COE 308 – Computer Architecture

Exam II – Fall 2008

Monday, January 12, 2009

7:00 – 9:00 pm

Computer Engineering Department

College of Computer Sciences & Engineering

King Fahd University of Petroleum & Minerals

Student Name: SOLUTION

Student ID: ______________________

Q1	/ 15	Q2	/ 15
Q3	/ 15	Q4	/ 25
Q5	/ 15	Q6	/ 15
Total	/ 100		

Important Reminder on Academic Honesty

Using unauthorized information 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.

Prepared by Dr. Muhamed Mudawar

Q1. (15 pts) Using the refined division hardware, show the **unsigned** division of:

Dividend = `11011001` by Divisor = `00001010`

The result of the division should be stored in the Remainder and Quotient registers. Eight iterations are required. Show your steps.

Iteration	Remainder	Quotient	Divisor	Difference
0: Initialize	`00000000`	`11011001`	`00001010`	
1: SLL, Diff	`00000001`	`10110010`	`00001010`	`< 0`
2: SLL, Diff	`00000011`	`01100100`	`00001010`	`< 0`
3: SLL, Diff	`00000110`	`11001000`	`00001010`	`< 0`
4: SLL, Diff	`00001101`	`10010000`	`00001010`	`00000011`
4: Rem = Diff	`00000011`	`10010001`		
5: SLL, Diff	`00000111`	`00100010`	`00001010`	`< 0`
6: SLL, Diff	`00001110`	`01000100`	`00001010`	`00000100`
6: Rem = Diff	`00000100`	`01000101`		
7: SLL, Diff	`00001000`	`10001010`	`00001010`	`< 0`
8: SLL, Diff	`00010001`	`00010100`	`00001010`	`00000111`
8: Rem = Diff	`00000111`	`00010101`		

Check:

Dividend = 11011001_2 = 217 (unsigned)

Divisor = 00001010_2 = 10

Quotient = 00010101_2 = 21 and Remainder = 00000111_2 = 7

Q2. (15 pts) A program, being executed on a processor, has the following instructions mix:

Operation	**Frequency**	**Clock cycles per instruction**
ALU	40 %	2
Load	20 %	10
Store	15 %	4
Branches	25 %	3

a) (3 pts) Compute the average clock cycles per instruction

Average CPI$_a$ = 0.4*2 + 0.2*10 + 0.15*4 + 0.25*3 = 4.15

b) (5 pts) Compute the percent of execution time spent by each class of instructions

Operation	**Frequency**	**CPI**	**CPI * Frequency**	**% Execution Time**
ALU	40 %	2	0.8	0.8 / 4.15 = 19.3%
Load	20 %	10	2.0	2.0 / 4.15 = 48.2%
Store	15 %	4	0.6	0.6 / 4.15 = 14.4%
Branches	25 %	3	0.75	0.75 / 4.15 = 18.1%

c) (5 pts) A designer wants to improve the performance. He designs a new execution unit that makes 80% of ALU operations take only **1** cycle to execute. The other 20% of ALU operations will still take **2** cycles to execute. The designer also wants to improve the execution of the memory access instructions. He does it in a way that **95%** of the **load** instructions take only **2** cycles to execute, while the remaining **5%** of the **load** instructions take **10** cycles to execute per **load**. He also improves the store instructions in such a way that each **store** instruction takes **2** cycles to execute.

Compute the new average CPI

```
Average CPIc =   0.8*0.4*1 + 0.2*0.4*2 +
                 0.2*0.95*2 + 0.2*0.05*10 +
                 0.15*2 + 0.25*3 = 2.01
```

d) (2 pts) What is the speedup factor by which the performance has improved in part **c**?

Speedup = 4.15 / 2.01 = 2.06 (I-count & clock are the same)

Q3. (15 pts) The following code fragment processes two double-precision floating-point arrays *A* and *B*, and produces an important result in register **`$f0`**. Each array consists of **`10000`** double words. The base addresses of the arrays *A* and *B* are stored in **`$a0`** and **`$a1`** respectively.

```
       ori     $t0, $zero, 10000
       sub.d   $f0, $f0, $f0
loop:  ldc1    $f2, 0($a0)
       ldc1    $f4, 0($a1)
       mul.d   $f6, $f2, $f4
       add.d   $f0, $f0, $f6
       addi    $a0, $a0, 8
       addi    $a1, $a1, 8
       addi    $t0, $t0, -1
       bne     $t0, $zero, loop
```

a) (3 pts) Count the total number of instructions executed by all the iterations (including those executed outside the loop).

`Instruction Count = 2 + 10000 * 8 = 80002`

Assume that the code is run on a machine with a **2 GHz** clock that requires the following number of cycles for each instruction:

Instruction	Cycles
addi, ori	1
ldc1	3
add.d, sub.d	5
mul.d	6
bne	2

b) (5 pts) How many cycles does it take to execute the above code?

`Clock cycles = 1 (ori) + 5 (sub.d) + 10000 * (2*3 (ldc1) + 6 (mul.d) + 5 (add.d) + 3*1 (addi) + 2 (bne))`

`= 6 + 10000 * 22 = 220006 cycles`

c) (3 pts) What is the execution time in nanoseconds?

`Execution time = cycles/clock rate = 220006/2 = 110003 nsec`

d) (2 pts) What is the average CPI for the above code?

`Average CPI = Clock Cycles / Instruction-Count =`

`Average CPI = 220006 / 80002 = 2.75`

e) (2 pts) What is the MIPS rate for the above code?

`MIPS rate = 80002 / 110 usec = 727.3 MIPS`

Q4. (25 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 addressing mode is register indirect only). This means that all load and store instructions will have the following format, where **Rs** is the register that contains the memory address.

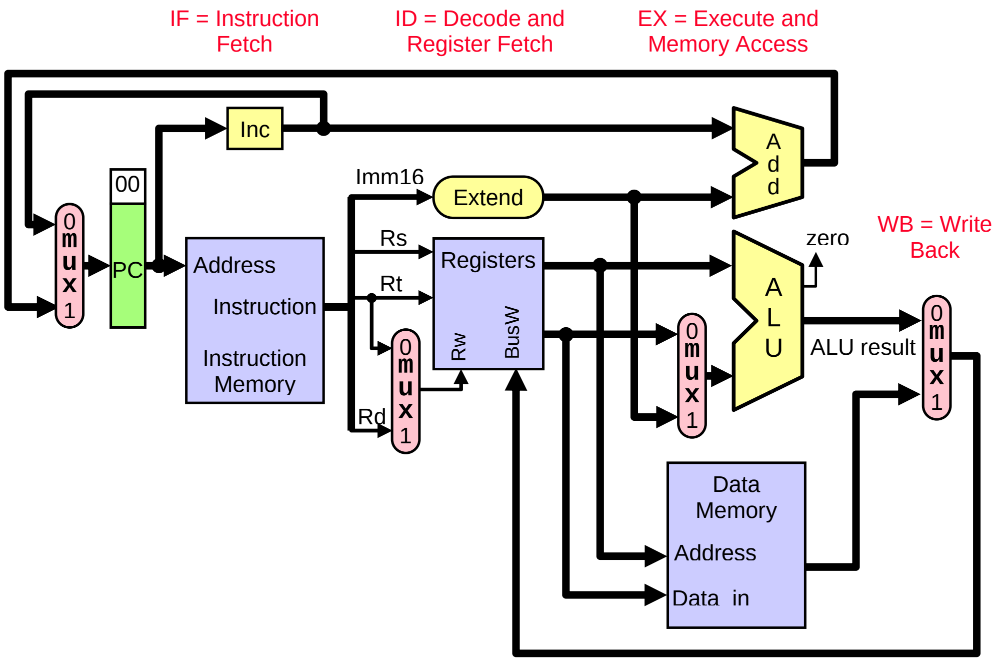
```
LW   Rt, (Rs)     # No immediate constant used
SW   Rt, (Rs)     # No immediate constant used
```

a) (10 pts) Draw the modified single-cycle datapath (rotate the page for wider drawing). Identify the changes that you are making to the single-cycle (non-pipelined) datapath.

Solution:

The major change is moving the Data Memory access to be done in parallel with the execute stage. There is no need for the ALU to do address calculation. So, Memory Access is done just after fetching the address from the register Rs and the data from register Rt.

The modified single-cycle datapath is shown below.



b) (7 pts) Assume that the operation delays for the major components are as follows:

Instruction Memory: 200 ps

Data Memory: 200 ps

ALU: 100 ps

Register file (read or write): 50 ps

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

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

BEFORE making the modification:

Cycle time = 200 + 50 + 100 + 200 + 50 = 600 ps

AFTER making the modification:

Cycle time = 200 + 50 + max(100, 200) + 50 = 200 + 50 + 200 + 50 = 500 ps

c) (8 pts) Because we have reduced the offset to zero 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 = IC * 1 * 600 ps = 600 IC

Modified Design Execution Time = 1.02 IC * 1 * 500 ps = 510 IC

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

What is the speedup factor?

Speedup factor = 600 / 510 = 1.176

Q5. (15 pts) Consider the following MIPS code sequence:

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

a) (6 pts) Identify all pairs of instructions that have RAW dependencies

```
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 pairs of instructions that have WAR dependencies

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

c) (2 pts) Identify all pairs of instructions that have WAW dependencies

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

d) (4 pts) Rewrite the above instructions to eliminate all WAR and WAW dependencies

Rename register **$2** as **$6** (or other unique name) in **add**, **sub**, and **sw**.
Rename register **$5** as **$7** (or other unique name) in **sub** and **sw**

```
lw  $5, 100($2)
add $6, $3, $5
sub $7, $5, $6
sw  $7, 100($6)
```

Q6. (15 pts) Consider the following MIPS assembly language code:

```
I1: ADD $4, $1, $0
I2: SUB $9, $3, $4
I3: ADD $4, $5, $6
I4: LW  $2, 100($3)
I5: LW  $2, 0($2)
I6: SW  $2, 100($4)
I7: AND $2, $2, $1
```

a) (8 pts) Show the timing of one loop iteration on a 5-stage (IF, ID, EX, MEM, WB) pipeline **without forwarding hardware**. Complete the timing table, showing all the stall cycles that are caused by data hazards. Label all stall cycles (Draw an X in the box). Compute the average CPI for the above code fragment.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1: ADD	**IF**	**ID**	**EX**	**M**	**WB**															
I2: SUB		**IF**	**X**	**X**	**ID**	**EX**	**M**	**WB**												
I3: ADD					**IF**	**ID**	**EX**	**M**	**WB**											
I4: LW						**IF**	**ID**	**EX**	**M**	**WB**										
I5: LW							**IF**	**X**	**X**	**ID**	**EX**	**M**	**WB**							
I6: SW										**IF**	**X**	**X**	**ID**	**EX**	**M**	**WB**				
I7: AND													**IF**	**ID**	**EX**	**M**	**WB**			

Total cycles = 17 (including time to fill the pipeline)

Average CPI = 17 / 7 = 2.428

b) (7 pts) Repeat part (a) on a pipeline that **supports forwarding**. Label all data forwards that the forwarding unit detects with an arrow between the stage handing off the data and the stage receiving the data. Compute the average CPI for the above code fragment.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1: ADD	**IF**	**ID**	**EX**	**M**	**WB**															
I2: SUB		**IF**	**ID**	**EX**	**M**	**WB**														
I3: ADD			**IF**	**ID**	**EX**	**M**	**WB**													
I4: LW				**IF**	**ID**	**EX**	**M**	**WB**												
I5: LW					**IF**	**X**	**ID**	**EX**	**M**	**WB**										
I6: SW							**IF**	**ID**	**EX**	**M**	**WB**									
I7: AND								**IF**	**ID**	**EX**	**M**	**WB**								

Total cycles = 12 (including time to fill the pipeline)

Average CPI = 12 / 7 = 1.714