CSE 378 Winter 2011 Homework 4
Due: Saturday, 3/12 at 11 pm via Catalyst. No late days.

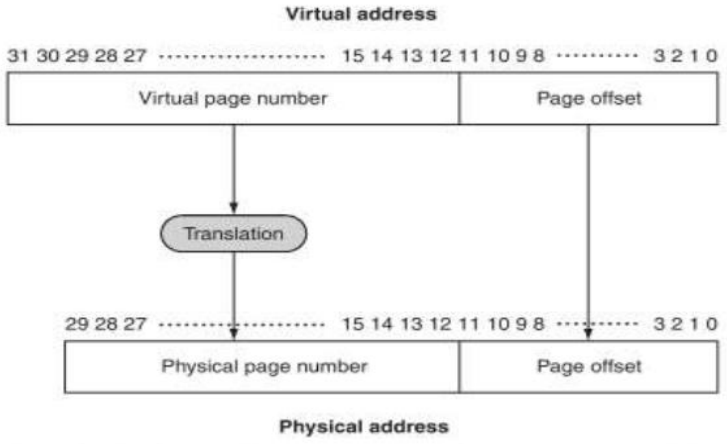

FIGURE 7.20 Mapping from a virtual to a physical address. The page size is 2^{12} = 4 KB. The number of physical pages allowed in memory is 2^{18}, since the physical page number has 18 bits in it. Thus, main memory can have at most 1 GB, while the virtual address space is 4 GB.

1. As we saw in class, page tables require fairly large amounts of memory, even when most of the entries are invalid. One way to reduce the memory footprint of page tables is to use a hierarchical page table. This way each process's first-level page table can easily fit in main memory and only necessary lower-level page tables need to be kept in memory. In such a system an address's virtual page number, as described in the figure above, is further divided into two fields: a "page table number" and a "page table offset". The page table number is used to index the first-level page table which provides an address for a second-level page table. The page table offset is then used to index the second-level page table to retrieve the physical page number which is then concatenated with the page offset to provide the physical address. One way to organize such a system is to have each second-level page table occupy exactly one page of memory. Assuming this design, a 32-bit virtual address space, a 16KB page size and 4 bytes per page table entry:

a. How many bits of the virtual address are used to index the first-level page table (the page table number)?

b. How many bits of the virtual address are used to index the second-level page table (the page table offset)?

c. How many first-level page tables can be stored in one page?

2. **Virtual Memory and TLBs**:

   a. Given the following stream of 32-bit virtual addresses, update the TLB and Page Table shown below. Assume 4 KB pages, a four-entry fully-associative TLB, and true LRU replacement where the fourth entry is the least recently used then the first, second then third (i.e. the third entry is the most recently used). If pages must be brought in from disk, increment to the next largest page number.

Addresses: 4095, 31272, 15789, 15000, 7193, 4096, 8912

TLB

Valid	Tag	Physical Page
1	11	12
1	7	4
1	3	6
0	4	9

Page Table

Valid	Physical page or in Disk
1	5
0	Disk
0	Disk
1	6
1	9
1	11
0	Disk
1	4
0	Disk
0	Disk
1	3
1	12
0	Disk
1	3
1	7
0	Disk

b. Repeat the question, but this time assume that pages are 16 KB (instead of 4 KB). Address sequence and tables are replicated below for your convenience.

Addresses: 4095, 31272, 15789, 15000, 7193, 4096, 8912

TLB

Valid	Tag	Physical Page
1	11	12
1	7	4
1	3	6
0	4	9

Page Table

Valid	Physical page or in Disk
1	5
0	Disk
0	Disk
1	6
1	9
1	11
0	Disk
1	4
0	Disk
0	Disk
1	3
1	12
0	Disk
1	3
1	7
0	Disk

c. What would be an advantage of having a larger page size? What would be a disadvantage, if any, of a larger page size?

3. **I/O:** Given the following two applications 1) A database management system for storing large multimedia files and 2) Starcraft 2 (a computer video game), answer the following questions about hard disk design.

   a. Is decreasing the sector size likely to help or hurt this application? Why?

   b. Would increasing disk rotation speed help or hurt this application? Why?

4. **Performance**: Our two favorite programs have the following distribution of instructions of type A, B, C, and D, and our current processor has the following CPI for those instructions:

	A	B	C	D
Fraction of program 1	20%	15%	55%	10%
Fraction of program 2	30%	30%	25%	15%
CPI	5	4	3	2

   a. We have heard there is a new processor that implements Type B instructions blazingly fast – each one now executes in only one cycle. What is maximum performance improvement we might expect from this new processor for program 1 and program 2?

   b. What would be our performance improvement if instead we just made our original processor run twice as fast? For program 1? For program 2?

5. **Cache Performance.** Bo-Lu Screen, a 378 student, was recently challenged to a coding competition by his friend Paige Fault. Each student wrote a function in C to convert an in-memory representation of a color image to grayscale. Whoever's code executed more quickly would be declared winner.

6.

Bo-Lu Screen's code:

```
unsigned byte[ ][ ] convToGray( unsigned byte img[ ][ ][ ], int width, int height,
                                        unsigned byte grayImg[ ][ ] ) {
        int x, y;
        unsigned byte R, G, B;

        for ( y = 0 ; y < height ; y++ ) {
                for ( x = 0 ; x < width ; x++ ) {
                        R = img[ y ][ x ][ 0 ];
                        G = img[ y ][ x ][ 1 ];
                        B = img[ y ][ x ][ 2 ];
                        grayImg[ y ][ x ] = 0.6*R + 0.3*G + 0.1*B;
                }
        }
        return grayImg;
}
```

Paige Fault's code:

```
unsigned byte[ ][ ] convToGray( unsigned byte img[ ][ ][ ], int width, int height,
                                        unsigned byte grayImg[ ][ ] ) {
        int x, y;
        unsigned byte R, G, B;

        for ( x = 0 ; x < width ; x++ ) {
                for ( y = 0 ; y < height ; y++ ) {
                        R = img[ y ][ x ][ 0 ];
                        G = img[ y ][ x ][ 1 ];
                        B = img[ y ][ x ][ 2 ];
                        grayImg[ y ][ x ] = 0.6*R + 0.3*G + 0.1*B;
                }
        }
        return grayImg;
}
```

Surprisingly, their code barely differs. However, Bo-Lu's code significantly outperformed Paige's on the same machine. Why did Bo-Lu's code execute faster than Paige's?