Assignment 4

All problems are due in the dropbox on mycourses by Thursday 05/03/12 @ 10pm.

1) Problem 4.9

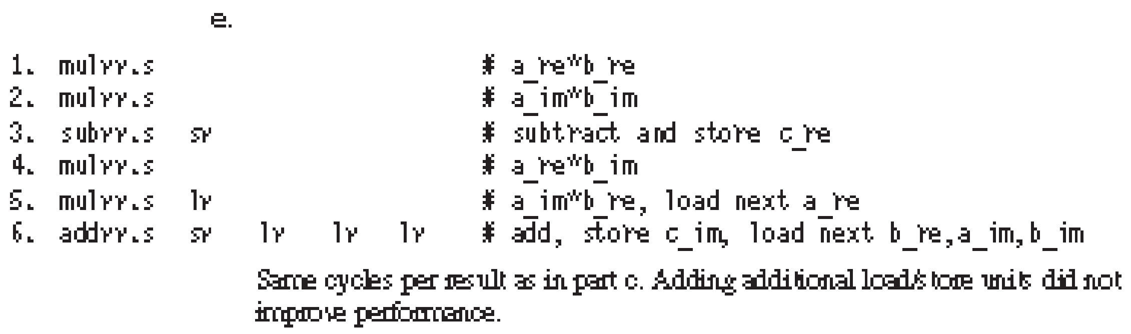
b. Assume MVL = 64:

```
        li          $VL,44        # perform the first 44 ops
        li          $r1,0         # initialize index
loop:   lv          $v1,a_re+$r1  # load a_re
        lv          $v3,b_re+$r1  # load b_re
        mulvv.s     $v5,$v1,$v3   # a+re*b_re
        lv          $v2,a_im+$r1  # load a_im
        lv          $v4,b_im+$r1  # load b_im
        mulvv.s     $v6,$v2,$v4   # a+im*b_im
        subvv.s     $v5,$v5,$v6   # a+re*b_re - a+im*b_im
        sv          $v5,c_re+$r1  # store c_re
        mulvv.s     $v5,$v1,$v4   # a+re*b_im
        mulvv.s     $v6,$v2,$v3   # a+im*b_re
        addvv.s     $v5,$v5,$v6   # a+re*b_im + a+im*b_re
        sv          $v5,c_im+$r1  # store c_im
        bne         $r1,0,else    # check if first iteration
        addi        $r1,$r1,#44   # first iteration,
                                  increment by 44
        j loop                    # guaranteed next iteration
else:   addi        $r1,$r1,#256  # not first iteration,
                                  increment by 256
skip:   blt         $r1,1200,loop # next iteration?
```

c.

```
1.    mulvv.s     lv        # a_re * b_re (assume already
                            # loaded), load a_im
2.    lv          mulvv.s   # load b_im, a_im*b_im
3.    subvv.s     sv        # subtract and store c_re
4.    mulvv.s     lv        # a_re*b_im, load next a_re vector
5.    mulvv.s     lv        # a_im*b_re, load next b_re vector
6.    addvv.s     sv        # add and store c_im
```

6 chimes


e.

```
1.  mulvv.s                        # a_re*b_re
2.  mulvv.s                        # a_im*b_im
3.  subvv.s   sv                   # subtract and store c_re
4.  mulvv.s                        # a_re*b_im
5.  mulvv.s   lv                   # a_im*b_re, load next a_re
6.  addvv.s   sv   lv   lv   lv    # add, store c_im, load next b_re,a_im,b_im
```

Same cycles per result as in part c. Adding additional load/store units did not improve performance.

Solve 4.9 a (5p) and d (5p).

2) Solve 4.13 a (5p) and b (10p).

3) Solve 4.14 a (5p), b (5p), and c (5p).

4) (10p) Read the paper posted on mycourses, and submit a ~500 words or less (~<1page) summary. Address the following:
   a. What is the paper talking about
   b. What did you find out new that you didn't know before
   c. What you agree and/or disagree

5) Project 4 (40p):

   a. This project assumes that you have a working CPU, with the specifications of the first three iterations of the project. If your current CPU does not meet all previous specifications, try to fix them and report them in this iteration of the project. **Save your current processor version for future use!**

   b. Keep your instruction memory as is. Change your data memory to a hierarchical organization, i.e. a data cache and data memory. The cache will have the following characteristics:

      i. DCache_Size = 32 Bytes. DM_Size = 256 Bytes.

      ii. The DCache is 2-way set associative.

      iii. There are 4 Bytes/Block. Hence a total of 256/4=64 blocks in DM.

      iv. Therefore, the DCache can store 8 blocks, or 4 groups of two blocks each (in the cache).

      v. If there are 4 groups of blocks, there will be 64/4=16 blocks/group in DM.

      vi. The 8-bit address is divided as follows:

         1. A7|A6 = Group ID

2. A5→A2 = Block ID

3. A1|A0 = Bytes within the block

vii. Block replacement strategy: replace not last used block (1-bit).

viii. Read hit – 1 clock cycle.

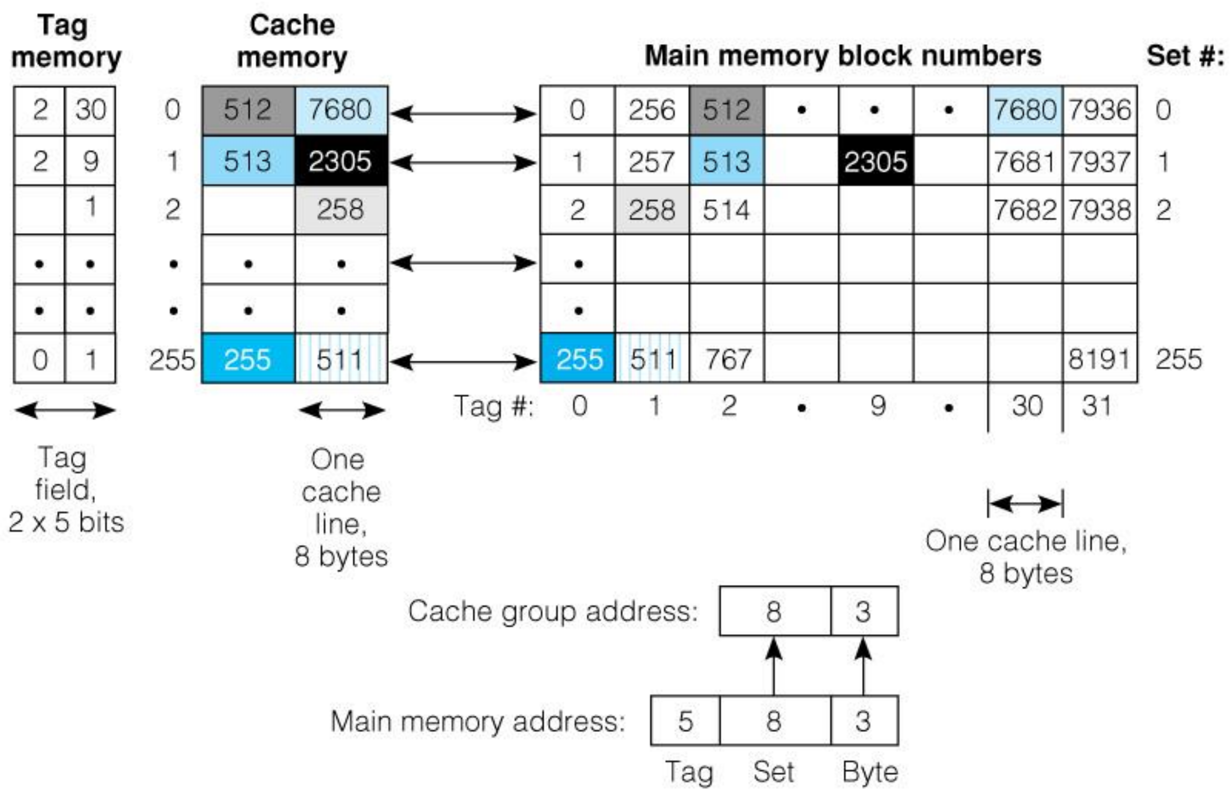
ix. Write hit – your choice, i.e. either write-through or write-back.

x. Read miss – bring block into cache first, and then provide required byte; transfer 1 byte/cycle ➔ 4 cycles/block; this accounts for the real latency and burst combination.

xi. Write miss – same as on a read miss.

c. You will need to design a Data-Cache-Memory Controller, separate from your current CPU. On a miss, either read or write, the pipeline will be stalled until the requested location (byte) is available for access in the cache. Thus, your current processor should not require any modifications, except for the stall mechanism in the case of a miss.

d. Below is a block diagram example of the necessary hardware:



e. Assume the following array of 16 elements (all values are in decimal: 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47; these are the first 16 prime numbers.
    i. Convert the following code to assembly:
```
for (i=16; i>0; i=i–1)
        x[i] = x[i] * 53;
for (i=16; i>0; i=i–1)
        x[i] = 53 / x[i];
```
    ii. Assemble the code, run and confirm that your processor is working properly. You don't need to collect any performance information at this time.

f. Along with the relevant design files, submit a report in which you describe your design choices and operation of your hierarchical data memory.