Programación en ensamblador[1]

Como cualquier otro lenguaje de programación, el ensamblador tiene varias versiones, pero entre ellas se conservan un conjunto de reglas para su correcta escritura, que podríamos decirle *gramática*. Para este curso nos basaremos en el ensamblador de 16 bits y lo emularemos con emu8086 (Ashkelon, 2014), pues así encapsularemos cualquier error que pudiera llevar al mal funcionamiento de la PC.

Los archivos de código no son más que archivos de texto plano con extensión *.asm en los cuales se escriben de forma ordenada las instrucciones en secciones determinadas: código, pila, datos. El código se puede comentar utilizando el signo ***";"***, pues el compilador ignorará todo lo que esté escrito en el renglón desde que se encuentra el ";" (Irvine, 2007, página 57).

Para la programación en ensamblador es muy importante reconocer la clasificación de las instrucciones relativa a las acciones que ejecutan. Así es que tenemos un conjunto de instrucciones para el movimiento de datos en la memoria, un segundo conjunto de funciones se encarga de todas las operaciones aritméticas y de lógica; finalmente, un tercer grupo se encarga de las funciones de control del proceso, es decir, sus bifurcaciones y decisiones.

El proceso de creación de un programa ejecutable a partir de un código fuente en ensamblador se da en un proceso de 2 pasos: primero se compila el código fuente y se obtiene un archivo con extensión *.obj al que le llamamos código objeto. Este código objeto debe ser enlazado en un segundo paso para crear el archivo ejecutable (con extensión *.exe).

Además, cuando requerimos realizar una operación matemática como sumar dos cantidades, restarlas, multiplicarlas e, incluso, dividirlas hacemos uso de las funciones aritméticas. No olvides que primero utilizaremos sólo números enteros positivos en los ejemplos, por lo que cuando hablamos de la división nos referimos a la división por módulo.

En otras ocasiones requerimos realizar operaciones lógicas (AND, OR, NOT). Estas operaciones siempre actúan a nivel bit, por lo que existe un conjunto de instrucciones que hacen comparaciones a nivel valor, pero éstas se encuentran más relacionadas al conjunto de operaciones de control, pues realizan un salto entre código en función del resultado de la comparación. Para mayor referencia puedes dar una hojeada a los capítulos 4 y 6 (Irvine, 2007).

[1] Documento elaborado por el Dr. Carlos Jiménez Gallegos.

El tercer grupo de instrucciones se encarga de conducir la ejecución del programa otorgándole la capacidad de tomar decisiones numéricas, y ceder el control del programa al exterior – como cuando se ejecutan interrupciones de hardware.

Formato de un programa en ensamblador

El modelo de programación para los microprocesadores Intel está basado en los siguientes registros de memoria (reproducción tomada de B. Brey, 2006, página 50):

FIGURA 2-1
El modelo de programación de los microprocesadores Intel, del 8086 hasta el Pentium 4.

Lo primero que debemos notar es que los registros pueden accederse por partes o grupos de 8, 16 y 32 bits. Por ejemplo, EAX es el registro acumulador de 32 bits, AX representa las 16 bits más bajos de EAX. AH y AL representan la parte alta de 8 bits y la parte baja (últimos 8 bits) del mismo registro AX. Es por ello que debemos diferenciar cuando hacemos una operación de 8 bits, de 16 o de 32. Aquí una breve descripción de cada registro:

Registro	Nombre	Uso
EAX, AX, AH, AL	Acumulador	Es un registro multipropósito que además se utiliza en operaciones como multiplicación, división y ajuste.
EBX, BX,BH,BL	Índice Base	Puede direccionar datos en la memoria o la dirección de desplazamiento de una posición de memoria. Es multipropósito.
ECX, CX, CH, CL	Contador	Aunque es un registro multipropósito se le llama contador porque en algunas instrucciones lo utilizan para contar repeticiones o ciclos.
EDX, DX, DH, DL	Datos	Guarda el resultado de la multiplicación y de la división, pero puede ser usado con otros propósitos.
SP	Apuntador de Pila	Apunta al último valor que se encuentra en la pila.
BP	Apuntador de Base	Apunta a una dirección de memoria.
DI	Índice Destino	Usado para las instrucciones de cadenas o arreglos.
SI	Índice Origen	Usado para apuntar a datos de cadena de origen para instrucciones de cadena.
IP	Apuntador de instrucciones	Apunta a la siguiente instrucción del programa.
FLAGS	Banderas	El registro de banderas nos informa el estado del microprocesador y de la última instrucción ejecutada.
CS	Código	Sección de memoria en donde está el código.
DS	Datos	Sección de memoria en donde están los datos.
ES	Extra	Segmento que utilizan algunas funciones de cadena para guardar datos.
SS	Pila	Segmento de memoria donde se localiza la pila.

Para el caso del registro de banderas FLAGS, en ellas cada bit representa una señal (bandera) especial. Aquí describimos las más importantes:

Bit	Bandera	Nombre	Descripción
0	C	Acarreo	Indica si hubo acarreo en la última operación. También puede indicar errores.
2	P	Paridad	Indica la paridad del último resultado: 0 para impar, 1 para par.
4	A	Acarreo Auxiliar	Guarda el medio acarreo, es decir, si hubo acarreo entre los bits 3 y 4, utilizado en operaciones en BCD.
6	Z	Cero	Indica si el último resultado fue cero.
7	S	Signo	Índica si el último bit del resultado es 1.
8	T	Trampa	Indica que el programa se detendrá en la condición de depuración.
9	I	Interrupción	Habilita las interrupciones.
10	D	Dirección	Indica la dirección (incremento o decremento).
11	O	Sobreflujo	Indica que ha habido un desbordamiento en una operación con signo.

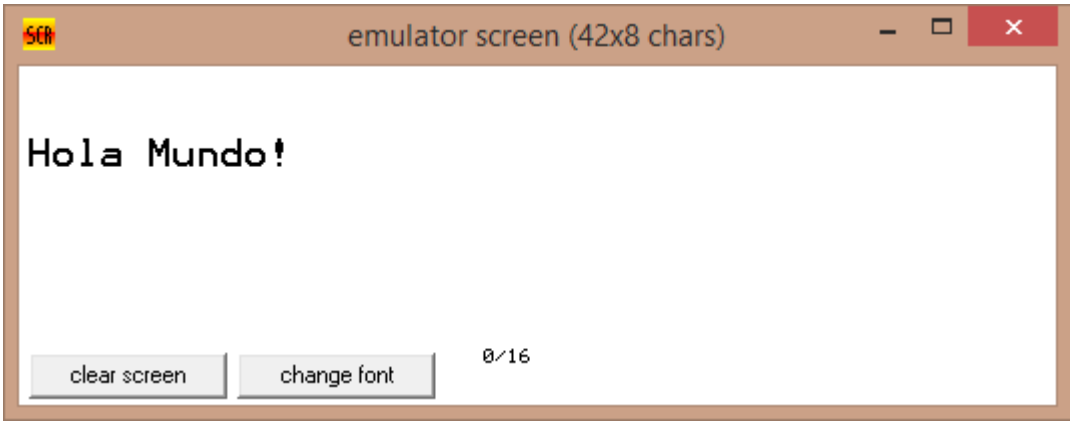
Siguiendo la tradición de todos los lenguajes de programación comenzaremos con el clásico ejemplo “Hola Mundo”. A continuación se encuentra el listado de Hola.asm.

```
.MODEL SMALL                ;Modelo pequeño

.STACK 100h                 ;Segmento de pila 256 posiciones

CR EQU 13                   ;Retorno de carro
LF EQU 10                   ;nueva línea

.DATA                       ;Segmento de datos

TEXTO DB LF,LF,CR,'Hola Mundo!$';Texto

.CODE                       ;Código del programa
```

```
MOV AX, @DATA              ;Cargar la dirección del segmento de datos

MOV DS,AX                  ;Cargar la dirección en el segmento de datos
LEA DX,TEXTO               ;Carga el texto

MOV AH,9                   ;imprimir en pantalla
INT 21h                    ;Llama al sistema operativo

MOV AH,4ch                 ;Función 4ch
INT 21h

END                        ;Fin del programa
```

Programa 1

Al emularlo se obtiene en la ventana screen (que es la emulación de pantalla):



Analicemos el programa anterior parte por parte. La directiva .**MODEL** le dice al compilador el tamaño del modelo de programa, en nuestro caso utilizaremos small aunque, incluso, este modelo pequeño es grande para lo que programaremos. La directiva .STACK representa el tamaño de pila que queremos utilizar, en este caso es del 100h posiciones, es decir, 256 bytes.

Enseguida aparecen dos comandos EQU, que son definiciones de constantes, éstas no son parte del sistema de memoria pues al momento de compilar la etiqueta es sustituida por el valor.

En este programa la etiqueta CR es sustituida por en decimal 13 y LF sustituida por el decimal 10. EQU nos sirve para utilizar etiquetas a manera de constantes que nos sirvan para indicar lo que representan, más que el valor numérico. Así, el valor ASCII 10 es el cambio de línea por lo que es más útil que aparezca como LF que como el decimal 10. Del mismo modo, el ASCII 13 es el retorno de carro y es mejor verlo en el código como CR que como el decimal 13.

Con la directiva .DATA señalamos el comienzo del segmento de datos. En esta sección se declaran los datos como cadenas de texto y arreglos numéricos con los que se quiere contar. En el programa sólo se define una cadena con nombre TEXTO que contiene la cadena Hola Mundo!, precedida de un retorno de carro y dos cambios de línea. El parámetro DB (data byte) indica que cada localidad de la cadena es de un byte, lo justo para representar un caracter del código ASCII.

A partir de la directiva .CODE se le indica al compilador que comienzan las instrucciones del programa. Es decir, que en este segmento comenzarán las instrucciones.

Tipos de instrucciones

Movimiento de datos

Ensamblador tienen una serie de funciones para mover datos. Por ahora vamos a hablar de las más importantes o más usuales: MOV, PUSH, POP, IN, OUT y LEA.

La instrucción MOV mueve datos de una localidad a otra, por ejemplo: MOV CL, CH copia el contenido de CH en CL. Si tenemos una variable en memoria MOV VAR, DL, copia el contenido de DL en la variable de nombre VAR. También se puede hacer lo contrario, es decir, pasar información de una variable a un registro: MOV DH, VAR. Incluso, es posible hacer asignaciones directas de los valores, estas asignaciones se llaman inmediatas, por ejemplo: MOV AH, 0FFh pone en AX el valor de 0FFh (hexadecimal).

En las asignaciones inmediatas, si el valor que se quiere asignar comienza con letra (como en el caso de FFh) se debe agregar un cero antes de la primera cifra (0FFh). El comando MOV trabaja para 8, 16 y 32 bits. El origen y el destino deben ser del mismo tamaño. Es importante siempre añadir la inicialización del segmento de datos, lo que se hace es colocar la dirección de DATA en DS.

```
.MODEL SMALL          ;Modelo pequeño

.STACK 100h           ;Segmento de pila 256 posiciones


;*** Algunas constantes ************

Quince EQU 15
Cero   EQU 0

.DATA                 ;Segmento de datos

; *** Algunas variables ************

Arreglo1 DB 10,20,30,40  ;Texto
Cadena DB 'UTEL$'
VAR DB 7

.CODE                 ;Código del programa


; **********************************
; Operaciones de movimiento de datos
; **********************************

; *** Siempre iniciar el segmento de datos
MOV AX, @DATA           ; Carga apuntador del
MOV DS, AX              ; segmento de datos en AX
; ****************************************

; Movimiento inmediato
 MOV AH,0FFH                    ; AH=FFh
```

```
; Movimiento de memoria a registro
 MOV AL, Quince              ;AL=0Fh

; movimiento de registro a registro 8 bits
 MOV BH, AL                  ; BH=0Fh

; movimiento de registro a registro 16 bits
 MOV BX, AX                  ; BX=FF0Fh

; movimiento de una variable a un registro
 MOV AH, VAR                 ; AH=7h

; movimiento de un registro a una variable
 MOV VAR, BH                 ; VAR=FFh

; movimiento de un arreglo a registro
; de la posición 2 del arreglo con valor
; decimal 30
 MOV AH, ARREGLO1[2]       ; AH=1Eh (30 decimal)

; Movimiento de registro a arreglo
; en la tercer posición
 MOV ARREGLO1[3], BH       ; vea la modificación del arreglo

MOV AH,4ch                 ;Función 4ch (finalizar)
INT 21h

END
```

Programa 2

PUSH y POP son funciones complementarias utilizadas para meter (PUSH) o sacar (POP) valores de la pila, sólo pueden usarse registros y variables de 16 y 32 bits, y sólo admite 8 bits si es inmediato. Su uso se ilustra en el siguiente programa. Recuerde que en una pila el primer valor en entrar, es el último en salir.

```
.MODEL SMALL          ;Modelo pequeño

.STACK 100h           ;Segmento de pila 256 posiciones


;*** Algunas constantes ************

Quince EQU 15
Cero   EQU 0
.DATA                 ;Segmento de datos

; *** Algunas variables ************


Cadena DB 'UTEL$'


.CODE                 ;Código del programa


; ********************************
; Operaciones de pila
;

; *** Siempre iniciar el segmento de datos
MOV AX, @DATA   ; Carga apuntador del
MOV DS, AX  ; segmento de datos en AX
```

```
; Uso de PUSH

 PUSH Quince    ; mete un 0Fh (15) inmediato
 MOV AX, 00FFh
 PUSH AX        ; mete un 00FFh
 PUSH Cero      ; mete un cero

; Uso de POP

POP AX                  ; AX=0000h
POP BX                  ; BX=00FFh
POP CX          ; CH=000Fh




MOV AH,4ch              ;Función 4ch
INT 21h

END
```

Programa 3

LEA es una instrucción que carga la dirección efectiva de un registro. Es necesaria cuando queremos pasar cadenas de caracteres a la pantalla, tal es el caso del Programa 1 en donde aparece LEA DX, TEXTO para cargar la dirección de la cadena “Hola mundo!”.

Instrucciones lógicas y aritméticas

Las instrucciones aritméticas que revisaremos serán la suma (ADD), la resta (SUB), multiplicación (MUL), división (DIV), comparación (CMP), Incremento (INC) y decremento (DEC). Las instrucciones lógicas comúnmente utilizadas son: or (OR), and (AND), negación (NOT) y or exclusivo (XOR).

La suma usa el prototipo ADD var1, var2 cuyo significado es var1=var1+var2, donde *var* es una variable, registro o memoria que pueden ser (del mismo tamaño) de 8, 16 y 32 bits. La resta actúa de manera similar: SUB AX, BX significa AX=AX-BX.

En cuanto al decremento y el incremento, son instrucciones que actúan sobre un solo registro o variable incrementando o decrementando en una unidad. INC AX significa que AX=AX+1 mientras que DEC BH significa BH=BH-1.

El caso de la división y la multiplicación son un poco diferentes. En la multiplicación de 8 bits, uno de los valores a multiplicar siempre debe de estar en AL. El resultado se coloca siempre en AX. Entonces, el comando MUL CL realiza la asignación AX= AL*CL. Para la operación de 16 bits uno de los valores debe de estar en AX, así, por ejemplo, MUL CX multiplica AX*CX y el resultado lo distribuye en los registros DX y AX (pues el resultado puede tener más bits).

La división se realiza de manera análoga, para 8 bits DIV CL significa que se divide AX entre CL, AH tendrá el cociente y AL el residuo. Para 16 bits DIV CX realiza la división de DX-AX entre CX, en AX estará el cociente y en DX el residuo. Es muy importante tener cuidado cuando hacemos multiplicaciones y divisiones pues la operación requiere localidades fijas para realizarse y debemos ubicar ahí datos que usaremos más adelante.

Analicemos el siguiente programa para revisar algunas de las operaciones aritméticas mencionadas:

```
.MODEL SMALL          ;Modelo pequeño

.STACK 100h           ;Segmento de pila 256 posiciones

;*** Algunas constantes ************

Quince EQU 15
Cero   EQU 0
.DATA             ;Segmento de datos

; *** Algunas variables ************


Cadena DB 'UTEL$'


.CODE             ;Código del programa
```

```
; *********************************
; Operaciones aritméticas
;

; *** Siempre iniciar el segmento de datos
MOV AX, @DATA      ; Carga apuntador del
MOV DS, AX      ; segmento de datos en AX

; *** suma 8 bits **
MOV AH,5              ; AH=5
ADD AH,2              ; AH=AH+2=7
MOV BL, Quince     ; BX=15 (0Fh)
ADD AH,BL             ; AH=AH+BL=22 (16h)

; *** suma 16 bits **
MOV AX,10h      ; AX=0010h
ADD AX,0F0h               ; AX=AX+F0h=0100h
MOV BX, Quince     ; BX=15 (0Fh)
ADD AX,BX             ; AX=AH+BL=010Fh

; *** resta 8 bits **
MOV AH,100      ; AH=100 (64h)
SUB AH,55         ; AH=AH-55=45 (2Dh)
MOV BL, Quince     ; BX=15 (0Fh)
SUB AH,BL             ; AH=AH-BL=30 (1Eh)

; *** resta 16 bits **
MOV AX,200h    ; AX=0200h
SUB AX,0F0h       ; AH=AH-F0h=0110h
MOV BX, Quince     ; BX=15 (0Fh)
SUB AX,BX             ; AH=AH-BL=0101h
```

```
; *** Multiplicación de 8 bits **
MOV AL, Quince     ; AL= 0Fh (15)
MOV CL, 3          ; CL= 3
MUL CL                    ; AX= 2Dh (45)

; *** Multiplicación de 16 bits **
MOV AX, Quince     ; AX= 0Fh (15)
MOV CX,1000        ; CX= 3E8h (1000)
MUL CX                    ; [DX AX]=0000 3A98h (15000)

MOV AH,4ch              ;Función 4ch
INT 21h

END
```

Programa 4

En cuanto a las operaciones lógicas, siempre actúan a nivel bit entre registros de 8 y 16 bits, y en caso de la instrucción NOT se aplica a un solo registro. Es fácil verificar su funcionamiento con un programa:

```
.MODEL SMALL          ;Modelo pequeño

.STACK 100h           ;Segmento de pila 256 posiciones

;*** Algunas constantes ************
Quince EQU 15
Cero   EQU 0

.DATA                 ;Segmento de datos

; *** Algunas variables ************

Cadena DB 'UTEL$'
```

```
.CODE                ;Código del programa


; **********************************
; Operaciones lógicas
;

; *** Siempre iniciar el segmento de datos
MOV AX, @DATA          ; Carga apuntador del
MOV DS, AX          ; segmento de datos en AX

MOV AL, Quince         ; AL=0Fh
OR AL,0FFh             ; AL=0FFh
AND AL,0F0h            ; AL=F0h
XOR AL,AL              ; AL=00h
NOT AL                          ; AL=0FFh

MOV AH,4ch             ;Función 4ch
INT 21h

END
```

Programa 5

Instrucciones para control de procesos

Las principales instrucciones para el control del proceso, o flujo de programa, son: el salto incondicional (JMP), los saltos condicionales (JXX), ciclo (LOOP) y las auxiliares CMP (compara) y TST (probar).

Para los saltos condicionales se reproduce la tabla de ellos del capítulo 6 (B. Brey, 2006, página 189):

TABLA 6-1 Instrucciones de salto condicional.

Lenguaje ensamblador	*Condición a evaluar*	*Operación*
JA	Z = 0 y C = 0	Salta si está por encima.
JAE	C = 0	Salta si está por encima o si es igual.
JB	C = 1	Salta si está por debajo.
JBE	Z = 1 o C = 1	Salta si está por debajo o si es igual.
JC	C = 1	Salta si hay acarreo.
JE o JZ	Z = 1	Salta si es igual o salta si es cero.
JG	Z = 0 y S = 0	Salta si es mayor que.
JGE	S = 0	Salta si es mayor o igual que.
JL	S != O	Salta si es menor que.
JLE	Z = 1 o S != O	Salta si es menor o igual que.
JNC	C = 0	Salta si no hay acarreo.
JNE o JNZ	Z = 0	Salta si no es igual o salta si no es cero.
JNO	O = 0	Salta si no hay desbordamiento.
JNS	S = 0	Salta si no hay signo (positivo).
JNP o JPO	P = 0	Salta si no hay paridad o si la paridad es impar.
JO	O = 1	Salta si hay desbordamiento.
JP o JPE	P = 1	Salta si hay paridad o si la paridad es par.
JS	S = 1	Salta si hay signo (negativo).
JCXZ	CX = 0	Salta si CX es igual a cero.
JECXZ	ECX = 0	Salta si ECX es igual a cero.

En el siguiente programa se muestra el funcionamiento de algunas de estas instrucciones.

```
.MODEL SMALL          ;Modelo pequeño

.STACK 100h           ;Segmento de pila 256 posiciones

;*** Algunas constantes ************
Quince EQU 15
Cero   EQU 0

.DATA                 ;Segmento de datos
```

```
; *** Algunas variables ************

Cadena DB 'UTEL$'


.CODE               ;Código del programa


; *********************************
; Operaciones de control de programa
;

; *** Siempre iniciar el segmento de datos
          MOV AX, @DATA      ; Carga apuntador del
          MOV DS, AX         ; segmento de datos en AX


inicio:     ; etiqueta

; *** JMP salto incondicional


          MOV AH, Quince ; AH=15;

          JMP parte2     ; Salto a parte2

          MOV AL, Cero   ; AL= 0  Nunca se ejecutará

parte2:             ; salta aquí


; *** LOOP ciclo de repetición
```

```
        MOV CX, 10     ; en CX se establece la cantidad
                ; de veces que se repetira el bloque
lazo:                  ; inicia el bloque de instrucciones
        PUSH CX        ; Estas instrucciones
      MOV AX, CX  ; se repiten
        LOOP lazo      ; aquí salta al inicio del bloque

; En la pila y AX se puede verificar como CX se va decrementando
; cada vez que termina un ciclo

; *** Saltos condicionales

        MOV AL,10      ; AL=10
        MOV BL,5       ; BL=5

      CMP AL,BL   ; compara AL con BL
      JA  es mayor ; salta si AL>BL
        MOV CL,0       ; no se ejecuta

es mayor:

        MOV AH,5           ;AH=10
      CMP AH,AL        ; compara AH, con AL
      JE  son iguales   ; salta si AH=AL
      MOV CL,0         ; esto no se ejecuta

son iguales:

      JMP inicio       ; este programa nunca ¡termina!
                       ; en algún punto se acabará el espacio
                ; en la pila
```

```
MOV AH,4ch          ;Función 4ch
INT 21h

END
```

Programa 6