

EE282H Midterm

Open Book
(Total time = 120 minutes, Total points = 120)

Name: (please print) _____ **SOLUTIONS** _____

I agree to abide by the terms of the Honor Code while taking this exam.

Signature: _____

GOOD LUCK!

Points: 1. (40) _____

2. (40) _____

3. (40) _____

Total: (120) _____

Question 1.0 Short Answer (40 points)

1.a (10 points)

What are the pros and cons of delayed branches?

Pros:

- + No implementation cost => natural behavior of the pipeline
- + Reduces CPI increase from branch delay => improves performance over stalling

Cons:

- Exposes part of ISA to the programmer which makes future changes to the architecture difficult to implement. Does not work well with dynamic branch prediction.
- Requires compiler scheduling to fill the branch delay slot
- Only works well with shallow pipelines => difficult to fill more than one delay slot
- Requires extra PCs to be saved to return from interrupts.

1.b (10 points)

Why is the VAX instruction set architecture difficult to pipeline? If you wanted to design a pipeline for the VAX ISA, how would you do it?

Variable length instructions make it difficult to fetch a new instruction every cycle. High variation in the amount of work per instruction and variety of complex addressing modes make a fixed pipeline that executes all instructions in a fully pipelined fashion complex to design.

The best way to pipeline the VAX ISA is to pipeline at the microinstruction level (done in VAX 8700). Microinstructions are much more regular (RISC instructions are like microinstructions) and can be pipelined much more easily than at the macroinstruction level.

1.c (10 points)

What are the advantages of an ID vs. EX control point? What are the disadvantages of the ID control point? Given an EX control point with always not-taken branch prediction, list all the changes you must make to the pipeline to implement the ID control point.

Advantages:

+ Reduces branch delay

Disadvantages:

-Requires more hardware because the ALU can not be used to compute the branch target

- Extra work required in ID stage will likely jeopardize the clock cycle time

Implementation:

Add PC-adder to ID stage and route out put of the adder to the PC mux.

Move quick-compare logic to the ID stage use output of the logic to control PC-mux

Move forwarding datapaths from the beginning of the EX stage to the end of the ID stage.

Adjust stall logic to squash instruction fetched when the branch is in the ID stage if the branch is taken.

1.d (10 points)

You are now a veteran of many simple pipelined processor designs. Your next assignment is to implement a dynamically scheduled processor with speculative execution. Your boss tells you that you should implement a processor with an instruction scheduling window size of 30 instructions but that you should only use a simple 2-bit branch predictor with a misprediction rate of 10%. You argue that a 30 instruction window will not be completely effective without better branch prediction. To prove your point you compute the effective average window size (EAWS) which factors in the effect of branch mispredictions. Assume that 20% of the instructions are branches and they are uniformly distributed in the instruction window. Also assume that the first instruction in the window is a branch. What is the EAWS of the proposed processor?

To compute EAWS we need to know the probability that any instruction in the window will be executed. Assume the following window configuration (B = branch, I = Non-branch)

B I I I I B I I I I B I I I I B I I I I B I I I I B I I I I

The probability that the first instruction, which is a branch, is executed is 1. The probability that the next 5 instructions are executed is the probability that the branch is correct (0.9). The probability that the next five instructions are executed is the probability that the first two branches are correct (0.9²). There are 6 branches in the 30 instruction window so the equation for EAWS is:

$$1 + (0.9 \times 5) + (0.9^2 \times 5) + (0.9^3 \times 5) + (0.9^4 \times 5) + (0.9^5 \times 5) + (0.9^6 \times 4) = 21.55$$

Question 2.0 ISA Design Comparison (40 points)

Pat Slack is an entry level assembly code programmer at SansTravail Industries. Being as lazy as he is, he is tired of writing code for standard DLX and believes that he could save a lot of time and instruction count coding with a CISC architecture. Seeing as how more than half the world is using x86 processors, Pat rationalizes that the x86 architecture must be superior in both speed and code density or else people would have switched to another architecture long ago. So Pat brings forth these arguments and some data regarding two benchmarks to you claiming that the entire company should switch to x86. His data is presented to you below.

Standard DLX

The instruction counts have been given below. Those instructions that may cause a single stall in the pipeline have been measured to see how often they actually cause a stall. This is noted as a percent of the number of instructions that stall. This is for a typical DLX pipeline where the branch is decided at the end of the execution stage. Pat reassures you that all of these numbers are in fact dynamic instruction counts.

	Benchmark 1		Benchmark 2	
Opcode	Ins Count #1	Stall%	Ins Count#2	Stall%
LW	25,200	32	1900	45
SW	7900	-	450	-
ALUOp	55,700	-	3500	-
Branch	10,400	39	320	66
Jumps	540	100	50	100
Totals	99740		6220	

x86-lite

The table is similar to DLX

	Benchmark 1		Benchmark 2	
Opcode	Ins Count #1	Stall%	InsCount#2	Stall%
LW	14,300	see table	1500	see table
SW	8300	see table	220	see table
ALUOp	49,500	see table	3500	see table
Branch	11,300	46	500	74
Jumps	440	100	90	100
Totals	83840		5810	

Since x86 makes the use of a wide variety of addressing modes, the stalling behavior is dependent upon which mode is used. This is shown in the following table (on next page). Note you can use this data for both benchmarks.

Mode	%used	Stall Behavior
Register	48	none
Immediate	23	none
Indirect	5	66% of ALUops stall 1 cycle, 34% of ALUops stall for 2 cycles 29% of LD/ST stall 1 cycle
Based	16	same as indirect
Indexed	6	same as indirect
Based+indirect	2	same as indirect

2.a (5 points)

Calculate the average CPI for the DLX processor. The first benchmark is used 75% of the time and the second benchmark is used only 25%. Please show all equations and all work for credit.

DLX

$$\text{CPI} = 1 + [(\text{lwcount} * \%stall + \text{branchcount} * \%stall + \text{jumpcount} * \%stall) / \text{numInstructions}]$$

$$\text{CPI benchmark1} = 1 + ((25200 * .32) + (10400 * .39) + (540 * 1)) / 99740 = 1.1269$$

$$\text{CPI benchmark2} = 1 + ((1900 * .45) + (320 * .66) + (50 * 1)) / 6220 = 1.179$$

$$\text{Average CPI} = (1.1269 * .75) + (1.1775 * .25) = 1.14$$

2.b (8 points)

Calculate the average CPI for the x86 processor using the same weightings as above.

X86 lite

ALU stalls

$$\text{Stalls} = (.05 + .06 + .02 + .16) * (.66(1) + .34(2)) = .3886$$

LD/ST stalls

$$\text{Stall} = (.05 + .06 + .02 + .16) .29 = .0841$$

CPI benchmark 1

$$= 1 + [(\text{lw/st count} * \%stalls + (\text{ALU} * \text{stallsalu}) + (\text{Branch} * 1) + (\text{CallRet} * 1))] / \text{numInstructions} = 1.3193$$

CPI benchmark 2

$$= 1 + [(\text{lw/st count} * \%stalls + (\text{ALU} * \text{stallsalu}) + (\text{Branch} * 1) + (\text{CallRet} * 1))] / \text{numInstructions}$$

$$= 1.338$$

$$\text{Avg CPI} = (1.3193 * .75) + (1.338 * .25) = 1.3239$$

2.c (7 points)

Despite the fact that the DLX has a 10% higher clock rate, Pat still claims that the x86 is slightly faster. Which machine is faster and by how much?

$$\text{ClockRateDLX} / \text{ClockRatex86} = 1.1$$

$$\text{IC DLX} / \text{ICx86} = (.75 * (99740) + .25 * (6220)) / (.75 * 83840) + (.25 * 5810) = 1.186$$

$$\text{Execution time} = \text{CPI} * \text{IC} * \text{CCT}$$

$$\text{CPU time x86} / \text{CPU time DLX} = (1.3239 * 1 * 1.1\text{cctDLX}) / (1.14 * 1.186 * \text{cctDLX}) = 1.077$$

So DLX is 7.7% faster than x86

2.d (10 points)

Lets say now that Pat wants to compromise with you. He suggests adding BGT(branch greater than), BLT(Branch less than), and BEQ(Branch equal to) to the existing DLX instruction set. He calls this new instruction set DLXenhanced. These new branch instructions take two register operands.

For example BEQ R1, R2, target - If R1 is equal to R2, branch to target.

Note: Standard DLX should not be confused with the DLX-lite ISA. Standard DLX does not contain BGT, BLT, and BEQ. It only contains BEQZ and BNEZ along with arithmetic instructions that set the condition code.

These new branch instructions can replace 80% of the existing branches in both benchmarks for DLX. Calculate the new CPI for DLXenhanced.

Benchmark 1

80% of branches = 8320 branches which can use the new format.

As a result of these new instructions, the number of ALUops will be reduced by 2080.

$$\text{CPI} = 1 + [(\text{lwcount} * \%stall + \text{branchcount} * \%stall + \text{jumpcount} * \%stall) / \text{numInstructions}]$$

$$\text{CPI} = 1 + [(25200 * .32 + 10400 * .39 + 540 * 1) / (99740 - 8320)] = 1.138$$

Benchmark 2

80% of branches = 256 branches

$$\text{CPI} = 1 + [(\text{lwcount} * \%stall + \text{branchcount} * \%stall + \text{jumpcount} * \%stall) / \text{numInstructions}]$$

$$\text{CPI} = 1 + [(1900 * .45 + 320 * .66 + 50 * 1) / (6220 - 256)] = 1.187$$

$$\text{Average CPI} = .75 * 1.138 + .25 * 1.187 = 1.15$$

2.e (10 points)

If DLXenhanced has the same clock rate as standard DLX, which machine is faster and by how much?

$$\text{IC DLX} / \text{IC DLXenhanced} = (.75 * (99740) + .25 * 6220) / (.91420 * .75) + (.25 * 5964) = 1.089$$

$$\text{CPU time DLXenhanced} / \text{CPU time DLX} = (1.15 * 1 * 1.00\text{cctDLX}) / (1.14 * 1.089 * \text{cctDLX}) = .926$$

So DLXenhanced is 7.9% faster than standard DLX

Question 3.0 Pipeline Design (40 points)

You've been hired by TA64 Systems to implement a new 64-bit processor that uses predication to reduce the penalties associated with branch prediction.

You are given a single-issue processor with the following pipeline stages:

- IF1 Instruction Fetch: First Cycle
- IF2 Instruction Fetch: Second Cycle
- RF Instruction Decode and Register Fetch
- EX ALU Execution. Branch Condition Resolved
- MEM1 Memory Stage #1
- MEM2 Memory Stage #2
- PC Predication Condition Resolved
- WB Write Back Stage

i	IF1	IF2	RF	EX	MEM1	MEM2	PC	WB							
i+1		IF1	IF2	RF	EX	MEM1	MEM2	PC	WB						
i+2			IF1	IF2	RF	EX	MEM1	MEM2	PC	WB					
i+3				IF1	IF2	RF	EX	MEM1	MEM2	PC	WB				
i+4					IF1	IF2	RF	EX	MEM1	MEM2	PC	WB			
i+5						IF1	IF2	RF	EX	MEM1	MEM2	PC	WB		
i+6							IF1	IF2	RF	EX	MEM1	MEM2	PC	WB	
i+7								IF1	IF2	RF	EX	MEM1	MEM2	PC	WB
i+8									IF1	IF2	RF	EX	MEM1	MEM2	PC
i+9										IF1	IF2	RF	EX	MEM1	MEM2
i+10											IF1	IF2	RF	EX	MEM1

Predication is a technique for squashing instructions based on the value of a register in order to eliminate small forward branches. For instance the following section of code:

```
0x0000    Bne r0,r1, 0xc
0x0004    NOP
0x0008    add r4,r3,r2
0x000c    lw r8,0(r4)
```

Can be reduced to this code:

```
0x0000    (r1) add.eq0 r4,r3,r1
0x0004    lw r8(r4)
```

The advantage is that there is no branch prediction for this small section of code and thus the pipeline is not flushed on a misprediction. The instruction is simply squashed in the PC stage.

3.a (10 points)

Assuming no predication, list the minimum number of physical forward paths to minimize data hazard stalls. Indicate the forwarding in the following manner: $EX_i \rightarrow EX_{i+1}$.

```
EXi->EXi+1
MEM1i->EXi+2
MEM2i->MEM1i+2
MEM2i->MEM2i+3
PCi->EXi+4
```

3.b (5 points)

Assuming that predication has been added to our processor model, what forwarding paths would we need to minimize data hazards? Could we remove any forwarding paths?

```
PCi->PCi+1
PCi->MEM1i+2
PCi->MEM2i+3
```

If one assumes that the processor can differentiate between predicated and non-predicated instructions than you can't remove any forwarding paths. However, if one assumes that the processor can't, then you must remove all the forwarding paths.

3.c (5 points)

What is the critical flaw in the design of the processor pipeline which makes scheduling code for this processor difficult? What would you do to fix it?

The critical flaw is that a store instruction may be predicated off and store the wrong value to the memory. To solve this move the predicate resolution stages much earlier in the pipeline.

3.d (4 points)

The project engineer on the design team wants to investigate the effect of predication on our processor model. To do this, first determine the maximum number of branch delay instructions we would need to add in order to remove all stalls due to taken branches.

if one assumes that the branches are predicated than the maximum delay slots are 6. However, if you assume that branches can't be predicated [a clarification was placed on the board to this effect], then the answer is 3.

3.e (10 points)

Assume that the CPI of our baseline model without predication is 1.4. By profiling several benchmarks, we determine that 10% of all instructions are branches, of which 70% are taken. On average, we can fill 2.5 branch delay slots. By adding predication, we can remove 15% of the branches and replace them with a single predicated instruction. Determine the CPI of the new machine.

If you got 3 branch delay slots:

$$\text{CPI}_{\text{new}} = (1.4 - 0.10 \cdot 0.15 \cdot 1.5) / (1 - 0.15 \cdot 0.10 \cdot 1.0) = 1.3984$$

If you got 6 branch delay slots:

$$\text{CPI}_{\text{new}} = (1.4 - 0.10 \cdot 0.15 \cdot 3.5) / (1 - 0.15 \cdot 0.10 \cdot 1.0) = 1.368$$

3.f (6 points)

How should precise interrupts be handled in this pipeline?

Precise interrupts should be handled as a normal DLX pipeline would except that interrupts that are generated by a predicated instruction should be squashed. An example is a divide by 0 interrupt that is predicated off if the denominator is zero. Partial credit was given for a reasonable pipelined interrupt handling scheme was described.