

EE 4363 / CSci 4203 Final Exam – Spring, 2007
(closed book, closed notes, no calculators, no electronic devices)
Be sure to clearly show how you obtain your answers to each question.

1. The IEEE single-precision floating-point format is composed of a 1-bit sign, an 8-bit biased exponent with a bias of 127 and a 23-bit fraction. Consider the following two operands X and Y in this format, which are specified in hex:

$$X = C1080000 \quad Y = C1200000$$

(a) (8 points) Determine the decimal value of each of the operands.

(b) (7 points) Determine the IEEE single-precision floating-point representation (specified in hex) for the product $X*Y$. (Perform the multiplication in decimal and then convert the result into the IEEE format.)

2. The following machine language VeSPA program is placed into the file v.out.

```
58 00 00 55
08 40 00 00
20 82 00 00
F8 00 00 00
```

(a) (8 points) Give the corresponding assembly language program.

(b) (7 points) Give the 32-bit contents (specified in hex) of the PC and registers r0 through r3, inclusive, and the values of the condition code bits C, V, Z and N after each instruction has been executed by behavioral.v.

3. Consider the following portion of a VeSPA assembly language program:

```
add  r14, r15, r16 ; instruction 1
sub  r16, r17, r15 ; instruction 2
or   r14, r16, r17 ; instruction 3
```

(a) (5 points) Identify any flow dependences, anti-dependences and output dependences that exist amongst these instructions. (In each case, specify the register that is involved.)

(b) (5 points) Suppose that instruction 1 is in the instruction fetch (IF) stage of the 5-stage pipelined VeSPA implementation during cycle 21. Determine which cycles instructions 1, 2 and 3 will be in the write back (WB) stage of the pipeline. (Include a table that shows the instructions moving through the pipeline as part of your answer.)

4. (10 points) Consider the following parameter values: memory is byte-addressable, the virtual address is 40 bits, the page size is 32K bytes and each page table entry is 32 bits. Determine the number of bits in the virtual page number field, the number of bits in the page offset field, the number of page table entries and the total size (in Mbytes) of the page table.

5. (15 points) Here is a series of address references given as word addresses: 5, 10, 2, 12, 5, 18, 2, 5, 10, 7, 26, 20, 12, 28. Show the hits and misses and the final cache contents for a 2-way set-associative cache with one-word blocks and a total size of 16 words. Assume that LRU replacement is used.

6. A program has a branch statement. When the program is executed, the outcomes of the branch are as follows (T = taken, N = not taken): N, T, T, N, T, T, T, N, N, T. List the predictions and give the prediction accuracy (expressed as a percentage) for each of the following four branch predictors:

- (a) (2 points) Always taken
- (b) (2 points) Always not taken
- (c) (4 points) 1-bit predictor, initialized to predict taken
- (d) (7 points) 2-bit predictor, initialized to weakly predict taken

7. (20 points) A file called summer.v contains the following:

```
module summer(x, y, z);
input  [2:0] x, y;
output [5:0] z;

wire  [5:0] e, f, g;

assign e = {x[2]&y[0], x[2]&y[0], x[2]&y[0], x[2]&y[0], x[1]&y[0], x[0]&y[0]};
assign f = {x[2]&y[1], x[2]&y[1], x[2]&y[1], x[1]&y[1], x[0]&y[0], 1'b0};
assign g = {x[2]&y[2], x[2]&y[2], x[1]&y[2], x[0]&y[2], 1'b0, 1'b0};
assign z = e + f + g;

endmodule
```

Another file called tbsummer.v contains the following:

```
module tbsummer; // summer testbench

reg  [2:0] x, y;
wire [5:0] z;

integer zval;

// instantiate the summer module
summer s1(x, y, z);

// perform a set of simulations and print the results
initial begin
    x = 3;
    y = 4;
    repeat (3) begin
        #10 zval = -z[5]*32 + z[4:0];
        $display($time, " x = %d , y = %d , z = %d", x, y, zval);
        x = x + 1;
    end
end

endmodule
```

If the command:

```
verilog tbsummer.v summer.v
```

is executed, give the output that will be produced by the \$display statement. Be sure to clearly show how you obtained your answer by including all of the intermediate calculations.

ADD* – Addition

*This instruction sets the condition code bits.

Assembly code notation

- a) ADD rdst, rs1, rs2
- b) ADD rdst, rs1, #immed16

Instruction encoding

a)	31 ... 27 00001	26 ... 22 rdst	21 ... 17 rs1	16 0	15 ... 11 rs2	10 ... 0 000 0000 0000
----	--------------------	-------------------	------------------	---------	------------------	---------------------------

b)	31 ... 27 00001	26 ... 22 rdst	21 ... 17 rs1	16 1	15 ... 0 immed16
----	--------------------	-------------------	------------------	---------	---------------------

HLT – Halt

Assembly code notation

HLT

Instruction encoding

31 ... 27 11111	26 ... 0 000 0000 0000 0000 0000 0000 0000
--------------------	---

LDI – Load Immediate

Assembly code notation

LDI rdst, #value

Instruction encoding

31 ... 27 01011	26 ... 22 rdst	21 ... 0 immed22
--------------------	-------------------	---------------------

NOT – Bit-wise logical complement

Assembly code notation

- a) NOT rdst, rs1

Instruction encoding

31 ... 27 00101	26 ... 22 rdst	21 ... 17 rs1	16 ... 0 0 0000 0000 0000 0000
--------------------	-------------------	------------------	-----------------------------------

AND – Bit-wise logical AND

Assembly code notation

- a) AND rdst, rs1, rs2
- b) AND rdst, rs1, #immed16

Instruction encoding

a)	31 ... 27 00100	26 ... 22 rdst	21 ... 17 rs1	16 0	15 ... 11 rs2	10 ... 0 000 0000 0000
----	--------------------	-------------------	------------------	---------	------------------	---------------------------

b)	31 ... 27 00100	26 ... 22 rdst	21 ... 17 rs1	16 1	15 ... 0 immed16
----	--------------------	-------------------	------------------	---------	---------------------