66.70 Estructura del Computador

Trabajo Práctico

2do Cuatrimestre 2012

Diseño de la lógica de un sistema de semáforos

Segunda entrega: "Solución programada"

Corrector: **Ing. Dario Novodvoretz**
Grupo: **8**
Integrantes:

Alumno	*E-Mail*
Pántano, Laura Raquel	laurapantano@yahoo.com
Extramiana, Federico	federicoextramiana@hotmail.com
Rossi, Federico Martín	federicomrossi@gmail.com

1. Introducción

Se ha solicitado desarrollar el sistema para el funcionamiento nocturno del par de semáforos que se encuentran en la esquina de una estación de Bomberos. A continuación se describe cómo debe ser el funcionamiento de los mismos:

- Ambos semáforos se encontrarán por defecto en un estado en el cual la luz amarilla se enciende de forma intermitente (1 seg. prendida – 1 seg. apagada);
- Cuando un peatón desea cruzar, debe pulsar el botón que se encuentra debajo del semáforo. En tal caso, los semáforos seguirán la siguiente secuencia:
  - Por 5 segundos se mantendrá encendida la luz amarilla en el semáforo 1, y se encenderá la luz roja en el semáforo 2.
  - Luego, se encenderá la luz verde del semáforo 1 dejando la luz roja en el semáforo 2. Se permanecerá en este estado por una duración de 30 segundos.
  - Transcurrido ese período, se encenderá la luz amarilla del semáforo 1 mientras que el semáforo 2 continua con la luz roja encendida.
  - Una vez transcurridos 5 segundos, se enciende la luz roja del semáforo 1 mientras que se pone en amarillo el semáforo 2. Se quedará en este estado por 5 segundos.
  - Ahora deberá permanecer el semáforo 1 en rojo mientras que el semáforo 2 prende únicamente la luz verde, quedando en este estado por otros 30 segundos.
  - Cumplido dicho tiempo, se procede a encender la luz amarilla exclusivamente en el semáforo 2, mientras que en el semáforo 1 se mantiene encendida la roja.
  - Luego de 5 segundos, se procede a volver al estado por defecto, en el cual ambos semáforos encienden de forma intermitente sus luces amarillas.
- En caso de volverse a presionar el botón para el cruce de los peatones mientras se ejecuta la secuencia anterior, éste no tiene ningún efecto;
- Existe también un botón que es utilizado al momento que deben salir los camiones de Bomberos. Cuando este es presionado, ambos semáforos deben pasar a encender su luz roja y su luz amarilla simultáneamente. Debido a que el tiempo requerido para la salida de los camiones no es conocido, se debe esperar a que este botón sea pulsado nuevamente para volver al estado por defecto de los semáforos (sin importar en qué estados se encontraban previamente);

2. Especificaciones

Se solicita diseñar un código *Assembly ARC* que implemente la lógica de control descripta anteriormente.

Los dos botones y las luces de los semáforos se encuentran mapeados a los bits de la dirección 0xD6000020 de la memoria, tal como se indica a continuación:

- **Bit 0**: Botón de peatón;
- **Bit 1**: Botón para salida de bomberos;

- **Bit 16**: Luz verde del semáforo 1;
- **Bit 17**: Luz amarilla del semáforo 1;
- **Bit 18**: Luz roja del semáforo 1;

- **Bit 24**: Luz verde del semáforo 2;
- **Bit 25**: Luz amarilla del semáforo 2;
- **Bit 26**: Luz roja del semáforo 2;

Por último, se debe suponer que la frecuencia del reloj del procesador es de 1 GHz.

3. Implementación

En el *Código 1* se muestra el código fuente en *Assembly ARC* correspondiente a la solución solicitada. Este se ha desarrollado en el marco del diagrama de estados presentado en el primer informe, en el sentido de que se ha mantenido la lógica de la existencia de tres secuencias separadas entre sí, las cuales se ejecutan de acuerdo a los eventos caracterizados por los pulsadores *BB* (botón de salida de camiones de bomberos) y *BP* (cruce de peatones). Recuérdese que la *secuencia A* representa el estado intermitente por defecto de los semáforos, la *secuencia B* representa el conjunto de estados de los semáforos en el cruce de peatones y la *secuencia C* representa la activación del botón para la salida de los camiones de bomberos.

Código 1: *"TP.Informe.Grupo.8.codigo.ARC.asm" - Solución programada*

```
1
2  ! ----------------------------------------------
3  ! 66.70 - Estructura del Computador
4  ! Facultad de Ingenieria
5  ! Universidad de Buenos Aires
6  !
7  ! Grupo:  8
8  ! Alumnos:  Pantano, Laura Raquel
9  !   Extramiana, Federico
10 !   Rossi, Federico Martin
11 ! ----------------------------------------------
12 ! RESOLUCION DEL TP EN ASSEMBLY ARC
13 ! Sistema de semaforos
14 !
15 ! Direccion de entrada 0xD6000020:
16 !
17 !   Bit0: Boton de peaton.
18 !   Bit1: Boton para salida de bomberos.
19 !   Bit16: Luz verde semaforo 1.
20 !   Bit17: Luz amarilla semaforo 1.
21 !   Bit18: Luz roja semaforo 1.
22 !   Bit24: Luz verde semaforo 2.
23 !   Bit25: Luz amarilla semaforo 2.
24 !   Bit26: Luz roja semaforo 2.
25 !
26 ! Utilizacion de registros:
27 !
28 ! %r1 - Direccion base de la region I/O de la memoria
29 ! %r2 - Registro auxiliar para armado de estados a activar
30 ! %r3 - Registro auxiliar para memorizacion de botones activos
31 ! %r4 - Registro auxiliar para verificacion de botones activos
32 ! %r5 - Contador de ciclos para el timer T1
33 !
34
35
36                 .begin
37                 .org    2048
38 BASE_IO         .equ    0x358000            ! Punto de comienzo de la region mapeada en memoria
39 IO              .equ    0x20                ! 0xD6000020 Posicion de memoria donde se encuentran
40                                             ! mapeadas las luces y pulsadores
41                 sethi   BASE_IO, %r1        ! %r1 <-- Direccion base de la region I/O de memoria
42 default:        and     %r2, 0x0, %r2       ! %r2 <-- 0x0
43                 st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2 (reset de los bits de la
44                                             ! posicion de memoria 0xD6000020)
45 secuenciaA:     call    T1                  ! Delay de 1 segundo
46                 ld      [%r1 + IO], %r3     ! %r3 <-- Valor de I/O mapeado en 0xD6000020
47                 and     %r3, 0x3, %r2       ! Guardamos en %r2 los dos primeros bits de %r3
48                                             ! Estado de luces LA1 (todas las luces apagadas)
49                 st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
50                 call    T1                  ! Delay de 1 segundo
51                 sethi   LA2, %r2            ! %r2 <-- LA2 en 22 bits mas significativos
52                 ld      [%r1 + IO], %r3     ! %r3 <-- Valor de I/O mapeado en 0xD6000020
53                 and     %r3, 0x3, %r3       ! Guardamos en %r3 los dos primeros bits de 0xD6000020
54                 add     %r2, %r3, %r2       ! %r2 <-- %r2 + %r3
55                 st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
56                 ba      secuenciaA          ! Reiniciamos secuencia A
57
58 secuenciaB:     sethi   LB1, %r2            ! %r2 <-- LB1 en 22 bits mas significativos
59                 st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
60                 call    T5                  ! Delay de 5 segundos
```

```
                sethi   LB2, %r2            ! %r2 <-- LB2 en 22 bits mas significativos
                st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
                call    T30                 ! Delay de 30 segundos
                sethi   LB3, %r2            ! %r2 <-- LB3 en 22 bits mas significativos
                st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
                call    T5                  ! Delay de 5 segundos
                sethi   LB4, %r2            ! %r2 <-- LB4 en 22 bits mas significativos
                st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
                call    T5                  ! Delay de 5 segundos
                sethi   LB5, %r2            ! %r2 <-- LB5 en 22 bits mas significativos
                st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
                call    T30                 ! Delay de 30 segundos
                sethi   LB6, %r2            ! %r2 <-- LB6 en 22 bits mas significativos
                st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
                call    T5                  ! Delay de 5 segundos
                ba      secuenciaA          ! Retornamos a la secuencia A

secuenciaC:     sethi   LC1, %r2            ! %r2 <-- LA2 en 22 bits mas significativos
                add     %r2, BB, %r2        ! %r2 <-- %r2 + BB (mantenemos activo el boton BB)
                st      %r2, [%r1 + IO]     ! 0xD6000020 <-- %r2
chequeoBB:      call    controlBB_d         ! Verificacion de BB desactivado
                ba      chequeoBB           ! Bucle para permanecer en la secuencia C



! Rutina de verificacion de pulsador BB activado
controlBB_a:    ld      [%r1 + IO], %r4     ! %r4 <-- Valor de I/O mapeado en 0xD6000020
                andcc   %r4, BB, %r0        ! Verifica si esta encendido el boton de bomberos
                bne     secuenciaC          ! Salto a la secuencia C
                jmpl    %r15+4, %r0         ! Retornamos a la rutina invocante

! Rutina de verificacion de pulsador BB desactivado
controlBB_d:    ld      [%r1 + IO], %r4     ! %r4 <-- Valor de I/O mapeado en 0xD6000020
                andcc   %r4, BB, %r0        ! Verifica si esta encendido el boton de bomberos
                be      default             ! Salto a la secuencia A
                jmpl    %r15+4, %r0         ! Retornamos a la rutina invocante

! Rutina de verificacion de pulsador BP activado
controlBP_a:    ld      [%r1 + IO], %r4     ! %r4 <-- Valor de I/O mapeado en 0xD6000020
                andcc   %r4, BP, %r0        ! Verifica si esta encendido el boton de peaton
                bne     secuenciaB          ! Salto a la secuencia B
                jmpl    %r15+4, %r0         ! Retornamos a la rutina invocante



! RUTINAS DE TIMERS
! Timers existentes para uso
!   T1  - Timer de 1 segundo
!   T5  - Timer de 5 segundos
!   T30 - Timer de 30 segundos

T1:             add     %r15, %r0, %r16     ! %r16 <-- %r15
                ld      [T1_ciclos], %r5    ! %r5 <-- T1_ciclos
T1_loop:        call    controlBB_a         ! Realizamos control del pulsador BB
                call    controlBP_a         ! Realizamos control del pulsador BP
                addcc   %r5, -1, %r5        ! Decrementamos cantidad restante de ciclos
                be      T_done              ! Finaliza cuando no quedan ciclos por ejecutar
                ba      T1_loop             ! Repetir bucle

T5:             add     %r15, %r0, %r16     ! %r16 <-- %r15
                ld      [T5_ciclos], %r5    ! %r5 <-- T5_ciclos
T5_loop:        addcc   %r5, -1, %r5        ! Decrementamos cantidad restante de ciclos
                be      T_done              ! Finaliza cuando no quedan ciclos por ejecutar
                call    controlBB_a         ! Realizamos control del pulsador BB
                ba      T5_loop             ! Repetir bucle

T30:            add     %r15, %r0, %r16     ! %r16 <-- %r15
                ld      [T30_ciclos], %r5   ! %r5 <-- T30_ciclos
T30_loop:       addcc   %r5, -1, %r5        ! Decrementamos cantidad restante de ciclos
                be      T_done              ! Finaliza cuando no quedan ciclos por ejecutar
                call    controlBB_a         ! Realizamos control del pulsador BB
                ba      T30_loop            ! Repetir bucle

T_done:         jmpl    %r16+4, %r0         ! Retorno a la rutina invocante
```

```
137
138 ! FIN RUTINA TIMER
139
140
141
142
143 ! Botones
144 BP              .equ    0x1                ! Estado activo del boton de cruce de peatones
145 BB              .equ    0x2                ! Estado activo del boton de salida de bomberos
146
147 ! Luces de la secuencia A
148 LA1             .equ    0x0                ! 0x00000000 - Todas las luces apagadas
149 LA2             .equ    0x8080             ! 0x02020000 - A1 y A2 prendidas
150
151 ! Luces de la secuencia B
152 LB1             .equ    0x10080            ! 0x04020000 - A1 y R2 encedidas
153 LB2             .equ    0x10040            ! 0x04010000 - V1 y R2 encendidas
154 LB3             .equ    0x10080            ! 0x04020000 - A1 y R2 encendidas
155 LB4             .equ    0x8100             ! 0x02040000 - R1 y A2 encendidas
156 LB5             .equ    0x4100             ! 0x01040000 - R1 y V2 encendidas
157 LB6             .equ    0x8100             ! 0x02040000 - R1 y A2 encendidas
158
159 ! Luces de la secuencia C
160 LC1             .equ    0x18180            ! 0x06060000 - R1, A1, R2 y A2 encendidas
161
162 ! Timers
163 T1_ciclos:      1                          ! Cantidad de ciclos a ejecutar en T1
164 T5_ciclos:      10                         ! Cantidad de ciclos a ejecutar en T5
165 T30_ciclos:     40                         ! Cantidad de ciclos a ejecutar en T30
166
167                 .end
```

En los siguientes apartados se hará mención de algunos puntos importantes de la implementación, a fin de lograr su completo entendimiento por parte del lector.

3.1. Acceso a la dirección mapeada

Como se ha especificado, los botones y las luces de los semáforos se encuentran mapeados a los bits de la dirección de memoria 0xD6000020. Se puede notar que no es posible acceder a esta dirección en forma directa mediante las instrucciones *ld* y *st*. Esto se debe a que estas permiten como máximo un valor de 13 bits, que se extiende con signo a 32 bits para el segundo registro origen.

Para poder lograr leer o almacenar en dicha dirección de memoria, se ha definido (línea 38) mediante la directiva *.equ* al símbolo *BASE_IO*, el cual contiene un número de 22 bits. Este representa un punto elegido como comienzo de la región mapeada en memoria. Se ha escogido de manera tal de que al ser almacenado en un registro mediante la instrucción *sethi*, estos 22 bits se encuentren en los bits más significativos del mismo. Esto se ha hecho en la línea 41, siendo *%r1* el registro destinado a reservar dicha dirección base.

Debajo de la definición de *BASE_IO* se declara el símbolo *IO* seteado con el valor que le falta a la posición *BASE_IO* para llegar a la dirección 0xD6000020. Por consiguiente, para poder acceder a la dirección donde se encuentran mapeados los estados de los botones y las luces de los semáforos bastará con establecer como dirección fuente o destino (dependiendo de la instrucción utilizada) a la suma del registro *%r1* y el valor del símbolo *IO*, tal como se muestra por ejemplo en las líneas 43 y 46.

3.2. Modificación de las entradas y salidas

De la misma forma en que se nos dificultaba acceder en forma directa a la dirección de memoria donde se encuentran mapeadas las entradas y salidas utilizadas, en este caso nos encontramos con que, al tratar de establecer un nuevo valor en dicha dirección mediante la instrucción *st*, no se nos posibilita almacenar directamente en memoria un valor de 32 bits ya que excede los 13 bits permitidos por la instrucción.

Para resolver esta problemática, primeramente se definieron al final de la implementación, símbolos que representan los distintos estados de las luces en las secuencias de manera tal que posean un valor de 22 bits

como máximo. Este valor se ha armado de manera tal que, al ser almacenado en un registro utilizando la instrucción *sethi*, los valores se encuentren en los 22 bits más significativos de este último. Por ejemplo, el segundo estado de la *secuencia A* debe poseer prendidas las luces A1 y A2, por lo que se necesita establecer en la dirección 0xD6000020 el valor 0x2020000 que en binario tiene un 1 en los bits 17 y 25, correspondientes a las luces amarillas. Entonces, el símbolo *LA2* que interpreta al segundo estado de la secuencia tendrá el valor hexadecimal de los últimos 22 bits desde la derecha del valor 0x2020000, es decir, 0x8080.

Por lo tanto, para almacenar este valor de 22 bits en la posición de memoria deseada, primeramente se deberán almacenar en los 22 bits más significativos de un registro para luego, mediante la instrucción *st*, ser copiado en memoria.

3.3. Timers

Para el manejo de los distintos tiempos intermedios entre estados de las secuencias existentes, se han definido tres rutinas correspondientes a los timers a utilizar, a saber: 1 segundo (T1), 5 segundos (T5) y 30 segundos (T30).

Se ha optado por implementar estos retardos realizando bucles de cierta cantidad de ciclos, los cuales fueron ajustados para obtener los delays esperados. Sin embargo, estos tiempos pueden variar ya que depende directamente de la frecuencia del procesador en donde se esta ejecutando el simulador del que se ha hecho uso para la puesta a prueba del presente desarrollo.

Inicialmente se ha supuesto una frecuencia de 1GHz para el procesador. Además se supuso que cada instrucción necesita de 4 ciclos para ser completada. De esta manera, cada instrucción tardará 4nS en ser ejecutada. Como se puede apreciar en el *Código 1*, la rutina correspondiente al timer de 1 segundo (T1) contiene 13 instrucciones que forman parte del bucle, lo que resulta en 52nS en cada ciclo del bucle. Como se desea obtener un delay de 1 segundo, entonces el bucle se deberá repetir 19.230.769 veces. En cambio, las rutinas correspondientes a los timers de 5 segundos (T5) y 30 segundos (T30) estan conformadas por un bucle de 8 instrucciones cada una. Entonces, para el timer T5 se necesitará recorrer el bucle 156.250.000 veces a fin de obtener un delay de 5 segundos, mientras que para el timer T30 será necesario recorrer el bucle 937.500.000 veces de manera tal de producir un retardo de 30 segundos.

Como se puede notar, la cantidad de repeticiones necesarias para obtener los distintos retardos son excesivamente mayores a los mostrados en el *Código 1* (símbolos *T1_ciclos*, *T5_ciclos* y *T30_ciclos*). Esto se debe, como ya se adelantó párrafos atrás, a que el simulador de ARC utilizado afecta considerablemente en el factor velocidad, además de que el software puede estar siendo ejecutado en una máquina con una frecuencia de procesador totalmente diferente a la supuesta. Por esta razón, en la implementación expuesta se ha establecido un número de ciclos que se ajustan aproximadamente a los delays esperados a la hora de correr el programa y ponerlo a prueba.

4. Comparación de las soluciones

Tanto la solución cableada como la solución programada poseen ventajas y desventajas. Es decir, decidir entre una u otra nos lleva a soluciones de compromisos. Los factores más significativos que establecen una diferencia entre ambos son la *velocidad*, la *facilidad de actualización*, el *costo* y la *complejidad*. Dependiendo de las necesidades que se desean satisfacer, podemos optar por una u otra, siendo estas soluciones perfectamente válidas en distintos marcos de desarrollo.

La solución programada es extensa y lenta pero es flexible y permite una implementación simple, en tanto que la solución cableada es rápida pero difícil de modificar, dando como resultado implementaciones más complejas.

Centrándonos en el sistema solicitado, es posible que la solución programada nos permita establecer un mayor ahorro de costos siendo que se utilizaría una cantidad notablemente menor de componentes electrónicos. A este hecho se le suma la reducción del espacio físico ocupado por el sistema de control. Por último, centrándonos en una hipotética situación de falla del sistema, la solución programada nos permite aminorar los espacios en los que se debe verificar dicha falla, mientras que la revisión del sistema cableado puede precisar una mayor constatación del funcionamiento de los dispositivos que lo conforman.