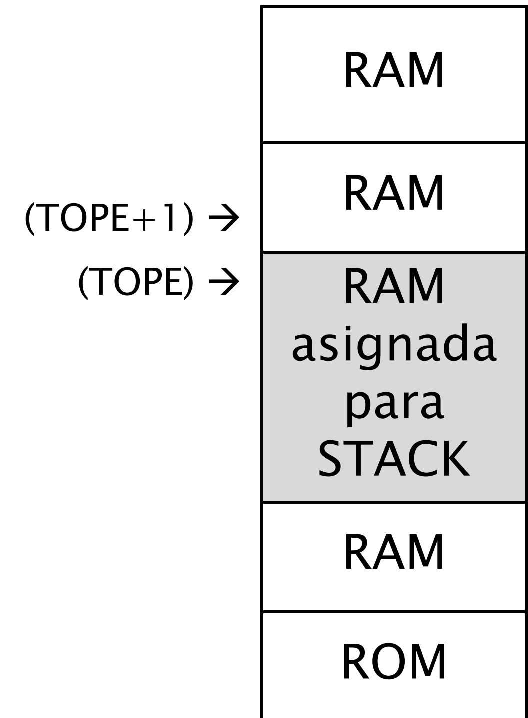
12. – Stack, Subrutinas, Parámetros, Variables

Introducción a los microprocesadores
2015

STACK

- Bloque de memoria RAM
  - DEBEMOS prever el tamaño asignado al Stack
  - NUNCA salir del tamaño asignado, de lo contrario:
    - Sobreescribo **variables** o **código en RAM**
    - “Almaceno” en **ROM** o **áreas vacías**



- Puntero
  - Registro SP (16 bits)
  - Apunta al último lleno
  - No se inicializa en ciclo de Reset
    - Si TOPE es la dirección más alta del Stack ➔ **LD SP, TOPE +1**
    - Si TOPE = 0xFFFF ➔ **LD SP, 0**
- “Crece” hacia abajo

Ejemplo:

PUSH BC

(SP – 1)← B

(SP – 2)← C

SP ← SP - 2

STACK

- ¿Quienes utilizan el STACK?
  - PUSH
  - POP
  - CALL
  - RET
  - Interrupciones (*)
  - RETN
  - RETI

PUSH, POP → Preservar registros

CALL, RET → Subrutinas

Interrupciones (*), RETN, RETI → Rutinas de atención a interrupciones (*)

- ¿Por qué puede crecer el STACK?
  - Anidamiento de subrutinas y/o interrupciones (*)
  - A medida que se agregan niveles de anidamiento crece el Stack.

(*) Cuando ocurre una interrupción, se ejecuta "automáticamente" una rutina llamada rutina de atención a la interrupción afectando el Stack de igual forma que una subrutina. Lo veremos en breve

Comunicación con la Subrutina

- Comunicación de:
  - Datos de entrada a la subrutina
  - Resultados

Pasaje de parámetros entre el programa llamante y la subrutina

- Alternativas para el pasaje de parámetros:
  - Registros internos
  - Direcciones reservadas de memoria
  - Stack
  - Puntero a una zona de memoria
  - Otras
  - Mezcla de las anteriores

Se debe acordar previamente el mecanismo de pasaje de parámetros y la ubicación de cada uno

Comunicación con la Subrutina

Secuencia de llamada

- No se trata de un CALL y nada más
- La invocación de una subrutina involucra una secuencia de llamada en la cual se realiza:
  - Preservar registros (*)
  - Preparación de los argumentos
  - Instrucción CALL
  - Obtención de resultado
  - Restablecer Stack
  - Restablecer registros (*)

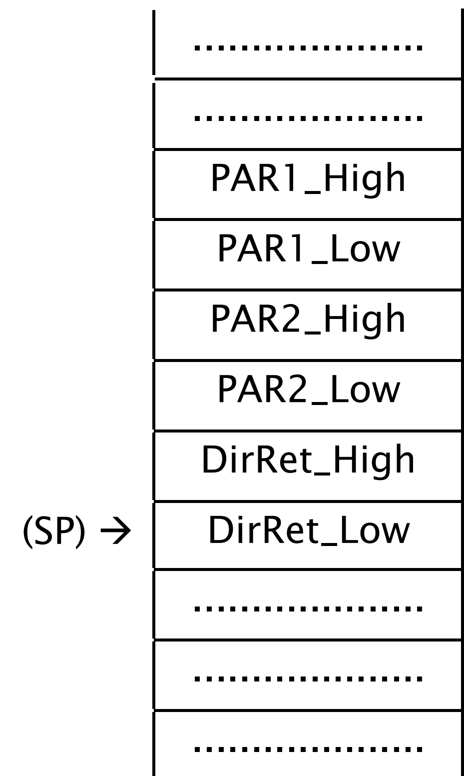
(*) eventualmente si así se requiere

Pasaje de parámetros en Stack

- Supongamos que debemos pasar 2 argumentos: PAR1 y PAR2
- Programa llamante:

```
LD BC, PAR1
PUSH BC
LD BC, PAR2
PUSH BC
CALL sub1
```

- Al ingresar en la subrutina el Stack está así:



Pasaje de parámetros en Stack

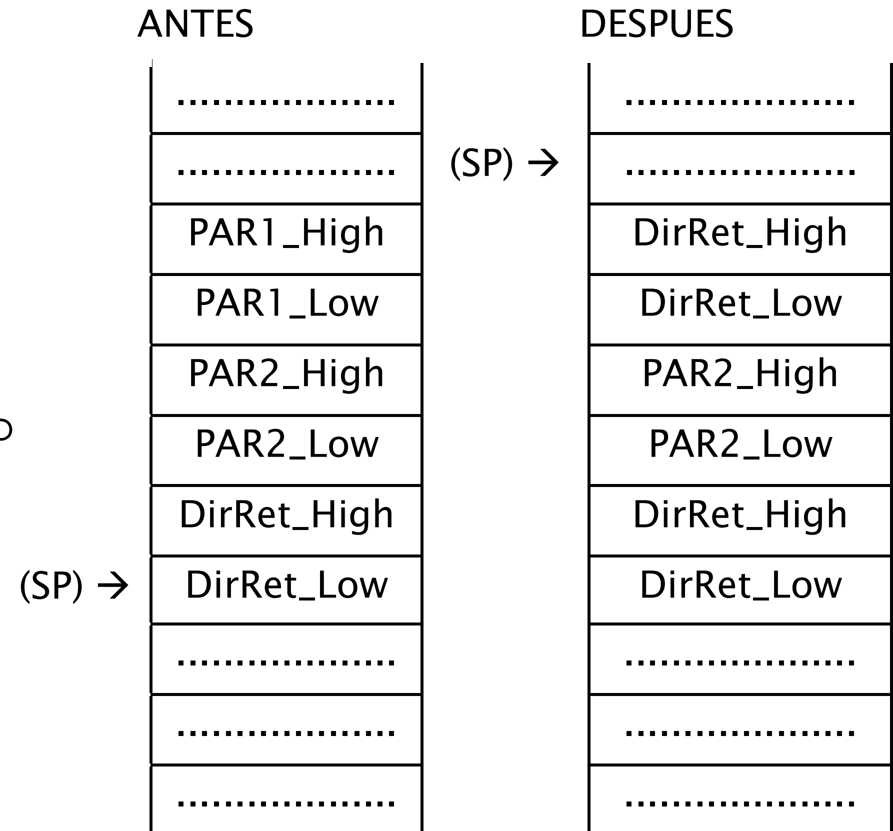
- Existen varias alternativas para retirar los argumentos del Stack en la subrutina:

  - **Método 1: POP y PUSH**
  - **Método 2: Instrucción EX (SP), HL**
  - **Método 3: Direccionamiento al stack**

Método 1: POP y PUSH

- Subrutina:

```
POP HL   ; HL ← dir de retorno
POP BC   ; BC ← param2
POP DE   ; DE ← param1
PUSH HL ; repongo dir de retorno
...
...
RET
```

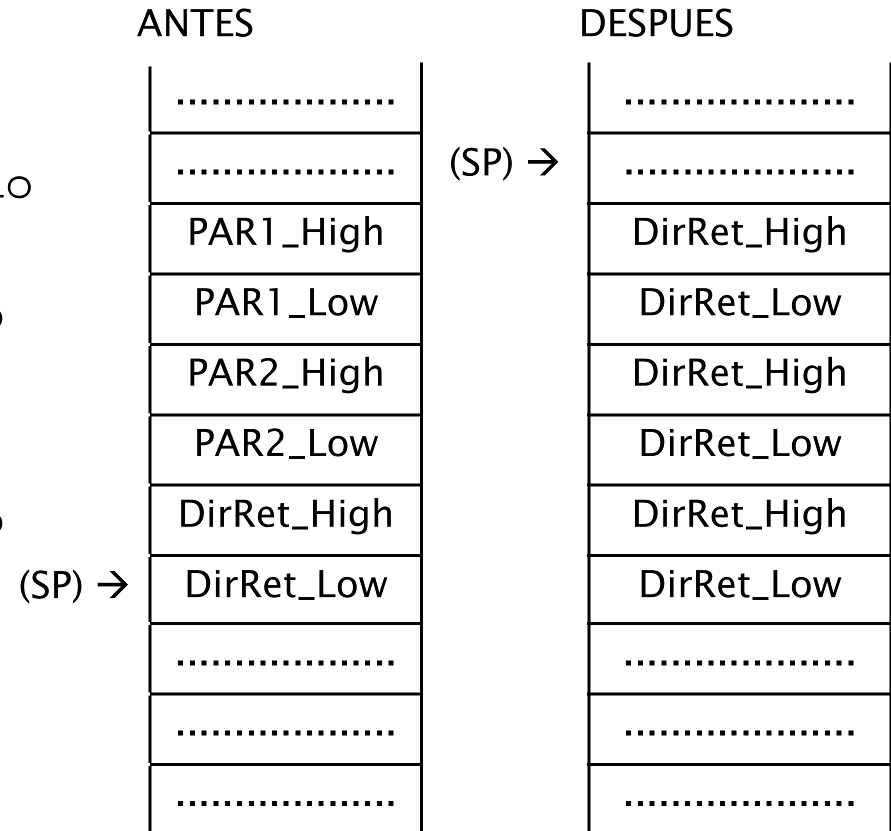


- Sencillo
- Destruye el contenido de los registros.
- El Stack queda alineado

Método 2: EX (SP), HL

- Subrutina:

```
POP HL          ; HL ← dir de retorno
EX (SP), HL     ; HL ← param2
                ; (SP) ← dir retorno
POP IX          ; IX ← dir retorno
EX (SP), IX     ; IX ← param1
                ; (SP) ← dir retorno
...
RET
```

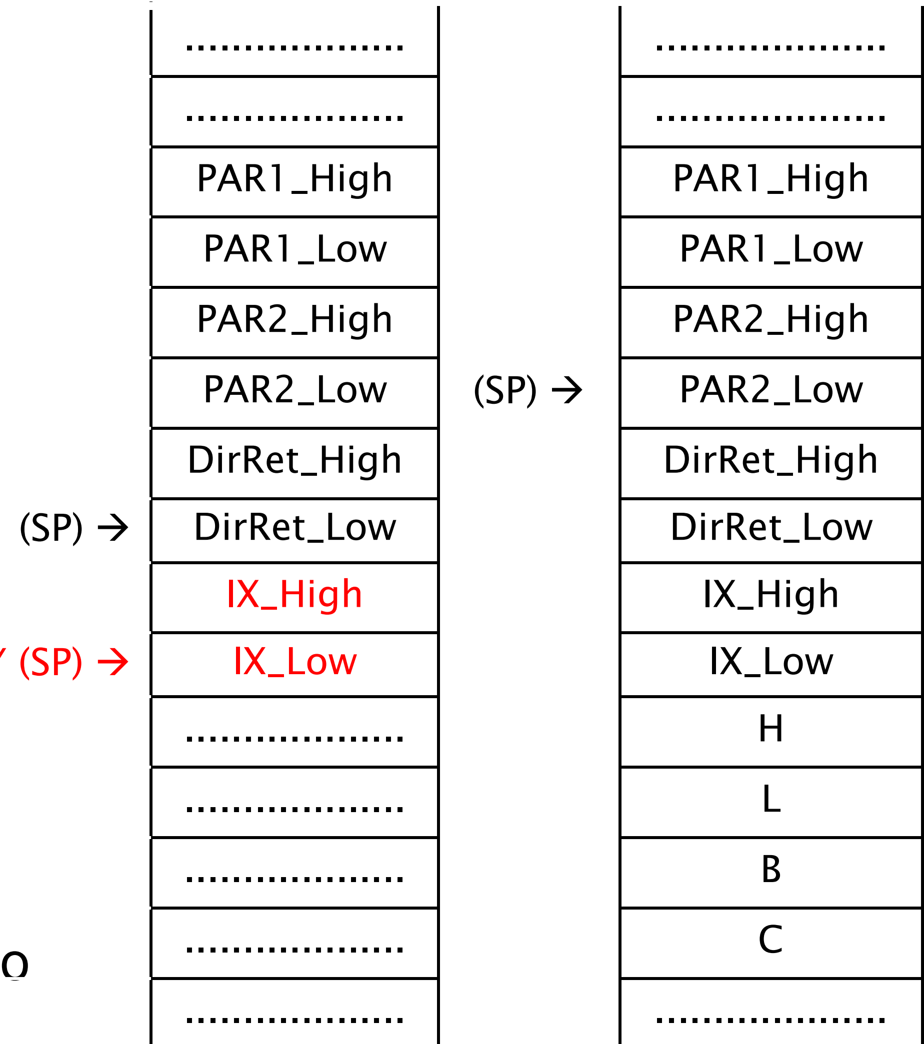


- Sencillo
- Destruye el contenido de los registros
- El Stack queda alineado

Método 3: Direccionamiento indirecto al Stack

- Subrutina:

```
PUSH IX
LD IX, 0
ADD IX, SP        ; IX ← SP (*)
PUSH HL
PUSH BC
LD H, (IX+4)
LD L, (IX+5)      ; HL ← par2
LD B, (IX+6)
LD C, (IX+7)      ; BC ← par1
...
POP BC
POP HL
POP IX
RET
```



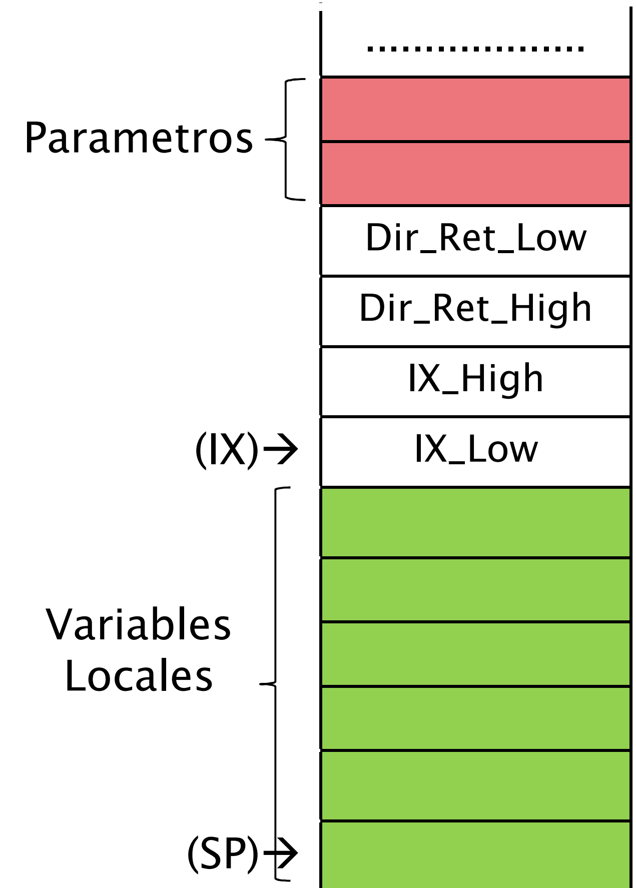
- Mas complicado
- Preserva registros
- El Stack queda desalineado

Reentrancia

- Reentrancia:
  - Nueva invocación antes de retornar de invocación anterior.
  - A propósito para algoritmos recursivos
  - No buscado cuando hay procesos concurrentes (p. ej. interrupciones)
- Problema:
  - Uso de memoria reservada para almacenamiento de parámetros o resultados intermedios
    - La segunda invocación destruye datos de la primera
    - Estructuras de datos con contenido incoherente
- Soluciones
  - Área de memoria independiente para cada invocación
  - Deshabilitar interrupciones

Reentrancia

- Área de memoria independiente...
  - Opción 1:
    - Dos copias de la subrutina (y de sus variables), una invocada desde programa principal y otra desde interrupciones (considerando que no se anidan interrupciones).
  - Opción 2
    - Área de memoria local para cada invocación
    - En el stack
    - Es lo que hace un compilador C o Pascal para variables locales de una función.
  - Deshabilitar interrupciones



Ubicación de variables en memoria

- En lenguajes como C y Pascal
- Variables Globales:
  - Existen durante toda la ejecución del programa
  - En direcciones reservadas de memoria
- Variables locales
  - Se crean en cada invocación de la función, desaparecen al retornar.
  - Área de memoria independiente para cada invocación
  - Se ubican en el stack

Programa en C vs Programa compilado

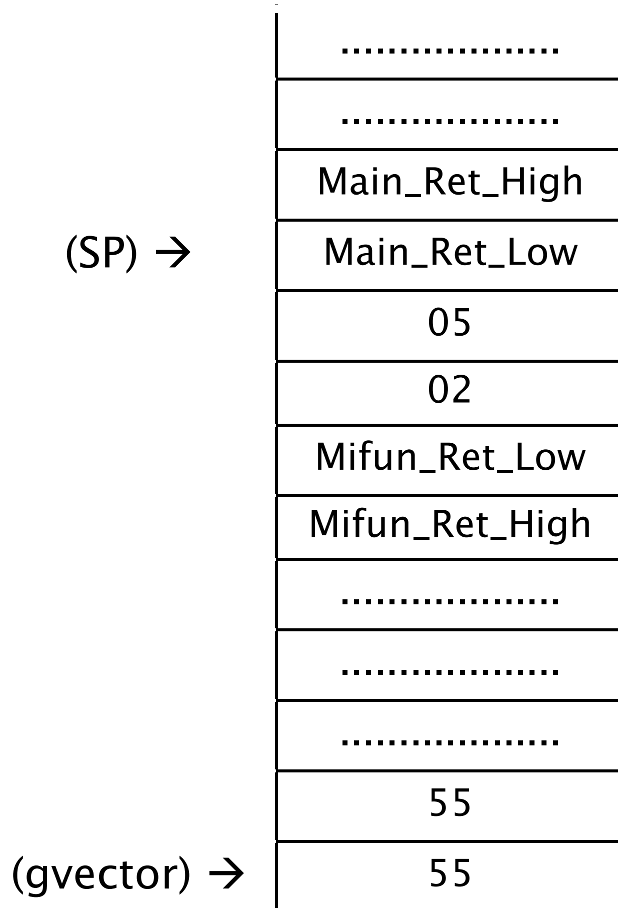
Programa en C:

```
char mifuncion(char pa, char pb){
   char lvector[10];
   lvector[0] = pa;
   lvector[9] = pb;

   return( pa + pb );
}

int gvector[10];
main(){
 gvector[0] = 0x5555;
 mifuncion(2,5);
 return( gvector[0] );
}
```

Programa compilado:

```
.area _DATA
_gvector::
       .ds 20
       .area _CODE
;sdcc_local.c:1: char
; mifuncion(char pa, char pb){
_mifuncion:
       push    ix
       ld      ix,#0
    …
;sdcc_local.c:11: main(){
_main:
;sdcc_local.c:12:
;     gvector[0] = 0x5555;
       ld      hl,#_gvector
       ld      (hl),#0x55
    …
       ret
```

Programa en C vs Programa compilado

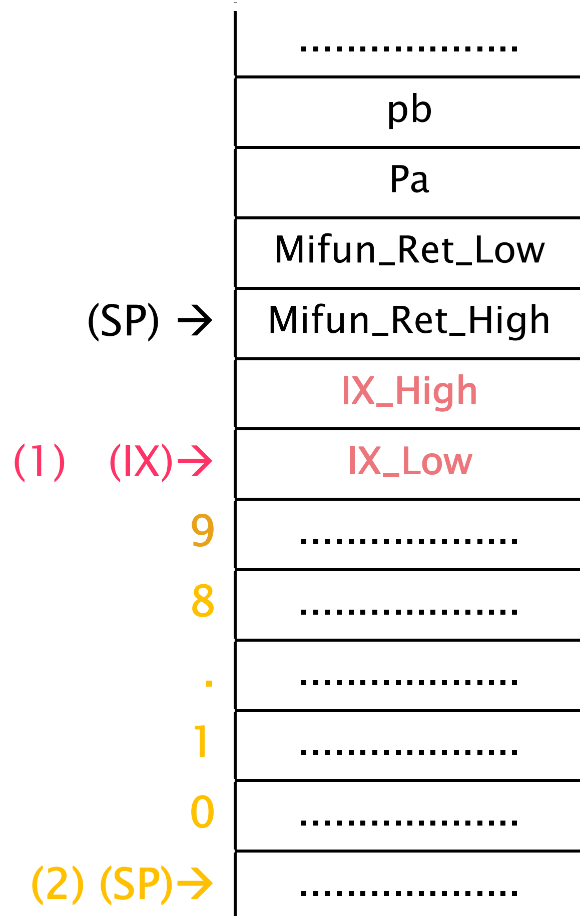



```
_main:
;sdcc_local.c:12: gvector[0] =
0x5555;
    ld      hl,#_gvector
    ld      (hl),#0x55
    inc     hl
    ld      (hl),#0x55
;sdcc_local.c:13: mifuncion(2,5);
    ld      hl,#0x0502
    push    hl
    call    _mifuncion
    pop     af
;sdcc_local.c:14:
return(gvector[0]);
    ld      hl,#_gvector
    ld      c,(hl)
    inc     hl
    ld      b,(hl)
;     reg = bc
    ld      l,c
    ld      h,b
    ret
```

Alinea SP

Devuelve en HL = 0x55 el resultado

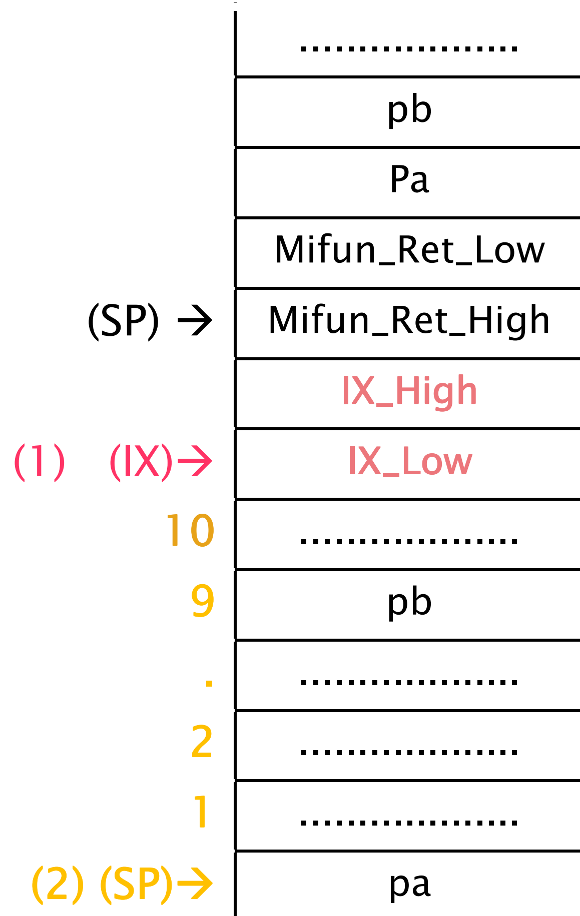
Programa en C vs Programa compilado




```
_mifuncion:  ;return
    push  ix
    ld    ix,#0
    add   ix,sp    ; (1)
    ld    hl,#-10
    add   hl,sp
    ld    sp,hl       ; (2)
    . . .
    . . .
;sdcc_local.c:6: return(pa+pb);
    ld    a,4(ix)     ;pa
    add   a,5(ix)     ;pb
    ld    l,a
;   genRet
    ld    sp,ix
    pop   ix
    ret
```

Devuelve en L el resultado

Programa en C vs Programa compilado




```
_mifuncion: ;var locales
    . . .
;sdcc_local.c:3: lvector[0] = pa;
    ld      hl,#0x0000
    add     hl,sp
    ld      c,l
    ld      b,h
;   genAssign (pointer)
    ld      a,4(ix)
    ld      (bc),a
;sdcc_local.c:4: lvector[9] = pb;
    ld      a,c
    add     a,#0x09
    ld      c,a
    ld      a,b
    adc     a,#0x00
    ld      b,a
;   genAssign (pointer)
    ld      a,5(ix)
    ld      (bc),a
;sdcc_local.c:6: return( pa + pb );
    . . .
```