Práctica 4 – Programación en ensamblador

Jaime Arana
Manuel Ferrero
3ºA

Inspección del código máquina:

TIPO-R

```
15  9D000190   03A0F021    ADDU FP, SP, ZERO
```

Instrucción: **addu rd, rs, rt** (suma)

Dicha instrucción tiene la siguiente distribución de bits:

000000 + rs(5 bits) + rt (5 bits) + rd(5 bits) + 00000 + 100001

Por lo tanto, si se sustituye por sus valores en decimal de los registros:

000000 + 29 + 0 + 30 + 00000 + 100001

Si pasamos estos a código máquina, que es la instrucción en hexadecimal, obtenemos:

03A0F021

TIPO-I

```
14  9D00018C   AFBE000C    SW FP, 12(SP)
```

Instrucción: **rd, desp16(rs)** (store word)

Dicha instrucción tiene la siguiente distribución de bits:

101011 + rs(5 bits) + rd(5 bits) + desplazamiento16

Por lo tanto, si se sustituye por sus valores en decimal de los registros:

101011 + 29 + 30 + 12

Si pasamos estos a código máquina, que es la instrucción en hexadecimal, obtenemos:

AFBE000C

TIPO-J

```
70  9D00020C   0B400075    J 0x9D0001D4
```

Instrucción: **j Dir28** (Salto incondicional)

Dicha instrucción tiene la siguiente distribución de bits:

000010 + constante << 2

Por lo tanto, si se sustituye por sus valores en decimal de los registros:

000010 + 0xD0001D4 << 2

Si pasamos estos a código máquina, que es la instrucción en hexadecimal, obtenemos:

0B400075

Ensamblador en línea:

```
#include <xc.h>
#include "Pic32Ini.h"

#define PIN_LED 0
#define PIN_PULSADOR 5

int main (void) {
    int valor_pulsador;

    // LED como salida, pulsador como entrada
    TRISC &= ~(1 << PIN_LED);
    TRISB |= (1 << PIN_PULSADOR);

    // LED empieza apagado
    LATC |= (1 << PIN_LED);

    while(1){
        valor_pulsador = ( PORTB >> PIN_PULSADOR ) & 1;

        // mientras el pulsador está pulsado = LED encendido
        if(valor_pulsador == 0){
            //LATC &= ~1;
            asm("  lui $v0, 0xBF88");   // 0xBF88 = -16504
            asm("  lw $v1, 25136($v0)");
            asm("  addiu $a0, $zero, -2");  // -2 = 0xFFFE
            asm("  and $v1, $v1, $a0");
            asm("  sw $v1, 25136($v0)");

        } else {
            LATC |= (1 << PIN_LED);
        }
    }
}
```

Función para generar retardos:

Para implantar la función que genera los retardos, creamos un retardo de 1ms e iteramos hasta llegar al valor del retardo solicitado.

```
#include <xc.h>
.text
.global Retardo
.ent Retardo
Retardo:
    beq a0, zero, Fin   # comprobar si retardo = 0
    la t0, T2CON
    sw zero, 0(t0)  # T2CON=0
    la t0, TMR2
    sw zero, 0(t0)  # TMR2=0
    li t5, 0xFFFFFDFF   # Mascara para poner IFS0bits.T2IF = 0
    la t1, IFS0
    lw t2, 0(t1)    # Cargo lo de IFS0
    and t2, t5, t2  # Aplico mascara
    sw t2, 0(t1)    # IFS0bits.T2IF=0
    addiu t0, zero, 0x1387    # 4999 en hexadecimal
    la t2, PR2
    sw t0, 0(t2)    # PR2 = 4999;
    ori t0, zero, 0x8000
    la t2, T2CON
    sw t0, 0(t2)    # T2CON = 0x8000;
    addu v0, zero, zero # i=0
For:
    sltu t0, v0, a0 # t0 = 1 si i < retardo_ms
    beq t0, zero, Fin # si i < retardo_ms falso, salimos del bucle
    nop
While:
    lw t0, 0(t1)    # Leo en t0 el registro IFS0
    li t3, 0x0200   # Mascara para poner todo a 0 menos el bit que me interesa
    and t4, t0, t3  # Aplico mascara
    beq t4, zero, While # Si todo es 0 ese bit que me interesa esta a 0
    nop
    la t1, IFS0
    lw t2, 0(t1)    # Cargo lo de IFS0
    and t2, t5, t2  # Aplico mascara --> guardada en t5
    sw t2, 0(t1)    # IFS0bits.T2IF = 0
    addi v0, v0, 1  # i++
    j For
    nop
Fin:
    jr ra
    .end Retardo
```

```c
#include <xc.h>
#include "Retardo.h"
#include "Pic32Ini.h"

#define PIN_LED 0

int main(void) {
    // LED como salida
    TRISC &= ~(1 << PIN_LED);

    // LED empieza apagado
    LATC |= (1 << PIN_LED);

    while(1){
        Retardo(500);

        LATC ^= (1 << PIN_LED);
    }
}
```