CSE 471 Autumn 1998

Computer Design and Organization

Midterm

Monday November 9th

NAME : ______________________________

Do all your work on these pages. Do not add any pages. Use back pages if necessary. Show your work to get partial credit.

This exam is worth 50 points. After each question, you will find the number of points it is worth. You should spend approximately x minutes on a question worth x points (e.g., 22 minutes on question 1 worth 22 points). That will leave you with 10 minutes to look over your work.

1. 22 points ____________
2. 8 points ____________
3. 12 points ____________
4. 8 points ____________

1. (22 points)

The 5 stage pipeline that is described in your book has often been called the LUI pipeline where LUI stands for *L*oad-*U*se *I*nterlock. *In this question, we'll assume that branch address calculation is done in stage ID and branch completion in stage EX.*

Another possible 5 stage pipeline is the AGI pipeline where AGI stands for *A*ddress *G*eneration *I*nterlock. The 5 stages of the AGI perform respectively

1. Stage 1 IF Instruction fetch like in LUI
2. Stage 2 ID Instruction decode and register read like in LUI
3. Stage 3 ADGEN Address generation in case we have a Load/Store instruction or a branch
4. Stage 4 EX/MEM Execute (in case of logical/arithmetic instruction and branch completion) or Access memory in case of Load/Store
5. WB Write back like in LUI

(a) (2 points)
Which of the LUI or AGI require more resources (registers and ALUs). Justify your answer.

(b) (5 points)
What is a RAW hazard? Give an example. How can the stalls due to these hazards be avoided in the LUI pipeline when you consider *only arithmetic operations* (be specific)? Can you do the same thing in the AGI pipeline?

(c) (8 points)
Assume that you have implemented all the forwarding that's possible. Consider the two sequences of instructions (a) and (b) below.

```
   (a)                          (b)
lw r1,0(r2)                  add r2,r1,r3
add r2,r1,r3                 lw  r4,0(r2)
```

Is there any stall in LUI for sequence (a)? If so why?

Is there any stall in LUI for sequence (b)? If so why?

Is there any stall in AGI for sequence (a)? If so why?

Is there any stall in AGI for sequence (b)? If so why?

(d) (4 points)
Assume a Branch Not Taken default prediction strategy. How many cycles are lost when you mispredict (i.e., the branch is taken) in each of LUI and AGI.

(e) (3 points)
Explain in a couple of sentences the major differences between LUI and AGI in terms of performance.

2. (8 points)

(a) (4 points)
Define WAR (Write After Read) and WAW (Write After Write) hazards. Give sample sequences of instructions illustrating these hazards (one sequence for WAR and one for WAW).

(b) (2 points)
Can WAR and WAW hazards occur in a single issue, single pipeline machine like the LUI (i.e., the one described in Hennessy & Patterson)? Justify your answer.

(c) (2 points)

WAR and WAW hazards can occur in machines with multiple pipes, single or multiple issue, and various latencies for each pipe. Name a common technique that has been used to solve the WAR and WAW hazards in this context.

3. (12 points)

Microprocessor X has a direct mapped branch target buffer (BTB) of 64 entries where each entry contains a tag and the next PC (30 bits) and no prediction bits. It also has a branch history table (BHT) consisting of 4K entries, where each entry is a 2-bit saturating counter.

Microprocessor X has a 5 stage pipeline. The target address computation is done during the ID stage and the resolution of the branch during the EXE stage.

(a) (1 point)
What is the size of the tag for each entry in the BTB.

(b) (11 points)
Explain how the BTB and BHT are accessed during the IF cycle and EXE cycle. Indicate which bits of the PC you use to access the two structures, when each of the structures is accessed, what information is needed from the processor and what information is returned to the processor etc. (Note that there is not a *single* exact solution to this problem. There are several possible variations).

4. (8 points)

Give the state diagram of the finite state machine described by the following Verilog module:

```
module satc(in,clk,state);
        input in,clk;
        output [1:0]state;
        reg [1:0]state;

        always @(posedge clk)
         begin
         case (state)
          0:  state = (in == 0) ? 1:3;
          1:  state = (in == 0) ? 1:0;
          2:  state = (in == 0) ? 1:3;
          3:  state = (in == 0) ? 2:3;
          default: state = (in == 0) ? 0:2;
        endcase
        end
endmodule
```