EECE 690/890
Homework #5
Due Tuesday 10/1/98

1. As discussed in class, it is important to keep our cordless phone's sleep time reasonably short (e.g. under 1 second) to avoid making the phone response time too slow. Ultimately this issue will define how fast the synthesizer must settle on each new frequency as it scans channels, and how fast the data slicer must settle (see last homework assignment).

a) Determine a reasonable spec for the synthesizer and data slicer combined settling time. For this calculation, assume a sleep cycle period of 1 second. Also assume that the phone's total current consumption and total battery capacity are such that it can receive for a total of 10 hours without using a sleep mode, and that marketing has told us that the phone must have an advertised standby time (receive time with sleep mode implemented) of one week.

b) Create a more detailed flow-chart (or pseudo-code description) of the channel scanning process which is shown in simplified form at the top of the Call Initiation flow chart. As a minimum, you should show the basic actions the microcontroller must take to scan the channels, and call out the I/O signals that it must control or monitor during the scan. You do not need to go to the level of working out the routine for sending individual bits to the synthesizer, but you should represent communicating with the synthesizer at some level.

HINT: Refer back to the phone block diagram and think about what events must take place to perform the scan.

2. The data sheet for the PIC17C44 microcontroller states that it contains 33 I/O pins. List the actual pins on the IC (by pin number) that can be used for I/O. HINT: It may help to look at the data sheet handout section on "Special Function Registers".

3. The PIC microcontroller used by your company in the past was a 16C84, which had very few I/O pins. When interfacing this device to the LCD, the LCD's "nibble mode" was used, in which the character to be displayed (an 8-bit ASCII quantity) was sent to the LCD 4 bits at a time. This allowed both the control lines and the data lines needed in communicating with the LCD to be implemented through a single 8-bit microcontroller port (PORTB).

The "write_lcd_chr" routine from your company's old program is shown below, together with variable and constant definitions that it uses. Execute this code on paper, and

determine what bit pattern exists on PORTB each time routine “clock_lcd” is called to latch the data into the LCD. Assume that the character in the W register is an upper-case ‘K’ (ASCII code 4bh) at the start of the routine.

You should show two 8-bit results (e.g. 10101010), one for each time that clock_lcd is called. Show your results with the MSB on the left.

You must also show your work! E.g. tabulate the values of variable temp_write_lcd_chr, register W, and port B at the end of each instruction’s execution to show how you arrived at your final results.

```
; variables
temp_write_lcd_chr   equ      20h

; constants
LCD_RS               equ      5


write_lcd_chr        movwf    temp_write_lcd_chr
                     swapf    temp_write_lcd_chr, W
                     andlw    0fh
                     movwf    PORTB
                     bsf      PORTB, LCD_RS
                     call     clock_lcd

                     movf     temp_write_lcd_chr, W
                     andlw    0fh
                     movwf    PORTB
                     bsf      PORTB, LCD_RS
                     call     clock_lcd

                     return
```