

Microcontrollers LAB 3

SPI communication using ASSEMBLY

Coordinator: L. Geurts

Cooperator: T. Stas

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 PIC18F2550 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	Mnemonic, Operands	Description
BYTE-ORIENTED OPERATIONS		BIT-ORIENTED OPERATIONS	
ADDWF f, d, a	Add WREG and f	BCF f, b, a	Bit Clear f
ADDWFC f, d, a	Add WREG and Carry bit to f	BSF f, b, a	Bit Set f
ANDWF f, d, a	AND WREG with f	BTFSC f, b, a	Bit Test f, Skip if Clear
CLRF f, a	Clear f	BTFSS f, b, a	Bit Test f, Skip if Set
COMF f, d, a	Complement f	BTG f, d, a	Bit Toggle f
CPFSEQ f, a	Compare f with WREG, skip =	CONTROL OPERATIONS	
CPFSGT f, a	Compare f with WREG, skip >	BC n	Branch if Carry
CPFSLT f, a	Compare f with WREG, skip <	BN n	Branch if Negative
DECf f, d, a	Decrement f	BNC n	Branch if Not Carry
DECFSZ f, d, a	Decrement f, Skip if 0	BNN n	Branch if Not Negative
DCFSNZ f, d, a	Decrement f, Skip if Not 0	BNOV n	Branch if Not Overflow
INCF f, d, a	Increment f	BNZ n	Branch if Not Zero
INCFSZ f, d, a	Increment f, Skip if 0	BOV n	Branch if Overflow
INFSNZ f, d, a	Increment f, Skip if Not 0	BRA n	Branch Unconditionally
IORWF f, d, a	Inclusive OR WREG with f	BZ n	Branch if Zero
MOVF f, d, a	Move f	CALL n, s	Call subroutine 1st word 2nd word
MOVFF f _s , f _d	Move f _s (source) to f _d (destination) 2nd word	CLRWDT —	Clear Watchdog Timer
MOVWF f, a	Move WREG to f	DAW —	Decimal Adjust WREG
MULWF f, a	Multiply WREG with f	GOTO n	Go to address 1st word 2nd word
NEGF f, a	Negate f	NOP —	No Operation
RLCF f, d, a	Rotate Left f through Carry	NOP —	No Operation
RLNCF f, d, a	Rotate Left f (No Carry)	POP —	Pop top of return stack (TOS)
RRCF f, d, a	Rotate Right f through Carry	PUSH —	Push top of return stack (TOS)
RRNCF f, d, a	Rotate Right f (No Carry)	RCALL n	Relative Call
SETF f, a	Set f	RESET	Software device Reset
SUBFWB f, d, a	Subtract f from WREG with borrow	RETFIE s	Return from interrupt enable
SUBWF f, d, a	Subtract WREG from f	RETLW k	Return with literal in WREG
SUBWFB f, d, a	Subtract WREG from f with borrow	RETURN s	Return from Subroutine
SWAPF f, d, a	Swap nibbles in f	SLEEP —	Go into Standby mode
TSTFSZ f, a	Test f, skip if 0		
XORWF f, d, a	Exclusive OR WREG with f		

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 compiler, 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.

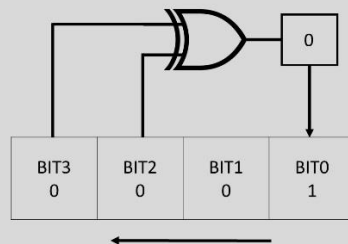
File Register

Unlike the declaration of a variable and its 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.

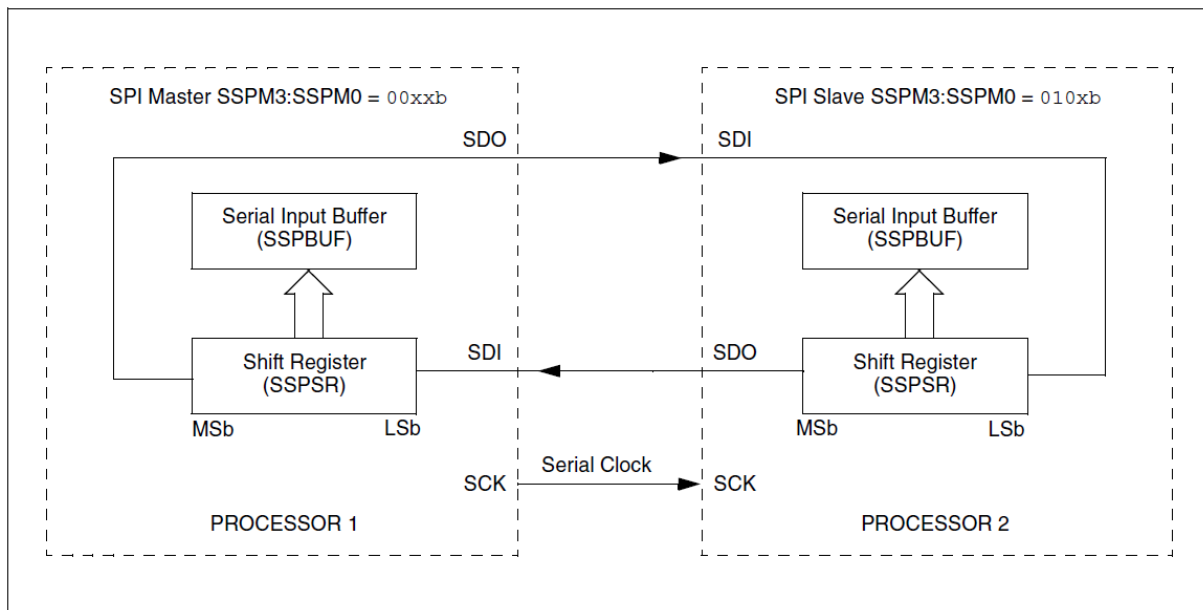


Figure 1: SPI Master/Slave connection with two PIC microcontrollers

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, its configuration and example code, have a look in the PIC18F25k50 data sheet chapter 16!



TABLE 16-1: REGISTERS ASSOCIATED WITH SPI OPERATION

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on page
ANSELA	—	—	ANSA5	—	ANSA3	ANSA2	ANSA1	ANSA0	147
ANSELB	—	—	ANSB5	ANSB4	ANSB3	ANSB2	ANSB1	ANSB0	148
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	IOCIE	TMR0IF	INT0IF	IOCIF	114
IPR1	ACTIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	123
PIE1	ACTIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	120
PIR1	ACTIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	117
PMD1	—	MSSPMD	CTMUMD	CMP2MD	CMP1MD	ADCMD	CCP2MD	CCP1MD	62
SSP1BUF	SSP1 Receive Buffer/Transmit Register								—
SSP1CON1	WCOL	SSPOV	SSPEN	CKP	SSPM<3:0>				252
SSP1CON3	ACKTIM	PCIE	SCIE	BOEN	SDAHT	SBCDE	AHEN	DHEN	255
SSP1STAT	SMP	CKE	D/A	P	S	R/W	UA	BF	251
TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	149
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	149
TRISC	TRISC7	TRISC6	—	—	—	TRISC2	TRISC1	TRISC0	149

Legend: Shaded bits are not used by the MSSP in SPI mode.

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.

TABLE 16-1: REGISTERS ASSOCIATED WITH SPI OPERATION

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on page
ANSELA	—	—	ANSA5	—	ANSA3	ANSA2	ANSA1	ANSA0	147
ANSELB	—	—	ANSB5	ANSB4	ANSB3	ANSB2	ANSB1	ANSB0	148
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	IOCIE	TMR0IF	INT0IF	IOCIF	114
IPR1	ACTIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	123
PIE1	ACTIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	120
PIR1	ACTIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	117
PMD1	—	MSSPMD	CTMUMD	CMP2MD	CMP1MD	ADCMD	CCP2MD	CCP1MD	62
SSP1BUF	SSP1 Receive Buffer/Transmit Register								—
SSP1CON1	WCOL	SSPOV	SSPEN	CKP	SSPM<3:0>				252
SSP1CON3	ACKTIM	PCIE	SCIE	BOEN	SDAHT	SBCDE	AHEN	DHEN	255
SSP1STAT	SMP	CKE	D/A	P	S	R/W	UA	BF	251
TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	149
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	149
TRISC	TRISC7	TRISC6	—	—	—	TRISC2	TRISC1	TRISC0	149