Name:____________________________

CSE 30321 – Computer Architecture I – Fall 2010
Final Exam

December 13, 2010

Test Guidelines:
1. Place your name on EACH page of the test in the space provided.
2. Answer every question in the space provided. If separate sheets are needed, make sure to include your name and clearly identify the problem being solved.
3. Read each question carefully. Ask questions if anything needs to be clarified.
4. The exam is open book and open notes.
5. All other points of the ND Honor Code are in effect!
6. Upon completion, please turn in the test and any scratch paper that you used.

Suggestion:
- Whenever possible, show your work and your thought process. This will make it easier for us to give you partial credit.

Question	Possible Points	Your Points
1	17	
2	10	
3	17	
4	15	
5	15	
6	17	
7	9	
Total	100	

Name:____________________________

Problem 1: (17 points)

Question A: (5 points)

A cache may be organized such that:

- In one case, there are more data elements per block and fewer blocks
- In another case, there are fewer elements per block but more blocks

However, in both cases – i.e. larger blocks but fewer of them OR shorter blocks, but more of them – the cache's total capacity (amount of data storage) remains the same.

What are the pros and cons of each organization? Support your answer with a short example assuming that the cache is direct mapped. Your answer must fit in the box below.

Solution:

If block size is larger:

- Con: There will be fewer blocks and hence a higher potential for conflict misses
- Pro: You may achieve better performance from spatial locality due to the larger block size
- Example: If you have a high degree of sequential data accesses, this makes more sense

If there are fewer elements per block and more blocks:

- Con: You may be more subject to compulsory misses due to the smaller block size
- Pro: You may see fewer conflict misses due to more unique mappings
- Example: If you have more random memory accesses, this makes more sense

Question B: (5 points)

Assume:

- A processor has a direct mapped cache
- Data words are *8 bits* long (i.e. 1 byte)
- Data addresses are *to the word*
- A physical address is *20 bits* long
- The tag is *11 bits*
- Each block holds 16 *bytes* of data

How many blocks are in this cache?

Solution:

Given that the physical address is 20 bits long, and the tag is 11 bits, there are 9 bits left over for the index and offset

We can determine the number of bits of offset as the problem states that:

- Data is word addressable and words are 8 bits long
- Each block holds 16 bytes

As there are 8 bits / byte, each block holds 16 words, thus **4** bits of **offset** are needed.

This means that there are 5 bits left for the index**. Thus, there are 2^5 or 32 blocks in the cache.**

Name:______________________________

Question C: (7 points)

Consider a *16-way set-associative* cache

- Data words are *64 bits* long
- Words are addressed to the *half-word*
- The cache holds 2 Mbytes of data
- Each block holds 16 data words
- Physical addresses are 64 bits long

How many bits of tag, index, and offset are needed to support references to this cache?

Solution:

We can calculate the number of bits for the **offset** first:

- There are 16 data words per block which implies that at least 4 bits are needed
- Because data is addressable to the ½ word, an additional bit of offset is needed
- Thus, the **offset** is **5 bits**

To calculate the **index**, we need to use the information given regarding the total capacity of the cache:

- 2 MB is equal to 2^{21} total bytes.
- We can use this information to determine the total number of *blocks* in the cache…
  - 2^{21} bytes **x** (1 block / 16 words) **x** (1 word / 64 bits) **x** (8 bits / 1 byte) = 2^{14} blocks
- Now, there are 16 (or 2^4) blocks / set
  - Therefore there are 2^{14} blocks **x** (1 set / 2^4 blocks) = 2^{10} or 1024 sets
- Thus, **10 bits of index** are needed

Finally, the remaining bits form the tag:

- 64 – 5 – 10 = 49
- Thus, there are **49 bits of tag**

To summarize: **Tag**: 49 bits; **Index**: 10 bits; **Offset**: 5 bits

Name:______________________________

Problem 2: (10 points)

Question A: (5 points)

The average memory access time for a microprocessor with 1 level of cache is 2.4 clock cycles

- If data is present and valid in the cache, it can be found in 1 clock cycle
- If data is not found in the cache, 80 clock cycles are needed to get it from off-chip memory

Designers are trying to improve the average memory access time to obtain a 65% improvement in average memory access time, and are considering adding a 2nd level of cache on-chip.

- This second level of cache could be accessed in 6 clock cycles
- The addition of this cache does not affect the first level cache's access patterns or hit times
- Off-chip accesses would still require 80 additional CCs.

To obtain the desired speedup, how often must data be found in the 2nd level cache?

Solution:

We must first determine the miss rate of the L1 cache to use in the revised AMAT formula:

AMAT	=	Hit Time	+	Miss Rate x Miss Penalty
2.4	=	1	+	Miss Rate x 80
Miss Rate	=	1.75%		

Next, we can calculate the target AMAT … i.e. $\text{AMAT}_{\text{with L2}}$:

Speedup	=	Time (old) / Time (new)
1.65	=	2.4 / Time (new)
Time (new)	=	1.4545 clock cycles

We can then again use the AMAT formula to solve for highest acceptable miss rate in L2:

$$1.4545 = 1 + 0.0175 \times (6 + (\text{Miss Rate}_{L2})(80))$$

Solving for Miss Rate_{L2} suggests that the highest possible miss rate is ~24.96%

Thus, as the hit rate is (1 – Miss Rate), the L2 hit rate must be ~75%.

Name:_____________________________

Question B: (5 points)

Assume that the base CPI for a pipelined datapath on a single core system is 1.

- *Note that this does NOT include the overhead associated with cache misses!!!*

Profiles of a benchmark suite that was run on this *single core chip* with an L1 cache suggest that for every 10,000,000 accesses to the cache, there are 308,752 L1 cache misses.

- If data is found in the cache, it can be accessed in 1 clock cycle, and there are no pipe stalls
- If data is not found in the cache, it can be accessed in 10 clock cycles

Now, consider a multi-core chip system where each core has an equivalent L1 cache:

- All cores references a common, centralized, shared memory
- Potential conflicts to shared data are resolved by snooping and an MSI coherency protocol

Benchmark profiling obtained by running the same benchmark suite on the multi-core system suggests that, on average, there are now 452,977 misses per 10,000,000 accesses.

- If data is found in a cache, it can still be accessed in 1 clock cycle
- On average, 14 cycles are now required to satisfy an L1 cache miss

What must the CPI of the multi-core *system* be for it to be worthwhile to abandon the single core approach?

Solution:

We first need to calculate the CPI of the single core system and incorporate the overhead of cache misses:

$$\begin{aligned} CPI &= 1 + (308{,}753 / 10{,}000{,}000)(10) \\ &= 1.309 \end{aligned}$$

We need to better this number with the multi-core approach. Thus:

$$\begin{aligned} 1.309 &> X + (452{,}977 / 10{,}000{,}000)(14) \\ 1.309 &> X + 0.634 \\ 0.675 &> X \end{aligned}$$

Thus, the CPI must be less than 0.675 → reasonable as execution will proceed in parallel.

Name:___________________________

Problem 3: (17 points)

Question A: (6 points)

Below, you are provided with a snapshot of a 4 entry, fully associative TLB and a page table.

- The TLB uses a "least recently used" replacement policy – i.e. the entry that has not been used for the longest time will be evicted if a new entry is to be added
- If needed, you may assume that the value of the page table register is 0
- The page size is 4 KB

Initial TLB State:

(Note that '1' = "Most Recently Used and '4' = "Least Recently Used")

(Note that '1' in the Valid column indicates that the entry is valid)

(Note that '0' in the Dirty column of the TLB indicates that data has NOT been modified)

Valid	Dirty	LRU	Tag	Physical Page #
1	0	3	0110	1000
1	0	4	0011	1101
1	0	2	1000	0110
1	0	1	0100	1010

Initial Page Table State:

Address	Valid	Physical Page #
0000	1	1110
0001	0	Disk
0010	1	1100
0011	1	1101
0100	1	1010
0101	1	1011
0110	1	1000
0111	1	1001
1000	1	0110
1001	1	0111
1010	1	0100
1011	0	Disk
1100	0	Disk
1101	1	0011
…	…	…

Show the final state of the TLB after processing the 2 virtual address sequence given below.

- If there is a page fault, the physical page number to be entered into the page table would be 0000; if there is another page fault, use physical page number 0001

Virtual addresses supplied: (MSB) 1011 ■ 0000 ■ 0001 ■ 0010 (LSB) (this is a load instruction)

(MSB) 1011 ■ 1101 ■ 1110 ■ 1111 (LSB) (this is a store instruction)

Name:______________________________

Solution:
The final TLB state is as shown below:
- The entry with tag 0011 has been evicted.
- Other LRU bits have been updated accordingly
- The fact that the last entry involves a write will cause the dirty bit to be set to 1.

Valid	Dirty	LRU	Tag	Physical Page #
1	0	4	0110	1000
1	**1**	**1**	**1011**	**0000**
1	0	3	1000	0110
1	0	2	0100	1010

Question B: (8 points)

- Assume that you have 4 Gbytes of main memory at your disposal
- 1 Gbyte of the 4 Gbytes has been reserved for process page table storage
- Each page table entry consists of:
    - A physical frame number
    - 1 valid bit
    - 1 dirty bit
    - 1 LRU status bit
- Virtual addresses are 32 bits
- Physical addresses are 26 bits
- The page size is 8 Kbytes

How many process page tables can fit in the 1 Gbyte space?

Solution:
We can first calculate the number of page table entries associated with each process.
- 8 KB pages implies that the offset associated with a VA/PA is 13 bits (2^{13} = 8KB)
- Thus, the remaining 19 bits are used to represent VPNs and serve as indices to the PT
- Thus, each PT has 2^{19} entries.

Next, each PA is 26 bits:
- 13 bits of the PA come from the offset, the other 13 come from a PT lookup
- Given that each PT entry consists of a PFN, a valid bit, a dirty bit, and a LRU bit, each PT entry will be 2 bytes

Therefore:

# of page tables = 2^{30} bytes available **x** (1 PT entry / 2 bytes) **x** (1 process / 2^{19} PT entries)

\# of page tables = 2^{10} or 1024.

Question C: (3 points)

If a virtual-to-physical address translation takes ~100 clock cycles, what is the most likely reason for this particular latency? Your answer must fit in the box below.

Solution:
- There was a TLB miss requiring a main memory access to read the page table
- The page table entry was valid and no page fault was incurred.

Name:_____________________________

Problem 4: (15 points)

This question considers the basic, MIPS, 5-stage pipeline (F, D, EX, M, WB).
(For this problem, you may assume that there is full forwarding for all questions.)

Question A: (5 points)

Assume that you have the following sequence of pipelined instructions:

```
lw      $6, 0($7)
add     $8, $9, $10
sub     $11, $6, $8
```

Where will the data operands that are processed during the EX stage of the subtract (sub) instruction come from? Also, you might take into account the following suggestions:

1. Draw a simple diagram to help support your answer.
2. If you are using more than a few sentences, or drawing anything put a simple picture to answer this question, you are making it too hard. You *don't* need to draw any control signals, etc.

Solution:

IF stage	**IF/ID latch**	ID stage	**ID/EX latch**	EX stage	**EX/Mem latch**	M stage	**M/WB latch**	WB stage

- The data associated with $6 in the subtract instruction will come from the M/WB latch
  - Could also say register file assuming you can read/write in same CC
- The data associated with $8 in the subtract instruction will come from the EX/M latch
- Muxes are needed to pick between the original input from the register file and forwarded data

Question B: (6 points)

Show how the instructions in the sequence given below will proceed through the pipeline:

- We will predict that the beq instruction is *not taken*
- When the beq instruction is executed, the value in $1 is equal to the value in $2

Solution:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
beq $1, $2, **X**	F	D	E												
lw $10, 0($11)		F	D												
sub $14, $10, $10			F												
X: add $4, $1, $2				F	D	E	M	W							
lw $1, 0($4)					F	D	E	M	W						
sub $1, $1, $1						F	D	D	E	M	W				
add $1, $1, $1							F	F	D	E	M	W			

Question C: (4 points)

For the instruction mix above, on what instruction results does the last add instruction depend on?

Solution:
Just the sub instruction before it; the sub produces the only data consumed by the add instruction

Name:______________________________

Problem 5: (15 points)
The multi-cycle and pipelined datapaths that we have discussed in class have generally been broken down into 5 steps:

1. Hardware to support an instruction *fetch*
2. Hardware to support an instruction *decode* (i.e. a register file read)
3. Hardware to support instruction *execution* (i.e. the ALU)
4. Hardware to support a *memory* load or store
5. Hardware to support the *write back* of the ALU operation back to the register file

Assume that each of the above steps takes the amount of time specified in the table below.

Fetch	Decode	Execute	Memory	Write Back
305 ps	275 ps	280 ps	305 ps	250 ps

*** *Note that these times* ***include*** *the overhead of performing the operation AND storing the data in the register needed to save intermediate results between steps.****

- *Thus, the times (Q) capture the critical path of the logic + latching overhead.*
- *After the Q seconds listed for each stage above, the data can be used by another stage*

Question A: (4 points)
Given the times for the datapath stages listed above, what would the *clock period* be for the entire datapath?

Solution:
- The clock period is defined by the longest time … here 305 ps.
- The clock rate is the inverse of the period … here, 3.28 GHz.
  - However, this is NOT the answer I am looking for.

Question B: (3 points)
In a pipelined datapath, assuming no hazards or stalls, how many seconds will it take to execute 1 instruction?

Solution:
1 instruction would need to proceed through all 5 stages of the pipeline
- Thus, the total time would be 305 ps x 5 stages = 1,525 ps or 1.525 x 10^{-9} s.

Name:______________________________

Question C: (4 points)

Assuming that N instructions are executed, and all N instructions are add instructions, what is the speedup of a pipelined implementation when compared to a multi-cycle implementation? Your answer should be an expression that is a function of N.

Solution:

For the multi-cycle approach:

- Each add instruction would take 4 clock cycles and each clock cycle would take 305 ps.
- Thus, the total time would be: 1220(N)

For the pipelined approach:

- For N instructions, we can apply the formula: NT + (S-1)T
- Thus, the total time would be:
  - = 305(N) + (5-1)(305)
  - = 305N + 1220 ps

Thus, the overall speedup is:

1220(N) / [305(N) + 1220]

Question D: (4 points) *This question should be answered independently of Questions A-C*

Assume you break up the memory stage into 2 stages instead of 1 to improve throughput in a pipelined datapath.

- Thus, the pipeline stages are now: F, D, EX, M1, M2, WB

Show how the instructions below would progress though this 6 stage pipeline. Like before, full forwarding hardware is available.

Solution:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $5, 0($4)	F	D	EX	M1	M2	WB									
add $7, $5, $5		F	D	D	D	EX	M1	M2	WB						
sub $8, $5, $9			F	F	F	D	EX	M1	M2	WB					

Name:______________________________

Problem 6: (17 points)

Question A: (4 points)

In your opinion, what are the 2 most significant impediments to obtaining speedup from N cores on a multi-core chip (irrespective of finding a good parallel algorithm)? You should also add a 1-2 sentence justification for each item. Your answer must fit in the box below.

Solution:
A good answer should mention 2 of the 3 things:
1. Cache coherency overhead
2. Contention for shared resources (i.e. the IC NW or higher level shared cache)
3. Latency

Question B: (9 points)

As was discussed in lecture, as more and more cores are placed on chip, it can make sense to connect them with some sort of interconnection network to support core-to-core communication.

Assume that you have a 6-core chip that you want to program to solve a given problem:
- You can use as few as 1 core or as many as 6 cores to solve the problem
- The problem require 450,000 iterations of a main loop to complete
- Each loop iteration requires 100 clock cycles
- Any startup overhead can be ignored
  - (i.e. instructions outside of the loop, to instantiate a new instance of the problem on another core, etc. etc.)

If more than 1 core is used to solve a problem, communication overhead must be added to the total execution time,
- Communication overhead is a function of the number of cores used to solve the problem, and is specified in the table below:

Number of cores used to solve problem	Communication overhead per iteration
1	0 cycles
2	10 cycles
3	20 cycles
4	30 cycles
5	40 cycles
6	50 cycles

- Thus, for example, the communication overhead if 2 cores are used is:
  - 10 cycles / iteration **x** 450,000 iterations = 4,500,000 cycles

How many cores should be used to solve this problem?

Name:______________________________

Solution:

We can write a simple expression for total execution time; assume:

- X is the number of cores used
- t_{loop} is the number of cycles per loop iteration
- N is the number of iterations for the problem

Thus:

T_{total} = $T_{interconnect}$ + T_{loops}

T_{total} = $(N)(X-1)(10)$ + $(N/X)(t_{loop})$

T_{total} = $10(N)(X) - 10(N) + 100(N/X)$

Subbing in problem data gives us:

T_{total} = $4.5\text{x}10^6(X) - 4.5\text{x}10^6 + (4.5\text{x}10^7)/(X)$

Plugging in 6 different numbers for X gives us:

X	**T_{total}**
1	4.500×10^7
2	2.700×10^7
3	**2.400×10^7**
4	2.475×10^7
5	2.700×10^7
6	3.000×10^7

The 3 core target is best.

Question C: (4 points)

- Assume that you have 10 cores that you can use to solve a problem in parallel
- 98% of your code is parallelizable

Can you get a speedup of 7? If so, how many cores are needed?

Solution:

This problem can be solved with Amdahl's law formula:

$$7 = \frac{1}{(1-0.98)+\frac{0.98}{X}}$$

If we solve for X, we find that we need 7.97 cores – or essentially 8 cores.

Name:____________________________

Problem 7: (9 points)
A snapshot of the state associated with 2 caches, on 2 separate cores, in a centralized shared memory system is shown below. In this system, cache coherency is maintained with an MSI snooping protocol. You can assume that the caches are direct mapped.

P0:

	Tag	Data Word 1	Data Word 2	Data Word 3	Data Word 4	Coherency State
Block 0	1000	10	20	30	40	M
Block 1	4000	500	600	700	800	S
…						
Block N	3000	2	4	6	8	S

P1:

	Tag	Data Word 1	Data Word 2	Data Word 3	Data Word 4	Coherency State
Block 0	1000	10	10	10	10	I
Block 1	8000	500	600	700	800	S
…						
Block N	3000	2	4	6	8	S

Question A: (3 points)
If P0 wants to write Block 0, what happens to its coherency state?

Solution:
Nothing. It has the only modified copy.

Question B: (3 points)
If P1 writes to Block 1, is Block 1 on P0 invalidated? Why or why not?

Solution:
No. The tags are different so this is different data.

Question C: (3 points)
If P1 brings in Block M for reading, and no other cache has a copy, what state is it cached in?

Solution:
It would still be cached in the shared state because this is not the MESI protocol.