Project Report

Garvit Arora **Apurb Agarwal**

Abstract

This report contains the documentation of **CS220 (Computer Organization) Assignment 7**. Our objective is to implement a new processor called **CSE-BUBBLE**. A single instruction gets executed in each clock cycle. Each of the **10** steps to complete the project have been documented below.

1 PDS1 - Registers and their usage protocol

We have **32** registers in our architecture. Each register stores **32** bits. The names, register numbers and protocol for using them are given below.

Register Nos.	Register names	Usage
0	`$zero`	value is always 0
1-3	`$v0-$v2`	return values
4-7	`$a0-$a3`	arguments
8-15	`$t0-$t7`	temporary values
16-23	`$s0-$s7`	variables
24-25	`$t8-$t9`	temporary values
28	`$gp`	global pointer
29	`$sp`	stack pointer
30	`$fp`	frame pointer
31	`$ra`	return address

2 PDS2 - Size of instruction and data memory

The memory element of our processor is called **VEDA**. It has two components, **instruction memory** and **data memory**. Both of them are 2D matrices. Since we use byte addressing, each row of both memory stores **8** bits (1 byte). Each instruction has **32** bits (4 bytes). Similarly each data element stored in the data memory is also **32** bits (4 bytes). Therefore we use 4 rows from the memory to represent each instruction as well as data element. There are **1024** rows in both the instruction memory as well as the data memory.

Memory Name	Size of 1 row	Number of rows
Instruction memory	8 bits	1024
Data memory	8 bits	1024

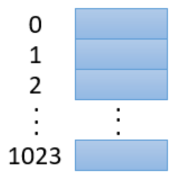

Figure 1: Memory

3 PDS3 - Instruction layout and encoding schemes

3.1 R type instructions

The general layout for R type instructions is given below.

opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

The encoding schemes for individual R type instructions are as follows.

instruction	opcode	rs	rt	rd	shamt	funct
`add r0, r1, r2`	0	r1	r2	r0	0	0
`sub r0, r1, r2`	0	r1	r2	r0	0	1
`addu r0, r1, r2`	0	r1	r2	r0	0	2
`subu r0, r1, r2`	0	r1	r2	r0	0	3
`and r0, r1, r2`	0	r1	r2	r0	0	4
`or r0, r1, r2`	0	r1	r2	r0	0	5
`sll r0, r1, k`	0	r1	na	r0	k	6
`srl r0, r1, k`	0	r1	na	r0	k	7
`slt r0, r1, r2`	0	r1	r2	r0	0	8

Note: Here `r0, r1, r2` denotes registers and `k` denotes any integer.

3.2 I type instructions

The general layout for I type instructions is given below.

opcode	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

The encoding schemes for individual I type instructions are as follows.

instruction	opcode	rs	rt	imm
`addi r0,r1,k`	4	r1	r0	k
`addiu r0,r1,k`	5	r1	r0	k
`andi r0,r1,k`	6	r1	r0	k
`ori r0,r1,k`	7	r1	r0	k
`lw r0,k1(r1)`	8	r1	r0	k1
`sw r0,k1(r1)`	9	r1	r0	k1
`beq r0,r1,k`	10	r0	r1	k
`bne r0,r1,k`	11	r0	r1	k
`bgt r0,r1,k`	12	r0	r1	k
`bgte r0,r1,k`	13	r0	r1	k
`ble r0,r1,k`	14	r0	r1	k
`bleq r0,r1,k`	15	r0	r1	k
`slti r0,r1,k`	15	r1	r0	k

Note: Here `r0` and `r1` denote registers. `k` and `k1` denotes integers and `k1` must always be divisible by 4 (due to byte addressing).

3.3 J type instructions

The general layout for J type instructions is given below.

opcode	address
6 bits	26 bits

The encoding schemes for individual J type instructions are as follows.

instruction	**opcode**	**address**
`j k`	`1`	`k`
`jr r0`	`2`	`r0`
`jal k`	`3`	`k`

Note: Here `r0` denotes a register. `k` denotes an address.

4 PDS4 - Instruction fetch

We send the instruction address (PC) to the VEDA memory module. One of the outputs from the module is the 32 bit instruction.

5 PDS5 - Instruction Decode

The input to this module is the opcode of an instruction and the output is the type of the instruction.

opcode	**instruction type**
0	R type
1-3	J type
4-16	I type

6 PDS6 - Arithmetic Logic Unit (ALU)

The Arithmetic Logic Unit takes in three inputs. Two of them (**a,b**) are operands on which the arithmetic operation is performed and the third is the control signal (**c**) which determines the type of arithmetic operation. Each of the arithmetic operations has its own module.

control signal	**arithmetic operation**
0	`AND`
1	`OR`
2	`add`
3	`shift left`
4	`shift right`
6	`sub`
7	`set less than`
12	`nor`

The ALU has three outputs. The first (**out**) stores the return value after performing the operation on the operands. Apart from that there are two more outputs. **z** stores either 0 or 1 depending on whether out is zero or not. **f** stores the first bit of **out**. **z** and **f** help us in the conditional branching operations.

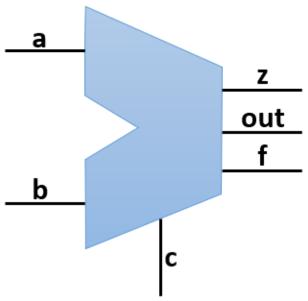

Figure 2: ALU

7 PDS7 - Branching Operations

For the unconditional branching operations like `j, jal` and `jr` we directly jump to the corresponding address. But for conditional branching we need to use the subtractor module from ALU. We use the output signals **z** and **f** to decide whether to branch or not. The control unit outputs two signals **branch** and **branchf** which are used in conditional branching operations. **branch** denotes the type of conditional branching operation and **branchf** denotes whether the instruction is a conditional branching operation or not.

Instruction	branch	condition to branch
`beq`	0	$\mathbf{z}$
`bne`	1	$\overline{\mathbf{z}}$
`bgt`	2	$\overline{\mathbf{z}} \cdot \overline{\mathbf{f}}$
`bgte`	3	$\overline{\mathbf{f}}$
`ble`	4	$\overline{\mathbf{z}} \cdot \mathbf{f}$
`bleq`	5	$\mathbf{z} \cdot \overline{\mathbf{f}} + \overline{\mathbf{z}} \cdot \mathbf{f}$

The instruction address (PC) branches only when **branchf** as well as the **condition to branch** (as defined above) are true.

8 PDS8 - Control Unit

We have used two control units.

8.1 Main Control Unit

This module takes the **opcode** of an instruction as input and outputs various control signals that decide the flow of data. The meaning of various control signals are listed below.

- **aluop** : helps to choose the control signal for alu
- **regDst** : chooses destination register
- **regWrite** : decided whether to write into a register
- **alusrc** : decides second operand of alu
- **memtoreg** : decides which data is written into the register
- **memread** : read from memory or not
- **memwrite** : write to memory or not
- **branch** : type of branch operation
- **branchf** : whether it is branching operation or not
- **jump** : type of jump operation

The values of the control signals for various opcodes are as follows.

opcode	aluop	rD	rW	as	mreg	mread	mw	b	bf	jump
0	2	1	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	2
3	0	0	1	0	0	0	0	0	0	3
4-5	0	0	1	1	0	0	0	0	0	0
6	3	0	1	1	0	0	0	0	0	0
7	4	0	1	1	0	0	0	0	0	0
8	0	0	1	1	1	1	0	0	0	0
9	0	0	0	1	0	0	1	0	0	0
10-15	1	0	0	0	0	0	0	op-10	1	0
16	1	0	1	1	0	0	0	0	0	0

8.2 ALU Control Unit

The ALU Control Unit takes **aluop** and **funct** as inputs. It has two outputs. **c** denotes the alu control signal which determines the arithmetic operation to be performed by the ALU. **chooseshift** denotes whether the operation to be performed is a shift operation or not.

aluop	funct	c	chooseshift
2	0	2	0
2	1	6	0
2	2	2	0
2	3	6	0
2	4	0	0
2	5	1	0
2	6	3	1
2	7	4	1
2	8	7	0
0	na	2	0
1	na	6	0
3	na	0	0
4	na	1	0

9 PDS9 - Bubble Sort

The MIPS code as well as the machine code for *Bubble Sort* are included in the zip file.

10 PDS10 - Execution

The execution of the processor for *Bubble Sort* was shown in the lab.

Note: All the relevant codes are included in the zip file.

END
Thank You