# 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*

```
! ------------------------------------------------
! 66.70 - Estructura del Computador
! Facultad de Ingenieria
! Universidad de Buenos Aires
!
! Grupo:  8
! Alumnos:  Pantano, Laura Raquel
!   Extramiana, Federico
!   Rossi, Federico Martin
! ------------------------------------------------
! RESOLUCION DEL TP EN ASSEMBLY ARC
! Sistema de semaforos
!
! Direccion de entrada 0xD6000020:
!
!   Bit0: Boton de peaton.
!   Bit1: Boton para salida de bomberos.
!   Bit16: Luz verde semaforo 1.
!   Bit17: Luz amarilla semaforo 1.
!   Bit18: Luz roja semaforo 1.
!   Bit24: Luz verde semaforo 2.
!   Bit25: Luz amarilla semaforo 2.
!   Bit26: Luz roja semaforo 2.
!
! Utilizacion de registros:
!
! %r1 - Direccion base de la region I/O de la memoria
! %r2 - Registro auxiliar para armado de estados a activar
! %r5 - Contador de ciclos para el timer T1
! %r6 - Contador de ciclos para el timer T2
! %r7 - Contador de ciclos para el timer T3
!

    .begin
    .org  2048
BASE_IO .equ  0x358000    ! Punto de comienzo de la region mapeada en memoria
IO    .equ  0x20      ! 0xD6000020 Posicion de memoria donde se encuentran
            ! mapeadas las luces y pulsadores

    sethi BASE_IO, %r1    ! %r1 <-- Direccion base de la region I/O de memoria
    ba  secuenciaB
secuenciaA: !call controlBB_a   ! Realizamos control del pulsador BB
    !call controlBP_a   ! Realizamos control del pulsador BP
    and %r1, %r0, %r2   ! %r2 <-- 0x0 - Estado de luces LA1
    st  %r2, [%r1 + IO] ! 0xD6000020 <-- %r2
    call  T1      ! Delay de 1 segundo
    and %r2, %r0, %r2   ! %r2 <-- 0x0
    sethi LA2, %r2    ! %r2 <-- LA2 en 22 bits mas significativos
    st  %r2, [%r1 + IO] ! 0xD6000020 <-- %r2
    call  T1      ! Delay de 1 segundo
    ba  secuenciaA    ! Reiniciamos secuencia A

secuenciaB: and %r2, %r0, %r2   ! %r2 <-- 0x0
    sethi LB1, %r2    ! %r2 <-- LB1 en 22 bits mas significativos
    st  %r2, [%r1 + IO] ! 0xD6000020 <-- %r2
    call  T5      ! Delay de 5 segundos
    and %r2, %r0, %r2   ! %r2 <-- 0x0
    sethi LB2, %r2    ! %r2 <-- LB2 en 22 bits mas significativos
    st  %r2, [%r1 + IO] ! 0xD6000020 <-- %r2
    call  T30     ! Delay de 30 segundos
```

```
    and %r2, %r0, %r2   ! %r2 <-- 0x0
    sethi LB3, %r2    ! %r2 <-- LB3 en 22 bits mas significativos
    st  %r2, [%r1 + IO] ! 0xD6000020 <-- %r2
    call  T5      ! Delay de 5 segundos
    and %r2, %r0, %r2   ! %r2 <-- 0x0
    sethi LB4, %r2    ! %r2 <-- LB4 en 22 bits mas significativos
    st  %r2, [%r1 + IO] ! 0xD6000020 <-- %r2
    call  T5      ! Delay de 5 segundos
    and %r2, %r0, %r2   ! %r2 <-- 0x0
    sethi LB5, %r2    ! %r2 <-- LB5 en 22 bits mas significativos
    st  %r2, [%r1 + IO] ! 0xD6000020 <-- %r2
    call  T30     ! Delay de 30 segundos
    and %r2, %r0, %r2   ! %r2 <-- 0x0
    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: and %r2, %r0, %r2   ! %r2 <-- 0x0
    sethi LC1, %r2    ! %r2 <-- LA2 en 22 bits mas significativos
    add %r2, BB, %r2    ! %r2 <-- %r2 + BB (mantenemos activo el bit de BB)
    st  %r2, [%r1 + IO] ! 0xD6000020 <-- %r2
secuenciaC_:  !call controlBB_d   ! Verificacion de BB desactivado
    ba  secuenciaC_   ! Bucle para permanecer en la secuencia C



! Rutina de verificacion de pulsador BB activado
controlBB_a:  ld  [%r1 + IO], %r2 ! %r2 <-- Valor de I/O mapeado en 0xD6000020
    and   %r2, BB, %r0    ! Verifica si esta encendido el boton de bomberos
    be  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], %r2 ! %r2 <-- Valor de I/O mapeado en 0xD6000020
    be  secuenciaA    ! 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], %r2 ! %r2 <-- Valor de I/O mapeado en 0xD6000020
    and   %r1, BP, %r0    ! Verifica si esta encendido el boton de peaton
    be  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:   ld  [T1_ciclos], %r5  ! %r5 <-- T1_ciclos
    ba  T_loop      ! Iniciamos timer
T5:   ld  [T5_ciclos], %r5  ! %r5 <-- T5_ciclos
    ba  T_loop      ! Iniciamos timer
T30:    ld  [T30_ciclos], %r5 ! %r5 <-- T30_ciclos
    ba  T_loop      ! Iniciamos timer

T_loop: andcc %r5, %r5, %r0   ! Verificar cantidad restante de ciclos
    be  T_done      ! Finaliza cuando no quedan ciclos por ejecutar
    sethi 0, %r0      ! No operation (NOP)
    add %r5, -1, %r5    ! Decrementamos cantidad restante de ciclos
    ba  T_loop      ! Repetir bucle
T_done:   jmpl  %r15+4, %r0   ! Retorno a la rutina invocante

! FIN RUTINA TIMER



! Botones
BP    .equ  0x1     ! Estado activo del boton de cruce de peatones
BB    .equ  0x2     ! Estado activo del boton de salida de bomberos
```

```
137
138 ! Luces de la secuencia A
139 LA1   .equ  0x0     ! 0x00000000 - Todas las luces apagadas
140 LA2   .equ  0x8080      ! 0x02020000 - A1 y A2 prendidas
141
142 ! Luces de la secuencia B
143 LB1   .equ  0x10080   ! 0x04020000 - A1 y R2 encedidas
144 LB2   .equ  0x10040   ! 0x04010000 - V1 y R2 encendidas
145 LB3   .equ  0x10080   ! 0x04020000 - A1 y R2 encendidas
146 LB4   .equ  0x8100      ! 0x02040000 - R1 y A2 encendidas
147 LB5   .equ  0x4100      ! 0x01040000 - R1 y V2 encendidas
148 LB6   .equ  0x8100      ! 0x02040000 - R1 y A2 encendidas
149
150 ! Luces de la secuencia C
151 LC1   .equ  0x18180   ! 0x06060000 - R1, A1, R2 y A2 encendidas
152
153 ! Timers
154 T1_ciclos:  1       ! Cantidad de ciclos a ejecutar en T1
155 T5_ciclos:  12        ! Cantidad de ciclos a ejecutar en T5
156 T30_ciclos: 100       ! Cantidad de ciclos a ejecutar en T30
157
158     .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 36) 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 40, 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 45 y 90.

## 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.

## 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.