Fall 2010

Name: Mark F.

Computer Architecture Test 1

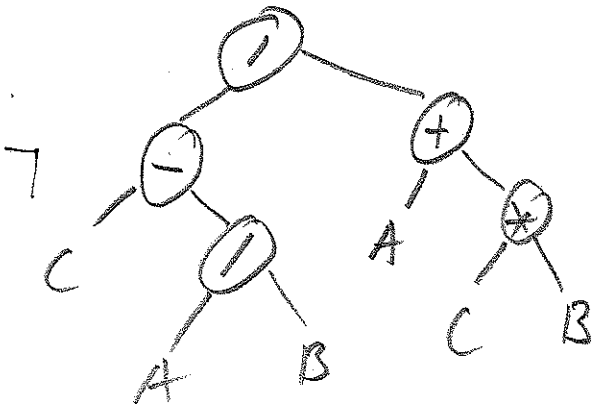
Question 1. (15 points) Consider the high-level assignment statement X = (C - A / B) / (A + C * B)

a) As in homework #1, write the LOAD and STORE assembly language instructions for this statement.

```
LOAD R1, A
LOAD R2, B
LOAD R3, C
DIV  R4, R1, R2
SUB  R4, R3, R4
MUL  R5, R3, R2
ADD  R5, R1, R5
DIV  R6, R4, R5
STORE R6, X
```

8

b) As in homework #1, write the 0-address (stack machine) assembly language instructions for this statement.



7

CAB / - ACB * + /

```
PUSH C
PUSH A
PUSH B
DIV
SUB
PUSHA
PUSH C
PUSH B
MUL
ADD
DIV
POP X
```

Question 2. (15 points) How does each of the following RISC (reduced instruction set computer) characteristic aid in instruction pipelining?

a) fixed-length instruction formats (e.g., all instructions 4 bytes)

The fetch stage can always read an instruction by reading the same # of bytes. Not variable time to perform.

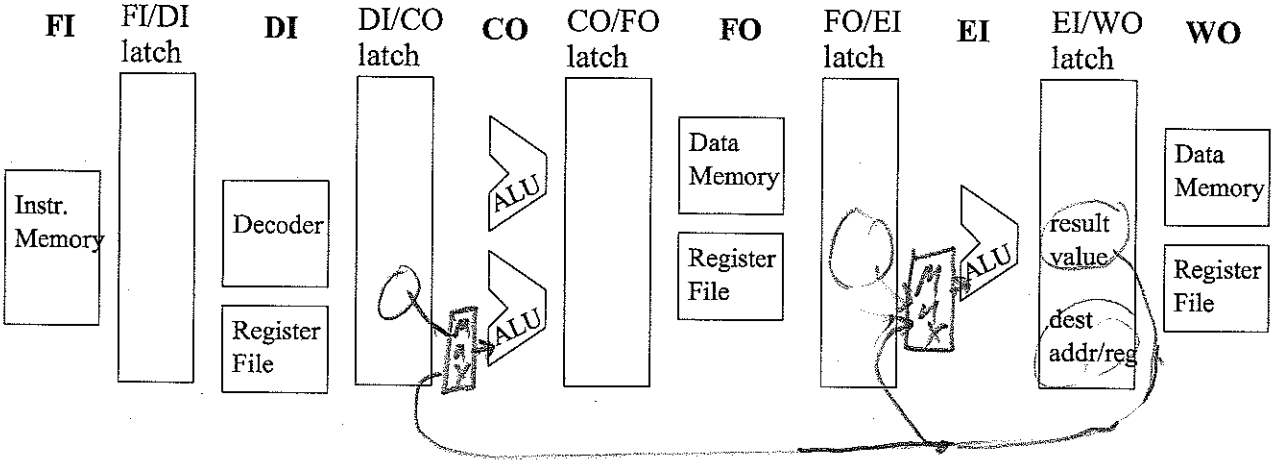
b) register-to-register arithmetic operations of a LOAD/STORE machine

Fetching register operands and write result to register is fast and constant in duration.

30

Fall 2010 Name: _____

Question 3. (25 points)



a) For the six stage pipeline of the text (see above), complete the following timing diagram **assuming NO by-pass signal paths.** Assume that we cannot write to a register and read from that same register in the same clock cycle. Recall that:

- The first register is the destination register, e.g., "ADD R2, R6, R7" performs R2 ← R6 + R7
- LOAD R1, 16(R2) - loads the value from memory at the address 16 + (address in R2) into R1
- STORE R2, 8(R6) - stores the value in R2 to memory at the address 8 + (address in R6)

Instructions		Time →																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1:	MUL R2, R3, R5	FI	DI	CO	FO	EI	WO														
I2:	LOAD R5, 8(R2)		FI	-	-	-	-	DI	CO	FO	EI	WO									
I3:	SUB R6, R5, R2							FI	DI	CO	-	-	FO	EI	WO						
I4:	STORE R6, 4(R2)								FI	DI	-	-	CO	-	-	FO	EI	WO			
I5:	ADD R6, R8, R9									FI	-	-	DI	-	-	CO	FO	EI	WO		

b) Complete the following timing diagram **assuming by-pass signal paths.**

Instructions		Time →																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1:	MUL R2, R3, R5	FI	DI	CO	FO	EI	WO														
I2:	LOAD R5, 8(R2)		FI	DI	-	-	CO	FO	EI	WO											
I3:	SUB R6, R5, R2			FI	-	-	DI	CO	FO	EI	WO										
I4:	STORE R6, 4(R2)						FI	DI	CO	FO	EI	WO									
I5:	ADD R6, R8, R9							FI	DI	CO	FO	EI	WO								

c) In the diagram at the top of the page add all by-pass signal paths used in part (b).

d) For the above program, indicate pairs of instruction that have **(one pair is enough for each type)**

i) write-read/read-after-write (RAW)/"true" data dependencies - I1 and I2 on R2

ii) output/write-write/write-after-write (WAW) dependencies - I3 and I5 on R6

iii) antidependencies/read-write/write-after-read (WAR) dependencies - I1 and I2 on R5
I4 and I5 on R6

Question 4. (25 points) Consider the following sequential search algorithm that searches an array for a specified "target" value. The index of where the "target" value is found is returned. If the "target" value is not in the array, then -1 is returned.

```
SequentialSearch (integer numberOfElements, integer target, integer array numbers[]) returns an integer
   integer test
   for test = 0 to (numberOfElements-1) do
      if number[test] == target then
         return test
      end if
   end for
   return -1
end SequentialSearch
```

(a) (b)

cond. PREDICT_NOT_TAKEN

cond. PREDICT_TAKEN +2

uncond. +3

+6 (a) Where in the code would unconditional branches be used and where would conditional branches be used?

+4 b) If the compiler could statically predict by opcode for the conditional branches (i.e., select whether to use machine language statements like: "BRANCH_LE_PREDICT_NOT_TAKEN" or "BRANCH_LE_PREDICT_TAKEN"), then which conditional branches would be "PREDICT_NOT_TAKEN" and which would be "PREDICT_TAKEN"?

c) Under the below assumptions, answer the following questions.

- numberOfElements = 100 and the "target" is **not** found in the array (i.e., an unsuccessful search)
- the outcome of conditional branches is known at the end of the EI stage
- target addresses of all branches is known at the end of the CO stage

i) If static predict-never-taken is used by the hardware, then what will be the total branch penalty (# cycles wasted) for the algorithm? (Here assume NO branch target buffer) For partial credit, explain your answer.

+8 for: 4 if: 4×100 end for: 2×100 = 604 cycles

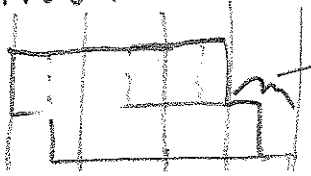
ii) If a branch target buffer with one history bit per entry is used, then what will be the total branch penalty (# cycles wasted) for the algorithm? (Assume predict-not taken is used if there is no match in the branch target buffer) For partial credit, explain your answer.

+7 for: 4 if: 4 end for: 2 = 10 cycles

25

Question 5. (5 points) How do superpipelining computers try to improve performance over traditional pipeline computers?

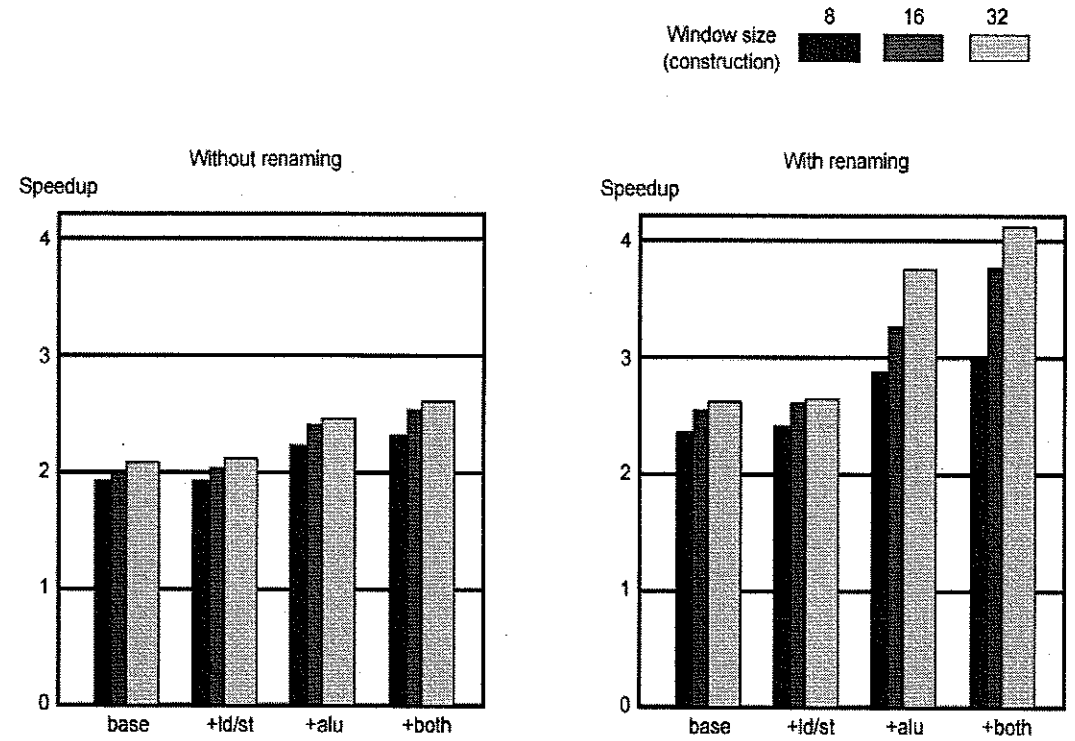
5 Each clock cycle is split into subcycles with instructions moving down the pipeline every subcycle.



eg. 2 subcycles per clock cycle, so two instr. complete per clock cycle.

30

Question 6. (15 points) Smith '95 studied the relationship between out-of-order issue, duplication of resources, and register renaming on the MIPS R2000 architecture. (See Figure 14.5 below)



The types of machine considered were:

1) base machine - no duplicate functional units, but can issue out-of-order
2) + ld/st: duplicates load / store functional unit that access data cache
3) + alu: duplicates ALU
4) + both: duplicates both load/store and ALU

Difference color bars are used to show the results for window sizes of 8 (black), 16 (dark gray), and 32 (light gray) instructions. The left graph show the results without register renaming and the right graph shows the results with register renaming. (Finally the questions)

a) Explain why register renaming is needed to significantly benefit from duplicate functional units.

Without reg. renaming the WAW and WAR dependences fill the window with instructions waiting on dependencies. By renaming regs we eliminate the WAW and WAR so more instr in the window are free to take advantage of duplicate fcn units

b) Explain why a large window size is important when renaming is used to benefit from duplicate functional units.

The larger window size allows enough independent (i.e, no RAW) instructions to be found to keep the duplicate functional units busy.