LABORATORIO DE
PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR x86-16bits

De la creación del programa al proceso en ejecución

Objetivo

El objetivo de esta práctica es conocer y diferenciar los conceptos de programa y proceso. Aprenderemos a desarrollar un programa en ensamblador y qué secciones lo componen.

Programa

Un programa no es un conjunto de instrucciones solamente. Un programa es un espacio de memoria organizado en secciones bien diferenciadas entre las que no pueden faltar la sección de código, la sección de datos y la sección de pila. El programa compilado se guarda en un fichero que tiene dos partes: el binario y una cabecera con información para el sistema operativo.

Las secciones de un programa

Un programa requiere de varias secciones de memoria. Evidentemente, una de ellas es la que contiene las instrucciones a ejecutar. Sin embargo, también necesitamos una sección que almacene los datos y una sección para la pila. Las secciones más habituales son:

Sección de código

La sección de código, sección de texto o simplemente texto, es la que contiene la secuencia de instrucciones a ejecutar. Normalmente es una sección de sólo lectura aunque algunos programas se automodifican o reservan pequeños espacios para datos en esta sección. Esta sección requiere de un puntero especial (puntero de instrucción o contador de programa) que señala la posición de la siguiente instrucción a ejecutar.

Sección de datos

La sección de datos contiene las variables globales (aquellas accesibles desde todo el programa) y las estáticas (su tiempo de vida abarca todo el tiempo de ejecución del programa) que contienen datos iniciales (inicializadas por el programador). Es una sección de lectura y escritura aunque si una variable se declara como constante se puede alojar en una zona de solo lectura.

Sección de bss

La sección bss (*Block Started by Symbol*) contiene variables estáticas no inicializadas, es decir, cuyo valor inicial es 0. Normalmente esta sección no se almacena en el fichero imagen del ejecutable sino que es el cargador del sistema operativo quien realiza la reserva de espacio en memoria principal y el relleno con 0.

Sección de *heap*

La sección *heap* se usa para hacer reservas dinámicas de memoria en tiempo de ejecución. La reserva de un bloque de memoria puede liberarse o incrementarse en tiempo de ejecución. Es una sección que puede crecer y, por tanto, no debe estar limitada por otras secciones. No aparece en el fichero imagen del ejecutable.

Sección de pila

La sección pila implementa un área de memoria con un tipo de acceso LIFO (*Last Input First Output*) utilizada en el manejo de procedimientos (subrutinas) que sirve para almacenar temporalmente los argumentos y variables locales. Esta sección requiere de un puntero especial (puntero de pila) que indica la posición de memoria de la cima de la pila. Adicionalmente, se suele utilizar otro puntero especial (puntero de marco de pila) que sirve para referenciar los argumentos y variables locales propios del procedimiento (subrutina) en curso.

Es habitual que la sección de pila y la sección de *heap* crezcan la una contra la otra de manera que, si el programa no ha sido bien diseñado, una sección puede llegar a sobrescribir los datos de la otra.

La cabecera de un programa

La cabecera almacena información para el cargador del sistema operativo. Algunos de los datos que proporciona esta cabecera son:

- tamaño de la cabecera
- tamaño del fichero binario
- tabla de direcciones de memoria absolutas
- máxima y mínima cantidades de memoria requeridas
- valores iniciales de los punteros de instrucción y de pila

¿Qué es la tabla de direcciones de memoria absolutas?

La imagen de un ejecutable se implementa en forma de fichero reubicable, es decir, que ha de funcionar igual sea cual sea el rango de posiciones de memoria principal que se le asignen. Todas aquellas referencias a posiciones de memoria relativas a la posición en curso no tienen problema de ambigüedad. En cambio, todas aquellas referencias absolutas son desconocidas *a priori*. Por ejemplo, el comienzo de la sección de datos puede ubicarse en cualquier posición de memoria principal y no se conocerá con precisión hasta que el sistema operativo no asigne un mapa de memoria al proceso.

Para solucionar este problema, lo que hacemos es referenciar las posiciones absolutas respecto al comienzo de la imagen del ejecutable y confeccionar una tabla con todas las posiciones de memoria absolutas que han de corregirse. Dicha tabla se guarda en la cabecera de la imagen del ejecutable. Una vez que el cargador conoce el rango del mapa de memoria asignado al proceso, modifica todos los valores de las direcciones absolutas de memoria sumando la posición de memoria inicial del proceso con la posición relativa que aparece en la imagen del ejecutable.

Veamos un ejemplo. Supongamos que la sección de datos comienza en la posición 1.024B respecto al comienzo de la imagen del ejecutable y supongamos que dicha imagen se va a cargar en la posición de memoria principal 3.456B. En consecuencia, la sección de datos comienza en la posición 3.456B + 1.024B = 4.480B

Lo mismo sucede con los punteros de instrucción y de pila. El cargador del sistema operativo calculará los valores efectivos de ambos punteros una vez se conoce el mapa de memoria asignado.

Formatos de cabeceras

Existen diferentes formatos para las cabeceras de las imágenes de los ejecutables. A continuación se presentan los más importantes:

- a.out (*assembler output*): formato original utilizado en entornos UNIX; obsoleto, evolucionó a COFF
- COFF (*Common Object File Format*): formato para ejecutables, código objeto y librerías compartidas en entorno UNIX; reemplazado UNIX por ELF sirvió de base para formatos del entorno Windows
- ELF (*Executable & Linkable Format*): formato para ejecutables, código objeto, librerías compartidas y volcado de memoria en entorno UNIX; admite una gran variedad de secciones en los ejecutables
- MZ (Mark Zbikowski): utilizado en el entorno DOS; evolucionó dando lugar a varias extensiones
- PE (*Portable Executable*): formato para ejecutables, código objeto, librerías compartidas (DLL), archivos de fuentes y otros usos en entorno Windows; es una evolución de COFF

Proceso

Un proceso es un programa en ejecución, es decir, una secuencia de instrucciones con una serie de recursos asociados y un estado. Los recursos asociados son el contador de programa, los datos de memoria, la pila y su puntero, los registros y operadores de la ruta de datos y los recursos de E/S (puertos, descriptores de ficheros, etc.). El estado guarda información del punto de ejecución, situación de procesamiento, propietario, privilegios, comunicaciones, mecanismo de devolución del control al sistema operativo, etc. El estado se almacena de diferentes maneras dependiendo de la arquitectura.

<u>Creación de un proceso</u>

El sistema operativo toma la imagen de un ejecutable, actualiza sus direcciones absolutas e inicializa los datos que lo requieran, copia la imagen actualizada en memoria principal, asigna recursos y transfiere el control a la primera instrucción.

Finalización de un proceso

Un proceso puede terminar de manera normal devolviendo el control al sistema, puede ser abortado por otro proceso o puede finalizar por error.

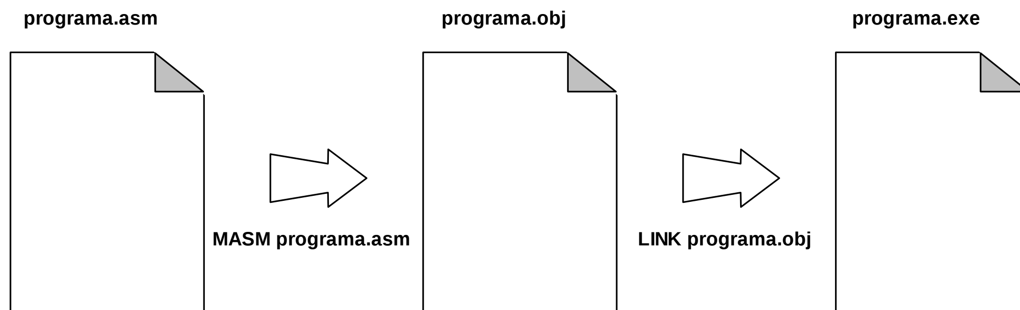
Estados de un proceso

Un proceso puede estar en ejecución, bloqueado (a la espera de algún evento externo) o listo (esperando a disponer de los recursos de ejecución del procesador).

Desarrollo de programas en ensamblador x86-16bits

Los ficheros ejecutables tienen extensión .EXE y se ajustan al formato MZ, es decir, la cabecera del fichero se ajusta a dicho formato MZ. El binario representa todos los segmentos declarados incluyendo la pila. Dependiendo del tipo de declaración, los segmentos pueden estar solapados parcial o totalmente o no estar solapados en absoluto. El tamaño del ejecutable no está limitado a 64KB.

El desarrollo de un programa .EXE parte de un fichero de código fuente en ensamblador (*programa.asm*) que se ensambla con **`masm5.1`** dando lugar a un fichero de módulo objeto (*programa.obj*) que finalmente se enalza (*linka*) con **`link`** generando el fichero ejecutable (*programa.exe*).



Diseño de secciones en ensamblador x86-16bits

El desarrollo de programas con lenguajes de alto nivel confía al compilador el diseño de las secciones de un programa. Por el contrario, el desarrollo de programas en ensamblador x86-16bits permite controlar totalmente este diseño si se considera necesario (***definición completa de segmentos***) o bien permitir que el programa ensamblador realice ese diseño de una manera transparente al programador (***definición simplificada de segmentos***).

Definición completa de segmentos

A continuación se muestra el "esqueleto" de varios programas que siguen la definición completa de segmentos. Se comenta cada uno de ellos al margen.

1	**Definición completa de segmentos Segmentos separados Salida mediante INT 20h**

```
ASSUME CS:codigo, DS:datos, SS:pila
;---------------------------------------------
; segmento de datos
datos    SEGMENT

   // DECLARACIÓN DE DATOS //

datos    ENDS
;---------------------------------------------
; segmento de pila
pila     SEGMENT STACK

   // DECLARACIÓN DEL TAMAÑO DE LA PILA //

   DB 256 DUP(?)    ; pila de 256 bytes
pila     ENDS
;---------------------------------------------
; segmento de código
codigo   SEGMENT
   main  PROC FAR

         ; dirección de retorno a la pila
         ; apunta al PSP (DS:0)
         PUSH DS
         XOR AX, AX
         PUSH AX

         ; inicializamos el segmento de datos
         MOV AX, DATOS
         MOV DS, AX

         // CÓDIGO DEL PROGRAMA //

         ; devolvemos el control al DOS
         RET
   main  ENDP
codigo   ENDS
;---------------------------------------------
; indicamos al programa ensamblador dónde
; comienza el código del programa
; sirve para dar valor a CS:IP
END main
```

La primera versión que presentamos es la más antigua por el método de salida (INT 20h).

Se declara el comienzo de cada segmento mediante una etiqueta y la directiva SEGMENT. Se declara el final con la misma etiqueta seguida de ENDS. El orden en el que se declaren será el orden en el que aparezcan en la imagen del ejecutable.

Siempre tiene que haber un segmento STACK (directiva SEGMENT seguida del operando STACK). En caso contrario el *linker* emitirá un error.

La directiva ASSUME indica qué registro de segmento sirve para direccionar cada segmento. El de código necesariamente debe usar CS.

El código fuente finaliza con la directiva END seguida de la etiqueta que marca el comienzo del código. Esto sirve para que CS:IP apunte a la dirección de arranque.

En este caso, el código se ha organizado como un procedimiento (*main*). Cuando se crea el proceso, el sistema operativo salta a la primera instrucción del procedimiento. Lo primero que hace dicho procedimiento es salvar en la pila la dirección de retorno (DS:0). Dicha posición apunta al PSP (área de memoria que guarda el estado del proceso). El PSP comienza con la instrucción INT 20h que devuelve el control al sistema.

La segunda tarea que realiza el procedimiento es cargar el registro DS con el valor de comienzo de la sección de DATOS.

2	**Definición completa de segmentos Segmentos separados Salida mediante INT 21h – servicio 4Ch**

```
ASSUME CS:codigo, DS:datos, SS:pila

   // RESTO DE SEGMENTOS //

;---------------------------------------------
; segmento de código
codigo   SEGMENT
   inicio: MOV AX, DATOS   ;inicializamos el
           MOV DS, AX      ;segmento de datos

           // CÓDIGO DEL PROGRAMA //

           MOV AX, 4C00h ;devolver el control
           INT 21h       ;al MS-DOS
codigo   ENDS
;---------------------------------------------
END inicio
```

En este caso solamente cambia, respecto al anterior, el modo en el que se devuelve el control al sistema operativo. Ahora se utiliza el servicio 4Ch de la INT 21h prescindiendo del PSP.

El segmento de código no adopta forma de procedimiento sino de secuencia de instrucciones.

Al igual que en el caso anterior, el código debe comenzar asignando valor al segmento de datos ya que si no se hace, no podrán reverenciarse los datos.

3	**Definición completa de segmentos** **Segmentos solapados** **Salida mediante INT 21h – servicio 4Ch**

```
COMUN  GROUP codigo, datos, pila
ASSUME CS:COMUN, DS:COMUN, SS:COMUN
;---------------------------------------------
; segmento de datos
datos    SEGMENT

   // DECLARACIÓN DE DATOS //

datos    ENDS
;---------------------------------------------
; segmento de pila
pila     SEGMENT STACK
   DB 256 DUP(?)
pila     ENDS
;---------------------------------------------
; segmento de código
codigo   SEGMENT
   inicio: MOV AX, COMUN   ;inicializamos el
           MOV DS, AX      ;segmento de datos

           // CÓDIGO DEL PROGRAMA //

           MOV AX, 4C00h ;devolver el control
           INT 21h       ;al MS-DOS
codigo   ENDS
;---------------------------------------------
END inicio
```

En este caso, los segmentos se solapan totalmente con la intención de que la imagen del ejecutable ocupe menos espacio.

La directiva GROUP agrupa varios segmentos lógicos en uno físico bajo un único nombre.

La carga del puntero del segmento de datos se realiza bajo el nombre común del grupo (COMUN en este ejemplo).

Las referencias a datos se harán anteponiendo a la etiqueta del dato el nombre del grupo. En este caso, COMUN:*nombre_dato*.

4	**Definición completa de segmentos** **Segmentos solapados** **Sin segmento de datos (datos en segmento de código)** **Salida mediante INT 21h – servicio 4Ch**

```
COMUN  GROUP codigo, pila
ASSUME CS:COMUN, DS:COMUN, SS:COMUN
;---------------------------------------------
; segmento de pila
pila     SEGMENT STACK
   DB 256 DUP(?)
pila     ENDS
;---------------------------------------------
; segmento de código
codigo   SEGMENT
   inicio:  JMP ppio

   // DECLARACIÓN DE DATOS //

   ppio:   MOV AX, COMUN   ;inicializamos el
           MOV DS, AX      ;segmento de datos

           // CÓDIGO DEL PROGRAMA //

           MOV AX, 4C00h ;devolver el control
           INT 21h       ;al MS-DOS
codigo   ENDS
;---------------------------------------------
END inicio
```

Con el fin de obtener una imagen de ejecutable aún más compacta, se puede evitar la declaración del segmento de datos. Ahora la reserva de espacio en memoria para los datos se realiza dentro del propio segmento de código.

El código comienza con una instrucción de salto incondicional que pasa por encima de la declaración de los datos.

Definición simplificada de segmentos

La definición simplificada de segmentos deja al programa ensamblador la tarea de diseño de los segmentos. Se invoca con la directiva DOSSEG. Los segmentos se declaran con las directivas .DATA, .CODE, .STACK (seguido del tamaño), etc.

<table>
<tr><th>1</th><th>Definición simplificada de segmentos</th><th>2</th><th>Definición simplificada de segmentos
Sin segmento de datos (datos en código)</th></tr>
<tr><td colspan="2">

```
dosseg
.model small
.stack 100h
.data

   // DECLARACIÓN DE DATOS //

.code
inicio:
   mov ax, @data
   mov ds, ax

   // CÓDIGO DEL PROGRAMA //

   mov ah, 4Ch
   int 21h
end inicio
```

</td><td colspan="2">

```
dosseg
.model small
.stack 100h
.code
inicio:
   jmp ppio

   // DECLARACIÓN DE DATOS //

ppio:
   mov ax, @code    ;ojo, no @data
   mov ds, ax

   // CÓDIGO DEL PROGRAMA //

   mov ah, 4Ch
   int 21h
end inicio
```

</td></tr>
</table>

Carga y finalización del proceso bajo MS-DOS

Para que el programa se convierta en proceso es necesario que el sistema operativo tome la imagen ejecutable, calcule y escriba las direcciones absolutas (*tabla de realocación* en el entorno MS-DOS), asigne recursos y transfiera el control al proceso. El estado del proceso de salva en un área de memoria contigua al binario conocida como PSP (*Program Segment Prefix*). El PSP tiene un tamaño de 256B = 100hB.

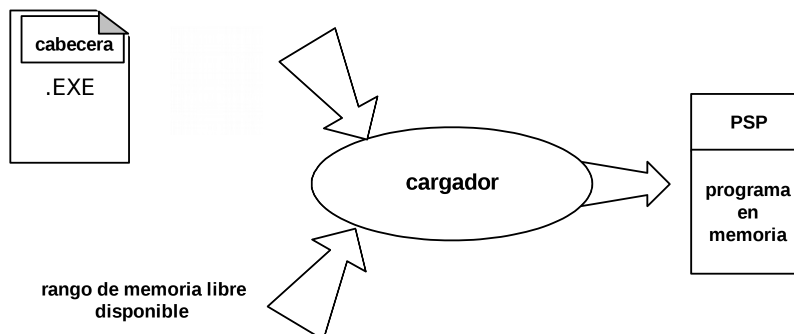
Como se ha indicado, para cargar los ficheros .EXE el sistema operativo debe actualizar las direcciones absolutas anotadas en la tabla de realocación sumando a cada valor de la tabla la dirección de comienzo del proceso.

En el momento en el que el sistema transfiere el control del proceso, los punteros adoptan estos valores:

- DS y ES apuntan al PSP
- CS e IP toman el valor especificado por la directiva END
- SS apunta al segmento de pila
- SP toma el valor del tamaño de la pila

Puesto que el sistema operativo asigna a DS y ES el puntero del PSP, es responsabilidad del programador comenzar el código asignando a DS (y ES si se usa) el valor correcto.

Para finalizar el proceso, ya hemos visto en los ejemplos de declaración de segmentos, que existen dos mecanismos disponibles: la llamada a la interrupción INT 20h y el servicio 4Ch de la interrupción 21h.



Prácticas

A) Creación de un ejecutable .EXE siguiendo la definición completa de segmentos.

Queremos crear un ejecutable que contenga la sencilla secuencia de operaciones aritméticas que se propone a continuación:

```
xor ax, ax
mov al, operando1
add al, operando2
mov resultado, ax
```

La declaración de datos será la mostrada a continuación. Se declaran tres variables. Dos de tamaño *byte* para los operandos y una de tamaño *word* para el resultado.

```
operando1 DB 10h
operando2 DB 20h
resultado DW 0000h
```

Crea 4 programas que incorporen este código usando los 4 “esqueletos” propuestos en esta práctica para desarrollar ejecutables usando la definición completa de segmentos. Una vez creados, pásaselos a DEBUG y determina el mapa de memoria de cada uno de ellos. Haz un dibujo y anota la dirección *base:desplazamiento* donde comienza cada segmento. Toma nota de los valores de los registros de segmento y de IP y SP antes de comenzar la ejecución y después de terminarla.

B) Creación de un ejecutable .EXE siguiendo la definición simplificada de segmentos.

Crea 2 programas usando los “esqueletos” propuestos en esta práctica para desarrollar ejecutables usando la definición simplificada de segmentos. Utilizando DEBUG, determina el mapa de memoria de cada uno. Haz un dibujo y anota la dirección *base:desplazamiento* donde comienza cada segmento. ¿Existe alguna diferencia con los mapas de memoria encontrados en el apartado A?.

La tarea que vamos a programar es la escritura de un mensaje en la pantalla. Para ello declaramos los datos siguientes:

```
mensaje   DB  "Hola a todos",13,10   ;Mensaje a escribir
longitud  EQU $ - mensaje            ;No es declaración; es etiqueta de nº entero
```

El código invocará al sistema operativo de la siguiente manera:

```
MOV CX, longitud
MOV DX, OFFSET mensaje
MOV BX, 1
MOV AH, 40h
INT 21h
```