Lenguaje ensamblador

Compiladores y Ensambladores

- Tanto los compiladores como los Ensambladores caen en la categoría de programas que llamamos traductores.
- Un traductor es un programa que acepta archivos en código fuente comprensibles para el humano y genera alguna clase de archivo binario.
- El archivo binario puede ser un archivo de programa ejecutable que el CPU puede comprender, o podría ser un archivo font, o un archivo de datos binarios comprimido, o alguno de los cientos de otros tipos de archivos binarios.
- Los programas traductores generan instrucciones de maquina que el CPU puede comprender.
- Un programa traductor lee un archivo en código fuente línea por línea, y escribe un archivo binario de instrucciones de maquina que realiza las acciones de computadora que el archivo de código fuente describe. Este archivo binario es llamado archivo de código objeto.
- Un compilador es un programa traductor que lee archivos en código fuente escrito en un lenguaje de alto nivel (tal como C++ o Pascal) y como salida escribe archivos en código objeto.
- Un ensamblador es un tipo especial de compilador, también es un programa traductor que lee archivos en código fuente y proporciona como salida archivos en código objeto para ser ejecutados por el CPU. Sin embargo, un ensamblador es un traductor diseñado específicamente para traducir lo que llamamos lenguaje ensamblador en código objeto.
- En el mismo sentido que un lenguaje compilador para Pascal o C++ compila un archivo en código fuente a un archivo en código objeto, decimos que un ensamblador ensambla un archivo en código fuente de lenguaje ensamblador a un archivo en código objeto. El proceso de traducción, es similar en ambos casos.
- El lenguaje ensamblador, sin embargo tiene una característica muy importante que se pone aparte de los de los compiladores, y es: el control total sobre el código objeto.

Lenguaje ensamblador

- **Definición [Duntemann]**: El lenguaje Ensamblador es un lenguaje traductor que permite tener un control total sobre cada instrucción generada por una máquina en particular mediante el programa traductor, tal programa traductor es llamado *Ensamblador*.
- Compiladores como Pascal o C++, realizan multitud de invisibles e inalterables procesos acerca de cómo una dada declaración del lenguaje será traducida en instrucciones de maquina.
- Por ejemplo en una simple instrucción en Pascal que asigna el valor 42 a una variable numérica llamada I será: I:= 42;
- Cuando el compilador de Pascal lee esta línea, realiza una serie de cuatro o cinco instrucciones de maquina donde se toma el valor 42 y se pone en memoria en una localidad codificada por el nombre I.
- Normalmente, el programador de Pascal no tiene idea de que esas cuatro o cinco instrucciones se realizan, y no tiene por completo forma de cambiarlas,

aun si supiera la secuencia de instrucciones de maquina, éstas son mas rápidas y mas eficientes que la secuencia que el compilador utiliza.

- El compilador de Pascal tiene su propia forma de generar instrucciones maquina, y no se tiene elección.
- Un ensamblador, sin embargo, tiene al menos una línea en el archivo de código fuente por cada instrucción de maquina generada.
- Tiene mas líneas para manejar muchas otras cosas, pero cada instrucción de maquina en el archivo de código objeto final es controlada por su correspondiente línea en el archivo de código fuente.
- Cada una de las muchas instrucciones de maquina en el CPU tiene un mnemónico correspondiente en lenguaje ensamblador.
- Como la palabra sugiere, estos mnemónicos son elementos de ayuda para auxiliar al programador a recordar un instrucción binaria de maquina en particular.
- Por ejemplo, el mnemónico para la instrucción binaria de maquina 9CH, que empuja las banderas del registro en la pila, es PUSHF, que es mas fácil de recordar que 9CH.
- Cuando se escribe el archivo de código fuente en lenguaje ensamblador, se colocan series de mnemónicos, típicamente un mnemónico por línea en el archivo de texto de código fuente.
- A continuación tenemos como ejemplo una porción de un archivo en código fuente:

```
MOV         AH, 12H           ; 12H  is Motor Information Service
MOV         AL, 03H           ; 03H is Return Current Speed function
XOR         BH, BH            ; Zero BH for safety's sake
INT         71H               ; Call Motor Services Interrup
```

- Aquí las palabras **MOV**, **XOR**, y **INT** son los mnemónicos, los números y los otros elementos inmediatamente a la derecha de cada mnemónico son los operandos de los mnemónicos.
- Hay varias clases de operandos para varias instrucciones de maquina, y algunas instrucciones (tal como **PUSH**) no tienen operandos.
- En conjunto, un mnemónico y sus operandos son llamados una instrucción.
- Y cuando hablamos de código binario específicamente nos referimos a una instrucción de maquina.
- La tarea mas importante del ensamblador es leer líneas del archivo de código fuente y escribir instrucciones de maquina en el archivo de código objeto.

BIBLIOGRAFÍA

- Jeff Duntemann
  Assembly Language Step-by-Step: Programming with DOS and Linux,
  Second Edition
  John Wiley & Sons © 2000 (613 pages)

1.7 Estructura de un programa en ensamblador

- Debug es una herramienta que nos sirve para conocer el funcionamiento de algunas instrucciones del lenguaje ensamblador, y permite realizar programas de tipo COM.
- **VER APÉNDICE B**
- MacroAssembler de Microsoft, nos permite con una estructura más sencilla y sólida, generar programas de tipo EXE y COM.
- Un programa EXE (EXEcution) difiere de un programa COM (COMmand) en que puede ser más grande de 64 k (los COM, al usar el mismo segmento para CS, DS, ES y SS no pueden superar este límite).
- Al trabajar con MacroAssembler, nuestro código fuente estará almacenado en un archivo ASCII, el cual deberá ser creado con un editor de textos. Esta característica nos permitirá gestionar el código fuente de una manera mucho más flexible.
- Es necesario conocer la estructura básica de un programa escrito en lenguaje ensamblador,
- Básicamente nuestros programas podrían estar englobados en cuatro bloques fundamentales:
    - Declaraciones y definiciones
    - Segmento de datos
    - Segmento de pila
    - Segmento de código

Declaraciones y definiciones
- definición de constantes, importación de código o información al compilador.

Segmento de datos
- Reserva de espacio para las variables que usa el programa.
- Para la creación de un segmento, se utiliza la directiva SEGMENT, la cual indica el comienzo del mismo.
- El segmento, en el código fuente, define un bloque de sentencias fuente, ya sean directivas o instrucciones.
- El final del segmento viene dado por la directiva ENDS (END Segment).

Segmento de pila
- Todos los programas han de llevar pila, con el fin de depositar la información necesaria para las llamadas a funciones, o bien almacenar datos temporalmente. En este bloque se define un tamaño para la pila.
- El segmento de pila se define como cualquier otro segmento, pero especificando la palabra reservada STACK.
- En este segmento reservamos únicamente espacio para la pila, es decir, definimos su tamaño.

Segmento de código
- Definición de todos los procedimientos e instrucciones del programa.

- Un procedimiento es un conjunto de sentencias, las cuales se engloban entre la directiva PROC (PROCedure) y la directiva ENDP (END Procedure).
- Programa que imprime en pantalla una frase para ejemplificar la estructura de un programa:

```
;=========================================================================
;=========================================================================
; PROGRAMA: pe01_hol.asm
; FUNCION : ESCRITURA DE UNA FRASE EN PANTALLA
;=========================================================================
;=========================================================================
; DECLARACION DE CONSTANTES
;=========================================================================
CR    EQU 13            ; Constante CR (Retorno de carro) En decimal o,
LF    EQU 0Ah           ; Constante LF (Salto de línea) en hex es igual
;=========================================================================
; DECLARACION DEL SEGMENTO DE DATOS
;=========================================================================
DATOS SEGMENT              ; Inicio del segmento de datos
      ;-------------------------------------------------------------
      MENSAJE DB CR,LF,'Hola Mundo !',CR,LF,'$'
      ;-------------------------------------------------------------
DATOS ENDS                 ; Fin del segmento de datos
;=========================================================================
; DECLARACION DEL SEGMENTO DE PILA
;=========================================================================
PILA  SEGMENT STACK        ; Inicio del segmento de pila
      ;-------------------------------------------------------------
      DB 64 DUP('PILA')    ; Inicialización de la pila
      ;-------------------------------------------------------------
PILA  ENDS                 ; Fin del segmento de pila
;=========================================================================
; DECLARACION DEL SEGMENTO DE CODIGO
;=========================================================================
CODIGO SEGMENT             ; Inicio del segmento de código
      ;-------------------------------------------------------------
      pe01_hol PROC FAR     ; Inicio procedimiento pe01_hol
            ;-----------------------------------------------------
            ASSUME CS:CODIGO,DS:DATOS,SS:PILA   ; Asignar segmentos
            ;-----------------------------------------------------
            MOV AX,DATOS    ; AX=Dirección del segmento de datos
            MOV DS,AX       ; DS=AX. Indicar el segmento de datos
            LEA DX,MENSAJE  ; DS:DX = dirección de MENSAJE
            ;-----------------------------------------------------
            MOV AH,9        ; Función DOS para escribir texto en pantalla
            INT 21H         ; Llamar a la interrupción del DOS
            ;-----------------------------------------------------
            MOV AX,4C00H    ; Función para terminar el programa
            INT 21H         ; y volver al DOS
            ;-----------------------------------------------------
      pe01_hol ENDP         ; Fin del procedimiento p1_hola
      ;-------------------------------------------------------------
CODIGO ENDS                ; Fin del segmento código
;=========================================================================
END pe01_hol              ; Empezar a ejecutar el procedimiento pe01_hol
;=========================================================================
;=========================================================================
;=========================================================================
```

Análisis del programa de ejemplo

- Declaraciones y definiciones
  - Las dos primeras sentencias funcionales del programa (sin tomar en cuenta los comentarios que inician con ";") permiten definir dos constantes: CR y LF.
  - Estas constantes no generan código objeto alguno, ya que indican al compilador que cuando encuentren en el código dichas constantes o símbolos, sustituya dichos símbolos por el valor indicado.

- Segmento de datos
  - En nuestro el ejemplo, hemos definido un segmento de datos llamado DATOS, y, dentro de éste, se han definido varias variables que se han inicializado mediante la directiva DB, la cual define bytes, en este caso con el valor ASCII de la cadena con que se crea.
  - El nombre de esta variable es: MENSAJE. Realmente es una etiqueta, la cual traducirá el compilador a su direccion de memoria de comienzo, dentro del segmento de datos.

- Segmento de pila
  - Definimos el tamaño de la pila, en este ejemplo añadimos 64 veces los bytes ASCII de la palabra PILA,
    con lo que otorgamos a la pila un tamaño de 256 bytes (64 x 4 bytes de la palabra PILA = 256 bytes).

- Segmento de código
  - El segmento de código, al igual que otros segmentos, queda englobado dentro del bloqueSEGMENT / ENDS.
  - Al comenzar este segmento, se define un procedimiento de tipo lejano, mediante la directiva:

```
Pe01_hol PROC FAR
```

  - Esta declaración sirve para información del compilador, con el fin de determinar dónde comienza un procedimiento.
  - Tras la definición de este procedimiento (que será el principal o inicial), se declaran los registros de segmento por defecto para direccionar los segmentos declarados en el programa. Esto se consigue mediante la directiva:

```
ASSUME CS:CODIGO,DS:DATOS,SS:PILA
```

  - Aún con esta declaración, el registro DS no apunta realmente al segmento de datos, por lo que se han definir las dos siguientes líneas de código:

```
MOV     AX,DATOS
MOV     DS,AX
```

  - Para llevar a cabo esta operación, se debe de utilizar un registro intermedio, en este caso AX. El segmento de código y el de pila no son necesarios direccionarlos, puesto que lo hace el DOS.
  - A través de la sentencia LEA, se deposita en el registro DX la dirección de memoria de una variable (offset dentro del segmento de datos, o sea, DS:DX).

- Tras determinar la dirección de ésta (que es una cadena), imprime una cadena en pantalla (el registro DX ya contiene la dirección de memoria donde se encuentra la cadena).
- Para ello, se deposita en el registro AH el número de servicio (9) y se llama a la interrupción 21H (interrupción de servicios estándar del DOS. Notar que aquí se especifica el sufijo H, puesto que está en hexadecimal).
- Después de imprimir la cadena, retorna (sentencia RET (RETurn)), que extrae de la pila la dirección de retorno y modifica IP para saber dónde debe continuar la ejecución.
- Este servicio del DOS permite concluir o finalizar el programa, retornando el control al DOS.

```
MOV   AX,4C00H
INT   21H
```

- Tras concluir el segmento de código, aparece esta sentencia:

```
END pe01_hol
```

  Con la cual se finaliza el programa fuente, e indica al compilador dónde se encuentra el procedimiento o dirección por donde debe empezar a ejecutarse el programa.

BIBLIOGRAFÍA

- http://club.telepolis.com/mydream/Asm/
  Rafael Hernampérez Martín

1.8 Ensamble, enlace y ejecución

Compilación del código fuente

- Escrito el código fuente de un programa en ensamblador, éste es solamente un texto en código ASCII.
- Es muy fácil editar, modificar, añadir o eliminar código de lo que será realmente el programa.
- Este texto ha de ser convertido a su equivalente en códigos de operaciones del microprocesador, por lo que se ha de ***"compilar"*** y ***"enlazar"***.
- Para llevar a cabo estas operaciones, disponemos de ensambladores como MacroAssembler o TurboAssembler.
- MacroAssembler dispone de los programas MASM y LINK, que compilan y enlazan al archivo fuente respectivamente.

Compilación

- Existen varios compiladores como son MASM y TASM para ambiente Windows así como NASM y GAS para ambientes Linux.
- Un compilador se encarga de comprobar los errores de sintaxis de un código fuente, además de algunos detalles del código, como detectar la presencia o no de un segmento de pila.
- Tras la revisión de la sintaxis, se procede a la traducción de sus sentencias a un archivo objeto (.OBJ), el cual aún no es un ejecutable completo.
- Esto nos permite preparar varios módulos por separado, para después enlazarlos en el programa .EXE final.
- El uso de módulos ahorra el tener que recompilar rutinas cada vez que hay una modificación en el código.
- La sintaxis completa de MASM es la siguiente:

```
MASM archivo.asm,archivo.obj,archivo.lst,archivo.crf
```

Donde:

```
archivo.asm -> Nombre del archivo con el código fuente
archivo.obj -> Nombre del archivo objeto a generar.
archivo.lst -> Nombre del archivo listado.
archivo.crf -> Nombre del archivo de referencias cruzadas.
```

- Un archivo listado contiene el código máquina y el puntero de programa asociado a cada sentencia.
- Además contiene los posibles errores que se pudieran generar durante la compilación, por lo que nos permitiría realizar un seguimiento del mismo para detectar errores y depurar.
- Un archivo de referencias cruzadas contiene la información sobre cada símbolo o constante, además de las sentencias donde se hace referencia.
- La sintaxis más rápida y normal que se utiliza es la siguiente:

```
MASM archivo;
```

- En este caso, genera, automática el archivo objeto (.OBJ) con el mismo nombre que el archivo fuente (en el caso de no haber errores).
- Si no se especifica el punto y coma del final, el compilador preguntará, uno a uno, el nombre de cada uno de los archivos citados anteriormente.

- Si archivo fuente contuviese errores, se especificaría el número de línea y el error que ha producido dicha línea.
- Al final, se informaría sobre el número de avisos (Warnings Errors) y el número de errores graves (Severe errors) que contiene el fuente.

Enlace o LINK

- Al igual que con los compiladores, existen ligadores para ambiente windows como son el LINK que se usa con le compilador MASM y el TLINK que se utiliza con el compilador TASM.
- Los programas LINK se encargan de enlazar los códigos objetos, completar las referencias y construir el archivo ejecutable (.EXE).
- La sintaxis del link o enlazador para el compilador MASM es la siguiente:

```
LINK archivo.obj+archivo.obj..,archivo.exe,archivo.map,archivo.lib,archivo.lib...

  archivo.obj+archivo.obj... -> Lista de los archivos objeto que
  conformarán el archivo ejecutable.

  archivo.exe -> Nombre del archivo ejecutable a generar.
  archivo.map -> Nombre del archivo mapa.
  archivo.lib+archivo.lib... -> Lista de los archivos de librería, de
  los cuales utilizan rutinas nuestro programa.
```

- Un archivo mapa contiene toda la información acerca de los segmentos generados en el programa (comienzo, final, nombre, tamaño...).
- Los archivos librerías son archivos objeto con rutinas preparadas para ser utilizadas en nuestros programas, con sólo hacer referencia a éstas.

[jjf]

Ejecución o corrida del programa

- Una vez ya realizado el ensamblado y el enlace se obtiene un archivo .EXE.
- El cual podemos ejecutar únicamente llamando al archivo con extención .EXE y dando enter en la computadora, con lo cual podemos ver la corrida de nuestro programa en pantalla por lo general.
- La sintaxis para la ejecución es la siguiente:

```
archivo.exe
```

BIBLIOGRAFÍA

- http://club.telepolis.com/mydream/Asm/
  Rafael Hernampérez Martín

- jjf

1.9 El entorno de programación

Creando un directorio de trabajo

- El proceso de crear y perfeccionar programas en lenguaje ensamblador involucra muchas clases diferentes de archivos de DOS, así como numerosas herramientas de software.
- Diferente al cuidadoso, completo e integrado ambiente de desarrollo en Pascal, Delphi o Visual Basic, en el desarrollo del lenguaje ensamblador se involucran una gran cantidad de elementos con algún montaje necesario.
- Se recomienda crear utilizando los comandos de DOS, un subdirectorio de desarrollo en el disco duro de la computadora, poniendo todas los distintos elementos en el, por facilidad podríamos llamarlo ASM, pero puede llevar el nombre que se desee.
- El directorio creado deberá contener lo siguiente:
  - El editor de texto o ambiente de desarrollo.
  - El programa ensamblador
  - Los subdirectorios derivados del desempaquetamiento de programas instalados.
  - El programa ligador.
  - El programa DEBUG. Una copia de DEBUG.EXE viene instalada en todas las copias de DOS en Windows, se debe encontrar el archivo DEBUG.EXE en su sistema y copiarlo al su directorio de trabajo.
  - Listados de programa de código fuente.
- **VER APÉNDICE C.**

Trabajando con los programas

- Ya creado el directorio de trabajo es necesario acceder a el por medio de un shell, que seria una ventana de DOS para el sistema operativo Windows y una terminal para los ambientes de Linux.
- Por medio del shell se van a ejecutar los programas:
  - Ensamblador
  - Ligador o LINK y
  - El DEBUG.
- Es necesario aprender los comandos y parámetros de los programa que se van a utilizar (así como DOS en su caso) para poder realizar adecuadamente la ejecución de los mismos.
- Aunque es suficiente con el shell para ejecutar los programas que se necesitan para desarrollar programas en el lenguaje ensamblador, existen herramientas graficas para ambiente Windows como son el Windbg que puede realizar tareas similares al DEBUG de DOS y entornos integrados de desarrollo como NASM-IDE.
- Es necesario mencionar que la sintaxis ocupada en programas para Linux es diferente a la que se ocupa para el ambiente Windows.

BIBLIOGRAFÍA

- Jeff Duntemann
  Assembly Language Step-by-Step: Programming with DOS and Linux, Second Edition
  John Wiley & Sons © 2000 (613 pages)

- jjf