```
$MODDE0CV ; Special Function Registers declaration for CV-8052

org 0000H ; Which code memory location to start at after reset
    ljmp pgrmstart  ; jump down to the label called "pgrmstart"

;These three values are associated with the timing of the circuit
;Tune them to set the "heartbeat" of the program
;The time between "heartbeats" is a function of product of these three
;But none of them can be over 255 (8 bit)
S_TICKS EQU #100
M_TICKS EQU #70
M_TICKS_DOUBLE EQU #140
L_TICKS EQU #150

; Look-up table for my student number, blank, and HELLO
; My student num: 48059760
T_StuNum:
    ;   0   6     7    9    5    0    8    4
    DB 40H, 02H, 78H, 10H, 12H, 40H, 00H, 19H
    ; BLANK
    DB 7FH
    ;   H    E    L   O
    DB 89H, 86H, 0xC7H, 40H
    ;   C    P    N       3    1    2
    DB 0xC6H, 8CH, 0xC8H, 0xB0H, 0xF9H, 0xA4H

;This is a "function", it takes two parameters. These parameters can be accessed using %0 and %1
Display_on mac
    mov dptr, #T_StuNum ; point to student number lookup table
    mov a, %1 ; Load second input into accumulator
    movc a, @dptr+a ; Read from table with the input offset. movc means reading from the code dataspace, and the @
is a dereference thing bc its a pointer
    mov %0, a ; Display number to given (first input) register
endmac

;This is a "function", it takes no parameters. It just displays the first 6 digits of my student number
Most_sig mac
    Display_on(r4, #00)
    Display_on(r5, #01)
    Display_on(HEX0, #02)
    Display_on(HEX1, #03)
    Display_on(HEX2, #04)
    Display_on(HEX3, #05)
    Display_on(HEX4, #06)
    Display_on(HEX5, #07)
endmac

;This is a "function", it takes no parameters. It just displays all blank
All_blnk mac
    Display_on(HEX0, #08)
    Display_on(HEX1, #08)
    Display_on(HEX2, #08)
    Display_on(HEX3, #08)
    Display_on(HEX4, #08)
    Display_on(HEX5, #08)
endmac

;This is a "function", it takes no parameters. It just displays the last 6 digits of my student number
Least_sig mac
    Display_on(HEX0, #00)
    Display_on(HEX1, #01)
    Display_on(HEX2, #02)
```

```
    Display_on(HEX3, #03)
    Display_on(HEX4, #04)
    Display_on(HEX5, #05)
endmac

;This is a "function", it takes no parameters. It just displays HELLO
Hello mac
    Display_on(HEX0, #08H)
    Display_on(HEX1, #12)
    Display_on(HEX2, #1)
    Display_on(HEX3, #1)
    Display_on(HEX4, #10)
    Display_on(HEX5, #09H)
endmac

;This is a "function", it takes no parameters. It just displays CPN312
CPN312 mac
    Display_on(HEX0, #18)
    Display_on(HEX1, #17)
    Display_on(HEX2, #16)
    Display_on(HEX3, #15)
    Display_on(HEX4, #14)
    Display_on(HEX5, #13)
endmac

MODE6LONG:
    ; Mode 6 - Hello() Most_sig() CPN312()
    ; if r4 == 0, Hello
    CJNE r4, #00, MODE6JUMP1
    Most_sig()
    mov r4, #01
    ljmp ENDTIME
    MODE6JUMP1:
    CJNE r4, #01, MODE6JUMP2
    CPN312()
    mov r4, #02
    ljmp ENDTIME
    MODE6JUMP2:
    Hello()
    mov r4, #00
    ljmp ENDTIME

MODE5LONG:
    ; Mode 5 - build it one by one
    ; if r4 == 0, Blank
    CJNE r4, #00, MODE5JUMP1
   All_blnk()
    ljmp ENDTIME
    MODE5JUMP1:
    CJNE r4, #01, MODE5JUMP2
    Display_on(HEX5, #07)
    ljmp ENDTIME
    MODE5JUMP2:
    CJNE r4, #02, MODE5JUMP3
    Display_on(HEX4, #06)
    ljmp ENDTIME
    MODE5JUMP3:
    CJNE r4, #03, MODE5JUMP4
    Display_on(HEX3, #05)
    ljmp ENDTIME
    MODE5JUMP4:
    CJNE r4, #04, MODE5JUMP5
```

```
    Display_on(HEX2, #04)
    ljmp ENDTIME
    MODE5JUMP5:
    CJNE r4, #05, MODE5JUMP6
    Display_on(HEX1, #03)
    ljmp ENDTIME
    MODE5JUMP6: ; else case
    Display_on(HEX0, #02)
    mov r4, #0xFFH ; to make sure it overflows
    ljmp ENDTIME




;This is a "function", it takes no parameters. It just displays the custom display for mode 7
Custom_disp mac
    ; remove 03 and 06
    Display_on(HEX0, #00)
    Display_on(HEX1, #01)
    Display_on(HEX2, #02)
    Display_on(HEX3, #04)
    Display_on(HEX4, #05)
    Display_on(HEX5, #07)
endmac

Scrll_left mac
    mov b, r5 ; "temp" keep

    mov a, r4
    mov r5, a
    mov r4, HEX5
    mov HEX5, HEX4
    mov HEX4, HEX3
    mov HEX3, HEX2
    mov HEX2, HEX1
    mov HEX1, HEX0
    mov HEX0, b
endmac

Scrll_right mac
    mov b, HEX0 ; "temp" keep

    mov HEX0, HEX1
    mov HEX1, HEX2
    mov HEX2, HEX3
    mov HEX3, HEX4
    mov HEX4, HEX5
    mov HEX5, r4
    mov a, r5
    mov r4, a
    mov r5, b
endmac

;For a 33.33MHz clock takes 30ns
WaitHalfSec:
    mov R2, #90
L3: mov R1, #250
L2: mov R0, #250
```

```
L1: djnz R0, L1 ; 3 machine cycles, 22.5us
    djnz R1, L2 ; 5.625ms
    djnz R2, L3 ; 0.5s approx
    ret


; The code under this label runs once, when the program starts
pgrmstart:
    ; Turns off LEDs and stuff
    mov SP, #0x7f
    mov LEDRA, #0 ; Bit addressable
    mov LEDRB, #0 ; Not bit addressable

    ; Start timer
    mov r1, S_TICKS
    mov r2, M_TICKS
    mov r3, L_TICKS



    ; Mode 0 by default, display first 6 digits
    Most_sig()

; The code under this label runs continuously
loop:
    ; Latching logic
    jb key.3, ENDLATCHtmp   ; jump if bit 3 of switch is = 1
    ljmp LATCHLOGIC
ENDLATCHtmp: ljmp ENDLATCH  ; I have to do this weird jumping bc jb can only jump up to 127 lines

; The code under this label has to do with latching our mode, and updating the display
; to the "first frame" of that mode
LATCHLOGIC:
    ; Evaluate initial display depending on mode
    mov a, SWA
   ANL a, #07           ; strip A to only the least significant 3 values by ANDing it with 00000111 (Not strictly nesc.)
    mov r0, a           ; store switch values in r0
    ; Think of this as a big Switch statement that finds what mode we are in
    ; There is a better way to do this, as a lot of the modes have the same start
    ; However, to start with this is a more clear way to see what is going on
    CJNE r0, #00, MODE1         ; jump if r0 != byte
    ; Mode 0
    Most_sig() ; display first 6 digits

    ljmp ENDLATCH
MODE1:  CJNE r0, #01, MODE2         ; jump if A != byte
    ; Mode 1
   All_blnk()
    Display_on(HEX0, #00)
    Display_on(HEX1, #01)

    ljmp ENDLATCH
MODE2:  CJNE r0, #02, MODE3         ; jump if A != byte
    ; Mode 2
    Most_sig() ; display first 6 digits
    ljmp ENDLATCH
MODE3:  CJNE r0, #03, MODE4         ; jump if A != byte
    ; Mode 3
    Most_sig() ; display first 6 digits
    ljmp ENDLATCH
MODE4:  CJNE r0, #04, MODE5         ; jump if A != byte
    ; Mode 4
```

```
    Least_sig() ; display last 6 digits
    mov r4, #00
    ljmp ENDLATCH
MODE5:  CJNE r0, #05, MODE6         ; jump if A != byte
    ; Mode 5
   All_blnk()
    mov r4, #00
    ljmp ENDLATCH
MODE6:  CJNE r0, #06, MODE7         ; jump if A != byte
    ; Mode 6
    Hello()
    mov r4, #00
    ljmp ENDLATCH
MODE7:  ; this should be the "else" case
    ; Mode 7
    Custom_disp()

    ljmp ENDLATCH
ENDLATCH:

; The code under this label has to do with timekeeping
TIMELOGIC:
    ; Timing logic
    djnz r1, ENDTICK
    mov r1, S_TICKS     ; if we got here, that means r1 is zero
    djnz r2, ENDTICK
    ; if we got here, that means r2 is zero
    JNB SWA.3, LONGERTIME ; Jump if bit = 0
    mov r2, M_TICKS
    ljmp SHORTERTIME
    LONGERTIME:
    mov r2, M_TICKS_DOUBLE
    SHORTERTIME:
    djnz r3, ENDTICK
    mov r3, L_TICKS ; if we got here, that means r3 is zero
    ljmp HEARTBEAT

    ENDTICK:
    ljmp ENDTIME


    ; these lines execute once a "heartbeat"
HEARTBEAT:
    cpl LEDRA.0     ;flip LED to visualize heartbeat

    ; Think of this as a big Switch statement that finds what mode we are in
    ; Modes that have time-dependant behavior: 2, 3, 4, 5, 6
    CJNE r0, #02, MODE3T        ; jump if r0 != byte
    ; Mode 2
    Scrll_left()
    ljmp ENDTIME
MODE3T: CJNE r0, #03, MODE4T        ; jump if r0 != byte
    ; Mode 3
    Scrll_right()

    ljmp ENDTIME
MODE4T: CJNE r0, #04, MODE5T        ; jump if A != byte
    ; Mode 4 - Flash
    ; if r4 == 0, blank
    CJNE r4, #00, FLASHJUMP
   All_blnk()
    mov r4, #01
```

```
    ljmp ENDTIME
    FLASHJUMP:
    Least_sig()
    mov r4, #00
    ljmp ENDTIME
MODE5:CJNE r0, #05, MODE6T        ; jump if A != byte
    ; Mode 5 - One by one
    inc r4
    ljmp MODE5LONG
MODE6:CJNE r0, #06, ENDTIME       ; jump if A != byte
    ; Mode 6 - Hello cycle
    ; Couldn't fit it in so had to do a scuffed long jump away then back
    ljmp MODE6LONG
ENDTIME:

    ljmp loop ; Go back up to loop to keep repeating forever
END
```