/* HPC_HW_2015_09_09
 * Author: Chen Fang

Problem 2.1

a)
There are 7 steps to complete one floating point addition.
If fetch and loads takes 2 nanoseconds each, then the total time:
t = 2 + 2 + 1*(rest 5 steps) = 9 nanoseconds

b)
Without pipelining, 1000 pairs of additions take up to:
Time = 1000*t = 1000*9 = 9000 nanoseconds

c)
With pipelining, the fetching for the next pair can be performed immediately
after the fetching for the current pair is completed, which takes 2 nanoseconds. So,
Time = t + (1000-1)*2 = 9+2000-2 = 2007 nanoseconds

d)
A cache miss on a fetch causes the pipelining to stall.
If there is a level 1 cache miss, level 2 cache is checked to see if data exists.
If that's a miss again, a fetch from the main memory is performed.

Problem 2.3

Generally speaking, a larger matrix tends to increase the numbers of cache misses,
while increase a larger cache tends to reduce cache misses.

Given MAX=8 and number of cache lines = 4, and assuming the cache line size = 4,
which remains unchanged compared with the example in the book:
For the first nested loop, there are two cache misses when each row is fetched.
As a row will not be revisited later, the total number of cache misses is 2*8=16.
For the second nested loop, there is one cache miss for each element
even though the number of cache lines increases, because old data are evicted
when cache lines are fully occupied. So the total number of cache missis is 8*8=64.

Problem 2.4

Since the lower 12 bits are used to locate a byte in a page, while the
upper 20 bits are used to identify the page number, the total number of pages can
reach up to 2^20=1,048,576.

Problem 2.6

It takes 10 cycles to load 64 bits, approximately 6.4bit/cycle.
So about 10 banks are needed to stream 64 bits in one go.

Problem 2.7

There are two sentences in the loop:

```
y[i] += a*x[i]
sum  += z[i]*z[i];
```

For CPU, the vector processor performs the first sentence in a vector fashion.
Several elements will be computed at the same time. The second sentence, however,
will be performed with the instruction that reduces the number of loads and stores.

For GPU, the first sentence will be executed in the same way as vector processor
except that there are much more working computational units at the same time.
Unfortunately, without special treatment, the second reduction process will be
processed in a serial way during which most of the units are idle and wasted.

```
*/
```