CmpE230 Fall '19
Homework 1

Selman Berk Özkurt, Emre Hoşer
2017402195, 2016400366

November 10th, 2019

1 Introduction

In this project, we will implement a multiplier that takes an octal multiplication operation directive from `stdin` and prints the output as an octal number to a new line to `stdout` using A86 assembler.

Our source code comprises a single `.asm` file that is organized as macros corresponding to different operation blocks performed. We chose this technique to enhance modular development and testing paradigm use. Most macros are called only once. This documentation is also organized in terms of these modules of the code.

2 Code

The program first takes the arguments into two memory words each, `VARa:VARb` for the first and `VARc:VARd` for the second. Later, we do the intermediate multiplications and add all the results to a `qword VARsum`. Lastly, we print the value in `VARsum` in octal format.

We initially wrote 0 to all memory slots and registers to clean garbage. They did interfere with the operation if we disabled doing this.

2.1 Formatted input

Later we begin taking input. We paste the modules getcstar and `getcnull` which take the characters from stdin, find the binary value of the ASCII characters and shift it into cx:bx[1] at the least significant side until reading the asterisk and new line characters, respectively. After executing those modules, what we have is data written in cx:bx which we then transfer into their logical locations VARa:VARb and VARc:VARd.

[1] which had been initialized to 0

2.2 Processing

We have to do a set of normal 16-bit multiplications and add the results as `qwords`. We multiply both 16-bit fields of both operands using native multiplication instruction and will have to make a long addition that will be discussed. The overall operation can be viewed as follows:

$$\begin{array}{cccc}
 & & \texttt{VARd*VARb}_{\text{high}} & \texttt{VARd*VARb}_{\text{low}} \\
 & \texttt{VARa*VARd}_{\text{high}} & \texttt{VARa*VARd}_{\text{low}} & \\
 & \texttt{VARb*VARc}_{\text{high}} & \texttt{VARb*VARc}_{\text{low}} & \\
+\ \texttt{VARb*VARd}_{\text{high}} & \texttt{VARb*VARd}_{\text{low}} & & \\
\hline
\texttt{VARsum[3]} & \texttt{VARsum[2]} & \texttt{VARsum[1]} & \texttt{VARsum[0]}
\end{array}$$

In order to do 64-bit additions we defined two `qword` variables, namely `VARsum` and `VARacc`. We put the operands to be added into these and run what we defined inside `summer` macro, which stores the result in `VARsum`.

2.2.1 `summer`

In order to reduce the variable and code use, we do all the binary 64-bit additions between `VARacc` and `VARsum` as discussed. One thing to consider is that individual 16-bit additions need their carry values to be preserved to actually accomplish the long addition. To do that, we added the least significant words using regular `add` and subsequent words using `adc` which adds the carry flag's value to the result. We used the same registers for all 4 additions, thereby fetching and storing the operands before and after each addition.

2.3 Formatted output

After the operations, what we have is the 64-bit result in binary format stored in the `VARsum` variable. In order to print this with the most significant part first, and using only the necessary number of octal numbers, we first separate the result into 3-bit parts beginning from the least significant by shifting the whole `qword`[2] until the remaining `sum` is zero[3]. We push these octal values into the stack frame that we manually create by pushing `bp` and moving `sp` into `bp`.

Then after processing the `sum` is complete, we pop these octal values and print them to `stdout` upon formating as ASCII characters. We use the FIFO property of the stack because we can determine the octal values lsb first, while we should print them in the reverse order.

At the end, we exit the process by making a system call.

[2]using `SHIFTSUM` component

[3]we check using `ZEROSUM` module