CS 202 Computer Organization and Architecture
Module 5
Processor Logic Design

Sheena N

Assistant Professor
Dept. of CSE
College of Engineering & Management Punnapra

sheena.cemp@gmail.com

May 1, 2017

Overview

1. Register Trasfer Logic
2. Inter Register Transfer
3. Arithmetic Microoperation
4. Logic Micooperation
5. Shift Micooperation
6. Conditional Control Statements

Register Transfer Logic

- Digital system
  - Sequential logic system constructed with flip flops and gates
  - Specified by means of state table
- Difficult to specify a large digital system with state table because of the huge number of states.
- Modular approach
  - System is partitioned into modular subsytems with a specific functional task
  - Modules constructed from such digital functions are registers, counters, decorders, multiplexers, arithmetic elements and control logic
  - Modules are interconnected with common data and control paths to form a digital computer system .
- Typical digital system module - processor unit of a digital computer
- To decribe Digital system module high level mathemetical notations are used - Register Trasfer Logic

Register Transfer Logic (cont.)

- Register Trasfer Logic
  - Regiters are the primitive component
  - Information flow and processing tasks among the data stored in registers are described in precise and concise mannner
  - Uses set of expressions & statements which resemble the statements in programming language

Register Transfer Logic (cont.)

- Components that form the basis of register transfer logic
  1. The set of registers in the system and their functions
  2. The binary coded information stored in the registers
  3. The operations performed on the information stored in the registers - Microoperations
  4. The control functions that initiate the sequence of operations

Register Transfer Logic (cont.)

- Registers
  - Emcompasses all type of registers such as shift registers, counters and memory units
  - Counter :- Funtion is to increment by 1 the information stored within it
  - Memory unit :- Collection of storage registers where information can be stored
  - Flip flop :- stand alone flip flop is a 1 bit register
  - Flip flops and associated gates of any sequential circuit are called a register

Register Transfer Logic (cont.)

- The binary coded information stored in the registers
  - Binary information stored in registers may be binary numbers, binary coded decimal numbers, alphanumeric characters, control information or any other binary coded information
  - The operations performed on the data stored in registers depend upon the type of data
  - Numbers are manipulated with arithmetic operations
  - Control information is manipulated with logic operations such as setting and clearing specified bits in the register

Register Transfer Logic (cont.)

- Microoperation
  - Operations performed in data stored in registers
  - Elementary operation that can be performed parallel during one clock pulse period
  - The result of operation may replace the previous binary information of a register or may be transfered to another register
  - Eg. of microoperation - Shift, count, clear, add & load
  - A counter with parallel load - perform microoperations increment & load
  - Bidirectional shift register - Peform shift left & shift right microoperation
  - Binry parallel adder - Used for implementing add micooperation on the content of two registers that hold binary numbers
  - A microoperation requires one clock pulse for the execution if the operation done in parallel
  - In serial computer a microoperation requires a number of clock pulses equal to the word time in the system.

Register Transfer Logic (cont.)

- Control information
  - Timing signals that sequence the operations one at a time
  - Certain conditions depend on the result of previous operations determine the state of control functions
  - Control function is a binary variable that when in one binary state initiates an opeartion and when in the other binary state inhibits the operation
- Register transfer language(Computer hardware description language)
  - Symbolic notation used for registers, for specifying operations on the contents of registers and specifying control functions
  - A statement in a register transfer language consists of control function and a list of microoperations

Register Transfer Logic (cont.)

Types of micooperations in digital system

- Interregister transfer microoperation
  - Do not change the information content when the binary information moves from one register to another
- Arithmetic operation
  - Perform arithmetic on numbers stored in registers
- Logic microoperation
  - Perform operations such as AND and OR on individual pairs of bits stored in registers
- Shift microoperation
  - Specify operations for shift registers

Register Transfer Logic (cont.)

Categories of binary information found in registers of digital computers

- Numerical data such as binary numbers or binary coded decimal numbers used in arithmetic computations
- Nonnumerical data such as alphanumeric characters or any other binary coded symbols used for special applications
- Instruction codes, addresses and any other control information used to specify the data processing requirements in the system

Inter Register Transfer

- Registers in a digital system are designated by capital letters(sometimes followed by numerals) to denote function of register
- Eg :- Register that holds address of memory - Memory address register designated by MAR
  other designations are A, B, R1, R2 and IR
- The cells or flipflops of n-bit register are numbered in sequence from 1 to n (from 0 to n-1) starting either from left or from right

Inter Register Transfer (cont.)

- 4 ways to represent register
  - Rectangular box with name of the register inside
  - The individual cells is assigned a letter with a subscript number
  - The numbering of cells from right to left can be marked on top of the box
  - 16 bit register is partitioned into 2 parts , bits 1 to 8 are assigned the letter L(for low) and bits 9 to 16 are assigned the letter H(for high)

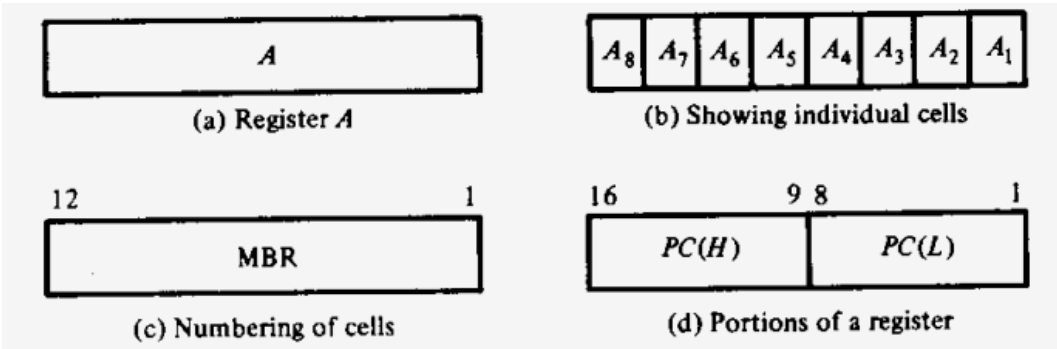

Figure: Block diagram of registers

Inter Register Transfer (cont.)

- Registers can be specified in a register transfer language with a declaration statement
- Eg :- Registers in the above figure can be defined with declaration statement such as
  DECLARE REGISTER A(8), MBR(12), PC(16)
  DECLARE SUBREGISTER PC(L) = PC(1-8), PC(H) = PC(9-16)
- Information transfer from one register to another is designated in sybolic form by means of replacement operator

$$A \leftarrow B$$

  Transfer of the contents of rgister B to register A.
  Content of source register do not change

Inter Register Transfer (cont.)

- Sometimes transfer occures under a predefined condition called control function which is a boolean function equal to 1 or 0
- Eg:-

$$x' T_1 : A \leftarrow B$$

Control function terminated with a colon
Transfer occurs when $x' T_1 = 1$ ie, x=0 and $T_1 = 1$

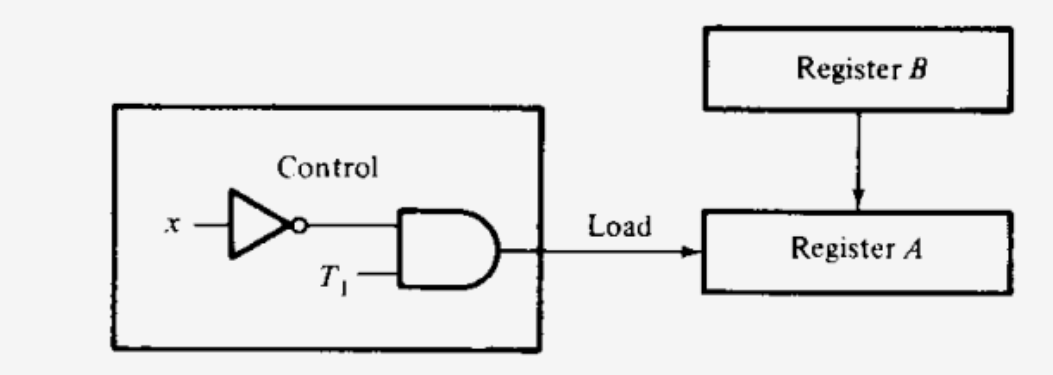

Figure: hardware implementation of the statement $x' T_1 : A \leftarrow B$

Inter Register Transfer (cont.)

Table: Basic symbols for register trnsfer logic

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	A, MBR, $R2$
Subscript	Denotes a bit of a register	A_2, B_6
Parentheses ( )	Denotes a portion of a register	$PC(H)$, $MBR(OP)$
Arrow $\leftarrow$	Denotes transfer of information	$A \leftarrow B$
Colon :	Terminates a control function	$x'T_0$:
Comma ,	Separates two microoperations	$A \leftarrow B$, $B \leftarrow A$
Square brackets [ ]	Specifies an address for memory transfer	$MBR \leftarrow M[MAR]$

Inter Register Transfer (cont.)

- Destination register receives information from two sources but not at the same time

$$T_1 : C \leftarrow A$$
$$T_2 : C \leftarrow B$$

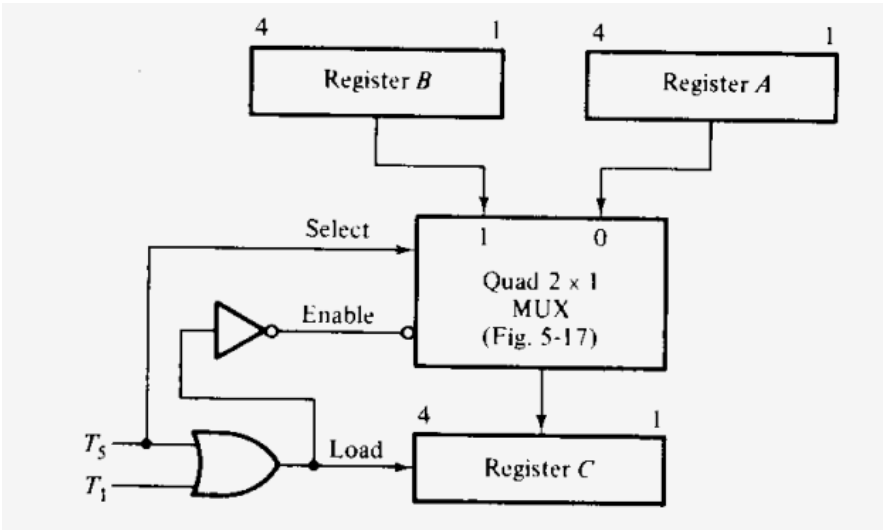

Figure: Use of multiplexer to transfer information from two sources into a single destination

Inter Register Transfer (cont.)

Bus transfer

- Paths must be provided to transfer information from one register to another in digital system

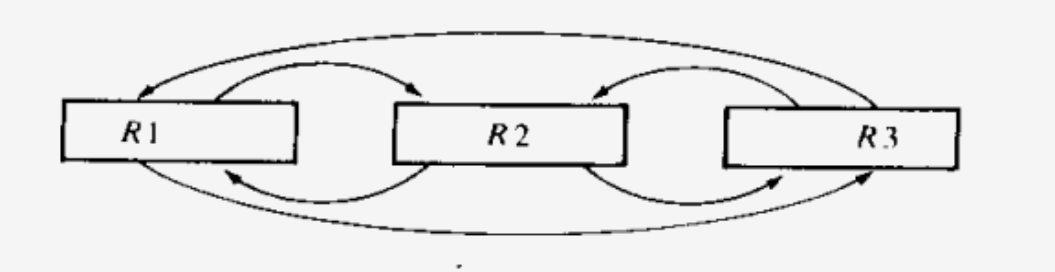

Figure: Transfer among three registers

- Eg :- Data tranfer among 3 registers - 6 data paths - each register requires a multiplexer to select between 2 sources
  If a register contains n flip flops - 6n lines - 3 multiplexers
- Number of registers increases number of interconnection lines and multiplexer increases

Inter Register Transfer (cont.)

- If restrict transfer one at a time then the number of paths among registers can be reduced

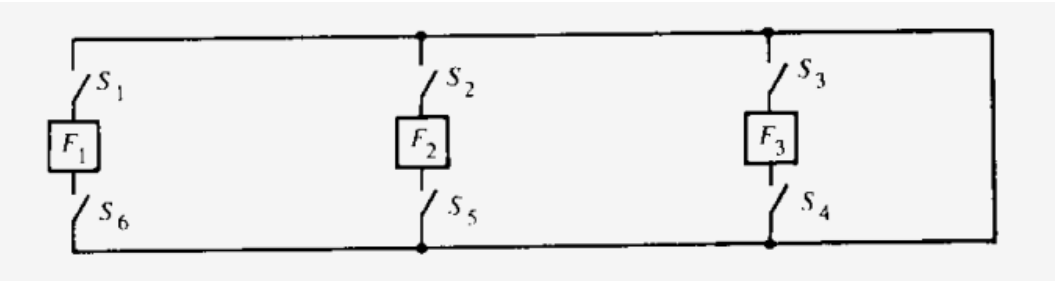

Figure: Transfer through one common line

- Each flipflop is connected to a common line through an electronic circuit that acts as a switch
- This scheme can be extended to registers with n flip flops and it requires n common lines

Inter Register Transfer (cont.)

- Bus :- Group of wires through which binary information is transfered
- For parallel transfer number of lines in the bus is equal to the number of bits in the registers
- A common bus system can be constructed with multiplexers and decoder
- Multiplexer selects one of the source register for the bus
- Decoder selects one destination register to transfer the information from the bus
- The 4-bits in the same significant position in the registers go through 4-to-1 line multiplexer to form one line of the bus
- 2 multiplexers are shown in figure - one for high order significant bit and other for low order significant bits
- For registers with n bits n multiplexers are needed to produce n line bus

Inter Register Transfer (cont.)

- n lines in the bus are connected to the n inputs of all registers
- The transfer of information from the bus into one destination register is accomplished by activting the load control of that register
- The particular load control activated is selected by outputs of the decoder when enabled.
- If decoder is not enabled no information will be transferred even though the multiplexers place the contents of a source register onto the bus
- Eg :- The statement $C \leftarrow A$ The control function that enables this transfer must select register A for the bus and register C for the destination
  The multiplexer and decoder selects inputs must be
  select source = 00(MUXs select register A)
  select destination = 10 (Decoder selects register C)
  Decoder enable =0 (Decoder is enabled)

Inter Register Transfer (cont.)

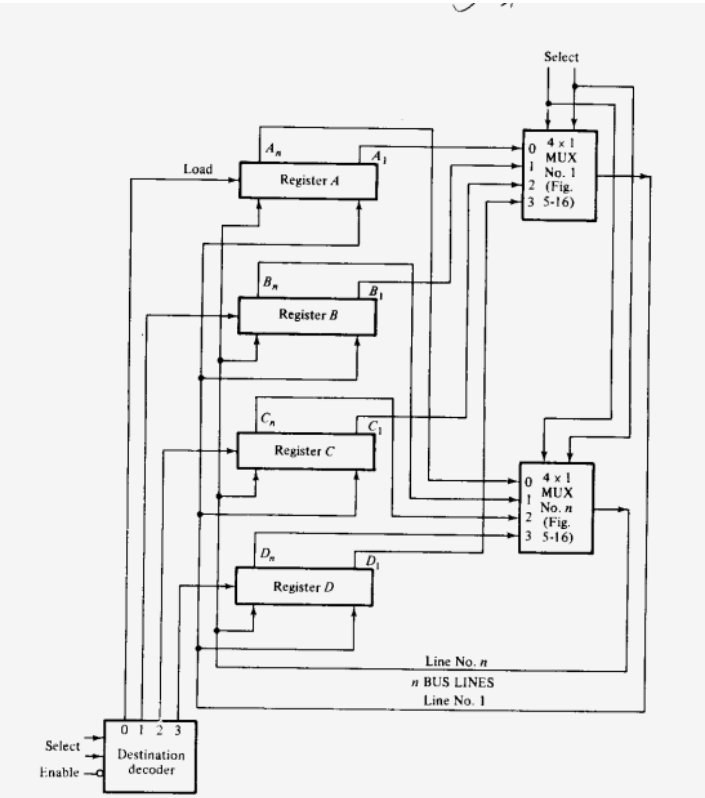

Figure: Bus system four registers

- on the next clock pulse the content of A, being on the bus, are loaded into register C

Inter Register Transfer (cont.)

Memory transfer

- Read operation :- Transfer of information from memory register to outside enviornment
- Write operation :- Transfer of new information into memory register selected
- Memory register is selected by an address Memory register is symbolised by the letter M

Inter Register Transfer (cont.)

- Only one address register is connected to the address terminals of memory
  - This register specifies the address
  - The letter M stands by itself in a statement designate a memory register selected by the address presently in MAR
  - A single memory buffer register MBR used to transfer data into and out of memory
  - 2 memory transfer operations - Read & Write
  - Read operation :- Transfer of information from selected memory register M specified by the address in MAR into MBR

    $$R : MBR \leftarrow M$$

    R is the control function that initiate the read operation
  - Write operation :- Trasfer of information from MBR into the register M selected by the address presently in MAR

    $$W : M \leftarrow MBR$$

    W is the control function that intiate write operation

Inter Register Transfer (cont.)

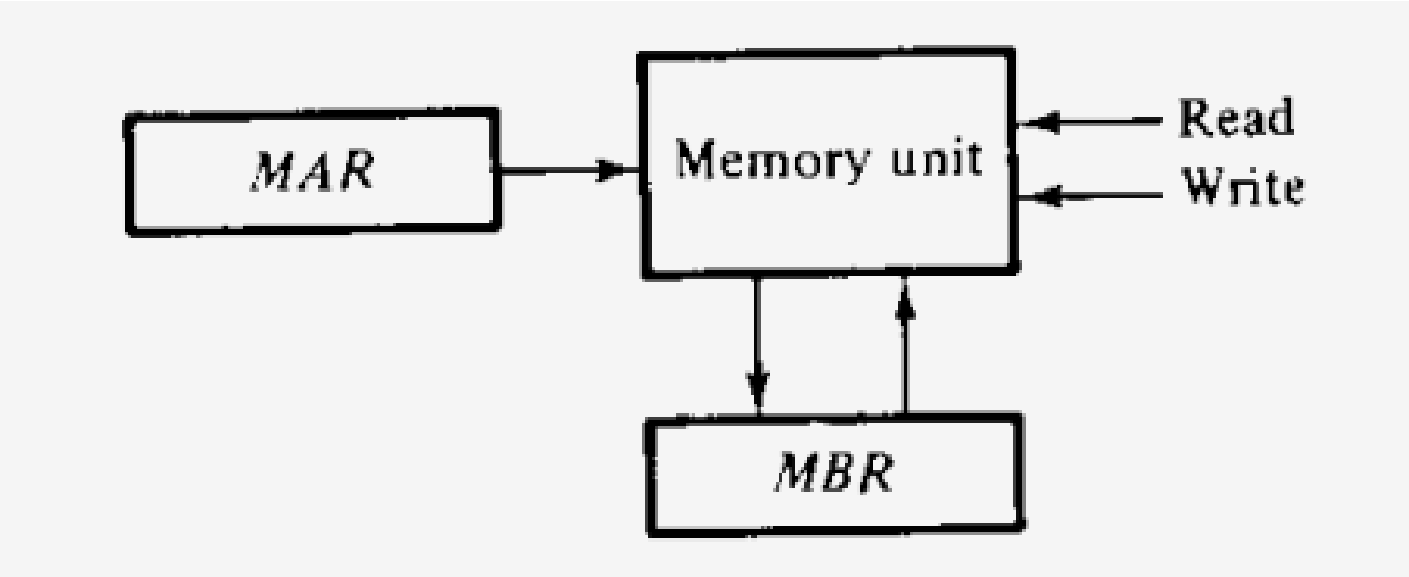

Figure: Memory unit that communicate with two external registers

Inter Register Transfer (cont.)

- The address lines form a common bus system to allow many registers to specify an address
  - The register specifies the address will be enclosed within square bracket after the symbol M
  - The address to the memory unit comes from an address bus
  - 4 registers are connected to the bus and any one may supply an address
  - The output of the memory can go to any one of 4 registers selected by the decoder
  - Data input to the memory comes from the data bus which selects one of the four registers
  - Write operation :- The transfer of information from register B2 to a memory word selected by the register A1 is

    $$\text{W: } M[A1] \leftarrow B2$$

  - Read operation :- R : $B0 \leftarrow M[A3]$

Inter Register Transfer (cont.)

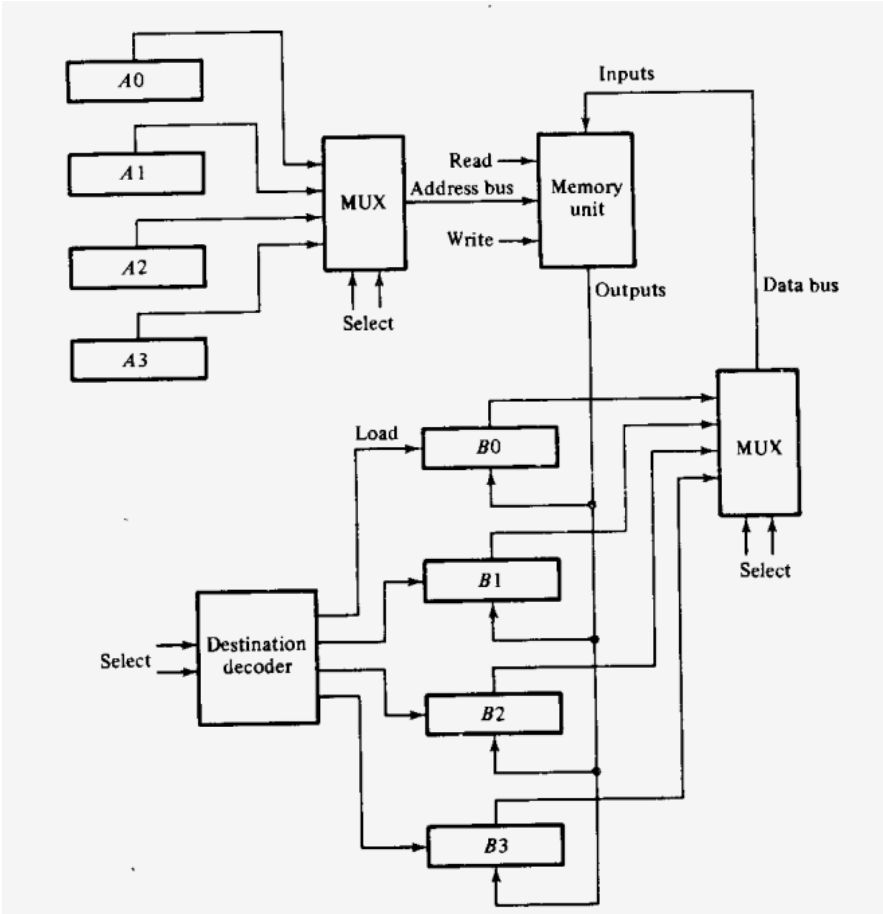

Figure: Memory unit that communicate with multiple registers

Arithmetic Microoperation

- Basic arithmetic microoperations - add, subtract, complement & shift
- The arithmetic add microoperations are defined by the statement

$$F \leftarrow A + B$$

  It states that the contents of register A are to be added to the contents of register B and the sum is transfered to register F
- To implement this statement require 3 registers A, B and F and a digital functon that performs the addition operation such as parallel addder

Arithmetic Microoperation (cont.)

Table: Arithmetic microoperation

Symbolic designation	Description
$F \leftarrow A + B$	Contents of A plus B transferred to F
$F \leftarrow A - B$	Contents of A minus B transferred to F
$B \leftarrow \overline{B}$	Complement register B (1's complement)
$B \leftarrow \overline{B} + 1$	Form the 2's complement of the contents of register B
$F \leftarrow A + \overline{B} + 1$	A plus the 2's complement of B transferred to F
$A \leftarrow A + 1$	Increment the contents of A by 1 (count up)
$A \leftarrow A - 1$	Decrement the contents of A by 1 (count down)

- Increment micropoperation symbolised by plus-one & implemented with an up counter
- Decrement micropoperation symbolised by minus-one & implemented with an down counter

Arithmetic Microoperation (cont.)

- There must be a direct relationship between the statements written in a register transfer language and the registers and digital functions which are required for the implementation
- Consider the statements

$$T_2 : A \leftarrow A + B$$
$$T_5 : A \leftarrow A + 1$$

Timing variable T_2 initiates an operation to add the contents of register B to the present contents of A with a parallel adder.
Timing variable T_5 increments register A with a counter.
The transfer of the sum from parallel adder into register A can be activated with a load input in the register.
Register be a counter with parallel load capability.

Arithmetic Microoperation (cont.)

- The parallel adder receives input information from registers A and B. The sum bits from the parallel adder are applied to the inputs of A and timing variable T_2 loads the sum into register A. Timing variable T_5 increments there by enabling increment input register
- Multiplication - Implemented with sequence of add & shift microoperations
- Division - Implemented with sequence of subtract & shift microoperations

Arithmetic Microoperation (cont.)

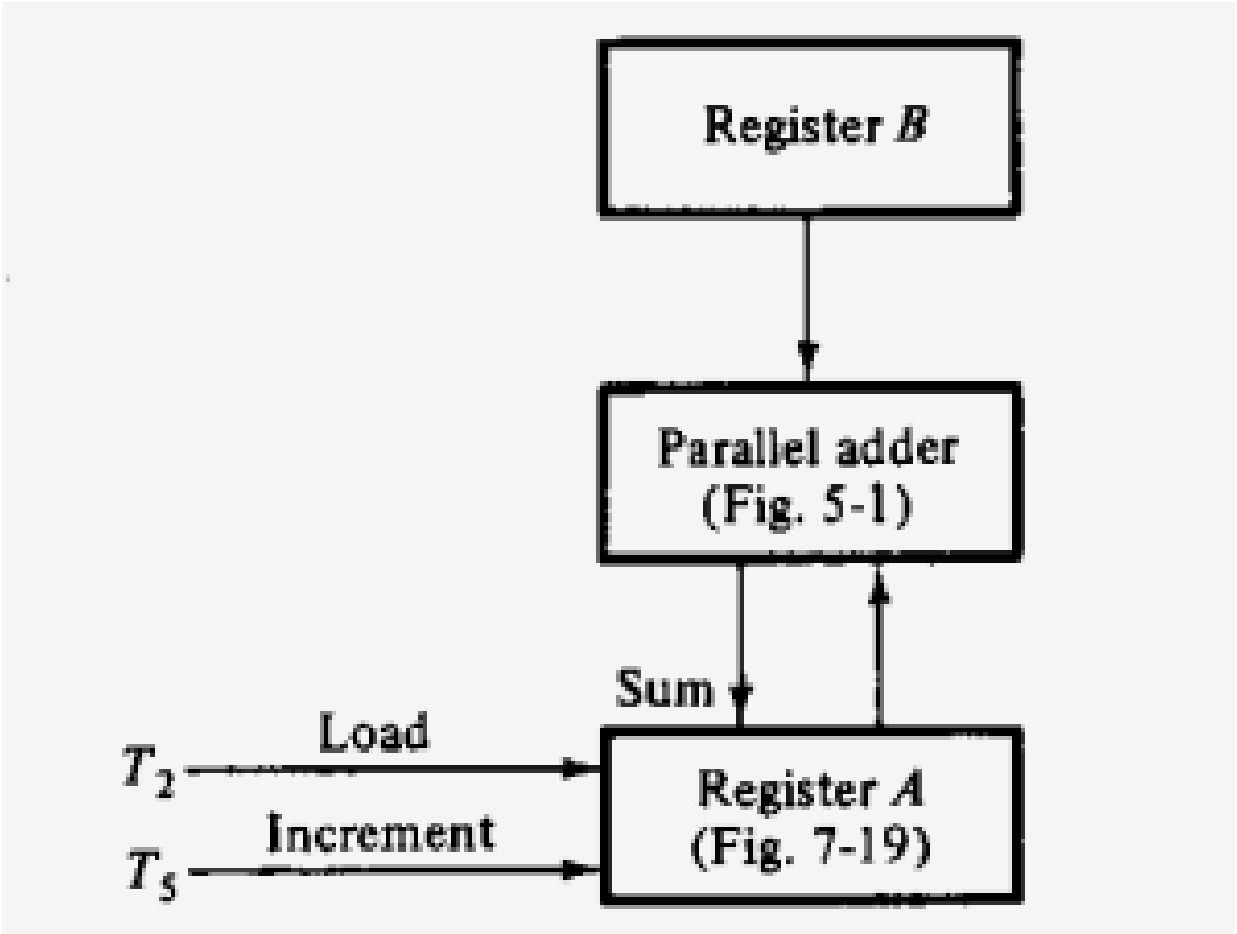

Figure: Implementation for addd and incrment microoperation

Logic Micooperation

- Specify binary operations for a string of bits stored in registers
- Consider each bit in the registers separately and treat as a binary variable
- Exclusive OR operation is symbolised by the statement

$$F \leftarrow A \oplus B$$

- The $+$ symbol has different meaning

$$T_1 + T_2 : A \leftarrow A + B, C \leftarrow D \vee F$$

The $+$ between T_1 and T_2 is an OR opeartion between 2 timing variables of a control function
The $+$ between A & B specifies an add microoperation

Logic Micooperation (cont.)

Table: Logic & shift microoperation

Symbolic designation	Description
$A \leftarrow \overline{A}$	Complement all bits of register A
$F \leftarrow A \vee B$	Logic OR microoperation
$F \leftarrow A \wedge B$	Logic AND microoperation
$F \leftarrow A \oplus B$	Logic exclusive-OR microoperation
$A \leftarrow \text{shl } A$	Shift-left register A
$A \leftarrow \text{shr } A$	Shift-right register A

Shift Micooperation

- Transfers binary information between registers in serial computers
- Used in parallel computers for arithmetic, logic and control operations
- Registers are shifted to the left or to the right
- No conventioal symbol for shift operation
- Here adopt symbols shl or shr
  shl - shift left
  shr - shift right
- Eg :-

$$A \leftarrow shl \text{ A - 1-bit shift to the left of register A}$$
$$B \leftarrow shr \text{ B - 1-bit shift to the right of register B}$$

- While the bits are shifted extreme flip flops receive information from the serial input

Shift Micooperation (cont.)

- Information transferred to extreme flip flop is not specified by shl or shr symbols
- Shift operation is accompanied with other microopeartion that specifies the value of the serial input for the bit transfer into the extreme fip flop
- Eg:-
  $A \leftarrow shl$, $A_1 \leftarrow A_n$ - Circular shift that tranfers the leftmost bit from A_n into the rightmost flipflop A_1
  $A \leftarrow shr$, $A_n \leftarrow E$ - Shift right opration with the leftmost flip flop A_n receiving the value of the 1-bit register E

Conditional Control Statements

- Specify a control condition by a conditional statement rather than a boolean control function
- Symbolised by if-then-else statement

  P : If(condition) then [microoperation(s)] else [microoperation(s)]

  mean that if control condition stated within the parentheses after the word if is true, then microoperation enclosed within the parentheses after the word then is executed otherwise the microoperation listed within after the word else is executed
- In any case the control function P must occur for anything to be done
- If else part of the statement is missing then nothing is executed if the condition is not true

Conditional Control Statements (cont.)

- Eg:-

$$T_2 : \text{If (C=0) then } (F \leftarrow 1) \text{ else } (F \leftarrow 0)$$

F is asssumed to be 1-bit register(flip flop) that can be set or cleared.

- If register C is a 1-bit register the statement is equivalent to the following statements

$$C'T_2 : F \leftarrow 1$$
$$CT_2 : F \leftarrow 0$$

- Same timing variable can occur in two separate control function. The valu of C is 0 or 1. So only one microoperation will be executed during T_2 depending on the value of C.
- If C has more than 1 bit the condition C = 0 means that all bits of C must be 0.
- Register C has 4 bits C_1, C_2, C_3 and C_4 the condition C=0 can be expressed with boolean function

Conditional Control Statements (cont.)

$$x = C_1' C_2' C_3' C_4' = (C_1 + C_2 + C_3 + C_4)'$$

variable x can be generated with a NOR gate.

- Conditional control statements now equivalent to 2 statements

$$xT_2 : F \leftarrow 1$$
$$x'T_2 : F \leftarrow 0$$

variable x = 1 if C = 0 but is equal to 0 if $C \neq 0$

A	B	D	F	C_{in}	H
011	011	011	100	0	010

Since we are selecting register R3 both A and B and destination D can select the bit pattern as 011. ALU will select OR operation by setting the code as 1000. Shift left operation can be specified by selecting H pattern as 010.

The control word for each microoperation is derived from the function table of the processor. The sequences of control words are stored in control memory. Based on the selection variables system will perform the sequence of micro operations. The scheme of producing control signals based on the control word is known as micro programmed control.

5.2.5 Design of Accumulator

Some processors distinguish one register from others and it is known as accumulator register. It is a multipurpose register capable of performing not only the add microoperation but many other microoperations as well. The organization of a processor unit with an accumulator register is shown below.

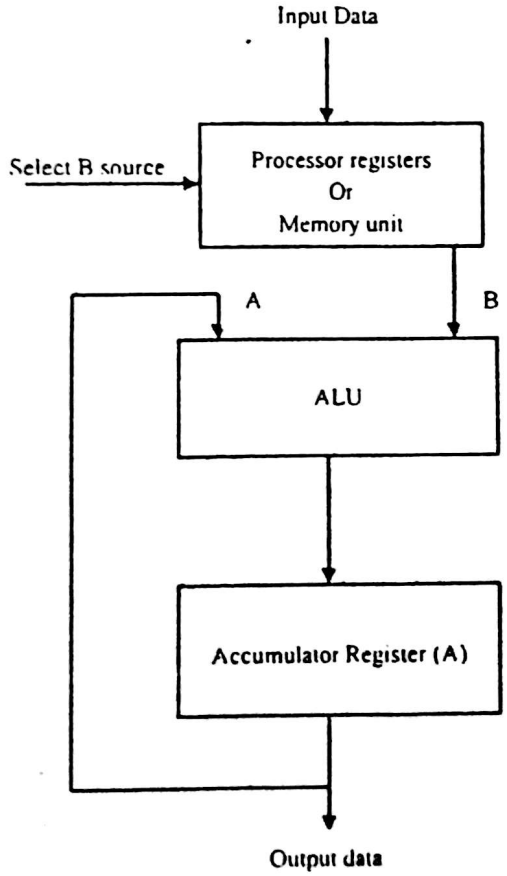

Fig 5.16: Processor with an accumulator register

The ALU associated with the register may be constructed as a combinational circuit. In this configuration, the accumulator register is essentially a bidirectional shift register with parallel load which is connected to the ALU. There is also a feedback connection from the output of accumulator register to one of the inputs in ALU. Because of this feedback connection, the accumulator register and its associated logic, when taken as one unit.

constitute a sequential circuit. The block diagram of the accumulator that forms as sequential circuit is shown below:

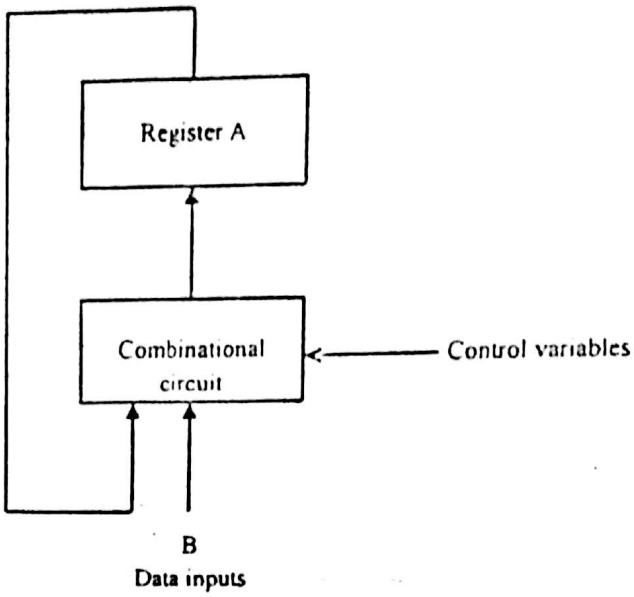

Fig 5.17: Block diagram of accumulator

The A register and associated combinational circuit constitute a sequential circuit. Here the combinational circuit replaces ALU, but it cannot be separated from the register. The accumulator register is denoted by A or AC (A register and associated combinational circuit).The external inputs to the accumulator are data inputs from B and the control variables (which will specify the microoperation to be executed).

Accumulator can also perform data processing operations .Total of nine operations are considered here for the design of accumulator circuit. These operations are described below.

Control variable	F with C_w=0	F with C_w=1
P1	A←A+B	Add
P2	A←0	Clear
P3	A←$\overline{A}$	Complement
P4	A←A ∧ B	AND
P5	A←AVB	OR
P6	A←A⊕B	Exclusive-OR
P7	A←shr A	Shift-right
P8	A←shl A	Shift-left
P9	A←A+1	Increment
Z bit	If(A=0)then(Z=1)	Check for zero

Table 5.9: List of microoperation for an accumulator

In all listed microoperations A is the source register. B register is used as the second source register. The destination register is also accumulator register itself. The nine control variables are considered as inputs to the sequential circuit. These variables are mutually exclusive. That means, only one variable must be enabled when a clock pulse occurs. The last entry in the table is a conditional control statement. It produces a binary 1 in the output variable Z when the content of register A is 0.

Design Procedure

Accumulator consists of n stages and n flip flops, numbered as A_1, A_2, A_3....An from right to left. It is convenient to partition the accumulator into n similar stages, with each stage consisting of one flip flop denoted by Ai, and one data input denoted by B_i, and the combinational logic associated with the flip flop. Each stage A_i is interconnected with neighbouring stage A_{i-1} on its right and A_{i+1} on its left. The register will be designed using JK type flip flops.

Each control variable P_i initiates a microoperation. Here nine microoperations are there by selecting control variables from P_1 to P_9. Accumulator is partitioned into n stages and each stage is again partitioned into those circuits that are needed for each microoperation. In the design procedure, we are designing various pieces separately and combine to form a one stage accumulator and then combine the stages to form a complete accumulator.

- **Add B to A (P_1)**

  Add microoperation is initiated when control variable P_1 is 1. To perform addition operation, accumulator can use a parallel adder composed of full adders. The full adder in each stage I will accept the input and (present state of A_i and data input B_i) and a previous carry bit C_i. Sum bit is transferred to flip flop A_i and output carries C_{i+1} is transferred to the next stage as input carry of that stage.

  The state table of a full adder, when considered as a sequential circuit is shown below.

Present State	Input		Next State	Flip-flop inputs		Output
A_i	B_i	C_i	A_i	JA_i	KA_i	C_{i+1}
0	0	0	0	0	X	0
0	0	1	1	1	X	0
0	1	0	1	1	X	0
0	1	1	0	0	X	1
1	0	0	1	X	0	0
1	0	1	0	X	1	1
1	1	0	0	X	1	1
1	1	1	1	X	0	1

Table 5.10: Excitation table for add microoperation

Columns of the table can be described as follows:

- Present state- the value of flip flop Ai before the clock pulse
- Next state- the value of flip flop Ai after the clock pulse (here, it will be determined by sum produced with inputs A_i, B_i and C_i)
- C_i - input carry
- C_{i+1} - output carry
- Flip flop inputs - shows the excitation input for the JK flip flop.

The excitation table of JK flip flop is shown below for reference.

Present state	Next State	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 5.11: List of microoperation for an accumulator

According to these values the above flip flop inputs are set. The flip flop input functions and the Boolean functions for the output are simplified in the maps as shown in fig 5.19.

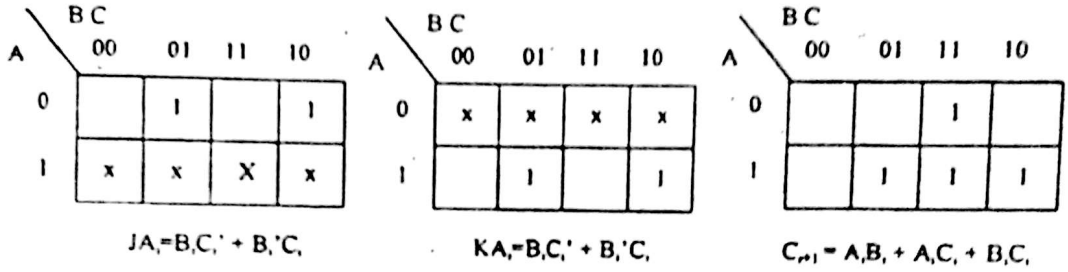

Fig 5.18: Map simplification for Add Microoperation

The J input of flip-flop Ai, designated by JA_i and the K input of flip flop A_i, designated by KA_i, do not include the control variable p1. These two equations should affect the flip flop only when P_1 is enabled. Therefore, they should be ANDed with control variable P_1. Then the equation becomes,

$$JA_i = B_iC_i'P_1 + B_i'C_iP_1$$

$$KA_i = B_iC_i'P_1 + B_i'C_iP_1$$

$$C_{i+1} = A_iB_i + A_iC_i + B_iC_i$$

- **Clear (P_2)**

Control variable P_2 clears all flip flops in register A. To cause this transition in a JK flip flop, we need only apply control variable P_2 to the K input of the flip flop. The J input will be assumed to be 0 if nothing is applied on it. The input functions can be written as:

$$JA_i = 0$$

$$KA_i = P_2$$

- **Complement(P_3)**

To cause this transition in a JK flip flop we need to apply P_3 to both J and K inputs.

$$JA_i = P_3$$

$$KA_i = P_3$$

- **AND (P_4)**

This microoperation is initiated with control variable P_4.This operation performs the logic AND operation between A_i and B_i and transfers the result to A_i. The excitation table for this operation is as shown below.

Present State	Input	Next State	Flip-flop Inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	0	0	X
1	0	0	X	1
1	1	1	X	0

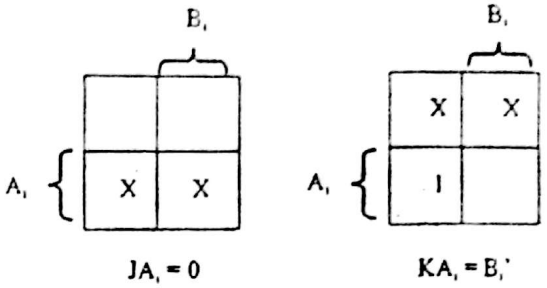

Fig 5.19: Excitation table for AND microoperations

The next state of A_i will be 1 only when the present state of A_i and data input B_i is 1s. The flip flop input functions can be simplified with the maps and the equations can be written as:

$$JA_i = 0$$

$$KA_i = B_i'$$

By including the control variable p4, the equation can be rewritten as:

$$JA_i = 0$$

$$KA_i = B_i'P_4$$

- **OR (P_5)**

Control variable P_5 initiates the logic OR operation between A_i and B_i. The result is transferred to A_i. The excitation table for this operation is as shown below.

Present State	Input	Next State	Flip-flop inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	1	X	0

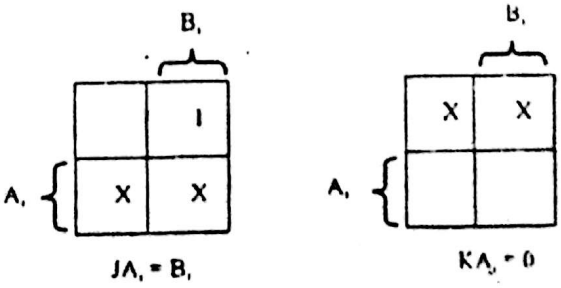

Fig 5.20: Excitation table for OR microoperations

The simplified equations in the maps dictate that the J input be enabled when $B_i=1$. When $B_i=0$, the present state and next state of Ai are the same. When $B_i=1$, the J input is enabled and the next state of A_i becomes 1. Input functions for the OR microoperation are:

$$JA_i = B_iP_5$$

$$KA_i = 0$$

- **Exclusive-OR (P_6)**

Control variable P_6 initiates the logic Exclusive-OR operation between A_i and B_i. The result is transferred to A_i. The excitation table and map simplification is as shown below.

Present State	Input	Next State	Flip-flop Inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	0	X	1

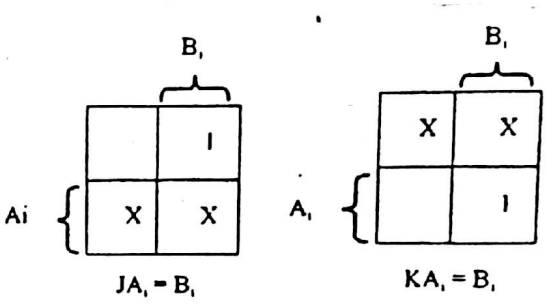

Fig 5.21: Excitation table for Exclusive-OR microoperations

The flip flop input functions are written as:

$$JA_i = B_iP_6$$

$$KA_i = B_iP_6$$

- **Shift-right (P_7)**

Control variable P_7 initiates the shift operation of A_i register one bit to the right. That is, the value of flip flop A_{i+1} is transferred to flip flop A_i. The flip flop input functions can be written as:

$$JA_i = A_{i+1}P_7$$

$$KA_i = A'_{i+1}P_7$$

- **Shift-left (P_8)**

Control variable P_8 initiates the shift operation of A_i register one bit to the left. That is, the value of flip flop A_{i-1} is transferred to flip flop A_i. The flip flop input functions can be written as:

$$JA_i = A_{i-1}P_8$$

$$KA_i = A'_{i-1}P_8$$

- **Increment (P_9)**

These operations increment the content of A register by one. The register behaves likes a synchronous binary counter with P_9 enabling the count. A 3 bit synchronous counter is shown in the following figure.

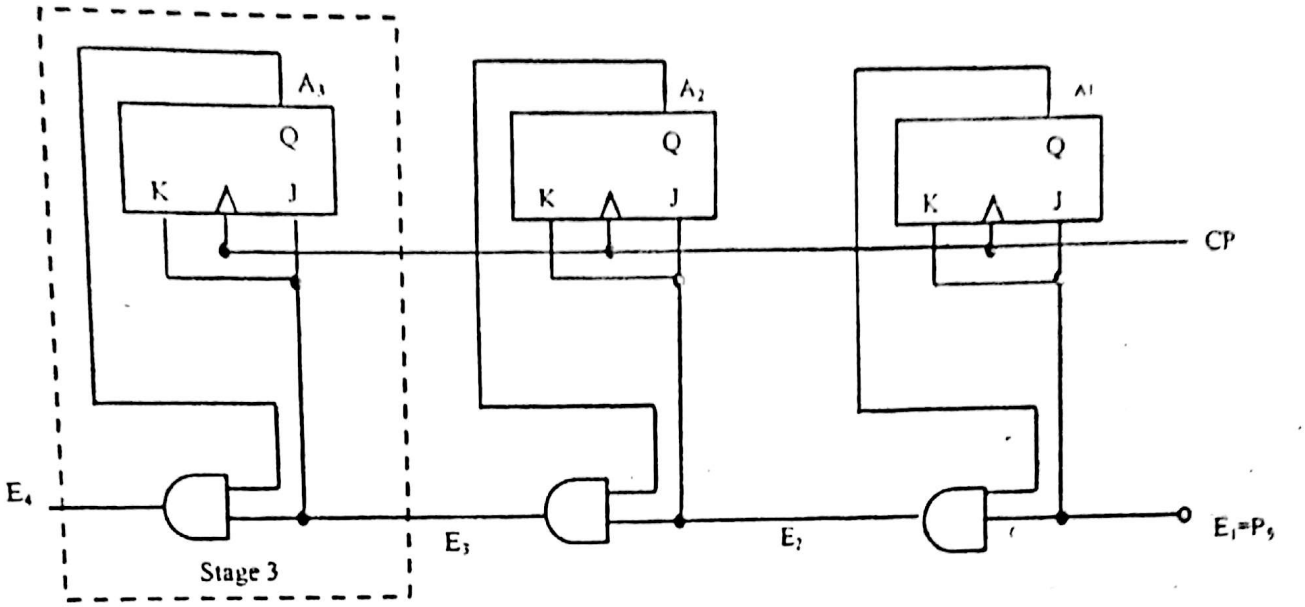

Fig 5.22: 3-bit synchronous binary counter

From the figure it can see that each stage is complemented when an input carry $E_i=1$. Each stage is generating an output carry E_{i+1} that is fed to the next stage on its left. The first stage is an exception, since it is complemented with the count-enable P_9. The Boolean function for a typical stage can be written as:

$$JA_i = E_i$$

$$KA_i = E_i$$

$$E_{i+1} = E_i A_i \qquad i=1,2,\ldots,n$$

$$E_1 = P_9$$

Input carry to the first stage of counter is E_1. It must be equal to the control variable p9 which enables the count. The input carry E_i is used to complement flip flop A_i. The input carry is ANDed with A_i to generate a carry for the next stage.

- **Check for Zero (Z)**

Variable Z is an output from the accumulator. This variable can be used to indicate a zero content in the A register. All the flip flops in the accumulator is cleared Z variable will be set to 1. When a flip flop is cleared, its complement output Q' is equal to 1.

The following figure shows the first three stages of the accumulator that checks for zero content. Each stage generates a variable Z_{i+1} by ANDing the complement output of A_i to an input variable Z_i. In this way, a chain of AND gates through all stages will indicate if all flips are cleared.

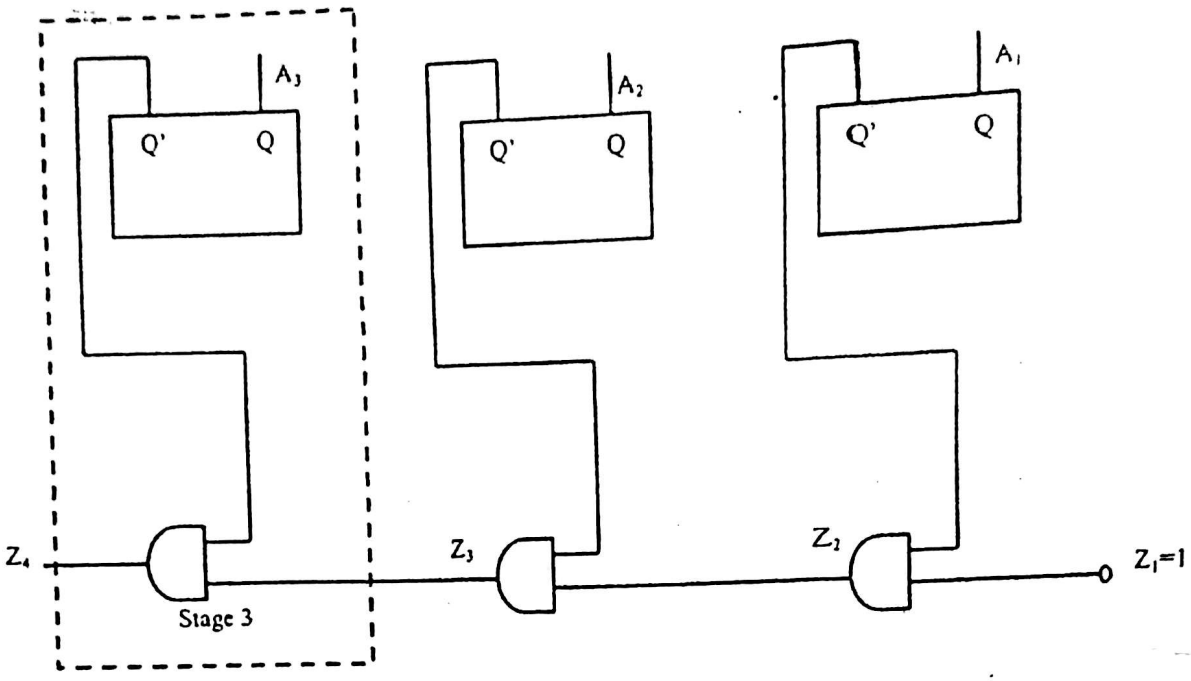

Fig 5.23: Chain of AND gates for checking the zero content of a register

The Boolean function for a typical stage can be expressed as:

$$Z_{i+1}=Z_iA_i' \qquad i=1,2,\ldots,n$$

$$Z_1=1$$

$$Z_{n+1}=Z$$

Variable Z becomes 1 if the output signal from the last stage Z_{n+1} is 1.

One stage of Accumulator

In the earlier sections we have derived the logic circuits for each individual microoperation that can be performed by the Accumulator. Now, we can combine them to all to form one stage of the Accumulator circuit. Combining all the input functions for the J and K inputs flip flop A_i produces a composite set of input Boolean functions for a typical stage.

$$JA_i=B_iC_i'P_1 + B_i'C_iP_1+P_3+B_iP_5+ B_iP_6+A_{i+1}P_7+A_{i-1}P_8+E_i$$

$$KA_i=B_iC_i'P_1 + B_i'C_iP_1+P_2+P_3+B_i'P_4+ B_iP_6+ A'_{i+1}P_7+ A'_{i-1}P_8+ E_i$$

Each stage in the accumulator must produce the carry for the next stage.

$$C_{i+1}= A_iB_i+ A_iC_i +B_iC_i$$

$$E_{i+1}= E_iA_i$$

$$Z_{i+1}=Z_iA_i'$$

The logic diagram for one typical stage of the Accumulator is shown below:

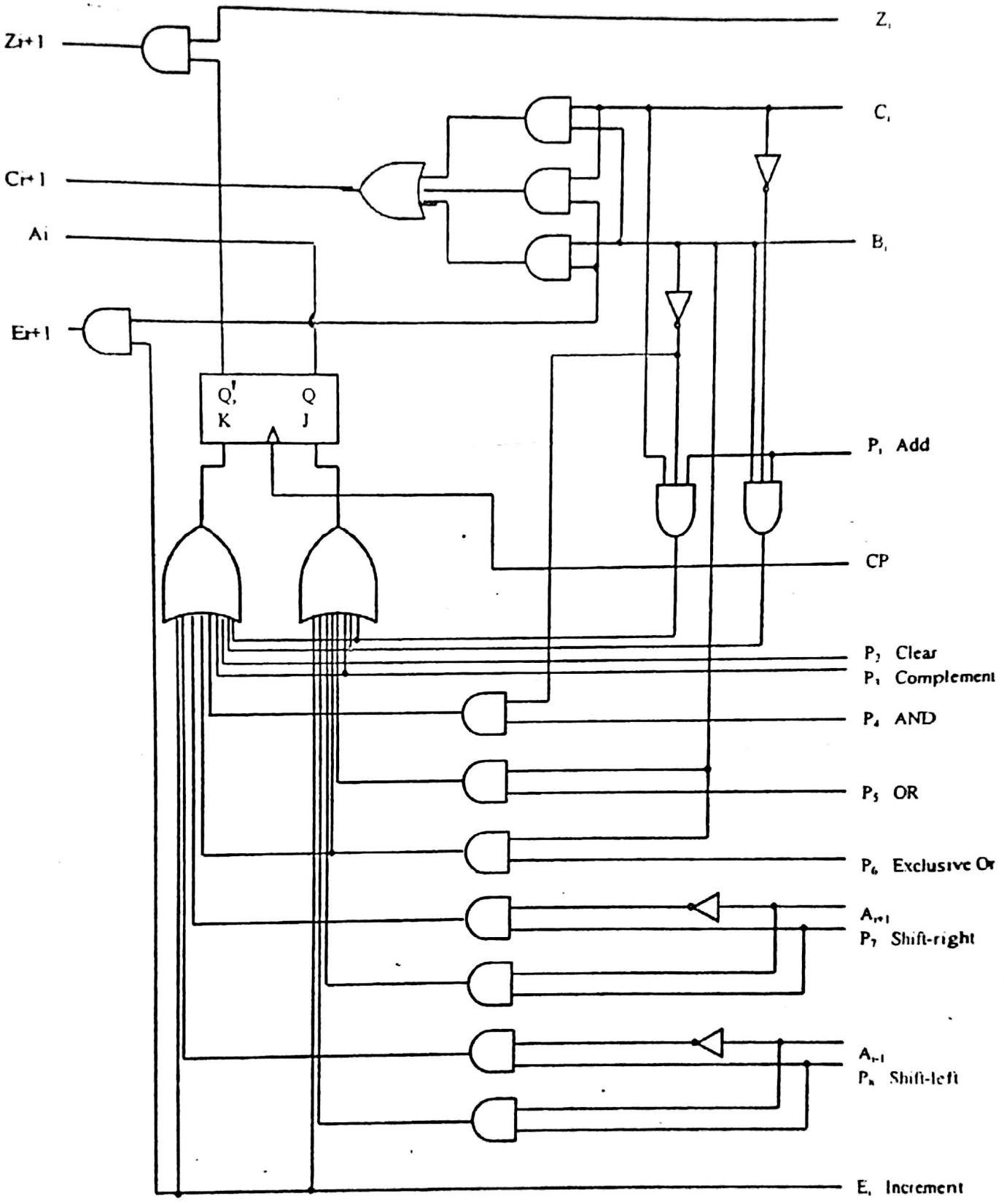

Fig 5.24: One typical stage of the accumulator

Each accumulator stage has eight control inputs from P_1 to P_8 that initiates one of eight possible microoperations. Control variable P_0 is applied only to the first stage to enable the increment operation through input E_1.

There are six other inputs in the circuit. Data input B_i, input carry C_i,A_{i-1} comes from the flip flop one position to the right,A_{i+1} comes from the flip flop one position to the left, E_i(carry input for the increment operation),and Z_i(used to form chain of zero detection).

There are four outputs to this circuit.

A_i	-	output of the flip flop
C_{i+1}	-	carry for the next stage
E_{i+1}	-	increment carry for the next stage
Z_{i+1}	-	carry for the next stage for zero detection.

Complete Accumulator

We have covered the design of one stage of Accumulator. For a complete accumulator there will be n stages like this. The inputs and outputs of each stage can be connected in cascade to form a complete accumulator. Here we are discussing the design of a 4 bit accumulator. The following diagram shows the design.

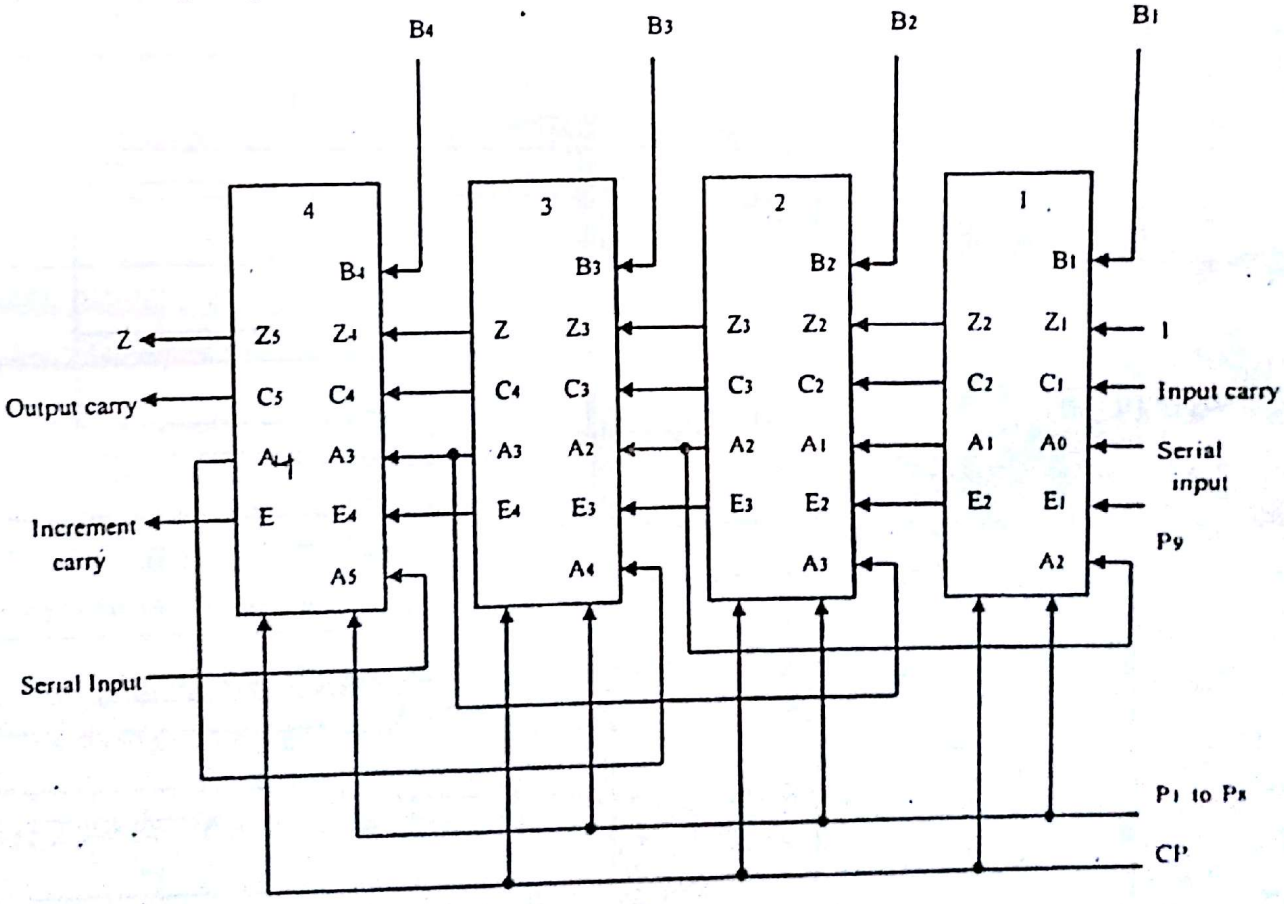

Fig 5.25: 4-bit accumulator constructed with four stages

The number on top of each block represents the bit position. All blocks receive 8 control variables p1 to p8 and the clock pulses from CP. The other six inputs and four outputs are same as with the typical stage.

The zero detect chain is obtained by connecting the z variables in cascade, with the first block receiving a binary constant 1. The last stage produces the zero detect variable Z.

Total number of terminals in the 4 bit accumulator is 25, including terminals for the A outputs. Incorporating two more terminals for power supply, the circuit can be enclosed within one IC package having 27 or 28 pins. The number of terminals for the control variable can be reduced from 9 to 4 if a decoder is inserted in the IC. In such cases, IC pin count is also reduced to 22 and the accumulator can be extended to 16 microoperations without adding external pins (That is, with 4 bits we can identify 16 operations).

6.1 Control Logic Design

Control unit is the nerve centre of a computer system. The main responsibility of control unit is the generation of timing and control signals. In this section, we are discussing the different methods of control unit design.

6.1.1 Control Organization

The control unit is the circuitry that controls the flow of information through the processor, and coordinates the activities of the other units within it. In a way, it is the "brain within the brain", as it controls what happens inside the processor, which in turn controls the rest of the PC. Control units control the flow of information with the help of control signals.

Functions of Control Unit

The control unit directs the entire computer system to carry out stored program instructions. The control unit must communicate with both the arithmetic logic unit (ALU) and main memory. The control unit instructs the arithmetic logic unit that which logical or arithmetic operation is to be performed .The control unit co-ordinates the activities of the all other units as well as all peripherals and auxiliary storage devices linked to the computer.

Design of Control Unit

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. As for example, during the fetch phase, CPU has to generate PC_{out} signal along with other required signal in the first clock pulse. In the second clock pulse CPU has to generate PC_{in} signal along with other required signals. So, during fetch phase, the proper sequence for generating the signal to retrieve from and store to PC is PC_{out} and PC_{in}.

To generate the control signal in proper sequence, a wide variety of techniques exists. Most of these techniques, however, fall into one of the two categories.

1. **Hardwired Control**
2. **Micro programmed Control**

Hardwired control units are constructed using digital circuits and once formed it cannot be changed. A micro programmed control unit itself decodes and execute instructions by means of executing micro programs.

6.1.2 Design of Hardwired Control

In this hardwired control techniques, the control signals are generated by means of hardwired circuit. The main objective of control unit is to generate the control signal in proper sequence.

Consider the sequence of control signal required to execute the add instruction. It is obvious that seven non-overlapping time slots are required for proper execution of the instruction represented by this sequence. Each time slot must be at least long enough for the function

specified in the corresponding step to be completed. Since , the control unit is implemented by hardwire device and every device is having a propagation delay , due to which it requires some time to get the stable output signal at the output port after giving the input signal. So, to find out the time slot is a complicated task. For the moment, for simplicity, let us assume that all slots are equal in diameter. Therefore the required controller may be implemented based upon the use of a counter driven by clock. Each state, or count, of this counter corresponds to one of the steps of the control sequence of the instructions of the CPU.

In the previous discussion (Refer chapter 2, section 2.1.3), we have mentioned control sequence for execution of two instructions only (one is for add and other one is for branching). Like that we need to design the control sequence of all the instructions. By looking into the design of the CPU, we may say that there are various instruction for add operation. As for example,

Add (R3), R1 Add the contents of memory location specified by R3 to the contents of register R1.

$R1 \leftarrow R1 + [R3]$

Add R2, R1 Add the contents of register R2 to the contents of register R1

$R1 \leftarrow R1 + R2$

The control sequence for execution of these two adds instructions are different of course, the fetch phase of all the instruction remain same. It is clear that control signals depend on the instruction, i.e. the contents of the instruction register. It is also observed that execution of some of the instructions depend on the contents of condition code or status flag register, where the control sequence depends in conditional branch instruction. Hence, the required control signals are uniquely determined by the following information:

- Contents of the control counter.
- Contents of the instruction register.
- Contents of the condition code and other status flags.

The status flags represent the various state of the CPU and various control lines connected to it, such as MFC status signal. The structure of control unit can be represented in a simplified view by putting it in block diagram. The detailed hardware involved may be explored step by step. The simplified view of the control unit is given in the figure 6.1.

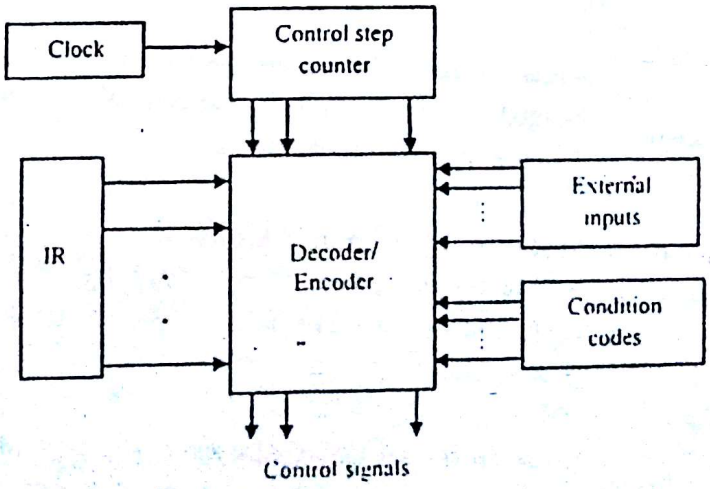

Fig 6.1: Control Unit Organization

The decoder/encoder block is simply a combinational circuit that generates the required control outputs depending on the state of all its input. The decoder part of decoder/encoder part provides a separate signal line for each control step, or time slot in the control sequence. Similarly, the output of the instruction decoder consists of a separate line for each machine instruction loaded in the IR, one of the output lines INS_1 to INS_n be set to 1 and all other lines are set to 0.

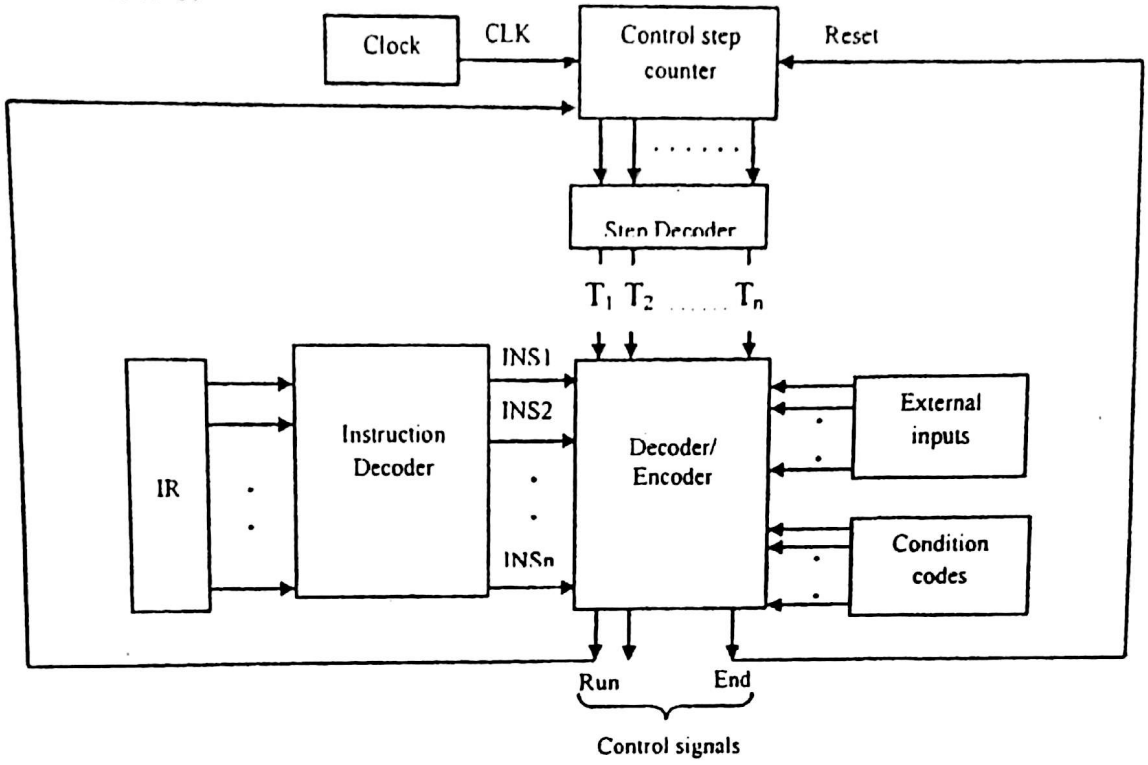

Fig 6.2: Control Unit Organization

All input signals to the encoder block should be combined to generate the individual control signals. Control step counter keeps tracks of the count of control steps. Each state or count of this control step corresponds to one control step. The step decoder provides a separate signal line for each step or time slot in the control sequence. The output of the instruction decoder consists of a separate signal line for each instruction. For any instruction loaded in the IR, one of the output lines INS_1 through INS_n is set to 1 and all other lines are set to zero. The encoder circuit combines all of these inputs and generates the control signals Y_{in}, Z_{out} etc.

It is required to generate many control signals by the control unit. These are basically coming out from the encoder circuit of the control signal generator. The control signals are lie PC_{in} , PC_{out} , Z_{in} , Z_{out} , MAR_{in} , add , End etc. In the previous discussions (chapter 2), we have mentioned the control sequence of the instruction, "Add (R3), R1", and "Control sequence for an unconditional branch instruction (BR)". Consider those CPU instructions.

By looking into the above instructions. we can write the logic function for Z_{in} as:

$$Z_{in} = T_1 + T_6.ADD + T_4.BR + \ldots\ldots\ldots\ldots$$

This means, control signal Z_{in} have to be generated for time cycles T1 for all instructions. in time cycle T6 for add instruction and in time cycle T_4 of BR instruction and so on.

Similarly, the Boolean logic function for add signal is

$$add = T_1 + T_6.ADD + T_4.BR + \ldots\ldots\ldots\ldots$$

These logic functions can be implemented by a two level combinational circuit of AND and OR gates. Similarly, the End control signal is generated by the logic function:

$$End = T_7.ADD + T_5.BR + (T_5.N + T_4.\overline{N}).BRN + \ldots\ldots\ldots\ldots$$

This End signal indicates the End of the execution of an instruction, so this End signal can be used to start a new instruction fetch cycle by resetting the control step counter to its starting value. The circuit diagram (Partial) for generating Z_{in} and End signal is shown in the diagram.

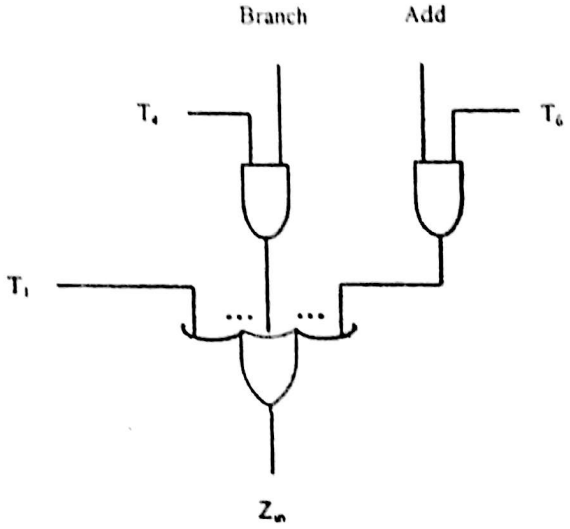

Fig 6.3: Generation of Z_{in} signal for the processor

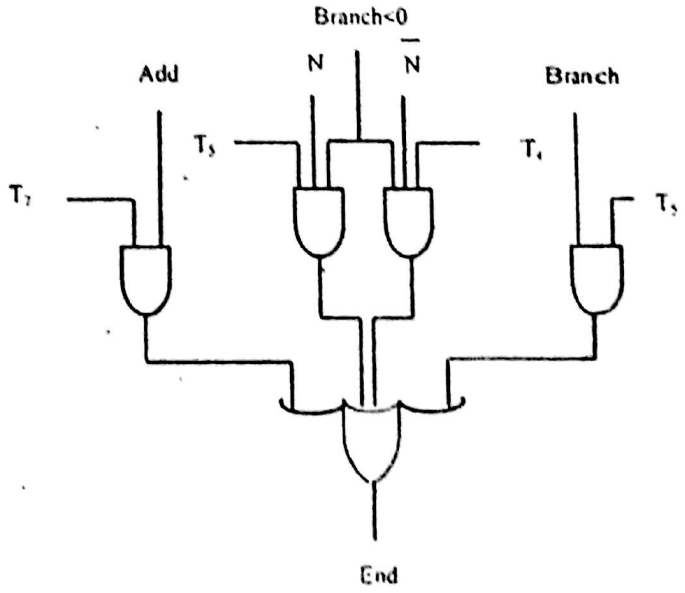

Fig 6.4: Generation of the End control signal

The signal ADD, BR, BRN etc are coming from instruction decoder circuits which depends on the contents of IR. The signal T_1, T_2, T_3 etc are coming out from step decoder depends on control step counter. The signal N (Negative) is coming from condition code register. When wait for MFC (WMFC) signal is generated, then CPU does not do any works and it waits for

an MFC signal from memory unit. In this case, the desired effect is to delay the initiation of the next control step until the MFC signal is received from the main memory. This can be incorporated by inhibiting the advancement of the control step counter for the required period. Let us assume that the control step counter is controlled by a signal called RUN. When set to 1, RUN causes the counter to be incremented by one at the end of every clock cycle. When RUN equal to zero, the counter stops counting. This is needed whenever the WMFC signal is issued, to cause the processor to wait for the reply from the memory. End signal starts a new instruction fetch cycle by resetting the control step counter to its starting value. The control hardwire we have discussed now 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, condition codes and external inputs. The output of the state machines are the control signals. The sequence of operations carried out by this machine is determined by the wiring of the logic elements; hence it is named as "Hardwired unit".

Advantages

1. They Operates at high speed.

Disadvantages

1. Have little flexibility.
2. Complexity of the instruction set it can implement is limited.

A complete Processor

A complete processor can be designed using the following structure as shown in figure 6.5.This structure has an instruction unit that fetches instruction from an instruction cache or from the main memory. To deal with integer and floating point data it has separate processing units. A data cache is inserted between the main memory and integer and floating point units. The processor is connected to the system bus and hence to the rest of the computer by means of a bus interface. A processor may include several units of integer and floating point units so that the rate of instruction execution will be increased.

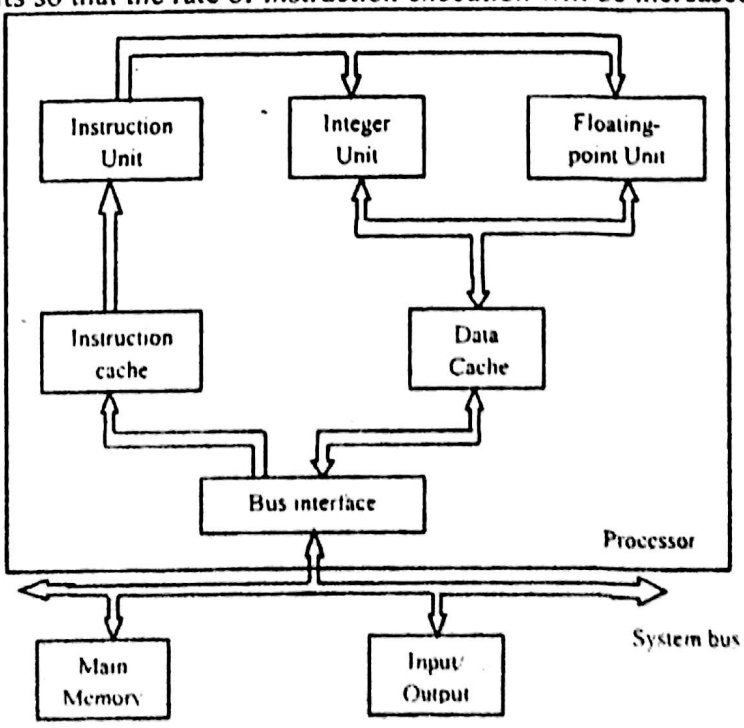

Fig 6.5: Block diagram of a complete Processor