COMP 1510 PROBLEM SET 01

Dewan Mukto, Student ID : 202004321

1) a) i) To convert 67 (base 10) to a 8-bit sign magnitude binary number (base 2), (‘//’ represents floor division to find the quotients and ‘%’ represents modulus operation to obtain remainders)

67 // 2 = 33, 67 % 2 = 1,
33 // 2 = 16, 33 % 2 = 1,
16 // 2 = 8, 16 % 2 = 0,
8 // 2 = 4, 8 % 2 = 0,
4 // 2 = 2, 4 % 2 = 0,
2 // 2 = 1, 2 % 2 = 0,
1 // 2 = 0, 1 % 2 = 1,

Reversing the order of remainders, we get = 1000011
Since this is a 8-bit sign magnitude number, the sign bit is 0

Thus, 67 (base 10) = **01000011** (base 2, sign magnitude)

ii) To convert -89 (base 10) to a 8-bit sign magnitude binary number (base 2), we repeat the same steps as for (a)(i) above.

Considering only the absolute value = 89,

89 // 2 = 44, 89 % 2 = 1,
44 // 2 = 22, 44 % 2 = 0,
22 // 2 = 11, 22 % 2 = 0,
11 // 2 = 5, 11 % 2 = 1,
5 // 2 = 2, 5 % 2 = 1,
2 // 2 = 1, 2 % 2 = 0,
1 // 2 = 0, 1 % 2 = 1,

Reversing the order of remainders, we get = 1011001
Since this is a 8-bit sign magnitude number, the sign bit is 1

Thus, -89 (base 10) = **11011001** (base 2, sign magnitude)

iii) We know, 67 (base 10) + (-89) (base 10) = -22 (base 10)
Let’s try this out in 8-bit sign magnitude arithmetic :

```
67 (base 10) =          01000011
-89 (base 10) =    +    11011001
                   ______________
                        00011100    = 28 (base 10)
```

Something is definitely wrong since the answer does not match with what we know it should be. This is why sign magnitude is not meant for performing calculations. The correct value could not be obtained.

b) i) Converting 67 (base 10) to 2’s complement (base 2) :

67 // 2 = 33, 67 % 2 = 1,
33 // 2 = 16, 33 % 2 = 1,
16 // 2 = 8, 16 % 2 = 0,
8 // 2 = 4, 8 % 2 = 0,
4 // 2 = 2, 4 % 2 = 0,
2 // 2 = 1, 2 % 2 = 0,
1 // 2 = 0, 1 % 2 = 1,

Reversing the order of remainders,
= 1000011

Filling up the remaining bits in front with 0,
= **01000011** (base 2, 2’s complement)

ii) Converting -89 (base 10) to 2’s complement (base 2) :

Once again, considering only the absolute value = 89,

89 // 2 = 44, 89 % 2 = 1,
44 // 2 = 22, 44 % 2 = 0,
22 // 2 = 11, 22 % 2 = 0,
11 // 2 = 5, 11 % 2 = 1,
5 // 2 = 2, 5 % 2 = 1,
2 // 2 = 1, 2 % 2 = 0,
1 // 2 = 0, 1 % 2 = 1,

Reversing the order of remainders,
= 1011001

Filling up the remaining bits in front with 0,
= 01011001

Some further steps since this is a negative number :

Flipping all the bits,
= 10100110

Adding 1 to it,
= 10100110 + 00000001
= **10100111** (base 2, 2’s complement)

iii) Reattempting the arithmetic calculation from (a)(iii),

67 (base 10) =		01000011	(base 2, 2’s complement)
-89 (base 10) =	+	10100111	(base 2, 2’s complement)
		11101010	= **-22** (base 10)

Yes, this time the correct value has been obtained!

2) a) Converting -119.627 to 32-bit IEEE 754 representation :

The decimal value is negative, so the sign bit is 1

Converting the absolute value of 119 (base 10) to binary :

119 // 2 = 59, 119 % 2 = 1,
59 // 2 = 29, 59 % 2 = 1,
29 // 2 = 14, 29 % 2 = 1,
14 // 2 = 7, 14 % 2 = 0,
7 // 2 = 3, 7 % 2 = 1,
3 // 2 = 1, 3 % 2 = 1,
1 // 2 = 0, 1 % 2 = 1,

Reversing the order of remainders,

= 1110111 (base 2)

As for the fractional part,

0.627 * 2 = 1.254
0.254 * 2 = 0.508
0.508 * 2 = 1.016
0.016 * 2 = 0.032
0.032 * 2 = 0.064

0.064 * 2 = 0.128
0.128 * 2 = 0.256
0.256 * 2 = 0.512
0.512 * 2 = 1.024
0.024 * 2 = 0.048

0.048 * 2 = 0.096
0.096 * 2 = 0.192
0.192 * 2 = 0.384
0.384 * 2 = 0.768
0.768 * 2 = 1.536

0.536 * 2 = 1.072
0.072 * 2 = 0.144
0.144 * 2 = 0.288
0.288 * 2 = 0.576
0.576 * 2 = 1.152

0.152 * 2 = 0.304
0.304 * 2 = 0.608
0.608 * 2 = 1.216
......

(keeps going on and on – only God knows how further; we only have a maximum of 23 bits to consider for the mantissa, so we can stop here)

Now, putting the integer and decimal portions together,

= 1110111.10100000100000110001001......

In normalized form :

= 1.11011110100000100000110001001..... x 2^6

Thus, the exponent is :

= 127 + 6 = 131 (base 10) = 10000101 (base 2)

The mantissa is :
11011110100000100000110 (cropped off at 23-bits)

Putting everything together,

= 1 10000101 11011110100000100000110

= **11000010111011110100000100000110** (IEEE 754)

Note : -119.627 could not be directly represented by the 32-bit IEEE 754 floating point representation. The value obtained above is equivalent to -119.6269989013671875 (the closest number possible using the IEEE 754 method. The full step-by-step details of conversion from the IEEE 754 value back to the decimal has not been shown since it was not part of the question.)

b) Converting 1 10001101 01010000000000000000000 to decimal,

The sign bit is 1, so the number is negative.

Exponent : 10001101 (base 2) = 141 – 127 = 14 (base 10)
Mantissa : 01010000000000000000000 (base 2) = 2^21 + 2^19 = 2621440 (base 10)

Normalized value = 1.01010000000000000000000 x 2^14

Un-normalized value = 0101010000000000.0000000 (base 2)
Since there is no fractional part for this number, we can ignore it

= 0101 0100 0000 0000 (spacing for easy visualization of the powers of 2 to apply)

Converting to decimal:

= 1 x 2^14 + 1 x 2^12 + 1 x 2^10
= 21504 (base 10)

Thus, the decimal representation of the 32-bit IEEE floating-point value is :

= **-21504.0** (base 10)

3) a) The value of the bias should be 2^(5-1) – 1 = 15

b) Converting 0.7 to a floating point number using this system,

The sign bit is 0, since the number is positive

0.7 * 2 = 1.4
0.4 * 2 = 0.8
0.8 * 2 = 1.6
0.6 * 2 = 1.2
0.2 * 2 = 0.4
0.4 * 2 = 0.8
0.8 * 2 = 1.6
0.6 * 2 = 1.2 (repeats from above)
....

So there is a repeating trailing sequence in the fractional part.

Thus, 0.7 = 0.1011001100... (repeats)

Mantissa (cropped off to 7 bits) = 0110011 (base 2, 7-bit)

Normalizing,

= 1.0110011 x 2^(-1)

So the exponent is = 15 - 1 = 14 (base 10) = 01110 (base 2, 5-bit)

Putting the pieces together,

= 0 01110 0110011
= **0011100110011** (in this floating point system)

Converting to decimal using this system,

= **0.10110011** (un-normalized, binary base-2)

=

$$\frac{1}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{0}{2^5} + \frac{0}{2^6} + \frac{1}{2^7} + \frac{1}{2^8}$$

= **0.69921875** (base 10, most accurate representation of 0.7 possible)

c) The next largest number that can be represented in this system is
= **0011100110111** (floating point)
= **0.10110111** (un-normalized, binary base-2)
=

$$\frac{1}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{0}{2^5} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^8}$$

= **0.71484375** (base 10)

d) The next smallest number that can be represented in this system is
= **0011100110010** (floating point)
= **0.10110010** (un-normalized, binary base-2)
=
$$\frac{1}{2^1}+\frac{0}{2^2}+\frac{1}{2^3}+\frac{1}{2^4}+\frac{0}{2^5}+\frac{0}{2^6}+\frac{1}{2^7}+\frac{0}{2^8}$$

= **0.6953125** (base 10)

e) Checking the differences between the adjacent possible values in that floating point system :

In base 10 :
0.71484375 – 0.69921875 = 0.015625
0.69921875 – 0.6953125 = 0.00390625

In base 2 :
0.10110111 – 0.10110011 = 0.000001 (base 2) = 2^(-6)
0.10110011 – 0.10110010 = 0.00000001 (base 2) = 2^(-8)

Thus, it can be seen that the spacing between the values are not always the same.

4) a)

```
   PROGRAM pendulumPeriod

   IMPLICIT NONE

   ! Variables are declared
   REAL,PARAMETER::pi = 3.141592654
   REAL::g = 9.8, L, T
   CHARACTER::std_val_g

   ! User is prompted for the input value(s)
   WRITE (*, "(a)", ADVANCE="no") "Enter value of L (in metres) : "
   READ (*, *) L
   WRITE (*, "(a)", ADVANCE="no") "Use the value of g = 9.8 m/s^2 ? (Y/N) : "
   READ (*, *) std_val_g

   ! For customized values of 'g'
   IF (std_val_g == 'N' .OR. std_val_g =='n') THEN
      WRITE (*, "(a)", ADVANCE="no") "Enter value of g (in metres per second squared) : "
      READ (*, *) g
   END IF

   ! Equation is used to perform the calculation
   T = 2 * pi * ( sqrt( L / g ))

   ! The result is output
   PRINT *, "The value of T is :", T, " seconds."

END PROGRAM pendulumPeriod
```

b)

```fortran
PROGRAM totalEnergy

IMPLICIT NONE

! Variables are declared
REAL::g = 9.8, m, v, h, total_energy
CHARACTER::std_val_g

! User is prompted for the input value(s)
WRITE (*, "(a)", ADVANCE="no") "Enter value of m (in kg) : "
READ (*, *) m
WRITE (*, "(a)", ADVANCE="no") "Enter value of v (in metres per second) : "
READ (*, *) v
WRITE (*, "(a)", ADVANCE="no") "Enter value of h (in metres) : "
READ (*, *) h
WRITE (*, "(a)", ADVANCE="no") "Use the value of g = 9.8 m/s^2 ? (Y/N) : "
READ (*, *) std_val_g

! For customized values of 'g'
IF (std_val_g == 'N' .OR. std_val_g =='n') THEN
   WRITE (*, "(a)", ADVANCE="no") "Enter value of g (in metres per second squared) : "
   READ (*, *) g
END IF

! Conditional statements to decide when to apply which formula
IF (v == 0) THEN
   ! For objects at rest, calculate only potential energy as total energy
   total_energy = m * g * h
ELSE IF (h == 0) THEN
   ! For objects at ground level, calculate only kinetic energy as total energy
   total_energy = ( m * v**2 ) / 2
ELSE
   ! For objects somewhere in between, calculate the sum of both energies
   total_energy =  (m * g * h) + (( m * v**2 ) / 2)
END IF

! The result is output
PRINT *, "The total energy of the object is :", total_energy, " joules."

END PROGRAM totalEnergy
```