

RCET 3375 Experiment 4

Delays

Goals: The student will be able to:

- Write programs that use loops to cause real-time delays.
- Calculate real time delays to a high degree of accuracy.
- Use a frequency counter to measure time and frequency.
- Flow chart all programs.
- Use portions of the program as a subroutine.
- Use the monitor delay subroutine.

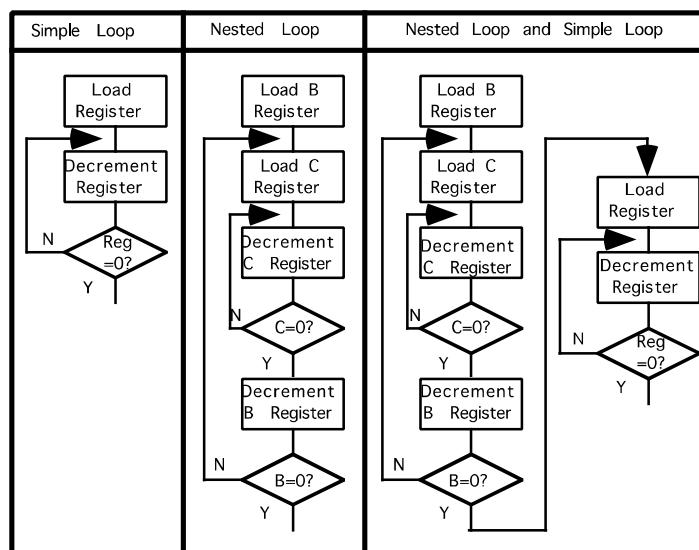
Background Information:

A common application for the microprocessor (μ p) is to control equipment or processes in real time, for example, control of the lights at an intersection where the sequence and timing of the lights is important. There are two ways to cause real time delays in a μ processor based system. One is to load a counter with the desired delay count then wait until the counter signals the μ p that the delay time has expired. Thus, loading a larger number into the delay timer would correspond to a longer delay time. This is called hardware delay and is the most efficient use of the CPU's time but it requires more hardware in the form of a timer IC. The other way is for the μ p to go into a loop of many iterations executing instructions that do nothing but waste time. The more iterations in the loop, the more delay time caused. This method is called software delay and is an inefficient use of CPU time because the CPU must repeatedly execute instructions for the sole purpose of spending time. If the CPU is counting down a software delay, it cannot be doing anything else. The software delay is the main focus of this experiment.

The simple loop shown below is one way of causing software delay in a program. By changing the number loaded into the register, the delay time can be changed. The advantage of this simple loop is that the increments in delay time are quite fine, about 4 μ s. The disadvantage is that the counter is only an eight-bit counter therefore, the maximum delay is only about 1 ms which would not be sufficient for timing a stop light or many other applications.

The next flow chart shown below is the nested loop. This method uses a simple in-line loop "nested" within another loop. This way the instruction within the inner loop will be executed (inner loop count) times (outer loop count) iterations. Or, if 0 is loaded into both the inner and outer loop counters, the CPU would execute the innermost instruction 65536 times. The advantage of nested loops then is that longer delay times may be generated. The disadvantage is that the increments in delay are courser. Loops may be nested as many times as necessary to obtain the desired delay.

To get the advantages of both simple loops and nested loops, follow a nested loop with a simple loop as shown in the last example below. The register values in the nested loop would be adjusted to cause a delay just short of that required. Then, adjust the in-line delay for the finer adjustment. If more accuracy than 4 μ s is required, no-ops may be embedded after the delay to add 1.3 μ s each for very fine delay adjustments.



RCET 3375 Experiment 4

It will be necessary to accurately calculate the execution time of a program. This is the procedure:

1. Determine the time for each Q state by measuring the frequency of the CLK signal of the chip with a frequency counter and calculate the time per Q state. Be accurate.
2. Using the instruction set in the Pic datasheet determine the number of Q states for each instruction and multiply by the number of loops on that instruction or group of instructions.
3. Multiply the total number of Q states by the time for one Q state.

Example:

CLK frequency is 3.073468 MHz

Time for one Q state = $1/3.073468 \text{ MHz} = 325.36535 \text{ ns}$

Program		# of Q States
MOVLW	0X00	4
MOVWF	COUNT	4
DOWNCOUNT		
DECFSZ	COUNT,1	4/8
GOTO	DOWNCOUNT	8

$$\text{Number of Q states} = (((\text{DECFSZ} + \text{GOTO}) 256) - 4 + 4 + 4)$$

$$\text{Number of Q states} = (((4 + 8) 256) - 4 + 4 + 4)$$

$$\text{Number of Q states} = 3076$$

Execution time of above program segment is:

$$\text{Execution Time} = (\text{Number of Q states})(\text{Time per Q state})$$

$$\text{Execution Time} = (3076)(325.36535 \text{ ns}) = 1.00082 \text{ ms}$$

The ability to accurately measure the delay generated by a program is sometimes a problem because one must determine which is the appropriate signal for use in triggering the Counter/Timer. One way to solve this problem is to write to a bit in a port in the program, then use this to trigger the frequency counter. So, if the program is running in a loop the counter will count the time between write of the instruction. If necessary, it is possible to measure the time for any program segment or monitor subroutine by embedding a write instruction immediately preceding and following the segment or subroutine. As the program is run, the first write starts the counter and the second write stops the counter. The counter now displays the elapsed time of the program.

Tasks:

1. Flowchart all programs before writing them and enter the flowcharts in your lab manual immediately before the programs. (Flow chart all programs from now on.)
2. Calculate and write a program to display a 5 in the display for .5 seconds. Then display a 0 in display for .5 seconds and repeat. Use in-line code to cause the delays (do not access the delays as subroutines). Use 8-bit registers only to count down the delays. The delay should be accurate to 1 μs . Include the assembly and machine code for this step.
3. Measure the time that each number is displayed in each program and note in your lab book. ie. show the measured time and the calculated time for each half of each program. (Be accurate)
4. Put the delay loops into a subroutine that is called from the main program. And repeat step 3. Be able to describe how the PC and the stack work in this step.