

Name: _____

CSE 30321 – Computer Architecture I – Fall 2009
Final Exam
December 18, 2009

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	15	
2	10	
3	20	
4	15	
5	15	
6	10	
7	15	
Total	100	

Name: _____

Problem 1: (15 points)

Question A: (5 points)

Briefly (in 4-5 sentences or a bulleted list) explain why many of the transistors on a modern microprocessor chip are devoted to Level 1, Level 2, and sometimes Level 3 cache. Your answer must fit in the box below!

Students' answers should say something to the effect of:

- Processing logic is faster than off-chip, 1-transistor DRAM – and this performance gap has been continually growing
- Idea: bring in subsets of (off-chip) main memory into faster on-chip memory (SRAM) that can operate at the speed of the processor
- Caches help to ensure faster “data supply times” to ensure that logic is not idle for larger number of CCs (e.g. the time to access off-chip memory)
- If instruction encodings and data for load instructions could not be accessed in 1-2 CCs, CPU performance would be significantly (and negatively) impacted.
- *A discussion of spatial vs. temporal locality should receive little to no credit – speed differentials are the most important consideration in this answer.*

In HW 8, you saw that some versions of the Pentium 4 microprocessor have two 8 Kbyte, Level 1 caches – one for data and one for instructions. However, a design team is considering another option – a single, 16 Kbyte cache that holds both instructions and data.

Additional specs for the 16 Kbyte cache include:

- Each block will hold 32 bytes of data (not including tag, valid bit, etc.)
- The cache would be 2-way set associative
- Physical addresses are 32 bits
- Data is addressed to the word and words are 32 bits

Question B: (3 points)

How many blocks would be in this cache?

Answer

- The cache holds 2^{14} bytes of data
- Each block holds 2^5 bytes
- Thus, there are $2^{14} / 2^5 = 2^9 = 512$ blocks

Name: _____

Question C: (3 points)

How many bits of tag are stored with each block entry?

Answer

We need to figure out how many bits are dedicated to the offset, index and tag. (Basically, this question asks how many bits of tag are needed.)

- Index:
 - o # of sets: $1024 / 2 = 256 = 2^8$
 - o Therefore 8 bits of index are needed
- Offset:
 - o # of words per block = $32 / 4 = 8$
 - o $2^3 = 8$
 - o Therefore 3 bits of offset
- Tag
 - o $32 - 3 - 8 = 21$ bits of tag

Therefore, 21 bits of tag need to be stored in each block.

Question D: (4 points)

Each instruction fetch means a reference to the instruction cache and 35% of all instructions reference data memory. With the first implementation:

- The average miss rate in the L1 instruction cache was 2%
- The average miss rate in the L1 data cache was 10%
- In both cases, the miss penalty is 9 CCs

For the new design, the average miss rate is 3% for the cache as a whole, and the miss penalty is again 9 CCs.

Which design is better and by how much?

Answer

$$\text{Miss penalty}_{v1} = (1)(.02)(9) + (0.35)(.1)(9) = .18 + .063 = 0.495$$

$$\text{Miss penalty}_{v2} = (.03)(9) = 0.270$$

V2 is the right design choice

Problem 2: (10 points)Question A: (4 points)

Explain the advantages and disadvantages (in 4-5 sentences or a bulleted list) of using a direct mapped cache instead of an 8-way set associative cache. Your answer must fit in the box below!

Answer

- A direct mapped cache should have a faster hit time; there is only one block that data for a physical address can be mapped to
- The above “pro” can also be a “con”; if there are successive reads to 2 separate addresses that map to the same cache block, then there may never be a cache hit. This will significantly degrade performance.
- In contrast, with a set associative cache, a block can map to one of 8 blocks within a set. Thus, if the situation described above were to occur, both references would be hits and there would be no conflict misses.
- However, a set associative cache will take a bit longer to search – could decrease clock rate.

Question B: (2 points)

Assume you have a 2-way set associative cache.

- Words are 4 bytes
- **Addresses are to the byte**
- Each block holds 512 bytes
- There are 1024 blocks in the cache

If you reference a 32-bit physical address – and the cache is initially empty – how many data words are brought into the cache with this reference?

Answer

- The entire block will be filled
- If words are 4 bytes long and each block holds 512 bytes, there are $2^9 / 2^2$ words in the block
- i.e. there are 2^7 or 128 words in each block

Question C: (4 points)

Which set does the data that is brought in go to if the physical address F A B 1 2 3 8 9 (in hex) is supplied to the cache?

Answer

We need to determine what the index bits are. From above, we know the offset is 9 bits (remember, data is byte addressable) – so we will need to break up the hex address into binary:

1111 1010 1011 0001 0010 0011 1000 1001

Our offset for this address is: 1 1000 1001

$1024 / 2 = 2^{10} / 2^1 = 512 = 2^9$ – therefore 9 bits of index are required.

These are: 01 ■ 0010 ■ 001 which implies the address maps to the 145th set.

Problem 3: (20 points)Question A: (5 points)

Explain (in 4-5 sentences or via a short bulleted list) why there is translation lookaside buffer on the virtual-to-physical address critical path. Your answer must fit in the box below!

Answer

- The page table for a process/program can be huge – and the entire page table will almost certainly not be cacheable.
- As a page table reference is required for every address translation, if every instruction and every data reference required a main memory lookup, performance would quickly and significantly degrade.
- The TLB is a fast cache for part of the page table – and if (a) the TLB contains 64-128 entries and (b) each page has $2^{14} - 2^{16}$ addressable entries, the seemingly small TLB can provide wide coverage

For the next question, refer to the snapshot of TLB and page table state shown below.

Initial TLB State:

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

Valid	LRU	Tag	Physical Page #
1	3	1111	0001
1	4	0011	0010
1	2	1000	1000
1	1	0100	1010

Initial Page Table State:

	Valid	Physical Page #
0000	0	0011
0001	1	1001
0010	1	0000
0011	1	0010
0100	1	1010
0101	0	0100
0110	1	1011
0111	0	0101
1000	1	1000
1001	1	0110
1010	1	1111
1011	1	1101
1100	1	0111
1101	0	1110
1110	1	1100
1111	1	0001

Name: _____

Also:

1. Pages are 4 KB
2. There is a 4-entry, fully-associative TLB
3. The TLB uses a true, least-recently-used replacement policy

Question B: (5 points)

Assume that the Page Table Register is equal to 0.

The virtual address supplied is:

(MSB) 1100 ■ 0010 ■ 0010 ■ 0100 (LSB)

What physical address is calculated? If you cannot calculate the physical address because of a page fault, please just write "Page Fault".

Answer:

- Our VPN is 1100
- This entry is *not* in one of the tag fields in the TLB ... so we need to look in the page table
- The entry in the page table is valid – and suggests that the PFN is 0111
- We can then concatenate 0111 to 0010 0010 0100
- Thus, our PA is: 0111 ■ 0010 ■ 0010 ■ 0100 (or 7224 hex)

Question C: (5 points)

Consider the following:

- Virtual address are 32 bits
- Pages have 65,536 (or 2^{16}) addressable entries
- Each page table entry has:
 - o 1 valid bit
 - o 1 dirty bit
 - o The physical frame number
- Physical addresses are 30 bits long

How much memory would we need to simultaneously hold the page tables for two different processes?

Answer:

- Our VPN is 16 bits long – so each page table will have 2^{16} entries
- There are also 16 bits left over for offset
- Thus, each physical frame number is $30 - 16 = 14$ bits
- Each page table entry will hold the above 14 bit PFN + 1 valid bit + 1 dirty bit
 - o Thus, each page table entry is 2 bytes.
- There are 2^{16} , 2-byte entries per process
 - o Thus, each page table requires 2^{17} bytes – or ~128 Kbytes
- Because each process has its own page table, a total of 2^{18} bytes – or ~256 Kbytes are needed

Name: _____

Question D: (5 points)

Assume that for a given system, virtual addresses are 40 bits long and physical addresses are 30 bits long. There are 8 Kbytes of addressable entries per page. The TLB in the address translation path has 128 entries. How many virtual addresses can be quickly translated by the TLB? Would changing the page size make your answer better or worse? – **Note that there are 2 questions here!**

Answer:

There are 128 (2^7) entries in the TLB and there are 8192 (2^{13}) entries per page. Therefore $2^7 \times 2^{13}$ implies 2^{20} (or 1 MB) addressable entries are covered by the TLB.

The answer would get better b/c 2^{13} would be larger.

Name: _____

Problem 4: (15 points)

This question considers the basic, MIPS, 5-stage pipeline. (For this problem, you may assume that there is full forwarding.)

Question A: (5 points)

Explain how pipelining can improve the performance of a given instruction mix. Answer in the box.

Answer

- Pipelining provides “pseudo-parallelism” – instructions are still issued sequentially, but their overall execution (i.e. from fetch to write back) overlaps
 - o (note – a completely right answer really needs to make this point)
- Performance is improved via higher throughput
 - o (note – a completely right answer really needs to make this point too)
 - o An instruction (ideally) finishes every short CC

Question B: (6 points)

Show how the instructions will flow through the pipeline:

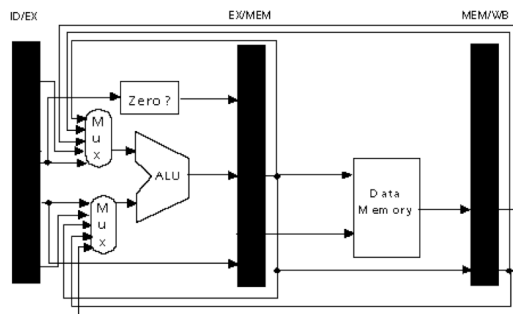
	1	2	3	4	5	6	7	8	9	10	11	12
lw \$10, 0(\$11)	F	D	E	M	W							
add \$9, \$11, \$11		F	D	E	M	W						
sub \$8, \$10, \$9			F	D	E	M	W					
lw \$7, 0(\$8)				F	D	E	M	W				
sw \$7, 4(\$8)					F	D	E	M	W			

Question C: (4 points)

Where might the sw instruction get its data from? Be very specific. (i.e. “from the lw instruction” is *not* a good answer!)

Answer

- Data is available from the lw in the MEM / WB register
- There could be a feedback path from this register back to one of the inputs to data memory
- A control signal could then select the appropriate input to data memory to support this lw – sw input combination



Name: _____

Problem 5: (15 points)

This question considers the basic, MIPS, 5-stage pipeline. (For this problem, you may assume that there is full forwarding.)

Question A: (5 points)

Using pipelining as context, explain why very accurate branch prediction is important in advanced computer architectures. Answer in the box below.

Answer

- If we need to wait for the branch to be resolved (e.g. after 3 stages) performance will be adversely affected – especially given that ~1 in every 6 instructions is a branch
 - o 3 stall CCs would be tacked on to the base CPI of 1

Question B: (7 points)

Show how the instructions will flow through the pipeline.

- You should assume that branches are predicted to be taken.
- You should assume that $0(\$2) == 4$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw \$1, 0(\$2)	F	D	E	M	W											
bneq \$0, \$1, X		F	D	D	E	M	W									
add \$2, \$1, \$1																
X: add \$2, \$1, \$0			F	F	D	E	M	W								
sw \$2, 4(\$2)					F	D	E	M	W							

Question C: (3 points)

What is the value that will be written to 4(\$2) in the last instruction?

Answer

- lw \$1, 0(\$2): $\$1 \leftarrow 4$
- bneq \$0, \$1, X: $\$0 \neq \1 , therefore goto X (also predict taken)
- add \$2, \$1, \$0: $\$2 \leftarrow 4 + 0 = 4$
- sw \$2, 4(\$2): $4(\$2) \leftarrow 4$

Problem 6: (10 points)

Consider an Intel P4 microprocessor with a 16 Kbyte unified L1 cache. The miss rate for this cache is 3% and the hit time is 2 CCs. The processor also has an 8 Mbyte, on-chip L2 cache. 95% of the time, data requests to the L2 cache are found. If data is not found in the L2 cache, a request is made to a 4 Gbyte main memory. The time to service a memory request is 100,000 CCs. On average, it takes 3.5 CCs to process a memory request. How often is data found in main memory?

$$\text{Average memory access time} = \text{Hit Time} + (\text{Miss Rate} \times \text{Miss Penalty})$$

$$\text{Average memory access time} = \text{Hit Time}_{L1} + (\text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1})$$

$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + (\text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

$$\text{Miss Penalty}_{L2} = \text{Hit Time}_{\text{Main}} + (\text{Miss Rate}_{\text{Main}} \times \text{Miss Penalty}_{\text{Main}})$$

$$3.5 = 2 + 0.03 (15 + 0.05 (200 + X (100,000)))$$

$$3.5 = 2 + 0.03 (15 + 10 + 5000X)$$

$$3.5 = 2 + 0.03 (25 + 5000X)$$

$$3.5 = 2 + 0.75 + 150X$$

$$3.5 = 2.75 + 150X$$

$$0.75 = 150X$$

$$X = .005$$

Thus, 99.5% of the time, we find the data we are looking for in main memory.

Problem 7: (15 points)

- Assume that for a given problem, N CCs are needed to process each data element.
- The microprocessor that this problem will run on has 4 cores.
- If we want to solve part of the problem on another core, we will need to spend 250 CCs for each instantiation on a new core.
 - o (e.g. if a problem is split up on 2 cores, 1 instantiation is needed)
- Also, if the problem is to run on multiple cores, an overhead of 10 CCs *per data element* is associated with each instantiation.
- We want to process M data elements.

Question A: (8 points)

If N is equal to 500 and M is equal to 100000, what speedup do we get if we run the program on 4 cores instead of 1? (*Hint – start by writing an expression!*)

Answer:

- To run the program on 1 core, $M \times N$ CCs are required:
 - o Thus, $500 \times 100,000 = 50,000,000$ CCs
- To run the program on 4 cores, we need to consider overhead:
 - o We need to start jobs on 3 other cores...
 - Thus, $250 \text{ CCs} \times 3 = 750 \text{ CCs}$ are needed
 - o We need to send a certain number of data elements to each core
 - i.e. $100,000 / 4 = 25,000$ elements must be sent to each core
 - Thus, an overhead of $25,000 \times 10 \times 3 = 750,000$ CCs is required
 - o We still need to do the computation itself...
 - ... but now this is split up over 4 cores...
 - Thus, the computation time is equal to $500 \times 100,000 / 4 = 12,500,000$ CCs
- We can use this analysis to create a generalized expression for running jobs on a single core and running jobs on multiple cores:
 - o Single core:
 - Execution time_{single core} = $M \times N$
 - o Multiple cores:
 - Execution time_{multiple cores} =
 - Time to process + Time to instantiate + Time to send data
 - $[(N \times M) / \# \text{ of cores}] + [(\# \text{ of cores} - 1)(250)] + [(M / \# \text{ of cores})(\# \text{ of cores} - 1)(10)]$
- Using these expressions, we can calculate single core and multi-core execution times:
 - o Single core = $500 \times 100,000 = 50,000,000$
 - o Multi-core = $(500 \times 100,000 / 4) + (3 \times 250) + [(100,000 / 4)(3)(10)]$
 - = $12,500,000 + 750 + 750,000$
 - = $13,250,750$
 - o Speedup = $50,000,000 / 13,250,750 = 3.77$

Name: _____

Question B: (7 points)

If N is equal to 250, M is equal to 120, the communication overhead increases to 100 CC per element, and the instantiation overhead remains the same (at 250 CC), how many cores should we run the problem on? Explain why.

Answer:

- We can use this expressions generated above:
 - o Single core:
 - Execution time_{single core} = M x N
 - o Multiple cores:
 - Execution time_{multiple cores} =
 - Time to process + Time to instantiate + Time to send data
 - $[(N \times M) / \# \text{ of cores}] + [(\# \text{ of cores} - 1)(250)] + [(M / \# \text{ of cores})(\# \text{ of cores} - 1)(100)]$
- The easiest way to solve this problem is to just compare all the cases by doing some trivial math
 - o Single Core
 - $250 \times 100 = 30,000$
 - o 2 Cores
 - = $(30,000 / 2) + (1)(250) + (120/2)(1)(100)$
 - = $15,000 + 250 + 6,000$
 - = $21,250$
 - o 3 Cores
 - = $(30,000 / 3) + (2)(250) + (120/3)(2)(100)$
 - = $10,000 + 500 + 8,000$
 - = $18,500$
 - o 4 Cores
 - = $(30,000 / 4) + (3)(250) + (120/4)(3)(100)$
 - = $7,500 + 750 + 9,000$
 - = $17,250$
- Thus, 4 cores still makes sense!