Computer Organization and Assembly Language

[Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.]

Question 1

A.

```
function strlen(char string[])
{
local int index= 0;
begin_loop:
if(string[index] = 0)        goto: end_loop;
++index;
Goto begin_loop;
End_loop;
}
```

B.

```
strlen:
# int i = 0
        addi    $sp, $sp, -88    # Set Stack pointer to the beginning of string
        lw      $a0, 0($sp)      # Load string address in $a0
#       sw      $zero, 0($sp)    # Initialize i in stack frame

begin_loop:
# if (string[i] == '\0')  ➔ goto end_loop
        lw      $t0, 0($s0)      # $t0 = i.
        add     $t0, $a0, $t0    # $t0 = string + i = &string [ i ]
        lbu     $t1, 0($t0)      # $a0[7:0] = string [ i ]
        beq     $t1, $zero, end_loop
```

```
# i++
        lw      $t0, 0($sp)     # $t0 = i
        addi    $t0, $t0, 1     # $t0 = i + 1
        sw      $t0, 0($sp)     # ++i
# goto begn_loop
        j       begin_loop      # Continue the loop
# end the loop
end_loop:
        addi    $sp, $sp, 88    # Deallocate stack frame
        mov     $v0,$t0         # Moving length in $v0
        jr      $ra             # Return
```

Question 2

Version 2:

```
function write_grid(grid,plain)
        local int col,index = 0, len = strlen(plain -1);
        int row = 0,col = 0;
        while row <= NUM_ROWS  do
                while col <= NUM_COLS     do
                        if index < len
                                grid[row][col] = plain[index]
                                ++index
                        else
                                grid[row][col] = ‘’;
                        end if
                end while
        end while
end function
```

Version 3

```
function write_grid(grid,plain)
        local int col,index = 0, len = strlen(plain -1);
        int row = 0,col = 0;



loop1:
                if row > NUM_ROWS goto end_loop1
loop2:
                if col > NUM_COLS    goto end_loop2
                        if index < len
                                grid[row][col] = plain[index]
                                ++index
                        else
                                grid[row][col] = ‘’;
                        end if
                        ++column
                        goto loop2
end_loop2:
                        ++rows
                        goto loop1
end_loop1:
end function
```

Version 4

```
function write_grid(grid,plain)
        local int col,index = 0, len = strlen(plain -1);
        int row = 0,col = 0;
loop1:
        if row > NUM_ROWS goto end_loop1
loop2:
                if  col > NUM_COLS   goto end_loop2
                        if index < len   goto skip
                        grid[row][col] = ‘’;

                        goto end_ifelse

skip:
                        grid[row][col] = plain[index]
                        ++index
end_ifelse:

                        end if
                        ++col
                        goto loop2
end_loop2:
                        ++row
                        goto loop1
end_loop1:

end function
```

Version 5:

```
        addi    $sp, $sp, -12           # Allocate 3 words in stack frame

# Loading Index, Column and row locations in $sp
        sw      $zero, 4($sp)           # index ← 0
        sw      $zero, 0($sp)           # col ← 0
        sw      $zero, 8($sp)           # row ← 0
        addi    $t7, $zero, ' '         # $t7 ← ' ' (ASCII value 32)

        li      $t9, 8          # NUM_ROWS = 8
        li      $t8, 10         # NUM_COLS  = 10

loop1:
        lw      $t0, 8($sp)                     # $t0 ← row
        beq     $t0, $t9, end_loop1

loop2:
        lw      $t0, 0($sp)                     # $t0 ← col
        bgt     $t0, $t8, end_loop2
startif:
        lw      $t0, 4($sp)                     # $t0 ← index
        blt     $t0, $v0, skip                  # length is stored in $v0

# grid[row][col] = plain[index]
        lw      $t0, 8($sp)             # $t0 ← row
        mul     $t0, $t0, $t8           # $t0 ← row · NUM_COLS
        lw      $t1, 0($sp)             # $t1 ← col
        add     $t0, $t0, $t1           # $t0 ← row · NUM_COLS + col
        add     $t0, $a0, $t0           # $t0 ← grid + row · NUM_COLS + col = &grid[row][col]
        sw      $t7, ($t0)                # Store the ASCII of ‘ ‘ at the grid [row][col]
```

```
        j       end_ifelse
skip:
# grid[row][col] = plain[index]
        lw      $t0, 8($sp)             # $t0 ← row
        mul     $t0, $t0, $t8           # $t0 ← row · NUM_COLS
        lw      $t1, 0($sp)             # $t1 ← col
        add     $t0, $t0, $t1           # $t0 ← row · NUM_COLS + col
        add     $t0, $a0, $t0           # $t0 ← grid + row · NUM_COLS + col = &grid[row][col]
        lw      $t5, 4($sp)               # Copying next word (plain[index]) from plain array
        sw      $t5, ($t0)                # Store the ASCII plain[index] to grid[row][col]


# ++index
        lw      $t0, 4($sp)             # $t0 ← index
        addi    $t0, $t0, 1             # $t0 ← index + 1
        sw      $t0, 4($sp)             # ++index
end_ifelse:

        lw      $t0, 0($sp)             # $t0 ← col
        addi    $t0, $t0, 1             # $t0 ← col + 1
        sw      $t0, 0($sp)             # ++col
        j       loop2
end_loop2:

        lw      $t0, 8($sp)             # $t0 ← row
        addi    $t0, $t0, 1             # $t0 ← row + 1
        sw      $t0, 8($sp)             # ++row
        j       loop1
end_loop1:
```

Question 3

1. What is the n in decimal fixed point form (ddd.ddddd)?

   -345.43210 x 10^{17}

2. What is n in binary fixed point form (bbb.bbbb), storing the first 25 bits following the binary point.

   11101111101100010001000 1. 0101001000101101000011100

3. What is the normalized binary number written in the form 1.bbb....bb x 2c.

   1. 11011111011000100010001 x 2^{25}

4. What are the 23 mantissa bits after the bits in bit position -24, -25, ... are eliminated.

   Mantissa bits are 11011111011000100010001 .

5. What is the biased exponent in decimal and in binary?

   The bias exponent in decimal is 191 and in binary is 10111111

6. Write 32-bits in order of s e m.

   1 10111111 11011111011000100010001

7. What is the final answer in 8-bit hexa-decimal.
   DFEFB111

Question 4

1. What is N in binary.
   1111 0100 1110 0011 . 1100 0010 1101 0001

2. What is value of the sign bit? What does it signify?
   The value of sign bit is 1. It signifies that the final number is negative.

3. What are the binary and decimal values of the biased exponent?
   The bias exponent in decimal is 233 and in decimal is 111 0100 1.

4. What is the mantissa of the binary?
   The mantissa of the binary is 110 0011 1100 0010 1101 0001

5. What is the decimal value of unbiased exponent?
   The decimal value of unbiased exponent is 233-127 = 106.

6. What is the decimal value of mantissa with leading 1. part?
   The decimal part of the mantissa is 6537937.

7. What is the final decimal real number?
   The final decimal real number is -1.4436069E32

Question 5

1. What is n in decimal fixed point form (ddd.ddddd)?
   Number N in decimal fixed point form is 182.75092 e-17

2. What is n in binary fixed point form (bbb.bbbb), storing first 110 bits.
   0.00000000000000000000000000000000000000000000000000010000011101011111001011111111001110100110001100110111000000001

3. What is the normalized binary number, written in the form 1.bbbbb...bbb × 2e, storing 54 bits following the binary point)
   1. 00000111010111110010111111111100111010011000110011100 x 2e-49

4. What are the 52 mantissa bits, after the bits in bit positions -53, -54, ... are eliminated using the round to nearest, ties to even mode?
   0000011101011111001011111111110011101001100011001100

5. What is the biased exponent in decimal and in binary?
   Biased exponent is 01111001110 in binary and 974 in decimal.

6. Write the 64-bits of the number in the order: s e m?

   0 01111001110 0000011101011111001011111111100111010011000110011100

7. Write the final number as a 16-Hexadigit number.
   3CE075F2FF9D319C