Ejemplos de uso de la práctica 4

Material proporcionado:

Fichero	Explicación
`ejemplos.zip`	Fichero que contiene todos los ejemplos descritos.

1. Introducción

A continuación se describen algunos ejemplos de uso de la aplicación a desarrollar en la práctica 4. En la corrección de práctica utilizaremos ejemplos *similares* a los aquí presentados para comprobar su funcionamiento.

Existen tres tipos de ejemplo distintos:

1. Ejemplos de memoria: para comprobar si la carga y grabado de los ficheros de memoria se realiza correctamente.
2. Ejecución simple de instrucciones: para comprobar casi de forma independiente si cada una de las instrucciones soportadas por la máquina funcionan correctamente.
3. Programas de ejemplo completos: con numerosas instrucciones que terminan haciendo un dibujo simple.

Las siguientes secciones describen uno a uno los ejemplos anteriores; se aconseja que se compruebe la práctica con cada uno de ellos *en el orden* aquí indicado. Para ello, se proporciona un fichero, `ejemplos.zip` que contiene todos los ejemplos aquí descritos. El fichero comprimido contiene un directorio por cada uno de los tipos de ejemplos (`01Mem`, `02Simp` y `03Prog`). En cada directorio, a su vez, aparece un directorio por cada ejemplo (`Ej1`,... `EjN`), donde se encontran los siguientes ficheros[1]:

[1] Algunos de ellos pueden no aparecer por no tener sentido en el ejemplo concreto

- `prog.asm`: programa a ejecutar.
- `memIn.txt`: estado inicial de la memoria.
- `memOut.txt`: estado final de la memoria esperado.
- `pantOut.txt`: estado final de la pantalla esperado, asumiendo que se ejecutó el programa con la pantalla inicialmente vacía.

2. Carga y grabación de ficheros de memoria

Los ejemplos consisten simplemente en la carga de diversos ficheros de memoria y su grabación a disco, para asegurarse de que se han leido correctamente y, además, que se han grabado correctamente. Por lo tanto, para probar la aplicación, basta con cargar cada uno de los ficheros (opción 3 del menú), y a continuación volcar el estado de la memoria a disco para ver si el fichero almacenado tiene las mismas posiciones que el original.

2.1. Ejemplo 1

Una única línea con una única posición de memoria.

```
05:-2300
```

2.2. Ejemplo 2

Varias líneas ordenadas.

```
5:2300
200:3
1000:256
14543:345
23432:9384
31432:4323
```

2.3. Ejemplo 3

Varias líneas desordenadas.

```
14543:345
5:2300
1000:256
23432:9384
31432:4323
200:3
```

2.4. Ejemplo 4

Ejemplo con posiciones repetidas. El valor final de la celda 23432 debe ser 1.

```
14543:345
5:2300
1000:256
23432:9384
31432:4323
23432:1
200:3
```

3. Ejecución de instrucciones simples

Los siguientes ejemplos ponen a prueba la ejecución de las instrucciones máquina que se indican en el enunciado. En la mayoría de los casos (a no ser que se indique lo contrario) la comprobación de si el programa ha funcionado correctamente consiste en comparar el estado final de la memoria.

3.1. Ejemplo 1: carga de valores

```
STORE [20000] 35
COPY [10000] [20000]
STORE [10] -15
```

El estado final esperado de la memoria es:

```
10:-15
10000:35
20000:35
```

3.2. Ejemplo 2: operaciones aritméticas

El siguiente programa suma y resta dos números.

```
STORE [20000] 35
STORE [20001] 25
SUM [20000] [20001]
SUB [20001] [20000]
```

El estado final esperado de la memoria es:

```
20000:60
20001:-35
```

3.3. Ejemplo 3: saltos

```
STORE [0] 0
STORE [1] 3
STORE [2] -3
STORE [11] 13
STORE [12] 16
STORE [13] 18
STORE [14] 19
JUMPZ [1] [14]
JUMPZ [2] [14]
STORE [101] 1
JUMPZ [0] [11]
STORE [201] 1
JUMPP [2] [14]
JUMPP [0] [12]
STORE [202] 1
JUMPP [1] [13]
STORE [203] 1
STORE [102] 1
```

El estado final esperado de la memoria es:

```
0:0
1:3
2:-3
11:13
12:16
13:18
14:19
101:1
102:1
```

Es decir, el programa *no* debe haber escrito en las posiciones de memoria `201`, `202` ni `203`, y debe haber escrito en las posiciones `101` y `102`. En caso de no ser así significará que alguno de los saltos no se ha ejecutado de forma correcta.

3.4. Ejemplo 4: lectura y escritura

El siguiente programa debe ejecutarse *con la pantalla vacía*, es decir, con todas sus posiciones con espacios[2]:

```
STORE [0] 1
STORE [1] 1
STORE [2] 1
READ [3]
READ [4]
```

[2] Así es como debe empezar la pantalla al arrancar la aplicación.

```
SUM [3] [4]
SUM [3] [2]
SUM [0] [2]
WRITE [3]
```

El estado de la memoria esperado es el siguiente:

```
0:2
1:1
2:1
3:65
4:32
```

En la pantalla, además, debe aparecer una 'Aén el primer carácter de la segunda línea.

4. Ejecución de programas complejos

4.1. Ejemplo 1: escritura de letras en la diagonal

El siguiente ejemplo escribe en la diagonal de la pantalla una serie de caracteres consecutivos, de tal forma que en la esquina superior izquierda aparece un carácter, debajo a la derecha de él aparece el siguiente, y así sucesivamente. El número de caracteres escritos es el indicado en la posición de memoria `12` al principio de la ejecución del programa. El último carácter que se escribirá es el que aparece en la posición `11`.

El programa siguiente incluye comentarios para mejorar la claridad aunque en la entrada a la aplicación deben suprimirse. De esta forma, si al ejecutarlo no se obtienen los resultados esperados, se puede ir ejecutando paso a paso comprobando si hace lo que debe.

```
; Carga de constantes
STORE [1001] 1   ; Constante 1
STORE [1002] 7   ; Etiqueta 1
STORE [1003] 10  ; Etiqueta 2
STORE [1004] 17  ; Etiqueta fin
; Inicializacion variables del bucle
COPY [201] [11]  ; [201] = carAux
COPY [202] [12]  ; [202] = i
; Inicio del bucle
etiqueta1:
SUB [202] [1001]
JUMPP [202] [1003] ; Si i >= 0, seguimos
JUMPP [1001] [1004] ; Salto seguro al final
; Cuerpo del bucle
etiqueta2:
; Escritura del caracter
COPY [0] [202]
```

```
COPY [1] [202]
SUM [0] [1001]
SUM [1] [1001]
WRITE [201]
; Decremento del caracter
SUB [201] [1001]
; Salto al principio del bucle
JUMPP [1001] [1002] ; Salto si constante 1 es positiva.
; Fin
etiquetaFin:
```

Para probar el programa, puede utilizarse la siguiente memoria inicial:

```
11:74
12:10
```

De esta forma, el programa escribirá 10 caracteres en la diagonal principal, siendo el último la 'J' (cuyo código ASCII es el 74).

Al final, el estado de la pantalla debería ser el siguiente (se numeran las filas y columnas por claridad; en el fichero generado únicamente debe aparecer el contenido de la pantalla):

```
          1         2
   12345678901234567890
   --------------------
 1|A                   |
 2| B                  |
 3|  C                 |
 4|   D                |
 5|    E               |
 6|     F              |
 7|      G             |
 8|       H            |
 9|        I           |
10|         J          |
   --------------------
```

Y el estado de la memoria debería ser:

```
0:1
1:1
11:74
12:10
201:64
202:-1
1001:1
1002:7
1003:10
```

```
1004:17
```

4.2. Ejemplo 2: dibujado de un cuadrado

Este ejemplo dibuja un cuadrado cuya esquina superior izquierda y longitud de lado viene dado en unas posiciones de la memoria. En particular, el programa recibe en las posiciones de memoria 20000 y 30000 las coordenads x e y de la esquina superior izquierda, y en la posición 200 la longitud del lado.

El programa en lenguaje de alto nivel es:

```
proc cuadrado(x, y, l)

    for i := 0 hasta l - 1
        escribe(x + i, y, '-')
        escribe(x + i, y + l - 1, '-')
    end for

    for i := 1 hasta l - 2
        escribe(x, y + i, '|')
        escribe(x + l - 1, y + i, '|');
    end for

end proc
```

Al pasarlo al ensamblador de nuestra máquina, el programa resultante es el siguiente[3]:

```
; Dibuja un cuadrado. Parámetros en las
; posiciones siguientes:
; [20000] = pos x de la esquina superior izquierda
; [30000] = pos y de la esquina superior izquierda
; [200] = longitud del lado
; Constantes
; [25000] = 0   [25001] = 1
; [25002] = '-' [25003] = '|'
STORE [25000] 0
STORE [25001] 1
STORE [25002] 45
STORE [25003] 124
; Etiquetas para saltos
; [2000] = inicioBucle1 [2001] = finBucle1
; [2002] = inicioBucle2 [2003] = finBucle2
STORE [2000] 10
STORE [2001] 22
STORE [2002] 23
```

[3]Igual que antes se incluyen comentarios en el texto que no aparecen en el fichero pasado a la aplicación.

```
STORE [2003] 36
; Variables:
; [1000] = i
; [1001] = aux para calculos
; Primer bucle
COPY [1000] [25000] ; Inicialiacion contador
; Condicion
COPY [1001] [1000]
SUB [1001] [200]
JUMPP [1001] [2001]
; Escribe(x+i, y, '-')
COPY [1] [20000]
SUM [1] [1000]
COPY [0] [30000]
WRITE [25002]
; Escribe(x+i, y+l-1, '-')
SUM [0] [200]
SUB [0] [25001]
WRITE [25002]
; Decremento contador
SUM [1000] [25001]
; Vuelta a empezar
JUMPP [25001] [2000]
; Segundo blcue
COPY [1000] [25001] ; Inicializacion contador
; Condicion
COPY [1001] [1000]
SUB [1001] [200]
SUM [1001] [25001]
JUMPP [1001] [2003]
; Write(x, y + i, '|')
COPY [1] [20000]
COPY [0] [30000]
SUM [0] [1000]
WRITE [25003]
; Write(x + l - 1, y + i, '-')
SUM [1] [200]
SUB [1] [25001]
WRITE [25003]
; Contador
SUM [1000] [25001]
; Vuelta a empezar
JUMPP [25001] [2002]
```

Para probar la correcta ejecución y dibujado del cuadrado se puede utilizar como estado inicial de la memoria el siguiente fichero:

```
20000:16
30000:6
200:10
```

Observa que con estos parámetros, el cuadrado no entra en una pantalla de 10 filas y 20 columnas, por lo que tendrás que cambiar *las contantes* `NUM_FILAS` y `NUM_COLUMNAS` a unos valores de al menos 15 y 25.

Al final de la ejecución, el estado de la pantalla será[4]:

```
          1         2
   12345678901234567890123456
   --------------------------
 1|                          |
 2|                          |
 3|                          |
 4|                          |
 5|                          |
 6|                ----------|
 7|                |         ||
 8|                |         ||
 9|                |         ||
10|                |         ||
 1|                |         ||
 2|                |         ||
 3|                |         ||
 4|                |         ||
 5|                ----------|
   --------------------------
```

Y la memoria contendrá los siguientes valores:

```
0:14
1:25
200:10
1000:9
1001:0
2000:10
2001:22
2002:23
2003:36
20000:16
25000:0
25001:1
25002:45
25003:124
30000:6
```

[4]Igual que antes, el cuadro exterior y números se dan como referencia.

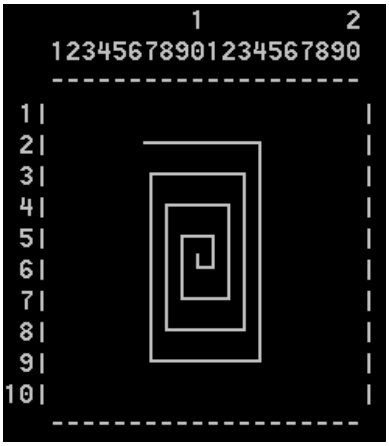

Figura 1: Salida del programa que dibuja una espiral

4.3. Ejemplo 3: espiral

El programa de este ejemplo muestra una espiral dibujada utilizando caracteres ASCII (figura 1). La esquina superior izquierda de la espiral viene determinada por el contenido de las posiciones de memoria `10` y `11` (para la x y la y respectivamente). La longitud del lado inicial (en caracteres) se indica en la posición `12`.

El programa en alto nivel tiene la forma siguiente:

```
proc espiral(x, y, l)
   mientras (l > 0)
      si (l = 1) entonces
         escribe(x, y, ESQSUPDER)
      sino
         for i := 0 hasta l-1
            si (i = 0) entonces
               escribe(x + i, y, '-')
               escribe(x + i, y + l - 1, ESQINFIZ)
            sino si (i <> l-1) entonces
               escribe(x + i, y, '-')
               escribe(x + i, y + l - 1, '-')
            sino
               escribe(x + i, y, ESQSUPDER)
               escribe(x + i, y + l - 1, ESQINFDER)
            fin si
         end for
         for i := 1 hasta l-2
            si (i = 1) entonces
               escribe(x, y + i, ESQSUPIZ)
               escribe(x + l - 1, y + i, '|')
            sino
```

```
                escribe(x, y + i, '|')
                escribe(x + l - 1, y + i, '|')
            fin si
        end for


    fin si
    l := l - 1;
    x := x + 1;
    y := y + 1;
  fin mientras
fin proc
```

Dada su complejidad, en la traducción a instrucciones de la máquina necesitaremos más posiciones de memoria y más instrucciones, por lo que para poder ejecutar el programa *tendrás que cambiar* las constantes `MAX_POSICIONES` y `MAX_INSTRUCCIONES`.

El programa en el ensamblador es:

```
; Constantes
; [100] = 1
; [101] = '|' [102] = '-'
; [105] = ESQSUPIZ [104] = ESQSUPDER
; [106] = ESQINFIZ [103] = ESQINFDER
STORE [100] 1
STORE [101] 179
STORE [102] 196
STORE [103] 217
STORE [104] 191
STORE [105] 218
STORE [106] 192
; Direcciones de las etiquetas para los saltos
; [500] = condBucle        [501] = cuerpoBucle
; [502] = casoBase         [503] = primerFor
; [504] = contadoresBucle  [505] = condFor1
; [506] = segundoFor       [507] = for1If1
; [508] = for1CondIf2      [509] = for1Contadores
; [510] = for1Else         [511] = fin
; [512] = condFor2         [513] = for2IfThen
; [514] = for2IfElse       [515] = for2Contadores
STORE [500] 27
STORE [501] 31
STORE [502] 33
STORE [503] 37
STORE [504] 90
STORE [505] 38
STORE [506] 66
STORE [507] 45
STORE [508] 51
```

```
STORE [509] 64
STORE [510] 59
STORE [511] 200
STORE [512] 67
STORE [513] 77
STORE [514] 83
STORE [515] 88
; Variables
; [300] = xaux     [301] = yaux     [302] = laux
; [303] = temporal para cálculos
; [304] = i
COPY [300] [10]
COPY [301] [11]
COPY [302] [12]
; condBucle
COPY [303] [302]
SUB [303] [100]
JUMPP [303] [501]
JUMPP [100] [511]
; cuerpoBucle
JUMPZ [303] [502]
JUMPP [100] [503]
; casoBase
COPY [1] [300]
COPY [0] [301]
WRITE [104]
JUMPP [100] [504]
; primerFor
STORE [304] 0
; condFor1
COPY [303] [304]
SUB [303] [302]
JUMPP [303] [506]
COPY [1] [300]
SUM [1] [304]
JUMPZ [304] [507]
JUMPP [100] [508]
; forIf1
COPY [0] [301]
WRITE [102]
SUM [0] [302]
SUB [0] [100]
WRITE [106]
JUMPP [100] [509]
; for1Cond2 (i <> l-1 <=> i-l+1 <> 0)
SUM [303] [100]
JUMPZ [303] [510]
COPY [0] [301]
WRITE [102]
```

```
SUM [0] [302]
SUB [0] [100]
WRITE [102]
JUMPP [100] [509]
; for1Else
COPY [0] [301]
WRITE [104]
SUM [0] [302]
SUB [0] [100]
WRITE [103]
; for1Contadores
SUM [304] [100]
JUMPP [100] [505]
; segundorFor
STORE [304] 1
; condFor2
COPY [303] [304]
SUB [303] [302]
SUM [303] [100]
JUMPP [303] [504]
COPY [0] [301]
SUM [0] [304]
; condicion del if
COPY [303] [304]
SUB [303] [100]
JUMPZ [303] [513]
JUMPP [100] [514]
; for2IfThen
COPY [1] [300]
WRITE [105]
SUM [1] [302]
SUB [1] [100]
WRITE [101]
JUMPP [100] [515]
; for2IfElse
COPY [1] [300]
WRITE [101]
SUM [1] [302]
SUB [1] [100]
WRITE [101]
; for2Contadores
SUM [304] [100]
JUMPP [100] [512]
; contadoresBucle
SUB [302] [100]
SUB [302] [100]
SUM [300] [100]
SUM [301] [100]
JUMPP [100] [500]
```

Para probar el programa, se puede utilizar el siguiente fichero de memoria de entrada:

```
10:7
11:2
12:8
```

La salida con esta configuración puede verse en la figura 1.