Data Representation:

Datatypes:

→ The datatypes found in the registers of digital computers may be classified as

(1) Numbers used in arithmetic Computation

(2) Letters of the alphabet used in data processing

(3) Other discrete symbols used for specific purposes

→ We represent each data in binary form in the registers

Number System:

radix:

→ A number System of base or radix r is a System that uses distinct symbol for r digits. To determine the quantity ~~of~~ the number ~~represents~~, It is necesary to multiply each digit by an integer power of r and then form the sum of all weighted digits.

Decimal:

→ The Decimal System number uses the radix 10 system.

→ The 10 Symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

→ The digit 824.5 is represented as.

$$8 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$$

which indicates that 8 hundreds plus 2 tens plus 4 units plus 5 tenths.

Binary:

→ The binary number system uses the radix 2.

The two digits will be 0 and 1.

→ The string of digits 101101 is interpreted as

$$1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 = 45$$

→ To distinguish between different radix numbers the digit will be enclosed in parentheris and the radix of the number inserted as a subscript.

Ex: $(101101)_2 = (45)_{10}$

octal & hexadecimal:

→ The octal (radix 8) and hexadecimal (radix 16)

The eight symbols are 0, 1, 2, 3, 4, 5, 6, 7. The sixteen symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

→ The symbols, A, B, C, D, E, F corresponds to the decimal numbers 10, 11, 12, 13, 14, 15 respectively.

→ The number in radix 'r' can be converted to the familar decimal system by forming the sum of the weighted digits.

Eg: octal 736.4 to decimal

$$(736.4)_8 = 7\times8^2 + 3\times8^1 + 6\times8^0 + 4\times8^{-1}$$

$$= 7\times64 + 3\times8 + 6\times1 + \frac{4}{8} = (478.5)_{10}$$

The decimal number of hexadecimal F3 is

$$(F3)_{16} = F\times16^1 + 3\times16^0 = (15\times16) + 3 = (243)_{10}$$

Convertion:

→ The Conversion of decimal 41.6875 into binary is done by first seperating the number into its integral part 41 and fraction part .6875.

→ The integral part is converted by dividing 41 by r=2 to an integer quotient 20 and a remainder 1.

→ The quotient is again divided by 2 to give a new quotient and remainder.

→ This process is repeated until the integer quotient becomes 0.

→ The first remainder will give the low-order bit of the converted binary number.

2 | 41
2 | 20 - 1
2 | 10 - 0
2 | 5 - 0
2 | 2 - 1
2 | 1 - 0
0 - 1

$(41)_{10} = (101001)_2$

→ The fraction part is converted by multiplying it by 2 to give an integer and a fraction.

→ The new fraction is again multiplied by 2 to give a new integer and a new fraction.

→ This process is repeated until fraction part becomes zero

→ finally two parts are combined to give total required conversion.

Integral part

$2 \mid 41$
$2 \mid 20 - 1$
$2 \mid 10 - 0$
$2 \mid 5 - 0$
$2 \mid 2 - 1$
$1 - 0$

$(41)_{10} = (101001)_2$

fraction part

0.6875

0.6875×2

1.3750×2

0.7500×2

1.5000×2

1.0000

$(0.6875)_{10} = 0.1011$

$\therefore (41.6875)_{10} = (101001.1011)_2$

Octal and hexadecimal numbers:

→ The conversion from and to binary, octal & hexadecimal plays an important role in digital computer.

→ Each octal digits represents to 3 and each hexadecimal digits represents to 4 binary digits.

→ Conversion from binary to hexadecimal is from four digits.

Ex:

A	F	6	3	Hexadecimal
1010	1111	0110	0011	Binary
1 2	7	5	4	3	octal.

Octal Number	Binary Code	Decimal equivalent
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7

Hexadecimal Number	Binary Code	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
:		
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15
14	0001 0100	20
F8	1111 1000	248

Decimal representation:

→ The binary number system is the most natural system for a Computer, but people are experienced to the decimal system.

→ One way to solve this Conflict is to Convert all i/p decimal numbers into a binary number

→ The another alternative bit assignment most commonly used for the decimal digits is the straight binary assignment listed in the following table is known as BCD.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
:	
10	0001 0000
248	0010 0100 1000

→ The only difference b/w a binary number and BCD is that in BCD each decimal digit is represented seperately by a four bits.

Complements:

→ Complements are used in digital Computers for simplying the subtraction operation and for logical manipulation

→ There are two types of Complements ① r's Complement ② (r-1)'s Complement when the values of base r is substituted, the two types are refferred → ~~The r's complement as~~ r's and 2's Complement for binay numbers and 10's & 9's Complement for decimal numbers.

(r-1)'s Complement:

9's complement Given a number N in base r having 'n' digits, the (r-1)'s Complement of N is defined as $(r^n - 1) - N$.

Sol: $r = 10 \quad \therefore r-1 = 9$

So the 9's Complemt is $(10^n - 1) - N$. Now 10^n represents a number that Consist of a Single 1 followed by n 0's. $10^n - 1$ is number represented by n 9's

Eg: with $n = 4$ we have $10^4 - 1 = 9999$

→ The 9's Complement is then obtained by subtracting each digit from 9.

Ex:- 9's Complement of 546700, 12389

$999999 - 546700 = 453299$

$99999 - 12389 = 87610$

1's complement:

for binary numbers $r=2$ & $r-1=1$ so the 1's complement of N is $(2^n-1) - N$

Ex: $n=4$, $(2^4-1) = 15 = (1111)_2$

The 1's complement of a binary number is formed by changing 1's into 0's and 0's into 1's.

Ex:- The 1's Complement of 1011001 is 0100110

→ The $(r-1)$'s Complement of octal or hexadecimal numbers are obtained by subtracting each digit from 7 and F (decimal 15) respectively.

r's Complement:

10's complement:

→ The r's Complement of a 'n' digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and 0 for $N=0$.

→ Thus the 10's Complement of the decimal 2389 is $(10000 - 2389) = 7611$

→ r's Complement is obtained by adding 1 to (r-1)'s complement Since $r^n - N = [(r^n-1) - N] + 1$

The 2's complement of binary 101100 is

$$010011 + 1 = 010100$$

Subtraction of unsigned numbers:

→ The direct method of subtraction uses the borrowing concept

→ The subtraction of two n digit unsigned numbers $M - N$ $(N \neq 0)$ in base r can be done as follows

1. Add the minuend M to the r's complement of the subtrahend N. This performs $M + (r^n - N) = M - N + r^n$
2. If $M \geq N$ the sum will produce an end carry r^n which is discarded and what is left is the result $M - N$
3. If $M < N$, the sum doesn't produce an end carry and is equal to $r^n - (N - M)$, which is the r's complemt of $(N - M)$

Ex: $72532 - 13250 = 59282$

The 10's Complement of 13250 is

$$99999 - 13250 = 86749 + 1 = 86750$$

$M = 72532$

10's complement of $N = 86750$

Sum = 159282

Discard the end Carry 1 and Answer = 59282

Ex M & N is M = 13250

N = 72532

produces negative 59282.

M = 13250

10's complent of N = 27468

Sum = 40718

There is no end carry

∴ Answer is negative 59282

which is 10's complement of 40718

10's Complent of 72532

99999 − 72532

= 27467 + 1

= 27468

10's complent of 40718

99999

(−) 40718

59281

(+) 1

59282

Binary Numbers:

① X = 1010100 Y = 1000011

X = 1010100

2's complent of Y = 0111101

10010001

Discard the end carry

∴ Answer is 0010001

2's complent of 1000011

0111100

+ 1

0111101

② X = 1000011

Y = 1010100

2's complent of Y = 0101100

There is no end carry

∴ Answer is negative 0010001 which is

2's complent of 1101111

1101111

0010000

1

0010001

2's complent of 1010100

0101011

+ 1

0101100

Fixed Point representation:

→ There are two ways, we are going to represent the position of the binary point in a register.

① fixed position

② floating point representation

→ Fixed point method assumes that the binary point is always fixed in one position.

→ There are two positions most widely used. They are

① A binary point in the extreme left of the register to make the stored number a fraction

② A binary point in the extreme right of the register to make the stored number as an integer.

Integer representation:

→ when an integer binary number is positive the sign is represented by 0 and the magnitude by a positive binary number.

→ when an integer binary number is negative the sign is represented by 1, but the rest of the number may be represented in one of three possible ways.

1. Signed magnitude representation (SMR)

2. Signed 1's Complement "

3. Signed 2's Complement "

→ The signed magnitude representation of a negative numbe
Consist of the magnitude and a negative sign. Remaing
2, we are going to perform either 1's (or) 2's complemt

Ex: +14 = 0000 1110 — 8 bit register.

Leftmost bits are zero in 8 bit register.

-14 can be represented in 3 ways.

→ The signed magnitude representation of -14 is obtained from +14 by Complementing only the sign bit.

Ex: -14 = 1000 1110 — (SMR)

→ The signed 1's Complemt representation of -14 is obtained by Complementig all the bits of +14, including the sign bit

Ex: -14 ~~—~~ 1111 0001 (Signed 1's Complent

→ The signed 2's Complemt is obtained by takin the 2's Complement of positive number including its sign bit

Ex -14 = 1111 0001 = 1111 0010

(+) 1

1111 0010 (Signed 2's Complement)

Arithmetic addition:

→ The addition of two numbers in the Signed Magnitude representation follows the rules of ordinary arithmetic.

→ If the sign are the same we add the two magnitudes and give the Sum the Common sign

→ If the signs are different, we subtract the smaller magnitude from the larger magnitude and give result the sign of the larger.

+6	00000110	−6	11111010 (2's comp)
+13	00001101	+13	00001101
+19	00010011	+7	1) 00000111

→ In Contrast 2's complement addition is stated as follows.

→ Add the two numbers, including their sign bits and discard the Carry out of the sign bit position.

→ The negative numbers must initially be in 2's complement and that if the sum obtained after the addition is negative, it is in 2's complement form.

+6	00000110	−6	11111010 (2's)
−13	11110011 (2's comp)	−13	11110011 (2's)
−7	11111001 (2's comp)	−19	11101101
			(2's comp)

Arithmetic Subtraction:

→ Subtraction of two signed binary numbers when negative numbers are in 2's Complement form is very simple and can be stated as follows:

→ Take the 2's complement of the subtrahend (including sign bit) and add it to the minuend (including sign bit). A carry out of sign bit is discarded

→ A subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed

→ This can be demonstrated through the relationship.

$$(\pm A) - (+B) = (\pm A) + (-B)$$

$$(\pm A) - (-B) = (\pm A) + (+B)$$

→ But changing a positive number to a negative number is easily done by taking its 2's complement. The reverse is also true.

Eg: $(-6) - (-13) = +7$

11111010 — 11110011

The subtraction is changed to addition by taking the 2's complement of the subtrahend (−13) to give (+13)

ie 11111010 + 00001101 = 100000111

By removing the end carry we obtain the correct answer 00000111 (+7)

Overflow:

→ When two numbers of n digits are added and the sum occupies n+1 digits, we say that an overflow occured.

→ An overflow is a problem in digital Computer because width of the registers is finite.

→ The detection of an overflow after addition of two binary numbers depends on sign of the number ie whether it is signed or unsigned number.

→ When two unsigned numbers are added, an overflow is detected from the end carry.

→ In case of signed number the left most bit always represents the sign and negative numbers are in 2's complement form.

→ When two signed numbers are added, the sign bit is treated as part of number and end carry doesn't indicate an overflow.

→ An overflow doesn't occur if one number is positive and another is negative

Eg:

Carries:	0 1		Carries:	1 0	
+70	0 1 0 0 0 1 1 0		−70	1 0 1 1 1 0 1 0	
+80	0 1 0 1 0 0 0 0		−80	1 0 1 1 0 0 0 0	
+150	1 0 0 1 0 1 1 0		−150	0 1 [illegible] 0 1 0	

→ Note that 8 bit result should have been positive but has a negative sign bit in first calculation and 8 bit result should have been negative but

→ however the carry out of sign bit position is taken as sign bit of the result, the 9 bit answer so obtained will be correct.

→ Since the answer cannot be accomodated within 8 bits we say that an overflow occured.

Overflow detection:

→ The overflow can be detected by observing the carry into the sign bit and carry out of the sign bit position.

→ If these two carries are not equal, an overflow condition produced.

→ If these two carries are applied to exclusive OR gate, an overflow will be detected, when o/p of the gate is equal to '1'

Decimal fixed point representation:

→ The representation of decimal number in a register is a function of the binary code used to represent a decimal digit.

→ A 4 bit decimal code requires four flipflops for each decimal digit. in binary code.

→ The representation of 4385 in BCD requires 16 flipflops ie four flipflops for each digits

0100 0011 1000 0101

→ By representing numbers in decimal we are wasting a considerable amount of storage space since number bits required to store a decimal number in binary code is greater than the number of bits needed for its equivalent binary representation.

→ However the advantage in using decimal representation is computer i/p and o/p data are generated by people who use only the decimal system

→

Floating point representation:

→ The floating point representation of a number has two parts

① first part represents a signed fixed point called mantissa

② The second part designates the position of the decimal point and is called the exponent.

→ The fixed point mantissa may be a fraction or an integer

Eg. The decimal number +6132.789

fraction	Exponent
+0.6132789	+04

→ The value of the exponent indicates that the actual position of the decimal point is four position right of the indicated decimal point in the fraction

→ The equivalent value is represented as $+0.6132789 \times 10^{+}$

→ The floating point is represented as

$$m \times r^{e}$$

Only the ~~mantissa~~ mantissa m and the exponent e are physically represented in the register (including sign)

→ A floating point binary number is represented in a similar manner except that it used base 2.

Eg: Binary number $+1001.11$ is represented with 8 bit fraction and 6 bit exponent as follows

fraction	exponent
01001110	000100

→ The fraction has a 0 in the leftmost position to denote positive. The decimal point of the fraction follows the sign bit but is not shown in the register.

→ The exponent has the equivalent binary number for +4

→ The floating point number is equivalent to

$$m \times 2^{e} = +(.1001110)_2 \times 2^{+4}$$

Normalization:

→ A floating point number is said to be normalized if the MSB of the mantissa is non zero.

Eg: the decimal number 350 is normalized but 00035 cannot be normalized.

→ Regardless of where the position of the radix point is assumed to be in the mantissa, the number is normalized only if its leftmost digit is nonzero.

Eg: 8 bit binary number 00011010 is not normalized because the MSB is zero.

→ The floating point numbers are more complicated than the fixed point numbers and their execution takes longer time and requires more complex H/w.

Error Detection Codes:

→ Binary information are transmitted through some communication medium which is subject to external noise that could change bits from 1 to 0 and vice versa.

→ In most practical system, there is always a finite probability of occurance of a errors

→ Two types of codes like error detection and error correction codes are used to detect and correct the errors

→ An error detection code is a binary code that detects digital errors during transmission. The detected errors cannot be corrected but their presence is indicated.

→ An error correcting code is a binary code that detects and corrects the bit errors occured during transmission

Parity bit:

→ The most common error detection code used is the parity bit.

→ A parity bit is an extra bit included with the binary message to make the total number of 1's either odd or even

→ The message of 3 bits and two possible parity bit is shown below

Message X Y Z	P(odd)	P(even)
0 0 0	1	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	0	1

→ P(odd) bit is choosen in such a way as to make sum of 1's (in all four bits) odd.

→ P(even) bit is choosen in such a way as to make sum of 1's even.

→ The even parity scheme has the disadvantage of having bit combination of all 0's while in the odd parity there is always one bit ie 1.

→ The p(odd) is a complement of p(even).

Parity generator:

→ During the transfer of information from one location to another, the parity bit is handled as follows.

→ At the sending end the message (in case 3 bits) is applied a parity generator, where the required parity bit is generated.

→ The message including the parity bit is transmitted to its destination.

Parity checker:

→ At the receiving end all the incoming bits are applied to parity checker that checks the proper parity bit is transmitted to its destination

→ An error is detected if the checked parity does not conform to the adopted parity.

→ The parity methods detects the presence of one, three or any odd number of errors. An even number of errors is not detected.

odd functions:

→ ① Parity generator and checker networks are logic circuits constructed with XOR and XNOR gates

→ ② An odd function is a logic function whose binary value is 1 iff an odd no. of variable are equal to 1

p(odd) is the complement of p(even), XNOR gate is used to complement

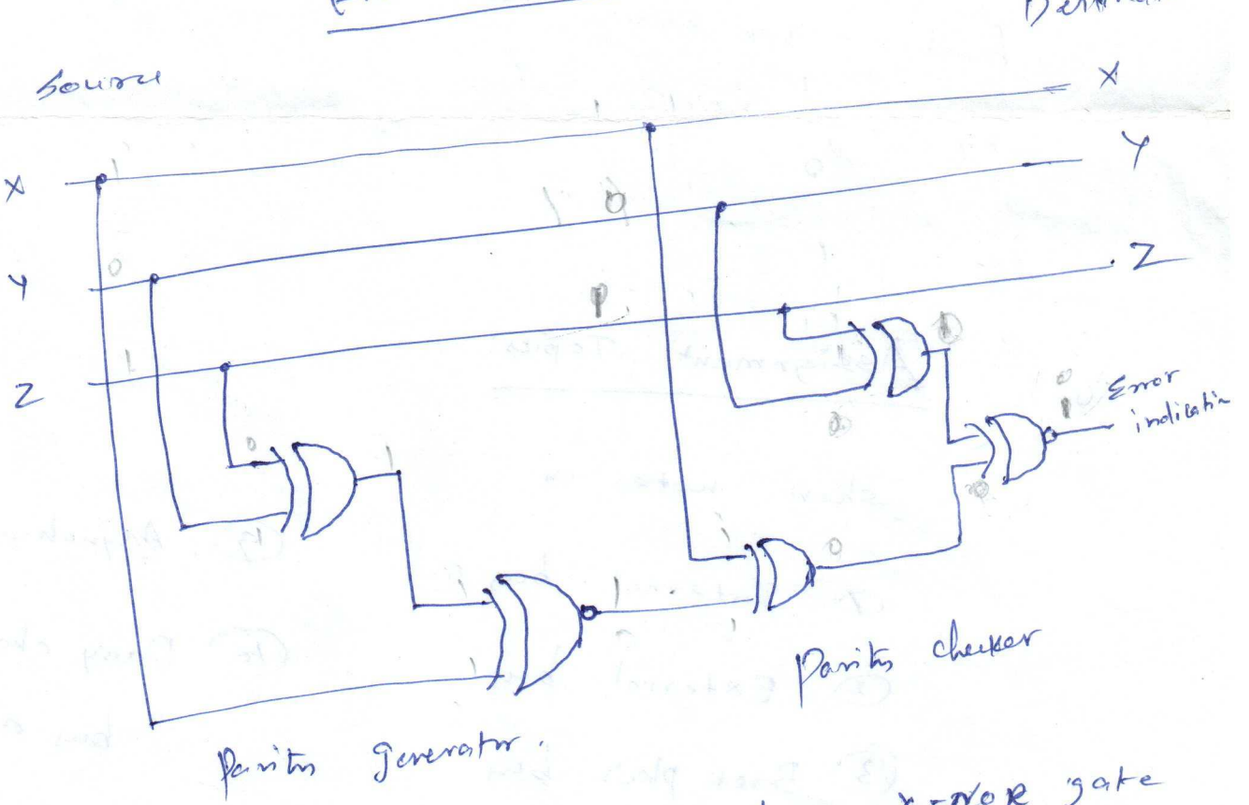
(2)

→ According to this definition P(even) function is the XOR of X, Y, Z, because it is equal to 1 when either one or all three of the variable are equal to 1

Example:

A 3 bit message to be transmitted with an odd parity bit. At the sending end, the odd parity bit is generated by a parity generator circuit which is shown below

Error detection with odd parity



→ This circuit consist of one XOR and one X-NOR gate Since P(even) is the XOR of XYZ and P(odd) is the Complement of P(even)

→ The message and the odd parity bits are transmitted to their destination where they are applied to the parity checkers.

→ An error has occured during transmission if the parity of the four bits received is even since binary information transmitted was originally odd.

→ The o/p of the parity checker would be 1 when an error occurs that is when number of 1's in the four i/p is even.

→ Since XOR function of 4 input is a odd function we again need to complement the o/p by using XNOR gate.

Assignment Topics:

write short notes on

1. Internal bus
2. External bus
3. Back plain bus
4. I/o bus
5. System bus
6. Address bus
7. Data bus
8. Synchronous bus
9. Asynchronous bus
10. Daisy chained based bus arbitration.

Important questions:

(1) a) What are the functional units? Explain each one

b) Explain about different bus in a practical Computer System.

(2) a) What are the different performance measures used to represent a computer system performance.

b) What do you mean by a parity bit? Explain with example how even and odd parity bits are generated. Is it possible to correct the errors using parity bits.

(3) a) Explain about sign magnitude and 2's complement approaches for representing the fixed point numbers.

b) Design a 4 bit odd parity generator and checker. Can parity bit be used for error detection. If so how?

(4) a) Explain about various buses such as internal, external, backplain, I/O, system, address, data, Synchronous and asynchronous

b) Explain about daisy chain bus arbitration.

Reference:

1. **Digital Design, 6th Edition, M. Morris Mano, Pearson Education, 2018.**

Prepared by:

Mr A Srinivasan,

Associate Professor,

Dept of CSE-DS