Sample Exam 1 Solutions

CIS 501, Spring 1998
Sample exam

1. Consider a workstation with no floating-point hardware which has to emulate FP operations with sequences of integer operations. Its clock cycle time is 10 ns. The following table lists the FP operation types and their relative frequencies, as well as the number of integer instructions needed to emulate each type of FP operation.

FP Operation	**Frequency**	**# Integer instr to emulate**
Add	50%	6
Multiply	40%	10
Divide	10%	20

Now consider adding a floating-point coprocessor to the workstation which allows FP operations to be performed directly (i.e. without emulation). However, adding the coprocessor will increase the cycle time by 20%. Consider a benchmark workload of 150 million instructions, 20% of which are FP operations and 80% of which are integer instructions. Assume a base CPI of 1 for integer instructions (and a CPI of 1 for FP instructions executed by the coprocessor).

(a) (6 points) Calculate the MIPS rating for each machine.

MIPS = $\frac{1}{CPI \times \text{cycle time} \times 10^6}$

MIPS$_{old}$ = $\frac{1}{1 \times 10 \times 10^{-9} \times 10^6}$ = 100 MIPS

MIPS$_{old}$ = $\frac{1}{1 \times 12 \times 10^{-9} \times 10^6}$ = 83.3 MIPS

(b) (6 points) Calculate the speedup from adding the coprocessor by comparing execution times.

Execution time = $\frac{IC}{MIPS \times 10^6}$

Speedup = $\frac{\text{Exec time}_{old}}{\text{Exec time}_{new}} = \frac{\text{IC}_{old}}{\text{IC}_{new}} \cdot \frac{\text{MIPS}_{new}}{\text{MIPS}_{old}} = \frac{.8 \times 150 + (.2)(150)(.5 \times 6 + .4 \times 10 + .1 \times 20)}{150} \cdot \frac{83.3}{100} = 2.17$

(c) (6 points) Assuming the coprocessor could be added without increasing the cycle time, calculate the speedup from adding it using Amdahl's Law.

there are $(.8 \times 150 + (.2)(150)(.5 \times 6 + .4 \times 10 + .1 \times 20)) = 390$ million instructions on the old machine.

% unenhanced = $\frac{.8 \times 150}{390}$ = .31, % enhanced = 1 - .31 = .69

on average, each FP takes .5(6) + .4(10) + .1(20) isntructions to emulate

Speedup = $\frac{1}{.31 + .69/9}$ = 2.6

2. Consider the standard DLX FP pipeline (i.e. with a pipelined floating point adder with a 4 cycle latency, and a pipelined FP multiplier with a 7 cycle latency). Furthermore, assume the pipeline uses delayed branches with a branch delay slot, and that branch resolution has been moved up into the ID stage. Thus the branch target may begin its IF stage at the same time the branch is beginning its EX stage. Consider the following segment of code:

```
loop: LD    F0, 0(R2)
      ADDI  R2, R2, #8
      MULTD F0, F0, F4
      SUB   R5, R4, R2
      BNZ   R5, loop
      ADDD  F2, F0, F2    ; branch delay slot
```

(a) (10 points) Show the timing for one iteration of this loop, assuming that no forwarding is available, but that a register fetch done in the same cycle as a write back will retrieve the value just written.

	clock cycle																			
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LD F0,0(R2)	F	D	X	M	W															
ADDI R2, R2, #8		F	D	X	M	W														
MULTD F0, F0, F4			F	-	D	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M	W						
SUB R5, R4, R3					F	D	X	M	W											
BNZ R5, loop						F	-	-	D	X	M	W								
ADDD F2, F0, F2									F	-	-	-	-	D	A_1	A_2	A_3	A_4	M	W
LD F0,0(R2)														F	D	X	M	W		

(b) (10 points) Show the timing for one iteration of ths loop, assuming that all possible forwarding is available (and again that a register fetch done in the same cycle as a write back will retrieve the value just written).

	clock cycle																			
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LD F0,0(R2)	F	D	X	M	W															
ADDI R2, R2, #8		F	D	X	M	W														
MULTD F0, F0, F4			F	D	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M	W							
SUB R5, R4, R3				F	D	X	M	W												
BNZ R5, loop					F	-	D	X	M	W										
ADDD F2, F0, F2							F	D	-	-	-	A_1	A_2	A_3	A_4	M	W			
LD F0,0(R2)								F	-	-	-	D	X	M	W					

(c) (4 points) Explain why this loop cannot be rescheduled to produce fewer stalls per iteration.

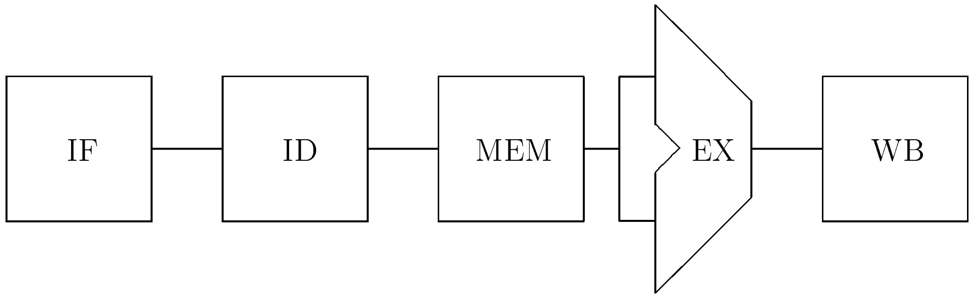
We currently have 4 stalls.

The following instructions must be carried out in order :

```
LD      F0, 0(R2)
             latency 1
MULTD   F0, F0, F4
             latency 6
ADDD    F2, F0, F2
```

Total latency is 7; we have at most three instructions to schedule in these latent slots (`ADDI, SUB, BNZ`), which still leaves 7-3 = 4 stalls.

3. We are designing a new machine, the Eniac-2001, and want to eliminate the load stalls found in the DLX. To achieve this, we decide to only support the register indirect addressing mode for loads and stores. This change lets us exchange the order of the EX and MEM stages in the pipeline (see diagram) and thus eliminate the load stall. The rest of the pipe is the same as the unoptimized DLX pipeline you implemented in the project.



(a) (5 points) Show how you would synthesize a register displacement instruction like `LW R1, 200(R3)` on the Eniac-2001.

```
ADDI    R4, R3, #200
LW      R1, (R4)
```

(b) (5 points) How many cycles does it take to execute the instructions used for the synthesis?

```
ADDI    F   D   M   X   W
LW          F   -   D   X   M   W
```

2 cycles + 1 stall = 3 cycles in all

(c) (7 points) Explain how this change to the pipeline affects control hazards.

Branches are still resolved in the EX stage, so the branch penalty has increased to 4.

(d) Explain how this change to the pipeline affects data forwarding/stalls for:

i. (6 points) ALU ops following ALU ops
No change. We still forward the result of the EX stage back into the EX stage.

ii. (6 points) Load/stores following ALU ops
A load or store using the result of an ALU op as an address will require one stall and a forward. A store using the result of an ALU op as the value to be stored will also require a stall and a forward.

iii. (6 points) ALU ops following Loads
Since MEM is now before Ex in the pipeline, this no longer requires a stall (but we still need to forward).

(Question 3, continued)

(e) (5 points) Explain how you could determine if the new pipeline would have improved performance.

It would depend on whether we have increased or decreased the overall stall count (since we have added some new stalls but eliminated some others). The simplest way would be to compare performance on a benchmark suite of typical programs.

4. Suppose we now also wanted to further change the Eniac-2001 from Problem 3 to once more support a displacement addressing mode for loads and stores by including an adder in the ID stage for effective address calculation. Call this machine the Eniac-2001++.

(a) (9 points) How would the data forwarding of the Eniac-2001++ be different from the Eniac-2001?

We need the value of the base register in the ID to calculate the effective address, thus we need to forward the results of the MEM and WB stages into the ID stage.

(b) (9 points) Would the performance of the Eniac-2001++ be better than that of the Eniac-2001? Why or why not?

The adder might require us to slow down the clock, but if it didn't we would probably get a performance win. Consider the instruction `LW R1, 200(R3)`.

Eniac-2001

```
ADDI    R4, R3, #200
LW      R1, (R4)
```

That costs us 3 cycles.

Eniac-2001++ Here we have 3 cases.

worst case : R3 is calculated in the immediately preceding isntruction by an ALU op. We have 2 stalls in this case, so it costs us 3 cycles.

next : R3 calculated by immediately preceding LW. This gives us 1 stall, hence costs 2 cycles.

best : no data hazards => no stalls, 1 cycle for the LW.

Thus we never do worse than the Eniac-2001, and sometimes do better.