McGill University
ECSE 425: COMPUTER ORGANIZATION AND ARCHITECTURE
Winter 2016
Midterm Examination

2:35 – 3:55 PM, February 17th, 2015

Duration: 80 minutes

- **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 14 pages long. Please check that you have all 14 pages.**
- **There are five questions for a total of 80 points, or one point per minute. 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.**
- **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: **_____SOLUTION_____**

Student Number: **_____**

Name: ID:

Q1: _____/10 **Q2: _____/20** **Q3: _____/20**

Q4: _____/15 **Q5: _____/15**

Total:

Name: ID:

Question 1: Short Answers to Long Questions (10 pts total)

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

a. (4 pts) Assume a 40-bit virtual address space, with 8 KB pages. Main memory has a 4 GB capacity. L1 is two-way set associative, virtually indexed, physically tagged, with 64B blocks. Assuming the L1 cache is as large as possible, how many bits are used for block tag, index, and offset?

A cache that is as big as possible with allocate an entire page per way: this is a 16KB cache.

Offset: 6 bits => 64B
Index: 13 bits => 8KB, but 6 for offset => 7 bits
Tag: 32 – 13 = 19 bits

b. (2 pts) In a 16KB, two-way set associative cache with 64B blocks, what fraction of the total required bits of storage are used for bookkeeping (i.e., *overhead*)? Assume the cache is *write-allocate*, *write-back*, and that the L2 cache is *exclusive*.

This is the same as the cache in (a)
16KB => 16384B of storage in the data array

7-bit index => 128 sets => 2 * 128 * 19b = 608B of storage in the tag array
A valid bit, and a dirty bit for each way within each set => 2 * 128 * 2b = 64B more

(608 + 64) / (16384 + 608 + 64) = 0.039

c. **(2 pts)** Describe the various performance, power, and cost trade-offs associated with *increasing* the size of an on-chip cache.

Increasing the size of a cache will

- Increase its area, decrease its yield, and therefore increase its cost;
- Decrease its hit rate, increasing its performance;
- Increase its access delay, decreasing its performance;
- Increase its capacitance, increasing its power consumption.

d. **(2 pts)** Re-schedule the following instructions to eliminate all stalls. Assume a standard five-stage pipeline with full hazard detection and forwarding.

```
LW    R2, 0(R1)
LW    R3, 4(R1)
ADD   R4, R2, R3
SW    R4, 8(R1)
LW    R5, 12(R1)
ADD   R6, R4, R5
```

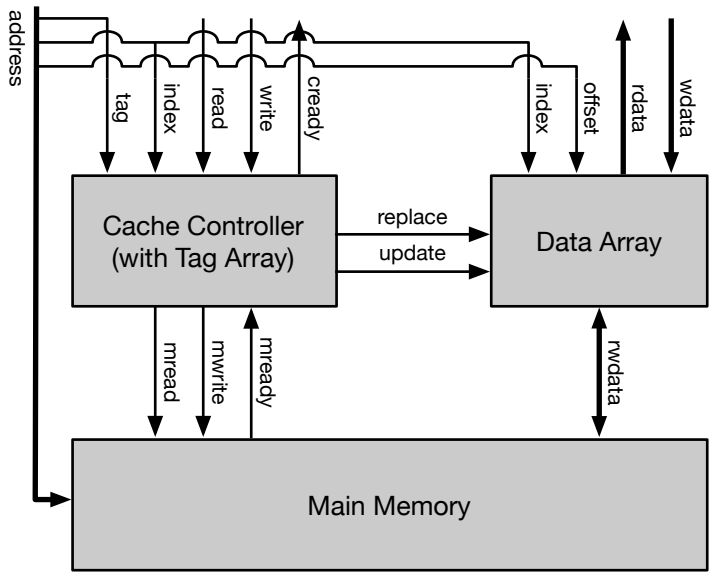
There are two stalls to eliminate: one before the first ADD, and one before the second ADD. Each disappears with a single change in the code: moving the final LW so it follows the earlier two:

```
LW    R2, 0(R1)
LW    R3, 4(R1)
LW    R5, 12(R1)
ADD   R4, R2, R3
SW    R4, 8(R1)
ADD   R6, R4, R5
```

Name: ID:

Question 2: This Cache is Out of Control (20 pts)

Consider the memory hierarchy below.



The cache controller implements a finite state machine to coordinate between the processor (not pictured), and memory, by manipulating the tag and data arrays. In this problem, you will design this finite state machine.

Assume the the cache is *write-allocate* and *write-back*, and that it blocks on writes to memory (*i.e.*, no write buffer is present). The cache controller finite state machine takes as input the following signals from the processor and main memory:

- `read`: asserted by the processor when it performs a read.
- `write`: asserted by the processor when it performs a write.
- `mready`: asserted by the memory when read data is available or write data has been saved.

The cache controller FSM also takes as input the following internal signals:

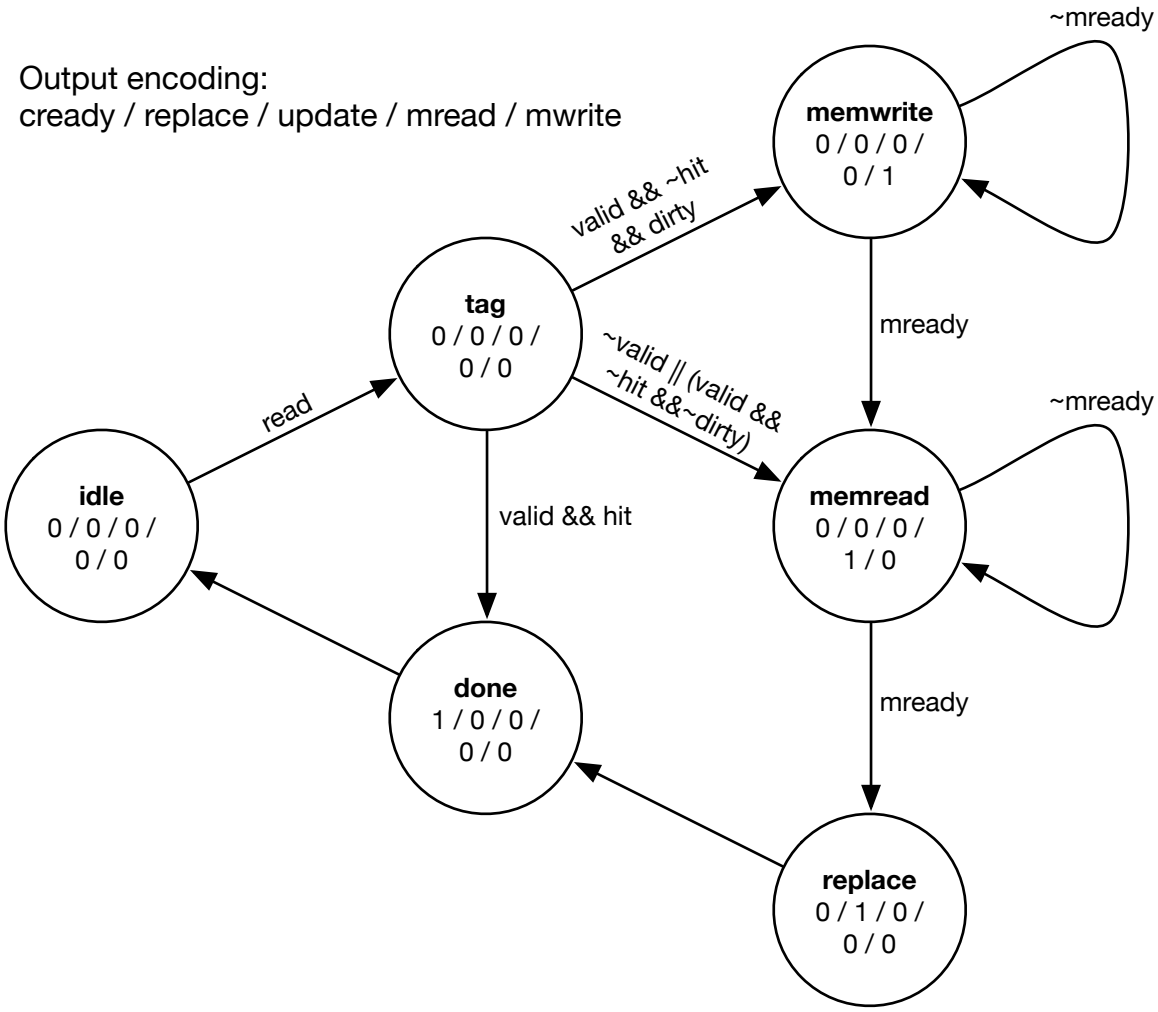
- `hit`: true when the addressed block matches the input tag.
- `valid`: true when the addressed block contains valid data.
- `dirty`: true when the addressed block contains dirty data.

The cache controller finite state machine asserts the following outputs as necessary:

- `cready`: asserted when read data is available or write data has been saved.
- `replace`: asserted when saving data from memory to the data array.
- `update`: asserted when saving data from the processor to the data array.
- `mread`: asserted when performing a read from main memory.
- `mwrite`: asserted when performing a write to main memory.

Design the *Moore* machine to process *read* requests from the processor using the states listed below. Clearly indicate which input conditions above (if any) are required for each transition. Also clearly indicate the condition of each output in each state. Use no more than one copy of any of the states. Note: some states may not be needed.

- *idle*: Initial state; leave when a read or write is requested by the processor.
- *tag*: For performing tag comparison; leave one cycle later.
- *replace*: For saving data read from memory to the data array; leave one cycle later.
- *update*: For saving data from the processor to the data array; leave one cycle later.
- *memread*: When reading from memory; leave when memory is ready.
- *memwrite*: When writing to memory; leave when memory is ready.
- *done*: When the cache access is complete (i.e., read data is available, or write has been completed); leave one cycle later.



Name: ID:

Additional Page for Question 2

Question 3: Improve Your Memory with Natural Supplements (20 pts)

Consider a pipelined processor that runs at 2 GHz and has a base CPI of 1.2 when all cache accesses hit. The only instructions that read data from or write data to memory are loads (16% of all instructions) and stores (12% of all instructions).

The memory system for this computer is composed of a split L1 cache. Both the I-cache and D-cache are direct-mapped and hold 64 KB each. The I-cache has a 1% miss rate; the D-cache has a miss rate of 10%.

The 1 MB inclusive, unified L2 cache has an access time of 15 ns. Of all memory references sent to the L2 cache in this system, 60% 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 50% of evicted data is dirty.

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

CT = 1/2e9 = 0.5 ns

$AMAT_{L1I} = HT_{L1I} + MR_{L1I} * (HT_{L2} + (1 + WBR) * MR_{L2} * MP)$
$= 1 + 0.01*(15/0.5 + (1 + 0.5) * 0.4*200/0.5)$
$= 3.7$ cycles

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

$AMAT_{L1D} = HT_{L1D} + (1 + WBR) * MR_{L1D} * (HT_{L2} + (1 + WBR) * MR_{L2} * MP)$
$= 1 + (1 + 0.5)*0.10*(15/0.5 + (1 + 0.5)*0.4*200/0.5)$
$= 41.5$ cycles

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

CPI = CPI_{base} + instruction access stalls per instruction + data access stalls per instruction
= 1.2 + (1) * (3.7 – 1) + 0.28 * (41.5 – 1) = 15.2 cycles

d. (8 pts) Now consider the effect of changing the size of the L1 D-cache. Assume that increasing the cache to 128 KB reduces the miss rate to 6% but increases the clock cycle time by 15%. What is the resulting speedup?

First, we need to determine the baseline using the PPE: IC * CPI * CT. IC is constant, so CPI * CT is sufficient.

CPI * CT = 15.2*0.5 ns = 7.6 ns/inst

Now, we need to determine CPI * CT for the new system. First, we need to determine the new cycle time.

CT_{new} = CT * 1.15 = 0.575 ns

$AMAT_{L1I} = HT_{L1I} + MR_{L1I} * (HT_{L2} + (1 + WBR) * MR_{L2} * MP)$
= 1 + 0.01*(15/0.575 + (1 + 0.5) * 0.4*200/0.575)
= 3.35 cycles

$AMAT_{L1D} = HT_{L1D} + (1 + WBR) * MR_{L1D} * (HT_{L2} + (1 + WBR) * MR_{L2} * MP)$
= 1 + (1 + 0.5)*0.06*(15/0.575 + (1 + 0.5)*0.4*200/0.575)
= 22.1 cycles

CPI = CPI_{base} + instruction access stalls per instruction + data access stalls per instruction
= 1.2 + (1) * (3.35 – 1) + 0.28 * (22.1 – 1) = 9.46 cycles

CPI * CT = 9.46*0.575 = 5.44 ns/inst

Finally, speedup: 7.6 / 5.44 = 1.4x!

Name: ID:

Question 4: Don't Fall Through that Branch! (15 pts)

Consider the following code:

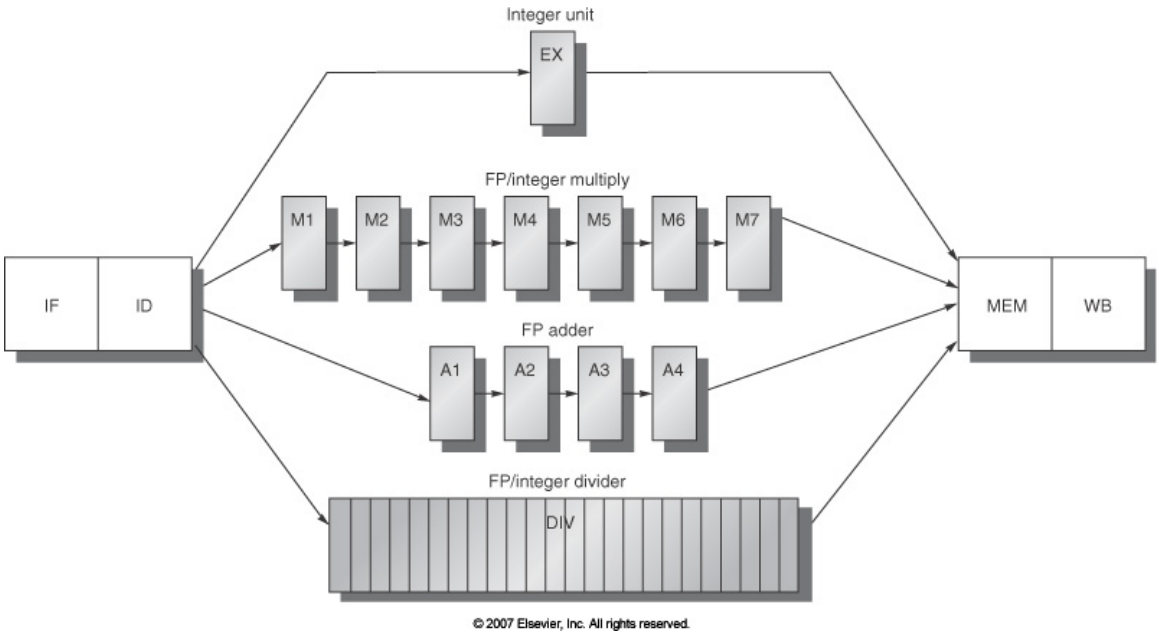
```
      LW    R6, 16(R1)
      ADDI  R7, R0, 2
EQ2:  BNEQ  R6, R7, EQ1
      ADD   R8, R2, R3
      MULT  R6, R5
      MFLO  R10
      J     END
EQ1:  ADDI  R7, R0, 1
      BNEQ  R6, R7, DEF
      SUB   R8, R2, R3
      LW    R9, 20(R1)
      MULT  R8, R9
      MFLO  R10
      J     END
DEF:  ADDI  R6, R0, 0
      ADDI  R8, R0, 0
      ADDI  R10, R0, 0
END:  SW    R6, 24(R1)
      SW    R8, 28(R1)
      SW    R10, 32(R1)
```

and corresponding initial memory state:

Address	Value
0x1000	5
0x1004	1
0x1008	3
0x100C	1
0x1010	2
0x1014	4
0x1018	8
0x101C	12
0x1020	3

Name: ID:

This code will run on the standard MIPS floating point pipeline, illustrated below.



Assume:

- Full hazard detection and forwarding logic;
- The register file supports one write and two reads per clock cycle;
- Branch targets and conditions and jump targets are resolved in ID;
- Branches are handled by predicting not taken;
- J is treated as a branch instruction for branch prediction purposes;
- MFLO is treated as an ALU operation for hazard detection and forwarding purposes;
- Split L1 instruction and data caches service all requests in one clock cycle;
- Two or more instructions may simultaneously pass through MEM (WB) as long as only one makes use of the memory (register file);
- Structural hazards are resolved by giving priority to older instructions.

Complete the chart on the next page to show the execution of the loop if R1 = 0x1000 initially. Indicate pipeline stages with {**F**, **D**, **X**, **Mi**, **Ai**, **M**, **W**}, and stalls with **S**.

Name: ID:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
LW R6, 16(R1)	F	D	E	M	W																																			
ADDI R7, R0, 2		F	D	E	M	W																																		
BNEQ R6, R7, EQ1			F	D	S	E	M	W																																
ADD R8, R2, R3				F	S	D	E	M	W																															
MULT R6, R5						F	D	M 1	M 2	M 3	M 4	M 5	M 6	M 7	M	W																								
MFLO R10							F	D	S	S	S	S	S	S	E	M	W																							
J END								F	S	S	S	S	S	S	D	E	M	W																						
ADDI R7, R0, 1															F	D	E	M	W																				
SW R6, R0, 0																F	D	E	M	W																			
SW R8, 28(R1)																	F	D	E	M	W																		
SW R10, 32(R1)																		F	D	E	M	W																	

Name: ID:

Question 5: I Should Have Known! (15 pts)

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

IF1 IF2 ID1 ID2 EX MEM1 MEM2 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 EX.

If 15% 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.

CPI = 1 + branch frequency * branch penalty
= 1 + branch frequency * (taken freq * taken penalty + untaken freq * untaken penalty)

Assume x is the fraction of branches taken.

Computer A – predicted not taken

Taken penalty: 4 – IF1, IF2, ID1, ID2 (for the mispredicted branch successor)
Untaken penalty: 0

CPI = 1 + 0.15 * (x * 4) = 1 + 0.6 * x

Computer B – predicted taken

Taken penalty: 3 – IF1, IF2, ID1
Untaken penalty: 4 – IF1, IF2, ID1, ID2

CPI = 1 + 0.15 * (x * 3 + (1 – x) * 4) = 1 + (0.45 * x + 0.6 * (1 – x)) = 1.6 – 0.15 * x

Solving for x

1 + 0.6 * x = 1.6 – 0.15 * x
0.75 * x = 0.6
x = 0.8

If 80% of branches are taken, the performance of the two computers will be equivalent.

Name: ID:

Additional Page