Name _____

Computer Architecture
EE 4720
Final Examination

11 May 1998, 10:00–12:00 CDT

Modified

Problem 1	_____	(20 pts)
Problem 2	_____	(30 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(30 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: An extended version of DLX is to include a new *morph* instruction, mnemonic `mrph`. Morph takes two arguments, an instruction address and a new instruction. After executing `mrph IADDR INSTR`, when execution reaches IADDR instead of executing the instruction at IADDR, INSTR is executed. This substitution only happens once, if execution reaches IADDR a second time the instruction at IADDR executes normally (unless `mrph` was executed again). For example consider:

```
 mrph POINTA, [subd f0, f2, f6]
 ...
LOOP:
 subi r2, r2, #3
POINTA:
 muld f0, f0, f2  ! On the first iteration subd will execute.
 bneq r2, LOOP
 addd f0, f0, f4
```

In the first iteration of the loop the `subd` instruction specified by `mrph` will be executed instead of the `muld`. After that the loop will execute normally. Morph might be useful for debuggers (the substituted instruction would be a jump to a debug routine).

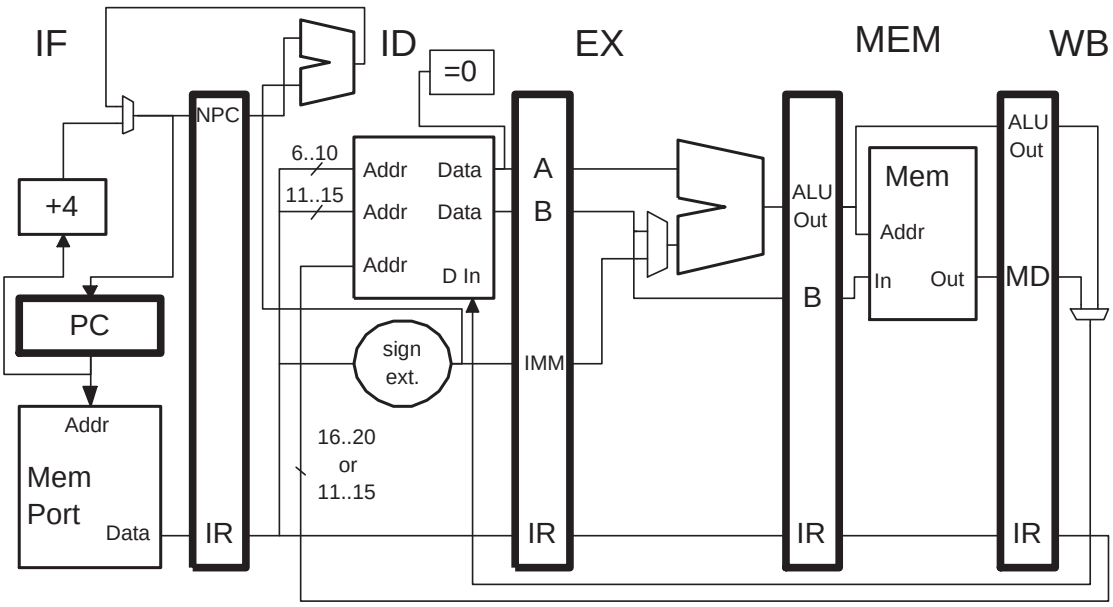
A system can have at most one morph active at any time. The morph instruction cannot modify memory[1].

(*a*) Determine a format for the morph instruction that fits naturally into the DLX ISA. (An instruction fits naturally if it uses an existing type and its implementation requires little new hardware.) The format should include the type and how the arguments are specified, that is how mrph's arguments relate to IADDR and INSTR. In other words, how is the address specified (not too difficult since many existing DLX instructions specify addresses) and how is the instruction specified (the interesting part). (The three DLX formats types are type-R, used by `add r1,r2,r3`; type-I, used by `addi r1,r2,#3`; and type-J, used by `j LOOP`. Don't confuse the assembly language instruction with the actual instruction format, the assembly language form used above may be misleading.) *Hint: If you're not sure what to do in this part, attempt the next part first.* (5 pts)

[1] Morph could be implemented in software by replacing the instruction at IADDR by a jump to a morph routine. The morph routine would execute the substituted instruction and then return.

(*b*) Modify the pipeline below to implement **mrph** using the format chosen above. The pipeline must always execute the code below correctly. (The code itself is correct.) (For partial credit if your design will not execute the code below correctly explain why and how it might be fixed.) (15 pts)

```
 bneq r10,L1
 mrph A,[add r1,r2,r3]
 j L2
L1:
 mrph SKIP,[add r0,r0,r0]
L2:
 ...
 bneq r1, SKIP     ! Hint: May cause trouble.
A:
 sub r1, r2, r3    ! Maybe add r1, r2, r3 substituted.
 lw r4, 0(r1)      ! Hint: May cause trouble.
SKIP:
 addi r4, r4, #12  ! Maybe add r0, r0, r0 substituted.
```



Problem 2: The program below executes on a 2-way, dynamically scheduled superscalar processor. The processor's features are summarized in the list below, the table gives details of the functional units. (The load/store unit computes the address in its first cycle and does the actual access in its second cycle. An operation does not enter the load/store unit until all its operands are ready.)

- ♢ Two-way superscalar instruction issue.
- ♢ Dynamically scheduled (see table), register renaming.
- ♢ CDB can accommodate results from any two functional units in any cycle.
- ♢ Reorder buffer for speculative execution and precise exceptions.
- ♢ Reorder buffer can retire as many as two instructions per cycle.
- ♢ Reservation stations, *not* reorder buffer, used for renaming.
- ♢ Zero-delay branch and branch target prediction. **No** branch folding.
- ♢ Branches do not have delay slots.

Name	Abbreviation	Latency	Initiation Interval	Reservation Station Nums
Load/Store	L	1	1	0-1
Integer	EX	0	1	2-3
F.P. Add	A	1	1	4-5
F.P. Mul.	M	5	2	6-7
Branch	BR	0	1	8-9
F.P. Divide	DIV	22	23	10-11

```
! When loop first entered r2-r1 large (loop iterates many times).
LOOP: ! LOOP = 0x1000
 lf   f0, 0(r1)     ! Don't overlook true dependencies on f0!
 mulf f0, f0, f2
 addf f0, f0, f3
 sf   4(r1), f0
 addi r1, r1, #8
 slt  r3, r1, r2
 bnez r3, LOOP
 div  f4, f5, f6
```

(*a*) Using the grid on the next page show the execution of the code above up to and including the last cycle shown on the grid. Assume perfect branch and branch target prediction, and no cache misses. Include instructions even if they have not yet finished executing at the end of the grid. (10 pts)

(*b*) Either determine and justify the CPI of an execution of a large number of iterations of the loop (ignoring cache misses and assuming perfect target prediction) or explain why it cannot easily be determined from the pipeline execution diagram. (A correct explanation of why it cannot be easily determined will get full credit, a correct CPI that left no time for problems 3 and 4 will get full credit for this subproblem and sympathy—but not credit—for omitting the others.) (4 pts)

First Iteration, Third Iteration, etc.?

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
`lf f0,0(r1)`																						
`mulf f0,f0,f2`																						
`addf f0,f0,f3`																						
`sf 4(r1),f0`																						
`addi r1,r1,#8`																						
`slt r3,r1,r2`																						
`bnez r3,LOOP`																						
`div f4,f5,f6`																						

Second Iteration, Fourth Iteration?, etc.?

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
`lf f0,0(r1)`																						
`mulf f0,f0,f2`																						
`addf f0,f0,f3`																						
`sf 4(r1),f0`																						
`addi r1,r1,#8`																						
`slt r3,r1,r2`																						
`bnez r3,LOOP`																						
`div f4,f5,f6`																						

(*c*) Suppose the second time the load (`lf`) executes it triggers a page fault exception, perhaps due to a bad load address.

In what cycle will the exception occur? Show the contents of the reorder buffer at that cycle and place a check mark next to instructions that have completed execution.

Explain how the reorder buffer will allow the exception to be precise. Specify what happens to the reorder buffer as a result of the exception, when it happens, and the contents of the buffer before and after it happens. At what cycle will the trap be inserted in the pipeline? (If solutions to the previous parts are not ready, make up a pipeline execution diagram just for this question.) (8 pts)

(*d*) What are the minimum number of reservation stations of each type needed to attain a minimum CPI on the code above? What is that CPI? (There is a grid on the next page to work this out, other methods may be faster.)(8 pts)

First Iteration, Third Iteration, etc.?

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
`lf f0,0(r1)`																						
`mulf f0,f0,f2`																						
`addf f0,f0,f3`																						
`sf 4(r1),f0`																						
`addi r1,r1,#8`																						
`slt r3,r1,r2`																						
`bnez r3,LOOP`																						
`div f4,f5,f6`																						

Second Iteration, Fourth Iteration?, etc.?

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
`lf f0,0(r1)`																						
`mulf f0,f0,f2`																						
`addf f0,f0,f3`																						
`sf 4(r1),f0`																						
`addi r1,r1,#8`																						
`slt r3,r1,r2`																						
`bnez r3,LOOP`																						
`div f4,f5,f6`																						

Problem 3: A system has a 64-bit address space ($a = 64$), addresses 16-bit characters ($c = 16$), and has a 256-bit data bus ($w = 256$).

(a) Show how memory devices could be connected to construct a 2-way set-associative cache with 1024-bit lines and 64 sets. Memory devices of any size can be used, be sure to specify their sizes (*e.g.*, $x\,\mathrm{b} \times 2^y$). Show only the connections needed to retrieve the data and tag information, determine if the access is a hit, and pass the data to the CPU. (10 pts)

(b) Suppose the cache is write-through. What is the capacity of the cache and how much memory is needed to implement it? Be sure to specify units. (5 pts)

(c) Find the addresses specified below. (5 pts)

- *Three* different address that are part of the same line and require exactly two loads to access. (Assume there are 256-bit load instructions.)

- Two addresses that can be in different lines but the same set.

- Three addresses that are in different sets.

Problem 4: Answer each question below.

(a) What is branch folding? In the implementations of DLX covered in class, why must the predictions used for branch folding always be correct? (6 pts)

(b) Why might the cost of a functional unit with an initiation interval of 2 be less than one performing the same operations but with an initiation interval of 1 but having the same latency? Given such a cost relationship, what should the minimum number of reservation stations be for a functional unit with an initiation interval of ι and a latency of λ? Explain. (6 pts)

(c) What are some difficulties that might be encountered in developing a superscalar implementation of a stack ISA? (For partial credit, list some distinguishing features of a stack ISA.) (6 pts)

(d) Explain how a reservation register can be used to detect MEM-stage structural hazards while an instruction is in the ID stage. (6 pts)

(e) What is the difference between a RAW hazard and a true dependency? (6 pts)