- Varias aplicaciones escritas en lenguaje de alto nivel requieren partes críticas escritas en ensamblador
- Implica poder llamar desde programas escritos en lenguaje de alto nivel compilables a rutinas escritas en ensamblador
- También es posible llamar desde programas escritos en ensamblador a rutinas escritas en lenguaje de alto nivel compilables
- Factible si programas de ensamblador siguen las convenciones (nomenclatura, paso de parámetros y resultados) de los lenguajes de alto nivel.
- Programa en C se compila a código objeto, programa en ensamblador se ensambla a código objeto y montador (linker) del compilador C genera el ejecutable enlazando ese ficheros objeto

Convenciones del lenguaje C relacionadas con:
- Uso de direcciones cortas (near) o largas (far) para acceder a datos (variables) y/o procedimientos
- Nomenclatura de segmentos, variables y procedimientos
- Paso de parámetros a procedimientos y devolución de resultados

Cuando se compila un programa en C se debe escoger un modulo de memoria, cada uno determina la ubicación en memoria de los segmentos lógicos (código, datos y pila) y si se usan direcciones cortas o largas para acceder a ellos.

Seis modelos de memoria en Turbo C:
- Tiny
- Small
- Medium
- Compact
- Large
- Huge

Punteros far → segmento y offset
Punteros Near → offset

- Tiny
  - mínima ocupación de memoria
  - los 4 segmentos son idénticos, el programa ocupa hasta 64KB
  - código, datos y pila en el mismo segmento físico
  - Programas compiladores en este modelo pueden convertirse en .COM (drivers) mediante EXE2BIN o con la opción /t del montador
  - Punteros cortos (near) para código y datos

- SMALL
  - programas pequeños en los que no es necesario mínima ocupación de memoria
  - un segmento físico para código (hasta 64KB) y otro para datos y pila (hasta 64KB)
  - Punteros cortos (near) para código y datos

- Medium
  - programas grandes que usan pocos datos
  - varios segmentos físicos para código (hasta 1MB) y uno para datos y pila (hasta 64KB)
  - punteros largos (far) para código y cortos (near) datos

- Compact (código near, punteros far)
  - programas pequeños que usan muchos datos
  - un segmento físico para código (hasta 64KB) y varios para datos y pila (hasta 1MB)
  - Punteros cortos (near) para código y largos (far) datos

- Large
  - programas grandes que usan muchos datos
  - Varios segmentos físicos para código (hasta 1MB) y para datos y pila (hasta 1MB). En total no puede superar 1MB
  - Punteros largos (far) para código y datos

- HUGE
  - similar al large con algunas ventajas y desventajas
  - Punteros normalizados (offset < 16)
  - Variables globales estáticas no pueden superar 64KB (posible manipular bloques de datos de más de 64KB)
  - compilador inserta código que actualiza automáticamente registros de segmento de datos (punteros a datos siempre normalizados)
  - Modelo más costoso en tiempo de ejecución

Resumen

Modelo	Segmentos		Punteros	
	Código	Datos Pila	Código	Datos
Tiny	64KB		Near	Near
Small	64KB	64KB	Near	Near
Medium	1 MB	64KB	Far	Near
Compact	64KB	1 MB	Near	Far
Large	1 MB	1 MB	Far	Far
Huge	1 MB	1 MB	Far	Far

Nomenclatura

- El compilador C siempre nombra de igual forma a los segmentos lógicos que utiliza
  - El segmento de código se llama _TEXT
  - El segmento _DATA contiene las variables globales inicializadas
  - El segmento _BSS contiene las variables globales no inicializadas
  - El segmento de pila lo define e inicializa el compilador de C en la función main
  - En los modelos pequeños de datos, todos los segmentos están agrupados con el nombre DGROUP
  - El compilador de C añade un _ delante de todos los nombres de variables y procedimientos

```
int a=12345
char b='A'
char c[]="Hola mundo"
int d=12;
```

```
_DATA SEGMENT WORD PUBLIC 'DATA'
PUBLIC _a, _b, _c, _d
_a DW 12345
_b DB 'A'
_c DB "Hola mundo", 0
_d DW 12
_DATA ENDS
```

```
main()
{
  funcion();
}
```

```
_TEXT SEGMENT BYTE PUBLIC 'CODE'
_main PROC FAR
  CALL _funcion
  RET
_main ENDP
_TEXT ENDS
```

- Las variables y procedimientos de ensamblador que sean accedidos desde programas en C deben llevar un _ por delante que no aparece en C
- En C se distingue entre mayúsculas y minúsculas es necesario que el ensamblador también las distinga con las opciones /mx o /ml
- En C un procedimiento que llama a otro, apila sus parámetros antes de ejecutar el CALL, los procedimientos de ensamblador que llamen a funciones de C también han de apilar sus parámetros
- Los parámetros se apilan en orden inverso a como aparecen
- Tras retornar de la subrutina se extraen los parámetros sumando al registro SP el tamaño en bytes de los parámetros
- Los parámetros de un byte (char) se apilan con dos bytes (el más significativo a 0)

Paso de parámetros

- Se apilan en formato little endian: palabra menos significativa en dirección menor byte menos significativo
- Para pasar por parámetro punteros a funciones o datos es necesario saber en qué modelo de memoria se está compilando el programa en C, para apilar el registro de segmento (modelo largo) o no apilarlo modelo corto

Paso de parámetros (ejemplo)

void funcion(char, long int, void *)

- llamada en modelo largo (punteros FAR para datos y código)

```
función ('P', 0x23, &V);

call - función
```

SO	SEG V
48	OFFSET V
46	00h
	00h
44	00h
	23h
42	00h
	50h
40	SEG @retorno
38	OFFSET @retorno

SP = BP → 36 BP Inicial

acceso a parámetros

```
- funcion PROC FAR
   push bp
   mov bp, sp          bx := offset V
   les bx, [bp+12]     es = SEG V
   mov dx [bp+10] 0x23
   mov ax  bp 18  segunda parte de 0x23
```

Devolución de resultados

- Las variables de retorno de una función con una longitud de 16 bits se devuelven al procedimiento llamante en AX y las de 32 bits en DX:AX

Ejemplo 1

```
int variable_C;
extern int dato_ar;
int funcion (int a, char far *p, char b);
main()
{ int a=123
  char b='F'
  char far *p
  variable_C=funcion (a, p, b);
}
DGROUP DATA, _BSS
_DATA SEGMENT WORD PUBLIC 'DATA'
   PUBLIC _dato_ar
   dato_ar DW 0
_DATA ENDS
_BSS SEGMENT WORD PUBLIC 'BSS'
   EXTRN_variable_C: WORD
   _fun_ar DW ?
_BSS ENDS
_TEXT SEGMENT BYTE PUBLIC 'CODE'
   ASSUME CS: _TEXT, DS: DGROUP
   PUBLIC _funcion
   _funcion PROC NEAR
      PUSH BP
      MOV BP SP
      MOV BX, [BP+4] → carga a
      LDS SI, [BP+6] → carga p en DS:SI
      MOV CX, [BP+10] CL='F' CH=00h
      ...
      MOV AX, CX
      POP BP
      RET
   _funcion ENDP
   _TEXT Ends
END
```

Definición de variables locales

- Las rutinas de C definen las variables locales que necesitan en la pila por encima de BP
- Se inicializan en el orden que son declaradas
- Se acceden mediante [BP-2], [BP-4] ...

La instrucción asm permite insertar en ensamblador en el programa C

```
main
{ int d1=5, d2=4
  asm {
        push cx
        push ax
        mov cx, d2
        cmp cx, 0
        jE final
  }
final:
  asm {
        ...
      }
```

No se debe emplear BP, ya que lo utiliza el compilador para el acceso de variables locales

aunque un 1 byte se cuenta 00 byte del char

int → 2 bytes char → 1 byte long → 4 bytes short → 2 bytes

Las cifras acaban en 0