LENGUAJE ASSEMBLER

MÁQUINA VIRTUAL - PARTE II

Cambios en la máquina virtual

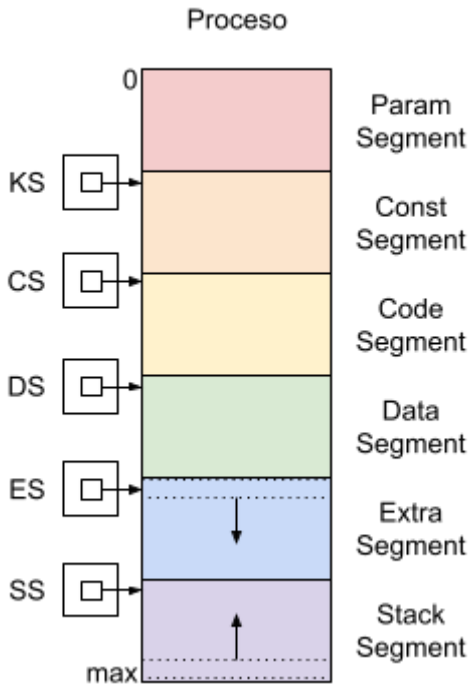
En esta segunda parte, la máquina virtual será capaz de trabajar con varios segmentos, cadenas de caracteres, símbolos, subrutinas y operandos de memoria con modificadores.

Cada proceso en la memoria quedará conformado por hasta seis segmentos:

- ***Param Segment***: contiene los argumentos pasados como parámetros a la ejecución.
- ***Code Segment***: contiene el código fuente; es apuntado por el registro CS.
- ***Data Segment***: se utiliza para los datos del proceso; es apuntado por el registro DS.
- ***Extra Segment***: reservado para el uso de memoria dinámica; es apuntado por el registro ES.
- ***Stack Segment***: dedicado exclusivamente para la pila del proceso; es apuntado por el registro SS.
- ***Const Segment***: reservado para el uso de constantes *strings*; es apuntado por el registro KS.

Los registros quedarán dispuestos de la siguiente manera:

Posición	Nombre	Descripción
0	CS	Segmentos
1	DS	
2	ES	
3	SS	
4	KS	
5	IP	Instruction Pointer
6	SP	Stack
7	BP	
8	CC	Condition Code
9	AC	Accumulator
10	EAX	General Purpose Registers
11	EBX	
12	ECX	
13	EDX	
14	EEX	
15	EFX	



Los registros **CS**, **DS**, **ES, SS y KS** contienen el puntero a sus respectivos segmentos. Los punteros se conforman del mismo modo que en la primera parte: los 2 bytes más significativos son la entrada en la tabla de descriptores de segmentos y los 2 bytes menos significativos el offset que debe ser 0.

Los registros **SP** y **BP** serán registros que utilizará el programador exclusivamente para manejar la pila.

Directivas

Las directivas no forman parte del lenguaje *assembler*, pero permiten dar indicaciones al traductor. Las directivas se pueden realizar en cualquier línea del código (normalmente suelen ser las primeras), indicando con \\ al comienzo de la misma. No puede haber ninguna otra instrucción o comentario en la misma línea.

Directiva *include*

Mediante esta directiva se puede incluir código *assembler* de otro archivo. En el proceso de traducción, el código se inserta en el lugar donde se encuentre la directiva. La sintaxis es la siguiente:

```
\\INCLUDE "filename.asm"
```

Donde ***filename.asm*** es la ruta y nombre del archivo que contiene el código a incluir. La ruta puede ser absoluta o relativa a la del archivo donde se encuentra la directiva. No está permitido hacer referencias circulares.

Directivas de definición de segmentos

El tamaño de algunos segmentos se puede definir mediante directivas que indican la cantidad de bytes que se deben reservar para cada uno. La sintaxis es la siguiente:

```
\\<SEGMENTO> <TAMAÑO>
```

Donde:
- ***SEGMENTO*** puede ser: ***DATA*** (*Data Segment*), ***EXTRA*** (*Extra Segment*) o ***STACK*** (*Stack Segment*).
- ***TAMAÑO*** es la cantidad (en decimal) de celdas de memoria destinadas a cada segmento.

Por ejemplo:

```
\\DATA 10
\\EXTRA 3000
\\STACK 5000
```

No necesariamente deben estar definidos los tres segmentos y tampoco en el mismo orden, pero cada directiva puede estar solo una vez en el código. Si alguno de los segmentos no está definido, se utiliza el tamaño 1024 como valor por defecto. El tamaño del *Code Segment* y del *Const Segment* no los define el programador directamente, sino que se calculan en el momento de la traducción, según el contenido del archivo. El tamaño del *Param Segment* debe ser calculado por la máquina virtual antes de cada ejecución.

El tamaño del segmento debe estar entre 0 (0x0000) y 65535 (0xFFFF). Aunque el programador no debería designar un tamaño mayor a la memoria disponible, 65535 es el máximo teórico que la arquitectura permitiría como tamaño de segmento. Sin embargo, la máquina virtual sólo dispondrá de una cantidad específica de memoria y, por lo tanto, la sumatoria del tamaño de los segmentos no podrá superarla.

Cadenas de caracteres

Las cadenas de caracteres (*strings*) se almacenan como secuencia consecutiva de caracteres en código ASCII (uno por cada celda de memoria de 1 byte) y finalizan con el carácter '\0' (0x00). Por ejemplo:

Offset	0	1	2	3	4	5	6	7	8	9	10	11
Hexadecimal	48	6F	6C	61	20	6D	75	6E	64	6F	21	00
Carácter	H	o	l	a		m	u	n	d	o	!	\0

Símbolos

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

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 o como el *offset* de un operando de memoria. Todos los símbolos se resuelven en la traducción y comparten la misma tabla, por lo que un rótulo y una constante no pueden compartir el mismo nombre.

Las líneas que definen las constantes no son instrucciones ejecutables y no generan código máquina. Suelen colocarse al principio del programa, pero podrían estar ubicadas en cualquier parte sin que afecte a la ejecución.

Constantes inmediatas

Una constante inmediata se define por la directiva *EQU*. Soporta las mismas bases que maneja el lenguaje: octal, decimal, hexadecimal y carácter. Por ejemplo:

```
BASE EQU 16
...
ADD EAX, BASE
```

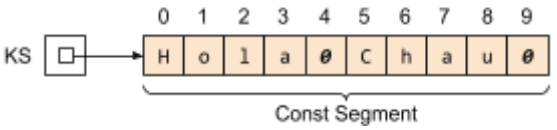
El símbolo *BASE* toma el valor 16 decimal. Por lo tanto, se suma 16 al registro EAX.

Constantes *strings*

Una constante *string* es similar a una inmediata, pero permite almacenar una cadena de caracteres en el *Const Segment* y el valor de la constante es el *offset* dentro del segmento. Por ejemplo:

```
TEXTO1 EQU "Hola"
TEXTO2 EQU "Chau"
```

El carácter 'H' de *TEXTO1* se almacenará en la celda apuntada por KS y el '\0' en la celda KS+4. El carácter 'C' de *TEXTO2* se almacenará en la celda KS+5 y el '\0' en la celda KS+9. Por lo tanto, el traductor reemplazará *TEXTO1* por el valor 0 y *TEXTO2* el valor 5, es decir el *offset* dentro del *Const Segment*.



Llamadas al sistema

3 (STRING 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 EDX. En CX (16 bits) se especifica la cantidad máxima de caracteres a leer. Por ejemplo:

```
MOV EDX, DS
ADD EDX, 123
MOV CX, 50
SYS 0x03
```

Si el usuario ingresa "Hola", lo almacenará comenzando por la celda 123 del *Data Segment*, donde colocará la 'H' y en la celda 127 colocará el carácter '\0'. Si en CX hubiera un 3 (en lugar de 50), habría almacenado "Hol" y '\0' en la celda 126.

4 (STRING WRITE): permite imprimir por pantalla un rango de celdas donde se encuentra un *string*. Inicia en la posición de memoria apuntada por EDX. Por ejemplo:

```
TEXTO EQU "Hola\n"
MOV EDX, KS
ADD EDX, TEXTO
SYS 0x04
```

Muestra "Hola" en la pantalla y baja una línea. Si el carácter '\n' no estuviera, el cursor se quedaría posicionado al final de la línea.

7 (CLEAR SCREEN): ejecuta una limpieza de pantalla. No requiere ningún registro configurado y tampoco modifica ninguno.

F (BREAKPOINT): pausa la ejecución y genera un archivo imagen con el estado actual de la máquina virtual. No requiere ningún registro configurado y tampoco modifica ninguno.

Operandos de memoria

Los operandos de tipo de acceso a memoria ahora soportan un modificador que permite indicar cuántos bytes serán afectados por la operación en la que se encuentre. Los modificadores son caracteres que se escriben como prefijos de los operandos y pueden ser:

b[...]	1 byte
w[...]	2 bytes
l[...]	4 bytes
[...]	

Por ejemplo:

```
MOV B[0], al     ;copia a la celda 0 del Data segment el contenido de AL.
MOV CX, w[edx]   ;copia a CX el contenido de las 2 celdas de un byte apuntadas por EDX.
MOV w[2], L[10]  ;copia el byte 12 al 2 y el 13 al 3 del Data segment.
```

Gestión de la pila

El *Stack Segment* será para uso exclusivo de la pila del proceso. La pila permite implementar de forma eficiente el trabajo con subrutinas: llamadas, retorno, pasaje de parámetros y recursividad.

Registros

Para trabajar con la pila, además del registro SS que contiene el puntero al comienzo del segmento, se utilizan los registros SP (*Stack Pointer*) y BP (*Base Pointer*). Siempre que estos registros se utilicen en indirecciones deben ser relativos al SS.

El SP se utiliza para apuntar al tope de la pila. Se inicializa con el 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 de memoria. Se puede utilizar para implementar pasaje de parámetros a través de la pila.

Instrucciones

PUSH: almacena un dato de 4 bytes en el tope de la pila. Requiere un solo operando que puede ser de cualquier tipo. Primero decrementa en 4 el valor del registro SP y luego guarda el valor del operando en la posición de memoria apuntada por SP.

```
PUSH AX     ;almacena en el tope de la pila el valor de AX expandido a 4 bytes
```

POP: extrae el dato del tope de pila y lo almacena en el único operando (puede ser de registro o memoria). Luego incrementa en 4 el valor del registro SP.

```
POP [1000]  ;almacena en la celda 1000 del data segment el valor en el tope de la pila
```

CALL: efectúa un llamado a una subrutina. Requiere un solo operando que puede ser de cualquier tipo. Primero almacena en el tope de la pila el valor del IP, que indica la dirección de memoria a la que se retornará luego de que la subrutina finalice. Luego, realiza un salto a la posición de memoria, relativa al CS, indicada por el operando.

```
CALL PROC1  ;ejecuta la subrutina rotulada como PROC1
CALL [ECX]  ;obtiene el valor X de la celda de memoria [ECX] y ejecuta la subrutina en CS+X
```

RET: efectúa una retorno desde una subrutina. No requiere parámetros. Extrae el valor del tope de la pila y realiza un salto a esa dirección de memoria.

Subrutina principal

El lenguaje *assembler* de la máquina virtual posee un rótulo reservado *“main”* (denominado *entry point*) que se utiliza para indicar la subrutina principal, que es dónde debe comenzar la ejecución del programa. Si el rótulo no existe, se utiliza por defecto la primera instrucción del programa. Además, al comenzar la ejecución, la máquina virtual deja cargada la pila con los siguientes valores:

- El tope de la pila contiene la dirección de retorno. Dado que se trata de la subrutina principal, esta dirección es un puntero fuera de *Code Segment* que ocasiona la finalización del proceso al ejecutar la instrucción RET.
- Un número entero que indica la cantidad de parámetros que se le han pasado a la máquina virtual al comenzar la ejecución del proceso.
- Un puntero al arreglo de punteros a *strings* (los parámetros).