Computer Science 605.411

Module 10 Example Set 1

1. A virtual memory system employs three levels of page map tables with 50-bit virtual addresses and 32-bit physical addresses. The page size is 16384 bytes. The size of each page map table is the same as the page size. PTEs (page table entries) within all tables are 4 bytes wide.
   a) (10) How many bits would be required for each virtual page number?
      Since the page size is 16384 bytes, 14 bits are required for the page offset. Hence 50-14 = 36 bits are required for the virtual page number.
   b) (10) Identify the numbers (counting from 0) of all page tables and page table entries that would be accessed in translating the virtual address 0x48D159E26AF0.
      Since there are 3 levels of page tables, the 36-bit virtual page number field would be subdivided into three 12-bit fields. The virtual address 0x48D159E26AF0 would be partitioned as 000100100011 010001010110 011110001001 101010111110000.
      Hence the page tables and entries that would be accessed are:
      000100100011 = 291 in the level 0 page table (the top level table)
      010001010110 = decimal 1110 in the level 1 page table
      011110001001 = decimal 1929 in the level 2 page table

2. A system employs a single inverted page map table in which each page table entry contains four items: a valid bit, a dirty bit, a 2-bit ID and the information needed to complete the page to frame translation. If this system employs 48-bit virtual addresses, 134217728 bytes of main memory, and 4096-byte pages,
   a) (5) what is the minimum physical address size in bits?
      A minimum of 27 bits would be required to address 134217728 bytes of main memory. $Log_2(134217728) = 27$.
   b) (5) what is the minimum size in bits of each entry in the inverted page map table?
      The number of bits in the page number is 48-12 = 36.
      Each entry in the inverted page map table would contain a 36-bit page number, a valid bit, a dirty bit, and a 2-bit ID for a total of 40 bits.
   c) (5) how many entries would the inverted page map table contain if the memory size was changed so that the physical address was 30 bits wide?
      A 30-bit physical address would correspond to 2^{30} bytes of memory, so there would be $2^{30}/2^{12} = 2^{18}$ frames. Hence there would be 262144 entries in the inverted page table.

3. Two independent non-communicating processes are running concurrently on a virtual memory system with a single unified cache. The processes do not share any physical memory frames. The first process reads from some virtual address followed by the second process reading from a different virtual address and each read causes a cache hit.

a) (10) Could these two references cause that same line within the cache to be accessed if the cache is referenced using virtual addresses? Explain your answer.

Yes. With a virtual memory system two processes may refer to the same virtual address but each would have its virtual address mapped to a different physical address. Even if the virtual addresses are not the they could contain the same tag and set or line number fields, so they could both map to the same cache line. This would be the case if the two addresses only differed in the offset field. Additional information such as the process ID would have to appear in the cache to avoid this problem if virtual addresses are used to access the cache.

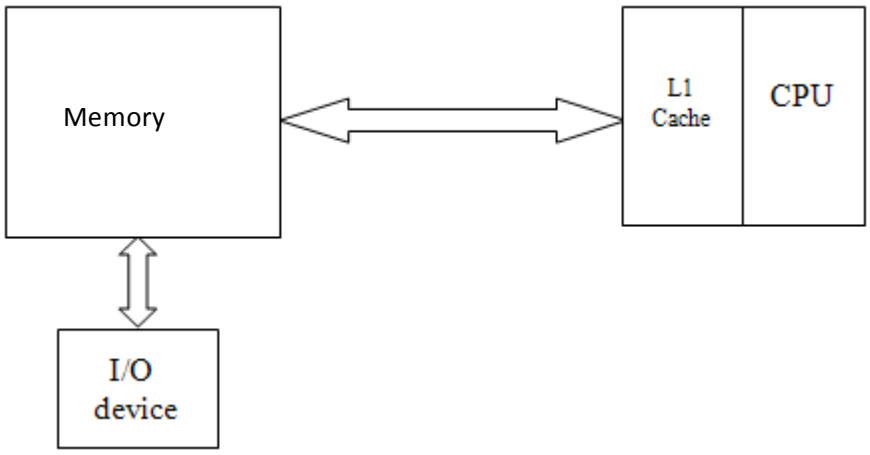
b) (10) Would your answer be the same if the cache had been referenced using physical addresses rather than virtual addresses? Explain.

No. The virtual addresses used by each process would have been translated into different and distinct physical addresses using separate page map tables. Since the two processes are independent and non-communicating, the OS would insure that the frames assigned to one process never match those assigned to the other. Therefore the two references could not map to the same cache line if physical addresses are used to access the cache.

4. (15) An executing program spends 65% of its time on disk I/O. If the disk system is replaced with an improved disk system that provides 50% greater throughput (i.e. the new disk system can perform the same operations as before but 50% faster), what overall speedup would be achieved for the program if the only change is to use the new and improved disk system?

If the disk system has 50% greater throughput, it will provide 1.5 times the throughput of the original system.
Based on Amdahl's law, if T is the total CPU time before the improvement then the time after the improvement is $T(0.35 + 0.65/1.5)$.
Hence the speedup = $T_{before}/T_{after} = T/\ T(0.35 + 0.65/1.5) = 1.28$

5. Suppose that our MIPS system employed a write allocate policy for its unified copy-back L1 cache. The system contains an I/O device with an integrated I/O controller that directly accesses memory without involving the cache as shown in the diagram below:



The following instructions are executed by the CPU on this system to store the 32-bit pattern 0xABCDEF12 into memory at the address 0x07E48080:

```
lui     $t0,0x07E4
ori     $t0,$t0,0x8080
lui     $t1,0xABCD
ori     $t1,$t1,0xEF12
sw      $t1,0($t0)
```

The memory employs little-endian storage order.
After these instructions have executed, the I/O device then writes a series of 4 bytes corresponding to 0x12, 0x34, 0x56, 0x78, in that order, into memory starting at address 0x07E48082. After the I/O device completes its writes, the CPU executes the following three instructions:

```
lui     $t0,0x07E4
ori     $t0,$t0,0x8080
lw      $t2,0($t0)
```

(10) a) Use 8 hex digits to show the 32-bit pattern loaded into $t2 by these instructions.
Due to the write-allocate policy, the write performed by the CPU will cause the block containing address 0x07E48080 to be updated and brought into the cache. Since the I/O write goes directly to memory bypassing the cache, the cache will not be updated to reflect the write performed by the I/O device. When the CPU again reads from address 0x07E48080, it will obtain from cache the same pattern that it had previously written. Therefore the contents of $t2 = 0xABCDEF12.

(10) b) If the memory had used big-endian storage order rather than little-endian storage order what would be the contents of $t2?

Using big-endian storage would affect the order of the bytes within a word in memory, however, when the word is read from memory or from cache the bytes are always placed into the register from high byte on the left to low byte on the right. Due to the cache hit, the CPU obtains the data from cache and $t2 would still contain 0xABCDEF12 the original value written by the CPU.

6. (10) The instruction sequence below runs on our non-pipelined multi-cycle datapath and employs polling to input a stream of characters coming from the keyboard attached to a USB interface that has a bandwidth of 115200 bytes per second (including any required overhead, such as parity, etc.). The CPU clock rate is a relatively slow 100 MHz. The hardware sets the ready bit within the control/status register each time a new character is received; the ready bit is automatically cleared to 0 when the input character is read by the instruction sequence. Delayed branching is not used since this is a non-pipelined system. No cache is used with this system.

```
        lui   $t3,0xFFFF      # point $t3 to the control/status register
loop:   lw    $t1,0($t3)      # read and check the ready bit
        andi  $t1,$t1,1       # mask off all but the ready bit
        beq   $t1,$0,loop     # if not set, continue checking
        nop
        lw    $t0,4($t3)      # read the input character
        beq   $0,$0,loop      # poll for the next character
```

While this code is running, a typist enters a stream of characters at the rate of 320 keystrokes per minute. On average, how many times will the ready bit be checked by this code between consecutive characters entered while the typist is typing? Assume that the beq instruction requires 3 cycles, the lw instruction requires 5 cycles and all other instructions require 4 cycles each.

The hardware is capable of transmitting 115200 characters per second, however the limiting factor is the typing speed. The time interval between keystrokes would be 60 seconds/320 = 0.1875 seconds. The cycle time is 1/100MHz = 10 ns. The time to execute the loop would be the sum of the execution time for the lw, andi & beq instructions = 5+4+3 = 12 cycles . Twelve cycles corresponds to 120ns. Hence the number of times the ready bit is checked between keystrokes = 0.1875/1.2e-7 = 1562500. Each time that the ready bit is found to be set, the final 3 instructions will also be executed. These account for 4+5+3 = 12 cycles or 120 ns. Hence after the first character arrives, the time interval during which the loop executes before the next character arrives is 187500000 – 120 = 187499880 ns. Therefore the loop will be executed 187499880/120 = 1562499 times between consecutive characters.