ECE IoT 505L – Embedded Programming – Lab Exercises

|  |  |  |
| --- | --- | --- |
| **S.No** | **Experiments** | **Tools used / Remarks** |
| 1. | 64 bit Addition – Assembly | Keil uVision IDE |
| 1. | Blink the LED – C Program |  |
| 2. | Read ADC output and display in Serial Console (UART) | Keil uVision IDE |
| 3. | I2C EEPROM Interface – C Program |
| 4. | Real Time Clock – C Application |
| 5. | Digital to Analog Converter (DAC) – C Program |
| 6. |  |
|  |  |
|  |  |  |
|  |  |
|  |  |

**Exercise 1**: Write an assembly program to add two 64 bit numbers in memory.

**Objective**: To write an assembly language program to add the two 64 bit numbers stored in the data memory of LPC1768 (ARM Cortex M3 Microcontroller from NXP).

**Let us consider two 64 bit numbers**

**X = 0x1234567812345678**

**Y = 0x2345678023456789 +**

**--------------------------------------**

**Z =** 0x3579BE013579BE01

**---------------------------------------**

**Update the memory 0x10000100 with the Least significant word of ”X” (0x12345678) (in Little Endian Order)**

**0x10000100 – 0x78**

**0x10000101 – 0x56**

**0x10000102 – 0x34**

**0x10000103 – 0x12**

**Update the memory 0x10000104 with the Most significant word of ”X” (0x12345678) (in Little Endian Order)**

**0x10000104 – 0x78**

**0x10000105 – 0x56**

**0x10000106 – 0x34**

**0x10000107 – 0x12**

**Update the memory 0x10000108 with the Least significant word of ”Y” (0x23456789) (in Little Endian Order)**

**0x10000108 – 0x89**

**0x10000109 – 0x67**

**0x1000010A – 0x45**

**0x1000010B – 0x23**

**Update the memory 0x1000010C with the Most significant word of ”Y” (0x23456789) (in Little Endian Order)**

**0x1000010C – 0x78**

**0x1000010D – 0x56**

**0x1000010E – 0x34**

**0x1000010F – 0x12**

**Algorithm:**

1. Move the address 0x10000100 to R0.
2. Load the 32 bit data pointed by R0 to R1
3. Move the address 0x10000104 to R0.
4. Load the 32 bit data pointed by R0 to R2
5. Move the address 0x10000108 to R0.
6. Load the 32 bit data pointed by R0 to R3
7. Move the address 0x1000010C to R0.
8. Load the 32 bit data pointed by R0 to R4
9. Add R1 and R3 and put the result in R1
10. Add R2 and R4 and put the result in R2
11. Store R1 at the memory location 0x10002108
12. Store R2 at the memory location 0x1000210C

**Assembly Program:**

**AREA mydata, DATA**

**X0 EQU 0x10000100**

**X1 EQU 0x10000104**

**Y0 EQU 0x10000108**

**Y1 EQU 0x1000010C**

**RE0 EQU 0x10000110**

**RE1 EQU 0x10000114**

**AREA gpio, CODE, READONLY, ALIGN=2**

**EXPORT \_\_main**

**ENTRY**

**\_\_main**

**LDR R0, =X0 ;Get the address of Least significant word(X0)'s address**

**LDR R1, [R0]**

**LDR R0, =X1 ;Get the address of Most significant word(X1)'s address**

**LDR R2, [R0]**

**LDR R0, =Y0 ;Get the address of Least significant word(Y0)'s address**

**LDR R3, [R0]**

**LDR R0, =Y1 ;Get the address of Most significant word(Y1)'s address**

**LDR R4, [R0]**

**ADDS R1, R1,R3**

**ADC R2, R2, R4**

**LDR R0, =RE0**

**STR R1, [R0]**

**LDR R0, =RE1**

**STR R2, [R0]**

**LBL B LBL**

**END**

**Output:**

Figure 1. Watch window showing PORTB value as 0x01

A screenshot of a computer

Description automatically generated

Figure 1. Memory Window View

**Work for Students: (To be submitted on or before 29/10/21)**

1. Modify the program to include carry.
2. Modify the program to do subtraction of two 64 bit numbers with borrow.
3. Write a program to Multiply two 64 bit numbers and get the 64 bit results

**Conclusion:** Thus, an assembly program is written to add two 64 bit numbers in memory and put the result back to memory.

Project folder: Arithmetic

**Exercise 2**: Write an assembly program to add two 8 bit numbers and show the sum and carry in the file registers for an PIC18 microcontroller with MPLAB X simulator.

**Objective**: To write an assembly language program to add two 8 bit numbers and show the sum and carry in the file registers for an PIC18 microcontroller with MPLAB X simulator.

**Algorithm:**

1. Take the number in the W (working ) register.
2. Add another number with the content of W Register
3. The SUM contained in W Register can be moved to General purpose File Register
4. Check the carry flag on the STATUS register.
5. If it is 1, then set memory location for Carry as 1
6. Go to step 6 (infinite loop)

**Assembly Program:**

**#include <xc.inc>**

**SUM set 10H**

**CAR set 11H**

**PSECT res\_vect, class=CODE, reloc=2**

**res\_vect:**

**goto main**

**PSECT code**

**main:**

**movlw 0x00**

**movwf CAR,a**

**MOVLW 95H ; WREG = 25H**

**;ADDLW 0x15 ; Add 34H to WREG**

**;ADDLW 21**

**ADDLW 10010101B**

**MOVWF SUM,a ; Save the added sum in W reg to loc 10H**

**BNC HERE**

**movlw 0x01**

**movwf CAR, a**

**HERE: GOTO HERE ; Infinite loop here**

**END res\_vect**

**Output:**

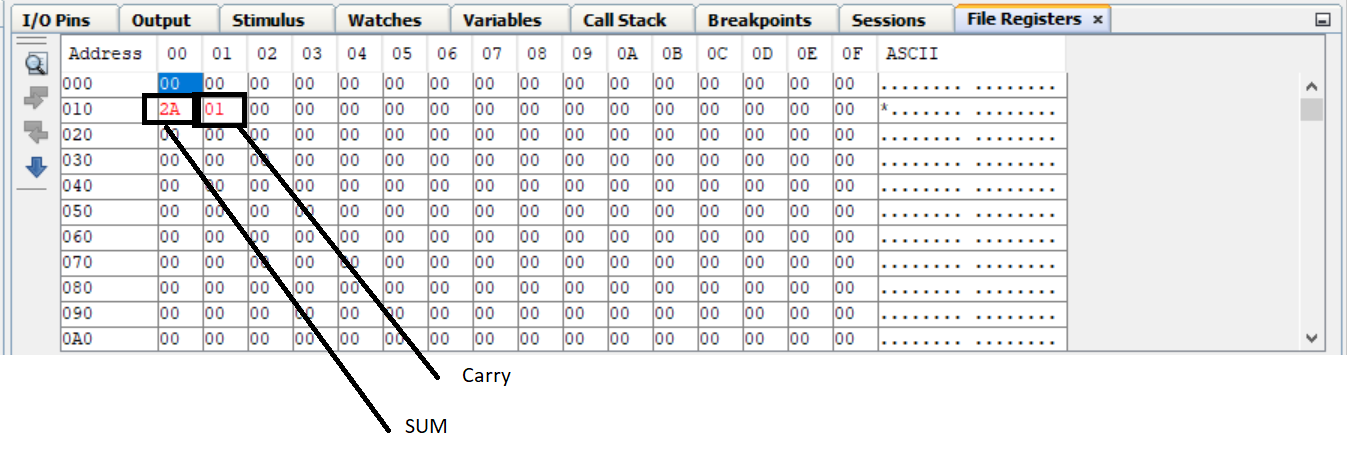


Figure 1. File Register window showing SUM and Carry

**Note: The configuration bits are not included which has to come after the inclusion of header files**

**Conclusion:** Thus, an assembly program is written to add two numbers and store the sum and carry at the file registers. And the same has been simulated using MPLAB X simulator and the snaps of the output File Register window are given.

Project: addition\_with\_carry

**Exercise 3**: Write an assembly program to subtract two 8 bit numbers and show the subtraction result and borrow in the file registers for an PIC18 microcontroller with MPLAB X simulator.

**Objective**: To write an assembly language program to subtract two 8 bit numbers and show the subtraction result and borrow in the file registers for an PIC18 microcontroller with MPLAB X simulator.

**Algorithm:**

1. Take the number in the W (working ) register.
2. Subtract the content of W Register from another number
3. The difference will be stored in W Register can be moved to General purpose File Register
4. Check the borrow flag on the STATUS register.
5. If it is 1, then set memory location for borrow as 1
6. Go to step 6 (infinite loop)

**Assembly Program:**

**#include <xc.inc>**

**RESLT set 10H**

**BORW set 11H**

**PSECT res\_vect, class=CODE, reloc=2**

**res\_vect:**

**goto main**

**PSECT code**

**main:**

**movlw 0x56**

**sublw 0x89**

**;BANKSEL SUM ; need to work on this**

**MOVWF RESLT,a ; Save the added sum in W reg to loc 10H**

**movlw 0x01**

**btfss STATUS,0**

**movwf BORW,a**

**HERE: GOTO HERE ; Infinite loop here**

**END res\_vect**

**Output:**

Graphical user interface, application, table, Excel

Description automatically generated

Figure 1. File Register window showing difference and Borrow

**Note: The configuration bits are not included which has to come after the inclusion of header files**

**Conclusion:** Thus, an assembly program is written to subtract two numbers and store the difference and borrow at the file registers. And the same has been simulated using MPLAB X simulator and the snaps of the output File Register window are given.

Project: Subtraction

**Exercise 4**: Write an assembly program to transmit a character from UART continuously and show that received characters at UART window of MPLAB X Simulator. Set the baud rate as 9600. Assume that the PIC18 has FOSC as 16MHz

**Objective**: To write an assembly language program to to transmit a character from UART continuously and show the characters at UART window of MPLAB X Simulator.

**Algorithm:**

1. Calculate the SPBRG for the baud rate of 9600. [ FOSC/(64\*(SPBRG+1)) ]
2. Load the calculated value (25) to SPBRG.
3. Configure TXSTA register. Make TXEN bit and BRGH bit as 1.
4. Enable UART reception on RCSTA.
5. Transmit 0x41 (ASCII value for A) and check whether the transmission is completed (Polling TRMT bit)
6. Go to step 5

**Assembly Program:**

**#include <xc.inc>**

**COUNT1 equ 21h ; These two values relate to the delay between flashes**

**COUNT2 equ 22h ;**

**PSECT res\_vect, class=CODE, reloc=2**

**res\_vect:**

**goto main**

**PSECT code**

**main:**

**;---CONFIGURE SPBRG FOR DESIRED BAUD RATE**

**movlw 25 ;baud rate = 9600bps**

**movwf SPBRG,a ;at 4MHZ**

**;---CONFIGURE TXSTA**

**movlw 00100100B**

**movwf TXSTA,a**

**;Configures TXSTA as 8 bit transmission, transmit enabled, async mode, high speed baud rate**

**movlw 10000000B**

**movwf RCSTA,a ;enable serial port receive**

**lp: movlw 0x41**

**movwf TXREG,a ;place the A character to TXREG**

**wthere:**

**btfss TRMT ;check if TRMT is empty**

**goto wthere ;if not, check again**

**goto lp**

**end res\_vect**

**Output:**

Text

Description automatically generated with low confidence

Figure 1. UART window showing the received characters

**Note: The configuration bits are not included which has to come after the inclusion of header files**

**Conclusion:** Thus, an assembly program is written to transmit a character continuously. And the same has been simulated using MPLAB X simulator and the snaps of the output File Register window are given.

Project: UART\_Tx

**Exercise 5**: Write a PIC assembly program to use the timer interrupt-based delay to change the bit pattern on PORTC

**Objective**: To write an assembly language program to change the bit pattern on PORTC. The delay between the change of bit pattern is done by using the timer interrupt.

**Algorithm:**

1. Set Global Interrupt Enable (GIE) and Peripheral Interrupt Enable (PIE) bits.
2. Set Timer1 Interrupt Enable bit.
3. Configure the Timer 1 prescaler.
4. Load appropriate value on Timer1 such that timer1 overflows after 50ms.
5. Start the Timer1.
6. Go to step 6.

**Timer1 ISR:**

1. Clear the Timer 1 Interrupt flag
2. At Timer1 ISR, decrement the counter (A2) which is initialized with the value of 100.
3. When the counter reaches 0, check the LSB of A4 and selects the bit pattern either 0xAA or 0xCC based on A4.0 bit value. If the count becomes 0, reload the counter with value 100.
4. Reload the Timer1 registers with the appropriate value for getting overflow after 50ms
5. Return from ISR

**Assembly Program:**

**#include <xc.inc>**

**A1 equ 0x20**

**A2 equ 0x21**

**A3 equ 0x22**

**A4 equ 0x23**

**PSECT res\_vect, class=code, reloc=2**

**res\_vect:**

**goto main**

**PSECT code**

**main:**

**CLRF STATUS,a**

**CLRF A4,a**

**CLRW**

**CLRF TMR1L,a**

**CLRF TMR1H,a**

**CLRF INTCON,a**

**CLRF PORTC,a**

**CLRF PORTB,a**

**MOVLW 0X64**

**MOVWF A1,a**

**MOVWF A2,a**

**BSF A4,0,a**

**BSF STATUS,5,a**

**CLRF TRISC,a**

**CLRF TRISB,a**

**BSF INTCON,7,a ;SET GLOBAL INTERRUPTS ENABLE BIT**

**BSF INTCON,6,a ;SET PERIPHERAL INTERRUPT ENABLE BIT**

**BSF PIE1,0,a ;TIMER1 INTERRUPT ENABLE BIT**

**BCF STATUS,5,a ;BANK 0 SELECTION**

**MOVLW 0XAA**

**MOVWF PORTC,a**

**MOVLW 0X10**

**MOVWF T1CON,a ;1:2 PRESCALE VALUE, TIMER1 OSC ENABLED**

**MOVLW 0X85 ;TIMER1 IS LOADED WITH A VALUE**

**MOVWF TMR1L,a ;EQUIVALENT TO 50ms DELAY**

**MOVLW 0X6D**

**MOVWF TMR1H,a**

**BSF T1CON,0,a**

**LOOP: GOTO LOOP**

**PSECT isr\_vect, class=code, reloc=2**

**goto isr**

**ORG 0x100**

**isr:**

**BCF PIR1,0 ; CLEAR THE INTERRUPT FLAG**

**DECFSZ A1,1**

**GOTO EXIT**

**MOVF A2,0 ;MOVE A2 CONTENT TO W**

**MOVWF A1,a**

**BTFSS A4,0,a**

**GOTO SEC**

**BCF A4,0,a ;IF A4 IS 0X01 DISPLAY 0XCC AT PORTC**

**MOVLW 0XCC**

**MOVWF PORTC, a**

**GOTO EXIT**

**SEC: BSF A4,0,a ;IF A4 IS 0X00 DISPLAY 0XAA AT PORTC**

**MOVLW 0XAA**

**MOVWF PORTC,a**

**EXIT: MOVLW 0X85**

**MOVWF TMR1L,a ;RELOAD THE TIMER**

**MOVLW 0X6D**

**MOVWF TMR1H,a**

**RETFIE**

**END res\_vect**

**Output:**

Graphical user interface, application

Description automatically generated

Figure 1. Watch window showing the PORTC with a value of 0xCC

Graphical user interface, application

Description automatically generated

Figure 2. Watch window showing the PORTC with a value of 0xAA

**Note: The configuration bits are not included which has to come after the inclusion of header files**

**Conclusion:** Thus, an assembly program is written to use timer interrupt-based delay for alternating between two bit patterns on PORTC has been done. And the same has been simulated using MPLAB X simulator and the snaps of the output File Register window are given.

Project: timer

**Exercise 6**: Write an assembly program to convert the given analog voltage as digital value and give those values to PORTB and PORTC using MPLAB X simulator.

**Objective**: To write an assembly language program to convert the given analog voltage as digital value and give that values to PORTB and PORTC using MPLAB X simulator.

**Algorithm:**

1. Configure RA0 as input. Configure AN0 as Analog input, ADC result as right justified and conversion clock as FOSC/32
2. Start the ADC Conversion.
3. Check for the end of conversion.
4. Transfer the ADC result high byte (ADRESH) to PORTB and Low byte (ADRESL) to PORTC.
5. Go to step 6 (Infinite loop)

**Assembly Program:**

**#include <xc.inc>**

**PSECT res\_vect, class=CODE, reloc=2**

**res\_vect:**

**goto main**

**PSECT code**

**main:**

**clrw**

**CLRF TRISC, a ;PORTC IS CONFIGURED AS OUTPUT**

**CLRF TRISB, a ;PORTB IS CONFIGURED AS OUTPUT**

**MOVLW 0X01**

**MOVWF TRISA, a ;RAO/AN0 IS CONFIGURED AS INPUT**

**MOVLW 0x8E ;AN0 CONFIGURED AS ANALOG INPUT**

**MOVWF ADCON1,a ;RESULT OF ADC IS RIGHT JUSTIFIED**

**MOVLW 0X01**

**MOVWF ADCON0,a ;CONVERSION CLK RATE Fosc/32 CHANNEL0**

**BSF ADCON0,1,a ;START THE CONVERSION**

**LOOP: BTFSC ADCON0,1,a ;CHECK FOR END OF CONVERSION**

**GOTO LOOP**

**MOVF ADRESH,W, a**

**MOVWF PORTB, a**

**MOVF ADRESL,W, a**

**MOVWF PORTC, a**

**GOTO main**

**END res\_vect**

**Input:**

Graphical user interface, application

Description automatically generated

Figure 1. Add the voltage for AN0, VDD and VSS in the stimulus and fire it.

**Output:**

Graphical user interface, table

Description automatically generated

Figure 2. Watch window showing the PORTB and PORTC having the digital value (0xFFC0) corresponding to 5V.

**Note: The configuration bits are not included which has to come after the inclusion of header files**

**Conclusion:** Thus, an assembly program is written to get the digital equivalent of analog value using ADC.

Project: adc