Hwk #2 – ISAs & ALUs
95 points
Due in class Thursday, 17 September, 2009

1. (20) Consider the following C code segment:

```
A = B + C;
B = A + C;
D = A – B;
```

a. Write an equivalent assembly language code for each of the four following ISAs; try to minimize the number of instructions you use.

You may use the following instructions:

load, store, add, sub, push, pop and negate

Assume all variables (A, B, C, and D) are initially in memory.

- **Accumulator**: all operations occur between a single register (the accumulator) and a memory location.
- **Stack**: All operations occur on top of the stack. Push and Pop are the only instructions that access memory; all others remove operands from the top of the stack and replace them with the result. You may assume the stack is sufficiently large to accommodate your needs (ie, it doesn't spill into memory).
- **Register-Memory**: All instructions can use at most two registers. Arithmetic operations involving a register and a memory location are permitted (the register is used as a source and destination).
- **Register-Register**: All arithmetic operations occur on registers. All instructions have three register operands.

b. Assume the target is an embedded application utilizing a 16-bit processor (addresses are 16 bits long as are the data operands). Further assume that the Register-Register architecture uses 16 registers. Finally, assume that all opcodes are 8-bits wide. Other than the opcode and operands, instructions do not contain any other information. Instruction length is always a whole number of bytes (ie, 1, 2, 3, etc. bytes). Instruction length is not fixed.

For each of the four architectures:

- How many instruction bytes are fetched?
- Which architecture is more efficient as measured by code size?
- How many data bytes are transferred to and from memory?
- Which architecture is more efficient as measured by memory (instructions + data) traffic?

c. Now, repeat part (b) for an application utilizing a 64-bit computer. That is, addresses and data operands are 64 bits long. The number of registers in the Register-Register architecture remains unchanged at 16. Instruction length is not fixed.

2. (15) You are to increase the number of registers in the MIPS ISA from 32 to 64. (Note that register width remains at 32 bits.)

   a. Draw the three MIPS instruction formats, showing which and how many bits are used for each field. You may not change the number of bits used for the opcode or shamt fields.

   b. Ignoring floating point operations, all R-format instructions (39 of them) use an opcode of 0 with the funct field (bits 5:0) specifying the actual operation (see the 3rd column from the right, Figure A.10.2 in the book). Propose changes to the instruction encoding to accommodate the change to 64 registers while supporting all 39 R-format instructions.

   c. How does your design affect the I- and J-format instructions?

3. (20) The following code fragment processes two arrays and produces a result in register $v0. Assume that each array consists of 2,500 words indexed 0 through 2,499, and that the base addresses of the arrays are stored in $a0 and $a1 respectively, and their sizes (2,500) are stored in registers $a2 and $a3 (also respectively).

```
        sll     $a2 $a2, 2
        sll     $a3, $a3, 2
        add     $v0, $zero, $zero
        add     $t0, $zero, $zero
outer:  add     $t4, $a0, $t0
        lw      $t4, 0($t4)
        add     $t1, $zero, $Zero
inner:  add     $t3, $a1, $t1
        lw      $t3, 0($t3)
        bne     $t3, $t4, skip
        addi    $v0, $v0, 1
skip:   addi    $t1, $t1, 4
        bne     $t1, $a3, inner
        addi    $t0, $t0, 4
        bne     $t0, $a2, outer
```

   a. Add comments to the code and describe in one sentence what is does. Make sure you state the value to be returned in $v0.

   b. Assume that the given code segment is run on a computer with a 2GHz clock that requires the following number of cycles for each instruction:

      add, addi, sll: 1 cycle
      lw, bne: 2 cycles

      In the worst case, how many seconds will take to execute this code?

4. (10) Problem 2.38.4 in the textbook.

5. (10) Problem 2.38.6 in the textbook.

6. (20) You are to design a 64-bit integer adder 3 different ways. Your design should achieve the best performance possible given the components and constraints as described below. In each case you need to provide:

- A diagram of the adder
- A computation showing the adder delay, similar to what was done in Lecture 5.

a. Your design can use 4-bit Carry-lookahead (CLAs) (as many as you need)
b. Your design can use 8-bit CLAs
c. Your design can use 4-bit CLAs, but these CLAs can only be connected to each other in a ripple-carry fashion.