ELEC 5200/ELEC 6200 Computer Architecture and Design
Test 1, Feb 11, 2019 **Total 33 points**
Broun 235, 11:00-11:50AM

Instructions: Please read all problems before writing your answers. Attempt all eight (8) problems. Be sure to revise your answers before turning them in. Good luck!

Problem 1: **(3 points)**

Define Instruction Set Architecture:

Answer:

An instruction set architecture (ISA) is the abstract interface between the hardware and low level software. It encompasses all of the information necessary to translate any program for machine processing. It includes the **instructions**, number and types of **registers**, and the **data transfer** modes (instructions) between registers, memory, and I/O.

Problem 2: **(5 points)**

Answer the following questions about a computer system:

(a) How does a compiler deal with pseudoinstructions?

Answer:

A compiler may treat pseudoinstructions and executable instructions alike.

(b) How does the MIPS assembler deal with pseudoinstructions? What rules should it follow?

Answer:

An assembler replaces each pseudoinstruction with one or more executable assembly language instructions. It should use a minimum number of executable instructions for each pseudoinstruction. Those executable instructions should use only the registers specified in the pseudoinstruction, and $0 and $1 (register at).

(c) Why does a pseudoinstruction not require an opcode?

Answer:

The hardware recognizes an instruction by decoding its opcode. Since a pseudoinstruction is replaced by other executable instructions with opcodes by the assembler, it is never executed by the hardware. Hence, it does not need an opcode.

(d) A compiler produces 1 million MIPS instructions. The same code when assembled results in 1.2 million instructions. Explain the difference. How does an assembler minimize the increase in the number of instructions?

Answer:

The increase is due to the presence of pseudoinctructions in the compiled code. Those are replaced by one or more executable instructions by the assembler. It should use a minimum number of executable instructions for each pseudoinstruction.

(e) Implement the following move pseudoinstruction using executable MIPS ISA:

```
mov     $r1, $r2        # r1 ← r2
```

Answer:

Pseudoinstruction mov can be expanded as:

add $r1, $r2, $zero or addi $r1, $r2, 0

Problem 3: **(6 points)**

Specify functions of various types of jump instructions in the MIPS instruction set. Which registers are written to and with what values by each type of jump instruction?

Answer: There are three types of jump instructions:

(1) **Jump (j)** causes the program to skip the next instruction and instead execute an instruction whose memory address is specified as the argument of the j instruction. The j instruction changes the contents of the program counter (PC). The new value written into PC is obtained from the 26-bit argument of the j instruction by first shifting the argument left by two bits and concatenating this 28 bit value with the 4 most significant bits of the current PC.

(2) **Jump and link (jal)** causes a subroutine to be executed. It first writes register 31 (return address register) with the memory address of the next instruction. Then the memory address where the called subroutine begins (specified as argument of the jal instruction) is written into PC.

(3) **Jump register (jr)** causes return from a called subroutine to the caller program or subroutine. It writes the content of register 31 into PC.

Problem 4: **(4 points)**

Assume that you would like to expand the size of the MIPS register file to 64 registers.

How would this impact the size of each bit field in an R-type instruction?

Answer:

For R-tye instructions the *rs*, *rt*, and *rd* fields must be extended from 5 bits to 6 bits. This leaves 8 bits for shamt and func fields combined if the opcode length remains unchanged (otherwise the instruction length must increase).

How would this impact the size of each bit field in a J-type instruction?

Answer:

All bitfields for J type instructions would remain unchanged.

Problem 5: **(3 points)**

Suppose the PC contains the value 0x2000 0000. The instruction at this address is the branch-on-equal (beq) MIPS instruction. Is it possible for this instruction to set the PC to address 0x2000 5000? If so, what should the value of the *immediate* field of the branch instruction be?

Answer:: Yes, it is possible. The value of the 16 bit immediate field should be 0x13FF (hex), 5,119 (dec), or 0001 0011 1111 1111 (bin) (any of the three are acceptable).

Branch address is equal to (PC+4) + *immediate**4. See pg 114 of textbook for more.

Problem 6: **(4 points)**

Show MIPS assembly code that would implement the following high level language code. Use the following register assignments: A is $t0, B is $t1, C is $t2, D is $t4, R is $v3.

```
A = (B + C) - 8*(D + R);
```

Comment your code.

Answer:

```
add   $t1, $t1, $t2   # $t1 = B + C
add   $t4, $t4, $v3   # $t4 = D + R
sll   $t4, $t4, 3     # $t4 = 8*(D + R)
sub   $t0, $t1, $t4   # $t0 = (B+C) – 8*(D+R)
```

Problem 7: **(8 points)**

A compiler designer is trying to decide between two code segments for a particular machine. The hardware designers have provided the following data below about the CPI for each class, and the instruction counts being considered for each code sequence.

Instr. class	CPI for this instr class
A	5
B	3

	Instruction count per instruction class	
Code sequence	A	B
1	4	6
2	6	3

How many clock cycles are required for:

a) Code sequence 1?

Answer:

Cycles required = CPI_A * $Num\_Instructions_A$ + CPI_B * $Num\_Instructions_B$
= 5 * 4 + 3 * 6 = 20 + 18 = 38 cycles

b) Code sequence 2?

Answer:

Cycles required = 5 * 6 + 3 * 3 = 30 + 9 = 39 cycles

c) Which code sequence is faster and how by how much?

Answer: Code sequence 1 is one clock cycle faster.

What is the average CPI for code sequence 1?

Answer:

Average CPI = CPI_A * PCT_INSTR_A + CPI_B * PCT_INSTR_B

$$= 5 \times \left(\frac{4}{4+6}\right) + 3 \times \left(\frac{6}{4+6}\right) = 5 \times 0.4 + 3 \times 0.6 = 3.8$$

Problem 8 (A Bonus Question): (3 points)

What is the von Neumann Bottleneck and how does it impact performance?

Answer:

Memory accesses are generally slow compared to the speed of the processor. The von Neumann bottleneck refers to the fact that the speed at which a processor can operate is limited, in practice, by the speed at which it can get data from memory. The larger the bottleneck (the difference in speed that memory data can be accessed vs the speed at which the cpu processes that data), the more time the cpu sits idle while waiting for data. Restricting most operands to cpu registers (and other approaches to be covered later) can limit the effect of the von Neumann bottleneck.