

# **Bloque 1**

## **Introducción a la CPU elemental**



Prácticas de  
Introducción a los computadores  
**Curso 2007-2008**

## **SESIÓN 1**

# **Introducción al simulador del computador elemental**

## **Objetivos**

En esta sesión se pretende que el alumno se familiarice con el uso del simulador del computador elemental.

En esta primera toma de contacto se aprenderá a manejar los registros y la memoria, se estudiará el funcionamiento básico del computador elemental (codificación y ejecución de instrucciones), se introducirán dos de los tipos de archivos que puede manejar la aplicación: archivos de simulación y archivos de memoria y se ejecitará la codificación de instrucciones y los modos de direccionamiento de la CPU elemental.

Es importante que comprendas lo que está ocurriendo cada vez que realizas una acción y que puedas prever el resultado de las mismas antes de que llevarlas a cabo.

## **Conocimientos y materiales necesarios**

Para poder realizar esta sesión el alumno debe:

- Acudir al laboratorio con una copia del juego de instrucciones de la CPU (disponible en la sección de material de teoría de la página web de la asignatura). También debe conocer el manejo de esta tabla para codificar las instrucciones de la CPU elemental.
- Conocer el funcionamiento básico de una arquitectura Von Neumann.

## Desarrollo de la práctica

### 1. Ejecución de instrucciones

Siguiendo las instrucciones de tu profesor de prácticas, arranca el programa simulador del computador elemental. Cuando lo hayas hecho, aparecerá la pantalla principal de la aplicación, en la que verás un diagrama de bloques de los principales elementos de una arquitectura Von Neumann: ALU, registros, memoria, módulos de E/S y buses. Los números de cuatro cifras que aparecen en diversos controles son números hexadecimales. Fíjate en los registros: todos están inicializados al valor 0000 excepto uno. ¿Cuál es?<sup>[1]</sup>

1

Vamos a modificar los registros R3 y R5 para que tengan los valores 2FE3h y 1A5Bh respectivamente. Para ello:

- ☐ Pulsa sobre el registro R3.
- ☐ En la ventana que aparecerá introduce el valor 2FE3h y luego pulsa el botón . (No es necesario añadir la h porque el simulador trabaja sólo con datos hexadecimales.)
- ☐ Siguiendo los mismos pasos, modifica el registro R5 para que tenga el valor 1A5Bh.

Fíjate en que los únicos valores que puedes modificar directamente de esta forma son los de los registros R0-R7 y PC. El valor del resto de los registros (MAR, MDR, IR...) lo asigna la aplicación durante la simulación.

Vamos a trabajar ahora con la memoria. Como sabes, la memoria se utiliza para almacenar instrucciones y datos. Vamos a utilizar la memoria para almacenar el código de la instrucción MOV R5, R3. Para ello:

- ☐ En primer lugar debes codificarla utilizando la tabla de codificación de instrucciones de la CPU elemental. Tras la codificación obtendrás un número hexadecimal de 4 cifras.
- ☐ Vamos a colocar ese número en la posición de memoria a la que apunte PC. Mira el valor de PC y anótalo<sup>[2]</sup>.
- ☐ Pulsa con el ratón sobre el dibujo etiquetado con MEMORIA (abajo a la izquierda). Aparecerá un menú en el que debes seleccionar la opción *Editor hexadecimal*.
- ☐ Se abrirá una ventana en la que se muestra dos listas de números. A la izquierda aparecen las direcciones de memoria y a la derecha el contenido de cada dirección. Se puede utilizar la barra de scroll para desplazarse por la memoria, pero para ir a una dirección concreta es más fácil utilizar el botón Ir a dirección... Pulsa sobre él.

2

- ❑ En la pantalla que te aparecerá escribe el valor que habías apuntado del PC.
- ❑ Ahora en la fila superior del editor hexadecimal estarás viendo el contenido de dirección de memoria a la que apunta el PC. Como es ahí donde queremos que esté la codificación de la instrucción `MOV R5, R3`, haz doble clic sobre ese número y, en la ventana que verás, escribe la instrucción codificada.

Vamos a comprobar que has codificado bien la instrucción:

- ❑ Cierra la ventana del editor hexadecimal pulsando el botón Aceptar.
- ❑ Vuelve a pulsar sobre el dibujo de la memoria pero ahora escoge la opción *Desensamblador* en el menú que te sale. Aparecerá una ventana muy similar a la del editor hexadecimal pero con una columna más para cada posición de memoria que contiene el mnemónico de la instrucción que se corresponde con el contenido de cada posición de memoria.
- ❑ Vete a la posición a la que apunta PC y comprueba que el mnemónico y los operandos de la instrucción en esa posición son los que deseamos: `MOV R5, R3`. Si no es así, repasa la codificación que has hecho, corrígela e introduce la nueva codificación en la posición a la que apunta PC (deberás usar de nuevo el editor hexadecimal).

Cuando llegues a este punto, R3 tendrá el valor 2FE3h, R5 el valor 1A5Bh y PC estará apuntando a la posición de memoria donde estará la codificación de la instrucción `MOV R5, R3`. Vamos a hacer que la CPU ejecute esta instrucción.

Recuerda que la ejecución de una instrucción consiste en traer esa instrucción de memoria, decodificarla en IR y hacer lo que indique su codificación. La instrucción que se trae es precisamente aquella que haya en la dirección donde apunta PC; por lo tanto, en cuanto le digamos al simulador que ejecute una instrucción ejecutará exactamente la que queremos que ejecute. Antes de hacerlo, contesta a las siguientes preguntas:

- ¿Qué valor tendrán R5<sup>[3]</sup> y R3<sup>[4]</sup> después de la ejecución?
- ¿Qué valor tendrá IR?<sup>[5]</sup>
- ¿Qué valor tendrá PC?<sup>[6]</sup>

3

4

5

6

Para ejecutar una instrucción puedes ir al menú *Ejecución* y escoger la opción *Instrucción completa*, que como verás tiene asignada la tecla rápida F8. Escoge esa opción. ¿Han quedado los registros como habías previsto?

Vamos a ejecutar ahora una instrucción aritmética: `ADD R0, R5, R3`. Codifícala y colócala en la posición de memoria donde creas que deba ir para que se ejecute cuando pulses F8. Antes de ejecutarla, contesta esta pregunta: ¿qué registro se modificará tras la ejecución de la instrucción y qué valor contendrá?<sup>[7]</sup> Comprueba después de ejecutar la instrucción si tu respuesta fue correcta.

7

## 2. Archivos .sim

El simulador permite en cualquier momento almacenar el estado de una simulación. Para ello utiliza archivos con extensión `.sim`. Vamos a probar esta característica.

- ❑ Escoge la opción de menú *Archivo*→*Guardar como*. Te saldrá el cuadro de diálogo típico para guardar. Escoge tu carpeta de usuario y pon como nombre `1-1cpu`. El programa le añadirá la extensión `.sim`.
- ❑ Fíjate en cómo están todos los registros de la CPU.
- ❑ Para comprobar que el archivo guarda correctamente los datos, vamos a reiniciar la CPU, mediante el menú *Ejecución*→*Reiniciar*.
- ❑ Fíjate en que todos los registros han vuelto a cero, excepto PC, que tiene el valor que tenía inicialmente. Puedes comprobar también que la memoria tiene ceros donde habías puesto las instrucciones.
- ❑ Ahora vamos a hacer que el simulador vuelva al estado en el que se encontraba cuando guardamos el archivo `.sim`. Escoge la opción del menú *Archivo*→*Abrir*. Te saldrá el cuadro de diálogo típico para abrir un archivo. Escoge tu carpeta de usuario y el tipo de archivo `.sim`. Elige a continuación el archivo `1-1sim`.
- ❑ Fíjate que se ha modificado el estado de la CPU para que sea exactamente el mismo que cuando guardaste el archivo. Además de los registros, comprueba que en la memoria están las instrucciones que introdujiste al principio de la práctica.

Los archivos `.sim` son muy útiles para recuperar el estado total de la simulación en cualquier momento, lo que permite, entre otras cosas, interrumpir la práctica en cualquier instante y reanudarla más adelante.

## 3. Archivos .mem

El simulador puede cargar otro tipo de archivos, los *archivos de memoria*, con extensión `.mem`, que permiten cargar en una zona de memoria una serie de números (que pueden ser código máquina de instrucciones o datos). A diferencia de los `.sim`, que afectan a todo el computador (registros, memoria, buses, señales, dispositivos de E/S), los archivos `.mem` sólo modifican las posiciones de memoria que se le indiquen.

Los archivos de memoria son archivos de texto creados por el usuario con la ayuda de un editor de textos ASCII. Su contenido debe ser una lista de números de 16 bits en notación hexadecimal, separados por espacios en blanco, tabulaciones o retornos de carro, que representan las instrucciones o datos que se quieren cargar en memoria.

Vamos a crear un archivo `.mem` que contenga el código máquina de las siguientes instrucciones y vamos a hacer que se cargue en la posición 300h:

1	SUB R0, R5, R3
2	MOV R5, R3
3	INC R6

Sigue estos pasos:

- ☐ Pulsa en la opción Ejecutar... y escribe `cmd`  
Esto te abrirá un “interfaz de comandos” (ventana de fondo negro). Desde esta ventana podrás editar ficheros de texto y, en futuras prácticas, “ensamblarlos”. En esta práctica el “ensamblado” lo harás a mano.
- ☐ En el interfaz de comandos escribe los comandos necesarios para llegar hasta la carpeta en que estás almacenando los ficheros de la práctica. Debes escribir `Z:` para cambiar a la unidad Z (tu unidad de usuario), y seguidamente el comando `cd` seguido de un nombre de carpeta para “entrar” en ella. Conviene que uses nombres de carpeta cortos, que no contengan espacios.  
Una vez estás en la carpeta adecuada, el comando `dir` te muestra los ficheros que hay en ella. Comprueba que son los que esperas.
- ☐ Escribe `edit 1-1mem.mem` y pulsa intro. Esto te abre un simple editor en el que puedes escribir los contenidos del archivo (no utilizamos el Bloc de notas porque añade por defecto la extensión `.txt` y necesitamos que la extensión sea `.mem`).
- ☐ Escribe la codificación de las instrucciones del listado anterior separando cada número por un retorno de carro.
- ☐ Guarda el archivo y sal del editor.

Vamos a comprobar qué ocurre cuando cargas este archivo.

- ☐ Reinicia el simulador.
- ☐ Pulsa sobre el dibujo que representa la memoria y escoge *Cargar desde archivo* en el menú que aparecerá.
- ☐ Escoge el archivo `1-1mem.mem` que antes guardaste.
- ☐ Te saldrá ahora un cuadro de diálogo en el que te pregunta la dirección de carga del contenido del archivo, es decir, a partir de qué dirección quieres que se carguen las instrucciones o datos que contenga el archivo. Introduce el valor 300h y pulsa aceptar.
- ☐ Busca el código del programa en memoria con el desensamblador. Comprueba que se han cargado las instrucciones que escribiste en el archivo `1-1mem.mem`.

- ☐ Modifica ahora PC para que se ejecute la segunda instrucción de las que has cargado.
- ☐ Pulsa **F8** y comprueba que se ejecuta la instrucción deseada.

Si ahora tuvieras que poner las instrucciones del archivo justo a continuación de las que acabas de cargar, ¿cómo lo harías? Inténtalo.

## 4. Modos de direccionamiento

A continuación iremos codificando y ejecutando una serie de instrucciones que usarán diferentes modos de direccionamiento. Para seguir la pista de dónde está almacenado el código máquina de cada instrucción y qué instrucción representa, se recomienda crear en un papel aparte una tabla como la siguiente:

Dirección de memoria	Código Máquina	Mnemónico Instrucción
...	...	...

que irás rellenando a lo largo de esta sesión de prácticas, cada vez que se te pida codificar una instrucción.

Vamos a estudiar los distintos modos de direccionamiento de la CPU elemental. En esta sesión ya has usado un modo de direccionamiento (instrucciones MOV R5, R3 y ADD R0, R5, R3) ¿Qué modo de direccionamiento se ha usado?

### 4.1. Preparación del simulador

- ☐ Tras arrancar el Simulador de la CPU, modifica los siguientes registros (pulsando con el ratón sobre ellos) y carga los valores que se especifican:

Registro	Valor
R2	FC23
R3	0004
R7	ABCD
PC	0200

- ☐ Además, carga el número 0201 en la dirección de memoria 0004.

## 4.2. Codificación y ejecución de instrucciones

- ❑ Codifica la instrucción `MOV R5, [R3]`. Carga el código de la instrucción en la memoria, de tal manera que sea la próxima instrucción a ejecutar. Apuntalo en la tabla que estás creando en papel aparte. No pulses `F8` todavía. Responde:

1. ¿Qué es lo hará esta instrucción?
2. ¿Con qué operandos trabaja?
3. ¿Qué valor aparecerá en R5 después de que se ejecute la instrucción?<sup>[8]</sup> ¿Y en R3?<sup>[9]</sup>
4. ¿Cuáles son los modos de direccionamiento que se utilizan en esta instrucción?

8

9

Pulsa `F8` y comprueba que ha ocurrido lo que habías previsto.

- ❑ Ahora deberás escribir las instrucciones necesarias para cargar en el registro R2 el valor 0020. ¿Cuántas instrucciones necesitas?<sup>[10]</sup> ¿Por qué? Codifícalas y cárgalas en memoria para que sean las siguientes instrucciones a ejecutar (a partir de la situación anterior). Pulsa `F8` las veces que necesites para que se cargue el registro R2. ¿Qué modos de direccionamiento has usado? ¿Por qué reciben ese nombre?
- ❑ Ahora tienes que guardar el contenido del registro R7 en la posición de memoria 0020. ¿Qué instrucción tienes que utilizar?<sup>[11]</sup> ¿Qué modos de direccionamiento hay que aplicar? Codifica la instrucción, cárgala en la memoria y ejecútala. Comprueba que efectivamente el dato se ha almacenado en la dirección deseada (usando para ello el editor hexadecimal de la memoria).
- ❑ Codifica y ejecuta en secuencia las instrucciones `MOV R7, R3` y `MOV R5, [R7]`. Antes de ejecutarlas intenta adivinar qué número aparecerá en R5 al terminar<sup>[12]</sup>.

10

11

12



## 5. Autoevaluación

### 5.1. Archivos en el disco

En tu carpeta de prácticas deberás tener los archivos de programa 1-1cpu.sim y 1-1mem.mem.

### 5.2. Ejercicios

- ⇒ Reinicia el simulador. Pon en el registro R3 el valor 103Bh y en R5 F1ACh. Coloca la codificación de la instrucción ADD R0, R5, R3 en la posición de memoria C000h y haz que sea la siguiente instrucción a ejecutar. Antes de ejecutarla, contesta a estas preguntas:

- ¿Qué valor tendrá el registro R0 tras la ejecución?<sup>[13]</sup>
- ¿Qué valor tendrán los flags del registro de estado?<sup>[14]</sup>

Ejecuta ahora la instrucción y comprueba si tus previsiones fueron acertadas.

- ⇒ Piensa ahora un valor que podrías poner en R3 para que haya overflow y haya carry<sup>[15]</sup>. Ponlo en R3 y modifica el PC para que se vuelva a ejecutar la instrucción anterior. Ejecútala y comprueba si has elegido bien el valor.
- ⇒ ¿Qué valor tendrías que poner en R3 para que el flag de cero estuviese a 1?<sup>[16]</sup> Introduce ese valor y comprueba si has elegido bien el valor.

13

14

15

16

## SESIÓN 2

# Control de flujo. Programación de algoritmos sencillos

## Objetivos

En esta sesión el alumno codificará instrucciones de control de flujo, con el objetivo de comprender cómo el flujo secuencial de la ejecución puede ser roto mediante los saltos, cómo se codifican éstos y cómo su ejecución afecta al registro PC.

Posteriormente se utilizará el juego de instrucciones de la CPU elemental para construir algoritmos sencillos que pueden dar lugar a programas. Se analizará el correcto funcionamiento de esos algoritmos usando el simulador y se presentará un nuevo tipo de archivo: “archivo ejecutable” que simplifica la carga de programas en el simulador.

El objetivo final de esta sesión es comprobar como, a partir de un conjunto restringido y relativamente simple de instrucciones básicas, una CPU puede ejecutar algoritmos y programas de mayor complejidad.

## Conocimientos y materiales necesarios

El alumno/a debe:

- Acudir al laboratorio con una copia del juego de instrucciones de la CPU (disponible en la sección de material de teoría de la página web de la asignatura). También debe conocer el manejo de esta tabla para codificar las instrucciones de la CPU elemental.
- Repasar el juego de instrucciones de la CPU teórica y conocer todos los tipos de instrucciones, los modos de direccionamiento posibles y el trabajo que realiza cada instrucción.
- Verificar que en su carpeta de prácticas están los ficheros generados en la sesión anterior. En particular, necesitará el llamado `1-1estado.sim`.

## Desarrollo de la práctica

### 1. Instrucciones de control de flujo

Vamos a trabajar ahora con las instrucciones que permiten controlar el flujo de ejecución de un programa.

Para comenzar, carga en el simulador el estado que guardaste al final de la sesión anterior (fichero 1-1estado.sim). Una vez recuperado el estado, el desarrollo continúa como se indica a continuación.

#### 1.1. Salto incondicional

- ❑ Tienes que colocar ahora en memoria (en la dirección correspondiente a la próxima instrucción a ejecutar) una instrucción que cause un salto hacia atrás, de modo que se vuelva a la primera instrucción de las que has cargado del archivo .sim (MOV R5, [R3]). Consulta la tabla en la que vas apuntando las instrucciones y la posición de memoria en las que éstas se encuentran. Responde:

1. ¿Cuál es el mnemónico de la instrucción de salto incondicional necesaria?<sup>[1]</sup>
2. ¿Cuál es el código máquina de esa instrucción?<sup>[2]</sup> Cárgala en memoria.
3. Pulsa **[F8]** (sólo una vez). ¿Qué valor tiene ahora el registro PC?<sup>[3]</sup> ¿Es el esperado?<sup>1</sup> Si no lo has hecho bien piensa en qué te has equivocado.<sup>2</sup>

1

2

3

- ❑ Pulsa de nuevo **[F8]** para ejecutar MOV R5, [R3]. Si, a continuación, sustituimos la instrucción MOV R5, [R3] por la instrucción JMP +6, y luego pulsamos 7 veces **[F8]** ¿Cuántas instrucciones *distintas* se ejecutarán?<sup>[4]</sup> ¿Cuál será el valor final del registro PC?<sup>[5]</sup> ¿Por qué? Compruébalo.

4

5

#### 1.2. Salto condicional

Pasaremos ahora a las instrucciones de salto condicional, pero en lugar de modificar la memoria “a mano”, usaremos un *archivo de memoria*. El formato de los archivos de memoria, ya ha sido explicado en la sesión anterior. Recuerda que debes usar el comando Edit en el menú Inicio→Ejecutar de Windows.

- ❑ Selecciona la opción del menú *Ejecución*→*Reiniciar*. Esto borrará todos los registros y la memoria.

<sup>1</sup>Si ese valor no es 0200 has realizado mal el cálculo.

<sup>2</sup>Si lo has hecho mal, puedes reintentar hacerlo bien cargando en el registro PC el valor anterior (0206) y modificando la instrucción de salto incondicional.

- ❑ Edita un archivo de memoria llamado `1-2prog1.mem` que contenga el código máquina de un programa que debe hacer lo siguiente:
  1. Cargar el registro R3 con el valor 8 decimal (se puede hacer con una o con dos instrucciones ¿Por qué? Hazlo con 2 instrucciones).
  2. Cargar el registro R6 con el valor 2. Hazlo también en 2 instrucciones.
  3. Decrementar el registro R3.
  4. Ejecutar la instrucción `BR.NZ -2`.
  5. Sumar los registros R3 y R6 y guardar el resultado en el registro R7.
- ❑ Carga el archivo de memoria en el simulador, a partir de la dirección 0300, e inicializa el registro PC con el valor 0300, para que la ejecución comience en la primera instrucción de las que has cargado en memoria. Comprueba con el desensamblador que has codificado correctamente todas las instrucciones.
- ❑ Responde: ¿Cuántas veces tendrás que pulsar `[F8]` antes de que se ejecute la última instrucción (la suma)?<sup>[6]</sup> La primera vez que se ejecuta la instrucción `BR.NZ -2` ¿qué instrucción se ejecutará a continuación?<sup>[7]</sup> Comprueba, pulsando `[F8]`, que estás en lo cierto.

6

7

### 1.3. Bucles

Examinemos lo que hemos hecho hasta ahora. En el archivo `1-2prog1.mem` tenemos un conjunto de instrucciones que hacen lo siguiente:

- Cargan en un registro (R3) un valor  $N$  determinado. En este caso  $N = 8$ .
- Se decrementa ese registro, de uno en uno, hasta cero (cuando la instrucción condicional es `BR.NZ`). Dicho de otra manera, podemos hacer una cuenta atrás desde un número  $N$  determinado.

Lo que vamos a hacer a continuación es aprovechar la cuenta para hacer algo en cada decremento.

- ❑ Copia el archivo de memoria `1-2prog1.mem` en un nuevo archivo llamado `1-2prog2.mem`.
- ❑ Edita `1-2prog2.mem` (con el comando `Edit`) y, delante del código de la instrucción `DEC R3` añade el código de la instrucción `INC R0`. Cambia el código de la instrucción `BR.NZ -2` por el de `BR.NZ -3`. Salva el archivo.
- ❑ Carga el archivo de memoria y ejecútalo hasta la instrucción `ADD`. ¿Qué valor tiene el registro R0? ¿Coincide con el valor de  $N$ ?<sup>[8]</sup>

8

## 2. Programación de un algoritmo simple

Vamos a construir en la CPU de ejemplo un programa que sea capaz de realizar la suma de una serie de números, es decir, a partir de una lista de números A, B, C, D, E, F, ..., lo que hay que hacer es calcular:

$$\text{suma} = A + B + C + D + E + F + \dots$$

La descripción del problema es la siguiente:

*A partir de una posición de memoria determinada tenemos una lista de números, uno a continuación de otro. El programa tiene que ir sumando uno a uno los números. El resultado se almacenará en una dirección de memoria dada. Son datos del problema: la dirección en la que se encuentra el primer número de la lista, la cantidad de números a sumar y la dirección de memoria en la que hay que dejar el resultado*

Para resolver este problema e implementarlo en la CPU de ejemplo, tenemos que seguir los siguientes pasos:

1. Plantear el algoritmo que resuelve el problema.
2. Expresar el algoritmo en el lenguaje ensamblador de la CPU de ejemplo.
3. Codificar de las instrucciones que componen el programa. (Compilación del lenguaje ensamblador)
4. Cargar en memoria el programa.
5. Inicializar el registro Contador de Programa.
6. Ejecutar el programa.

Analizando la operación a realizar vemos que, si la cantidad de números a sumar es mayor que 2, la suma acumulada ya no se podrá realizar de manera directa en nuestra CPU porque la instrucción de suma de la CPU teórica solamente puede sumar dos operandos de cada vez. Necesitaremos construir un algoritmo que nos permita realizar la suma de cualquier cantidad de números.

Lo primero que necesitamos es un *almacén* para el resultado (lo que hemos llamado suma). En nuestro caso, podemos optar por una posición de memoria o por un registro de la CPU. Utilizaremos un registro de la CPU, en concreto R0. A este elemento le iremos sumando los números de uno en uno. Realizaremos la operación:

$$\text{suma} = \text{suma} + A$$

Si suma está inicializado a 0, en suma tendremos el valor de A. Después realizaremos:

$$\text{suma} = \text{suma} + B$$

y de esta manera tendremos en suma el resultado de sumar A y B. Después haremos:

$$\text{suma} = \text{suma} + C$$

y así sucesivamente hasta sumar todos los números de la lista.

Por lo tanto, nuestro algoritmo de suma sería algo así:

```
Inicializar contador de números a sumar
Inicializar "suma" con cero
Inicializar un registro con la dirección en que está el primer número
Repetir
    Tomar un numero de la lista
    Añadirlo (sumarlo) a "suma"
    Pasar (apuntar) al siguiente numero de la lista
    Decrementar contador de números a sumar
    Si el contador es distinto de 0 volver a "Repetir"
    Si no (el contador es 0) salir del bucle
```

Como la lista de números se encuentra en memoria, tendremos que usar un registro para acceder a ellos, ya que esta CPU sólo admite el modo de direccionamiento a memoria indirecto a través de registro. Este registro comienza apuntando al primer elemento de la lista, pero irá aumentando en cada repetición del bucle para apuntar a los sucesivos elementos.

Después de realizar la suma, tendríamos que guardar el resultado en la memoria. Nos quedaría por hacer:

```
Inicializar registro que apunta a la posición de memoria donde se
    guardará el resultado
Guardar el resultado en esa posición de memoria.
```

Ahora que sabemos lo que tenemos que hacer, vamos a identificar los distintos elementos que necesita el algoritmo y qué almacén (registro) de nuestra CPU de ejemplo utilizaremos para cada uno.

Elemento del algoritmo	Registro
Acumulador (suma)	R0
Dirección donde está el dato a sumar	R1
Contador de elementos que aún quedan por sumar	R2
Dato a sumar, temporalmente almacenado en registro	R3
Dirección donde se dejará el resultado	R4

Vamos a ver cómo nos queda el programa expresado ya en instrucciones de la CPU de ejemplo para los siguientes datos:

- Dirección de comienzo de la lista de números: 25FD
- Dirección de memoria en la que se almacena la suma: 25F0
- Cantidad de números que contiene la lista: 4 (los números son: 12, 3, -20, 7)

Nuestro programa, expresado en lenguaje ensamblador de la CPU teórica, es el siguiente (se han quitado alguno de los mnemónicos):

```

???? R0,R0,R0    ; Inicializar suma a cero

MOVL R1, FDh     ; Inicializar registro que nos apunta
MOVH R1, 25h     ; al principio de la lista de números

MOVL R2, 4       ; Inicializar contador de números a sumar
????

bucle:           ; Indicador de inicio del bucle
MOV R3, [R1]     ; Coger un número de la lista
ADD R0, R0, R3   ; Añadirlo a suma
INC R1           ; Pasar al siguiente número de la lista
DEC R2           ; Decrementar contador de números a sumar
BR_NZ bucle     ; Si quedan números (resultado mayor que 0),
                ; volver al inicio del bucle

MOVL R4, F0h     ; Inicializar registro que apunta a la
MOVH R4, 25h     ; posición de memoria donde se guardará el resultado
????           ; Guardar el resultado en la memoria

```

Ahora que tienes parte del programa haz lo siguiente:

- ☐ Completa el programa con las instrucciones que faltan.
- ☐ Con ayuda de la tabla de codificación, codifica las instrucciones del programa (compilar el programa).
- ☐ Crea un archivo de memoria llamado 1-2prog3.mem con el código máquina del programa.
- ☐ Carga el archivo de programa en el simulador, a partir de la dirección de memoria 3100h. Usando el desensamblador, comprueba que has codificado correctamente el programa.
- ☐ Codifica en hexadecimal los datos de la lista de números, escribe su codificación en un archivo llamado 1-2datos1.mem y carga ese archivo en la memoria a partir de la dirección 25FDh.
- ☐ Comprueba el desarrollo del programa: inicializa PC con el valor 3100h (dirección de inicio del programa) y ejecuta paso a paso las instrucciones pulsando la tecla **F8**. ¿Qué aparece al final en la dirección de memoria 25F0h?<sup>[9]</sup>

9

### 3. Archivos ejecutables y otro algoritmo sencillo

En el último ejercicio de la sesión anterior hemos visto como podíamos hacer un pequeño programa que es capaz de contar hasta un número  $N$  determinado.

Vamos a aprovechar ese trabajo para hacer un pequeño programa que sea capaz de realizar la suma acumulada de  $N$  números.

El programa, dado un número  $N$ , tiene que calcular el valor:

$$\text{suma\_acumulada} = 1 + 2 + 3 + \dots + N$$

En el archivo de memoria `1-2prog2.mem` tenemos un algoritmo que nos permite “contar” hasta  $N$ . Si a la vez que vamos contando, también vamos sumando, nuestro problema de realizar la suma acumulada queda resuelto.

Vamos a aprovechar por tanto este archivo, y a la vez vamos a presentar un nuevo tipo de archivo manejado por el simulador: el “archivo ejecutable”.

Como has observado, a la hora de cargar un programa en la CPU se requieren ciertos pasos, siempre los mismos:

1. Cargar un archivo de memoria, *especificando la dirección de carga*.
2. Inicializar el registro PC para que apunte a la primera instrucción que queremos ejecutar.
3. Inicializar otros registros de la CPU (contadores, etc) con el valor 0.

Estas tareas son automatizadas si usamos “archivos ejecutables” en lugar de “archivos de memoria”. Un archivo ejecutable para el simulador de CPU es muy similar a un archivo de memoria, pero contiene, además del código a cargar, la dirección de carga, el valor inicial de PC y el valor inicial de R7 (este último es necesario pues está relacionado con la pila, como veremos en futuras sesiones).

Los archivos ejecutables para el simulador, de momento se crearán de la misma forma que los archivos de memoria, es decir, escribiéndolos “a mano” con el programa EDIT. Externamente se distinguen porque su nombre termina en `.eje` (en lugar de `.mem`). En el siguiente bloque de prácticas presentaremos una herramienta que permitirá crear archivos ejecutables de forma más cómoda.

Al leer un “archivo ejecutable”, el simulador de CPU interpreta los números que allí encuentra de la siguiente forma:

1. El primer número indica en qué dirección de memoria se cargarán los datos del fichero.
2. El segundo número es el valor con que se inicializará PC.
3. El tercer número es el valor con que se inicializará R7.
4. Los restantes números hasta fin de fichero, son equivalentes a los de un “archivo de memoria”, y se irán cargando la dirección especificada en 1.

Los restantes registros de la CPU (R0 a R6) se inicializan automáticamente al valor 0, así como toda la memoria que no ha sido cargada desde el archivo, que también recibe el valor 0 (borrando por tanto otros programas o datos que pudieran haber en memoria).

Con esta información ya podemos empezar a hacer cosas:

- ❑ Copia el archivo llamado `1-2prog2.mem` que tenías en tu carpeta de prácticas y dale a la copia el nombre `1-2prog4.eje`.



- ❑ Abre, con el editor de texto EDIT, el archivo 1-2prog4.eje.
- ❑ Añade al principio del fichero los números 0200, 0200 y 0300 (que representan, como ya se ha dicho, la dirección de carga, el valor inicial de PC y el valor inicial de R7)
- ❑ Guarda el archivo ejecutable y sal del editor
- ❑ Carga el archivo ejecutable en el simulador (usando el menú *Archivo*→*Abrir*. Comprueba cómo PC y R7 se han inicializado al valor especificado en el archivo.
- ❑ Usa el desensamblador para examinar lo que hay en la memoria, a partir de la dirección 0200h, y copia en un papel el código máquina y los mnemónicos de cada instrucción.
- ❑ Modifica (sobre el papel) las instrucciones de carga del registro R6 para que se inicialice correctamente para trabajar como el acumulador de la suma.
- ❑ Modifica (sobre el papel) la instrucción de suma del programa. Tienes que cambiarla de posición en el programa y modificarle los operandos. Tienes que conseguir que, en cada iteración, realice una suma y la acumule sobre el registro R6. ¿Dónde tienes que colocar la instrucción ADD? ¿Cuáles tienen que ser sus operandos?<sup>[10]</sup> Observa que también tienes que modificar la instrucción de salto condicional.
- ❑ Lleva a cabo las modificaciones que has diseñado sobre el papel, editando el fichero 1-2prog4.eje y cambiando sus contenidos.
- ❑ Carga el nuevo ejecutable y comprueba que hace lo que se espera de él, ejecutándolo paso a paso mediante **F8**. El resultado final debe ser 24h

10

## 4. Autoevaluación

### 4.1. Archivos en el disco

En tu disco de prácticas tendrás que tener los archivos de memoria 1-2prog1.mem, 1-2prog2.mem y 1-2prog3.mem y el archivo ejecutable 1-2prog4.eje.

### 4.2. Ejercicios

- ⇒ Considera el programa que has codificado en 1-2prog1.mem. Si quisieras sustituir la instrucción DEC R3 por una instrucción SUB que hiciera lo mismo. ¿Cuántas instrucciones tendrías que añadir al programa para conseguirlo?<sup>[11]</sup>

11

- ⇒ Carga el archivo 1-2prog2.mem en la memoria (elige tú mismo la dirección que quieras) e inicializa el registro PC para apunte a la primera instrucción del programa. Modifica (directamente sobre la memoria) las instrucciones que inicializan R3 para que se cargue el valor 12 decimal en lugar del 8.

Escribe un nuevo programa (en un archivo de memoria) a partir del que hay en 1-2prog2.mem, haciendo los cambios necesarios para sustituir la instrucción INC R0 por una instrucción ADD que tenga el mismo efecto. Cárgalo y ejecútalo para verificar que funciona.

- ⇒ Reinicia el simulador. Codifica la instrucción ADD R3, R5, R1. Cárgala en una dirección de memoria cualquiera. En la dirección siguiente codifica la instrucción de salto condicional BR.Z +5. Ahora, en la dirección de memoria a la que se saltaría con ese BR.Z, codifica una instrucción de salto incondicional que lleve a la instrucción ADD.

Prepara el registro PC para que apunte a la instrucción ADD y carga a mano los registros R3, R5 y R1 de tal manera que al pulsar **F8** tres veces se ejecuten ambas instrucciones de salto: la condicional y la incondicional.

- ⇒ Cambia la instrucción de salto condicional del ejercicio anterior por todas las posibles condiciones. Carga los registros de manera adecuada para probar cada una de las instrucciones.

### 4.3. Un programa completo

Ahora vamos a plantear un programa que debes realizar desde cero y sin ayuda. Tendrás que pensar e implementar todo el algoritmo. El problema es el siguiente:

*Realizar un programa que procese una lista de números determinando cuántos de ellos son positivos.*

La lista estará formada por los números: -1, 4, 76, 90, -45, 34, -9, 23, -3

Los números estarán almacenados a partir de la posición de memoria A000

El resultado debe almacenarse en la posición de memoria: B000

PISTA: se recorre la lista de uno en uno. Si es positivo se cuenta, si no, no.

#### 4.3.1. Condicionamientos

Para realizar el programa tendremos que respetar las siguientes condiciones:

- ❑ Se usará el registro R1 como contador, para lo cual deberá inicializarse con el número de elementos de la lista y se decrementará en cada iteración. Cuando R1 alcance el valor 0, terminará el procesamiento.

- ☐ Usar el registro R5 para apuntar a los números de la lista.
- ☐ Usar el registro R6 para apuntar a la dirección de memoria donde se va a almacenar el resultado.
- ☐ Usar el registro R4 como contador de números positivos. (Debe ser inicializado a 0 e incrementado cada vez que se encuentre un positivo).
- ☐ Una vez codificado tu algoritmo, debes escribir su código en un archivo ejecutable con el nombre 1-2prog6.eje. Los datos, en cambio, puedes meterlos “a mano” en memoria, o preparar un archivo .mem con ellos y cargarlo en la dirección A000h, como tú prefieras (pero recuerda que al cargar un .eje, los contenidos del resto de la memoria se borran).

## SESIÓN 3

# Pasos de ejecución de instrucciones del computador elemental

## Objetivos

Durante esta sesión el alumno se familiarizará con las señales de control y el modo de simulación *manual* de la CPU elemental.

Para ello, aprenderá cómo cambiar el modo de funcionamiento del simulador del computador elemental, cuáles son las diferencias entre el modo de funcionamiento *normal* y el modo de funcionamiento *manual* y cómo efectuar operaciones con el simulador de la CPU elemental en este nuevo modo.

## Conocimientos y materiales necesarios


Para un aprovechamiento positivo de esta práctica se requiere que el alumno:

- Sepa cómo codificar, cargar y ejecutar un programa (archivo ejecutable) para la CPU teórica en modo de simulación *normal*.
- Conozca cómo funciona una *unidad de control* y tenga claros los conceptos *paso de ejecución* y *señal de control*.
- Sepa cómo utilizar la documentación del computador elemental que se le ha suministrado, y haya estudiado los apuntes de teoría en los que se señalan, para cada grupo de instrucciones, las señales de control activas en cada paso de ejecución.
- Verifique que su carpeta de prácticas contenga los archivos generados en prácticas anteriores. En particular, en esta sesión se necesitarán los archivos 1-2prog3.mem y 1-2datos1.mem.

## Desarrollo de la práctica

### 1. Ejecución de instrucciones paso a paso en modo normal

En sesiones anteriores, el alumno ya ha ejecutado programas en el computador elemental y ha tenido ocasión de comprobar que, cada vez se pulsa la tecla **F8**, se ejecuta una instrucción completa. Esa pulsación tiene, por tanto, el mismo efecto que la generación de tantos ciclos de reloj como pasos de ejecución tiene la instrucción.

Si en lugar de la tecla **F8** se pulsa la tecla **F7**, o se pulsa con el ratón sobre el botón que muestra un flanco ascendente (situado en la esquina superior izquierda de la ventana de la aplicación, ) , sólo transcurrirá un pulso de reloj, lo que permite observar las señales que la unidad de control genera en cada paso de ejecución. Para completar la ejecución de una instrucción el alumno deberá pulsar **F7** tantas veces como pasos de ejecución tenga esa instrucción.

Antes de comenzar a utilizar el modo de simulación *manual*, y para que el alumno se familiarice con las señales de control de la CPU teórica, se ejecutarán paso a paso algunas instrucciones de un programa desarrollado en sesiones anteriores. Para ello:

- ❑ Carga en memoria el archivo 1-2prog3.mem en la memoria del computador elemental (a partir de la dirección 3100h), y el archivo con los datos (1-2datos1.mem) a partir de la dirección 25FDh.
- ❑ Inicializa el registro PC al valor adecuado para ejecutar el programa que acabas de cargar.
- ❑ Pulsa la tecla **F8** hasta que en el registro de instrucción IR aparezca la instrucción MOVH R1, 25h. En este instante, PC apuntará a la siguiente instrucción a ejecutar, que será MOVL R2, 4.
- ❑ Pulsa la tecla **F7**. Tu pulsación tiene el mismo efecto que si el reloj del sistema generase un nuevo pulso de reloj, por lo que la unidad de control genera las señales correspondientes a la fase de búsqueda de instrucción (primer paso de ejecución, T1). Observa cómo la unidad de control del simulador del computador elemental indica el paso de ejecución en el que se encuentra y las señales que ha activado. Observa también el contenido de los registros PC, IR, MAR, MDR y TMPS, y contesta:
  - ¿Por qué el registro MAR tiene ese valor?
  - Observa el contenido del registro TMPS. ¿En qué registro se almacenará ese valor?<sup>[1]</sup> ¿Por qué no se ha almacenado todavía?
  - Durante el paso de ejecución T1, ¿ha cambiado el valor del registro MDR?<sup>[2]</sup> ¿Lo hará más adelante? Si lo hace, ¿en qué paso de ejecución lo hará y por qué?<sup>[3]</sup>

1

2

3

- ❑ Pulsa dos veces más la tecla **F7** para completar las fases de incremento del PC y de decodificación de la instrucción (pasos T2 y T3). Observa el valor del registro PC. ¿A dónde apunta?<sup>4</sup> ¿Es lógico? ¿En qué paso de ejecución ha cambiado, en T2 o en T3?<sup>5</sup>
- ❑ ¿Cuál es el contenido del registro MDR?<sup>6</sup> ¿Qué necesitarías para haber predicho el contenido de MDR a partir del registro MAR?
- ❑ ¿Qué contiene el registro IR? ¿Qué contenía MDR? ¿Es una coincidencia? ¿Puede afirmarse que el contenido de los registros IR y MDR es siempre el mismo sea cual sea el paso de ejecución de la instrucción?<sup>7</sup>
- ❑ Si pulsases de nuevo la tecla **F7** (no lo hagas), ¿cambiaría el valor de IR?<sup>8</sup> ¿Cuántas veces crees que cambiará hasta que finalice la ejecución de la instrucción? ¿Cuántas veces tendrás que pulsar la tecla **F7**, tras terminar la ejecución de la instrucción actual, hasta que el registro IR cambie de valor?<sup>9</sup>
- ❑ Pulsa la tecla **F7** hasta llegar a la instrucción `MOV R3, [R1]`. Ejecutala paso a paso y observa el ciclo de espera que aparece antes de completar la lectura de memoria. ¿En qué paso ocurre<sup>10</sup> y por qué es necesario?
- ❑ Sigue ejecutando el programa y observa cómo en la instrucción de escritura en memoria, `MOV [R4], R0`, hay que esperar un ciclo de reloj para que el resultado quede escrito en la posición de memoria seleccionada.

4

5

6

7

8

9

10

## 2. El modo de simulación manual del computador elemental

El simulador del computador elemental no sólo permite ejecutar paso a paso las instrucciones de un programa, sino que proporciona un modo de simulación especial, que da al alumno la posibilidad de generar señales de control a su antojo: el modo de simulación *manual*.

Para pasar al modo de simulación *manual*, debe pulsarse la tecla **F5**, o elegir la opción *Simulación manual* del menú *Ejecución*. El simulador del computador elemental indica el modo de simulación elegido en la esquina derecha de la barra de estado de la aplicación. Una nueva pulsación de la tecla **F5** hará que el simulador vuelva al modo de simulación *normal* (ver figura 3.1). ¡Cuidado! Al cambiar de modo, los contenidos de la memoria y los registros se borran, como si se hubiera realizado una operación de *Reiniciar*.

La característica principal de este modo de simulación es que *los registros PC e IR pierden su funcionalidad*. Es el alumno quien debe seleccionar las señales de control que se activarán en cada ciclo de reloj y pulsar la tecla **F7** para activarlas.

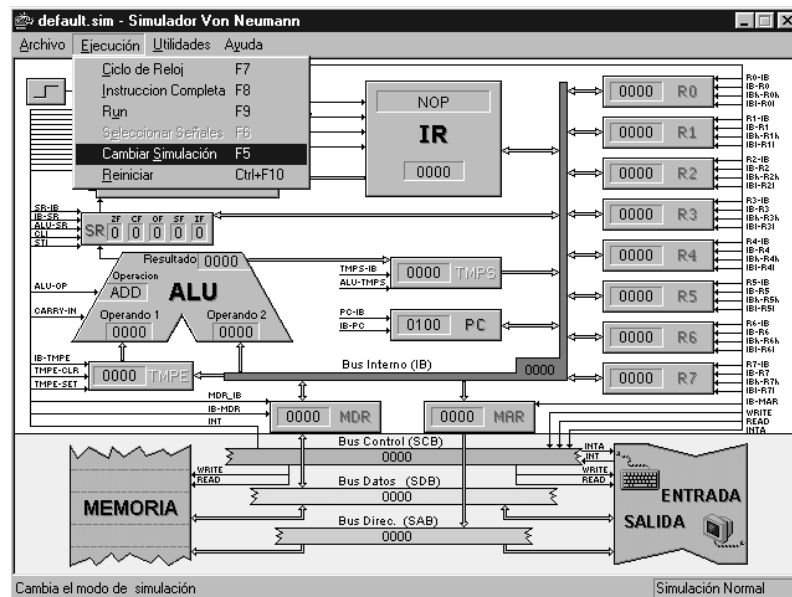


Figura 3.1: Selección del modo de funcionamiento manual

Para ilustrar este modo de funcionamiento se calculará la suma de dos operandos almacenados en dos registros diferentes generando las señales de control adecuadas, y se almacenará el resultado en un tercer registro, diferente a los dos anteriores:

- ❑ Si no lo estaba ya, pon el computador en modo de simulación manual (pulsas **F5**).
- ❑ Introduce los valores 1000h y 2000h en los registros R0 y R1 respectivamente (pulsas sobre ellos con el ratón e introduce su valor).
- ❑ Escribe en un papel las señales de control activas en cada paso de ejecución de la instrucción ADD R2, R0, R1. Fíjate en las señales activas durante los pasos T4, T5 y T6.
- ❑ Usando el botón izquierdo del ratón, haz clic en la ventana de la unidad de control. Emergerá un diálogo que muestra todas las señales de control del computador elemental y que te permitirá elegir las que se activarán con el siguiente pulso de reloj (ver figura 3.2).
- ❑ Para que la ALU pueda sumar dos operandos diferentes, uno debe almacenarse en el registro temporal de entrada y el otro debe estar en el bus interno cuando se active la señal de control ADD. Utilizando el diálogo mencionado anteriormente, y fijándote en las señales que has escrito en el papel, selecciona las señales de control necesarias para llevar el primer operando, almacenado en R0, al registro temporal de entrada. Después, pulsas el botón **Aceptar**.
- ❑ Pulsas la tecla **F7** para activar las señales que has seleccionado, y observa lo que ocurre. Comprueba que el registro temporal de entrada queda cargado con el valor adecuado.



Figura 3.2: Selección de señales en modo de funcionamiento manual

- ❑ Ahora usarás el bus interno para llevar a la ALU el segundo operando (el almacenado en R1) y efectuar la suma. Fijándote en los pasos de ejecución de la instrucción ADD R2, R0, R1, marca las señales de control necesarias, acepta tu selección y actívalas mediante la tecla **F7**.
- ❑ Por último, almacena el resultado en el registro R2.
- ❑ Para fijar los conocimientos que has adquirido, efectúa con los mismos operandos las operaciones AND, almacenando el resultado en los registros R3 y R4.

### 3. Autoevaluación

- Usando el modo de simulación manual, genera las señales correspondientes a las fases de búsqueda de instrucción, incremento del PC y decodificación de una instrucción (pasos T1, T2 y T3), almacenada en la dirección de memoria 1000h, y cuya codificación es 5800. Si has tenido éxito, tras generar las señales de control del paso T3 podrás saber cuál es el mnemónico de dicha instrucción sin mirar las tablas de codificación: ¿cómo?<sup>[11]</sup>
- Escribe (y comprueba posteriormente con el simulador del computador elemental) las señales de control necesarias para sumar dos operandos, almacenados en los registros R0 y R1, y almacenar el resultado en la dirección de memoria apuntada por el registro R2.
- Haz lo mismo para conseguir copiar una palabra desde la dirección de memoria apuntada por el registro R0 a la dirección de memoria apuntada por el registro R1. No deberás alterar ningún registro de propósito general (R0-R7). Comprueba que efectivamente escribes el dato en la memoria. ¿Cuántos pasos de ejecución has necesitado?<sup>[12]</sup>

11

12