CS220 Assignment 8 Report

Aarav Oswal
230012

PDS1 - Decide the registers and their usage protocol.

We have decided to use 32 general purpose registers each of width 32 bits similar to the MIPS architecture as follows:

- $zero ($0): Always zero
- $at ($1): Reserved for assembler
- $v0-$v1 ($2-$3): Function results
- $a0-$a3 ($4-$7): Arguments
- $t0-$t7 ($8-$15): Temporaries
- $s0-$s7 ($16-$23): Saved registers
- $t8-$t9 ($24-$25): Temporaries
- $k0-$k1 ($26-$27): Reserved for OS
- $gp ($28): Global pointer
- $sp ($29): Stack pointer
- $fp ($30): Frame pointer
- $ra ($31): Return address

For floating point registers, we have decided to use 32 registers each of size 32 bits, all of which can be used for storing floating point numbers except the register $31 which denotes cc.

Apart from these we have decided to use the following special registers each of size 32 bits:

- PC: Program Counter
- hi and lo: For storing multiplication results of operations mul, madd and maddu

PDS2 - Decide upon the size for instruction and data memory.

Since the PC is 32 bits wide, we have decided to use the Instruction Size as 32 bits.
We have decided to keep the size of Instruction memory as 512 words or 2KB and the size of Data Memory also as 512 words or 2KB.

PDS3 - Design the instruction layout for R-, I- and J-type instructions and their respective encoding methodologies.

General Structure of encoding:

1. R-Type Instructions:

31-26 (6)	25-21 (5)	20-16 (5)	15-11 (5)	10-6 (5)	5-0 (6)
opcode	rs	rt	rd	shamt	func

2. I-Type Instructions:

31-26 (6)	25-21 (5)	20-16 (5)	15-0 (16)
opcode	rs	rt	imm

3. R-Type Instructions:

31-26 (6)	25-0 (26)
opcode	address

R-type Instructions:
sll, srl, sla, sra, jr, madd, maddu, mul, add, addu, sub, subu, amd, or, xor, not, slt, add.s, sub.s, mov.s, mfc1, mtc1, c.eq.s, c.le.s, c.lt.s, c.ge.s, c.gt.s

I-type Instructions:
addi, addiu, slti ,seq, andi, ori, xori, lui, beq, bne, bgt, bgte, ble, bleq, bleu, bgtu

J-type Instructions:
j, jal

Encoding of all Instructions:

Class	Instruction	Instruction Encoding
Arithmetic	add r0, r1, r2	opcode = 0, rs, rt, rd, shamt = 0, func = 32

	sub r0, r1, r2	opcode = 0, rs, rt, rd, shamt = 0, func = 34
	addu r0, r1, r2	opcode = 0, rs, rt, rd, shamt = 0, func = 33
	subu r0, r1,r2	opcode = 0, rs, rt, rd, shamt = 0, func = 35
	addi r0, r1,c	opcode = 8, rs, rt, imm = c
	addiu r0, r1, c	opcode = 9, rs, rt, imm = c
	madd r0, r1	opcode = 0, rs, rt, rd = 0, shamt = 0, func = 28
	maddu r0, r1	opcode = 0, rs, rt, rd = 0, shamt = 0, func = 29
	mul r0, r1	opcode = 0, rs, rt, rd = 0, shamt = 0, func = 30
Logical	and r0, r1, r2	opcode = 0, rs, rt, rd, shamt = 0, func = 36
	or r0, r1, r2	opcode = 0, rs, rt, rd, shamt = 0, func = 37
	andi r0, r1, c	opcode = 12, rs, rt, imm = c
	ori r0, r1, c	opcode = 13, rs, rt, imm = c
	not r0, r1	opcode = 0, rs, rt = 0, rd, shamt = 0, func = 39
	xori r0, r1, c	opcode = 14, rs, rt, imm = c
	xor r0, r1, r2	opcode = 0, rs, rt, rd, shamt = 0, func = 38
	sll r0, r1, c	opcode = 0, rs, rt = 0, rd, shamt = c, func = 0
	srl r0, r1, c	opcode = 0, rs, rt = 0, rd, shamt = c, func = 1
	sla r0, r1, c	opcode = 0, rs, rt = 0, rd, shamt = c, func = 2
	sra r0, r1, c	opcode = 0, rs, rt = 0, rd, shamt = c, func = 3
Data Transfer	lw ro, c(r1)	opcode = 35, rs, rt, imm = c
	sw ro, c(r1)	opcode = 43, rs, rt, imm = c
	lui ro, c	opcode = 15, rs = r0, rt = r0, imm = c
Conditional Branch	beq r0, r1, c	opcode = 24, rs, rt, imm = c
	bne r0, r1, c	opcode = 25, rs, rt, imm = c
	bgt r0, r1, c	opcode = 26, rs, rt, imm = c
	bgte r0, r1, c	opcode = 27, rs, rt, imm = c
	ble r0, r1, c	opcode = 28, rs, rt, imm = c

	bleq r0, r1, c	opcode = 29, rs, rt, imm = c
	bleu r0, r1, c	opcode = 30, rs, rt, imm = c
	bgtu r0, r1, c	opcode = 31, rs, rt, imm = c
Unconditional Branch	j c	opcode = 2, addr = c
	jr r0	opcode = 0, rs, rt = 0, rd = 0, shamt = 0, func = 8
	jal c	opcode = 3, addr = c
Comparison	slt r0, r1, r2	opcode = 0, rs, rt , rd, shamt = 0, func = 42
	slti r0, r1, c	opcode = 10, rs, rt, imm = c
	seq r0, r1, c	opcode = 11, rs, rt, imm = c
Floating point	mfc1 r0, f0	opcode = 1, rs = f0, rt = 0, rd = r0, shamt = 0, func = 8
	mtc1 f0, r0	opcode = 1, rs = r0, rt = 0, rd = f0, shamt = 0, func = 9
	add.s f0, f1, f2	opcode = 1, rs, rt, rd, shamt = 0, func = 0
	sub.s f0, f1, f2	opcode = 1, rs, rt, rd, shamt = 0, func = 1
	c.eq.s cc f0, f1	opcode = 1, rs, rt, rd = 31, shamt = 0, func = 24
	c.le.s cc f0, f1	opcode = 1, rs, rt, rd = 31, shamt = 0, func = 25
	c.lt.s cc f0, f1	opcode = 1, rs, rt, rd = 31, shamt = 0, func = 26
	c.ge.s cc f0, f1	opcode = 1, rs, rt, rd = 31, shamt = 0, func = 27
	c.gt.s cc f0, f1	opcode = 1, rs, rt, rd = 31, shamt = 0, func = 28
	mov.s cc f0, f1	opcode = 1, rs, rt = 0, rd, shamt = 0, func = 6
Exit	exit	opcode = 63

Architecture of my IITK-Mini-MIPS