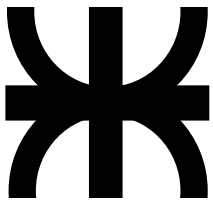


UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL SAN NICOLÁS
INGENIERIA EN ELECTRÓNICA

TÉCNICAS DIGITALES III

TRABAJO PRÁCTICO N° 3

ENLACE DE RUTINAS DE ASSEMBLER EN UN LEGUAJE DE ALTO NIVEL COMO C

AÑO 2004

TEORIA DEL TRABAJO PRACTICO Nº 3:

Por qué combinar

Ventajas de combinar lenguajes

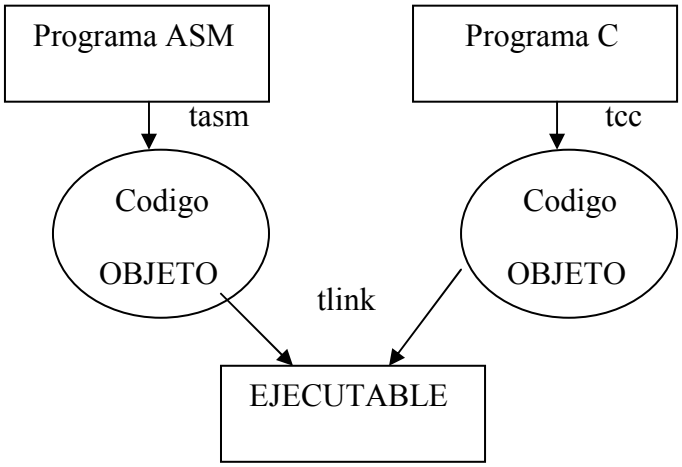
- Realizar el trabajo fuerte en un lenguaje de alto nivel, lo que facilita la programación global.
- Conseguir la máxima eficiencia en los módulos críticos, usando el lenguaje de más bajo nivel.
- Al combinar lenguajes, se puede obtener una combinación óptima entre eficiencia y facilidad de programación, en cada sección del programa.

Ligado de módulos múltiples

Herramientas

- 1 - Utilizar el ensamblador en linea (inline asm) que esta disponible en algunos compiladores. asm { ..... }

- 2 - Escribir un fuente en C, otro en ensamblador y construir un único programa ejecutable:



Cada módulo se compila o ensambla por separado, obteniendo el archivo objeto (.OBJ)

Se incluyen todos los módulos al construir el ejecutable, en el proceso de ligado.

Se requiere una estructura que defina la interfaz entre los distintos módulos

Para enlazar rutinas en lenguaje ensamblador con C tenemos 2 alternativas:

Creación de un ejecutable con módulos múltiples

En ensamblador

```
TASM uno

TASM dos

TLINK uno dos, resultado
```

C con ensamblador

```
TASM uno

TCC -c dos

TLINK uno dos,resultado
```

Como se puede ver de los ejemplos que tenemos arriba, el proceso es similar; se crean varios objetos, ensamblando o compilando (con la opción -c); y luego, el TLINK se encarga de ligar los módulos en un ejecutable.

Normalmente, el segundo caso se puede hacer en forma tal que el compilador de C llame al TLINK automáticamente (lo que es más conveniente, pues requeriremos menos parámetros):

```
TASM uno

TCC -Eresultado.exe dos.c uno.obj
```

Para que el programa en ensamblador se pueda enlazar con el programa escrito en C debe cumplir una serie de requisitos:

Requisitos para la unión

Características del .ASM

- Usar un modelo de memoria compatible con el archivo .C; preferiblemente, el mismo
- Iniciar el fuente con la directiva DOSSEG, que indica al ensamblador que ordene los segmentos en la forma estándar en lenguajes de alto nivel.
- Definición "externa" o "pública" de las variables que se van a compartir
- Directivas EXTERN, PUBLIC y GLOBAL
- Manejo correcto de parámetros de funciones, por medio del uso de la pila acorde al "orden C" de parámetros, y al modelo de memoria

Variables Compartidas

Características generales

Hay una serie de registros que no se pueden alterar: estos registros son: CS, DS, SS, BP, SP. Además si en el programa en C se usan variables de tipo 'register' tampoco se podrá modificar SI y DI.
Si queremos modificar alguno de estos registros, habrá que salvarlos en la pila.

Utilizar los nombres apropiados de variables y funciones, todos los nombres de variables y funciones declarados en C se pueden usar en ensamblador precedidos del simbolo “_“ .

Ej:

```
int var;   /* variable declarada en C */
```

La usariamos de la siguiente forma en ensamblador:

```
_var
```

Entonces :

- Las variables definidas por un compilador de "C" internamente se preceden por un subguión (_)
- Las variables compartidas por distintos módulos deben ser definidas en "C" como globales, no estáticas (automáticas).
- Se debe usar la opción apropiada en el ensamblador para que sea "sensitivo" a mayúsculas/minúsculas, como es el caso de "C" (/ml en TASM)

PUBLIC y EXTRN

Declarar los nombres a compartir como públicos o externos, en ensamblador usaremos EXTRN o PUBLIC y en C usaremos extern.

Ej: programa en C que llama a una función en ensamblador:

```
extern int primo(int n); /* prototipo de la funci¢n */
```

y en ensamblador haríamos:

```
PUBLIC _primo
_primo PROC NEAR .....
```

Entonces :

Toda variable o función que esté definida en el módulo actual, y deba ser usada por otro módulo, se declara como PUBLIC

Sintaxis:

```
PUBLIC Simbolo
```

Toda variable o función definida en otro módulo, que se vaya a usar en el actual,

se declara como EXTRN

Sintaxis:

```
EXTRN Simbolo:Tipo
```

Tipos válidos

Para su uso en EXTRN

1. BYTE - Equivale a "char" de "C"

2. WORD - Equivale a "int", "short" o "enum" de "C". También se usa para sustituir a los apuntadores cortos (near *)
3. DWORD - Equivale a "long" de "C". También se usa para "float", y para los apuntadores largos (far *)
4. QWORD - Equivale a 8 bytes, como "double" en "C".

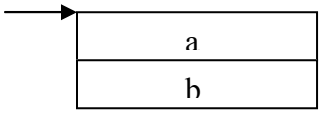
Reglas para el manejo de parámetros de "C"

Debemos conocer el sistema de paso de parámetros que utiliza el lenguaje. En el caso del C los parámetros se pasan por la pila, colocándose de derecha a izquierda.

Como los parámetros se colocan en la pila, se hace "push" de sus valores, de derecha a izquierda; es decir, el último parámetro está más abajo en la pila. El programa en C se encarga de eliminarlos de la pila al volver.

Ej: funcion(a,b)

```
        _funcion        /* dir. Retorno*/
        ret
```



El valor de retorno de una función se coloca en:

- AL, si es de tamaño byte
- AX, si es de tamaño word (o near *)
- DX:AX, si es double word (o far *)

NOTA: la dirección de retorno puede ocupar 2 o 4 bytes dependiendo de si es un salto de tipo NEAR o FAR.
En C son DD los tipos: unsigned long, long, flota, puntero de tipo far (char far *ptr;)
(solo se admiten valores de retorno de 16 o 32 bits)

MODELOS DE MEMORIA

Usar nombres de segmento compatibles con el programa en C.
Existen los siguientes modelos:

```
.MODEL TINY | SMALL | COMPACT | LARGE | HUGE
 dependiendo del modelo que usemos en C :
.CODE    ;segmento de código
.DATA    ;segmento de datos
.CONST   ;segmento de constantes
.STACK   ;segmento de pila
```

Podemos hacer .STACK 100H y reservamos 100h bytes de pila. Tambien podemos hacer: MOV AX,@DATA

Generación Del EXE final en Turbo C

tcc programa.c programa.asm

A continuación se dá un ejemplo:

Código en C :

```
/* devuelve los numeros primos hasta el 1000 */
#include <stdio.h>
extern int primo(int n);  /* definimos el prototipo de la funci¢n */
main()
{
  int i,n;
  n=1000;
  for (i=2;i<=n;i++)
    if (primo(i))        /* llamamos a la funci¢n externa en asm */
      printf("%d ",i);
  return 0;
}
```

Código en ASM :

```
; Decide si un número que se le pasa como parámetro es primo o no
; Devuelve false (0) o true (!0) al programa en C
; Se invoca desde C: primo(i) siendo i de tipo int (16 bits)
.MODEL SMALL          ;Usamos el modelo SMALL y debe coincidir con el
                      ; usado en C
.CODE                 ;Solo usamos el segmento de código
PUBLIC    _primo      ;Se declara el procedimiento como público
                      ; para que pueda ser llamado desde otro
                      ; módulo
_primo  PROC NEAR
        PUSH BP       ;Guardamos BP
        MOV BP, SP    ;Recuperamos en BP el puntero de la pila
        MOV SI, [BP+4] ;Recuperamos en SI el parámetro que le hemos
                      ; pasado [BP+4] (2 del BP + 2 de la dirección
                      ; de retorno)
        MOV BX, 2     ;BX contiene el divisor (de 2 a SI)
BUCLE:  XOR DX, DX    ;Dividendo DX&AX
        MOV AX, SI    ;Divisor BX
        CMP AX, BX    ;Si ya se ha dividido por todos
        JE  PRIMO     ; es que es primo
        DIV BX        ;Dividimos DX&AX / BX
        OR DX, DX     ;DX contiene el resto de la divisi¢n
        JZ NOPRIMO    ; si es 0 el numero no es primo
        INC BX        ;Incrementamos el divisor
        JMP BUCLE     ; volvemos a dirvidir
PRIMO:  MOV AX, 1     ;Colocamos el valor de retorno a 1 (true en C)
        JMP FIN       ;
NOPRIMO:XOR AX, AX ;Colocamos el valor de retorno a 0 (falso en C)
FIN:    POP BP        ;Quitmos el valor de BP que nosotros guardamos
                      ; al principio
        RET           ;Retornamos
_primo  ENDP
        END
```

ANEXO :

*Otra forma de compilar:
tcc -c programa1.c tasm programa2.asm

tlink c:\tcc\lib\c0?.obj programa1.obj programa2.obj, programa.exe,, c:\tcc\lib\c?.lib,, /3

donde ? es el modelo de memoria (S)mall, (L)arge, ...
/3 por si quieres compilar rutinas con 32bits