TRABAJO PRÁCTICO

MÁQUINA VIRTUAL - PARTE I

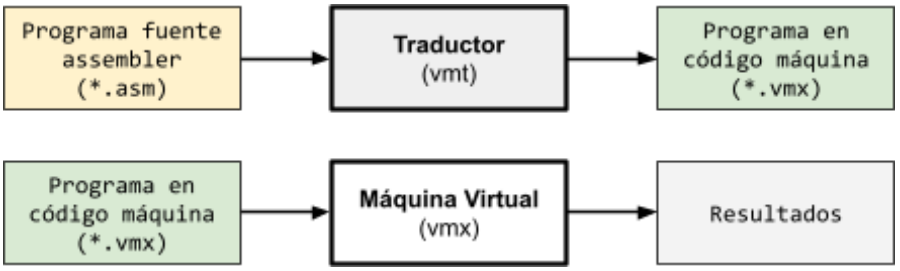
Introducción

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 con el programa (*vmt.exe*) que provee la cátedra.

Procesos

TRADUCCIÓN (Traductor): donde se 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): este proceso debe obtener las instrucciones a ejecutar desde el archivo generado por el proceso Traductor (**.vmx*), configurar la memoria principal y los registros, interpretar las instrucciones, emular su funcionamiento y producir los resultados de su ejecución.



Traductor

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

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

Donde:

- ***vmt.exe*** es el programa ejecutable del proceso 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.exe filename.vmx [-d]
```

Donde:
- ***vmx.exe*** es el programa ejecutable del proceso 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 *Assembler* correspondiente al código máquina cargado en la memoria principal.

Descripción 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
- 16 registros de 4 bytes (se utilizan 11 en esta primera parte)
- Procesador con capacidad para:
    - decodificar instrucciones en lenguaje máquina
    - direccionar a cada byte de la memoria principal

Ejecución

El archivo binario (****.vmx***) contiene un encabezado y el código de máquina a ejecutar. La máquina virtual, primero debe leer el encabezado para validar que se puede ejecutar y de ser así cargar el código en la memoria principal. Luego, armará la tabla de descriptores de segmentos e inicializará los registros. A continuación **comienza la ejecución**, la cual consiste en: leer la instrucción (apuntada por el registro IP), interpretar los operandos y la operación, ubicar el registro IP en la próxima instrucción y realizar la operación; y se repite hasta que se ejecute una instrucción STOP o el registro IP apunte fuera del segmento de código. Solo se puede ejecutar un único programa por vez.

Programa

Como se dijo previamente, el programa es el resultado de la traducción y el punto de entrada de la máquina virtual. Por convención, deberá tener extensión ***.vmx*** para ser identificado fácilmente como un archivo ejecutable por la máquina virtual. El programa binario tendrá, además del código máquina, información en una cabecera (*header*) para poder identificarlo como tal. De modo que el ejecutor examinará el *header* para determinar si es capaz de ejecutar ese programa y configurar el espacio de memoria asignado al mismo. Inmediatamente después del *header* se encuentra el código que se deberá cargar íntegramente en la memoria principal.

Header		
Nº byte	**Campo**	**Valor**
0 - 4	Identificador	“VMX24”
5	Versión	1
6 - 7	Tamaño del código	—

Memoria principal

La memoria principal de la máquina es donde se encontrará íntegramente el código y los datos del programa en ejecución (proceso). La máquina no posee sistema operativo, por lo que las funciones del mismo serán emuladas. Como se describió previamente, la memoria deberá tener una capacidad para 16384 bytes. 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).

Al iniciar la máquina virtual, se debe leer el archivo, con el código del programa, ingresado como parámetro y crear el proceso, ubicándolo en la memoria separado en dos segmentos:

- El segmento de código: donde estará el programa completo byte a byte.
- El segmento de datos: ocupará todo el resto de la memoria.

Tabla de descriptores de segmentos

La tabla de descriptores de segmentos consta de 8 entradas de 32 bits y se inicializa en el momento de la carga del programa. Cada entrada de la tabla de segmentos 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 son para almacenar la cantidad de bytes del mismo (tamaño).

Base (2 bytes)	**Size** (2 bytes)

Por definición, la primera entrada (posición 0 de la tabla) guardará la información del segmento de código, mientras que la segunda (posición 1 de la tabla) guardará la información del segmento de datos.

Punteros

Cada segmento podría estar ubicado en cualquier parte de la memoria. Es por ello 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 direcciones 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.

Entonces, para acceder a la memoria se debe conocer el segmento y un desplazamiento (*offset*) dentro del mismo. Un puntero a memoria consta de 4 bytes: **2 bytes para el código de segmento** y **2 bytes para el *offset***. El código de segmento almacena la posición del mismo en la tabla de descriptores 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).

Registros

Si bien en esta primera parte la máquina virtual utilizará solo 11 registros, deberá tener la capacidad para almacenar 16, los cuales se codifican de la siguiente manera:

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	

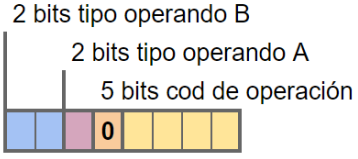
Antes de comenzar la ejecución, la máquina virtual debe inicializar los registros CS y DS con punteros al comienzo del segmento de código y del segmento de datos, respectivamente. Es decir, en los 16 bits más significativos deberán almacenarse las posiciones de la tabla de descriptores de segmentos, mientras que los 16 bits menos significativos se rellenan con 0. Por lo tanto, dado que por definición el descriptor del segmento de código se encuentra en la posición 0 de la tabla y el descriptor del segmento datos en la posición 1, 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. Luego de procesada la instrucción, pero antes de su ejecución, el registro IP deberá quedar apuntando al primer byte de la siguiente instrucción.

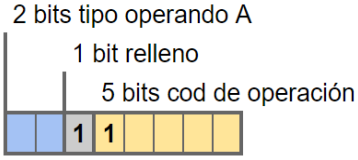
Instrucciones en lenguaje máquina

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

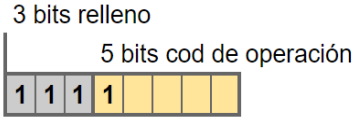
Instrucción con 2 operandos



Instrucción con 1 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 24 instrucciones, las cuales se listan a continuación junto con sus códigos de operación asociados en hexadecimal, clasificadas según la cantidad de operandos.

2 Operandos		1 Operando		0 Operandos	
Mnemónico	**Código**	**Mnemónico**	**Código**	**Mnemónico**	**Código**
MOV	00	SYS	10	STOP	1F
ADD	01	JMP	11		
SUB	02	JZ	12		
SWAP	03	JP	13		
MUL	04	JN	14		
DIV	05	JNZ	15		
CMP	06	JNP	16		
SHL	07	JNN	17		
SHR	08	LDL	18		
AND	09	LDH	19		
OR	0A	NOT	1A		
XOR	0B				
RND	0C				

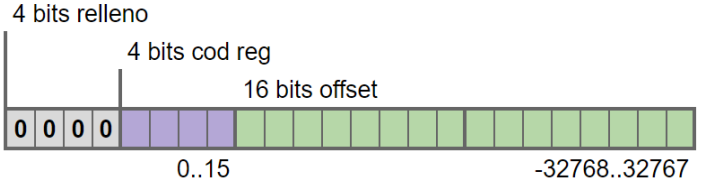
Operandos

Se admiten tres tipos de operandos:

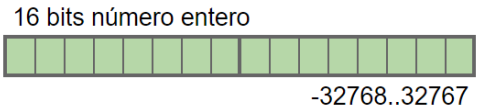
Código binario	Tipo	Tamaño
00	memoria	3 bytes (11)
01	inmediato	2 bytes (10)
10	registro	1 byte (01)
11	ninguno	0 bytes (00)

NOTA: el tamaño del operando puede obtenerse negando los dos bits correspondientes al código.

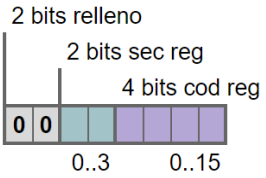
Operando de memoria: el dato es el contenido de una posición de memoria principal, relativa al comienzo de algún segmento (es decir una dirección lógica).



Operando inmediato: El dato es directamente el valor del operando.



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.

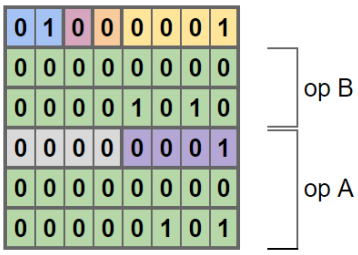


Los 2 bits para identificar el sector de registro se codifican de la siguiente manera:

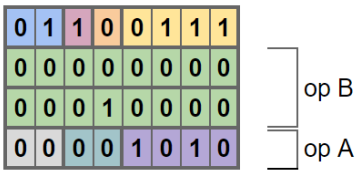
Código binario	Descripción	Ejemplo
00	registro de 4 bytes	EAX
01	4to byte del registro	AL
10	3er byte del registro	AH
11	registro de 2 bytes	AX

Ejemplos

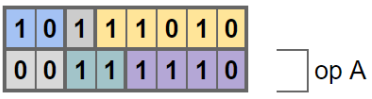
ADD [5], 10



SHL EAX, %10



NOT EX



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 AL, mientras que CL y CH contendrán la cantidad de celdas y su tamaño, respectivamente. En la pantalla se debe mostrar un prompt (*[XXXX]:*) que indique en hexadecimal (4 dígitos) la dirección física en la memoria principal de la celda en la que se encuentra cada dato.

Errores

La máquina virtual debe ser capaz de detectar 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 al calcular la dirección física de un dato dentro de un segmento, la misma apunta a un byte que se encuentra fuera de los límites del segmento.

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 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] 41 00 0A 01 00 05    |  ADD     [DS+5], 10
[0006] 67 00 10 0A          |  SHL        EAX, 16
[000A] BA 3E                |  NOT         EX
```

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.