CS220- Course Project (Assignment 7)

Submitted By:-

Harsh Oza (210411)

Sujal Lalawat (211066)

Submitted To:-

Respected Urbi Mam

[PDS1]:-

Deciding the registers and their usage protocol :

We have decided to take an array of 32 registers of 5 bits each

[5:0] reg[31:0]

These registers stores the addresses.

1) reg[0] => 0 (Zero)
2) reg [1] => large constants
3) reg[2] and reg [3] => return value from function
4) reg[4] - reg[7] => function arguments
5) reg[8] - reg[17] => temporary variables
6) reg[18] - reg[25] =>saved temporary variables
7) reg[26] and reg[27] => os kernel
8) reg[28] => global pointer
9) reg[29] => stack pointer
10) reg[30] => frame pointer
11) reg[31] => return pointer

[PDS2]:-

In [31:0]memory_veda[1023:0]

First 25 words for Instruction memory

Remaining for Data memory

[PDS3]:-

Instruction layout for R-type , I-type and J-type instructions :

1) R-type :

Opcode(000000)	**rs**	**rt**	**rd**	**shamt**	**Function**
000000(31-26)	5 bits(25-21)	5 bits(20-16)	5 bits(15-11)	5 bits(10-6)	6 bits(5-0)

2) I- type :

Opcode	**rs**	**rt**	**Immediate Address**
6 bits(31-26)	5 bits(25-21)	5 bits(20-16)	16 bits(15-0)

3) J- type :-

Opcode	**Address**
6 bits(31-26)	26 bits(25-0)

List of all opcodes :

Out of 25 Instructions in Instruction memory Some of them are :

Instruction	Opcode / Function
add	32
addu	33
addi	8
addiu	9
sub	34
subu	35
and	36
andi	12
or	37
ori	13
xor	38
sll	0
srl	2
lw	35
sw	43
beq	4
bne	5

<u>Encoding Methodology:</u>

The encoding methodology for each type of instruction is as follows:

1) R-type instruction encoding:

opcode: 6 bits, set to 0b000000 for all R-type instructions rs: 5 bits, the source register 1 index

rt: 5 bits, the source register 2 index

rd: 5 bits, the destination register index

shamt: 5 bits, the shift amount (used only for shift instructions)

funct: 6 bits, the function code that specifies the operation to be performed

2) I-type instruction encoding:

opcode: 6 bits, set to the appropriate opcode for the instruction

rs: 5 bits, the source register index

rt: 5 bits, the destination register index

immediate: 16 bits, the immediate value used in the operation

3) J-type instruction encoding:

opcode: 6 bits, set to the appropriate opcode for the instruction

address: 26 bits, the jump target address

For example, consider the following instructions:

add r0, r1, r2

addi r0, r1, 1000

j 100

These instructions will be encoded as follows:

1] add r0, r1, r2:

opcode: 0b000000

rs: 0b00001 (r1)

rt: 0b00010 (r2)

rd: 0b00000 (r0)

shamt: 0b00000

funct: 0b100000 (add)

2] addi r0, r1, 1000:

opcode: 0b001000

rs: 0b00001 (r1)

rt: 0b00000 (r0)

immediate: 0b0000001111101000 (1000 in 2's complement)

3] j 100:

opcode: 0b000010

address: 0b00000000000000000000000001100100 (100 in binary, shifted left 2 bits to account for byte addressing)

[PDS 9]:-

```
# Bubble Sort Algorithm in MIPS

# Array A[0...n-1] to be sorted

# Assume n = 10
```

```
.data
A: .word 5, 2, 1, 4, 8, 7, 10, 3, 6, 9  # Data to be sorted
n: .word 10                    # Length of array A

.text
.globl main

main:
la $t0, A      # Load address of A into $t0
lw $t1, n      # Load n into $t1

li $t2, 1      # Set $t2 to 1 to enter the first loop
loop1:
beq $t2, $t1, end    # Exit if i equals n

li $t3, 0      # Set $t3 to 0 to enter the second loop
li $t4, 1      # Set $t4 to 1 for comparison with $t3
loop2:
bge $t4, $t1, next   # Exit if j equals n
sll $t5, $t4, 2      # Multiply j by 4 to get the memory location of A[j]
add $t5, $t5, $t0    # Add the memory location of A to get the memory location of A[j]
lw $t6, ($t5)        # Load A[j] into $t6
sll $t7, $t3, 2      # Multiply i by 4 to get the memory location of A[i]
add $t7, $t7, $t0    # Add the memory location of A to get the memory location of A[i]
lw $t8, ($t7)        # Load A[i] into $t8
ble $t6, $t8, skip   # If A[j] is not less than A[i], skip to next iteration
sw $t6, ($t7)        # Swap A[i] and A[j] in memory
sw $t8, ($t5)

skip:

addi $t3, $t3, 1   # Increment j by 1

addi $t4, $t4, 1

j loop2

next:

addi $t2, $t2, 1     # Increment i by 1

j loop1

end:

li $v0, 10       # End program

syscall
```