Assembly language programming using modules

Ken Clowes (kclowes@ee.ryerson.ca)

November 22, 2000

Contents

1 Introduction

This tutorial demonstrates how to use, modify and create *modules* in assembly language programming. The assembly language used is the DECUS variant of the 6811 assembly language used in ELE538. The reader should also have some familiarity with the assembly language coding standards[Clo] that are used here.

Before beginning, we briefly define what we mean by a *module*:

> A module is object code and documentation. The object code provides useful subroutines that can be used by programmers to simplify their own programs. The way the module and its routines are implemented is of no concern to the user of the module. (Indeed, the implementation language need not even be assembler.) All the user needs is clear documentation about the contents of the module. The routines can then be used and executable code produced by linking the application program with the module(s) used.

You can obtain the source code and try out the examples here by copying the file `~kclowes/public/modEx.tgz` into a directory and unpacking the files with the command:

```
zcat modEx.tgz | tar xvf -
```

2 A simple example

As a simple example, consider the first program that should be written in any language—getting the computer to say "Hello world".

If there are no existing subroutines or modules, this is a complex process at the assembly language level. The programmer would have to know the details of the serial communication device being used, test the status register and wait until it was empty before sending each character, loop through the characters in the string and so forth.

However, if there is an existing subroutine called `putstr` that sends each character in a null-terminated string whose starting address is specified by register IX to the serial port, the program becomes trivial as illustrated by the following code:

```
        .area _CODE
main::
        ldx #hi         ; IX = address of "Hello world!" string
        jsr putstr
        swi

        .area DATA
hi:     .asciz "Hello world!"
```

Using a module (i.e. object code) instead of cutting and pasting the source code for an existing subroutine offers the advantages of encapsulation and information hiding.

2.1 Exercise

To ensure that this works, type `make hello` in the directory where you unpacked the examples. You can then download the `hello.s19` file to the 6811 and run it with the Buffalo monitor command `go 6200`.

Edit the source code (file: `hello.asm`) to (for example) change the string or print it out twice and check that your version of “hello” works.

2.2 Some nitty gritty details

Notice that the assembly code is divided into two *areas* (with the `.area` directives.) Data goes into the eponymously named segment and the actual instructions go into the `_CODE` segment. The starting addresses of these segments are assigned at link time (by default, we start the `DATA` segment at 0x6000 and the `_CODE` segment at 0x6200). We suggest you follow the same conventions. The concepts of segments or areas and the linking process are described in greater detail in Appendix A.

2.3 What modules and subroutines are available?

Look at the on-line documentation. on-line documentation on the ELE 538 home page.

3 A second example (myName.asm)

In this example, we print a greeting prompting the user for their name. We then calculate the number of characters in the name and print a reply. The following modules and subroutines are used:

stdio module Uses:

- `putstr`: To output a null-terminated string.
- `getline`: Reads a line (terminated by hitting return or enter).

stdlib Uses:

- `atoi_u16`: Converts a 16-bit unsigned integer to a string of ascii digits of the number's decimal representation.

strings Uses:

- `strlen`: Determines the number of characters (length) of a string.

```
                .area DATA
greeting:       .asciz "Enter your name: "
replyStart:     .ascii "Hi "
me:             .ds 60
replyMid:       .ascii ". (Your name has "
len:            .ds 6
replyEnd:       .asciz " characters in it.)\n"
        ; Actual code
        .area _CODE
main::
        ldx #greeting
        jsr putstr
        ldx #me
        jsr getline
        jsr strlen
        clra
        ldx #len
        jsr itoa_u16
        ldx #replyStart
        jsr putstr
        ldx #replyMid
        jsr putstr
        ldx #replyEnd
        jsr putstr
        swi
```

3.1 Exercise

Write a program that prompts the user to enter two numbers (one per line), calculate the sum and print the result. Your program should work for both positive and negative numbers.

4 Modifying a module

5 Creating a module

A Object code and linking

```
   ;file foo.asm
        .area FOO
        .asciz "FOO ONE"

        .area BAR
        .asciz "BAR ONE"

      ;file bar.asm
        .area FOO
        .asciz "FOO TWO"

        .area BAR
        .asciz "BAR TWO"
```

```
/usr/local/gnu6811/bin/as6811 foo.asm
/usr/local/gnu6811/bin/as6811 bar.asm

/usr/local/gnu6811/bin/aslink -bFOO=0x6000 -bBAR=0x7000 -o foobar foo.o bar.o

/usr/local/gnu6811/bin/aslink -bFOO=0x6000 -bBAR=0x7000 -o barfoo bar.o foo.o



vt6811 foobar.s19
```

```
>md 6000 600f

6000 46 4F 4F 20 4F 4E 45 00 46 4F 4F 20 54 57 4F 00 FOO ONE FOO TWO
>

vt6811 barfoo.s19
>md 6000 600F

6000 46 4F 4F 20 54 57 4F 00 46 4F 4F 20 4F 4E 45 00 FOO TWO FOO ONE
>
md 7000 700F

7000 42 41 52 20 54 57 4F 00 42 41 52 20 4F 4E 45 00 BAR TWO BAR ONE
>
```

References

[Clo] Ken Clowes. *Assembly Language Coding Standards.* `file:~kclowes/public/CodingStds/CodingStdAsm.ps`