Basic Computer

Computer Architecture Course

Sepideh Bayati

Negin Firouzian

Supervised by

Dr. Hamed Farbeh

25 May 2018

Contents

Basic Computer Design

A basic computer is designed with minimum hardware components to perform the basic tasks that a computer should perform (Mano 1993). The organization of the computer is defined by its internal registers. The internal organization of a digital system is defined by sequence of micro-operations that perform on data stored in its registers. The computer is capable of executing various micro-operations and can be instructed to perform a sequence of operations.

Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register. The control unit then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations.

An instruction code is a group of bits that instruct the computer to perform a specific operation. It is made up of 16 bits, and divided into three parts:

- Two bits (I_1, I_0) to specify the addressing mode.
- Four bits binary code to specify the operation.
-

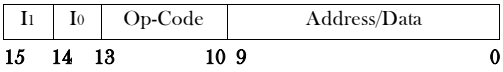

Figure 1: Instruction format.

Table 1 illustrates the addressing mode of the proposed computer, which is used to:

- Reduce the number of bits in the address field of the instruction, and
- Give user flexibility in dealing with counters, pointers ...etc
-

Table 1: Addressing Modes.

I1	I0	Addressing mode
0	0	Immediate addressing *($ address)*
0	1	Direct addressing
1	1	Indirect addressing *(# address)*

The computer needs registers for manipulating data and a register for holding a memory address. Nine registers are required for the proposed computer, as shown in Table 2. Some registers (such as AC, MAR, and MBR) may receive data from several multiplexed sources.

Table 2: List of Registers for the Basic Computer.

Register	Bits No.	Function
Memory Buffer Reg. (MBR)	16	Holds memory word
Memory Address Reg. (MAR)	10	Holds address for memory
Address Reg. (AR)	10	Holds operand/instruction add.
Accumulator (AC)	8	Processor register
Counter Reg. (CR)	8	Holds count for loops
Program Counter (PC)	10	Holds address of instruction
Operation Reg. (OPR)	4	Holds code of operation
Input Reg. (INP)	8	Holds input character
Output Reg. (OUTR)	8	Holds output character

The basic computer has eight registers, a memory unit, and a control unit. Paths must be provided to transfer information from one register to another and between memory to registers. A more efficient scheme for transferring information in a system with many registers is to use a common bus. A multiplexer can be used to design the common bus. The connection of the registers to the common bus system is shown in Fig. 2.

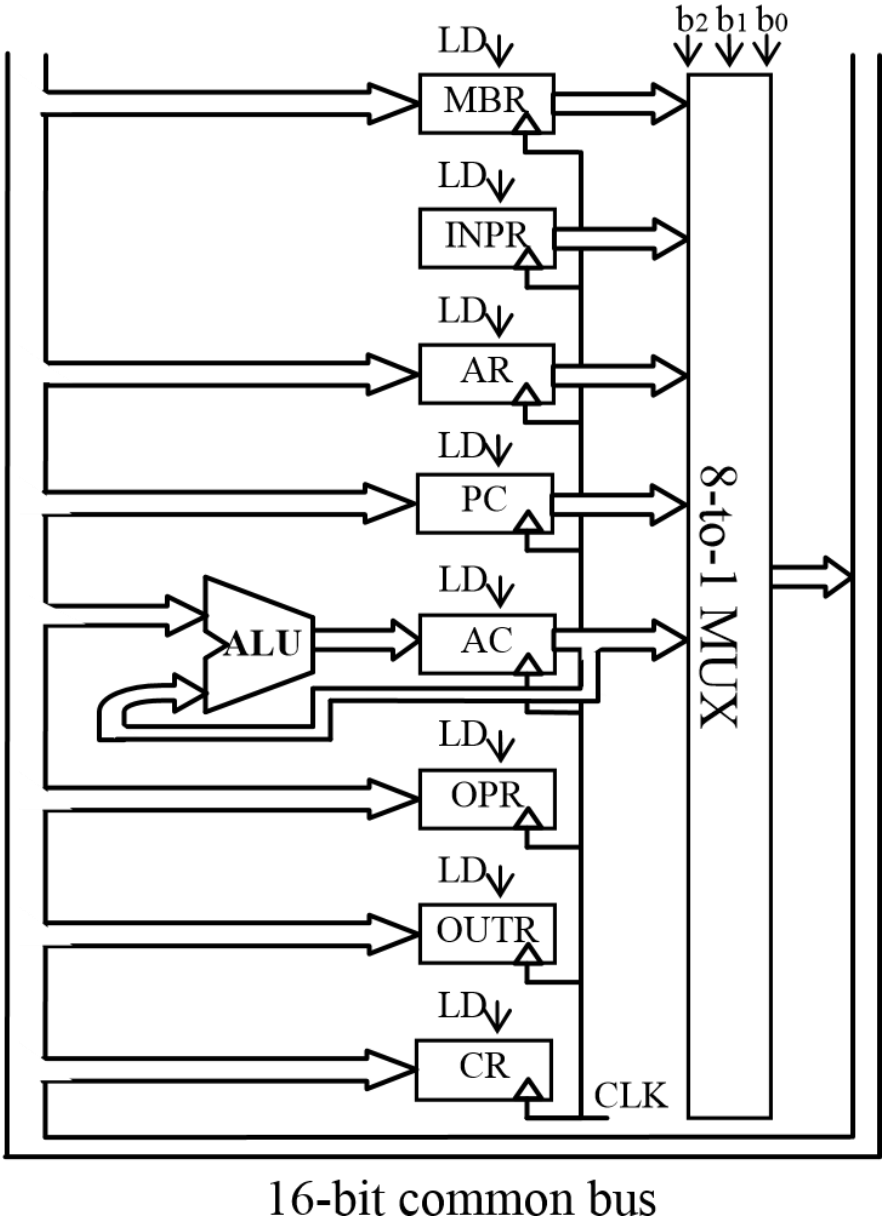

Figure 2: Basic computer registers connected to a common bus.

Computer Instructions

The basic computer has three basic instruction code formats. A memory-reference instruction uses ten bits to specify either an address or an operand and two bits to specify the addressing mode (I_0, I_1). For immediate addressing it is 00, 01 for direct addressing, and 11 for indirect address. The register-reference instructions are recognized by the operation code 1111 and a 00 in the left most bits of the instruction. A register-reference instruction specifies an operation on the AC register. An operand from memory is not needed; therefore, the other 10 bits are used to specify the operation to be executed. Similarly, an input-output instruction does not need a reference to memory and is recognized by the operation code 1111 and 11 in the left most bits of the instruction. The remaining 10 bits are used to specify the type of the input-output operation. This technique allows having up to 35 different operations, as given in Table 3.

Table 3: Basic computer instruction formats

MEMORY REFERENCE INSTRUCTIONS				
	Code for the 6 left most bits			
Symbol	I1I0=00	I1I0=01	I1I0=11	Description
NOP	000000	010000	110000	No operation
ADD	000001	010001	110001	Add memory word to AC
SUB	000010	010010	110010	Subtract memory word from AC
AND	000011	010011	110011	AND memory word to AC
OR	000100	010100	110100	OR memory word to AC
XOR	000101	010101	110101	XOR memory word to AC
LDA	000110	010110	110110	Load memory word to AC
STA	000111	010111	110111	Store content to AC in memory
BUN	001000	011000	111000	Branch unconditionally
BSA	001001	011001	111001	Branch and save return address
DSZ	001010	011010	111010	Decrement and skip if zero
LDC	001011	011011	111011	Load CR
RET	001100	011100	111100	Return
BZ	001101	011101	111101	Branch if zero
BC	001110	011110	111110	Branch if carry
REGISTER REFERENCE INSTRUCTIONS (Hexadecimal code)				
CLA	3C01			Clear AC
CLS	3C02			Clear all status flags
CMA	3C04			Complement AC
SRA	3C08			Shift right AC
SLA	3C10			Shift left AC
INC	3C20			Increment AC
HALT	3C40			Terminate program
INPUT-OUTPUT REFERENCE INSTRUCTIONS				
INP	BC01			Input character to AC
OUT	BC02			Output character from AC
SKI	BC04			Skip on input flag
SKO	BC08			Skip on output flag
ION	BC10			Interrupt on
IOF	BC20			Interrupt off
SFI	BC40			Set input flag
SFO	BC80			Set output flag

Timing and Control

The timing for all registers in the basic computer is controlled by a master clock generator. The control signals are generated by the control unit and provide control inputs for the multiplexer in the common bus, control inputs in processor registers, and micro-operations for the accumulator. The control unit, consists of three decoders, a sequence counter, and a number of control logic gates. The control unit gets the operation code from the OPR through a 4x16 decoder. Bits 14 & 15 of the instruction code are transferred to two flip-flops designated by the symbols I_1& I_0 . Bits 0 through 9 are applied to the control logic gated directly from the MBR through the common bus. The outputs of the counter are decoded into 4 timing signals T0 through T3. The sequence counter (SC) can be incremented or cleared. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 2x4 decoder. Once, the counter is cleared, causing the next active timing signal to T0. The decoder is used to determine the cycle to be performed.

Instruction Cycles

A program residing in the memory unit of the computer consists of a sequence of instructions. Each instruction cycle is subdivided into a sequence of sub-cycles. The value of three flip-flops is entered into a decoder to determine the cycle to be served, as illustrated in Table 4. In this computer each instruction cycle is divided into the following five sub cycles:

Table 4: Cycles Combination.

Flip-flops			Cycle
G	F	R	
0	0	0	Fetch & Decode
0	0	1	Indirect
0	1	0	Direct
0	1	1	Execution
1	0	0	Interrupt

Fetch and Decode Cycle

Initially, the program counter is loaded with the address of the first instruction in the program. The sequence counter is cleared to 0, providing a decoded timing signal T0. After each clock pulse, sequence counter is incremented by one, so that the timing signals go through a sequence T0, T1, T2 and T3. The micro-operations for the fetch and decode cycle can be specified by the following statements:

$$
\begin{aligned}
&C_0T_0 : PC \rightarrow MAR \\
&C_0T_1 : M(MAR) \rightarrow MBR,\ PC+1 \rightarrow PC \\
&C_0T_2 : MBR < I_0, I_1 > \rightarrow (I_0, I_1), \\
&\qquad MBR < OP\_Code > \rightarrow OPR, \\
&\qquad MBR < Address > \rightarrow AR \\
&\overline{I_0}C_0T_3 : \quad 1 \rightarrow R,\ 1 \rightarrow F, 0 \rightarrow G \qquad \text{(ExecuteCycle)} \\
&\overline{I_1}I_0C_0T_3 : O \rightarrow R,\ 1 \rightarrow F, 0 \rightarrow G \qquad \text{(DirectCycle)} \\
&I_1I_0C_0T_3 : 1 \rightarrow R,\ 0 \rightarrow F, 0 \rightarrow G \qquad \text{(IndirectCycle)}
\end{aligned}
$$

Indirect Cycle

At this cycle, the effective address of the operand is to be read from memory. The AR holds the address of memory word, which contains the effective address of the operand.

$$
\begin{aligned}
&C_1T_0 : AR \rightarrow MAR \\
&C_1T_1 : M(MAR) \rightarrow MBR \\
&C_1T_2 : MBR < Address > \rightarrow AR \\
&C_1T_3 : 0 \rightarrow R,\ 1 \rightarrow F, 0 \rightarrow G \qquad \text{(Direct Cycle)}
\end{aligned}
$$

Direct Cycle

The effective address of the operand may be read during two time pulses. Therefore, to disable the delay of waiting for T3, the sequence counter may be cleared at time T2. Thus, the next time pulse will be T0 of the execution cycle and not T3 of the indirect cycle.

$$
\begin{aligned}
&C_2T_0 : AR \rightarrow MAR \\
&C_2T_1 : M(MAR) \rightarrow MBR, \\
&C_2T_2 : 1 \rightarrow R,\ 1 \rightarrow F,\ 0 \rightarrow G, \\
&\qquad 0 \rightarrow CC \quad \text{(Execution Cycle)}
\end{aligned}
$$

Execute cycle

At this cycle, the fetched instruction (register-reference, memory reference, or input-output-reference) is executed. The type of the instruction is decided according to the following table:

Table 5: Combinations of Instruction Types

q15	I0	I1	Instruction Type
0	X	X	MRI
1	0	0	RRI
1	0	1	IOI

Interrupt Cycle

The interrupt cycle is initiated after the execute cycle if the interrupt flip-flop (*INF*) is equal to 1. The flip-flop is set to 1 manually using a switch, and it may be set during the execution of the program by the instruction ION. The flip-flop is reset to 0 whenever the interrupt is served or by the instruction *IOF*.

This basic computer serves the interrupt by saving the next sequential instruction in memory address 0, and then it starts execution from address 1 in the memory. The micro operations required for this instruction are:

$$C_4T_0 : PC \rightarrow MBR < Data >, 0 \rightarrow INF$$
$$C_4T_1 : 0 \rightarrow PC, 0 \rightarrow MAR$$
$$C_4T_2 : MBR < Data > \rightarrow M(MAR), PC + 1 \rightarrow PC$$
$$C_4T_3 : 0 \rightarrow R, 0 \rightarrow F, 0 \rightarrow G \qquad (Fetch\ Cycle)$$

Table 6: Micro-operations of all instructions.

Instr.	Micro-operations.
Memory Reference Instructions (MRIs)	
NOP	No micro-operation
ADD	$q_0C_3T_0$: AC + MBR < Data > → AC
SUB	$q_1C_3T_0$: AC − MBR < Data > → AC
AND	$q_2C_3T_0$: AC ∩ MBR < Data > → AC
OR	$q_3C_3T_0$: AC ∪ MBR < Data > → AC
XOR	$q_4C_3T_0$: AC ⊕ MBR < Data > → AC
LDA	$q_5C_3T_0$: MBR < Data > → AC
STA	$q_6C_3T_0$: AR → MAR, AC → MBR < Data > $q_6C_3T_1$: MBR < Data > → M(MAR)
BUN	$q_7C_3T_0$: AR → PC
BSA	$q_8C_3T_0$: PC → MBR< Address>, AR → MAR $q_8C_3T_1$: MBR< Address> → M(MAR), AR+1 → AR $q_8C_3T_2$: AR → PC
DSZ	$q_9C_3T_0$: MBR < Data > −1 → MBR < Data > $q_9C_3T_1$: MBR < Data > → M(MAR), MBR < Data > → AC $q_9C_3T_2$: IF(AC = 0) Then PC + 1 → PC
LDC	$q_{10}C_3T_0$: MBR < Data > → CR
BZ	$q_{11}C_3T_0$: if ZF = 1 then PC + 1 → PC else MBR < Add> → PC
BC	$q_{12}C_3T_0$: if CF = 1 then PC + 1 → PC else MBR < Add> → PC
Register Reference Instructions (RRIs)	
CLA	rB_0T_0 : 0 → AC
CLS	rB_1T_0 : 0 → All Status Flags
CMA	rB_2T_0 : $\overline{AC}$ → AC
SRA	rB_3T_0 : SHR(AC) → AC, 0 → AC(7)
SLA	rB_4T_0 : SHL(AC) → AC, 0 → AC(0)
INC	rB_5T_0 : AC + 1 → AC
HALT	rB_6T_0 : 0 → StartIndex
Input/Output Reference Instructions (IORIs)	
INP	pB_0T_0 : INPR → AC, 0 → FGI
OUT	pB_1T_0 : AC → OUTR, 0 → FGO
SKI	pB_2T_0 : IF(FGI = 1) then PC + 1 → PC
SKO	pB_3T_0 : IF(FGO = 1) then PC + 1 → PC
ION	pB_4T_0 : 1 → INF
IOF	pB_5T_0 : 0 → INF
SFI	pB_6T_0 : 1 → FGI
SFO	pB_7T_0 : 1 → FGO

Table 6: Micro-operations of all instructions.

Note that:

- $q_{15}\overline{I_1}\overline{I_0}C_3$ = r (Common to all RRIs).
- MBR(n)=B_n[MBR (0-9) that specifies the operation].
- $q_{15}I_1\overline{I_0}C_3$ = p (Common to all input-output instructions).
- MBR(n)=B_n[MBR (0-9) that specifies the operation].

Complete Computer Description

The instruction cycle flowchart is shown in Fig. 3. Figure 4 shows the schematic diagram of the basic computer. It is compound of one 16-bit register, three 10-bit registers, four 8-bit registers, one 4-bit register, eight D-Flip-flops, one 1024x16-bit RAM, one ALU, one 8-bit inverter, two 2-to-1 Multiplexer, one 4-to-1 Multiplexer, one 4-to-16 decoder, one 3-to-8 decoder, one 2-to-4 decoder, and one 2-bit sequence counter.

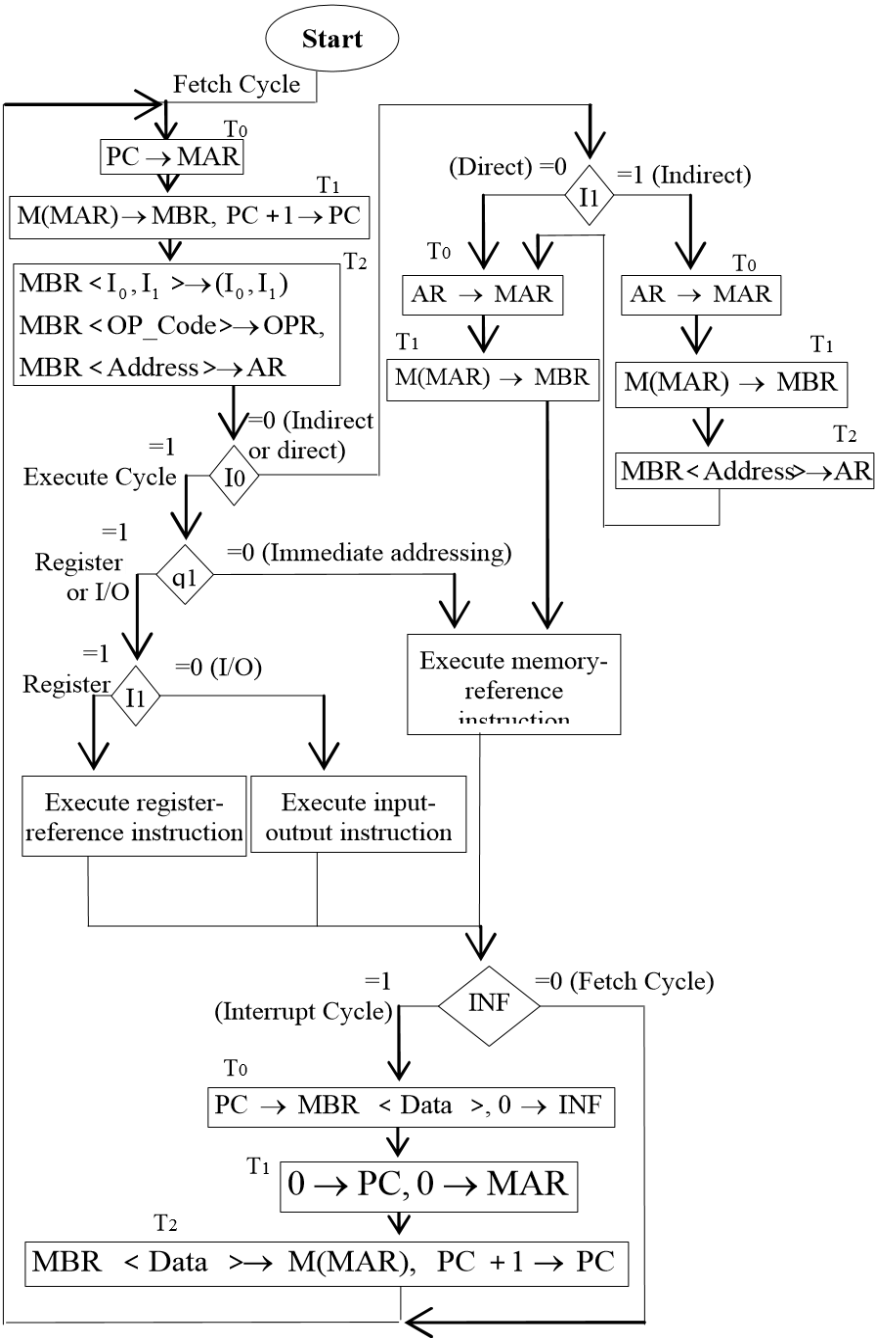

Figure 3: Flowchart of instruction cycle.

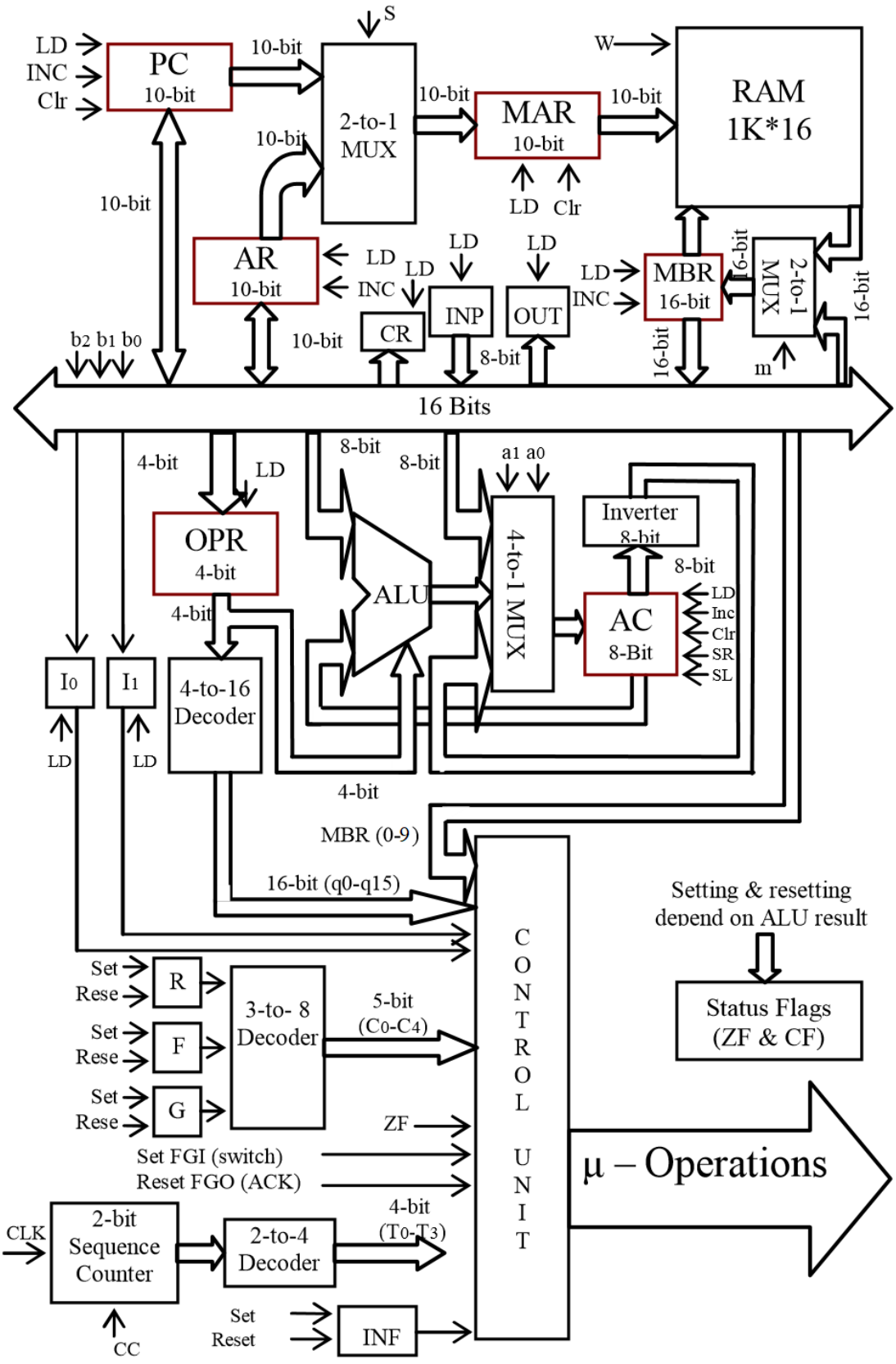

Figure 4: Basic computer block diagram.

Handling Instructions

1. All your project should be implemented by VHDL not Verilog or ...
2. About 10% of your score would be devoted to the report you deliver within your projects code. In this report you would explain the whole job and anything special you have done.
3. In addition to bonus sections mentioned beforehand, implementation of this project on Altera DE2 FPGA boards will result in extra mark as well.
4. You can simulate your code with softwares like: Modelsim, Altera Quartus, GHDL, Xilinx ISE, Active HDL, Xilinx Vivado Design Suite,. . . (note that Modelsim and Quartus has got free versions and GHDL is totally free so it's better to obey Copyright) but the TA's laptop is only equipped with Modelsim and in case of delivery you should be able of running your code in that environment.
5. All questions would be answered by the course's email: computerarchitecture2018@gmail.com .
6. This project could be done both individually and in a group:
   - Individual:
     a. your score which is out of 100 would be multiplied by 1.3.
   - As a member of a group:
     a. your score which is out of 100 would be multiplied by 1.
     b. We assume each of the two members had been involved in every single line of project so the project would be delivered individually.
     c. One of group members should email the names of group mates by 97/3/8 23:55 to the course's email. at 97/3/8 23:56 you are considered as a student eager to complete the project alone.
7. Place all your modules and report into a .zip named as "StudentID_FirstName_LastName.zip" before upload. if any other module is used in your implementation but hasn't been mentioned in this document place it in its proper place next to modules within the same hierarchy.
8. Deadline of the project is at 12 AM on 97/4/18 and cannot be extended, so try to make the deadline.
9. In case of delivery, your code will be downloaded by the responsible TA from Moodle, so the only way to convey your code is Moodle and in if you need to reform your code please upload it when possible to be used in the due date.

Cheating alert

a. All your codes would be processed, both manually and automatically and in case of any similarity by any means, both of individuals involved, would be fined by getting -35 percent of the project's score (note that if pushing your code to Github or any other VCS, exposing your code to a friend or ... results in unexpected similarities of others, you ALL, are responsible!).
b. Any cooperation beyond members of the group is forbidden!
c. The only source you are allowed to copy form, is github.com/AUT-CEIT/Arch2018 repo which has been devoted to this course, copying any other source from the Internet, ... would be considered just like from another classmate.
d. By the way, in case of any similarity with any of the previous students of this course, you blame the same.

Remember that, any HDL (Hardware Design Language) is a piece of art, and could be really enjoyable, if you try your best to understand what's going on instead of just doing it to make it end.

Good Luck