Name: SAMPLE **ID:** **Signature:**

Course 304-425B Fall 2003 --- Computer Architecture and Organization

Mid-term examination – October 27, 2003

Instructions

- **This is a closed book examination**. Calculators and **one sheet** of notes written both sides are allowed.
- Explain every result **when asked**. Marks will be given for clear, concise solutions.
- State any assumption required for an answer if it not clear in the text of the question.
- Please **sign** this paper at the top of the page, write your **name** and student **number** legibly there and put your initials on all the answer sheets. Make sure they are returned well secured together.
- There are [illegible] questions and you are given 50 minutes to complete this exam.
- Put all your answers in the places indicated. Any other location will not be marked.

---

1) Performance (10 points) Certain floating-point-optimized machines do not have a FP divide instruction. Instead they have a reciprocal unit, so divisions require two instructions: take the reciprocal then multiply. But because it is easier to design a pipelined reciprocal unit than a pipeline divide unit, we want to find how much pipelining in this unit would make the enhancement worthwhile. Assume that the CPI of the reciprocal instruction is equal to the CPI for the unpipelined division divided by the number of stages. The clock cycle time is the same. The benchmarks ran on the original machine with the unpipelined divide unit yields the following CPIs and instruction counts (In millons of instructions):

	Ld/St	Int-ALU	FP-Add	FP-Mult	FP-Divide	Branches	Others
CPI	0.2	0.5	2	8	16	0.1	1
counts	10	12	5	3	1	5	0.000001

What is the minimum number of stages for the reciprocal unit needed to make the enhancement worthwhile? Show how you arrive at the result.

$$CPUTIME_1 = CC_{TIME} \sum IC_i \times CPI_i = CC_{TIME}\left(\sum_{\substack{\text{ALL EXCEPT}\\ \text{DIVISIONS}}} IC_i * CPI_i + 10^6\, 16\right)$$

$$CPUTIME_2 = CC_{TIME} \qquad = CC_{TIME}\left(\sum_{\substack{\text{ALL EXCEP}\\ \text{DIVISIONS}}} IC_i * CPI_i + 10^6 \frac{16}{n} + 10^6\, 8\right)$$

$$CPUTIME_2 < CPUTIME_1 \Rightarrow \frac{16}{n} + 8 < 16$$

$$\Rightarrow \frac{16}{n} < 8 \Rightarrow \underline{n > 2}$$

2) Performance (10 points) Consider the 5 stage MIPS style single pipeline. We want to find a good estimate of the speedup given by upgrading its original unified cache. There are 30% Load/Stores. Before enhancement, all memory accesses miss at a rate of 3% and the penalty is 30CC per miss. After enhancement which involves setting up a split cache, the miss rate is 2% for instruction fetches but remains at 3% for data accesses. Penalty is the same.

$$\text{UNIFIED} \quad CPIREAL = 1 + 0.03 * 30 + 0.3\ 0.03\ 30$$
$$= 1 + 0.9 + 0.27 = 2.17$$

AMBDAHL'S

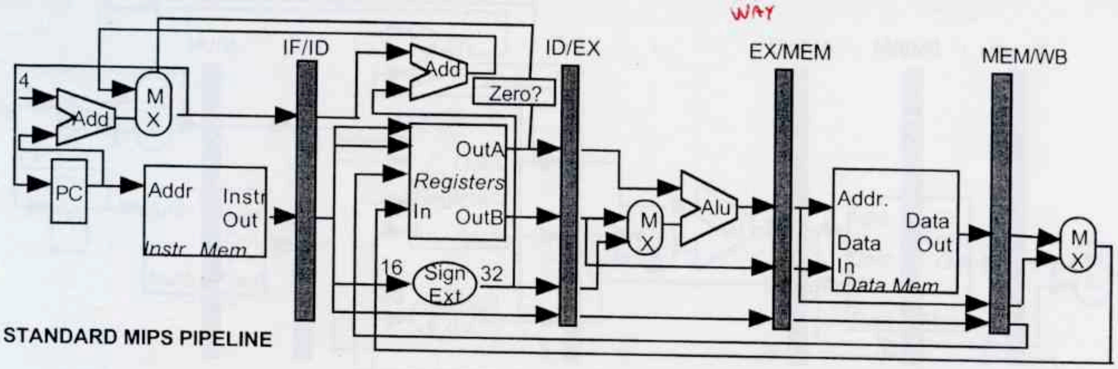
$$\text{SPLIT} \quad CPIREAL = 1 + 0.02 * 30 + 03 \cdot 0.03\ 30$$
$$= 1 + 0.6 + 0.27 = 1.87$$

$$\underline{SU = 1,16}$$

NOTE: $FE = .9/2.17 = 0.415$
$SU_E = 0.03/0.02 = 1.5$

$$SU = \frac{1}{.585 + \frac{.415}{1.5}} = 1.16$$

ıteger Pipelining (20 points) Consider the 5-stage MIPS pipeline with simple branch delay running a loop to ompute the sum of all the elements of an array. Assume full hazard detection and forwarding hardware including to and from the data memory. The jump intruction is implemented in such WAY that it has no delay slot.



STANDARD MIPS PIPELINE

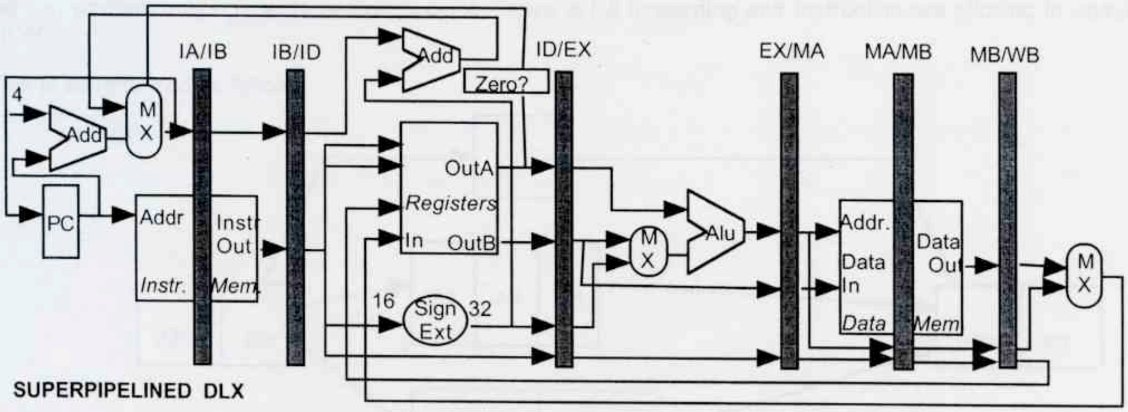
```
       AND    R1,R0,R0        // clear running sum
LOOP:  DSLT   R5,R2,R3        // compare running pointer to &A[n-1] (n length of array)
       BEQZ   R5,OUT          // if pointer exceeds bound, exit loop
       NOP                    // NOP instruction to handle the branch delay
       LD     R4, 0(R2)       // load element
       DADD   R1,R1,R4        // do running sum
       DADDI  R2,R2,#8        // increment pointer by 8 bytes
       J      LOOP            // loop back
OUT:   AND    R2,R0,R0        // clear R2 for starting an unrelated computation
       AND    R4,R0,R0        // same for R4
```

Schedule this code to minimize stalls.

```
       AND    R1,R0,R0
LOOP   DSLT   R5,R2,R3
       LD     R4,0(R2)      OK TO LOAD ONCE TOO MANY.
       BEQZ   R5,OUT
       DADDI  R2,R2,#8      OK TO INC R2 ONCE TOO MANY.
       DADD   R1,R1,R4
       J      LOOP
OUT :  AND    R2,R0,R0
       AND    R4,R0,R0
```

+ OTHER
POSSIBILITIES

Integer Pipelining (20 points). Consider a superpipelined 7 stage pipeline where the memory accesses take two clock cycles:



SUPERPIPELINED DLX

Assuming full hazard detection and forwarding hardware including to and from the data memory. Assume that the branch is handled by pure delay, that jumps bevave like branches, i.e. compute jump address in ID stage (notice that despite the fact that there are now 2 delay slots, the code will work with only 1 NOP).

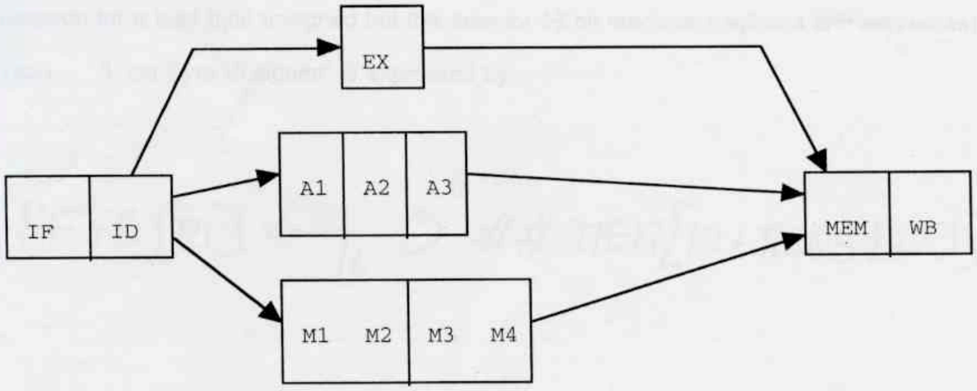
```
        LD      R1,0(R5)     // load number
        BNEZ    R1,THEN      // if not zero then
        NOP
        DADDI   R2,R0,#1     // else temp=1
        J       STORE
        NOP
        NOP
THEN:   DADD    R2,R1,R1     // if not zero then double number in temp
STORE:  SD      R2,0(R6)     // store result (1 or n*2)
```

Use the chart below to show the timing of this code when the branch is taken.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LD	IA	IB	ID	EX	MA	MB	WB													
BNEZ		IA	IB	S	S	S	ID	-	-	-										
NOP			IA	S	S	S	IB	ID	EX	MA	MB	WB								
DADDI							IA	IB	ID	EX	MA	MB	WB							
DADD								IA	IB	ID	EX	MA	MB	WB						
SD									IA	IB	ID	EX	MA	MB	WB					

, **Floating Point Pipelining (20 points)** Assume that the exec stage of the FP addition takes three clock cycles and is fully pipelined so its initiation rate is one addition per clock cycle, while ~~that~~ the multiplication takes four clock cycles but is partially pipelines resulting in an initiation rate of one new multiplication per two clock cycles. The register file can perform only one write per clock cycle. There is full forwarding and instruction are allowed to complete out-of-order.

This is summarized as follows.



Produce the timing of the following code sequence.

```
L.D    F8,0(R1)
MUL.D  F2,F8,F4
MUL.D  F6,F4,F4
S.D    F2,0(R1)
ADD.D  F8,F4,F4
S.D    F6,0(R2)
L.D    F6,0(R3)
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L.D	IF	ID	EX	ME	WB															
MUL.D		IF	ID	S	M1	M2	M3	M4	ME	WB										
MUL.D			IF	S	ID	S	M1	M2	M3	M4	ME	WB								
S.D					IF	S	ID	EX	S	ME	WB									
ADD.D							IF	ID	S	A1	A2	A3	WB							
S.D								IF	S	ID	EX	ME	S	WB						
L.D										IF	ID	EX	S	ME	WB					

ISA (10 points) Recall the specification of instructions using a register transfer language, for example on the 64 bit MIPS:

`LH R1,10(R2)` "Load Half Word Signed" is expressed by

$$\text{Regs[R1]} \leftarrow_{64} (\text{Mem[10+Regs[R2]]}_0)^{48} \text{ \#\# Mem[10+Regs[R2]] \#\# Mem[11+Regs[R2]]}$$

Find a RTL expression for a load byte unsigned but this time for 16 bit machine (registers and addresses)

`LBU R1,10(R2)` "Load Byte Unsigned" is expressed by

$$\text{REGS[R1]} \leftarrow_{16} 0^{8} \text{ \#\# MEM[10+REGS[R2]]}$$

7) Exceptions (10 points)

Consider this sequence of two instructions where we assume that the `MULT` instruction has an exec stage of one cycle like other ALU instructions.

```
SD   R2,0(R1)
MULT R2,R3,R3
```

Suppose that two exceptions occur: in the MEM stage of the store (page fault) and in the EX stage of the multiplication (overflow) both of which should be re-startable.

Briefly explain the problem and what must be done to ensure precision.

SD	IF	ID	EX	MEM	WB	
MULT		IF	ID	EX	MEM	WB

SD RAISES AN EXCEPTION IN THE MEM STAGE, AND SO DOES MULT. IF THE MULT IS ALLOWED TO COMPLETE (I.E. WRITE BACK), BECAUSE OF "WAR" HAZARD, INCORRECT RESTARTING COULD FOLLOW. THE PRECISE BEHAVIOR IS PRESERVED IF THE PAGE FAULT IS RESOLVED BEFORE R2 IS ERASED. IN OTHER WORDS, EXCEPTIONS MUST BE RESOLVED (AND INSTRUCTIONS RESTARTE) IN THE ORDER OF THE INSTRUCTIONS THAT RAISED THEM (THIS IS IMPLEMENTED WITH A STATUS VECTOR ASSOCIATED WITH EACH INSTRUCTION).

END OF EXAM