

EE 371 LAB 3 F05

HEXADECIMAL DATA and SUBROUTINES

Did you know that a pun is a rare medium well done?

Schedule: Week of Sept 19: Do the lab
Week of Sept 26: Last chance to demo the lab and hand in the assembler listing.

1. Pre-Lab: Come to the lab with a source file ready to do the following programming assignment.

Specifications: Our best customer at our embedded systems company, *Two Dot Engineering*, requires a program that will convert a single byte to two "ASCII-hex" encoded bytes to allow the byte's binary data to be printed on the screen. For example, if the byte is %1010 0001 the upper nibble, %1010 is to be converted to \$41 (the ASCII code for \$A) and the lower nibble, %0001, is to be to \$31 (the ASCII equivalent of \$1.) Our crack Two Dot Engineering design team decides to implement this with a subroutine that could be used in a variety of applications. We come up with the following specifications for the subroutine `bin_to_ASCII`:

`bin_to_ASCII`: This subroutine accepts a binary number in the B register and returns two ASCII-hex digits in the D register. For example, if B = %1111 0101 (\$F5), the D register will return \$4635 where the \$46 in the A register is the ASCII code for \$F (upper case F) and the \$35 in B is the ASCII code for \$5. These characters can then be printed on the screen.

Input to the subroutine:

B register contains the binary data to be converted

Output from the subroutine:

D register contains the two ASCII-Hex characters.

Only the D register and the condition code register are allowed to be changed by this subroutine.¹

Of course, as is the case with all Two Dot Engineering software, the *ECE Department Assembly Language Practice* document will be followed.

2. Helping Hand #1: Your `bin_to_ASCII` routine will take the binary contents of the B register and convert each nibble in the byte to the equivalent ASCII code. The design to do this is the following:
 - ; Convert a nibble in the range of \$0 to \$F to the ASCII equivalent
 - ; IF the number is higher than 9
 - ; THEN subtract 9 and OR with \$40
 - ; ELSE OR with \$30

To verify for yourself how this works, check out the ASCII codes for the hex digits \$0-\$F.

3. Helping Hand #2: If you have a byte to convert to ASCII, you have to convert each nibble separately. You can do the most significant nibble first and then the least significant. To do this, you have to isolate each nibble. An algorithm to do this is:
 - ; Isolate the most significant nibble from a byte
 - ; Logical shift right (see the Rotate/Shift instructions) four bits
 - ; Convert to ASCII-hex
 - ; Isolate the least significant nibble from a byte
 - ; AND the byte with \$0F
 - ; Convert to ASCII-hex

Name _____

Partner _____

Meeting Day: _____ Hr: _____

Demo: _____

Code _____

¹ When we say a register is not modified by the subroutine we mean that on exit the register must have the same value as on entry. You may, of course, use registers as long as you push them at the beginning of the subroutine and pull them at the end.

4. Helping Hand #3: If a subroutine is in a separately assembled source file (it doesn't have to be, but it is a good idea) the module must make its starting location known to the rest of the world. This is done including the following code in the module:

```
XDEF my_module ; Define the label my_module for external programs
; My module's code is below:
my_module:
    pshx ; Save the x register on the stack
    etc.
    pulx ; Restore the x register
    rts ; Return to the calling program
```

The program that calls your subroutine module has to include the following:

```
XREF my_module ; Tell the assembler that my_module is somewhere else
```

Then any calling program can jump to the subroutine by:

```
jsr my_module
```

5. Helping Hand #4: In addition to the subroutine (function) described above you will need a "main" program to test your routine. I have written one for you to use called N\EE371\lab03_f05\lab3_test.asm.
6. CodeWarrior Specifics: After you create a new project, copy (don't add) the lab3_test.asm program and save it as the "main.asm" program in your project. You can change the name from main.asm if you wish. Then create your subroutine:



Click on the New File Icon and after typing in your subroutine save it (**Save As...**) in your sources folder in your project. Then add it to the project: **Project > Add ...asm to Project**.

Create an assembler list file: **Edit > Simulator Settings > Assembler for HC12 > Options > Generate a listing file**. The .lst file will be in the **bin** directory in your project. You will need to hand in the listing for your subroutine.

Make the program to run in the simulator and demonstrate that it works.

7. Demonstration: You are to demonstrate the program to the customer's representative in the lab. Make sure you can show it working for a variety of situations.
8. Grading: Hand in your list file - one list per group. Demo 10 points (-2 points for each demo not working **100% correctly**), program .LST file 10 points [you will be graded on your use of labels, equates, orgs, comments and other assembler language features - only one listing per group need be submitted].