KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

CAMPUS GROUP T LEUVEN

Microcontrollers LAB 3

SPI communication using ASSEMBLY

Coordinator: L. Geurts

Cooperator: T. Stas

.

Academic Year 2019-2020

Programming in Assembly

While C programming language has the advantage of being readable and easily understandable, it suffers from some shortcomings including slow speed and inefficient implementations. Assembly code, as a low-level language, allows PIC programmers to have more control over how an algorithm is implemented, leading often to faster and more efficient implementations.

Assembly instructions

Following table gives a brief overview of the instruction set of a PIC18F25k50 microcontroller. More information and examples on every instruction and its operands can be found in the datasheet from the microcontroller under chapter 27.

Mnemonic, Operands		Description
BYTE-ORIENTED OPERATIONS		
ADDWF	f, d, a	Add WREG and f
ADDWFC	f, d, a	Add WREG and Carry bit to f
ANDWF	f, d, a	AND WREG with f
CLRF	f, a	Clear f
COMF	f, d, a	Complement f
CPFSEQ	f, a	Compare f with WREG, skip =
CPFSGT	f, a	Compare f with WREG, skip >
CPFSLT	f, a	Compare f with WREG, skip <
DECF	f, d, a	Decrement f
DECFSZ	f, d, a	Decrement f, Skip if 0
DCFSNZ	f, d, a	Decrement f, Skip if Not 0
INCF	f, d, a	Increment f
INCFSZ	f, d, a	Increment f, Skip if 0
INFSNZ	f, d, a	Increment f, Skip if Not 0
IORWF	f, d, a	Inclusive OR WREG with f
MOVF	f, d, a	Move f
MOVFF	f_s, f_d	Move f_s (source) to 1st word f_d (destination) 2nd word
MOVWF	f, a	Move WREG to f
MULWF	f, a	Multiply WREG with f
NEGF	f, a	Negate f
RLCF	f, d, a	Rotate Left f through Carry
RLNCF	f, d, a	Rotate Left f (No Carry)
RRCF	f, d, a	Rotate Right f through Carry
RRNCF	f, d, a	Rotate Right f (No Carry)
SETF	f, a	Set f
SUBFWB	f, d, a	Subtract f from WREG with borrow
SUBWF	f, d, a	Subtract WREG from f
SUBWFB	f, d, a	Subtract WREG from f with borrow
SWAPF	f, d, a	Swap nibbles in f
TSTFSZ	f, a	Test f, skip if 0
XORWF	f, d, a	Exclusive OR WREG with f

Mnemonic, Operands		Description
BIT-ORIENTED OPERATIONS		
BCF	f, b, a	Bit Clear f
BSF	f, b, a	Bit Set f
BTFSC	f, b, a	Bit Test f, Skip if Clear
BTFSS	f, b, a	Bit Test f, Skip if Set
BTG	f, d, a	Bit Toggle f
CONTROL OPERATIONS		
BC	n	Branch if Carry
BN	n	Branch if Negative
BNC	n	Branch if Not Carry
BNN	n	Branch if Not Negative
BNOV	n	Branch if Not Overflow
BNZ	n	Branch if Not Zero
BOV	n	Branch if Overflow
BRA	n	Branch Unconditionally
BZ	n	Branch if Zero
CALL	n, s	Call subroutine 1st word 2nd word
CLRWDT	—	Clear Watchdog Timer
DAW	—	Decimal Adjust WREG
GOTO	n	Go to address 1st word 2nd word
NOP	—	No Operation
NOP	—	No Operation
POP	—	Pop top of return stack (TOS)
PUSH	—	Push top of return stack (TOS)
RCALL	n	Relative Call
RESET		Software device Reset
RETFIE	s	Return from interrupt enable
RETLW	k	Return with literal in WREG
RETURN	s	Return from Subroutine
SLEEP	—	Go into Standby mode

Mnemonic, Operands		Description
LITERAL OPERATIONS		
ADDLW	k	Add literal and WREG
ANDLW	k	AND literal with WREG
IORLW	k	Inclusive OR literal with WREG
LFSR	f, k	Move literal (12-bit) 2nd word to FSR(f) 1st word
MOVLB	k	Move literal to BSR<3:0>
MOVLW	k	Move literal to WREG
MULLW	k	Multiply literal with WREG
RETLW	k	Return with literal in WREG
SUBLW	k	Subtract WREG from literal
XORLW	k	Exclusive OR literal with WREG

Table 1: PIC18F2550 instruction set

Environment

Because in this session, you will use assembly language in MPLABX, you will have to use an assembly compiler.

Start the environment, then click on File >> New Project to create a new project. Unlike when using C language, you need to choose MPASM as compiler.

There is no need to configure the properties of the project any more, unlike the XC8 complier, the MPASM compile is quite handy.

The next step is to create an .asm file in your project. There is an assembly template on Toledo. This template shows you how to include header files, initialize registers, and handle interrupts. You may download this and add it to your project folder.

Building and Debugging

After importing an assembly file into your project, you may run or debug the code (no need for linker file). The building and debugging procedures are the same as that of a C project. You may use the same debug tools (*Variables* and *Stimulus*) as well.

NOTE: When using debug tools, your reset, interrupt, and start vectors should point to 0x000, 0x008, 0x018, and 0x020 respectively. Before building a HEX file to download onto the board, do not forget to change the reset, interrupt, and start vectors back to 0x1000, 0x1008, 0x1018, and 0x1020. This ensures that your code skips the bootloader and that all parts are executed properly.

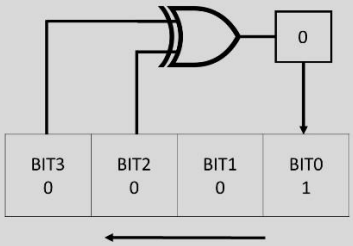
File Register

Unlike the declaration of a variable and it's data type in C, when you want to use a variable in assembly code you need to give an address location to the File Register of your microcontroller. When in simulation mode, you can have a look at the values stored in the File Register.

Select Window → Simulator → File Registers

Assignment:

Have a look at the template program on toledo "AsmTemplate.asm". This example shows an 8-bit linear feedback shift register (LFSR) using EXOR functions. This can be done using the following algorithm:



RC0 is used as enable signal, the result will be displayed on PORTB.
Try to predict the first 3 values the LFSR will generate. Try to simulate the template program and check if your prediction was correct.

Try to download the template program. Test the functionality on your microcontroller board.
If the program runs at full speed, it is not possible for human eye to see the LED change. Create and call the delay function in your assembly program:

```
CALL    delay1          ; call the delay loop
```

SPI

The Serial Peripheral Interface (SPI) is a three-pin, synchronous communication protocol that allows a master device to initiate communication with and transfer data to and from a slave device. It is implemented in the PIC by a hardware module called the Synchronous Serial Port or the Master Synchronous Serial Port (MSSP). It allows for serial communication between two or more devices at a high speed and is reasonably easy to implement.

To understand the concept of SPI, you will connect two PIC boards together and communicate between them using this protocol.

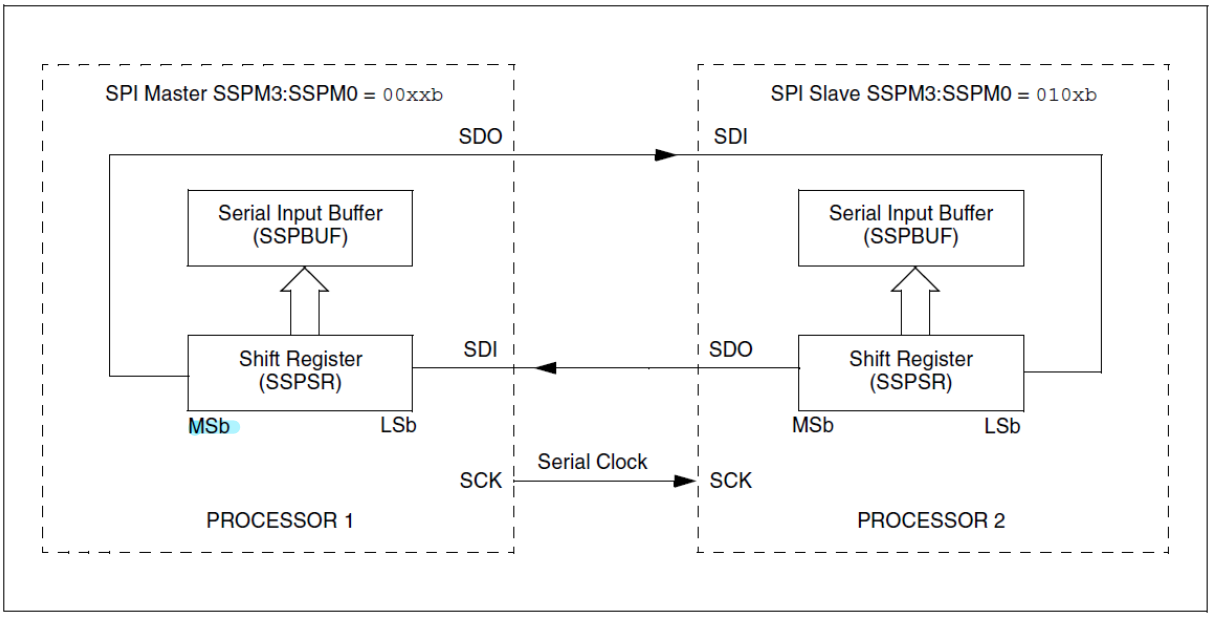


Figure 1: SPI Master/Slave connection with two PIC microcontrollers

The MSSP module consists of a transmit/receive shift register (SSPSR) and a buffer register (SSP1BUF). The SSPSR shifts the data in and out of the device, MSB first. The SSPBUF holds the data that was written to the SSPSR until the received data is ready. Once the eight bits of data have been received, the byte is moved to SSP1BUF register. Then, the Buffer Full detect bit, BF (SSP1STAT<0>) and the interrupt flag bit, SSPIF, are set.

The SPI Master clock should be driven by TMR2, with Timer2 period register PR2 set to 0xFF. When TMR2 is equal to PR2, it will generate a clock pulse for the master SPI.

For more information on the MSSP module, it's configuration and example code, have a look in the PIC18F25k50 data sheet chapter 16!

Assignment "COUNTER over SPI":

1. **Counter implementation**
   Start by implementing a simple 8-bit binary counter on PORTB. Use RA4 to toggle UP/DOWN mode. If the counter program runs at full speed, it is not possible for human eye to see the LED change. Create and call the delay function in your assembly program:
   ```
   CALL    delay1        ; call the delay loop
   ```
2. **One PIC board**
   Configure your PIC as a Master SPI. Transmit the counter values via the SPI Master. Connect SDO directly to SDI of the same board. Display the received 8-bit result on 8 LED's. Display the SPI data line on an oscilloscope. Pay attention to the pinout of your microcontroller (SDI and SDO).
3. **Two PIC board**
   Connect your master PIC to another PIC that is configured as an SPI slave. The master is generating the counter values, the slave receives those and shows them on 8 LED's connected to PORTB.

Assignment "Master as AD convertor":

In the previous lab session we configured the AD convertor to take samples on an analog input. Try to implement this code in assembly instructions. Connect a potentiometer to the analog input of the SPI master, sample this input and send the result to the SPI slave. The slave displays the received value on PORTB.

Assignment "Master as AD convertor – Slave as PWM generator":

In the previous lab session we configured the CCP module to generate a PWM signal. Try to implement this code in assembly instructions on the SPI slave. Use the AD result that you receive from the SPI slave from previous assignment to make an adjustable Duty Cycle.