**Introduction**

In this lab we had to program a 68HC11 microcontroller to create a complex hexadecimal and decimal counter and display it on a single seven-segment display by utilizing interrupts. The counter counts up (or down) to 0-255 in decimal and 00-FF in hexadecimal. There are a total of eleven user inputs that will change a certain setting for the counter. To do all this will require a lot of memory, so the biggest challenge will be to optimize the program to fit everything in.

**Preliminary Work**

Previously, I used all of the legs of a single 7404 chip because I could not get any of the other chips to work. During this lab I tried a friend’s 7406 chip, but it still did not work. I must have been wiring the 7406 chip and resistors incorrectly. I proceeded anyway with two lights missing, the decimal point and one segment. Since the circuit is the same as last lab I omitted the wire list and circuit diagram in this report.

I really struggled making flow charts, because of how long it took to make them by hand and on a PC. It took incredibly long if I wanted to make any edits to it too, so I jumped straight into pseudo code.

**Pseudo code**

//new definitions

PORTB EQU $1004 ;7-segment display (accB)

fast\_speed\_s1 EQU 1

fast\_speed\_s2 EQU 4

normal\_speed\_s1 EQU 2

normal\_speed\_s2 EQU 8

time\_base2 (see case 'N')

ORG data

counter FCB $00 //actual counter variable, not CNTR

NUM FCB $00, $01, $02, $03, $04, $05, $06, $07, $08, $09, $0A, $0B, $0C, $0D, $0E, $0F

HEX FCB $C0, $CF, $92, $86, $8D, $A4, $A0, $CE, $80, $8C, $88, $A1, $F0, $83, $B0, $B8

stride FCB $00

temp\_variable //hopefully can use this variable multiple times to save space

//booleans

decimal\_bool

up\_bool

last\_digit

ORG main

input

(

display menu on the console

pause the counter

inchar

//switch (inchar) or a bunch of if/else statements that change settings according to inchar

case +: start from beginning of data tables using '#'

case -: start from ending of data tables using the specific memory address

case D: if(decimal){run code}

case H: else{run hexadecimal code} [very large if else, try to used shared code when possible]

case R: set counter to 0 and set stride to 1

case S: if(0>char<F){add count accordingly} else{not in the requirements}

case E: JSR to a loop of displaying the last\_digit in port B plus a decimal point

case P: BEQ 'P' to get out of loop

case N: STAA normal\_speed\_s1/s2/(s3) onto CNTR or add/subtract or just set time\_base(?)

case F: STAA fast\_speed\_s1/s2/(s3) onto CNTR or add/subtract

case Q: SWI

)

The program will be set to run the default counter from the beginning.

Many commands (R, S, N, F) simply involve changing values.

For example, reset will look something like this:

RESET LDAB #0 ;counter=0

STAB counter

LDAB #1 ;stride=1

STAB stride

The rest of the commands (D, H, E, P, +, -) will use BNE and BEQ against their associated boolean variables.

For example, the + and - input may look like this:

LDAB up\_bool

BNE count\_down

LDX #NUM ;points to the beginning of NUM

LDY #DAT ;points to the beginning of DAT2

count\_down LDX $0032

LDY $0065

There will be two states for both decimal and hexadecimal. In decimal, the first two digits will use the first state because they have the same time base. The second state will be the digit containing the ‘remaining’ time.

**Problems**

For the first two days of nearly eight hours of work, I could not get the 7-segment display to count. In computer science the compiler tells you where the error is, and if not you can debug it step by step, similar to wookie. In wookie the variables (count & portB) were correct but it still did not display, so the problem in my code was beyond's wookie's capabilites which led me to believe the problem was in an important part of the program such loading the correct registers. This was extremely frustrating and I tried everything. I think the problem was the combination of possibilities that could go wrong, but I finally did get it to work.

**Conclusion**

So far, the program counts in hexadecimal. I quickly added a reset function which worked without a problem. I imagine the rest of the functions shouldn’t be a problem either. After I get the program fully written, I will have to try to optimize the code as much as possible. By Sunday I should have everything completely working, but may have to separate the programs into two due to memory constraints, having one for decimal and the other for hexadecimal. I will send the “How To” along with the program, it shouldn’t be complicated.