Name: **ID:**

McGill University

ECSE 425 -- Computer Organization and Architecture

Fall 2009

FINAL EXAMINATION SOLUTIONS

9:00 am – 12:00 pm, December 11, 2009

Duration: 180 minutes

Question 1. Short Answers (35 points)
There are 2 parts to this question.

1) Part 1: There are 10 sub questions in this part (3 points each)

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

a) What is the difference between multithreading and simultaneous multithreading (SMT)?
Multithreading exploits TLP, it can be run on any machine (single or multiple issue, uni- or multi-processor) . One thread at each clock cycle.
SMT exploits both TLP and ILP, it must be run on a multiple issue machine with dynamic scheduling. Multiple threads at each CC.

b) What is the shared memory multiprocessor model? Can it be applied to distributed memory multiprocessor systems?
The shared memory multiprocessor model uses a shared address space among all processors.
It can be applied to physically distributed memory multiprocessors systems.

c) Name two major challenges in parallel processing using multiprocessors.
One must parallelize programs; the serial portion of the program becomes the bottleneck.
The latency to remote memory is longer.

d) Why is cache coherency not an issue in uni-processor system but is an issue in shared-memory multi-processor systems?
In a uni-processor, only 1 processor has access to the data and no other processors can modify it. In shared-memory multiprocessor systems, other processors can access and modify the data.

e) The bus-based broadcast snooping protocol serializes all the coherence traffic. Name an advantage and a disadvantage of this serialization.
Serializing preserves memory access order, such as RAW WAW WAR among all processors.
However the latency can be longer, the bus can become the bottle neck with increasing number of processors.

f) A challenge of multiprocessor systems is to build operations that appear atomic. Give an example of a sequence of instructions that can be used for this purpose.

```
try: ll R1,0(R2)
     some operation on R1
     sc R1,0(R2)
     beqz R1,try
```

g) What is the difference between write allocate and no-write allocate?

In write allocate, the block must be transferred to the cache on a write miss, followed by a write hit action (write back or write through). In no-write allocate, there is no block transfer to cache on a write miss and the data is written directly to memory.

h) Why caches with virtual index physical tag can help reduce the hit time compared to physically addressed caches?

With virtual index physical tag, cache read using the virtual index and address translation can be done in parallel, versus an all physically addressed cache, the hardware must translate the address first, then read the cache using the translated (physical) index.

i) Give an example of a technique to reduce cache miss penalty.

1. Use multilevel caches.
2. Fetch the critical word first then the rest of the block, or early restart (fetch in order but continue the CPU instruction as soon as the needed word arrives, while fetching the rest of the block).
3. Prioritize read misses over writes by using a write buffer.

j) What is the difference between AMAT and CPUtime? Which one is a more accurate performance measure for a computer system?

AMAT: average time it takes to access memory

CPU time: average time it takes to run a sequence of instructions. The CPU time includes the AMAT in its formula.

CPU time is a more accurate performance measure as it gives more realistic performance since not all instructions access memory.

Part 2 (5 points)

The current focus in computer design has shifted from getting more instructions per clock cycle on a single CPU (by having multiple pipelines with multiple issues) to having multiple CPUs (each CPU is either single issue or multiple issue). Does it mean that exploiting instruction level parallelism is no longer useful? What factors do you think will influence the success of multiple CPU architecture? Provide your answer in a short paragraph (no more than half a page).

Answer:

Exploiting ILP is still useful. What the shift means is that current techniques for exploiting ILP on a single processor are good enough and it is not worthwhile to invent new techniques to exploit ILP because of diminishing return (at least with the current technology).

Name: **ID:**

The current focus is on new techniques to exploit TLP, where multiprocessors seem most suitable. (Each processor however can implement techniques for exploiting ILP within each thread.)

In current and future systems, both ILP and TLP are going to exist. The balance however is unclear and depends on application. Application is a major factor that influences the success of an architecture.

Other important factor is the software. Multiple processors provide the platform for exploiting TLP, but the software must also be parallelized to take advantage of this platform before we see major multiprocessor successes.

Technology that affects the speed of inter-processor communications is also a factor.

Power consumption is likely to increase for multiple processors so efficient power management is another factor.

Question 2. Multi Processors (30 points)

There are 3 parts to this question.

Part a) (10 points)

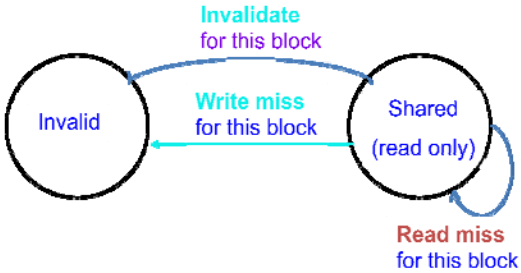
Consider the write-back invalidate snooping protocol with 3 states: Invalid, Shared and Exclusive. List all **bus requests** to a shared block in a cache and show the corresponding state transitions in the finite state machine for this protocol.

Answer

Shared ------read miss------>Shared

Shared ------write miss------> Invalid

Shared ------invalidate------> Invalid



Part b) (10 points)

Assume that words x1 and x2 are in the same cache block, which is in the **Shared** state in the caches of both processors P1 and P2. Assuming the following sequence of events, identify if each event is a hit or a miss, and each miss as a true sharing miss or a false sharing miss. Any miss that would occur if the block size were one word is designated a true sharing miss.

Time	P1	P2
1	Read x1	
2		Write x2
3	Write x1	
4		Read x2
5	Write x2	

Answers:

Time	P1	P2	Hit/Miss? True or false sharing miss? Why?
1	Read x1		Hit since x1 in shared state
2		Write x2	True sharing miss since P1 also has x2 in shared state
3	Write x1		False sharing miss since P1 also has x1 in shared state
4		Read x2	False sharing miss since P1 has the block containing x2 in exclusive state even though it did not modify x2
5	Write x2		True sharing miss since P2 also has x1 in shared state

Part c) (30 points)

Consider a 4 processor distributed shared-memory system. Each processor has a single direct-mapped cache that holds four blocks, each containing two words with addresses separated by 4. To simplify the illustration, the cache address tag contains the full address and each word shows only two hexadecimal characters, with the least significant word on the right. The cache states are denoted M, S, and I for Modified, Shared, and Invalid. The directory states are denoted DM, DS, and DI for Directory Modified, Directory Shared, and Directory Invalid. This simple directory protocol uses messages given in Table 2c on the next page.

Assume the cache contents of the four processors and the content of the main memory as shown in Figure 2c below. A CPU operation is of the form

P#: <*op*> <*address*> [<-- <*value*>]

where P# designates the CPU (e.g., P0), <op> is the CPU operation (e.g., read or write), <address> denotes the memory address, and <value> indicates the new word to be assigned on a write operation.

For the sequence of CPU operations given below, show the changes in the contents (including coherence state, tags, and data) of the caches and memory after the each operation has completed. What value is returned by

each read operation? For each operation, what is the sequence of messages passed on the bus? You can use the table on the following page to help you with the bus messages.

Note: The tags are in **hexadecimal**

P0				P1				P2				P3			
state	tag	data		state	tag	data		state	tag	data		state	tag	data	
I	100	26	10	I	100	26	10	S	120	02	20	S	120	02	20
S	108	15	08	M	128	2D	68	S	108	15	08	I	128	43	30
M	110	F7	30	I	110	6F	10	I	110	6F	10	M	130	64	00
I	118	C2	10	S	118	3E	18	I	118	C2	10	I	118	40	28

	Memory			
address	state	Sharers	Data	
100	DI		20	00
108	DS	P0,P2	15	08
110	DM	P0	6F	10
118	DS	P1	3E	18
120	DS	P2,P3	02	20
128	DM	P1	3D	28
130	DM	P3	01	30

P0: read 130
P3: write 130 <-- 20
P2: read 11C

Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Processor P has a read miss at address A; request data and make P a read sharer
Write miss	Local cache	Home directory	P, A	Processor P has a write miss at address A; request data and make P the exclusive owner
Invalidate	Local cache	Home directory	A	Request to send invalidates to all remote caches that are caching the block at address A
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at address A
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared
Fetch/invalidate	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache
Data value reply	Home directory	Local cache	D	Return a data value from the home memory
Data write back	Remote cache	Home directory	A, D	Write back a data value for address A

P = requesting processor number, A = requested address, and D = data contents

Table 2c. Messages for a simple directory protocol.

To show the bus messages, use the following format:

Bus {message type, requesting processor, address, data}

Example: Bus {read miss, P0, 100, --}

To show the contents in the cache of a processor, use the following format:

P# <block #> {state, tag, data}

Example: P3 <block 0> {S, 120, 02 20}

To show the contents in the memory, use the following format:

M <address> {state, [sharers], data}

Example: M <120> {DS, [P0, P3], 02 20}

Answers:

P0: read 130

Bus{data write back,110,F7 30} sent by P0 to directory
M.110{DI,,F7 30}
Bus{read miss,P0,130} sent by P0 to directory
Bus{fetch,130} sent by directory to P3
P3.B2{S,130,64 00}
Bus{data write back,130,64 00} sent by P3 to directory
Bus{data value reply,64 00} sent by directory to P0
P0.B2{S,130,64 00}; returns 00
M.130{DS,{P0,P3},64 00}

P3: write 130 <-- 20

P3.B2{M,130,64 20}
Bus{invalidate, 130} sent by P3 to directory
M.130{DM,P3,64 00}
Bus{invalidate, 130} sent by directory to P0
P0.B2{I,130,64 00}

P2: read 11C

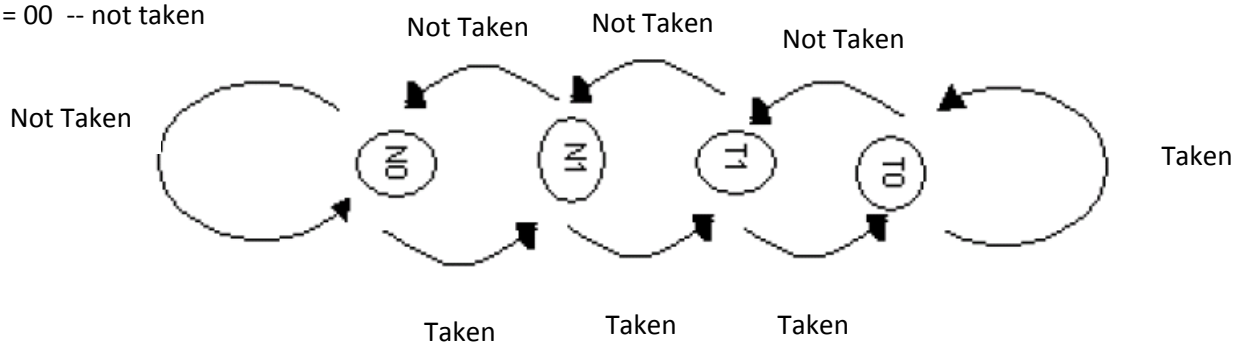
Bus{read miss,P2,11C} sent by P2 to directory
M.118{DS,{P1,P2},3C 18}
Bus{Data value reply,3C 18} sent by directory to P2
P2.B3{S,118,3C 18}; returns 3C

Name: **ID:**

Question 3. Memory Hierarchy (30 points)
There are 3 parts to this question.

Part a) (5 points)
Draw the finite state machine for a 2-bit local predictor using the saturating counter in the space below.

T0 = 11 -- taken
T1 = 10 -- taken
N1 = 01 -- not taken
N0 = 00 -- not taken



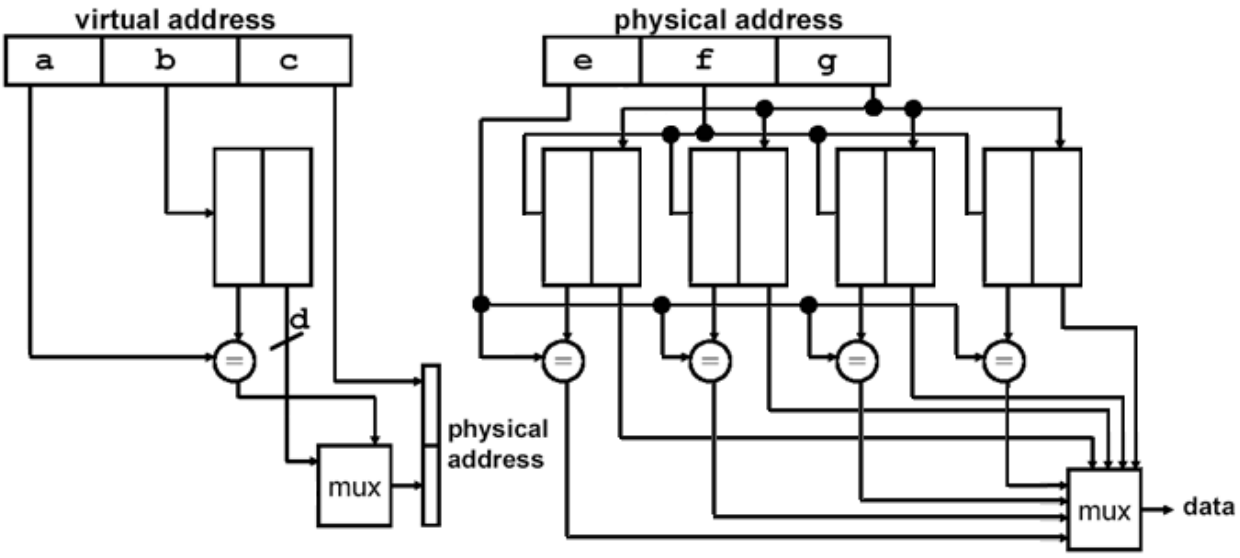
Part b) (15 points)
Consider a Virtual Memory / Cache system with the following properties:

Virtual address size	64 bits, byte-addressable
Physical address size	30 bits, byte-addressable
Block size	32 bytes
Page size	64 kbytes
Total cache data size	32 kbytes
Cache associativity	4-way set associative
TLB associativity	1-way set associate
TLB size	1024 entries in total

Name the address fields and calculate the bit-size of each field in the following figure.

	a	b	c	d	e	f	g
Name	TLB tag	TLB index	Page offset	Physical page table	Cache tag	Cache index	Block offset
Bit-size	38	10	16	14	17	8	5

$$\text{blocksize} = 32 = 2^g$$
$$\#\,\text{blocks/set} = (32 * 1024/32)/4 = 2^f$$
$$\text{physicaladdresssize} = e + f + g$$
$$\text{pagesize} = 64 * 1024 = 2^c$$
$$\#\text{pages} = 1024 = 2^b$$
$$\text{virtualaddresssize} = a + b + c$$
$$\text{physicaladdresssize} = \text{d} + \text{c}$$



Part c) (10 points)

Consider a MIPS machine with a byte-addressable main memory and the following specifications:

Data cache size	1 kB
Block size	64 B

The following C program representing a dot-product (with no optimizations) is executed on this computer.

```
int i;
int a[256], b[256];
int c;

for ( i = 0; i < 256; i++ )
{
   c = a[i] * b[i] + c;
}
```

Assume that the size of each array element is one word of size 4 bytes and the elements are stored in consecutive memory locations in array index order. Array `a` starts at address `0x0000`, `b` at `0x0400`. What is the miss rate given a 2-way set associative cache? Show your calculations.

Answer:

Given that a[i] and b[i] will never replace each other, and given that we read every elements once and in order, the number of misses will correspond to the number of blocks required to hold arrays a and b. Array a requires 16 blocks to hold its 256 words, therefore 16 blocks will be transferred from memory to the cache throughout the loop. The same applies to array b.

Throughout the loop, 32 blocks will be transferred from memory to the cache, and there will be 512 memory accesses. That makes a total of 32 misses out of 512 memory accesses and gives a miss rate of 6.25%.

Name: **ID:**

Question 4. Pipelining and Instruction Level Parallelism (55 points) There are 4 parts to this question.

For all 4 parts, use the following snippet of code

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

Also, use the following execution time for each unit:

Functional unit	Cycles to execute
FP add	3
FP mult	6
Load/store	2
Int ALU	1

Part a) (10 points)

Identify all hazards in the snippet of code.

Potential data hazards:

RAW:
L.D F0, 0(R1) -> ADD.D F0,F0,F4
ADD.D F0,F0,F4 -> MUL.D F2,F0,F2
L.D F2,0(R2) -> MUL.D F2,F0,F2
MUL.D F2,F0,F2 -> S.D F2,0(R2)
DADDUI R1,R1,#-8 -> BNEZ R1,loop

WAW:
L.D F2,0(R2) -> MUL.D F2,F0,F2
L.D F0,0(R1) -> ADD.D F0,F0,F4

WAR:
L.D F0,0(R1) -> DADDUI R1,R1,#-8
L.D F2,0(R2) -> DADDUI R2,R2,#-8
S.D F2,0(R2) -> DADDUI R2,R2,#-8

Name: **ID:**

Part b) (15 points)

Part b.i) Unroll the loop twice (2 iterations per new loop) and schedule it on a 5-issue VLIW machine using the provided table.

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

Clock cycle	Memory reference 1	Memory reference 2	FP operation 1	FP operation 2	Integer operation/branch
1	L.D F0,0(R1)	L.D F6,-8(R1)			
2	L.D F2,0(R2)	L.D F8,-8(R2)			
3			ADD.D F0,F0,F4	ADD.D F6,F6,F4	
4					
5					
6			MUL.D F2,F0,F2	MUL.D F8,F6,F8	
7					
8					
9					DADDUI R1,R1,#-16
10					DADDUI R2,R2,#-16
11					BNEZ R1, loop (with delay slot)
12	S.D F2,16(R2)	S.D F8,8(R2)			BNEZ R1, loop (without delay slot)

Part b.ii)In this VLIW machine, at least how many times do you need to unroll the loop to get the maximum efficiency? Answer: Without getting into too many complexities, we can unroll 6 times easily to get 6 iterations per 14 CC. We can also unroll 10 times to get 10 iterations per 19 CC by performing another 4 iterations while idling. The true maximum efficiency is when one of the unit is always busy. Given enough registers, unrolling 32 times will give the best efficiency where every memory reference slot will always be busy.
Assume an infinite loop, what is the average number of clock cycles per iteration?
Answer: 6 clock cycles per iteration for part b. 1.5 clock cycles per iteration for maximum efficiency.

Name: **ID:**

Part c) (15 points)

Part c.i) Consider a single issue dynamically scheduled machine without hardware speculation. Assume that the functional units are pipelined and that all memory accesses hit the cache. There is a memory unit with 5 load buffers and 5 store buffers. Each load or store takes 2 cycles to execute, 1 to calculate the address, and 1 to load/store the data. 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	Number of func. units	Number of reservation stations
Integer ALU	1	5
FP adder	1	3
FP multiplier	1	2
Load	1	5
Store	1	5

Now dynamically schedule 2 iterations of the original loop on this machine ***without speculation***. Show the clock cycle number of each stage of the dynamically scheduled code in the table below. Assume at least one cycle delay between successive steps of every instruction execution sequence (issue, execution start, write back).

Instruction	Operands	Issue	Execution Start	Write Back
L.D	F0,0(R1)	1	2	4
ADD.D	F0,F0,F4	2	5	8
L.D	F2,0(R2)	3	4	6
MUL.D	F2,F0,F2	4	9	15
S.D	F2,0(R2)	5	16*	17
DADDUI	R1,R1,#-8	6	7	9
DADDUI	R2,R2,#-8	7	8	10
BNEZ	R1,loop	8	10	11
L.D	F0,0(R1)	12(1)	13	16
ADD.D	F0,F0,F4	13	17	20
L.D	F2,0(R2)	14	16	18
MUL.D	F2,F0,F2	15	21	27
S.D	F2,0(R2)	16	28	29
DADDUI	R1,R1,#-8	17	18	19
DADDUI	R2,R2,#-8	18	19	21
BNEZ	R1,loop	19	20	22

*S.D here can calculate address while waiting for F2.

(1) L.D must wait for branch to return branch decision

Part c.ii) In this dynamically scheduled code, assume an infinite loop, what is the average number of clock cycles per old iteration?
Answer: 11 CC per iteration given that the second iteration can only start at cycle 12.

Part d) (15 points)

Now we use the dynamic scheduling hardware to build a speculative machine that can issue and commit 2 instructions per cycle. Again assume that the functional units are pipelined and that all memory accesses hit the cache. There is a memory unit with 8 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.

Functional unit type	Number of functional units	Number of reservation stations per functional unit
Integer ALU	2	4
FP adder	2	3
FP multiplier	2	2
Load	2	4

Part d.i) Schedule 2 iterations of the original code on this ***speculative*** machine in the table below. Assume at least one cycle delay between successive steps of every instruction execution sequence (issue, execution start, write back, commit). Assume two common data buses.

Instruction	Operands	Issue	Execution Start	Write Back	Commit
L.D	F0,0(R1)	1	2	4	5
ADD.D	F0,F0,F4	1	5	8	9
L.D	F2,0(R2)	2	3	5	9
MUL.D	F2,F0,F2	2	9	15	16
S.D	F2,0(R2)	3	4	5	16
DADDUI	R1,R1,#-8	3	4	6	17
DADDUI	R2,R2,#-8	4	5	6	17
BNEZ	R1,loop	4	7	8	18
L.D	F0,0(R1)	5	7	9	18
ADD.D	F0,F0,F4	5	10	13	19
L.D	F2,0(R2)	6	7	9	19
MUL.D	F2,F0,F2	6	14	20	21
S.D	F2,0(R2)	7	8	10	21
DADDUI	R1,R1,#-8	7	8	10	22
DADDUI	R2,R2,#-8	8	9	11	22
BNEZ	R1,loop	8	11	12	23

Part d.ii) Assume an infinite loop and no ROB overflow, what is the average number of clock cycles per iteration on this speculative machine? Compare with parts b and c.
Answer: It takes 4 CC per old iteration. The VLIW performs best, however it requires software scheduling. Note also that the VLIW is 5-issue whereas the speculative machine is double issue. Also, this speculative machine with double issues performs two loops in less than half the cycles required by the non speculative machine in part c.

Question 5. Performance (30 points)
There are 3 parts to this question.

Part a) Reliability and Amdahl's law. (10 points)
Consider a system in which the components have the following MTTF (in hours):

CPU	1,000,000
Hard disk	200,000
Memory	500,000
Power supply	100,000

Part a.i)
Assume that if any component fails, then the system fails. What is the system MTTF?

$$\text{MTTF}_{\text{system}} = \frac{1}{1000000^{-1} + 200000^{-1} + 500000^{-1} + 100000^{-1}} = 55555.55 \text{ hours}$$

Part a.ii)
You buy an additional hard drive and bring the total hard disk MTTF to 600000 hours, which provides 3 times improvement. Using Amdahl's law, compute the improvement in the whole system reliability?

The hard disk contributes $\frac{200000^{-1}}{1000000^{-1}+200000^{-1}+500000^{-1}+100000^{-1}} = 27.7\%$ of the total MTTF.

$$\text{improvement} = \frac{1}{1 - 27.7\% + \frac{27.7\%}{3}} = 1.22727$$

Part b) Cache performance (10 points)
Consider a memory system with latency of 60 clocks. The transfer rate is 4 bytes per clock cycle and that 30% of the transfers are dirty. There are 32 bytes per block and 25% of the instructions are data transfer instructions. There is no write buffer. In addition, the TLB takes 40 clock cycles on a TLB miss. A TLB does not slow down a cache hit. For the TLB, make the simplifying assumption that 0.5% of all references is not found in TLB, either when addresses come directly from the CPU or when addresses come from cache misses.

If the base CPI with a perfect memory system is 1.5, what is the CPI for a 16-KB two-way set-associative unified cache using write back with cache miss rate of 1.6%?

Compute the effective CPI for this cache with the real TLB.

Answers:

Since this is a unified cached, both instruction and data share the same cache and have the same transfer rate on block replacements.

Virtually addressed cache: An address from CPU will go through the cache first, and only on a cache miss it goes through the TLB.

$$CPI = 1.5 + (100\% + 25\%)(1.6\%)\big((68 + (0.5\%)40) + (30\%)(68 + (0.5\%)40)\big) = 3.2732$$

Physically addressed cache: An address from CPU will go through the TLB first, then through the cache.

$$CPI = 1.5 + (100\% + 25\%)(1.6\%)(68 + (30\%)68) + (100\% + 25\%)(0.5\%)(40) = 3.518$$

Part c) Branch prediction performance (10 points)

Suppose we have a deeply pipelined processor, for which we implement a branch-target buffer for the conditional branches and branch folding for the unconditional branches.

For the conditional branches, assume that the misprediction penalty is always 4 cycles and the buffer miss penalty is always 3 cycles. Assume 90% branch-target buffer hit rate and 90% target address accuracy, and 15% conditional branch frequency.

For branch folding that stores the target instructions of the unconditional branches, assume also a 90% hit rate and 5% unconditional branch frequency. Assume also that the hit target instruction can bypass the fetch stage and start immediately in the decode stage.

How much faster is this processor versus a processor that has a fixed 2-cycle branch penalty for both unconditional and conditional branches? Assume a base CPI without branch stalls of 1.

Answer:

$$CPI_{fixed} = 1 + (15\% + 5\%)(2) = 1.4$$

CPI of deeply pipelined processor assuming that the bypassing only happens for unconditional branches

$$\begin{aligned} CPI_{new} &= 1 + (15\%)(0.9 \times 0.9 \times 0 + 0.9 \times 0.1 \times 4 + 0.1 \times 0.9 \times 3 + 0.1 \times 0.1 \times 4) \\ &\quad + (5\%)(0.9 \times (-1) + 0.1 \times 2) \\ &= 1.0655 \end{aligned}$$

The deeply pipelined processor is 1.31 times faster than the fixed 2-cycle branch processor.