6.115 Laboratory 1: Getting Information Into and Out of the Microcontroller

David Ologan

June 9, 2022

1 Exercise 1 : See Shiny Lights

- Assemble the test program above using AS31. Download the resulting HEX file to the R31JP using your Kable and TeraTerm. run the program using either MINMON "G" command or the MON/RUN and reset switches on the R31JP. What do you see on the R31JP lights?

After running the program above on the R31JP, the right most LED on the LED Bank of the R31JP is lit green.

Listing 1: Right LED

```
1  ; This little program turns an LED on.
2
3  mov P1, #00h  ; Clear the LED Bank
4  mov P1, #01h  ; Turn on a Single Light
5  loop:
6     sjmp loop
```

- Explain the purpose of the loop in the test program above.

The purpose of the loop in the test program above is to make sure the program acts as intended. After Assigning 01h to Pin P1, we don't want it to keep reading whatever is up next in memory. Essentially, it gives us control of our program by making sure it acts only as intended.

- Write and test a program to verify which light is connected to Port 1's most-significant bit (MSB).

The LED connected to Port 1's most-significant bit(MSB) is the third one from the left. After sending 80h to P1, that LED is the only one lit. Essentially, the first two LED's from the left aren't in use.

Listing 2: Right LED

```
1  ; This little program turns the MSB on.
2
3  mov P1, #00h  ; Clear the LED Bank
4  mov P1, #80h  ; Turn on a Single Light
5  loop:
6     sjmp loop
```

- Write another short program (5 lines or less) that is functionally identical to the test program, but which uses the clr and setb commands to activate the light. You may use other commands as well, but be sure to use the clr and setb commands in your program.

Listing 3: SETB/CLR

```
1  main:
2  CLR A;
3  SETB ACC.0;
4  mov P1, A  ; Set Value of P1.0
5  loop:
6     sjmp loop
```

- Experiment with the light bank to see if you can make an interesting static (unchanging pattern, e.g. every other light off.

Listing 4: Every Other

```
1  main:
2  mov P1, #00h  ; Clear the LED Bank
3  mov P1, #55h  ; Turn on Every Other Light
4  loop:
5     sjmp loop
```

- Next, consider the problem of making a visibly flashing light or an interesting dynamic pattern on the lights, such as left-to-right and back again), e.g. from the classic TV show or movie of your choice. Think about whether you could accomplish this with the nop, ljmp, and clr commands. What problems might be encountered in the attempt to make a flashing light strictly with mov and nop commands? Can you make things work using the djnz command?

If you wanted to make a dynamic pattern on the lights, you could accomplish this by wasting machine cycles in between turning the lights on and off to simulate a changing pattern. If you only used a nop and a mov, you would need a lot of nops to show the

changes. Your unit of time would be in strictly nops. Making things work with the djnz command would be possible, since you could use the delay to simulate the progression of the dynamic pattern across the lights. Essentially, you would light one led, wait the given delay, and then turn on the next LED, until the pattern finished.

2 Exercise 2: Serial Communications with a Personal Computer

- Complete program segment 2 with register with register constants for 9600-baud communication.
- Assemble the complete program consisting of code segments 1-4, and test your code by connecting the R31JP to a PC. Use a communications program like TeraTerm in Windows on the PC to send and receive characters to and from the micro-controller.

Listing 5: Register Constants for 9600 baud

```
1  .org 00h       ; power up and reset vector
2      ljmp start ; when the micro wakes up, jump to the beginning of
3                 ; the main body or loop in the program, called "start"
4  .org 100h      ; and located at the address location 110h in external mem
5  start:
6      lcall init
7      loop:
8          lcall getchr
9          lcall sndchr
10         sjmp loop
11                        ; set up serial port with 11.0592 Mhz crystal.
12 init:                  ; user timer 1 for 9600 baud serial communication
13     mov tmod, #20h     ; set timer 1 for auto reload mode 2
14     mov tcon, #41h     ; run timer 1
15     mov th1, #0fdh     ; set 9600 baud with xtal = 11.059 mhz
16     mov scon, #50h     ; set serial control reg for 8 bit data
17     ret                ; and mode 1
18
19 getchr:
20     jnb ri, getchr     ; wait till character received
21     mov a, sbuf        ; get character and put in the accumulator
22     anl a, #7fh        ; mask off 8th bit
23     clr ri             ; clear serial "receive status" flag
24     ret
25
26 sndchr:
27     clr scon.1         ; clear the ti complete flag
28     mov sbuf, a;       ; move a character from acc to sbuf
29     txloop:
30         jnb scon.1, txloop ; wait till chr is sent
31     ret
```

- How could program segment 1 be rewritten to use less program memory? There are at least three changes that could be made. Do these changes make the code easier to understand, harder to understand, or leave it about the same?

Program Segment 1 could be rewritten to use less program memory by starting at 00h instead of 100h. We don't use interrupts in this code so this pre-allocated space is unnecessary. Also, instead of using lcall, you could get rid of your subroutines and do everything in main. Getting rid of lcall and the subroutines would complicate your code, since the subroutines allow you to distinguish individual operations.

- In your lab report, also include the register constants for program segment 2 that you would have used if we had instead wanted 2400-baud communication. You do not need to recompile your actual program for 2400-baud operation. Simply note the constants you would have used for 2400 baud communication in your lab report.

In order to run at 2400-baud, we would need to change TH1 to F4h and SMOD = 0. If SMOD = 1, TH1 would need to be E8.

3 Exercise 3: Make an ASCII table and Improve the Typewriter

- Add a single line to Program Segment 1 that causes the ASCII code for each character typed to not only be echoed back to the PC (by the routine sndchr) but also to be displayed on the LED bank connected to Port 1.

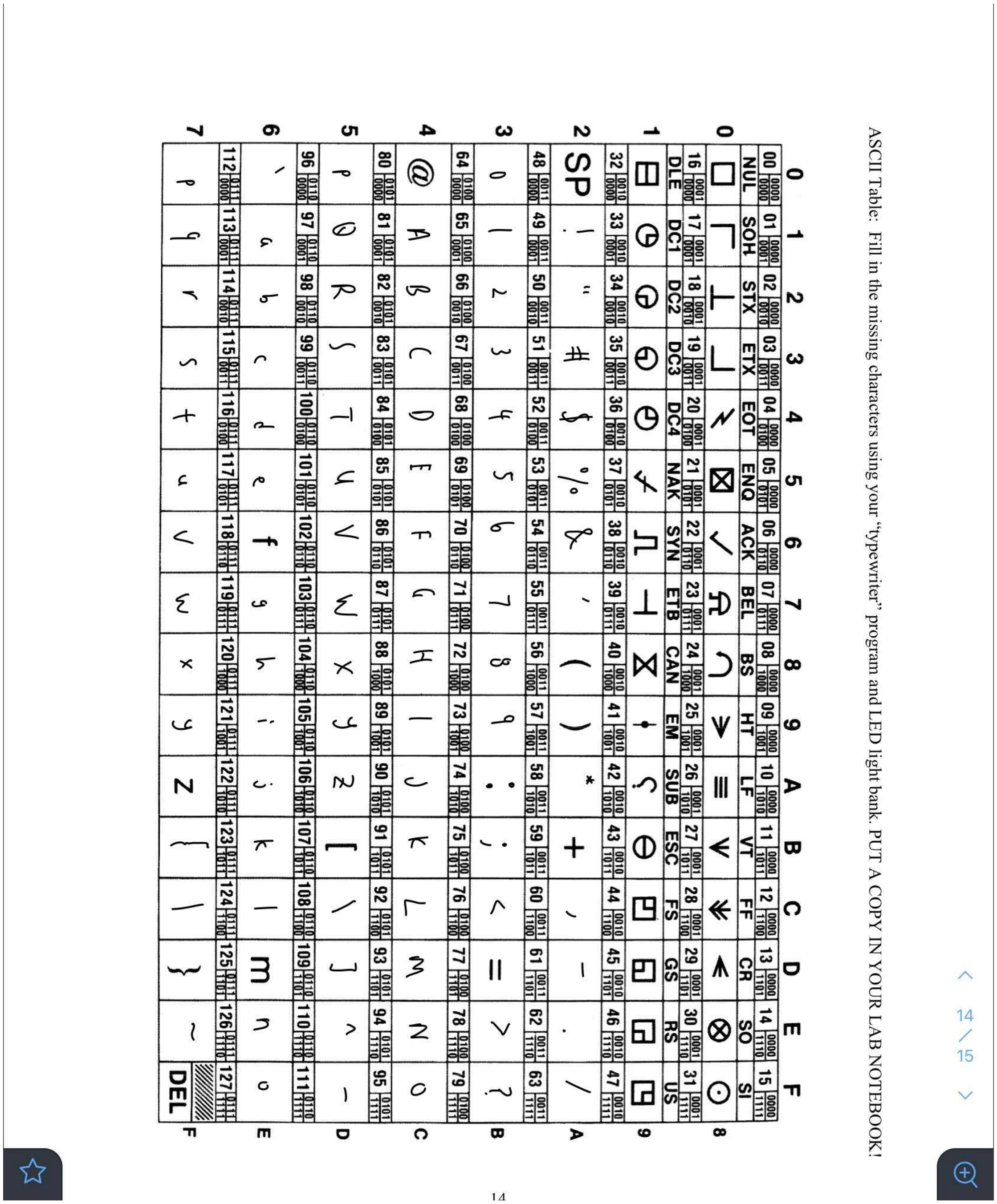
- Implement an "auto-wrap". After the typist has entered 65 key strokes, have the micro-controller automatically issue a carriage return (back to the beginning of the line) AND a line feed (drop down to the next line). Add a subroutine called crlf to your program to provide this operation. The ASCII code fro a linefeed operation is the decimal number 10. The ASCII code for a carriage return operation is the decimal number 13. Use the djnz command to count 65 key strokes.

Listing 6: Typewriter with Carriage Return

```
1  .org 00h       ; power up and reset vector
2      ljmp start ; when the micro wakes up, jump to the beginning of
3                 ; the main body or loop in the program, called "start"
4  .org 100h      ; and located at the address location 110h in external mem
5  start:
6      lcall init
7      mov R0, #41h; Initialize counter for 65 characters
8      loop:
9          lcall getchr
10         lcall sndchr
11         mov P1, a      ; Read letter output to the
12         DJNZ R0, loop  ; decrement R1, if 0, run crlf sub-function
13         lcall crlf     ;carriage return, new line, reset counter
14         sjmp loop
15                        ; set up serial port with 11.0592 Mhz crystal.
16 init:                  ; user timer 1 for 9600 baud serial communication
17     mov tmod, #20h     ; set timer 1 for auto reload mode 2
18     mov tcon, #41h     ; run timer 1
19     mov th1, #0fdh     ; set 9600 baud with xtal = 11.059 mhz
20     mov scon, #50h     ; set serial control reg for 8 bit data
21     ret                ; and mode 1
22
23 getchr:
24     jnb ri, getchr     ; wait till character received
25     mov a, sbuf        ; get character and put in the accumulator
26     anl a, #7fh        ; mask off 8th bit
27     clr ri             ; clear serial "receive status" flag
28     ret
29
30 sndchr:
31     clr scon.1         ; clear the ti complete flag
32     mov sbuf, a;       ; move a character from acc to sbuf
33     txloop:
34         jnb scon.1, txloop ; wait till chr is sent
35     ret
36
```

```
37  crlf:
38      mov A, #0Dh; Carriage Return #13d, #0Dh
39      lcall sndchr
40      mov A, #0Ah; Linefeed #10d,#0Ah
41      lcall sndchr
42      mov R0, #41h; Reset the Counter to 3
43      ret; return to DJNZ call
```

- Use your modified typewriter program to fill in the missing characters on the ASCII table attached to the end of this lab, a sort of periodic table for computer nerds. The first two rows of control codes/special characters have been filled in for you, along with a few other characters to give you the point. you will notice patterns in the table. Please do not guess codes and do not look them up in a reference until you have filled the table using your own program Check each one with a key press. Notice that the table provides each code as a decimal, binary (low and high nibble), and hexadecimal number.



4 Exercise 4: Make a Simple Calculator

- The ASCII code for a number is 3h in hex, e.g. the ASCII code for the digit 9 is 57 in decimal or 39 in hex. Notice that adding d0h to the hex ASCII code for a digit produces the result 0h in hex. This trick can be used to directly convert ASCII codes for single digits back to the actual single digits.

- Write and test your simple calculator program. Notice that the program is intended to work like an HP calculator, i.e., the user "pushes" two three-digit numbers on to the stack, then enters an operation (addition or subtraction). this operation causes two numbers to be popped off the stack, added, and the results displayed automatically on the LED Bank. The User does not have to press an "equals" sign to see a result. Make sure that your program uses the stack of the micro-controller, i.e. makes use of the push and pop instructions. Make sure that your

program does not require a "reset" of the R31JP after each calculation. Your program should keep producing results as long as the user types arguments.

I didn't end up restarting in a new file, so the only version of my calculator program is the modified version. However to accomplish the desired result of displaying the answer on the LED, I would uncomment lines 30 and 45 and send the result from the accumulator to P1. See Exercise 5

- Find out what happens when your overflow or underflow your program. For example, try adding the numbers 250 and 010. Can you explain these results? How is the carry bit in the microcontroller involved in these calculations?

When you overflow or underflow your program, the overflow carry counter is incremented by 1, the count restarts from 0. It loops back around when you need to borrow or you go over the count. I try to think of it like an 8th bit that keep track of things

5 Exercise 5: Modify the Calculator

- Modify your program so that, in addition to displaying the results on the LED bank, the program also displays the sum or the difference on the PC screen.

Listing 7: Modified Calculator

```
1  .org 00h
2      ljmp start ; power up and reset vector
3
4  .org 100h
5
6  start:
7      lcall init        ;Run init to start serial
8      start2:
9      mov R0, #03h      ; Initialize counter for 3 characters
10     mov R1, #02h      ; Initialize counter for when both numbers are recieved
11     loop:
12         lcall getchr
13         lcall sndchr
14         ;mov P1, a; Read letter output to the LED
15         lcall cnvrt    ; converts ASCII to hex, pushes to stack
16         DJNZ R0, loop  ; decrement R0, if not 0 go to loop, tells you when a number has been
               completely entered
17         lcall crlf     ; If so new line protocol, start with next number
18         DJNZ R1, loop  ; decrement R1, if not 0 go to loop, if both numbers have been received
19         lcall two      ; leave me with two 8 bit numbers on the stack
20             lcall getchr
21             lcall sndchr              ; wait for operation entry
22             CJNE A, #2Bh, subtract    ; If not subtract, got to add
23
24             addition:
25                 pop 03
26                 pop 04                ; Pop Off both binary numbers, R4 is the first number,
                       R3 the second
27                 ;mov P1, R4
28                 mov A, R3             ; Move R3 to the acc
29                 ADD A, R4             ; Add R3 and R4
30                 ;mov P1, A; Send binary number into the LED
31                 lcall reverse         ; Convert binary number into ASCII again
32                 lcall crlf            ; Get a new line
33                 mov A, R7             ; Send back the answer in ASCII starting with hundreds
34                 lcall sndchr
35                 mov A, R6             ; Send tens place next
```

```
36                lcall sndchr
37                mov A, R5             ; Send ones place next
38                lcall sndchr          ; Starting with the 100ths place, send the characters
                      back in order
39                sjmp endArithmetic    ; Exit this loop to end arithmetic
40            subtract:
41                pop 03
42                pop 04                ; Pop Off both binary numbers, R4 is the first number,
                      R3 the second
43                mov A, R4             ; Move R4 into acc
44                SUBB A,R3             ; Subtract R3 from R4 which we just put into the
                      accumulator
45                ;mov P1, A
46                lcall reverse         ; Convert binary number into ASCII again
47                lcall crlf            ; Get a new line
48                mov A, R7             ; Send back the answer in ASCII starting with hundreds
49                lcall sndchr
50                mov A, R6             ; Send tens place next
51                lcall sndchr
52                mov A, R5             ; Send ones place next
53                lcall sndchr          ; Starting with the 100ths place, send the characters
                      back in order
54                sjmp endArithmetic    ; Exit this loop to end arithmetic
55            endArithmetic:
56                sjmp start2           ; reset the program start back at the beginning after
                      giving us the desired answer
57            ;sjmp loop2
58
59    init:
60        mov tmod, #20h
61        mov tcon, #40h
62        mov th1, #0fdh
63        mov scon, #50h
64        ret
65
66    getchr:
67        jnb ri, getchr
68        mov a, sbuf
69        anl a, #7fh
70        clr ri
71        ret
72
73    sndchr:
74        clr scon.1
75        mov sbuf, a; Move contents of the accummulator into sbuf
76        txloop:
77            jnb scon.1, txloop
78        ret
79
80    crlf:
81        mov A, #0Dh    ; Carriage Return #13d, #0Dh
82        lcall sndchr
83        mov A, #0Ah    ; Linefeed #10d,#0Ah
84        lcall sndchr
85        mov R0, #03h   ; Reset the Counter to 3
86        ret            ; return to DJNZ call
87
88    cnvrt:             ;convert input into binary, push to stack
89        pop 03
90        pop 04         ;Pop off the return address
91        mov R5, acc    ;move ASCII number from accumulator into R5
92        mov A, #0d0h   ;Convert ASCII to Hex
```

```
93      ADD A, R5       ;Move Hex value into accumulator
94      push acc        ;Push the number into the stack
95      push 04
96      push 03         ;Reload return address
97      ret
98
99
100 two:
101 ;The stack has 6 hex numbers that need to be converted into 2 bytes for the stack
102     pop 03
103     pop 04              ; Pop off return address
104                         ; Second number
105     pop 05              ; pop off ones place
106     pop 06              ; pop off tens place
107     pop 07              ; pop off hundreds place
108
109     mov A, #64h         ; move 100 into the accumulator
110     mov B, R7           ; move R7 into B (hundreds place)
111     MUL AB              ; Multiply hundreds place by 100
112     mov R7, A           ; Move result back into R7
113
114     mov A, #0Ah         ; move 10 into acc
115     mov B, R6           ; move R6 into B (tens place)
116     MUL AB              ; Multiply tens place by 10
117     mov R6, A           ; Move result back into R6
118
119     mov A, R5           ; Move R5 into acc
120     ADD A, R6           ; Add R6 to R5 (tens to ones)
121     ADD A, R7           ; Add R7 to R6 and R5 (hundreds, tens, ones)
122     mov R2, A           ; Save binary second number in R2
123
124                         ; First number
125     pop 05              ; pop off ones place
126     pop 06              ; pop off tens place
127     pop 07              ; pop off hundreds place
128
129     mov A, #64h         ; move 100 into acc
130     mov B, R7           ; move R7 into B (hundreds place)
131     MUL AB              ; Multiply hundreds place by 100
132     mov R7, A           ; Move result back into R7
133
134     mov A, #0Ah         ; move 10 into acc
135     mov B, R6           ; move R6 into B
136     MUL AB              ; Multiply tens place by 10
137     mov R6, A           ; Move result back into R6
138
139     mov A, R5           ; Move R5 into acc
140     ADD A, R6           ; Add R6 to R5 (tens to ones)
141     ADD A, R7           ; Add R7 to R6 and R5 (hundreds, tens, ones)
142     push acc            ; First number added to the stack first
143     mov A, R2           ; Second Number added to acc
144     ;mov P1, a
145     push acc            ; Second number added to the stack next
146     push 04             ; Re add Return address to Stack
147     push 03
148     ret
149
150 reverse:
151     ; takes answer and returns it as three ASCII characters
152     ; start with 8-bit answer in accumulator, push three hex numbers to the stack
153     ; A has the number
154
```

Decimal	Keypad Code	ASCII
1	0FH	31h
2	0EH	32h
3	0DH	33h
4	0BH	34h
5	0AH	35h
6	09H	36h
7	07H	37h
8	06H	38h
9	05H	39h
0	02H	30h

Table 1: Keypad to ASCII Translation

```
155     mov B, #64h    ; moves 100 into B
156     DIV AB         ; divides accumulator value by 100, gets the hundredths place
157     SUBB A, #0d0h  ; converts the hundredths place
158     mov R7, A      ; A has the hundreds place, save it in R7
159
160     mov A, B       ; Take the remainder and move into acc
161     mov B, #0Ah    ; Move 10 into B
162     DIV AB         ; Divide A by 10 to get the tens place
163     SUBB A, #0d0h  ; Convert to ASCII
164     mov R6, a      ; Move tens place into R6
165
166     mov A, B       ; Move Remainder into A
167     ;mov P1, A
168     SUBB A, #0d0h  ; Convert Ones place into ASCII
169     ADD A, #01h    ; Account for Carry Flag
170     mov R5, A      ; Save ones place in R5
171     ret
```

6 Exercise 6: Add a keypad to the calculator

- Write a simple program that allows you to verify that the keypad and 74C922 are working. When a key is pressed and released display the data nibble from the 74C922 on the PC Screen. Send a byte to the PC, with the most significant nibble set to a convenient value, and the least significant nibble set to the data from the 74C922. Write down a table that shows the data nibble provided by the 74C922 for each key. Your ASCII table may be helpful here. Note that the key pressed does not necessarily correspond to the nibble you get.

- Use the db instruction to create a data table called keytab that converts the data nibble from 74C922 to a data byte that, at least for the digits on the keypad, allows a key press to be converted to a digit. That is write an assembly routine that loads the accumulator with the byte that corresponds to the numerical value of a pressed key if that key is a digit.

Listing 8: Verify Keypad Function/ Data table

```
1  .org 00h
2      ljmp start ; power up and reset vector
3
4  .org 100h
5
6  start:
7      lcall init ; intialize serial port
8      loop:
9          lcall getkey  ; call getkey, grabs a keypress from the keypad
```

```
10          lcall fix      ; converts the random nibble into an ascii code
11          lcall sndkey   ; sends the ascii character to the PC
12      sjmp loop
13
14  init:
15      mov tmod, #20h
16      mov tcon, #41h
17      mov th1, #0fdh
18      mov scon, #50h
19      ret
20
21  getkey:
22      jnb P3.3, getkey          ; jump if bit not set, wait for the button press
23      mov P1, #0FFh             ; Set Port 1 high to be read
24      pressdone:
25          jb P3.3, pressdone    ; When the press ends, keep going otherwise wait for it to end
26          mov a, P1             ; Reading to Port 1
27          clr C                 ; Clear the carry flag
28          SUBB A, #0F0h         ; Make first nibble all 0s
29          mov P1, a             ; Writing to Port 1, to see output
30          ret
31
32  sndkey:
33      clr scon.1
34      mov sbuf, a               ; Move contents of the accumulator into sbuf
35      txloop:
36          jnb scon.1, txloop
37      ret
38
39  fix:
40      inc a                     ; increment accumulator
41      movc a, @a+pc             ; grab value from data table, replace it with the correct code
42      ret
43      .db 00h, 00h, 30h, 00h, 00h, 39h, 38h, 37h, 00h, 36h, 35h, 34h, 00h, 33h, 32h, 31h ;
            Datatable with conversion to ASCII characters
```

- Create a calculator that once again computes the sum or difference of two-three digit numbers. Use the keypad to enter each of the three-digit numbers. Use the PC to show all the digits and results. You may continue to use the PC to enter + or -.

Listing 9: Calculator with Keypad

```
1   .org 00h
2       ljmp start        ; Power up and reset vector
3
4   .org 100h
5
6   start:
7       lcall init        ; Run init to start serial
8       start2:
9       mov R0, #03h      ; Initialize counter for 3 characters
10      mov R1, #02h      ; Initialize counter for both numbers
11      loop:
12          lcall getkey      ; Get Key Press from Keypad
13          lcall fix         ; Convert KeyPad Press to ASCII using data table
14          ;mov P1, A
15          lcall sndchr
16          ;mov P1, a        ; Read letter output to the LED
17          lcall cnvrt       ; converts ASCII to hex, pushes to stack
18          DJNZ R0, loop     ; decrement R0, if not 0 go to loop, tells you when a number has
                been completely entered
```

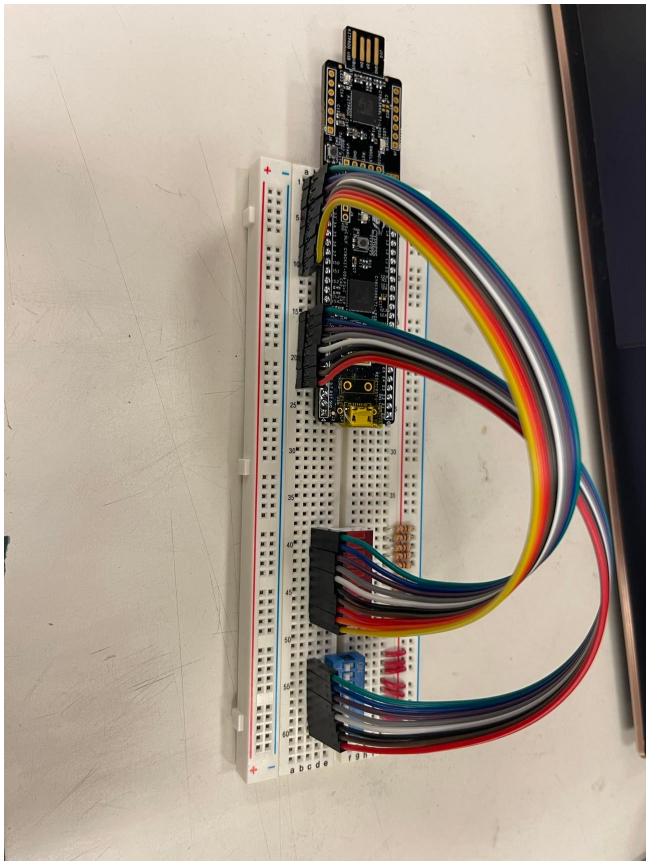
```
19          lcall crlf          ; If so new line protocol, start with next number
20          DJNZ R1, loop       ; decrement R1, if not 0 go to loop, if both numbers have been
                recieved
21          lcall two           ; leave me with two 8 bit numbers on the stack
22              lcall getchr
23              lcall sndchr    ; wait for operation entry
24              CJNE A, #2Bh, subtract;      ; If not Subtraction do addition
25
26              addition:
27                  pop 03
28                  pop 04                  ; Pop Off both binary numbers, R4 is the first
                        number, R3 the second
29                  ;mov P1, R4
30                  mov A, R3               ; Move R3 into acc
31                  ADD A, R4               ; Add R3, and R4
32                  ;mov P1, A              ; Send binary number into the LED
33                  lcall reverse           ; Convert binary number into ASCII again
34                  lcall crlf              ; Get New Line
35                  mov A, R7               ; Send ASCII answer in order, hundreds first
36                  lcall sndchr
37                  mov A, R6               ; Then tens place
38                  lcall sndchr
39                  mov A, R5               ; Finally ones place
40                  lcall sndchr            ; Starting with the 100ths place, send the characters
                        back in order
41                  sjmp endArithmetic      ; Go to end to restart process
42              subtract:
43                  pop 03
44                  pop 04                  ; Pop Off both binary numbers, R4 is the first
                        number, R3 is the second
45                  mov A, R4               ; Move R4 into acc
46                  SUBB A,R3               ; Subtract R3 from R4 which we just put into the
                        accumulator
47                  ;mov P1, A
48                  lcall reverse           ; Convert the binary number into ASCII again
49                  lcall crlf              ; Get New Line
50                  mov A, R7               ; Send ASCII answer in order, hundreds first
51                  lcall sndchr
52                  mov A, R6               ; Then tens place
53                  lcall sndchr
54                  mov A, R5               ; Finally one's place
55                  lcall sndchr            ; Starting with the 100ths place, send the characters
                        back in order
56                  sjmp endArithmetic      ; Go to end to restart process
57              endArithmetic:
58                  sjmp start2             ; reset the program start back at the beginning after
                        giving us the desired answer
59              ;sjmp loop2
60
61  init:
62      mov tmod, #20h
63      mov tcon, #40h
64      mov th1, #0fdh
65      mov scon, #50h
66      ret
67
68  getchr:
69      jnb ri, getchr
70      mov a, sbuf
71      anl a, #7fh
72      clr ri
73      ret
```

```
74
75  sndchr:
76      clr scon.1
77      mov sbuf, a           ; Move contents of the accummulator into sbuf
78      txloop:
79          jnb scon.1, txloop
80      ret
81
82  crlf:
83      mov A, #0Dh       ; Carriage Return #13d, #0Dh
84      lcall sndchr
85      mov A, #0Ah       ; Linefeed #10d,#0Ah
86      lcall sndchr
87      mov R0, #03h      ; Reset the Counter to 3
88      ret               ; return to DJNZ call
89
90  cnvrt:            ; Convert input into binary, push to stack
91      pop 03
92      pop 04        ; Pop off the return address
93      mov R5, acc   ; move ASCII number from accumulator into R5
94      mov A, #0d0h  ; Convert ASCII to Hex
95      ADD A, R5     ; Move Hex value into accumulator
96      push acc      ; Push the number into the stack
97      push 04
98      push 03       ; Reload return address
99      ret
100
101
102 two:
103 ;The stack has 6 hex numbers that need to be converted into 2 bytes for the stack
104     pop 03
105     pop 04        ; Pop off return address
106                   ; second number
107     pop 05        ; Pop off ones place
108     pop 06        ; Pop off tens place
109     pop 07        ; Pop off hundreds place
110                   ; Second Number
111     mov A, #64h   ; Move 100 into acc
112     mov B, R7     ; Move R7 to B (hundreds place)
113     MUL AB        ; Multiply hundreds place by 100
114     mov R7, A     ; Move converted hundreds place into R7
115
116     mov A, #0Ah   ; Move 10 into acc
117     mov B, R6     ; Move R6 into acc (tens place)
118     MUL AB        ; Multiply tens place by 10
119     mov R6, A     ; Move converted tens place into R6
120
121     mov A, R5     ; Move ones place into acc
122     ADD A, R6     ; Add tens place to ones place (R6+R5)
123     ADD A, R7     ; Add R7 to R6 and R5.
124     mov R2, A     ; Save binary second number into R2
125
126                   ; First number
127     pop 05        ; Pop off ones place
128     pop 06        ; Pop off tens place
129     pop 07        ; Pop off hundreds place
130
131     mov A, #64h   ; Move 100 into acc
132     mov B, R7     ; Move R7 into B (hundreds place)
133     MUL AB        ; Multiply hundreds place byb 100
134     mov R7, A     ; Move converted hundreds place into R7
135
```

```
    mov A, #0Ah    ; Move 10 into acc
    mov B, R6      ; Move R6 into acc (tens place)
    MUL AB         ; Multiply tens place by 10
    mov R6, A      ; Move converted tens place into R6

    mov A, R5      ; Move ones place into acc
    ADD A, R6      ; Add tens place to ones place (R6+R5)
    ADD A, R7      ; Add R7 to R6 and R5
    push acc       ; First number added to the stack first
    mov A, R2      ; Move second number into acc
    ;mov P1, a
    push acc       ; Second number added to the stack next
    push 04        ; Readd return address onto stack
    push 03
    ret

reverse:
    ; takes answer and returns it as three ASCII characters
    ; start with 8-bit answer in accumulator, push three hex numbers to the stack
    ; A has the number

    mov B, #64h       ; Moves 100 into B
    DIV AB            ; Divides Answer by 100 to get hundreths place
    SUBB A, #0d0h     ; Convert to ASCII
    mov R7, A         ; Move hundredths place into R7

    mov A, B          ; Move remainder into acc
    mov B, #0Ah       ; Move 10 into B
    DIV AB            ; Divide remainder by 10 to get tens place
    SUBB A, #0d0h     ; Convert to ASCII
    mov R6, a         ; Move converted tens place into R6

    mov A, B          ; Move remainder into acc
    ;mov P1, A
    SUBB A, #0d0h     ; Convert ones place to ASCII
    ADD A, #01h       ; Account for Carry Flag
    mov R5, A         ; Move converted ones place into R5
    ret

getkey:
    jnb P3.3, getkey      ; Jump if bit not set, wait for key press
    mov P1, #0FFh         ; Set Port 1 high to be read
    pressdone:
        jb P3.3, pressdone ; If press detected, wait for it to end
        mov a, P1         ; Reading to Port 1
        clr C
        SUBB A, #0F0h     ; Make first nibble 0s
        ;mov P1, a; Writing to Port 1
        ret

fix:
    inc a                 ; increment acc
    movc a, @a+pc         ; get correct ASCII translation from data table
    ret
    .db 00h, 00h, 30h, 00h, 00h, 39h, 38h, 37h, 00h, 36h, 35h, 34h, 00h, 33h, 32h, 31h ; data
        table with correct ASCII codes
```

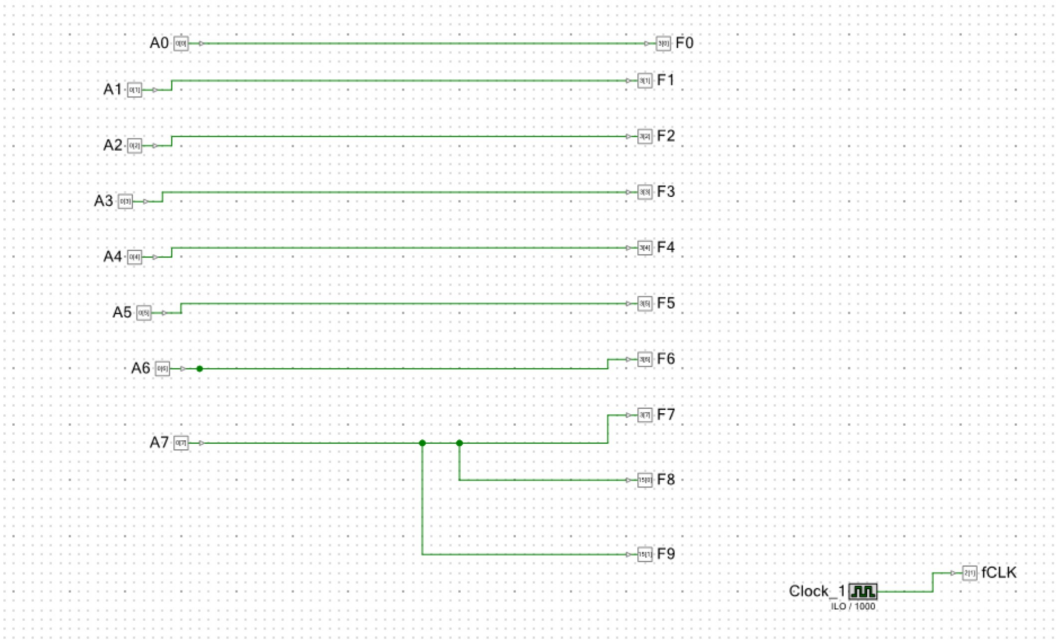
7 Exercise 7: Build your Kovid Konsole

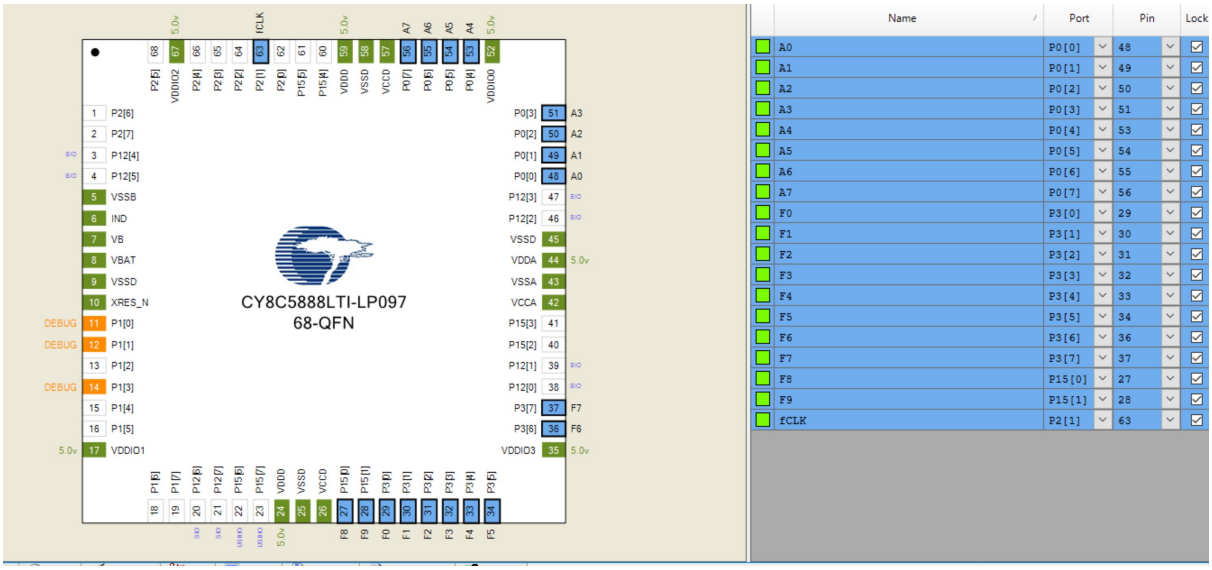
- Carefully wire up your Kovid Konsole. Include good pictures of your Konsole in your lab report.



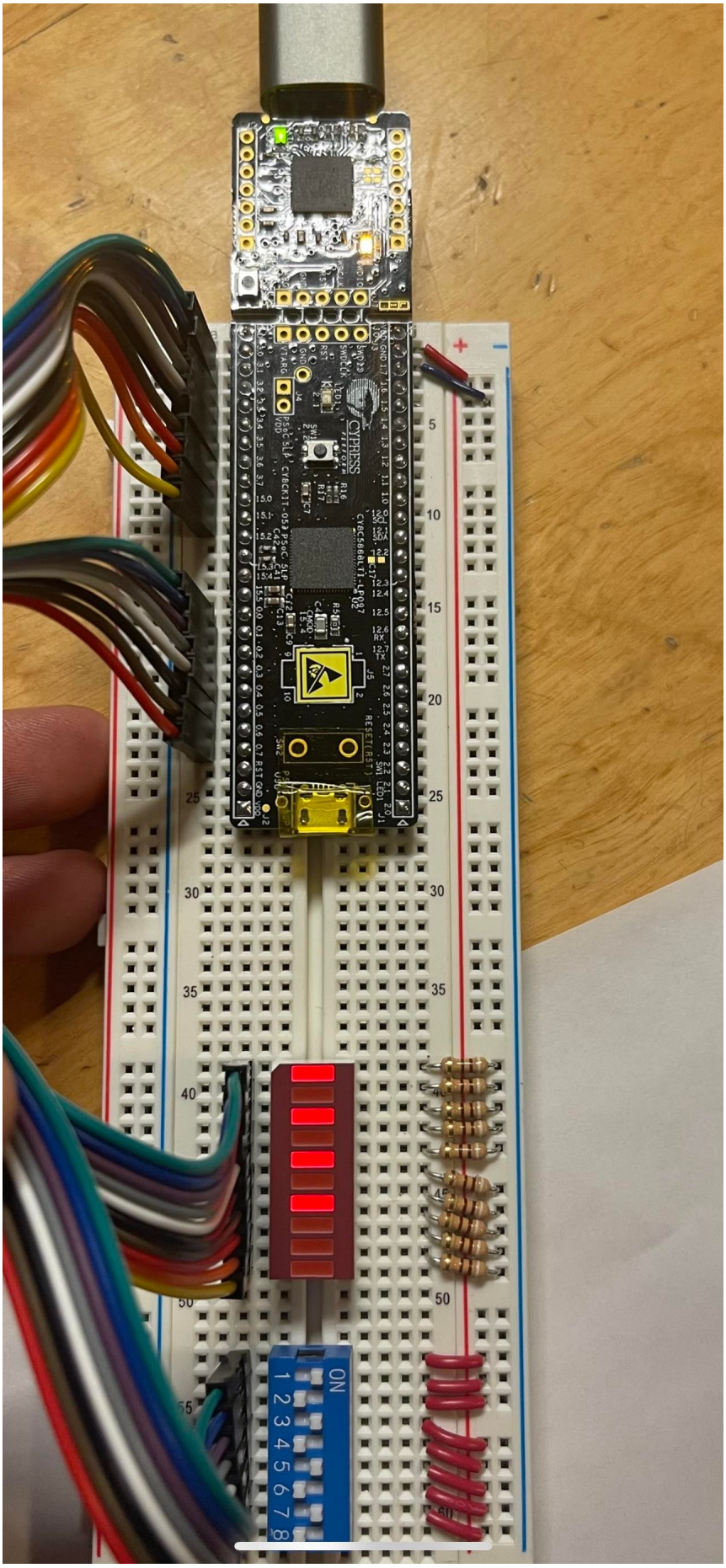
8 Exercise 8: Test your Konsole

- Test your Konsole by trying different switch configurations. Active Switches should create glowing LED's.
- Take screenshots of your Creator Project. Document in your lab report.





- Take a picture of your Kovid Konsole with an interesting light and switch pattern that convincingly shows that your switches correspond to your light pattern. Document in your lab report.


ON
1 2 3 4 5 6 7 8