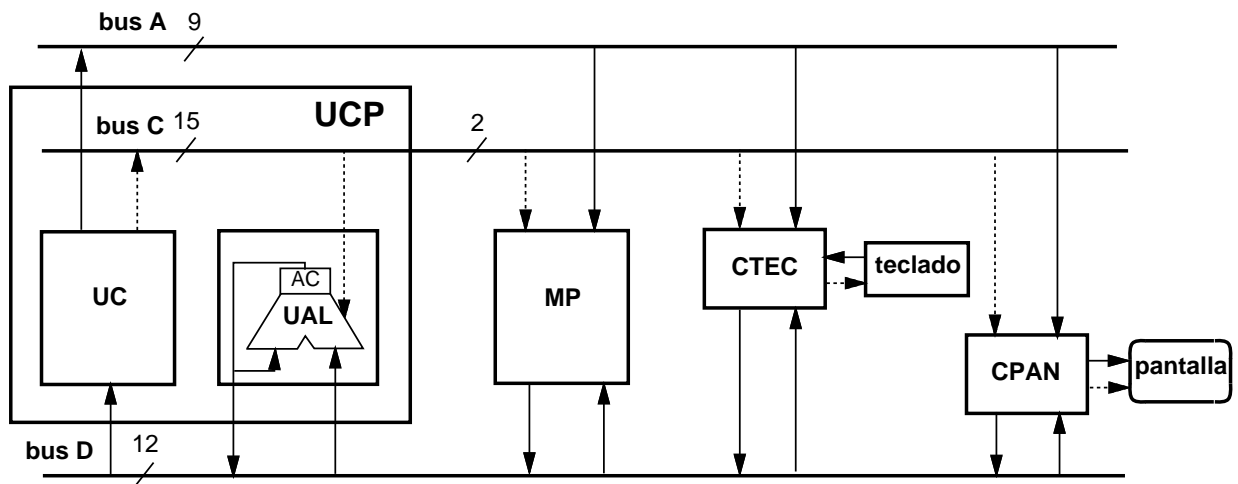
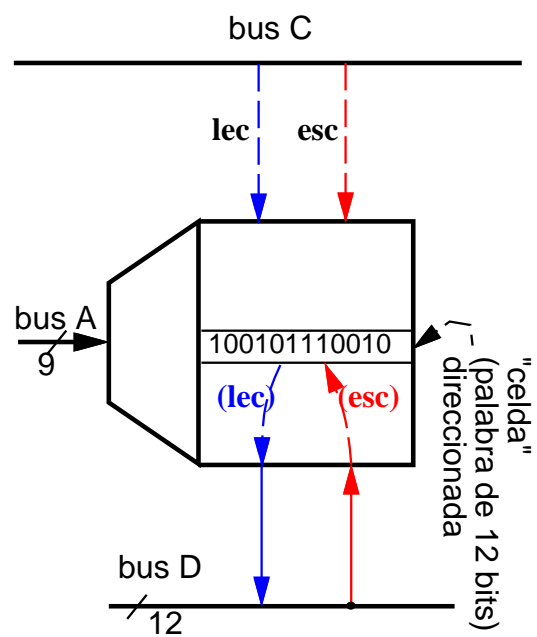
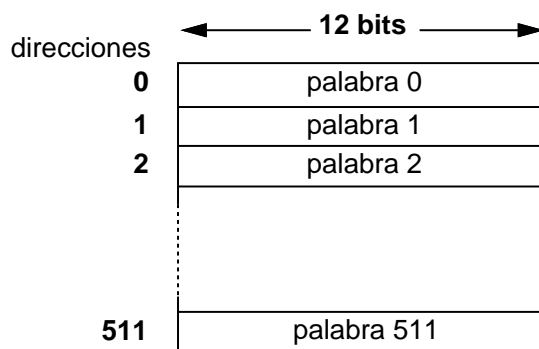


Símplez: modelo estructural

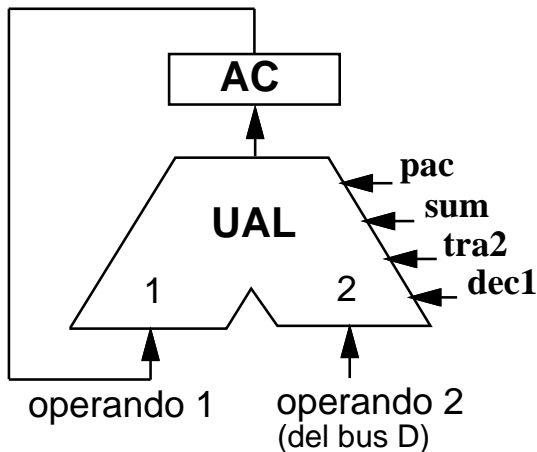


Símplez: memoria principal

Capacidad, organización y funcionamiento:



Símplez: unidad aritmética



Operaciones (notación «RT»):

* **pac**: $0 \rightarrow AC$

* **sum**: $(AC) + (\text{bus D}) \rightarrow AC$

* **tra2**: $(\text{bus D}) \rightarrow AC$

* **dec1**: $(AC) - 1 \rightarrow AC$

Símplez: unidad de control

Repite indefinidamente:

1. *Extrae (lee)* una instrucción de la MP.

2. La *interpreta*:

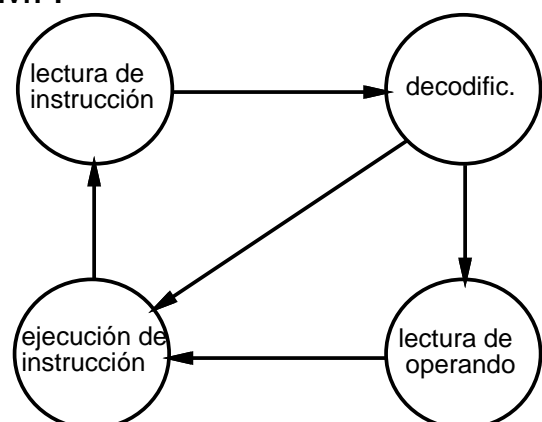
a) *Descodifica* la instrucción.

b) En su caso, *extrae* el operando (de la MP o de periférico).

c) *Ejecuta* la instrucción.

Incluye, en su caso, escribir en la MP o en periférico.

3. Genera dirección de la siguiente (normalmente, ya en el paso 1).



- Teclado y pantalla (trivializados)
- Caracteres codificados en ISO «Latin 1» (ISO 8859–1) o «Latin 9» (ISO 8859–15)
- Comunicación a través del acumulador
- Tiempos para las transferencias mucho mayores que los de la UAL y la MP
- Ocupan las direcciones 508 a 511 («robadas» a la MP)

Símplez: modelo funcional (1)

Representación de información numérica
(sólo números enteros no negativos):

- En binario, con 12 bits
- Rango: de 0 a $2^{12} - 1 = 4095$
- Ejemplos:
 - D'4 = B'000000000100 = Q'0004 = H'004
 - D'10 = B'000000001010 = Q'0012 = H'00A
 - D'1022 = B'001111111110 = Q'1776 = H'3FE
 - D'4095 = B'111111111111 = Q'7777 = H'FFF

Símplez: modelo funcional (2)

Representación de información textual (caracteres):

- Código ISO Latin9 en los 8 bits menos significativos de una palabra

- Ejemplos:

'a' = B'01100001 = Q'141 = H'61 = 97

'à' = B'11100000 = Q'340 = H'E0 = 224

'€' = B'10100100 = Q'244 = H'A4 = 164

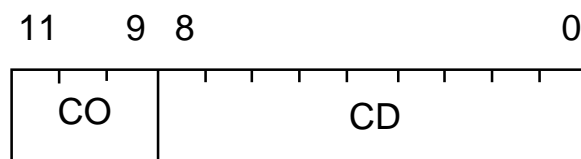
' ' = B'00100000 = Q'040 = H'20 = 32

'ret' = B'00001101 = Q'015 = H'0D = 13

'nl' = B'00001010 = Q'012 = H'0A = 10

Símplez: modelo funcional (3)

Formato de instrucciones:



CO : Código de operación ($2^3 = 8$)

CD : Campo de dirección ($2^9 = 512$)

Símplez: modelo funcional (4)

Repertorio de instrucciones:

CO (bin)	CO (oct)	CO (nem)	Significado («RT»)
000	0	ST	(AC) → MP[(CD)]
001	1	LD	(MP[(CD)]) → AC
010	2	ADD	(AC) + (MP[(CD)]) → AC
011	3	BR	siguiente instrucción en MP[(CD)]
100	4	BZ	si cero, siguiente instrucción en MP[(CD)]
101	5	CLR	0 → AC
110	6	DEC	(AC) – 1 → AC
111	7	HALT	para

Suma de dos enteros

■ resultado = operando1 + operando2;

■ **Asignación de direcciones:**

operando1: D'300 = Q'454

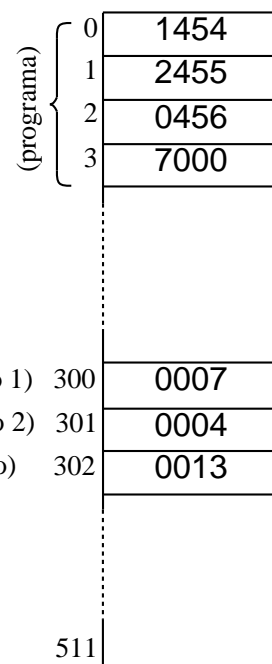
operando2: D'301 = Q'455

resultado: D'302 = Q'456

■ **Mapa de memoria:** ⇒

Dir.MP (dec.)	Cont. (oct.)	Cont. (nem.)	Comentarios
------------------	-----------------	-----------------	-------------

[0]	1454	LD	/300 ; op1 → AC
[1]	2455	ADD	/301 ; (AC)+op2 → AC
[2]	0456	ST	/302 ; (AC) → res
[3]	7000	HALT	



Progresión aritmética (1)

```

■ ai = a0; suma = a0;
  for(int c = 1; c <= n; c++) {
    ai = ai + r;
    suma = suma + ai;
  }

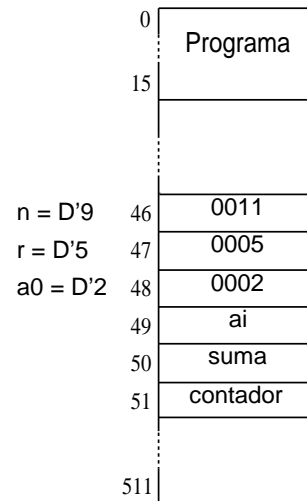
```

■ Asignación de direcciones (por ej.):

```

n      : D'46 = Q'56
r      : D'47 = Q'57
a0     : D'48 = Q'60
ai     : D'49 = Q'61
suma  : D'50 = Q'62
c      : D'51 = Q'63 (contador)

```



Resultado:
 $a_0 + a_1 + \dots + a_9 = 2 + 7 + \dots + 47 = 245$

■ Mapa de memoria: \Rightarrow

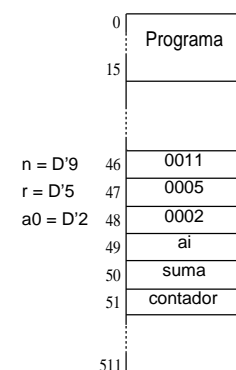
Progresión aritmética (2)

Asignaciones iniciales (suponemos que en [52] hay «1»):

C	Mnemónico	Octal
ai = a0;	LD /48	1060
	ST /49	0061
suma = a0;	ST /50	0062
c = 1;	LD /52	1064
	ST /51	0063

Asignaciones dentro del bucle:

C	Mnemónico	Octal
ai = ai + r;	LD /49	1061
	ADD /47	2057
	ST /49	0061
suma =	ADD /50	2062
suma + ai;	ST /50	0062



Resultado:
 $a_0 + a_1 + \dots + a_9 = 2 + 7 + \dots + 47 = 245$

Control del bucle: Comprobar si $c \leq n$ y seguir o no seguir.

Pero sólo tenemos una instrucción de bifurcación, BZ,

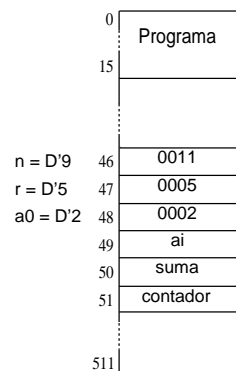
¡y no hay instrucción para restar!

Progresión aritmética (3)

Solución: contar hacia atrás hasta que $c = 0$

```
ai = a0; suma = a0;
for(int c = n; c != 0; c--) {
    ai = ai + r;
    suma = suma + ai;
}
```

C	Mnemónico	Octal	Dir.MP
ai = a0;	LD /48	1060	[0]
	ST /49	0061	[1]
suma = a0;	ST /50	0062	[2]
c = n;	LD /46	1056	[3]
	ST /51	0063	[4]
¿c != 0?	BZ /15	4017	[5]
ai = ai + r;	LD /49	1061	[6]
	ADD /47	2057	[7]
	ST /49	0061	[8]
suma =	ADD /50	2062	[9]
suma + ai;	ST /50	0062	[10]



Resultado:
 $a_0 + a_1 + \dots + a_9 = 2 + 7 + \dots + 47 = 245$

C	Mnemónico	Octal	Dir.MP
c--	LD /51	1063	[11]
	DEC	6000	[12]
	ST /51	0063	[13]
	BR /5	3005	[14]
	HALT	7000	[15]



Interpretación de contenidos de la MP (1)

dir. MP	cont. (binario)	cont. (octal)	cont. (hex.)
[0]	101 000 000 000	Q'5000	H'A00
[1]	111 111 111 010	Q'7772	H'FFA
...
[5]	010 000 000 001	Q'2001	H'401
...

¿Qué hay en estas palabras? (interpretación)

Depende ...



Interpretación de contenidos de la MP (2)

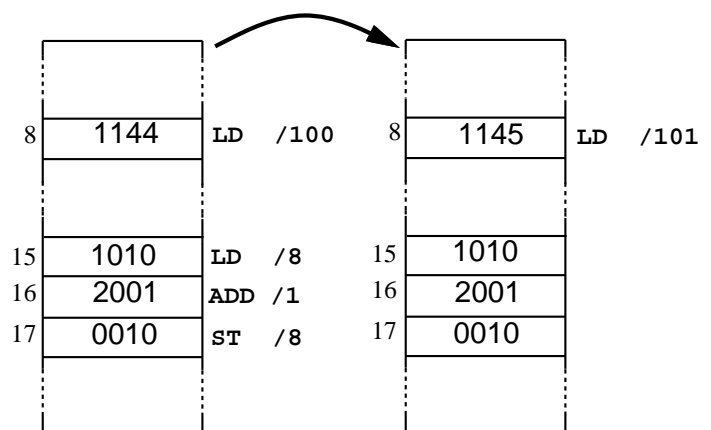
Dir.MP/cont.	Interpretación como ...		
	instrucción	número	carácter
[0] Q'5000	CLR	D'2560	NUL
[1] Q'7772	HALT	D'4090	ú
[5] Q'2001	ADD /1	D'1025	[control]

- ¿Qué ocurre si la UC ejecuta [5]?
- ¿Qué ocurre si empieza ejecutando [0]?

En el nivel de máquina convencional no hay «tipos»
(salvo en algunas máquinas)

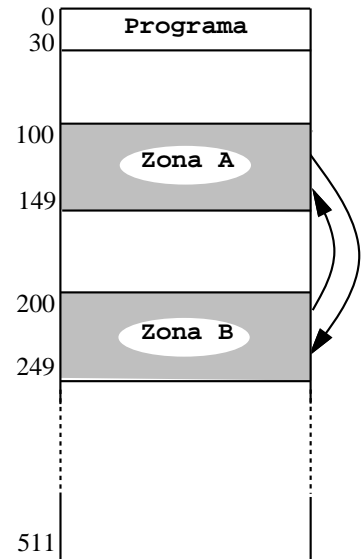
Modificación de instrucciones

[0] 3002 BR /2
 [1] 0001
 [2] 5000 CLR
 [3] ...
 [8] 1144 LD /100
 [9] ...
 [15] 1010 LD /8
 [16] 2001 ADD /1
 [17] 0010 ST /8
 [18] ...
 [25] 3010 BR /8
 [26] ...



Intercambio («swapping») en la MP

```
int temp;
int[] a = new int[50];
int[] b = new int[50];
---
---
for (int i = 0; i<a.length; i++) {
    temp = a[i];
    a[i] = b[i];
    b[i] = temp;
}
```

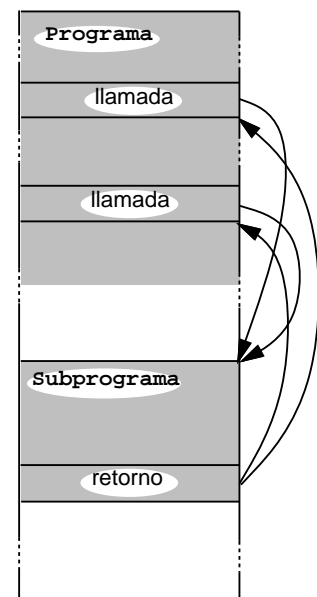


Ejercicio: Traducir al lenguaje de Símplez, suponiendo que el compilador pone las constantes 50 y 1 en las direcciones [1] y [2] y reserva la dirección [3] para la variable *i* y la [4] para *temp*.

Subprogramas

Implementación de procedimientos y funciones («métodos») en el nivel de máquina convencional.

- *Transmisión de los argumentos* (paso de parámetros).
En Símplez, únicamente *por valor*.
A través del AC, o de palabras de la MP (variables globales).
- *Preservación de la dirección de retorno*.
En Símplez, «construyendo» la instrucción de retorno, BR /dir, antes de cada llamada.



Subprograma para restar

Estas instrucciones implementan el subprograma, suponiendo que:

- El minuendo (mayor que el sustraendo) se pasa por el acumulador.
- El sustraendo (mayor que 0) se pasa por la dirección 202.
- Antes de llamarlo (con una instrucción BR /200) se modifica adecuadamente la última instrucción.

Ejercicio: Escribir las instrucciones necesarias de un programa que a partir de la dirección [10] requiere hacer la operación $(50) - (51) \rightarrow 52$, y a partir de [20], $(100) - (101) \rightarrow 102$.

```
[200] 3313 BR /203
[201] 0000
[202] 0000 ; sustr.
[203] 6000 DEC
[204] 0311 ST /201
[205] 1312 LD /202
[206] 6000 DEC
[207] 4323 BZ /211
[208] 0312 ST /202
[209] 1311 LD /201
[210] 3313 BR /203
[211] 1311 LD /201
[212] 3000 BR /0
```

Lenguaje ensamblador

- Etiquetas, pseudoinstrucciones, directivas...
- ⇒ Nivel de máquina simbólica

```
Programa fuente
-----
                ORG 200
[200] 3313 RESTA BR /BUCLE
[201] 0000 RESULT RES 1
[202] 0000 SUSTR RES 1
[203] 6000 BUCLE DEC
[204] 0311                ST /RESULT
[205] 1312                LD /SUSTR
[206] 6000                DEC
[207] 4323                BZ /FINAL
[208] 0312                ST /SUSTR
[209] 1311                LD /RESULT
[210] 3313                BR /BUCLE
[211] 1311 FINAL LD /RESULT
[212] 3000                BR /0
                END
```

Comunicaciones con los periféricos

Cada periférico tiene un **puerto de datos** y un **puerto de estado**.

Puerto	Dirección
estado de la pantalla	508
datos de la pantalla	509
estado del teclado	510
datos del teclado	511

- LD /511: lee el último carácter tecleado.
- LD /510: lee el estado del teclado («1» = preparado para enviar un carácter; «0» = no preparado).
- ST /509: saca por pantalla el carácter del AC.
- LD /508: lee el estado de la pantalla («1» = preparada para recibir un carácter; «0» = no preparada, ocupada con el anterior).

Espera activa («*busy waiting*»)

Para escribir en la pantalla:

```
[20] 1774 LD /508 ; (en [20], por ejemplo)
[21] 4024 BZ /20 ; si no preparada sigue probando
[22] 1005 LD /5 ; carga el caracter (en [5], p. ej.)
[23] 0775 ST /509 ; y lo envia a la pantalla
```

Para leer del teclado:

```
[40] 1776 LD /510 ; (en [40], por ejemplo)
[41] 4050 BZ /40 ; si no preparado sigue probando
[42] 1777 LD /511 ; carga el caracter
[43] 0003 ST /3 ; y lo almacena en [3] (p. ej.)
```

Análisis de tiempos en la espera activa

Supongamos estos datos:

- Tiempo de ejecución para las instrucciones BR, BZ, CLR, DEC y HALT: 1 ciclo de MP = 200 ns.
- Tiempo de ejecución para las instrucciones ST, LD y ADD: 2 ciclos de MP = 400 ns.
- Tiempo de escritura de un carácter en la pantalla:
 $1/30 \text{ s} \approx 33 \times 10^6 \text{ ns}$.

— Si inmediatamente después de ST 509 se inicia otro bucle de espera para escritura, ¿cuántas veces se ejecutan las dos instrucciones del bucle?

— $33 \times 10^6 / 600 = 55.000$ veces

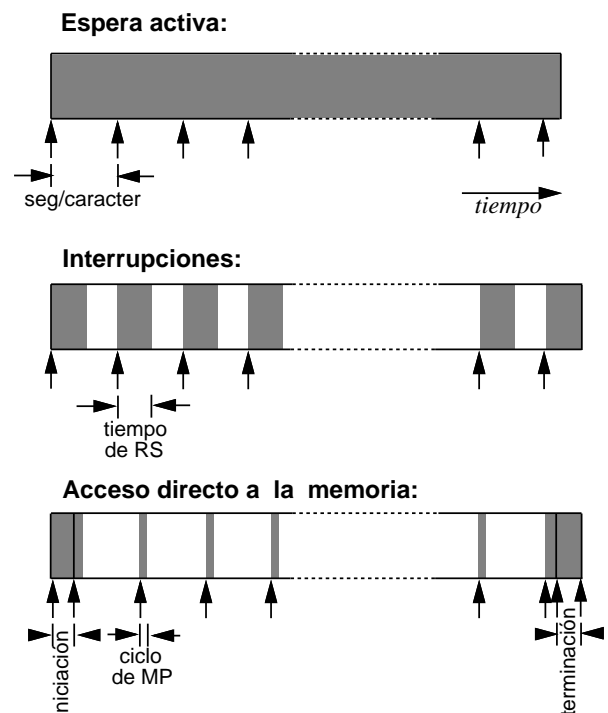
Introducción a otros mecanismos

Interrupciones:

- «iniciativa» del periférico
- interrumpe a la UCP
para cada carácter
- la UCP ejecuta una RS
para cada carácter

Acceso directo a la memoria:

- el periférico se comunica directamente con la MP
- sólo interrumpe al final de la transferencia de un bloque
- va «robando» ciclos a la UCP



Símplez: conclusiones

- ♣ Conveniencia de más «facilidades hardware»
 - ⇒ instrucciones SUB, CALL/RET, lógicas y de desplazamiento
- ♣ Capacidad de direccionamiento ridícula
 - ⇒ modo de direccionamiento indirecto.
- ♣ Modificación de instrucciones para recorrer la MP
 - ⇒ modo de direccionamiento indexado.
- ♣ Pocos y pobres tipos de datos
 - ⇒ facilidades hardware para «empalmar» palabras.
- ♣ Comunicación con periféricos bloquea a la UCP
 - ⇒ interrupciones, ADM.

Programación: conclusiones (1)

- ♣ **Programación en el nivel de máquina convencional:**
 - Propensa a errores y costosa en desarrollo
 - Programas ligados a la máquina
 - Aparición del *nivel de máquina simbólica*
 - No utilizada, salvo casos especiales
- ♣ **Automodificación de los programas:**
 - Programas no reutilizables
 - Propensión a errores
 - Absolutamente «prohibido» en ingeniería del software
 - Lenguajes de alto nivel no lo permiten
 - ¿Aprendizaje?

♣ Comunicaciones con los periféricos:

- Interrupciones y ADM permiten obtener mejor rendimiento de la máquina, pero programación delicada y costosa
- Aparición del *nivel de máquina operativa*