Use unsigned int for representing addresses and masks

The virtual address space is with 32 bits, each address has 32 bits. The size of the **unsigned int** type in c/c++ is 32 bits in both the 32-bit and 64-bit systems, using **unsigned** because an address cannot be negative.

Reading trace file

File tracereader.c can be compiled into your program like any other file in a multifile compilation. You will need to call NextAddress to get the next address. Please note that it takes a FILE pointer (output of fopen), and a *pointer* to an object of type p2AddrTr. This means that you need to allocate an object of that type. Please note that we are using the term object loosely here, this is not a C++ object, but rather a structure that is typedefed in the tracereader.h file. It has several fields, but the most important one is addr which will contain the logical/virtual address. The return code lets you know if an address was found or not.

Read the tracerreader.c code for further details. You only **need to call the NextAddress** function, see below for a quick example:

p2AddrTr mtrace;

```
unsigned int vAddr;

if(NextAddress(tracef_h, &mtrace)) //tracef_h - file handle from fopen
{
  vAddr = mtrace.addr;
}
```

Bit masking and shifting

Refer to ***bitmasking-demo.c*** for a demo of bit masking and shifting

Logical/virtual to physical address translation

Let us look at an example where we translate 0041f760 -> 0001f760 with a single level page table (10 bits for page index) and 4 MB (2^22) page size.

0041f760 can be rewritten in binary as follows:

0000 0000 0100 0001 1111 0111 0110 0000

We had 10 bits of page index, so we mask as follows:

0000 0000 0100 0001 1111 0111 0110 0000

1111 1111 1100 0000 0000 0000 0000 0000 (mask) bit-wise and

0000 0000 0100 0000 0000 0000 0000 0000

We can compute the offset by constructing a 22 bit mask of 1s, and do the same kind of bit-wise and operation (above) to get 0x1f760 for the offset.

To get the logical/virtual page number, we right shift the above value by 22 bits, we are left with 0000 0000 01. Writing this in groups of four for easy translation to hex, 00 0000 0001, we see that this is logical page 1. Assuming logical page 1 is mapped to physical frame 0, so we take 0 * page size and add the offset of 1f760. Since 0 times anything is still zero, the physical address would be 0 + 1f760, or 17f60.

Map type in the page table data structure

First, refer to ***pageTableDS.pdf***.

Specifically, the Map essentially is the representation of a page table entry for mapping a logical/virtual page number to a physical frame page number. At a minimum, it is a frame index, but you might choose to include a valid bit like a real page table would. If you do not, you need another way of determining whether or not a frame has been allocated to a page. The last level of your page table will point to an instance of type Map.

What is a frame? A frame is a block of memory that is the same size as a logical/virtual page. When you access a logical/virtual page, we first find what frame it is associated with and then find how many bytes we are into the frame (offset). See the question about logical to physical address translation above.

Process arbitrary number of mandatory command line arguments (for processing numbers of bits for levels)

See the yellow highlighted portion below.

```
#include <unistd.h>

int main(int argc, char **argv) {

  // other stuff (e.g. declarations)

  /*
   * This example uses the standard C library getopt (man -s3 getopt)
   * It relies on several external variables that are defined when
   * the getopt.h routine is included.  On POSIX2 compliant machines
   * <getopt.h> is included in <unistd.h> and only the unix standard
   * header need be included.
   */
  while ( (Option = getopt(argc, argv, "n:o:")) != -1) {
    switch (Option) {
    case 'n': /* Number of addresses to process */
      // optarg will contain the string following -n
```

```
      // Process appropriately (e.g. convert to integer atoi(optarg))

      break;
    case 'o':     /* produce map of pages */
      // optarg contains the output method…
      break;
    default:
      // print something about the usage and die…
      exit(BADFLAG);  // BADFLAG is an error # defined in a header
    }
}

/* first mandatory argument, optind is defined by getopt */
idx = optind;

/* argv[idx] now has the first mandatory argument: the trace file
    path*/
 /* argv[idx+1] through argv[argc-1] would be the arguments for
    specifying the number of bits for each page table level, one number
    per each level */
 if (idx < argc)
 {
  …….

 }
```

Debugging tip

If you are using gdb, you can print out a number in hexadecimal with p/x; e.g. p/x logical_addr if logical_addr contains a value you want to inspect. I find it much easier to think about bit patterns in hexadecimal, base 10 is a nightmare when you are trying to think about bit positions. Remember printf and cout also have methods to print hex as well.