McGill University
ECSE 425: COMPUTER ORGANIZATION AND ARCHITECTURE
Winter 2015
Final Examination

9:00 AM – 12:00 Noon, April 22th, 2015

Duration: 3 hours

Examiner: Prof. B. H. Meyer **Associate Examiner: Prof. W. J. Gross**

- **Write your name and student number in the space below. Do the same on the top of each page of this exam.**
- **The exam is 18 pages long. Please check that you have all 18 pages.**
- **There are six questions for a total of 160 points. Not all parts of all questions are worth the same number of points; read the whole exam first and spend your time wisely!**
- **This is a closed-book exam. You may use two double-sided sheets of notes; please turn these sheets in with your exam.**
- **Faculty standard calculators are permitted; no cell phones or laptops.**
- **Clearly state any assumptions you make.**
- **Write your answers in the space provided. Show your work to receive full credit, and clearly indicate your final answer.**

Name: _____

Student Number: _____

Q1: _____/40 **Q2: _____/30** **Q3: _____/20**

Q4: _____/20 **Q5: _____/20** **Q6: _____/30**

Total:

Question 1: Are These *Short Answer* Questions? (40 pts total)

Respond to each of the following; two to four sentences should be adequate in each case.

a. (8 pts total) Consider the following memory hierarchy. L1 is virtually indexed, physically tagged, and two-way set-associative. L2 is physically indexed and 16-way set-associative. L1 is write-back; L2 is write-through; data to main memory is buffered. Describe, in order of occurrence, all of the steps taken to service a read request when:

i. (4 pts) The access results in a TLB hit, L1 cache miss, L2 cache hit.

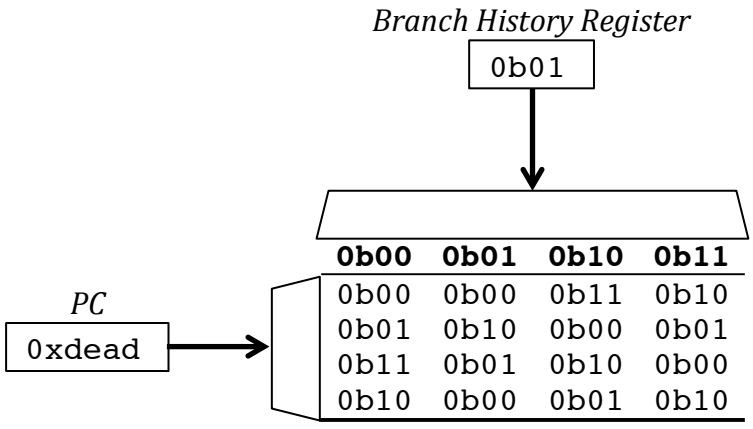
ii. (4 pts) The access results in a TLB miss, L1 cache hit.

b. (4 pts) List two reasons clock cycle time deviates from the ideal in a pipelined processor.

c. (4 pts total) Consider the following branch predictor state for a (2, 2) *correlating branch predictor*. All state is given in binary (e.g., `0bxy` where `x` and `y` are bits). Assume `0b1` indicates *taken* while `0b0` indicates *not taken*.

i. (2 pts) Given a branch instruction at address `0xdead`, *circle the branch predictor state* that is used to determine whether to take or not take the branch. *Hint*: not all state is used; don't circle everything.

ii. (2 pts) What prediction is made? Indicate how predictor state will be updated.



d. (8 pts) Schedule the following code for execution on hardware capable of (i) *fine-grained multi-threading* (FGMT) and (ii) *simultaneous multi-threading* (SMT). Assume that two identical threads are available; schedule a single iteration of the loop for each. Assume both the FGMT and SMT processors issue up two instructions of any type per cycle. Assume that integer instructions require one (1) cycle, memory accesses require two (2) cycles (addresses are calculated in the first cycle, and memory is accessed in the second), and floating point addition (fully pipelined) requires four (4) cycles. Complete the table, indicating the instruction (Inst X) issued and the thread (Thrd X) from which it is issued, in each cycle.

```
L0: L.D     F0,0(R1)
    L.D     F6,-8(R1)
    ADD.D   F4,F0,F2
    ADD.D   F8,F6,F2
    S.D     F4,0(R1)
    S.D     F8,-8(R1)
    DADDUI  R1,R1,#-16
    BNE     R1,R2,L0
```

FGMT

Cycle	Thrd 1 Inst 1	Thrd 2 Inst 2
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		

SMT

Cycle	Thrd 1 Inst 1	Thrd 2 Inst 2
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		

e. (8 pts) Schedule the following, unrolled, code for VLIW execution on a processor capable of issuing an integer (Int) instruction, floating-point instruction (FP), and memory access instruction (Mem) each cycle. Complete the table, indicating which instructions are issued each cycle, under the assumption that integer instructions require one (1) cycle, memory accesses require two (2) cycles (addresses are calculated in the first cycle, and memory is accessed in the second), and floating point addition (fully pipelined) requires four (4) cycles.

```
L0: L.D     F0,0(R1)
    L.D     F6,-8(R1)
    L.D     F10,-16(R1)
    L.D     F14,-24(R1)
    ADD.D   F4,F0,F2
    ADD.D   F8,F6,F2
    ADD.D   F12,F10,F2
    ADD.D   F16,F14,F2
    S.D     F4,0(R1)
    S.D     F8,-8(R1)
    S.D     F12,-16(R1)
    S.D     F16,-24(R1)
    DADDUI  R1,R1,#-32
    BNE     R1,R2,L0
```

Cycle	Int	FP	Mem
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

f. **(2 pts)** What is the performance advantage of multi-lane SIMD hardware?

g. **(2 pts)** What is the challenge associated with exploiting the performance advantage of multi-lane SIMD hardware?

h. **(2 pts)** Why do vector architectures generally not utilize caches?

i. **(2 pts)** How do GPU architectures hide memory access latency?

Question 2: Cache with Cachet (30 pts)

Consider a hierarchical cache with the following contents. All state, tags and data are in hexadecimal format. Assume that L1 and L2 are *exclusive*. Assume *little-endianness*, i.e., that the least significant byte in a word is stored in the least significant address.

L1 Cache

Set	Valid	Dirty	Tag	Data
0	0	0	34	83 ab 57 88 31 e7 a0 5e 8f 4c e4 df 63 77 c6 5e
1	0	0	1f	d2 ce aa 7c 82 23 d7 d7 1f 64 76 c7 a3 dc 19 78
2	1	0	d3	e6 65 8d 3b e3 53 f8 d7 f8 40 3d f2 80 42 0a f0
3	1	1	2d	0c b8 00 e4 d7 60 58 b0 71 4e 10 f7 b9 5f 58 c7
4	0	0	67	5f 91 85 64 de 45 ca e8 84 b7 c0 41 6e 5c d4 64
5	1	1	d3	be 8a 35 b0 9b 75 70 48 08 06 a0 bd 13 c6 69 a0
6	0	0	8d	2a a4 7b bd 72 a1 f7 51 19 3b 55 fc d8 52 30 35
7	1	0	00	30 4b 12 c0 2f 51 8d 3a 8e 57 c7 8b be 44 93 a2
8	1	0	ab	df cd 5a f7 7f 4b cd b3 58 a5 4d 46 f5 30 91 e6
9	0	0	12	ee f9 86 7d 24 dd 98 7c 8c a7 de ec bc a5 06 cb
10	1	1	e3	2c 5b af c6 92 09 cf ca a9 e5 f2 87 d2 91 9b 1d
11	1	0	df	e8 af 5b 2e 4a 77 e0 0c 1d d0 ce 0b b8 ac a6 9c
12	0	0	ee	9d b7 11 e9 68 7c a3 4f 91 17 2f b3 3d 85 92 33
13	1	0	1d	cd 8b 2f 5d c8 de 16 90 d2 4e e3 20 12 83 f2 62
14	0	0	a0	d5 d0 a3 4b 9e 1a d4 55 a3 9b 1d 98 26 38 a5 e0
15	1	1	55	a5 65 2b 9f 55 20 72 e6 25 23 07 e1 bf 7f 72 1e

L2 Cache

Set	Valid	Dirty	Tag	Data
0	0	0	22	d0 3c 82 d0 e4 04 93 50 9c 4f 9a 4a f1 6d 21 22
1	1	1	d2	54 9d 9b 59 ba aa a5 d0 66 ea 4d 7c b6 ef 9d 9d
2	1	0	17	05 ff 99 48 d9 a8 86 0b 07 60 74 60 97 42 c0 51
3	1	0	8d	ea b4 e8 01 cf 8c 54 96 3a 8f 5e b1 d1 58 53 07
4	0	0	88	20 03 80 6e 6b c1 71 3b 12 2f 3c 59 f0 6c f9 1c
5	1	0	2d	99 ec 4f d6 40 df 28 4d 0f 8f 64 66 ec 10 ff cc
6	1	1	22	ec de a2 6c 4b 93 ad f4 5b ed 48 2a cf b4 ca 45
7	1	0	8d	97 73 42 76 bb e0 4c 40 bd eb 5c b1 74 72 6a c4
8	0	0	a9	10 08 90 d3 59 48 d4 5d af 5c 97 b7 66 f2 fa 3c
9	0	0	0e	85 8f 34 e7 8a f9 2e d9 c5 1f 21 85 88 a0 03 15
10	1	1	31	5c 6a ec f3 e0 72 84 97 55 65 cc a1 7b 9d 61 24
11	1	0	b8	77 93 42 c7 a1 98 44 2b 15 90 1f d7 cf a5 94 40
12	1	1	ab	e5 05 bf ac 67 2d 7b f1 76 e9 cc 16 7d e8 1a c3
13	1	0	4d	c9 27 66 71 42 df 83 6d ca a9 59 e3 a9 fe 15 18
14	0	0	ab	5c ea ab 14 8a fa 11 b0 00 40 46 6c 72 89 0f f8
15	1	1	1c	a0 0b 47 8d b1 3e 6b 42 7d be 76 b8 1e 0c 45 88

a. **(4 pts)** What data is delivered when a read request is made for a 16-bit word at `0xabcd`? Indicate the changes made to any cache blocks in L1 or L2 as a consequence of servicing the request.

b. **(6 pts)** Indicate the changes made to the cache (both L1 and L2) when `0x1234` is written to address `0x8d34`.

Now consider a pipelined processor that runs at 2 GHz and has a base CPI of 0.9 when all cache accesses hit. The only instructions that read data from or write data to memory are loads (15% of all instructions) and stores (10% of all instructions).

The memory system for this computer is composed of a split L1 cache. Both the I-cache and D-cache are 2-way set-associative and hold 32 KB each. The I-cache has a 1% miss rate; the D-cache has a miss rate of 15%.

The 1 MB inclusive, unified L2 cache has an access time of 20 ns. Of all memory references sent to the L2 cache in this system, 80% are satisfied without going to main memory. Main memory has an access latency of 200 ns. Assume that L2 and main memory transfer times are accounted for in the access times above.

Assume that all caches use a write-back policy, and 40% of evicted data is dirty.

c. (4 pts) What is the average memory access time (in cycles) for instruction accesses?

d. (4 pts) What is the average memory access time (in cycles) for data accesses?

e. (4 pts) What is the overall CPI of the system?

f. (4 pts) What speedup is achieved if an L3 cache is added with a *local hit rate* of 90% and access time of 40 ns?

g. (4 pts) Rather than improve the memory hierarchy as in part (d), would it be more advantageous to adjust compilation to reduce the number of memory accesses? What speedup is achieved if the fraction of loads and stores is reduced to 10 and 5% respectively? Assume the optimization removes the memory accesses and does not replace them with other instructions.

Question 3: You're So Predictable (20 pts)

Two computers A and B are identical except for their branch prediction schemes. Each computer uses a seven-stage pipeline:

IF ID1 ID2 EX1 EX2 MEM WB

Computer A uses a static predicted not-taken scheme. Computer B uses a static predicted taken scheme. In each case, branch targets are calculated in ID2, and branch conditions are resolved in EX2.

If 20% of instructions are conditional branches, what fraction of these branches must be taken for the two computers to have equivalent performance? Assume each computer achieves the ideal CPI for every other instruction other than conditional branches.

Question 4: UnRFLMAO! (20 pts)

Consider the following C code.

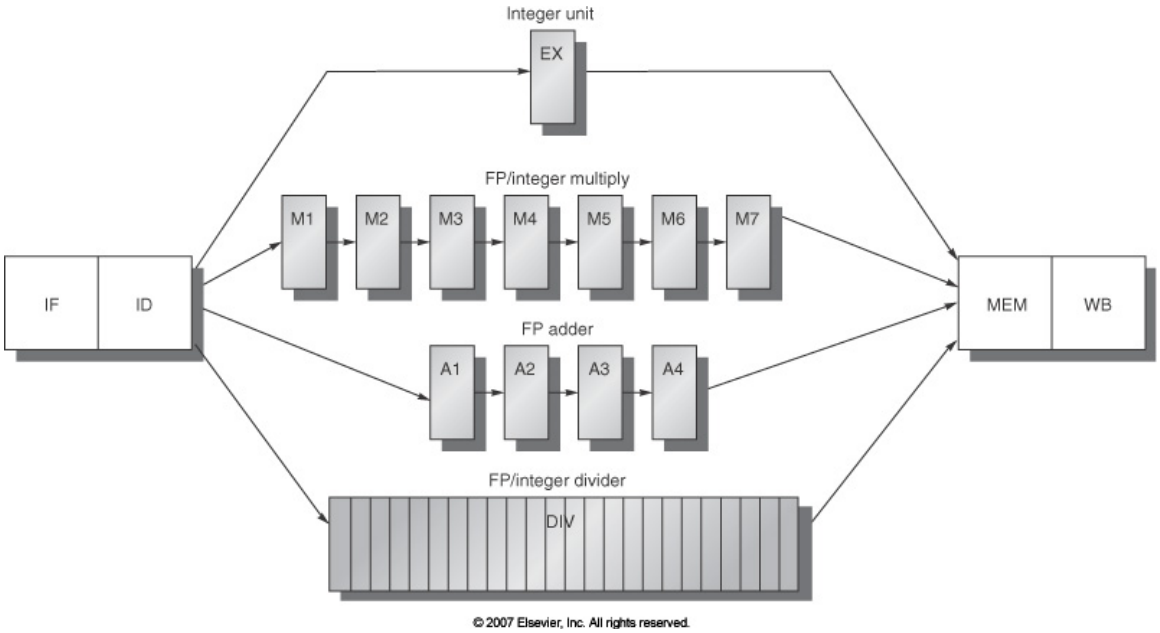
```
for (int i=n; i≥0; i++) {
     sum[i+1] = sum[i] + a;
}
```

This code can be implemented with the following assembly.

```
       LI      R2, n
       LI.D    F2, a
Loop:  L.D     F4, 0(R1)
       ADD.D   F6, F2, F4
       S.D     F6, 8(R1)
       DADDUI  R1, R1, #8
       DADDUI  R2, R2, #-1
       BNEZ    R2, Loop
```

`LI` loads an immediate value in an integer register; `LI.D` loads a double-precision floating-point value in a floating-point register using the FP adder.

Unroll the loop once and reschedule the instructions to minimize delay within the loop body. Assume the standard MIPS floating point pipeline (forwarding, split L1 caches, etc):



Further assume that control hazards are managed using a *branch delay slot*. On the following page, write out the re-scheduled instructions, including their operands. Insert placeholders for any stalls not eliminated by rescheduling.

Additional Page

Question 5: This One is Double-Wide! (20 pts)

Consider once more the code from Q4 (note that `n` has been set to 2, and `a` to 8).

```
        LI      R2, #2
        LI.D    F2, #8
Loop:   L.D     F4, 0(R1)
        ADD.D   F6, F2, F4
        S.D     F6, 8(R1)
        DADDUI  R1, R1, #8
        DADDUI  R2, R2, #-1
        BNEZ    R2, Loop
```

This time, execute the code to completion assuming a *super-scalar* out-of-order processor with support for hardware speculation, and the following functional units:

Functional Unit	Cycles to Execute	No. of Functional Units
Integer ALU	1	4
FP Adder (used by LI.D, too)	4	2
FP Multiplier	7	1
Load/Store	*2	4

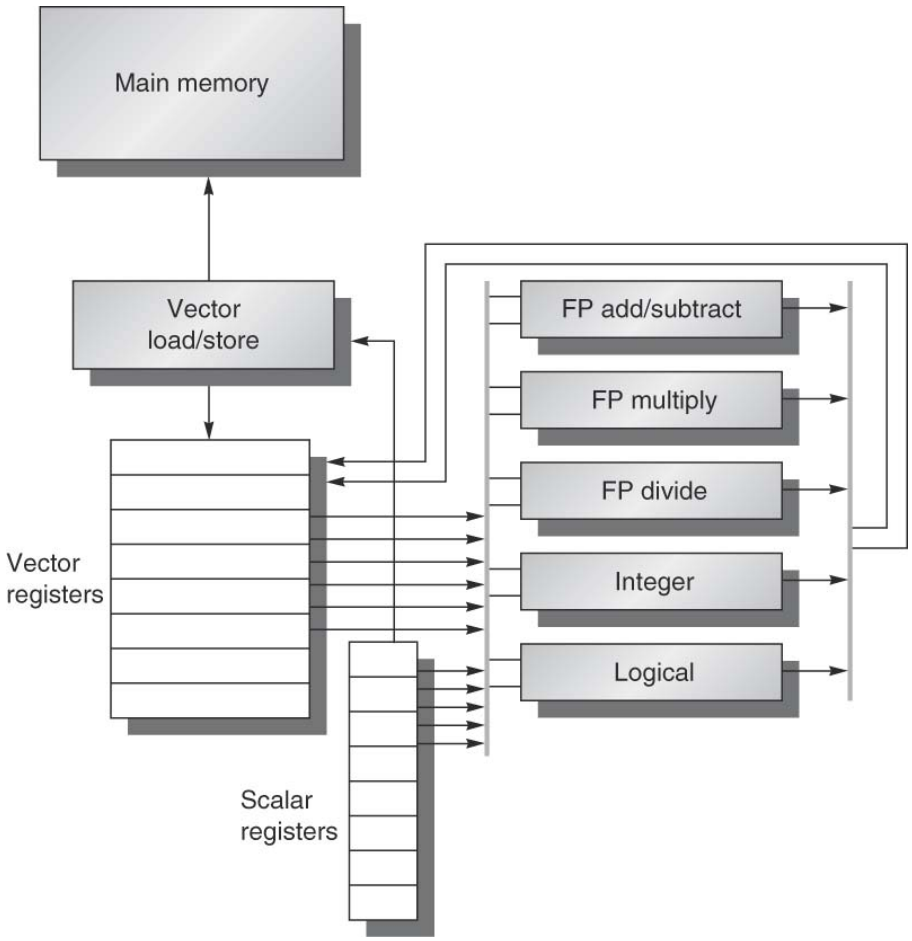
* The first cycle of Load/Store execution is used to calculate the effective address, and can proceed even if other operands are not yet ready.

Assume: the processor can issue and commit *two instructions per cycle*, a single reservation station per functional unit, and *perfect branch prediction*. Complete the following table.

Inst.	Inst. Issue	Begin EX	Finish EX	Write CDB	Commit
`LI R2, #2`	1	2	2	3	4
`LI.D F2, #8`	1	2	5	6	7

Questions 6: *Vector* Sounds Like an 70s Superhero (30 pts total)

Assume the following vector architecture with 64-element vector registers, and one functional unit of each type.



Further assume the following *start-up times* for each functional unit above:

Functional Unit	Start-up Time
FP add/subtract	**6**
FP multiply	**7**
FP divide	**20**
Vector load	**12**

Each functional unit may begin a new operation the cycle after the last element of the previous operation enters the pipeline. *I.e.*, when two vector store instructions execute one after the other, the start-up time penalty is paid only once.

Now consider the following VMIPS assembly sequence.

```
LV       V1,Ra     # V1[i] <- Mem(Ra+i)
MULV.D   V2,V1,V3  # V2[i] <- V1[i] * V3[i]
ADDV.D   V4,V1,V3  # V4[i] <- V1[i] + V3[i]
SV       Rb,V2     # Mem(Rb+i) <- V2[i]
SV       Rc,V4     # Mem(Rc+i) <- V4[i]
```

a. **(5 pts)** Assuming *no chaining* is available, identify the *convoys* in the above assembly and use *chimes* to estimate the number of cycles required to execute the above VMIPS assembly.

b. **(5 pts)** Now assuming *chaining* is available. Identify the *convoys* in the above assembly and use *chimes* to estimate the speedup that is possible.

c. **(10 pts)** Assuming *chaining* is available, now use the start-up latencies for each functional unit to determine the number of cycles required to execute the above assembly. How much error is introduced when using *chimes* to approximate latency?

Now consider the following C code.

```
for (int i=0; i<n; i++) {
     sum = sum + Y[i]*b;
     W[i+1] = W[i] * Y[i+1];
}
```

d. (4 pts) Identify any dependencies in the above code, and indicate their type.

e. (6 pts) Re-write the above loop to increase loop-level parallelism.