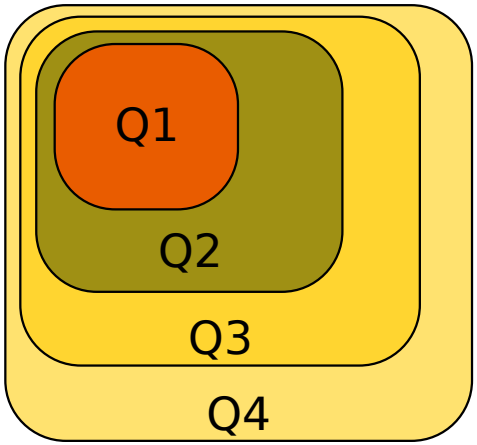
Modularización y reuso de código

Organización de computadoras

Universidad Nacional de Quilmes

http://orga.blog.unq.edu.ar

Repaso



1. Máscaras
2. Repeticiones
3. Arreglos
4. **Q4**
   1. Modo de direccionamiento indirecto (y registro indirecto)
   2. Operaciones lógicas
5. ¿Pila?

Repaso: Máscaras

Repaso: Repeticiones

Example (¿Cómo encender el piloto del calefón?)

1. poner la perilla en posición piloto
2. acercar un fósforo mientras se presiona la perilla
3. mantener presionando aproximadamente 20 segundos
4. Si al liberar la perilla el piloto se apaga, volver al paso (1), sino seguir con el paso (5)
5. ...

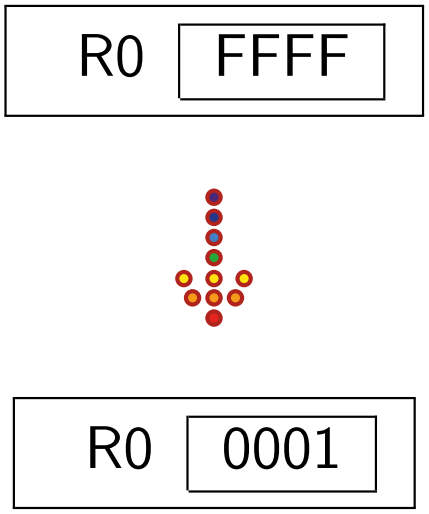
Repaso: Arreglos

⋮	⋮
000A	1er valor
000B	2do valor
000C	3er valor
000D	4to valor
000E	5to valor
000F	6to valor
⋮	⋮

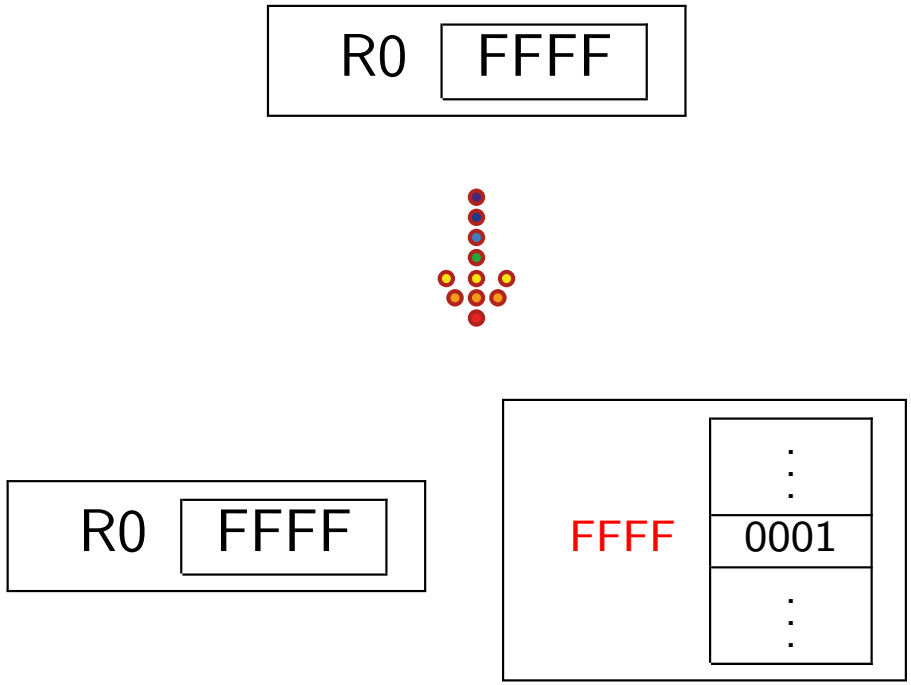
Arreglo de valores
Posiciones de memoria consecutivas que contienen una colección de elementos. Cada elemento puede ocupar mas de una celda.

Repaso: Modo indirecto

MOV R0,0x0001

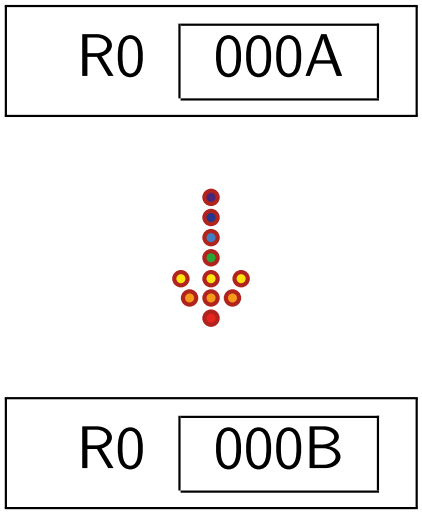


MOV [R0],0x0001

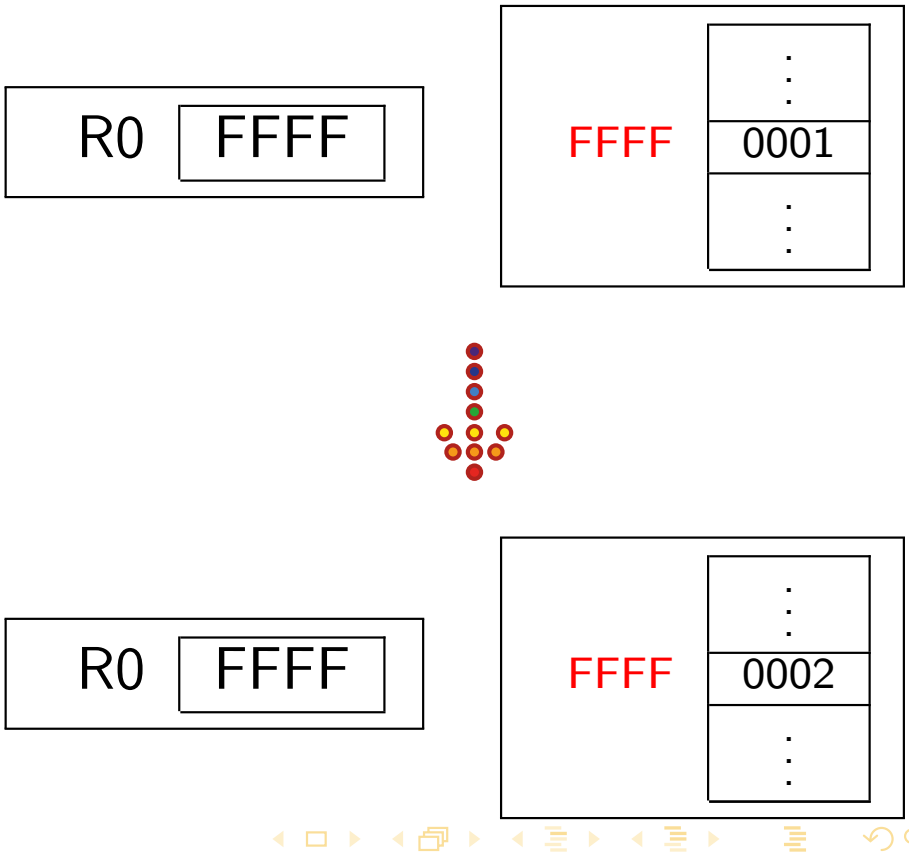


Repaso: Modo indirecto

ADD R0,0x0001



ADD [R0],0x0001



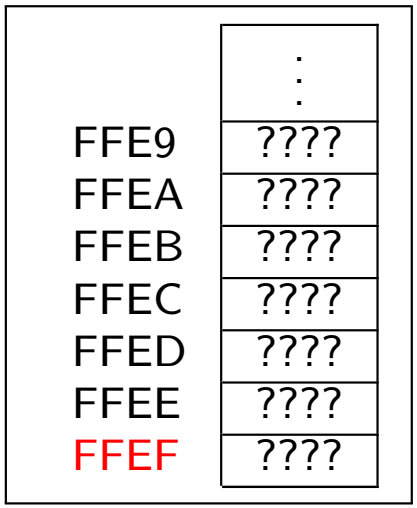
La Pila

Estructura de Pila (Stack)

- La pila es un sector especial de la memoria
- Los datos se organizan *apilados*:
  1. Cuando se escribe en la pila, se lo agrega "sobre" el último agregado
  2. Cuando se lee de la pila, se lo saca del "tope" de la pila
- El seguimiento del tope de pila se lleva mediante un registro especial SP (Stack Pointer)

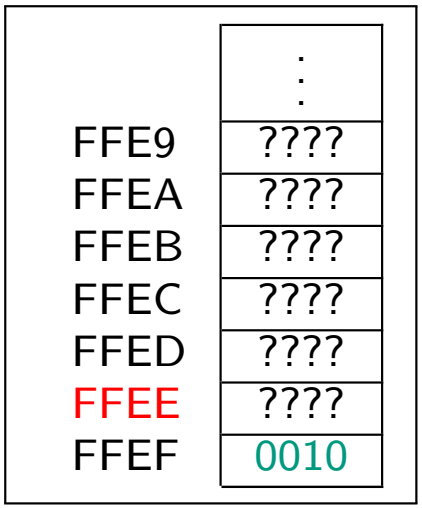
Funcionamiento de la pila

1. Estado original: Tope de pila en FFEF



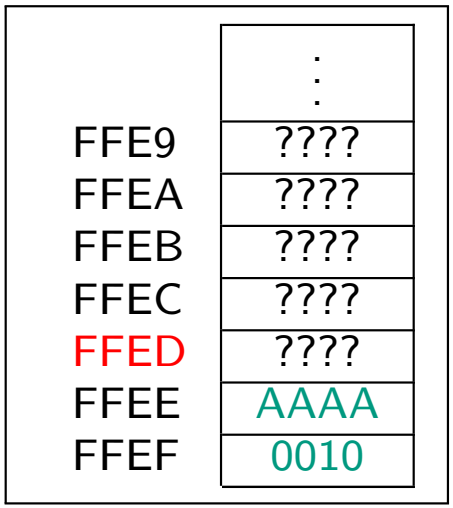
Funcionamiento de la pila

1. Estado original: Tope de pila en FFEF
2. Apilar elemento: 0010. Tope de pila en FFEE



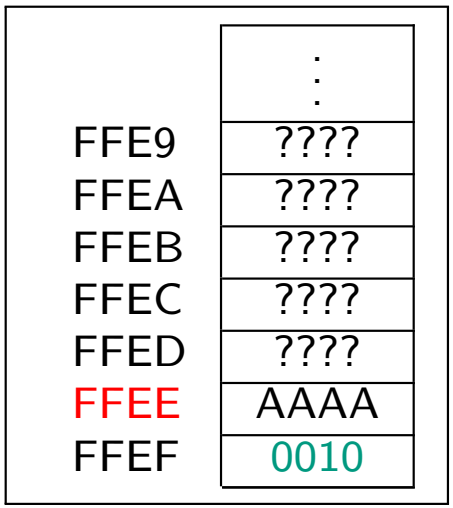
Funcionamiento de la pila

1. Estado original: Tope de pila en FFEF
2. Apilar elemento: 0010. Tope de pila en FFEE
3. Apilar elemento: AAAA. Tope de pila en FFED



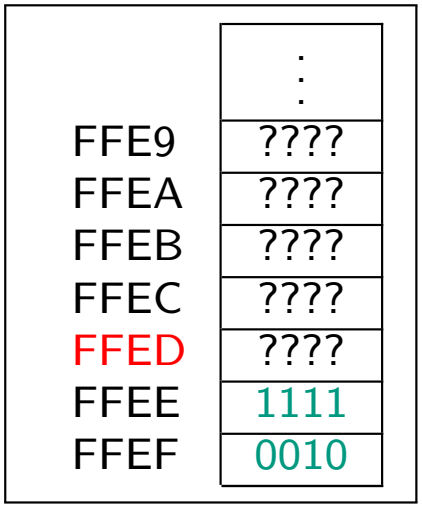
Funcionamiento de la pila

1. Estado original: Tope de pila en FFEF
2. Apilar elemento: 0010. Tope de pila en FFEE
3. Apilar elemento: AAAA. Tope de pila en FFED
4. Desapilar elemento. Tope de pila en FFEE



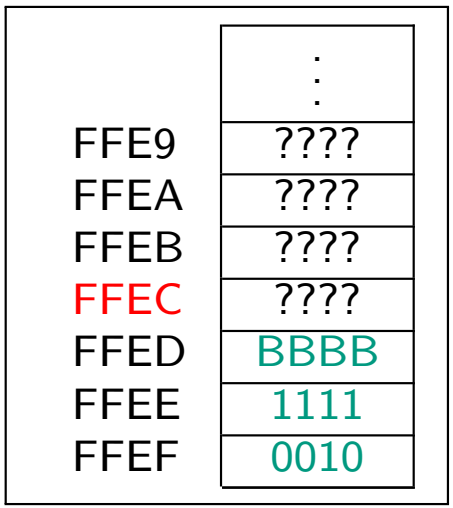
Funcionamiento de la pila

1. Estado original: Tope de pila en FFEF
2. Apilar elemento: 0010. Tope de pila en FFEE
3. Apilar elemento: AAAA. Tope de pila en FFED
4. Desapilar elemento. Tope de pila en FFEE
5. Apilar elemento: 1111. Tope de pila en FFED

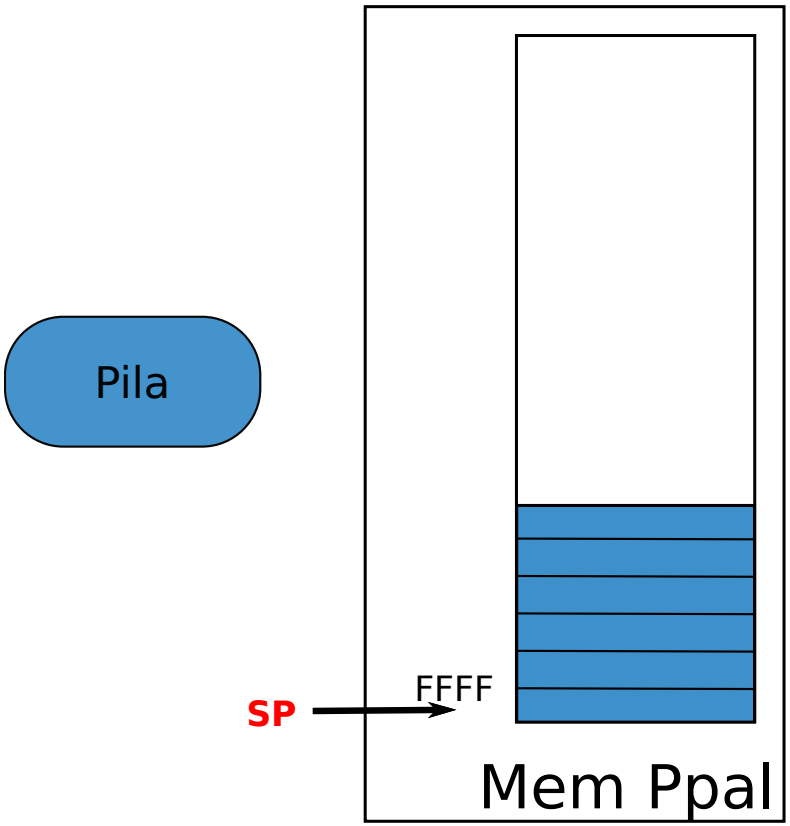


Funcionamiento de la pila

1. Estado original: Tope de pila en FFEF
2. Apilar elemento: 0010. Tope de pila en FFEE
3. Apilar elemento: AAAA. Tope de pila en FFED
4. Desapilar elemento. Tope de pila en FFEE
5. Apilar elemento: 1111. Tope de pila en FFED
6. Apilar elemento: BBBB. Tope de pila en FFEC



Implementación de Pila



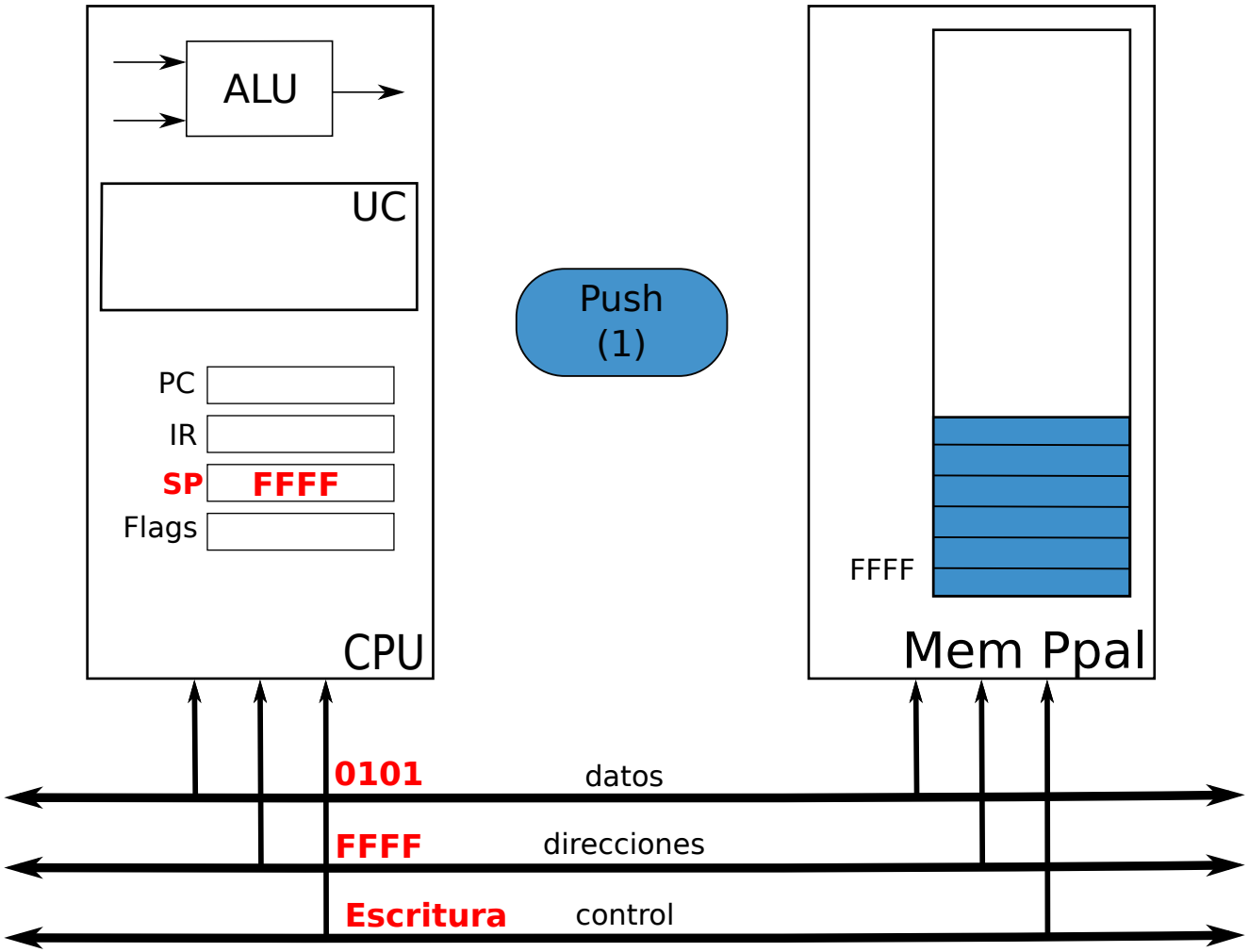
SP (Stack Pointer) contiene la dirección de la primer celda de memoria **disponible** de la pila.

Estructura de Pila: Push

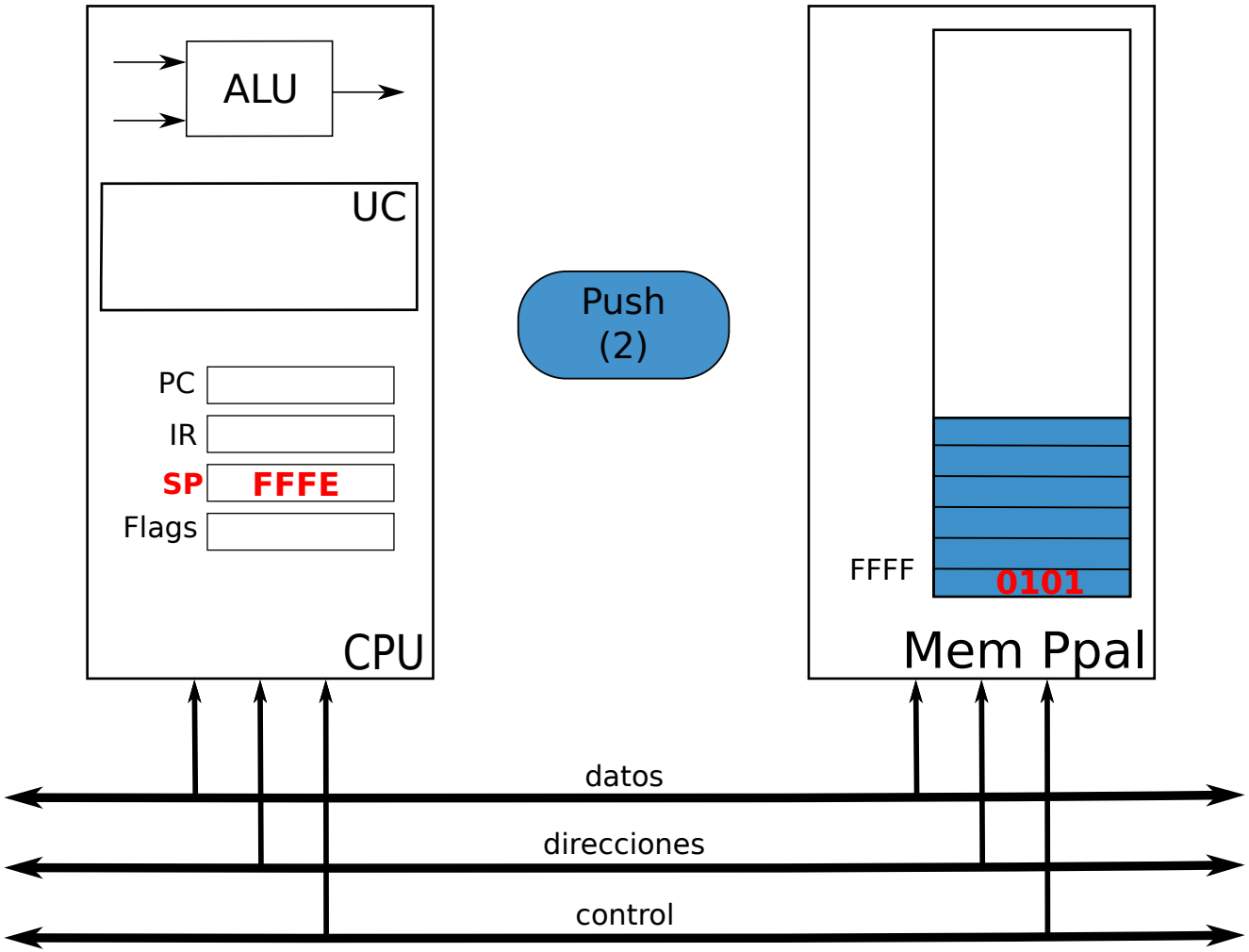
Push

1. Se hace una escritura del dato que está en el bus de datos en la dirección que está en SP
2. Se decrementa SP (así sigue cumpliendo la condición)

Estructura de Pila: Push



Estructura de Pila: Push

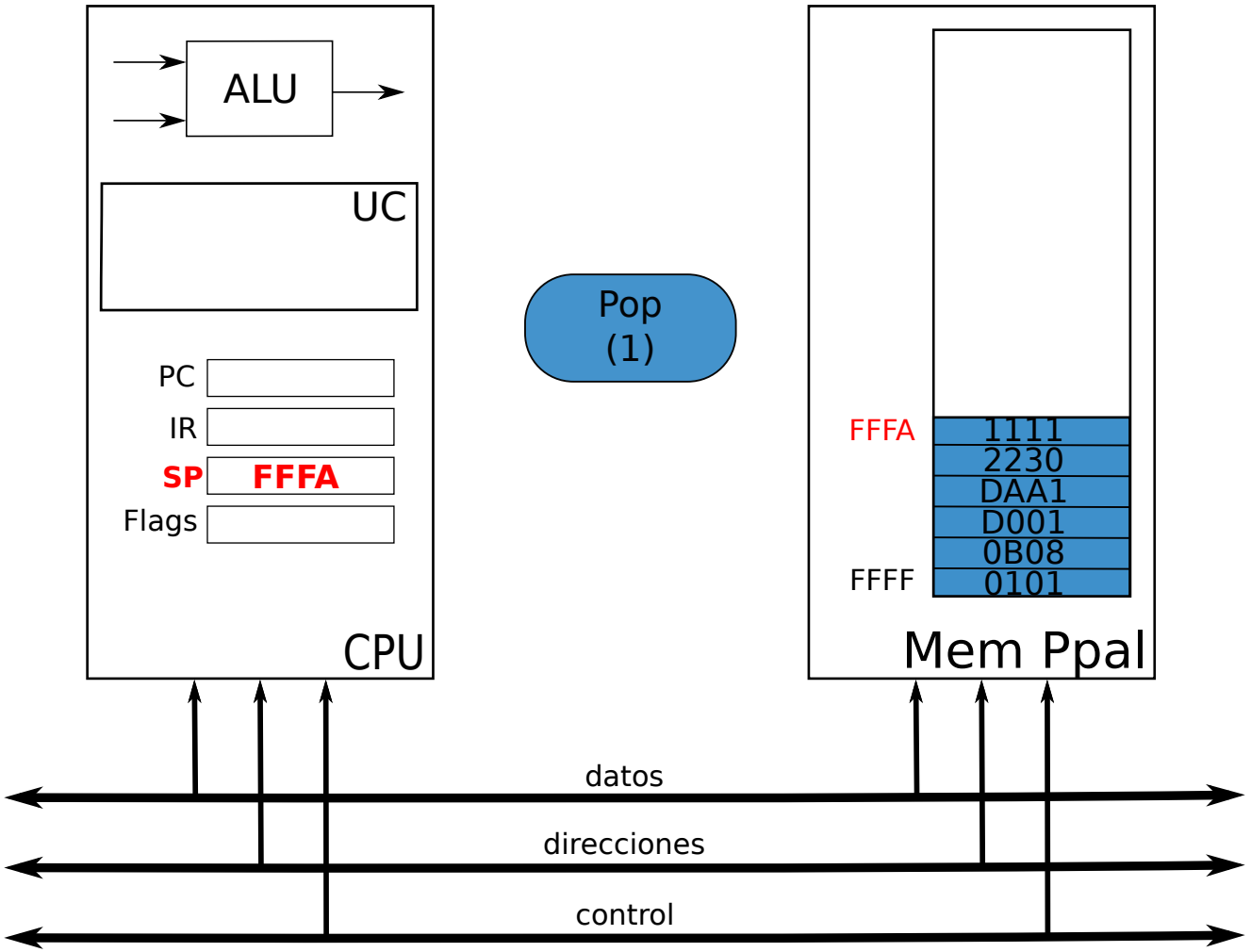


Estructura de Pila: Pop

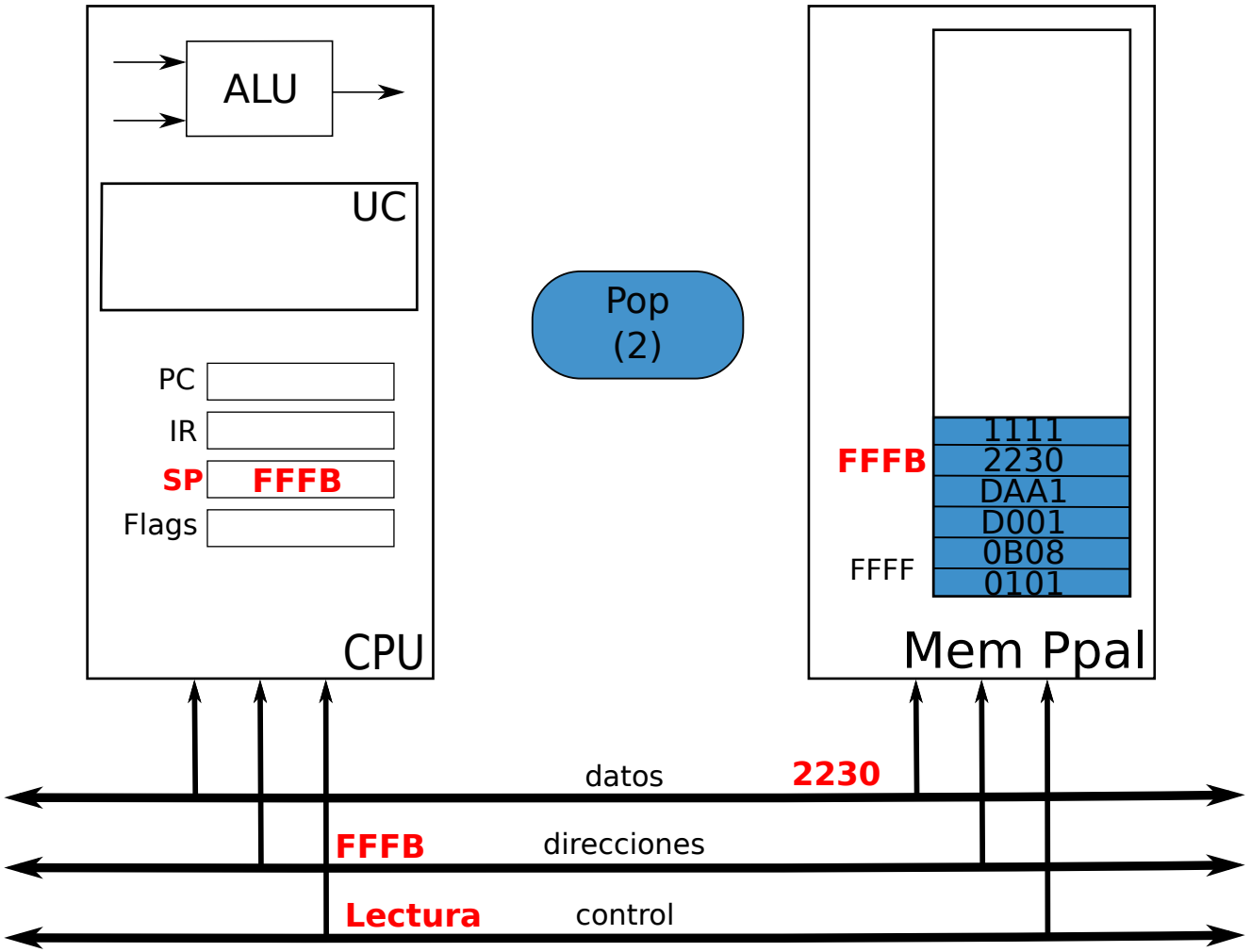
Pop

1. Se incrementa SP (para que haga referencia a un dato dentro de la pila)
2. Se hace una lectura de la dirección que está en SP

Estructura de Pila: Pop



Estructura de Pila: Pop



Estructura de Pila

- El tamaño y la ubicación de la pila está definido por la arquitectura.
- El pop no blanquea el tope de la pila.
- Cuando se hace push se pierde el valor que tenía la celda (por definición de escritura)

Estructura de Pila

sumar los 2 números al tope de la pila y apilar el resultado

```
POP R1
POP R2
ADD R1,R2
PUSH R1
```

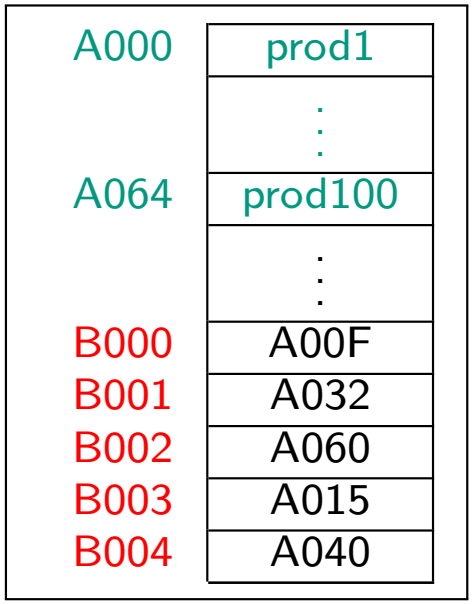
Desafío del maxi-kiosko

- Se tiene:
  1. tabla (arreglo) de precios unitarios de los 100 productos.
  2. tabla con los 5 productos que están de oferta esta semana, que contiene las direcciones donde están los precios unitarios de esos productos
- Se necesita:
  1. guardar los precios originales
  2. aplicar un descuento del 25 % a todos los productos que están en oferta

Desafío del maxi-kiosko

Ejemplo

1. Tabla de precios: A000..A064
2. Productos en oferta (son 5): B000..B004



Desafío del maxi-kiosko

Bosquejando el programa: ¿Que hay que hacer?

- Primero: Hacer una copia de la tabla de precios original
- Luego: Recorrer la tabla de ofertas
- Además: Aplicar un descuento a una celda

Modularización

Modularización

Modularizar
Dividir un problema *grande* en problemas mas *pequeños*

Problemón

Problemita 1

Problemita 2

Modularización

Desafío del maxi-kiosko

Programa principal

Copiar tabla original

Recorrer tabla de ofertas y aplicar descuento

Subrutina
Programa auto-contenido. Es sub porque se la piensa para ser utilizada dentro de otro programa

Modularización

Copiar tabla original

```
        MOV R0, 0xA000   ; direccion del 1er elemento
        MOV R1, 0xC000   ; direccion del inicio de la copia
arriba: MOV [R1], [R0]   ; copia un elemento
        ADD R1, 0x0001   ; dirección del sig
        ADD R0, 0x0001   ; dirección del sig
        CMP R0, 0xA064   ; compara con la dir del ultimo
        JLEU arriba      ; si es menor o igual, salta
```

Modularización

Recorrer tabla de descuentos y aplicar

```
        MOV R3, 0xB000   ; direccion de la primer oferta
        MOV R1, 0x0005   ; tamaño del arreglo de ofertas
otro:   MOV R6, [R3]     ; copia el precio actual
        MUL R6, 0x0019   ; multiplica por 25
        DIV R6, 0x0064   ; divide por 100
        SUB [R3], R6     ; descuenta
        ADD R3, 0x0001   ; dirección del sig
        SUB R1, 0x0001   ; uno menos para procesar
        JNE otro         ; si R1 no es cero, salta
```

¿Cómo se integran las partes?

Modularización: integrar las partes

Encapsular las subrutinas

```
copiar:   MOV R0, 0xA000
          MOV R1, 0xC000
arriba:   MOV [R1], [R0]
          ADD R1, 0x0001
          ADD R0, 0x0001
          CMP R0, 0xA064
          JLEU arriba
          RET
```

```
descontar:  MOV R3, 0xB000
            MOV R1, 0x0005
otro:       MOV R6, [R3]
            MOV R5, [R6]
            MUL R5, 0x0019
            DIV R5, 0x0064
            SUB [R6], R5
            ADD R3, 0x0001
            SUB R1, 0x0001
            JNE otro
            RET
```

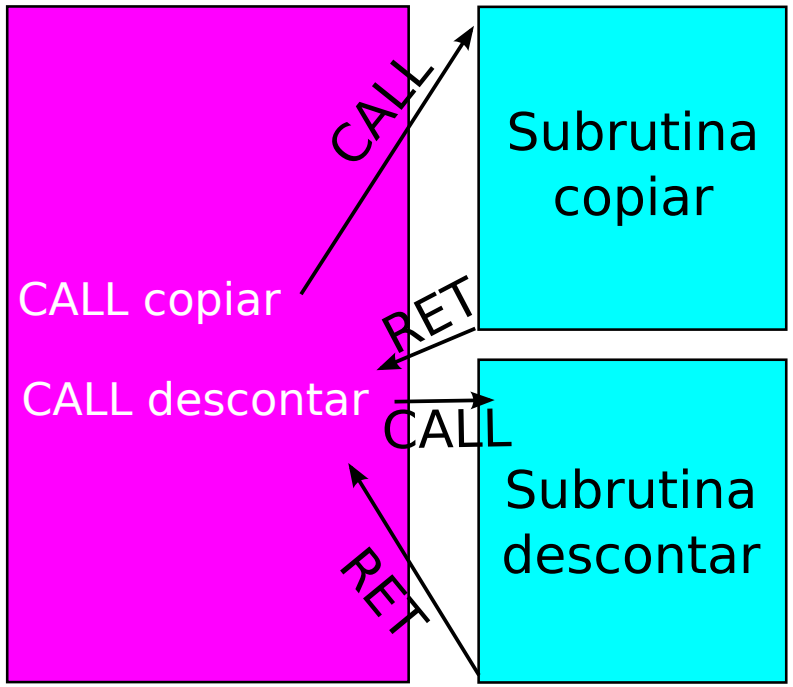
Llamar a las subrutinas

```
; Programa principal
call copiar
```

Modularización: CALL y RET

Instrucción CALL
Desvía el flujo del programa a la instrucción que define la etiqueta

Instrucción RET
Permite restituir el flujo del programa a la instrucción siguiente del último llamado



La planilla de Orga

- Se tiene dos arreglos con las notas de los alumnos de orga, a partir de la celda 9000 y A000 respectivamente.
- La longitud de los arreglos está en la primer celda de cada uno.
- Se necesita determinar si el alumno cuyas notas están en las celdas 9002 y A002 aprobó la materia (ambas notas deben ser mayores o iguales a 4), en cuyo caso se debe poner un 1 en el registro R1 y 0 en caso contrario.

La planilla de Orga: ejemplo

⋮	⋮
9000	0004
9001	0008
9002	0004
9003	000A
9003	0003
⋮	⋮
A000	0004
A001	0006
A002	0002
A003	0008
A003	000A
⋮	⋮

- Hay 4 estudiantes
- El segundo estudiante se sacó 4 en el primer parcial y 2 en el segundo

La planilla de Orga

Poner un 1 en el registro R1 si las notas de las celdas 9002 y A002 son mayores o iguales a 4, o 0 en caso contrario.

```
            CMP [9002], 0x0004
            JCS leFueMal
            CMP [A002], 0x0004
            JCS leFueMal
            MOV R1, 0x0001
            JMP fin
leFueMal:   MOV R1, 0x0000
fin:        ...
```

La planilla de Orga: Contar los alumnos que aprobaron ambos parciales

Recorrer el arreglo

verificar si $nota1 >= 4$ y $nota2 >= 4$

Reuso

Reuso

Reusar
Escribir subrutinas que puedan ser usadas en diferentes situaciones

Problema 1
- Subrutina 1
- Subrutina 2

Problema 2
- Subrutina 1
- Subrutina 3

Reuso

La planilla de Orga: Contar los alumnos que aprobaron ambos parciales

```
aprobar:    CMP [9002], 0x0004
            JCS leFueMal
            CMP [A002], 0x0004
            JCS leFueMal
            MOV R1, 0x0001
            JMP fin
leFueMal:   MOV R1, 0x0000
fin:        ...
```

```
            MOV R7, [9000]      ;long del arreglo
            MOV R6, 0x0000      ;contador
otro:       call aprobar
            ADD R6,R1
            SUB R7, 0x0001
            JNE otro
```

¿Cuál es el error?

Parámetros

Parámetros

La rutina **aprobar** debe ser mas flexible

```
aprobar:    CMP [9002], 0x0004
            JCS leFueMal
            CMP [A002], 0x0004
            JCS leFueMal
            MOV R1, 0x0001
            JMP fin
leFueMal:   MOV R1, 0x0000
fin:        ...
```

```
aprobar:    CMP [R0], 0x0004
            JCS leFueMal
            CMP [R2], 0x0004
            JCS leFueMal
            MOV R1, 0x0001
            JMP fin
leFueMal:   MOV R1, 0x0000
fin:        RET
```

Reuso

La planilla de Orga: Contar los alumnos que aprobaron ambos parciales

```
aprobar:    CMP [R0], 0x0004
            JCS leFueMal
            CMP [R2], 0x0004
            JCS leFueMal
            MOV R1, 0x0001
            JMP fin
leFueMal:   MOV R1, 0x0000
fin:        RET
```

```
            MOV R7, [9000]      ;long del arreglo
            MOV R0, 0x9001      ;primer arreglo
            MOV R2, 0xA001      ;segundo arreglo
            MOV R6, 0x0000      ;contador
otro:       call aprobar
            ADD R6,R1
            ADD R0, 0x0001
            ADD R2, 0x0001
            SUB R7, 0x0001
            JNE otro
```

Reuso

Contar alumnos que NO aprobaron ambos parciales

```
aprobar:    CMP [R0], 0x0004
            JL leFueMal
            CMP [R2], 0x0004
            JL leFueMal
            MOV R1, 0x0001
            JMP fin
leFueMal:   MOV R1, 0x0000
fin:        RET
```

```
            MOV R7, [9000]
            MOV R0, 0x9001
            MOV R2, 0xA001
            MOV R6, 0x0000
otro:       CALL aprobar
            CMP R1, 0x0001
            JE noLoSumo
            ADD R6, 0x0001
noLoSumo:   ADD R0, 0x0001
            ADD R2, 0x0001
            SUB R7, 0x0001
            JNE otro
```

Reuso

Ejercicio

Se cuenta con una subrutina que aplica un descuento dado como parámetro, sobre una celda cuya dirección se pasa como parámetro:

```
descontar:  MOV R6, [R3]
            MUL R6, R4
            DIV R6, 0x0064
            SUB [R3], R6
            RET
```

Hacer un programa que aplique un 10 % de descuento al valor en la celda 6565, un 30 % al valor en la celda AAAA y un 45 % al valor en 0367.

Reuso

Ejercicio

Hacer un programa que aplique un 10 % de descuento al valor en la celda 6565, un 30 % al valor en la celda AAAA y un 45 % al valor en 0367.

```
MOV R3, 0x6565
MOV R4, 0x000A      ; 10 %
CALL descontar;
MOV R3, 0xAAAA
MOV R4, 0x001E      ; 30 %
CALL descontar;
MOV R3, 0x0367
MOV R4, 0x002D      ;45 %
CALL descontar;
```

```
descontar:  MOV R6, [R3]
            MUL R6, R4
            DIV R6, 0x0064
            SUB [R3], R0
            RET
```

Contratos

Ejercicio

En dos grupos

Equipo A Escribir una rutina que calcule el promedio de las notas de un alumno, asumiendo que las notas están en R4 y R5. El resultado debe dejarse en R6.

Equipo B Escribir un programa que calcule el promedio máximo entre los estudiantes de la comisión. Las notas están en dos arreglos que comienzan en 0B00 y 0C00.

Inconvenientes

¿Cuál es el problema del código no documentado?

- Para entender lo que hace una rutina se debe tratar de comprender el código
- Es dificil detectar errores secundarios: Cuando la rutina modifica algo no esperado
- Es dificil modificar el código para nuevos requerimientos

Contratos

¿Cómo documentar el código?

Especificar:

- **Requiere** Qué necesita la rutina (Parámetros y precondiciones)
- **Retorna** En que variable (registro o memoria) se retorna el resultado
- **Modifica** Que variables auxiliares se utilizan (registros, memoria, flags)

Ejercicio: documentar la rutina aprobar

```
aprobar:    CMP [R0], 0x0004
            JCS leFueMal
            CMP [R2], 0x0004
            JCS leFueMal
            MOV R1, 0x0001
            JMP fin
leFueMal:   MOV R1, 0x0000
fin:        RET
```

Requiere Las notas están en las celdas apuntadas por R0 y R2. Los valores están en $BSS(16)$

Retorna un 1 en R1 si ambas notas son mayores o iguales a 4, 0 en caso contrario

Modifica flags

Ejercicio: documentar la rutina copiar:

```
copiar:   MOV R0, 0xA000
          MOV R1, 0xC000
arriba:   MOV [R1], [R0]
          ADD R1, 0x0001
          ADD R0, 0x0001
          CMP R0, 0xA064
          JLEU arriba
          RET
```

Requiere El arreglo tiene 100 elementos y comienza en A000

Retorna El arreglo copiado a partir de la celda C000

Modifica R0, R1, flags

Ejercicio: documentar la rutina descontar:

```
descontar:  MOV R3, 0xB000
            MOV R1, 0x0005
otro:       MOV R6, [R3]
            MUL R6, 0x0019
            DIV R6, 0x0064
            SUB [R3], R0
            ADD R3, 0x0001
            SUB R1, 0x0001
            JNE otro
            RET
```

Requiere El arreglo de ofertas tiene 5 elementos y comienza en B000

Retorna Las direcciones indicadas en el arreglo de ofertas

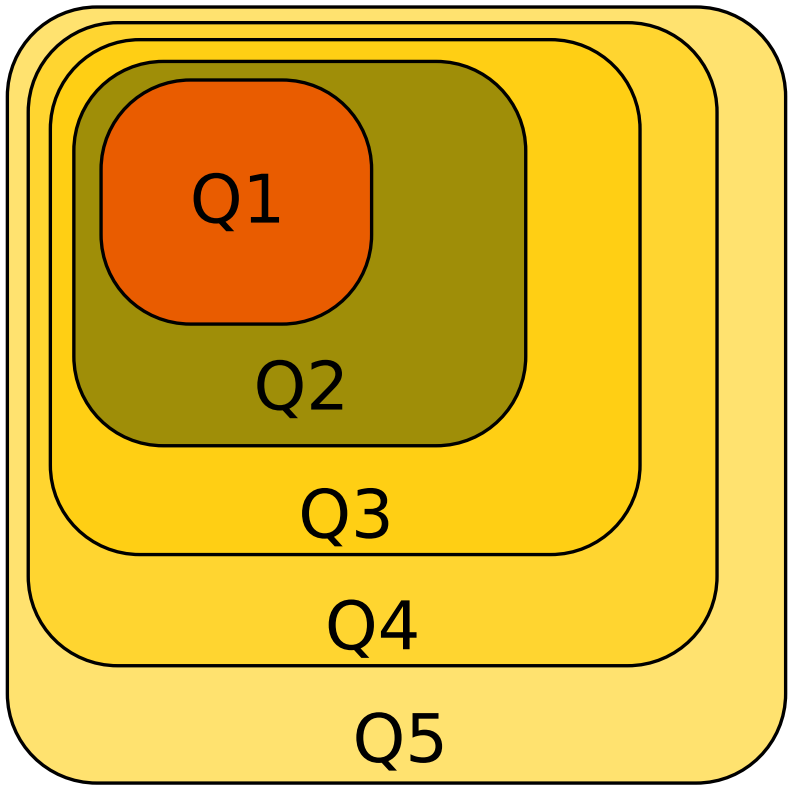
Modifica R3, R1, R6 flags

Arquitecturas **Q**

... quinto acto ...

Arquitectura **Q5**

Arquitectura **Q5**



Arquitectura **Q5**

- Tiene 8 registros de uso general de 16 bits: R0..R7
- Tiene direcciones de 16 bits
- Tiene registros no visibles al programador:
  - *Program counter* de 16 bits.
  - Registro de flags (**ZNCV**) de 16 bits.
  - *Stack Pointer* de 16 bits. Comienza en la dirección FFEF .
- permite 3 modos de direccionamiento:
  - modo registro: el valor buscado está en un registro
  - modo inmediato: el valor buscado está codificado dentro de la instrucción
  - modo directo: el valor buscado está contenido en una celda de memoria
  - modo indirecto: la dirección del valor buscado está contenido en una celda de memoria
  - modo registro indirecto: la dirección del valor buscado está contenido en un registro

Arquitectura **Q5**: formato de instrucciones

- Operaciones de tipo 1 (MUL,MOV,ADD,SUB,CMP,DIV,AND,OR)

Cod_Op (4b)	Modo Destino (6b)	Modo Origen (6b)	Operando Destino (16b)	Operando Origen (16b)

- Operaciones de tipo 2 (Un operando Origen: JMP, PUSH CALL)

Cod_Op (4b)	Relleno (000000)	Modo Origen (6b)	Operando Origen (16b)

- Operaciones de tipo 3 (Un operando Destino:)

Cod_Op (4b)	Modo Destino (6b)	Relleno (000000)	Operando Origen (16b)

- Operaciones de tipo 4 (Saltos condicionales y relativos)

Prefijo (1111)	Cod_Op (4)	Desplazamiento(8) (8b)

- Operaciones de tipo 5 (instrucciones sin operandos: RET)

Cod_Op (4b)	Relleno (000000000000)

Arquitectura **Q5**: Operaciones de tipo 1

Tipo 1: Aritméticas y lógicas

Cod_Op (4b)	Modo Destino (6b)	Modo Origen (6b)	Operando Destino (16b)	Operando Origen (16b)

Operación	**CodOp**
MUL	0000
MOV	0001
ADD	0010
SUB	0011
CMP	0110
DIV	0111
AND	0100
OR	0101

Arquitectura **Q5**: Operaciones de tipo 2

Tipo 2: Un operando Origen

Cod_Op (4b)	Relleno (000000)	Modo Origen (6b)	Operando Origen (16b)

Operación	**CodOp**	**Efecto**
JMP	1010	PC ← dirección
PUSH	1110	[SP] ← Origen; SP ← SP - 1
CALL	1011	[SP] ← PC; SP ← SP - 1; PC ← Origen

Arquitectura **Q5**: Operaciones de tipo 3

Tipo 3: Un operando Destino

Cod_Op (4b)	Modo Destino (6b)	Relleno (000000)	Operando Origen (16b)

Operación	**CodOp**	**Efecto**
NOT	1001	Dest $\leftarrow$ NOT Dest (bit a bit)
POP	1101	SP $\leftarrow$ SP + 1; Dest $\leftarrow$ [SP]

Arquitectura **Q5**: Operaciones de tipo 4

Tipo 4: Salto condicional (relativo) - 1 de 2

Prefijo (1111)	Cod_Op (4b)	Desplazamiento(8b)

Salto	**Codop**	**Descripción**	**Condición**
JE	0001	Igual / Cero	Z
JNE	1001	No igual	$\overline{Z}$
JLEU	0100	Menor o igual sin signo	$C^{\vee} Z$
JGU	1100	Mayor sin signo	$\overline{(C^{\vee} Z)}$
JCS	0101	Menor sin signo	C
JNEG	0110	Negativo	N

Arquitectura **Q5**: Operaciones de tipo 4

Tipo 4: Salto condicional (relativo) - 2 de 2

Prefijo (1111)	Cod_Op (4b)	Desplazamiento(8b)

Salto	**Codop**	**Descripción**	**Condición**
JVS	0111	Overflow	V
JLE	0010	Menor o igual con signo	$Z^{\vee}(N \oplus V)$
JG	1010	Mayor con signo	$\overline{(Z^{\vee}(N \oplus V))}$
JL	0011	Menor con signo	$N \oplus V$
JGE	1011	Mayor o igual con signo	$\overline{(N \oplus V)}$

Arquitectura **Q5**: Operaciones de tipo 5

Tipo 5: Instrucciones sin operandos

Cod_Op (4b)	Relleno (000000000000)

Operación	**CodOp**	**Efecto**
RET	1100	PC $\leftarrow$ [SP+1]; SP $\leftarrow$ SP + 1

Ejercicio

Hacer un programa que cuente cuantos números de un arreglo son pares, tienen el bit 5 en 1 y son múltiplos de 5

¿Preguntas?