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 32 registros de 4 bytes, pero solo se utilizan 17 en esta primera parte.

Código	Nombre	Descripción	Código	Nombre	Descripción
0	LAR	Acceso a memoria	16	AC	Acumulador
1	MAR		17	CC	Código de condición
2	MBR		18	-	Reservado
3	IP	Instrucción	19	-	
4	OPC		20	-	
5	OP1		21	-	
6	OP2		22	-	
7	-	Reservado	23	-	
8	-		24	-	
9	-		25	-	
10	EAX	Registros de propósito general	26	CS	Segmentos
11	EBX		27	DS	
12	ECX		28	-	Reservado
13	EDX		29	-	
14	EEX		30	-	
15	EFX		31	-	

- **LAR, MAR y MBR:** son utilizados por el procesador para comunicarse con la memoria principal.
- **IP (Instruction Pointer):** se usa para apuntar a la próxima instrucción a ejecutar dentro del segmento de código.
- **OPC, OP1 y OP2:** almacenan la instrucción en lenguaje máquina que se está ejecutando.
- **EAX a EFX:** sirven para almacenar datos y realizar operaciones durante la ejecución.
- **AC (Accumulator):** se utiliza para algunas operaciones especiales y también pueda ser utilizado para almacenar datos auxiliares.
- **CC (Condition Code):** contiene los bits N y Z que informan sobre el resultado de la última operación matemática o lógica ejecutada.
- **CS y DS:** almacenan los punteros al comienzo de los segmentos de código y datos, respectivamente.

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.

Operandos

Operando inmediato

El dato es directamente el valor del operando. Se pueden tener valores numéricos en base 10, 2, 8 y 16. Las bases 2, 8 y 16 se representan anteponiendo 0b, 0o (o simplemente 0) y 0x al dato, respectivamente. Además, se puede usar el apóstrofe (') para indicar valores ASCII.

Ejemplos:
97, **0o141** o **0141**, **0x61**, **0b1100001**
'a o **'a'** (valor decimal 97)
fin (rótulo)

Operando de registro

Accede a alguno de los registros de la máquina virtual, identificándolo por su nombre.

Ejemplos:
EAX, **AC**, **DS**, **CS**

Operando de memoria

Accede a los 4 bytes ubicados en la 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 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 dos 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,EDX        ;Carga en EAX el valor del registro EDX
MOV EBX,[8]        ;Carga en EBX 4 bytes desde la celda de memoria 8 hasta la 11
MOV [12],10        ;Carga desde la celda de memoria 12 hasta la 15 el valor decimal 10
```

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,[10]       ; multiplica EAX por el valor de la celda 10, dejando el resultado en EAX
SUB [EBX+10],1     ; resta 1 al valor de la celda de 4 bytes apuntada por EBX+10
DIV ECX,7          ; divide el valor de ECX por 7, el resultado queda en ECX y el resto en AC
```

CMP: similar a la instrucción SUB, el segundo operando se resta del primero, pero éste no almacena el resultado, solamente se modifican los bits N y Z 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
```

SHL, SHR, SAR: realizan desplazamientos de los bits almacenados en un registro o una posición de memoria y afectan al registro CC. SHL y SHR efectuan corrimientos a la izquierda y a la derecha (respectivamente) y los bits que quedan libres se completan con ceros. SAR también desplaza a la derecha, pero los bits de la izquierda propagan el bit anterior. Es decir, si el contenido es un número negativo, el resultado también lo será, porque agrega unos. Si es un número positivo, agrega ceros.

```
SHL EAX,1          ; corre los 32 bits de EAX una posición a la izquierda
                   ; (equivale a multiplicar EAX por 2)
SHR [200],EBX      ; corre a la derecha los bits de la celda 200,
                   ; la cantidad de veces indicada en EBX
SAR [50],2         ; corre los bits de la celda de 4 bytes que comienza en 50 dos posiciones
                   ; a la derecha, pero conservando el signo (equivale a dividir [50] por 4)
```

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 EAX,EBX        ; efectua el AND entre EAX y EBX, el resultado queda en EAX
```

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

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

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

Instrucciones con un 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 indicada por CS+2 si los bits N y Z de CC son cero (>0)
JN EBX      ; se salta a la celda indicada en EBX 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 la celda con rótulo fin si el bit N o el bit de Z está en 1 (<=0)
```

NOT: efectúa la negación bit a bit del operando y afecta al registro CC.

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

Instrucciones sin operandos

STOP: detiene la ejecución del programa.

Llamadas al sistema

1 (READ): permite almacenar los datos leídos desde el teclado a partir de la posición de memoria apuntada por EDX. El registro ECX indica la cantidad de celdas en los 2 bytes menos significativos y el tamaño de las mismas en los 2 bytes más significativos. El modo de lectura depende de la configuración almacenada en EAX con el siguiente formato:

Valor	Bit	Significado
0x10	4	1: interpreta binario
0x08	3	1: interpreta hexadecimal
0x04	2	1: interpreta octal
0x02	1	1: interpreta caracteres
0x01	0	1: interpreta decimal

Ejemplo 1:

Código	Pantalla

`MOV EAX, 0x01`
`MOV EDX, DS`
`ADD EDX, 11`
`LDL ECX, 2`
`LDH ECX, 2`
`SYS 0x1`
	[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 EAX, 0x02`
`MOV EDX, DS`
`ADD EDX, 11`
`LDL ECX, 4`
`LDH ECX, 1`
`SYS 0x1`
	[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. El registro ECX indica la cantidad de celdas en los 2 bytes menos significativos y el tamaño de las mismas en los 2 bytes más significativos. El modo de escritura depende de la configuración almacenada en EAX con el siguiente formato:

Valor	Bit	Significado
0x10	4	1: escribe binario
0x08	3	1: escribe hexadecimal
0x04	2	1: escribe octal
0x02	1	1: escribe caracteres
0x01	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`
`LDH ECX, 1`
`LDL ECX, 4`
`MOV EAX, 0x12`
`SYS 0x2`
	[XXXX]: 0b1001000 H
	[XXXX]: 0b1101111 o
	[XXXX]: 0b1101100 l
	[XXXX]: 0b1100001 a

Ejemplo 2:

Código	Pantalla

`MOV [10], 0x41`
`SHL [10], 8`
`OR [10], 'a'`
`MOV EDX, DS`
`ADD EDX, 12`
`LDL ECX, 1`
`LDH ECX, 2`
`MOV EAX, 0x0F`
`SYS 0x2`
	[XXXX]: 0x4161 0o40541 Aa 16737

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

Ejemplo de un programa completo

Ejemplo de un programa en *Assembler* para contar la cantidad de bits de un número ingresado.

```
inicio:     mov     eax, 0b01       ; seteo para leer en decimal
            mov     edx, DS         ; guardar en el data segment
            add     edx, 4          ; ...en la posición 1
            ldh     ecx, 0x04       ; leer celdas de 4 bytes
            ldl     ecx, 0x01       ; leer una sola celda
            sys     0x1             ; system call para leer
            xor      ac, ac         ; reseteo el ac (ac = 0)
            mov     eax, [edx]      ; copio 4 bytes de memoria a registro
otro:       cmp     eax, 0          ; comparo con cero
            jz      fin             ; si es cero terminé
            jnn   sigue             ; si no es negativo salta
            add      ac, 1          ; si es negativo acumula 1
sigue:      shl     eax, 1          ; desplazo un bit a la izquierda
            jmp    otro             ; continua el loop
fin:        add     edx, 4          ; incremento para usar otra posición
            mov   [edx], ac         ; copia a memoria el ac
            mov     eax, 0b01       ; seteo para escribir decimal
            ldh     ecx, 0x04       ; escribir celdas de 4 bytes
            ldl     ecx, 0x01       ; escribir una sola celda
            sys     0x2             ; system call para imprimir
            stop                    ; detiene la ejecución
```