LENGUAJE ASSEMBLER

MÁQUINA VIRTUAL - PARTE I

Descripción de la máquina virtual

El proceso (programa en ejecución) en la memoria principal se divide en dos segmentos:

- El **segmento de código** (*Code Segment*) almacena el código del programa en lenguaje máquina.
- El **segmento de datos** (*Data Segment*) se utiliza para almacenar datos durante la ejecución.

Registros

La máquina virtual posee 16 registros de 4 bytes, pero solo se utilizan 11 en esta primera parte.

Posición	Nombre	Descripción
0	CS	Segmentos
1	DS	
2		Reservado
3		
4		
5	IP	Instruction Pointer
6		Reservado
7		
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 y DS** almacenan los punteros al comienzo de los segmentos de código y datos, respectivamente.

El **registro IP (Instruction Pointer)** se usa para apuntar a la próxima instrucción a ejecutar dentro del segmento de código.

El **registro AC (Accumulator)** se utiliza para algunas operaciones especiales y también podrá ser utilizado para almacenar datos auxiliares.

El **registro CC (Condition Code)** informa sobre el resultado de la última operación matemática o lógica ejecutada. De los 32 bits que tiene, solamente se usarán dos:

- El bit más significativo es el '**bit indicador de signo**' (bit N). Valdrá 1 cuando la última operación matemática o lógica haya dado por resultado un valor negativo, y 0 en otro caso.
- El segundo bit más significativo es el '**bit indicador de cero'** (bit Z). Valdrá 1 cuando la última

operación matemática o lógica haya dado por resultado cero, y 0 en cualquier otro caso.

Los **registros de uso general (General Purpose Registers)**, desde EAX hasta EFX, sirven para almacenar datos y realizar cálculos durante la ejecución.

Formato de la instrucción

Cada línea del programa fuente puede contener una sola instrucción. Cada instrucción se compone como máximo de **un rótulo**, **un mnemónico** (palabra que representa un código de operación), **dos, uno o ningún operando** separados por coma (dependiendo del tipo de instrucción) y **un comentario**. Con la siguiente sintaxis:

```
RÓTULO:     MNEMÓNICO     OPN_A, OPN_B ;COMENTARIO
```

Lo único obligatorio para ser considerado instrucción es el **mnemónico**, todo lo demás (dependiendo de la instrucción) puede no estar o ser opcional. Puede haber líneas de código que estén en blanco o que solo tengan comentarios.

La disposición de las instrucciones en el programa fuente no están sujetas a una tabulación predeterminada, es decir que puede haber una cantidad variable de caracteres separadores delante de una instrucción y entre las partes que la forman. Se consideran separadores a los espacios en blanco y al carácter de tabulación.

Operandos

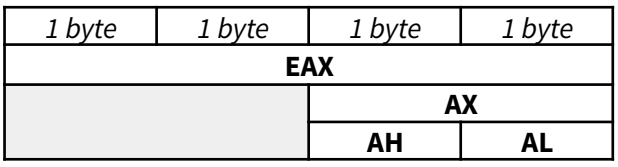
Operando inmediato

El dato es directamente el valor del operando. Se pueden tener valores numéricos en base 10, 8 y 16, que se representan con los símbolos #, @ y % respectivamente delante del dato. El símbolo # es opcional. Además se puede usar el apóstrofe (') para indicar valores ASCII.

Ejemplos:
#65 o **65**, **@101**, **%41**
'A o **'A'** (valor decimal 65)
fin (rótulo)

Operando de registro

El dato es el contenido, o parte, de alguno de los registros de la máquina virtual. Se especifican por el identificador del registro. Para los registros EAX a EFX, se puede especificar un “pseudónimo” para identificar el sector y la cantidad de bytes del registro a la que se accede. El prefijo “E” indica “extendido” quiere decir que se accede a los 4 bytes, sin prefijo se accede sólo a los 2 bytes menos significativos y con el sufijo L o H (en lugar de la X) indica el byte bajo (L = low) o el byte alto (H = high) como se indica en la figura:



IMPORTANTE: el registro solo tiene 4 bytes (32 bits). Si se asigna un valor al EAX, y luego otro valor al AX, esta última operación sobrescribe los últimos 2 bytes del registro EAX.

Ejemplos:

EAX o **eax** (case insensitive) accede al registro extendido de 32 bits EAX
FH accede al tercer byte del registro EFX

Operando de memoria

El dato se conforma con los 4 bytes ubicados en memoria a partir de la dirección indicada. Se utiliza el siguiente formato:

[<registro>±<desplazamiento>]

El registro debe contener un puntero a una dirección de memoria (como se definió previamente), y el desplazamiento es un número entero positivo, el cual puede sumar o restar posiciones de memoria.

Tanto el registro como el desplazamiento son opcionales, pero al menos uno de los dos debe estar presente. Si se omite el registro, se utilizará el DS. Si no se indica un desplazamiento, se asume 0.

Ejemplos:

[EDX] se accede a la posición de memoria apuntada por el registro EDX (que debe estar correctamente conformado como un puntero)
[EBX+10] se obtiene la dirección de memoria apuntada por EBX y se desplaza 10 bytes para acceder al dato.
[ECX-4] se accede al valor que se encuentra 4 bytes antes de la dirección apuntada por ECX.
[DS+8] es equivalente a **[8]**

Instrucciones (mnemónicos)

Instrucciones con 2 operandos

MOV: asigna a un registro o posición de memoria un valor, que puede ser el contenido de otro registro, posición de memoria o un valor inmediato.

```
MOV EAX,10          ;Carga en EAX el valor 10 decimal
MOV EBX,CX          ;Carga en EBX el valor de los 16 bit menos significativos del registro CX
MOV EDX,[12]        ;Carga en EDX 4 bytes desde la celda de memoria 12 hasta la 15.
```

ADD, SUB, MUL, DIV: realizan las cuatro operaciones matemáticas básicas. El primer operando debe ser de registro o memoria, ya que es donde se guarda el resultado. El resultado de estas instrucciones afecta el valor del registro CC. El DIV tiene la particularidad de que además guarda el resto de la división entera (módulo) en AC.

```
ADD EAX,2           ;Incrementa EAX en 2
MUL EAX,CL          ;Multiplica EAX por CL dejando el resultado en EAX
SUB [EBX+10],1      ;Resta 1 del valor de la celda de 4 bytes apuntada por EBX+10
DIV [10],7          ;Divide el valor de la celda 10 por 3,
                    ;el resultado queda en la celda 10 y en AC el resto
```

SWAP: intercambia los valores de los dos operandos (ambos deben ser registros y/o celdas de memoria).

CMP: similar a la instrucción SUB, el segundo operando se resta del primero, pero éste no almacena el resultado, solamente se modifican los bits NZ del registro CC. Es útil para comparar dos valores y generalmente se utiliza antes de una instrucción de salto condicional.

```
CMP EAX,[1000]     ;Compara los contenidos de EAX y la celda 1000
```

AND, OR, XOR: efectúan las operaciones lógicas básicas bit a bit entre los operandos y afectan al registro CC. El resultado se almacena en el primer operando.

```
AND AX,BX          ;efectúa el AND entre AX y BX, el resultado queda en AX
```

SHL, SHR: realizan desplazamientos a izquierda o a derecha, respectivamente, de los bits almacenados en un registro o una posición de memoria. También afectan al registro CC.

En SHL los bits derechos que quedan libres se completan con ceros.

En SHR los bits de la derecha propagan el bit anterior, es decir si el contenido es un número negativo el resultado seguirá siendo negativo, porque agrega 1. Si era un número positivo, agrega 0.

```
SHL AX,1           ;corre los 16 bits de AX una posición a la izquierda
                   ;No invade los 16 bits altos de EAX (equivale a multiplicar AX por 2).

SHR [200],BX       ;corre a la derecha los bits de la celda de 4 bytes que comienza en 200,
                   ;la cantidad de veces indicada en BX.
```

RND: carga en el primer operando un número aleatorio entre 0 y el valor del segundo operando.

Instrucciones con 1 operando

SYS: ejecuta la llamada al sistema indicada por el valor del operando.

JMP: efectúa un salto incondicional a la celda del segmento de código indicada en el operando.

```
JMP 0 ;asigna al registro IP la dirección de memoria 0 donde se almacenó la instrucción 1.
```

JZ, JP, JN, JNZ, JNP, JNN: realizan saltos condicionales en función de los bits del registro CC. Requieren de un solo operando que indica el desplazamiento dentro del segmento de código.

Instrucción	Bit N	Bit Z	Condición	Instrucción	Bit N	Bit Z	Condición
JZ (== 0)	0	1	Cero	**JNZ** (!= 0)	1	0	Negativo o positivo
					0	0	
JP (> 0)	0	0	Positivo	**JNP** (<= 0)	1	0	Negativo o cero
					0	1	
JN (< 0)	1	0	Negativo	**JNN** (>= 0)	0	1	Cero o positivo
					0	0	

```
JP 2        ;se salta a la celda 2 del CS si los bits N y Z de CC son cero. (>0)
JN BX       ;se salta a la celda indicada en BX si el bit N de CC está en 1. (<0)
JZ [8]      ;se salta a la celda indicada en la celda 8 si el bit Z de CC es 1. (==0)
JNZ fin     ;se salta a la celda con rótulo fin si el bit Z de CC está en cero. (!=0)
JNP fin     ;se salta a "fin" si el bit N está en 1 o el bit Z está en 1.(<=0)
```

LDH: carga en los 2 bytes más significativos del registro AC, con los 2 bytes menos significativos del operando. Esta instrucción está especialmente pensada para poder cargar un inmediato de 16 bits, aunque también se puede utilizar con otro tipo de operando.

LDL: carga en los 2 bytes menos significativos del registro AC, con los 2 bytes menos significativos del operando. Esta instrucción está especialmente pensada para poder cargar un inmediato de 16 bits, aunque también se puede utilizar con otro tipo de operando.

NOT: efectúa la negación bit a bit del operando y afectan al registro CC. El resultado se almacena en el primer operando.

```
NOT [15]          ;invierte cada bit del contenido de la posición de memoria 15.
```

Instrucciones sin operandos

STOP: detiene la ejecución del programa. No requiere parámetros.

Llamadas al sistema

1 (READ): permite almacenar los datos leídos desde el teclado a partir de la posición de memoria apuntada por EDX, guardándolo en CL celdas de tamaño CH. El modo de lectura depende de la configuración almacenada en AL con el siguiente formato:

Valor	Bit	Significado
%08	3	1: interpreta hexadecimal
%04	2	1: interpreta octal
%02	1	1: interpreta caracteres
%01	0	1: interpreta decimal

Ejemplo 1:

Código	Pantalla

MOV AL, %01
MOV EDX, DS
ADD EDX, 11
MOV CL, 2
MOV CH, 2
SYS %1
	[XXXX]: 23
	[XXXX]: 25

Al finalizar la lectura, las posiciones de memoria 11 a 14 (relativas al DS) quedarán con los valores 0, 23, 0 y 25 (respectivamente).

Ejemplo 2:

Código	Pantalla

MOV AL, %02
MOV EDX, DS
ADD EDX, 11
MOV CL, 4
MOV CH, 1
SYS %1
	[XXXX]: H
	[XXXX]: o
	[XXXX]: l
	[XXXX]: a

Al finalizar la lectura, las posiciones de memoria 11 a 14 (relativas al DS) quedarán con los valores 72, 111, 108 y 97 (respectivamente).

2 (WRITE): muestra en pantalla los valores contenidos a partir de la posición de memoria apuntada por EDX, recuperando CL celdas de tamaño CH. El modo de escritura depende de la configuración almacenada en AL con el siguiente formato:

Valor	Bit	Significado
%08	3	1: escribe hexadecimal
%04	2	1: escribe octal
%02	1	1: escribe caracteres
%01	0	1: escribe decimal

Cuando el caracter ASCII no es imprimible, escribe un punto (.) en su lugar.

Ejemplo 1:

Código	Pantalla

`MOV [3],'a'`
`MOV [2],'l'`
`MOV [1],'o'`
`MOV [0],'H'`
`MOV EDX, DS`
`ADD EDX, 3`
`MOV CH, 1`
`MOV CL, 4`
`MOV AX, %02`
`SYS %2`
	[XXXX]: H
	[XXXX]: o
	[XXXX]: l
	[XXXX]: a

Ejemplo 2:

Código	Pantalla

`MOV [10], %41`
`SHL [10], 8`
`OR [10], 'a'`
`MOV EDX, DS`
`ADD EDX, 12`
`MOV CL, 1`
`MOV CH, 2`
`MOV AL, %0F`
`SYS %2`
	[XXXX]: %4161 @40541 Aa 16737

NOTA: *XXXX* corresponde a la dirección de memoria de la celda.