Lab 1 (All Sections) Prelab: Introduction to MIPS instructions

Name:

Kerry Zeng

Sign the following statement:

On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work

Kerry Zeng

I Read It!!

Please mark the following Checkbox to indicate you have read all the material of the "Lab Procedures and Policies" document ☑.

1 Objective

The main objective of this prelab is to familiarize you with several basic MIPS instructions and their appropriate encoding formats.

2 Prerequisites

Before proceeding, you should learn basic MIPS instructions such `add, sub, addi, lw, sw` and their appropriate formats (refer to Sections 2.2-2.5 of the text book).

3 Introduction

Before commanding a computer's hardware, you must speak its language. The words of a computer language are called *instructions*, and its vocabulary is called an *instruction set.* For example, the following `add` instruction tells a computer to add the two variables `b` and `c` and store the result in `a`.

```
add a, b, c
```

In this lab, we will explore several instructions in the MIPS instruction set and understand their encoding formats.

4 Instruction encoding

Instructions inside the computer are represented as a series of ones and zeros (binary digits).

For example the instruction `add $t0, $s1, $s2` is encoded as:

000000	10001	10010	01000	00000	100000
6bits	5bits	5bits	5bits	5bits	6bits

Each of these segments of an instruction is called a *field*. The first and last fields in combination tell the MIPS computer that this instruction performs addition. The second field specifies the number of the register which corresponds to the first source operand of the addition operation ($17 = \$s1$). The third field specifies the second source operand ($18 = \$s2$), and the fourth field specifies the register to hold the result ($8 = \$t0$). The fifth field is unused in this instruction, so it is set to 0.

This layout of the instruction is referred to as instruction format. In MIPS, the size of each instruction is exactly of 32 bits. The instruction format in MIPS can be categorized as register (R), immediate (I), or jump (J). The above example is an R-format instruction. For this lab, we will concentrate on the register (R) and immediate (I) formats.

Note: A *word* is defined as the unit of access in a computer. In MIPS, all data and instruction accesses are 32-bits and hence a 32-bit value is called a word in MIPS.

Most of the processors have a restriction on the alignment of data. In MIPS, we have this alignment restriction; words in MIPS must start at addresses that are multiples of 4.

For the following questions, feel free to use the MIPS reference card that can be found on the last page.

5 Questions

1. Write down the representation of R- and I-format instructions. Explain the purpose of each field in the instruction.

   R format instructions are used for register type instructions. For example things like adding or shifting registers
   B31-26 | B25-21 | B20-16 | B15-11 | B10-6 | B5-0
   1. This is the opcode and it determines which operation to perform. 6 bits
   2. The second one is the destination register where the result of the operation is stored.
   3. This is one of the operands.
   4. This is the second operand.

2. Which of the following are Register or Immediate type instructions? Place the appropriate letter (R or I) next to them:

sub	R	andi	I	beq	I
lw	I	slt	R	srl	R

3. Write the binary encoding for the following instructions. Clearly indicate the different fields in the encoding.

add $s1,$s2,$s3	000000 10010 10011 10001 00000 100000
addi $s1, $s2,40	001000 10010 10001 0000000000101000
sub $s6,$s7,$t3	000000 10111 01011 10110 00000 10010

4. Translate the following high-level code into assembly language. Assume variables $a - c$ are held in registers $\$s0 - \$s2$ and $f - j$ are in $\$s3 - \$s7$

   ```
   a=b-c;
   ```

```
f=(g+h)-(i+j);
```

```
a=b-c
sub $s0,$s1,$s2

f=(g+h)-(i+j)
add $t0, $s4, $s5
add $t1, $s6, $s7
sub $s3, $t0, $t1
```

5. Write an assembly program to swap the contents of 2 variables stored in registers $4 and $5.

```
addi $t0,$5,$0
addi $4,$5,$0
addi $5,$t0,$0

uses $s0 to temporarily store the contents of 5 and then does the swap.
```

6. Read the tutorial on the MARS MIPS simulator. In lab we will be using MARS to simulate a simple MIPS processor. The MARS tutorial can be found at:

   http://courses.missouristate.edu/KenVollmar/MARS/tutorial.htm

   You can install MARS on your computer to get better understanding of the simulator.

   I have read the MARS tutorial ☑.

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	`add`	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$
Add Immediate	`addi`	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	`addiu`	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	`addu`	R	R[rd] = R[rs] + R[rt]		$0 / 21_{hex}$
And	`and`	R	R[rd] = R[rs] & R[rt]		$0 / 24_{hex}$
And Immediate	`andi`	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	`beq`	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	`bne`	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	`j`	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	`jal`	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	`jr`	R	PC=R[rs]		$0 / 08_{hex}$
Load Byte Unsigned	`lbu`	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	`lhu`	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	`ll`	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	`lui`	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	`lw`	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	`nor`	R	R[rd] = ~ (R[rs] | R[rt])		$0 / 27_{hex}$
Or	`or`	R	R[rd] = R[rs] | R[rt]		$0 / 25_{hex}$
Or Immediate	`ori`	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	`slt`	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		$0 / 2a_{hex}$
Set Less Than Imm.	`slti`	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	`sltiu`	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	`sltu`	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	$0 / 2b_{hex}$
Shift Left Logical	`sll`	R	R[rd] = R[rt] << shamt		$0 / 00_{hex}$
Shift Right Logical	`srl`	R	R[rd] = R[rt] >> shamt		$0 / 02_{hex}$
Store Byte	`sb`	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	`sc`	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	`sh`	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	`sw`	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	`sub`	R	R[rd] = R[rs] - R[rt]	(1)	$0 / 22_{hex}$
Subtract Unsigned	`subu`	R	R[rd] = R[rs] - R[rt]		$0 / 23_{hex}$

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS