EE 2004
Week 6 Tutorial Solution

1. Creating delay using loops

1.1. Calculate the time delay generated by the following code fragment if [DELAY_L] and [DELAY_H] are set to be 0x00 and 0x04 respectively. Validate your answer using the Stopwatch tool (Debugger → Stopwatch) in MPLAB Sim.

```
DelayLoop:  decfsz  DELAY_L, F, A
            bra DelayLoop
            decfsz  DELAY_H, F, A
            bra DelayLoop
```

1.3. Change the two lines of "`bra DelayLoop`" to "`goto DelayLoop`". Determine the time delay generated for the same initial values of [DELAY_L] and [DELAY_H]. Again, validate your answer using the Stopwatch tool (Debugger → Stopwatch) in MPLAB Sim.

Note: We define the loop formed by the first two lines by the term "DELAY_L loop" in the discussion below. Also, throughout this course, we use the convention [DELAY_L] to denote value contained in register with address DELAY_L.

Sequence of events	***Instr. cycles (1.1 use `bra`)***	***Instr. cycles (1.3 use `goto`)***
1. DELAY_L loop executes for the 1st time	*767*	*768 (last decfsz skips, which takes 3 instr. cycles since `goto` is a 2-word instr.)*
2. Executes the last two lines with [DELAY_H] decrementing from 04 to 03 (decfsz does not skip)	*3*	*3*
3. DELAY_L loop executes for the 2nd time	*767*	*768*
4. Executes the last two lines with [DELAY_H] decrementing from 03 to 02 (decfsz does not skip)	*3*	*3*
5. DELAY_L loop executes for the 3rd time	*767*	*768*
6. Executes the last two lines with [DELAY_H] decrementing from 02 to 01 (decfsz does not skip)	*3*	*3*
7. DELAY_L loop executes for the 4th time	*767*	*768*
8. Executes the last two lines with [DELAY_H] decrementing from 01 to 00 (decfsz skips)	*2*	*3 (last decfsz takes 3 instr. cycle)*
Total	*767*4 + 3*3 + 2 = 3079*	*768*4 + 3*4 = 3084*

For the bra version (1.1):

Suppose the clock frequency = 4MHz. Duration of each clock cycle = 0.25µs.
*Duration of each instruction cycle = 4*0.25us = 1µs.*

*Total time delay generated = 3079*1µs = 3079µs or 3.079ms*

You can derive a mathematical expression to compute the total number of instruction cycles generated in terms of the initial values of [DELAY_H].

Total number of inst. cycles
= 767[DELAY_H] + 3*([DELAY_H]-1) + 2*
= 770[DELAY_H] -1*

1.2. Using the Stopwatch tool, validate that the bra instruction takes two machine cycles to execute. Explain why two machine cycles are required.

See Chapter 3 Part A.

1.4. Use the loop provided in Question 1.1 as a building block, write a subroutine to generate a 8ms delay.

First assume that [DELAY_L] is set to be 0x00 and let x be the initial value of [DELAY_H]. The number of instruction cycles generated by the nested loop in 1.1 is a function of x. Now you want to generate a 8ms delay, which means that you need to generate 8000 instruction cycles using the nested loop. Solve the following equation:

Set 770x-1 = 8000 -> x = 10.4

If [DELAY_H] is set to be d'10', the total number of instruction cycles generated are 770(10)-1 = 7699.

We need 301 instruction cycles more.

If we initialize [DELAY_L] to be 00, the first two lines repeat 256 times. If [DELAY_L] is initialized differently, we would have fewer repetitions.

We can run the DELAY_L loop for 11 times, but in the first run, we are repeating the first two lines for fewer than 256 times to generate a delay of 301-3 = 298 instruction cycles. (Why 298? Because when decrementing [DELAY_H] from 11 to 10, the last two lines of the loop takes 3 inst. cycles. See the table below.)

How do we initialize [DELAY_L] to achieve this?

The first two lines execute for 3 instruction cycles if decfsz does not skip and 2 inst. cycles otherwise. Here is the formula relating the number of instruction cycles generated with the initial [DELAY_L]:

3([DELAY_L]-1) + 2 = 3*[DELAY_L]-1 = 298 → [DELAY_L] = 99.67*

Since [DELAY_L] must be an integer, initialize [DELAY_L] to d'99', thereby generating a delay of 296 instr. cycles. Add 2 nop at the bottom of the loop to make up for the difference.

Putting it all together, you have the following sequence of events:

Sequence of events	***Instr. cycles***
DELAY_L loop executes for the 1st time	*296*
Executes the last two lines with [DELAY_H] decrementing from d'11' to d'10' (decfsz does not skip)	*3*
DELAY_L loop executes for the 2nd time	*767*
Executes the last two lines with [DELAY_H] decrementing from d'10' to d'09' (decfsz does not skip)	*3*
.....	.....
.....	.....
DELAY_L loop executes for the 11th time	*767*
Executes the last two lines with [DELAY_H] decrementing from d'01' to d'00' (decfsz skips)	*2*
2 nops	*2*
Total	*8000*

```
            cblock 0x000
                  DELAY_L
                  DELAY_H
            endc

      org 0x0000
      movlw d'99'
      movwf DELAY_L, A
      movlw d'11'
      movwf DELAY_H, A ;Note that we did not consider the
                       ;time required to run these four
                       ;lines.

      ;;;;;;;; Time = 0
DelayLoop: decfsz  DELAY_L, F, A
       bra DelayLoop
       decfsz  DELAY_H, F, A
       bra DelayLoop

      nop
      nop
      ;;;;;;;; Time = 8 ms

            bra $
```

2. Subroutine and stack

Answer questions below using the following code listing:

```
Program
Memory        LINE            SOURCE
Address


              00012 COUNT equ 0x00
              00013 MyReg equ 0x01
              00014 DELAY_H equ 0x02
              00015 DELAY_L equ 0x03
              00016
              00017 ORG 0x000000
000000        00018 goto Main
              00019 ORG 0x000030
000030        00020 Main:     movlw 0x00
000032        00021           movwf COUNT
000034        00022           movlw 0xFF
000036  D???  00023 Back:     rcall Display
000038 EF?? F??? 00024        goto Back
00003C        00025 Display:  decfsz COUNT, F
00003E        00026           bra Display
000040        00027           movf COUNT, W
000042        00028           movwf PORTB
000044 EC?? F??? 00029        call Delay
000048        00030           return
00004A        00031 Delay:    clrf    DELAY_H
00004C        00032           clrf    DELAY_L
00004E D???   00033 DelayLoop: rcall DelayLoopLow
000050        00034            decfsz  DELAY_H
000052 EF?? F??? 00035              goto    DelayLoop
000056        00036            return
000058        00037 DelayLoopLow: decfsz  DELAY_L
00005A D???   00038               bra DelayLoopLow
00005C        00039               return
```

2.1. Identify the contents of the stack, stack pointer and the TOS register after the execution of Line 23, 29, 30 and 33. Step through the program by pressing F7 and inspect the contents of the "Hardware Stack" (Go to the "View" menu and select "Hardware Stack") and the STKPTR, TOS and PCLAT (Go to the "Watch" window and type STKPTR, TOS and PCLAT under the "Symbol Name" column) to validate your answers.

See next page.

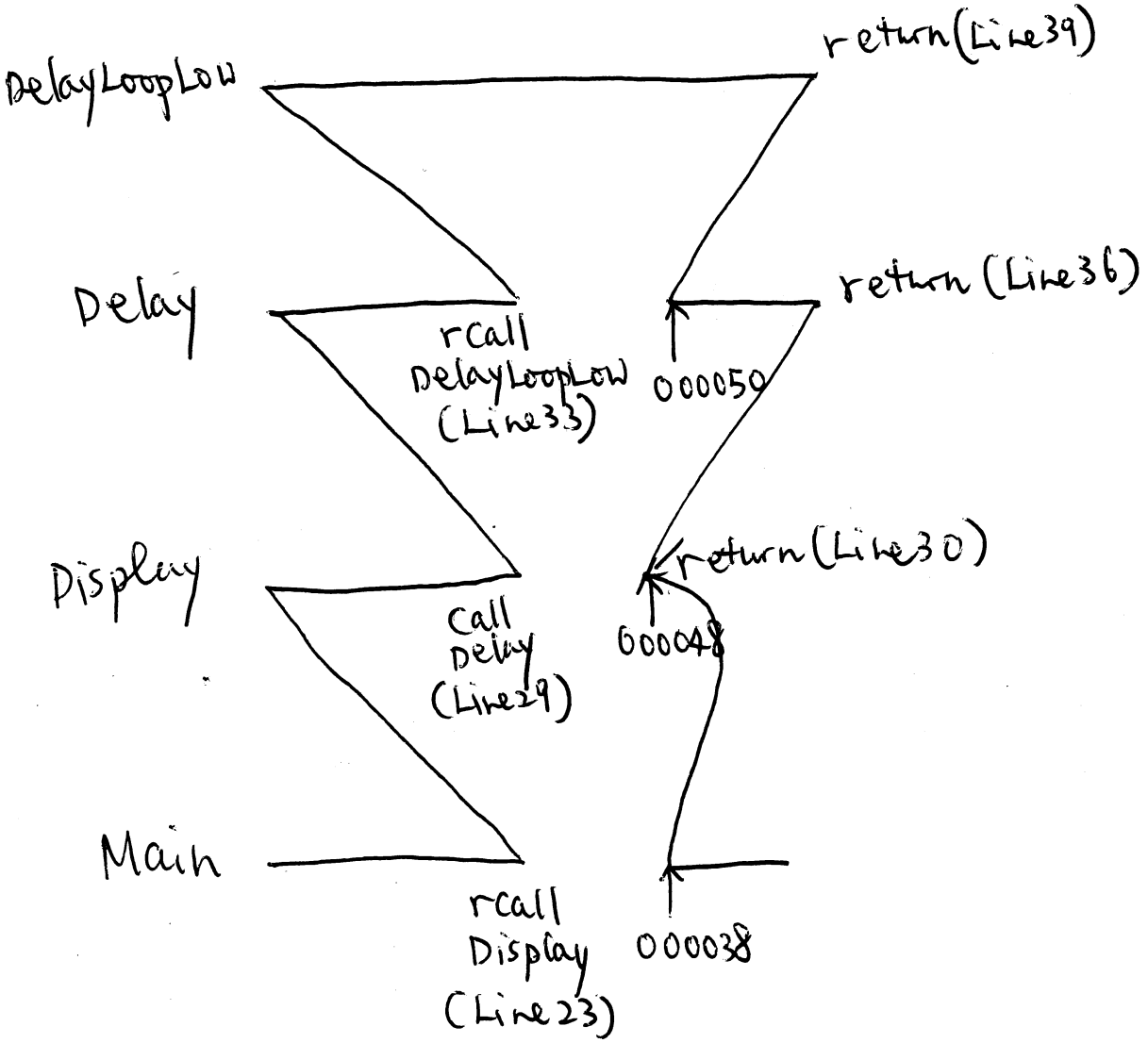
2.2. Determine the opcode of the instructions in Line 23, 24, 29, 33, 35 and 38 and validate your answers by inspecting the .lst file.

You can get the answer from the .lst file you generated.

2.3. What is the difference between the `call` and `rcall` instructions?

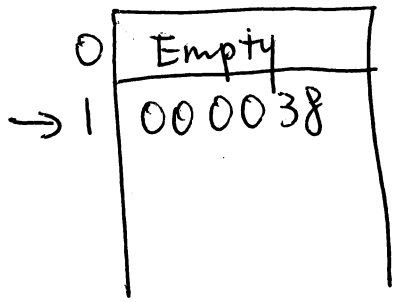
See Chapter 3 Part B.

1


DelayLoopLow
return (Line39)
Delay
return (Line36)
rcall
DelayLoopLow
(Line33)
000050
Display
return (Line30)
call
Delay
(Line29)
000048
Main
rcall
Display
(Line23)
000038

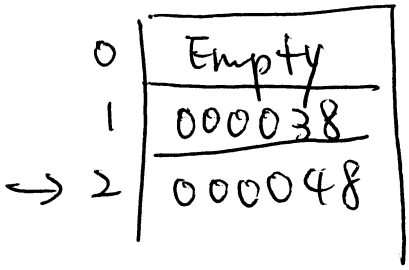
(1)

Line 23



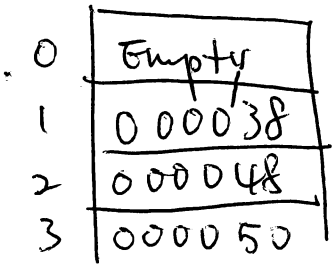
STKPTR = 1
TOS = 000038
PC = 00003C

Line 29



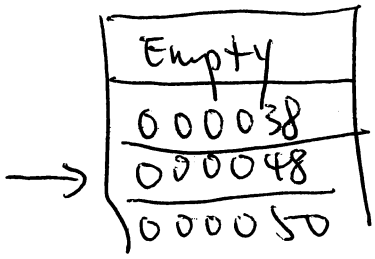
STKPTR = 2
TOS = 000048
PC = 00004A

Line 33



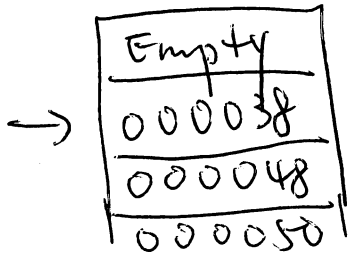
STKPTR = 3
TOS = 000050
PC = 000058

Line 39



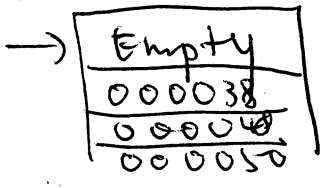
STKPTR = 2
TOS = 000048
PC = 000050

Line 36



STKPTR = 1
TOS = 000038
PC = 000048

Line 30



STKPTR = 0
TOS = 000000
PC = 000038