CS433 Midterm

Prof Josep Torrellas

October 19, 2017

Time: 1 hour + 15 minutes

Name:

Instructions:

1. This is a closed-book, closed-notes examination.
2. The Exam has 4 Questions. Please budget your time.
3. Calculators are allowed.
4. Please write your answers neatly. Good luck!

Problem No.	Maxm Points	Points Scored
1	30	
2	30	
3	30	
4	30	
Total	120	

1. **Pipelining. Control Hazards [30 points]**

   a. [**3 points**] List the 4 general ways of dealing with branch prediction statically. Hint: one is freezing the pipeline.

   [Answer]
   Freezing pipeline
   Always predict taken
   Always predict nontaken
   Delayed branches

   b. [**2 points**] List one pro and one con of Delayed Branches.

   [Answer]
   Pro: Simple hardware (no need stall pipeline)
   Pro: If can find a useful instruction, no cycle wasted
   Con: If cannot find a useful instruction, waste a cycle

   c. [**2 points**] Suppose we have a simple in-order processor like the one in the book with a 1-delay slot for branches. Consider codes (a) through (c):

```
       ADD R1,R2,R3             ADD R1,R2,R3              ADD R1,R2,R3
       NOP                      NOP                       NOP
       BEQ R4 label             BEQ R1 label              BEQ R1 label
       [                        [                         [
       ADD R7,R7,R10            ADD R7,R7,R10             ADD R7,R9,R10
       JMP end                  JMP end                   JMP end
       NOP                      NOP                       NOP
label: ADD R7,R7,R12     label: ADD R7,R7,R12      label: ADD R7,R9,R12
end:                     end:                      end:

       (a)                      (b)                       (c)
```

   What is the best instruction to put in the delay slot in code (a)? Explain why (in here and in subsequent questions, if no explanation, then no points).

   [Answer]
   ADD R1,R2,R3 because it is a useful, independent instruction

d. **[2 points]** In the code above, what is the best instruction to put in the delay slot in code (b)? Explain why.

[Answer]
Can only put NOP. Cannot put ADD R1,R2,R3 because it has a data dependence with the branch. Cannot put any instruction from the target or fall through path because, in case of misprediction, would destroy a needed value.

e. **[7 points]** In the code above, what is the best instruction to put in the delay slot in code (c) if R2+R3=0 60% of the time? Show the resulting code. In this case: (i) what are the instructions executed when R2+R3=0, and (ii) what are the instructions executed when R2+R3!=0?

[Answer]
Put the first instruction from the target. Be careful that we need to keep the label "label". Resulting code is on the left; code executed when R2+R3=0 is in center; code executed when R2+R3!=0 is on right.

```
       ADD R1,R2,R3    ADD R1,R2,R3     ADD R1,R2,R3
       NOP             NOP              NOP
       BEQ R1 end      BEQ R1 end       BEQ R1 end
       ADD R7,R9,R12   ADD R7,R9,R12    ADD R7,R9,R12
       ADD R7,R9,R10                    ADD R7,R9,R10
       JMP end                          JMP end
       NOP                              NOP
label: ADD R7,R9,R12
end:
```

f. [**7 points**] Repeat the whole previous question if R2+R3=0 40% of the time.

[Answer]

```
       ADD R1,R2,R3        ADD R1,R2,R3     ADD R1,R2,R3
       NOP                 NOP              NOP
       BEQ R1 label        BEQ R1 label     BEQ R1 label
       ADD R7,R9,R10       ADD R7,R9,R10    ADD R7,R9,R10
       JMP end      label: ADD R7,R9,R12    JMP end
       NOP                                  NOP
label: ADD R7,R9,R12
end:
```

g. [**7 points**] If R2+R3=0 50% of the time, which code do you prefer, the one in question e or the one in question f? Why?

[Answer]
The one in question f because the one in question e replicates one instruction.

2. **Software ILP** [**30 points**]
Consider an in-order single issue machine like the one considered in class. There is 1 FP multiplier, taking 8 cycles to perform a multiply, and 1 FP adder, taking 3 cycles to perform an addition. Both are pipelined. Branches are resolved in the ID stage and there is 1 branch delay slot. There is full forwarding, including forwarding from the end of an EX to the MEM stage for stores. Now consider this code fragment:

```
loop L.D    F0, 0(R1)
     L.D    F2, 8(R1)
     MUL.D  F6, F0, F0
     ADD.D  F4, F2, F0
     ADD.D  F6, F6, F4
     S.D    F6, 0(R2)
     DADDUI R1, R1, #16
     DADDUI R2, R2, #8
     DSUBUI R3, R3, #1
     BNEZ   R3, loop
```

A. Extracting ILP

a. [**12 points**] Reschedule the code to minimize stalls. How many stalls are there? Please show the resulting code and explain the stalls.

[Answer]
There are 3 stalls.

```
loop L.D    F0, 0(R1)
     L.D    F2, 8(R1)
     MUL.D  F6, F0, F0
     ADD.D  F4, F2, F0
     DADDUI R1, R1, #16
     DADDUI R2, R2, #8
     DSUBUI R3, R3, #1
     3 STALLS
     ADD.D  F6, F6, F4
     BNEZ   R3, loop
     S.D    F6, -8(R2)
```

b. **[12 points]** Unroll the loop and reschedule the instructions to eliminate all stalls. Only unroll the *minimum* number of times to remove all stalls. How many iterations were unrolled? Explain.

[Answer]
16 cycles, 2 iterations

```
loop L.D    F0, 0(R1)
     L.D    F8, 16(R1)
     MUL.D  F6, F0, F0
     MUL.D  F14, F8, F8
     L.D    F2, 8(R1)
     L.D    F10, 24(R1)
     ADD.D  F4, F2, F0
     ADD.D  F12, F10, F8
     DSUBUI R3, R3, #2
     DADDUI R1, R1, #32
     ADD.D  F6, F6, F4
     ADD.D  F14, F14, F12
     DADDUI R2, R2, #16
     S.D    F6, -16(R2)
     BNEZ   R4, loop
     S.D    F14, -8(R2)
```

B. Short Answer [**6 points**]

a. [**2 points**] What are 2 disadvantages of loop unrolling?

   [Answer]
   Disadvantages: code size increases, register pressure increases

b. [**4 points**] What are 2 differences between dynamically scheduled superscalar and VLIW processors?

   [Answer]
   Superscalar - Issues multiple arbitrary instructions, instructions dynamically schedule.

   VLIW - Issues a fixed number of different types of instructions, instructions packaged together at compile time, if parallel instructions cannot be found, put NOP in its slot.

3. **Branch Prediction [30 points]**

A. Branch Prediction Schemes

Consider the following code with two branches, B1 and B2. R0 always contains 0, and R1 initially contains the memory address of the first element of the array which is initialized to [0, 0, 1, 2, 3, 4, 5, 6, 7, 8]. Assume the memory is byte-addressable and the size of integer is 4 bytes.

```
      ADD   R2 R0 R0
      ADD   R3 R0 R0
LOOP: ADD   R4 R1 R2
      LW    R5 0(R4)
      ANDI  R5 R5 #1
      BEQZ  R5 EVEN      <- B1
      ADDI  R3 R3 #1
EVEN: ADDI  R2 R2 #4
      SUBI  R4 R2 #40
      BNEQZ R4 LOOP      <- B2
```

a. [**4 points**] Explain what the code does. Which values R2 and R3 contain when exiting the loop (i.e., when B2 is not taken)?

[Answer] It counts the number of odd integers in the array. R2 will be 40 and R3 will be the number of odd numbers in the array.

b. [**8 points**] Assume that 2-bit saturating counters are used for branch prediction. Complete the tables below and calculate the prediction accuracy for B1 and B2.

B1:

Step	State	Prediction	Actual Outcome
1	00	N	T
2	01	N	T
3	10	T	N
4	01	N	T
5	10	T	N
6	01	N	T
7	10	T	N
8	01	N	T
9	10	T	N
10	01	N	T

B2:

Step	State	Prediction	Actual Outcome
1	00	N	T
2	01	N	T
3	10	T	T
4	11	T	T
5	11	T	T
6	11	T	T
7	11	T	T
8	11	T	T
9	11	T	T
10	11	T	N

B1 Prediction Accuracy: 0 / 10 = 0%
B2 Prediction Accuracy: 7 / 10 = 70%

c. [**8 points**] Assume the 2-bit prediction scheme explained in the lecture, which needs two consecutive mispredictions to change the prediction. Complete the tables below and calculate the prediction accuracy for B1 and B2. In the tables, 00 means strong-not-taken.

B1:

Step	State	Prediction	Actual Outcome
1	00	N	T
2	01	N	T
3	11	T	N
4	10	T	T
5	11	T	N
6	10	T	T
7	11	T	N
8	10	T	T
9	11	T	N
10	10	T	T

B2:

Step	State	Prediction	Actual Outcome
1	00	N	T
2	01	N	T
3	11	T	T
4	11	T	T
5	11	T	T
6	11	T	T
7	11	T	T
8	11	T	T
9	11	T	T
10	11	T	N

B1 Prediction Accuracy: 4 / 10 = 40%
B2 Prediction Accuracy: 7 / 10 = 70%

B. Branch Target Buffer

a. [**10 points**] Consider a branch target buffer for conditional branches. Assume the followings:

   – The BTB hit rate is 80%.
   – The prediction accuracy for conditional branches in the BTB is 75%.
   – For conditional branches not in the BTB, the branch taken frequency is 55%.
   – A correctly predicted conditional branch has no penalty.
   – If a conditional branch hits in the BTB but is provided a wrong prediction, there is a penalty of two cycles.
   – If a conditional branch misses in the BTB but is taken, there is a penalty of three cycles.

   What is the branch penalty in cycles?

   [Answer]
   Branch Penalty
   = (Hit in BTB and Wrong Prediction) * 2
   + (Miss in BTB and Taken) * 3
   = (80% * 25%) * 2 + (20% * 55%) * 3
   = 0.4 + 0.33 = 0.73 cycles

4. **Tomasulo Algorithm [30 points]**

This problem concerns Tomasulo algorithm with dual issue and hardware speculation. Assume the followings (which is similar to what we had in the homework):

- There is one integer functional unit that takes 1 cycle, one FP Add unit that takes 4 cycles, one FP Multiply unit that takes 5 cycles, and one FP Divide unit that takes 16 cycles.
- Functional units are not pipelined.
- Memory accesses use the integer functional unit to perform effective address calculation.
- Stores access memory during the CM stage while loads access memory during the EX stage.
- Stores do not need the CDB or the WB stage.
- If an instruction moves to the WB stage in cycle x, then an instruction that is waiting for the same functional unit (due to a structural hazard) can start execution in cycle x.
- An instruction waiting for data on the CDB can move to the EX stage in the cycle after the CDB broadcast.
- Only one instruction can write to the CDB in one clock cycle.
- Whenever there is a conflict for a functional unit or the CDB, assume that the oldest (by program order) of the conflicting instructions gets access, while others are stalled.
- The result from the integer functional unit is also broadcast on the CDB and forwarded to dependent instructions through the CDB (just like any floating point instruction).
- There are unlimited reorder buffer entries and reservation stations.
- Two instructions can commit per cycle.
- The BNEZ instruction uses the integer functional unit for its comparison and does not need the WB stage.
- Assume that an instruction after a branch cannot issue in the same cycle as the branch; the earliest it can issue is in the cycle immediately after the branch (to give time to access the branch history table and/or buffer). Any other pair of instructions can issue in the same cycle.
- There is one branch delay slot.

Complete the following table. For each instruction, fill in the cycle numbers for each pipeline stage (CM stands for commit). Then indicate where its source operands are read from (use RF for register file, ROB for reorder buffer, and CDB for common data bus). You do not have to fill entries marked with –. Some entries are filled in for you.

Instruction	IS	Operand 1	Operand 2	EX	WB	CM
L.D F0 0(R1)	1	RF	RF	2	3	4
L.D F6 8(R1)	1	RF	RF	3	4	5
DIV.D F2 F0 F6	2	CDB	CDB	5-20	21	22
ADD.D F4 F2 F6	2	CDB	CDB	22-25	26	27
MUL.D F8 F6 F4	3	CDB	CDB	27-31	32	33
S.D F4 16(R1)	3	–	RF	4	–	33
S.D F8 24(R1)	4	–	RF	5	–	34
DADDUI R1 R1 -32	4	RF	RF	6	7	34
BNEZ R1 target	5	CDB	–	8	–	35
MUL.D F10 F2 F6	6	CDB	RF	22-26	27	35