CS220 IITK-MIPS Report

PDS1

Question: - Decide the registers and their usage protocol.

Answer: - We have used register symbols same as used in **QTSPIM**. We have decided to implement **32 registers**, each one of which is **32 bits wide**. All the registers are used as storage registers except the last register, which is used for **"ra"**.

PDS2

Question: - Decide upon the size of instruction and data memory in VEDA.

Answer: - We have kept our **Program Counter (PC) 32 bits wide**. Therefore, PC is 32*1 bits. We have kept our **Data memory and Instruction memory** of **32 words each** (i.e., each word is 32 bits wide). Therefore, **Data memory and Instruction memory** are 32*32 bits each. We have used the memory for both data and instruction memory.

Size (in bits)	Registers	Usage protocol
32	instruction	Stores the current instruction to be executed
6	ins_op_code	Stores opcode of current instruction
5	ins_rs	Stores address of first source register of current instruction
5	ins_rt	Stores address of second source register of current instruction
5	ins_rd	Stores address of destination register of current instruction
5	ins_shamt	Stores shift amount of current instruction
6	ins_func	Stores function code of current instruction
26	address	Stores address of current J type instruction
16	imm	Stores address of current I type instruction
2	R_type	Stores whether current instruction is R type (1) or not (0)
2	J_type	Stores whether current instruction is J type (1) or not (0)
2	I_type	Stores whether current instruction is I type (1) or not (0)
32	rs	Stores the data at address of first source register of current instruction
32	rt	Stores the data at address of second source register of current instruction
32	rd	Stores the data at address of destination register of current instruction

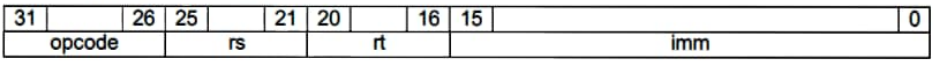
PDS3

Question: - Designing the instruction layout of R, I and J-type instructions and their respective encoding methods.

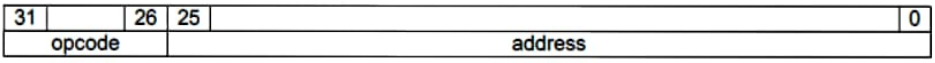
Answer: -

General structure of instruction encoding: -

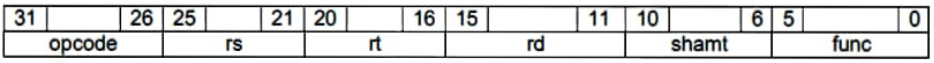
1. **I-Type (immediate)**



2. **J-Type (jump)**



3. **R-Type (register)**



Instruction encoding: -

1. **R-Type**

For a given instruction, we first check the **opcode**. If that comes out to be **0**, then that instruction is **R type**. Then we check the specific instruction by **func code**.

Instruction type	Instruction	Instruction encoding
R type	add r0, r1, r2	opcode = 0, rs, rt, rd, shamt, func = 32
	sub r0, r1, r2	opcode = 0, rs, rt, rd, shamt, func = 34
	addu r0, r1, r2	opcode = 0, rs, rt, rd, shamt, func = 33
	subu r0, r1, r2	opcode = 0, rs, rt, rd, shamt, func = 35
	and r0, r1, r2	opcode = 0, rs, rt, rd, shamt, func = 36
	or r0, r1, r2	opcode = 0, rs, rt, rd, shamt, func = 37
	sll r0, r1, 10	opcode = 0, rs, rt, rd, shamt, func = 0
	srl r0, r1, 10	opcode = 0, rs, rt, rd, shamt, func = 2
	jr r0	opcode = 0, rs, rt, rd, shamt, func = 8
	slt r0, r1, r2	opcode = 0, rs, rt, rd, shamt, func = 42

2. **J-Type**

For a given instruction, we first check the **opcode**. If that comes out to be **2 or 3,** then that instruction is **J type**.

Instruction type	Instruction	Instruction encoding
J type	j 10	opcode = 2, address as per instruction
	jal 10	opcode = 3, address as per instruction

3. **I-Type**

For a given instruction, we first check the **op code**. If that comes out to be anything else except **{0,2,3},** then that instruction is **I type**.

Instruction type	Instruction	Instruction encoding
I type	addi r0, r1, 1000	opcode = 8, rs, rt, imm
	addiu r0, r1, 1000	opcode = 9, rs, rt, imm
	andi r0, r1, 1000	opcode = 12, rs, rt, imm
	ori r0, r1, 1000	opcode = 13, rs, rt, imm
	lw r0, 10(r1)	opcode = 35, rs, rt, imm
	sw r0, 10(r1)	opcode = 43, rs, rt, imm
	beq r0, r1, 10	opcode = 4, rs, rt, imm
	bne r0, r1, 10	opcode = 5, rs, rt, imm
	bgt r0, r1, 10	opcode = 7, rs, rt, imm
	bgte r0, r1, 10	opcode = 1, rs, rt, imm
	ble r0, r1, 10	opcode = 6, rs, rt, imm
	bleq r0, r1, 10	opcode = 11, rs, rt, imm
	slti 1, 2, 100	opcode = 10, rs, rt, imm