Chapter 3
Computer Architecture

Note by Dr. Abraham. This chapter is being revised. Updates will be posted.

Introduction

Computers could be made to handle analog or digital signals. Today's computers are digital and use two discrete signals such as on and off. Every instruction to be implemented in a CPU, referred to as the instruction set, must be predetermined and the circuitry set up to execute them. The more the instructions in the set such as in the Complex Instruction Set Computers (CISC), the easier the life of the programmer, but, more complex and slower the machine. Many of the instructions in a CISC machine can be replaced by multiple simpler instructions. Such is the case in the Reduced Instruction Set Computers (RISC). RISC favors a minimal number of instructions in the set. The computer architecture is a branch of study that deals with designing of processors and their instruction sets, and organizing various components to facilitate most efficient communication between them without significantly increasing cost. Modern processors are called microprocessors. Each family of microprocessors has its own unique architecture and instruction set.

A computer system consists of the input modules, the processing unit and the output modules. There are some components that provide for both input and output, such as the primary and secondary memories. The processor obtains the instructions to execute from the memory. These instructions are stored in the memory by the operating system when a program is loaded. The processor itself has three basic functional units, arithmetic logic unit (ALU), control unit, and the registers. The CPU reads one instruction at a time from the memory, decodes that instruction, fetches any data that is needed based on the decoded instruction, executes that instruction and writes the results back. In as much as the digital computers use zeros and ones or binary digits (Bits) the readers are encouraged to explore binary mathematics as well as basics of Boolean algebra. A brief introduction to these subjects are given here.

Binary numbers

Binary numbers are much easier to use than the decimal system, but it appears more difficult because we are used to the decimal system. In the decimal system we have to keep track of ten symbols whereas in binary we will only have two know two symbols, 0 and 1. The next higher number to 1 in binary is a 0 with a carry of 1. The sequence from 0 to 15 decimal goes like this in binary: 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010. 1011, 1100, 1101, 1110, 1111. It follows that if have a bank of 4 switches we can turn the off or on to represent this set of numbers. If we had 8 switches we could represent all unsigned integers from 0 to 255. Most computers use a byte (8 bits) as one unit. Multiple bytes called a "word" can be addressed together as well. A word size is generally the size of the registers found in the CPU (discussed later). In order to calculate the value of 11110101 in decimal we start with the right most digit and move to the left multiplying the number (0 or 1) at each digit with the digit value. The digit value is the digit position from right to left raised to the power of two. The rightmost bit is called the least significant bit (LSB) and the left most bit is the most significant bit (MSB).

Number from right to left	**Digit value**	**Value**
1	1	1 x 1 = 1
0	2	0 x 2 = 0
1	4	1 x 4 = 4
0	8	0 x 8 = 0
1	16	1 x 16 = 16
1	32	1 x 32 = 32
1	64	1 x 64 = 64
1	128	1 x 128 = 128
	Total	**245**

Conversion from decimal to binary can be achieved by performing sequence of integer division and modulus operations. For example, to convert decimal 9 to binary, perform an integer division of 9 by 2, giving a 4 with a remainder of 1. This 1 will be the right most bit (least significant bit) in the binary number. Now continue the integer division of 4 with 2 and you have a 2 with a remainder of 0. This 0 will be the next digit holder in the binary. Divide the 2 with a 2 and you have a 1 and a remainder of 0. The 1 will be the left most digit (most significant bit) and the 0 next to it. The binary number will be 1001. Here is another example:

Convert decimal 127 to binary:

Integer Division	Binary Number							
127 \ 2 = 63 R 1								1
63 \ 2 = 31 R 1							1	1
31 \ 2 = 15 R 1						1	1	1
15 \ 2 = 7 R 1					1	1	1	1
7 \ 2 = 3 R 1				1	1	1	1	1
3 \ 2 = 1 R 1			1	1	1	1	1	1
1 \ 2 = 0 R 1		1	1	1	1	1	1	1

Convert decimal 64 to binary

Integer Division	Binary Number							
64 \ 2 = 32 R 0								0
32 \ 2 = 16 R 0							0	0
						0	0	0

16 \ 2 = 8 R 0
8 \ 2 = 4 R 0
4 \ 2 = 2 R 0
2 \ 2 = 1 R 0

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

Principles of addition and subtractions are the same as in the decimal system: addition involves carrying a one when adding 1+1, and subtraction involves borrowing a 1 when subtracting 1 from a 0. Let us perform addition and subtraction on 11010 and 01111. The carry is indicated by a left or right arrow depending on the direction of the borrow.

Addition:

```
        ←     ←     ← ←
        1     1     0   1   0          =        26
        0     1     1   1   1          =        15
-------------------------------                ------
     1  0     1     0   0   1          =        41
```

Subtraction:

```
              →     →   →   →
        1     1     0   1   0          =        26
        0     1     1   1   1          =        15
-------------------------------                ------
        0     1     0   1   1          =        11
```

The discussion on binary numbers thus far only dealt with unsigned integers. In order to represent a signed number in a computer a method has to be devised to indicate the sign. Examples of such methods are, signed-magnitude, ones-complement, and twos-complement. In signed magnitude the most significant bit (MSB) is set aside to indicate the sign, zero for positive and 1 for negative. The ones-complement is obtained by complementing the bits. The MSB of a negative number will begin with a one and a positive number with a zero. 0011 will represent +3 and 1100 (complement of the positive 3) will represent -3. Ones-complement allows for simpler arithmetic manipulation circuitry, but the number zero has two different representations. In twos-complement a negative number is represented by complementing the bits of the positive number and adding a 1. In all three representations the positive numbers are represented the same way. In twos complement a +3 represented as 0011 and a –3 is represented as 1101 (complement of 0011 is 1100, and add a 1). Deciphering a twos-complement number is done as follows. If a binary number's MSB is a zero then it is a positive and no further action is necessary. If the MSB is 1, it is a negative number. Suppose the number is 10000001, we know it is a negative number. Complement the bits and add a 1, we get 01111110 +1 = 01111111 which is 127. The sign is already determined to be negative it is –127.

Everything in a digital computer must be represented in the binary format. This implies that a specific binary sequence must be established for representing the alphabets, numeric symbols and other special symbols that are in use in English as well as other languages of the world. Similar decision must be made on how images will be represented in the binary format. An agreement was reached in 1968 to represent all English alphabets using a particular sequence of zeros and ones, and this coding scheme is known as the American Standard Coding for Information Interchange (ASCII). If a program can export information into ASCII format then another program can import it. The standard ASCII character set consists of 128 decimal numbers ranging from zero through 127 assigned to letters, numbers, punctuation marks, and the most common special characters. The Extended ASCII Character Set also consists of 128 decimal numbers and ranges from 128 through 255 representing additional special, mathematical, graphic, and some foreign characters. The first 32 (0 to 31) ASCII codes are for special purposes such as line feed, carriage return, form feed, bell, etc. The ASCII can only represent languages that use Roman Alphabets such as English. A more recent standard, the Unicode provides a unique number for every character regardless of the platform, the program, or the language. Now every alphabet of every language can be represented in a computer. To represent an ASCII character we need 7 bits and an extended ASCII character requires 8 bits; therefore, we set aside a byte for it and a byte happens to be the smallest addressable memory unit. Unicode uses two bytes to represent one character. This means we can have 65,536 possible symbols in a given alphabet. Another encoding scheme that IBM mainframes used is the Extended Binary Coded Decimal Interchange Code (EBCDIC).

Other numbering schemes

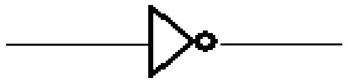
Since the basic addressable memory unit is a byte an octal (base eight) makes sense and some early computers used the octal system. Octal numbering system uses 8 symbols, 0 to 7. The next number after 7 is 10. Hexadecimal (base 16) numbers give more compactly written numbers and programmers use it very often. It uses 15 symbols 0-9 and A-F. The A stands for decimal 10, B for 11, C for 12, D for 13, E for 14 and F for 15. These 15 symbols can be represented using four bits or a nibble. For example , 1111 will represent F (decimal 15). Two hexadecimal digits can be represented in one byte; FF representing 255. In hexadecimal, after the F, the next number is 10 (decimal 16).

Boolean Operations

The mathematics used in digital systems is the special case of Boolean Algebra in which the variables only assumes only two values (0 or 1). The basic operations of Boolean Algebra are: NOT, AND, OR, XOR, NAND and NOR.

NOT Gate

The NOT operation gives the inverse (complement) of a Boolean variable. For example NOT(True) is false.



If X is the input values to the Gate and Y is the Output

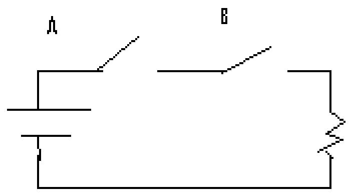
Then Y is been calculated as = $\overline{\mathbf{X}}$

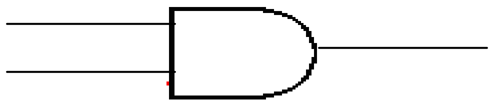
Truth Table for NOT Gate

X	**Y**
0	1
1	0

<u>AND Gate</u>

The AND operation is a logical multiplication. An example of an AND operation is, IF US CITIZEN AND AGE GREATER THAN EQUAL TO 18 THEN ELIGIBLE TO VOTE. There are two inputs, citizenship and age, and one output, voting eligibility, in this example. Both inputs and the output may assume only true or false. In a situation where a person is a US citizen and 20 years of age, the inputs will be TRUE and TRUE. The output also will be TRUE. If one or both of the inputs are FALSE then the output will be FALSE.





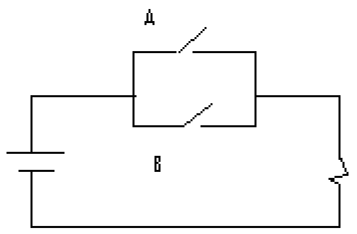
This is a two input AND gate. A and B are the input values to the AND Gate and Y is the output. TheY is been calculated as **Y = A.B**. The table formed using the different combinations of the inputs is known as the Truth Table.

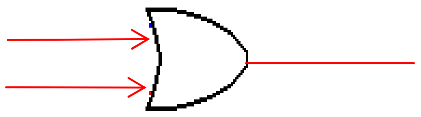
Truth Table for AND gate

A	**B**	**Y**
0	0	0
0	1	0
1	0	0
1	1	1

<u>OR Gate</u>

The OR operation is a local addition. An example of an OR operation is: If grade == 90 or grade >90 then Grade = 'A'. In an OR operation only one condition has to be true for the output to be true.





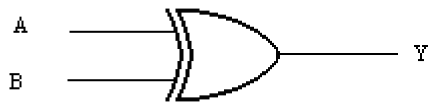
If A and B are the Input values to the Gate and Y is the Output

Then Y is been calculated as **Y = A + B**

Truth Table for OR Gate

A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	1

<u>XOR Gate</u>



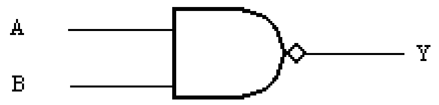
If A and B are the Input values to the Gate and Y is the Output

Then Y is been calculated as **Y=** $\mathbf{A\overline{B} + \overline{A}B}$

<u>Truth Table</u>

A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

<u>NAND Gate</u>



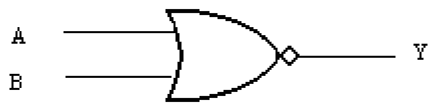
If A and B are the Input values to the Gate and Y is the Output

Then Y is been calculated as **Y=** $\mathbf{\overline{A.B}}$

<u>Truth Table</u>

A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate



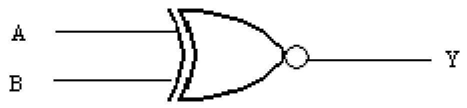
If A and B are the Input values to the Gate and Y is the Output

Then Y is been calculated as **Y=** $\overline{\mathbf{A}+\mathbf{B}}$

Truth Table

A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

XNOR gate



(1 ⊕ 0)'

If A and B are the Input values to the Gate and Y is the Output

Then Y is been calculated as **Y=** $\overline{\mathbf{A}\overline{\mathbf{B}}+\overline{\mathbf{A}}\mathbf{B}}$

Truth Table

A	**B**	**Y**
0	0	1
0	1	0

1	0	0
1	1	1

Associative Law

(A·B)·C = A·(B·C) = A·B·C
(A+B)+C = A+(B+C) = A+B+C

Distributive Law

A·(B+C) = (A·B) + (A·C)

Commutative Law

A·B = B·A
A+B = B+A

Precedence

AB = A·B
A·B+C = (A·B) + C
A+B·C = A + (B·C)

DeMorgan's Theorem

$\overline{\mathbf{A.B}} = \overline{\mathbf{A}} + \overline{\mathbf{B}}$ (NAND)
$\overline{\mathbf{A} + \mathbf{B}} = \overline{\mathbf{A}} \cdot \overline{\mathbf{A}}$ (NOR)

Given below is a C++ program to emulate a half-adder.

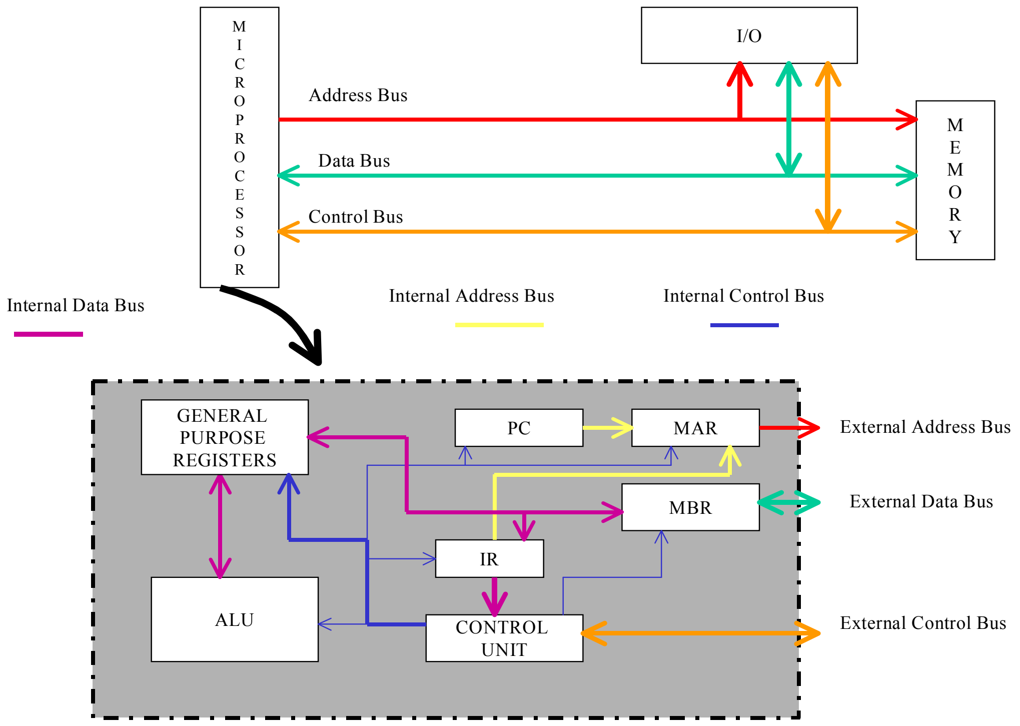
```
#include <iostream.h>

void main()
{
        short x,y, sum, carry;
        bool a,b,c,d;
        cout <<"Enter two bits to add seperated by a space ";cin >> x>>y;
        a=x==1; b=y==1;  //actually c++ assigns 0 for false and 1 for true, we could have
read these directly;
        c=(a||b) && !(a&&b);
        d=(a&&b);
        sum=c?1:0; //if c is true then sum gets 1 else sum gets 0
        carry=d?1:0;
        cout <<x <<"+"<<y <<"="<<carry<<sum<<endl;
        if (a = true) cout <<"it is true";
        if (a) cout << "it is ture";

}
```

Computer Programming Languages:
Computers perform operations such as moving data and data manipulation by activating switches and gates. Instructions to do that also must be given in the form of 1s and 0s or on and off. When we program computers using just ones and zeros we are using the "machine language". Indeed the computers were programmed this way until a machine language program was written to translate English sounding symbols into zeros and ones. Such a program is called the Assembler. One assembly language instruction will be translated into one machine instruction. The high level language compilers such as C++ and Visual Basic translate one high level instruction to several machine level instructions.

Structure of a Microprocessor

International Business Machines (IBM) used the Intel-8088 microprocessor in their first personal computer (PC) introduced in 1981. A microprocessor has two functional units, the execution unit (EU) and the bus interface unit (BIU). The EU consists of the control logic for decoding and executing instructions, an ALU for performing arithmetic and logic operations and a set of general purpose registers. The BIU contains control logic that controls all external bus operations. A bus is a set of wires that connects the various components of a computer together and transfers signals and data between them. The number of lines a bus has depends on the architecture of the microprocessor and is directly proportional to the CPU speed.

A block diagram of a hypothetical microprocessor is given in Fig____. This diagram shows the PC- Program Counter, MBR- Memory Buffer Register, MAR- Memory Address Register, ALU- Arithmetic Logic Unit, IR- Instruction Register, and the General Purpose Registers. This block diagram also shows the address bus, data bus and the control bus going out of the CPU. The interconnections (bus) connect components inside the CPU (internal bus) and components external to the CPU (external bus). There are three different functions for the bus, memory address transfers, data transfers and control signal transfers. One set of physical wires may handle all three operations one at a time, or there may be different sets of wires for each, the address bus, data bus and the control bus.



BLOCK DIAGRAM OF A MICROPROCESSOR

Fig: 1

The CPU makes extensive use of the registers, the fastest storage available to the CPU, when fetching, decoding, and executing instructions. When a program begins execution, the program counter (PC, a register inside the CPU) has the address of the next instruction to fetch; this address is placed there initially by the operating system and updated automatically by the CPU. There are three additional registers, the instruction register (IR), memory address register (MAR) and the memory buffer register (MBR) that work together to fetch the instruction. The address from the PC is moved to the MAR. The reason for this is that the address bus is connected to the MAR and all addresses issued must go through this register. Then the address contained in the MAR is placed on the address bus and the READ line is asserted on the control bus. The memory whose address is found on the address bus places its contents on the data bus. The only register that is connected to the data bus is the MBR, and all data should go in an out through this register. Thus the data on the data bus is copied into the MBR. The instruction is now copied on to the IR to free up the MBR to handle another transfer. The contents of the PC will be now incremented by one instruction. The instruction is then decoded and executed as described later in this chapter. In practice, more than one instruction is read during an instruction cycle. Additional instructions are kept in temporary storage registers such as the Instruction Buffer Register (IBR). Those registers hold memory address will be large enough to hold the maximum addressable memory and the those registers that hold instruction or data will be as large as the word size of the memory.

In actuality, the logical address found inside the PC may be converted to actual physical address by a memory management unit (MMU). This allows flexibility of relocating a program. For example, a physical address may be computed by adding the contents of a base register such as the CS, and an offset register such as the BX. There may also be a relocation register involved as well.

Additional registers that are found in the CPU are Stack Pointer (SP), Index Register (XR), Processor Status Register (XR), and Data Registers (DRs). A stack is a certain portion of the memory that keeps track of subroutine calls and other information. The SP contains the address of the top of that memory. The index register keeps track of arrays. The processor status register is used to record conditions that occur within the CPU such as an overflow. The data registers are the general purpose registers that a programmer uses to store data that are being manipulated by the CPU. Data from the memory will be brought into these registers to be manipulated by the CPU. The results of ALU operations will be stored in these data registers before being written back to the RAM.

Operations of a microprocessor

The power of a CPU is sometimes quantified by addressing capability or by how many bits of data that can be transferred during a fetch. If we say that the hypothetical processor is 16 bit, then 16 bits of data should be transferred at the same time. The power of the CPU is also denoted by the speed in which the data is transferred. We will now analyze the working of the hypothetical processor.

The size of the PC depends on the number of address lines of the CPU. Let us assume that the CPU has 16 address lines. Then the PC should be able to address 2^{16} locations, which is equal to 65,536. Initially the CPU starts its operation based on the contents of the PC. Assume that initially the PC contains the value 0000h where “h” stands for hexadecimal notation.

The Fetch Cycle:

The address of the next instruction contained in the PC is transferred to the MAR . MAR places this address on the address bus and the corresponding memory places its contents on to the data bus. The MBR is connected to the data bus and it receives the instruction or the data contained in that address. At the start up, the CPU performs an instruction fetch. Instruction fetch involves reading the instruction stored in the memory. The location from which the instruction is read is derived from the PC. A summary of the instruction fetch is given below:

MAR $\leftarrow$ [PC]

```
MBR ← [Memory]
PC  ← [PC] +1
IR  ← [MBR]
```

The Decode Cycle:

An instruction will have an opcode and none, one or more operands (data to operate on). The opcode is the operation performed by the CPU. The number of bits assigned to the opcode will determine the maximum number of instructions that processor is allowed to have. For example, suppose that the length of an instruction (opcode and operand) is 16 bits, 4 bits are allocated for the instructions and the remaining 12 bits are allocated for an operand. Four bits can give us a maximum of sixteen instructions (4^2). Each of these 16 bit patterns, ranging from decimal 0 to 15, or binary 0 to 1111, or hexadecimal 0 to F will stand for a particular instruction. For illustration purposes, an imaginary computer may have the following instruction set:

Bin	Dec	Hex	Instruction	Explanation
0000	0	0	LD	load
0001	1	1	ST	store
0010	2	2	A	add
0011	3	3	AI	add immediate
0100	4	4	S	subtract
0101	5	5	SI	subtract immediate
0110	6	6	M	multiply
..				
..				
1111	15	F	J	jump

This hypothetical computer has only one user addressable register, which is the accumulator (AC). When a load (LD) is executed, the contents of the register as indicated by the operand is brought into the AC, the reverse happens in a store (ST). When an add (A) is executed, the value contained in the memory location as indicated by the operand is added to the contents of the AC, and the result is placed back in the AC. The add immediate (AI) differs from the add (A) in that the operand is what is added, not the content of the memory pointed by the operand.

Suppose a program that is already in execution has arrived at a point that it needs to handle these two statements: B = B + A and C = B + 2. Also suppose that the variable A is kept in memory location 200h, the variable B in 201h, and the variable C in 202h. The values in each are 5, 3 and 0 respectively. The code segment for this section of the program is given in figure 2. This figure shows the memory location of each instruction, assembly mnemonics of each instruction and its equivalent machine code.

Memory Location	**Instruction**	**Machine code in Hex**	**Description of Instruction**
100h	LD A	0200	Load variable A from the memory to the accumulator
101h	A B	2201	Add accumulator with the variable B
102h	ST B	1201	Store the value in accumulator to memory
103h	AI 2	3002	Add accumulator with 2
104h	ST C	1202	Store the value in accumulator to memory

Figure 2

There are three registers that need watching, the Accumulator (AC), Program Counter (PC), and the Instruction Register (IR). The PC contains 100h, which means that the next instruction should be fetched from memory location 100 hexadecimal. The control unit fetches the instruction contained in the address indicated by the PC, which is 100h. Please refer to figure 3, step 1. The instruction 0200 is brought into the IR. The IR now contains 0200. The instruction is decoded and separated into the opcode of 0h and operand of 200h. Recall that this is based on the assumption that 4 bits (one hexadecimal digit) are used for the opcode and 12 bits (three hexadecimal digits) are used for the operand. Once the instruction is fetched the PC is automatically incremented, and now contains 101h. The opcode 0 indicates a load, and the operand is fetched from memory location 200h and loaded into the accumulator (AC). Now the accumulator contains 5. See step 2 of figure 3.

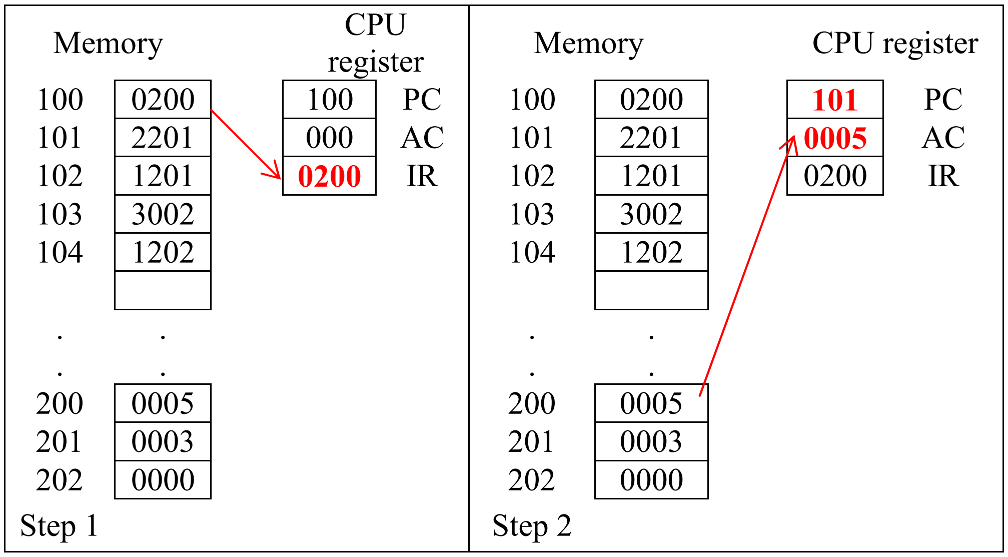


Figure 3

Next, the instruction from 101h is fetched and placed in the IR. Please refer to figure 4, step 3. The PC is incremented to 102h. The content of IR is decoded and based on the opcode of 2h, the operand located in address 201h is added to the AC. The AC now has a value of 8 (figure 4, step 4).

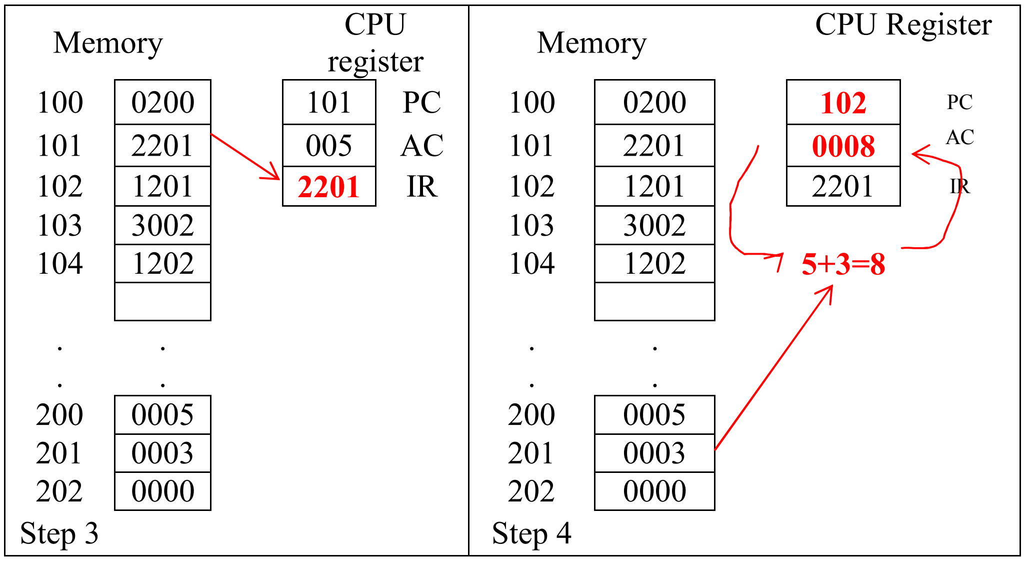
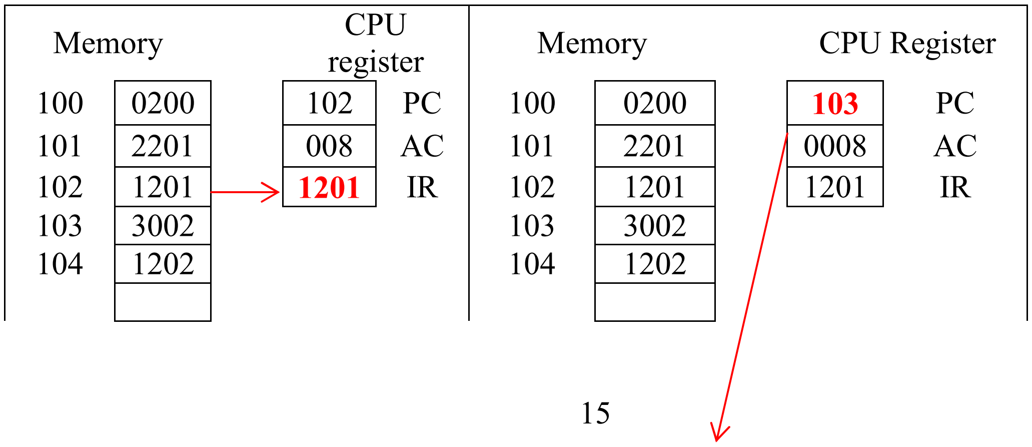


Figure 4

The next instruction whose address is contained in the PC is fetched and placed in the IR giving it a value of 1201. Please refer to Figure 5 step 5. The PC is incremented to 103h. The contents of IR (1201) is decoded and based on the opcode of 1h, the contents of the AC is saved in the address indicated by the operand, which is 201h. Address 201 (variable B) now has a value of 8 (Figure 5, step 6).



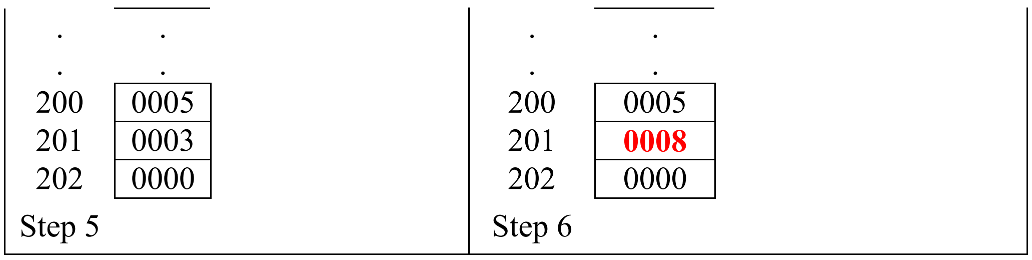


Figure 5

Next, the instruction contained in 103h (PC content) is fetched next and placed in the IR giving it a value of 3002. See Figure 6, step 7. The PC is incremented to 104h The contents of IR is decoded, and the opcode of 3h is an add immediate. No data fetching is necessary and the 2 is added to the AC. The new value of the accumulator is now Ah or decimal 10. (Figure 6, step 8).

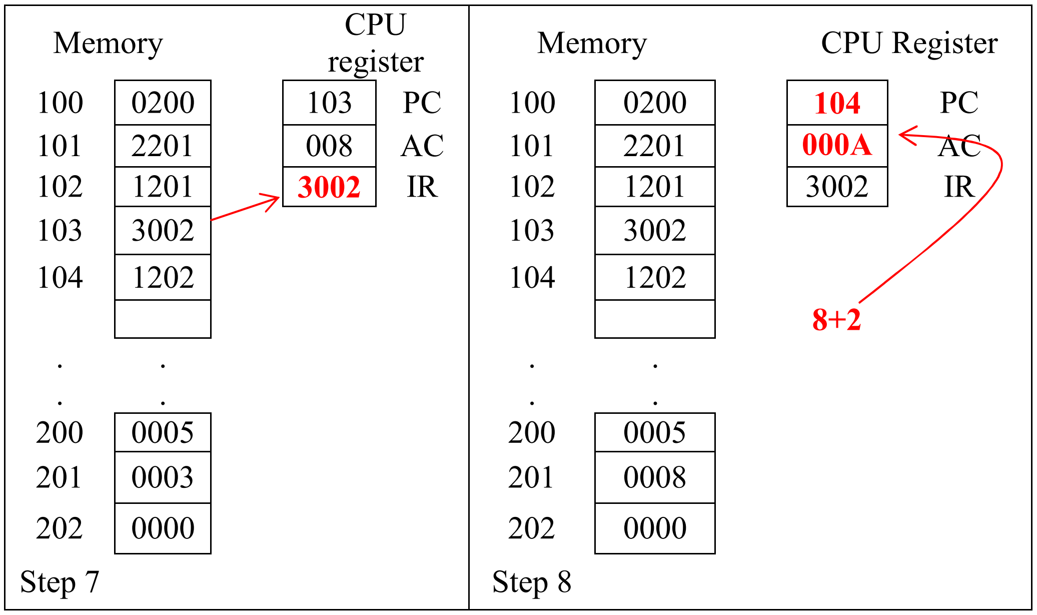
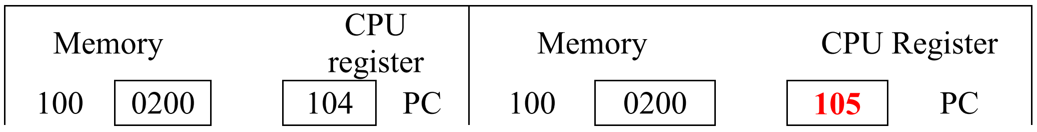


Figure 6

The last instruction is now fetched and placed in IR. Please refer to Figure 7. The PC is incremented to 105h. The PC now contains 1202, and it is decoded to give the opcode of 1h and operand of 202. Since hex 1 is a store, the value of AC (the value is Ah) is stored in memory location 202h. It was already mentioned that 202h is the memory location assigned to C.



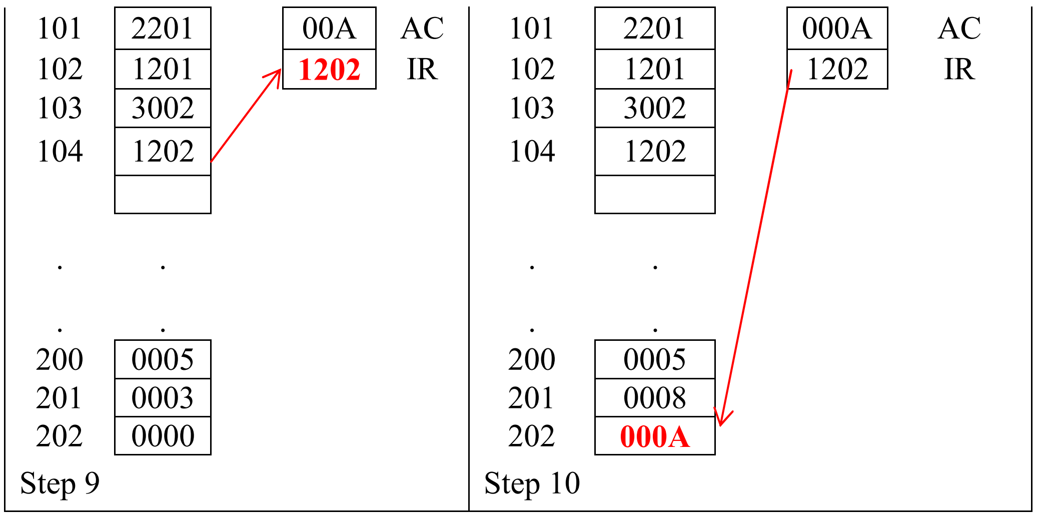


Figure 7

From the above operations one can see that defining the CPU instruction set is a difficult task. First the architect will have to decide the size of the address bus and data bus. From this the instruction format is derived and the instruction set is defined. The operation of a computer in executing a program consists of a sequence of instruction cycles with one machine instruction per cycle. Each instruction cycle is made up of smaller units. The function of the smaller units can be considered as fetch, indirect, execute and interrupt. The control unit carries out these instructions. The control unit accomplishes these steps using micro-programs and micro-operations. The elements of the program execution are shown in the figure.

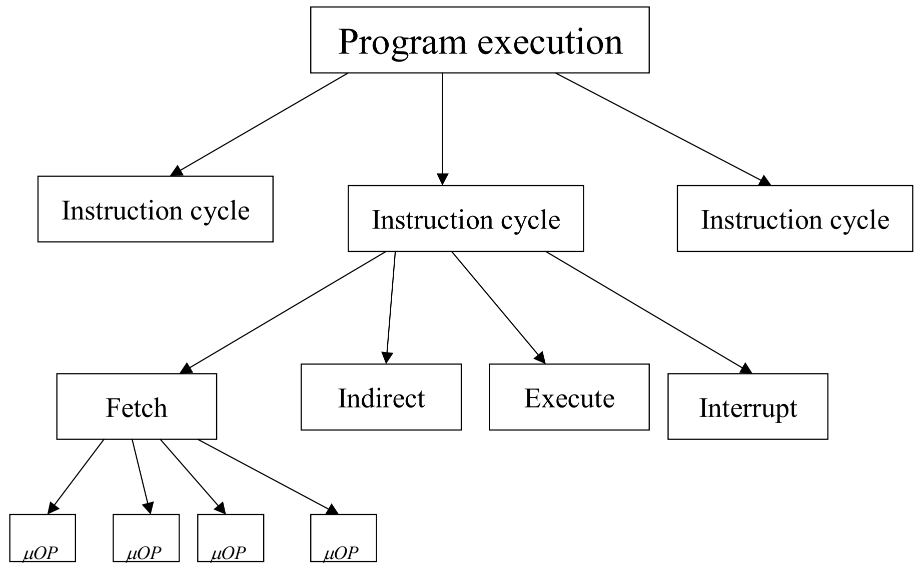


Fig: 5

Fetch Cycle:
A fetch cycle works as shown in figure 2. The proper sequence of events must be followed. Thus (MAR (PC)) must precede (MBR Memory) because the memory read operation makes use of the address in the MAR.

Conflicts must be avoided. One should not attempt to read to and write from the same register in one time unit, because the results would be unpredictable. For e.g. the microperations (MBR Memory) and (IR MBR) should not occur during the same time unit. Once an instruction is fetched then it fetches the source operands. If there is an indirect addressing involved then it moves to the indirect cycle. If it is a direct

Indirect cycle

When there is an indirect addressing to be performed, then an indirect cycle must precede the execute cycle. The dataflow is given below

$$MAR \leftarrow (IR(Address))$$
$$MBR \leftarrow Memory$$
$$IR \leftarrow (MBR(Address))$$

Fig: 6

The address field of the instruction is transferred to the MAR.

Execute cycle

This cycle depends on the opcode. For example, for the instruction ADD R1, X, the following sequence will occur.

$$\begin{aligned} &MAR \leftarrow (IR(Address)) \\ &MBR \leftarrow Memory \\ &R1 \leftarrow (R1) + (MBR) \end{aligned}$$

Fig: 7

Here the execution begins with the IR containing the ADD instruction. Initially the address portion of the IR is loaded into the MAR. Then the referenced memory location is read. The contents of R1 and MBR are added by the ALU.

Interrupt cycle

At the end of the execute cycle a test is made to determine whether any enabled interrupts have so occurred. If so the interrupt cycle occurs. The nature of the interrupt cycle varies from machine to machine. A simple illustration is given below

$$\begin{aligned} &MBR \leftarrow (PC) \\ &MAR \leftarrow Save\_address \\ &PC \leftarrow Routine\_address \\ &Memory \leftarrow (MBR) \end{aligned}$$

Fig: 8

Based on the approach explained above we can generate a flow chart for instruction cycle as given below, ICC is an instruction cycle code register. Based on the contents of the ICC register we can determine what type of instruction is to be done. An e.g. is given below

00: Fetch
01: Indirect
10: Execute
11: Interrupt

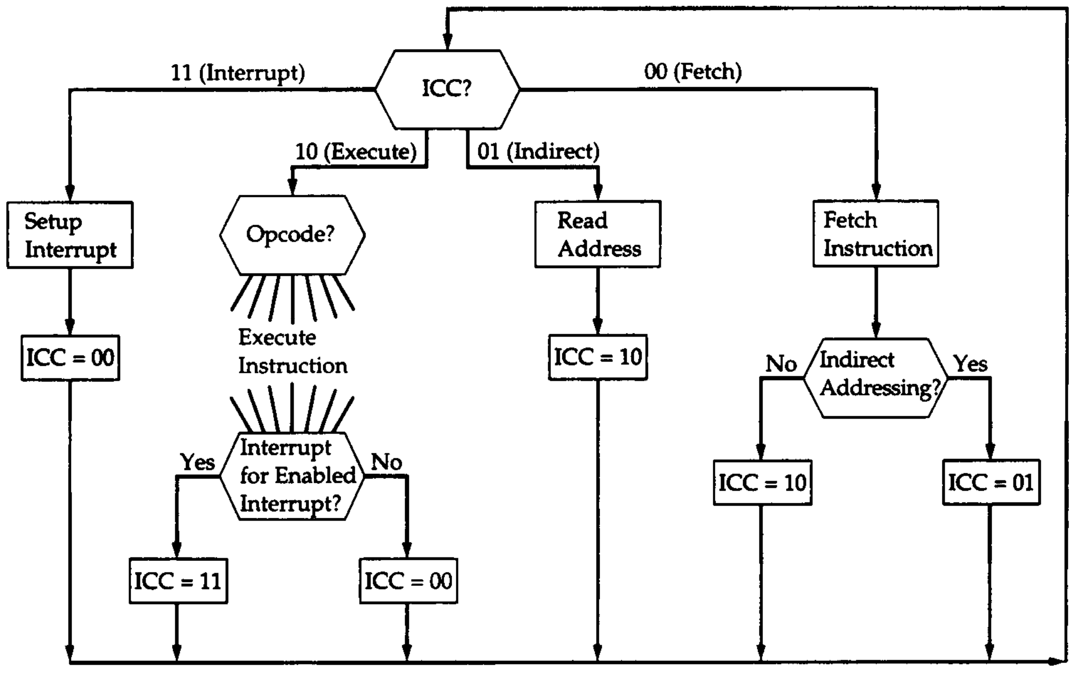


Fig: 9

We have analyzed the behavior or functioning of the processor into minute operations called micro operations. These microperations are the functional requirement of the control unit of a microprocessor. Based on the studies we have conducted we have to do the following steps initially for designing a microprocessor

1. Define the basic elements of the processor
2. Describe the micro operation
3. Determine the functions that the control unit must perform.

The basic elements of a processor are ALU, registers, internal paths and external data paths and now we have the control unit. Registers are used to store data internal to the processor. Internal data paths are used to move data between registers and between register and ALU. External data paths link registers to memory and I/O modules. External data paths are otherwise known as system bus. The control unit causes operations to happen within the processor.

The micro-operations are summarized to the following categories

Transfer data from one register to another.
Transfer data from a register to an external interface
Transfer data from an external interface to a register
Perform an arithmetic or logic operation, using registers for input and output.

How control signals are generated?

We have defined the elements that make up the processor (ALU, registers, data paths) and the micro-operations that are performed. For the control unit to perform its function, it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behavior of the system. These are the external specifications of the control unit. Internally, the control unit must have the logic required to perform its sequencing and execution functions. Let us discuss about the interaction between the control unit and the other elements of the processor. Figure 10 is a general model of the control unit, showing all of its inputs and outputs.

Clock: The control unit keeps track of its operations using the clock. The control unit causes one micro-operation (or a set of simultaneous micro-operations) to be performed for each clock pulse. This is sometimes referred to as the processor cycle time, or the clock cycle time.

Instruction register: The opcode of the current instruction is used to determine which micro-operations to perform during the execute cycle.

Flags: Flags can be considered as single bit registers. These are needed by the control unit to determine the status of the processor and the outcome of previous ALU operations. For example, for the increment-and-skip-if-zero (ISZ) instruction, the control unit will increment the PC if the zero flag is set.

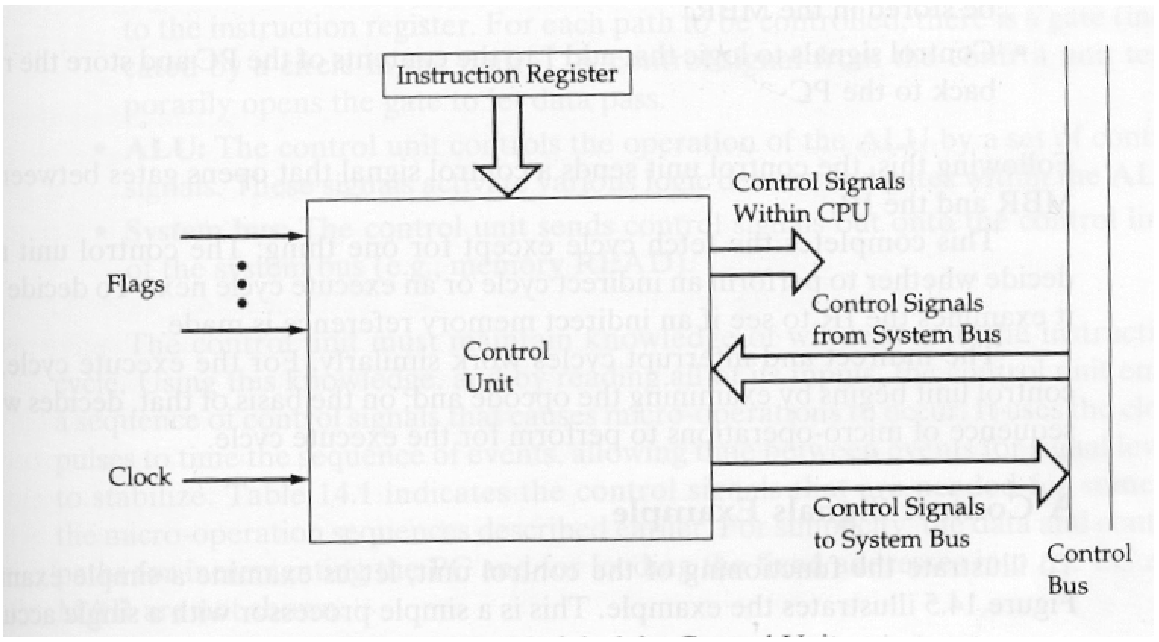

Instruction Register
Control Signals Within CPU
Flags
Control Signals from System Bus
Control Unit
Clock
Control Signals to System Bus
Control Bus

Fig: 10

Control signals from control bus: The control bus portion of the system bus provides signals to the control unit, such as interrupt signals and acknowledgements. There are two types of Control signals 1) Control signals within the processor 2) Control signals to control bus. Control signals within the processor are of two types. They are controls signals to move data moved from one register to another and those that activate specific ALU functions. Control signals to the control bus are for memory read and write, and control signals to the I/O modules.

Three types of control signals are used: those that activate an ALU function, those that activate a data path, and those that are signals on the other external system bus or other external interface. All of these signals are applied directly as binary inputs to individual logic gates.

Let us see how a fetch cycle works. The control unit keeps track of where it is in the instruction cycle. At a given point, it knows that the fetch cycle is to be performed next. The first step is to transfer the contents of the PC to the MAR. The control unit does this by activating the control signal that opens the gates between the bits of the PC and the bits of the MAR. The next step is to read a word from the memory into the MBR and increment the PC. The control unit does this by sending the following signals simultaneously:

A control signal that opens gates, allowing the contents of the MAR onto the address bus. A memory read control signal on the control bus will generate a control signal that opens the gates, allowing the contents of the data bus to be stored in the MBR. Next it will generate control signals to logic that add 1 to the contents of the PC and store the result back to the PC. Following this, the control unit sends a control signal that opens gates between the MBR and the IR. This completes the fetch cycle except for one thing. The control unit must decide whether to perform an indirect cycle or an execute cycle next. To decide this it examines the IR to see if an indirect memory reference is made. The indirect and interrupt cycles work similarly. For the execute cycle, the control unit begins by examining the opcode and, on the basis of that, decides the sequence of micro-operations to perform for the execute cycle.

Control Signals Example

To illustrate the functioning of the control unit, let us examine a simple processor

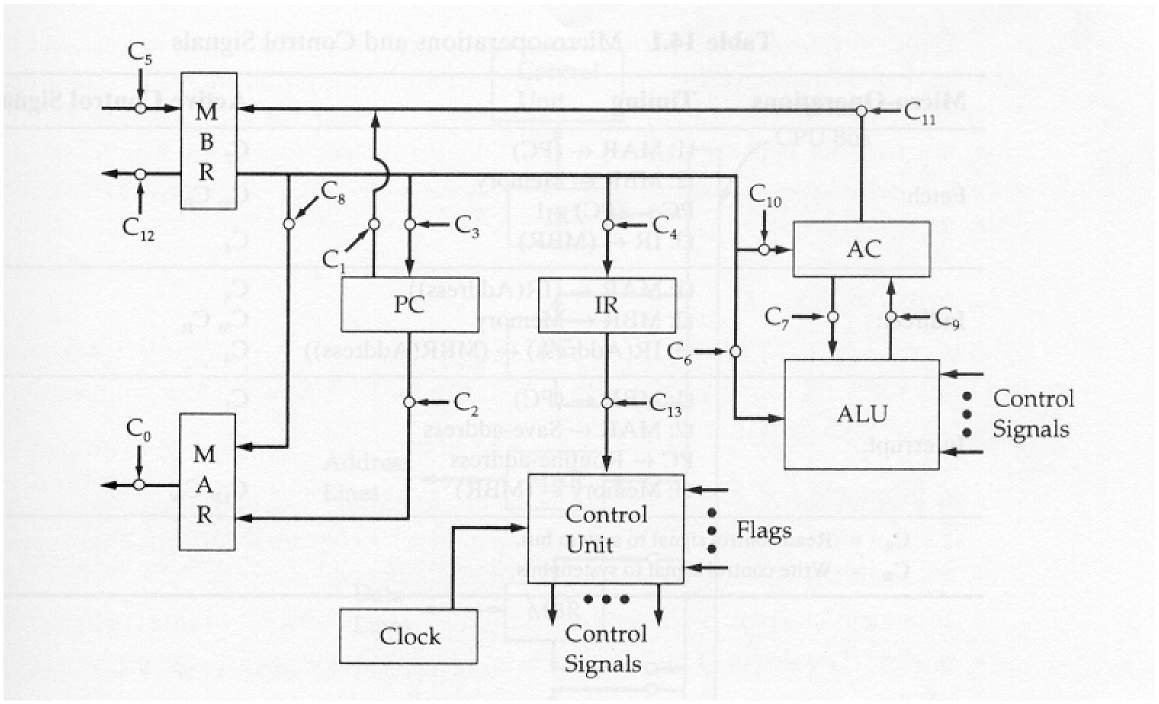


Fig: 11

This is a simple processor with a single accumulator. The data paths between elements are indicated. The control paths for signals from the control unit are not shown, but the terminations are labeled C_i and indicated by a circle. The control unit receives inputs from the clock, the instruction register, and flags. With each clock cycle, the control unit reads all of its inputs and emits a set of control signals. Control signals go to three separate destinations:

Data paths: The control unit controls the internal flow of data. For example, on the instruction fetch, the contents of the memory buffer register (MBR) are transferred to the instruction register (IR). For each path to be controlled, there is a gate (indicated by a circle in the figure). A control signal from the control unit temporarily opens the gate to let data pass.

ALU: The control unit controls the operation of the ALU by a set of control signals. These signals activate various logic devices and gates within the ALU.

System bus: The control unit sends control signals out onto the control lines of the system bus (e.g., memory READ).

The control unit must maintain knowledge of where it is in the instruction cycle. Using this knowledge, and by reading all of its inputs, the control unit emits a sequence of control signals that causes micro-operations to occur. It uses the clock pulses to time the sequence of events, allowing time between events for signal levels to stabilize. Table 14.1 indicates the control signals that are needed for some of the micro-operation sequences described earlier. For simplicity, the data and control paths for incrementing the PC and for loading the fixed addresses into the PC and MAR are not shown.

Micro-operation	Timing	Active control signal

Fetch	MAR ← (PC)
	MBR ← (memory)
	PC ← (PC)+1
	
	IR ← (MBR)
		C_2
		C_5, C_R
		
		C_4

Indirect:	MAR ← (IR(Address))
	MBR ← (memory)
	IR(Address) ← (MBR(Address))
		C_8
		C_5, C_R
		C_4

Interrupt:	MBR ← (PC)
	MAR ← Save_address
	PC ← Routine_address
	Memory ← (MBR
		C_1
		
		C_{12}, C_W

Fig: 12

Internal Processor Organisation

Fig: 13 explain an internal processor bus. The ALU and all processor registers are connected by a single internal bus. Gates and control signals are provided for movement of data onto and off the bus from each register. X and Y are two new registers other than mentioned above used for temporary storage. ALU cannot be used for storing any value. As soon as the ALU performs its function the output is available on the bus

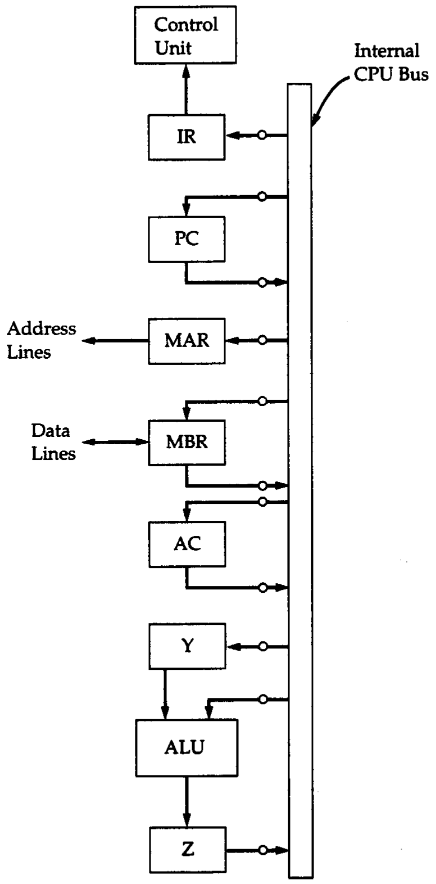


Fig: 13

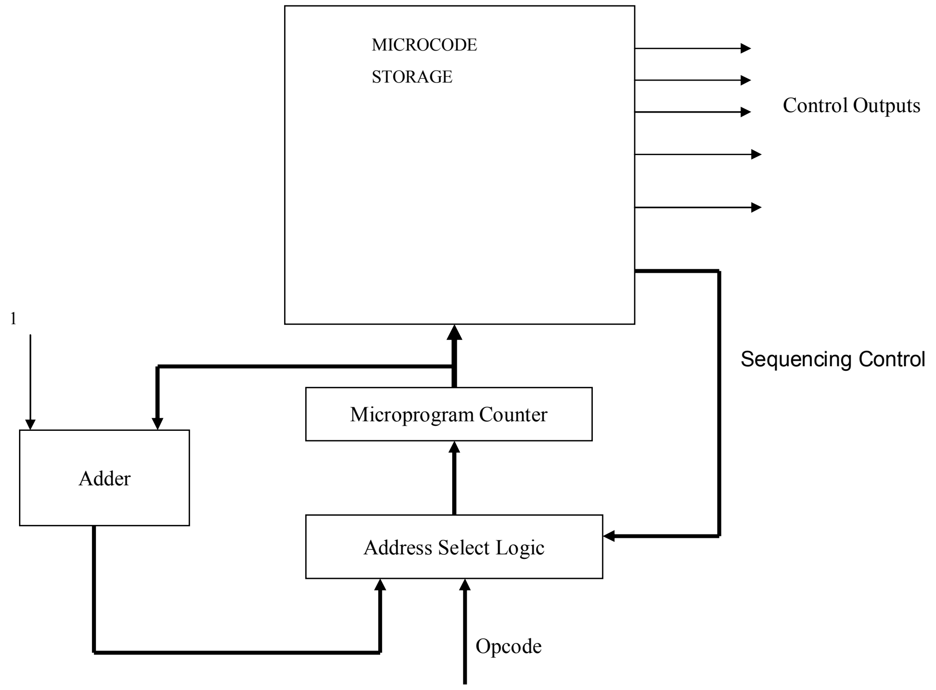
Design of Control Unit

The Control unit can be designed using hardware logics or by using a control memory. In hardware type of design the logics are designed using basic building digital circuits like AND, OR, NOR etc. The signals to the C_i are generated using the Opcode. The hardware design can be generated by implementing Truth Table, K-maps and also using State machine. The implementation of hardware type of control logic is expensive, but faster. The second method is to use a control memory. In this case we have to program the control memory. This method is called microprogramming. A simple example of programming the control memory is shown below

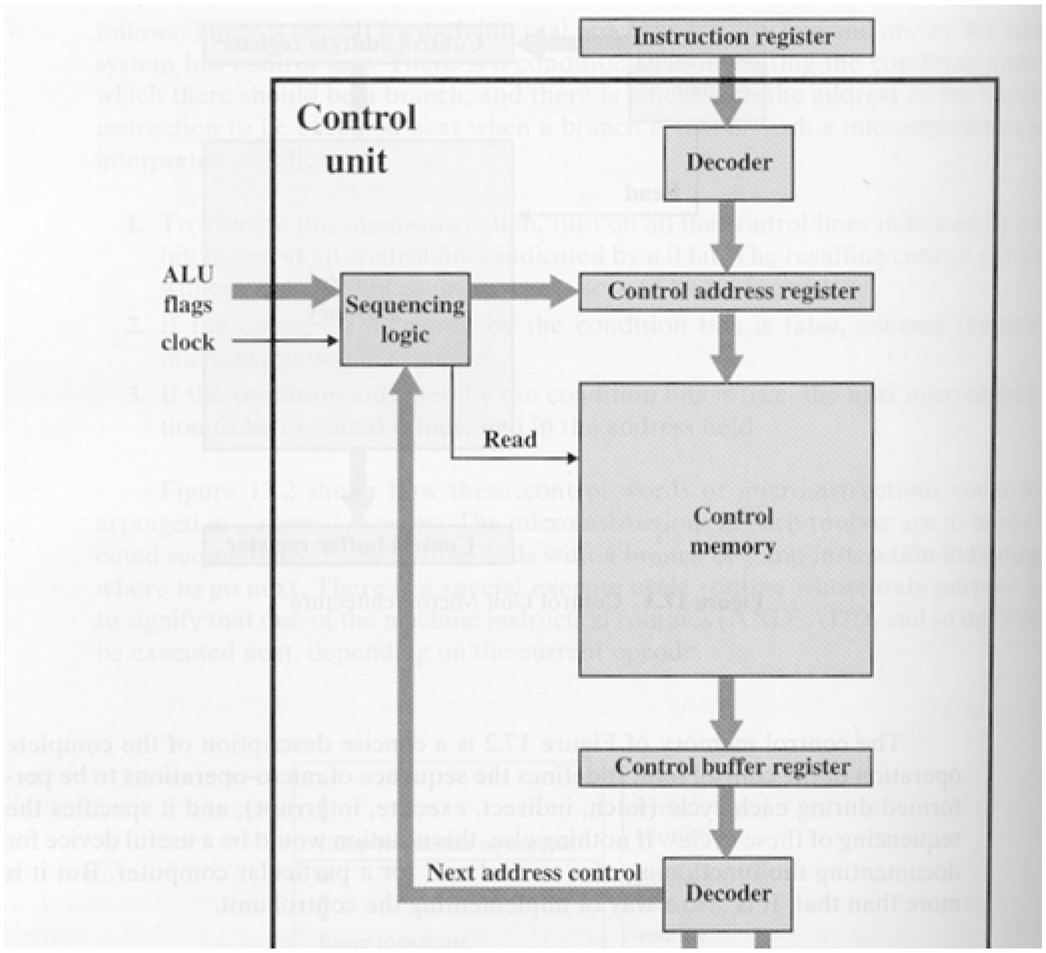
Opcode				Control Outputs				
				C_2 C_5, C_R C_4	C_2 C_5, C_R C_4	C_2 C_5, C_R C_4	C_2 C_5, C_R C_4	C_2 C_5, C_R C_4
0	0	0	0	1	0	0	0	0

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

The table contained in the Cs will be stored in the control memory which is called microcode storage.



The micro program is directly stored in the memory. The micro program counter determines the next microinstruction. The combination of the current micro program counter, incrementer, dispatch tables and address select logic forms a sequencer that selects the next microinstruction. The microcode can be implemented using ROM, or PLA (Programmable Logic Array). The sequential control is implemented using finite state machine which. Design of a state machine is an entirely different subject.



Next we will discuss the various stages of a simple program control. This figure is based on a different architecture and different instruction format. The Architecture is almost similar to the hypothetical microprocessor. Here the address lines, data lines and control lines are shown more explicitly. Let us implement the instruction

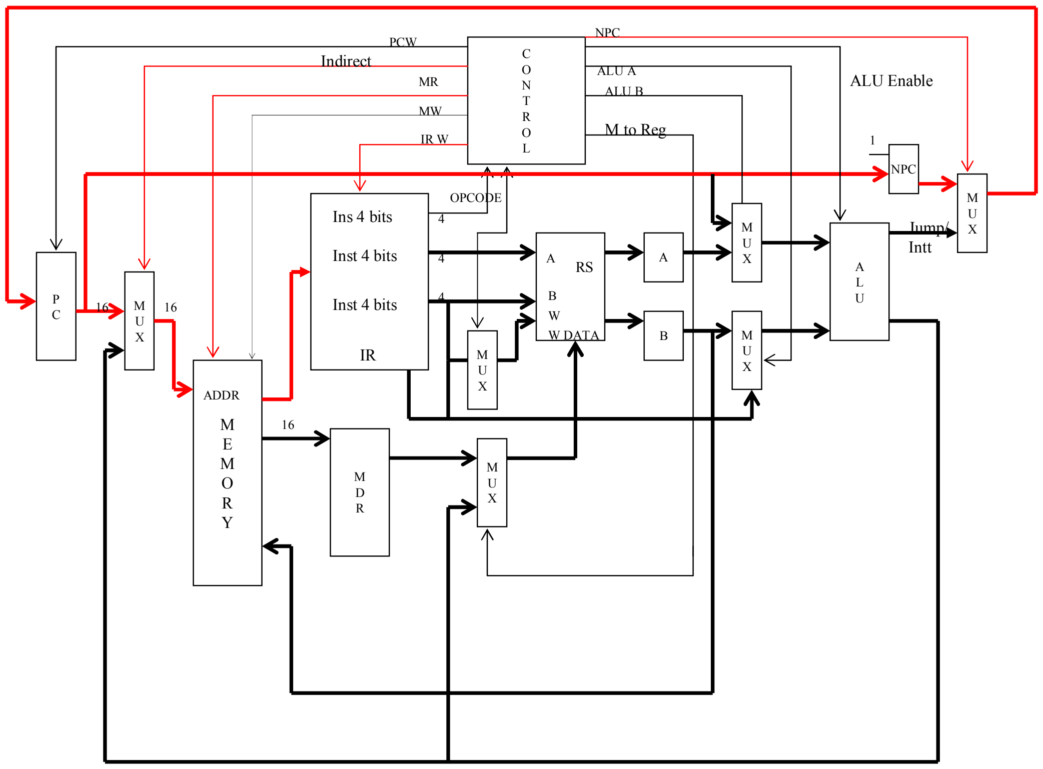
ADD A, B

This instruction performs the operation addition where the contends of the register A is added to the contends of Register B and stored in the Register B. Here we have to do an instruction fetch, decode and execute cycle. There is only one fetch operation (different from the way which the accumulator architecture was discussed)

The instruction format is as follows

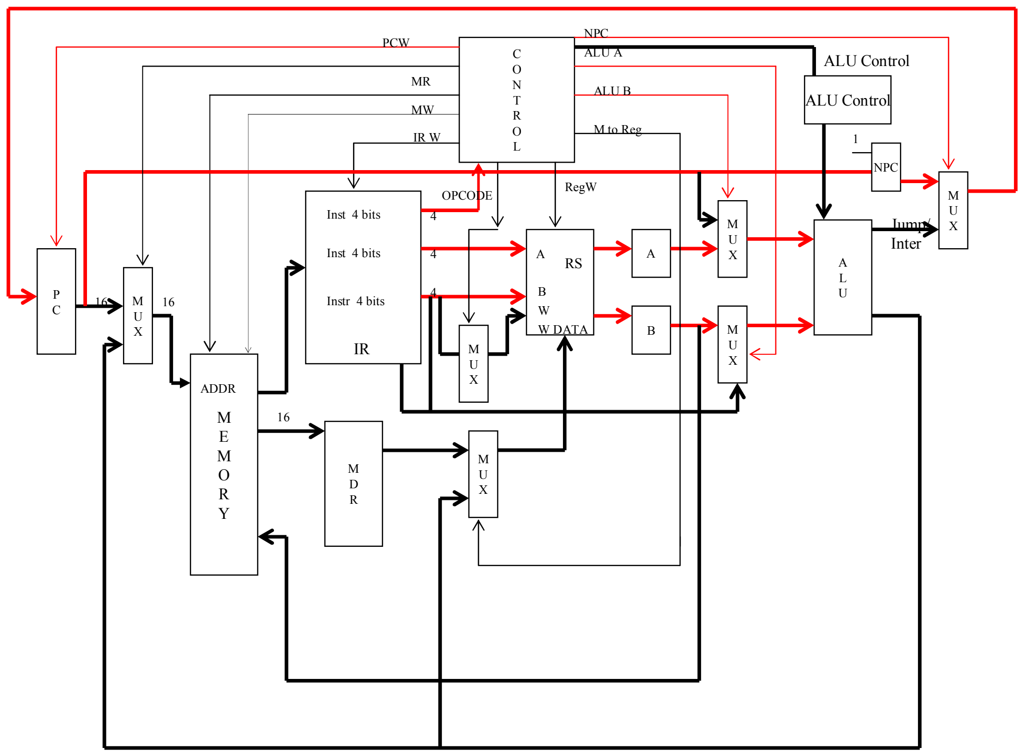
Opcode 4 bits	Reg 1 4 bits	Reg 2 4 bits	

In the above instruction there is no memory fetch and hence memory write is not taking place.



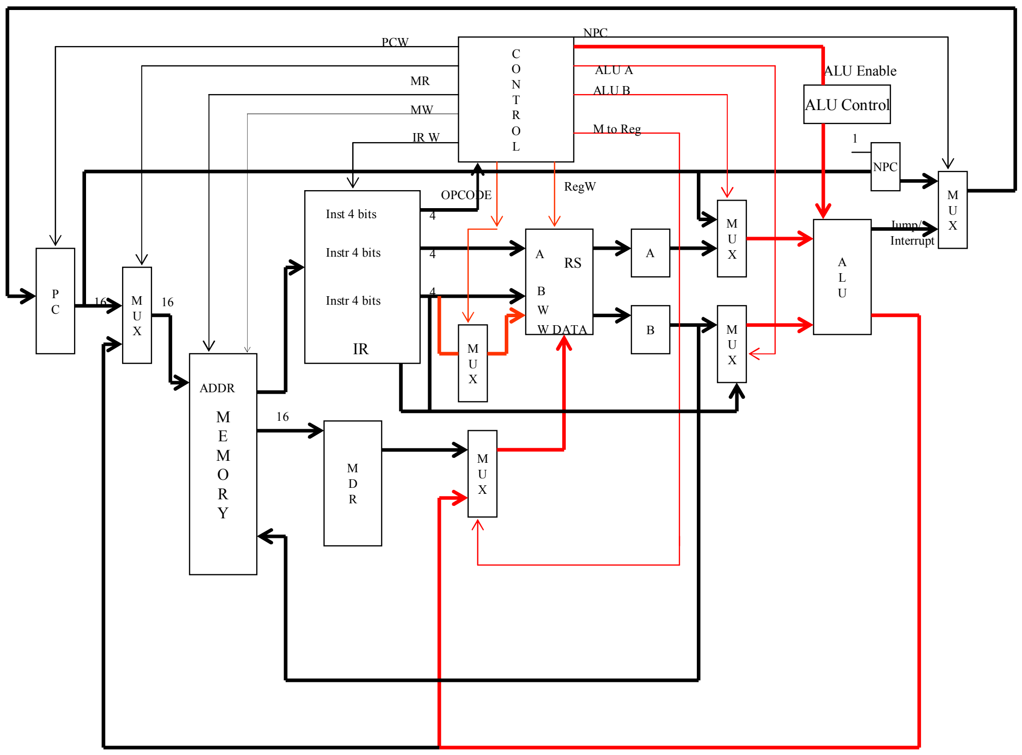
ADD A,B Instruction fetch

During Instruction fetch only the signals marked with red color are active. The operation starts with PC. The PC will generate a 16 bit address. The MUX will select only the address from PC since it is an instruction fetch (Control signal is Indirect). (The address from the ALU will be selected when there is an indirect addressing). Then there is a Memory read enable signal (Control signal is Memory Read: MR). The data read from the memory is written into the Instruction register enabled by the control signal IRW. Now the instruction register have the contends for the next stage which is Decode stage.



ADD A,B Decode

Based on the Opcode in the instruction register the Control circuit decides which all signal has to be generated. First thing to do is to increment the program counter. The control signal for incrementing the PC is PC write. The block shown as the RS contains all the registers. Based on the IR it selects the registers and outputs the contends of the registers into the temporary register A and B. ALU (Arithmetic Logic Unit) requires two inputs to perform the operation. Before the input to the ALU there are two muxes. These muxes will be selected in such a way that only the outputs from the temporary registers will be transferred to the ALU. What type of operation should the ALU perform depends on the next stage which is execute stage.



ADD A,B Execute

The operation which ALU should perform depends on the ALU Control. In this case it is addition. So the ALU control will send out Control signals for addition. In the ALU control the control signal will be multiple. The result of the addition has to be stored in the register. For this the output of ALU will be fed to a MUX (this MUX selects the data from the ALU or from the memory based on the control signal Memory to Register. In this case the control signal will be given in such a way that only the data from the ALU will be select. The output will be written into the register. Which register to be written will be given by the RegisDest Mux.