UNIVERSIDAD DE GRANADA.

ESCUELA TECNICA SUPERIOR DE INGENIERIAS INFORMATICA Y DE TELECOMUNICACIÓN.

Departamento de Arquitectura y Tecnología de Computadores.

TECNOLOGÍA Y ORGANIZACIÓN DE COMPUTADORES.

PRÁCTICA 8.

DESCRIPCIÓN A NIVEL RT DE UN COMPUTADOR SENCILLO.

11 GRADO EN INGENIERÍA INFORMÁTICA.

PRÁCTICA 8.

DESCRIPCIÓN A NIVEL RT DE UN COMPUTADOR SENCILLO REALIZADO EN LogicWorks 4.1.

Objetivos:

- *Experimentar con el computador sencillo CS1_LogicWorks_versión 1.b en LogicWorks (versión 4.1). El diseño de este computador es descrito teóricamente en el Tema 5.*
- *En esta práctica, se muestra cómo realizar e introducir en CS1_LogicWorks un programa, ejecutarlo y analizar su comportamiento. También, sus posibilidades para realizar seguimientos de las microoperaciones que tienen lugar durante la ejecución un programa a nivel de cada ciclo máquina.*

8.1 Introducción.

En esta sección describimos la implementación y organización de CS1 en LogicWorks (versión 4.1), mostrando sus diferentes "vistas" para seguir la ejecución de un programa. Antes de nada hay que descargar ***CS1_VersiónEstudiante_LW4_Vb*** desde el directorio donde está este guión de prácticas en la plataforma SWAD. El fichero está comprimido (tipo *.zip). Hay que descomprimirlo para obtener el fichero en formato ****.cct*** que es el que acepta el simulador LogicWorks 4 (LW4 versión 4.1). A a partir de ahora usaremos la abreviatura **CS1** para referirnos al circuito ***CS1_VersiónEstudiante_LW4_Vb.cct***.

Una vez abierto CS1 en el simulador LW4, observaremos diferentes vistas de CS1. La ***primera vista*** es general (ver Figura 8.1), en donde podemos observar la organización global de la unidad de procesamiento, UP, con visualizadores que muestran el contenido de los registros en hexadecimal, tales como: contador de programa PC, registro de direcciones MAR, registro de instrucciones IR, registro temporal RT y acumulador AC.

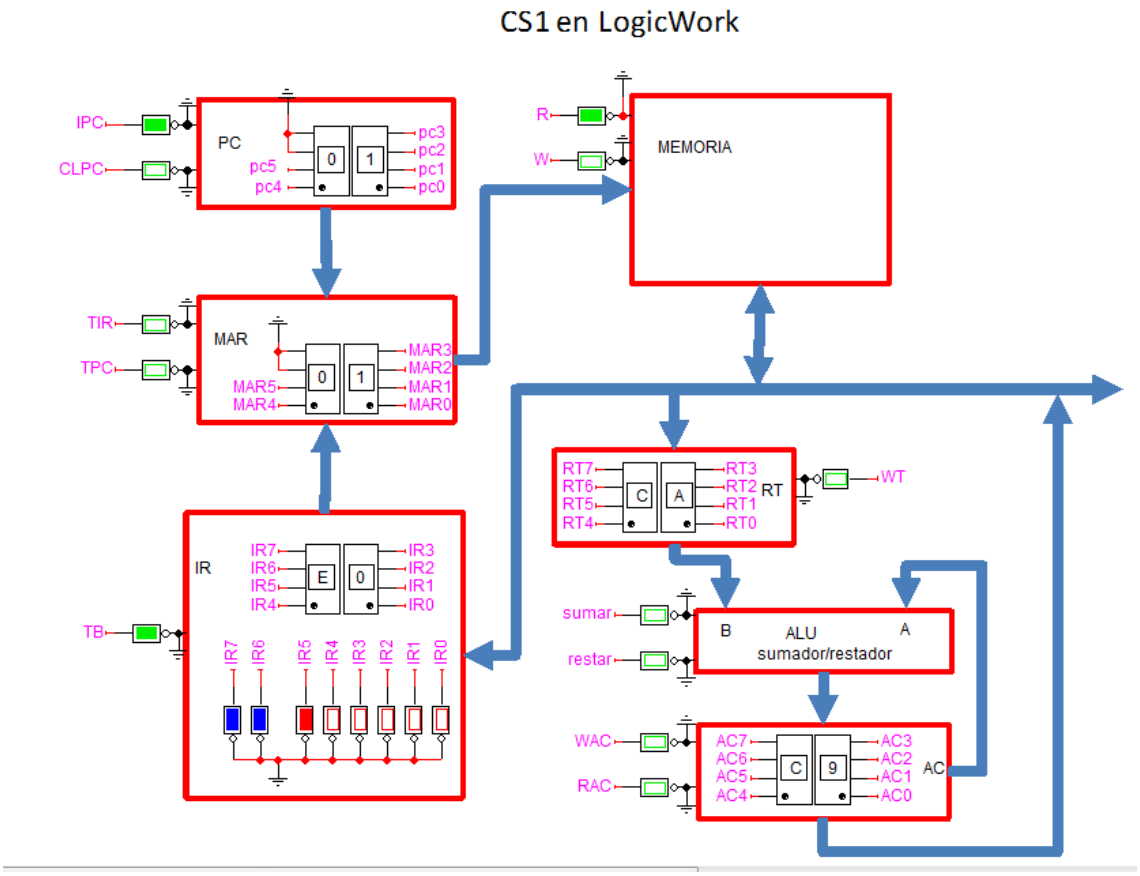


Figura 8.1

También, se visualizan con "leds" las señales de control de cada componente de la unidad de procesamiento. Estas señales las activa la unidad de control. Durante la ejecución de un programa, veremos cómo cambian los contenidos de los registros y las señales de control, en cada ciclo de reloj.

La ***segunda vista*** de CS1 (ver Figura 8.2(a)), es la organización real y detallada de los componentes y buses que conforman la unidad de proceso. Si seleccionamos un componente y hacemos doble "Clic" con el botón izquierdo del ratón, LogicWorks mostrara la realización de dicho componente. Hay que tener en cuenta, que en la "Versión Estudiante" de CS1, el circuito que implementa realmente cada componente ha sido realizado tomando primitivas de la librería "Simulation Logic.clf" del LogicWorks, y no es objetivo de análisis por parte del estudiante. Para dicho análisis, se incluye una "imagen" (no es un circuito) de la realización que debería tener el componente, según lo explicado en teoría en los temas dedicados al análisis y diseño de sistemas secuenciales síncronos. Esto se hace para "invitar" al estudiante (si lo desea y

como actividad extra) a su realización y posterior comprobación, mediante la sustitución de dicho componente por el diseñado por el estudiante. En la figura 8.2(b) se presenta un ejemplo del contenido "imagen" del registro temporal RT.

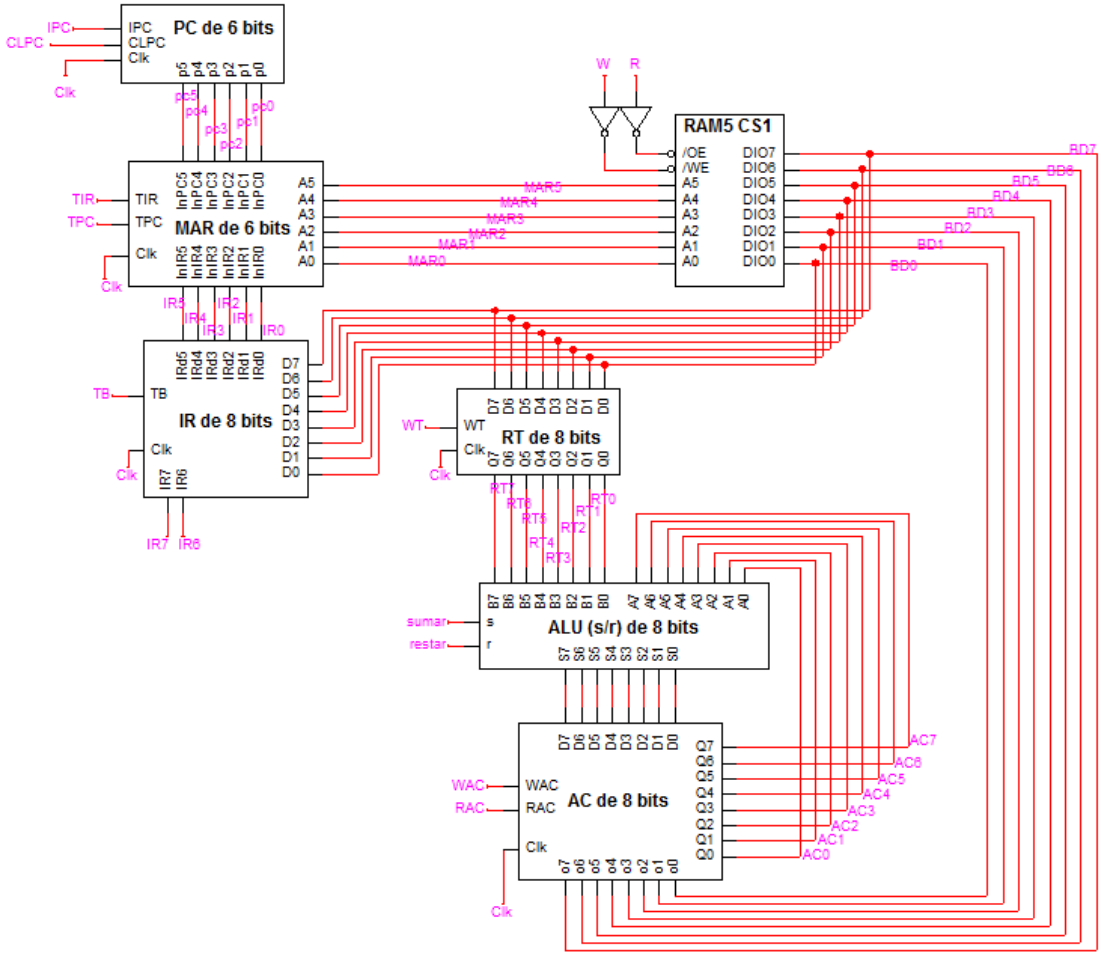


Figura 8.2(a) Unidad de Proceso detallada en LogicWorks.

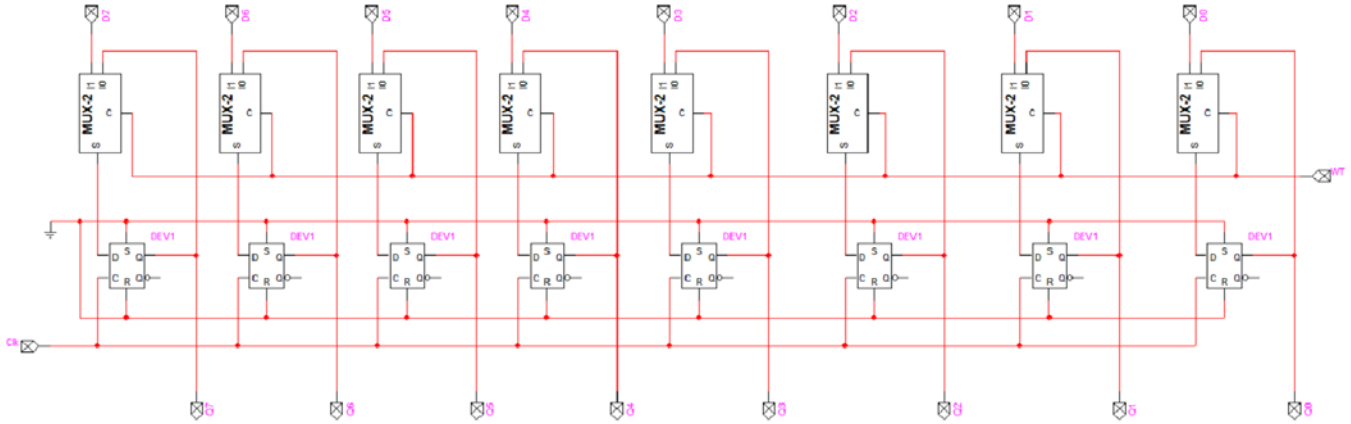


Figura 8.2(b)Vista de la "imagen" que contiene el componente: Registro RT.

Respecto al contenido y edición de la memoria RAM5 CS1, LogicWorks dispone de la siguiente forma de hacerlo. Primero seleccione el módulo RAM5 CS1 con un clic en el botón derecho del ratón, después nos vamos al icono que se muestra en la figura 8.3(a).

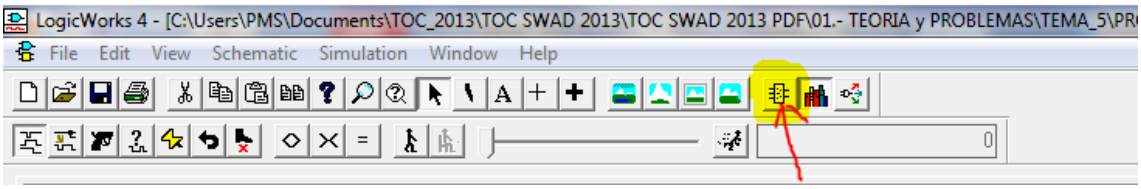


Figura 8.3(a)

Al pulsarlo obtendremos la vista de la figura 8.3(b). Pulsaremos en la opción "Siguiente". Después de esto, nos aparece otro cuadro de dialogo, como el de la figura 8.3(c) . Volvemos a seleccionar "Siguiente", y aparece el cuadro de la figura 8.3(d), que muestra el contenido en hexadecimal de la memoria.

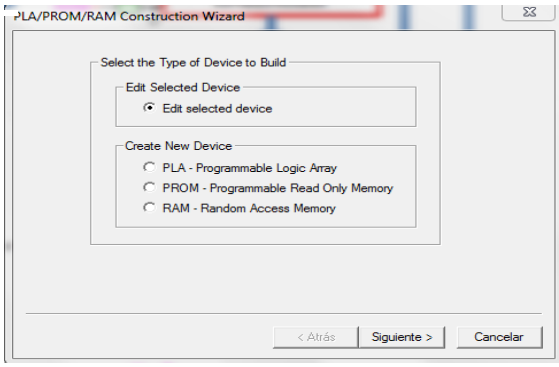


Figura 8.3(b)

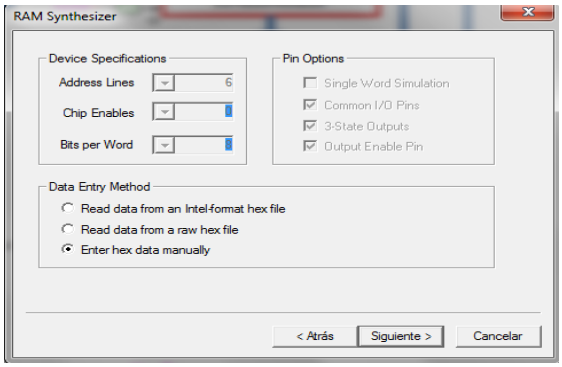


Figura 8.3(c)

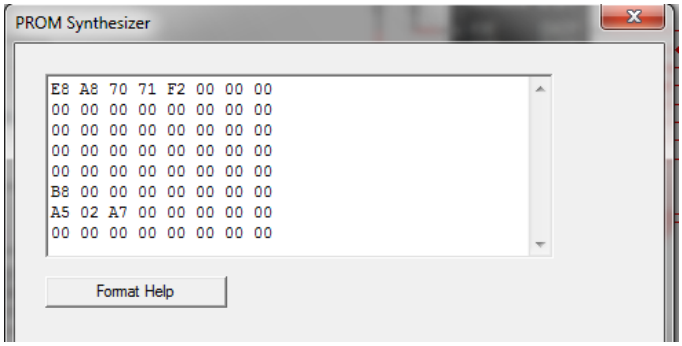


Figura 8.3(d)

Desde este cuadro, podemos situarnos con el ratón y editar los contenidos de la memoria. Las direcciones de memoria son consecutivas, de modo que las dos primeras filas corresponden a direcciones de 00 a 0F, el segundo par de filas, de 10 a 1F, el tercer par de filas, de 20 a 2F, y por último, el cuarto par de filas a direcciones de 30 a 3F. Como ejemplo, podemos ver que el dato B8, que aparece en la figura 8.3(d), es un dato ubicado en la segunda fila del tercer par y primera columna, por tanto su dirección es la 28 en hexadecimal.

Para facilitar qué posiciones de memoria en hexadecimal corresponden a los contenidos almacenados en la RAM5 CS1 de LogicWorks, proporcionamos la figura 8.3(d).

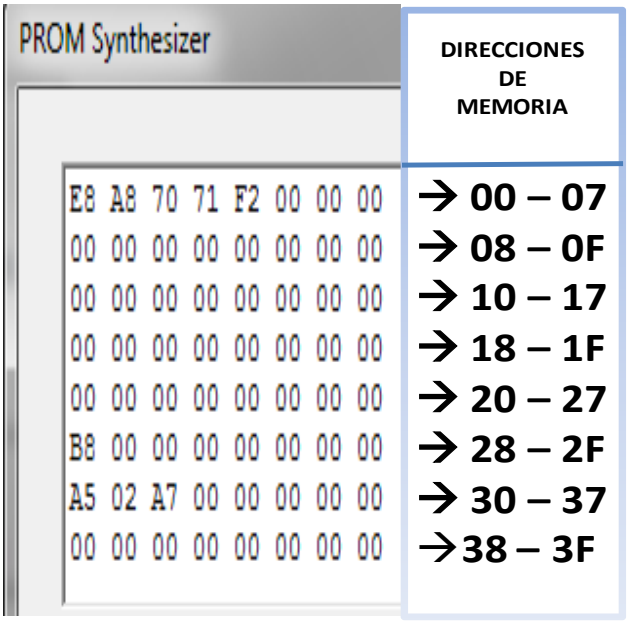


Figura 8.3(d.)

La ***tercera vista*** de CS1 (ver figuras 8.4) se dedica a la unidad de control. En la figura 8.4(a) se ve la realización en LogicWorks de la unidad de control, cuyo diseño se explica en el Tema 5. Junto a esta unidad de control se han añadido diversos elementos (ver figura 8.4(b)), como visualizadores leds para ver, en cada ciclo de reloj, el estado y bloque ASM en el que se encuentra dicha unidad. También, se proporciona una "imagen" de la Carta ASM de procesado junto con las señales de control que se activan. Mediante visualizadores leds, se pueden ver las señales de control que se activan en cada ciclo de reloj y su efecto en la unidad de procesamiento. Para ello, se adjuntan textos junto con a dichos visualizadores para indicar la/s microoperación/es de transferencia a registros que se van a producir como consecuencia de las señales de control que estén activas en un determinado ciclo de reloj. Se puede, por tanto, realizar un seguimiento minucioso de las fases de captación y ejecución de las instrucciones del programa que se esté ejecutando. Dicho programa tiene que estar almacenado en la memoria RAM5_CS1 a partir de la dirección cero, siendo la última instrucción la de STOP.

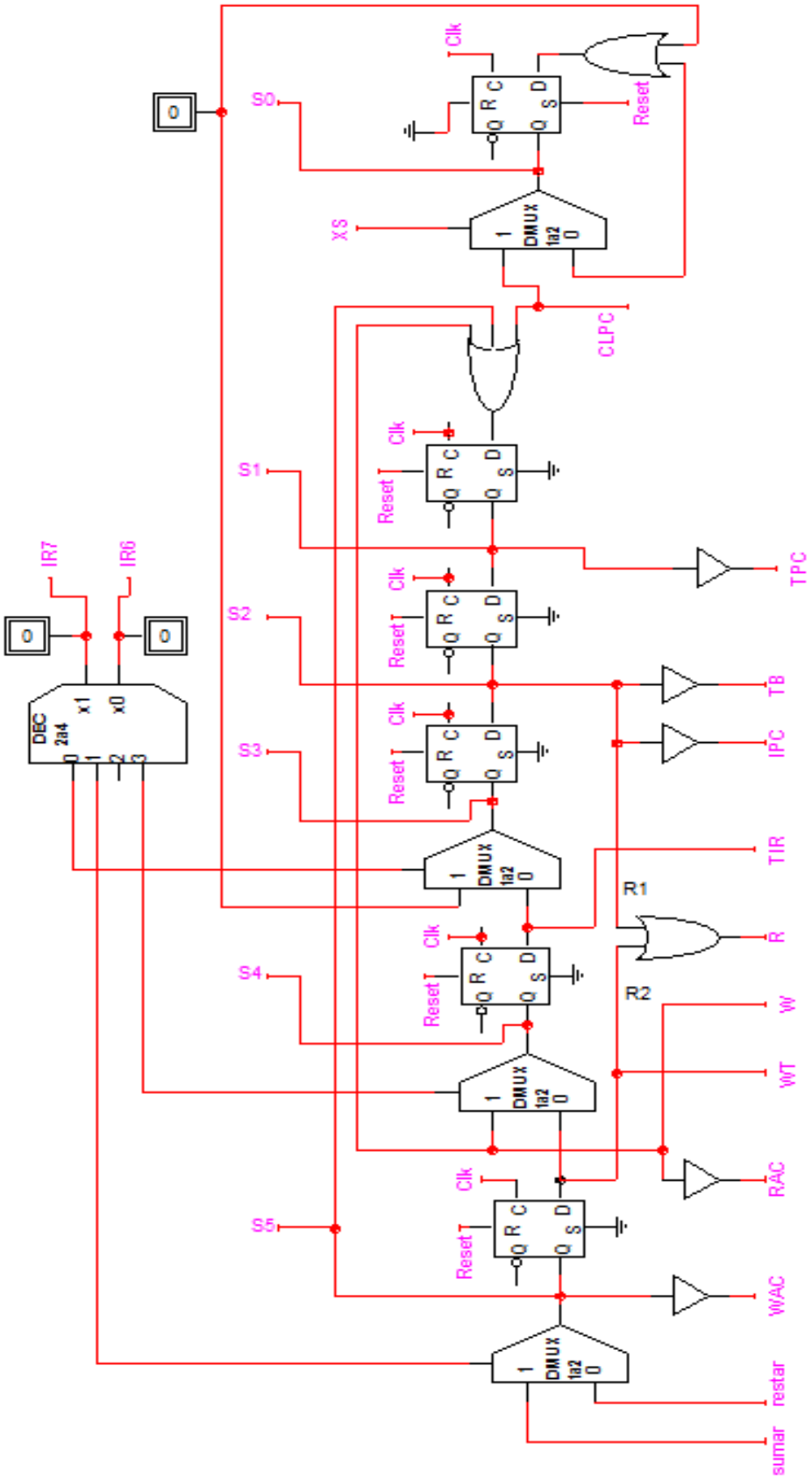


Figura 8.4(a.). Unidad de control de CS1 en LogicWorks 4.1

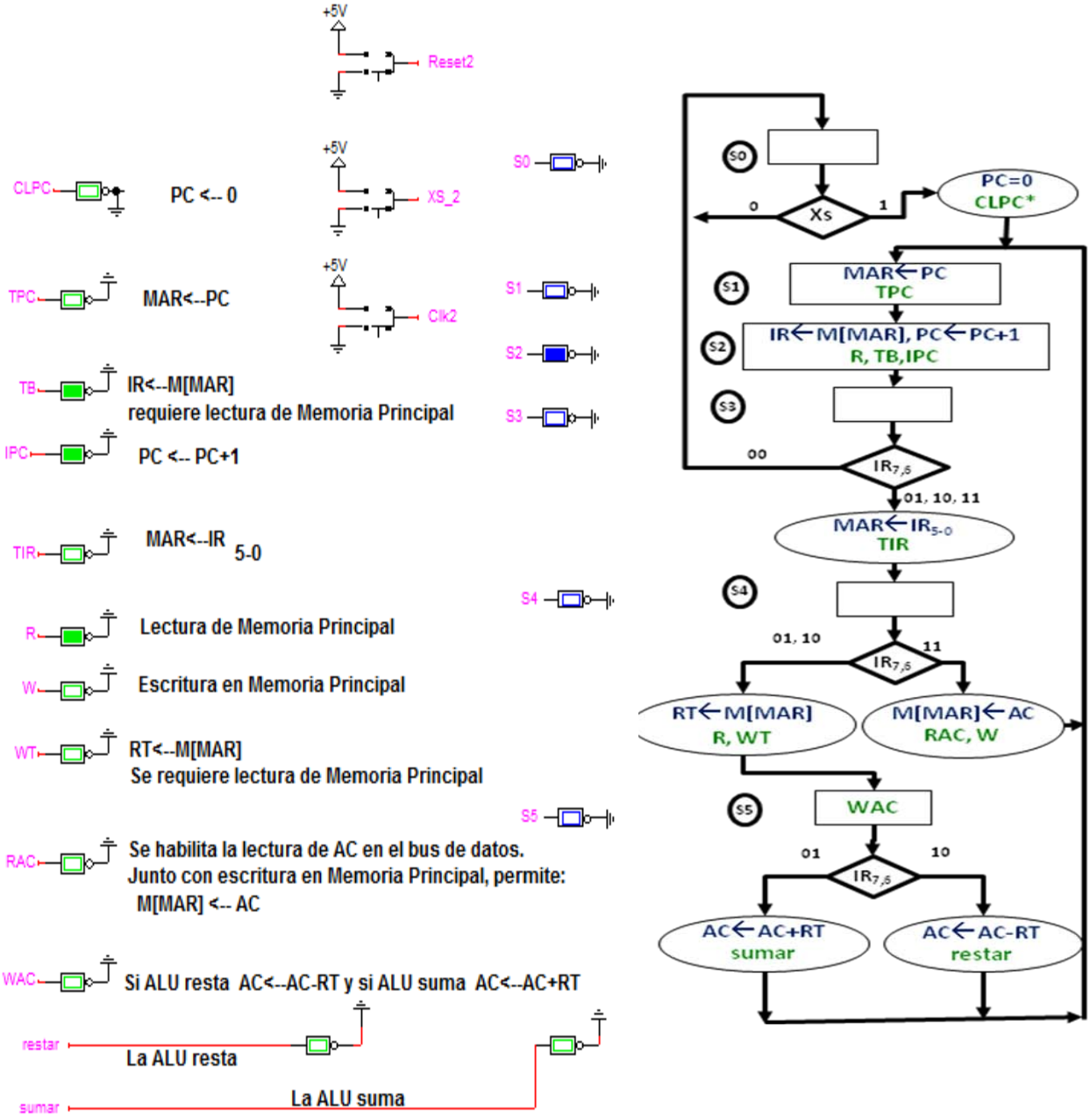


Figura 8.4(b). Unidad de control de CS1 en LogicWorks 4.1

8.2 Cómo realizar un programa y cargarlo en memoria.

Lo primero es definir en ensamblador, con el repertorio de instrucciones de CS1 (ver *Figura 8.4(a)*), el programa que deseamos realizar.

Ensamblador ($DirDato en hexadecimal)	Descripción RT	Formato de la Instrucción en binario	
		CO	Dirección del Dato en binario
STOP	Fin ejecución	00	X X X X X
ADD $DirDato	AC ← AC + M($DirDato)	01	A_5 A_4 A_3 A_2 A_1 A_0
SUB $DirDato	AC ← AC - M($DirDato)	10	A_5 A_4 A_3 A_2 A_1 A_0
STA $DirDato	M($DirDato) ← AC	11	A_5 A_4 A_3 A_2 A_1 A_0

Figura 8.4(a) Repertorio de Instrucciones de CS1

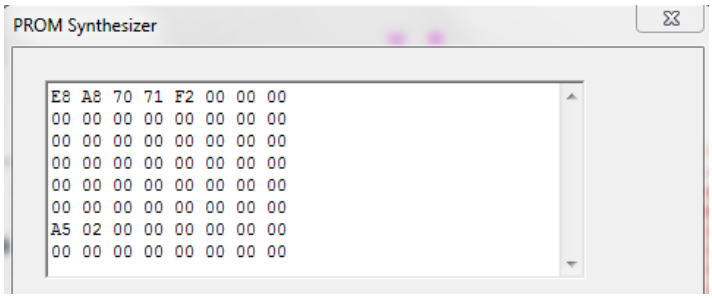
Por ejemplo supongamos que se desea realizar un programa que llamaremos **ProgSUMA2** que sume dos datos que están en las direcciones, dadas en hexadecimal, **30 y 31** y que el resultado se almacene en la dirección **32**. Una solución para para el programa **ProgSUMA2** viene dada en la *Figura 8.4(b)*. Obsérvese que las dos primeras instrucciones, sirven para poner el acumulador AC a cero. Para ello, la primera instrucción almacena en la **dirección 28** de memoria el contenido actual del acumulador (**datAux**) y la segunda instrucción resta este valor con el mismo dato que fue almacenado en la **dirección 28**, consiguiendo dejar el acumulador a cero. La tercera instrucción suma el **cero** del acumulador con **dat1** que hay almacenado en memoria en la **dirección 30**, y se carga en el acumulador. La cuarta instrucción suma **dat1** del acumulador con **dat2** que hay almacenado en memoria en la **dirección 31**, por tanto en el acumulador ya tenemos el resultado deseado, **datResul=dat1+dat2**, que se carga en el acumulador. Por último, la quinta instrucción almacena el resultado (**datResul**) en la **dirección** de memoria **32**.

Programa en ensamblador ($DirDato en hexadecimal)	Descripción RT del programa ProgSUMA2	Instrucción en binario		Instrucción en hexadecimal
		CO 2 bits	Dirección del dato en binario con 6 bits	
STA $28	M($DirDato de datAux=28) ← AC	11	10 1000	E8
SUB $28	AC ← AC - M($DirDato de datAux=$28)	10	10 1000	A8
ADD $30	AC ← AC + M($DirDato de dat1=$30)	01	11 0000	70
ADD $31	AC ← AC + M($DirDato dat2=$31)	01	11 0001	71
STA $32	AC ← AC + M($DirDato datResul=$32)	11	11 0010	F2
STOP	Fin ejecución	00	----------	00

*Figura 8.4(b) Programa **ProgSUMA2.***

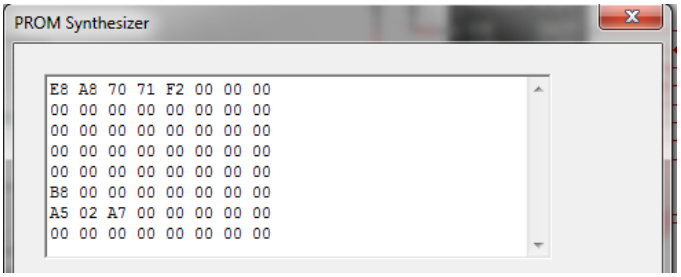
Para cargar el programa en memoria hay que editar la memoria RAM5 CS1, con la utilidad del LogicWorks, descrita en la sección 8.1 (ver figuras 8.3). El programa debe almacenarse en direcciones consecutivas de memoria partiendo de la dirección 0. Por último, hay que introducir en memoria los datos con los que va a operar el programa. En nuestro ejemplo, vamos a suponer que **dat1=A5** (**en la dirección 30**) y **dat2=02** (**en la dirección 31**). La figura 8.4(c) muestra, cómo deben quedar los contenidos de memoria correspondientes a las direcciones (en hexadecimal): **de la 00 a la 05, donde se almacenan las instrucciones del programa** y en **las direcciones 30 y 31 los datos de entrada dat1=A5 y dat2=02** (no importa el contenido de la memoria en el resto de las direcciones).

Figura 8.4(c)



Después de ejecutar el programa **ProgSUMA2**, tendríamos que observar el resultado correcto, almacenado en la **dirección 32** de la memoria, como el obtenido en la Figura_8.4(d). Observad que en la **dirección 28** aparece **B8,** que corresponde a **datAux**, es decir el valor que tenía el acumulador antes de lanzar la ejecución del programa.

Figura 8.4(d)



8.2 Ejecución del programa y su análisis con CS1 en LogicWorks.

Una vez tenemos en memoria el programa y los datos de entrada, podemos ejecutarlo. La primera vez que abrimos CS1 en LogicWorks, es conveniente activar el pulsador de "Reset1 o Reset2" (ver figura 8.5), esto lleva a CS1 al estado de **espera S0**. Lo siguiente es definir si queremos utilizar **modo manual (paso a paso)** o **automático**

para la señal de reloj. En general, el modo paso a paso interesa para analizar con detalle todos los eventos que se producen en cada ciclo de reloj. El modo automático, está orientado a ejecutar el programa de una manera rápida. También, para obtener cronogramas del funcionamiento de CS1. Para iniciar la ejecución de un programa, **pulsamos XS_1** o **XS_2**, este pulso, entra en el módulo XS, que genera un pulso sincronizado que se envía a CS1. Esto hace que el sistema, deje de esperar en el estado **S0** y pase a los siguientes estados. Cuando la ejecución del programa termina, CS1 vuelve al **estado de espera S0**, esperando una nueva pulsación en **XS_1** o **XS_2**, para iniciar otra ejecución de un programa.

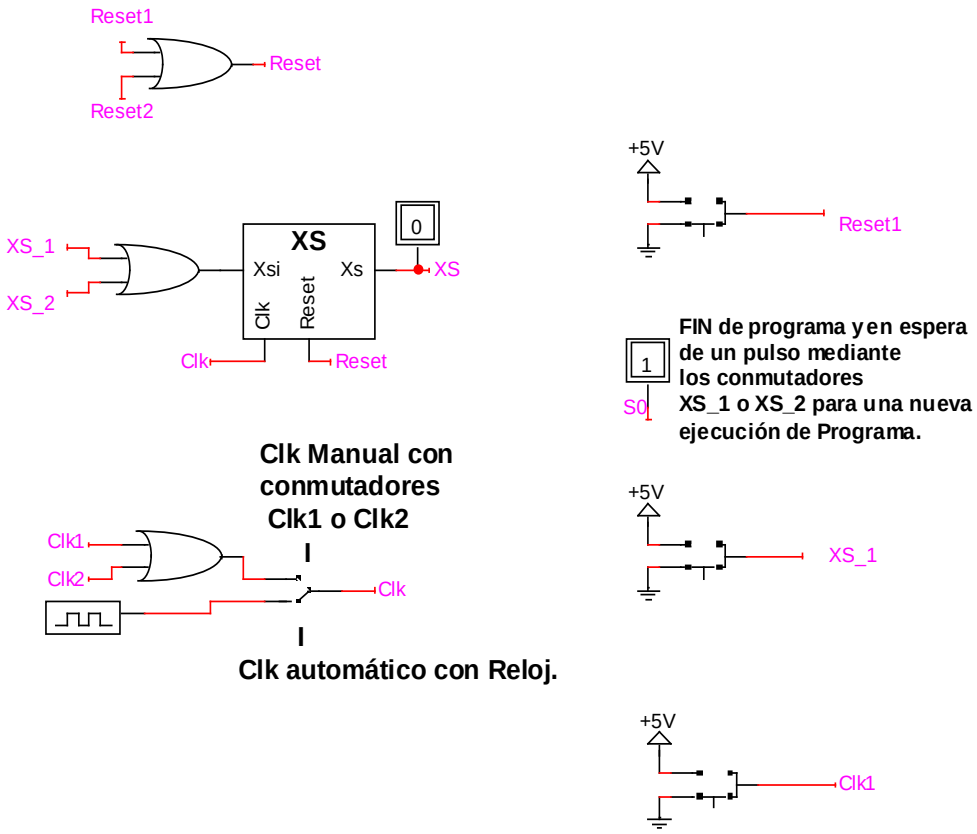


Figura 8.5

El análisis y seguimiento del programa se puede obtener a través de las 3 "vistas" de CS1 explicadas en el apartado 8.1, incluyendo la información dada en los cronogramas.

8.3 Realización de la Práctica 8.

1) En el computador original CS1 suministrado en SWAD, una vez descomprimido, "CS1_VersiónEstudiante_LW4_Vb.cct", se incluye una memoria RAM5_CS1 que almacena ceros en todas las posiciones de memoria. Si ejecutamos CS1, el computador interpreta, como programa almacenado, una sola instrucción que es la STOP.
a) Ejecute dicho programa y vea su cronograma. Obsérvese los dos ciclos de captación de instrucción y un ciclo más, que consiste en llevar a CS1 al estado de espera S0.
b) Edite ahora RAM5_CS1, colocando en la dirección 00 la instrucción 0F y ejecute el programa. ¿Observa alguna diferencia respecto al apartado a)?, ¿Qué instrucción es 0F?, ¿Qué código de operación tiene?.

2) Edite en la memoria RAM5_CS1 el programa **ProgSUMA2**, descrito en ***el apartado 8.2***, con los **datos de entrada dat1=A5 y dat2=02** en **las direcciones 30 y 31,** respectivamente**. a)** Ejecute el programa. Después de esto, vea el contenido de la RAM5_CS1, correspondiente a la **dirección 32,** donde se almacena el resultado. ¿Es correcto el resultado obtenido?. **b)** Obtenga su cronograma con LogicWorks, y deduzca a través de él lo siguiente: **b1)** Número de ciclos de reloj que consume el programa, **b2)** ***Las microoperaciones*** que se producen a nivel de transferencia a registros, durante la fase de ejecución de la instrucción ***STA $32*** (*M($32)*$\leftarrow$*AC*).
c) Realice otras ejecuciones de **ProgSUMA2,** cambiando los datos **dat1** y **dat2**, comprobando los diferentes resultados que se obtienen en la **dirección 32** de la memoria.

3) Se desea confeccionar una versión del programa ProgSUMA2, que opere con datos de entrada dat1 y dat2 que estén ubicados en direcciones distintas. Concretamente, dat1 en la dirección 38 y dat2 en la dirección 39. Además, deseamos que el resultado se almacene en la dirección 3A. A dicho programa lo denominaremos ***ProgSUMA2_vb***.
a) Realice una tabla similar a la indicada en la *Figura 8.4(b)* para el nuevo programa.
b) Edite en la memoria RAM5_CS1, las nuevas instrucciones en hexadecimal, a partir de la direeción 00, así como los datos dat1=A5 y dat2=02 en las nuevas direcciones indicadas.
c) Ejecute el programa ProgSUMA2_vb y compruebe que el resultado almacenado en memoria es correcto.