CEG453 Lab Help
PC-SBC INTERFACE AND CROSS-C COMMANDS

(Modified from the CEG411 Lab Help written by Dr. Awwal)

I. INTRODUCTION

The Arnewsh SBC68K (SBC-Single board computer) is a small, inexpensive microcomputer based on the popular Motorola 68000 microprocessor. It contains the following components:

- MC68000 microprocessor (central processing unit).
- ROM (contains the monitor program TUTOR)
- 128 KB static RAM
- MC68230 PI/T (parallel interface/timer for I/O and timing)
- MC68681 DUART (Dual Universal Asynchronous Receiver & Transmitter for serial I/O)

The User Memory (addresses $000900 - $01FFFF) should be used for the laboratory projects. Note that our program is loaded at address 1000. If you have more questions about the board, such as, "Where are the output ports located? (i.e. which connector to use, J7 or J4 ?), refer to the SBC68K USER'S MANUAL, REV. 1.1 provided in the lab. Leave the book in the lab for others to use. If you want a copy for yourself, you can purchase it at the bookstore.

II. INSTRUCTIONS FOR USING THE ARNEWSH SBC68K SIMULATOR BOARDS

Step 1
Check that the SBC68K board is connected (1) to the PC's COM1 port through the J5 (Terminal) connector and (2) to its power supply (make sure the power strip is switched on!)

Step 2
Power up the PC, and log on using the class/group account assigned to you by the TA.

Step 3
Use the PC's monitor and keyboard as I/O devices for the SBC68K "resident monitor program," (a.k.a. the TUTOR.)

(a) Double click on the MC68000 icon available on the desktop. Alternatively, you may bring up WinNT's hyper-terminal program and load the file "c:\mc68000\mc68000.ht", which contains the appropriate serial communications settings for the SBC68K (COM1, 19,200 baud, 8 bit, no parity, with 1 stop bit.) Check these communications settings by clicking on [File]→[Properties]→[Configure…]

(b) Hit the "carriage return" key and a prompt "`TUTOR 1.32 >`" should appear on the screen. At this point, the terminal program is no longer an I/O device for the PC, it has become an I/O device for the SBC's TUTOR program!

Step 4
Prepare an S-Record File on the PC, which may be downloaded to the SBC68K. Refer to the section "HOW TO USE THE CROSS-CODE-C PROGRAM TOOLS" in this help file.

Step 5
Download an S-Record File from the PC:

(a) On the TUTOR prompt type "`LO1`" so the SBC68K starts listening to the serial link.
(b) On the hyper-terminal program, select [Transfer]→[Send Text File…] and then browse through the directories to select a downloadable file (*.DWN).

Now begin executing your program by typing:

```
TUTOR 1.32 > .PC 1000
TUTOR 1.32 > GO
```

Note: Make sure you add adding the following line at the top of your C program

```
asm("     move.w #$2400,SR");
```

This instruction will properly initialize the MC68000 status register.

Note
To edit text files you may use one of these three options
- The DOS "edit" utility (i.e., on the DOS prompt type "`edit proj1.c`".)
- The standard "notepad" program available in Windows.
- The "pfe" (Programmers File Editor) program that has been installed in your desktop. This is the recommended option.

III. SOME USEFUL TUTOR COMMANDS

1. To trace a program (Insight: T=1 in the Status Register)

   ```
   TUTOR 1.32 > .PC 1000
   TUTOR 1.32 > T
   ```

   Continue to hit return. All the registers will be displayed. You see the instruction, and the effect of the last instruction on screen. Note: The instruction shown at the bottom of the screen will be executed next.

2. Setting "breakpoints": When you single-step (trace) through the program, it may be long and painful before reaching the place you want to investigate. To avoid that, you may set a breakpoint right at the address you want the program to stop. You may use MD as above to find the address, say, 1076, and then set a break point using

   ```
   TUTOR 1.32 > BR 1076
   ```

   Now if you type

   ```
   TUTOR 1.32 > GO 1000
   ```

it will stop at the break point, you can continue thereafter using T

3. Remove breakpoints using

```
TUTOR 1.32 > NOBR.
```

4. When displaying memory which has only data (created by DC or DS directives) but no instruction, use simply

```
TUTOR 1.32 > MD 2000 20
```

   without the ";DI" option. The hex along with ASCII representation of 20 bytes of memory is displayed starting at address 2000.

IV. HOW TO USE THE CROSS-CODE-C PROGRAM TOOLS

Note: These operations are done under a DOS prompt. These tools are in the "path" so you don't need to tell DOS where to find them. The options shown below with the commands are only a few of the options available on these tools. The options are case sensitive. For more information, see the instruction book on the desk in room 339.

1. **Convert a C source file into a relocatable object file, with an assembly listing file**

```
DOS prompt > cc68000 -o LIST= proj1.c
```

   The cc68000 program will create a relocatable object file, proj1.o from the input source file, proj1.c. The -o LIST= option will create an assembly language listing file, proj1.lst. If you desire another file name for your listing file, add it after the equals sign in the command line. If you want to see an assembly language source file created from your C file, use the -S option. For example:

```
DOS prompt > cc68000 -S proj1.c
```

   The output of this operation will be an object file, proj1.o and an assembly file, proj1.s. Your C code will be used to comment the generated assembly code.

2. **Convert an assembly source file into an object file using the CrossCodeC tools**

```
DOS prompt > as68000 -L lab1.s
```

   This will create a relocatable object file, lab1.o, from the program file, lab1.s. The -L option will create a listing file, lab1.lst, of the assembly language program.

3. **Use the linker to link the programs you created and make a load file lab1.out:**

```
linker  -f c:\xcode\ecb.spc lab1.o abcxyz.o c:\xcode\lib\xcode\libc.a -o lab1.out
```

Notes:

- You need to have previously created (or otherwise copied) abcxyz.o before running the linker program.
- The -f switch, and its argument the file named c:\xcode\linker.spc tells the linker how to link the files. This file is IMPORTANT for you to specify the target memory map.
- The file c:\xcode\lib\xcode\libc.a is needed if you are linking in a C program that uses C library functions. If not linking C programs, omit the argument from the command string.
- The -o option lets you select the name for your output file, rather than accepting the default. Check to see that the file "lab1out" was created. This file has resolved all relocatable addresses, replacing them with the actual addresses to be used in SBC memory.

4. **Create absolute assembly listing files:**

We have assembly listings of the relocatable code produced in the compile process. However we don't have an "assembly language equivalent" of the "linked load module" as it will be placed into SBC memory. This is very useful, if not necessary, for debugging. To get the listing with absolute addresses displayed, we use the absolute listing tool:

```
abs lab1.out lab1.lst abcxyz.lst
```

Note that this requires that we have produced listing files previously. These files are processed using the information in the lab1.out file and the output will be two files: lab1.abs and abcxyz.abs.

5. **Create s-record downloader files:**

We now need to make a file in a format suitable for downloading to the SBC board. The format the SBC expects is the Motorola S-Record format. It contains a sequence of records of memory load addresses, object hex code, and record check sums in ASCII. To create it use the downloader tool:

```
down -d mot lab1.out
```

This will create a downloader file named lab1.dwn from the input file lab1.out The switch -d mot tells the program down to make the output file Motorola S-Record compatible. The file lab1.dwn can be loaded into the SBC68K using the hyper-terminal's Transfer utility.

V. USING A MAKEFILE ON THE PC

For assembly language programs:

```
CC =  as68000
CFLAGS = -L
OBJS = lab4.o abcxyz.o
LST = lab4.lst abcxyz.lst

ex1:  $(OBJS)
      linker –f c:\xcode\ecb.spc $(OBJS) -o lab4_load.out
      abs lab4.out $(LST)
      down -d mot lab4.out

lab4.o: lab4.s
      $(CC) $(CFLAGS) lab4.s
abcxyz.o: abcxyz.s
      $(CC) $(CFLAGS) abcxyz.s

clean:
      del *.o
      del *.lst
      del *.abs
      del lab4.out
```

Now type “make”, and all the needed files will be produced. When executing “make clean” the directory will be cleared of the user files.

For C language programs:

```
CC = cc68000
CFLAGS = -o LIST=
OBJS = proj4.o abcxyz.o
LST = proj4.lst abcxyz.lst

ex4: $(OBJS)
      linker   -f   c:\xcode\ecb.spc   $(OBJS)   c:\xcode\lib\xcode\libc.a   -o
proj4.out
      abs proj4.out $(LST)
      down -d mot proj4.out

proj4.o: proj4.c
      $(CC) $(CFLAGS) proj4.c

abcxyz.o: abcxyz.asm
      as68000 -L abcxyz.asm
```