Project 3: Long Multiplication

Due: Thursday October 10, 2002

Objective

The objective of this project is to gain experience writing more complex assembly language programs.

Assignment

Write an assembly language program that takes two arbitrarily long strings from the user. These strings should be interpreted as unsigned hexadecimal numbers. Your program must then multiply these two numbers and output the product as a hexadecimal number. The algorithm you should follow to do this multiplication is the same long multiplication algorithm you learned in elementary school.

Implementation Issues:

1. You will find the code to read in and convert two arbitrarily long hexadecimal numbers in:

   `/afs/umbc.edu/users/c/h/chang/pub/cs313/readhex.asm`

   There's also code for printing out long binary numbers.
2. Although your code must essentially work for input numbers of any size, you are allowed to limit the user's inputs to 64 digit hexadecimal numbers. This is so we do not have to deal with memory allocation issues, but nevertheless your program should be able to handle more digits by changing the definition of a single constant.
3. It is OK to multiply the numbers byte by byte instead of 32-bit word by 32-bit word.
4. Numbers must be stored in little-endian format.
5. You will find the ADC (add with carry) instruction useful.
6. Remember that IMUL is for signed numbers, so you must use MUL instead of IMUL.
7. The syntax for a MUL instruction to multiply the AL register with a memory location is something like:

   ```
   MUL  byte [esi]
   ```

   The AL register is implicit and must not be specified.
8. You may assume that multiplying an n-byte number with an m-byte number results in an (n+m)-byte product.
9. You will need to carefully plan what information is stored in which registers and what information is stored in memory.
10. Complex addressing modes such as `[esi + ecx]` will come in handy.
11. You can check your result using `dc` the "desktop calculator". Type in `16 o 16 i` to put the program in hexadecimal mode. Note that `dc` uses Reverse Polish Notation, so adding 5 to 3 is accomplished by `5␣3␣+␣p`. The final `p` prints out the result to the screen. Type `man dc` to obtain full documentation on `dc`.

Turning in your program

Use the UNIX script command to record some sample runs of your program. Your sample runs should thoroughly exercise your program using inputs that are long enough and complex enough to demonstrate that your program handles multiple word multiplication correctly.

You should submit two files: 1) your assembly language program and 2) a typescript file of your sample runs. The class name for submit is 'cs313-0101' and the assignment name is 'proj3'. The UNIX command to do this should look something like:

```
submit cs313-0101 proj3 multiply.asm typescript
```