UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EECE 476 – Computer Architecture
Final Exam
July, 2009

Time Allowed: 2.5 hours

Instructor: Dr. Tor Aamodt

Instructions:

1. **Read each question carefully before attempting to solve it.**
2. **Questions are <u>not</u> ordered in increasing level of difficulty, so if you get stuck on a question, considering looking at other questions.**
3. **Please attempt to solve <u>all</u> questions.**
4. **State any assumptions you think are necessary, but ensure your assumptions are reasonable.**
5. **Keep all your answers on the exam sheet. There is extra space for answering questions on the last two pages, which you can use if you need to.**

LAST Name: Solution
First Name: _____

Student Number: _____

Q1.	/ 8
Q2.	/ 10
Q3.	/ 10
Q4.	/ 7
Q5.	/ 8
Q6.	/ 6
Q7.	/ 6
Q8.	/ 5
Q9.	/ 5
Q10.	/ 8
Q11.	/ 6
Total:	/ 79

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Question 1: [8 marks] "Computer Architecture Jeopardy"

These problems are similar to the "Jeopardy Game" on TV. The answers are shown and you are to provide the *best* correct questions. For each answer there may be more than one *appropriate* question: you need to provide the *best* one.

(a) Answer: This hardware structure "guesses" whether a branch is "taken" or "not-taken".

Question: Branch Predictor ?

(b) Answer: This provides each program with the appearance of its own large memory space independent of how many programs are running together on a computer.

Question: Virtual Memory ?

(c) Answer: This hardware enables the reduction or elimination of stalls otherwise required to avoid RAW hazards in pipelined microprocessors.

Question: Forwarding ?

(d) Answer: This type of cache miss occurs more frequently when the number of processors in a shared memory multiprocessor is increased.

Question: Coherence misses ?

(e) Answer: This class of instruction set architecture uses instructions which access a source register that is then used as a destination register, but this register is not explicitly encoded as part of the instruction.

Question: accumulator ?

(f) Answer: This causes a RAW hazard if hardware reorders two instructions.

Question: true data dependence ?

(g) Answer: This limits the number of instructions that can enter the writeback stage each cycle in Tomasulo's algorithm.

Question: common data bus ?

(h) Answer: This *view of memory* ensures that a read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.

Question: coherent ?

Question 2 [10 marks]: TRUE OR FALSE: Label each statement with "T" or "F". Be very clear. Answers that look like a combination of "T" and "F" will be marked wrong.

(a) If performance improves by 10% for a silicon area increase of 10%, then the ratio of performance per unit cost (e.g., in dollars) stays the same. F

(b) It is possible to get a speedup by a factor of 7 (i.e., "seven times faster") by optimizing the performance a portion of a program that originally used up 80% of the overall execution time of the program. F

(c) The scoreboard algorithm may need to stall an instruction due to a "name dependency" even when there is no RAW hazard. T

(d) Instructions in the MIPS64 instruction set architecture are encoded using 32-bits. T

(e) The store instruction "SW R1,0(R2)" modifies the contents of register R2. F

(f) The memory alignment property imposes the requirement that different levels of cache use the same block size in a RISC microprocessor system. F

(g) A write-through invalidate cache coherence protocol uses a state machine with two states (per cache block). T

(h) The immediate addressing mode encodes one of the source operands for an instruction as part of the instruction itself, rather than getting the value from a register or memory. T

(i) The there is no benefit to using reservation stations if a processor has a reorder buffer since a reorder buffer already supports precise interrupts. F

(j) An advantage of snoop based cache coherence protocols is they are more scalable than directory based cache coherence. F

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Question 3 [10 marks]: Short answer.

(a) What is the difference between a load instruction and a store instruction?

a load reads from memory while a store writes to memory

(b) What is the motivation for using "multiple issue" to improve performance?

to reduce ideal CPI below 1.0

(c) What is a precise exception?

when hardware ensures no instruction after the faulting instruction has updated programmer visible state and all instructions before the faulting instruction have.

(d) When might a branch predictor that uses a two-bit state machine be expected to be more accurate than a predictor that uses a single-bit assuming the same number of rows are used in the tables for both predictors.

when branches are biased to be mostly taken, or mostly not taken (so one misprediction should not change the prediction)

(e) A benchmark suite contains a number of different programs. How is it possible to compare performance of several computers on this benchmark suite, when a different program is fastest on each different computer?

by averaging the speedups relative to a base machine (eg using the geometric mean)

(f) What does SIMD stand for?

single instruction, multiple data

(g) When is an invalidation message sent in a directory cache coherence protocol?

when a processor wants to modify a block in the shared or modified state in the directory and not in modified state in its cache.

(h) Give a brief description of "false sharing".

this occurs when two (or more) processors access a single cache line invalidating each other's copy, but no communication takes place between them using that line.

(i) The outcomes ("taken", "not-taken") of a particular branch follow a pattern that appears random. How might it be possible for hardware to predict the outcomes of this branch with close to 100% accuracy?

by correlating on an earlier branch

(j) Briefly describe how load locked (LL) and store conditional (SC) are able to provide the functionality of an atomic read-modify-write, but without the hardware having to guarantee that a LL-SC pair is always executed atomically.

by optimistically loading the value and then checking as part of the store whether another processor had accessed the same cache block as the load locked before completing the store operation.

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Question 4 [7 marks total]: You are trying to figure out whether to build a fabrication facility to manufacture your company's microprocessors or to pay an external dedicated chip foundry to fabricate your chips for you. It costs \$1 billion to build a new fabrication facility. If your company builds a fabrication facility, there will be less money to design its next generation microprocessor, likely reducing the price it can be sold for. On the other hand, your company will be able to reduce wafer costs if it has its own fabrication facility. Assume that if your company builds a fabrication facility it will develop a microprocessor code named "Takakkaw" with an area of 240 mm^2 that would sell for \$200 each. If your company pays an external dedicated chip foundry it will instead develop a microprocessor code named "Swiftcurrent" having an area of 380 mm^2 that would sell for \$500 each. In both cases assume a defect rate of 0.9 defects per cm^2, wafer yield of 1.0, wafers are 300 mm in diameter, $\alpha = 4$, negligible packaging and test costs, and a final test yield of 1.0. Assume the external dedicated chip foundry charges \$5000 per wafer but the cost per wafer if you run your own fabrication facility is \$2500 (this is on top of the \$1 billion for building the facility in the first place).

(a) [4 marks] What is the cost per chip (IC cost) of the "Swiftcurrent" microprocessor?

$$\text{dies per wafer} = \frac{\pi (300/2)^2}{380} - \frac{\pi \cdot 300}{\sqrt{2 \cdot 380}} = 186.015 - 34.187 = 151.828$$

round down to 151

$$\text{die yield} = 1.0 \times \left(1 + \frac{0.9 \times 3.80}{4}\right)^{-4} = 0.08445$$

$$\text{IC cost} \approx \text{Die Cost} = \frac{\$5000}{151 \cdot 0.08445} = \$392.10$$

≈ due to assumption of negligible packaging & test cost + final test yield of 1.0

(b) [3 marks] Assuming you would sell the same number of chips regardless of which chip you build, how many chips do you need to sell to earn a greater profit by building your own fabrication facility taking into account the cost of building the fabrication facility?

$$N \cdot (\$200 - \text{Cost}_{TAK}) - \$10^9 > N \cdot (\$500 - \$392.10)$$

$$\text{dies per wafer}_{TAK} = \frac{\pi (300/2)^2}{240} - \frac{\pi \cdot 300}{\sqrt{2 \cdot 240}} = 294.524 - 43.018 = 251$$

$$\text{die yield} = 1.0 \times \left(1 + \frac{0.9 \times 2.4}{4}\right)^{-4} = 0.1778$$

$$\text{IC cost} \approx \text{Die cost} = \frac{2500}{251 \cdot 0.1778} = \$56.02$$

$$N \cdot \$143.98 - \$10^9 > N \cdot \$107.9$$

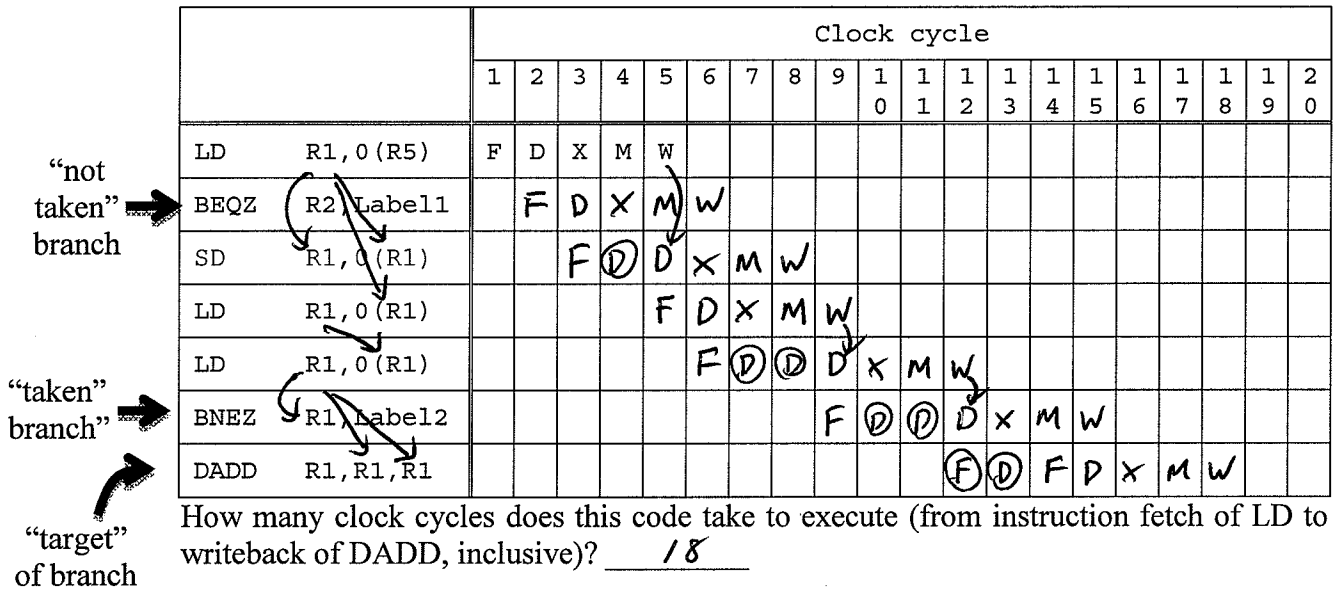
$$\Rightarrow N > \frac{10^9}{143.98 - 107.9} \Rightarrow N > 27.7 \text{ million chips}$$

(this is comparable to # of x86 CPUs sold per year by AMD)

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

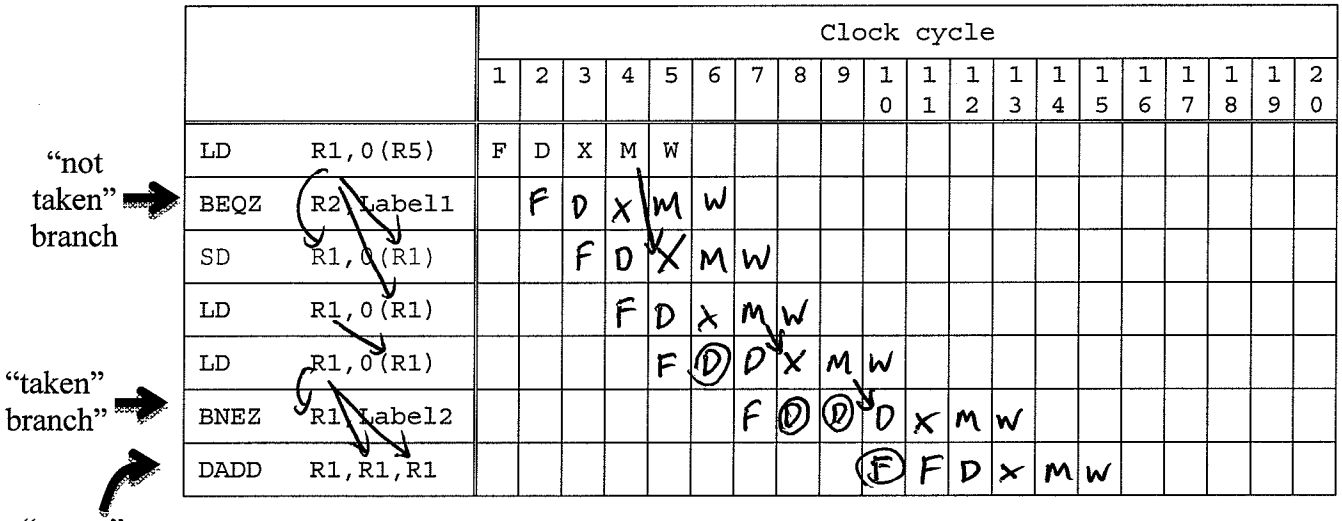
Question 5: In the following questions you determine the number of cycles it takes for some MIPS64 assembly code to execute on the simple five-stage pipeline. For all pipeline diagrams, you **must** indicate clearly where forwarding occurs using an arrow to indicate when a pipeline stage forwards a value to another pipeline stage.

5(a) [4 marks] Complete the table below to determine how many cycles it takes the following code to execute on the classic 5-stage pipeline studied in class without any support for forwarding except for *forwarding through the register file*. Assume branches are resolved in execute. Assume the lone branch in the code is "taken".



How many clock cycles does this code take to execute (from instruction fetch of LD to writeback of DADD, inclusive)? 18

5(b) [4 marks] Now analyze the *same* code on the classic 5-stage pipeline studied in class assuming complete support is included for all forwarding paths required to reduce or eliminate stalls and assuming branches are resolved in decode.



How many clock cycles does this code take to execute (from instruction fetch of LD to writeback of DADD, inclusive)? 15

Question 6 [6 marks]: Consider the distributed shared-memory system illustrated on the next page which uses a writeback invalidate protocol. Each processor has a single direct-mapped cache that holds 4 blocks each with 2 words (recall that a word is four bytes). In the figure the two words in a block are separated by a dotted line (least significant word on the right). For simplicity the tag field contains the full address (in hexadecimal).

The cache states are defined as follows:

- M: modified
- S: shared
- I: Invalid

The directory states are denoted DM, DS, and DI for Directory Modified, Directory Shared, and Directory Invalid.

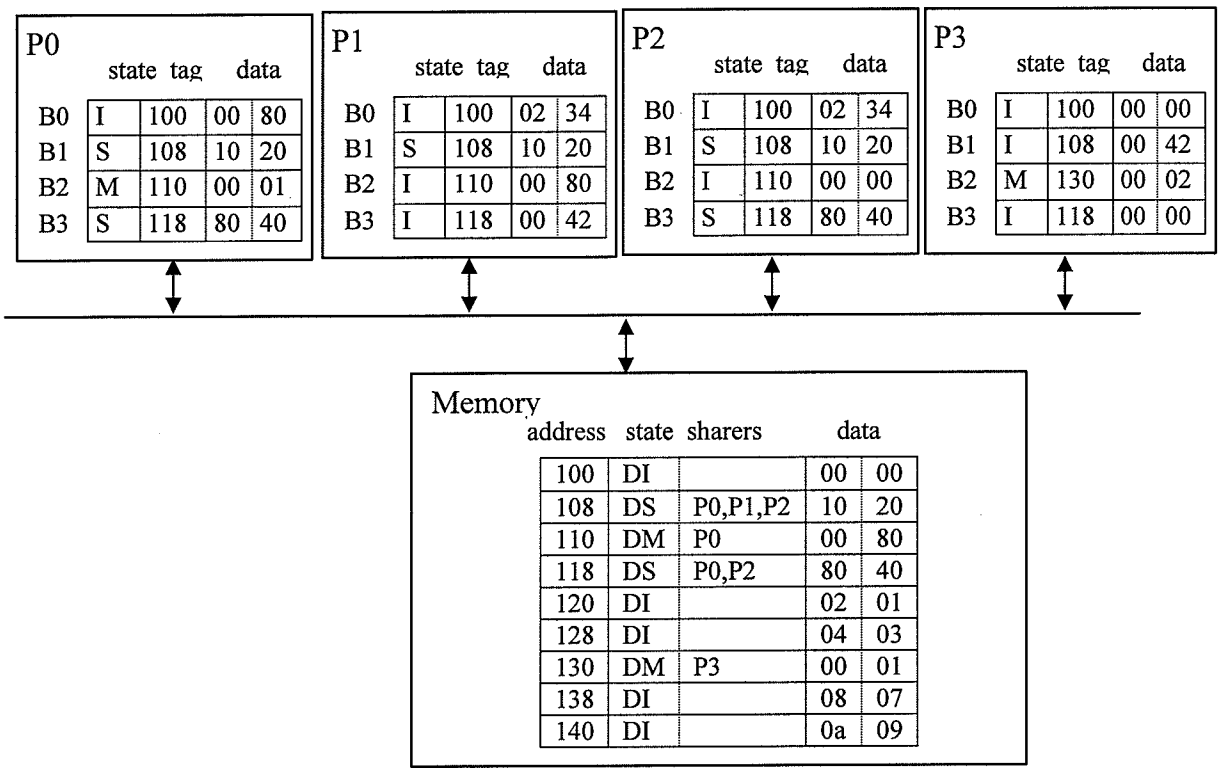
Assume CPU operations of the form:

```
P#: <op> <memory address> [<-- <value>]
```

Where P# designates the CPU (e.g., P0) , <op> is the CPU operation (read or write a single word), and <value> indicates the new word to assign during a write operation.

For each part of this question (a), through (f) assume the initial cache and memory state in the figure on the following page. Show the final state (coherence state, address tag, and data) for the changed lines (either data or state) in each cache and memory. Also, what is the value returned by each read operation?

Use this notation: "P#.Block#: (new state, address, new block contents)" to denote a particular block in a particular processor. For memory changes use "M.*address*: (new directory state, new sharers, new block contents)".



P0

	state	tag	data	
B0	I	100	00	80
B1	S	108	10	20
B2	M	110	00	01
B3	S	118	80	40

P1

	state	tag	data	
B0	I	100	02	34
B1	S	108	10	20
B2	I	110	00	80
B3	I	118	00	42

P2

	state	tag	data	
B0	I	100	02	34
B1	S	108	10	20
B2	I	110	00	00
B3	S	118	80	40

P3

	state	tag	data	
B0	I	100	00	00
B1	I	108	00	42
B2	M	130	00	02
B3	I	118	00	00

Memory

address	state	sharers	data	
100	DI		00	00
108	DS	P0,P1,P2	10	20
110	DM	P0	00	80
118	DS	P0,P2	80	40
120	DI		02	01
128	DI		04	03
130	DM	P3	00	01
138	DI		08	07
140	DI		0a	09

Analyze the effect of following sequences (there is additional space on the next page):

(a) P0: read 100

P0.B0(S,100,00 00) returns 0

M.100 C DS, P0, 00 00)

(b) P0: read 110

no change returns 01

(c) P1: write 108 <-- 46

P0.B1(I,108,10 20) P2.B1(I,108,10 20)

P1.B1 (M,108,10 46) M.108(DM, P1, 10 20)

(d) P0: write 110 <-- 15

P0.B2(M,110,00 15)

(e) P2: write 11c <-- 07

P0: read 118 returns 40

P0.B3(S,118,0740) P2.B3(I,118,0740) M.118(DS,P0,0740)

~~P1.B3(I,118,~~

(f) P0: write 134 <-- 54

P2: read 110 returns 01

P3: write 130 <-- 08

~~P0.B2(I,130,54 08) P0:~~

~~P2.B2(S,~~ next page

Additional space for question 6:

(i) (i) M.110(DI, ~~[illegible]~~, 00 01)

~~(ii)~~ P0.B2(M, ~~S4~~ 130, S4.02)

M.130(DM, ~~[illegible]~~ P0, 00 02).

(ii) M.110(OS, P2, 00 01) *

P2.B2(S, 110, 00 01) *

(iii) P0.B2(I, 130, S4 02) *

M.130(DM, P3, S4 02) *

P3.B2(M, 130, S4 08) *

Question 7 [6 marks] Determine how many times a **1-bit** branch predictor with **2-bits of correlation** correctly predicts a branch for the following code assuming it is executed 5 times and ignoring other branches (including any branch needed to cause the code to be executed repeatedly) with register R1 initially equal to the value in the left hand column of the table below each time the code is encountered. Also, assume each branch has it's own entry in the 1-bit predictor tables, the 1-bit predictor table entries are initialized to N ("not taken"), and the correlation bits are all initialized to 0.

```
        BEQZ    R1,L2       ; branch b1
        DADDI   R3,R1,#-1
        BNEZ    R3,L1       ; branch b2
        ...
L1:     DADDI   R3,R1,#-2
        BNEZ    R3,L2       ; branch b3
        ...
L2:
```

Fill in the following table, where each row represents a pass through the code. For each entry under a "prediction" heading, indicate the values in all tables separated by a slash (/). For each branch prediction, circle the entry used to make predictions. Note that if "b1" is taken it "skips" past "b2" and "b3", so if this happens leave the entries for "b2" and "b3" empty in the row corresponding to current iteration.

R1 = ?	b1 prediction	b1 action	new b1 prediction	b2 prediction	b2 action	new b2 prediction	b3 prediction	b3 action	new b3 prediction
-1	N/N/N/(N)	NT	N/N/N/N	N/N/N/(N)	T	N/N/N/T	N/N/(N)/N	T	N/N/T/N
1	(N)/N/N/N	NT	N/N/N/N	N/(N)/N/T	NT	N/N/N/T	N/N/T/(N)	T	N/N/T/T
2	N/N/(N)/N	NT	N/N/N/N	N/(N)/N/T	T	N/T/(N)/T	N/N/(T)/T	NT	N/N/N/T
1	N/(N)/N/N	NT	N/N/N/N	N/T/N/(T)	NT	N/T/N/N	N/N/N/(T)	T	N/N/N/T
0	N/N/(N)/N	T	N/N/T/N						
2	(N)/N/T/N	NT	N/N/T/N	N/(T)/N/N	T	N/T/N/N	N/N/(N)/T	NT	N/N/N/T

Number of mispredictions: 7

Question 8 [5 marks] Consider the following MIPS64 assembly code:

```
DIV.D   F2,F4,F6
MULT.D  F2,F2,F8
S.D     F2,0(R1)
DIV.D   F8,F10,F12
ADD.D   F2,F6,F8
```

Evaluate the number of clock cycles it takes the above code to execute assuming a single-issue Tomasulo's algorithm.

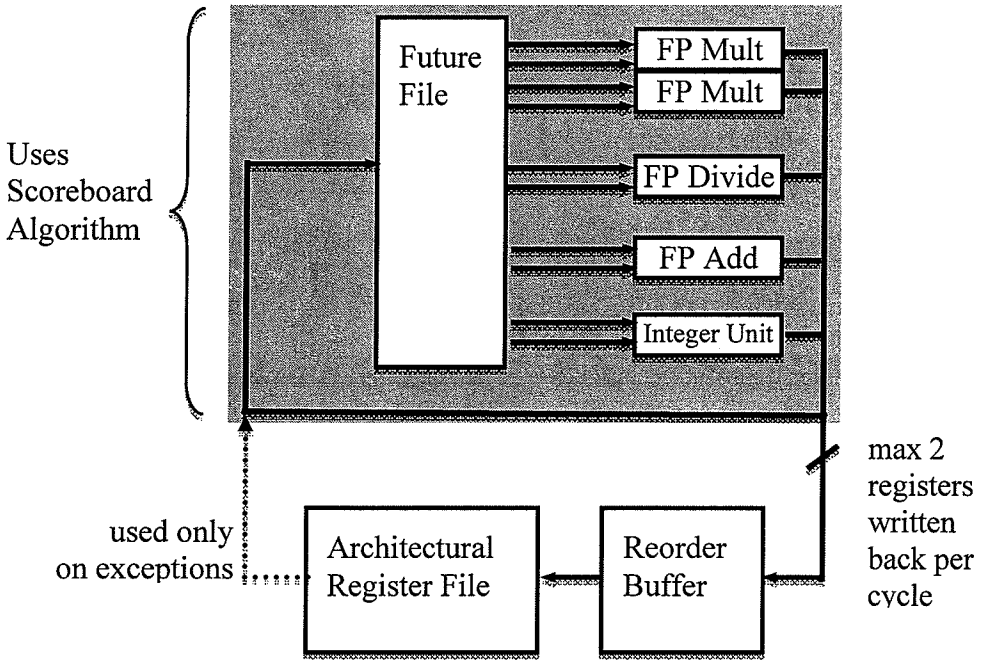
Assume the following:

- All function units are fully pipelined (a new operation can start on the function unit each cycle provided it source operands are available).
- There is no forwarding between function units; results are communicated by the common data bus (CDB).
- ~~Up to one instruction can commit per cycle.~~
- The issue (IS), and write result (WB) stages each take 1 cycle.
- The execution stage (EX) does both the effective address calculation and the memory access for loads. For store instructions, the execution stage (EX) takes a single cycle and only does the effective address calculation while the WB stage writes the store value to memory.
- DIV.D takes 20 cycles in EX; ADD.D takes 2 cycles in EX; MULT.D takes 10 cycles in EX.
- There are as many reservation stations as necessary.

	IS	EX	WB	Comment
DIV.D F2,F4,F6	1	2	22	
MULT.D F2,F2,F8	2	23	33	
S.D F2,0(R1)	3	4	34	
DIV.D F8,F10,F12	4	5	25	
ADD.D F2,F6,F8	5	26	28	

Question 9 [5 marks]: In class we learned about how the reorder buffer could support precise exceptions when using Tomasulo's algorithm. The reorder buffer can also be used in a computer that uses the Scoreboard algorithm. One way to do this is shown in the figure below in the context of a multiple issue processor that can issue two instructions per cycle. The top portion (shaded area) operates like the normal scoreboard algorithm except that two operands can issue and writeback on a given clock cycle. The box labeled "Future File" acts as the register file during normal operation when there is no exception. The "Architecture Register File" keeps the precise state of the machine in case of exceptions. Initially, the contents of the Future File and Architecture Register File are exactly the same. Values are read by the function units from the Future File and written back into the Future File just as they were read and written from/to the register file in the normal Scoreboard algorithm. The reorder buffer in the lower part of the figure has one entry per instruction in the pipeline, which is allocated in program order when an instruction issues. As each instruction writes its results (possibly out of order) the result is written both into the Future File and the Reorder Buffer. Once an instruction becomes the oldest instruction in the processor, its value is committed from the Reorder Buffer into the Architecture Register File. If the oldest instruction caused an exception (e.g., a "page fault") the contents of the Reorder Buffer is flushed and the contents of the Architectural Register File is copied into the Future File and control is transferred to an exception handler.

note that you probably wouldn't want to build a processor this way



On the following page, you will analyze an instruction sequence executing on the above hardware organization under some additional assumptions (outlined on the next page).

Assume the following:

- The pipeline stages are I (Issue); RO (Read Operands); EC (Execution Complete); WB (Writeback); COM (Commit); The first four stages behave similar to the way they would with the Scoreboard algorithm. Commit is used when an instruction leaves the reorder buffer to update the Architectural Register File.
- Up to **two** instructions can issue, writeback and commit per cycle. Any number of registers can be read in a given clock cycle in RO and any number of instructions can complete EC in a given clock cycle.
- Function units are not pipelined. Function units become available the cycle after the WB stage of the prior instruction to use the function unit.
- There is no forwarding between function units; results must first be written into the Future File during WB, and can only be read during RO the following clock cycle.
- The issue I, RO, WB, and COM stages each take 1 cycle.
- There are two floating point multiply units, one floating-point divide unit, one floating point addition unit, and one integer unit.
- The execution stage does both the effective address calculation and the memory access for loads using the integer unit in a single clock cycle.
- DIV.D takes 20 cycles in EX; ADD.D takes 2 cycles in EX; MULT.D takes 10 cycles in EX.

Fill in the table below indicating the clock cycle when each instruction reaches the corresponding stage of the pipeline. **State any additional assumptions you feel are necessary but ensure your assumptions are reasonable.**

	I	RO	EC	WB	COM	Comment
DIV.D F2,F4,F6	1	2	22	23	24	
MULT.D F4,F2,F8	1	24	34	35	36	RO: F2
L.D F10,0(R1)	2	3	4	5	36	
MULT.D F8,F10,F12	2	6	16	25	37	RO: F10, WB: F8, COM in ord
ADD.D F6,F4,F14	3	36	38	39	40	RO: F4,
MULT.D F8,F10,F12	26	27	37	38	40	I: SH,
ADD.D F2,F6,F8	40	41	43	44	45	I: SH

Question 10 [8 marks total]: For question (a) and (b) below use a *3-way* (yes, I mean 3, it is not a typo) set associative cache with 2-byte blocks, 4 index bits and LRU replacement.

(a) [2 marks] What is the capacity of this cache measured in bytes? (Please show how you calculate your answer so we can attempt to give you part marks if your answer is incorrect.)

$3 \times 2 \times 2^4 = 96\ B$

(b) [3 marks] In the table below is a series of byte address references (in decimal). Label each reference in the list as a hit or a miss.

Address	Hit/Miss?
0	m
16	m
64	m
48	m
0	h
32	m
160	m
64	m
17	h
145	m
49	h
50	m
51	h
32	h

0 32 64 96 128 160 1
2
4
6
8
10
12
14
16 48 80 112 144
18 50

(c) [2 marks] List and describe two forms of "locality".

spatial locality - references to nearby locations
temporal locality - references to recently accessed locations

(d) [1 mark] What does it mean that a cache is "no write allocate"?

when a store misses in the cache, a cache line is not allocated.

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Question 11 [6 marks total]:

(a) [3 marks] Suppose the branch and memory reference frequencies (as a percentage of all instructions) are as follows: Conditional branches 15%, jumps and calls 10%, memory reference 20%. Also suppose 60% of all conditional branches are taken and 8% of all memory references miss in the data cache with a miss latency of 20 cycles. We are examining a **four**-deep pipeline with stages instruction fetch (IF), instruction decode (ID), execute (EX), and writeback (WB). The memory accesses occur during the EX pipeline stage. Jumps are resolved in ID and conditional branches are resolved in EX. Assuming branches and jumps are predicted **not-taken**, and ignoring stalls due to data dependencies, but assuming the pipeline stalls on a data cache miss, what is the average CPI assuming an ideal CPI of 1.0?

$$CPI = 1 + 0.15 \times 0.6 \times 2 + 0.1 \times 1 + 0.2 \times 0.08 \times 20$$
$$= 1.6$$

(b) [3 marks] Consider the following MIPS64 assembly code:

```
Loop:   L.D     F0,0(R1)
        ADD.D   F4,F0,F2
        S.D     F4,0(R1)
        DADDUI  R1,R1,#-8
        BNE     R1,R2,Loop
```

assume branch delay slot

Assume the number of iterations ***n*** is large but unknown. Find the theoretically *optimal* number of unrollings ***k*** as a function of ***n***, using the latencies in the following table assuming an in-order single-issue MIPS pipeline:

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Hints: (1) Recall you need two loops: one that iterates ***n/k*** times which is unrolled ***k*** times, and one that iterates ***n* mod *k*** times that is **not** unrolled. **(2)** Given that ***n*** is large, but unknown, a reasonable approximation to **(*n* mod *k*)** is simply ***k/2***.

unrolled loop cycles $= 3(n - n \bmod k) + 2\left\lfloor \frac{n}{k} \right\rfloor + 6(n \bmod k)$

$= 3n + 2\left\lfloor \frac{n}{k} \right\rfloor + 3(n \bmod k)$

$C \approx 3n + 2\frac{n}{k} + 3 \cdot \frac{k}{2}$

$\frac{dC}{dk} = 0 \Rightarrow -2\frac{n}{k^2} + \frac{3}{2} = 0$

$\Rightarrow k = \sqrt{\frac{4n}{3}}$