Lecture 13a

1) The machine code below runs in a CV-8052 processor that takes 1 clock per cycle with a 33.3333MHz clock. Find how long it takes this code to execute. (Warning: there is a loop you have to consider!) 7E 08 0E 00 8E 80 00 00 DE FA

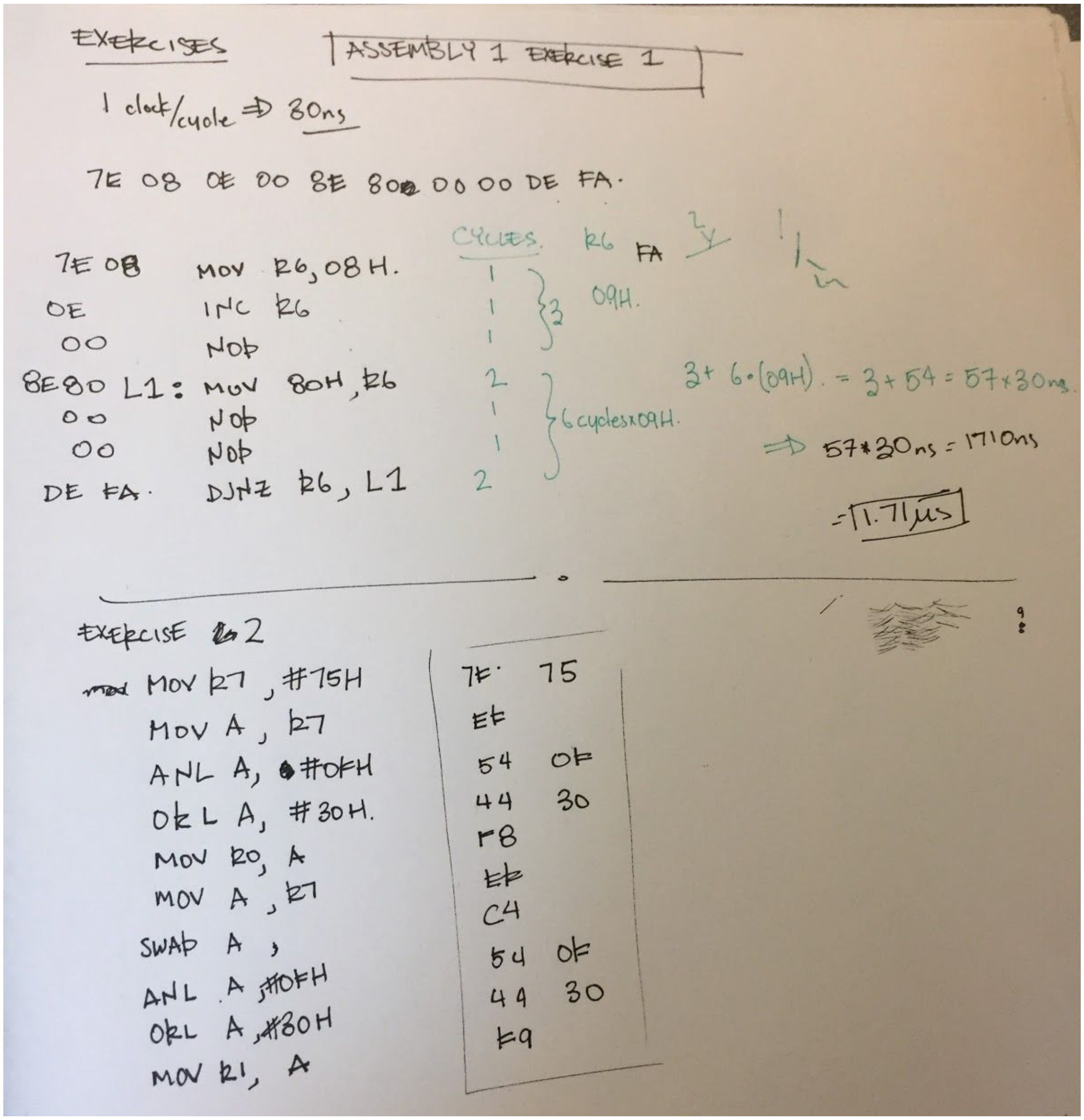
address	opcode	instruction	cycle
0000		Org 00H	
0000		myprogram:	
0000	7E 08	Mov r6, #08h	2
0002	0e	Inc r6	1
0003	00	nop	1
0004		L1:	
0004	8E 80	Mov 80H, R6	2
0006	00	nop	1
0007	00	nop	1
0008	DE FA	Djnz r6, L1	2 (if no jump) 3 (if jump)

4 + 8*7 + 6 = 66; 66*30ns = 1.98us (Jason did this)
Amar: 4 + 9 * 7 - 1 = 66

2) Assemble by hand (both op-codes and operands) the program below. MOV R7, #75H MOV A, R7 ANL A, #0FH ORL A, #30H MOV R0, A MOV A, R7 SWAP A ANL A, #0FH ORL A, #30H MOV R1, A

7F 75 EF 54 0F 44 30 F8 EF C4 54 0F 44 30 F9

3) Modify the programs of examples 3 and 4 so that they turn off all unused LEDs by writing zero to them. The SFR addresses for the LEDS are: LEDR0-7=E8H, LEDR8-9=95H.**Lecture 13**



EXERCISES

ASSEMBLY 1 EXERCISE 1

1 clock/cycle ⇒ 30ns

7E 08 0E 00 8E 80 00 00 DE FA.

		CYCLES.	R6 FA
7E 08	MOV R6,08H.	1	
0E	INC R6	1	3 — 09H.
00	NOP	1	
8E 80 L1:	MOV 80H,R6	2	3+ 6·(09H). = 3+54 = 57×30ns.
00	NOP	1	6 cycles×09H.
00	NOP	1	⇒ 57*30ns = 1710ns
DE FA.	DJNZ R6, L1	2	= $\boxed{1.71\mu s}$

EXERCISE 2

MOV R7 , #75H	7F· 75
MOV A , R7	EF
ANL A, #0FH	54 0F
ORL A, #30H.	44 30
MOV R0, A	F8
MOV A , R7	EF
SWAP A ,	C4
ANL A ,#0FH	54 0F
ORL A ,#30H	44 30
MOV R1, A	F9

Lecture 13b

1)Explain the differences between these two assembly instructions: mov a, #10H mov a, 10H

The first moves the hex number 10 to the accumulator
The second moves the value stored at location 10 to the accumulator

2)There are two ways to access the internal memory of the 8051/8052 microcontroller: directly and indirectly. Explain why the combination of these two instructions mov R0, #0A0H mov a, @R0 and this supposedly equivalent instruction mov a, 0A0H Result with different values in the accumulator.

The first is indirect addressing. It is addressing A0H in upper 128 ram, using R0 as a pointer.
The second iis direct addressing the SFR memory location A0H.

3) **Exercise 3 - Lecture 13B**

```
$MODDE0CV
org 0x0000
    ljmp setup

Wait1s:
     mov R2, #178
L3:  mov R1, #250
L2:  mov R0, #250
L1:  djnz R0, L1 ; 3 machine cycles-> 3*30ns*250=22.5us
     djnz R1, L2 ; 22.5us*250=5.625ms
     djnz R2, L3 ; 5.625ms*180=1s (approximately)
     ret

; Look-up table for 7-seg displays.
T_7seg:
     DB 40H, 79H, 24H, 30H, 19H
     DB 12H, 02H, 78H, 00H, 10H
     DB 08H, 03H, 46H, 21H, 06H
     DB 0EH

setup:
    mov R7, #0
main:
    mov dptr, #T_7seg
    ; digit 6
    mov a, R7
    anl a, #0xf
    movc a, @a+dptr
    mov HEX0, a
    inc R7
    lcall Wait1s
    cjne R7, #00001010B, main
```

```
    mov R7, #0
    ljmp main
END
```

Q5)

```
decrdptr MAC
     Mov a, DPL
     JNZ L1
     Dec DPH
L1:  Dec DPL
ENDMAC
```

Lecture 14

1)Write an assembly subroutine for the 8051 that checks if a 32-bit number stored in registers R0 to R3 is zero.

```
L1:
        Mov a, R0
        Jnz NotZero
        Mov a, R1
        Jnz NotZero
        Mov a, R2
        Jnz NotZero
        Mov a, R3
        Jnz NotZero
        Ljmp Zero
        ret
```

Where Zero and NotZero are other labels which define the operations after determining if the 32-bit number is 0.

2)Write the assembly equivalent of this piece of C code (the size of int is 2 bytes): unsigned int x, y; unsigned char z; . [other code comes here] . if (x>y) z=0; else z=1;

;NOTE: this will work iff the MSByte of X or Y is stored in the lower memory location

```
DSEG
X: DS 2
Y: DS 2
Z: DS 2


[Other Code]

Mov R0, #X
Mov R1, #Y
Mov R7, #2

L1:
Clr c
Mov a, @R1
Subb a, @R0
Jc IsGreater
Inc R1
Inc R2
Djnz R7, L1
Mov R0, #Z
Mov @R0, 1
Ljmp forever

IsGreater:
Mov R0, #Z
Mov @R0, 0

Forever:
	Ljmp forever
```

Lecture 15

Write an assembly program to multiply the 24-bit binary number stored in registers R2, R1, R0 (R0 is the least significant byte) by 10 (decimal). Save the result in R3, R2, R1, R0. Use the MUL AB instruction.

```
	clr c
```

```
mov a, #10
mov b, R0
mul ab
mov R0, a
mov R4, b
mov a, #10
mov b, R1
mul ab
addc a, R4
mov R1, a
mov R4, b
mov a, #10
mov b, R2
mul ab
addc a, R4
mov R2, a
mov R5, #0
mov a, b
addc a, R5
mov R3, a
```

Find square root of DPH-DPL and store it in R7. Use binary search

```
; this only works if DPH-DPL is a perfect square, but if you want to stop it from
doing an infinite loop and return a rounded value just add a check for r0 > r1
; EDIT: returns approx value now

Sqrt:
      Mov R0, #00 ; low
      Mov R1, #FF ; high
Search:
      Clr c
      Mov a, R1
      Subb a, R0
      Jc Found

      Clr c
      Mov a, R0
      Add a, R1
      Rrc a ; div by 2
      Mov r2, a ; mid
      Mov b, a
      Mul ab
      Mov R3, a
      Mov R4, b

      Clr c
      Subb a, DPH
      Jc less
```

```
      Clr c
      Mov a, DPH
      Subb a, R3
      Jc greater

      Clr c
      Mov a, r4
      Subb a, DPL
      Jc less
      Clr c
      Mov a, DPL
      Subb a, R4
      Jc greater

Found:
      Mov R7, R2 ; found the solution
      Ret

Less:
      Mov R0, R2
      Ljmp Search

Greater:
      Mov R1, R2
      Ljmp Search
```

A common way of passing parameters to a function is via the stack. Modify the function WaitHalfSec so that it receives the number of half-seconds to wait in the stack. (Note: this problem is not as trivial as it sounds. You may need to increment and/or decrement register SP to solve this problem)

//Explanation: This assumes somewhere else (i.e. a main/forever loop), we passed in how many 1s waits we wanted into register 3, R3 and used the following commands
push AR3
lcall Wait1s

The lcall adds two more bytes to the SP which is why we decrement sp to get to where R3 is stored on the stack

Wait1s:
dec sp
dec sp
pop R3
L4: mov R2, #180
L3: mov R1, #250
L2: mov R0, #250
L1: djnz R0, L1 ; 3 machine cycles-> 3*30ns*250=22.5us
djnz R1, L2 ; 22.5us*250=5.625ms
djnz R2, L3 ; 5.625ms*180=1s (approximately)
djnz R3, L4

```
inc sp
inc sp
inc sp
Ret
```

Most C programs pass parameters to functions via the stack. Also C programs use the stack to allocate automatic variables (local variables defined within the function). This works fine most of the time, but sometimes a condition commonly known as “stack overflow” occurs. Explain what causes “stack overflow”.

Wikipedia: Call stack pointer exceeds the stack bound
In other words, stack too big

Lecture 16

3)Write an Interrupt service routine for timer 0 that generates a 1 kHz square wave in pin P0.0 of the CV-8052 processor.

(Is it asking just for an ISR or for the whole code????)

```
org 0h
ljmp InitTimer

org 0bh
	ljmp timer0_ISR
timer0_ISR:
	cpl P0.0
	clr TR0 ; Disable timer 0
	mov TH0, #high(RELOAD_TIMER0_500us )
	mov TL0, #low(RELOAD_TIMER0_500us )
	clr TF0 ;Clear the timer flag
	setb TR0 ; Enable timer 0
	 reti

XTAL equ 33333333
FREQ equ 2000 ; 0
RELOAD_TIMER0_500us equ 65536-(XTAL/(12*FREQ))

InitTimer:
Setb EA
Setb ET0
mov a, TMOD
anl a, #11110000B ; Clear bits for timer 0, keep bits for timer 1
orl a, #00000001B ; GATE=0, C/T*=0, M1=0, M0=1: 16-bit timer
mov TMOD, a
```

```
clr TR0 ; Disable timer 0
mov TH0, #high(RELOAD_TIMER0_10ms )
mov TL0, #low(RELOAD_TIMER0_10ms )
clr TF0 ;Clear the timer flag
setb TR0 ; Enable timer 0

Forever:
	Ljmp Forever
```

4)Write an interrupt service routine for timer 2 that increments a two digit BCD counter displayed in the 7-segment displays HEX1 and HEX0 of the CV-8052 every second. Make sure that the ISR for this question and the ISR from the previous question can run concurrently in the same processor.

```
Org 0
	Ljmp InitTimer
Org 2bh
	Ljmp MyIsr

MyISR:
	Push Acc
	Clr TF2
	Dec R7
	Mov A, R7
	Jz Increment

	Ljmp Return

Increment:
	Inc R0
	Mov A, R6
	Subb A, R0
	Jz reset_r0
	Ljmp Return:
Reset_r0:
	Mov R0, #0
	Ljmp return:
Return:
	Pop Acc
	reti

XTAL equ 33333333
FREQ equ 100 ; 1/100Hz=10ms
RELOAD_TIMER0_10ms equ 65536-(XTAL/(12*FREQ))

InitTimer:
```

```
Setb EA
Setb ET2
Clr TR2
Clr TF2
Clr Rclk
Clr Tclk
Clr  TCON2.1
Mov RCAP2L=#low(RELOAD_TIMER0_10ms)
Mov RCAP2h=#high(RELOAD_TIMER0_10ms)
Mov R7 #100
Mov R6, #100
Setb TR2

Forever:
        lcall hex2bcd_16bit  ;assume they have already been written using R0
        ; cus I don't want to
        lcall Display_BCD
        Ljmp Forever
```

6)Program profiling is used to find the usage of resources by a piece of code (a subroutine, for example). A profile value often needed is execution time. Show how to use timer 0 to find out the execution time of a subroutine.

```
Org 0h
    Ljmp Find_time
Org 0bh
    Ljmp timer0isr
Timer0isr:
    Inc R0
    Clr TR0
    Mov TH0, #0
    Mov TL0, #0
    Setb TR0
reti


Find_Time:
    Setb, ET0 ;enables timer 0 interrupt
    Mov a, Tmod
    Anl a, #1111000B
    Orl a, #00000001B
    Mov TH0, #0
    Mov TL0, #0
    Mov R0, #0 ;stores overflow in R0
    Clr TF0
    Setb TR0
    Lcall Function
```

```
CLR TR0
ret

;number of clks (x12???) it took stored in THL and TH0
Total number of times TH0-TL0 is incremented is TH0-TL0 + R0*FFFFH
Time (in seconds) =  total num times incremented*12/33.33E6
```

Lecture 18

2) Time-to-distance Table look-up by value (Binary Search)

Notes:

- myTime = time data, for which distance is wanted
- Time2Dist = look-up table with time constants stored

- R0 and R2 defines the range of numbers (distances) we are considering, R0>R2
- R1 = (R0 - R2)/2 = distance currently being considered
- (R6,R7) = time corresponding to distance R1

```
MOV R2, #0
MOV R0, #250
MOV R1, #125
MOV DPTR, #Time2Dist

Search:
    ; start by filling (R6,R7) with the time corresponding to the distance value currently in R1
    MOV A, R1
    MOVC A, @A+DPTR
    MOV R7, A
    MOV A, R1
    INC A
    MOVC A, @A+DPTR
    MOV R6, A
    MOV A, R7
    ; now that we have (R6,R7), we can compare this time value to the time data we have in myTime, to
    see if R1 is the distance value that matches myTime
    CLR C
    SUBB A, myTime+1 ; subtracting: R7 - (low byte of myTime)
    MOV A, R6
    SUBB A, myTime ; subtracting: R6 - (high byte of myTime)
    JNC Bigger ; If carry == 0, (R6, R7) > myTime, so we have to look at a bigger distance R1
    JNZ Smaller ; Else, if a != 0, then (R6, R7) < myTime, so we have to look at a smaller R1
    RET ; If carry == 1, and a == 0, then (R6,R7)==myTime, so we have the right R1!
Bigger:
    ; re-adjust the lower bound R2 to be higher
    MOV A, R1
    MOV R2, A
```

```
        MOV A, R0
        CLR C
        JNC NewDist
Smaller:
        ; re-adjust the upper bound R0 to be lower
        MOV A, R1
        MOV R0, A
        CLR C
        JNC NewDist
NewDist:
        ; re-calculate R1 and go back to check whether this new value is the correct distance
        SUBB A, R2
        RRC A
        MOV R1, A
        CJNE A, AR0, Search
```

3) •Design an extended ‘XRAM’ memory decoder for the 8051 microcontroller so that:

Address range Function Access type 0000H to 7FFFH RAM (32k) Read/Write 8000H to 8FFFH EEPROM (4k) Read/Write 9000H to 9FFFH RAM (4k) Read/Write A000H to AFFFH Input Read Only B000H to BFFFH Output Write Only C000H to CFFFH EPROM Read Only D000H to DFFFH Reserved for future use Read/Write E000H to EFFFH Reserved for future use Read/Write F000H to FFFFH Reserved for future use Read/Write

Final, Question 1:

Address	Opcode/Operands	Instruction
3000		org 3000H
3000		BCD_X_20:
3000		; BCD*2
3000	EC	MOV A, R4
3001	2C	ADD A, R4
3002	D4	DA A
3003	FC	MOV R4, A
3004	ED	MOV A, R5
3005	3D	ADDC A, R5
3006	D4	DA A
3007	FD	MOV R5, A
3008		; Multiply BCD*2 by 10
3008	7904	MOV R1, #4
300A	C3	L1: CLR C
300B	EC	MOV A, R4
300C	33	RLC A
300D	FC	MOV R4, A
300E	ED	MOV A, R5
300F	33	RLC A
3010	FD	MOV R5, A
3011	D9 F7	DJNZ R1, L1
3013	22	RET

Question 2:

```
90 00 03          MOV DPTR, #0x0003
C3                CLR C
94 20             SUBB A, #0x20
75 F0 06          MOV B, #0b110
A4                MUL AB
25 82             ADD A, DPL
F8 82             MOV DPL, A
E5 F0             MOV A, B
35 83             ADDC A, DPH
F5 83             MOV DPH, A
```

Question 3:

Instruction	Cycles	Note
`Wait:`		
`push psw`	3	
`push acc`	3	
`push AR0`	3	
`mov a, R1`	1	`a = R1`
`add a, R1`	1	`a = 2*R1`
`add a, #50`	2	`a = 2*R1 + 50`
`mov R0, a`	1	`R0 = 2*R1 + 50`
`W1: djnz R0, W1`	2 if R0 = 0, else 3	Runs R0 times, jumps to self, total execution time is 3*R0 - 1 cycles.
`pop AR0`	3	
`pop acc`	3	
`pop psw`	3	
`ret`	3	

`mov R1, #40`	2	
`lcall Wait`	3	

Runtime:

Steps	Cycles
Call subroutine	3 *don't think this is necessary as it is asking for the time it takes for the **subroutine** to run
Push registers	9
Compute R0	5
Waiting Loop	3 * (2 * R1 + 50) - 1 == 6 * R1 + 149
Pop register	9
Return	3
Fixed execution time	3 + 9 + 5 + 149 + 9 + 3 = 178
Variable execution time	6 * R1

Overall execution time: 6 * R1 + 178 cycles
Counting time from the lcall to when it returns.
For R1 = 40, execution time is 418 cycles.

b)

T = [180R1+5250] x 10^(-9)   *Note: this is with 175 cycles instead of 178

Question 4:

Write a SHORT (fewer bytes as possible) assembly subroutine for the 8051 microcontroller to perform the operation R=M-S, where R, M, and S are defined as: DSEG at 40H M: DS 8 S: DS 8 R: DS 8

Question 5:

Right now, if the sum of the 8 inputs overflows 16 bits, the result will be incorrect. Fix: Sum to a 24 bit result, then after divide that by 8, the MSB will be 0 and can be discarded.

^true, see solution #2 for an attempt at solving this problem

```
Avg_16: ; this might just be horribly wrong or overly complicated
    push psw
    push acc
    push AR0
    push AR2
    mov R2, #8 ; sum eight numbers
    mov R0, #40H ; address to start
    mov DPL, #0
    mov DPH, #0
    avg_16_L1: ; sum the numbers
            mov A, DPL
            add A, @R0
            mov DPL, A
            inc R0
            mov A, DPH
            addc A, @R0
            mov DPH, A
            inc R0
            djnz R2, avg_16_L1
    mov R2, #3 ; Loop Counter
    avg_16_L2: ; Divide the [DPH,DPL] by 8
            clr c
            mov A, DPH
            RRC A
            mov DPH, A
            mov A, DPL
            RRC A
            mov DPL, A
            djnz R2, avg_16_L2
    pop AR2
    pop AR0
    pop acc
    pop psw
    Ret

Solution #2:

DSEG at 50H
X: DS 3             ; to make it a 3-byte register

MOV R0, #40H
MOV R1, #8
MOV R2, #3

LJMP L1

COUNT:
INC X+2
DJNE R1, L1
LJMP L2
```

```
L1:
MOV A, X+0
ADD A, @R0
MOV X+0, A
INC R0
MOV A, X+1
ADDC A, @R0
MOV X+1, A
CJNE C, #0, COUNT
DJNE R1, L1

L2:
CLR C
RRC X
DJNE R2, L2

MOV DPL, X+0
MOV DPH, X+1
```