ECE 565 Computer Architecture

Midterm

Wednesday, Oct 28, 2009

Name: Solution

Limit your answers to the space provided. Unnecessarily long answers will be penalized. If you use more space than is provided, you are probably doing something wrong. Use the back of the page for any scratch work.

Some problems may take longer than others. I have given a time estimate per question. Plan your time accordingly.

Show your work to get partial credit. A correct answer without any explanation to back it up may NOT get any credit.

Please make sure that you have all the 16 pages of the exam.

Problem 1 _____ (out of 12 points)

Problem 2 _____ (out of 16 points)

Problem 3 _____ (out of 12 points)

Problem 4 _____ (out of 32 points)

Problem 5 _____ (out of 9 points)

Problem 6 _____ (out of 13 points)

Problem 7 _____ (out of 6 points)

Total _____ (out of 100 points)

1 Basic Pipelining (12 points) (12 minutes)

1.1 (12 points)

Assuming the following six-stage pipeline:

Fetch | Decode | Execute | Memory | Execute2 | Write-back

In this pipeline, fetch, decode, memory and write-back work exactly as in the five-stage pipeline. Arithmetic instructions that are dependent on an immediately-preceding load instruction pass through Execute without any processing and use Execute2. All other arithmetic instructions and memory instructions use Execute as in the five-stage pipeline.

Assume that the cache access fits in the single cycle of the memory stage and that appropriate bypasses are built into the new pipeline.

1.1.1 (4 points)

In what cases does the new pipeline solve the load-use-stall problem in the five-stage pipeline? Fill in **one** instruction before **or** after the load below to illustrate such a case.

ld r4, 8[r12]
add -, r4, -

1.1.2 (4 points)

In what cases does the new pipeline **not** solve the load-use-stall problem in the five-stage pipeline? Fill in **one** instruction before **or** after the load below to illustrate such a case.

ld r4, 8[r12]
ld -, [r4]

1.1.3 (4 points)

In what cases does the new pipeline incur stalls that do **not** exist in the five-stage pipeline? Fill in **one** instruction before **or** after the code below to illustrate such a case.

ld r2, 0[r24]

add r1, r2, r3

add -, r1, -

2 Performance (16 points) (16 minutes)

2.1 CPI (8 points)

Assume that a processor has two adders each of which takes one cycle for an add and two multipliers each of which takes four cycles for a multiplication. Assuming that 25% of instructions in a program are multiplies and the rest are adds, compute the **range** of CPI (cycles per instruction) that may be achieved by the program, given that there are no delays in the system other than the above operational latencies.

Hint: Think carefully - this question is not about business-as-usual CPI calculation!

All instrs. dependent

$$CPI = \frac{25 \times 4 + 75 \times 1}{100} = 1.75$$

All instrs. independent

$$CPI = \frac{\max\left(\frac{25 \times 4}{2}, \frac{75 \times 1}{2}\right)}{100} = 0.5$$

State the conditions under which the end-points of your range would be achieved.

2.2 Optimal pipeline depth (8 points)

You wish to deepen a given pipeline to increase its clock speed. At 5 stages, each stage has 36 gate-delays of useful work and 1 gate-delay of latch overhead (i.e., the latch adds 1 gate-delay to the clock period). In the 5-stage pipeline, each instruction stalls for 1 cycle, and the average CPI is 2. Assume that the average stall cycles per instruction goes up linearly as you deepen the pipeline (i.e., 6 stages incur 6/5 cycles of stall, 10 stages incur 2 cycles of stall, and so on).

What is the **optimum** number of stages to minimize average execution time of an instruction? In reality, stalls increase more than linearly but that will make the question too hard.

Hint: Count execution time in gate delays.

Clock period $= \left(\frac{180}{n} + 1\right)$

CPI $= \left(1 + n/5\right)$

Time $= \left(\frac{180}{n} + 1\right)\left(1 + n/5\right)$

for minimum time, $n = 30$

3 Short Questions (12 points) (10 minutes)

3.1 (4 points)

In Bhandarkar and Clark's RICS vs. CISC paper, why might the results be questionable? Explain using the Iron Law.

Time = clk period x CPI x #instrs

compilers can affect CPI and #instrs. The paper assumes equal effect for both RISC & CISC compilers which may be untrue.

3.2 (4 points)

In the paper by Agerwala et al. on challenges for the next decade, what is the key reason behind optimal pipeline depth for power-performance (as in energy-delay product) to be shallower than that for performance?

Better clock speed can improve performance at most linearly (usually sub-linear due to memory effects) but power increases cubically. So optimal power-performance depths are a lot less than optimal performance-only depths.

3.3 (4 points)

In Borkar's paper on scaling, compare the clock speed improvement as predicted by scaling to that seen in microprocessors until recently.

Scaling → $\sqrt{2}$ x Real → 2x

the extra $\sqrt{2}$ comes from architecture and circuit optimizations to facilitate deeper pipelines.

4 Dynamic Scheduling (32 points) (35 minutes)

4.1 Renaming (8 points) (10 minutes)

Consider the following code which is dispatched to reservation stations using Tomasulo's renaming implemented using a **physical** register file (PREG) to hold **completed** values and the **architectural** register file (AREG) to hold **committed** values. Assume the following: (1) the instructions occupy physical register 10 onwards, (2) **all** the instructions have been renamed and dispatched, (3) the first two instructions completed before *mult* is fetched, and *sub* and *add* have completed, and the *mult* is **just** about to complete.

ld f6, 34(r2) -- PREG#10, committed

ld f2, 40(r3) -- PREG#11, completed, not committed

mult f4, f2, f4 -- PREG#12, **just** about to complete

sub f7, f6, f2 -- PREG#13, completed

div f1, f4, f6 -- PREG#14

add f4, f7, f2 -- PREG#15, completed

Fill in **all** the empty entries in the **two** tables below.

Reservation Stations								
FU	**Name**	**Busy**	**Op**	**Value$_j$**	**Value$_k$**	**PREG$_j$**	**PREG$_k$**	**PREG#**
1	ld	N	DON'T CARE					
2	Add1	N	DON'T CARE					
3	Add2	N	DON'T CARE					
4	Mul/div1	Y	mul	–	Arch [F4]	P11	–	P12
5	Mul/div2	Y	div	–	Arch [F6]	P12	–	P14

Register Rename Table			
Arch Reg#	**PREG#**	**in PREG**	**in AREG**
f0	–	N	Y
f1	P14	N	N
f2	P11	Y	N
f3	–	N	Y
f4	P5	Y	N
f5	–	N	Y
f6	–	N	Y
f7	P13	Y	N

4.2 Precise interrupts (15 points) (15 minutes)

4.2.1 (3 points)

What are the basic requirements for precise interrupts in terms of state change done by instructions before and after an excepting instruction?

All instructions before the excepting instruction must be complete - must have updated the relevant state.

No instruction after the excepting instruction must have changed any state.

4.2.2 (8 points)

Consider the following code sequence in an out-of-order issue pipeline with physical-register renaming:

```
ld1 r4, r20, r21

ld2 r18,  0[r20]

add r5, r4, r20          # last use of r4

ld3  r11, 0[r12]

sub r4, r5, r6           # overwrites r4
```

ld3 will cause imprecise state but correct execution

Assume that any of ld1, ld2, or ld3 can cause an exception, and that an **erroneous** design results in the freeing of ld1's physical register just after sub is decoded.

Exception at which load may cause not only imprecise state but also incorrect execution?

ld2

Exception at which load will not affect the preciseness of state or the correctness of execution?

ld1

4.2.3 (4 points)

Modern pipelines use the history buffer to hold previous rename register number to enable precise interrupts and branch speculation (as opposed to the previous value as described in Jim Smith's paper). However, there is no need for extra bypasses into the history buffer. Why?

With values, bypasses are needed because the previous value may not be available when the over-writing instruction is dispatched. But with rename register numbers, the previous number is guaranteed to exist because renaming finishes within one cycle.

4.3 Hazards (9 points) (10 minutes)

4.3.1 (3 points)

What hazards does renaming remove?

WAR, WAW

4.3.2 (3 points)

In what key aspect are hazards through memory different than hazards through registers?

Registers → known at decode
Memory → known only after address calculation.

4.3.3 (3 points)

Why is it important to issue loads out of program order while stores proceed in program order?

Later instructions depend on loads. So loads need to be done fast else will cause many stalls.

Stores do not have this problem.

AND
out-of-order stores are hard to fix if there are exceptions.

5 Branch Prediction (9 points) (10 minutes)

5.1 (3 points)

What is the problem with 1-bit counters?

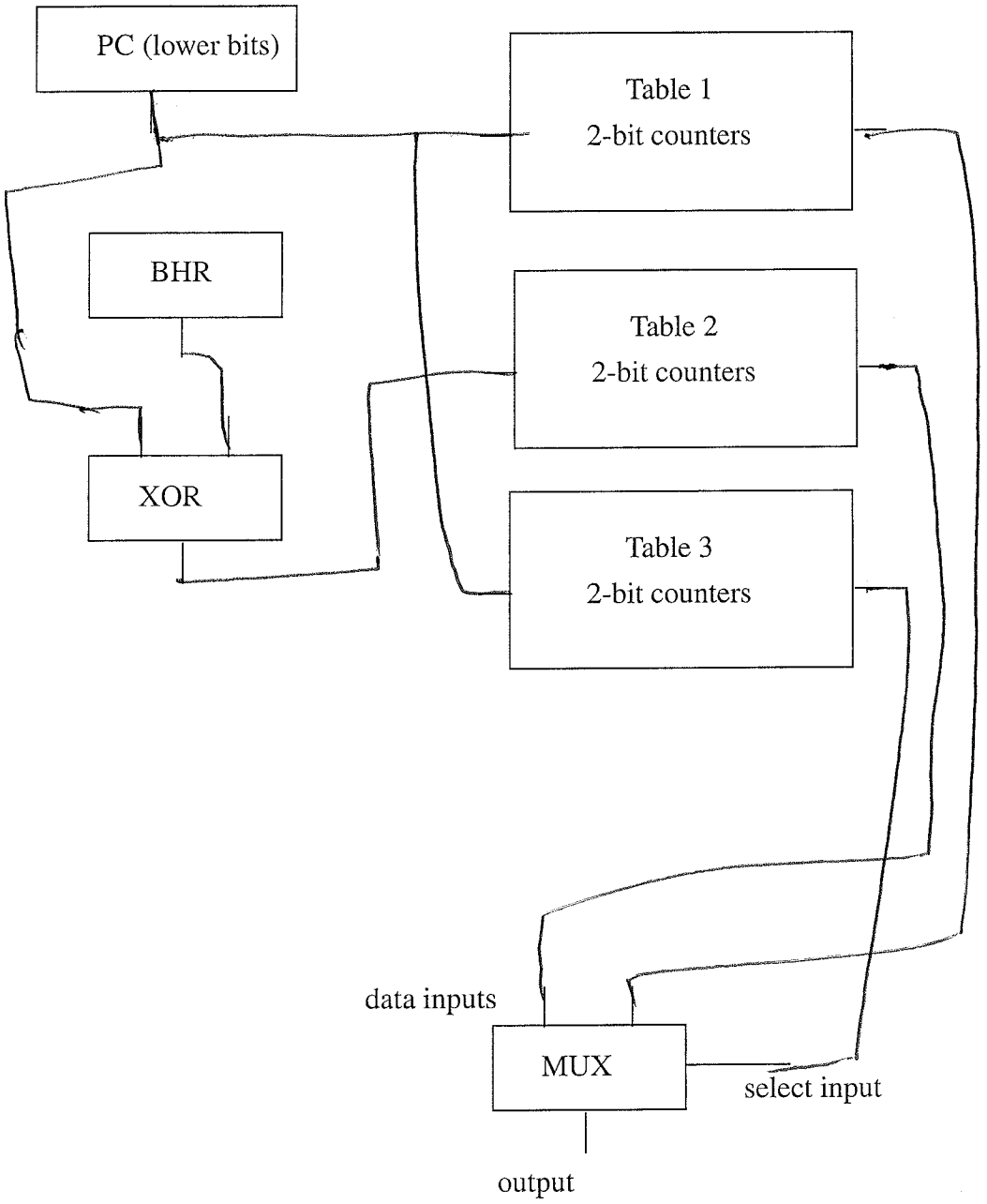
They mispredict at every loop entry & exit.

5.2 (6 points)

The next page shows an incomplete block diagram of a tournament predictor which chooses between two predictors — one that correlates locally and the other that correlates globally.

The following are available to you (next page): Lower-order bits of the PC, a single branch history register (BHR), as many XOR gates as needed, three tables with 2-bit saturating counters, and a multiplexor that picks one of the two data inputs based on the select input.

Draw the required connections among the boxes shown, keeping in mind that a box may be connected to more than one other box.


PC (lower bits)
Table 1
2-bit counters
BHR
Table 2
2-bit counters
XOR
Table 3
2-bit counters
data inputs
MUX
select input
output

6 Compiler Scheduling (13 points) (20 minutes)

6.1 Software Pipelining (8 points) (12 minutes)

Software-pipeline the following loop to maximize performance:

```
for (i=0; i< N; i++)
  sum += A[i]*B[i];      # sum is in register f20
LOOP:
  LD      f0, 0[r1]      # slow
  LD      f2, 0[r2]      # slow
  MUL     f4, f0,f2      # slow
  ADD     f20, f20, f4   # slow
  ADDI    r1, r1, 8      # fast - no need to schedule
  ADDI    r2, r2, 8      # fast - no need to schedule
  BLT     r1, r3, LOOP   # branch if r1 < r3; r3 holds &(A[N])
```

Wrire your answer on the next page.

```
LD    F0, 0[r1]
LD    F2, 0[r2]
LD    F4, 8[r1]
LD    F6, 8[r2]
MUL   F14, F0, F2
addi  r1, r1, 16
addi  r2, r2, 16
```

Loop:

```
ADD   F20, F20, F14
MUL   F14, F4, F6
LD    F4, 0[r1]
LD    F6, 0[r2]
addi  r1, r1, 8
addi  r2, r2, 8
blt   r1, r3, Loop

ADD   F20, F20, F14
MUL   F14, F4, F6
ADD   F20, F20, F14
```

6.2 Trace Scheduling (5 points) (8 minutes)

Trace schedule the following code (assembly is shown below the code) to maximize performance:

```
x = a + c;                    # scalar accesses

if (ptr1 != NULL)  {          # if condition is mostly true

      a = *ptr1 + 4;          # ptr1 load causes stalls

}

      add rx, ra, rc

      beq  rptr1, NULL, AROUND

      ld r2, 0[rptr1]                 # causes stalls

      add ra, r2, 4

AROUND:
```

The compiler knows that (1) rptr2 is usually not equal to NULL but **not always**, (2) ptr1 does **not** refer to x Assume that you have special opcodes for loads and sentinel instructions to handle null-pointer dereferences.

"ld-null reg, addr" flags null-pointer reference.

"sentinel-null reg label" checks for null-pointer dereference.

You must show the scheduled code AND the repair code, if any. Write the answer on the next page.

```
LD_NULL     r2, 0[rptr1]
add         rx, ra, rc
sentinel_null   r3, AROUND
add         ra, r2, 4
```

AROUND:

From the code we see that r2 is a "temporary" and therefore does not need any repair if ld-null fails.

7 Vector/SMT (6 points) (5 minutes)

7.1 (3 points)

Deep superscalar pipelines incur stalls due to data and branch hazards. How do deep vector pipelines avoid such stalls?

vector instructions do not have data or control hazards, by definition.

7.2 (3 points)

How does an SMT pipeline ensure that register dependencies are correctly maintained in the presence of multiple threads?

Each thread gets its own rename table (and a few other small resources).