SAP1-Plus Ultra

Group:- G7

Team:-

1.Ibrahim Samy   Section 1
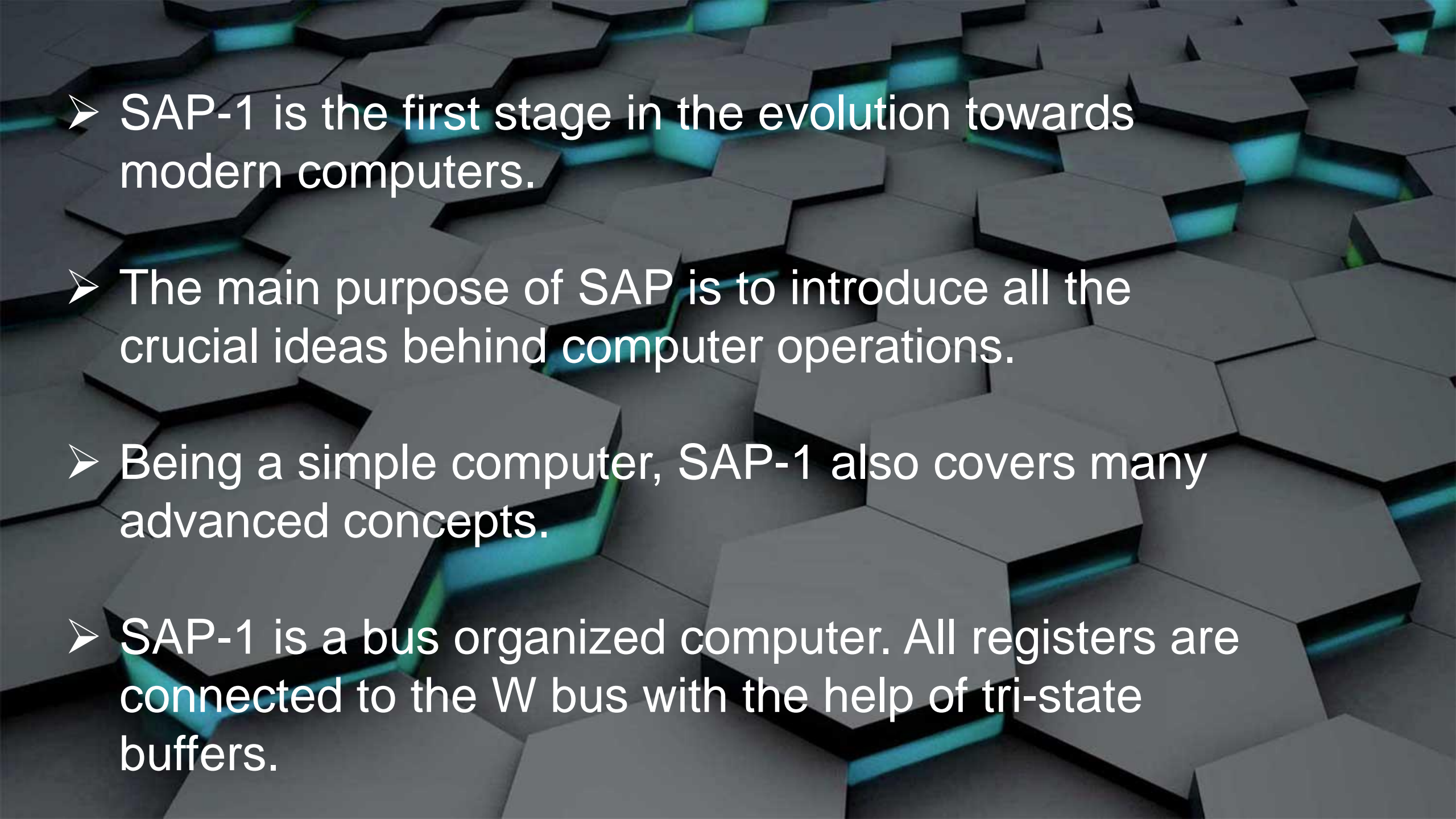2.Yahia Mohammed   Section 4
3.Yahia Shaban   Section 4
4.Youssef Samy   Section 4

Youssef Samy

➢ SAP-1 is the first stage in the evolution towards modern computers.

➢ The main purpose of SAP is to introduce all the crucial ideas behind computer operations.

➢ Being a simple computer, SAP-1 also covers many advanced concepts.

➢ SAP-1 is a bus organized computer. All registers are connected to the W bus with the help of tri-state buffers.

- **Simple-As-Possible.**
- **One output device with 8 LEDs**
- **32 bytes of Memory.**
- **7 instructions**
  - **3 with 1 operand**
  - **4 with no operand.**
- **Accumulator Architecture**
  - **Accumulator**
  - **Out Register**
  - **B Register**
  - **Word size 8-bit**
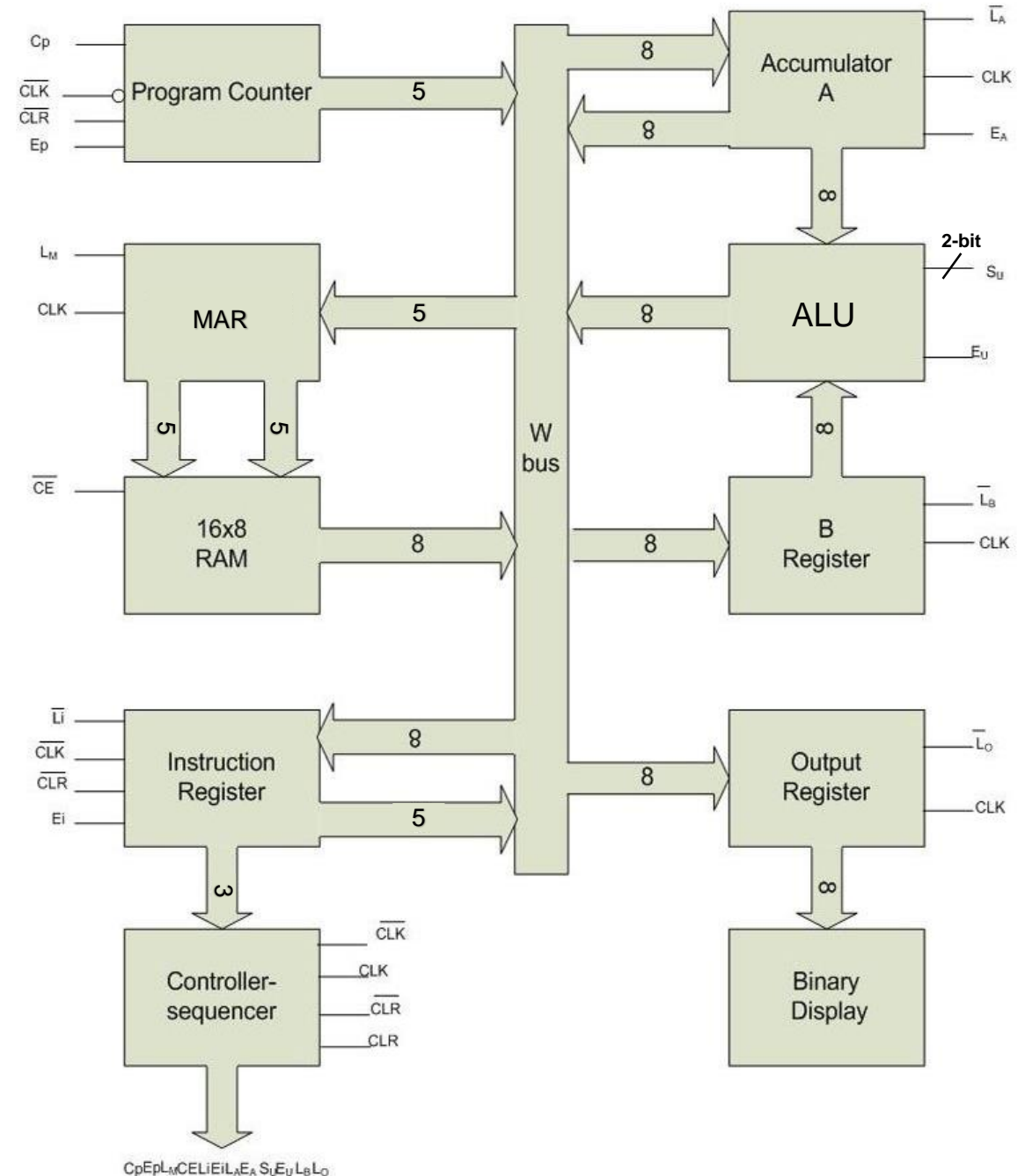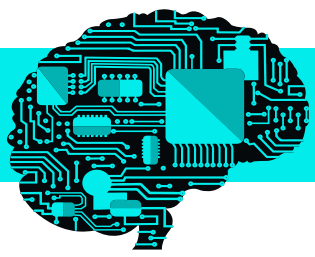
Architecture

➢ **8-bit "W" bus.**

➢ **5-bit program counter, only counts up, it starts counting from 0 and counts up to 31.**

➢ **5-bit Memory Address Register (MAR).**

➢ **32 Byte Memory.**

➢ **8-bit (1 Byte) Instruction Register (IR).**

➢ **5-cycle controller with 13-bit control signal**

➢ **8-bit Accumulator.**

➢ **8-bit B Register.**

➢ **8-bit ALU.**

➢ **8-bit Output Register.**
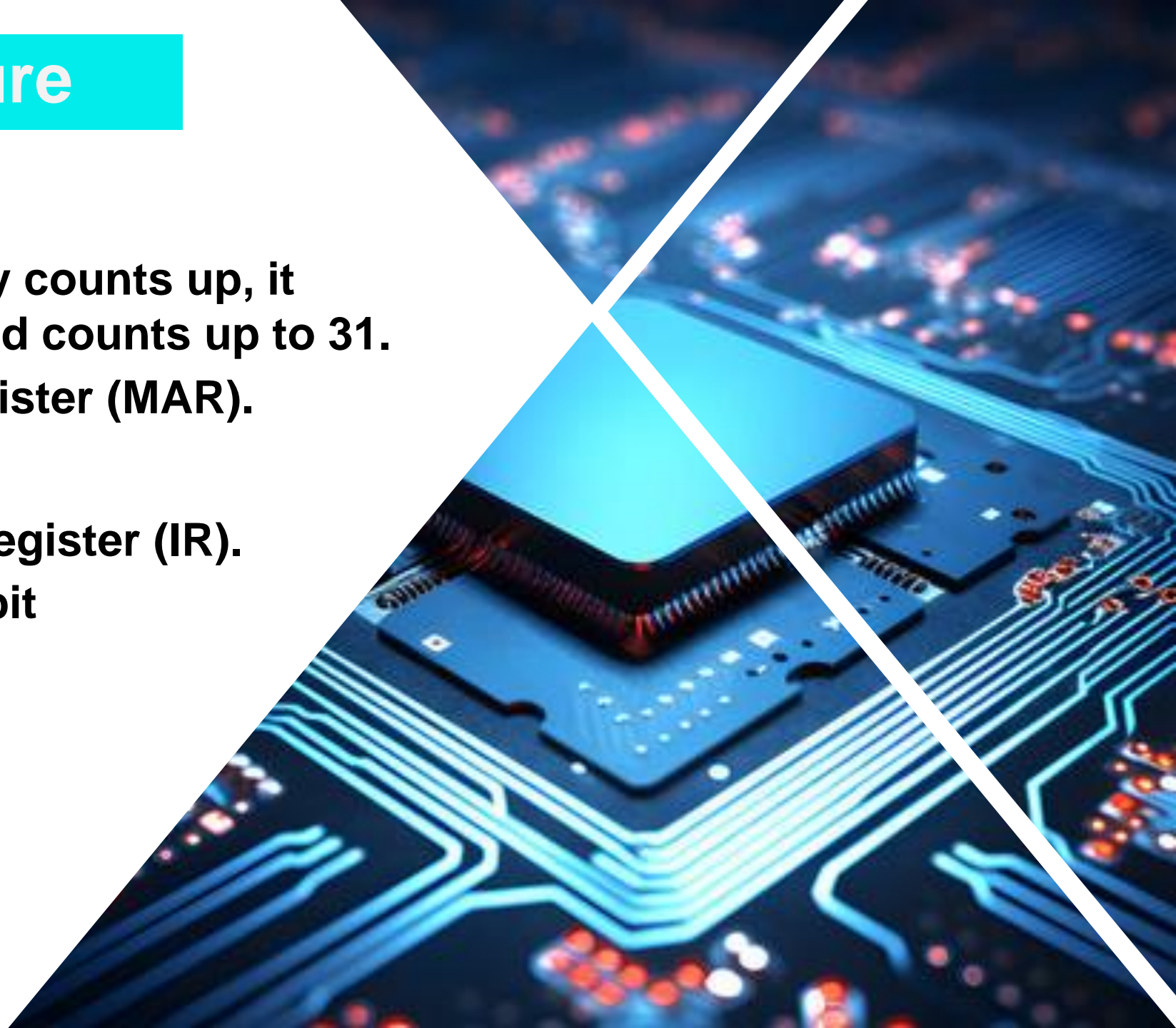
Processor Updating

How to make the SAP-1 Better

01 **RAM Size**
✓ Make the RAM Bigger

02 **ISA**
✓ Add More Instructions

03 **Execute Time**
✓ Decrease The execution time

04 **ADDER/SUBTRACTOR**
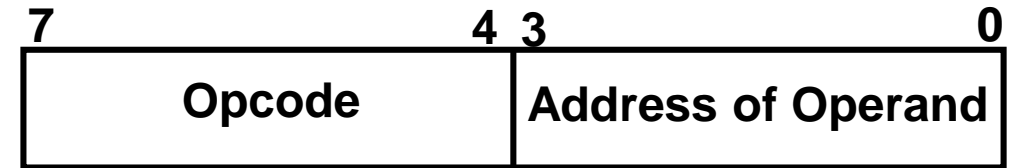✓ Add more functionality

Goals

Old vs New (Instruction)

Old

Instruction

The instruction consists of 8bits:
Divided to:
- **4 bits address**
- **4 bits opcode**

7	Opcode	4	3	Address of Operand	0

Instruction
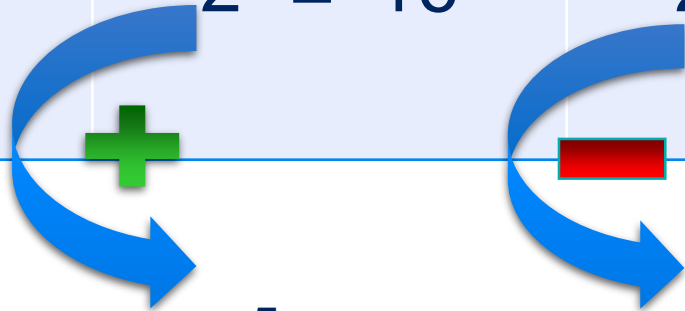
The instruction consists of 8bits:
Divided to:
- **5 bits address**
- **3 bits opcode**

New

7	Opcode	5	4	Address of Operand	0
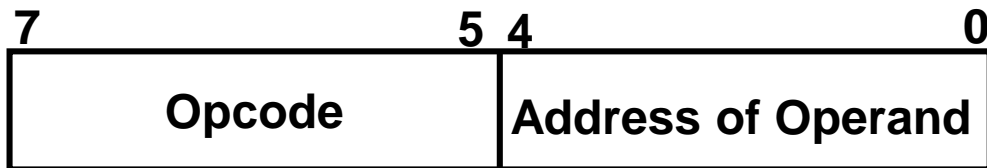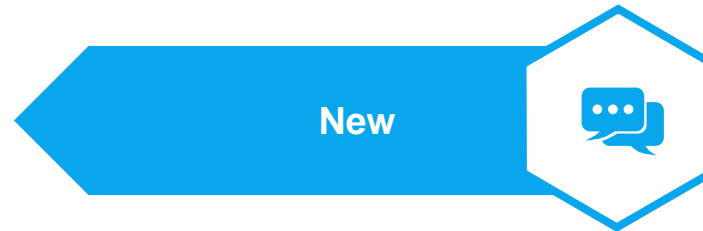
Result

Version	Accessible Locations	Max No of Instructions
Old	$2^4 = 16$	$2^4 = 16$
New	$2^5 = 32$	$2^3 = 8$

The #instructions has decreased but still **sufficient**

Old vs New (ALU)

Old

ADDER/SUBTRACTOR

- **It can only do some arithmetic operations such that ADD & SUB**

New

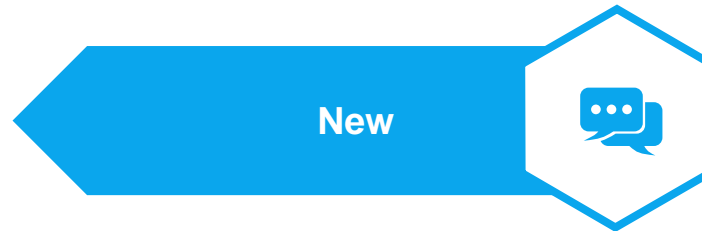**ALU**

- **By adding some logical operations such that Logical Shift Right & Shift left**
- **It has become an ALU**

Result

ADD/SUB

ALU
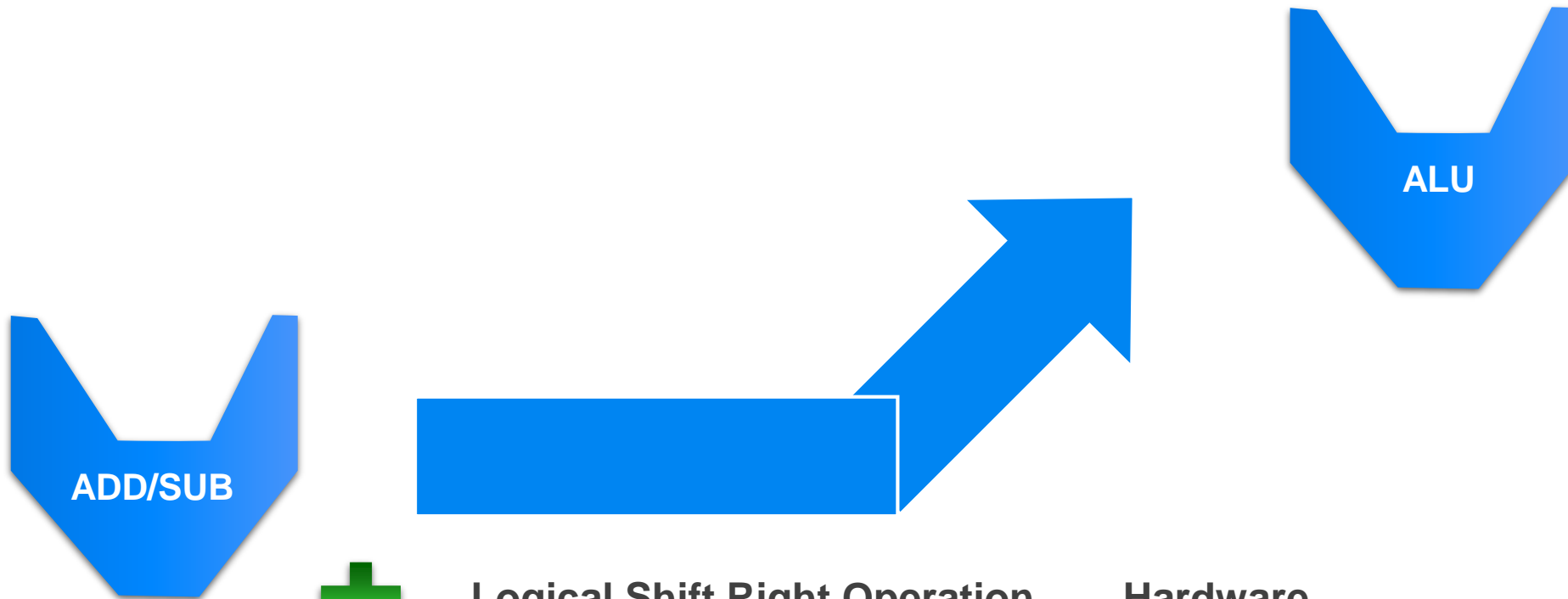
➕ **Logical Shift Right Operation**   **Hardware**

➕ **Logical Shift Left Operation**   **Hardware**

➕ **Multiply Operation**   **Software**

➕ **Division Operation**   **Software**

Old vs New (Controller)

Old

Instruction Execute Time
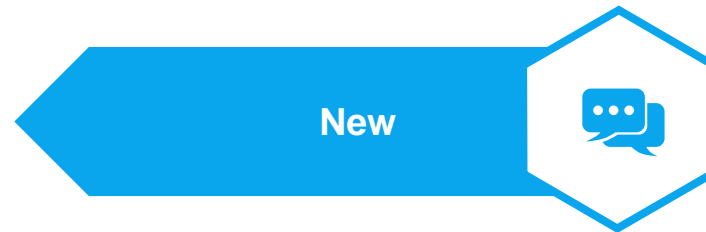
- **Any Instruction Takes 6 time signals to be executed**
  - **3 time signals for fetch**
  - **3 time signals for execute**

New

Instruction Execute Time

- **Make the fetch takes only 2 time signals**
- **Reset the signal counter immediately after the instruction execution finish**

Result

Make the Processor Faster

Less Power Consumption

Yahia Mohamed

Blocks

ALU

- **SAP-1 uses a 2's complement adder-subtractor. When input S_u is (logic 00), the sum is:**

$$S = A + B$$

- **When S_u is (logic 01), the Shift Right is:**
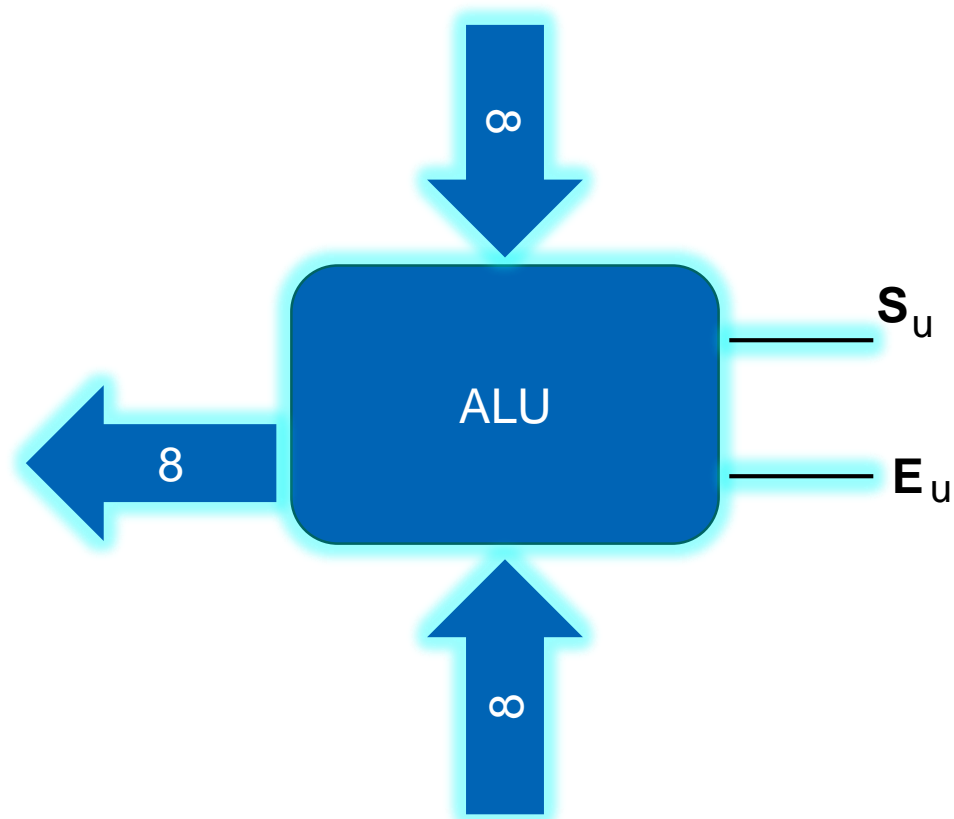
$$S = A >> 1$$

- **When S_u is (logic 10), Shift left is:**

$$S = A << 1$$

- **When S_u is (logic 11), the sub is:**

$$S = A + B' + 1$$

- **The ALU is asynchronous and its contents change as soon as the input changes**

- **When E_U is high, these contents appear on the W bus.**

Accumulator

- To add/sub/shift two 8-bit numbers A and B, the accumulator register stored the number A.

- The Accumulator has two outputs.
    - One output goes to the **ALU**
    - The other goes to the bus through tri-state buffers.

- It also stores the (answer of two values) output of **ALU** through w-bus, when $\overline{L_A}$ is low.

- It's value is appeared on w-bus when **E$_A$** is high, which can then be read by output register.
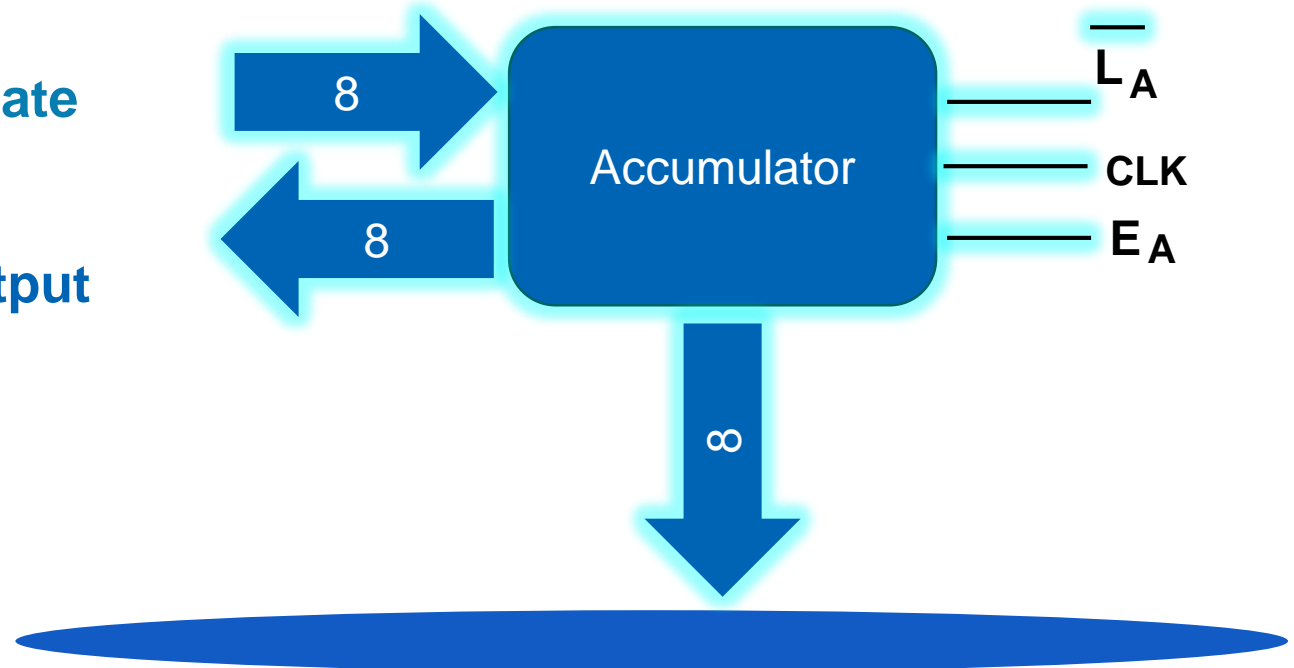
B Register

- **To add/sub/ shift two 8-bit numbers A and B, the B register stored the number B.**

- **It supplies the number to be added or subtracted from the contents of accumulator to the ALU.**

- **When data is available at W-bus and $\overline{\text{L}_\text{B}}$ goes low, at the rising clock edge, B register loads that data.**

8

8

B Register

$\overline{\text{L}}_\text{B}$

CLK
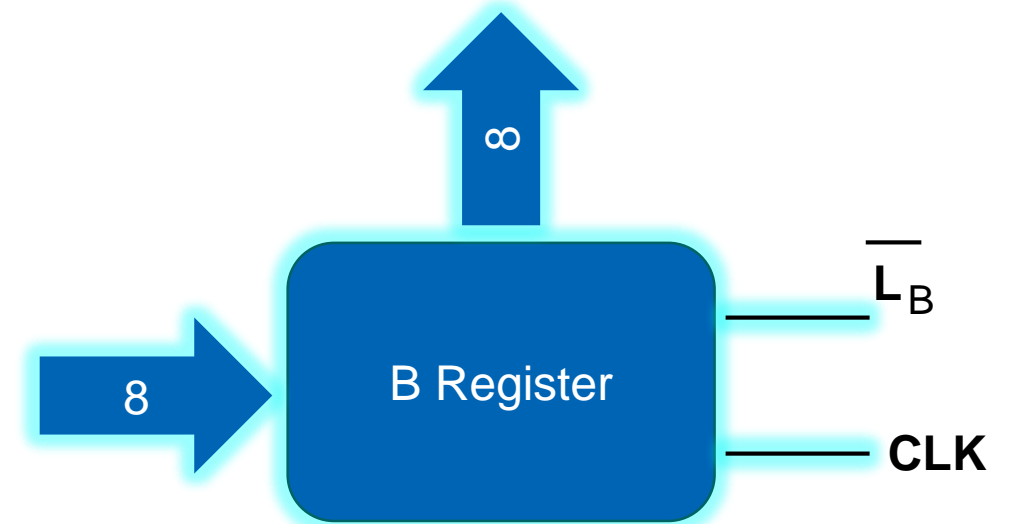
Program counter

- When SAP-1 is **Running Mode (Cp is active),** the (5-bit) address is generated by the Program Counter which is then stored into the MAR through W bus **(Ep is active).**

- The MAR stores the (5-bit) **address of data and instruction** which are placed in memory.

- A bit later, the MAR applies this 5-bit address to the RAM, where Data or instruction is read from RAM

C_P

$\overline{CLK}$

$\overline{CLR}$

E_P

Program counter

5
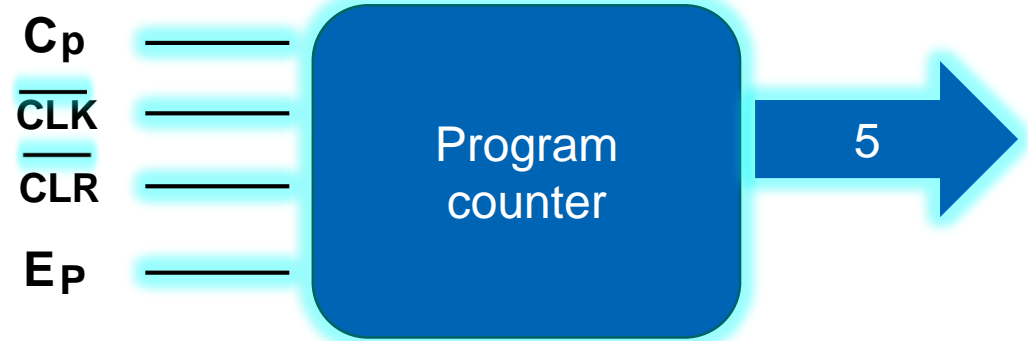
MAR

- **The MAR stores the (5-bit) address of data and instruction which are placed in memory.**

- **When SAP-1 is Running Mode, the (5-bit) address is generated by the Program Counter which is then stored into the MAR through W bus.**

- **A bit later, the MAR applies this 5-bit address to the RAM, where Data or instruction is read from RAM.**

L_M

CLK

MAR

5

5
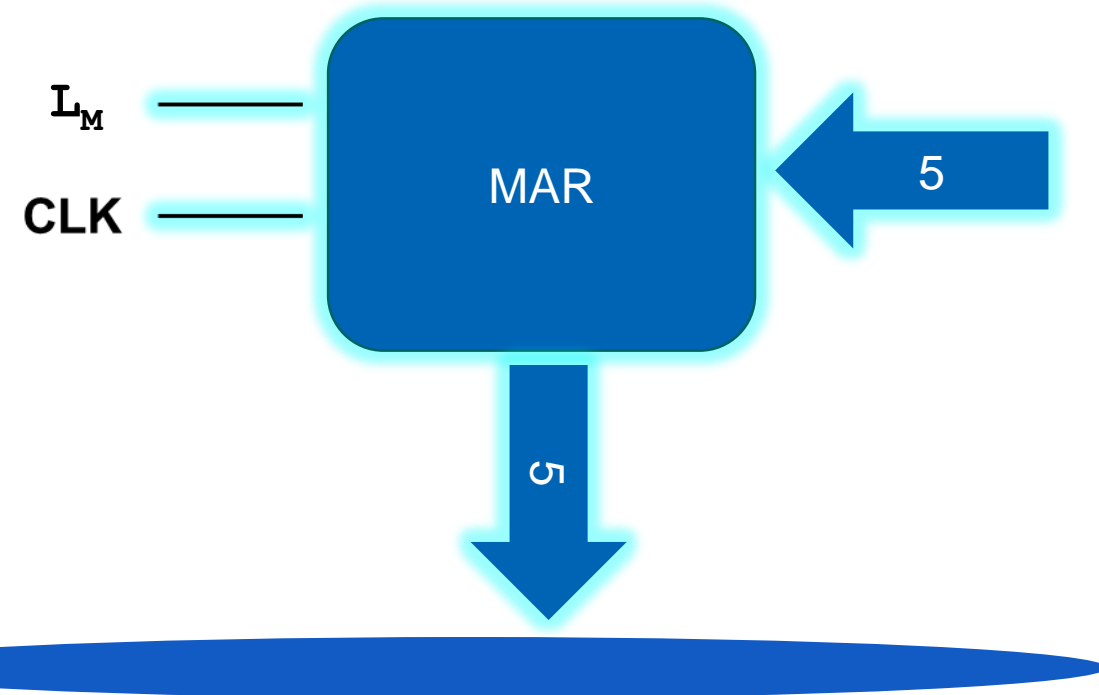
Memory

- In initial design, the **RAM** is a **32 x 8** static TTL RAM. It means there are **32 memory locations** (from 0 to 31) and each location contains an 8-bit of data/instruction.

- You can program the RAM to be used for address and data. This allows you to store a program and data in the memory before a computer run.

- During a computer run, the RAM receives **5-bit addresses** from the MAR and a read operation is performed,

- in this way, the instruction or data stored in the RAM and when $\overline{C_E}$ is low the data placed on the W bus for use in some other part of the computer.

5

RAM
32*8

8

$\overline{C_E}$
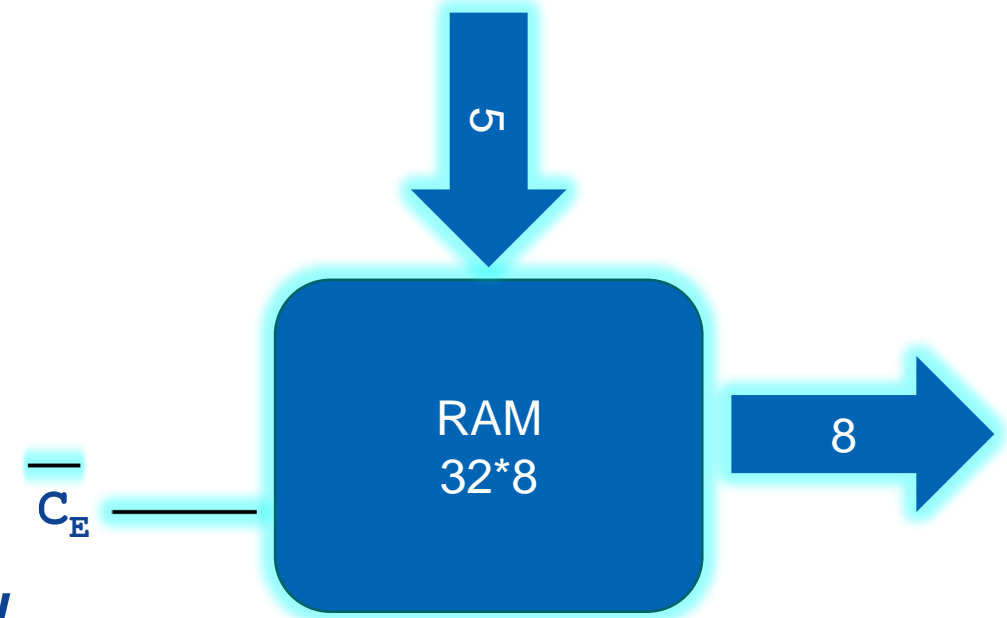
Instruction Register

- **When the instruction is placed at bus from memory, the Instruction Register stores this instruction on the next rising clock edge.**

- **The contents of the instruction register are split into two parts.**

- **The upper part (3-bit) is a two-state output that goes directly to the block labeled "Controller-sequencer"**

- **The lower part (5-bit) is a three-state output that is read onto the bus when $\overline{E_I}$ is low**

$\overline{L_I}$

CLK

$\overline{E_I}$

Instruction Register
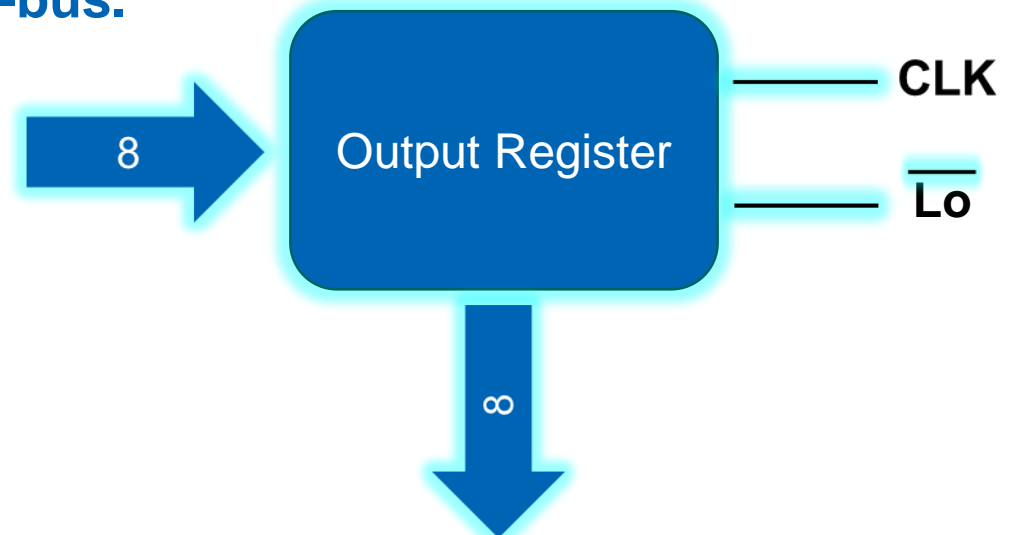
8

5

3
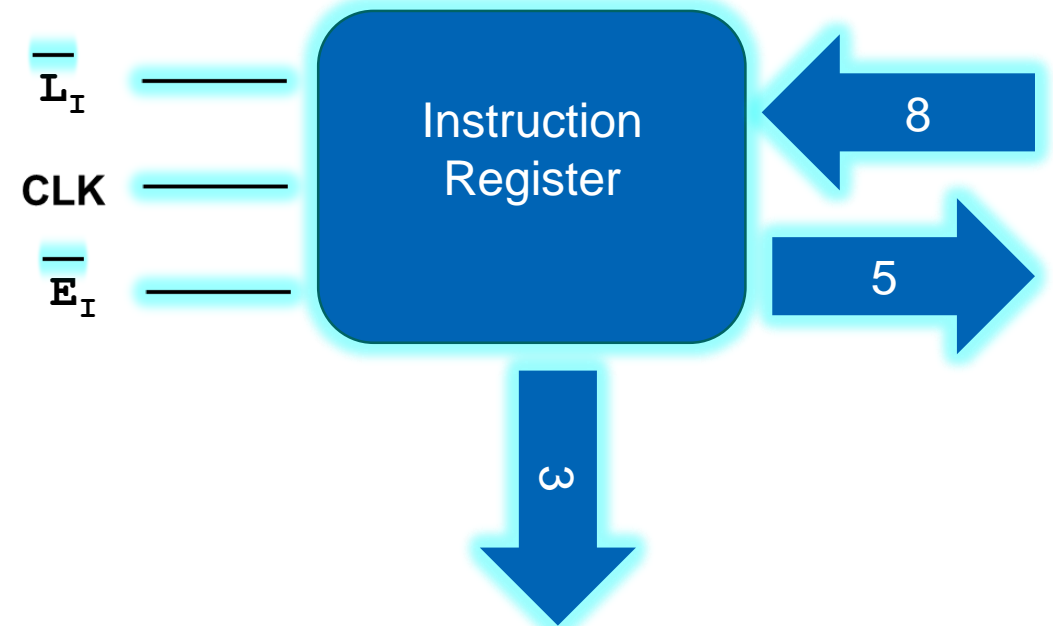
Output Register

At the end of an arithmetic operation the accumulator contains the word representing the answer.

Then answer stored in the accumulator register is then loaded into the output register through W-bus.

This is done in the next positive clock edge when E_A is **high** and L_O is **low**.

Yahia Mohamed

Controller

- **The Control Unit is responsible for linking all the components together**

- **The Controller receives 3-bit opcode from IR register**

- **Then it executes the Instruction using 13-bits different Control Signals**

- **Control Signals :-**

$$C_P \quad E_P \quad \overline{L_M} \quad \overline{C_E} \quad \overline{L_I} \quad \overline{E_I} \quad \overline{L_A} \quad E_A \quad S_{U(2bits)} \quad E_U \quad \overline{L_B} \quad \overline{L_O}$$

Opcode

3

Controller / Sequencer

13
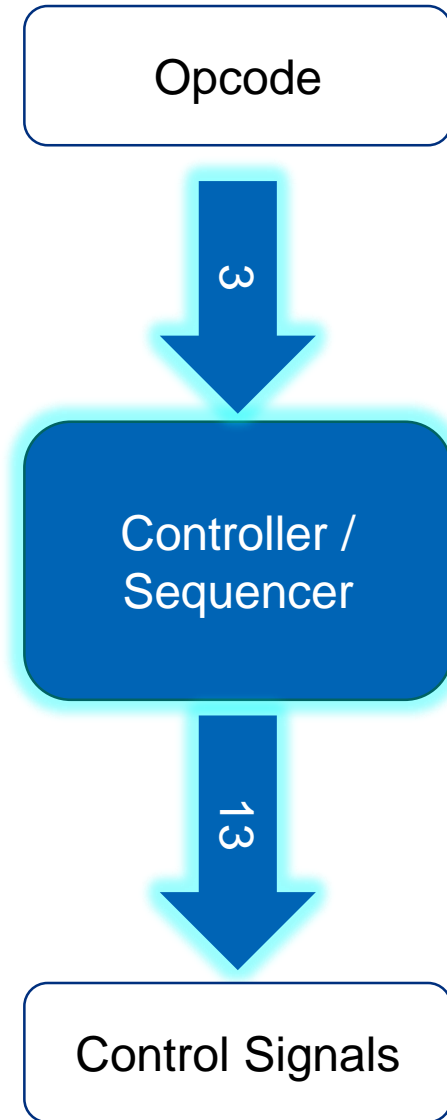
Control Signals

Control Signals

Signal	Size	Active	Function
C_P	1 bit	High	Increase the PC
E_P	1 bit	High	Write PC content on BUS
L_M	1 bit	Low	Load MAR from BUS
C_E	1 bit	Low	Write RAM content on BUS
L_I	1 bit	Low	Load IR from BUS
E_I	1 bit	Low	Write IR content on BUS
L_A	1 bit	Low	Load ACC from BUS
E_A	1 bit	High	Write ACC content on BUS
S_U	2 bits	N/A	Select ALU operation
E_U	1 bit	High	Write ALU content on BUS
L_B	1 bit	Low	Load Register B from BUS
L_O	1 bit	Low	Load OUT Register from BUS

Controller / Sequencer

$C_P E_P \overline{L_M} \overline{C_E} \overline{L_I} \overline{L_A} \overline{E_I} E_A S_U E_U \overline{L_B} \overline{L_O}$
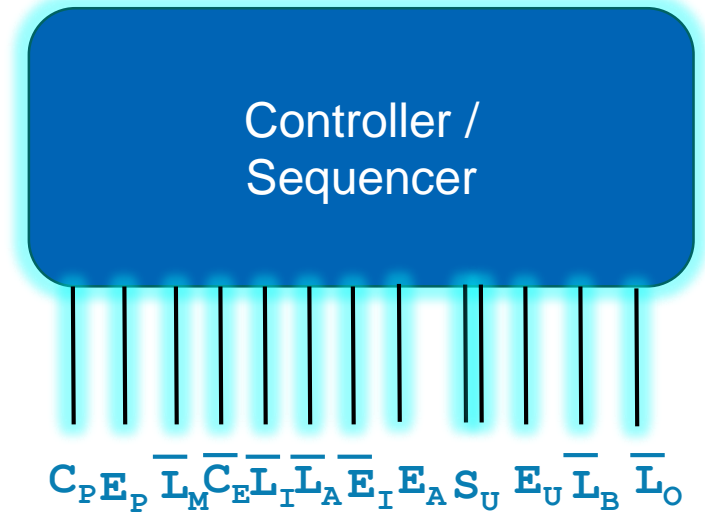
Ibrahim Samy

Instruction Set 😊

7		5	4		0
Opcode			Address of Operand		

SUB
opcode
010 ADDRESS

AC ← AC-B

LDA
opcode
000 ADDRESS
AC←MEM[MAR]

SHIFT RIGHT
opcode
100 XXXX
AC ← srl AC

ADD
opcode
001 ADDRESS

AC←AC+B

OUT
opcode
101 XXXX

OUT←AC

SHIFT LEFT
011 XXXX

AC← sll AC

HLT
opcode
111 XXXX

CLK←0
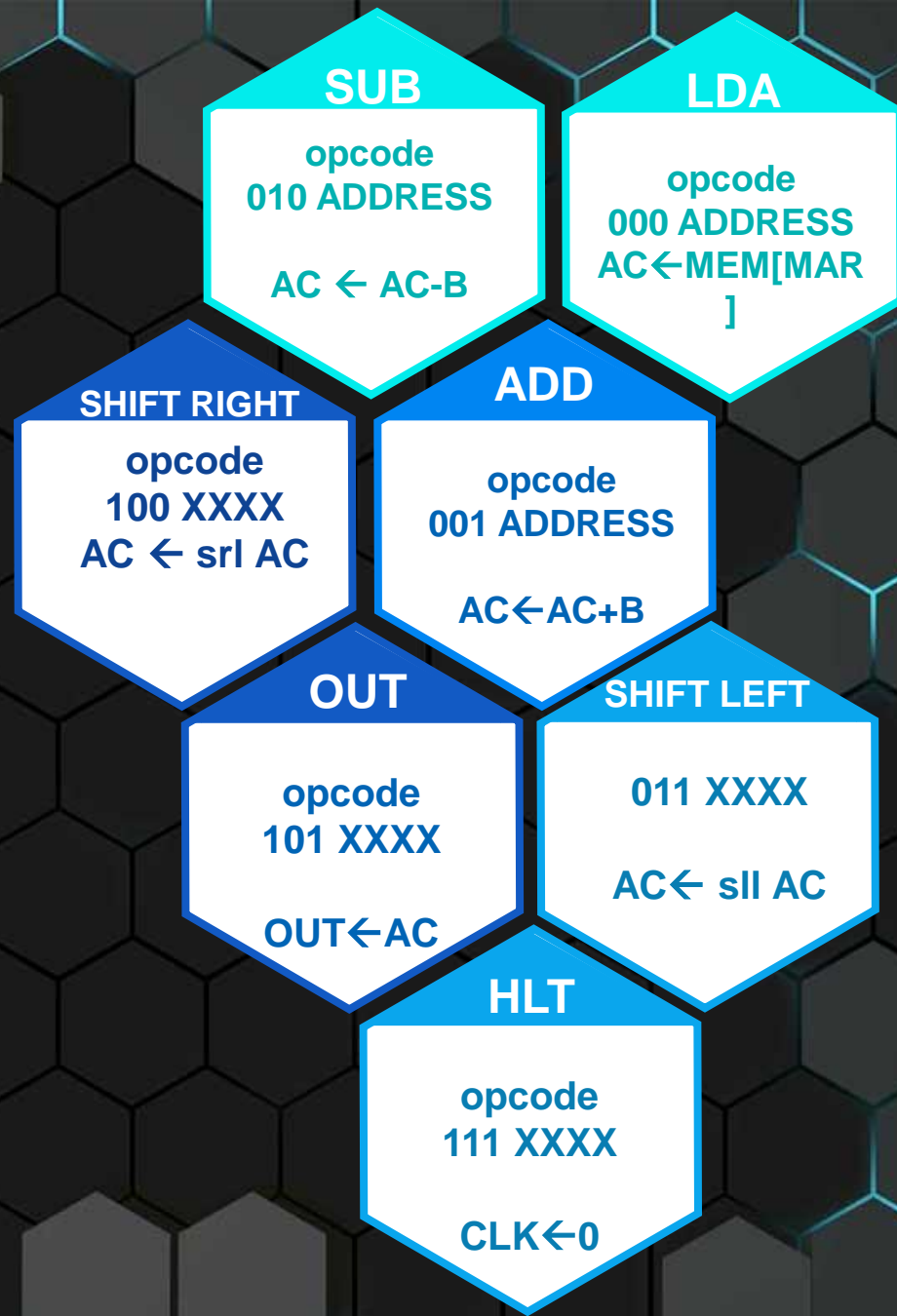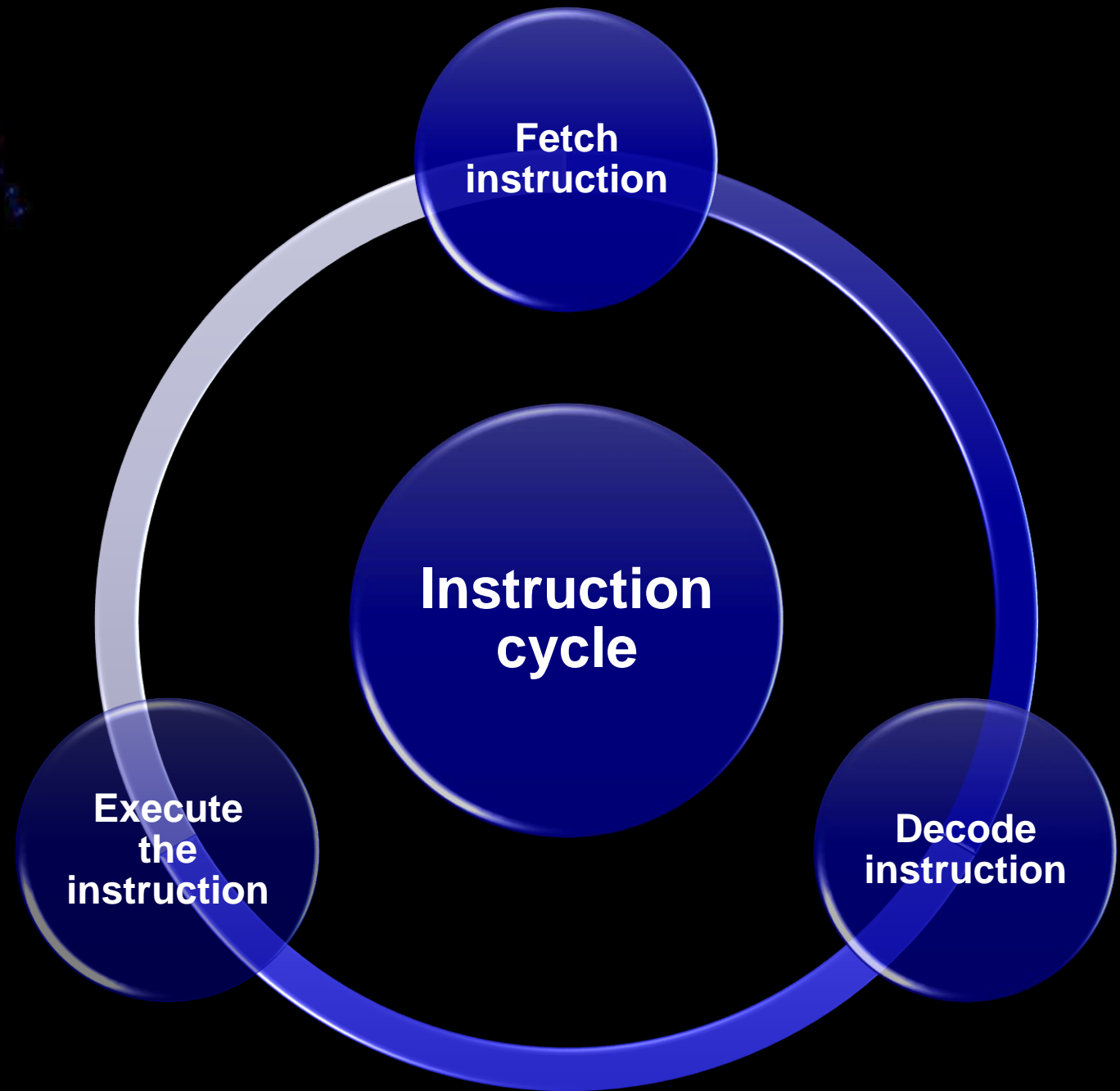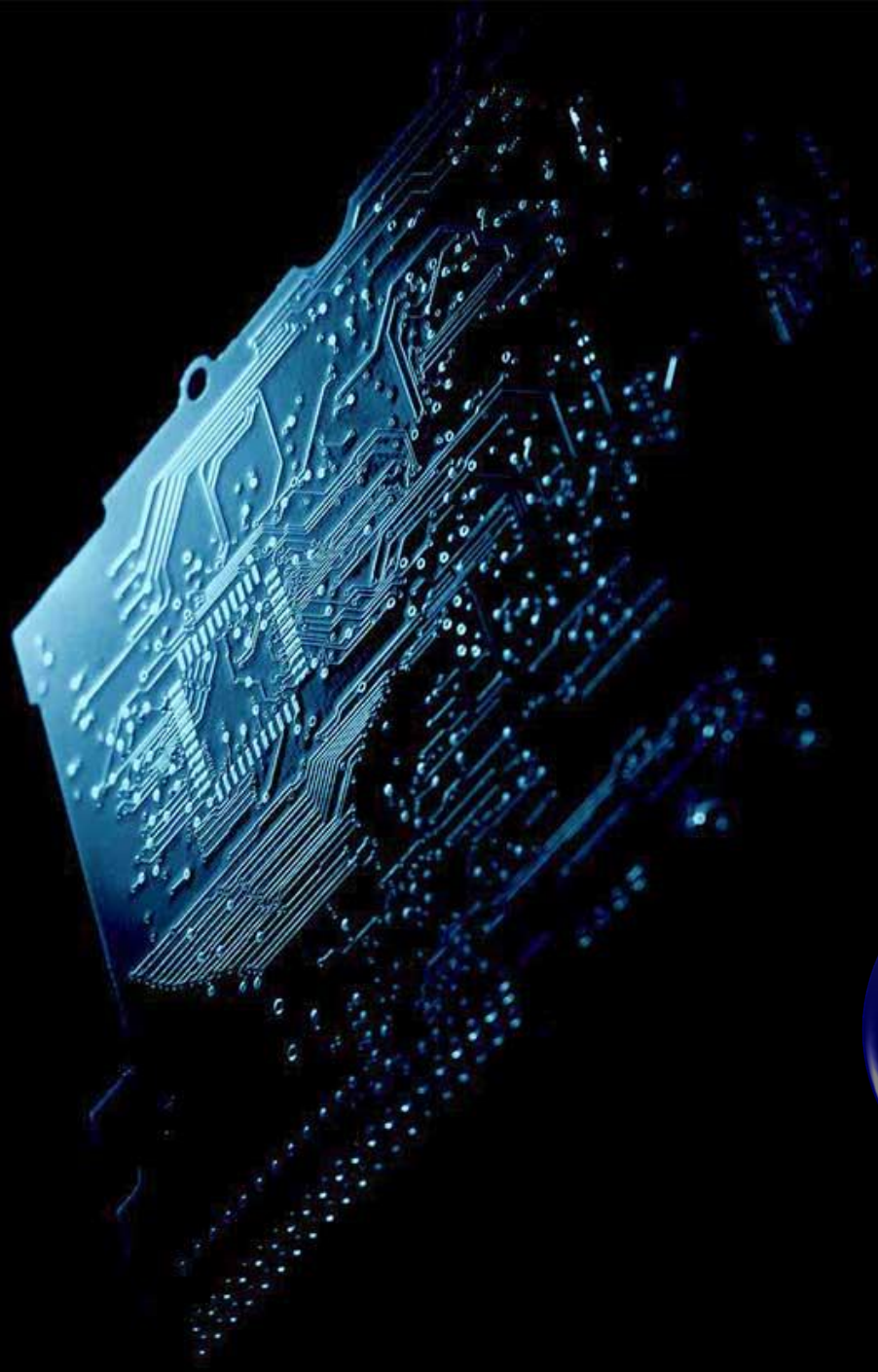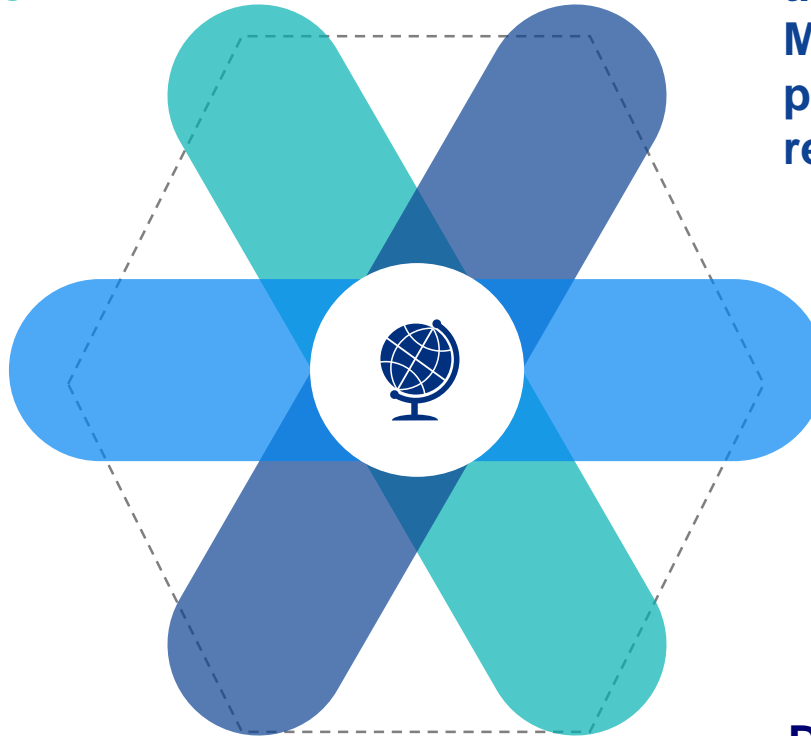
Fetch
T1,T2

During the **address state**, E_P and L_M are active; this means that the program counter sets up the MAR via the W bus.

During the **memory state**, CE' and L'_I are active. The addressed RAM word sets up the instruction register via the W bus. Midway through the memory state, a positive clock edge loads the instruction register with the addressed RAM word.

$$IR \leftarrow M[AR]$$

A positive clock edge occurs midway through the address state; this loads the MAR with the contents of the PC.

$$AR \leftarrow PC$$

This sets up the program counter to count positive clock edges. Halfway through the increment state, a positive clock edge hits the program counter and advances the count by 1.

The **address, increment,** and **memory states** are called the **fetch cycle** of SAP-I.

During the **increment state**, C_P , $\overline{Li}$ is the only active control bits.

$$PC \leftarrow PC+1$$
$$IR \leftarrow M[AR]$$

Execution Cycle :

- The next three states (T_3, T_4, and T_5) are the execution cycle of SAP-1

- The register transfers during the execution cycle depend on the particular instruction being executed.

- What follows are the control routines for different SAP-1 instructions.

Macro Instructions :

- **The instructions we have been programming with (LDA, ADD, SUB, . . .) are sometimes called macro-instructions to distinguish them from micro-instructions.**

- **Each SAP-1 macroinstruction is made up of maximum three micro-operations. For example, the LDA macroinstruction consists of the two micro-operations.**
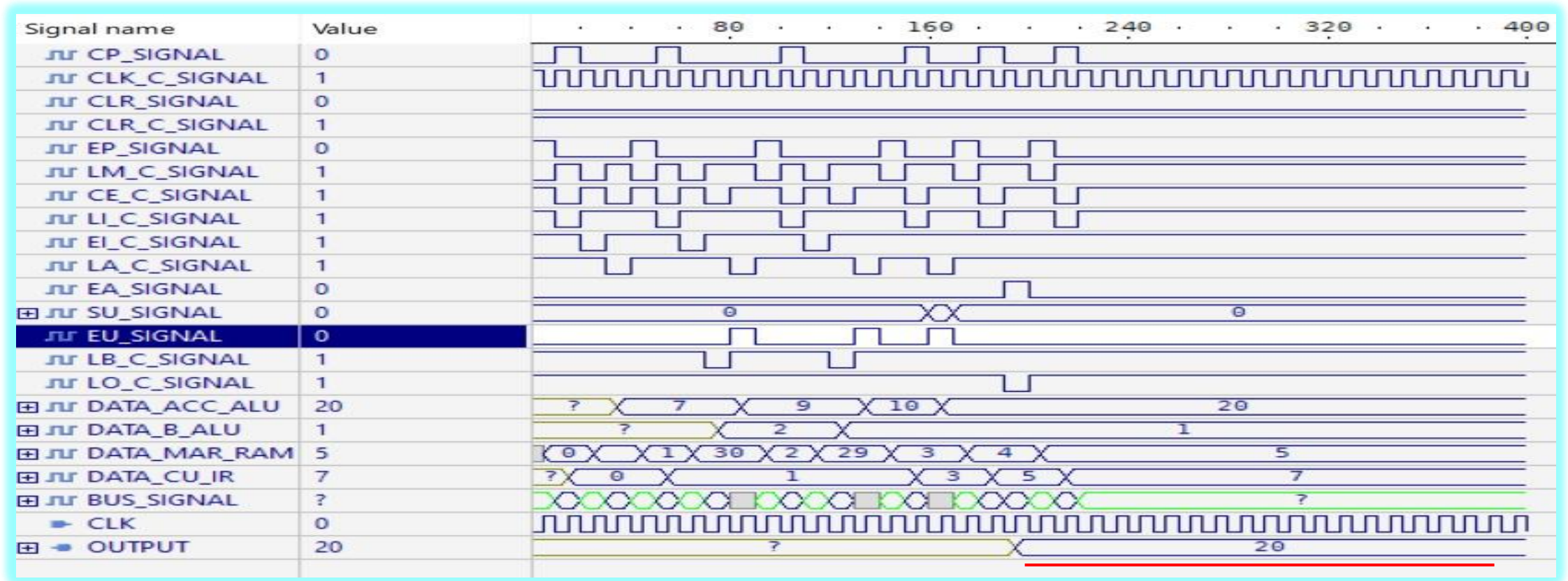
Micro operation :

- **The controller-sequencer sends out control words, on during each T state or clock cycle.**

- **These words are like directions telling the rest of the computer what to do.**

- **Because it produces a small step in the data processing, each control word is called a micro-operation.**

Youssef Samy

Ex:1

Signal name	Value	
CP_SIGNAL	0	
CLK_C_SIGNAL	1	
CLR_SIGNAL	0	
CLR_C_SIGNAL	1	
EP_SIGNAL	0	
LM_C_SIGNAL	1	
CE_C_SIGNAL	1	
LI_C_SIGNAL	1	
EI_C_SIGNAL	1	
LA_C_SIGNAL	1	
EA_SIGNAL	0	
SU_SIGNAL	0	
EU_SIGNAL	0	
LB_C_SIGNAL	1	
LO_C_SIGNAL	1	
DATA_ACC_ALU	20	
DATA_B_ALU	1	
DATA_MAR_RAM	5	
DATA_CU_IR	7	
BUS_SIGNAL	?	
CLK	0	
OUTPUT	20	

instruction

```
when "00000" => RAM_OUT <= "00011111"; -- LDA
when "00001" => RAM_OUT <= "00111110"; -- ADD
when "00010" => RAM_OUT <= "00111101"; -- ADD
when "00011" => RAM_OUT <= "01100000"; -- shift left
when "00100" => RAM_OUT <= "10100000"; -- out
when "00101" => RAM_OUT <= "11100000"; -- hlt
```
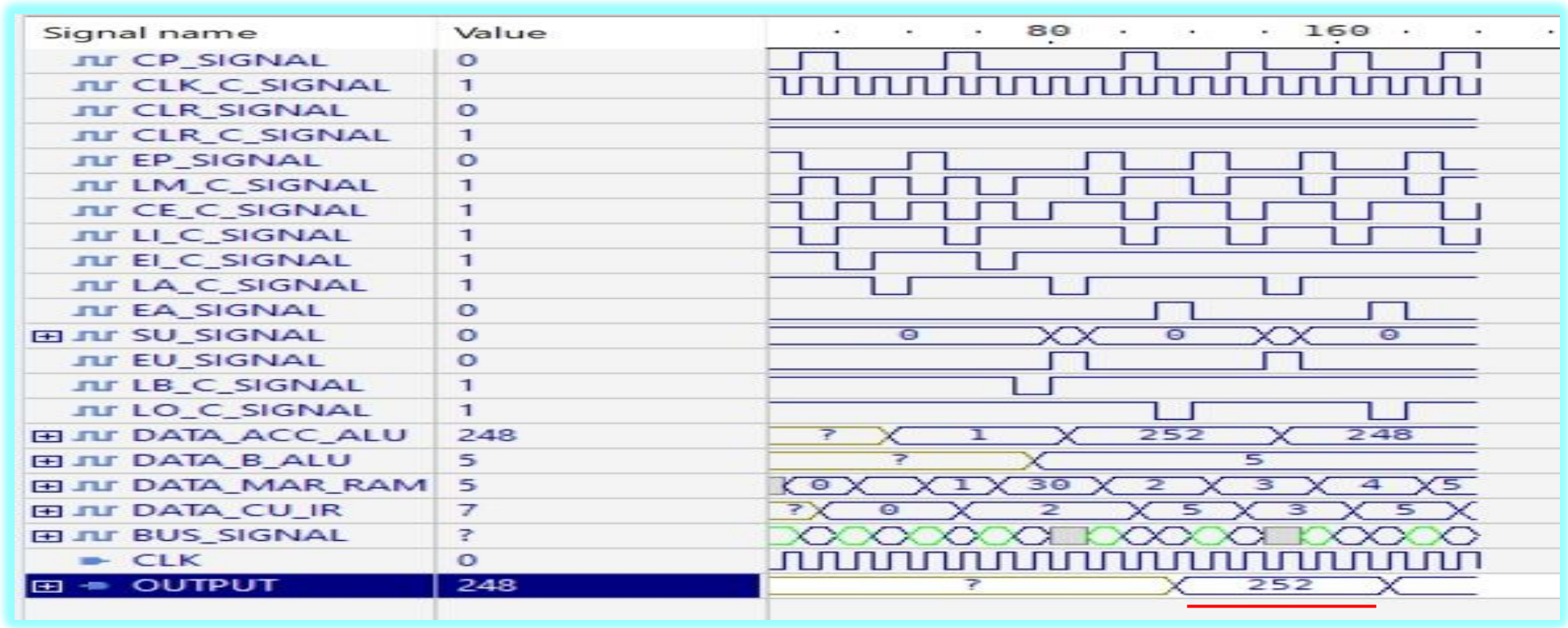
Data in memory

```
when "11100" => RAM_OUT <= "01000000";
when "11101" => RAM_OUT <= "00000001";  --1
when "11110" => RAM_OUT <= "00000010";  --2
when "11111" => RAM_OUT <= "00000111";  --7
```

Ex:2

Signal name	Value			80		160	

instruction **Data in memory**

```
when "00000" => RAM_OUT <= "00011111"; -- LDA
when "00001" => RAM_OUT <= "01011110"; -- SUB    when "11110" => RAM_OUT <= "00000101";  --5
when "00010" => RAM_OUT <= "10100000"; -- OUT    when "11111" => RAM_OUT <= "00000001";  --1
```

Thanks' for your time