Programación en VCore - Parte 1

Aspectos básicos del procesador y la computadora

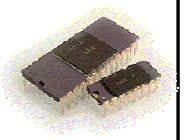
- La memoria de computador en detalle.
- El uso de la pila para almacenar datos.
- Paso de argumentos a funciones.
- Acceso a periféricos por medio de puertos.
- Mecanismos de espera activa para la pantalla y el teclado.

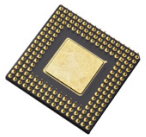
La memoria de VCore en detalle

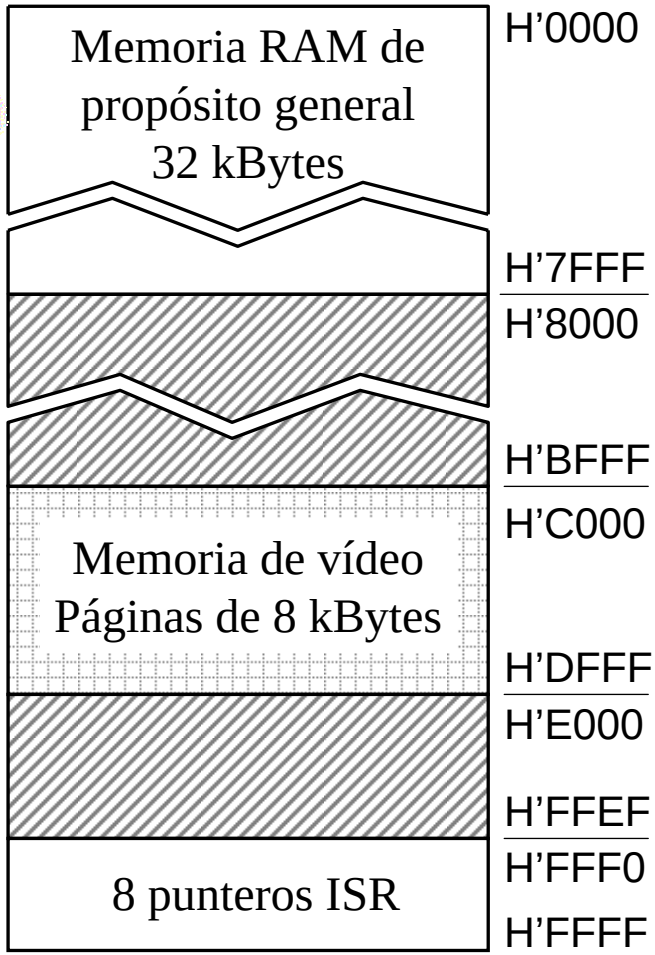
- El programa debe estar almacenado en los primeros 32 kbytes del espacio de direccionamiento de VCore.
- Un acceso a zonas de memoria sin asignación produce una interrupción de "fallo de acceso a memoria" (*memory fault*).
- El acceso a memoria puede realizarse en modo 8 o 16 bits:

```
ld    .1,/Variable16bit
ld.b  .2,/Variable8bit
```

- Los vectores de interrupción y la memoria de video están mapeados en el espacio de direcciones de VCore.







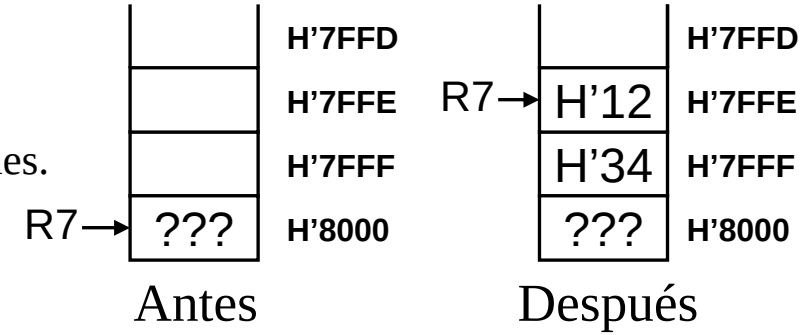
La pila: introducción de valores

- Introducir en la pila un valor de 16 bits

`push #H'1234`

Secuencia de ejecución de la instrucción:

1. Decrementar puntero de pila (R7) dos posiciones.
   $R7 \leftarrow (R7) - 2$
2. Almacenar contenido en la pila.
   $MP[(R7)]_{16bits} \leftarrow H'1234$

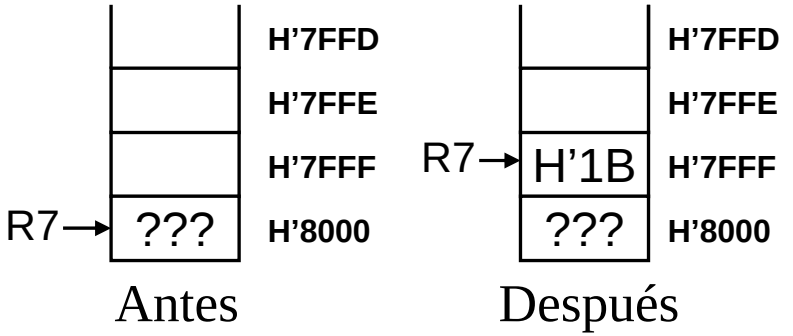


- Introducir en la pila un valor de 8 bits

`push.b #H'1B`

Secuencia de ejecución de la instrucción:

1. Decrementar puntero de pila (R7) una posición.
   $R7 \leftarrow (R7) - 1$
2. Almacenar contenido en la pila.
   $MP[(R7)]_{8bits} \leftarrow H'1B$



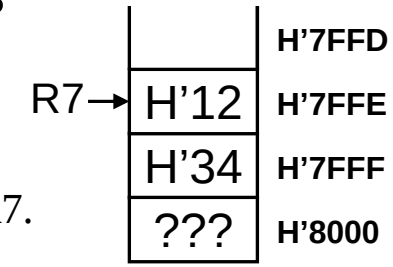
La pila: extracción de valores

- Extraer en la pila un valor de 16 bits

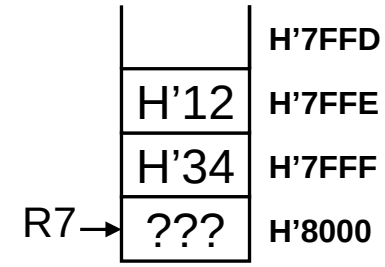
`pop .1`

Secuencia de ejecución de la instrucción:

1. Leer 16 bits memoria cuya dirección esta en R7.
   $R1 \leftarrow (MP[(R7)])_{16bits}$
2. Incrementar R7 en dos posiciones (16 bits).
   $R7 \leftarrow (R7) + 2$



Antes



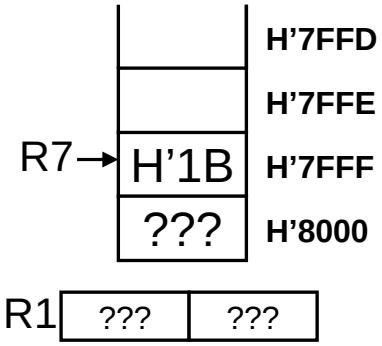
Después

- Extraer en la pila un valor de 8 bits

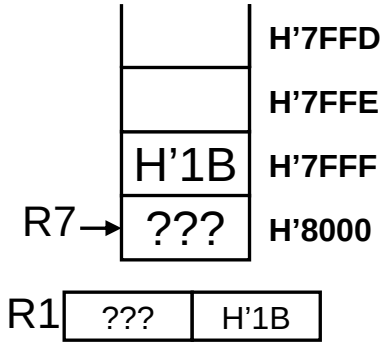
`pop.b .1`

Secuencia de ejecución de la instrucción:

1. Leer 8 bits memoria cuya dirección esta en R7.
   $R1 \leftarrow (MP[(R7)])_{8bits}$
2. Incrementar R7 en una posición (8 bits).
   $R7 \leftarrow (R7) + 1$



Antes



Después

La pila: llamadas a funciones (subrutinas)

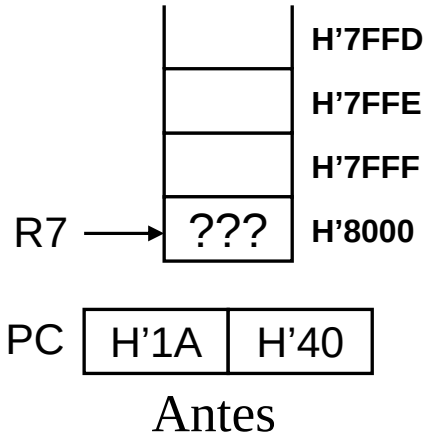
- Llamada a una subrutina

```
H'1A40: call #H'21B0
```

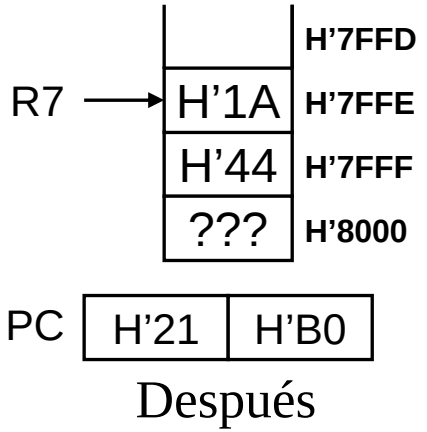
Secuencia de ejecución de la instrucción:

1. Calcular siguiente valor contador programa (PC)
   PC ← (PC) + 4
2. Decrementar puntero de pila (R7) dos posiciones.
   R7 ← (R7) – 2
3. Almacenar contador de programa en pila
   $MP[(R7)]_{16bits} \leftarrow (PC)$
4. Saltar al valor especificado en la instrucción
   PC ← (CD) en este caso por ser dir. inmediato.

La posición de la siguiente instrucción después de **`call`** queda almacenada en la pila para poder ser recuperada después con la instrucción **`rts`**.



Antes



Después

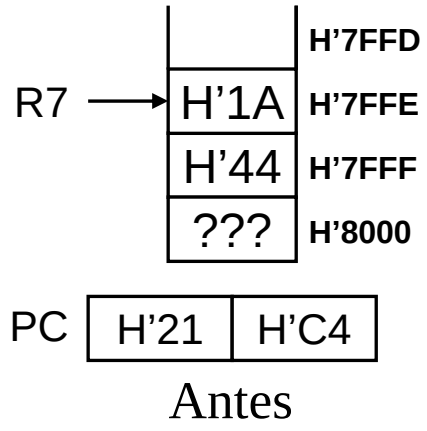
La pila: retorno de funciones (subrutinas)

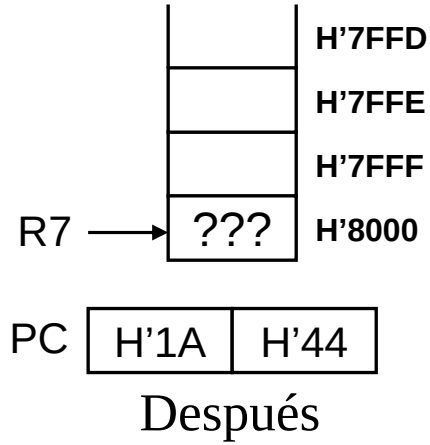
- Retorno de una subrutina

`H'21C4: rts`

Secuencia de ejecución de la instrucción **`rts`**:

1. Extrae un valor de 16 bits de la pila y lo carga en PC
   $PC \leftarrow (MP[(R7)])_{16bits}$
2. Incrementa el puntero de pila (R7) dos posiciones.
   $R7 \leftarrow (R7) + 2$





Ejemplo de llamada a una función (también llamada subrutina):

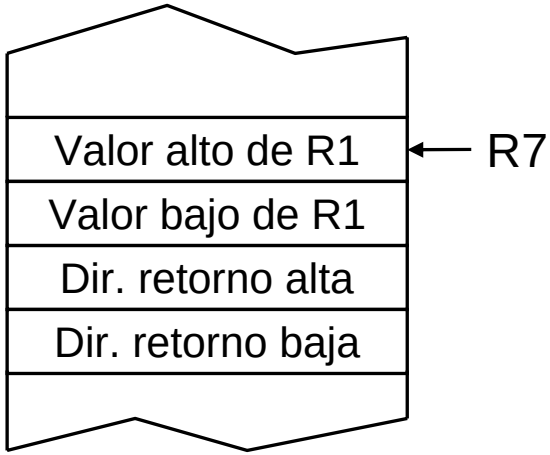
```
    ...
    ld  .1,#17          ; Carga sumando 1
    ld  .2,#25          ; Carga sumando 2
    call SumaDosValores ; Llama a función suma
    st  .1,/Resultado   ; Guarda resultado
    ...
SumaDosValores:         ; Comienzo de la función
    add .1,.2           ; Suma los dos valores
    rts                 ; Retorna a llamador
```

Ejemplo del uso de la pila para guardar datos 1/2

Multiplicación por 10 modificando solo el registro R0

mult10.asm

```
; Función: Mult10
; Descripción: multiplica un número
;   por 10 y devuelve el resultado.
; Parámetros:
;   R0: número a multiplicar
; Retorna:
;   R0: resultado de la multiplicación
Mult10:
       push   .1     ; guarda R1
       asl    .0     ; R0 = n x 2
       ld     .1,.0  ; R1 = R0
       asl    .1,#2  ; R1 = n x 8
       add    .0,.1  ; R0 = n x 10
       pop    .1     ; recupera R1
       rts           ; retorna a llamador
       END
```



Ejemplo del uso de la pila para guardar datos 2/2

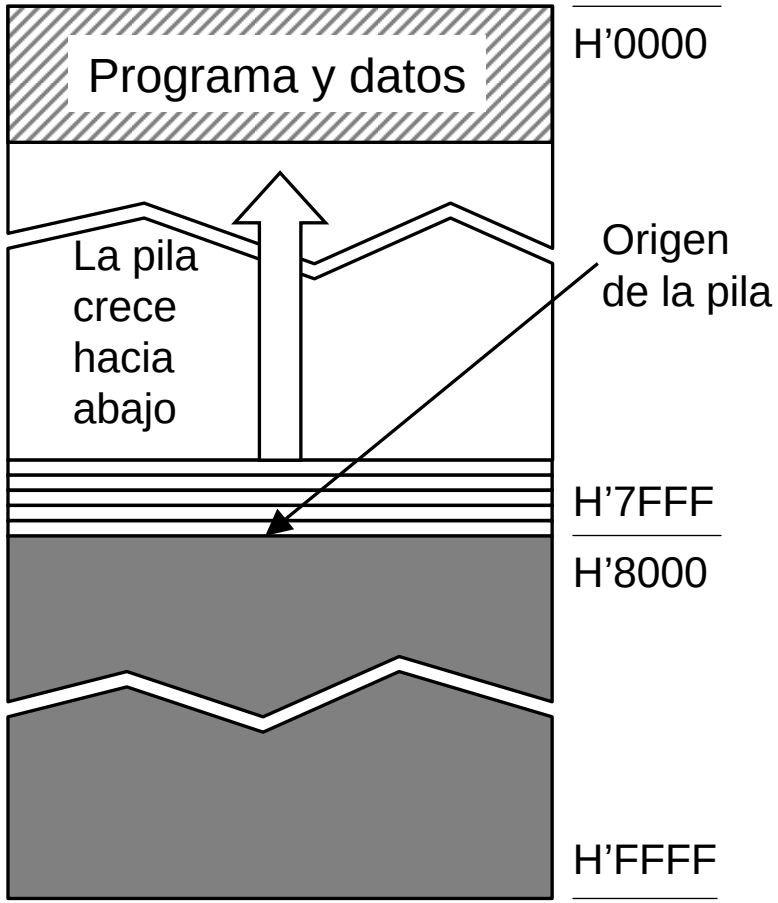
Programa llamador completo para multiplicar una tabla de datos

multlista.asm

```
         ORG    H'100             ; Origen de ensamblado.
Inicio:  LD     .7,#H'8000        ; Carga valor inicial puntero pila
         LD     .1,#TablaOrigen   ; Carga valor inicial tabla origen
         LD     .2,#TablaDestino  ; Carga valor inicial tabla destino
         LD     .3,#100           ; Carga número de elementos
Bucle:   LD     .0,[.1++]         ; Lee un elemento en R0 (parámetro)
         CALL   #Mult10           ; Multiplica elemento leído por 10
         ST     .0,[.2++]         ; Almacena resultado multiplicación
         DEC    .3                ; Decrementa el número de elementos
         BNZ    #Bucle            ; restante y si no es cero continua.
         HALT                     ; La tabla se ha tratado: detención.

         INCLUDE  "mult10.asm"     ; Incluye código fichero mult10.asm

TablaOrigen:    DW  50 Dup(4,7)   ; Reserva espacio para tabla origen.
TablaDestino:   DW  100 Dup(?)    ; Reserva espacio para tabla destino.

         ORG    H'FFFE            ; Coloca el vector de Reset al
Reset:   DW     Inicio            ; comienzo de nuestro programa.
         END                      ; Final del fichero fuente.
```

Consejos para el uso de la pila

- La pila decrece en posiciones, por lo que conviene colocar su origen al final de la memoria (R7 = H'8000)
- Un anidamiento muy profundo en llamadas a subrutinas o un almacenamiento excesivo puede desbordar el espacio reservado a la pila y sobreescribir zonas de programa o datos (*stack overflow*).
- El uso de la pila es lento ya que exige realizar accesos a memoria. Se debe usar la pila solo cuando es necesario.



Paso de argumentos a funciones

Paso por memoria

```
  st   .1,/VarComp
  call Funcion
     …
VarComp: dw ?
Funcion:
  ld   .3,/VarComp
     …
  rts
```

Ventajas:
- Gran capacidad

Inconvenientes:
- Lento: usa la memoria
- No es reentrante.

Generalmente desaconsejado

Paso por registro

```
  ld   .1,#1234
  ld   .2,#5678
  call Funcion
     …
Funcion:
  add  .1,.2
     …
  rts
```

Ventajas:
- Rapidez

Inconvenientes:
- Poca capacidad
- Reentrante usando pila

Función rápida con pocos argumentos de llamada

Paso por pila

```
  push  #1232
  call  Funcion
     …
Funcion:
  pop   .1
  add   .1,#47
     …
  rts
```

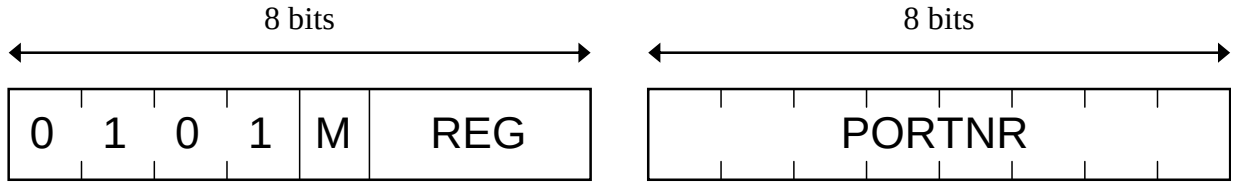
Ventajas:
- Gran capacidad
- Reentrante

Inconvenientes:
- Lento: usa la memoria

Funciones recursivas o con muchos argumentos

Acceso a periféricos por medio de puertos

Instrucciones de propósito específico IN y OUT



M: tipo de acceso a un puerto
- M = 0: **`IN`** : acceso de lectura
- M = 1: **`OUT`** : acceso de escritura

REG: número de registro (R0 a R7)

PORTNR: número de puerto (0 a 255)

Ejemplos:

```
in   .1,/160
out  .2,/193
```

Puertos	Descripción
160	Puerto de estado del teclado
161	Puerto de datos del teclado
180	Estado del temporizador
181	Contador del temporizador
192	Estado de la tarjeta gráfica
193	Escritura de carácteres
194	Selección de página de video
195,196	Posición X e Y del cursor
197,198	Color y fondo de carácter

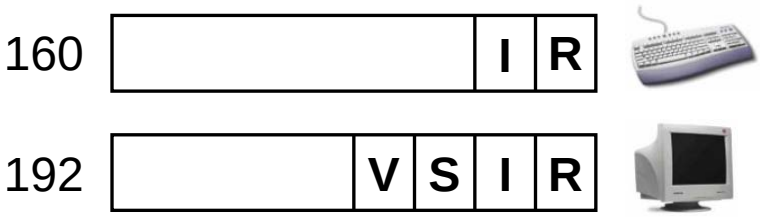
Atención: acceso a periféricos tamaño byte (8 bits)

Rutinas de espera activa para teclado y pantalla

espact.asm

```
;Función: EscCar
;Descripción: escribe carácter en pantalla
;Parámetros: R0: carácter a escribir
;Retorna: nada ; Modifica R1 !!!
EscCar:  in   .1,/192
         and  .1,#1
         bz   #EscCar
         out  .0,/193
         rts
;Función: LeeCar
;Descripción: lee carácter del teclado
;Parámetros: ninguno
;Retorna: R0: caracter leído del teclado
LeeCar:  in   .0,/160
         and  .0,#1
         bz   #LeeCar
         in   .0,/161
         rts
         END
```

Indicadores de los puertos de estado



prespact.asm

```
;Prueba rutinas de espera activa
      ORG H'100
INCLUDE "espact.asm"
Inicio:   ld   .7,#H'8000
Bucle:    call #LeeCar
      call    #EscCar
      br  #bucle
      ORG H'FFFE
      DW  Inicio
      END
```