DIY Address Translation

This exercise lets you try out address translation in hierarchical paging. To keep things manageable, we have to use a tiny logical and physical address space, a little bit larger than the example we worked through in class. This example will let us see some of the same things that go on with a real paged memory system.

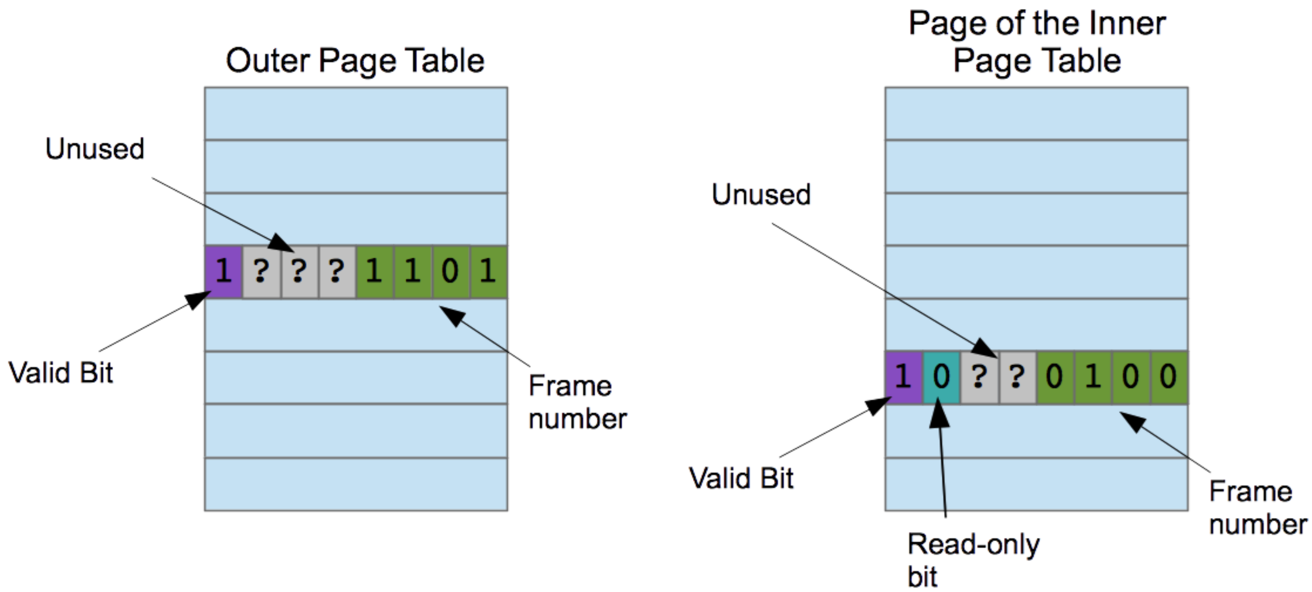
On the course homepage in Moodle, you'll find a file, **memory.txt**. Part of this file is shown below. This file gives the contents of physical memory, one byte per line. Physical addresses are 7 bits, so each byte of memory is shown (in binary) with its 7-bit physical address to the left (also given in binary). Memory is broken into frames of 8 bytes each.

```
Top-Level Page Table in Frame Ten.

    physical       memory   frame
     address     contents   number
    (binary)     (binary)
-------------------------------------
     0000000     01111000   frame 0000 (0)
     0000001     01100000
     0000010     11101010
     0000011     00010011
     0000100     01101101
     0000101     01000011
     0000110     10100010
     0000111     10010100
-------------------------------------
     0001000     00111100   frame 0001 (1)
     0001001     10011000
     0001010     00111111
     0001011     10100010
     0001100     00100001
     0001101     10110111
     0001110     10101110
     0001111     10111101
-------------------------------------
     0010000     00011001   frame 0010 (2)
     0010001     11001110
     0010010     01010100
     0010011     01010101
     0010100     11011011
     0010101     01110010
     0010110     00111010
     0010111     11111010
-------------------------------------
     0011000     10000111   frame 0011 (3)
     0011001     11000011
     0011010     10011000
     0011011     01010111
     0011100     11101010
     0011101     10000011
     0011110     11000110
     0011111     01100010
```

On this computer, pretend a logical address is 9 bits. The system uses two-level paging. The outer page table fills one frame, with each entry stored in one byte (so, the outer page table has 8 entries). Each page of the inner page table occupies one frame. The inner page table has entries that are just one byte, so each page of the inner page table also contains 8 entries.

The following figure illustrates the format of the outer and inner page tables. For the outer table, the figure shows what entry 3 might contain. A one in the high-order bit indicates that this entry is valid, the process may access pages reached via this entry. The next three bits are unused, and the low-order four bits give the frame where the corresponding page of the inner page table resides. Here, for example, page 3 of the inner page table is valid, and it resides in frame 13 (decimal for 1101).



A page of the inner page table has the format illustrated on the right. Each entry is stored in a byte. The high-order bit of the entry is the valid bit, and the next bit is the read-only bit (one for read-only, zero for writable). The next two bits are unused, and the four low-order bits give the memory frame where the corresponding page of the process resides. Let's say this is an illustration of page 3 of the page table. Then, the line shown in this figure would tell us about page 29 of the process (the first page of the page table tells us abut pages 0 .. 7 in the process, the next page of the page table tells us about 8 .. 15, then next 16 .. 23, so this page tells us about pages 24 .. 31 of the process). According to the figure, page 29 of the process is valid, it's not read only and it resides in frame 4 (decimal for 0100).

You'll complete this exercise by completing the Moodle online quiz named Exercise 12 Quiz. The file, memory.txt, shows the contents of memory you'll be using. In this file, the outer page table is in frame ten.

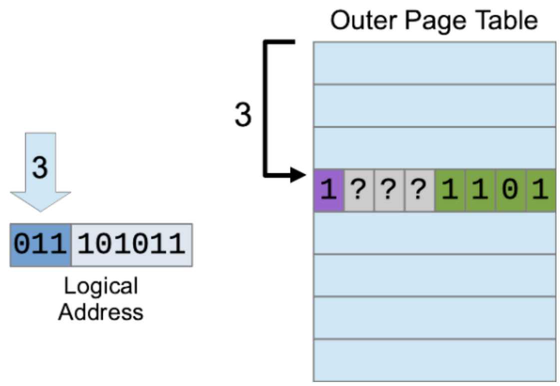
Your job is to report what will happen during address translation for a collection of memory references. Each memory reference is either a read or a write, followed by a 9-bit logical address. For example, given the following memory reference:

- write 011101011

You would report that this translates to physical address 0100011.

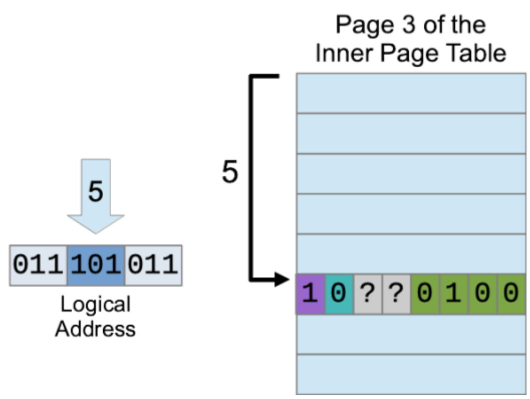
Why does this address translate to 0100011? Remember, in hierarchical paging, you'll be breaking the logical address into multiple fields. In this architecture, the page table is broken into eight (page-sized) pieces, so the outer page table contains eight entries. The high-order three bits in the address tell you what part of the page table you need.

The following figure shows part of the outer page table. For logical address 011101011, the high-order three bits have a value of 3. Line 3 of the outer page table says this part of the page table is valid, and it's in frame 13 of memory.



The following shows part of page 3 of the (inner) page table, from frame 13 of memory. It has 8 entries, which makes sense as there are 3 more bits remaining in the page number. The next three bits from the logical address say that we want entry 5 from this part of the page table. Looking at entry 3, we see that this page of the process is valid and it's not read only. It's in frame 4 of memory.

5



Using the frame number (0100) as the high-order bits and the remaining 3 bits of the logical address (011) as the offset, we get a physical address of 0100011.

Some of the given memory accesses you have to translate may not be permitted. The following two examples are both bad. One is an attempt to access an invalid page, and the other is an attempt to write to a read-only page. For cases like these, you'll just answer with an X.

- read 111000111
- write 101111010