Louis Coyle

ECE300

1/19/14

HW 02

1. The data word size of the PIC18F452 is 8-bits. The instruction word size is 16-bits. No, all instructions do not execute in a single instruction cycle. For example, MOVFF takes two instruction cycles and two instruction words, as the first contains the opcode and the source and the second contains the 12-bit destination address. Also, conditional instructions and any instruction that modifies the PC (including BRA, GOTO) can take up to 3 cycles. For these instructions, the next pipelined instruction must be flushed with a NOP (in the case of a conditional, only if it evaluates true). If the flushed instruction is a 2-word instruction, it will need to flush with another NOP, making a maximum of 3 cycles.
2. MOVFF is a non-atomic, two cycle, two instruction word instruction. If the MOVFF operation gets interrupted while attempting to modify an INTCON register then the behavior is undefined. For example, if a change on RB occurs when attempting to clear the RBIE flag using MOVFF, then the interrupt could still occur after the two-cycle instruction is completed, specifically if the change in RB occurs before the data is written in the execute portion of MOVFF’s second instruction cycle. This can be avoided simply by using the BSF instruction, which is atomic.
3. MOVFF lets you access the entire memory space. Although MOVF is a single word instruction, if you want to access memory outside of the access bank with MOVF, you must first set the BSR (bank select register) with MOVLB to select the 256-byte region in RAM that you need to access and set a = 1 in the MOVF instruction. This effectively makes MOVF take 2 cycles in the case that the variables in your loop are outside of the access bank or are in different bank select registers. For this reason, MOVFF could be advantageous, especially in the case that the loop variables are scattered throughout the RAM, as it allows you to ignore the BSR and access anywhere in the 12-bit address space in one instruction, increasing code density.
4. Following is my disassembly for the code snippet:

MOVFF 0, j (1 cycle)

LOOP:

RLNCF j (1 cycle)

MOVLW 0x01 (1 cycle)

BTFSS PORTB, 0 (1 cycle if == 0, 2 cycles if == 1)

GOTO ELSE (2 cycles)

IORWF j, 1, 0 (1 cycle)

GOTO LOOP (2 cycles)

ELSE

MOVLW 0xFE (1 cycle)

ANDWF j, 1, 0 (1 cycle)

GOTO LOOP (2 cycles)

With the above disassembly for the code snippet, I have found that the maximum amount of cycles in between signal reads at the BTFSS instruction is 8. Using the HS+PLL oscillator configuration allows the PIC18F452 to have a 40MHz clock, meaning that 1 cycle takes 25 nanoseconds to complete. Therefore, if there are 8 cycles between signal reads, then there are 200 nanoseconds between reads of RB0. So, RB0 can’t change any faster than every 200ns, or pulses of the signal may be missed. In order to ensure that pulses will not be missed, the period of the signal must be >= 400ns, and so the frequency must be <= 2.5MHz. Including this code in another program would mean additional cycles in between signal reads (at the very least two more cycles to GOTO or BRA to this code), increasing the amount of time between these reads. Therefore, the signal wouldn’t be able to change this fast, or pulses may be missed.

1. If one uses the RLCF instruction rather than RLNCF instruction, the bit that is rotated out will be saved in the carry flag of the status register. One could then look in the status register for that 9th bit.
2. Hardware exceptions can be implemented on the PIC18F452 using its interrupt features. For example, one can trigger an interrupt on a change of pin RB0, causing program execution to switch to the interrupt vector at 0x08 and for appropriate code for this exception to be executed. Similarly, one could make hardware breakpoints by manually setting the RBIF flag before or after where one would like to have a breakpoint, causing a break in program execution once this flag is set. From the interrupt vector location, one could view the status of the memory and the saved WREG value, just as with a typical hardware breakpoint. The number of different exception types is limited by the number of interrupts that the PIC is capable of (the number of different interrupt flags that can be set or cleared). On top of this, the interrupts must be unused in their typical facility. The PIC18F452 in particular, has interrupt flag bits for RB, TIMR0, INT0, INT1, INT2, PSP, AD, RC, TX, SSP, CCP1, TMR, TMR1, EE, BCL, LVD, TMR3, and CCP2, all of which will trigger an interrupt if their necessary interrupt enable bits are set. This gives 18 possible interrupt sources, minus the number of sources already in use by the program that can be used as different exception types.