CSE 230 - Assignment 3
Mohseen - mmukadda
Vishnu - vnnarang

1: Code file submitted.

2:

(2 pts) Pseudoinstructions. We discussed commonly-used pseudoinstructions in §2.3.8 of the Ch. 2 lecture notes. Remember that pseudoinstructions are software-emulated instructions (or virtual instructions) in the sense that the assembler replaces pseudoinstructions with one or more hardware instructions (or physical instructions) that achieve the desired effect. Suppose you are writing a MIPS assembler and you can create your own pseudoinstructions. Two common instructions found in many different instruction sets are one that increments an integer value in a register (to increment means add one) and one that decrements the integer value (subtracts one). You decide to implement two pseudoinstructions

inc reg # reg ← reg + 1
dec reg # reg ← reg - 1

For both inc and dec, what would be the equivalent hardware instruction(s)?

Answer:

Increment and decrement are both operations involving 1 as an immediate value. We can use addi to implement both as follows:

```
# Increment $reg by 1 (assuming we have a register $reg with us )
addi $reg, $reg, 1        # $reg ← $reg + 1

# Decrement $reg by 1 (assuming we have a register $reg with us )
addi $reg, $reg, -1       # $reg ← $reg - 1
```

3:

(5 pts) Instruction Encoding. What would be the MIPS32 encoding of this instruction: or $t8, $s3, $t5? Write your answer as an 8-hexdigit integer, e.g., 0x12345678, and for full credit show/explain how your answer was obtained.

Answer:

'or' is a R-format instruction (refer the Green card) and it's opcode is 000000.
R-format instruction is of the format:

op	6 bits	31:26	Opcode
rs	5 bits	25:21	First source register operand
rt	5 bits	20:16	Second source register operand
rd	5 bits	15:11	Destination register operand
shamt	5 bits	10:6	Shift amount - only for shifting instructions
funct	6 bits	5:0	Function code combined with opcode to identify Instruction

Hence for **or $t8, $s3, $t5** instruction we have the following:

Syntax: or $dst, $src1, $src2 # $dst ← $src1 | $src2

op:	000000	
rs:	10011	# $s3 is Register number 19 from green card
rt:	01101	# $t5 is Register number 13
rd:	11000	# $t8 is Register number 24
shamt	00000	# Shift amount is 0 as this is not a shift instruction.
funct	100101	# 25 hex is the function code from green card

Hence, the MIPS encoding for the given instruction is:
000000 10011 01101 11000 00000 100101 (op:rs:rt:rd:shamt:funct)

Which in hex format is: 0x026DC025

4.

(5 pts) Instruction Encoding. What would be the encoding of this instruction:
lw $a1, -16($fp)? Write your answer as an 8-hexdigit integer, e.g., 0x12345678, and for full credit show/explain how your answer was obtained.

Answer:

'lw' is an I-format instruction.

I format instruction has following fields:

op	6 bits	31:26	Opcode
rs	5 bits	25:21	First source register operand
rt	5 bits	20:16	Source or destination register operand
imm	16 bits	15:0	A 16 bit signed immediate

Hence for **lw $a1, -16($fp)** instruction we have the following:

syntax: lw $rt, offset($base)

op:	100011	# Opcode for lw is 23hex
rs:	11110	# $fp is Register number 30 in Green Card
rt:	00101	# $a1 is Register number 5 in Green Card
imm:	1111111111110000	# 2's complement of -16

Hence MIPS encoding for the instruction is:
100011 11110 00101 1111111111110000 (op:rs:rt:imm)

Which in Hex format is 0x8FC5FFF0

5.

(13 pts) You are hacking away on a MIPS32 system trying to understand how the code in a MIPS binary executable file works. You have loaded the binary into your debugger and are focused on the following memory addresses and the 8-hex digit values stored in them,

0x0040_0000: 0x3C011001
0x0040_0004: 0x34300000
0x0040_0008: 0x8E080000

0x0040_000C: 0x21080001
0x0040_0010: 0xAE080000

You know these values represent instructions because you know that 0x00400000 is the starting address of the text segment in MIPS programs. Decode each of these instructions and write the corresponding symbolic assembly language instructions. While playing around with the program in the debugger, you discover that memory addresses 0x10010000–0x10010003 appear to be allocated to an integer variable; let us call this variable x. Knowing that, please explain with one or two English sentences what the above five instructions are doing.

Answer:

Decoding the instructions:

i) 0x3C011001

- convert it into binary:
  0011 1100 0000 0001 0001 0000 0000 0001
- Opcode (first 6 bits is : 0011 11) is : f hex
  This is the opcode for the **lui** instruction (from the green card).
- Syntax for this instruction is: lui $dst, label(31:16)
  It loads the 16 upper bits from the label into the 16 upper most bits of the $dst register.
- $dst in this case is (00 000 0 0001).
  We have rs: $zero and $rt: $at (Assembler temporary).
- From the remaining bits we have imm: 4097

From the above information we see that the instruction 0x3C011001 is loading 4097 immediate value (upper 16 bits) into the assembler temporary register $at.

ii) 0x34300000

- convert it into binary:
  0011 0100 0011 0000 0000 0000 0000 0000
- Opcode (first 6 bits is : 0011 01) is : D hex
  This is the opcode for the **ori** instruction (from the green card).
- Syntax for this instruction is: ori rt, rs, imm
- It performs the OR operation on the rs register and the imm, and store the result in rt register
- rs (25:21) in this case is (00 001) $1 -> $at (assembler temporary).
  rt (20:16) would be: (1 0000) $16 -> $s0 (saved temporary).
- Imm would be: (0000 0000 0000 0000) -> 0 (16 bits)

Therefore, we can conclude from the given encoding, 0x34300000 can be decoded to
ori $s0, $at, 0

iii) 0x8E080000

- Convert it into binary:
  1000 1110 0000 1000 0000 0000 0000 0000

- Opcode (first 6 bits is : 1000 11) is : 23 hex
  This is the opcode for the **lw** instruction (from the green card).
  syntax: lw $rt, offset($base)
- rs: 10 000 = 16 (i.e. Register $s0)
- rt: 0 1000 = 8 (i.e. Register $t0)
- imm: 0000 0000 0000 0000 = 0 (i.e. Immediate value is 0)
- Hence the instruction decoded is: **lw $t0, 0($s0)**

iv) 0x21080001

- Convert it into binary:
  0010 0001 0000 1000 0000 0000 0000 0001
- Opcode (first 6 bits is : 0010 00) is : 08 hex
  This is the opcode for the **addi** instruction (from the green card).
  syntax: addi $dest, $src1, $src2 (R[rt] = R[rs] + SignExt-Imm)
- rs: = 01 000 = 8 (i.e. Register $t0)
- rt: 0 1000 = 8 (i.e. Register $t0)
- imm: 0000 0000 0000 0001 = 1 (i.e. Immediate value is 0)
- Hence the instruction decoded is: **addi $t0, $t0, 1**
  (i.e. it's incrementing the value in $t0 by 1).

v) 0xAE080000

- Convert it into binary:
  1010 1110 0000 0100 0000 0000 0000 0000
- Opcode (first 6 bits is : 1010 11) is : 17 hex
  This is the opcode for the **lw** instruction (from the green card).
  syntax: sw $rt, offset($base)
- rs: 10 000 = 16 (i.e. Register $s0)
- rt: 0 0100 = 4 (i.e. Register $a0)
- imm: 0000 0000 0000 0000 = 0 (i.e. Immediate value is 0)
- Hence the instruction decoded is: **sw $a0, 0($s0)**