TRABAJO PRÁCTICO

MÁQUINA VIRTUAL - PARTE I

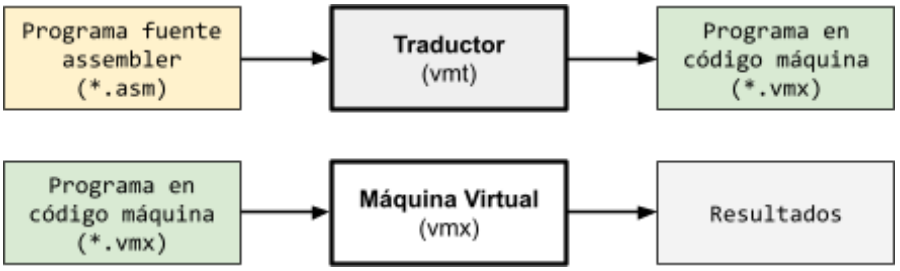
Introducción

El trabajo práctico consiste en realizar una aplicación, en un lenguaje de programación a elección, que emule la ejecución de un programa en el lenguaje máquina de una computadora que se describe en este documento. El programa a ejecutar se encuentra previamente escrito en el lenguaje *Assembler* de la máquina virtual y traducido a lenguaje máquina con el programa traductor (*vmt*) que provee la cátedra.

Procesos

TRADUCCIÓN (Traductor): debe leer el código fuente *Assembler* de un archivo de texto (**.asm*), traducirlo a código máquina y generar otro archivo binario codificado (**.vmx*), que es el programa que se ejecutará en la máquina virtual.

EJECUCIÓN (Máquina Virtual): debe leer el archivo generado por el Traductor (**.vmx*), configurar la memoria principal y los registros, interpretar las instrucciones y emular su funcionamiento para producir los resultados de su ejecución.



Traductor

El traductor, provisto por la cátedra, se utiliza desde una consola del siguiente modo:

```
vmt filename.asm [filename.vmx] [-o]
```

Donde:

- ***vmt*** es el programa ejecutable del Traductor.
- ***filename.asm*** (obligatorio) es la ruta y nombre del archivo de texto donde está escrito el código fuente que será traducido (puede ser cualquier nombre con extensión *.asm*).
- ***filename.vmx*** (opcional) es la ruta y nombre del archivo generado por el Traductor, que contiene el programa en lenguaje máquina (puede ser cualquier nombre con extensión *.vmx*). Si se omite, se crea un archivo con el mismo nombre que el ***.asm*** pero con extensión ***.vmx***. Si el archivo ya existe, se sobrescribe.
- ***-o*** (opcional) es un flag o bandera opcional para indicar que se omita la salida por pantalla de la traducción. Este flag no omite los mensajes de error producidos durante la traducción.

Máquina virtual

Se debe entregar el código fuente y el ejecutable compilado de la máquina virtual, la cual debe poder utilizarse desde una consola del siguiente modo:

```
vmx filename.vmx [-d]
```

Donde:

- ***vmx*** es el programa ejecutable del Ejecutor o Máquina Virtual.
- ***filename.vmx*** (obligatorio) es la ruta y nombre del archivo con el programa en lenguaje máquina (puede ser cualquier nombre con extensión ***.vmx***).
- ***-d*** (opcional) es un flag que fuerza a la máquina virtual a mostrar el código desensamblado (*disassembler)*, es decir, un código *Assembler* que se corresponde con el código máquina cargado en la memoria principal.

Componentes de la máquina virtual

La máquina virtual a implementar en esta primera parte, debe tener los siguientes componentes:

- Memoria principal (RAM) de 16 KiB
- Tabla de descriptores de segmentos
- 32 registros de 4 bytes (se utilizan 17 en esta primera parte)
- Procesador con capacidad para:
    - decodificar instrucciones en lenguaje máquina
    - direccionar a cada byte de la memoria principal
    - realizar operaciones aritméticas y lógicas en 32 bits

Memoria principal

La memoria principal de la máquina es donde se encontrará íntegramente el código y los datos del programa en ejecución (proceso). La memoria deberá tener una capacidad para 16384 bytes (16 KiB). Las direcciones físicas de la memoria comienzan en 0 para acceder al primer byte (el byte más bajo) y 16383 para acceder al último (el byte más alto).

En esta primera parte, el **segmento de código** contendrá el programa completo en lenguaje máquina y se ubicará al comienzo de la memoria, mientras que el **segmento de datos** ocupará todo el resto de la memoria disponible.

Tabla de descriptores de segmentos

La tabla de descriptores de segmentos permite definir la ubicación y el tamaño de cada segmento del proceso en la memoria principal. Consta de 8 entradas de 32 bits, cada una se divide en dos partes: los primeros 2 bytes son para guardar la dirección física de comienzo del segmento (base) y los siguientes 2 bytes la cantidad de bytes que ocupa. Se inicializa en el momento de la carga del programa.

En esta primera parte, la primera entrada (posición 0) guardará la información del segmento de código, mientras que la segunda (posición 1) guardará la información del segmento de datos. Es decir que la tabla quedará conformada de la siguiente manera:

	Base (2 bytes)	**Tamaño (2 bytes)**
0	0	Tamaño del código
1	Tamaño del código	16 KiB - Tamaño del código

Registros

Si bien en esta primera parte la máquina virtual utilizará solo 17 registros, deberá tener la capacidad para almacenar 32, los cuales se codifican de la siguiente manera:

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	-	

Programa

El programa es el resultado de la traducción y el punto de entrada de la máquina virtual. Por convención, tiene extensión ***.vmx*** para ser identificado fácilmente como un archivo ejecutable por la máquina virtual. Además del código en lenguaje máquina, el programa binario posee al comienzo una cabecera con la siguiente estructura:

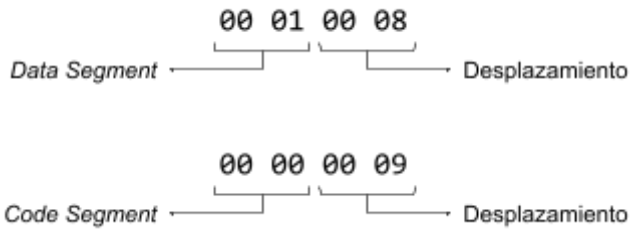
Nº byte	Campo	Valor
0 - 4	Identificador	“VMX25”
5	Versión	1
6 - 7	Tamaño del código	—

Direcciones de memoria principal

Direcciones lógicas (punteros)

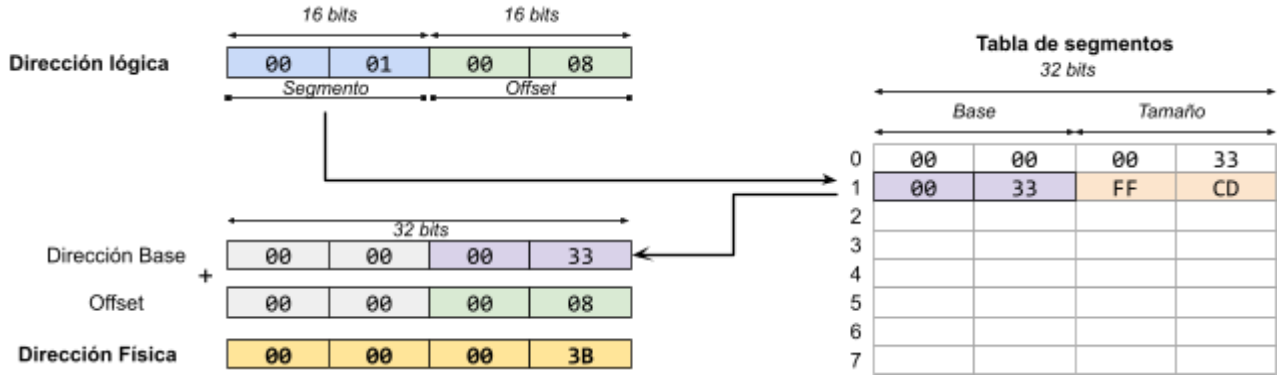
Cada segmento podría estar ubicado en cualquier parte de la memoria. Es por eso que el programa no puede tener una dirección física para acceder a una celda de memoria. En su lugar, debe utilizar **direcciones lógicas**, que son relativas a cada segmento. Durante la ejecución, la máquina virtual se encargará de traducir esa dirección lógica en una física y acceder a la celda de memoria específica.

Por lo tanto, para acceder a la memoria se debe conocer el segmento y un desplazamiento dentro del mismo. Un puntero a memoria consta de 4 bytes: **2 bytes para el código de segmento** y **2 bytes para el desplazamiento**. El código de segmento indica su posición en la tabla de descriptores de segmentos. Por ejemplo, para acceder al byte 8 del segmento de datos se deberá utilizar la dirección lógica 00 01 00 08 (hexadecimal). Si se debe acceder al byte 9 del segmento de código, se deberá utilizar la dirección lógica 00 00 00 09 (hexadecimal).

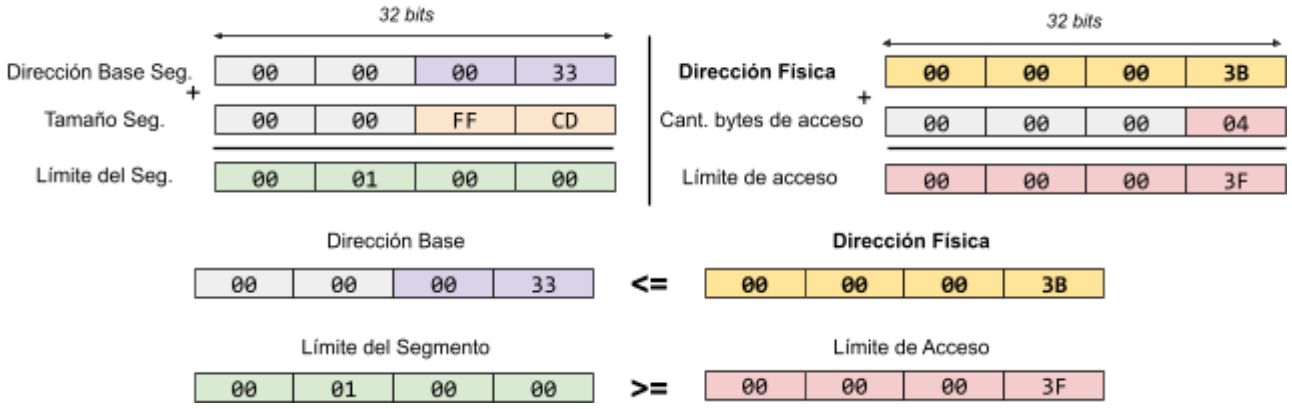


Direcciones físicas

La **dirección física** es directamente la posición de un byte en la memoria principal a partir de la cual se comienza a leer o escribir. Durante la ejecución, para acceder a un dato de la memoria, la máquina virtual debe traducir las direcciones lógicas en físicas en cada acceso. Para ello, se toma el código de segmento de la dirección lógica para obtener la dirección base del mismo, a través de la tabla de segmentos y, finalmente, se le suman los 16 bits menos significativos de la dirección lógica (el desplazamiento) para formar la dirección física de la memoria a la cual se debe acceder. Por ejemplo:



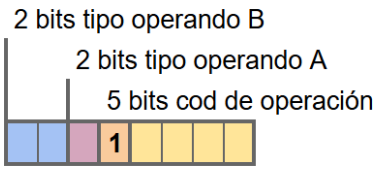
Luego de obtener la dirección física, y sabiendo cuántos bytes van a acceder, ya sea para lectura o escritura de la memoria, la máquina virtual debe garantizar que el acceso se encuentre dentro del segmento especificado en la dirección lógica, para ello debe utilizar el tamaño del segmento. por ejemplo, si se quieren acceder a 4 bytes desde la dirección física del ejemplo anterior:



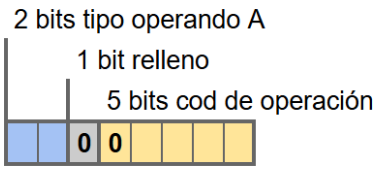
Instrucciones en lenguaje máquina

Cada instrucción en lenguaje máquina se compone de un código de operación y sus operandos. Existen instrucciones con dos operandos, un operando o ninguno. El primer byte de la instrucción siempre contendrá los tipos de operandos y el código de operación, codificados de la siguiente manera:

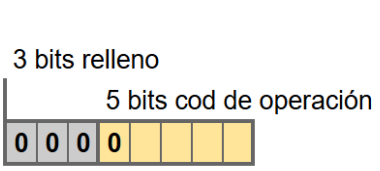
Instrucción con dos operandos



Instrucción con un operando



Instrucción sin operandos



Luego, los siguientes bytes contienen los operandos. **La instrucción no tiene una longitud fija**, sino que dependerá de la cantidad y los tipos de sus operandos. Tanto los operandos como sus tipos se codifican en lenguaje máquina en el orden inverso al que se encuentran en el lenguaje *Assembler*.

Códigos de operación

El lenguaje *Assembler* es una representación del lenguaje máquina, donde las instrucciones se describen con un **mnemónico**. En esta primera parte solo se implementarán 26 instrucciones, las cuales se listan a continuación junto con sus códigos de operación en hexadecimal.

Dos operandos		Un operando		Sin operandos	
Mnemónico	**Código**	**Mnemónico**	**Código**	**Mnemónico**	**Código**
MOV	10	SYS	00	STOP	0F
ADD	11	JMP	01		
SUB	12	JZ	02		
MUL	13	JP	03		
DIV	14	JN	04		
CMP	15	JNZ	05		
SHL	16	JNP	06		
SHR	17	JNN	07		
SAR	18	NOT	08		
AND	19				
OR	1A				
XOR	1B				
SWAP	1C				
LDL	1D				
LDH	1E				
RND	1F				

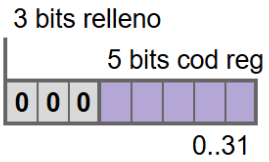
Operandos

La máquina virtual admite tres tipos de operandos, que codifican de la siguiente manera:

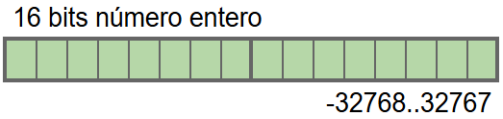
Código binario	Tipo	Tamaño
00	ninguno	0 bytes
01	registro	1 byte
10	inmediato	2 bytes
11	memoria	3 bytes

NOTA: el tamaño del operando en bytes coincide con su correspondiente código binario.

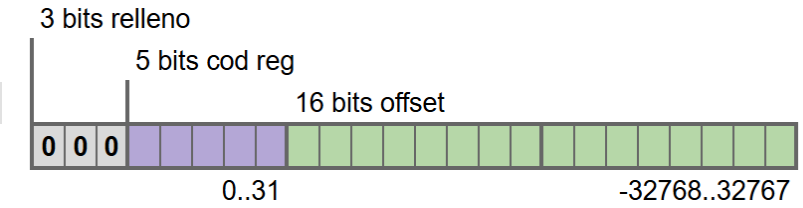
Operando de registro: su valor es el código del registro al cual se accede.



Operando inmediato: su valor es directamente el valor del operando.

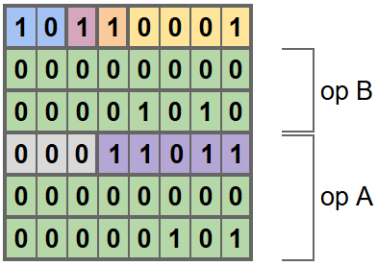


Operando de memoria: su valor se compone por el código del registro y el desplazamiento correspondientes. La posición en la memoria principal a la cual se accede es relativa al comienzo de algún segmento (es decir, una dirección lógica).

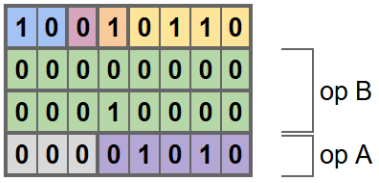


Ejemplos

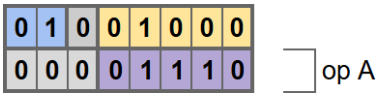
ADD [5], 10



SHL EAX, 0x10



NOT EEX



Llamadas al sistema

La instrucción SYS, en esta primera parte, debe soportar las llamadas al sistema READ (1) y WRITE (2). En ambos casos, la posición de memoria inicial estará indicada en EDX y el modo de lectura/escritura en EAX, mientras que ECX contendrá la cantidad de celdas en los 2 bytes menos significativos y el tamaño de las mismas en los 2 bytes más significativos. En la pantalla se debe mostrar un prompt (*[XXXX]:*) de 4 dígitos hexadecimales, que indique la dirección física de la celda en la que se encuentra cada dato.

Ejecución

Inicialmente, la máquina virtual debe leer el encabezado del programa para verificar si es capaz de ejecutar el programa y, de ser así, cargar el código en la memoria principal, armar la tabla de descriptores de segmentos e inicializar los registros.

Los registros CS y DS se cargan con punteros al comienzo del segmento de código y del segmento de datos, respectivamente. Es decir, en los 16 bits más significativos deberán almacenarse las posiciones de la tabla de descriptores de segmentos, mientras que los 16 bits menos significativos se rellenan con 0. Por lo tanto, CS será igual a 00 00 00 00 y DS será igual a 00 01 00 00 (en hexadecimal). Por otro lado, el registro IP debe inicializarse con un puntero a la primera instrucción del código. En otras palabras, al comienzo de la ejecución deberá tener el mismo valor que el registro CS.

A continuación comenzará la ejecución, la cual consiste en:

- Leer la instrucción apuntada por el registro IP
- Almacenar el código de operación de la instrucción en el registro OPC
- Guardar en los registros OP1 y OP2 los operandos A y B, respectivamente:
    - el byte más significativo del registro contendrá el código binario del tipo de operando
    - en los restantes tres bytes quedará el valor del operando, tal cual como está codificado en la instrucción almacenada en la memoria principal
    - en el caso de que el operando no exista, el registro tendrá un 0
- Ubicar el registro IP en la próxima instrucción (sumar el tamaño de la instrucción actual)
- Realizar la operación correspondiente a la instrucción

La ejecución se debe repetir hasta que el registro IP apunte fuera del segmento de código. Si se ejecuta una instrucción STOP, la máquina virtual debe asignar un -1 (0xFFFFFFFF) al registro IP. De esta manera, el código de segmento dejará de corresponderse con el segmento de código o con cualquier otro de los que podrían almacenarse en la tabla de descriptores de segmentos.

Cada vez que se realiza una operación en la memoria, se debe cargar en el registro LAR la dirección lógica a la que se quiere acceder y la cantidad de bytes en la parte alta del registro MAR (los 2 bytes más significativos). Luego de realizar la traducción a una dirección física, el resultado debe almacenarse en la parte baja del registro MAR (los 2 bytes menos significativos). En el registro MBR debe quedar el valor con el cual se está operando, ya sea el valor que se desea almacenar en el caso de una escritura o el que se obtuvo después de la lectura. La lectura de la instrucción no debe modificar ninguno de estos registros.

El registro CC debe informar sobre el resultado de la última operación matemática o lógica ejecutada. De los 32 bits que posee, solamente se usarán los primeros dos. El bit más significativo es el indicador de signo (bit N), que valdrá 1 cuando la última operación matemática o lógica haya dado por resultado un valor negativo y 0 en cualquier otro caso. El segundo bit más significativo es el indicador de cero (bit Z), que valdrá 1 cuando la última operación matemática o lógica haya dado por resultado cero y 0 en cualquier otro caso.

Errores

La máquina virtual debe ser capaz de detectar, al menos, los siguientes errores:

- **Instrucción inválida:** cuando el código de operación de la instrucción a ejecutar no existe.
- **División por cero:** cuando al ejecutar la instrucción DIV, el valor del segundo operando es 0.
- **Fallo de segmento:** cuando el código de segmento de una dirección lógica excede el tamaño de la tabla de descriptores o cuando la dirección física apunta a un byte que se encuentra fuera de los límites del segmento, excepto en el caso de la lectura de la instrucción.

Ante la ocurrencia de cualquiera de estos errores, la máquina virtual debe informarlo e inmediatamente abortar la ejecución del proceso.

Disassembler

Si a la máquina virtual se le indica que muestre el código *Assembler* (***-d***), deberá mostrar una línea por cada instrucción con el siguiente formato:

```
[0000] XX XX XX XX          | MNEM       OP_A,       OP_B
```

- **[0000]** es la dirección física de memoria donde está alojada la instrucción, expresada con 4 dígitos hexadecimales.
- **XX XX XX XX** es la instrucción completa (de longitud variable) en hexadecimal, agrupada por bytes.
- **MNEM** es el mnemónico correspondiente al código de la instrucción.
- **OP_A** y **OP_B** son los operandos A y B, respectivamente, expresados en decimal.

Por ejemplo:

```
[0000] B1 00 0A 1B 00 05    |  ADD     [DS+5],         10
[0006] 96 00 10 0A          |  SHL        EAX,         16
[000A] 48 0E                |  NOT        EEX
```

Los rótulos, comentarios y constantes con formato no pueden ser mostrados tal cual fueron escritos en el código *Assembler* porque no existen en el código máquina.