Answers

Q01) Convert 21_4 to base 10

$21_4 = 1 * 4^0 + 2 * 4^1 = 9$

Q02) Assume we are dealing with 8-bit binary numbers for this question. Complete the following table. The values here are represented in signed two's complement format;

Decimal	Binary	Hexadecimal
31	0001 1111	1F
55	0011 0111	37
-11	1111 0101	F5
9	0000 1001	09
-22	1110 1010	EA

Q03) The following addition and subtraction operations are to be carried out with 4-bit two's complement number. For each operation, calculate the result and label each as OVERFLOW, or CORRECT

3 + 5

```
 3 is  0011
 5 is  0101
       1000      (OVERFLOW)
```

6 - 1

```
 6 is  0110
-1 is  1111
       0101      (CORRECT)
```

-3 - 3

```
-3 is  1101
-3 is  1101
       1010      (CORRECT)
```

Answers

Q04) State the two problems with the 'sign magnitude' representation of signed numbers.

1) Two zeros. 2) Can't do arithmetic (add)

Q05) What's the range (min to max) of numbers you can represent with 5 bits using;

1) unsigned numbers 0 to 31

2) signed numbers — one's complement -15 to 15

3) signed numbers — two's complement -16 to 15

Q06) What's the output of the following C Code — assume size of int is 8 bytes, and address of x is 80 (decimal)?

```
int main()
{
        int x[] = {3, 5, 1, 15, 10, 30};
        int *p = x;
        printf("%p, %d", p + 2, *(p + 4));  //You can show address in decimal
        return 0;
}                    96, 10
```

Q07) What's the value of register $s0, $s1, and $s2 after executing this MIPS code;

```
        .data
                myVariable: .word 25
        .text
                addi $t0, $zero, 7
                sw $t0, myVariable($zero)
                lw $s0, myVariable($zero)
                add $s1, $s0, $t0
                sub $s2, $s0, $s1
```

$s0 = 7

$s1 = 14

$s2 = -7

Q08) True/False;

1. MIPS belong to the RISC category of CPU architectures

   True

2. Assembly languages are architecture-independent; you can use the same assembly code for any CPU architecture

   False

3. In MIPS32 (the MIPS we use), one word is 8 bytes

   False

Q09) How can you write the following line of C code in MIPS;

s0 = s1 + s2 + s3 - s4;

Answer to Q09 goes here:

```
add $s0, $s1, $s2
add $s0, $s0, $s3
sub $s0, $s0, $s4
```

Q10) Convert the following C to MIPS;

```
int myArray[20];
void main()
{
        myArray[0] = 20;
        myArray[5] = 15;
        myArray[10] = 10;
        // use syscall to end program
}
```

Answer to Q10 goes here:

```
.data
    myArray: .space 80
.text
    addi $t0, $zero, 20
    sw $t0, myArray($zero)

    addi $t0, $zero, 15
    addi $t1, $zero, 20
    sw $t0, myArray($t1)

    addi $t0, $zero, 10
    addi $t1, $zero, 40
    sw $t0, myArray($t1)

    addi $v0, $zero, 10
    syscall
```

Answers

Q11) Convert the following C to MIPS;

```
int globalValue = 9;

void main()
{
        globalValue =  subtractFromGlobal(5);
        // use syscall to end program
}

int subtractFromGlobal(int num)
{
        return globalValue + num;
}
```

```
.data
        globalValue: .word 9
.text
        main:
                addi $a0, $zero, 5              # Placing argument in $a0
                jal subtractFromGlobal          # Calling function
                sw $v0, globalArray($zero)# globalValue = returned value
                addi $v0, $zero, 10             # These two lines will quit
                syscall                         # the program

        subtractFromGlobal:
                lw $t0, globalValue($zero)  # Get the value of globalValue
                add $v0, $t0, $a0               # Add it to $a0; put result in $v0
                jr $ra                          # Go back to the caller (main)
```

Q12) Convert the following C to MIPS;

```
void main()
{
        int myArray[50];
        int i = 0;
        while (i < 50)
        {
                myArray[i] = 5 + i;
                i = i + 1;
        }
        // use syscall to end program
}
```

```
.text
        main:
                addi $sp, $sp, -200  # Allocate space for array
                addi $s0, $zero, 0   # i = 0
                BEGIN:               # beginning of our loop
                slti $t0, $s0, 50
                beq $t0, $zero, AFTER
                sll $t1, $s0, 2      # Calculating offset
                add $t2, $sp, $t1    # Adding offset to array address
                addi $t3, $s0, 5     # Calculating value
                sw $t3, 0($t2)       # Changing array on stack
                addi $s0, $s0, 1     # i = i + 1
                j BEGIN
                AFTER:
                addi $v0, $zero, 10
                syscall
```

Answers

Q13) What's the output of the following code;

```
void main()
{
        int var_1 = -15.75760643445; // size of int is 4 bytes
        printf("Value is: %x", var_1);
}                                          ff ff ff f1
```

Q14) Given that the array of ints 'my_array' contains the following elements [10, 15, 19, 11, 17]; also, assume that the register $s0 points to (or contains the address of) the first element in the array. Fill the table below (given the code has executed) — Size of int is 4 bytes;

```
addi $s1, $zero, 20
lw $t3, 4($s0)
lw $t4, 8($s0)
add $s2, $t3, $t3
addi $t6, $zero, 18
sub $s6, $zero, $t6
```

Register	$s1	$s2	$s6	$t3	$t4	$t6
Value	20	30	-18	15	19	18

Q15) Given that the array of ints 'my_array' contains the following elements [7, 11, 23, 12, 10]; also, assume that the register $s0 points to (or contains the address of) the first element in the array. List the values of 'my_array' (the entire array) after executing the code below — Size of int is 4 bytes;

```
lw $s4, 8($s0)        # $s4 = 23
lw $s6, 0($s0)        # $s6 = 7
sw $s4, 0($s0)        # arr[0] = 23    Final array: [23, 11, 23, 11, 7]
lw $s3, 4($s0)        # $s3 = 11
sw $s3, 12($s0)       # arr[3] = 11
sw $s6, 16($s0)       # arr[4] = 7
```