ICS 233 - Computer Architecture & Assembly Language

Final Exam – Fall 2007

Wednesday, January 23, 2007
7:30 am – 10:00 am

Computer Engineering Department
College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals

Student Name: SOLUTION

Student ID: ______________________________

Q1	/ 10	Q2	/ 20
Q3	/ 15	Q4	/ 15
Q5	/ 20	Q6	/ 25
Total	/ 105		

Important Reminder on Academic Honesty

Using unauthorized information or notes on an exam, peeking at others work, or altering graded exams to claim more credit are severe violations of academic honesty. Detected cases will receive a failing grade in the course.

Q1. (10 pts) Consider the following MIPS code sequence:

```
lw  $5, 100($2)
add $2, $3, $5
sub $5, $5, $2
sw  $5, 100($2)
```

a) (5 pts) Identify all the RAW dependencies between pairs of instructions.

```
lw  $5, 100($2)  and  add $2, $3, $5
lw  $5, 100($2)  and  sub $5, $5, $2
add $2, $3, $5   and  sub $5, $5, $2
add $2, $3, $5   and  sw  $5, 100($2)
sub $5, $5, $2   and  sw  $5, 100($2)
```

b) (3 pts) Identify all the WAR dependencies between pairs of instructions

```
lw  $5, 100($2)  and  add $2, $3, $5
add $2, $3, $5   and  sub $5, $5, $2
```

c) (2 pts) Identify all the WAW dependencies between pairs of instructions

```
lw  $5, 100($2)  and  sub $5, $5, $2
```

Q2. (20 pts) We have a program core consisting of five conditional branches. The program core will be executed millions of times. Below are the outcomes of each branch for one execution of the program core (T for taken and N for not taken).

Branch 1: T-T-T-T-T
Branch 2: N-N-N
Branch 3: T-N-T-N-T-N-T-N
Branch 4: T-T-T-N-N-N
Branch 5: T-T-T-N-T-T-T-N-T

Assume that the behavior of each branch remains the same for each program core execution. For dynamic branch prediction schemes, assume that each branch has its own prediction buffer and each buffer is initialized to the same state before each execution. List the predictions and the accuracies for each of the following branch prediction schemes:

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

a) **Branch 1: prediction = T-T-T-T-T, correct = 5, wrong = 0**
Branch 2: prediction = T-T-T, correct = 0, wrong = 3
Branch 3: prediction = T-T-T-T-T-T-T-T, correct = 4, wrong = 4
Branch 4: prediction = T-T-T-T-T-T, correct = 3, wrong = 3
Branch 5: prediction = T-T-T-T-T-T-T-T-T, correct = 7, wrong = 2

Total correct = 19, Total wrong = 12, Accuracy = 19/31 = 61.3%

b) **Branch 1: prediction = N-N-N-N-N, correct = 0, wrong = 5**
Branch 2: prediction = N-N-N, correct = 3, wrong = 0
Branch 3: prediction = N-N-N-N-N-N-N-N, correct = 4, wrong = 4
Branch 4: prediction = N-N-N-N-N-N, correct = 3, wrong = 3
Branch 5: prediction = N-N-N-N-N-N-N-N-N, correct = 2, wrong = 7

Total correct = 12, Total wrong = 19, Accuracy = 12/31 = 38.7%

c) **Branch 1: prediction = T-T-T-T-T, correct = 5, wrong = 0**
Branch 2: prediction = T-N-N, correct = 2, wrong = 1
Branch 3: prediction = T-T-N-T-N-T-N-T, correct = 1, wrong = 7
Branch 4: prediction = T-T-T-T-N-N, correct = 5, wrong = 1
Branch 5: prediction = T-T-T-T-N-T-T-T-N, correct = 5, wrong = 4

Total correct = 18, Total wrong = 13, Accuracy = 18/31 = 58.1%

d) **Branch 1: prediction = T-T-T-T-T, correct = 5, wrong = 0**
Branch 2: prediction = T-N-N, correct = 2, wrong = 1
Branch 3: prediction = T-T-T-T-T-T-T-T, correct = 4, wrong = 4
Branch 4: prediction = T-T-T-T-T-N, correct = 4, wrong = 2
Branch 5: prediction = T-T-T-T-T-T-T-T-T, correct = 7, wrong = 2

Total correct = 22, Total wrong = 9, Accuracy = 22/31 = 71%

Q3. (15 pts) Consider a **direct-mapped** cache with **128 blocks**. The block size is **32 bytes**.

a) (3 pts) Find the number of tag bits, index bits, and offset bits in a 32-bit address.

Offset bits = 5

Index bits = 7

Tag bits = 32 – 12 = 20 bits

b) (4 pts) Find the number of bits required to store all the valid and tag bits in the cache.

Total number of tag and valid bits = 128 * (20 + 1) = 2688 bits

c) (8 pts) Given the following sequence of address references in decimal:

20000, 20004, 20008, 20016, 24108, 24112, 24116, 24120

Starting with an **empty cache**, show the **index** and **tag** for each address and indicate whether a hit or a miss.

Address = Hex	**Offset (5 bits)**	**Index (7 bits)**	**Tag**	**Hit or Miss**
20000 = 0x4E20	0x00 = 0	0x71 = 113	4	Miss (initially empty)
20004 = 0x4E24	0x04 = 4	0x71 = 113	4	Hit
20008 = 0x4E28	0x08 = 8	0x71 = 113	4	Hit
20016 = 0x4E30	0x10 = 16	0x71 = 113	4	Hit
24108 = 0x5E2C	0x0C = 12	0x71 = 113	5	Miss (different tag)
24112 = 0x5E30	0x10 = 16	0x71 = 113	5	Hit
24116 = 0x5E34	0x14 = 20	0x71 = 113	5	Hit
24120 = 0x5E38	0x18 = 24	0x71 = 113	5	Hit

Q4. (15 pts) A processor runs at 2 GHz and has a CPI of 1.2 without including the stall cycles due to cache misses. Load and store instructions count 30% of all instructions.

The processor has an I-cache and a D-cache. The hit time is 1 clock cycle. The I-cache has a 2% miss rate. The D-cache has a 5% miss rate on load and store instructions.

The miss penalty is 50 ns, which is the time to access and transfer a cache block between main memory and the processor.

a) (3 pts) What is the average memory access time for instruction access in clock cycles?

Miss penalty = 50 ns * 2 GHz = 100 clock cycles

AMAT = hit time + miss rate * miss penalty = 1 + 0.02 * 100 = 3 clock cycles

b) (3 pts) What is the average memory access time for data access in clock cycles?

AMAT = 1 + 0.05 * 100 = 6 clock cycles

c) (4 pts) What is the number of stall cycles per instruction and the overall CPI?

Stall cycles per instruction = 1 * 0.02 * 100 + 0.3 * 0.05 * 100 = 3.5 cycles

Overall CPI = 1.2 + 3.5 = 4.7 cycles per instruction

d) (5 pts) You are considering replacing the 2 GHz CPU with one that runs at 4 GHz, but is otherwise identical. How much faster does the new processor run? Assume that hit time in the I-cache and the D-cache is 1 clock cycle in the new processor, and the time to access and transfer a cache block between main memory and the processor is still 50 ns.

For the new processor running at 4 GHz:

Miss penalty = 50 ns * 4 GHz = 200 clock cycles

Stall cycles per instruction:

(1 * 0.02 + 0.3 * 0.05) * 200 = 7 cycles

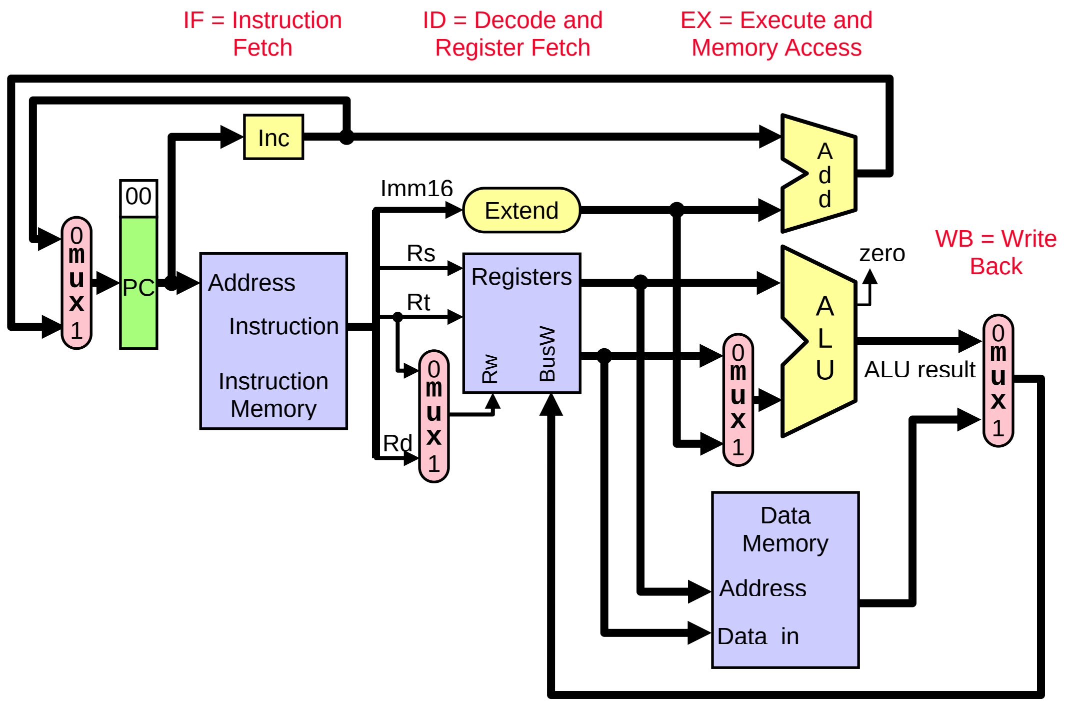
Overall CPI = 1.2 + 7 = 8.2 cycles per instruction

Speedup = (CPI_c / CPI_d) * (Clock $Rate_d$ / Clock $Rate_c$) = (4.7 / 8.2) * (4/2) = 1.146

Q5. (20 pts) Consider the following idea: we want to modify all **load** and **store** instructions in the instruction set such that the offset is always 0. The **load** and **store** instructions can be of the R-type and there is NO need for the ALU to compute the memory address. This means that all load and store instructions will have the following format, where **Rs** is the register that contains the memory address.

```
LW  Rt, (Rs)    # No immediate constant used
SW  Rt, (Rs)    # No immediate constant used
```

a) (10 pts) Draw the modified **single-cycle** datapath. **Identify the changes** that you are making to the single-cycle datapath.



b) (4 pts) Assume that the operation delays for the major components are as follows:

Instruction Memory: 200 ps

Data Memory: 200 ps

ALU: 150 ps

Register file (read or write): 100 ps

Ignore the delays in the multiplexers, control, PC access, extension logic, and wires.

What is the cycle time for the single-cycle datapath BEFORE and AFTER making the modification?

BEFORE making the modification:

Cycle time = 200 + 100 + 150 + 200 + 100 = 750 ps

AFTER making the modification:

Cycle time = 200 + 100 + max(150, 200) + 100 = 200 + 100 + 200 + 100 = 600 ps

c) (6 pts) Because we have removed the offset in all load and store instructions, all original load-store instructions with non-zero offsets would now require an additional **ADDI** instruction to compute the address. This will increase the instruction count.

Suppose we have a program in which 20% of the instructions are load-store instructions. Assume further that only 10% of the original load-store instructions have a non-zero offset and would require an additional **ADDI** instruction to compute the address.

What is the percent increase in the instruction count when additional **ADDI** instructions are used?

Percent increase in the instruction count = 20% * 10% = 2% (for additional ADDI)

Which design is better, the original one that allowed non-zero offsets, or the modified one with zero offsets, and why?

Execution Time = Instruction Count * CPI * Clock Cycle

CPI = 1 in both cases because this is single-cycle design

Original Design Execution Time = I-Count * 1 * 750 ps = 750 I-Count

Modified Design Execution Time = 1.02 I-Count * 1 * 600 ps = 612 I-Count

Modified Design is better because it takes less time to execute program

What is the speedup factor?

Speedup factor = 750 / (600*1.02) = 1.225

Q6. (25 pts) Use the following MIPS code fragment:

```
I1:   ADDI    $3, $0, 100          # $3 = 100
I2:   ADD     $4, $0, $0           # $4 = 0
Loop:
I3:   LW      $5, 0($1)            # $5 = MEM[$1]
I4:   ADD     $4, $4, $5           # $4 = $4 + $5
I5:   LW      $6, 0($2)            # $6 = MEM[$2]
I6:   SUB     $4, $4, $6           # $4 = $4 – $6
I7:   ADDI    $1, $1, 4            # $1 = $1 + 4
I8:   ADDI    $2, $2, 4            # $2 = $2 + 4
I9:   ADDI    $3, $3, -1           # $3 = $3 – 1
I10:  BNE     $3, $0, Loop         if ($3 != 0) goto Loop
```

a) (10 pts) Show the timing of one loop iteration on the 5-stage MIPS pipeline **without forwarding hardware**. Complete the timing table, showing all the stall cycles. Assume that the branch will stall the pipeline for 1 clock cycle only.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
I1: ADDI	**IF**	**ID**	**EX**	**M**	**WB**																				
I2: ADD		**IF**	**ID**	**EX**	**M**	**WB**																			
I3: LW			**IF**	**ID**	**EX**	**M**	**WB**																		
I4: ADD				**IF**	stall	stall	**ID**	**EX**	**M**	**WB**															
I5: LW					2 stall	cycles	**IF**	**ID**	**EX**	**M**	**WB**														
I6: SUB								**IF**	stall	stall	**ID**	**EX**	**M**	**WB**											
I7: ADDI									2 stall	cycles	**IF**	**ID**	**EX**	**M**	**WB**										
I8: ADDI												**IF**	**ID**	**EX**	**M**	**WB**									
I9: ADDI													**IF**	**ID**	**EX**	**M**	**WB**								
I10: BNE														**IF**	stall	stall	**ID**								
I3: LW															2 stall	cycles	**IF**	**IF**	**ID**	**EX**	**M**	**WB**			
I4: ADD																	**1 delay**	**cycle**	**IF**	stall	stall	**ID**	**EX**	**M**	**WB**

$\longleftarrow$ **Time of one loop iteration = 15 cycles** $\longrightarrow$

b) (5 pts) According to the timing diagram of part **(a)**, compute the number of clock cycles and the average CPI to execute ALL the iterations of the above loop.

There are 100 iterations
Each iteration requires 15 cycles =
8 cycles to start the 8 instructions in loop body + 7 stall cycles
There are 2 additional cycles to start the first 2 instructions before the loop.
Therefore, total cycles = 100 * 15 + 2 (can be ignored) = 1502 cycles ≈ 1500 cycles
Total instruction executed = 2 + 8 * 100 = 802 instructions (counting first two)
Average CPI = 1502 / 802 = 1.87
If we ignore first two instructions and the time to terminate last iteration then
Average CPI = 1500/800 = 1.88 (almost same answer)

c) (5 pts) Reorder the instructions of the above loop to fill the load-delay and the branch-delay slots, without changing the computation. Write the code of the modified loop.

```
    ADDI    $3, $0, 100     # $3 = 100
    ADD     $4, $0, $0      # $4 = 0
Loop:
    LW      $5, 0($1)       # $5 = MEM[$1]
    LW      $6, 0($2)       # Moved earlier to avoid load-delay
    ADDI    $3, $3, -1      # Moved earlier
    ADD     $4, $4, $5      # $4 = $4 + $5
    ADDI    $1, $1, 4       # $1 = $1 + 4
    ADDI    $2, $2, 4       # $2 = $2 + 4
    BNE     $3, $0, Loop    # if ($3 != 0) goto Loop
    SUB     $4, $4, $6      # Fills branch delay slot
```

Other re-orderings are possible as long as we avoid the load delay and we fill branch delay slot with an independent instruction. We should be able to reduce the stall cycles to 0.

d) (5 pts) Compute the number of cycles and the average CPI to execute ALL the iteration of the modified loop. What is the speedup factor?

There are 100 iterations
Each iteration requires 8 cycles =
8 cycles to start the 8 instructions in loop body + 0 stall cycles
There are 2 additional cycles to start the first 2 instructions before the loop
+ 4 additional cycles to terminate the ADDI instruction in the last iteration.
Therefore, total cycles = 100 * 8 + 6 (can be ignored) = 806 cycles ≈ 800 cycles
Total instruction executed = 2 + 8 * 100 = 802 instructions (counting first two)
Average CPI = 806 / 802 = 1.00
If we ignore first two instructions and the time to terminate last iteration then
Average CPI = 800/800 = 1.00 (almost same answer)
Speedup Factor = $CPI_{part\text{-}b}/CPI_{part\text{-}d}$ = 1.88/1.00 = 1.88