Guía de programación ATMega328pa

Autor: Alberto J. Molina

Última modificación: 6/05/13

Índice

1. Pseudocódigo / Lenguaje Ensamblador
2. Sentencias de asignación
3. Sentencias de control
4. Bucles
5. Subrutinas
6. Interrupciones

Indice

1. Pseudocódigo / Lenguaje Ensamblador
2. Sentencias de asignación
3. Sentencias de control
4. Bucles
5. Subrutinas
6. Interrupciones

Pseudocódigo / Lenguaje Ensamblador

- La programación en bajo nivel es compleja. Requiere el conocimiento de:

  - La arquitectura del procesador.

  - La organización de datos en la memoria y de los modos de direccionamiento.

  - El amplio juego de instrucciones.

- Y precisa que el programador tenga en cuenta la notación numérica de los datos, a la hora de realizar operaciones con ellos (comparaciones, detección de overflows, etc.).

NADA DE ESTO SERÍA NECESARIO EN LENGUAJES DE ALTO NIVEL

Pseudocódigo / Lenguaje Ensamblador

- Sin embargo, la programación en bajo nivel permite al programador el control pleno de los recursos de la máquina (p.e: rutina de retardo temporal)

- El programador de bajo nivel requiere realizar dos tareas:

    - Determinar el algoritmo

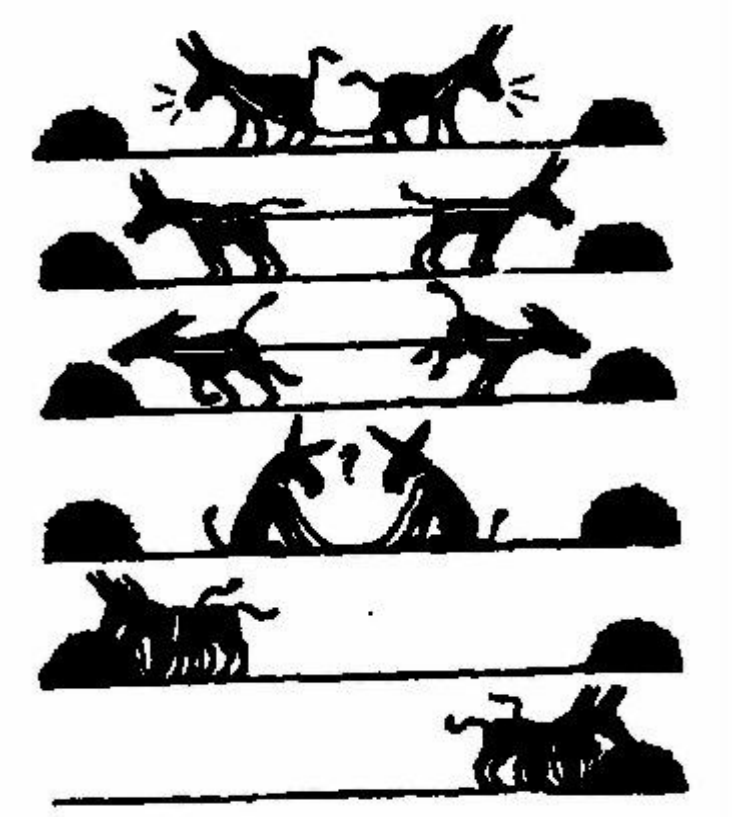
    - Expresarlo en bajo nivel

Pseudocódigo / Lenguaje Ensamblador

- Intentar realizar esas dos tareas a la vez dificulta enormemente la programación y reduce la probabilidad de éxito.

- Hagamos las dos tareas por separado.

DIVIDE Y VENCERÁS



Pseudocódigo / Lenguaje Ensamblador

- Procedimiento a seguir pasaría:

1) Describir el algoritmo en lenguaje de alto nivel (pseudocódigo).

2) Traducir dicho pseudocódigo a ensamblador (procedimiento que siguen los compiladores).

Indice

1. Pseudocódigo / Lenguaje Ensamblador
2. Sentencias de asignación
3. Sentencias de control
4. Bucles
5. Subrutinas
6. Interrupciones

Sentencias de asignación I

vble1 ← const

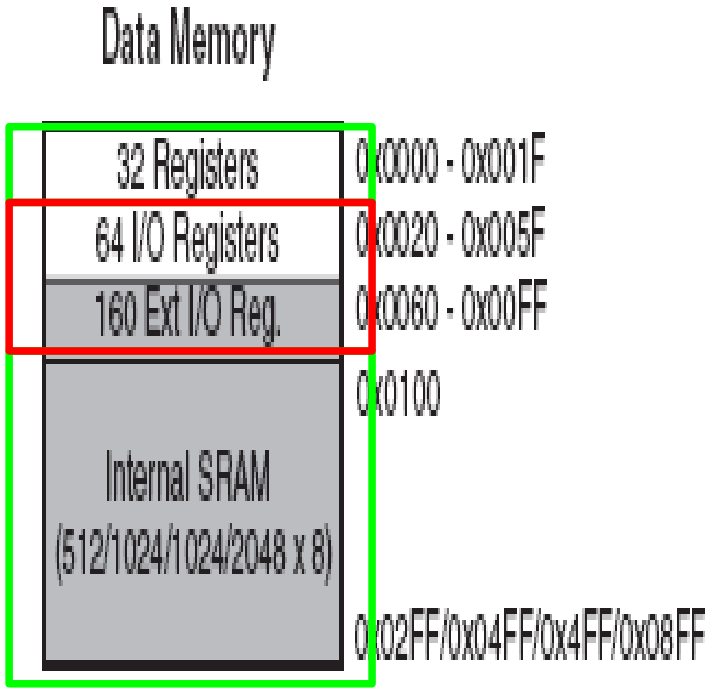
vble1 ← vble2

vble1 ← (vble2)

a[i] ← vble

¿Dónde se encuentran las variables?

Sólo para Operaciones de ES

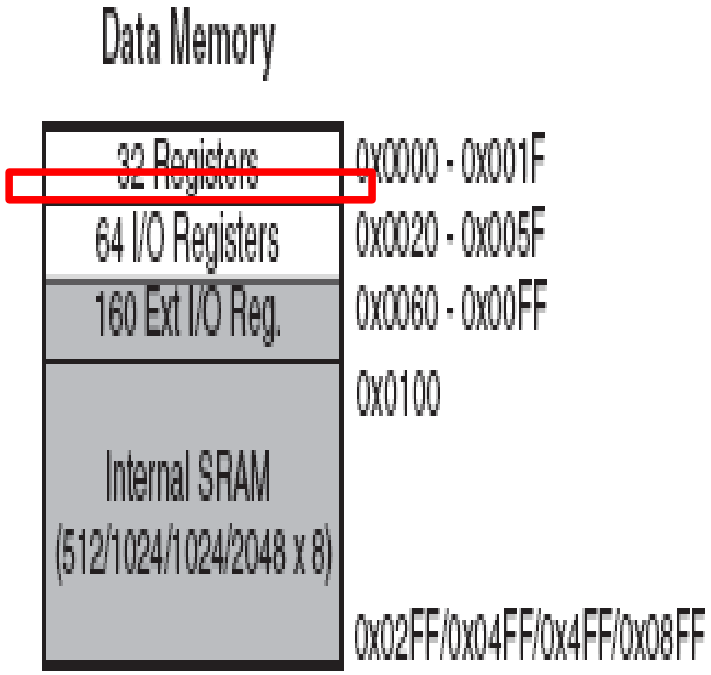


Sentencias de asignación II (Inicialización)

vble1 ← const

Ej.: A ← 10

```
.def A = Rd ;d =[16,31]

.equ const = 10

LDI A, const
```

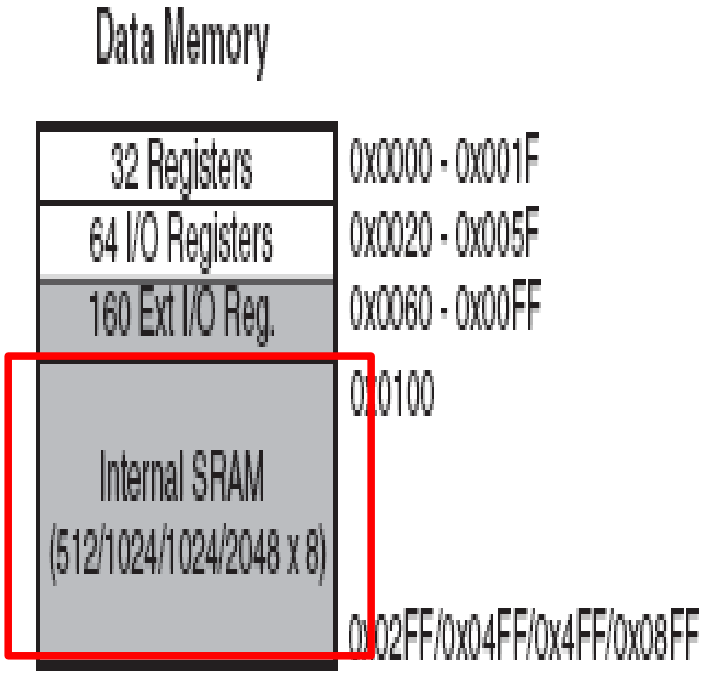


Sentencias de asignación III (inicialización)

vble1 ← const

Ej.: A ← 10

```
.def temp = R16;
.def A = dir; dir =[$100,RAMEND]
.equ const = 10
LDI temp, const
STS A, temp
```



Sentencias de asignación IV

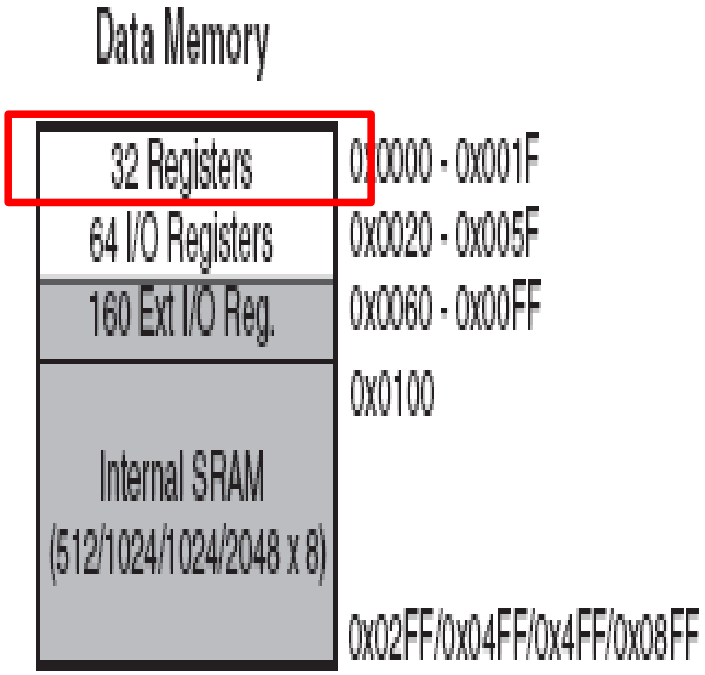
vble1 ← vble2

Ej.: A ← C

.def A = Rd1;

.def B = Rd2; d1,d2 = [0,31]

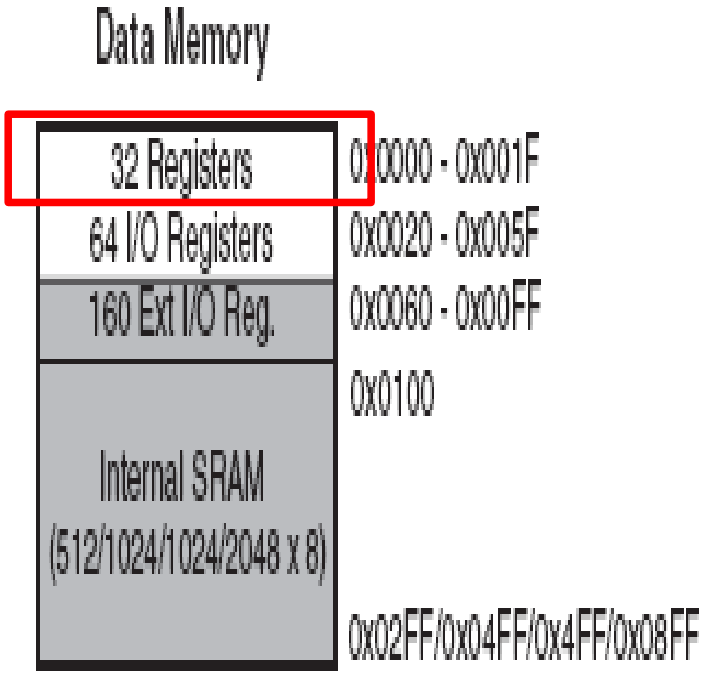
MOV A,B



Sentencias de asignación V

vble1 ← vble2 (tamaño W)

Ej.:A ← B



.def AH = R1

.def AL = R0

.def BH = R3

.def BL = R2

MOVW AH:AL,BH:BL

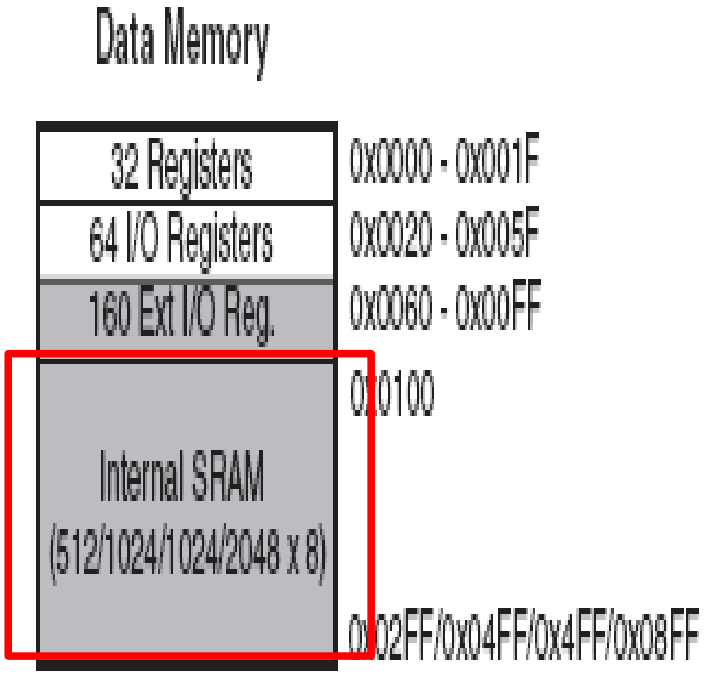
(Recordad notación little-endian)

Sentencias de asignación VI

vble1 ← vble2

Ej.: A ← C

```
.equ A = dir1;
.equ B = dir2; dir1,2 = [$100,RAMEND]
.def temp = R16
LDS temp, B
STS A,temp
```

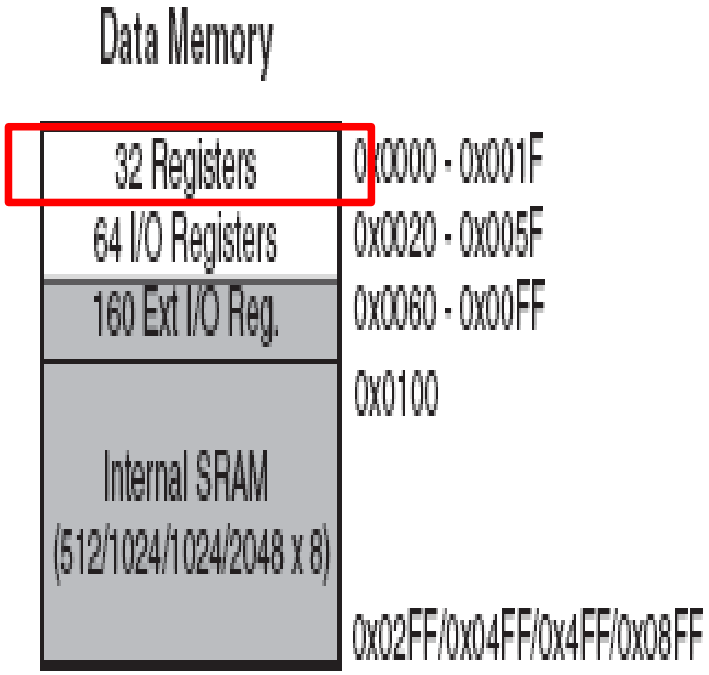


Sentencias de asignación VII

vble1 ← (vble2)

Ej.: A ← (punt)

```
; (Registros X,Y,Z)

.def A = Rd ; d=[0,31]

LD  A, Z
```



Sentencias de asignación VIII

vble1 ← (vble2)

vble2 ← vble2+1

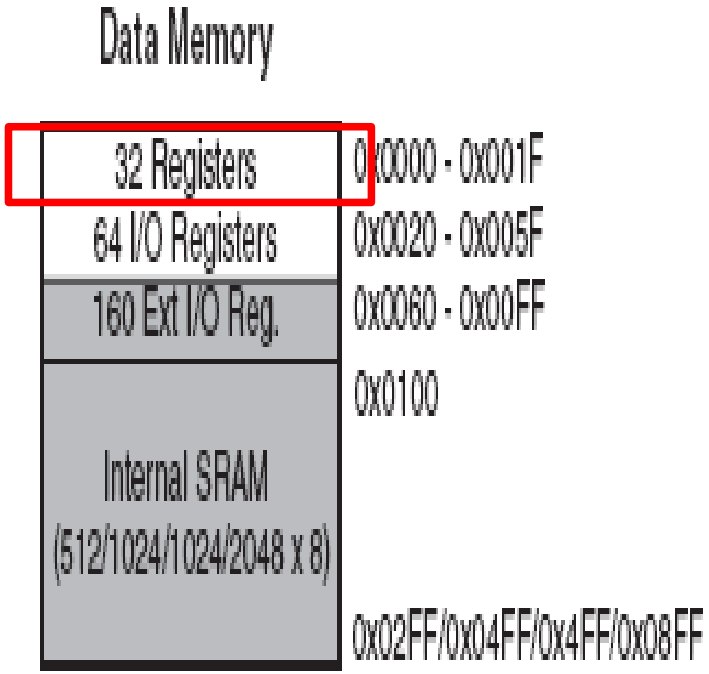
Ej.: A ← (punt)

punt ← punt+1

; (Registros X,Y,Z)

.def A = Rd ; d=[0,31]

LD A, Z+



Sentencias de asignación IX

vble2 ← vble2-1

vble1 ← (vble2)

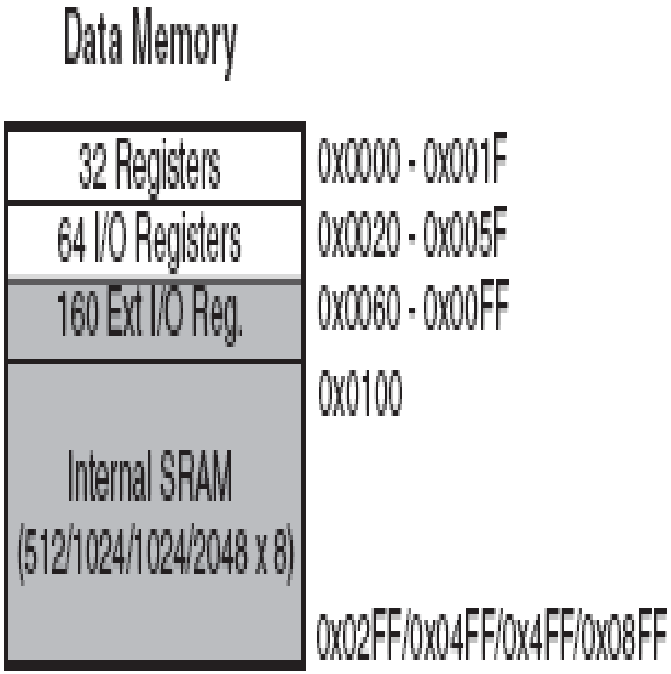
Ej.: punt ← punt-1

A ← (punt)

; (Registros X,Y,Z)

.def A = Rd ; d=[0,31]

LD A, -Z

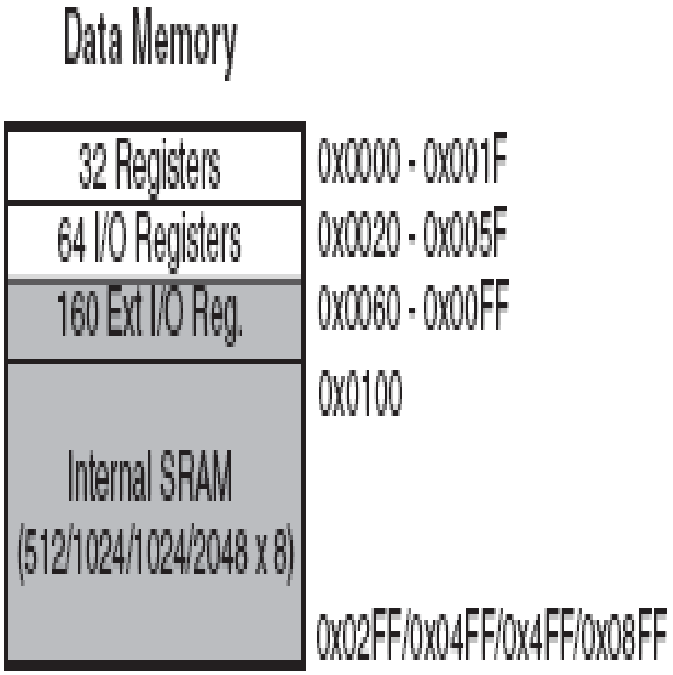


Sentencias de asignación X

vble1 ← (vble2+D)

Ej.: A ← (punt+2)

```
; (Registros Y,Z)
.def A = Rd ; d=[0,31]
LDD  A, Y+2
```



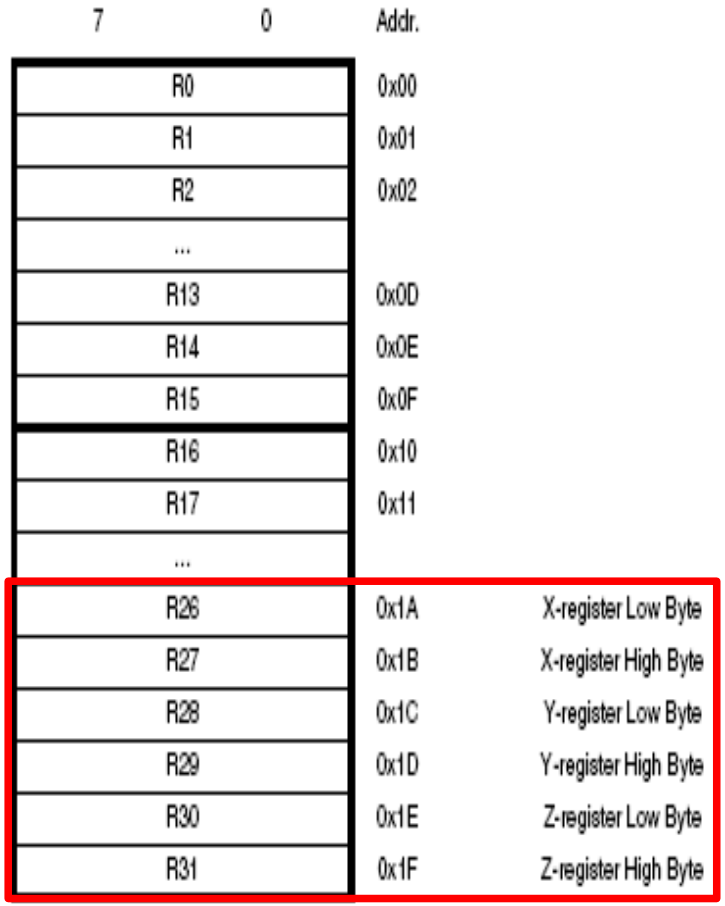
Sentencias de asignación XI

Para inicializar un puntero

Z ← $0100

LDI ZL, low($0100)

LDI ZH, high($0100)



Sentencias de asignación XII (acceso a matrices)

vble1 ← Matriz[vble2]

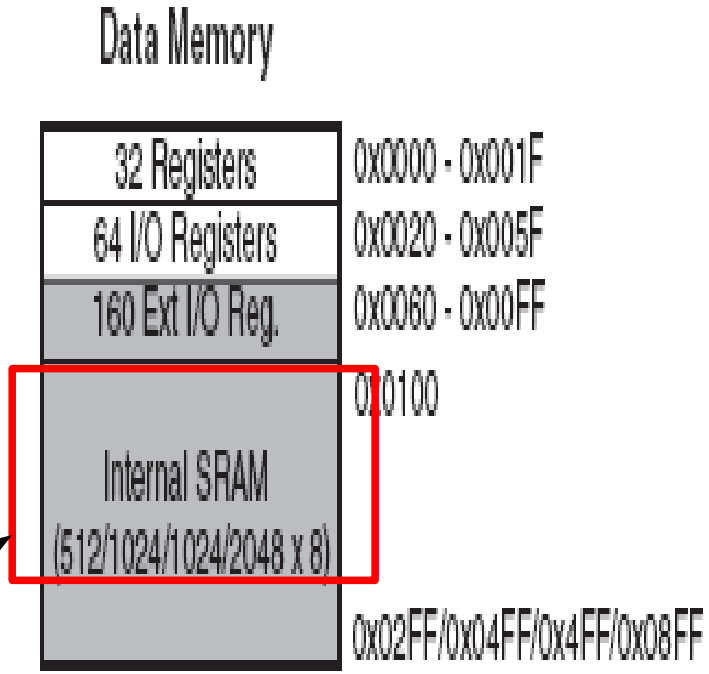
Vble1 ← Matriz[const]

Matriz[const] ← Vble1

Matriz[vble2] ← Vble1

Ej.: A ← B[C], A ← B[2];

¿Dónde se guardan las matrices de datos?



Sentencias de asignación XIII (acceso a matrices)

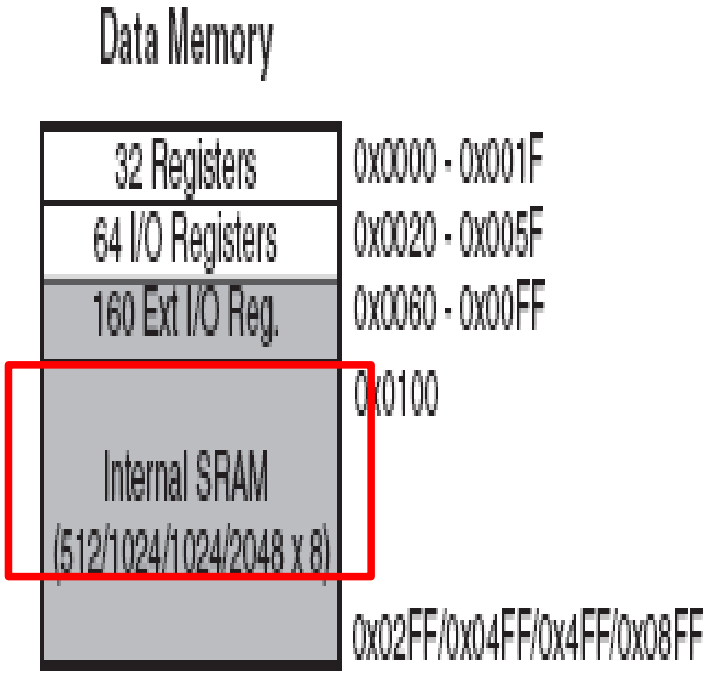
Procedimiento de creación de una matriz (I)

.equ tam = …… ;tamaño de la matriz

.dseg

.org $100 ;A partir de esta dirección

Matriz: .byte(tam) ;Reserva #tam bytes de la SRAM interna a partir de la dirección Matriz que es igual a $100



Sentencias de asignación XIV (acceso a matrices)

Procedimiento de creación de una matriz (II)

;Ahora hay que rellenar la matriz con datos (procedimiento manual)

LDI R16, dato0

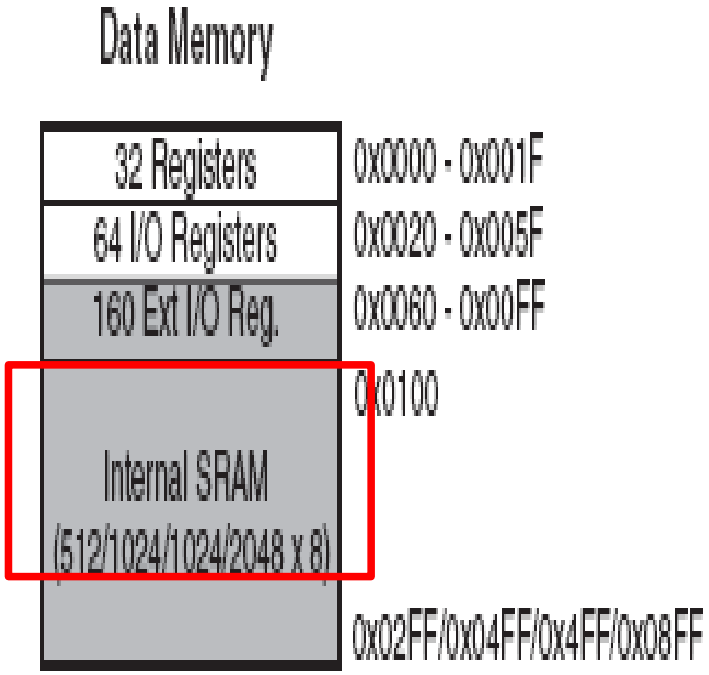
STS Matriz, R16

El ensamblador substituye Matriz por $100

LDI R16, dato1

STS Matriz+1,R16

El ensamblador substituye Matriz+1 por $101

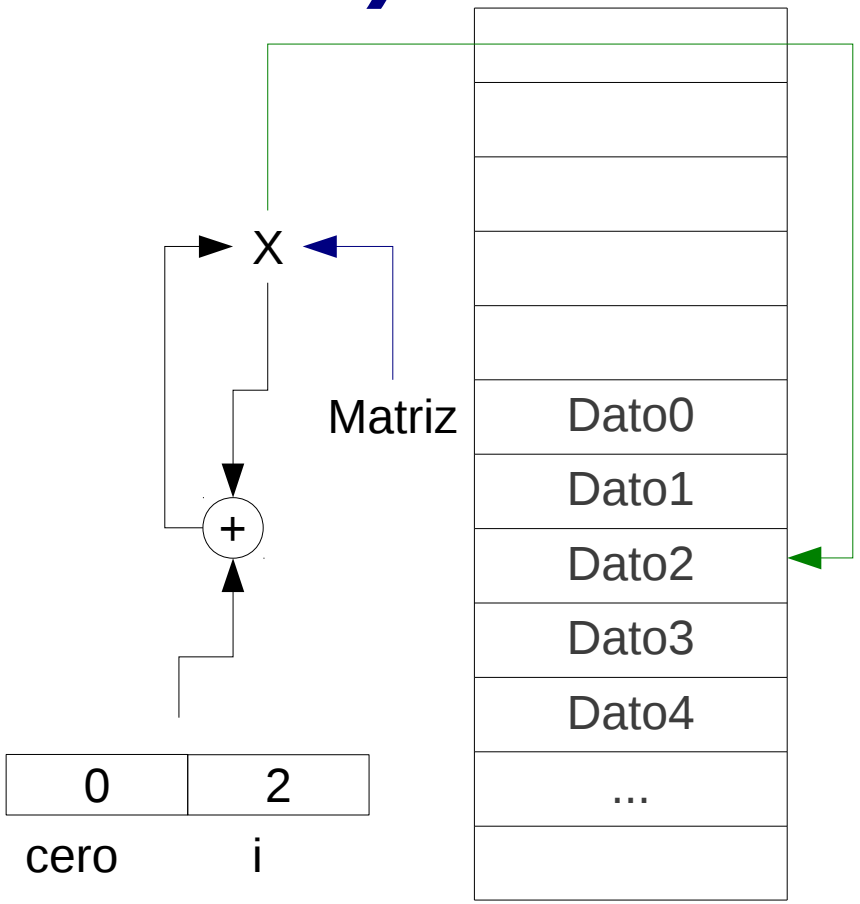


Sentencias de asignación XV (acceso a matrices)

Acceso al elemento Matriz[vble1]

Ej.: a ← Matriz[i]

```
.def i = R3
.def a = R1
.def cero = R0
CLR R0
LDI XL, low(Matriz)
LDI XH,high(Matriz)
ADD XL,i
ADC XH,cero
LD  a, X
```

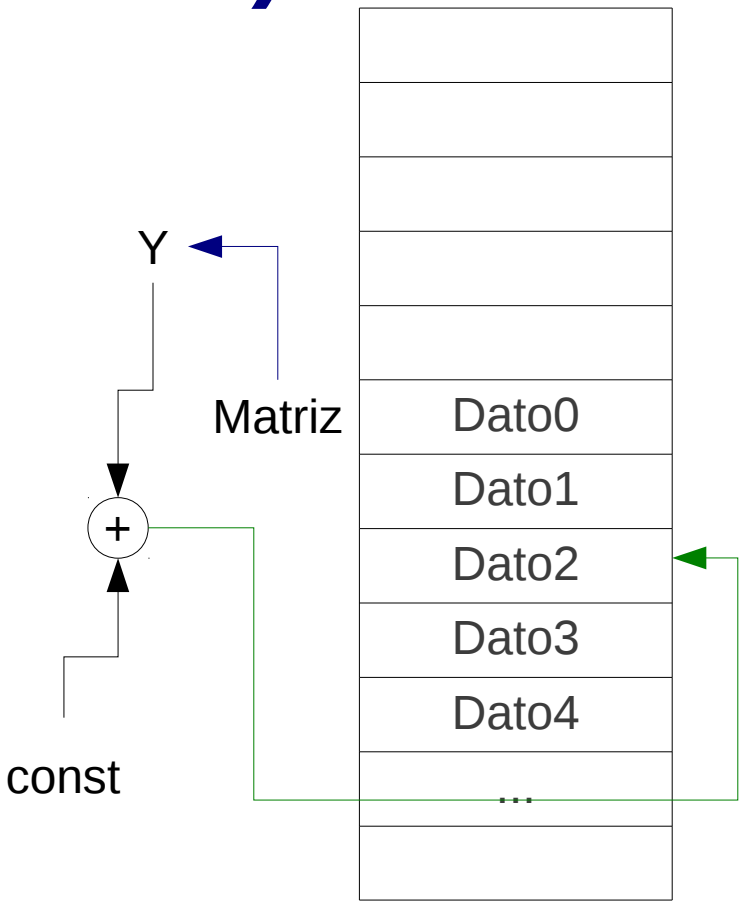


Sentencias de asignación XVI (acceso a matrices)

Acceso al elemento Matriz[const]

Ej.: Matriz[2] ← A

```
.equ const = 2  ;[Entre 0 y 63]
.def A =R2
LDI YL, low(Matriz)
LDI YH,high(Matriz)
STD Y+const,A
```



Índice

1. Pseudocódigo / Lenguaje Ensamblador
2. Sentencias de asignación
3. Sentencias de control
4. Bucles
5. Subrutinas
6. Interrupciones

Sentencias de control (I)

```
si (condición)

      ……. ; cuerpo para condición
            ;cierta

sino

      ……. ; para condición falsa

fsi
```

Tipos de condiciones:

A == B (igualdad)

A != B ó A <> B (desigualdad)

A > B (A mayor que B)

A >= B (A mayor o igual que B)

A < B (A menor que B)

A <= B (A menor o igual que B)

A(b) (consulta el bit b de la variable A)

A y B pueden ser:
- ambas variables
- una variable y una constante

A(b) es el bit de la variable A

Sentencias de control (II)

```
Si ( condición)

    ;{cuerpo cierto}

Sino

    ;{cuerpo falso}

fsi
```

```
      CP  A,B
      Brxx  Si
Sino:
            ; {cuerpo falso}
      JMP fsi
Si:
            ;{cuerpo cierto}
Fsi: ...  ;Instrucción siguiente
```

A = Rd
B = Rr

Si A o B son variables de la SRAM, deberemos moverlas a la zona de registros

Test (CP Rd,Rr)	Booleana	Mnemonico	Comentario
Rd ≥Rr	(N ⊕ V) = 0	BRGE	Signo
Rd < Rr	(N ⊕ V) = 1	BRLT	Signo
Rd = Rr	Z = 1	BREQ	Signo/Sin signo
Rd ≠ Rr	Z = 0	BRNE	Signo/Sin signo
Rd ≥ Rr	C = 0	BRCC/BRSH	Sin signo
Rd < Rr	C = 1	BRCS/BRLO	Sin signo
Carry	C=1	BRCS	Simple
Sin carry	C=0	BRCC	Simple
Negativo	N=1	BRMI	Simple
Positivo	N=0	BRPL	Simple
Overflow	V=1	BRVS	Simple
Sin overflow	V=0	BRVC	Simple
Cero	Z=1	BREQ	Simple
No cero	Z=0	BRNE	Simple

ES IMPORTANTE TENER EN CUENTA SI LAS VARIABLES A COMPARAR SON CON SIGNO O SIN SIGNO

Sentencias de control (III)

```
Si ( condición)

    ;{cuerpo cierto}

Sino

    ;{cuerpo falso}

fsi
```

```
        CPI  A,cte
        Brxx  Si
Sino:
              ; {cuerpo falso}
        JMP fsi
Si:
              ;{cuerpo cierto}
Fsi: ...  ;Instrucción siguiente
```

A = Rd ;d=[16,31]
B = Cte ;de 8 bits

Test (CP Rd,Rr)	Booleana	Mnemonico	Comentario
Rd ≥Rr	(N ⊕ V) = 0	BRGE	Signo
Rd < Rr	(N ⊕ V) = 1	BRLT	Signo
Rd = Rr	Z = 1	BREQ	Signo/Sin signo
Rd ≠ Rr	Z = 0	BRNE	Signo/Sin signo
Rd ≥ Rr	C = 0	BRCC/BRSH	Sin signo
Rd < Rr	C = 1	BRCS/BRLO	Sin signo
Carry	C=1	BRCS	Simple
Sin carry	C=0	BRCC	Simple
Negativo	N=1	BRMI	Simple
Positivo	N=0	BRPL	Simple
Overflow	V=1	BRVS	Simple
Sin overflow	V=0	BRVC	Simple
Cero	Z=1	BREQ	Simple
No cero	Z=0	BRNE	Simple

ES IMPORTANTE TENER EN CUENTA SI LAS VARIABLES A COMPARAR SON CON SIGNO O SIN SIGNO

Índice

1. Pseudocódigo / Lenguaje Ensamblador
2. Sentencias de asignación
3. Sentencias de control
4. Bucles
5. Subrutinas
6. Interrupciones

Bucles (I)

A = Rd
B = Rr

Mientras (condición)

; { cuerpo bucle}

Fin mientras

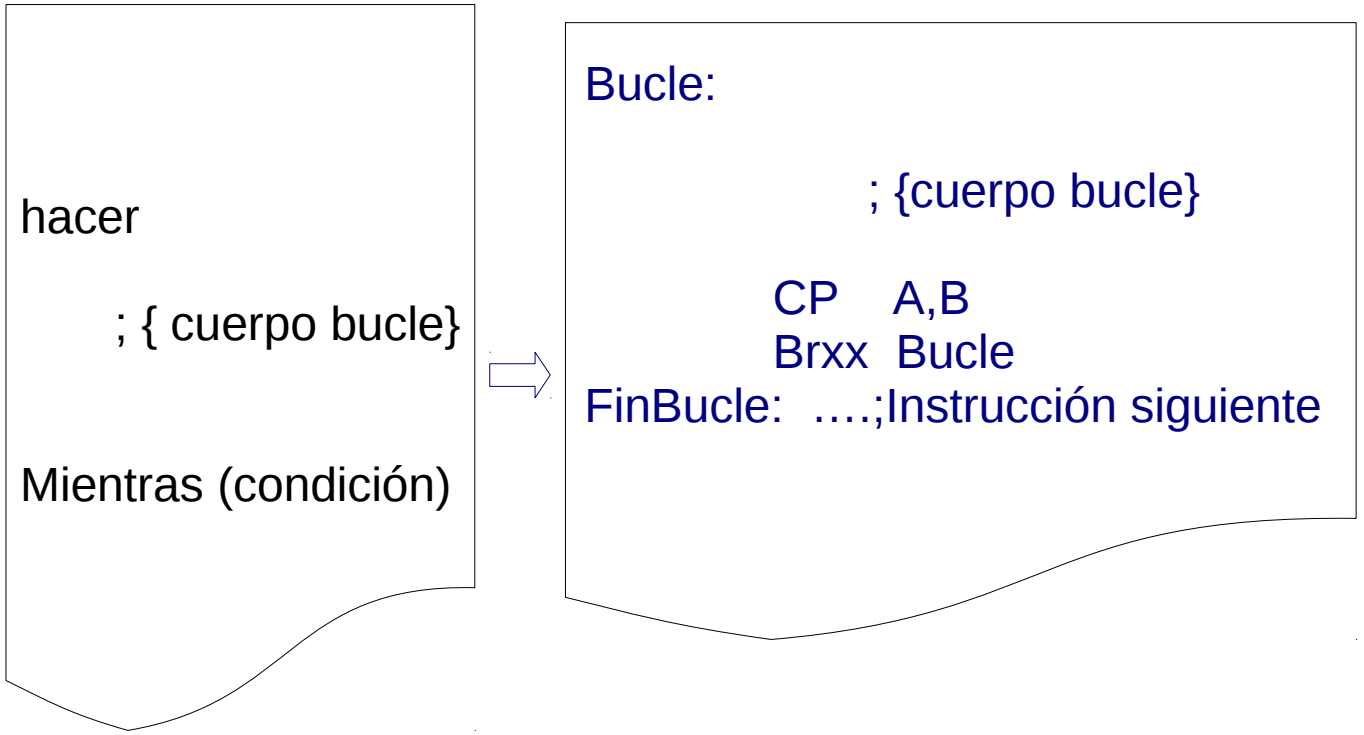
```
Bucle:   CP    A,B
         Brxx  FinBucle
               ; {cuerpo bucle}
               JMP bucle
FinBucle: ….;Instrucción siguiente
```

Condición opuesta

Test (CP Rd,Rr)	Booleana	Mnemonico	Comentario
Rd ≥Rr	(N ⊕ V) = 0	BRGE	Signo
Rd < Rr	(N ⊕ V) = 1	BRLT	Signo
Rd = Rr	Z = 1	BREQ	Signo/Sin signo
Rd ≠ Rr	Z = 0	BRNE	Signo/Sin signo
Rd ≥ Rr	C = 0	BRCC/BRSH	Sin signo
Rd < Rr	C = 1	BRCS/BRLO	Sin signo
Carry	C=1	BRCS	Simple
Sin carry	C=0	BRCC	Simple
Negativo	N=1	BRMI	Simple
Positivo	N=0	BRPL	Simple
Overflow	V=1	BRVS	Simple
Sin overflow	V=0	BRVC	Simple
Cero	Z=1	BREQ	Simple
No cero	Z=0	BRNE	Simple

EL CUERPO DEL BUCLE SE EJECUTA SIEMPRE QUE LA CONDICIÓN SEA QUE SE EVALÚA SEA CIERTA

Bucles (II)



A = Rd
B = Rr

EL CUERPO DE UN HACER MIENTRAS SE EJECUTA AL MENOS UNA VEZ
Y SE REPITE SIEMPRE QUE LA CONDICIÓN EVALUADA SEA CIERTA.

Índice

1. Pseudocódigo / Lenguaje Ensamblador
2. Sentencias de asignación
3. Sentencias de control
4. Bucles
5. Subrutinas
6. Interrupciones

Subrutinas (I)

Subr Nombre

;{cuerpo subrutina}

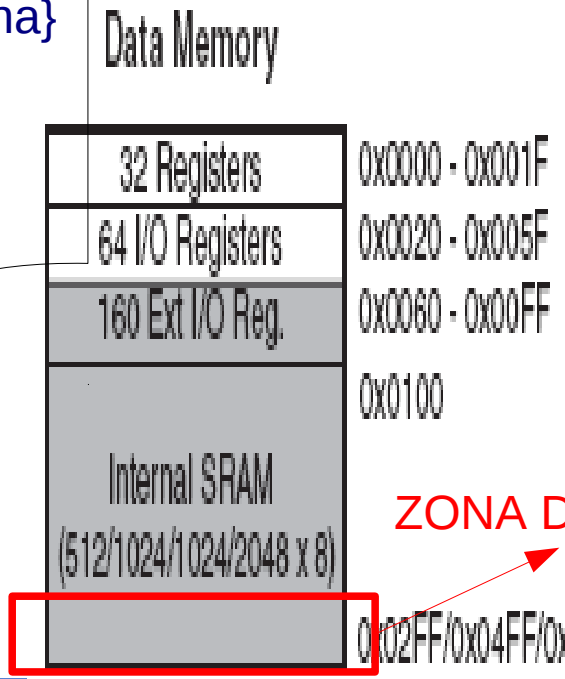
Fin Nombre

⇨

Nombre:

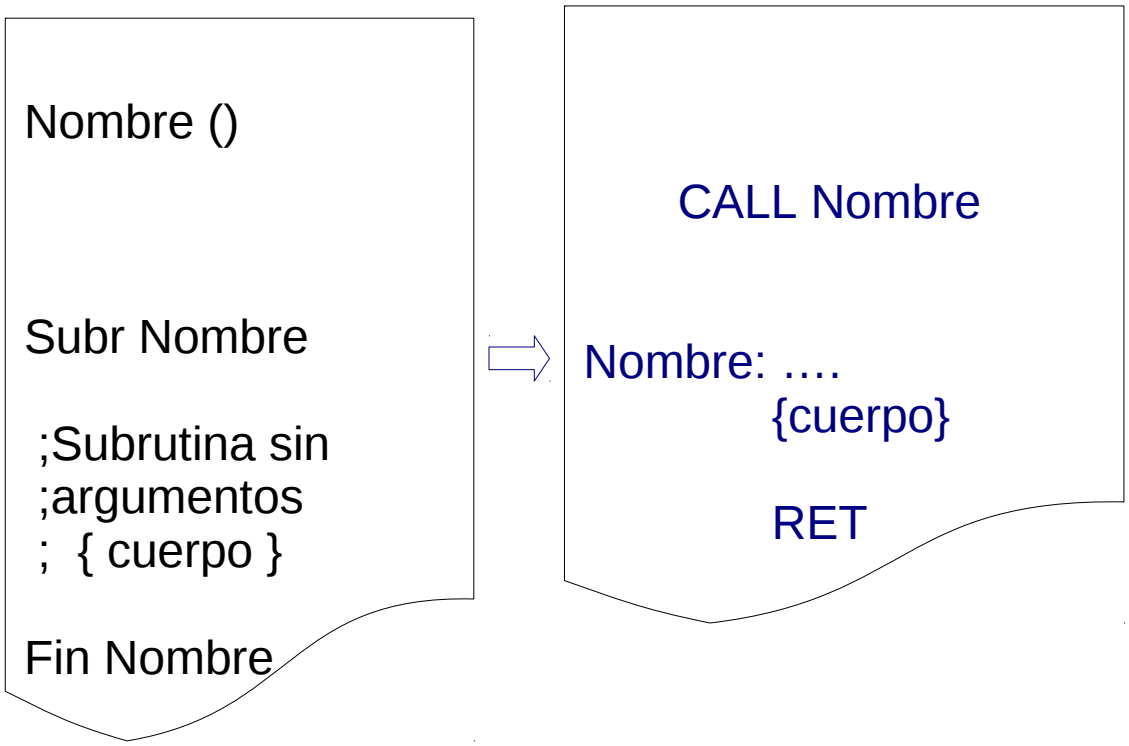
; {cuerpo subrutina}

FinNombre: RET

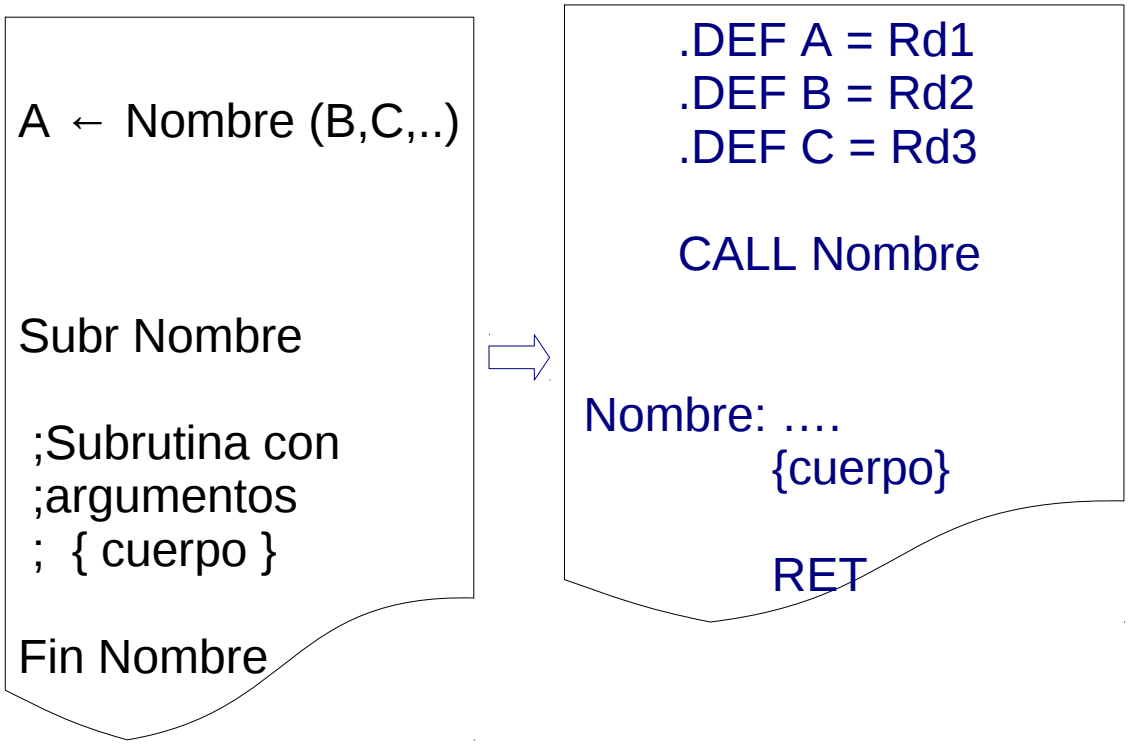


- EL PUNTERO SP ESTÁ INICIADO A RAMEND
- EVITAR USAR LA ZONA ALTA DE LA SRAM PARA DATOS DEL PROGRAMA.

Subrutinas (II)
Llamada a una subrutina

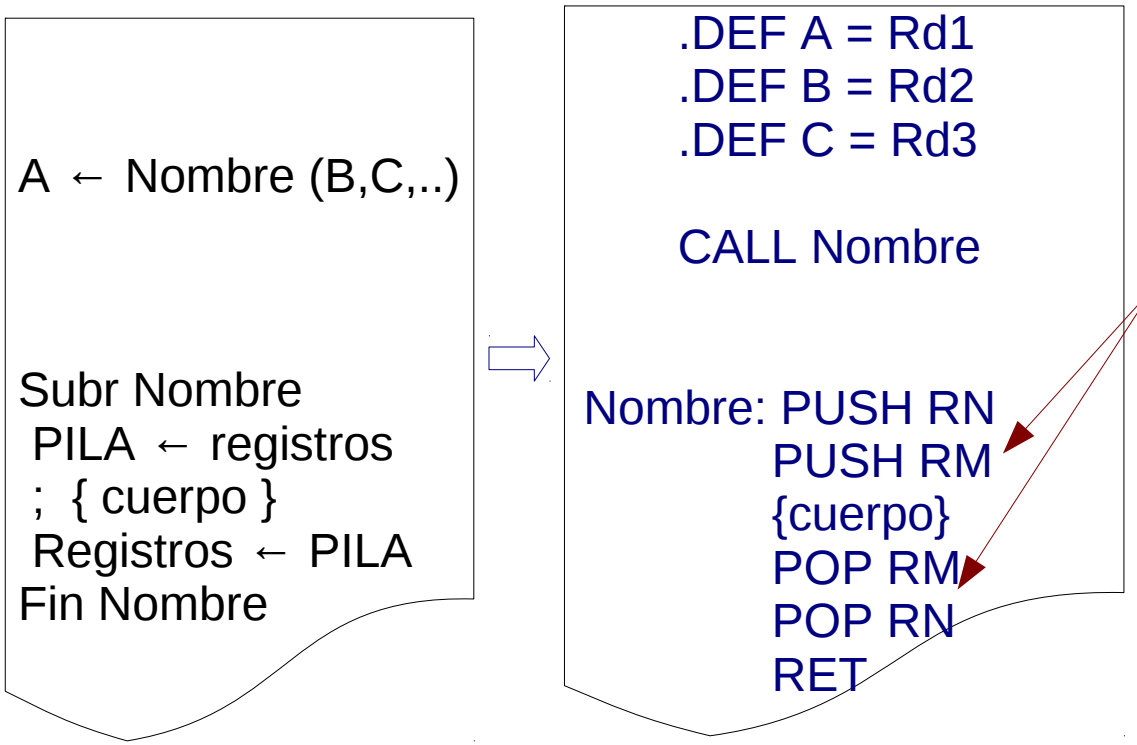


Subrutinas (III)
Llamada a una subrutina



UTILIZAMOS REGISTROS PARA ALMACENAR LOS RESULTADOS Y LOS ARGUMENTOS DE LA SUBRUTINA

Subrutinas (IV)
Llamada a una subrutina

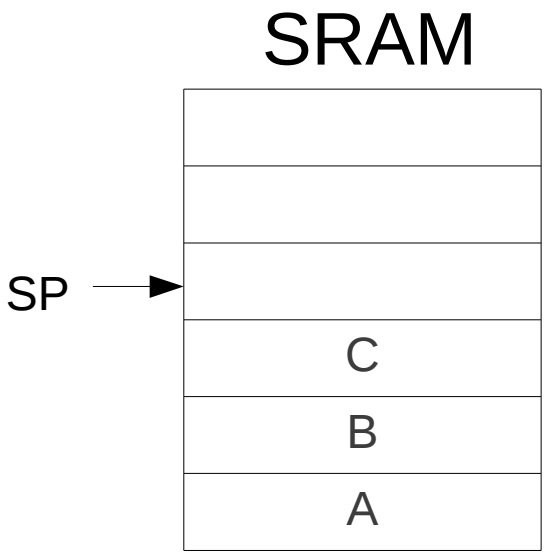


CONVIENE GUARDAR EL CONTENIDO DE LOS REGISTROS QUE VAYAN A SER MODIFICADOS POR LA SUBRUTINA. ¿DÓNDE? → EN LA **PILA**

Subrutinas (V)

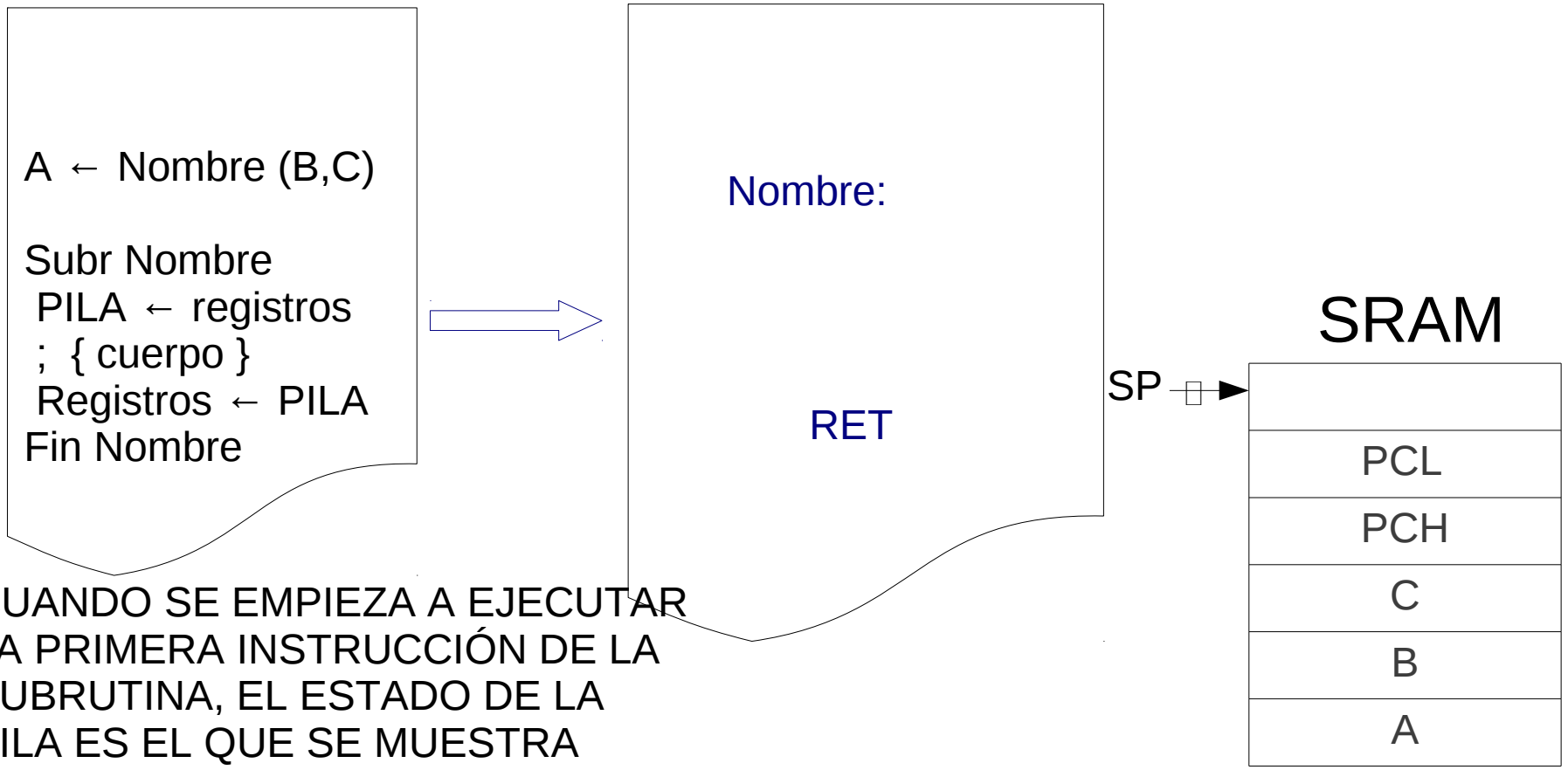
```
A ← Nombre (B,C)

Subr Nombre
 PILA ← registros
 ; { cuerpo }
 Registros ← PILA
Fin Nombre
```

```
PUSH A
PUSH B
PUSH C
CALL Nombre
POP C
POP B
POP A
```



ES MÁS HABITUAL PASAR LOS ARGUMENTOS Y LA SALIDA DE UNA SUBRUTINA A TRAVÉS DE LA PILA

Subrutinas (VI)

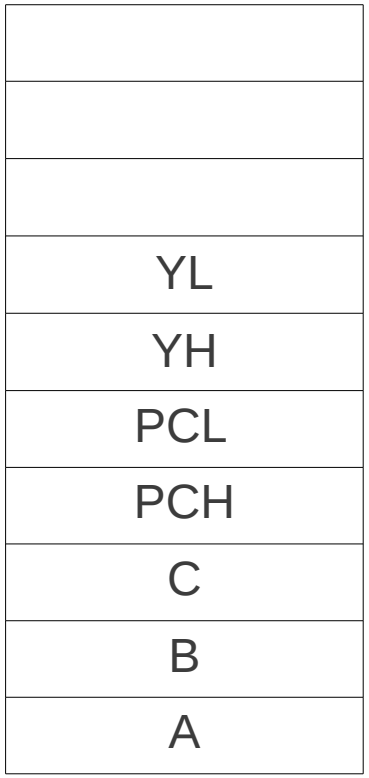


CUANDO SE EMPIEZA A EJECUTAR LA PRIMERA INSTRUCCIÓN DE LA SUBRUTINA, EL ESTADO DE LA PILA ES EL QUE SE MUESTRA

Subrutinas (VI)

```
A ← Nombre (B,C)

Subr Nombre
 PILA ← registros
 ; { cuerpo }
 Registros ← PILA
Fin Nombre
```

```
Nombre:
        PUSH YH
        PUSH YL
        IN  YL,SPL
        IN YH,SPH



        POP YL
        POP YH
        RET
```



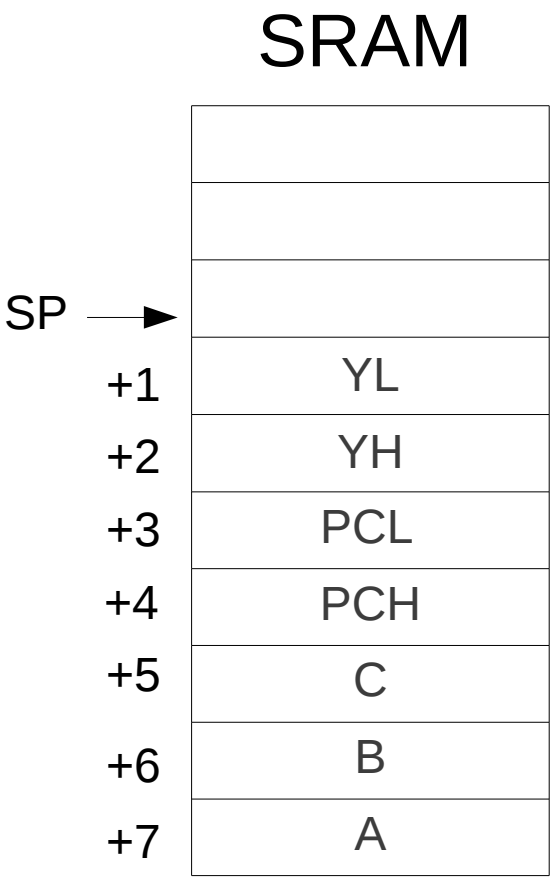
COPIAMOS EL SP A UN REGISTRO Y, Z (PERO PREVIAMENTE SE GUARDA)

Subrutinas (VI)

A ← Nombre (B,C)

Subr Nombre
 PILA ← registros
 ; { cuerpo }
 Registros ← PILA
Fin Nombre

```
Nombre:
        PUSH YH
        PUSH YL
        IN  YL,SPL
        IN YH,SPH

        ;{cuerpo}

        POP YL
        POP YH
        RET
```

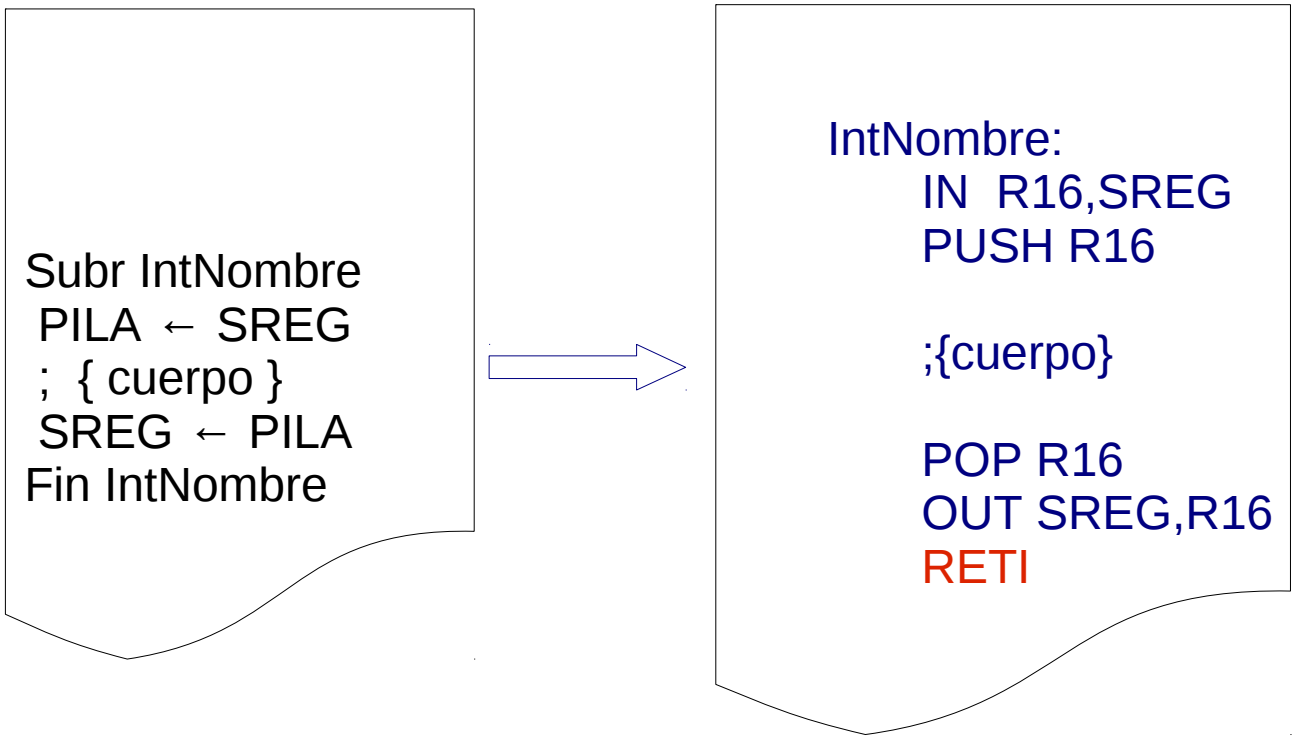


PARA LEER EL DATO C USAREMOS: LDD Y+5
PARA LEER EL DATO B USAREMOS: LDD Y+6
PARA MODIFICAR EL RESULTADO A: STD Y+7

Índice

1. Pseudocódigo / Lenguaje Ensamblador
2. Sentencias de asignación
3. Sentencias de control
4. Bucles
5. Subrutinas
6. Interrupciones

Interrupciones (I)



IDÉNTICO FORMATO QUE PARA LAS SUBRUTINAS. EN ENSAMBLADOR SE ESCRIBE RETI AL FINAL EN LUGAR DE RET. NO OLVIDAR SALVAR EL SREG

Interrupciones (II)

```
Subr IntNombre
 PILA ← SREG
 ; { cuerpo }
 SREG ← PILA
Fin IntNombre
```

```
        .CSEG
        .ORG VectInt
        JMP IntNombre
        ……..
IntNombre:
                IN  R16,SREG
                PUSH R16

                ;{cuerpo}

                POP R16
                OUT SREG,R16
                RETI
```

LAS RUTINAS DE INTERRUPCIÓN TIENEN QUE ESTAR “INSTALADAS” EN SU RESPECTIVOS VECTORES DE INTERRUPCIÓN.