EE 308 – LAB 1

ASSEMBLER, SIMULATOR, AND MONITOR

Introduction and Objectives

This laboratory introduces you to the following 68HC12 assembly language programming tools:

- The 68HC12 assembler `CA6812`. This runs on the PC.
- The 68HC12 simulator `ZAP`. This also runs on the PC.
- The `D-Bug 12` monitor that runs on the 68HC12.

An assembler takes assembly language code in a form that a human can reasonably read with a little practice, translates it to machine code which a microprocessor can understand. and stores them in a `.s19` file which the 68HC12 monitor program can understand. In this lab we will use the Cosmic Software CA6812 assembler, which is on the PCs in the Digital/Microcontrollers lab. This is copyrighted software which is licensed for use on our lab computers only. There is a demo version of this assembler (as well as the Cosmic C compiler and the ZAP simulator) are available from Cosmic at `http://www.cosmic-us.com/hc12_des.html`. It should be adequate for most of the labs in this course.

The Cosmic CA6812 assembler produces an output file with a `.h12` extension which the `ZAP` simulator uses. It also produces an output file with a `.s19` extension which can be loaded into and run on the 68HC12. The `D-Bug 12` monitor, running from HC12 EPROM, loads `S19` records into the HC12 and provides some tools for debugging loaded programs.

The `ZAP` simulator simulates the operation of the 68HC12 on a computer. This has two advantages: you can try a program without having a 68HC12, and you can easily see information (such as register contents, number of cycles to complete, etc.) which are difficult or impossible to observe on an actual microcontroller. Again, the full `ZAP` simulator is a licensed program which cannot be distributed, but the demo version will be adequate for most of the labs in this course.

The relationship between these various programs is illustrated in Figure 1.

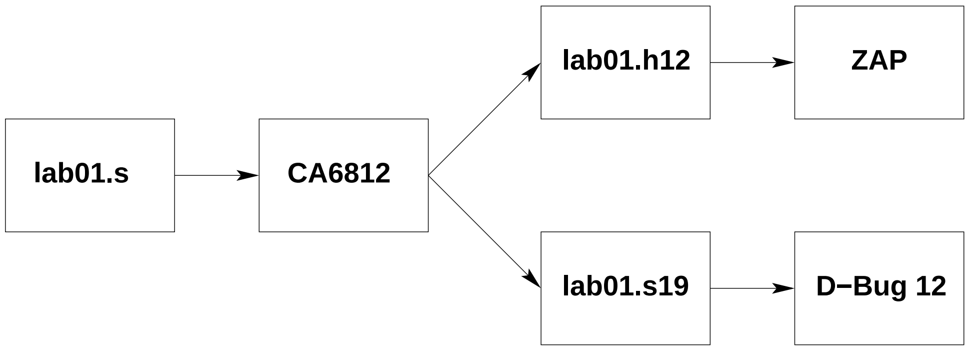

Figure 1: The relationship between `CA6812`, `ZAP`, and `D-Bug 12`.

Pre-Lab

You should read this entire lab **before coming to lab**, and answer all of the questions in the pre-lab section **before coming to lab**.

The D-Bug 12 monitor allows you to interact with the HC12 microcontroller. You can use it to load programs, to find out what is in the HC12's registers and memory, to change the contents the registers and memory, and many other things. The `D-Bug 12` commands of interest for this lab are: `ASM`, `BF`, `BR`, `NOBR`, `G`, `HELP`, `LOAD`, `MD`, `MDW`, `MM`, `MMW`, `RM`, `RD`, and `T`. Read the descriptions of these commands in Chapter 5 of the **D-Bug 12 Reference Guide**.

Questions to answer before lab:

- What is the difference between the `MM` and `MMW` commands?
- How would you change the value of the `B` accumulator to `0x55`? (There are several ways to do this; you only need to find one way.)

Assembly language is used to write programs for the HC12. An assembly language instruction will be converted by the assembler into a single machine instruction for the HC12. The assembly language commands of interest to us for this lab are: `LDAA`, `STAA`, `TAB`, `ABA`, and `ASRA`. Read the descriptions of these commands in the **68HC12 CPU12 Core Users Guide**. At this point you will not understand everything the guide says about these instructions, but you should understand enough to get you through this lab.

Figure 2 is a simple program for the HC12. We will use it to illustrate how to use the assembler, the simulator and D-Bug 12. Note: We will always use the `SWI` instruction to end a program. This instruction will transfer control the the HC12 from the program to the D-Bug 12 monitor.

```
; 68HC12 demo program
; Bill Rison
; 1/15/03

; This is a program to divide and multiply a number by two,
; and store the results in memory

         title   "LAB 1 Demo Program"

evbram:  equ     $1000        ;0x1000 is start of user ram on 68HC12
prog:    equ     evbram       ;Start program at beginning of RAM
data:    equ     evbram+$1000 ;Memory area for data

CODE:    section .text    ;The stuff which follows is program code
         org     prog     ;set program counter to 0x1000
         ldaa    input    ;Get input data into ACC A
         asra             ;Divide by 2
         staa    result   ;Save the result
         ldaa    input    ;Get input data into ACCA
         tab              ;Put the same number into ACCB
         aba              ;Add the number to itself (multiply by two)
         staa    result+1 ;Save the result
         swi              ;End program (return to D-Bug 12 monitor)



DATA:    section .data    ;The stuff which follows is data
         org     data
input:   dc.b    $07      ;first input data
result:  ds.b    2        ;reserve two bytes for results
```

Figure 2: An assembly language program used to get started with the `CA6812`, `ZAP`, and `D-Bug 12`.

Question to answer before lab: What are the contents of the `A` register after each instruction of the program shown in Figure 2 executes?

We will use `D-Bug 12` to explore the memory of the HC12 on your evaluation board. The memory map for your 68HC12 is shown on page 24 of the **MC9S12DP256B Device Users Guide**. (Look the the figure labeled "Normal Single Chip".) Much of the memory is used by the `D-Bug 12` monitor. For this class you can use RAM from 0x1000 to 0x2FFF, and EEPROM from 0x0400 to 0x0FFF.

Use a text editor to enter the program shown in Figure 2 before coming to lab and save it under the name `lab01.s`. (You can also use a web browswer to download the program.)

Also create the file `lab01.lkf` which contains the following lines:

```
# Link file for program lab01.s
+seg .text -b 0x1000 -n .text   # program start address
+seg .data -b 0x2000 -n .data   # data start address
lab01.o                         # application program
```

To assemble the program open a `COMMAND` window. (One way to do this is to go to `START`, the `RUN`, and enter the program name `CMD`.) You can assemble the program using the `ca6812` assembler with the commands:

```
>  c:\cx32\ca6812 -a -l -xx -pl  lab01.s
>  c:\cx32\clnk -o lab01.h12 -m lab01.map lab01.lkf
>  c:\cx32\chex -o lab01.s19 lab01.h12
```

(Note: If you use the demo version of the compiler, you will need to use a different path to the assembler: instead of using `c:\cx32\`, you will need to use `c:\cosmic\eval12\`. Also, you can save typing by using a batch file to execute these commands. To do this, follow the instructions at the end of the lab.)

The `ca6812` command will create a file called `lab01.o` which contains the machine codes in a special form which other programs can read. It also creates the file `lab01.ls` which shows what op codes were generated by the assembler. The `clnk` command will translate the `lab01.o` file into a form which the `ZAP` simulator can understand. It also creates a file `lab01.map` which shows where in memory the program will be loaded. The `chex` command will create a `.s19` file, which is the file you will use with the HC12 evaluation board.

Bring a disk with the program on it to the lab.

As you gain experience you will operate independently in the lab. However, for the first few labs you should be pestering the lab assistants to distraction to make sure you get everything that you can from the time in the lab. We are there to help. Your part is to take the pre-lab seriously and show up for the lab session prepared.

The Lab

Create a directory for this course (say, `D:\EE308`). Inside this directory create a subdirectory for this lab (say, `D:\EE308\LAB01`). Change to this subdirectory, type in the `lab01.s` program, and assemble it with the commands shown above.

Some **questions** on the output of the assembler:

- Look at the `lab01.ls` file. Where will the machine code for the instruction `staa result` be stored in the HC12?
- What machine code is generated for the `staa result` mnemonic? Is this what you expected? (Look up the `STAA` instruction in your **HCS12 Core Users Guide** to determine what code this instruction should generate.)

- At what address will `result` be located in the HC12 memory?

Start the `ZAP` simulator.

Once you are in the simulator, go to the `Setup` menu, `Path` submenu, and `Append` the path where your program is located. Load the `lab01.h12` file using the `File` menu.

Go to the `Show` menu and click on `Memory`. Give a starting address of `0x2000` and a `Data` format. For `Size` select `Byte`. Adjust your desktop so you can see the `Registers`, `Disassembly`, `Command`, and `Memory` windows. Set the program counter to the start of your program by typing the following into the `Command` window:

```
Zap> eval $pc=0x1000
```

In the `Disassembly` window you should see the `lab01` program.

Question

- What value do the HC12 registers have in the `Register Window`?

Trace, or single step, through your program using the `istep` command in the command window, or clicking on the `Footstep` icon, and observe what is going on in the `Disassembly` and `Register` windows. (Note: Do not try to execute the `SWI` instruction on the simulator. It will do unexpected things.)

When you are done single stepping, do the following:

- Set a breakpoint at the `swi` instruction. Do this by double-clicking on the address of the `swi` instruction in the `Disassembly` window. The address should become hightlighted in red. You need to set a breakpoint so the simulator will stop at this address – it does not have D-Bug 12 loaded to recognize the `swi` instruction.
- Reset the program counter to `0x1000`. You can reset the program counter by entering the command

  ```
  Zap> eval $pc=0x1000
  ```

  in the `Command` window, or by double-clicking on the `pc` in the `Register` window and typing in the desired value.
- Run the entire program. You can run the program by entering the command

  ```
  Zap> go
  ```

  in the `Command` window, or by clicking on `Green Light` icon on the icon bar.

Check that you have the expected values in `result` and `result + 1` after the program has finished.

Change the input data at address `0x2000` from `$07` to `$be`, and rerun the program. Check the results.

Some **questions** on the simulator:

- How do the contents of the `A` register compare to what you predicted in the Pre-Lab after the execution of each in instruction?
- Change the contents of the `B` register to `0xAC`. Change the contents of the `A` register to `0xDC`. Set the stack pointer to `0x09D0`. Hint: How did you change the program counter?
- How can you put `0x3421` in the `X` register? How can you verify the change took place? Do it to see if your ideas are correct.
- What do the four columns in the `Disassembly` window represent?
- Reset the program counter to `0x1000` and rerun the program. Display memory to look at `input` and `result`. Is `result` equal to `input`/2? If not, why?

Exit from the simulator. Take out your HCS12 Evaluation Board (EVB), and make sure Jumpers `NO_AUTO`, `MODC` and `MEM_EN` are off, and jumper `JP1` is on. Connect your EVB to your computer and power supply. Open a `Hyperterminal` to talk to the HC12. Your lab instructor will help you connect to the HC12 if you have a problem.

To download your program to your EVB type `LOAD` at the `D-Bug 12` prompt. Then select the `Transfers` menu, `Send Text File` submenu, and select the file `lab01.s19` to send. Wait for the HC12 to respond with `>`.

Using your EVB do the same things that you did on the simulator, and answer all of the questions from the simulator section that are applicable to the EVB. In addition:

- What is the `D-Bug 12` command to change the Program Counter?
- Use the `BF` command to load `0x55` into memory locations `0x2800` to `0x2FFF`. Use the `MD` command to verify that it worked.

Creating an Assembly Batch File

To save typing, you can create an assembly batch file. To do this, use a text editor to create the following file, and call it (for example) `asm.bat`:

```
c:\cx32\ca6812 -a -l -xx -pl  %1.s
c:\cx32\clnk -o %1.h12 -m %1.map %1.lkf
c:\cx32\chex -o %1.s19 %1.h12
```

Now to assemble the file `lab01.s`, give the command:

```
>  asm lab01
```