McGill University ECSE 425
Computer Organization and Architecture
all 2009
Second Midterm Examination

18:35 – 19:55, November 16, 2009

Duration: 80 minutes

- **There are 4 questions for a total of 100 points. There are 12 pages. Please check that you have all 12 pages.**
- **This is a closed-book exam. You can bring 1 single-sided sheet of hand-written notes. This sheet of notes must be entirely hand-written, no portions may be machine-produced of photocopied.**
- **Calculators are permitted, but no cell phones or laptops are allowed.**
- **Write your name and student number in the space below. Do the same on the top of each sheet of this exam.**
- **State any assumption.**
- **Write your answers in the space provided.**

Name: _____

Student Number: _____

Q1: _____ **Q2:** _____

Q3: _____ **Q4:** _____

Total: _____

Question 1. Short Answers (20 points)
There are 10 sub question (2 points each)

For each question below, provide a short answer in 1-2 sentences.

a) What is the difference between dynamic scheduling and speculation?
Speculation is essentially dynamic scheduling with branch prediction.
Dynamic scheduling: in-order issue but out-of-order execution and out-of-order completion
Speculation: need to enforce in-order completion via in-order commit

b) Give an example of memory disambiguation. Is it easier to solve in hardware or software?

```
MULTD   R0, R0, R2(20)
SD      R4, R6(10)
```

The two memory references R2(20) and R6(10) may be to the same address.
In general, memory disambiguation is when two memory accesses point to the same address but have different bases and offsets. The problem is easier to solve in hardware where the calculated address is known.

c) What is the function of the common data bus in the Tomasulo organization?
Carry the results of the functional units to the reservation stations, load/store buffers and register file.

d) What are the advantages of in-order commit for speculation and exception handling?
Allows exact break points for precise exception handling and for backing up from a wrongly predicted branch.

e) What is the main difference between superscalar and VLIW processors?
Superscalar: dynamic issue
VLIW: static issue (long instruction words are already ordered and predetermined in code)

f) Why does increasing the set associativity in an L1 cache also increase the processor clock cycle?
More tags to search, the search circuitry is more complex, thus the latency increases.

g) What is the advantage of increasing the set associativity of a cache?
It decreases the miss rate

h) Why is virtual memory still useful even though we can afford large memory nowadays?
Hides segmented memory with a continuous memory space.
Protects memory space of one program from another.

i) Virtually addressed cache (for both index and tag) can avoid address translation for cache lookup, but can create other problems. Name one such problem.
Memory space protection

j) What is the function of the table lookaside buffer (TLB)?
To cache a portion of the address translation table for faster address translation, bypassing the translation table access in the main memory.

Question 2. Branch Prediction (25 points)
There are 2 parts to this question.

Part a) (15 points)

Use the following snippet of code for this question

```
a = 0;
b = 0;
for (i = 0 ; i < 6 ; i++) // B1
   if ((x[i] % 2) == 0) // B2
      a = a + 1;
   if ((x[i] % 4) == 0) // B3
      b = b + 1;
```

Given x = [8 3 8 2 9 3], fill in the table on the next page for a (1,1) correlating branch predictor. The branch predictors are all initialized to the not taken state, and the last branch prior to this snippet of code was not taken.

```
a = 0;
b = 0;
for (i = 0 ; i < 6 ; i++) // B1
   if ((x[i] % 2) == 0) // B2
      a = a + 1;
   if ((x[i] % 4) == 0) // B3
      b = b + 1;
x = [8 3 8 2 9 3]
```

#	Branch	Outcome	State when prev. branch T	State when prev. branch T	Prediction	Update T	Update T
1	B1	T	N	N	N	T	N
2	B2	T	N	N	N	N	T
3	B3	T	N	N	N	N	T
4	B1	T	T	N	N	T	T
5	B2	N	N	T	T	N	N
6	B3	N	N	T	N	N	T
7	B1	T	T	T	T	T	T
8	B2	T	N	N	N	N	T
9	B3	T	N	T	T	N	T
10	B1	T	T	T	T	T	T
11	B2	T	N	T	T	N	T
12	B3	N	N	T	T	N	N
13	B1	T	T	T	T	T	T
14	B2	N	N	T	T	N	N
15	B3	N	N	N	N	N	N
16	B1	T	T	T	T	T	T
17	B2	N	N	N	N	N	N
18	B3	N	N	N	N	N	N
19	B1	N	T	T	T	N	T

The Grayed out entries are the states used for the prediction and the states being updated.

Part b) (10 points)

Consider the tournament branch predictor given in Figure XX. The local predictor uses 10 bits of the branch address to index the branch history table. To make the local predictions, the most recent 12 branch outcomes are used to find the corresponding 3-bit predictor information. The global predictor uses the history of the last 8 branches; each entry in the global predictor is a standard 2-bit predictor. To choose from among a global predictor and a local predictor, the tournament predictor uses 2-bit counters. Compute the size of this tournament branch predictor.

Solution

The local history table is indexed with 10 bits of the branch address. Thus, the table has 2^10, 1024, entries and each entry requires 12 bits to hold the most recent 12 branch outcomes.

The local predictor is indexed using the most recent 12 branch outcomes. Thus, the table has 2^12, 4096, entries and each entry contains a 3-bit predictor.

The global predictor is indexed using the last 8 branches. Thus, the table has 2^8, 256, entries and each entry contains a 2-bit predictor.

The chooser is indexed using the last 8 branches. Thus, the table has 2^8, 256, entries and each entry contains a 2-bit chooser.

Adding the size of each table, we get the total number of bits require for the tournament branch predictor: 1024*12 + 4096*3 + 256*2 + 256*2 = 25600 bits.

Question 3. Tomasulo/Speculation (30 points)
There are 2 parts to this question.

Use the following snippet of code, representing a dot-product loop, for both parts.

```
loop:  L.D     F0, 0(R1)
       L.D     F4, 0(R2)
       MULT.D  F0, F0, F4
       ADD.D   F2, F0, F2
       SUBI    R1, R1, #8
       SUBI    R2, R2, #8
       BNEZ    R1, loop
```

Part a) (15 points)

Consider a dynamically scheduled machine with hardware speculation. Assume that the functional units are not pipelined and that all memory accesses hit the cache. There is a memory unit with 5 load buffers. The reorder buffer has 50 entries. The reorder buffer can function as a store buffer, so there are no separate store buffers. Each load or store takes 2 cycles to execute, 1 to calculate the address, and 1 to load/store the data. Assume a branch predictor with 0% misprediction rate. Assume there are dedicated integer functional units for effective address calculation and branch condition evaluation. The other function units are described in the following table.

Func. unit type	Cycles to execute	Number of func. units	Number of reservation stations
Integer ALU	1	1	5
FP adder	4	1	3
FP multiplier	15	1	2

Fill in the table on the next page with the clock cycle number that each instruction issues, begins and ends execution, writes its result, and commits for the first three iterations of the loop. Assume no multiple write backs nor commits.

Func. unit type	Cycles to execute	Number of func. units	Number of reservation stations
Integer ALU	1	1	5
FP adder	4	1	3
FP multiplier	15	1	2
Load	2	1	5

```
loop:  L.D     F0, 0(R1)
       L.D     F4, 0(R2)
       MULT.D  F0, F0, F4
       ADD.D   F2, F0, F2
       SUBI    R1, R1, #8
       SUBI    R2, R2, #8
       BNEZ    R1, loop
```

Loop iteration	Code		Issue	Exec. Start	Exec. End	Write Back	Commit
1	L.D	F0, 0(R1)	1	2	3	4	5
1	L.D	F4, 0(R2)	2	4 (3)	5 (4)	6 (5)	7(6)
1	MULT.D	F0, F0, F4	3	7(6)	21 (20)	22 (21)	23 (22)
1	ADD.D	F2, F0, F2	4	23 (22)	26 (25)	27 (26)	28 (27)
1	SUBI	R1, R1, #8	5	6	6	7	29 (28)
1	SUBI	R2, R2, #8	6	7	7	8	30 (29)
1	BNEZ	R1, loop	7	8	8	9	31 (30)
2	L.D	F0, 0(R1)	8	9	10	11	32 (31)
2	L.D	F4, 0(R2)	9	11 (10)	12 (11)	13 (12)	33 (32)
2	MULT.D	F0, F0, F4	10	22 (1) (21)	36 (35)	37 (36)	38 (37)
2	ADD.D	F2, F0, F2	11	38 (37)	41 (40)	42 (41)	43 (42)
2	SUBI	R1, R1, #8	12	13	13	14	44 (43)
2	SUBI	R2, R2, #8	13	14	14	15	45 (44)
2	BNEZ	R1, loop	14	15	15	16	46 (45)

(1) Must wait for the MULT in the previous iteration to exit the functional unit before this one enters.

( ) Values in bracket if load can calculate address while another load accesses memory.

Part b) (15 points)

Show the contents of the Reservation Stations, the registers, and the Re-Order Buffer when the 2nd iteration's `L.D F4,0(R2)` begins execution.

Cycle 10

Reservation stations								
Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	Address
Load 1	No	L.D					#1	Reg[R1]
Load 2	No	L.D					#2	Reg[R2]
Load 3	Yes	L.D					#8	#5-8
Load 4	Yes	L.D					#9	#6-8
Load 5								
FP Add 1	Yes	ADD.D		Reg[F2]	FP Mult 1		#4	
FP Add 2	Yess	ADD.D			FP Mult 2	FP Add 1	#11	
FP Add 3								
FP Mult 1	Yes	MULT.D	Mem[R1]	Mem[R2]			#3	
FP Mult 2	Yes	MULT.D			Load 3	Load 4	#10	

Reorder buffer					
Entry	Busy	Instruction	State	Destination	Value
1	No	`L.D`	Commit	F0	Mem[Reg[R1]]
2	No	`L.D`	Commit	F4	Mem[Reg[R2]]
3	Yes	`MULT.D`	Exec	F0	
4	Yes	`ADD.D`	Exec	F2	
5	Yes	`SUBI`	Write result	R1	Reg[R1]-8
6	Yes	`SUBI`	Write result	R2	Reg[R2]-8
7	Yes	`BNEZ`	Write result		
8	Yes	`L.D`	Exec	F0	Mem[#5-8]
9	Yes	`L.D`	Exec	F4	
10	Yes	`MULT.D`	Issue	F0	
11	Yes	ADD.D	Issue	F2	
12					
13					
14					
15					

	R1	R2	F0	F2	F4
Reorder #	#5	#6	#10	#11	#9
Busy	Yes	Yes	Yes	Yes	Yes

Question 4. Memory Hierarchy (25 points)
There are 3 parts to this question.

Part a) (15 points)

For a given benchmark running on a given processor, 28% of the instructions are loads, and 12% are stores. Using that benchmark, we want to choose between a direct-mapped cache and a 2-way set associative cache. Both caches use write-back protocol with write-allocate.

The processor's cache requires 2 clock cycles for every write hit, and 1 clock cycle for every read hit. The penalty for transferring a block from and to the main memory is 100 ns. Whenever the cache needs to replace a block, there is a 30% chance that the block it is replacing is dirty.

The processor clock speed with direct-mapped cache is 2GHz. Because of the extra time for tag search, the clock cycle with 2-way set associative cache is 1.2 times greater than that with the direct mapped cache.

Using the miss rates from the following table, calculate the effective CPU time for both the direct-mapped and 2-way set associative caches architectures. Which cache is better and why? Use a base CPI of 1CC.

	Cache read miss rate	Cache write miss rate
Direct mapped	16%	8%
2-way set associative	12%	7%

Solution

$$\begin{aligned} Time/Instr &= 1CC + \%_{read} \times \%_{readmiss} \times (100ns + 30\% \times 100ns) + \%_{write} \\ &\quad \times \left(1CC + \%_{writemiss} \times (100ns + 30\% \times 100ns)\right) \\ &= (1 + \%_{write})CC + (\%_{read} \times \%_{readmiss} + \%_{write} \times \%_{writemiss}) \times 130ns \\ &= 1.12CC + (1.28 \times \%_{readmiss} + 0.12 \times \%_{writemiss}) \times 130ns \end{aligned}$$

$$\%_{read} = 100\% + \%_{load} = 1.28$$

$$CPU_{time-direct} = IC(1.12CC + 27.872ns)\big|_{CC=\frac{1}{2E9}} = IC \times 28.432ns$$

$$CPU_{time-2way} = IC(1.12CC + 21.06ns)\big|_{CC=\frac{1.2}{2E9}} = IC \times 21.732ns$$

The 2-way set associative cache is better because the CPU time is better.

Part b) (10 points)

The following tables show the contents of an L1 and L2 caches. Both caches have a block size of 4 bytes. Also, both of them are 2-way set associative and are byte addressable. Note that all the values in the tables are in hexadecimal notation.

L1 Cache	Set 1		Set 2	
Index	Tag	Block content	Tag	Block content
0x0	0x0BD2	E7 E1 FA FB	0x61AE	F7 02 EF 5F
0x1	0x0BD2	20 82 35 40	0xCB93	28 59 54 B8
0x2	0xD2CF	E9 D3 00 08	0xB1E2	F8 78 C4 C0
0x3	0xB1E1	A1 E2 4B AA	0xB1E2	F5 08 E4 A4
0x4	0xB1E2	18 F8 68 72	0x7212	7C 41 94 1A
0x5	0xF342	47 4B A8 C7	0x7212	CC DF 2E 4A
0x6	0xF342	8C 00 6D 60	0xA575	24 52 A9 C1
0x7	0x7052	F5 1F 2B 00	0xB599	6B F8 8C 24

L2 Cache	Set 1		Set 2	
Index	Tag	Block content	Tag	Block content
0x0	0x30D7	F7 02 EF 5F	0x05E9	E7 E1 FA FB
0x1	0x05E9	20 82 35 40	0x05E9	C0 F4 63 D2
0x2	0x112A	09 24 66 8C	0x58F1	F8 78 C4 C0
0x3	0x112A	46 E4 6C BC	0x58F1	F5 08 E4 A4
0x4	0x58F1	18 F8 68 72	0x3909	7C 41 94 1A
0x5	0x79A1	47 4B A8 C7	0x3909	CC DF 2E 4A
0x6	0x79A1	8C 00 6D 60	0x79A1	61 64 E6 27
0x7	0x112A	AD C1 96 4D	0x3829	F5 1F 2B 00
0x8	0x112A	8C 00 6D 60	0X5B01	7C 41 94 1A
0x9	0x0F0A	A7 DE 9F 5A	0x65C9	28 59 54 B8
0xA	0x6967	E9 D3 00 08	0x6967	F5 1F 2B 00
0xB	0x6967	A7 CD 66 26	0x58F0	A1 E2 4B AA
0xC	0x6967	8C BA AF B8	0x6164	C1 1A 2E 76
0xD	0x6164	EA CD 34 37	0x6164	08 26 32 8B
0xE	0x0F0A	F5 94 1F 73	0x52BA	24 52 A9 C1
0xF	0x5ACC	6B F8 8C 24	0x0F0A	2B D2 E4 0A

The following table shows the size in bits of each address fields

L1	Block address		Block offset
	Tag	Index	
	16	3	2
L2	Block address		Block offset
	Tag	Index	
	15	4	2

What is the value returned from the multilevel cache system when we request a byte from the following addresses

i. 0x163C51 located in L1 0xB1E2-index4,byte1 (0x68)
ii. 0x185933 located in L2 0x6164-index6,byte3 (0xC1)
iii. 0x017A41 located in L1 0x0BD2-index4,byte1 (0xFA)

Solution

i. 0x68
0x163C51 can be decoded into the L1 address fields 0xB1E2-4-1 (tag-index-byte format). This is a hit in the L1 cache.

ii. 0xC1
0x185933 can be replaced into the L1 address fields 0xC2C9-4-3. This is a miss in the L1 cache. It can be replaced into the L2 address fields 0x6164-C-3. This is a hit in the L2 cache.

iii. 0xFA
0x017A41 can be replaced into the L1 address fields 0x0BD2-0-1. This is a hit in the L1 cache.