TRABAJO PRÁCTICO

TRADUCTOR ASSEMBLER - MÁQUINA VIRTUAL

PARTE I

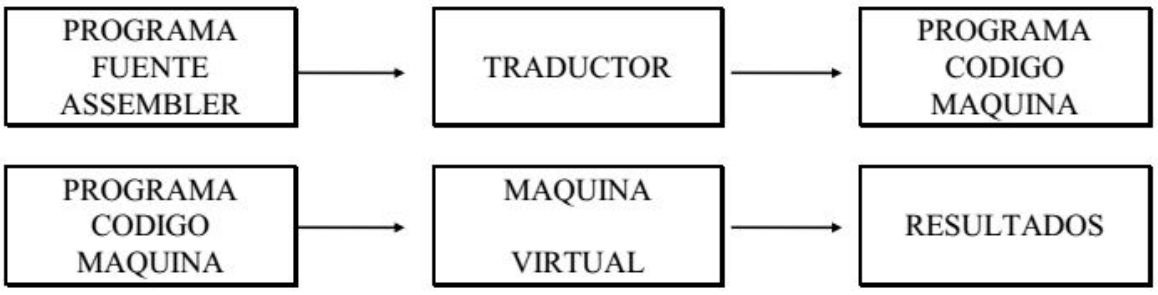
1 - INTRODUCCIÓN

El trabajo práctico consiste en realizar un programa en PASCAL, C/C++, Java, etc que lea un programa fuente escrito en un lenguaje assembler particular, lo traduzca al código máquina de una 'arquitectura virtual' y finalmente simule su ejecución.

PROCESOS:
Se pueden diferenciar dos procesos:

TRADUCCIÓN: donde se debe leer el código fuente assembler, traducirlo a código máquina, y almacenarlo en la memoria RAM y los registros de la 'máquina virtual', o en un archivo de imagen, para su posterior ejecución.

EJECUCIÓN: este proceso debe obtener las instrucciones a ejecutar desde la memoria (junto con sus operandos), interpretarlas y simular su funcionamiento.



PROGRAMA: La máquina virtual es un programa del cual se deben entregar los fuentes y el ejecutable compilado para Win32 o Linux. Se debe poder utilizar el programa ejecutable pasándole parámetros del siguiente modo:

```
mv.exe [-op] [codigo.asm] [maquina.img]
```

Donde "mv.exe" es el programa ejecutable (en caso de hacerlo en java mv.java) y el resto los argumentos, opcionales o no (dependiendo de la operación que se quiera realizar). Los archivos ".asm" son archivos de texto donde está escrito el código fuente que es traducido por la máquina virtual. Los archivos ".img" son archivos binarios donde se encuentran los valores de los registros y el estado de la memoria de la máquina virtual en un instante determinado.

Las operaciones se especifican en el argumento -op y pueden ser

- **-t** o **-T** (Traslate - Traducción): lee e interpreta el código assembler especificado en el archivo codigo.asm (puede ser cualquier nombre con extensión .asm) y almacena el

programa en código máquina en el archivo maquina.img (puede ser cualquier nombre con extensión .img)
- **-x** o **-X** (Execute - Ejecución): levanta la imágen del archivo maquina.img y ejecuta la máquina virtual
- **-a** o **-A** (All - Traducción y Ejecución): es la opción por omisión, si no se deja explícita otra opción. Lee e interpreta el código assembler especificado en el archivo codigo.asm y lo ejecuta.

Ejemplos (suponiendo que la máquina virtual está compilada en mv.exe):

Para traducir y ejecutar un programa escrito en lenguaje assembler (programa.asm).

```
> mv -a programa.asm
> mv programa.asm
```

Para traducir el programa escrito en assembler (programa1.asm) a código máquina y almacenarlo en un archivo de imagen (prg1.img).

```
> mv.exe -t programa1.asm prg1.img
```

Para ejecutar un programa compilado en una imagen (prg1.img)

```
> mv -x prg1.img
```

2 - DESCRIPCIÓN DE LA MÁQUINA VIRTUAL

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

- Memoria RAM de 2000 celdas de 4 bytes cada una.
- 9 registros de 4 bytes cada uno:
  - 6 registros de uso general: AX, BX, CX, DX, EX y FX.
  - 1 registro contador de programa: IP.
  - 1 registro de códigos de condición: CC.
  - 1 registro de segmento: DS.

En la memoria RAM de la máquina virtual se debe almacenar el programa traducido a código máquina. Las celdas que quedan libres se podrán usar para almacenar los datos durante la ejecución.
El registro de segmento DS (Data Segment) se utilizará para apuntar a la primer celda de la zona de datos.
Las direcciones efectivas de los datos se calcularán en base al valor del registro DS más un desplazamiento que se obtendrá de los operandos de las instrucciones.

Dirección del dato = Valor DS + Desplazamiento

El registro IP (instruction pointer) se usará para apuntar a la próxima instrucción a ejecutar dentro del segmento de programa.

Los registros de uso general servirán para almacenar datos y realizar cálculos durante la ejecución.

El registro CC (condition code) informará sobre el resultado de la última operación matemática o lógica ejecutada. De los 32 bits que tiene, solamente se usarán dos.
El bit menos significativo es el 'bit indicador de cero'. Valdrá 1 cuando la última operación matemática o lógica haya dado por resultado cero, y 0 en cualquier otro caso.
El bit más significativo es el 'bit indicador de signo'. Valdrá 1 cuando la última operación matemática o lógica haya dado por resultado un valor negativo, y 0 en otro caso.

La máquina virtual debe ser capaz de interpretar y procesar una serie de instrucciones que se explican más adelante en la sección EJECUCIÓN.

3 - TRADUCCIÓN

Cada línea del programa fuente assembler puede contener una sola instrucción. Traducidas todas las líneas del programa se deberá dar comienzo a la ejecución y/o almacenamiento en el archivo de imagen.

La traducción debe mostrar por pantalla el resultado del proceso, especificando la dirección de memoria donde se almacenó el código de máquina, el código de máquina mismo y la instrucción a la que pertenece.

El programa fuente puede contener líneas de comentarios, las cuales comienzan con un asterisco (*). Estas líneas no generan código ejecutable, pero igualmente se deberán mostrar por pantalla.

También se pueden agregar comentarios en la misma línea de la instrucción luego de un ";" (punto y coma).

INSTRUCCIONES DEL LENGUAJE ASSEMBLER

A continuación se lista el conjunto de instrucciones a implementar y el código de máquina asociado (en base hexadecimal).

Instrucción	COD	Instrucción	COD	Instrucción	COD	Instrucción	COD
				JMP	20		
MOV	01	AND	31	JE	21	SYS	81
ADD	02	OR	32	JG	22		
SUB	03	NOT	33	JL	23		
MUL	04	XOR	34	JZ	24		
DIV	05			JP	25		
MOD	06			JN	26		
CMP	13	SHL	37	JNZ	27		
SWAP	17	SHR	38	JNP	28		
RND	19			JNN	29	STOP	8F

En el apartado SEMÁNTICA DE INSTRUCCIONES se especifica cada una de las instrucciones.

Las instrucciones admiten tres categorías de 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: #10 o 10 (valor decimal 10), @17 (valor 15 decimal), %100 (valor 256 decimal)
‘A o ‘A’ (valor decimal 65)

Operando de registro: El dato es el contenido de alguno de los registros de la máquina virtual. Se especifican por el nombre del registro.
Ejemplo: AX o ax (case insensitive) representa el contenido del registro AX.

Operandos directo: el dato es el contenido de una posición de memoria RAM. El operando indica el desplazamiento dentro de la zona de datos respecto del valor de DS. Se expresa en base 10 y entre corchetes.
Ejemplo: Si el operando es [11] y el registro DS contiene el valor 500, el dato se ubicará en la celda 511 (decimal).

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.

LLAMADAS AL SISTEMA (System Calls o Interrupciones por software)

Un System Call es un mecanismo por el cual el programador solicita al sistema una acción, y por lo tanto se detiene el programa en ejecución resguardando los registros, para ejecutar un rutina de atención a un servicio generalmente asociado a los recursos del computador (teclado, monitor, archivo, etc). Cuando finaliza la rutina de atención, se restauran los registros y el programa retorna a su ejecución normal.

En esta máquina virtual los System Calls se implementan utilizando la instrucción **SYS e indicando el código de la operación**. El código de operación es un número entero positivo (constante decimal) y se ubica como primer (y único) operador de la instrucción. y pueden ser:

Código	Significado
1	READ
2	WRITE
3	DUMP

En el apartado de SEMÁNTICA DE LAS SYSTEM CALLS se especifica el comportamiento de cada uno.

INSTRUCCIONES DEL LENGUAJE MÁQUINA

El formato general de una instrucción de máquina es el siguiente:

16 bits	8 bits	4 bits	4 bits
Código de Operación	**Sin uso**	**Tipo Oper. 1**	**Tipo Oper. 2**

Los últimos ocho bits se utilizan para codificar los tipos de los operandos (4 bits para cada uno de los operandos).

Tipo Operando	Código binario
INMEDIATO	0000
DE REGISTRO	0001
DIRECTO	0010

Ejemplo: MUL AX,[32]
Esta instrucción almacena en el registro AX el resultado de la multiplicación del contenido del registro AX por el contenido de la celda 32 del segmento de datos. El código de operación de MUL es 04h, por lo tanto, la traducción a código máquina de esta instrucción es la siguiente:

```
Codigo de operacion         04h -> 00000000 00000100                    = 00 04 00 00 H
Tipo operando 1 (registro)  01b ->                              0001     = 00 00 00 10 H
Tipo operando 2 (directo)   10b ->                                  0010 = 00 00 00 02 H
                                  ----------------------------------   -------------
Instrucción de máquina    -------> 00000000 00000100 00000000 00010010 = 00 04 00 12 H
```

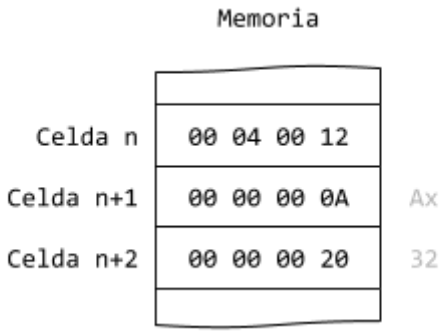
Es decir que 00040012h significa *"multiplicar un registro por el contenido de una celda de memoria y guardar el resultado en el mismo registro"*.

Para cada instrucción máquina, la información sobre los operandos se almacena en las dos celdas de memoria que siguen. Es decir que **cada instrucción de assembler ocupa tres celdas de memoria** RAM.
A los registros se les deben asignar códigos para poder referenciarlos:

Registro	Código Hexadecimal
DS	0000 0002
IP	0000 0008
CC	0000 0009
AX	0000 000A
BX	0000 000B
CX	0000 000C
DX	0000 000D
EX	0000 000E
FX	0000 000F

Por lo tanto, para la instrucción assembler dada, la traducción se deberá almacenar en tres celdas de memoria. La primer celda contendrá el código de instrucción y los tipos de operandos, y la segunda y tercera celda contendrán los valores de los operandos.

por ejemplo: MUL AX,[32]



La salida por pantalla que debe mostrar el módulo traductor debe respetar el siguiente formato:

[0000 0000] 0004 0012 0000 000A 0000 0020 1: MUL AX,[32]

El valor encerrado entre corchetes (hexadecimal) es la dirección de memoria donde se almacenó la instrucción, luego se ubican los valores hexadecimales de las tres celdas y por último la instrucción traducida con el número de línea (decimal) correspondiente del programa fuente. En este caso la instrucción sería la primera del programa. Las líneas de comentario no se deben tener en cuenta para la numeración.

Ejemplo de un programa completo

```
* Carga dos valores en EX y BX, los suma y muestra
MOV   EX, 1001
MOV   BX, 100
ADD   EX, BX     ;suma
MOV   [1], EX
MOV    AX,%1101
MOV    DX,1
MOV    CX,1
SYS    2
```

La traducción deberá mostrar por pantalla lo siguiente:

```
* Carga dos valores en AX y BX, los suma y muestra
[0000 0000]: 0001 0010 0000 000E 0000 03E9        1: MOV        EX, 1001
[0000 0002]: 0001 0010 0000 000B 0000 0064        2: MOV        BX, 100
[0000 0004]: 0002 0011 0000 000E 0000 000B        3: ADD        EX, BX     ;suma
[0000 0006]: 0001 0021 0000 0001 0000 000E        4: MOV       [1], EX
[0000 0008]: 0001 0010 0000 000A 0000 1101        5: MOV        AX, %1101
[0000 0010]: 0001 0010 0000 000D 0000 0001        6: MOV        DX, 1
[0000 0012]: 0001 0010 0000 000C 0000 0001        7: MOV        CX, 1
[0000 0014]: 0081 0000 0000 0002 0000 0000        8: SYS         2
```

RÓTULOS (labels)

Un rótulo es un "nombre" que puede asignarse a una instrucción, para luego poder hacer referencia a ella en una instrucción de salto.

Se define como una cadena de caracteres seguida del carácter **dos puntos**. Por ejemplo:

```
OTRO: ADD[1000],1
```

Hace que OTRO tenga como valor el número de línea de la instrucción ADD.
Las instrucciones de salto pueden hacer uso de los rótulos en sus operandos. Por ejemplo:

```
JMP OTRO
```

Indica realizar un salto incondicional a la instrucción ADD del ejemplo.

Los rótulos permiten independizar los saltos de los números de línea, con lo cual, si se agregan o quitan líneas al programa, no será necesario recalcular dichos saltos.

ERRORES

En la traducción se deben detectar, como mínimo, los siguientes errores:

- Error de sintaxis es el de una instrucción inexistente, en cuyo caso mostrará un mensaje indicandolo. La línea no será traducida, las tres celdas que la forman quedarán en FFFF FFFF FFFF FFFF.
- No se encuentra el rótulo (o label) cuando se hace referencia a un rótulo y el mismo no se encuentra en ninguna línea. Deberá mostrar FFFF FFFF en lugar del argumento del rótulo.

Ante alguno de estos errores la traducción deberá continuar, pero el programa no será ejecutado.

4 - EJECUCIÓN

Una vez traducido el programa fuente assembler, el código máquina correspondiente estará alojado en la memoria RAM de la Máquina Vitual listo para ejecutarse.
Para procesar dicho código, se irá obteniendo cada instrucción máquina (y su operandos), desde la memoria haciendo uso del registro IP (puntero a instrucción), se interpretará y se realizará la acción que corresponda.

La estructura general del módulo de ejecución es la siguiente:

```
IP = 0
WHILE (0<=IP AND IP<DS) DO
    BEGIN
        <OBTENER PROXIMA INSTRUCCION>
        <OBTENER OPERANDOS>
        <INTERPRETAR INSTRUCCION>
        <EJECUTAR INSTRUCCION>
    END
```

SEMÁNTICA DE LAS INSTRUCCIONES

A continuación se detalla el significado de las instrucciones del lenguaje assembler. Cuando se hace referencia a una dirección de memoria, debe interpretarse como una dirección relativa, cuyo valor deberá sumarse al contenido del registro de segmento de datos (DS) para calcular la dirección efectiva.

MOV: Permite asignar 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 AX,10   ; Carga en AX el valor 10 decimal
MOV BX,CX ; Carga en BX el valor del registro CX
MOV DX,[10]; Carga en DX el valor almacenado en la celda de memoria 10.
```

ADD, SUB, MUL, DIV: Efectúan 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.

```
ADD AX,2      ; Incrementa AX en 2
MUL AX,BX  ; Multiplica AX por BX dejando el resultado en AX
SUB [10],BX  ; Resta BX del valor de la celda 10, y el resultado queda en la celda 10
```

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

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

AND, OR, XOR, NOT: Efectúan las operaciones lógicas básicas bit a bit entre los operandos y afectan al registro CC. El resultado se almacena en el primer operando. La instrucción NOT requiere un solo operando.

```
AND AX,BX  ; efectúa el AND entre AX y BX, el resultado queda en AX
NOT [15]        ; invierte cada bit del contenido de la posición de memoria 15.
```

SHL, SHR: shift left y shift right permite efectuar el desplazamiento (a izquierda y derecha) de los bits almacenados en un registro o una posición de memoria. Los bits que quedan libres se completan con ceros. También afectan al registro CC.

```
SHL AX,1; corre los 16 bits de AX una posición a la izquierda (es equivalente a multiplicar AX por 2).
SHR [200],BX; corre a la derecha los bits de la celda 200, la cantidad de veces indicada en BX.
```

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

JMP: efectúa un saldo incondicional al número de línea o rótulo del programa fuente indicado en el operando.

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

JE, JG, JL: comparan el primer operando con el contenido del registro AX, y en función del resultado efectúan un salto al número de línea o rótulo indicado por el segundo operando.

```
JG BX,5      ; si BX es mayor que AX se salta a la línea 5
JE [1000],10 ; si el valor de la celda 1000 es igual a AX se salta a la línea 10
JL CX,BX     ; si CX es menor que AX se salta a la línea indicada por BX.
```

JZ, JP, JN, JNZ, JNP, JNN: permiten efectuar saltos condicionales en función de los bits del registro CC. Requieren de un solo operando que indica el número de línea donde se debe bifurcar.

```
JP 2         ; se salta a la línea 2 si el bit de signo de CC es cero.
JN BX        ; se salta a la línea indicada en BX si el bit de signo está en 1.
JZ [8]       ; se salta a la línea indicada en la celda 8 si el bit de cero de CC es 1.
JNZ 10       ; se salta a la línea 10 si el bit de cero está en cero.
```

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

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

MOD: Carga en el primer operando el resto de dividir el primer operando por el segundo. En este caso modifica el CC acorde al resultado de la división. Ejemplo -5 mod 2 = 1 pero CC quedará configurado con negativo

SEMÁNTICA DE LOS SYSTEM CALLS

A continuación se detalla el significado de cada interrupción, la codificación necesaria en los registros y cómo debe responder.

SYS 1 (READ): Permite almacenar en un rango de celdas de memoria los datos leídos desde el teclado. Almacenando lo que se lee por teclado en la posición de memoria apuntada por DX hasta CX celdas de memoria como máximo (si se ingresan más caracteres que los especificados en CX, se finaliza la lectura).
El modo de lectura depende de la configuración de cada bit del registro AX

Valor	Bit	Significado
%8000	15	
%4000	14	
%2000	13	
%1000	12	1: Escribe un prompt [0000 0000]: con la dirección de memoria.
%0800	11	
%0400	10	

%0200	9	
%0100	8	1: Interpreta el contenido luego del endline según la especificación de los 4 bits menos significativos. 0: lee caracter a caracter, y guarda uno por celda.
%0080	7	
%0040	6	
%0020	5	
%0010	4	
%0008	3	1: interpreta hexadecimal
%0004	2	1: interpreta octal
%0002	1	
%0001	0	1: interpreta decimal

Ejmeplo1:

Código	Pantalla

`MOV AX, %1101`
`MOV DX, 11`
`MOV CX, 2`
`SYS 1`
	[0011]: 23 <Enter>
	[0012]: 25 <Enter>

Al finalizar la lectura, las celdas 11 y 12 quedarán con los valores 23 y 25 respectivamente.

Ejmeplo2:

Código	Pantalla

`MOV AX, %1000`
`MOV DX, 11`
`MOV CX, 50`
`SYS 1`
	[0011]: Hola <Enter>

Las celdas quedarán de la siguiente manera: 11 = ‘H’, 12 = ‘o’, 13 = ‘l’, 14 = ‘a’, 15 = %00

SYS 2 (WRITE): Muestra en pantalla el contenido del rango de celdas especificado. Comenzando por la celda apuntada por DX y muestra una cantidad de celdas CX

Al igual que la lectura, la forma de escritura se configura en AX

Valor	Bit	Significado
%8000	15	
%4000	14	
%2000	13	
%1000	12	1: Escribe un prompt [0000 0000]: con la dirección de memoria.
%0800	11	
%0400	10	
%0200	9	
%0100	8	1: Agrega endline después de imprimir cada celda
%0080	7	
%0040	6	
%0020	5	

%0010	4	1: imprime solo el byte menos significativo como caracter.
%0008	3	1: imprime la celda de memoria en hexadecimal
%0004	2	1: imprime la celda de memoria en octal
%0002	1	1: imprime la celda de memoria como binario (core image)
%0001	0	1: imprime la celda de memoria en decimal

Cuando el caracter ASCII no es imprimible escribe un punto '.' en su lugar.

Ejemplo 1:

Código	Pantalla

`MOV [1],´H´`
`MOV [2],´o´`
`MOV [3],´l´`
`MOV [4],´a´`
`MOV [5],%00`
`MOV DX,1`
`MOV CX,4`
`MOV AX, %0010`
`SYS 2`
	Hola

Ejemplo 2:

Código	Pantalla

`< mismo código de ejemplo 1 >`
`OR AX, %1000`
`OR AX, %0100`
`OR AX, %0010`
`SYS 2`
	[0000 0001]: H
	[0000 0002]: o
	[0000 0003]: l
	[0000 0004]: a

Ejemplo 3:

Código	En archivo

`MOV [10], %41`
`SHL [10], 8`
`OR [10], %61`
`MOV DX, 10`
`MOV CX, 1`
`MOV AX, %010F`
`SYS 2`
	[0001]: %4161 @40541 Aa #16737

SYS 3 (DUMP): Es igual al SYS 2 (WRITE) y además muestra el contenido de todos los registros de la máquina con el formato:

[DS]: 0000
[IP]: 0000
[CC]: 0000
etc…

Según el formato especificado en los últimos 4 bits de AX (idem SYS 2)

ESPECIFICACIÓN DE LOS ARCHIVOS DE IMAGEN (*.IMG)

A continuación se detalla cómo debe almacenarse y recuperarse los archivos *.img.

Los archivos son binarios puros que se leen en celdas de 4 bytes.
En las primeras 16 celdas se guardarán los registros usando el código como posición. Por ejemplo el IP se encontrará en la novena celda (comienza a contar desde 0). En las siguientes 2000 celdas se almacena el contenido de la memoria.