University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Spring 1999 John Kubiatowicz

Homework Quiz (HW #2)
Solutions

February 16, 1999
CS152 Computer Architecture and Engineering

This quiz covers one of the problems from homework #2.
Good Luck!

Your Name:	
SID Number:	
Discussion Section:	

Total:	

A better multiply for the Single Instruction Computer

In the homework, you were introduced to the "Subtract and branch if negative" instruction:

```
sbn a, b, c      # Mem[a] = Mem[a] – Mem[b]; If Mem[a] < 0 goto c;
```

Problem 3.30 asked you to develop a multiply routine that was not particularly efficient. Consider the following alternative for a 16-bit computer. This performs c = a * b, modifying b in the process:

```
c = 0;
for (index = 32768; index >= 1; index = index / 2) {
     If (b >= index) { b = b – index; c = c - index * a; }
}
c = -c;
```

To implement this, we will assume a macro facility that lets us generate groups of instructions many times.

1. Create a macro called "`inner`" which performs the inner instructions of our multiply. This takes 4 memory locations as input: `a, b, index,` and `scaled_a`, and computes:

   ```
   if (b >= index) {b = b – index; c = c – scaled_a;}
   ```

   Assume one temporary memory location (`temp`). Note that your code can contain an empty label called "exit:" at the end.

   ```
   inner(b, c, index, scaled_a, temp):
           sbn   b, index, nogo        # do test (b < index)
           sbn   c, scaled_a, exit     # b >= index. Fix C
     nogo: sbn   temp,temp,.+1         # Create
           sbn   temp,index,.+1        #    -index
           sbn   b,temp,.+1            # Fix up B (undo test)
     exit:
   ```

2. Create a macro called "`double`" which puts (2*Mem[in]) ⇒ Mem[out], using Mem[temp] as a temporary location. This must not alter the "in" memory location:

   ```
   double(out, in, temp):
           sbn   temp,temp,.+1         # temp = 0
           sbn   temp,in,.+1           # temp = -in
           sbn   temp,in,.+1           # temp = -2in
           sbn   out,out,.+1           # out = 0
           sbn   out,temp,.+1          # out = 2in
   ```

3. Using your macros, write a routine "multiply" that will take a and b as input, and place the result in c. ***Make this work for 4 bit variables***. Assume that the memory location "`one`" contains the constant 1. Also, assume that you can use your macros like instructions:

   For instance:
   ```
   double(two,one,mytemp)
   inner(q,r,other,two,anothertemp)
   ```

   The macro facility expands each macro call into the set of instructions in the macro definition, while substituting the actual arguments for formal ones. Thus, for instance, the first call to "double" above will substitute `two`⇒`out`,`one`⇒`in`, and `mytemp`⇒`temp` in your definition of double.

   *Hint: you must first create a bunch of constants like 2 and (2 * a), then use them.*

   Multiply_4_bit:
   ```
      double(two,one,temp)              # Create const 2
      double(four,two,temp)             # Create const 4
      double(eight,four,temp)           # Create const 8
      double(two_a,a,temp)              # Create const 2 * a
      double(four_a,two_a,temp)         # Create const 4 * a
      double(eight_a,four_a,temp)       # Create const 8 * a
      sbn   neg_c,neg_c,.+1             # neg_c = 0
      inner(b,neg_c,eight,eight_a,temp)# Deal with bit 8
      inner(b,neg_c,four,four_a,temp)  # Deal with bit 4
      inner(b,neg_c,two,two_a,temp)    # Deal with bit 2
      inner(b,neg_c,one,one_a,temp)    # Deal with bit 1
      sbn   c,c,.+1                    # c = 0
      sbn   c,neg_c,.+1                # c = -neg_c
   done:
   ```

4. If you were to make a 16-bit version of this, what is the worst-case number of cycles that it would take? Is this much better than the multiply from the homework?

   Each double takes 5 cycles. The maximum number of cycles for an inner operation is 4 cycles (the negative path). A 16 bit version would have 15 * 2 * 5 cycles from `double` ops, 16 * 4 cycles from `inner` ops and 3 cycles of random sbn instructions. So, 150+64+3 = 217 cycles.

   This is **much** better than the solution in your homework (which had a loop of multiple instructions that could have up to 65535 iterations).