Computer Science 605.611

Problem Set 10 Answers

**(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 0xddddddddd) of the instruction that is fetched?

Since the cache holds 262144 four-byte instructions, its size = 4*262144 = 1048576 bytes. Each set contains 4 ways (i.e., lines), so the total number of sets in the cache is 1048576 / (4*256) = 1024. Therefore 10 bits are required for the set number. Each line is 256 bytes, so 8 bits are needed for the offset within a line.
Hence the tag field requires 32 - 10 - 8 = 14 bits.
The second instruction within each line is at offset 4. The 14-bit tag field contains the value 0x345, the 10-bit set field contains the value 0x29 and the 8-bit offset field contains the value 4.
This corresponds to address 00001101000101 0000101001 00000100 = 0x0D142904

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 = 514__, Line# = ___8992___, Offset = ___36__

The cache contains 2097152/64 = 32768 lines,
so the line number field requires $\log_2(32768) = 15$ bits.
The offset field width = $\log_2(64) = 6$ bits.
So the leftmost 32 – (15+6) = 11 bits contain the tag.
0x4048C824 = 01000000010 010001100100000 100100
Therefore the tag = 01000000010 = 0x202 = decimal 514
Line number = 010001100100000 = 0x2320 = decimal 8992
Offset = 100100 = 0x24 = decimal 36

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:

a) (5) 128 bytes
If the line size is 128 bytes, then the number of lines in the D-cache = 524288/128 = 4096, so the line number field contains 12 bits and the offset field is 7 bits.
X resides at address 0x3D5A4BFC = 00111101010110100100101111111100, which maps to line 010010010111= 0x497
Y resides at address 0x3D5A4B08 = 00111101010110100100101100001000 which maps to line 010010010110 = 0x496.
When X is first read a miss occurs and the block containing X is brought into line 0x497 of the D-cache. When Y is read a miss also occurs and the memory block containing Y is brought into line 0x496 of the cache. When the sum is written to X, a cache write hit occurs in line 0x497. So 2 read misses and 1 write hit occurs for the D-cache.
#read hits = _0__ #read misses = __2___
#write hits = _1__ #write misses = _0__

b) (5) 1024 bytes
For 1024-byte lines, the number of lines in the D-cache = 524288/1024 = 512, so the line number field contains 9 bits and the offset contains 10 bits.

X resides at address 0x3D5A4BFC = 00111101010110100100101111111100, which maps to line 010010010 = 0x92
Y resides at address 0x3D5A4B08 = 00111101010110100100101100001000 which also maps to line 010010010= 0x92.
Both X and Y reside in the same memory block. Recall that the memory block size and the cache line size are always the same.
When X is first read a miss occurs and the block containing X is brought into line 0x92 of the D-cache. When Y is read, a hit occurs since X and Y are within the same cache line. When the sum is written, a cache write hit occurs.
So 1 read miss and 1 read hit occurs in the D-cache. One write hit and zero write misses occur in the D-cache.
#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?

For a 2-way set associative caches, each set contains two lines so the set size is 2 * 32 bytes = 64 bytes.

Hence the number of sets in the cache is 4194304/64 = 65636, so 16 bits are required for the set number. For 32-byte lines the offset can range from 0 to 31 and requires 5 bits.

0xC0404248 = 11000000010 0000001000010010 01000 so the address maps to set 0000001000010010 binary = 0x0212 = decimal 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?

Each set is 4 * 64 bytes = 256 bytes in size.

Hence the number of sets in the cache is 8388608/256 = 32768, so 15 bits are required for the set number. For 64-byte lines, the offset can range from 0 to 63 and requires 6 bits.

0x4248C040 = 01000010010 010001100000001 000000 so the address maps to set 010001100000001 binary = 0x2301 = decimal 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?

Due to the write-allocate policy, a new line is brought into set 7 in response to the miss. The pseudo-LRU bit pattern 100 selects way 0 to be replaced.

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

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?

Since way 2 was accessed, B2 is set to 1 and B0 is set to 0. Hence the 3 bits are changed from 010 to 110.

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

Reading, writing or replacing a line is considered an "access"

LRU bits are updated for each access

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.

Read hit ratio = ___90.00%_

For look-through mode, the average access time is given by

ta = tc + (1-h)*tm so (1-h) = (ta – tc)/tm

h = 1 - (ta – tc)/tm = 1 - (14-8)/60 = 54/60 = 0.90 or 90%

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 = __11.64 ns_

For look-aside mode, the average access time is given by

ta = h*tc + (1-h)*tm = 0.93*8 + 0.07*60 = 7.44 + 4.20 = 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.
There are 268435456*4 = 1073741824 = $2^{30}$ bytes of physical memory. So at least 30 bits are needed for the physical address.

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.
In addition to the valid bit, the dirty bit, and the 6-bit task id, each page table entry (PTE) must also contain the page number of the page that occupies the corresponding frame. The 45-bit virtual address contains a 13-bit offset for the 8192-byte pages. Hence the number of bits in the virtual page number is the virtual address size minus the offset size = 45-13 = 32.
Therefore each PTE must contain at least a 32-bit page number, a valid bit, a dirty bit and a 6-bit task ID for a total of 40 bits. Five bytes are required to hold all 40 bits. There is a separate PTE for each frame and there are 1073741824 ÷ 8192 = 131072 frames. So the inverted page map table must be at least 131072*5 bytes = 655360 bytes in size.

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.

There are **32768 /8192** = 4 frames of physical memory available. For each of the 30 columns (0 through 29) this code accesses rows 0 through 349 in sequence. That is, consecutive accesses hop from one row to the next. Each row is 32*4=128 bytes in size.

So each page can hold 8192/128 = 64 rows. The code accesses rows 0 through 349 causing page faults as indicated in the following table:

| Row access causing page fault | rows loaded in response to page fault | Frame used |
|---|---|---|
| 0 | 0 – 63 | 0 |
| 64 | 64 – 127 | 1 |
| 128 | 128 – 191 | 2 |
| 192 | 192 – 255 | 3 |
| 256 | 256 – 319 | 0 |
| 320 | 320 – 351 | 1 |

There are 6 page faults from the first iteration of the inner loop. Due to the LRU page replacement, each page will be replaced before it is required a second time. Therefore, each iteration of the inner loop will cause 6 page faults. The outer loop is executed 30 times, so the total number of page faults is 30*6 = 180.

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.

Each page contains 4096/128 = 32 rows. This code accesses even numbered rows 0 through 348 of the matrix (125 rows). Within

each row the 15 even numbered columns 0 through 28 are accessed in sequence. Hence after a page is loaded in response to a fault, the inner loop generates no additional page faults since the references made in the inner loop remain on the page. The code causes page faults as indicated in the following table:

| Row access causing page fault | rows loaded in response to page fault | Frame used |
|---|---|---|
| 0 | 0 – 31 | 0 |
| 32 | 32 – 63 | 1 |
| 64 | 64 – 95 | 2 |
| 96 | 96 – 127 | 3 |
| 128 | 128 – 159 | 4 |
| 160 | 160 – 191 | 5 |
| 192 | 192 - 223 | 6 |
| 224 | 224 - 255 | 7 |
| 256 | 256 - 287 | 0 |
| 288 | 288 - 319 | 1 |
| 320 | 320 - 351 | 2 |

Each page fault brings in 32 rows, so all 350 rows will have been loaded after 11 page faults. Hence only 11 pages faults are caused by this code.

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?

Page map tables contain a separate PTE for each virtual page. Since there are 8 entries in the page map table, there are 8 virtual pages, so the page number requires 3 bits. A 16-byte page requires a 4-bit offset. So the minimum number of bits needed for the virtual address is 3+4 =7.

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

The problem states that there are four frames, so two bits are required for the frame number. Frames are the same size as pages, so each frame is 16 bytes in size, which requires a 4-bit offset. The minimum number of bits in the physical address is 2+4 = 6.

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

Virtual address 0x48 corresponds to page 4. The second entry in the TLB indicates that page 4 resides in frame 1, so a TLB hit occurs.

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

0x34 = 0110100 as a 7-bit binary number

So the page number is 3 (binary 011) and the offset is 4 (binary 0100).

There is no entry in the TLB for page 3, so a TLB hit does not occur. Checking the PMT, we see that page 3 maps to frame 2. So virtual address 0x34 maps to physical address 100100 binary = 0x24.