Computer Science 605.611

Problem Set 10

(Recall that MIPS memory is byte addressable; its size is measured in units of bytes not bits.)

1. (5) A MIPS system employs a 4-way set associative I-cache (i.e., instruction cache) that can hold a total of 262144 instructions and uses a 256-byte line size. The second instruction within way3 of set 0x29 is fetched. If way3 has tag 0x345, what is the corresponding 32-bit physical address (expressed in hex 0xdddddddd) of the instruction that is fetched?

256 bytes requires offset field of 8 bits
The second instruction is at a offset of 2 * 4 (bytes per MIPS instruction) = 00001000
4 way set associative cache requires 2 bits to specify set
way3 corresponds to 11
remaining 22 bits corresponds to the tag
0x345 = 0000000000 0011 0100 0101
Full address is
0000 0000 0000 1101 0001 0111 0000 1000 = 0x000d1708

2. (5) A fetch of an instruction from address 0x4048C824 on a MIPS systems causes a hit in the I-cache. The direct-mapped I-cache is 2097152 bytes in size and has 64-byte cache lines. Show, in decimal, the tag, the line number and the offset within the line for the instruction that is fetched. Tag = _____, Line# = _____, Offset = _____

2,097,152 = 2 ^ 21, thus the line field must be 21 bits long
64 = 2 ^ 6, thus the offset must be 6 bits long
32 – 21 – 6 = 5, thus the tag must be 5 bits long

0x4048C824 = 0100 0000 0100 1000 1100 1000 0010 0100

Tag = 01000 = 8
Offset = 100100 = 36
Line = 00001001000011001000000 = 74528

3. The MIPS instruction sequence below shows one way to implement the high level statement: X = X+Y

```
lui     $t0,0x3D5A          ; load $t0 with base address 0x3D5A0000
lui     $t2,0x3D5A          ; load $t2 with the same base address
lw      $t1,0x4BFC($t0)     ; read variable X into $t1
lw      $t3,0x4B08($t2)     ; read variable Y into $t3
add     $t1,$t1,$t3         ; compute X + Y
```

```
sw      $t1, 0x4BFC($t0)      ; store sum back into X
```

The 32-bit variables X and Y reside at physical memory addresses 0x3D5A4BFC and 0x3D5A4B08 respectively.
If the corresponding machine code is executed on a system with a D-cache (data cache) that is a direct mapped, write allocate, copy-back cache of size 524288 bytes, how many D-cache hits and how many D-cache misses occur if the D-cache is initially empty and the line size is:

0x3D5A4BFC / 128 = 0011 1101 0101 1010 0100 1011 1111 1100 / 100 0000
=00111101010110100100101111
0x3D5A4B08 / 128 = 1101 0101 1010 0100 1011 0000 1000 / 1000000
= 00111101010110100100101110
(map to different cache addresses, thus first read does not load second value to cache)

a) (5) 128 bytes #read hits = 0 #read misses = 2
#write hits = 1 #write misses = 0

0x3D5A4BFC / 1024 = 0011 1101 0101 1010 0100 1011 1111 1100 / 10 0000 0000
=00111101010110100100101
0x3D5A4B08 / 1024 = 1101 0101 1010 0100 1011 0000 1000 / 10 0000 0000
= 00111101010110100100101
(map to the same cache addresses, thus first read loads second value to cache)

b) (5) 1024 bytes #read hits = 1 #read misses = 1
#write hits = 1 #write misses = 0

4. a) (5) Assume that our MIPS system employs a 2-way set associative D-cache that has a total data storage capacity of 4194304 bytes and a 32-byte line size. To which set does address 0xC0404248 map?

0xC0404248 = 1100 0000 0100 0000 0100 0010 0100 1000
In this case each set contains 2 lines, and is therefore 2 * 32 = 64 bytes in size
The number of sets is 4194304 / 64 = 65536
The set field requires lg2(65536) = 16 bits
The width of the offset is lg2(32) = 5 bits
11000000010 0000001000010010 01000
Therefore the address maps to set 0000001000010010 = 530

b) (5) Assume instead that the D-cache has a total data storage capacity of 8388608 bytes and a 4-way set associative organization with a 64-byte line size. To which set does address 0x4248C040 map?

0x4248C040 = 0100 0010 0100 1000 1100 0000 0100 0000
In this case each set contains 4 lines, and is therefore 4 * 64 = 256 bytes in size
The number of sets is 8388608 / 256 = 32768
The set field requires lg2(32768) = 15 bits
The width of the offset is lg2(64) = 6 bits
01000010010010 010001100000001 000000
Therefore the address maps to set 010001100000001 = 8961

5. Recall that with a write-allocate policy, if a cache miss occurs the memory block containing the referenced address is first loaded into the cache and then updated. Suppose that for a 4-way set associative cache, the 3 pseudo-LRU bits for set 7 are currently 100 and the set is full. The cache employs a write-allocate policy and the next memory reference that maps to set 7 is a write miss. The pseudo-LRU bits are described in the sub-module on cache replacement policies.

a) (10) Which of the 4 lines (way0, way1, way2 or way3) within set 7 will be replaced in response to the write miss?

B2 B1 B0	Way to replace
000	0
001	2
010	1
011	2
100	0
101	3
110	1
111	3

Bits 100 correspond to replacing way0.

b) (5) Suppose that the reference to set 7 had instead been a read hit in way2, and that the three pseudo-LRU bits for set 7 were 010 before the read was performed. After the read hit, what is the pattern for the three pseudo-LRU bits?

Way accessed	Effect on LRU bits
0	1->B1, 1->B0
1	0->B1, 1->B0
2	1->B2, 0->B0
3	0->B1, 0->B0

Read hit in way 2 converts B2 to 1 and B0 to 0, while B1 was 1 initially
The B2 B1 B0 pattern after the read is 110

6. A system employs a unified L1 cache with a read access time of 8ns. The main memory has a read access time of 60ns.

a) (6) Using the decimal format dd.dd% (i.e., 2 digits before and 2 digits after the decimal point) for your answer, what read hit ratio is required to produce an average read access time of 14ns if the cache operates in look-through mode.

14 ns = r * (8 ns) + (1 – r) * (8 ns + 60 ns)
14 = 8r – 68r + 68
54 = 60r
Read hit ratio = 90.00%

b) (6) If instead, the cache operates in look-aside mode and has a read hit ratio of 93%, what is the average read access time in nano-seconds?

Average read access time = 0.93 * 8 ns + 0.07 * 60 ns
Average read access time = 11.64 ns

7. A virtual memory system employs 45-bit virtual addresses, uses a page size of 8192 bytes and contains 268435456 words of physical memory (each word, as usual, is 32 bits).

a) (4) State the minimum number of bits required for physical addresses on this system.

268435456 = 2 ^ 28 words
Each word has 4 = 2^2 bytes
Thus the minimum number of bits required for the physical address is 2 + 28 = 30 bits

b) (5) What is the minimum size (in bytes) of the inverted page map table that could be used for this system if each entry in the inverted page map table contains only a valid bit, a dirty bit, a 6-bit task id plus any required addressing bits.

The number of bits in the page number = 45 – lg2(8192) = 32
Each entry in the page map table will contain 1 valid bit, 1 dirty bit, 6 task bits, and the 32 bit page number for a total of 40 bits or 5 bytes
#entries = 268435456 * 4 / 8192 = 131072 entries
size of PMT = #entries * entry size = 131072 * 5 = 655,360 bytes

8. A matrix of 32-bit integers, x[512][32] (i.e. 512 rows with 32 elements per row) resides at memory address 3EA4000 on a machine with a virtual memory system that employs a page size of 8192 bytes. Assume that the amount of physical memory available to contain pages from the matrix is 32768 bytes, and that an LRU page replacement algorithm is used. Initially, all page map table entries are invalid (i.e., none of the required pages are in memory). Considering only the page faults generated in referencing the matrix (i.e., ignoring those that might result from instruction fetches or referencing the loop indices), indicate how many page faults would occur in executing the C language instruction sequence shown below. Note that in the C language, matrices are stored in row major order (i.e., one row after another).

a) (10)
```
for (j = 0; j < 30; j++) {
    for (k = 0; k < 350; k++)    x[k][j]  = x[k][j] + 1;
}
```
Number of page faults due to accessing the matrix = 180
Explain how you arrived at your answer.

32768 / 8192 = 4 pages
row 0 stored at 0x3EA4000 = 0011 1110 1010 0100 0000 0000 0000
row 1 stored at 0x3EA4010 = 0011 1110 1010 0100 0000 0001 0000
row 2 stored at 0x3EA4020 = 0011 1110 1010 0100 0000 0010 0000
8196 byte page size can store 8196*8/32/32 = 64 rows at once.
In outer loop iteration 0:
- In inner loop 0-63 there is one page fault to load the 1st 64 rows
- In inner loop 64-127 there is one page fault to load the 2nd 64 rows
- In inner loop 128-191 there is one page fault to load the 3rd 64 rows

In inner loop 192-255 there is one page fault to load the 4th 64 rows

In inner loop 256-319 there is one page fault to load the 5th 64 rows, overwriting the 1st page

In inner loop 320-383 there is one page fault to load the 6th 64 rows, overwriting the 2nd page

There are 6 page faults in the 0^{th} iteration of the outer loop.

In outer loop iteration 1:

In inner loop 0-63 there is one page fault to load the 1st 64 rows, since those were overwritten by the 5^{th} inner iteration of the 0^{th} outer loop

In inner loop 64-127 there is one page fault to load the 2nd 64 rows, since those were overwritten by the 6^{th} inner iteration of the 0^{th} outer loop

In inner loop 128-191 there is one page fault to load the 3rd 64 rows, since those were overwritten by the 1st inner iteration of the 1st outer loop

…

6 page faults per iteration of the outer loop.

6*30 = 180 page faults

b) (10) Suppose that instead, the system employs 4096-byte pages and has a total of eight 4096-byte frames of physical memory. An LRU page replacement algorithm is used and all page map table entries are initially set to invalid. Considering only the page faults generated in referencing the matrix (i.e. ignoring those that might result from instruction fetches or the loop indices), indicate how many page faults occur in executing the instruction sequence shown below using row major storage order for the matrix.

```
k=0;
while( k < 350 ) {
        j = 0;
        while( j < 30)   {
                x[k][j]  = x[k][j] + 1;
                j = j + 2;
        }
   k = k + 2;
}
```

Number of page faults due to accessing the matrix = 11

Explain how you arrived at your answer.

4096 byte frames * 8 frames = 32768 bytes main memory

4096 byte page size can store 4096*8/32/32 = 32 rows at once.

In the outer loop, there is one page fault to load the first 32 rows (k<32), while there are no page faults on the inner loop since all elements are already stored in memory

In the outer loop, there is one page fault to load the second 32 rows (32 <= k < 64), and there are still no page faults for the inner loop.

For 350 rows, there will be 11 page faults (350/32 -> round up).

9. A computer uses a byte-addressable virtual memory system with a TLB (translation look-aside buffer) that contains four entries, a 2-way set associative cache, and a page map table for a process P. Cache blocks are 8 bytes in size, while pages are 16 bytes in size. Assume, for the sake of this problem, that main memory contains only 4 frames and that the TLB and page table contents for Process P are as shown below. There is a separate PTE (page table entry) for each of the pages in the virtual address space. Based on the information in these tables, answer the following questions.

TLB	
page	frame
0	3
4	1
–	–
–	–

	Page Table	
	frame	Valid
0	3	1
1	0	1
2	-	0
3	2	1
4	1	1
5	-	0
6	-	0
7	-	0

a) (4) What is the minimum number of bits required for a virtual address?

Virtual memory has size 2*4*16 bytes, thus needs lg2(128) = 7 bits for the virtual address

b) (4) What is the minimum number of bits required for a physical address?

Main memory has size 4*16 bytes, thus needs lg2(64) = 6 bits for the physical address

c) (3) If the virtual address 0x48 is referenced, does a TLB hit occur?

0x48 = 100 1000
Offset is the last 4 bits
page # is the first 3 bits (100 references page 4)
Page 4 references frame 1, thus there is a TLB hit

d) (3) Does referencing virtual address 0x34 cause a TLB hit? To what physical address (if any) does virtual address 0x34 correspond?

0x34 = 011 0100
Offset is the last 4 bits
page # is the first 3 bits (011 references page 3)

Page 3 will not cause a hit in the TLB