COMPUTER ARCHITECTURE paper - IV

Assignment - I

1. Explain about fixed point and floating point Repressention.

Fixed point Represention

This represention has fixed number of bits for integer part and for fractional part. for example if given fixed point represention is 1111 , FFFF , they you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed point number representation . the sign field, integer field and fractional field .

usigned fixed point	integral	Fraction	
Signed fixed point	sign	integer	fraction

we can represent these number using

⇒ signed representation: range from $-(2^{(k-1)}-1)$ to $2^{(k-1)}-1)$ for k bits

⇒ 1's complement representation: range from $-(2^{(k-1)}-1)$ to $2^{(k-1)}-1)$ for k bits

⇒ 2's complementation representation: range from $-(2^{(k-1)})$ to $(2^{(k-1)}-1)$ for k bits

2's complementation representation is prefered in computer system because of unambiguous property and easier for arithemetic operations

example: Assume number is using 32-bit format which reserve 1 bit for the sign 15 bits for the integer part and 16 bits for the fractional part.

Then -43.625 is represented as following

1	000000000101011	1010000000000000
Sign bit	integer part	Fractional part

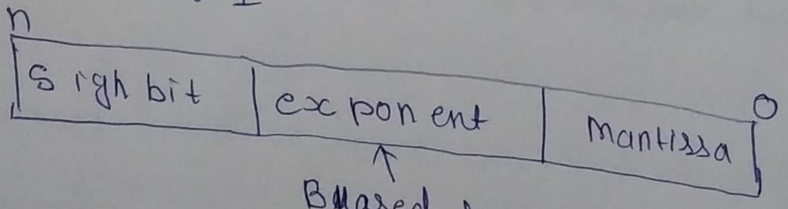
where, 0 is used to represent + and 1 is used to represent, 00000000000101011 is 15 bit binary value for decimal 43 and 1010000000000000 is 16 bit binary value for fractional 0.625

Floating point Representation

This representation doesnot reserve a specific number of bits for the integer part or the fractional part. Instead it reverse a certain number of bits for the number and certain number of bits to say where within that number the decimal place sits.

The floating number representation of a number has two part the first part represents a signed fixed point number called mantissa.. flating point is always interpreted to represent a number in the following $M \times r^e$

only the mantissa m and the exponent e are physically represented in the register. A floating-point binary number is represented in a similar mannar except that is uses base 2 for the exponent. A floating point number is said to be normalized if the most significant digit of the mantissa 1



So, actual number is $(-1)^s (1+m) \times 2^{(e-Bias)}$, where S is the sign bit, m is the mantissa, e is the exponent value, and Bias is the bias number.

Note that signed integers and exponent are represented by either sign rebrentation or ones complement representation, or two's complement representation.

The floating point representation is more flexible. Any non-zero number can be represented in the normalized from of $\pm (1.b_1 b_2 b_3) 2 \times 2^n$ This is normalized from of a number x.

2. What are CPU Registers? Explain Them.

CPU registers are small fast storage location within the centeral processing unit (CPU) of a computer. They are used to hold data that the CPU needs to acces quickly while performaning operations. Here som common type of CPU registers and their function.

Accumulator (ACC) used to store Intermediate arithemetic and logic results.

program counter (PC) Holds the address of the next instruction to be executed.

Instruction Register (IR) Contains the current instruction being executed

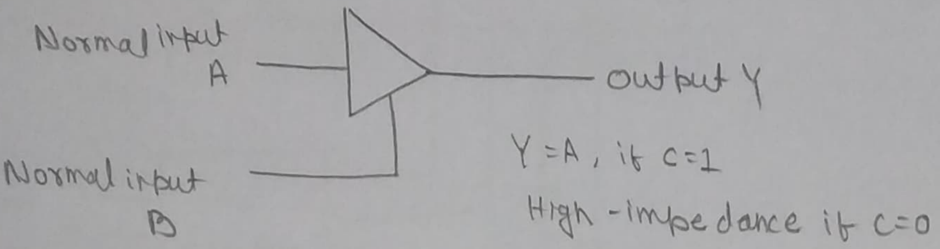
Memory Address Register (MAR) Holds the memory address of data that needs to be accessed.

Memory data Register (MDR) Store the data being transfered to or from the memory location pointed to by the MAR

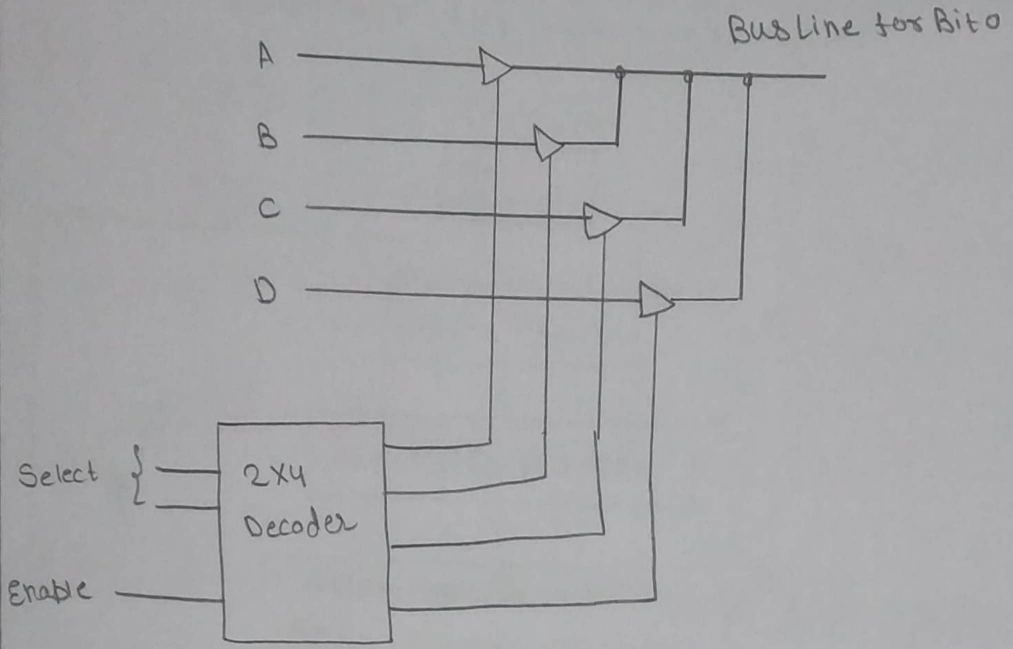
Status Register: Holds flags that indicate the status of the CPU, such as zero carry, overflow and sign flags

3. Construct a Bus System for four Registers using three state Bus Buffers.

A bus system can also be constructed with three-state gates instead of multiplexers. The three State gate is a digital circuit that exhibits three States. Two of the states are signals, equivalent to Logic 1 and 0 as in a conventional gate the third state is high-impedance state. Three gats may perform any conventional Logic, such as AND or NAND. However the one most commonly used in the design of a bus system is the buffer gate.



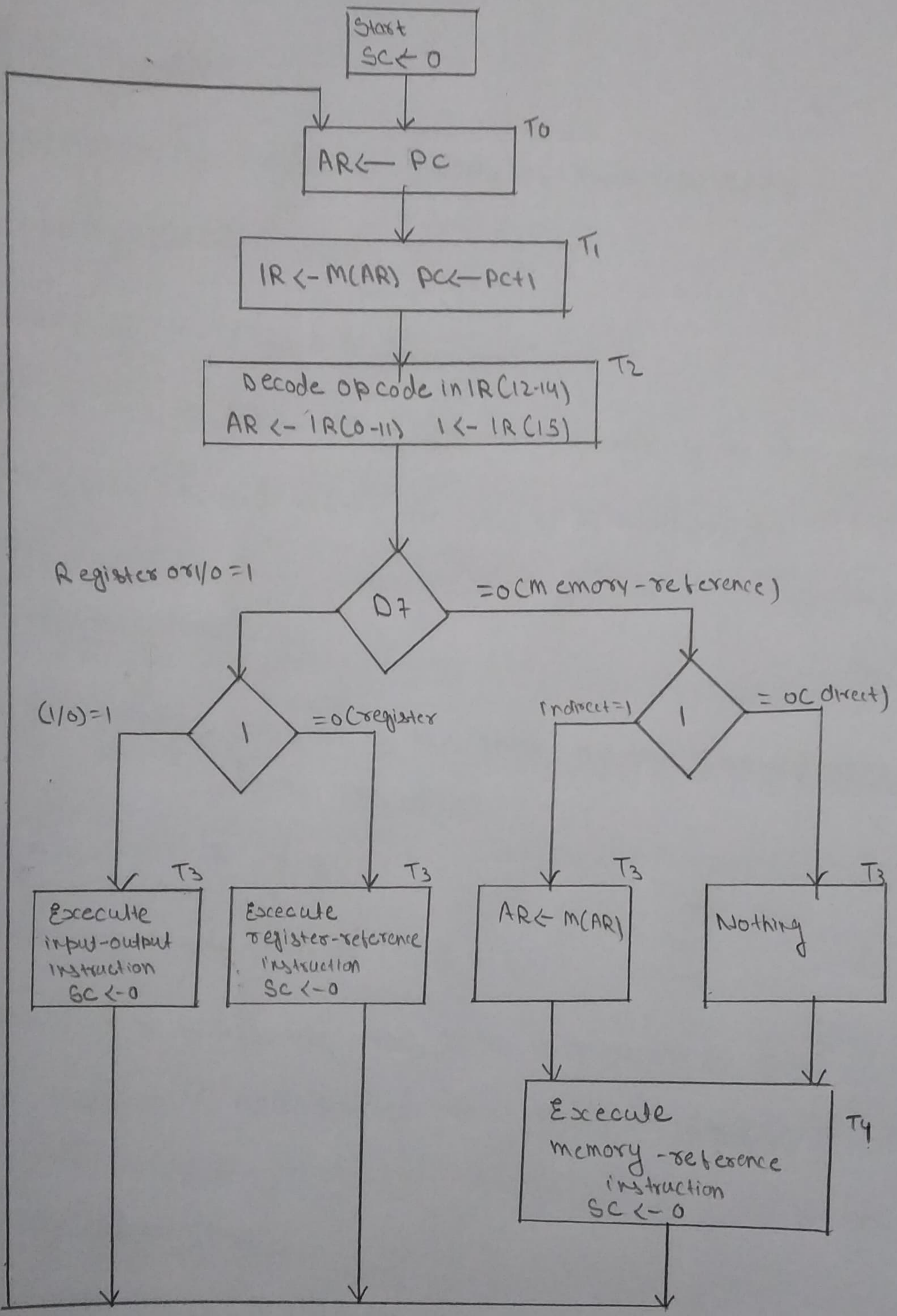
It is distinguished from a normal buffer by having both a normal input and a control input. The control input determines the output state. when the control input is equal to 1, the output is enabled and the gate behaves like any conventional buffer; with the output equal to the normal input. when the control input is 0, the output is disabled and the gates goes to a high-impedance state, regardless of the value in the normal input. Because of this feature, a large number of three-state gate output can be connected with wires to form a common bus line without endangering loading effects.



The Construction of a bus system with three state buffers is demonstrated in above figure. The outputs of four buffers are connected together to form a single bus line. The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line. No more than one buffer may be in the active state at any given time.

One way to ensure that no more than one contool input is active at any given time is to use a decoder, as shown in the diagram when the enable input of the decoder is 0, all of its foour output are 0.

4 Draw a flowchart for instruction cycle.



Start SC ← 0
AR ← PC
T0
IR ← M(AR) PC ← PC+1
T1
Decode op code in IR (12-14) AR ← IR (0-11) I ← IR (15)
T2
Register or I/O = 1
D7
= 0 (memory-reference)
(I/O) = 1
I
= 0 (register
Indirect = 1
I
= 0 (direct)
T3
Execute input-output instruction SC ← 0
T3
Execute register-reference instruction SC ← 0
T3
AR ← M(AR)
T3
Nothing
Execute memory-reference instruction SC ← 0
T4

5 Write Arithemetic and Logic operations.

Arithemetic operations

Addition (ADD)

Discription ⇒ Adds the values of two operands

example ⇒ 'A = A+B'

Assembly ⇒ 'ADD A, B'

operation ⇒ Adds the value in register B to the value in register A, and stores the result in register A.

Substraction (SUB)

Discription ⇒ Substracts the value of the second operand from the first.

Example ⇒ 'A = A-B'

Assembly instruction ⇒ 'SUB A, B'

operation ⇒ Substracts the value in register B from the value in register A, and Stores the result in register A

multiplication (MUL)

Discription ⇒ multiplies two operands

example ⇒ 'A = A*B'

Assembly instruction ⇒ 'MUL A, B'

operation ⇒ multiplies the value in register A by the value in register B. and stores the result in register A.

Division (DIV)

Description ⇒ Divides the first operand by the second

example ⇒ 'A = A/B'

Assembly instruction ⇒ 'DIV A.B'

operation ⇒ Divides the value in register A by the value in register B. and stores the result in register A.

Increment (INC)

Description ⇒ increase the value of an operand by one

example ⇒ 'A = A+1'

Assembly instruction ⇒ 'INC A'

operation ⇒ Adds 1 to the value in register A

Decrement (DEC)

Description ⇒ Decrease the value of an operand by one.

example ⇒ 'A = A-1'

Assembly instruction ⇒ 'DEC A'

operation ⇒ substract 1 from the value in register A.

Logic operations

AND

Description ⇒ performs a bitwise AND operation on two operands

example ⇒ 'A = A ANB'

Assembly instruction ⇒ 'AND A, B'

operation ⇒ performs a bitwise AND between the values in registers A and B. and stores the result in register A

OR

Description ⇒ performs a bitwise OR operation on two operands

example ⇒ 'A = A OR B'

Assembly instruction ⇒ 'OR A, B'

operation ⇒ performs a bitwise OR between the value in registers A and B and stores the result in register A

NOT

Description ⇒ performs a bitwise NOT operation on an operand

example ⇒ 'A=NOT A'

assembly instruction ⇒ 'NOT A'

operation ⇒ Inverts all bits in the value of register A