Daniyal Khan
3765942

CS-2253

Assignment #1

Question 1:

R-type is used for arithmetic, logic and shift operations.

|opcode (6)| rs (5) | rt (5) | rd (5) | shamt (5) | funct (6)

Opcode is 0 for r-type.rs and rt are the source registers and rd is the destination register. Shamt is used for shift amount. Funct is used to tell ALU which operation to perfom (eg. Add).

Example: add $t0, $t1, $t2

This means: $t0 = $t1 + $t2

I-type is used for instructions that use constants or memory.

|opcode (6)| rs (5) | rt (5) | immediate (16) |

6 bit opcode. rs is the source register and rt is the target register.

Immediate is the constant value or offset.

Example: addi $t0, $t1, 10

This means: $t0 = $t1 + 10

J-type is used to perform unconditional jumps to an address.

| opcode (6) | target address (26) |

6 bit opcode. Target is 26 bit part of address.

Example: j LOOP
This jumps to the label LOOP.

Question 2:

```
li $t0, 10
li $t1, 20

bgt $t0, $t1, ELSE            # if $t0 > $t1 go to ELSE
sll $t2, $t0,1                # t2 = $t0 * 2
j END
ELSE:
sll $t2, $t1, 1               # t2 = $t1 * 2
END:
```

- bgt $t3, $zero, ELSE: Branches to ELSE if the condition is true (i.e., x is greater than y).
- j END: Unconditional jump to skip the else block once the then block runs.

Question 3:

```
li $s0, 0

li $s1, 1

li $t0, 5                          # outer loop limit var

li $t1, 3                          # inner loop limit var


outerloop:

     li $s2, 1
innerLoop:

     mul $s3, $s1, $s2

     add $s0, $s0, $s3


     addi $s2, $s2, 1              # $s2++

ble $s2, $t1, innerLoop


addi $s1, $s1, 1                   # $s1++

ble $s1, $t0, outerLoop
```

In MIPS, nested loops are implemented manually using branching and labels, because there is no built-in for or while loops. We use conditional branch instructions and registers to track loop variables.We use saved registers ($s0-$s7) for variables we want to keep across function calls, and temporary registers ($t0-$t9) for short-term values.

Question 4:

```
li $s0, 0                    # sum = 0
li $t0, 0                    # loop var i
li $t2, 5

loop:
mul $t1, $t0, 4              # i * 4
add $t3, $s3, $t1            # base + offset
lw $t4, 0($t3)
add $s0, $s0, $t4            # $s0 = $s0 + $t4

addi $t0, $t0, 1             # $t0++
blt $t0, $t2, loop
```

In this program, we access elements of an array stored in memory using base + offset addressing. The base address of the array is stored in register $s3, and the offset is computed by multiplying the loop index by 4 (since each integer is 4 bytes). The effective address of each array element is computed by adding this offset to the base address. The lw instruction is then used to load the word (integer) from memory into a register

Question 5:

```
# Perform bitwise AND between $t0 and $t1
and $t2, $t0, $t1             # $t2 = $t0 & $t1

# Shift the result in $t2 left by 3 bits (multiply by 8)
sll $t2, $t2, 3               # $t2 <<= 3

# Perform a bitwise OR between $t2 and $t3
or $t1, $t2, $t3          # $t1 = $t2 | $t3 (reuse $t1 to save space)

# Perform a bitwise XOR between the result and $t4
xor $t5, $t1, $t4             # $t5 = $t1 ^ $t4



beq $t5, $zero, ELSE          # if $t5 == 0, go set $t6 = 1
li $t6, 0                     # else, $t6 = 0
j END                         # jump to end

ELSE:
li $t6, 1

END:
```