PHYS 234 Lab 9: MIPS Assembly and Machine Code

Ethan Yang

The goal of this lab is to construct a program in machine language, and then convert them into binary and hexadecimal instructions. The program would perform 4 actions: store a value in $s0, which is also register 16, store another value in $s1 (register 17), store the sum of the two values in $s2 (register 18), and store this sum in address 32. The MIPS Assembly code is written below:

```
addi $s0, $0, 1
addi $s1, $0, 2
add $s2, $s0, $s1
sw $s2, 32($0)
```

The first two lines use the add immediate (addi) instruction to initialize 2 different values and store them in $s0 and $s1. Line 1 adds the immediate, or constant value, 1 to register $0, which has a value of 0, and stores the result 1 in the destination register $s0. Line 2 adds the immediate 2 to register $0 and stores the result 2 to $s1.

Line 3 uses the add instruction to sum the values in register $s0 and $s1 and store the result in $s2. As a result, $s2 would store 1 + 2 = 3.

Line 4 uses the store word (sw) instruction to store the value in $s2 to the address calculated by adding an offset, which is 32 in this case, to the address of the base register, $0 in this case. Therefore, the value 3 would be stored into address 32.

To convert these instructions into binary and hexadecimal, we need to identify each line's type of machine language and how to format their op (operation code), rs (first source register), rt (second source register), and etc. Since addi and sw are I-type instructions, their binary machine language is formatted in the order of op, rs, rt, and imm. Add is an R-type instruction, and its binary machine language is formatted in the order of op, rs, rt, rd, shamt, and funct. The op code and funct is 6 bits, rs, rt, rd, and shamt are 5 bits, and imm is 16 bits. The op code for addi is 8, 43 for sw, and 0 for R-type instructions. The funct for add is 32. We can plug in the value for each of them to get the binary instruction, and then convert them into hexadecimal. The processes are shown below:

Instruction	op	rs	rt	imm	Binary	Hexadecimal
addi $s0, $0, 1	8	0	16	1	001000 00000 10000 0000000000000001	0x20100001
addi $s1, $0, 2	8	0	17	2	001000 00000 10001	0x20110002

					0000000000000010	
sw $s2, 32($0)	43	0	18	32	101011 00000 10010 0000000000100000	0xAC120020

Instruction	op	rs	rt	rd	shamt	funct	Binary	Hexadecimal
add $s2, $s0, $s1	0	16	17	18	0	32	000000 10000 10001 10010 00000 100000	0x02119020

As a result, we translated the Assembly machine code into hexadecimal instructions, as shown in the rightmost columns of the tables, and rewritten in order below:

0x20100001
0x20110002
0x02119020
0xAC120020