RGPVNOTES.IN

Subject Name: **Computer Architecture**

Subject Code: **IT-4005**

Semester: **4th**

UNIT-II

Fixed-Point representation

Fixed Point Representation

The positive numbers are represented as unsigned numbers but for negative values. For example, the arithmetic uses plus '+' or minus '-'sign to indicate positive or negative numbers. But in binary notation, o is used to indicate positive and 1 is used to indicate negative numbers. In addition to the sign, a number may have a binary or decimal point to represent fractions, integers or mixed integers.

Integer representation

When the number is positive, a '0' is used to represent the positive number or when the number is negative, the sign is represented by 1. And, the rest of the number is represented as by any one of the following method.

- Signed Magnitude representation
- Signed 1's complement representation
- Signed 2's complement representation

Consider, an 8-bit representation of +14

- Signed magnitude representation

+14 ss 00001110
-14 ss 10001110

- Signed 1's complement representation

+14 ss 01110001
-14 ss 11110001

- Signed 2's complement representation

+14 ss 01110010
-14 ss 11110010

Addition and Subtraction

Four basic computer arithmetic operations are addition, subtraction, division and multiplication.

Addition and Subtraction with signed magnitude

Consider two numbers having magnitude A and B. When the signed numbers are added or subtracted, there can be 8 different conditions depending on the sign and the operation performed as shown in the table below:

Operation	Add magnitude	When A > B	When A < B	When A ss B
(+A) + (+B)	+(A + B)	--	--	--
(+A) + (-B)	--	+(A - B)	-(B - A)	+(A - B)
(-A) + (+B)	--	-(A - B)	+(B - A)	+(A - B)
(-A) + (-B)	-(A + B)	--	--	--
(+A) - (+B)	--	+(A - B)	-(B - A)	+(A - B)
(+A) - (-B)	+(A + B)	--	--	--
(-A) - (+B)	-(A + B)	--	--	--
(-A) - (-B)	--	-(A - B)	+(B - A)	+(A - B)

From the table, we can derive an algorithm for addition and subtraction as follows:

Addition (Subtraction) Algorithm:

- When the signs of A & B are identical, add the two magnitudes and attach the sign of A to the result.
- When the sign of A & B are different, compare the magnitude and subtract the smaller number from the large number. Choose the sign of the result to be same as A if A > B, or the complement of the sign of A if A < B. If the two numbers are equal, subtract B from A and make the sign of the result positive.

Hardware Implementation

The hardware consists of two registers A and B to store the magnitudes, and two flip-flops As and Bs to store the corresponding signs. The results can be stored in the register A and As which acts as an accumulator. The subtraction is performed by adding A to the 2's complement of B. The output carry is transferred to the flip-flop E. The overflow may occur during the add operation which is stored in the flip-flop A Ë... F. When m ss 0, the output of E is transferred to the adder without any change along with the input carry of '0". The output of the parallel adder is equal to A + B which is an add operation. When m ss 1, the content of register B is complemented and transferred to parallel adder along with the input carry of 1. Therefore, the output of parallel is equal to A + B' + 1 ss A – B which is a subtract operation.

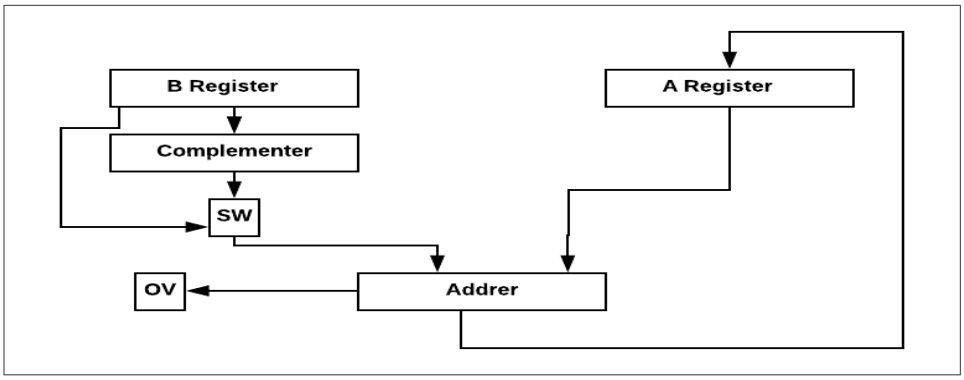

Fig 2.2 Hardware Implementation of addition & subtraction

As and Bs are compared by an exclusive-OR gate. If outputss0, signs are identical, if 1 signs are different.

- For Add operation, identical signs dictate addition of magnitudes and for operation identical signs dictate addition of magnitudes and for *subtraction*, different magnitudes dictate magnitudes be added. Magnitudes are added with a microoperation EA
- Two magnitudes are subtracted if signs are different for add operation and identical for subtract operation. Magnitudes are subtracted with a microoperation EA ss B and number (this number is checked again for 0 to make positive 0 [Asss0]) in A is correct result. E ss 0 indicates A < B, so we take 2's complement of A.

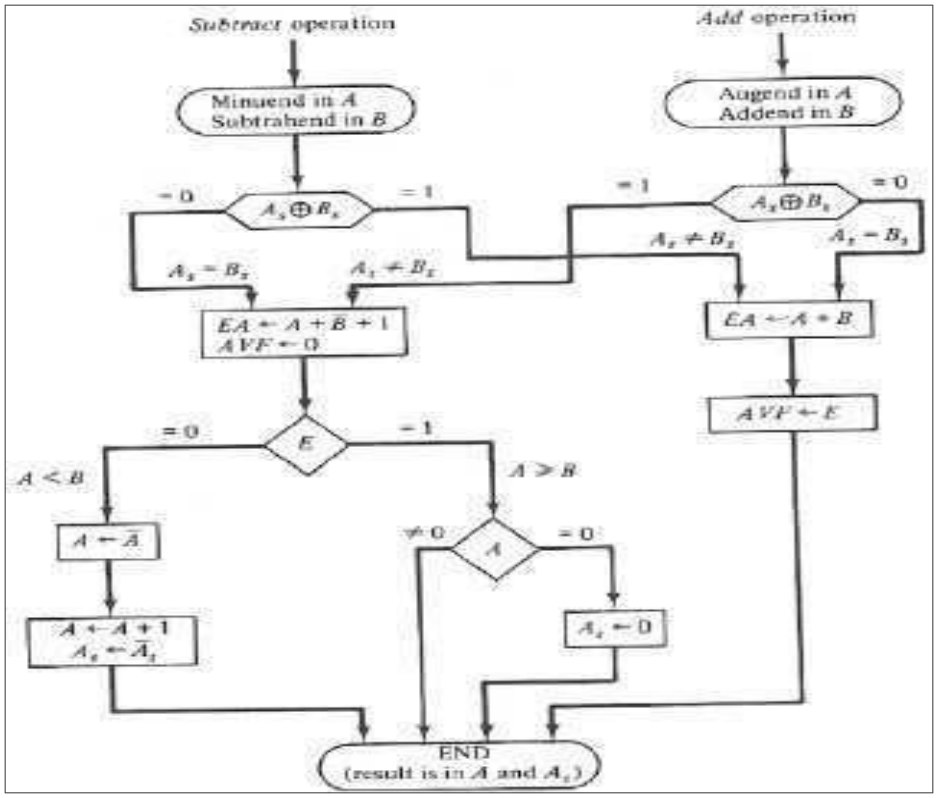

Fig 2.3 Flowchart of addition & subtraction

Multiplication Hardware Implementation and Algorithm

Generally, the multiplication of two final point binary number in signed magnitude representation is performed by a process of successive shift and ADD operation. The process consists of looking at the successive bits of the multiplier (least significant bit first). If the multiplier is 1, then the multiplicand is copied down otherwise, 0's are copied. The numbers copied down in successive lines are shifted one position to the left and finally, all the numbers are added to get the product.
The hardware for the multiplication of signed magnitude data is shown in the figure below.

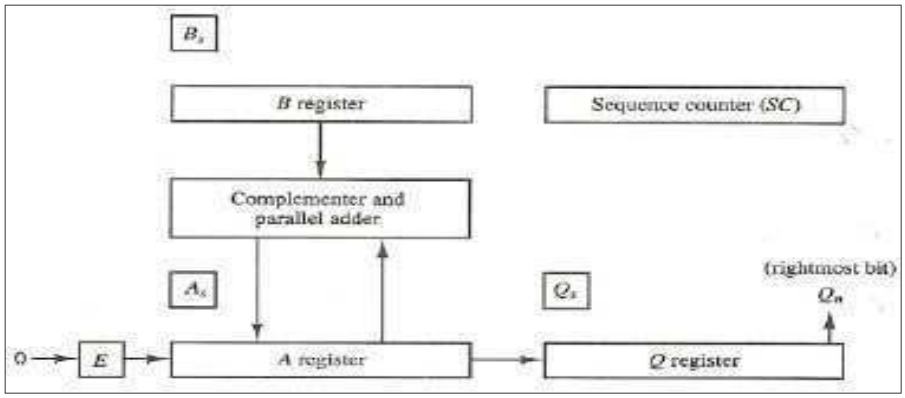

Fig 2.4 **Multiplication Hardware Implementation**

Initially, the multiplier is stored q register and the multiplicand in the B register. A register is used to store the partial product and the sequence counter (SC) is set to a number equal to the number of bits in the multiplier. The sum of A and B form the partial product and both shifted to the right using a statement "Shr EAQ" as shown in the hardware algorithm. The flip flops As, Bs & Qs store the sign of A, B & Q respectively. A binary '0" inserted into the flip-flop E during the shift right.

Hardware Algorithm

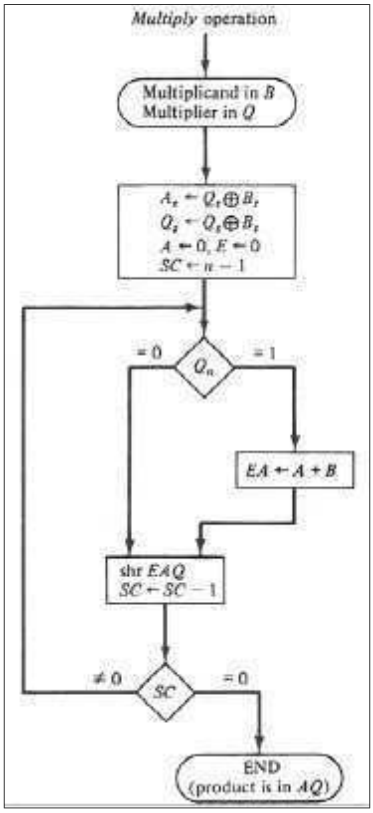

Fig 2.5 **Multiplication Flowchart**

Example: Multiply 23 by 19 using multiply algorithm.

multiplicand	E	A	Q	SC
Initially,	0	00000	10011	101(5)
Iteration1(Qnss1), add B first partial product shrEAQ,	0	00000 +10111 10111		
	0	01011	11001	100(4)
Iteration2(Qnss1) Add B Second partial product shrEAQ,	1	01011 +10111 00010	11001	
	0	10001	01100	011(3)
Iteration3(Qnss0) shrEAQ,	0	01000	10110	010(2)
Iteration4(Qnss0) shrEAQ,	0	00100	01011	001(1)
Iteration5(Qnss1 Add B Fifth partial product shrEAQ,	0	00100 +10111 11011	01011	
	0	01101	10101	000
Final Production AQ	0110110101			

The final product is in register A & Q. therefore, the product is 0110110101.

Booth Algorithm

The algorithm that is used to multiply binary integers in signed 2's complement form is called booth multiplication algorithm. It works on the principle that the string 0's in the multiplier doesn't need addition but just the shifting and the sting of 1's from bit weight 2^k to 2^m can be treated as $2^{k+1} - 2^m$ (Example, +14 ss 001110 ss $2^{3ss1} - 2^1$ ss 14). The product can be obtained by shifting the binary multiplication to the left and subtraction the multiplier shifted left once.

According to booth algorithm, the rule for multiplication of binary integers in signed 2's complement form are:

- The multiplicand is subtracted from the partial product of the first least significant bit is 1 in a string of 1's in the multiplicand.
- The multiplicand is added to the partial product if the first least significant bit is 0 (provided that there was a previous 1) in a string of 0's in the multiplier.
- The partial product doesn't change when the multiplier bit is identical to the previous multiplier bit.

This algorithm is used for both the positive and negative numbers in signed 2's complement form. The hardware implementation of this algorithm is in figure below:

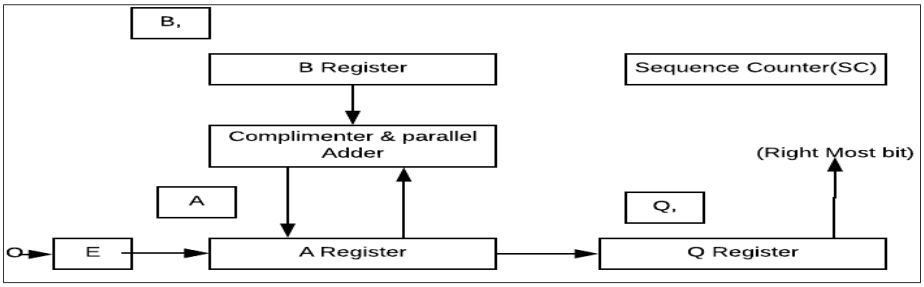

Fig 2.6 Booth hardware implementation

Numerical Example: Booth algorithm

BRss10111(**Multiplicand**)

QRss10011(**Multiplier**)

Array Multiplier

The multiplication algorithm first checks the bits of the multiplier one at time and form partial product. This is a sequential process that requires a sequence of add and shift microoperation. This method is complicated and time consuming. The multiplication of 2 binary numbers can also be done with one microoperation by using combinational circuit that provides the product all at once.

Example.

Consider that the multiplicand bits are b1 and b0 and the multiplier bits are a1 and a0. The partial product is c3c2c1c0. The multiplication two bits a0 and a1 produces a binary 1 if both the bits are 1, otherwise it produces a binary 0. This is identical to the AND operation and can be implemented with the AND gates as shown in figure.

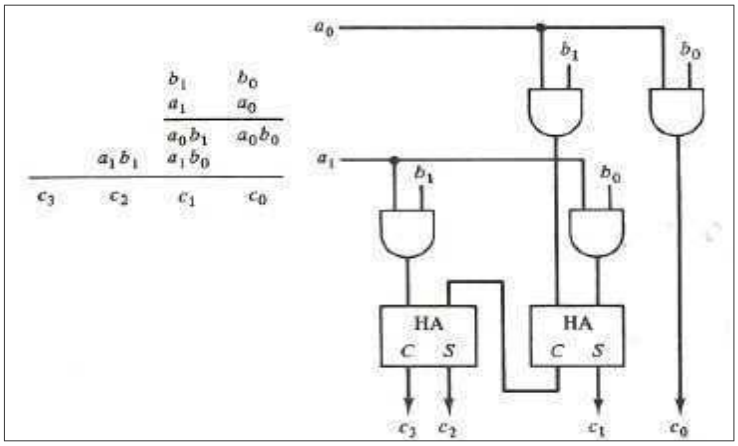

Fig 2.7 2-bit by 2-bit array multiplier

Division Algorithm

The division of two fixed point signed numbers can be done by a process of successive compare shift and subtraction. When it is implemented in digital computers, instead of shifting the divisor to the right, the dividend or the partial remainder is shifted to the left. The subtraction can be obtained by adding the number A to the 2's complement of number B. The information about the relative magnitudes of the information about the relative magnitudes of numbers can be obtained from the end carry,

Hardware Implementation

The hardware implementation for the division signed numbers is shown id the figure.

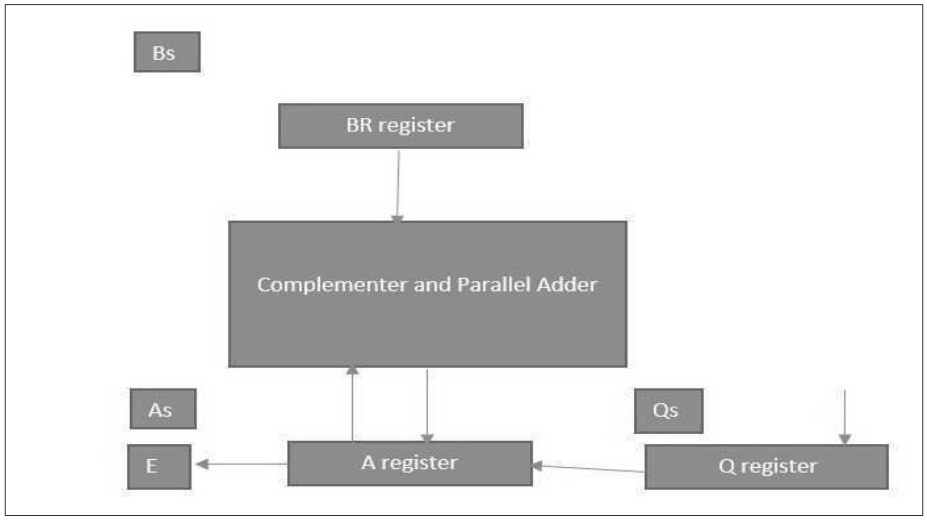

Fig 2.8 Division Algorithm

The divisor is stored in register B and a double length dividend is stored in register A and Q. the dividend is shifted to the left and the divider is subtracted by adding twice complement of the value. If E ss 1, then A >ss B. In this case, a quotient bit 1 is inserted into Qn and the partial remainder is shifted to the left to repeat the process. If E ss 0, then A > B. In this case, the quotient bit Qn remains zero and the value of B is added to restore the partial remainder in A to the previous value. The partial remainder is shifted to the left and approaches continues until the sequence counter reaches to 0. The registers E, A & Q are shifted to the left with 0 inserted into Qn and the previous value of E is lost as shown in the flow chart for division algorithm.

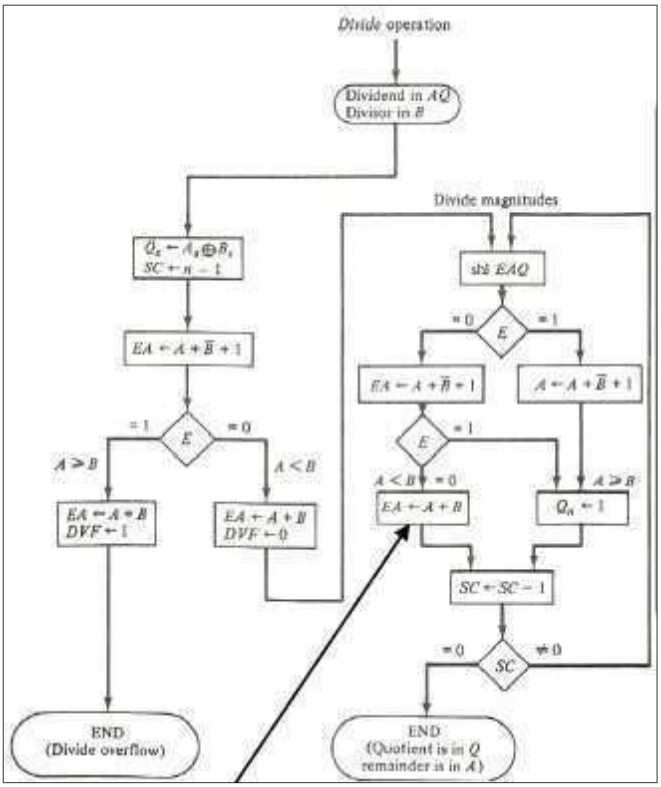

Fig 2.9 Flowchart division algorithm

This algorithm can be explained with the help of an example.
Consider that the divisor is 10001 and the dividend is 01110.

Divisor B = 10001, $\overline{B} + 1$ = 01111

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B} + 1$		01111		
$E = 1$	1	01011		
Set $Q_n = 1$	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00101		
Set $Q_n = 1$	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	11001	00110	
Add B		10001		
Restore remainder	1	01010		2
shl EAQ	0	10100	01100	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00011		
Set $Q_n = 1$	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	10101	11010	
Add B		10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Fig 2.10 Division Algorithm

Restoring method

Method described above is restoring **method** in which partial remainder is restored by adding the divisor to the negative result. Other methods:

Non-restoring method: In contrast to restoring method, when A -B is negative, B is not added to restore A but instead, negative difference is shifted left and then B is added. How is it possible? Let's argue:
Divide Overflow
The division algorithm may produce a quotient overflow called dividend overflow. The overflow can occur of the number of bits in the quotient are more than the storage capacity of the register. The overflow flip-flop DVF is set to 1 if the overflow occurs.
The division overflow can occur if the value of the half most significant bits of the dividend is equal to or greater than the value of the divisor. Similarly, the overflow can occuessr if the dividend is divided by a 0. The overflow may cause an error in the result or sometimes it may stop the operation. When the overflow stops the operation of the system, then it is called divide stop.

Floating-Point Representation
The floating-point representation of a number has two parts: *mantissa* and *exponent*
Mantissa: represents a signed, fixed-point number. Maybe a fraction or an integer
Exponent: designates the position of the decimal (or binary) point
Example1: decimal number +6132.789 is represented in floating-point as:
Fraction exponent
+0.6132789 +04

Floating-point is interpreted to represent a number in the form: $m * r^e$. Only the mantissa m and exponent *e* are physically represented in resisters. The radix *r* and the radix point position are always .
Example2: binary number +1001.11 is represented with an 8-bit fraction and 6-bit exponent as,
Fraction exponent
+01001110 000100
or equivalently,
$m * 2^e$ ss $+(.1001110)2 * 2^{+4}$
Normalization
A floating-point number is said to be *normalized* if the most significant digit of the mantissa is nonzero. For, example, decimal number 350 is normalized but 00035 is not.

Hardwired & micro-programmed control unit
To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired Control unit and Micro-programmed control unit.

Hardwired Control Unit–

The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as "hardwired".

- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.
- Hardwired control is faster than micro-programmed control.
- A controller that uses this approach can operate at high speed.

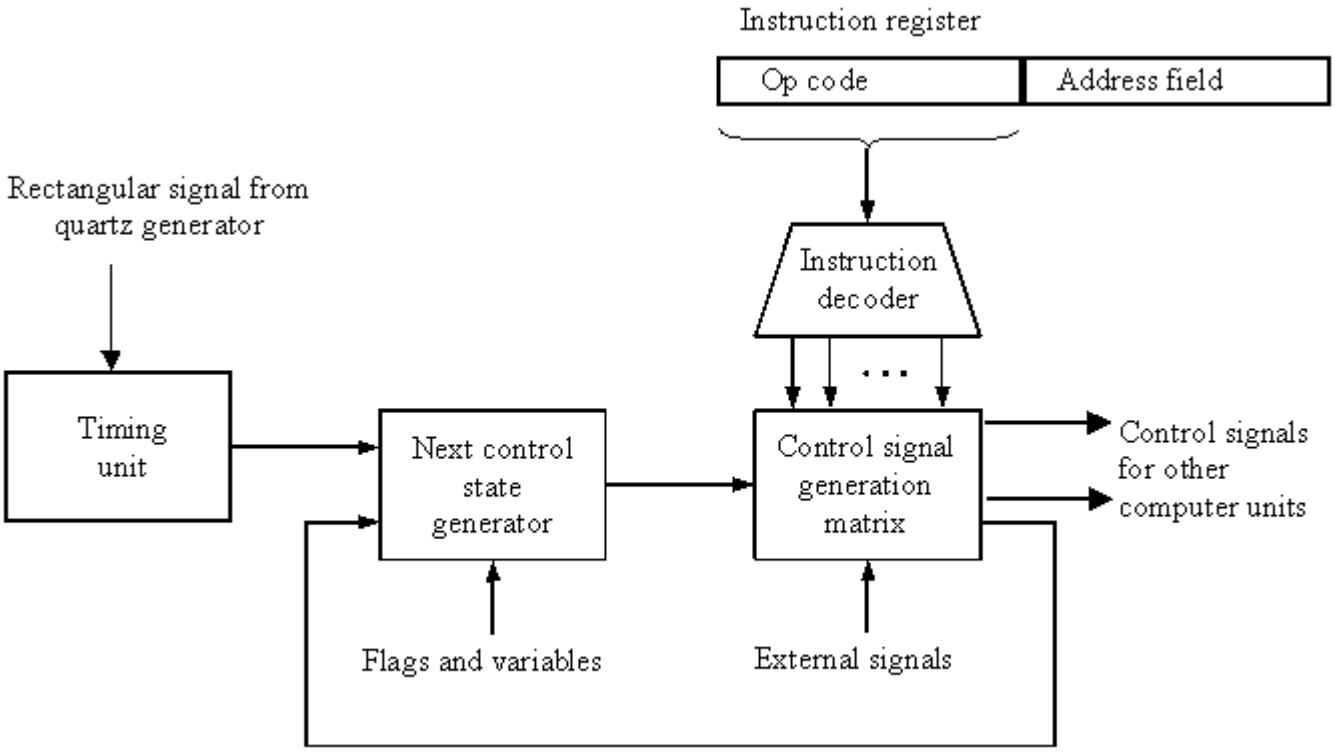

Fig2.11 Hardwired Control Unit

Micro-programmed Control Unit –

- The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.
- Control signals are generated by a program are similar to machine language programs.
- Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

Terminology:

- **Control Word :** A control word is a word whose individual bits represent various
- control signals.
- **Micro-routine :** A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.
- **Micro-instruction :** Individual control words in this micro-routine are referred to as microinstructions.
- **Micro-program :** A sequence of micro-instructions is called a micro-program, which is stored in a ROM or RAM called a Control Memory (CM).
- **Control Store :** the micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the Control Store.

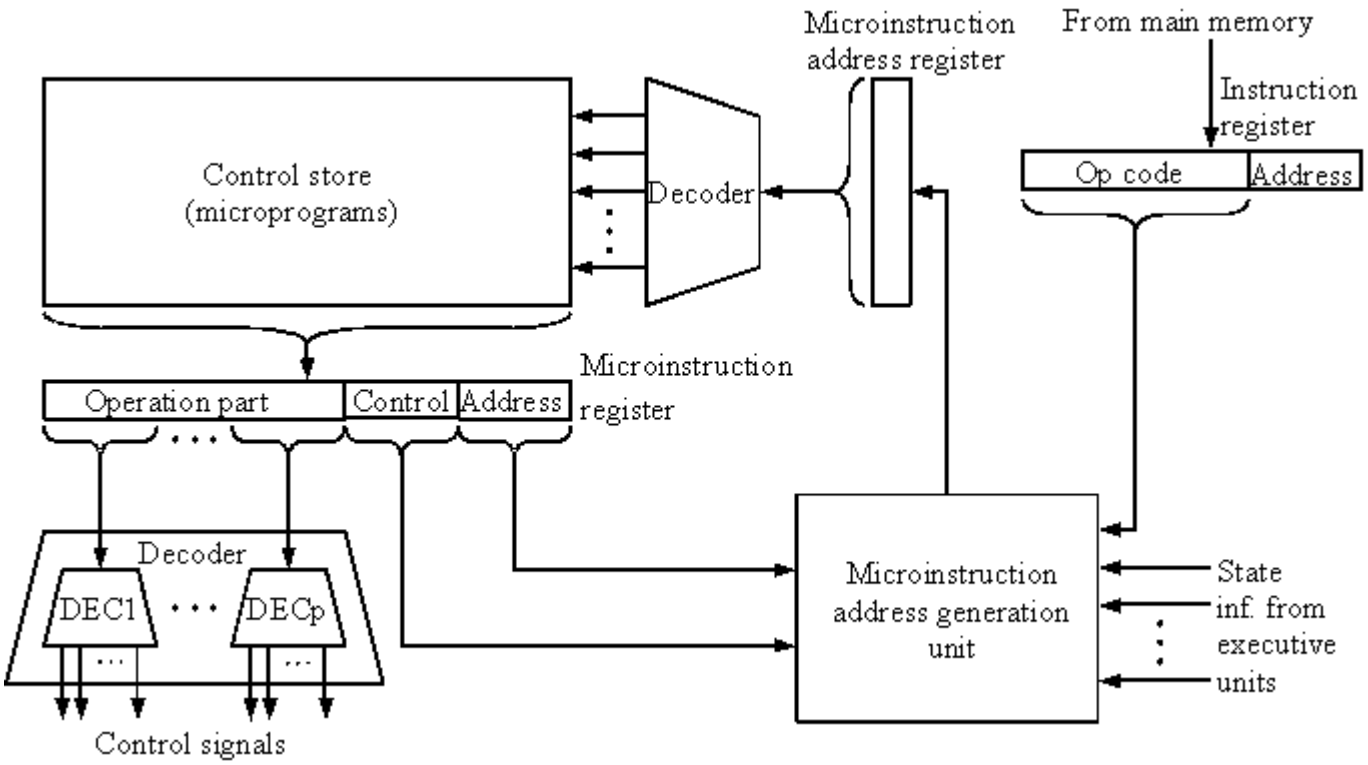

Fig 2.12 Microprogramed Control Unit

Control Memory

* The control unit in a digital computer initiates sequences of micro operations.
* The complexity of the digital system is derived form the number of sequences that are performed
* When the control signals are generated by hardware, it is hardwired
* In a bus-oriented system, the control signals that specify micro operations are groups of bits that select the paths in multiplexers, decoders, and ALUs.
* The control unit initiates a series of sequential steps of micro operations
* The control variables can be represented by a string of 1’s and 0’s called a control word
* A micro programmed control unit is a control unit whose binary control variables are stored in memory .
* The control unit initiates a series of sequential steps of micro operations
* The control variables can be represented by a string of 1’s and 0’s called a
* control word
* A micro programmed control unit is a control unit whose binary control variables are stored in memory
* A sequence of microinstructions constitutes a micro program
* The control memory can be a read-only memory
* Dynamic microprogramming permits a micro program to be loaded and uses a writable control memory
* A computer with a micro programmed control unit will have two separate
* memories: a main memory and a control memory
* The micro program consists of microinstructions that specify various internal control signals for execution of register micro operations
* These microinstructions generate the micro operations to:

* fetch the instruction from main memory
* evaluate the effective address
execute the operation
* Return control to the fetch phase for the next instruction.
* The control memory address register specifies the address of the microinstruction
* The control data register holds the microinstruction read from memory
* The microinstruction contains a control word that specifies one or more
* Micro operations for the data processor

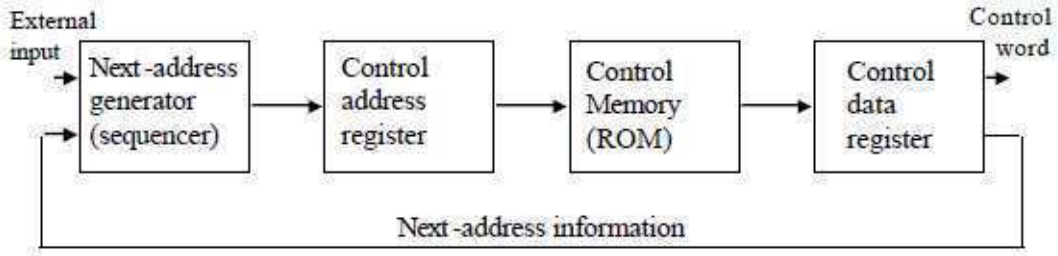

Fig 2.13 Control Memory

Micro Program sequencer

In computer architecture and engineering, a sequencer or micro sequencer generates the addresses used to step through the micro program of a control store. It is used as a part of the control unit of a CPU or as a stand-alone generator for address ranges. Usually the addresses are generated by some combination of a counter, a field from a microinstruction, and some subset of the instruction register. A counter is used for the typical case, that the next microinstruction is the one to execute. A field from the microinstruction is used for jumps, or other logic. Since CPUs implement an instruction set, it's very useful to be able to decode the instruction's bits directly into the sequencer, to select a set of microinstructions to perform a CPU's instructions. Most modern CPUs are considerably more complex than this description suggests. They tend to have multiple cooperating micro machines with specialized logic to detect and handle interference between the micro machines.

A micro program sequencer for micro programmed control unit develops micro program consecutive addresses, branches to subroutines with address saving and possible return to micro program, as well as interrupting micro program forcings with address saving of the interrupted micro programs.

In order to allow the double saving of micro program and subroutine addresses in case of concurrent interruptions and branches, the sequencer is provided with two address generation loops each including a register. The two loops have a common portion to which they accede through a multiplexer (23). The first loop (23, 25, 22, 21, 30, 31) is further coupled to a saving register stack (20). While the first loop executes the saving of a micro program address and the latching of a branch address received from the second loop, the second loop (23, 25, 24, 39, 17, 18, 42, 19, 27, 29) executes a first updating and-, related latching of interrupting micro program address. During the following cycle, by command of the first microinstruction of the interrupting micro program, the second loop performs a first updating and related latch of the interrupting micro program address and the first loop saves into the register stack (20) the branch address and performs a second updating and related latching of the interrupting micro program address.

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in