4) AC=1 (P_4)		
$\texttt{SUB}_1$ `addr`	s3=1, CALU=00 (T_4) MARC=1 (P_4)	s2=1, s7=1, CALU=11 (T_5) AC=1 (P_5)	
$\texttt{SUB}_2$ `value`	s3=1, s7=1, CALU=11 (T_4) AC=1 (P_4)		

EXAMPLE OF LOGICAL EQUATION

$s3 = T_4$

THANK YOU

PDF version of the slides

Back to the begin - Back to the outline