TRABAJO PRÁCTICO

TRADUCTOR ASSEMBLER - MÁQUINA VIRTUAL

PARTE II

0 - FORMATO DE ENTREGA

Se deberá entregar, por mail a los miembros de la cátedra, un archivo Zip, Rar o 7zip con el siguiente contenido:

- Los fuentes completos de la máquina virtual.
- El programa ejecutable (.EXE compilado para Windows o Linux).

PROGRAMA: Al programa de la máquina virtual se le agrega nuevos argumentos con archivos para utilizar.

```
mv.exe [-op] [codigo.asm] [maquina.img]  [archivo.ext]*
```

Donde:

- [-op] es la operación, al igual que en la primera parte: T: Traducción, X: Ejecución, A: Traducción y ejecución.
- [codigo.asm] es el código fuente en assembler de la MV.
- [maquina.img] es el archivo de imagen binario que genera la traducción.
- [archivo.ext]* es una lista nombres de archivos con cualquier extensión.

Todos los argumentos pueden, o no, ser opcionales dependiendo de la configuración:.

- En ausencia de [-op] la opción por defecto es A (Traducción y ejecución).
- Se debe especificar *.asm cuando las opciones son A o T
- Se debe especificar *.img cuando la opción es X, si no está el nombre del archivo y la opción es A no se crea la imagen.
- La lista de archivos siempre va al final y comienzan a numerarse desde 0 (cero), la lista es obligatoria dependiendo del uso en el código de la máquina, y pueden ser archivos de cualquier extensión.

El orden de los argumentos siempre es el mismo, sin embargo hay que tener en cuenta que algunos argumentos pueden estar ausentes.

1 - INTRODUCCIÓN

En esta segunda parte del trabajo práctico se deberá ampliar la máquina virtual para que soporte instrucciones de manejo archivos, Strings, de pila, llamado y retorno de subrutinas; además de un nuevo tipo de operando: el operando indirecto y un nuevo tipo símbolo: constantes.

También se incrementa el tamaño de la memoria a 32KiB, es decir 8192celdas de 4 bytes, se incorpora un nuevo segmento de memoria: “Stack Segment” con su respectivo registros de control: SS.

2 - MANEJO DE ARCHIVOS

La máquina virtual podrá crear, abrir, leer/escribir, cerrar y eliminar archivos por medio de System Calls (interrupciones). Los archivos tendrán un *handle* (número identificador de archivo) para referenciarlos y se vinculan con archivos de la máquina real que han sido pasado como parámetros.

Por ejemplo si se ejecuta la máquina virtual:

```
mv.exe -x programa.img archivo1.txt archivo2.dat
```

archivo1.txt tendrá *handle* = 0 y archivo2.dat tendrá *handle* = 1.

Internamente la máquina deberá manejar una tabla de archivos (máximo 10 archivos), donde contenga para cada uno: el handle, el nombre, si está abierto o no.

Para hacer las operaciones sobre un archivo se utilizará el System Call: SYS, donde el argumento indicará la operación:

Código	Significado
3D	OPEN FILE modo lectura escritura
3E	CLOSE FILE
3F	READ FILE
40	WRITE FILE
42	SEEK (saltar a una posición)

Se utilizará información adicional de los registros:

AX	Indicará cómo debe leer o escribir, del mismo modo que en todas las interrupciones
BX	Tiene el handle del archivo
CX	Indicará la cantidad (de bytes o líneas) a leer/escribir del archivo según AX Si AX = %0002 => CX cuenta celdas (4Bytes) Si AX = %0010 => CX cuenta bytes Si AX = %0100 => CX cuenta líneas
DX	Indicará la posición de memoria en la cual se guardan o extraen los datos. En caso del SEEK, se especifica SEEK_SET(0) si debe comenzar a contar desde el comienzo del archivo (CX>= 0), SEEK_CUR(1) si debe contar desde la posición corriente, SEEK_END(2) si debe contar desde el final del archivo (CX <= 0).

Ejemplos:

Abrir el archivo archivo2.dat

```
      mov BX,1
      SYS %3D
```

Leer 8 bytes del archivo y ponerlos en las celdas 1001 a 1002

```
      mov AX,%0002
      mov BX,1
      mov CX,2
      mov DX,1001
      SYS %3F
```

Finalmente cada vez que se utiliza la instrucción SYS, se setea el registro CC del siguiente modo:

- Si se produjo un error, en el CC se configura en 1 el bit de Signo.
- Si el puntero del archivo supera el final (EOF), en el CC se configura en 1 el bit de Cero.

De este modo se pueden manejar errores, y recorrer archivos, por ejemplo:

```
       mov AX,%0002
       mov BX,1
       mov CX,1
       mov DX,1001
       SYS %3D
leo:   JN error
       JZ sigue
       SYS %3F
       add DX,1
       jmp leo
sigue: …

error: …
```

3 - MANEJO DE LA PILA

El acceso a memoria tipo pila permite implementar en forma eficiente el trabajo con subrutinas (llamada, retorno, pasaje de parámetros, recursividad).

Por defecto, la pila tendrá un tamaño de 200 celdas. Para definir un tamaño distinto se puede indicar la cláusula "**STACK=tamaño**" a continuación de la palabra reservada \\ASM.

Por ejemplo, para definir una pila de 100 celdas se escribirá la orden:

```
\\ASM STACK=100
```

El segmento de pila se ubicará al final de la memoria RAM de la máquina virtual.

La palabra reservada \\ASM podría estar en cualquier línea de código, aunque por convención, de ser necesaria, se escribe al inicio del archivo.

REGISTROS

La máquina virtual deberá contar con tres registros adicionales para el manejo de pila: SS (Stack Segment), SP (Stack Pointer) y BP (Base Pointer).

El registro SS, indicará el inicio del segmento de pila. Su valor se obtiene del siguiente cálculo:

SS = 8192 - Tamaño de la pila

El registro SP apuntará al tope de la pila, tomando como valor inicial el tamaño de la misma.

SP = Tamaño de la pila

La pila va creciendo hacia las posiciones inferiores de memoria, por lo tanto el valor de SP se irá decrementando cuando se guarden datos en la pila y se aumenta cuando se sacan datos.

El registro BP, servirá para acceder a celdas dentro de la pila, haciendo uso del operando indirecto. Se puede utilizar para implementar pasaje de parámetros a través de la pila.

Cuando se utilicen los registros BP o SP en indirecciones, se deberá considerar el valor de SS como la dirección base para el cálculo de las direcciones efectivas (y no el valor del registro DS).

Es decir, que para el modo de direccionamiento indirecto, los registros BP y SP deben trabajar con el registro SS, mientras que el resto de los registros lo deben hacer con el registro DS.

Como los registros SP y BP pueden ser operandos de una instrucción, se deben codificar como se hizo con el resto de los registros de la máquina virtual:

Registro	Código Hexadecimal
SS	0004
SP	0006
BP	0007

DETECCIÓN DE ERRORES PILA

El programa debe detectar en tiempo de ejecución los siguientes errores:

1. Stack overflow (desbordamiento de pila): se produce cuando se intenta insertar un dato en la pila y ésta está llena.
2. Cuando se intenta sacar un dato de la pila y ésta está vacía.

En ambos casos se debe mostrar un mensaje por pantalla detallando el tipo de error y abortar la ejecución.

INSTRUCCIONES

Las nuevas instrucciones a implementar son:

INSTRUCCIÓN	COD HEXA
PUSH	44
POP	45
PUSHALL	46
POPALL	47
CALL	40
RET	48

PUSH: Permite almacenar un dato en el tope de la pila. Requiere un solo operando que es el dato a almacenar. El mismo puede ser de cualquier tipo.

```
PUSH AX  ;  decrementa en uno el valor de SP y almacena en la posición apuntada por este registro el valor de AX.
```

POP: Extrae el dato del tope de pila y lo almacena en el operando (que puede ser registro, memoria o indirecto).

```
POP [1000]  ;  el contenido de la celda apuntada por SP se almacena en la celda 1000 y luego el valor de SP se incrementa en 1.
```

PUSHALL: Permite almacenar todos los registros de uso general en el tope de la pila.
Equivale a:

```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH EX
PUSH FX
```

POPALL: Extrae del tope de la pula y asigna todos los valores a los registros de uso general.
Equivale a:

```
POP FX
POP EX
POP DX
POP CX
POP BX
POP AX
```

CALL: Permite efectuar un llamado a una subrutina. Requiere un solo parámetro que es la posición de memoria a donde se desea saltar (por supuesto, puede ser un rótulo).

CALL PROC1 ; se salta a la instrucción rotulada como PROC1. Previamente se guarda en la pila la dirección memoria siguiente a esta instrucción (dirección de retorno).

RET: Permite efectuar una retorno desde una subrutina. No requiere parámetros. Extrae el valor del tope de la pila y efectúa una salto a dicha dirección.

4 - MANEJO DE STRINGS

Se incorporan nuevas instrucciones para el manejo de cadena de caracteres (SMOV, SCMP, SLEN).

EN MEMORIA

Los strings se almacenan como secuencia consecutiva de caracteres en código ASCII, ocupando cada carácter una celda de memoria (de 32 bits)y se finaliza con con el carácter ´\0´ (es decir %00000000).

Posición de memoria relativa al origen del String	(+0)	(+1)	(+2)	(+3)	(+4)	(+5)	(+6)	(+7)	(+8)	(+9)	(+10)	(+11)
Valor Hexa en memoria (ultimo bytes)	48	6F	6C	61	20	6D	75	6E	64	6F	21	00
Carácter	H	o	l	a	*espacio*	m	u	n	d	o	!	*fin*

INSTRUCCIONES

Las nuevas instrucciones a implementar son: SMOV, SCMP y SLEN.

Instrucción	Código Hexadecimal
SLEN	50
SMOV	51
SCMP	53

SLEN: Permite obtener la longitud de un String. Tiene 2 operandos, el primero es donde se guarda la longitud, el segundo es la dirección de memoria donde se encuentra el String. El primer operando puede ser de registro, directo o indirecto; el segundo operando solo puede ser directo o indirecto.

SMOV: Permite copiar un String de una posición de memoria a otra con la misma mecánica del MOV. Tiene 2 operandos, el primero indica la dirección de memoria destino y el segundo la dirección de memoria origen. Los operandos pueden ser directos o indirectos.

SCMP: Permite comparar 2 Strings, consiste en restar el carácter apuntado por el primer operando menos el carácter apuntado por el segundo operando, y modificar el CC acorde al resultado obtenido. Si el resultado es 0 continua con el segundo carácter. Finaliza cuando la resta resulta distinto de 0 o cuando haya llegado al final de una de las cadenas. Los operandos pueden ser directos o indirectos.

Ejemplo: `SCMP [AX],[BX]`

`[AX] ->"Hola"` `[BX] ->"HOLA"`	`[AX] ->"0h"` `[BX] ->"0h"`	`[AX] ->"0h"` `[BX] ->"0h..."`
"H" - "H" = %48 - %48 = 0 "o"-"O"= %6F- %4F= 32 (32=%20=' ')	"O" - "O" = %4F - %4F = 0 "h" - "h" = %68 - %68 = 0 "\0" - "\0" = %00 - %00 = 0	"O" - "O" = %4F - %4F = 0 "h" - "h" = %68 - %68 = 0 "\0" - "." = 0 - %2E = -46
CC = %0000	CC = %0001	CC = %8000

A continuación se detallan las modificaciones de cada system calls, adaptadas a esta segunda parte.

SYS 1 (READ): Permite almacenar en un rango de celdas de memoria los datos leídos desde el teclado. Almacena lo que se lee en la posición de memoria apuntada por DX.
Si CX tiene un valor mayor a 0 lee tantos datos como se indique en CX, poniendo un dato en una celda de memoria contigua partiendo desde la apuntada por DX. Si CX es igual a -1, se lee de a un byte/caracter y se guarda 1 por celda (utilizando el byte menos significativo de cada celda), hasta que se lea el fin de línea y se guarda en la última celda '/0'. Si CX es igual a 0, no se lee nada.
El modo de lectura depende de la configuración de cada bit del registro AX, no se modifica.

SYS 2 (WRITE): Muestra en pantalla el contenido del rango de celdas especificado. Comenzando por la celda apuntada por DX y muestra una cantidad de celdas CX, si CX tiene el valor -1 entonces imprime todas las celdas desde DX hasta que encuentra un %0000 (no imprimible).
Al igual que la lectura, la forma de escritura se configura en AX.

SYS 3 (DUMP): Es igual al SYS 2 (WRITE) y además muestra el contenido de todos los registros, sin cambios.

5 - SÍMBOLOS

El lenguaje assembler de MV, ya disponía de un tipo de símbolo: los rótulos (o labels), a estos se le agregan las constantes.

CONSTANTES

Una constante se define por la directiva EQU. Por ejemplo:

```
BASE  EQU  #16
```

Hace que el símbolo BASE tenga el valor 16 decimal. Luego, dicho símbolo podrá utilizarse en una instrucción. Por ejemplo:

```
ADD AX,BASE
```

Suma 16 al registro AX.

Las directivas EQU no son instrucciones ejecutables y no generan código máquina (son pseudo-instrucciones), por lo tanto son ignoradas al contar las líneas de código del mismo modo que los comentarios. Suelen colocarse al principio del programa, pero podrían estar ubicadas en cualquier parte sin que afecte a la ejecución. Deben soportar las mismas bases que maneja el lenguaje: octal, decimal, hexadecimal o carácter.

NOTAS ADICIONALES

- En el manejo de símbolos no se deben diferenciar mayúsculas de minúsculas.
  (Ej:‘TOPE’, ‘Tope’ y ‘tope’ son el mismo símbolo).

- La longitud máxima de un símbolo es de 10 caracteres.

- Las constantes al igual que los rótulos, se resuelven en la traducción y comparten la misma tabla. Es decir si existe un rótulos llamado “otro” que se le asigna la línea 10, no puede existir una constante llamada “otro”, porque se tomará como símbolo duplicado.

- Generalizando, tanto los rótulos como las constantes se reemplazan como **valores inmediatos** dentro de las instrucciones pudiéndose utilizar indistintamente en cualquier instrucción que admita un argumento inmediato.

DETECCIÓN DE ERRORES EN SÍMBOLOS

El traductor debe ser capaz de detectar errores de **símbolos duplicados e indefinidos**. En ambos casos deberá informar el error, continuar con la traducción pero no realizar la ejecución del programa.

6 - OPERANDO INDIRECTO

La máquina virtual debe ser capaz de manejar un nuevo tipo de operando: el indirecto, que se codifica como 11 (binario).

Por ejemplo:

```
MOV AX , #1000
MOV BX , [AX]
```

Almacena en BX el contenido de la dirección de memoria apuntada por AX (en este caso el valor de la celda 1000).

Dependiendo del registro utilizando varia el registro base:

- Si se utilizan los registros AX a FX la dirección es relativa al DS.
- Si se utilizan los registros BP o SP la dirección es relativa al SS.
- No se permite el uso de los registros IP, CC, DS o SS.

Además, a este tipo de operando se le puede añadir una constante positiva o negativa (en base 10) o un símbolo.

Ejemplo 1:

```
MOV CX , [AX+2]
```

Suponiendo que AX vale 1000, esta instrucción almacena en CX el contenido de la celda de memoria 1002.

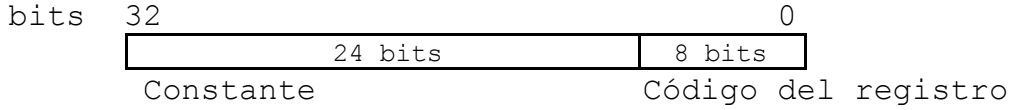
Ejemplo 2:

```
Vector EQU #100
       MOV DX , [BX+Vector]
```

Suponiendo que BX vale 3, esta instrucción almacena en DX el contenido de la celda de memoria 103. Resumiendo, el formato de un operando indirecto es:

[<registro> {+ / **-** <valor decimal>/<símbolo>}**]** (las llaves indican partes opcionales)

En la celda de memoria RAM donde se almacene el operando indirecto, se debe registrar información sobre el registro y la constante. Para ello se usará el siguiente formato.



Por ejemplo, el operando [CX+23], se debe codificar de la siguiente forma:

```
Constante 23:   00000000 00000000 00010111          b
Registro  CX:   ________________________ 00001100 b
                00000000 00000000 00010111 00001100 b (%170C)
```