MODULE-2

ARCHITECTURES FOR PROGRAMMABLE DIGITAL SIGNAL PROCESSING DEVICES

❖ LEARNING OBJECTIVES

- ➢ Introduction, Basic Architectural Features
- ➢ DSP Computational Building Blocks.
- ➢ Bus Architecture and Memory.
- ➢ Data Addressing Capabilities.
- ➢ Address Generation Unit.
- ➢ Programmability and Program Execution.
- ➢ Features for External Interfacing.

2.1 Basic Architectural Features

A programmable DSP device should provide instructions similar to a conventional microprocessor. The instruction set of a typical DSP device should include the following,

a. Arithmetic operations such as ADD, SUBTRACT, MULTIPLY etc
b. Logical operations such as AND, OR, NOT, XOR etc
c. Multiply and Accumulate (MAC) operation
d. Signal scaling operation

In addition to the above provisions, the architecture should also include,

a. On chip registers to store immediate results
b. On chip memories to store signal samples (RAM)
c. On chip memories to store filter coefficients (ROM)

EXAMPLE NO 1: Investigate the basic feature that should be provided in DSP architecture to be implement the following Nth order filter (FIR)

$$y(n) = \sum_{i=0}^{n-1} \mathrm{h(i)x(n-i)}$$

Where x(n) donates the i/p samples, Y(n) donates the o/p samples, h(i) the i th filter coefficient x(n-i) is the i/p samples i samples earlier than x(n)

SOLUTION:
The FIR filter requires the following basic features for implementing

1. A ROM to store the signals samples i.e x(n), x(n-1), x(n-2), …………etc
2. An MAC unit to perform multiply and accumulate operation
3. An accumulator to store the result immediately
4. A resistor to point to the current signal sample being used (coefficient pointer)
5. A counter to keep track of the count
6. A shifter to shift the i/p samples appropriately.

2.2 DSP Computational Building Blocks

- Each computational block of the DSP should be optimized for functionality and speed and in the mean while the design should be sufficiently general so that it can be easily integrated with other blocks to implement overall DSP systems.
- Following are the basic building blocks that are essential to carry out DSP computations
  - Multiplier
  - Shifter
  - MAC unit
  - ALU

2.2.1 Multipliers

- The advent of single chip multipliers paved the way for implementing DSP functions on a VLSI chip.
- Parallel multipliers replaced the traditional shift and add multipliers now days. Parallel multipliers take a single processor cycle to fetch and execute the instruction and to store the result. They are also called as Array multipliers.
- The key features to be considered for a multiplier are:
  a. Accuracy
  b. Dynamic range
  c. Speed
- The number of bits used to represent the operands decides the accuracy and the dynamic range of the multiplier. Whereas speed is decided by the architecture employed.
- If the multipliers are implemented using hardware, the speed of execution will be very high but the circuit complexity will also increases considerably.
- Thus there should be a trade off between the speed of execution and the circuit complexity. Hence the choice of the architecture normally depends on the application.

➢ *Parallel Multipliers*

- Consider the multiplication of two unsigned numbers A and B. Let A be represented using m bits as (Am-1 Am-2 …….. A1 A0) and B be represented using n bits as (Bn-1 Bn-2 …….. B1 B0).
- Then the product P (P_{m+n-1}……….P_2, P_1, P_0)of these two numbers is given by,

$$A = \sum_{i=0}^{m-1} \mathrm{Ai}\ \ 2^{i}$$

$$B = \sum_{j=0}^{n-1} \mathrm{Bj}\ \ 2^{j}$$

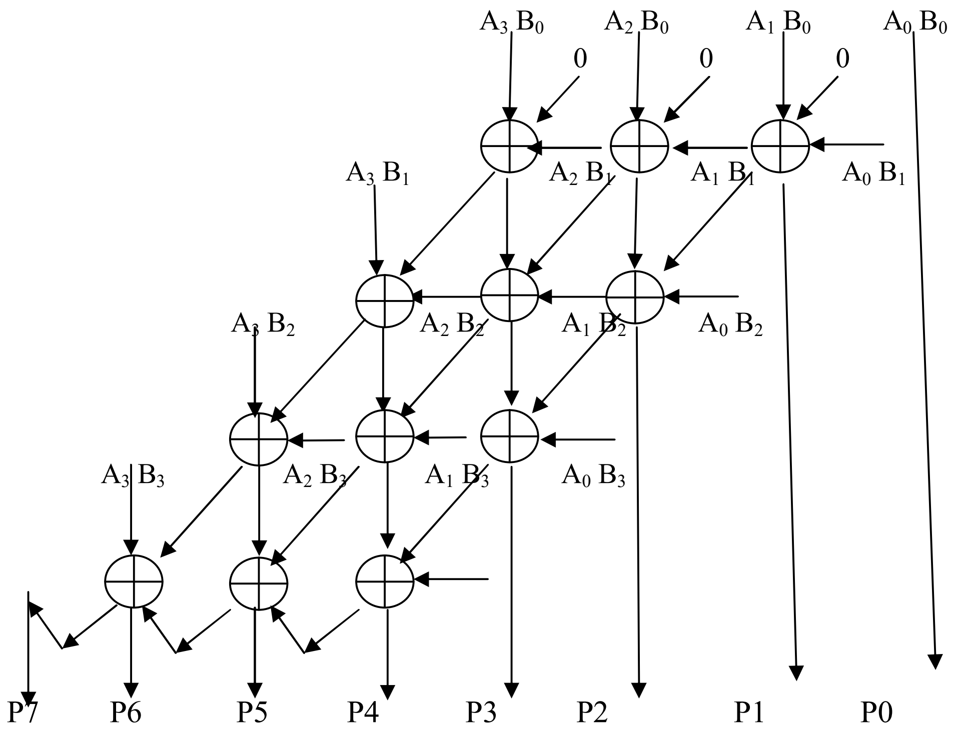
$$P = A \times B = \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} \mathrm{Ai}\ \ \mathrm{Bj}\ \ 2^{i+j}$$

- Each bit of P is obtained by a summation of its bits AiBj using an array of single bits address
- The multiplication operation of bits is as shown bellow, where A=A_3A_2 A_1 A_0 and B=B_3 B_2 B_1 B_0
- The structure is regular and required twelve 3 i/p and 2 o/p adders
- It must have n×n multipliers, the number of adders required is n(n-1)

n×n MULTIPLIERS and n(n-1) ADDERS

			×	A3 B3	A2 B2	A1 B1	A0 B0
				A3B0	A2B0	A1B0	A0B0
			A3B1	A2B1	A1B1	A0B1	↓
		A3B2	A2B2	A1B2	A0B2	↓	↓
	A3B3	A2B3	A1B3	A0B3	↓	↓	↓
↓	↓	↓	↓	↓	↓	↓	↓
P7	P6	P5	P4	P3	P2	P1	P0

- This operation can be implemented paralleling using 4 × 4 Braun multiplier whose hardware structure is as shown in the figure 2.1.
- This structure must have n×n=4×4=16 multipliers,
- The number of adders required is n(n-1)=4(4-1)=4×3=12 with 3i/p and 2 o/p s per adder.

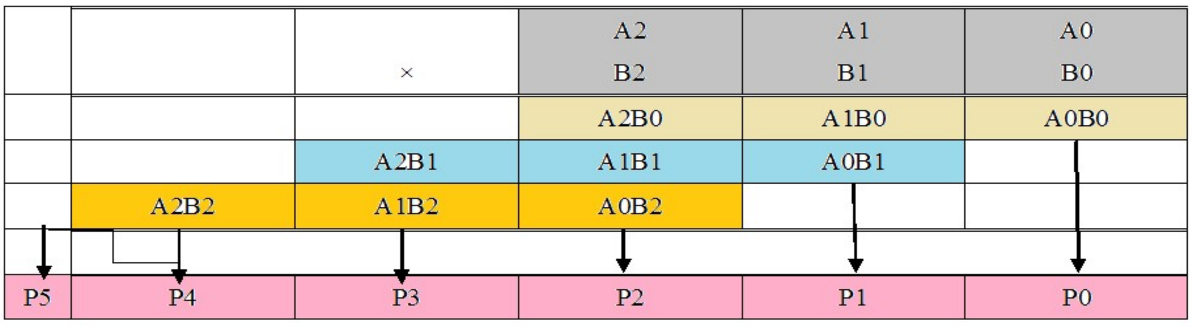


LIMITATON OF BRAUN MULTIPLICATION

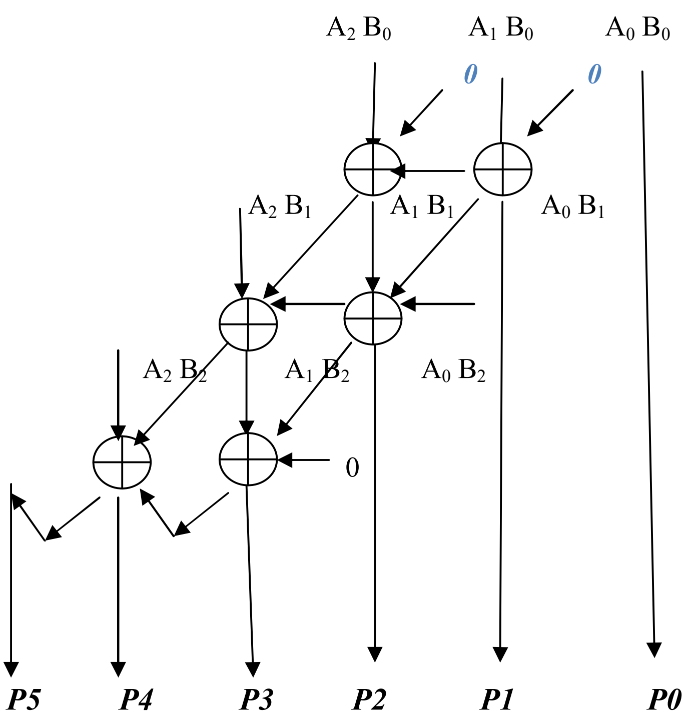
- Braun multiplier does not take into the sign of th numbers that are being multiplied.
- Hardware is required before and after the multiplication of signed numbers which are represented in 2 s compliment.

The structure of 3×3 BRAUN MULTIPLIERS

- let A= A_2 A_1 A_0 , and B= B_2 B_1 B_0
- Product P=A×B=P0, P1, P2,……Pm+n-1
- The structure has n×n multipliers i.e 3×3=9, n(n-1)=3(3-1)=6 adders with 3 i/p and 2 o/p per adder.



		×	A2 B2	A1 B1	A0 B0
			A2B0	A1B0	A0B0
		A2B1	A1B1	A0B1	
	A2B2	A1B2	A0B2		
P5	P4	P3	P2	P1	P0



➢ *Multipliers for Signed Numbers*

- In the Braun multiplier the sign of the numbers are not considered into account.
- Hardware is required before and after the multiplication of signed numbers which are represented in 2 s compliment.
- In order to implement a multiplier for signed numbers, additional hardware is required to modify the Braun multiplier. The modified multiplier is called as Baugh-Wooley multiplier.
- Consider two signed numbers A and B,

$$A= -A_{m-1}\, 2^{m-1} + \sum_{i=0}^{m-2} Ai\ \ 2^{i}$$
$$B= -B_{n-1}\, 2^{n-1} + \sum_{j-2}^{n-2} Bj2^{j}$$
$$P=A_{m-1}\, B_{n-1}\, 2^{m+n-2} + \sum_{i=0}^{m-2}\sum_{j=0}^{n-2} Ai\ Bj\ \ 2^{i+j} - \sum_{i=0}^{m-2} AiBj\ \ 2^{h-1+I}\ \ - \sum_{j=0}^{n-2}\ A_{m-1}B_{j}\ \ 2^{m-1+j}$$

➢ Baugh-Wooley multiplier(4×4 multiplier).

- Product of two number is given by

$$A= -A_{m-1}\, 2^{m-1} + \sum_{i=0}^{m-2} Ai\ \ 2^{i}$$
$$B= -B_{n-1}\, 2^{n-1} + \sum_{n-2}^{n-2} Bj2^{j}$$

$$P=A_{m-1}\, B_{n-1}\, 2^{m+n-2} + \sum_{i=0}^{m-2}\sum_{j=0}^{n-2} Ai\, Bj\ 2^{i+j} - \sum_{i=0}^{m-2} AiBj\ 2^{h-1+I}\ - \sum_{j=0}^{n-2}\ A_{m-1}B_j\ 2^{m-1+j}$$

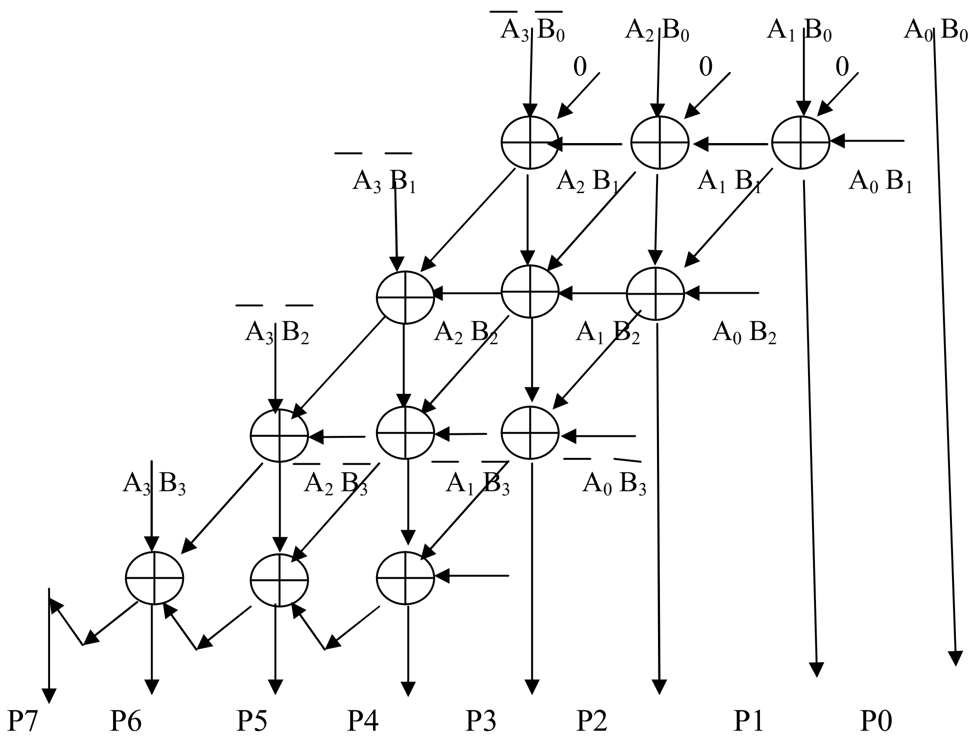
- The two subtraction terms can be expressed as addition of 2 s compliment numbers For example: Let m=4 and n=4

$$\sum_{i=0}^{m-2} Ai\ B_{n-1}=A_0\ B_3 + A_1\ B_3 + A_2\ B_3$$

$$\sum_{j=0}^{n-2} A_{m-1}\ B_{n-1}=A_0\ B_3 + A_1\ B_3 + A_2\ B_3$$

- Therefore the terms $A_0\ B_3$, $A_1\ B_3$, $A_2\ B_3$, $A_0\ B_3$, $A_1\ B_3$, $A_2\ B_3$ are to be complimented

				A3 B3	A2 B2	A1 B1	A0 B0
			×				
				$\overline{A3B0}$	A2B0	A1B0	A0B0
			$\overline{A3B1}$	A2B1	A1B1	A0B1	
		$\overline{A3B2}$	A2B2	A1B2	A0B2		
	A3B3	$\overline{A2B3}$	$\overline{A1B3}$	$\overline{A0B3}$			
↓	↓	↓	↓	↓	↓	↓	↓
P7	P6	P5	P4	P3	P2	P1	P0



$\overline{A_3}\,\overline{B_0}$
$A_2\,B_0$
$A_1\,B_0$
$A_0\,B_0$
0
0
0
$\overline{A_3}\,\overline{B_1}$
$A_2\,B_1$
$A_1\,B_1$
$A_0\,B_1$
$\overline{A_3}\,\overline{B_2}$
$A_2\,B_2$
$A_1\,B_2$
$A_0\,B_2$
$A_3\,B_3$
$\overline{A_2}\,\overline{B_3}$
$\overline{A_1}\,\overline{B_3}$
$\overline{A_0}\,\overline{B_3}$
P7
P6
P5
P4
P3
P2
P1
P0

➢ Baugh-Wooley multiplier (3×3 multiplier).

- Product of two number is given by

  $A= -A_{m-1}\,2^{m-1} + \sum_{i=0}^{m-2} Ai\;\;2^{i}$

  $B= -B_{n-1}\,2^{n-1} + \sum_{n-2}^{n-2} Bj2^{j}$

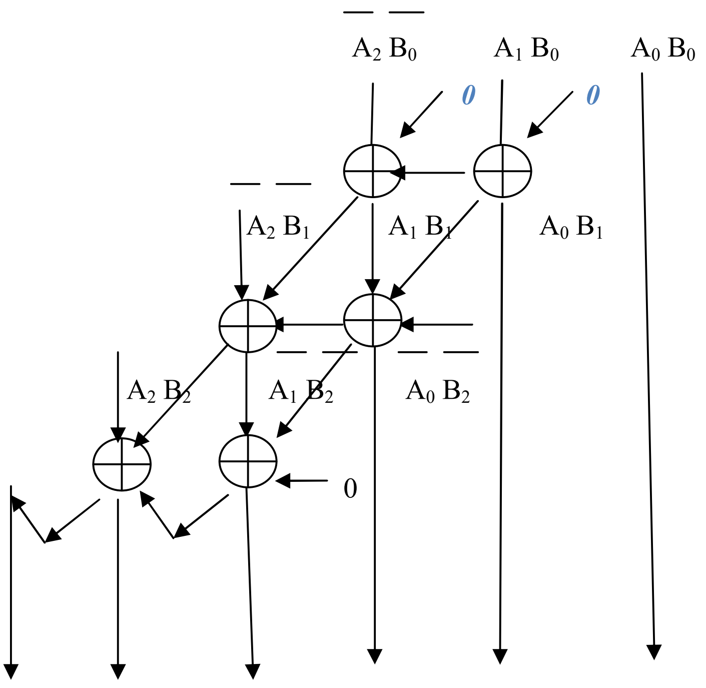
  $P=A_{m-1}\,B_{n-1}\,2^{m+n-2} + \sum_{i=0}^{m-2}\sum_{j=0}^{n-2} Ai\;Bj\;\;2^{i+j} - \sum_{i=0}^{m-2} AiBj\;\;2^{h-1+I}\;\; - \sum_{j=0}^{n-2}\;\; A_{m-1}B_j\;\;2^{m-1+j}$
- The two subtraction terms can be expressed as addition of 2 s compliment numbers For example: Let m=4 and n=4

$$\sum_{i=0}^{m-2} Ai\;B_{n-1}=A_0\;B_2 + A_1\;B_2$$

$$\sum_{j=0}^{n-2} A_{m-1}\;B_j=A_2\;B_1 + A_2\;B_0$$

- Therefore the terms $A_1\;B_2$, $A_0\;B_2$, $A_2\;B_1,$ $A_2\;B_0$ are to be complimented

	×		A2 B2	A1 B1	A0 B0
			$\overline{A2B0}$	A1B0	A0B0
		$\overline{A2B1}$	A1B1	A0B1	
	A2B2	$\overline{A1B2}$	$\overline{A0B2}$		
↓	↓	↓	↓	↓	↓
P5	P4	P3	P2	P1	P0



P5 P4 P3 P2 P1 P0

EXAMPLE NO :1

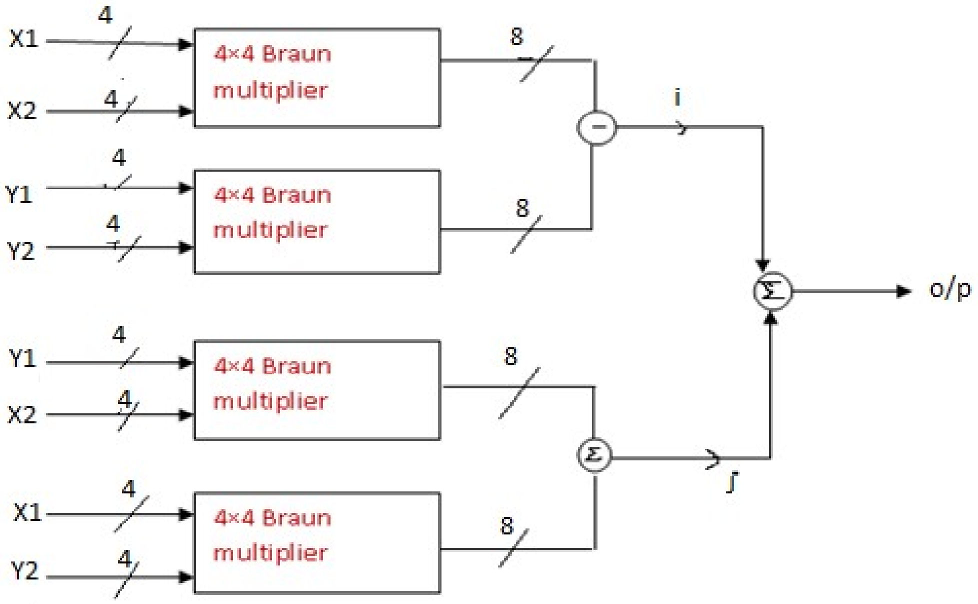
Suggest a scheme to implement a multiplier to multiply two complex numbers using 4×4 Braun multiplier as the building block.

SOLUTION:

Consider two complex numbers $X_1 + jY_1$ and $X_2 + jY_2$

$(X_1 + jY_1) \times (X_2 + jY_2) = X_1X_2 + j X_2Y1 + j X_1Y_2 - Y_1Y_2$

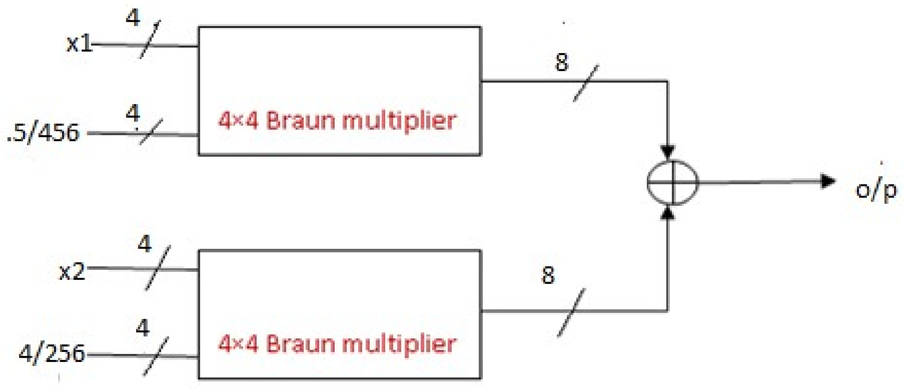
To implement this we need 4×4nmultiplier and three add/substracters



EXAMPLE NO 2:

$(0.5X_1 + 4 X_2)/256$ Using 4×4 Braun multiplier as the building block.

SOLUTION:



➢ ***Speed***

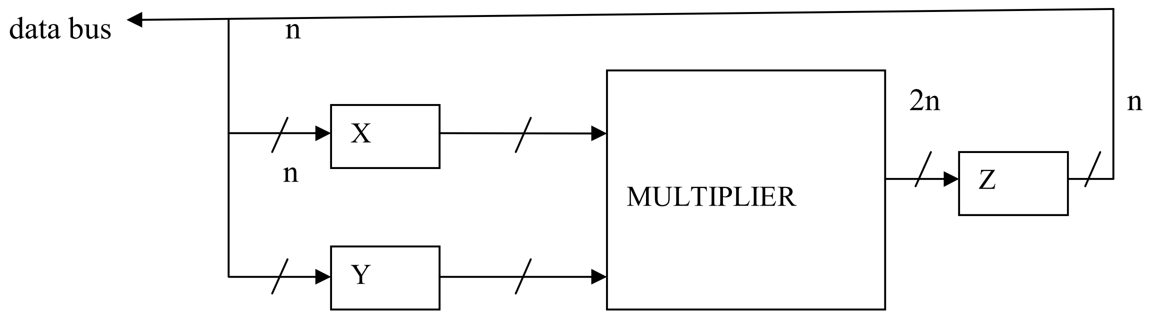
- Conventional Shift and Add technique of multiplication requires n cycles to perform the multiplication of two n bit numbers.
- Whereas in parallel multipliers the time required will be the longest path delay in the combinational circuit used. As DSP applications generally require very high speed, it is desirable to have multipliers operating at the highest possible speed by having parallel implementation.

➢ ***Bus Widths***

- Consider the multiplication of two n bit numbers X and Y. The product Z can be at most 2n bits long.
- In order to perform the whole operation in a single execution cycle, we require two buses of width n bits each to fetch the operands X and Y and a bus of width 2n bits to store the result Z to the memory.
- Although this performs the operation faster, it is not an efficient way of implementation as it is expensive.
- Many alternatives for the above method have been proposed. One such method is to use the program bus itself to fetch one of the operands after fetching the instruction, thus requiring only one bus to fetch the operands. And the result Z can be stored back to the memory using the same operand bus.
- But the problem with this is the result Z is 2n bits long whereas the operand bus is just n bits long. We have two alternatives to solve this problem,

  a. Use the n bits operand bus and save Z at two successive memory locations. Although it stores the exact value of Z in the memory, it takes two cycles to store the result.

  b. Discard the lower n bits of the result Z and store only the higher order n bits into the memory. It is not applicable for the applications where accurate result is required.
- Another alternative can be used for the applications where speed is not a major concern. In which latches are used for inputs and outputs thus requiring a single bus to fetch the operands and to store the result (Fig 2.2).



X, Y,Z are latches or buffer

Fig 2.2: A Multiplier with Input and Output Latches

2.2.2 Shifters

Shifters are used to either scale down or scale up operands or the results.

The following scenarios give the necessity of a shifter

a. While performing the addition of N numbers each of n bits long, the sum can grow up to n+log2 N bits long. If the accumulator is of n bits long, then an overflow error will occur. This can be overcome by using a shifter to scale down the operand by an amount of log2N.
b. Similarly while calculating the product of two n bit numbers, the product can grow up to 2n bits long. Generally the lower n bits get neglected and the sign bit is shifted to save the sign of the product.
c. Finally in case of addition of two floating-point numbers, one of the operands has to be shifted appropriately to make the exponents of two numbers equal.

From the above cases it is clear that, a shifter is required in the architecture of a DSP.

PROBLEM NO 1:

It is required to find the sum of 64, 16 bit numbers. How many bits should the accumulator have so that the sum can be computed without the occurrence of overflow error or loss of accuracy?

SOLUTION:

The sum of 64, 16 bit numbers can grow up to (16+ log2 64)=22 bits long.

Hence the accumulator should be 22 bits long in order to avoid overflow error from occurring.

PROBLEM NO 2:

In the previous problem, it is decided to have an accumulator with only 16 bits but shift the numbers before the addition to prevent overflow, by how many bits should each number be shifted?

SOLUTION:

As the length of the accumulator is fixed, the operands have to be shifted by an amount of log2 64 = 6 bits prior to addition operation, in order to avoid the condition of overflow.

PROBLEM NO 3:

If all the numbers in the previous problem are fixed point integers, what is the actual sum of the numbers?

The actual sum can be obtained by shifting the result by 6 bits towards left side after the sum being computed. Therefore

$$\text{Actual Sum} = \text{Accumulator content} \times 2^{6}$$

$$\boxed{\text{Actual Sum} = \text{Accumulator content} \times 2^{6}}$$

➢ *Barrel Shifters*

- In conventional microprocessors, normal shift registers are used for shift operation.
- As it requires one clock cycle for each shift, it is not desirable for DSP applications, which generally involves more shifts.
- In other words, for DSP applications as speed is the crucial issue, several shifts are to be accomplished in a single execution cycle. This can be accomplished using a barrel shifter,
- This connects the input lines representing a word to a group of output lines with the required shifts determined by its control inputs.
- For an input of length n, log2 n control lines are required. And an additional control line is required to indicate the direction of the shift.
- The block diagram of a typical barrel shifter is as shown in figure 2.3.
- The control i/p also determines the direction of the shift(left or right)
- If i/p word has n bits and shift from 0 to n-1 bits from 0 to n-1 bits positions to the right or left are to be implemented.
- The control i/p requires $\log_2 n$ lines to determine the number of bits to be shifted.
- Bits shifted out of the i/p word are discarded and the new bits position are filled with zeros in case of left shift.
- In case of right shift, the new bits positions are replicated with the most significant bits to maintain the sign of the shifted result.
- Implementation of ***4 bits right shift barrel shifter*.**

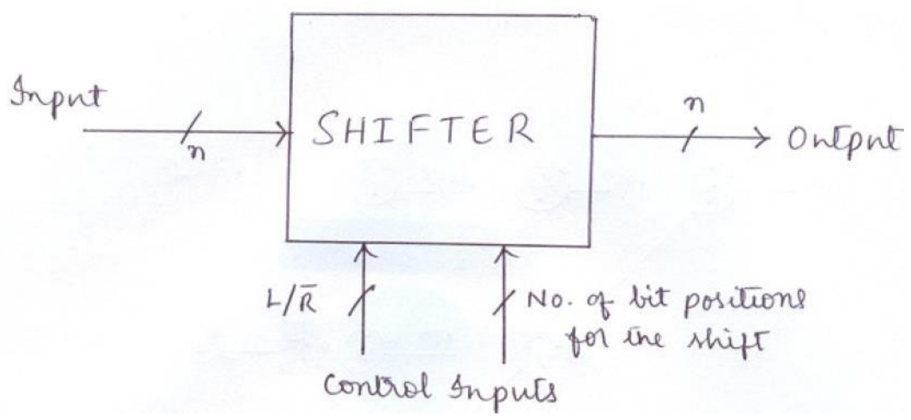
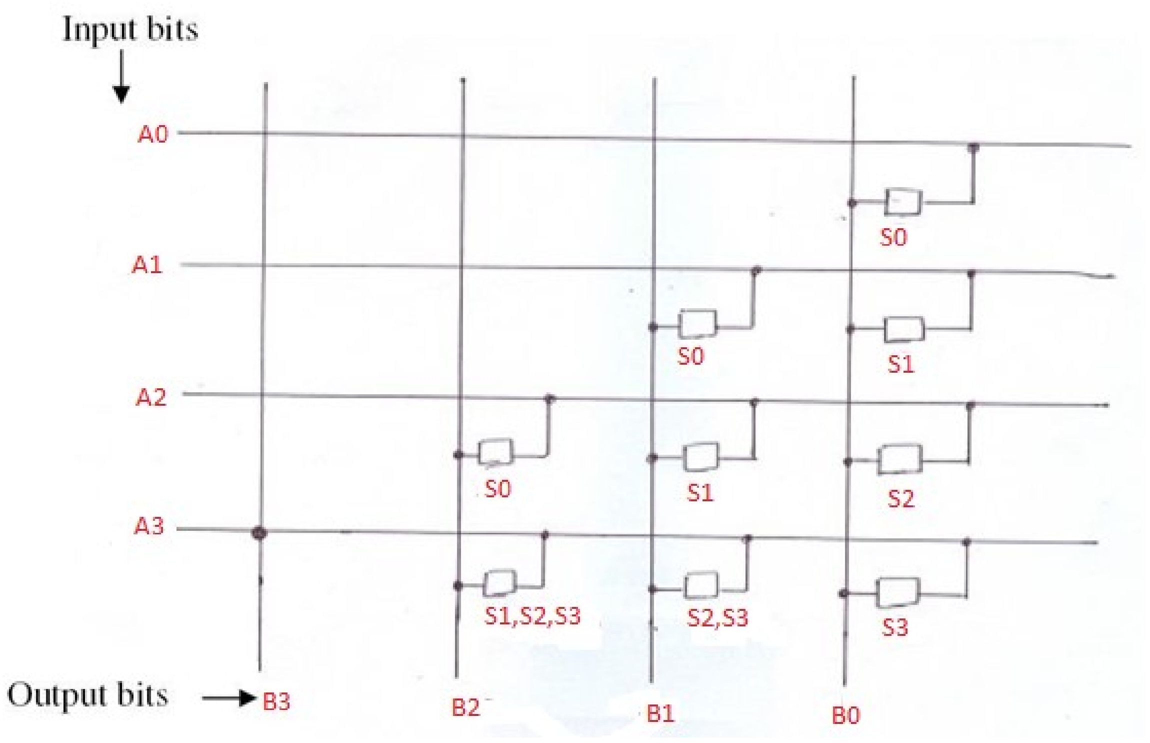


Fig 2.3 A Barrel Shifter

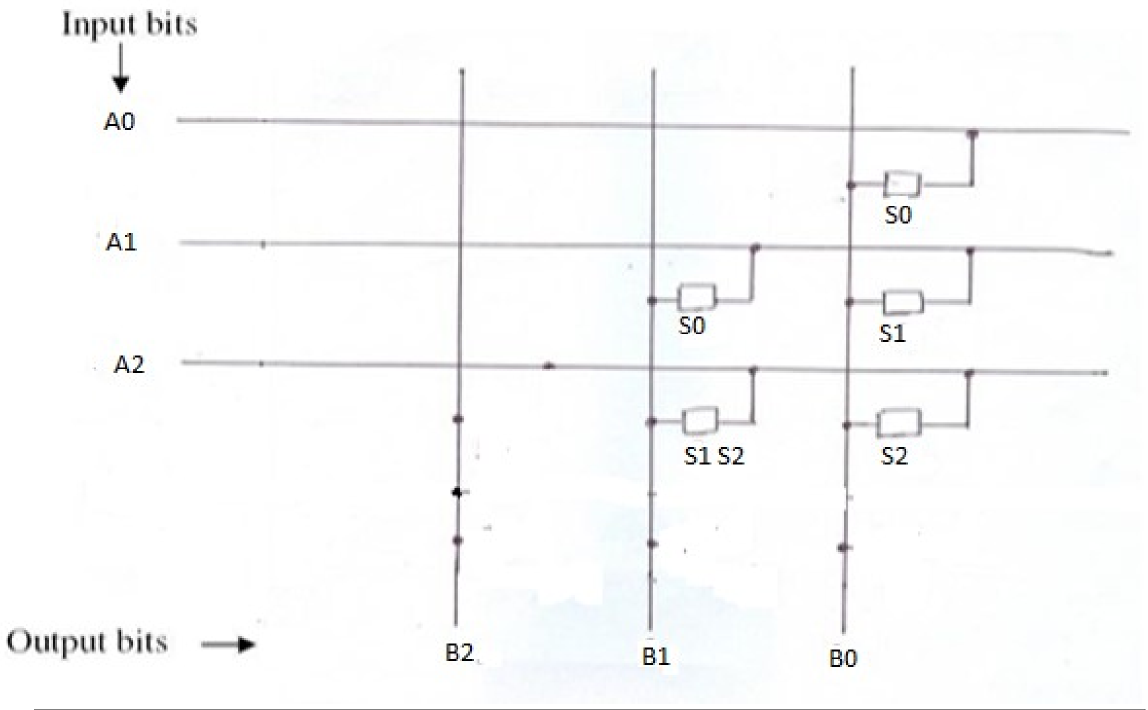


INPUT	SHIFT	OUTPUT(B2,B1,B0)
A3 A2 A1 A0	0(S0)	A3 A2 A1 A0
A3 A2 A1 A0	1(S1)	A3 A3 A2 A1
A3 A2 A1 A0	2(S2)	A3 A3 A3 A2
A3 A2 A1 A0	3(S3)	A3 A3 A3 A3

Fig 2.4 Implementation of a 4 bit Shift Right Barrel Shifter

- Figure 2.4 depicts the implementation of a 4 bit shift right barrel shifter. Shift to right by 0, 1, 2 or 3 bit positions can be controlled by setting the control inputs appropriately.
- As a barrel shifter is a combination logic circuit, the time taken to implement the shift is the total combination deley involve in decoding the control lines and setting up the path from the i/p lines to the o/p lines.

- Implementation of ***3 bits right shift barrel shifter.***



INPUT	SHIFT	OUTPUT(B2,B1,B0)
A2 A1 A0	0(S0)	A2 A1 A0
A2 A1 A0	1(S1)	A2 A2 A1
A2 A1 A0	2(S2)	A2 A2 A2

2.2.3 Multiply and Accumulate Unit

- Most of the DSP applications require the computation of the sum of the products of a series of successive multiplications. In order to implement such functions a special unit called a multiply and Accumulate (MAC) unit is required.
- A MAC consists of a multiplier and a special register called Accumulator. MACs are used to implement the functions of the type A+BC.
- A typical MAC unit is as shown in the figure 2.5.

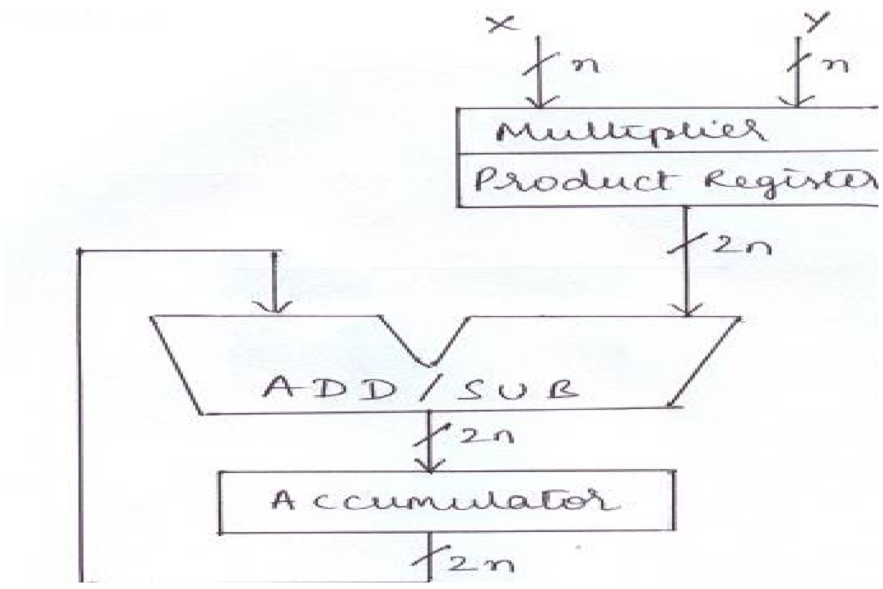


Fig 2.5 A MAC Unit

- Although addition and multiplication are two different operations, they can be performed in parallel.
- By the time the multiplier is computing the product, accumulator can accumulate the product of the previous multiplications.
- If N products are to be accumulated, N-1 multiplications can overlap with N-1 additions.
- During the very first multiplication, accumulator will be idle and during the last accumulation, multiplier will be idle. Thus N+1 clock cycles are required to compute the sum of N products.

PROBLEM NO 1:

If a sum of 256 products is to be computed using a pipelined MAC unit, and if the MAC execution time of the unit is 100nsec, what will be the total time required to complete the operation?

As N=256 in this case,

MAC unit requires N+1=257execution cycles.

As the single MAC execution time is 100nsec,

The total time required will be, (257*100nsec)=25.7usec

PROBLEM NO 2:

Consider a MAC unit whose inputs are 16 bit numbers. If 256 products are to be summed up in this MAC, how many guard bits should be provided for the accumulator to prevent overflow condition from occurring?

As it is required to calculate the sum of 256, 16 bit numbers, the sum can be as long as $(16+ \log_2 256)=24$ bits.

Hence the accumulator should be capable of handling these 22 bits. Thus the guard bits required will be (24-16)= 8 bits.

The block diagram of the modified MAC after considering the guard or extention bits is as shown in the figure

PROBLEM NO 3:

What should be the minimum width of the accumulator in a DSP device that receives 10n bits of A/D samples and is required to add 64 of them without causing an overflow?

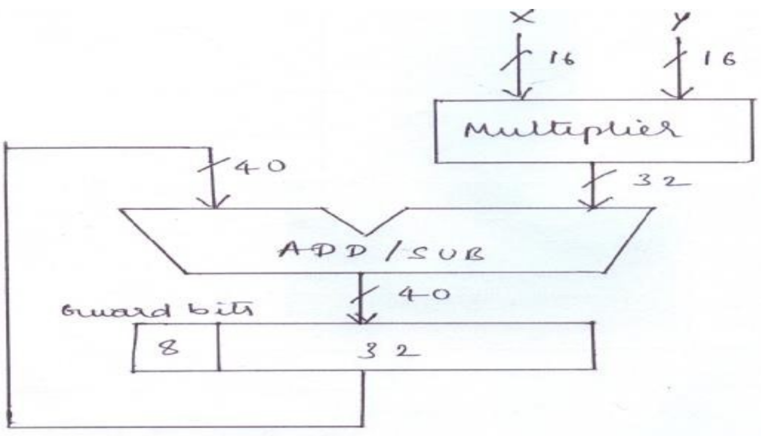
SOLUTION:

As it is required to calculate the sum of 64, 10bits number

The sum can be calculated as $(n + \log_2 N)$

n=10, N=64 i.e $10 + \log_2 64 = 10 + 6 =16$

hence the accumulator should be capable of handling these 16 bits thus the guard bits required will be (16-10)=6 bits



PROBLEM NO 4:

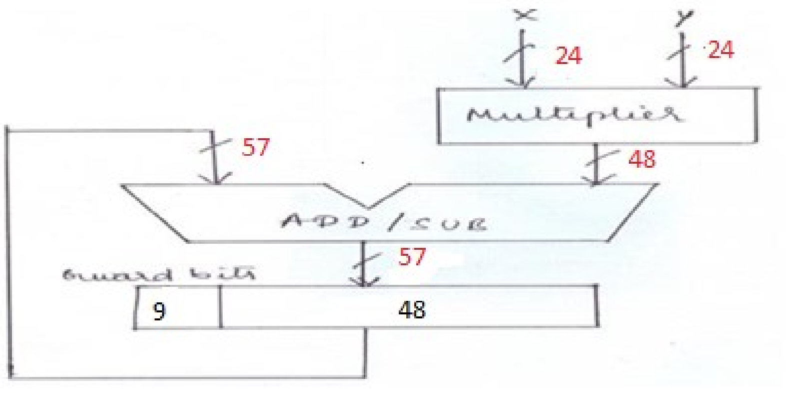
Explain guard bits in a MAC unit of a DSP. Consider a MAC unit whose input are 24 bit number, How many gard bits to provided if 512 products have to be added in the accumulator to prevent overflow condition? What is the overall size of the accumulator required?

SOLUTION:

To handle the growth of the accumulator sum guard bits are used. They are also called as extension bits, When the computation of the required sum of product is completed, the extension bits may be shifted by the required amount and saved as a single word.

Guard bits required for 512 = $\log_2 512=9$ Product to prevent overflow

Overall size = 24 + 24 + 9 = 57 bits.



- **Overflow and Underflow**

  While designing a MAC unit, attention has to be paid to the word sizes encountered at the input of the multiplier and the sizes of the add/subtract unit and the accumulator, as there is a possibility of overflow and underflows. Overflow/underflow can be avoided by using any of the following methods viz

  a. Using shifters at the input and the output of the MAC
  b. Providing guard bits in the accumulator
  c. Using saturation logic

- **Shifters**

  Shifters can be provided at the input of the MAC to normalize the data and at the output to de normalize the same.

- **Guard bits**

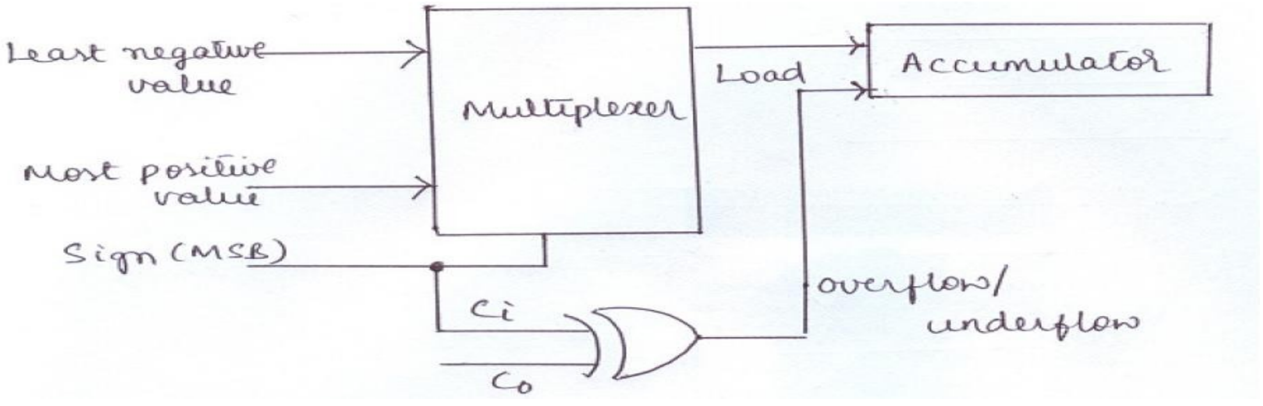
  As the normalization process does not yield accurate result, it is not desirable for some applications. In such cases we have another alternative by providing additional bits called guard bits in the accumulator so that there will not be any overflow error. Here the add/subtract unit also has to be modified appropriately to manage the additional bits of the accumulator.

- **Saturation Logic**
  - Overflow/ underflow will occur if the result goes beyond the most positive number or below the least negative number the accumulator can handle.
  - The overflow/underflow error can be resolved by loading the accumulator with the most positive number which it can handle at the time of overflow and the least negative number that it can handle at the time of underflow.

- This method is called as saturation logic.
- A schematic diagram of saturation logic is as shown in figure 2.7.
- In saturation logic, as soon as an overflow or underflow condition is satisfied the accumulator will be loaded with the most positive or least negative number overriding the result computed by the MAC unit.
- If carry C_i is not equal to the carry out C_0 the overflow/underflow condition occurs

Fig 2.7: Schematic Diagram of the Saturation Logic

2.2.4 Arithmetic and Logic Unit



- A typical DSP device should be capable of handling arithmetic instructions like ADD, SUB, INC, DEC etc and logical operations like AND, OR , NOT, XOR etc.
- The block diagram of a typical ALU for a DSP is as shown in the figure 2.8.
- It consists of status flag register, register file and multiplexers.

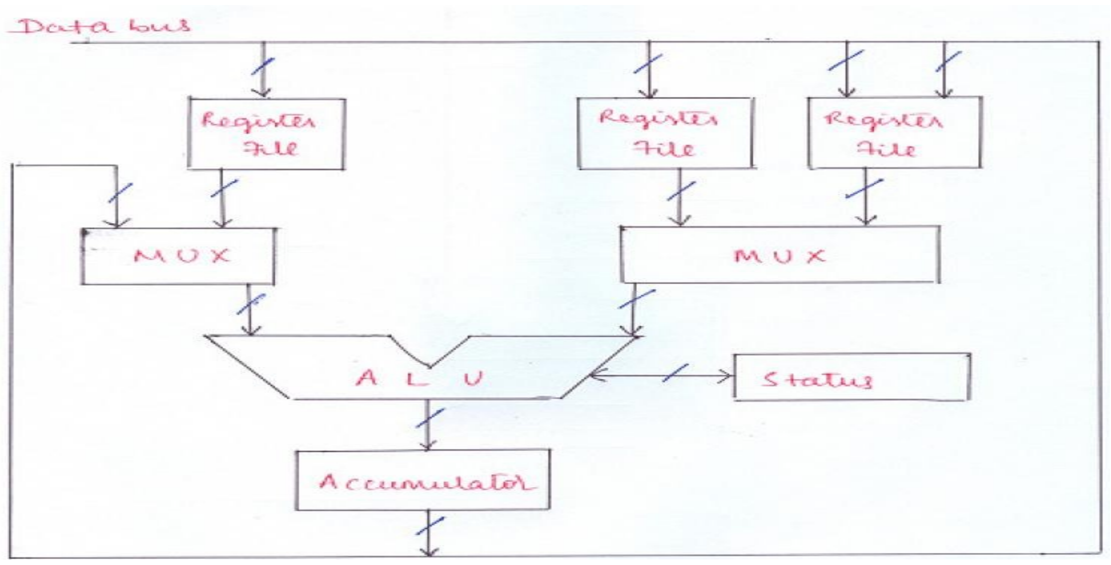


Fig 2.8 Arithmetic Logic Unit of a DSP

- **Status Flags**
  ALU includes circuitry to generate status flags after arithmetic and logic operations. These flags include sign, zero, carry and overflow.
  Example: if the execution of any instruction result in overflow, overflow flag is set otherwise it is reset.

- **Overflow Management**
  Depending on the status of overflow and sign flags, the saturation logic can be used to limit the accumulator content.

- **Register File**
  Instead of moving data in and out of the memory during the operation, for better speed, a large set of general purpose registers are provided to store the intermediate results.

2.3 Bus Architecture and Memory

- Conventional microprocessors use Von Neumann architecture for memory management wherein the same memory is used to store both the program and data (Fig 2.9).
- Although this architecture is simple, it takes more number of processor cycles for the execution of a single instruction as the same bus is used for both data and program.

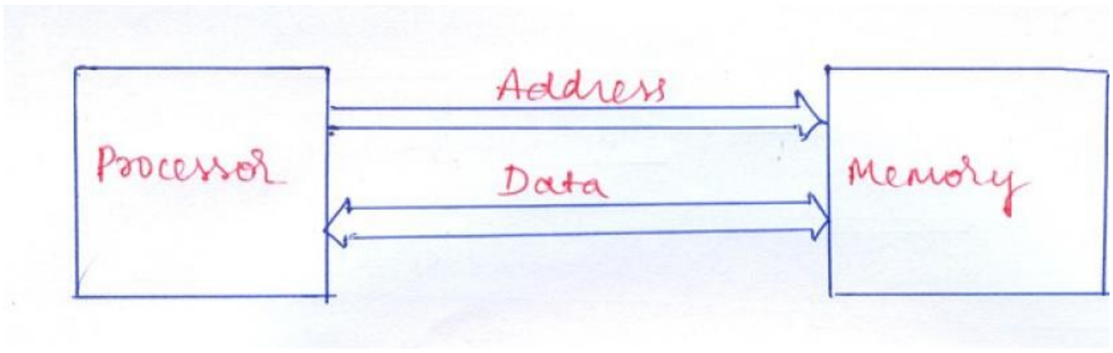


Fig 2.9 Von Neumann Architecture

- In order to increase the speed of operation, separate memories were used to store program and data and a separate set of data and address buses have been given to both memories, the architecture called as **Harvard Architecture**.
- It is as shown in figure 2.10.

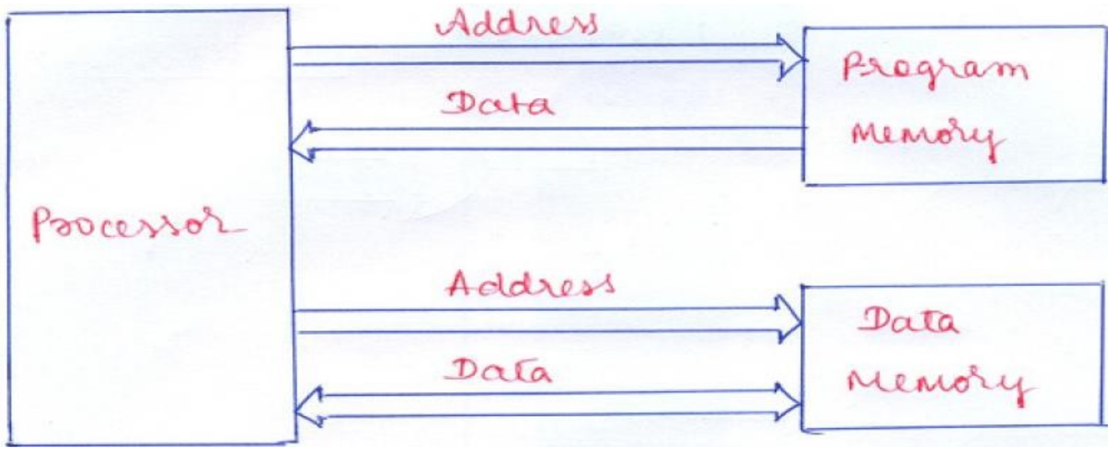


Fig 2.10 Harvard Architecture

- Although the usage of separate memories for data and the instruction speeds up the processing, it will not completely solve the problem.
- As many of the DSP instructions require more than one operand, use of a single data memory leads to the fetch the operands one after the other, thus increasing the delay of processing.
- This problem can be overcome by using two separate data memories for storing operands separately, thus in a single clock cycle both the operands can be fetched together (Figure 2.11).

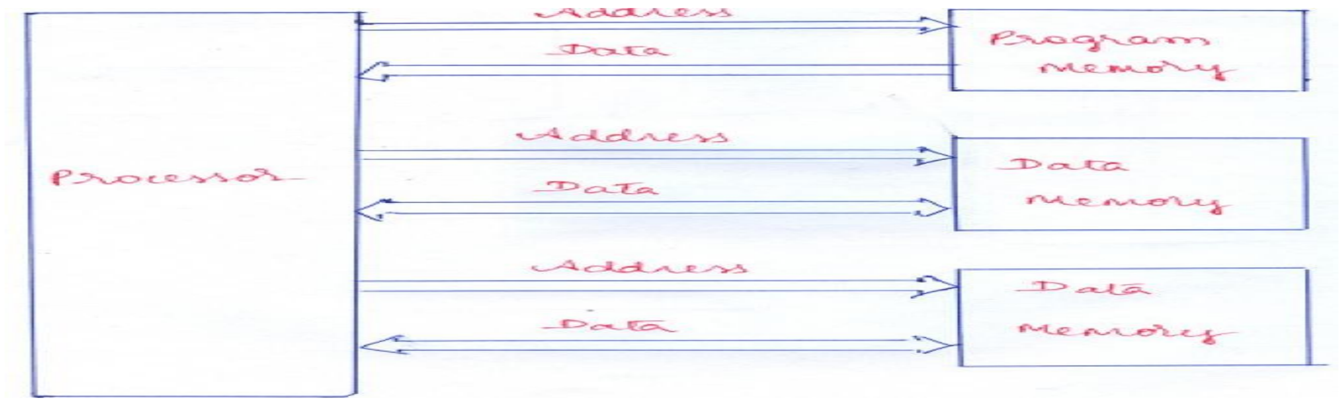


Fig 2.11 Harvard Architecture with Dual Data Memory

- Although the above architecture improves the speed of operation, it requires more hardware and interconnections, thus increasing the cost and complexity of the system.
- Therefore there should be a tradeoff between the cost and speed while selecting memory architecture for a DSP.

2.3.1 On-chip Memories

In order to have a faster execution of the DSP functions, it is desirable to have some memory located on chip. As dedicated buses are used to access the memory, on chip memories are faster. Speed and size are the two key parameters to be considered with respect to the on-chip memories.

Speed

On-chip memories should match the speeds of the ALU operations in order to maintain the single cycle instruction execution of the DSP.

Size

In a given area of the DSP chip, it is desirable to implement as many DSP functions as possible. Thus the area occupied by the on-chip memory should be minimum so that there will be a scope for implementing more number of DSP functions on- chip.

2.3.2 Organization of On-chip Memories

- Ideally whole memory required for the implementation of any DSP algorithm has to reside on- chip so that the whole processing can be completed in a single execution cycle.
- Although it looks as a better solution, it consumes more space on chip, reducing the scope for implementing any functional block on-chip, which in turn reduces the speed of execution. Hence some other alternatives have to be thought of.
- The following are some other ways in which the on-chip memory can be organized.

a. As many DSP algorithms require instructions to be executed repeatedly, the instruction can be stored in the external memory, once it is fetched can reside in the instruction cache.
b. The access times for memories on-chip should be sufficiently small so that it can be accessed more than once in every execution cycle.
c. On-chip memories can be configured dynamically so that they can serve different purpose at different times.

2.4 Data Addressing Capabilities

Data accessing capability of a programmable DSP device is configured by means of its addressing modes. The summary of the addressing modes used in DSP is as shown in the table below.

Table 2.1 DSP Addressing Modes

Addressing Mode	Operand	Sample Format	Operation
Immediate	Immediate Value	ADD #imm	#imm +A $\longrightarrow$ A
Register	Register Contents	ADD reg	reg +A $\longrightarrow$ A
Direct	Memory Address Register	ADD mem	mem+A $\longrightarrow$ A
Indirect	Memory contents with address in the register	ADD *addreg	*addreg +A $\longrightarrow$ A

2.4.1 Immediate Addressing Mode

In this addressing mode, data is included in the instruction itself.
The capability to include data as part of the instruction is provided by the immediate addressing mode.
Example : A DSP processor may allow the programmer to write the instruction

ADD #imm

To add the value represented by imm to the accumulator register

#imm + A→A this operation is implemented

Example : FIR coefficient are examples for this kind.

2.4.2 Register Addressing Mode

- In this mode, one of the registers will be holding the data and the register has to be specified in the instruction.
- In this register addressing mode a processor register provides the operand using

this addressing mode the DSP processor may provide an instruction

- ADD reg

- To implement reg + A→A

2.4.3 Direct Addressing Mode

- In this addressing mode, instruction holds the memory location of the operand.
- In the direct addressing mode a memory operand is specified by providing its memory address,for a instant DSP processor may allow an instruction

  ADD mem
- To implement mem + A → A
- A single sample store in a memory location can be accessed using direct addressing mode.

2.4.3 Indirect Addressing Mode

- In this addressing mode, the operand is accessed using a pointer.
- A pointer is generally a register, which holds the address of the location where the operands resides. Example: to add to the accumulator A, the content of the memory location whose address is held in addreg,
- The following instruction is implemented

  ADD *addreg which means *addreg + A →A
- Indirect addressing mode can be extended to inculcate automatic increment or decrement capabilities, which has lead to the following addressing modes.

Table 2.2 Indirect Addressing Modes

Addressing Mode	Sample Format	Operation
Post Increment	ADD *addreg+	A $\longrightarrow$ A + *addreg addreg $\longrightarrow$ addreg+1
Post Decrement	ADD *addreg-	A $\longrightarrow$ A + *addreg addreg $\longrightarrow$ addreg-1
Pre Increment	ADD +*addreg	addreg $\longrightarrow$ addreg+1 A $\longrightarrow$ A + *addreg
Pre Decrement	ADD -*addreg	addreg $\longrightarrow$ addreg-1 A $\longrightarrow$ A + *addreg
Post_Add_Offset	ADD *addreg, offsetreg+	A $\longrightarrow$ A + *addreg addreg $\longrightarrow$ addreg+offsetreg
Post_Sub_Offset	ADD *addreg, offsetreg-	A $\longrightarrow$ A + *addreg addreg $\longrightarrow$ addreg-offsetreg
Pre_Add_Offset	ADD offsetreg+,*addreg	addreg $\longrightarrow$ addreg+offsetreg A $\longrightarrow$ A + *addreg
Pre_Sub_Offset	ADD offsetreg-,*addreg	addreg $\longrightarrow$ addreg-offsetreg A $\longrightarrow$ A + *addreg

PROBLEM NO:1

What are the memory addresses of the operands in each of the following cases of indirect addressing modes? In each case, what will be the content of the *addreg* after the memory access? Assume that the initial contents of the *addreg* and the *offsetreg* are 0200h and 0010h, respectively.

a. ADD **addreg*
b.ADD +addreg***
c.* ADD *offsetreg+,*addreg
d.* ADD **addreg,offsetreg-

Instruction	Addressing Mode	Operand Address	addreg Content after Access
ADD **addreg-*	Post Decrement	0200h	0200-01=01FFh
ADD +**addreg*	Pre Increment	0200+01=0201h	0201h
ADD *offsetreg+,*addreg*	Pre_Add_Offset	0200+0010=0210h	0210h
ADD **addreg,offsetreg-*	Post_Sub_Offset	0200h	0200-0010=01F0h

PROBLEM NO:2

Identify the addressing modes of the operand in each of the following instruction and their operation

1) ADD B 2) ADD 5678h 3) ADD + *addreg 4) ADD *addreg, offsetreg-

SOLUTION:

1) ADD B $\rightarrow$Register addressing mode
   Operation: B +A$\rightarrow$A
   Content of register B is added with content of accumulator store the result in accumulator.

2) ADD 567 h $\rightarrow$Direct addressing mode
   Operation : A$\leftarrow$5678h

3) ADD + *addreg $\rightarrow$ Pre increment indirect addressing mode
   Operation: addreg $\leftarrow$addreg + 1
   A $\leftarrow$ A+*addreg
   Content of addreg is incremented by 1, content of memory location pointed by *addreg is added with A and stored in A.

4) ADD *addreg, offsetreg-$\rightarrow$ Post subs tract offset indirect addressing mode
   Operation: *adreg + A $\rightarrow$ A
   addreg $\leftarrow$addreg – offset.
   Content of memory location pointer by *addreg is added with A and stored the result in accumulator.
   Content of addreg is subtracted with the content of offsetreg stored result in addreg.

2.4.6 Special Addressing Modes

For the implementation of some real time applications in DSP, normal addressing modes will not completely serve the purpose. Thus some special addressing modes are required for such applications.

- ***Circular Addressing Mode***

  While processing the data samples coming continuously in a sequential manner, circular buffers are used. In a circular buffer the data samples are stored sequentially from the initial location till the buffer gets filled up. Once the buffer gets filled up, the next data samples will get stored once again from the initial location. This process can go forever as long as the data samples are processed in a rate faster than the incoming data rate.

  Circular Addressing mode requires three registers viz

  a. Pointer register to hold the current location (PNTR)

  b. Start Address Register to hold the starting address of the buffer (SAR)

c. End Address Register to hold the ending address of the buffer (EAR)

There are four special cases in this addressing mode. They are

a. SAR < EAR & updated PNTR > EAR
b. SAR < EAR & updated PNTR < SAR
c. SAR >EAR & updated PNTR > SAR
d. SAR > EAR & updated PNTR < EAR

The buffer length in the first two case will be (EAR-SAR+1) whereas for the next tow cases (SAR- EAR+1)

The pointer updating algorithm for the circular addressing mode is as shown below.

```
; Pointer Updating Algorithm

Updated PNTR ←—— PNTR ± increment

If SAR < EAR
        And if Updated PNTR > EAR then
                New PNTR ←—— Updated PNTR – Buffer size
        And if Updated PNTR < SAR then
                New PNTR           Updated PNTR + Buffer size

If SAR > EAR
        And if Updated PNTR > SAR then
                New PNTR ←—— Updated PNTR – Buffer size
        And if Updated PNTR < EAR then
                New PNTR ←—— Updated PNTR + Buffer size

Else
        New PNTR ←—— Updated PNTR
```

Four cases explained earlier are as shown in the figure 2.12.

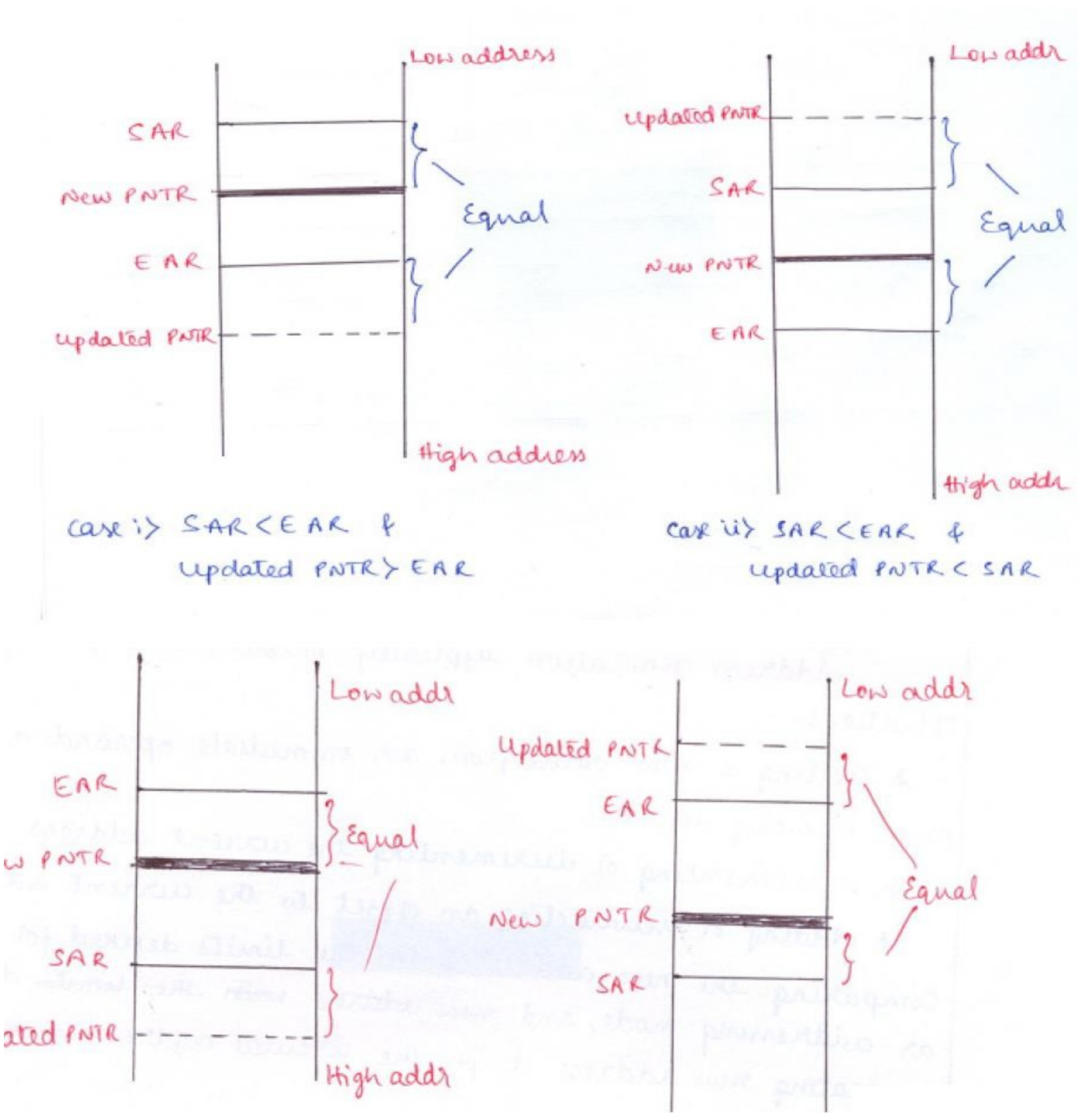


Fig 2.12 Special Cases in Circular Addressing Mode

PROBLEM NO:1

A DSP has a circular buffer with the start and the end addresses as 0200h and 020Fh respectively. What would be the new values of the address pointer of the buffer if, in the course of address computation, it gets updated to

a. 0212h
b. 01FCh

SOLUTION:

Buffer Length= (EAR-SAR+1) = 020F-0200+1=10h

a. New Address Pointer= Updated Pointer-buffer length = 0212-10=0202h
b. New Address Pointer= Updated Pointer+ buffer length = 01FC+10=020Ch

PROBLEM NO: 2

Repeat the previous problem for SAR= 0210h and EAR=0201h

SOLUTION:

Buffer Length= (SAR-EAR+1)= 0210-0201+1=10h

c. New Address Pointer= Updated Pointer- buffer length = 0212-10=0202h
d. New Address Pointer= Updated Pointer+ buffer length = 01FC+10=020Ch

PROBLEM NO: 3

A digital signal processor has a circular buffer with a start and the end addresses as 0200h and 0310h. What is the circular buffer size? What would be the new value of the addresses pointer of the abive buffer if in the course of address computation it gets updated to i)0336h ii)0192h

SOLUTION:

SAR = 0200H   EAR = 0192 H
NOW SAR<EAR
BUFFER SIZE =EAR –SAR +1=0310 – 0200 +1=0111h

a) UPNTER = 0336h
NEWPNTR=UPNTR –BUFERSIZE =0336h-0111h=0225h

b) UPPNTR =0192 h,   UPNTR< SAR
NEWPNTR = UPNTR + BUFFERSIZE = 0192 h + 0111h =0303 h

➢ ***Bit Reversed Addressing Mode***

To implement FFT algorithms we need to access the data in a bit reversed manner. Hence a special addressing mode called bit reversed addressing mode is used to calculate the index of the next data to be fetched. It works as follows. Start with index 0. The present index can be calculated by adding half the FFT length to the previous index in a bit reversed manner, carry being propagated from MSB to LSB.

Current index= Previous index+ B (1/2(FFT Size))

Important note: The reverse carry add operation can be provided in the architecture to implement special addressing mode.

PROBLEM NO: 1

Compute the sequence in which the i/p data should be ordered for a 8 point DIT-FFT

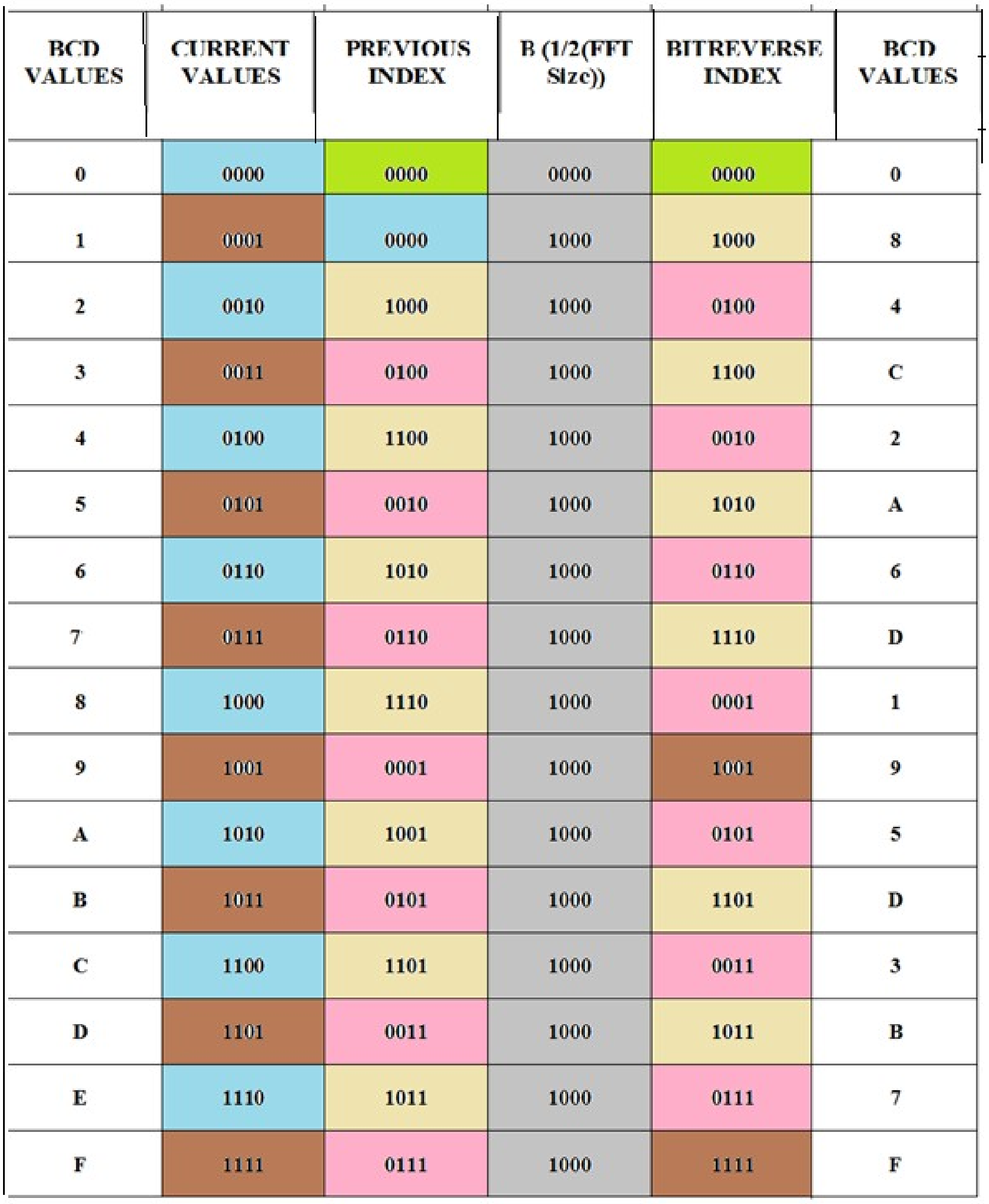
SOLUTION: start with index 0
First index would be 000
FFT length is 8
Therefore the first index would be (000)
Next index can be calculated by adding half the FFT length, in this case it is (100) to the previous index. i.e. Present Index= (000)+B (100)= (100)
Similarly the next index can be calculated as Present Index= (100)+B (100)= (010)

Current index= Previous index+ B (1/2(FFT Size))

BCD VALUES	CURRENT VALUES	PREVIOUS INDEX	B (1/2(FFT Size))	BITREVERSE INDEX	BCD VALUES
0	000	000	000	000	0
1	001	000	100	100	4
2	010	100	100	010	2
3	011	010	100	110	6
4	100	110	100	001	1
5	101	001	100	101	5
6	110	101	100	011	3
7	111	011	100	111	7

PROBLEM NO: 2
Compute the sequence in which the i/p data should be ordered for a 8 point DIT-FFT
SOLUTION: start with index 0
First index would be 0000
FFT length is 16
Therefore the first index would be (0000)
Next index can be calculated by adding half the FFT length, in this case it is (1000) to the previous index. i.e. Present Index= (0000)+B (100)= (1000)
Similarly the next index can be calculated as Present Index= (1000)+B (1000)= (0100)

Current index= Previous index+ B (1/2(FFT Size))



BCD VALUES	CURRENT VALUES	PREVIOUS INDEX	B (1/2(FFT Size))	BITREVERSE INDEX	BCD VALUES
0	0000	0000	0000	0000	0
1	0001	0000	1000	1000	8
2	0010	1000	1000	0100	4
3	0011	0100	1000	1100	C
4	0100	1100	1000	0010	2
5	0101	0010	1000	1010	A
6	0110	1010	1000	0110	6
7	0111	0110	1000	1110	D
8	1000	1110	1000	0001	1
9	1001	0001	1000	1001	9
A	1010	1001	1000	0101	5
B	1011	0101	1000	1101	D
C	1100	1101	1000	0011	3
D	1101	0011	1000	1011	B
E	1110	1011	1000	0111	7
F	1111	0111	1000	1111	F

2.5 Address Generation Unit

The main job of the Address Generation Unit is to generate the address of the operands required to carry out the operation. They have to work fast in order to satisfy the timing constraints. As the address generation unit has to perform some mathematical operations

in order to calculate the operand address, it is provided with a separate ALU.
Address generation typically involves one of the following operations.

a. Getting value from immediate operand, register or a memory location
b. Incrementing/ decrementing the current address
c. Adding/subtracting the offset from the current address
d. Adding/subtracting the offset from the current address and generating new address according to circular addressing mode
e. Generating new address using bit reversed addressing mode

The block diagram of a typical address generation unit is as shown in figure 2.13.

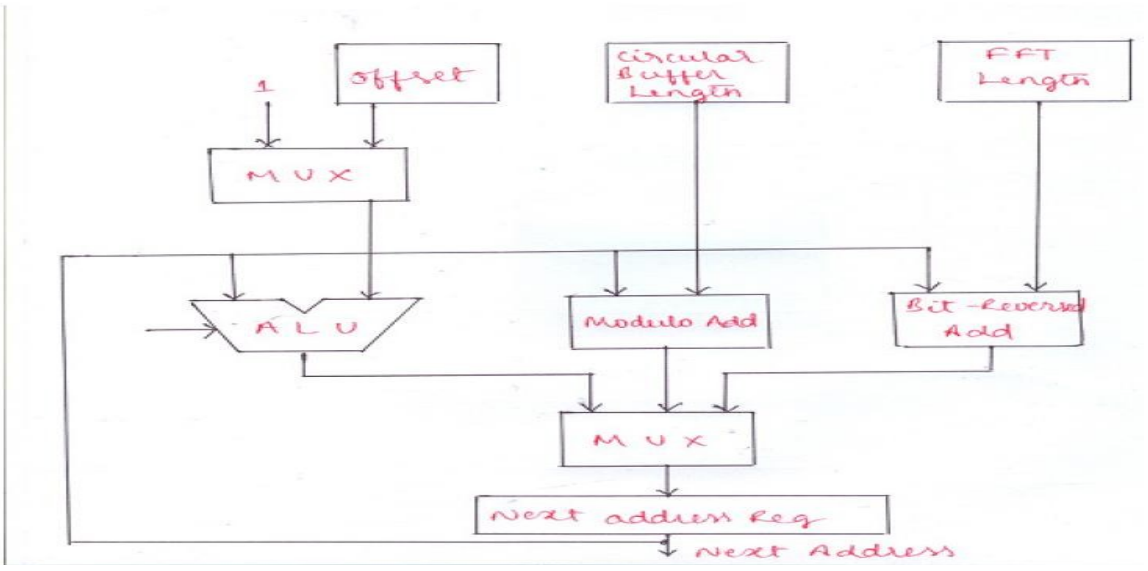


Fig 2.13 Address generation unit

2.6 Programmability and program Execution

- A programmable DSP device should provide the programming capability involving branching, looping and subroutines.
- The implementation of repeat capability should be hardware based so that it can be programmed with minimal or zero overhead. A dedicated register can be used as a counter.
- In a normal subroutine call, return address has to be stored in a stack thus requiring memory access for storing and retrieving the return address, which in turn reduces the speed of operation. Hence a LIFO memory can be directly interfaced with the program counter.

2.6.2 Program Control

- Like microprocessors, DSP also requires a control unit to provide necessary control and timing signals for the proper execution of the instructions.
- In microprocessors, the controlling is micro coded based where each instruction is divided into microinstructions stored in micro memory. As this mechanism is slower, it is not applicable for DSP applications.
- Hence in DSP the controlling is hardwired base where the Control unit is designed as a single, comprehensive, hardware unit. Although it is more complex it is faster.

2.6.3 Program Sequencer

- It is a part of the control unit used to generate instruction addresses in sequence needed to access instructions. It calculates the address of the next instruction to be fetched. The next address can be from one of the following sources.
  a. Program Counter
  b. Instruction register in case of branching, looping and subroutine calls
  c. Interrupt Vector table
  d. Stack which holds the return address
- The block diagram of a program sequencer is as shown in figure 2.14.

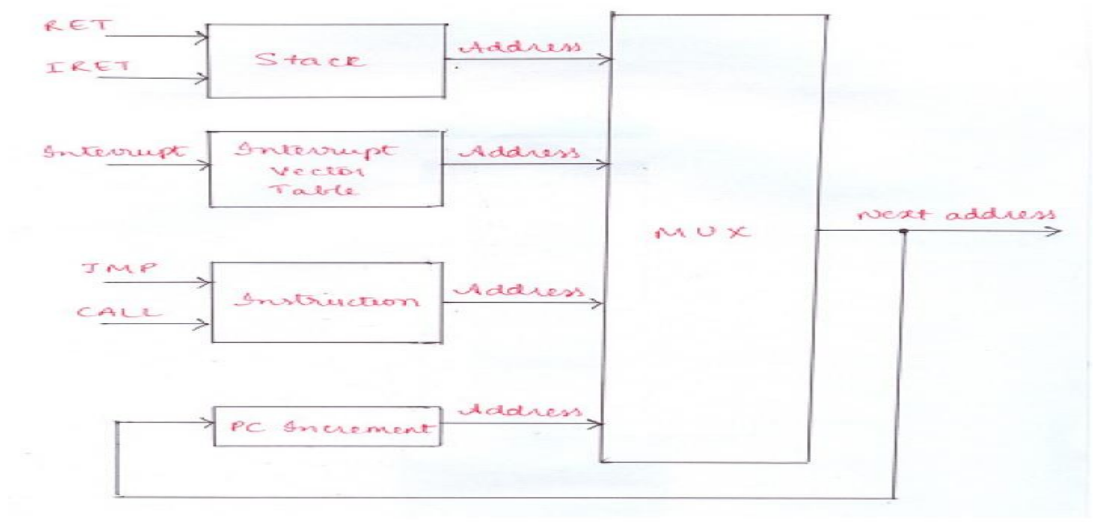


Fig 2.14 Program Sequencer

Program sequencer should have the following circuitry:

a. PC has to be updated after every fetch
b. Counter to hold count in case of looping
c. A logic block to check conditions for conditional jump instructions
d. Condition logic-status flag

ASSIGNMENT NO: 2 (Recommended Questions)

1. Implement a 3-bit shift right barrel shifter, Tabulate the output for different bit shifts.[**JUNE/JULY-2011,10M**]
2. Discuss the role of saturation logic. Explain with the help of a block diagram. [**june/july-2011,7M. June/july-2016,6M**]
3. With a neat block diagram, explain arithmetic logic unit (ALU) of DSP system.[**DEC-2011,6M**. **DEC-2016,6M**]
4. Explain the operation of Barrel shifter, with an example.[**dec-2011,5M. DEC-2016,5M**]
5. Explain i)Circular addressing mode ii)Parallelism iii)Guard bits.[**dec-2011,9M**]
6. Explain implementation of 4 bit shift left barrel shifter, with a neat diagram.[**june-2012,6M**]
7. Suggest a scheme to implement a multiplier to multiply two complex numbers using 4×4 Braun multiplier as the bulding blocks. .[**june-2012,6M**]
8. Draw a structure to multiply two 4 bit signed numbers A and B. .[**june-2012,8M**]
9. Give the structure of a 4×4 Braun multiplier, explain its concept. What modification is required to carry out multiplication of signed numbers? Comment on speed of the multiplier. [**DEC-2012,10M.DEC-2014,6M.dec-2014,8M.june/july-2015,6M.DEC-2025,8M. June/july-2016, 8M**]
10. Explain the guard bits in a MAC unit of a DSP. Consider a MAC unit whose inputs are 24-bit numbers. How many guard bits should be provided if 512 products have to be added in the accumulator to prevent overflow condition? What is the overall size of the accumulator required? ,[**dec-2012,10M. dec-2014,6M**]
11. Why circular buffers are required in DSP procesors? How they are implemented? ,[**dec-2012,2M. dec-2014,6M**]
12. Explain the different techniques used to prevent overflow condition and underflow condition occurring in MAC unit? [**june/july-2013,4M**]
13. Explain the different ways in which the on-chip memory can be organized efficiently and cost-effective manner. **[june/july-2013,4M]**
14. Explain the register pointer updating algorithm for circular buffer addressing mode. . **[june/july-2013,4M.dec-2014,8M]**
15. Compute the sequence in which the input data should be ordered for a 16 point DIT FFT using BIT reversed addressing mode.[**DEC-2015,4M]**
16. Explain implementation of 8- tap FIR filter, (i) pipelined using MAC units and (ii) parallel using two MAC units. Draw block diagrams.[**june/july-2014,6M**]
17. What is the role of a shifter in DSP? Explain the implementation of 4-bit shift right barrel shifter, with a diagram.[**june/july-2014,6M**]

18. Identify the addressing modes of the operands in each of the following instructions & their operations i)ADD B ii) ADD #1234h iii) ADD 5678h iv) ADD +*addreg **v)ADD** *addreg, offsetreg- **[june/july-2014,6M]**
19. It is required to find sum of 256 numbers each is represented by 16 bits. How many bits should the accumulator have so that the sum is computed without overflow?It is decided to have a accumulator with 16 bits, by how many bits each number is shifted to avoid overflow?[**june/july-2014,4M]**
20. With a neat block diagram, Explain address generation unit of DSP system**.[June/july-2026,6M]**
21. Explain how the circular addressing mode and bit reversal addressing mode are implemented in a DSP. **[DEC-2025,8M]**
22. What are the memory address of the operands in each of the following cases of indirect addressing model? In each case what will be the content of address register after the memory access? Assume that the in itial content of address register and the offset register are 0300h and 0020h i) ADD *addreg **ii)** ADD +*addreg **iii)** ADD +*offset, +*addreg **iv)** ADD *addreg,offset. **[DEC-2025,4M]**
23. Explain the purpose of program sequencer with a neat block diagram. **[June/july-2014, 6M. JUNE/JULY-2011,6M]**
24. With a neat block diagram explain ALU of DSP system. ,**[dec-2012,8M]**
25. Explain i) circular buffer addressing mode ii) Parallelism iii) Guard bits.**[DEC-2016,9M]**
26. The 256 unsigned numbers, 16 bit each are to be summed up in a processor. How many guard bits are needed to prevent overflow.**[JUNE/JULY-2011,3M]**
27. How will you implement an 8X8 multiplier using 4X4 multipliers as the building blocks.