

Universidad nacional Autónoma de México

Facultad de Ingeniería

Microcomputadoras

Tarea 3: Sumador de 2 números

- Alumno: Alfonso Murrieta Villegas
- Profesor: Rubén Anaya García

Realizar un programa que realice la suma de dos números de 8 bits en formato decimal y entregue el resultado también en formato decimal, representar el resultado en dos registros.

Propuestas de diseño y consideraciones

Consideración 1:

La entrada y salida debe ser en decimal por lo que:

1. Debe haber una conversión de datos con un flujo como el siguiente:



2. Al haber una conversión de sistema numérico debemos contemplar el mismo algoritmo (Más adelante en el pseudocódigo mencionaré a detalle este apartado)

Consideración 2:

1. La posición en el sistema de numeración determina el peso, es decir, en la primera posición del número guardado en el registro es la decena y el segundo número es la unidad, por lo que debe separar como dos variables para realizar por separado la conversión y tratamiento del dato
 1. Variable de entrada 1 = Decena
 2. Variable de entrada 2 = Unidad
2. Respecto a lo anterior debe considerarse a detenimiento el acarreo de cada número además de otros componentes del algoritmo

Consideración 3:

1. Recordemos que hay un algoritmo propuesto por el profesor además del ajuste que debemos considerar al realizar las debidas conversiones

Algoritmo Ajuste a decimal

State of C Bit Before DAA (Column 1)	Upper Half-Byte of ACCA (Bits [7:4]) (Column 2)	Initial Half-Carry H Bit from CCR (Column 3)	Lower Half-Byte of ACCA (Bits [3:0]) (Column 4)	Number Added to ACCA by DAA (Column 5)	State of C Bit After DAA (Column 6)
0	0-9	0	0-9	00	0
0	0-8	0	A-F	06	0
0	0-9	1	0-3	06	0
0	A-F	0	0-9	60	1
0	9-F	0	A-F	66	1
0	A-F	1	0-3	66	1
1	0-2	0	0-9	60	1
1	0-2	0	A-F	66	1
1	0-3	1	0-3	66	1

Pseudocódigo

A continuación un ejemplo de como se haría en general un algoritmo en MIPS , cabe destacar que para nuestro microcontrolador debido a la arquitectura y a la disponibilidad de mnemónicos cambiará tanto la cantidad de líneas como el mismo código sin embargo la lógica sería relativamente similar

```
INICIO
    ;OBTENER VALORES => Hacerlo 2 veces para cada posición de ambos números de
    entrada
    mov ah,01h
    int 21h ; Obtener el primer dígito
    mov var1,a1 ; Guardarlo en la respectiva variable

    mov ah,01h
    int 21h
    sub a1,30h
    add a1,var1

    mov d1,a1
    ; La suma sería una etiqueta propia para contemplar la complejidad de nuestro
    problema
    add d1,30h

    ;Imprimir los valores
    mov ah,02h*
    int 21h
END
```

Como propuesta de la suma en nuestro microcontrolador , a continuación planteo una generalización de la respectiva subrutina ya en ensamblador

```
SUMA
    MOVF    VARIABLE1,W
    MOVWF   POS_NUM
    MOVF    VARIABLE2,W
    ADDWF   POS_NUM
```

```

    BTFSC    STATUS,C
    BSF      BIT_C,0

    BTFSC    STATUS,DC
    BSF      BIT_DC,0

    MOVF     POS_NUM,W
    MOVWF    AUX1 ; Para el carry

```

Código y explicación por segmentos

Para este apartado será explicado parte por parte el programa para hacer más legible el algoritmo y la lógica empleada:

Variables y registros

A continuación el código encargado de instanciar todas las variables a ocupar durante el programa , lo más destacable es la posición o la ubicación de los registros encargados de la entrada y salida de datos

```

PROCESSOR 16f877
INCLUDE <p16f877.inc>

;VAR FOR DATA
IN_1    EQU H'20'
IN_2    EQU H'22'
POS_C   EQU H'24'
POS_D   EQU H'25'

;INPUT DATA
CAR_C   EQU H'58'
CAR_D   EQU H'59'
N1_SIZ  EQU H'60'
N1_AUX  EQU H'61'
N2_SIZ  EQU H'62'
N2_AUX  EQU H'63'

;OUTPUT DATA - AUX VARS
OUT_AUX EQU H'5E'
CONV    EQU H'50'
COUNT_AUX EQU H'5F'

OUT_SIZ EQU H'64'
OUT_AUX_2 EQU H'65'

    ORG 0
    GOTO INICIO
    ORG 5

```

Asignación y primeros pasos

Al igual que cualquier programa, siempre tenemos un apartado "main" o "index" en este caso la subrutina INICIO es la encargada de empezar con la asignación y primeras manipulaciones de los datos ingresados en los registros previos:

```
INICIO
    CLRFB    POS_C
    CLRFB    POS_D
    CLRFB    CAR_C
    CLRFB    CAR_D

    MOVLW    0X04
    MOVWF    COUNT_AUX

    MOVLW    0X01
    MOVWF    OUT_AUX

    MOVFB    IN_1,W
    MOVWF    CONV

    BCF      STATUS,C
    CALL     CHECK_P_D

    MOVLW    0X04
    MOVWF    COUNT_AUX

    MOVLW    0X00
    MOVWF    OUT_AUX

    MOVFB    IN_2,W
    MOVWF    CONV

    BCF      STATUS,C
    CALL     CHECK_P_D
    GOTO     ADD_P
```

NOTA: Al inicio hacemos una limpieza general para posteriormente realizar las llamadas de las 3 distintas subrutinas principales

Validación, posicionamiento y algoritmo

Para este apartado es donde a priori se hace la validación del tamaño, de los números ingresados a demás del apartado de conversión para posteriormente hacer el llamado de la subrutina de la suma

```
CHECK_P_D
    RRF      CONV
    BCF      STATUS,C
    DECFSZ   COUNT_AUX
    GOTO     CHECK_P_D

MAYOR
    MOVLW    0X0A
    SUBWF    CONV,0
    BTFSC    STATUS,C
    GOTO     FIN

ASIG_1
```

```

        MOVF      CONV,W
        BTFSC     OUT_AUX,0
        GOTO      ASIG_2
        GOTO      ASIG_AUX
ASIG_2
        MOVWF     N1_SIZ
        MOVF      IN_1,W
        GOTO      CHECK_V_D
ASIG_AUX
        MOVWF     N2_SIZ
        MOVF      IN_2,W
        GOTO      CHECK_V_D
CHECK_V_D
        MOVWF     CONV
        SWAPF     CONV,1
        MOVLW     0X04
        MOVWF     COUNT_AUX
CHECK_V_D_2
        RRF       CONV
        BCF       STATUS,C
        DECFSZ    COUNT_AUX
        GOTO      CHECK_V_D_2
MENOR
        MOVLW     0X0A
        SUBWF     CONV,0
        BTFSC     STATUS,C
        GOTO      FIN
ASIG_B
        MOVF      CONV,W
        BTFSC     OUT_AUX,0
        GOTO      ASIG_B1
        GOTO      ASIG_B2
ASIG_B1
        MOVWF     N1_AUX
        RETURN
ASIG_B2
        MOVWF     N2_AUX
        RETURN
FIN
        MOVLW     0XFF
        MOVWF     POS_C
        MOVWF     POS_D
        GOTO      FIN

```

SUMA

Para este apartado realmente no hay mucha complejidad , la subrutina lo que realiza es meramente la suma de los datos previamente cargados, analizados y además convertidos.

De manera general lo único aquí destacables mencionar que se hace la llamada de la subrutina CHECK_OUT debido a que debemos convertir el valor ahora en decimal además de contemplar las debidas consideraciones del profesor (Los ajuste respecto al decimal).

```

ADD_P
        MOVF      IN_1,W
        MOVWF     POS_D

```

```

MOVWF    IN_2,W
ADDWF    POS_D
BTFSF    STATUS,C
BSF       CAR_C,0
BTFSF    STATUS,DC
BSF       CAR_D,0
MOVLW    0X04
MOVWF    COUNT_AUX
MOVWF    POS_D,W
MOVWF    CONV
BCF       STATUS,C
CALL     CHECK_OUT
GOTO     CH_OUT_2

```

Conversión a decimal , escritura en registros y revisión de cada estado o condición inicial del ajuste decimal

Para este apartado realmente hay muchísima complejidad debido a que debemos considerar todas las consideraciones del ajuste a decimal (Planteados al inicio de este trabajo), además de evidentemente realizar la conversión debida para posteriormente escribir los datos en los debidos registros

NOTA: Es importante destacar que la mayor parte de este apartado se hacen bastantes llamadas a los diferentes estados o condiciones del ajuste decimal.

```

CHECK_OUT
    RRF     CONV
    BCF     STATUS,C
    DECFSZ  COUNT_AUX
    GOTO    CHECK_OUT

ASIG_R
    MOVF    CONV,W
    MOVWF   OUT_SIZ
    MOVLW   0X04
    MOVWF   COUNT_AUX
    MOVF    POS_D,W
    MOVWF   CONV
    BCF     STATUS,C
    SWAPF   CONV

CHECK_OUT2
    RRF     CONV
    BCF     STATUS,C
    DECFSZ  COUNT_AUX
    GOTO    CHECK_OUT2

ASIG_R2
    MOVF    CONV,W
    MOVWF   OUT_AUX_2
    RETURN

CH_OUT_2
    BTFSF   CAR_C,0
    GOTO    CAR_C_0
    GOTO    CAR_C_1

CAR_C_0
    MOVLW   0X0A

```

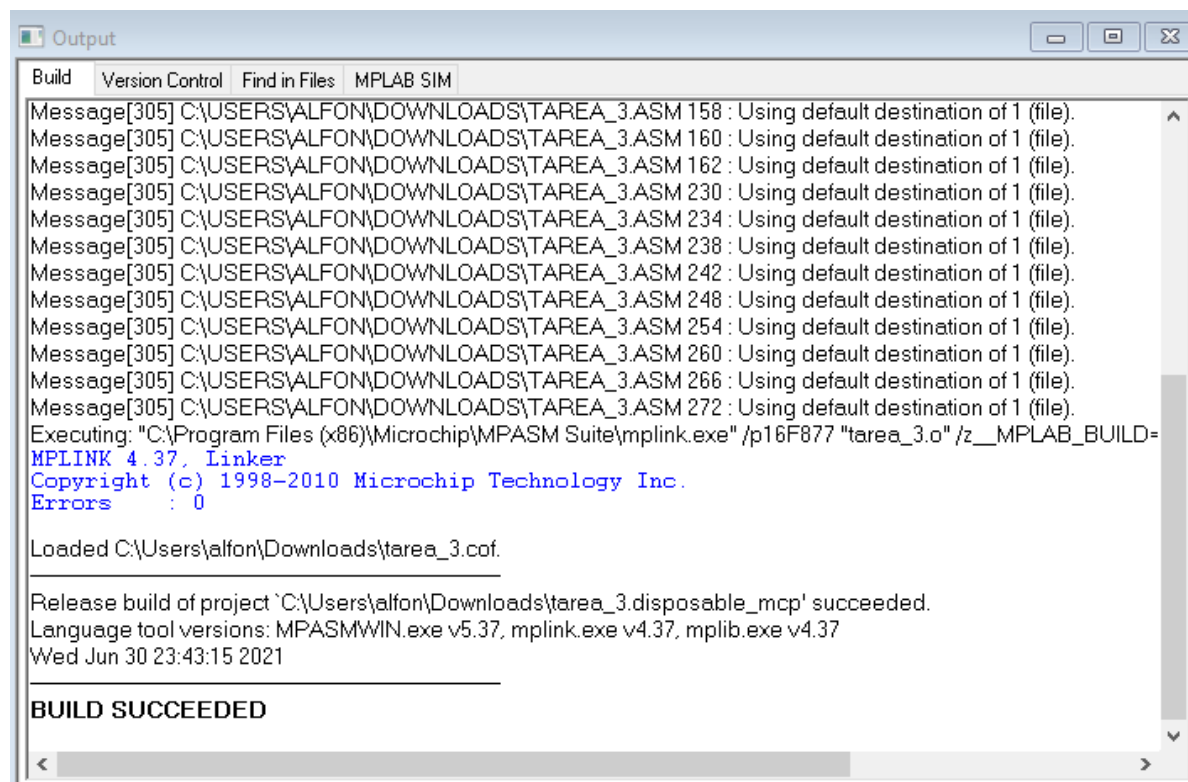
```

        SUBWF    OUT_SIZ,W
        BTFSS    STATUS,C
        GOTO     STA_FIR
        GOTO     STA_SEC
STA_FIR
        BTFSS    CAR_D,0
        GOTO     CAR_D_0
        GOTO     CAR_D_1
CAR_D_0
        MOVLW    0X0A
        SUBWF    OUT_AUX_2,W
        BTFSS    STATUS,C
        GOTO     STA_1
        GOTO     STA_2
CAR_D_1
        GOTO     STA_3
STA_SEC
        BTFSS    CAR_D,0
        GOTO     CAR_D2_0
        GOTO     CAR_D2_1
CAR_D2_0
        MOVLW    0X0A
        SUBWF    OUT_AUX_2,W
        BTFSS    STATUS,C
        GOTO     STA_4
        GOTO     STA_5
CAR_D2_1
        GOTO     STA_6
CAR_C_1
        MOVLW    0X03
        SUBWF    OUT_SIZ,W
        BTFSS    STATUS,C
        GOTO     STA_AUX
        GOTO     STA_9
STA_AUX
        MOVLW    0X0A
        SUBWF    OUT_AUX_2,W
        BTFSS    STATUS,C
        GOTO     STA_7
        GOTO     STA_8

```

Ejecución y prueba del código

A continuación se muestra la compilación exitosa del código:



The screenshot shows the 'Output' window of the MPLAB IDE. The window has a title bar with standard Windows controls and a menu bar with 'Build', 'Version Control', 'Find in Files', and 'MPLAB SIM'. The main text area displays the following output:

```
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 158 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 160 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 162 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 230 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 234 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 238 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 242 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 248 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 254 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 260 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 266 : Using default destination of 1 (file).
Message[305] C:\USERS\ALFON\DOWNLOADS\TAREA_3.ASM 272 : Using default destination of 1 (file).
Executing: "C:\Program Files (x86)\Microchip\MPASM Suite\mplink.exe" /p16F877 "tarea_3.o" /z__MPLAB_BUILD=
MPLINK 4.37, Linker
Copyright (c) 1998-2010 Microchip Technology Inc.
Errors      : 0

Loaded C:\Users\alfon\Downloads\tarea_3.cof.

Release build of project 'C:\Users\alfon\Downloads\tarea_3.disposable_mcp' succeeded.
Language tool versions: MPASMWIN.exe v5.37, mplink.exe v4.37, mplib.exe v4.37
Wed Jun 30 23:43:15 2021

BUILD SUCCEEDED
```

Una vez compilado y ensamblado nuestro código, a continuación una tabla con los resultados obtenidos tras varios ejempos:

NOTA: La entrada del primer número se da en el registro 20 , la entrada 2 está en el registro 22 y la salida se dan en los registros 24 y 25

Operación
a resolver

Salida en MPLAB

$21 + 12 = 33$

The screenshot shows the MPLAB IDE v6.56 interface. The main window displays assembly code for a PIC16F877. The code includes instructions like MOVWF, ADDWF, and GOTO. The File Registers window is open, showing a table of registers with addresses 00 to 0F. The Output window shows a successful build message.

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000	--	00	B1	08	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	21	00	12	00	00	33	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050	03	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
060	02	01	02	08	03	00	00	00	00	00	00	00	00	00	00	00
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
080	--	FF	B1	08	00	3F	FF	FF	FF	07	00	00	00	00	--	--
090	--	00	FF	00	--	--	--	--	02	--	--	--	--	--	--	--
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	--	00	B1	08	--	00	--	--	--	00	00	00	00	00	--	--
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

$10 + 10 = 20$

The screenshot shows the File Registers window in MPLAB IDE v6.56. It displays a table of registers with addresses 00 to 110. The table includes columns for Hex and Symbolic values. The ASCII column shows the corresponding ASCII values for the registers.

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
000	--	00	B1	08	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	10	00	10	00	00	20	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	20	00	00	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
060	01	00	01	00	02	00	00	00	00	00	00	00	00	00	00	00
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
080	--	FF	B1	08	00	3F	FF	FF	FF	07	00	00	00	00	--	--?
090	--	00	FF	00	--	--	--	--	02	00	--	--	--	--	00	00
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	--	00	B1	08	--	00	--	--	--	00	00	00	00	00	00	00
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

$5 + 17 = 22$

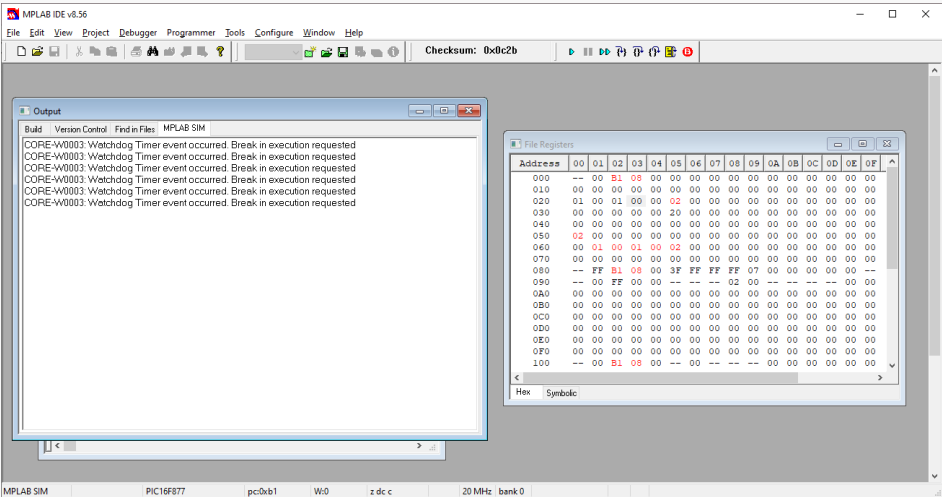
The screenshot shows the MPLAB IDE v6.56 interface. The main window displays the Output window, which shows a successful build message. The File Registers window is also open, showing a table of registers with addresses 00 to 100. The table includes columns for Hex and Symbolic values.

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000	--	00	B1	08	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	05	00	17	00	00	20	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	20	00	00	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050	0C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
060	00	05	01	07	01	0C	00	00	00	00	00	00	00	00	00	00
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
080	--	FF	B1	08	00	3F	FF	FF	FF	07	00	00	00	00	--	--
090	--	00	FF	00	--	--	--	--	02	--	--	--	--	--	--	--
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	--	00	B1	08	--	00	--	--	--	00	00	00	00	00	00	00

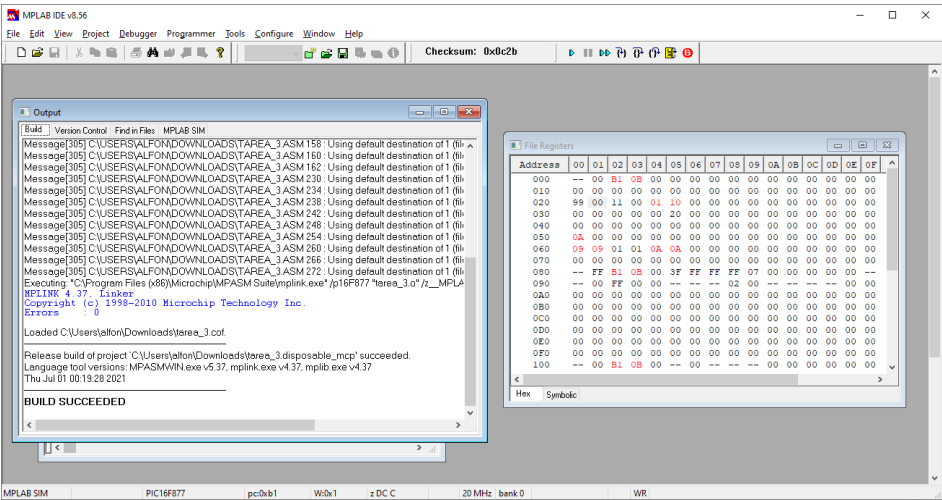
Operación
a resolver

Salida en MPLAB

1+1=2



99+11



NOTA: En la última imagen se aprecia como en el registro 24 se tiene a la centena del número decimal

File Registers																
Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000	--	00	B1	0B	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	99	00	11	00	01	10	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	20	00	00	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
060	09	09	01	01	0A	0A	00	00	00	00	00	00	00	00	00	00
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
080	--	FF	B1	0B	00	3F	FF	FF	FF	07	00	00	00	00	00	--
090	--	00	FF	00	00	--	--	--	02	00	--	--	--	--	00	00
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	--	00	B1	0B	00	--	00	--	--	--	00	00	00	00	00	00

