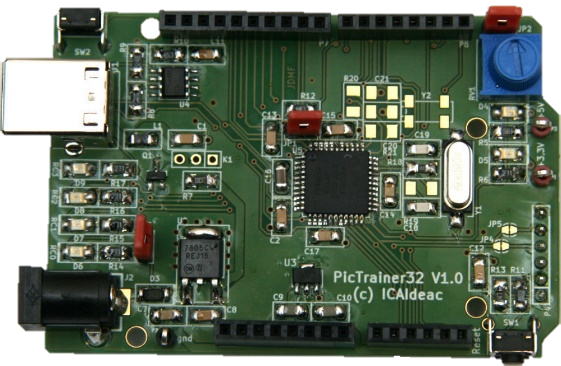
|  |
| --- |
| **Microprocesadores** |
| Práctica 4: Programación en ensamblador |
| Francisco Martín Martínez |

|  |
| --- |
| Jorge Calvar, Fernando Santana  1 de marzo de 2021 |



Contenido

[Introducción 2](#_Toc66722919)

[2. Inspección del código máquina generado por el compilador 2](#_Toc66722920)

[2.1 Ejercicio 3](#_Toc66722921)

[Tipo I 3](#_Toc66722922)

[Tipo R 3](#_Toc66722923)

[Tipo J 3](#_Toc66722924)

[3. Ensamblador en línea 4](#_Toc66722925)

[4. Función para generar retardos en ensamblador 5](#_Toc66722926)

[Función Retardo 5](#_Toc66722927)

[Main 6](#_Toc66722928)

[Conclusión 7](#_Toc66722929)

[Anexo 8](#_Toc66722930)

# Introducción

En esta práctica nos familiarizaremos con los temporizadores del PIC32MX230F064D. También aprenderemos a realizar programas que incluyan otros ficheros .c y a crear archivos de cabecera.

# 2. Inspección del código máquina generado por el compilador

En este apartado vamos a ver cómo inspeccionar el código máquina generado por el compilador. Para ello vamos a partir del programa que enciende el LED RC0 cuando se pulsa el pulsador RB5

#include <xc.h>

#define PIN\_PULSADOR 5

**int** main(**void**)

{

**int** pulsador;

TRISC = 0; *// Configuramos todo como output*

LATC = 0xFFFE; *// El LED es activo a nivel bajo*

TRISB = 1 << PIN\_PULSADOR; *// Pin de pulsador como input*

**while** (1)

{

*// Se lee el estado del pulsador*

pulsador = (PORTB >> PIN\_PULSADOR) &1;

**if** (pulsador == 0)

{

LATC &= ~1;

}

**else**

{

LATC |= 1;

}

}

}

# 2.1 Ejercicio

Compilamos el programa en modo debug y mostramos el código máquina de la siguiente forma:

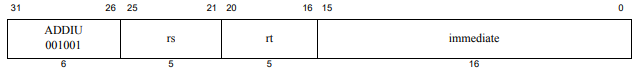


El código maquina se encuentra anexo al final del documento.

A continuación analizaremos una instrucción de cada tipo

## Tipo I

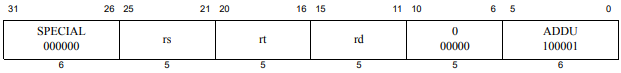
9D000194 27BDFFF0 ADDIU SP, SP, -16



|  |  |  |  |
| --- | --- | --- | --- |
| 001001 | 11101 | 11101 | 1111 1111 1111 0000 |
| ADDIU | 29 SP | 29 SP | complemento a 2 16 (-16) |

Suma -16 al registro del puntero a la pila. Aunque solo queremos mover cuatro posiciones cada w*ord* tiene 4 bytes, al ser el PIC32 de 32 bits, tal y como explica su nombre.

## Tipo R



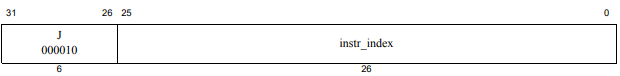
9D0002EC 03A0F021 ADDU FP, SP, ZERO

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | 11101 | 00000 | 11110 | 00000 | 100001 |
| special | 29 sp | zero | 30 fp | 0 | ADDU |

Esta instrucción copia en *Frame Pointer* el contenido del registro del *Stack Pointer*. Se trate de una instrucción de tipo R ya que recibe tres direcciones de registros en su entrada.

## Tipo J

9D00023C 0B400081 J 0x9D000204



|  |  |
| --- | --- |
| 000010 | 11010000000000000010000001 |
| J | dirección |

Salto incondicional a la dirección de la instrucción donde empieza el bucle de scan, es decir, while(1). Es de tipo J al recibir solo un inmediato.

# 3. Ensamblador en línea

El compilador no optimiza al máximo al generar el código maquina a partir del código en C. Si queremos optimizar los tiempos para ahorrar ciclos de reloj podemos usar el ensamblador en línea recodificando las instrucciones manualmente. Otra opción sería adquirir la versión PRO del compilador XC32, que cuesta 24,60€ al mes.

A continuación, se recodifica el programa expuesto en el ejercicio 3

#include <xc.h>

#define PIN\_PULSADOR 5

**int** main(**void**)

{

**int** pulsador;

TRISC = 0; *// Configuramos todo como output*

LATC = 1; *// El LED es activo a nivel bajo*

TRISB = 1 << PIN\_PULSADOR; *// Pin de pulsador como input*

**while**(1){

*// Se lee el estado del pulsador*

pulsador = (PORTB>>PIN\_PULSADOR) & 1;

**if**(pulsador == 0){

*//LATC &= ~1;*

asm(" lui $v0, 0xBF88"); *//0xBF88 = -16504*

asm(" lw $v1, 25136($v0)");

asm(" addiu $a0, $0, -2");

asm(" and $v1, $v1, $a0");

asm(" sw $v1, 25136($v0)");

}**else**{

LATC |= 1;

}

}

}

# 4. Función para generar retardos en ensamblador

Se va a escribir una función que genere un retardo con el temporizador 2 utilizando el siguiente encabezado:

## Función Retardo

*/\**

*\* File: Retardo.h*

*\* Author: Jorge & Fernando*

*\**

*\* Created on February 5, 2020*

*\*/*

**int** Retardo (**uint16\_t** retardo\_ms);

En este caso, hemos aprovechado que todos los retardos son múltiplos de 1 ms para que el temporizador siempre tenga 1 ms de periodo y lo utilizaremos varias veces hasta conseguir el retardo deseado.

En concreto, la función en ensamblador sigue realiza los siguientes pasos:

1. Comprueba si el retardo es 0, en cuyo caso la función devolverá directamente un 0.
2. A continuación, se inicializa el temporizador, configurando T2CON, el bit correspondiente de IFS0 y TMR a 0. Y el periodo a 4999, lo que corresponde con 1 ms.
3. Encendemos el temporizador activando el bit 15 de T2CON.
4. Entramos en un bucle y esperamos tantas veces a que termine el temporizador como milisegundos hay que esperar.
5. Paramos el temporizador (T2CON = 0) y devolvemos un 0.

#include <xc.h>

.text # sección de código

# Se define el nombre de la función global para poder llamarla desde C

.global Retardo

.ent Retardo # Introduce el símbolo Retardo en el código para depuración

Retardo:

*addi* v0, zero, 0

# Se recibe el periodo en ms en a0

*beq* a0, zero, Fin0 # return 0 si periodo == 0

# T2CON = 0

*la* t1, T2CON

*sw* zero, 0(t1)

# IFS0bits.T2IF = 0

*la* t4, IFS0

*addi* t9, zero, 0x200 # t9 = 1 << 9

*nor* t9, t9, zero # t9 = not (1<<9(

*lw* t0, 0(t4)

*and* t0, t0, t9 # IFS0 &= ~(1<<9)

*sw* t0, 0(t4)

*la* t2, TMR2 # TMR2 = 0

*sw* zero , 0(t2)

# PR2 = 4999

*addi* t0, zero, 4999

*la* t3, PR2

*sw* t0, 0(t3)

# Encendemos

*addi* t0, zero, 0x8000

*sw* t0, 0(t1) # T2CON = 1<<15

Buc: *beqz* a0, FBuc

*addi* t9, zero, 0x200 # t9 = 1 << 9

*nor* t9, t9, zero # t9 = not (1<<9(

*lw* t0, 0(t4)

*and* t0, t0, t9 # IFS0 &= ~(1<<9)

*sw* t0, 0(t4)

*sw* zero, 0(t2) # TMR2 = 0

# Esperamos 1ms

Esp: *lw* t0, 0(t4) # t0 = IFS0

*andi* t0, t0, 0x200 # t0 = IFS0 & (1<<9)

*beqz* t0, Esp

nop

*addi* a0, a0, -1

*j* Buc

nop

# T2CON = 0

FBuc: *sw* zero, 0(t1)

*j* Fin0

nop

Fin1: *addi* v0, zero, 1

Fin0: *jr* ra

.end Retardo

## Main

Finalmente se crea un main para probar la función. El programa cambia el estado del LED RC0, que inicialmente esta apagado, cada 500ms. Es decir, el LED parpadea con una frecuencia de 1Hz.

*/\**

*\* File: main4.c*

*\* Author: Jorge & Fernando*

*\**

*\* Created on February 5, 2020*

*\*/*

#include <xc.h>

#include "Pic32Ini.h"

#include "retardo.h"

#define PIN\_LED 0

**int** main(**void**) {

InicializarReloj();

*// Pines del LED como digital*

ANSELC &= ~(1 << PIN\_LED);

LATA = 0; *// En el arranque dejamos todas las salidas a 0*

LATB = 0;

LATC = 0xF; *//apagados al empezar*

*// Como salidas*

TRISA = 0;

TRISB = 0;

TRISC = 0;

**while**(1){

Retardo(1000); *//0.5s para que la frecuencia de parpadeo = 1Hz*

LATC ^= 1 << PIN\_LED; *// Se invierte el LED*

}

}

## Conclusión

En esta práctica hemos aprendido a:

* Generar un listado en ensamblador de un programa en C para ver el código generado por el compilador.
* Insertar instrucciones en ensamblador en línea dentro de un programa en C
* Escribir funciones en ensamblador para ser llamadas desde C.

# Anexo

El archivo consta de tres columnas: la primera contiene o bien el número de línea del código fuente en C o bien la dirección de la instrucción de código máquina. La segunda columna contiene el código máquina y la tercera o bien la instrucción en C o bien la instrucción en ensamblador.

1: #include <xc.h>

2: #define PIN\_PULSADOR 5

3:

4: **int** main(**void**)

5: {

9D0001D8 27BDFFF0 ADDIU SP, SP, -16

9D0001DC AFBE000C SW FP, 12(SP)

9D0001E0 03A0F021 ADDU FP, SP, ZERO

6: **int** pulsador;

7:

8: TRISC = 0; *// Configuramos todo como output*

9D0001E4 3C02BF88 LUI V0, -16504

9D0001E8 AC406210 SW ZERO, 25104(V0)

9: LATC = 1; *// El LED es activo a nivel bajo*

9D0001EC 3C02BF88 LUI V0, -16504

9D0001F0 24030001 ADDIU V1, ZERO, 1

9D0001F4 AC436230 SW V1, 25136(V0)

10: TRISB = 1 << PIN\_PULSADOR; *// Pin de pulsador como input*

9D0001F8 3C02BF88 LUI V0, -16504

9D0001FC 24030020 ADDIU V1, ZERO, 32

9D000200 AC436110 SW V1, 24848(V0)

11:

12: **while**(1){

13: *// Se lee el estado del pulsador*

14: pulsador = (PORTB>>PIN\_PULSADOR) & 1;

9D000204 3C02BF88 LUI V0, -16504

9D000208 8C426120 LW V0, 24864(V0)

9D00020C 00021142 SRL V0, V0, 5

9D000210 30420001 ANDI V0, V0, 1

9D000214 AFC20000 SW V0, 0(FP)

15: **if**(pulsador == 0){

9D000218 8FC20000 LW V0, 0(FP)

9D00021C 14400009 BNE V0, ZERO, 0x9D000244

9D000220 00000000 NOP

16: LATC &= ~1;

9D000224 3C02BF88 LUI V0, -16504

9D000228 8C426230 LW V0, 25136(V0)

9D00022C 2404FFFE ADDIU A0, ZERO, -2

9D000230 00441824 AND V1, V0, A0

9D000234 3C02BF88 LUI V0, -16504

9D000238 AC436230 SW V1, 25136(V0)

17: }**else**{

18: LATC |= 1;

9D000244 3C02BF88 LUI V0, -16504

9D000248 8C426230 LW V0, 25136(V0)

9D00024C 34430001 ORI V1, V0, 1

9D000250 3C02BF88 LUI V0, -16504

9D000254 AC436230 SW V1, 25136(V0)

19: }

20: }

9D00023C 0B400081 J 0x9D000204

9D000240 00000000 NOP

9D000258 0B400081 J 0x9D000204

9D00025C 00000000 NOP

21:

22: }