EE282 Computer Architecture and Organization
Midterm Exam

February 13, 2001

(Total Time = 120 minutes, Total Points = 100)

Name: (please print) _____**Wolfe - Solution**_____

In recognition of and in the spirit of the Stanford University Honor Code, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature:_____

This examination is open notes, open book. You may not, however, collaborate in any manner on this exam. You have two hours to complete the exam. Before starting, please make sure you have all 9 pages.

1	
2	
3	
4	
Total	

1) Short Answer Questions (28 points – 4 points each unless marked)

A) A program consist of 10 subroutines (S_1 through S_{10})– each run once. You know the CPI for each subroutine and how many instructions are in each subroutine. How should the mean CPI for the entire program be computed?

$$Avg.\ CPI = \frac{\sum_i \left(Instructions_i \cdot CPI_i\right)}{\sum_i \left(CPI_i\right)}$$

B) A well-encoded variable-length instruction set produces larger programs than a well-encoded fixed-length instruction set (true or false).

False

C) All accumulator machines must use self-modifying code to traverse a linked list (true or false).

False

D) Increasing a 5-stage DLX pipeline to a 7-stage pipeline by breaking the execute stage into three stages will add how many additional stall cycles to the following code:

```
ADD   R1,R2,R1
LW    R2, 0(R2)
ADD   R1,R2,R1
J     label
```

2 additional stalls

```
Old Pipeline:       F D X M W
                      F D X M W
                        F D - X M W

New Pipeline:       F D X X X M W
                      F D X X X M W
                        F D - - - X M W
```

E) What is the asymptotic prediction accuracy of a two-bit branch predictor on the repeating pattern NNTNNTNNNTNNTNNTT… (T=taken, N=not taken). (length =17)

10/17

F) You need to make a choice between a pipelined DLX implementation without any bypassing or interlocking (ability to stall) and one with bypassing and stalling. You know exactly which programs you are going to execute. Why might you get better performance from the implementation without any bypassing or interlocking?

Faster clock speed – or something similar

G) Adding a faster floating-point unit to a machine will give a speedup of 500% (6x faster) on FP instructions. The FP instructions make up 10% of the dynamic instruction count on the original machine and the average CPI for FP instructions on the original machine is 9 cycles and the average CPI for all other instructions on the original machine is 1 cycle. What is the overall speedup of the machine due to this modification.

Original CPI = .1*9+.9*1=1.8

New CPI = .1*1.5+.9*1 = 1.05

Speedup = 1.8/1.05=1.714 or 71.4%

2) ISA (20 points)

A simple processor can be built with only an accumulator (A) and an instruction pointer (IP) as registers. Assume we have such a machine with the following instruction set:

Mnemonic	Function	Name	Encoding		
			Byte 2	Byte 1	Byte 0
LD X	A <- M(X)	Load	00000001	XXXXXXXX	XXXXXXXX
LDI X	A <- X	Load Immediate	00000010	XXXXXXXX	XXXXXXXX
LDX	A <- M(A)	Load Indexed	00000011	00000000	00000000
ST X	M(X) <- A	Store	00000100	XXXXXXXX	XXXXXXXX
STX X	M(A) <- M(X)	Store Indexed	00000101	XXXXXXXX	XXXXXXXX
ADD X	A <- A+M(X)	Add	00000110	XXXXXXXX	XXXXXXXX
SUB X	A <- A-M(X)	Subtract	00000111	XXXXXXXX	XXXXXXXX
BR X	IP <- X	Branch	00001000	XXXXXXXX	XXXXXXXX
BZ X	If (A==0) IP <- X else IP <- IP+3	Branch if Zero	00001010	XXXXXXXX	XXXXXXXX
BN X	If (A<0) IP <- X else IP <- IP+3	Branch if Negative	00001011	XXXXXXXX	XXXXXXXX

A is the contents of the accumulator. M(X) is the contents of memory address X. M(A) is the memory address pointed to by the accumulator. The machine is **byte addressed**. The accumulator is 16 bits wide. The IP is 16 bits wide. Instructions are 24 bits long.

We need to write a program to compare 2 strings S0 and S1. *Each character in each string is **16-bits** long* (unicode). Each string is terminated by a NULL (0x0000).

a) Assume we have a simple computer system using this processor that has 32K bytes of ROM and 8 bytes of RAM. We therefore need to store the program and the strings in ROM. Using this instruction set on this machine, write an assembly program fragment to compare the strings S0 and S1 and set the memory location same to 1 if they are equal, and to 0 if they are not. Remember – that no matter what you put into the program – the contents of RAM are undefined at the beginning or program execution.

```
            org   0x0000            ; (RAM)
equal:      dw    0x0000

; add more variables here



            org   0x8000            ; (ROM)
S0:         dw    ??????????????    ; some unknown string of unknown length
            dw    0x0000
S1:         dw    ??????????????    ; some unknown string of unknown length
            dw    0x0000

            org   0x8800
start:

; add code here
```

For example:

```
        org    0x0000
same:   dw     0x0000
count:  dw     0x0000
temp:   dw     0x0000

        org    0x8800
        ldi    0x0000
        st     count
        st     same

loop:   add    S0
        ldx
        st     temp
        ldi    S1
        add    count
        ldx
        sub    temp
        bz     samchr
        ldi    0x0001
        st     same
        br     done

samchr: ld     temp
        bz     done
        ldi    2
        add    count
        st     count
        br     loop

done:
```

Pipelining (26 points)

Consider the following 10-stage pipelined processor that we will call SUPER-DLX.

F1	Instruction fetch 1: send address to I-cache
F2	Instruction fetch 2: receive instruction from I-cache
D1	Instruction decode begins,
D2	Complete Decode, Instruction type identified
RR	read the registers, branch target address available
A1	start ALU operation; resolve branch condition
A2	ALU operations completed (address for load/store calculated)
M1	Memory 1: send address to D-cache
M2	Memory 2: read/write data to D-cache (store data required here)
WB	Write results back to register file

In SUPER-DLX, the writes to the register file occur in the first half of the cycle and the register reads occur in the second half of the cycle. You can assume that the F and M stages are pipelined so that one instruction fetch (F) and one data access (M) can begin each cycle.

A. (8 points) Assume that all possible useful bypass paths will be incorporated into this pipeline. List all of the bypass paths. Use the form X -> Y meaning that there is a bypass from the end of stage X to the beginning of stage Y.

```
A2 -> A1      M2 -> M2

M1 -> A1      A2 -> R

M2 -> A1      M1 -> R

M2 -> M1      M2 -> M2
```

(6 points) List all bypass paths that are exercised to execute the following code on the SUPER-DLX.
How many stall cycles will occur, if any?

```
1.    LW    R1, 4(R4)
2.    ADD   R3, R2, R1
3.    ADD   R4, R1, R3
4.    SW    R4, 0(R6)
```

Forwarding Paths

A2 -> A1 **M2 -> M2** **M2 -> A1**

Number of Stall Cycles	**4**

B. (6 points) For the following piece of code how many stall cycles will occur from the first SUB to the instruction at END if the branch is taken? Assume that if the branch is taken and the branch direction is correctly predicted then the branch target is always correctly predicted. Answer the question for no prediction (wait for branch condition & target before fetching), predict not taken, and predict taken.

```
1.     SUB   R3, R7, R8
2.     BEZ   R3, END     // branch if R3 == 0
       ...
       ...
END:   SW    R3, 4(R9)
```

scheme	# Stall Cycles
no prediction	**6**
predict not taken	**6**
predict taken	**1**

D. (6 points) For each of the following instruction sequences, specify any forwarding path used and/or the number of stall cycles that are required on the SUPER-DLX.

Type	Instruction Sequence	Forwarding Path	Stalls
1	ALU Rx, -, - Store Rx, 0(-)	**M2 -> M2**	**0**
2	Load Rx, 0(-) ALU -, -, Rx	**M2 -> A1**	**3**
3	Load Rx, 0(-) Branch Rx, target	**M2 -> A1**	**3**
4	ALU Rx, -, - Branch Rx, target	**A2 -> A1**	**1**
5	Load Rx, 0(-) ALU Ry, R0, R0 ALU -, -, Rx	**M2 -> A1**	**2**
6	ALU Rb, -, - Store -, 0(Rb)	**A2 -> A1**	**1**

3) Branch Prediction (26 points)

Consider the following branch patterns:

```
    1  2  3  4  5  6  7  8  9  10   1 ..
b1: N  N  N  N  T  N  T  T  T  T  | N  N
b2: N  T  T  T  N  T  N  N  N  N  | N  T
b3: T  T  T  T  T  T  T  T  T  N  | T  T
```

The inner loop has three branches that execute in sequence b1, b2, b3, with the pattern shown. The inner loop always executes for 10 iterations. It resides within an outer loop that executes thousands of times.

(a) (12 points) In the steady state, what is the prediction accuracy for each of the following predictors on these three branches. A 1 entry BHT means that there is only 1 history remembered for the entire processor. A huge BHT means that each branch in this code has its history tracked separately.

Fill in the blanks in the table.

Predictor	Accuracy on b1	Accuracy on b2	Accuracy on b3
Static, predicting not taken	**5/10**	**6/10**	**1/10**
One-bit predictor with a 1-entry BHT	**6/10**	**1/10**	**5/10**
One-bit predictor with a huge BHT	**6/10**	**6/10**	**8/10**
Two-bit predictor with a huge BHT	**5/10**	**5/10**	**9/10**

Suppose that this machine has a 10-stage pipeline with the following stages

F1 – start the fetch, predict if the instr. is a branch, whether it is taken or not, and if taken, the target address.
F2 – complete the fetch
D1 – decode the instruction – know it's a branch at the end of D1,
D2 – complete decode
RR – read the registers– branch target address available at the end of this stage
A1 – start ALU operation; resolve branch condition
A2 – complete ALU operation
M1 – start memory operation
M2 – complete memory operation
WB – write the result to registers

(b) (4 points) What is the penalty for a mispredicted branch?

5 cycles

(c) (4 points) What is the penalty when a branch is correctly predicted as taken, but the branch target address is incorrectly predicted?

4 cycles

(d) (6 points) Assume the following

1. There are no stalls in this pipeline except for branch instructions.
2. Branch instructions account for 25% of all instructions
3. 20% of branch instructions are taken
4. Branch direction is correctly predicted 90% of the time
5. The target address for a taken branch is correctly predicted 80% of the time

What is the CPI for this machine?

CPI = .75*1 + .25 * (.2*(.9*.8*1+.8*.2*5+.1*6) +.8*(.9*1+.1*6)) = 1.161