

EMI

MATERIA : ANALISIS DISEÑO DE SISTEMAS
BASADOS EN MICROPROCESADORES

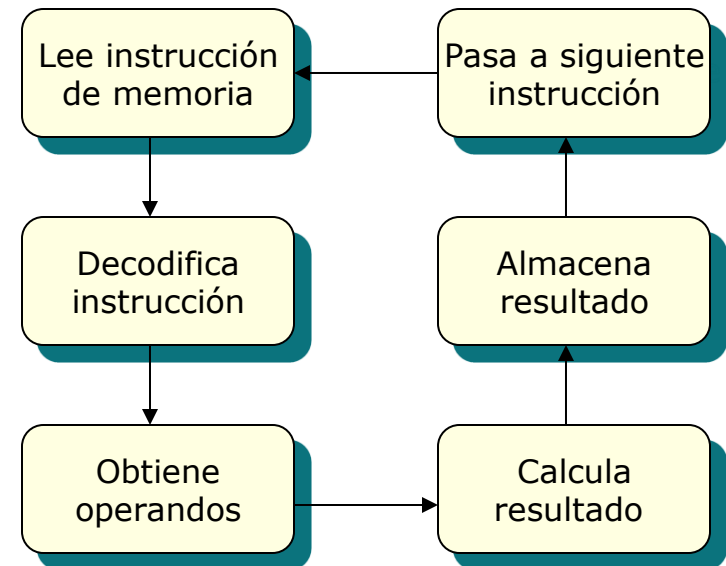
CODIGO : SIS – 03 2 12

DOCENTE : CESAR MARTIN
SUAREZ SUAREZ

El computador

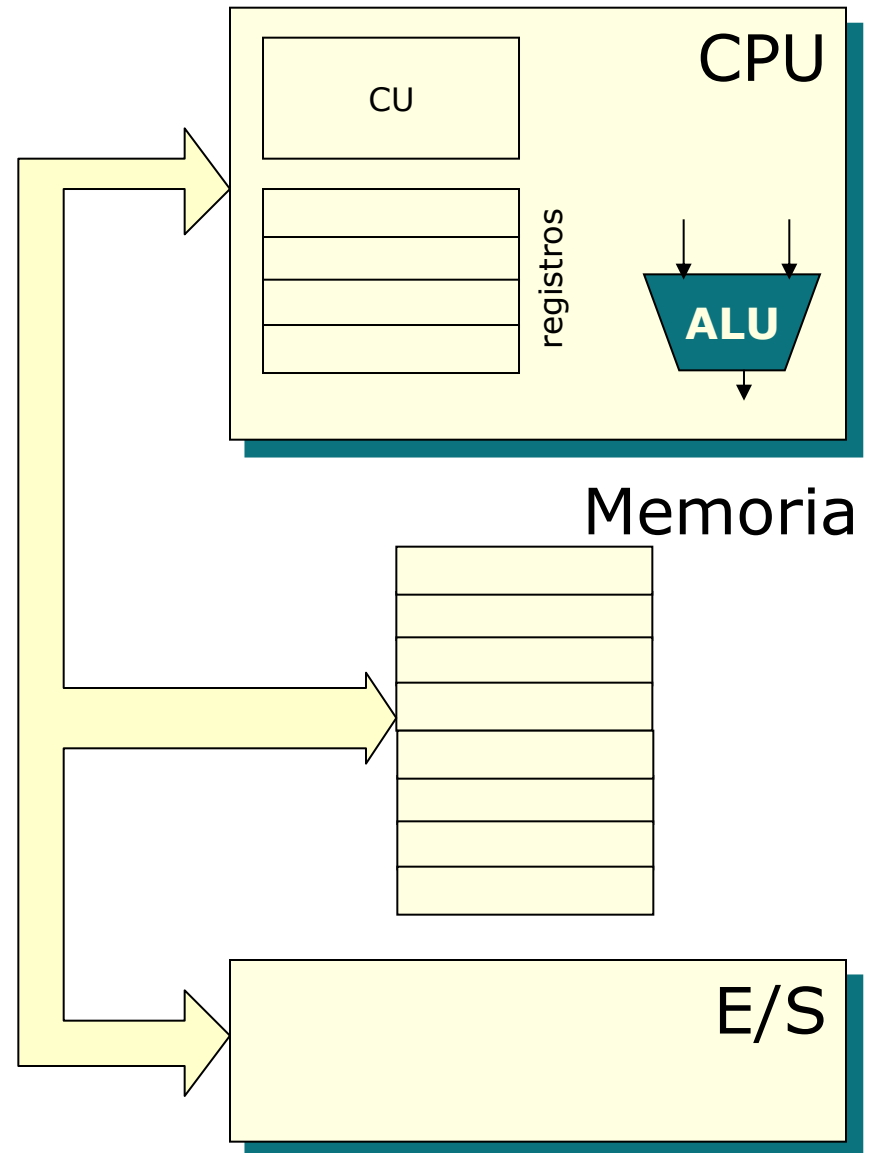
- Máquina de ejecución secuencial de instrucciones
 - Programa=secuencia de instrucciones
- Instrucciones y datos almacenados en memoria
 - Memoria organizada en direcciones
- Instrucciones, datos y operaciones en binario

Dirección	Contenido	
000..00	01101100	
000..01	00101110	
000..10	11111100	
	...	
Inst1:	MOV X,3	} Instrucciones codificadas en binario
Inst2:	MOV Z,X	
Inst3:	ADD Z,Y	
	...	
X:	0	} Datos codificados en binario
Y:	7	
Z:	2	
	...	



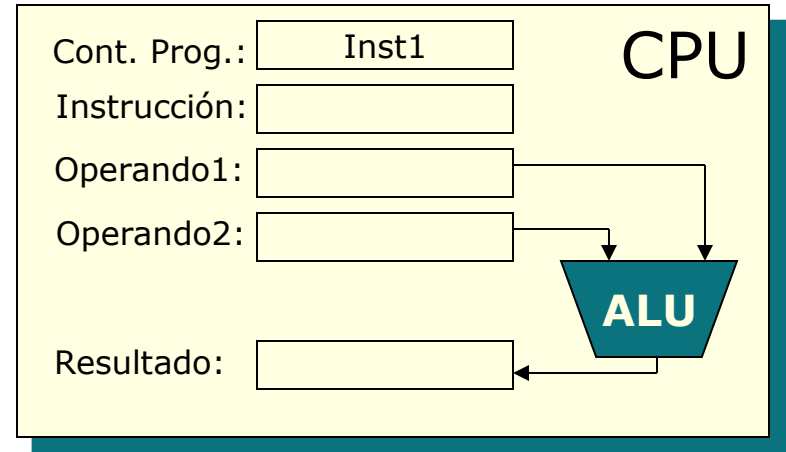
Partes del computador

- Unidad de Control (CU): organiza el funcionamiento del computador
- Unidad Aritmético Lógica (ALU): ejecuta operaciones aritméticas (+, -, *, /) y lógicas a nivel de bit (AND, OR, ...)
- Registros: lugares de almacenamiento temporal de información
- Memoria: lugar principal de almacenamiento de información (código y datos)
- Dispositivos de E/S: permiten la comunicación con el exterior



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

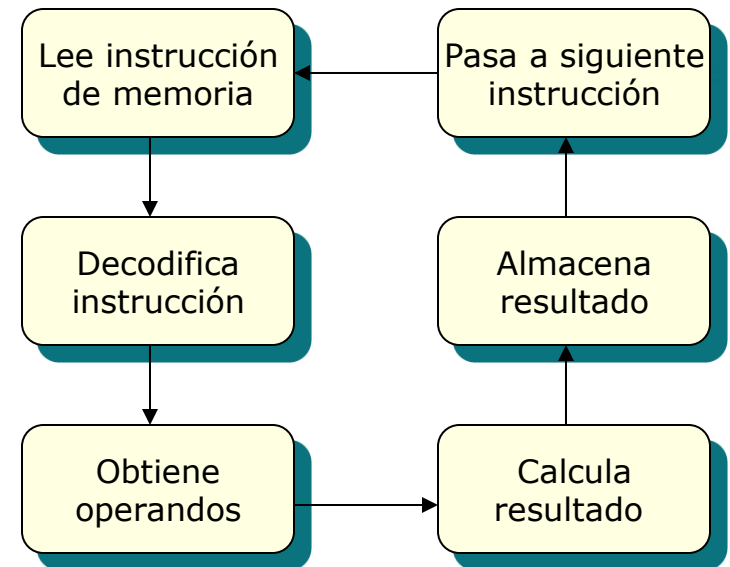


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000000
Y:	00000111
Z:	00000010
	...

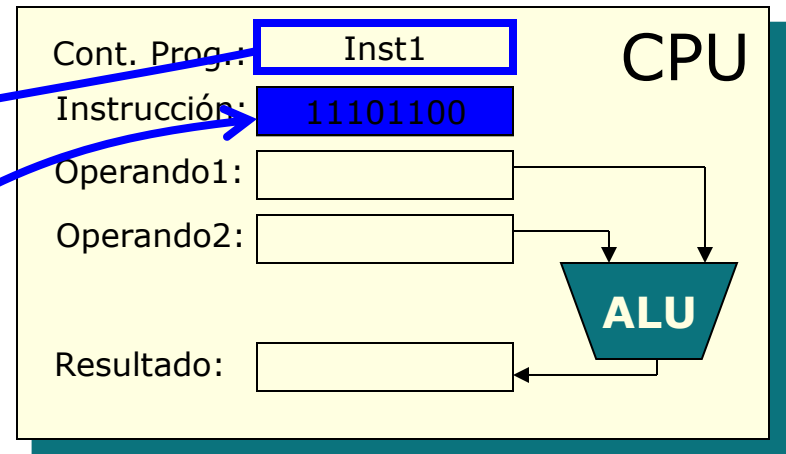
MOV X,3
MOV Z,X
ADD Z,Y

0
7
2



Ejecución de instrucciones

- Ejemplo: $x=3$; $z=x+y$;



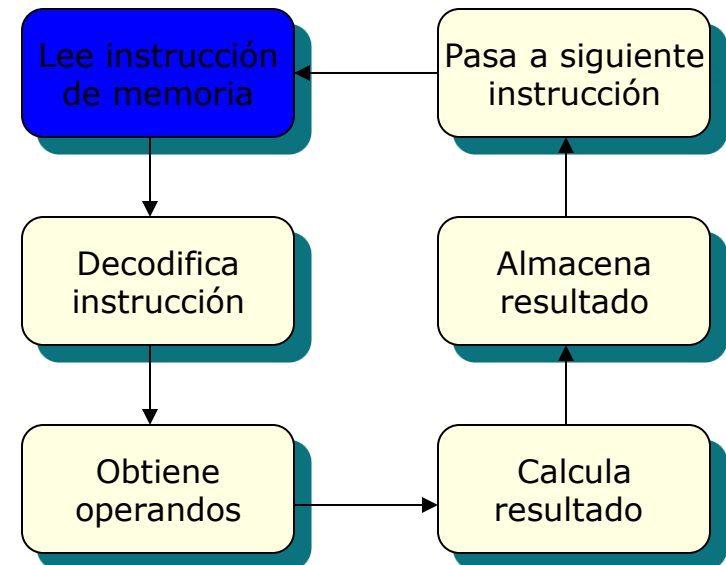
Dirección **Contenido**

000..00	01101100
000..01	00101110
000..10	11111100
...	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
...	...
X:	00000000
Y:	00000111
Z:	00000010
...	...

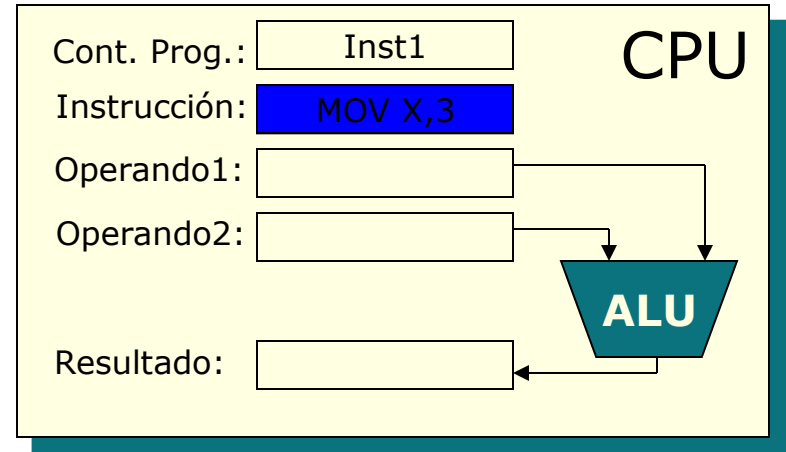
MOV X,3
MOV Z,X
ADD Z,Y

0
7
2



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

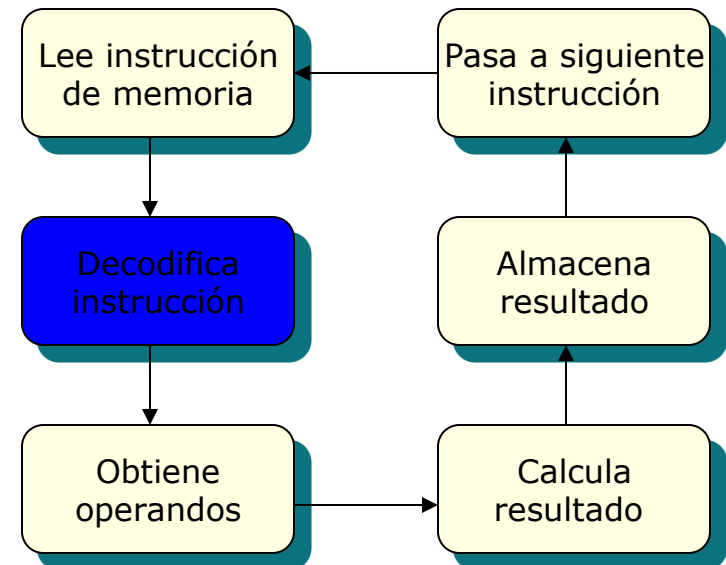


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000000
Y:	00000111
Z:	00000010
	...

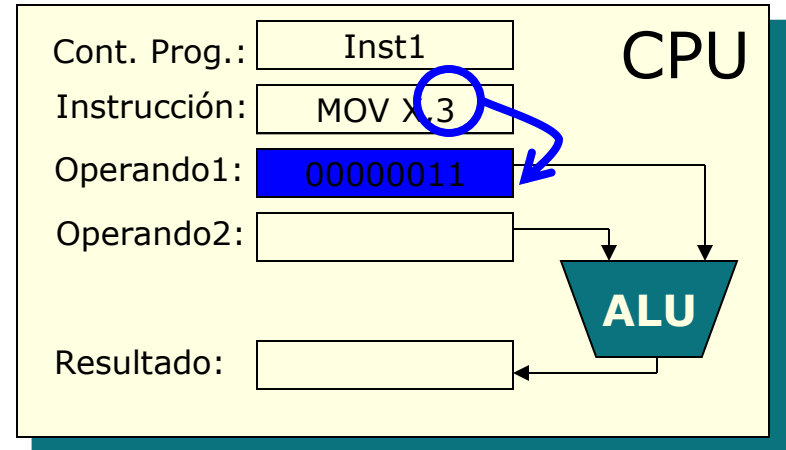
MOV X,3
MOV Z,X
ADD Z,Y

0
7
2



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

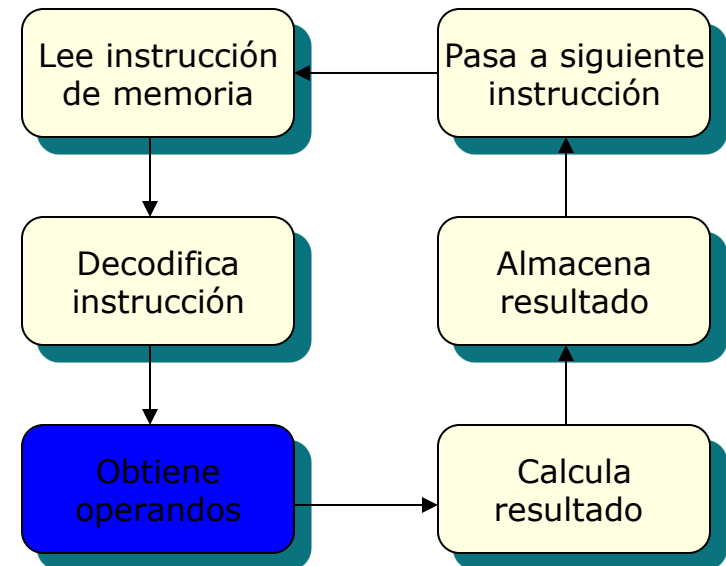


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000000
Y:	00000111
Z:	00000010
	...

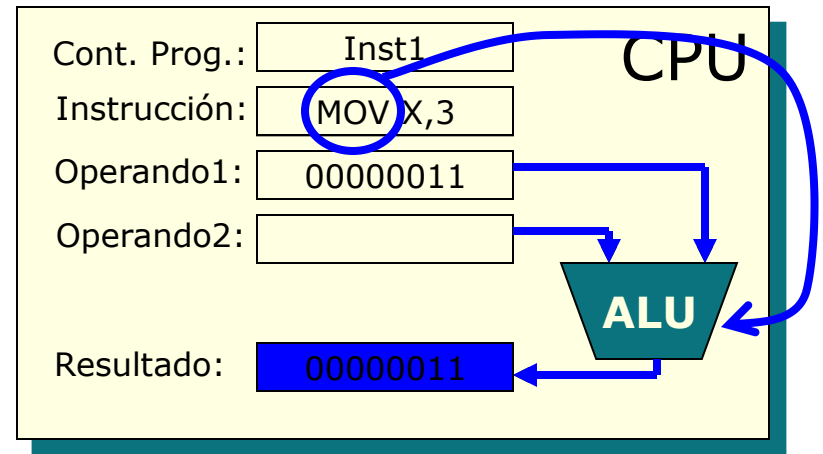
MOV X,3
MOV Z,X
ADD Z,Y

0
7
2



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

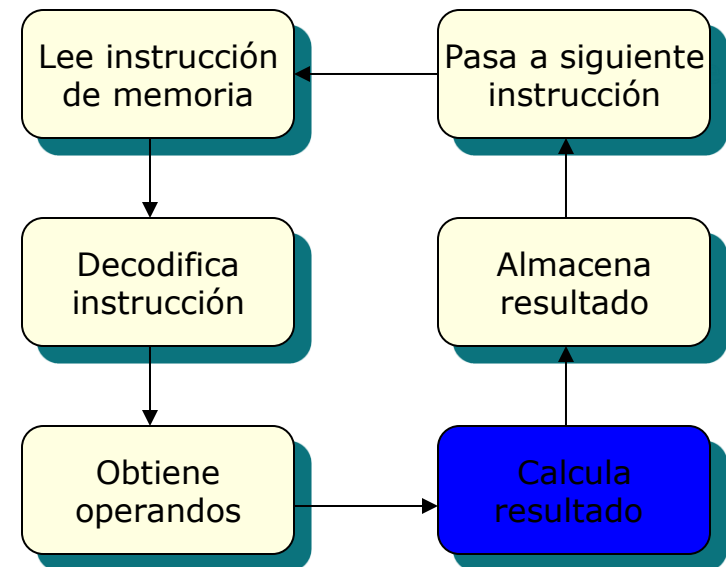


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000000
Y:	00000111
Z:	00000010
	...

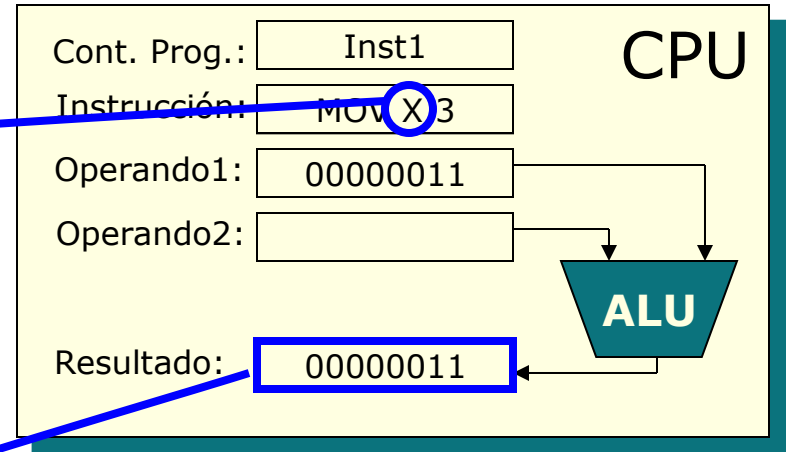
MOV X,3
MOV Z,X
ADD Z,Y

0
7
2



Ejecución de instrucciones

- Ejemplo: $x=3$; $z=x+y$;



Dirección **Contenido**

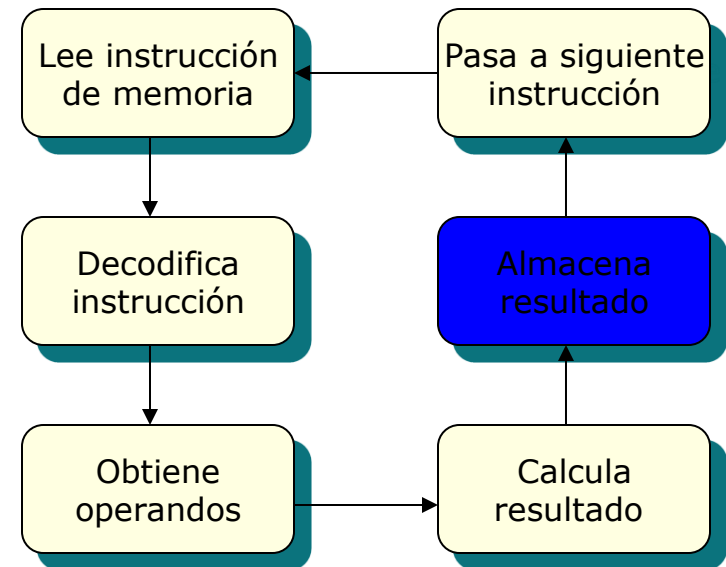
000..00	01101100
000..01	00101110
000..10	11111100
	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...

X:	00000011
Y:	00000111
Z:	00000010
	...

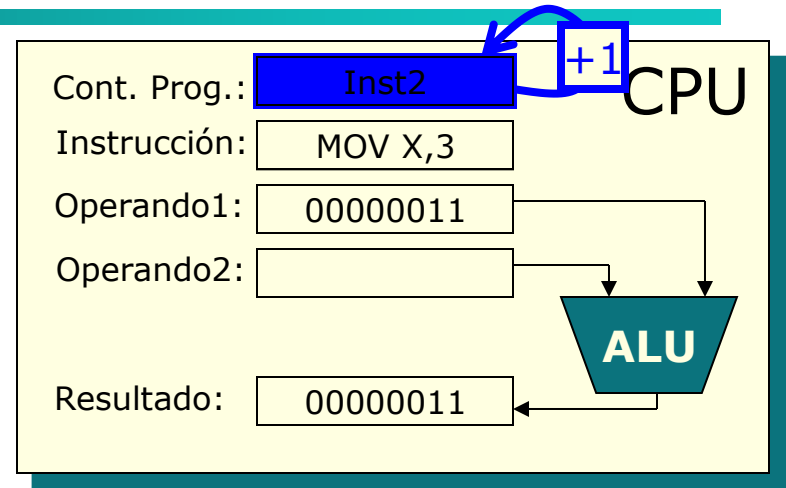
MOV X,3
MOV Z,X
ADD Z,Y

3
7
2



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

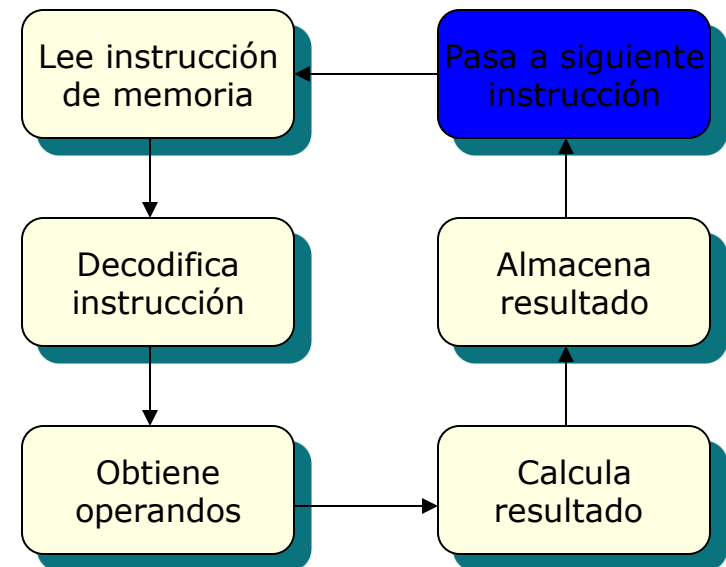


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000011
Y:	00000111
Z:	00000010
	...

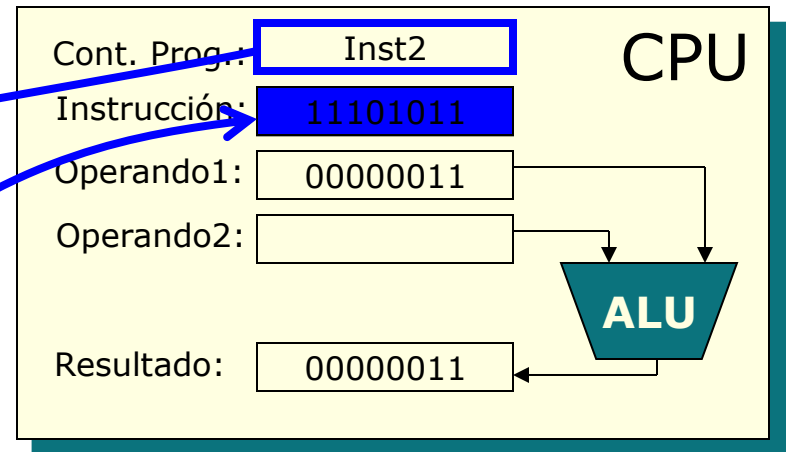
MOV X,3
MOV Z,X
ADD Z,Y

3
7
2



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$



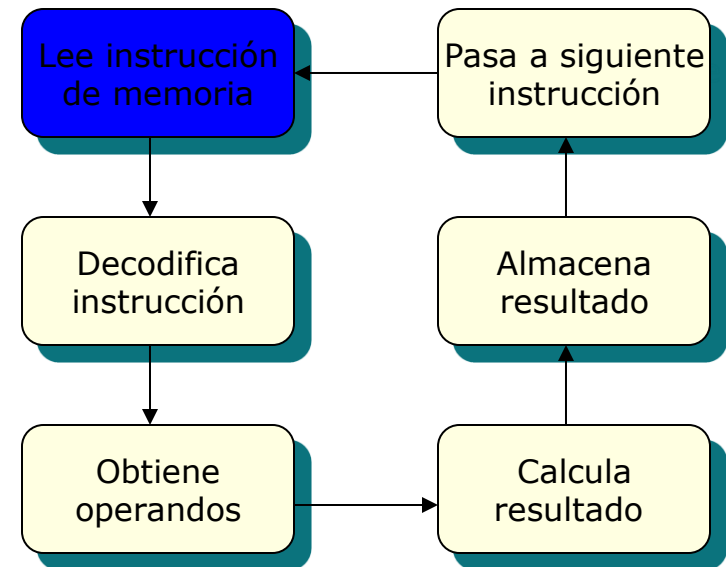
Dirección **Contenido**

000..00	01101100
000..01	00101110
000..10	11111100
...	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
...	...
X:	00000011
Y:	00000111
Z:	00000010
...	...

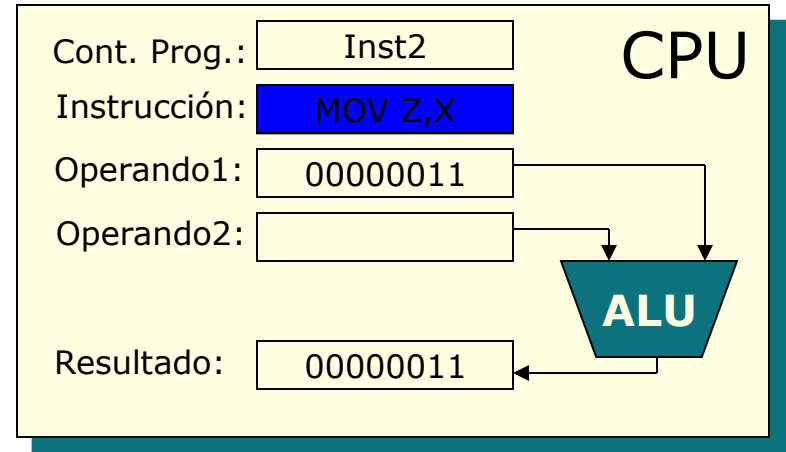
MOV X,3
MOV Z,X
ADD Z,Y

3
7
2



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

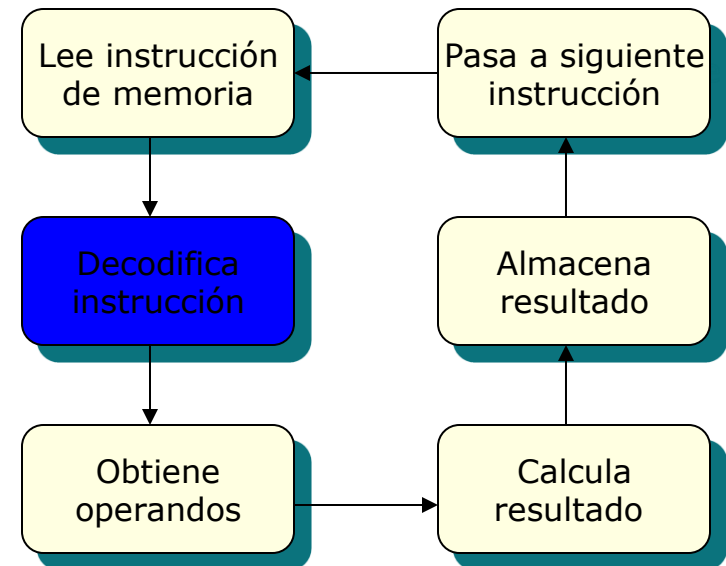


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000011
Y:	00000111
Z:	00000010
	...

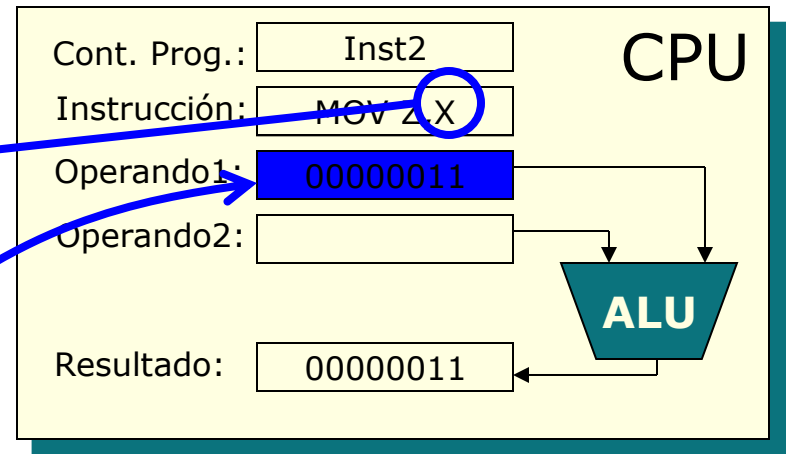
MOV X,3
MOV Z,X
ADD Z,Y

3
7
2



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$



Dirección **Contenido**

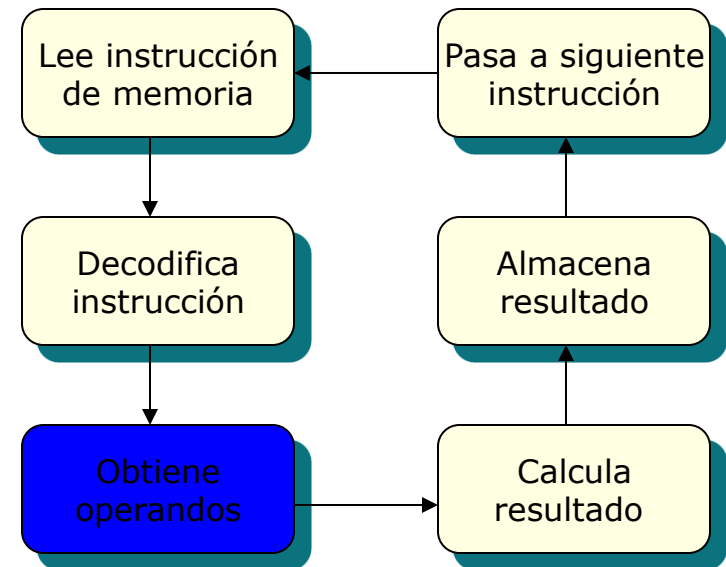
000..00	01101100
000..01	00101110
000..10	11111100
	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...

X:	00000011
Y:	00000111
Z:	00000010
	...

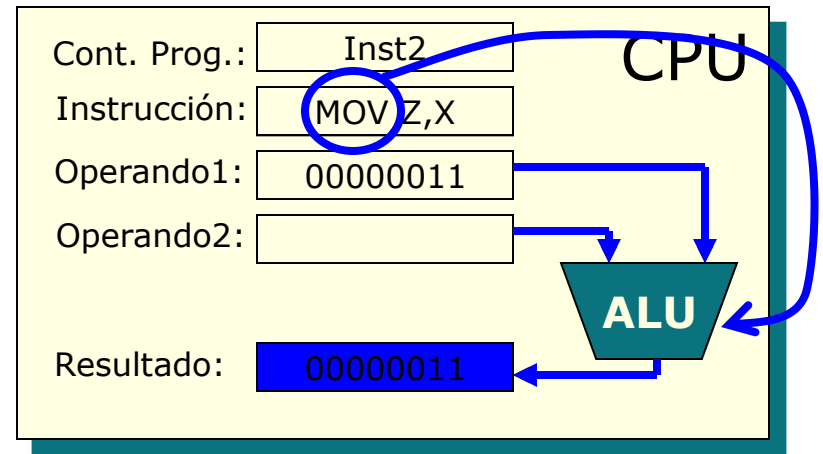
MOV X,3
MOV Z,X
ADD Z,Y

3
7
2



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

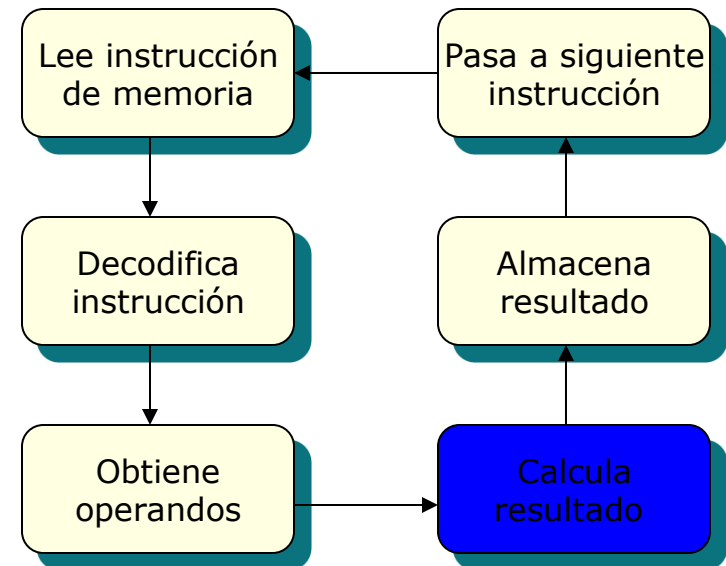


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000011
Y:	00000111
Z:	00000010
	...

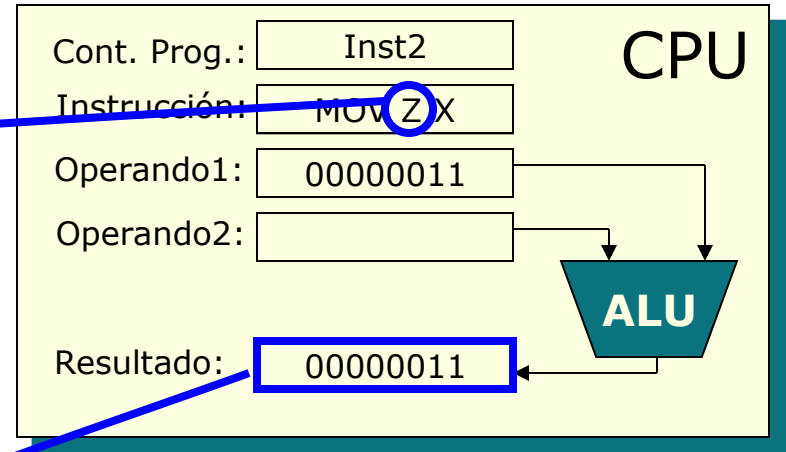
MOV X,3
MOV Z,X
ADD Z,Y

3
7
2



Ejecución de instrucciones

- Ejemplo: $x=3$; $z=x+y$;



Dirección **Contenido**

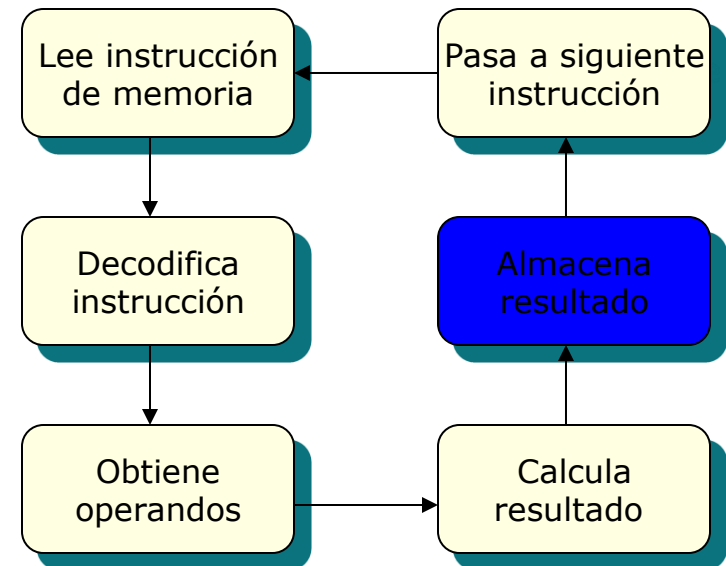
000..00	01101100
000..01	00101110
000..10	11111100
	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...

X:	00000011
Y:	00000111
Z:	00000011
	...

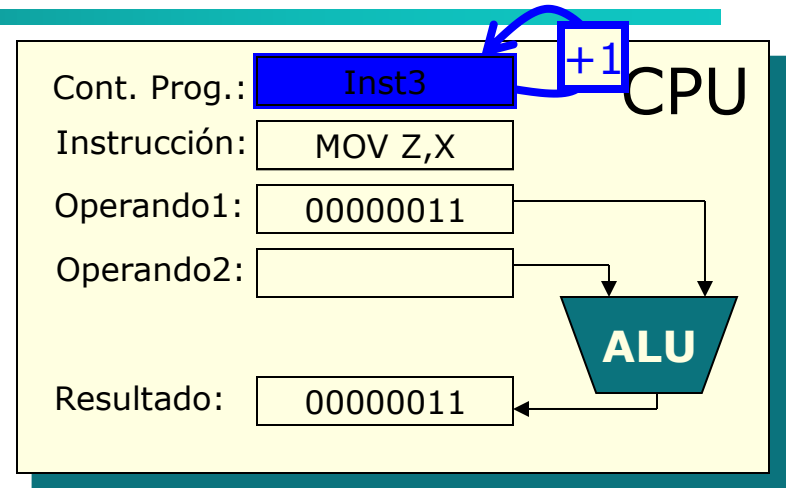
MOV X,3
MOV Z,X
ADD Z,Y

3
7
3



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

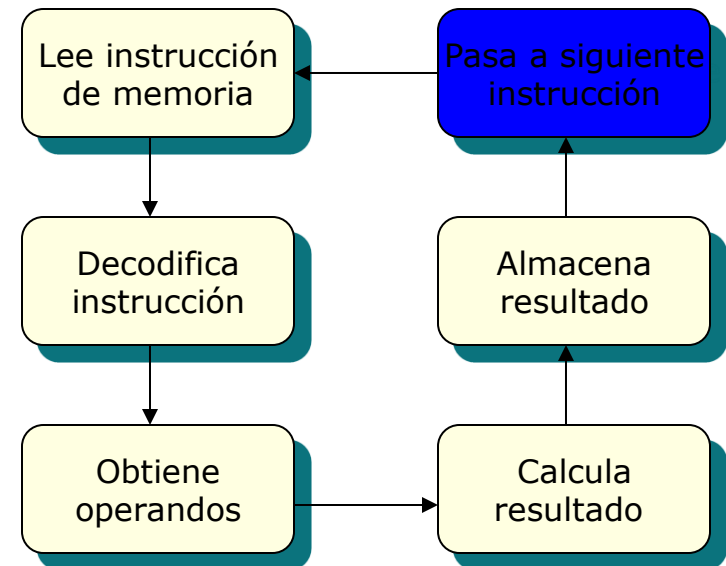


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000011
Y:	00000111
Z:	00000011
	...

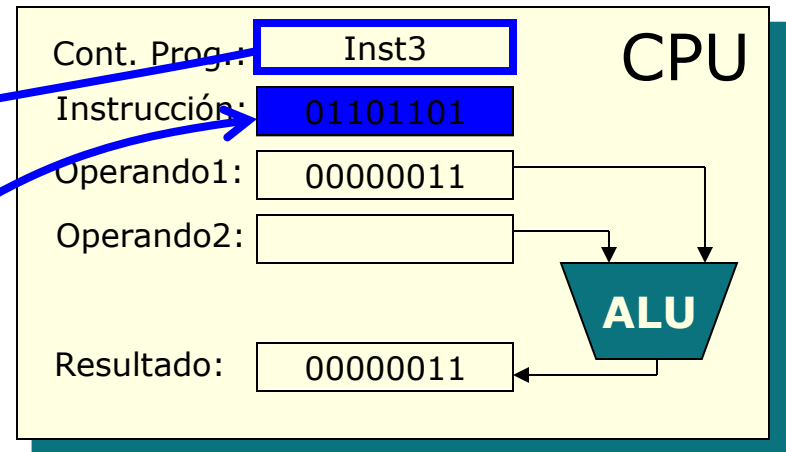
MOV X,3
MOV Z,X
ADD Z,Y

3
7
3



Ejecución de instrucciones

- Ejemplo: $x=3$; $z=x+y$;



Dirección **Contenido**

000..00	01101100
000..01	00101110
000..10	11111100
	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...

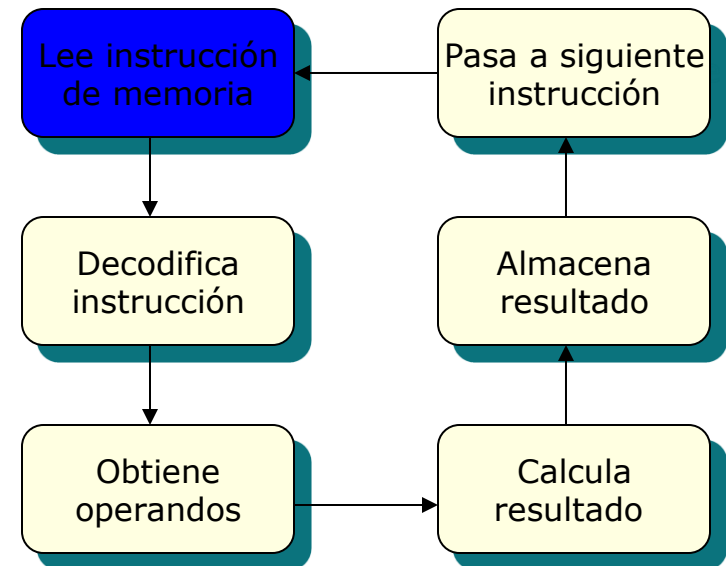
X:	00000011
Y:	00000111
Z:	00000011
	...

MOV X,3
MOV Z,X
ADD Z,Y

3

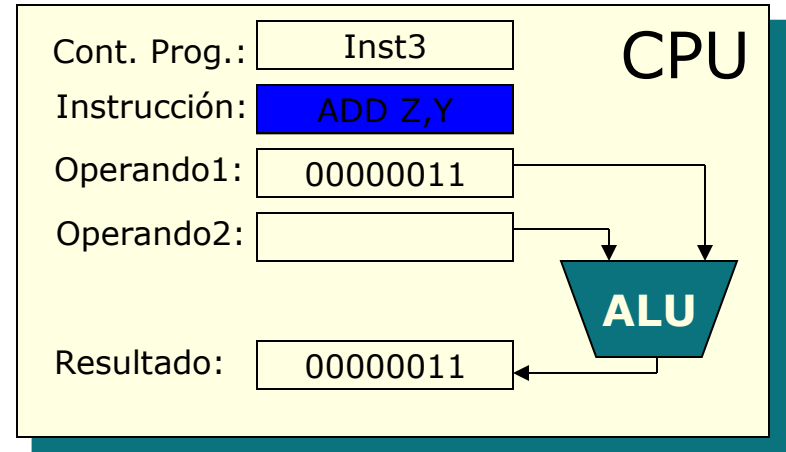
7

3



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$



Dirección Contenido

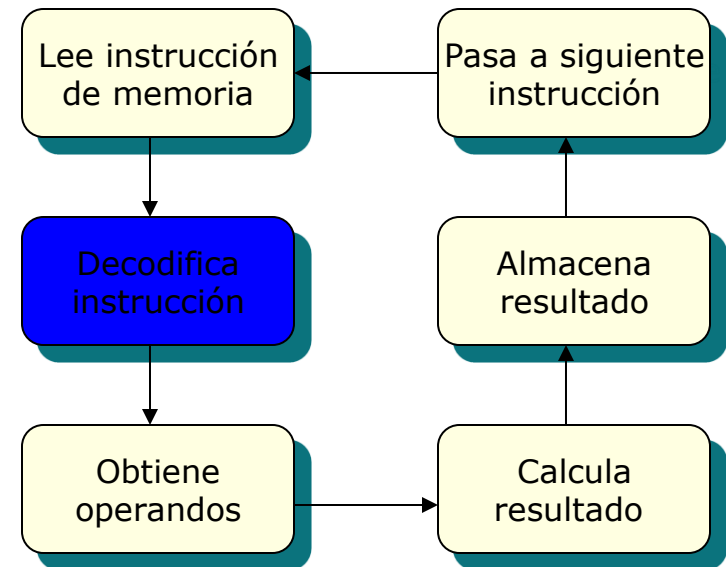
000..00	01101100
000..01	00101110
000..10	11111100
	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...

X:	00000011
Y:	00000111
Z:	00000011
	...

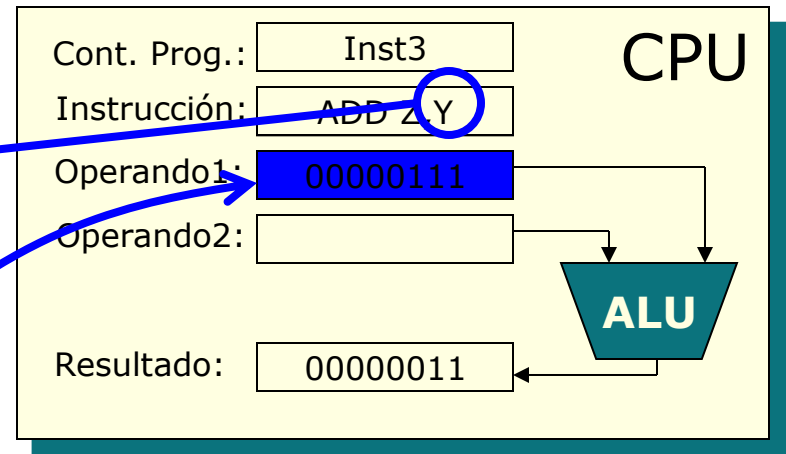
MOV X,3
MOV Z,X
ADD Z,Y

3
7
3



Ejecución de instrucciones

- Ejemplo: $x=3$; $z=x+y$;



Dirección **Contenido**

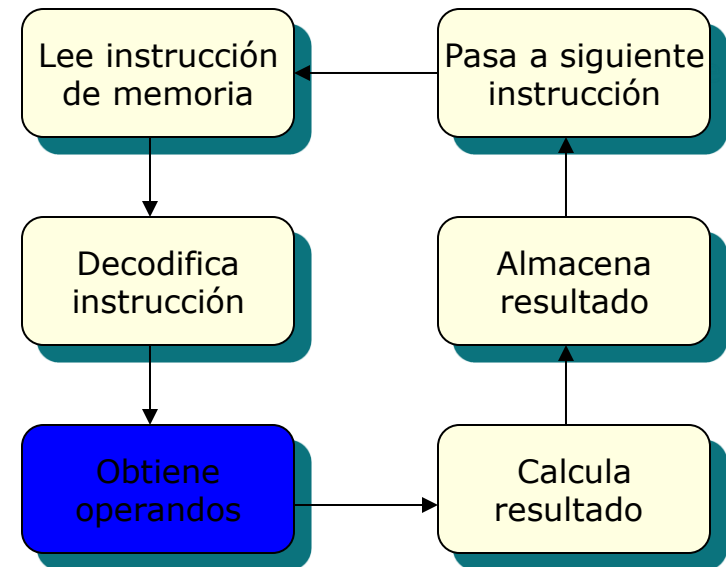
000..00	01101100
000..01	00101110
000..10	11111100
	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...

X:	00000011
Y:	00000111
Z:	00000011
	...

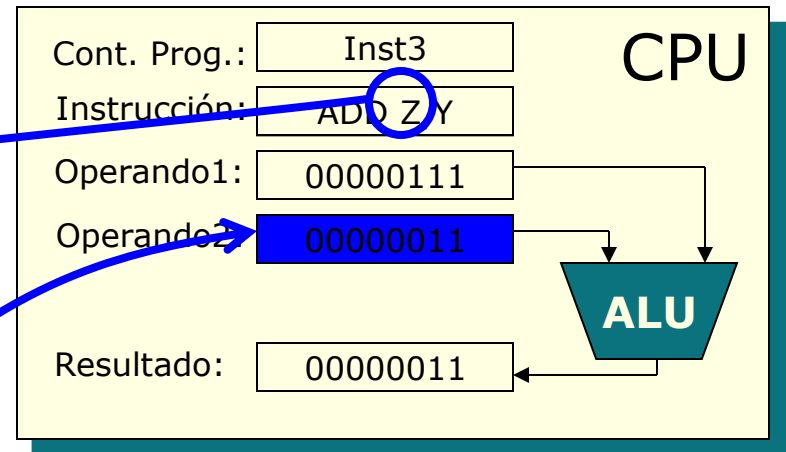
MOV X,3
MOV Z,X
ADD Z,Y

3
7
3



Ejecución de instrucciones

- Ejemplo: $x=3$; $z=x+y$;



Dirección **Contenido**

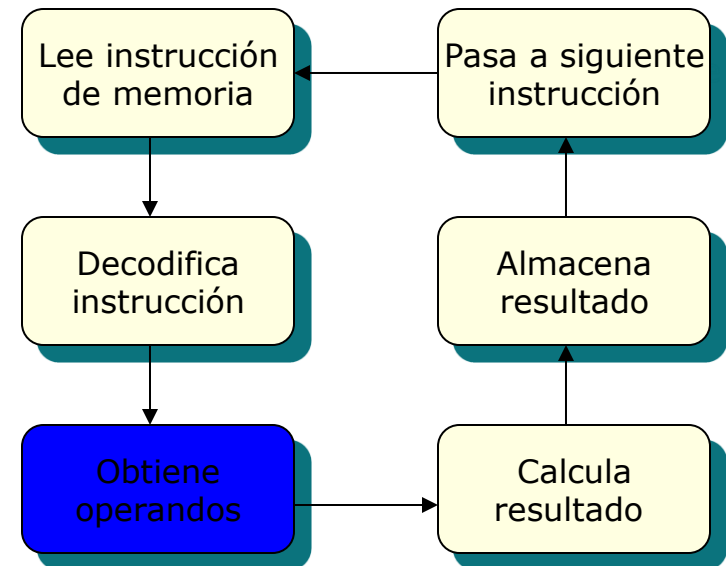
000..00	01101100
000..01	00101110
000..10	11111100
...	

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
...	

X:	00000011
Y:	00000111
Z:	00000011
...	

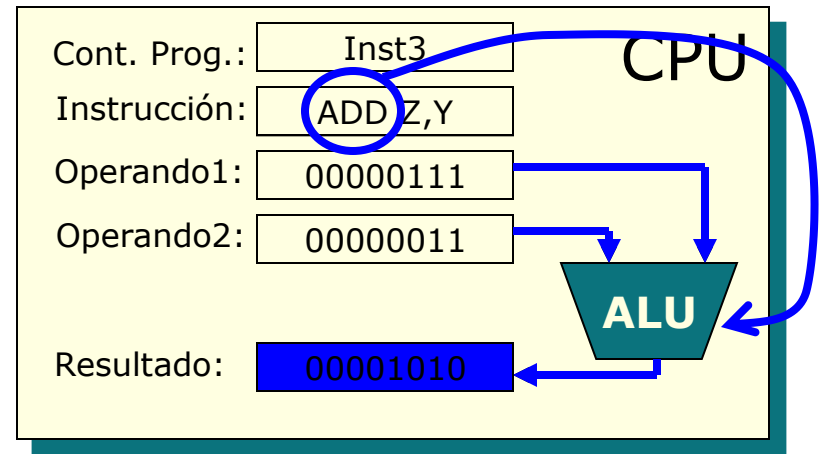
MOV X,3
MOV Z,X
ADD Z,Y

3
7
3



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

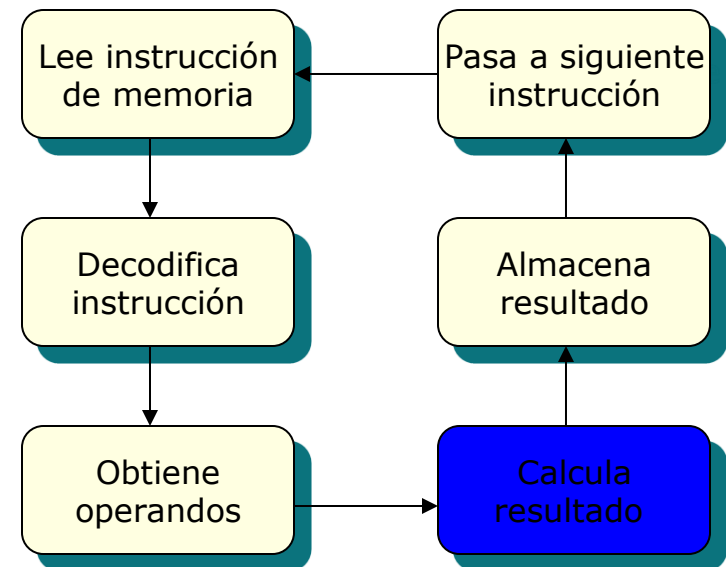


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000011
Y:	00000111
Z:	00000011
	...

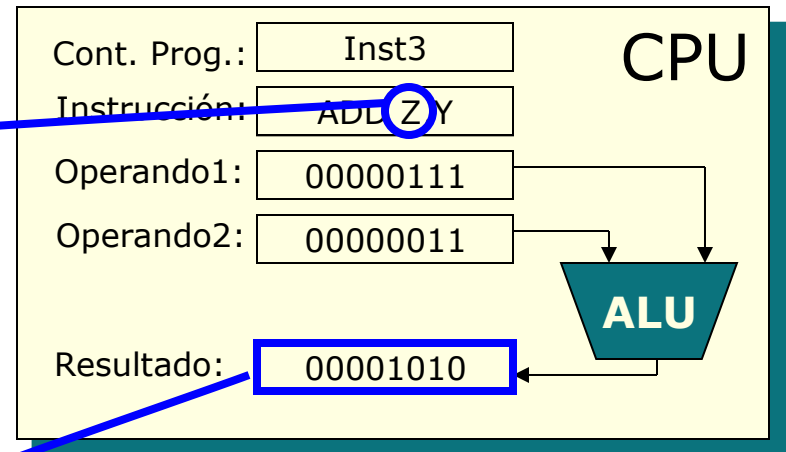
MOV X,3
MOV Z,X
ADD Z,Y

3
7
3



Ejecución de instrucciones

- Ejemplo: $x=3$; $z=x+y$;



Dirección **Contenido**

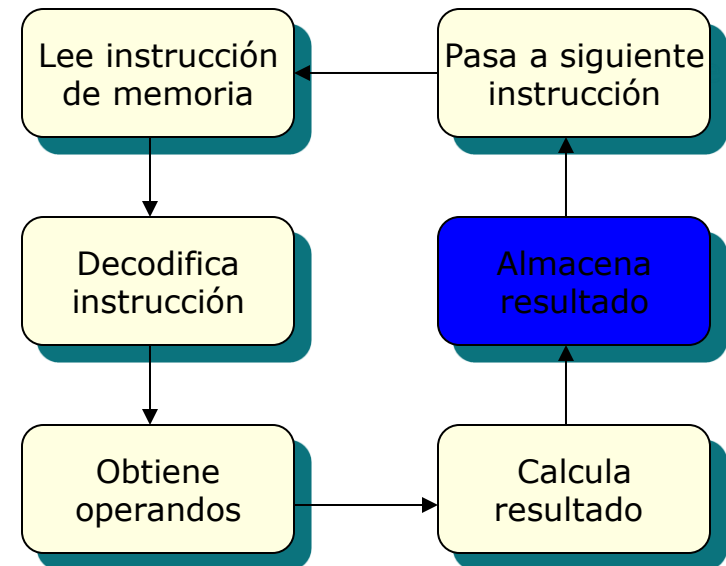
000..00	01101100
000..01	00101110
000..10	11111100
	...

Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...

X:	00000011
Y:	00000111
Z:	00001010
	...

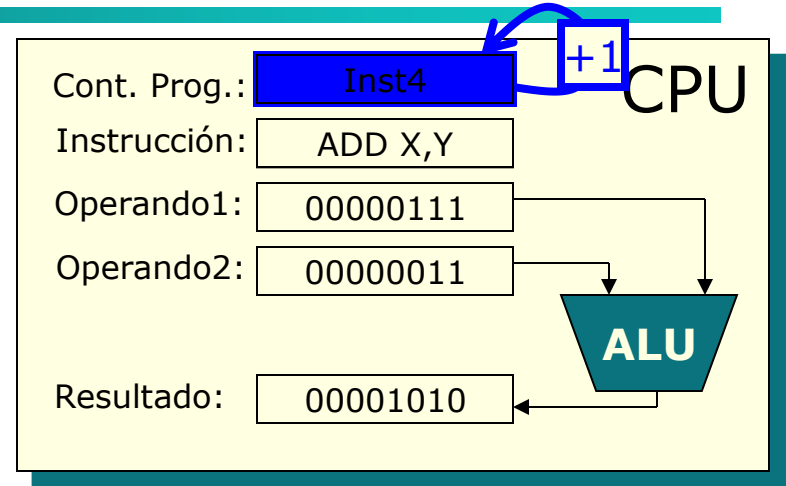
MOV X,3
MOV Z,X
ADD Z,Y

3
7
10



Ejecución de instrucciones

- Ejemplo: $x=3; z=x+y;$

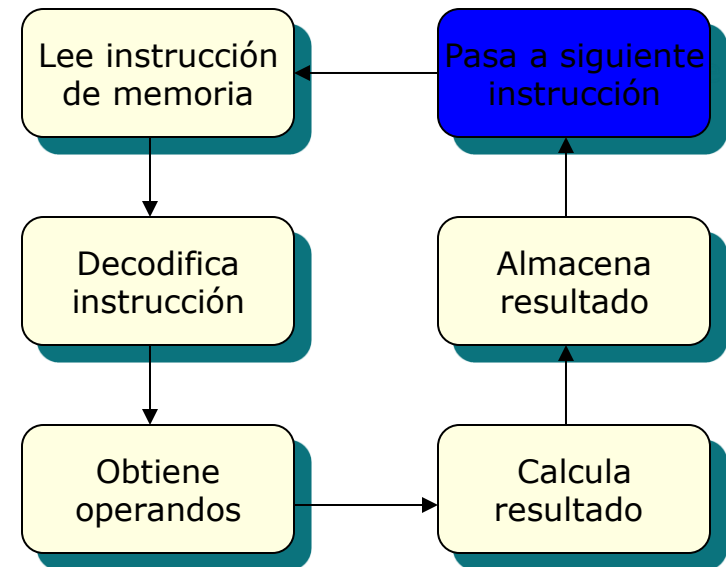


Dirección Contenido

000..00	01101100
000..01	00101110
000..10	11111100
	...
Inst1:	11101100
Inst2:	11101011
Inst3:	01101101
	...
X:	00000011
Y:	00000111
Z:	00001010
	...

MOV X,3
MOV Z,X
ADD Z,Y

3
7
10



Lenguajes de alto y bajo nivel

- El computador ejecuta lenguaje máquina:
 - Secuencias de 0s y 1s interpretadas como instrucciones
 - Instrucciones muy sencillas (copiar, sumar, ...)
 - Difícil de escribir, leer y depurar por el hombre
 - Fácil de almacenar e interpretar por el computador
- El programador escribe lenguaje de alto nivel:
 - Lenguaje textual con instrucciones no ejecutables directamente por el computador
 - Instrucciones más complejas
 - Fácil de escribir, leer y depurar por el hombre
 - Necesita un programa compilador o intérprete para convertir a secuencia de instrucciones de máquina

Lenguajes de alto y bajo nivel

- Ejemplo:

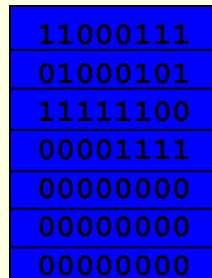
Programa de alto nivel

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    int i;
    float x;

    i=7;
    x=1;
    while (i>=1)
    {
        x=x*i;
        i=i-1;
    }

    printf("El factorial es %f",x);
}
```



11000111
01000101
11111100
00001111
00000000
00000000
00000000
00000000

Programa de máquina

```
mov     DWORD PTR _i$[ebp], 7
mov     DWORD PTR _x$[ebp], 1F
$L906:  cmp     DWORD PTR _i$[ebp], 1
        jl     SHORT $L907
        fld     DWORD PTR _i$[ebp]
        fmul    DWORD PTR _x$[ebp]
        fstp    DWORD PTR _x$[ebp]
        mov     eax, DWORD PTR _i$[ebp]
        sub     eax, 1
        mov     DWORD PTR _i$[ebp], eax
        jmp     SHORT $L906
$L907:  fld     DWORD PTR _x$[ebp]
        sub     esp, 8
        fstp    QWORD PTR [esp]
        push    OFFSET FACTORIAL
        call    _printf
        add     esp, 12
```

Alto nivel

- Ventajas

- Programación más cerca del lenguaje humano y matemático
- Lenguajes estructurados y tipados
- Facilidad de escritura, lectura y depuración
- La complejidad se deja al compilador
- Código portable de una máquina a otra

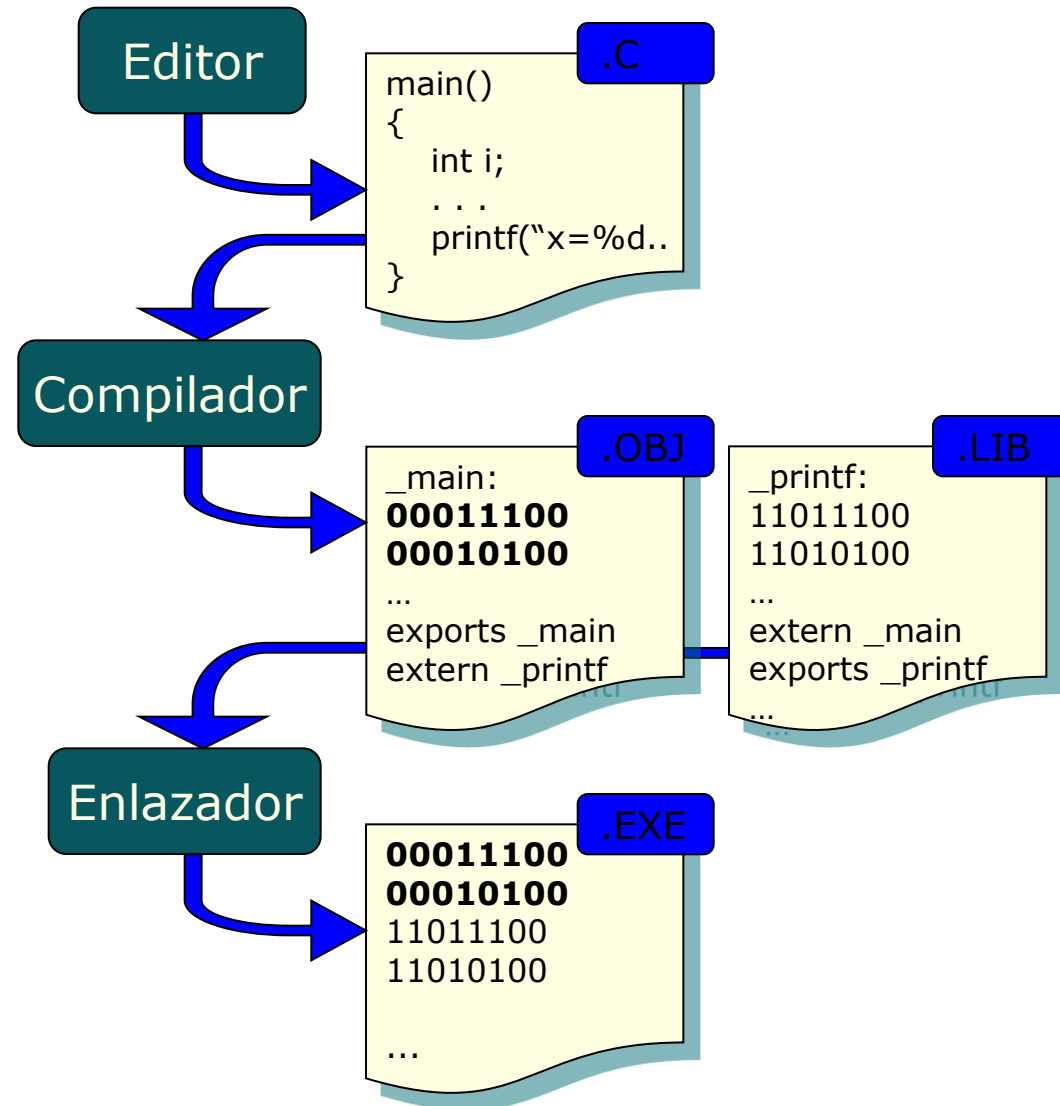
- Inconvenientes

- Dificultad de acceso a todos los recursos de la máquina
- El código máquina resultante suele ser más grande y lento

Alto nivel → código máquina

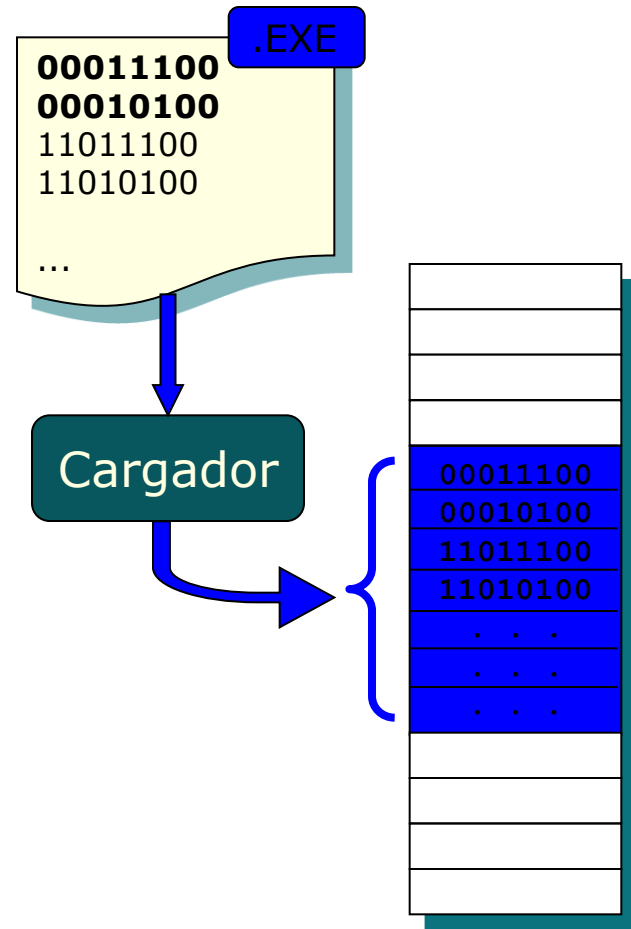
- Pasos en la obtención de código máquina:

- Escribir código fuente (programa editor)
- Compilar código fuente (programa compilador)
- Enlazar código objeto y librerías (programa enlazador)



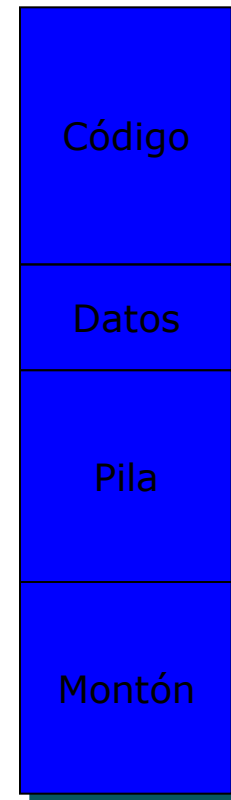
Ejecución del programa

- Ejecución de un programa:
 - Cargar en memoria el archivo ejecutable (programa cargador)
 - Poner en el PC la dirección de la 1ª instrucción del programa
 - O bien cargar en memoria y ejecutar paso a paso con un programa depurador



El programa en memoria

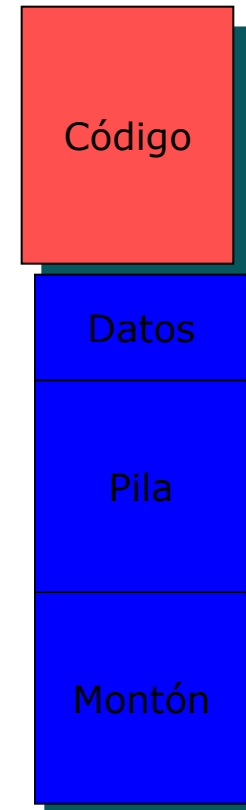
- Partes del programa:
 - Código
 - Conjunto de funciones
 - Datos
 - Variables globales (datos)
 - Variables locales: Pila (stack)
 - Variables de almacenamiento dinámico: Montón (heap)



La zona de memoria de código

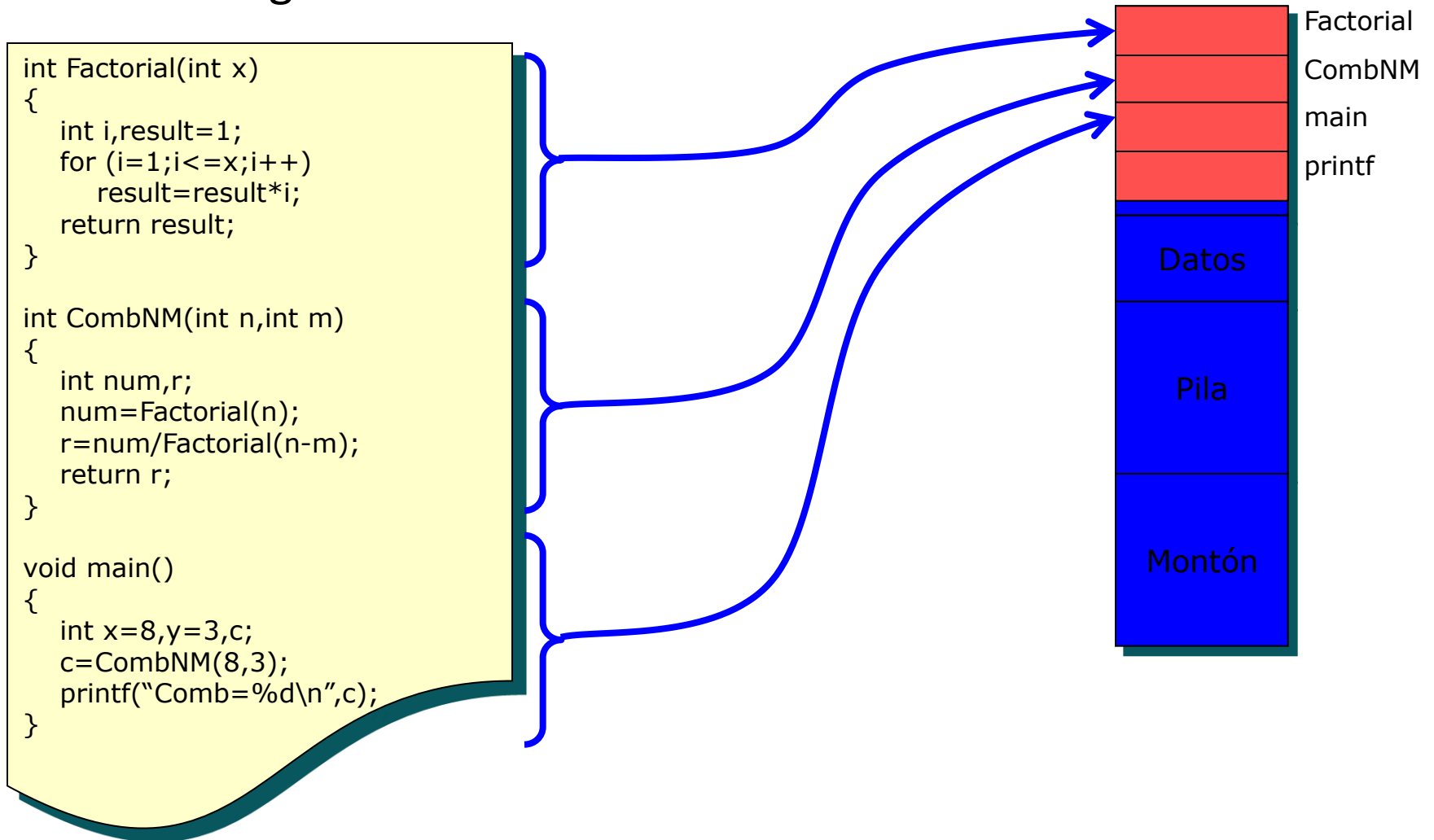
- El código está dividido en funciones
 - Función: conjunto de instrucciones que cumplen un objetivo común, separable del resto y, en lo posible, reutilizable.
 - Elementos de la función:
 - Parámetros
 - Variables locales
 - Sentencias de ejecución
 - Valor devuelto

```
tipodvto Funcion(tipo1 par1,tipo2 par2, ...)  
{  
    tipoL1 vloc1,vloc2,vloc3;  
    tipoL2 vloc4,vloc5;  
    ...  
    Sentencias;  
  
    return valor;  
}
```



La zona de memoria de código

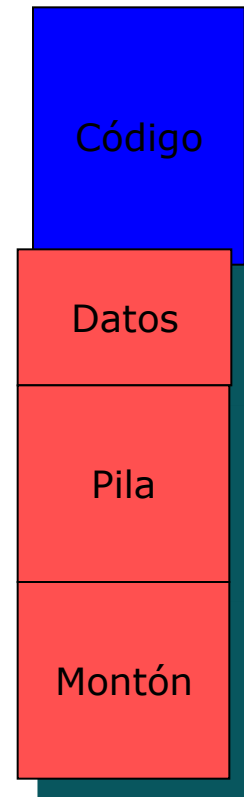
- El código está dividido en funciones



La zona de memoria de datos

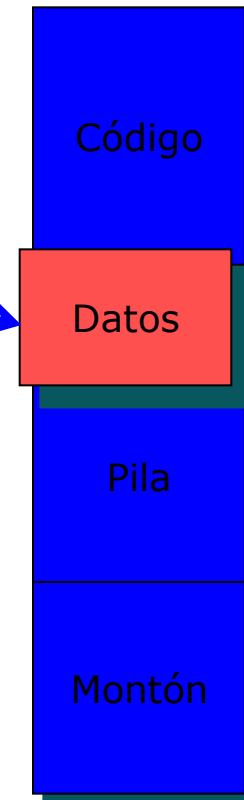
- Los datos son las variables utilizadas:
 - Variable: zona reservada en direcciones de memoria consecutivas, cuyo contenido puede ser modificado y reutilizado por el programa.
 - Tipos de variables:
 - Variables globales
 - Variables locales
 - Parámetros de función
 - Variables de almacenamiento dinámico

```
tipoG1 vglob1,vglob2;  
  
tipodvto Funcion(tipo1 par1,tipo2 par2, ...)  
{  
    tipoL1 vloc1,vloc2,vloc3;  
    tipoL2 vloc4,vloc5;  
    ...  
}
```



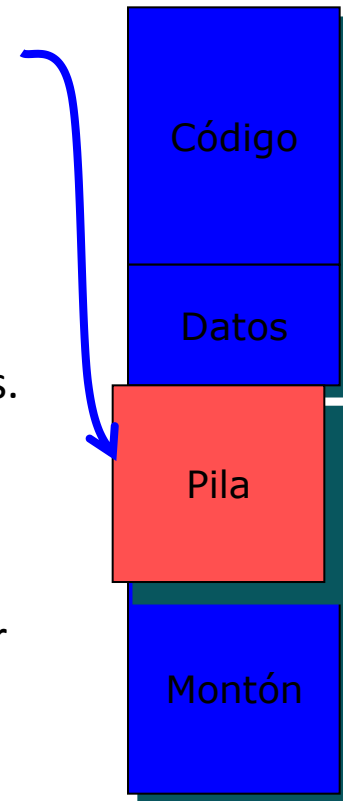
La zona de memoria de datos

- Los datos son las variables utilizadas:
 - Variables globales:
 - Se alojan en el segmento de datos
 - Su espacio se reserva al cargar el programa y se mantiene en toda su duración
 - Todas las funciones tienen acceso a estas variables



La zona de memoria de datos

- Los datos son las variables utilizadas:
 - Variables locales y parámetros de función:
 - Se alojan en el segmento de pila.
 - Su espacio se reserva al empezar a ejecutar la función en que están declaradas, y se libera al terminar su ejecución.
 - Sólo la función que las declara puede utilizarlas.
 - Las variables locales y parámetros se van apilando en memoria al llamar a otras funciones, y desapilando al salir.
 - Los parámetros son variables locales cuyo valor inicial se da en la llamada a la función.



La zona de memoria de datos

- El funcionamiento de la pila

```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```



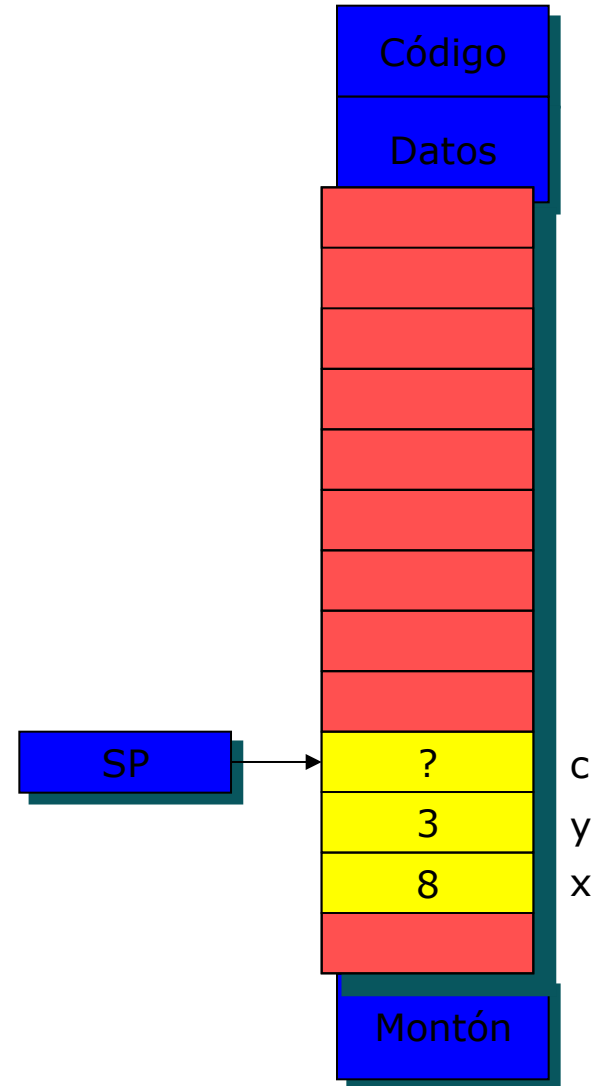
La zona de memoria de datos

- El funcionamiento de la pila

```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```



La zona de memoria de datos

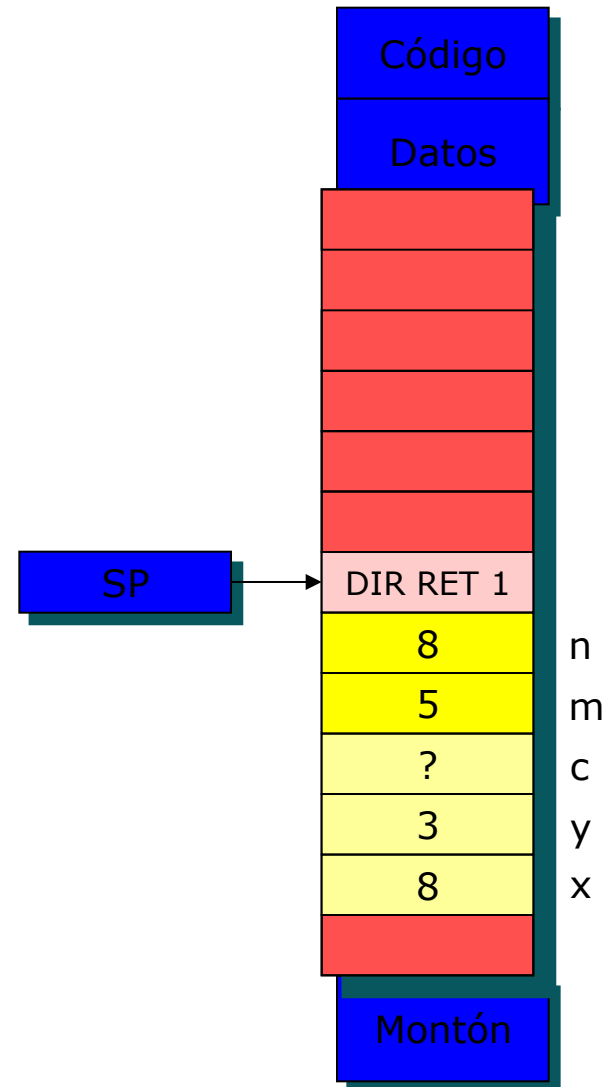
- El funcionamiento de la pila

```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 1:



La zona de memoria de datos

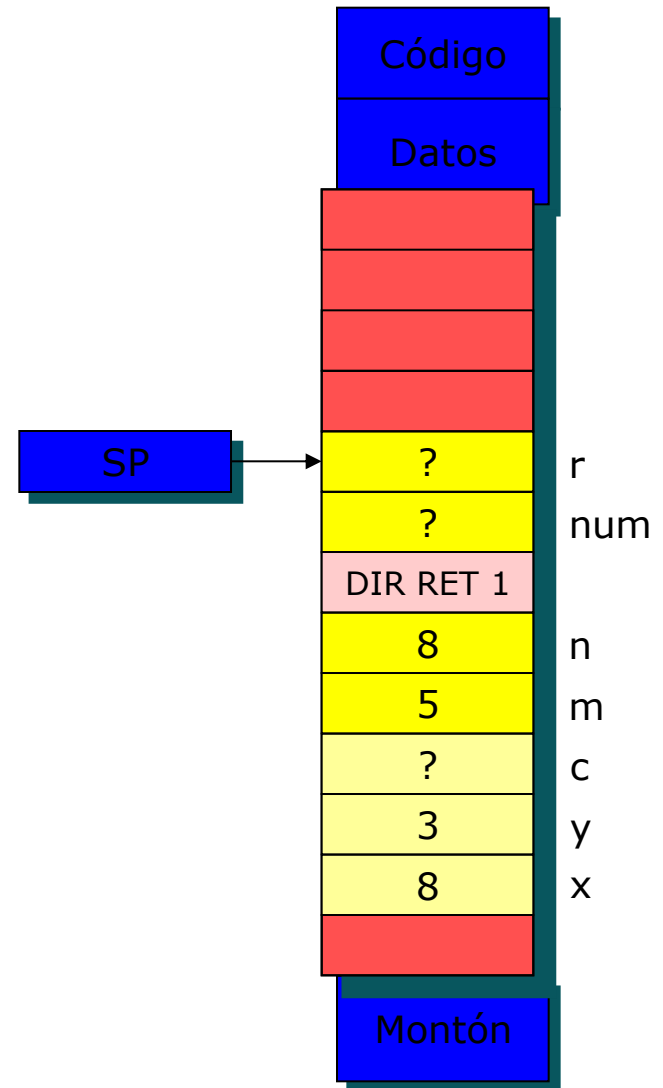
- El funcionamiento de la pila

```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 1:



La zona de memoria de datos

- El funcionamiento de la pila

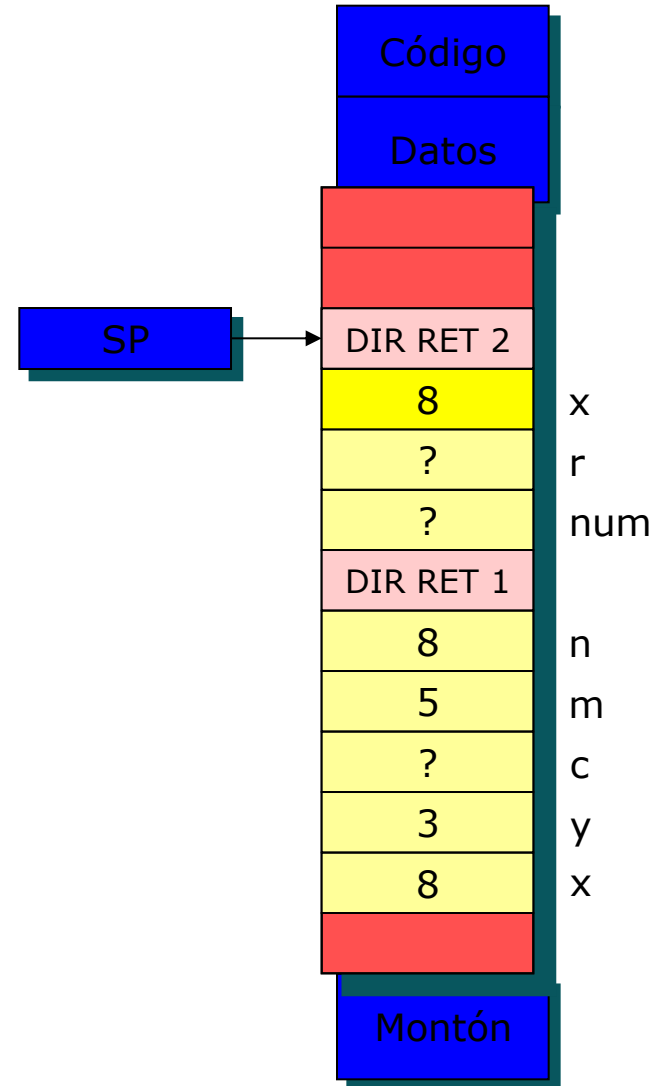
```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 2:

DIR RET 1:



La zona de memoria de datos

- El funcionamiento de la pila

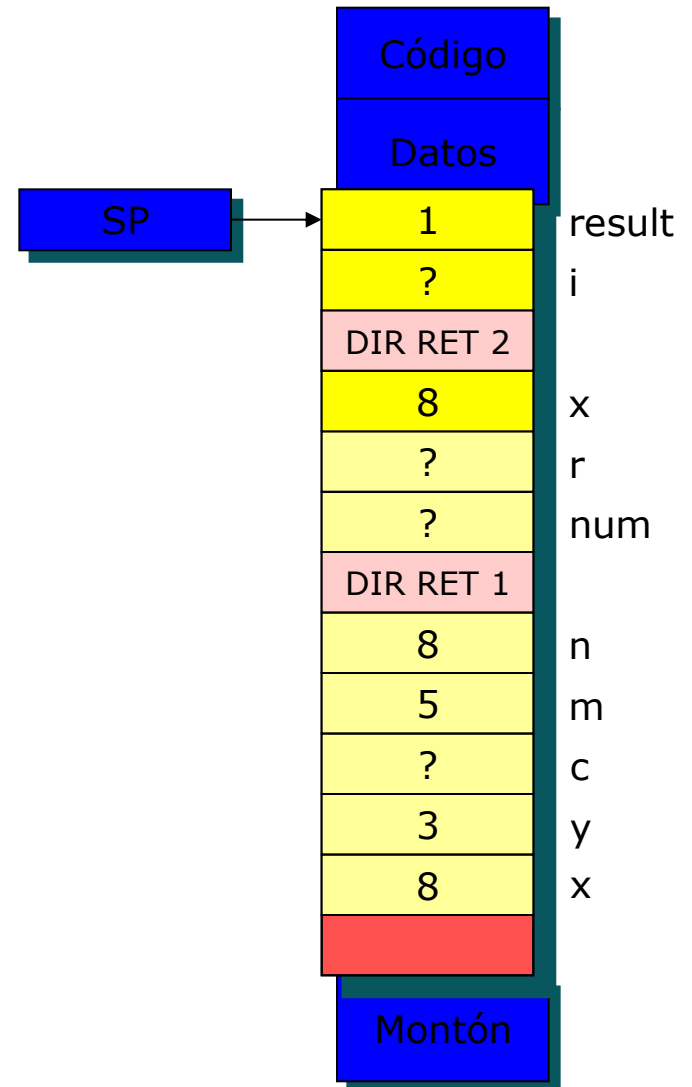
```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 2:

DIR RET 1:



La zona de memoria de datos

- El funcionamiento de la pila

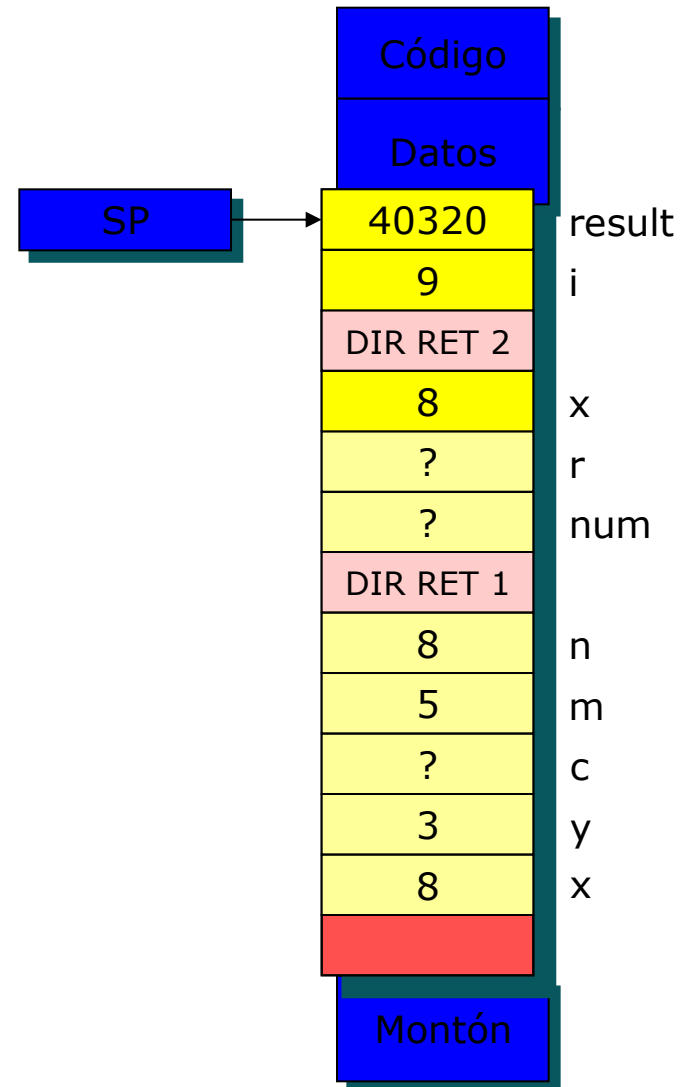
```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 2:

DIR RET 1:



La zona de memoria de datos

- El funcionamiento de la pila

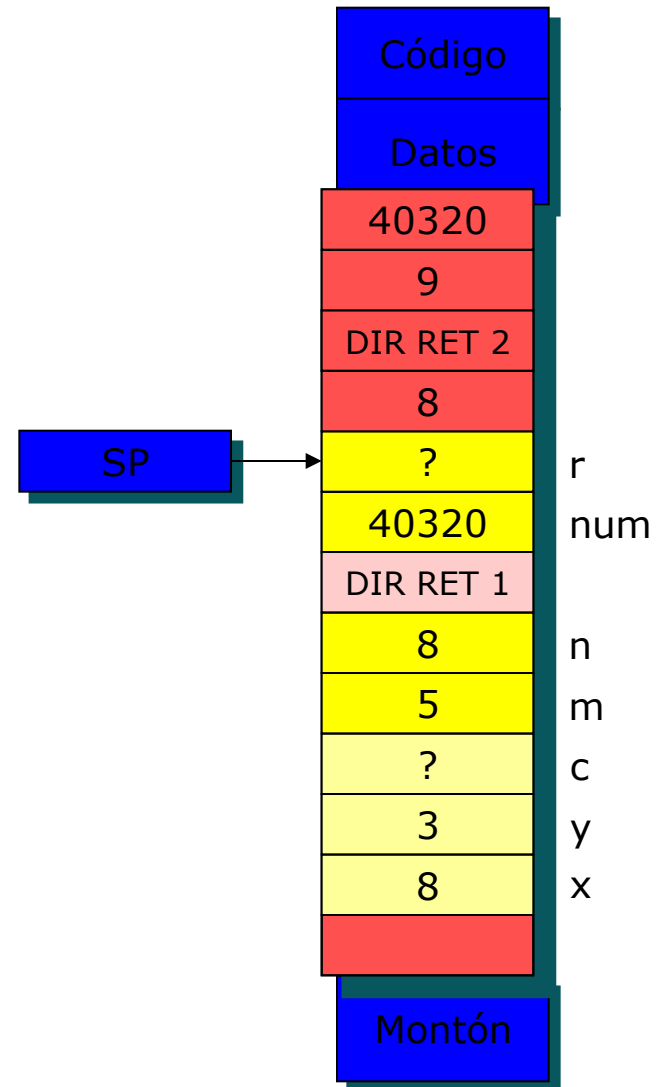
```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET →

DIR RET 1:



La zona de memoria de datos

- El funcionamiento de la pila

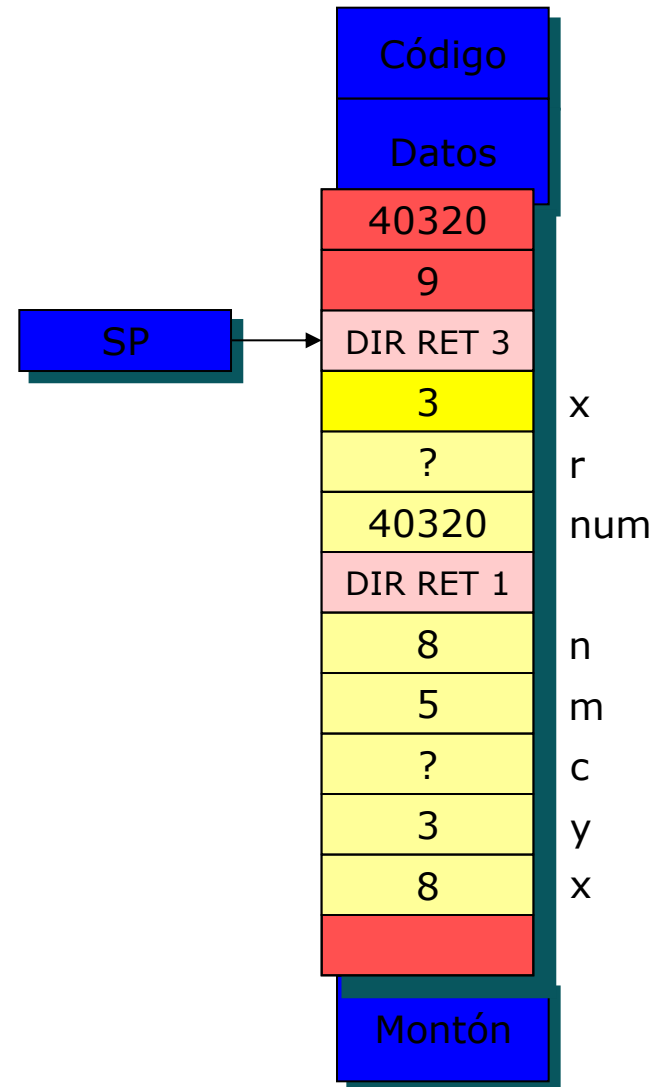
```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 3:

DIR RET 1:



La zona de memoria de datos

- El funcionamiento de la pila

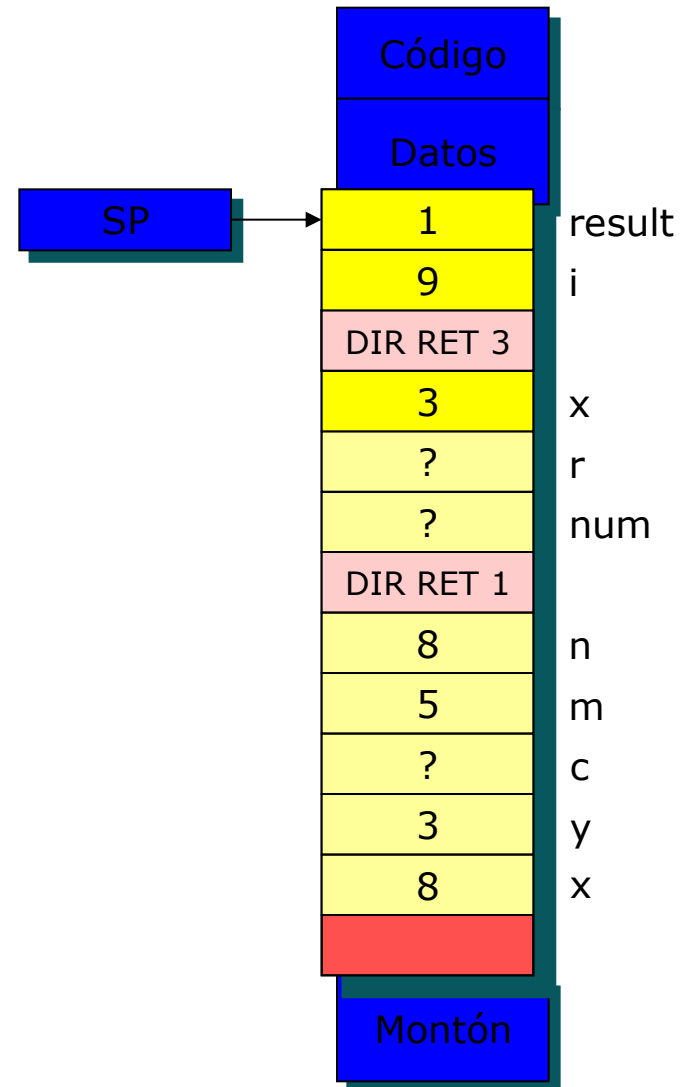
```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 3:

DIR RET 1:



La zona de memoria de datos

- El funcionamiento de la pila

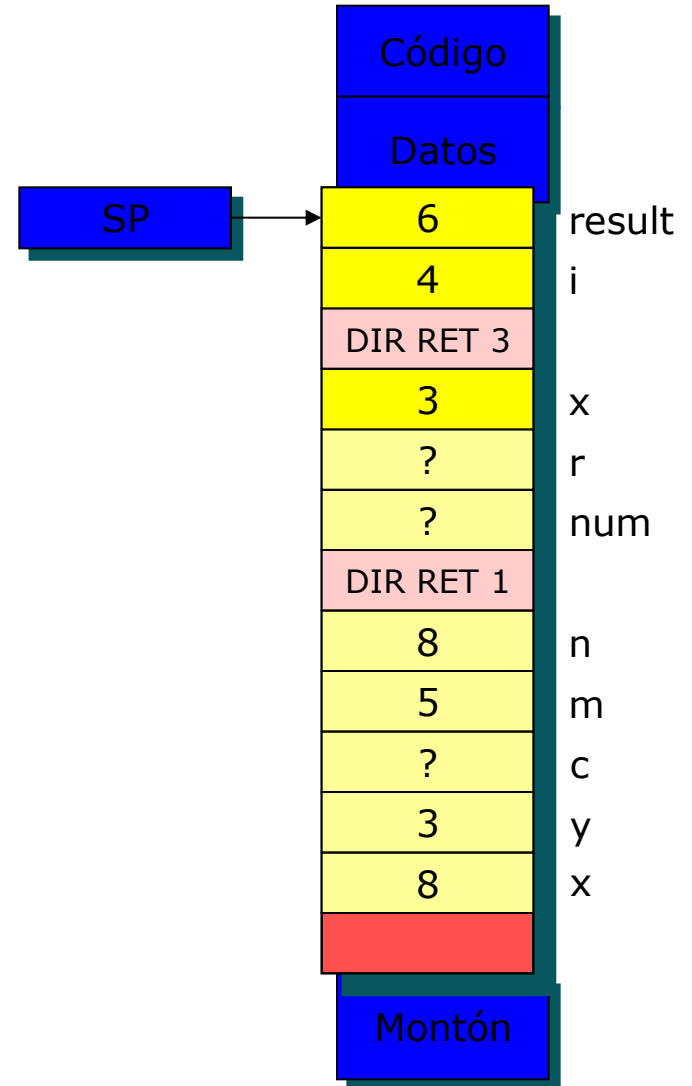
```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 3:

DIR RET 1:



La zona de memoria de datos

- El funcionamiento de la pila

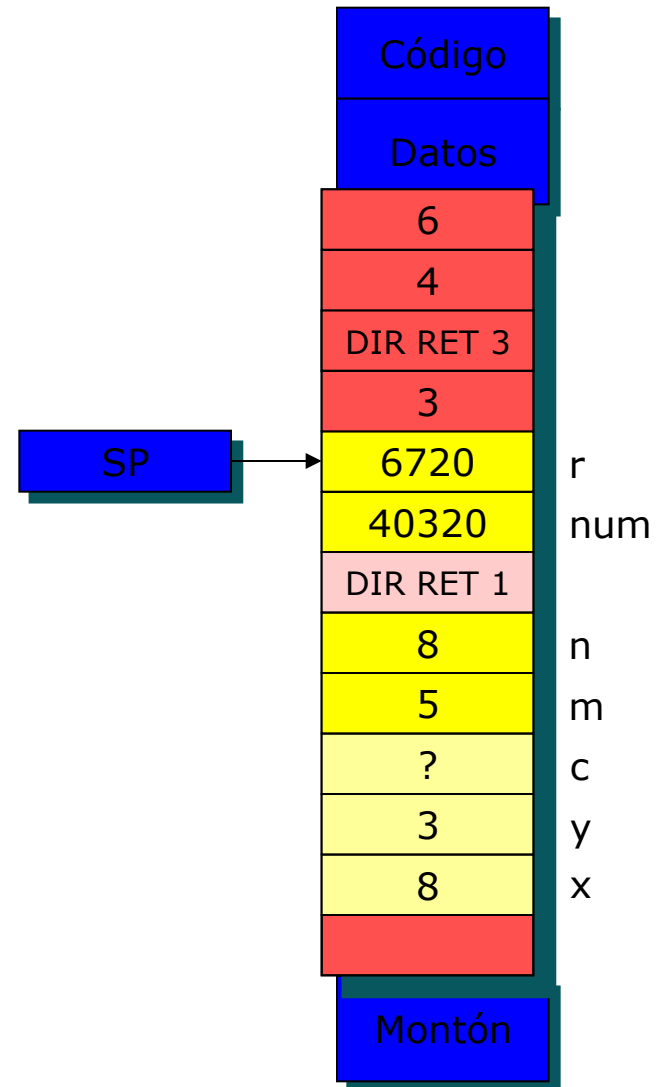
```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET 5

DIR RET 1:



La zona de memoria de datos

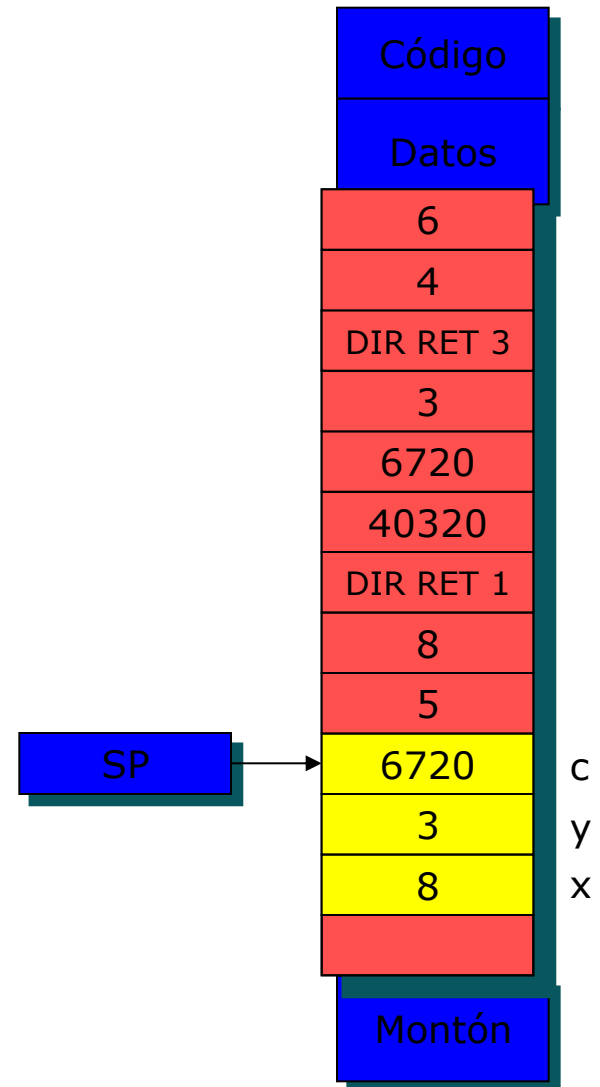
- El funcionamiento de la pila

```
int Factorial(int x)
{
    int i,result=1;
    for (i=1;i<=x;i++)
        result=result*i;
    return result;
}

int CombNM(int n,int m)
{
    int num,r;
    num=Factorial(n);
    r=num/Factorial(n-m);
    return r;
}

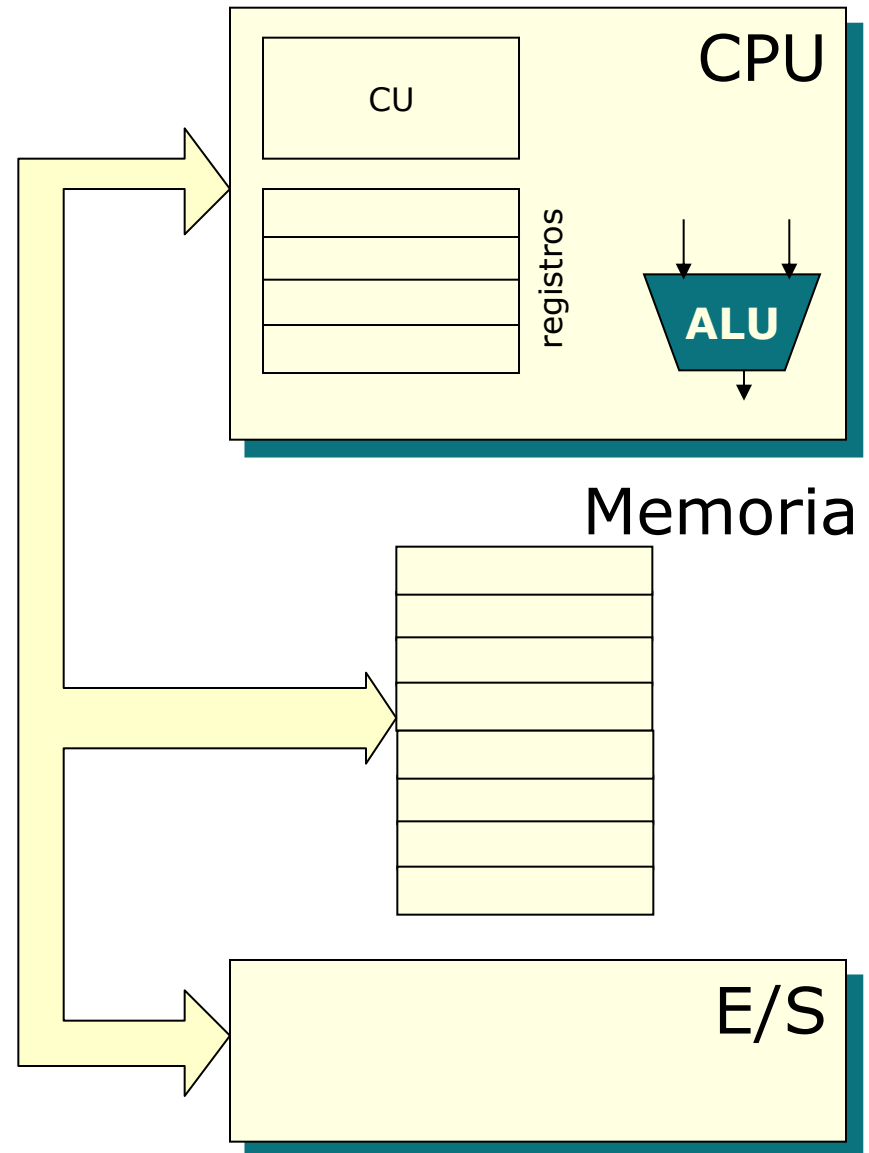
void main()
{
    int x=8,y=3,c;
    c=CombNM(x,y+2);
    printf("Comb=%d\n",c);
}
```

DIR RET →



Los dispositivos de Entrada/Salida

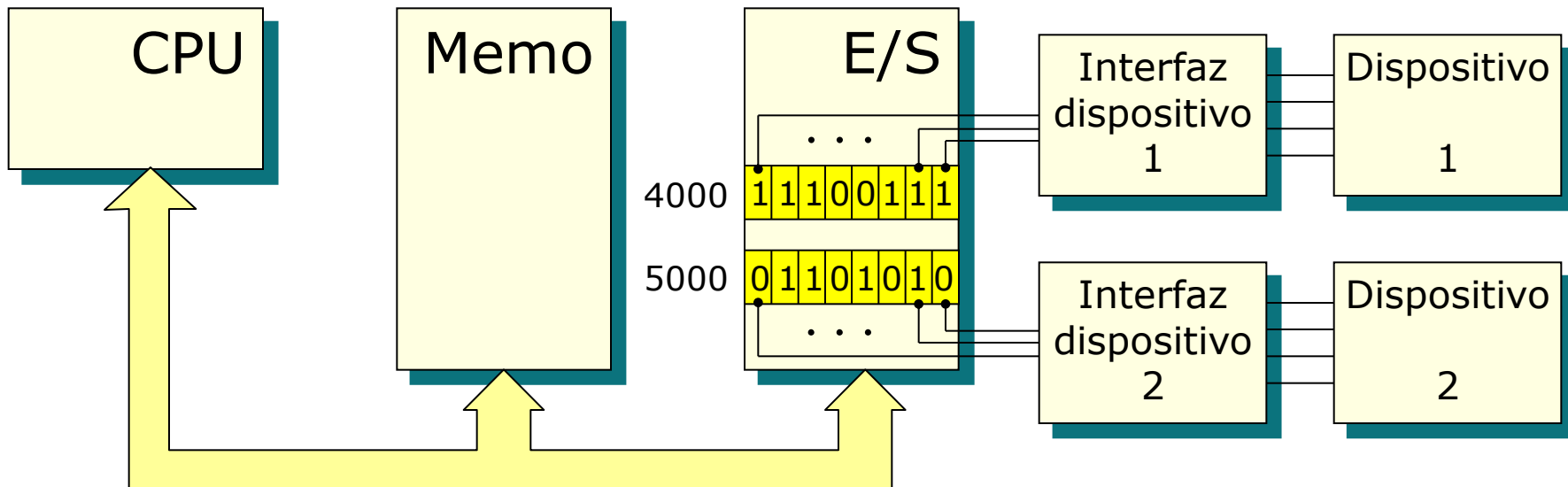
- Dispositivos de E/S: permiten la comunicación del computador con el exterior
 - Entrada: teclado, ratón, conversor A/D, entrada digital, temporizador, ...
 - Salida: pantalla, impresora, conversor D/A, salida digital, ...
 - E y S: disco duro, comunicación serie, paralelo, red, ...



Programación de E/S: puertos

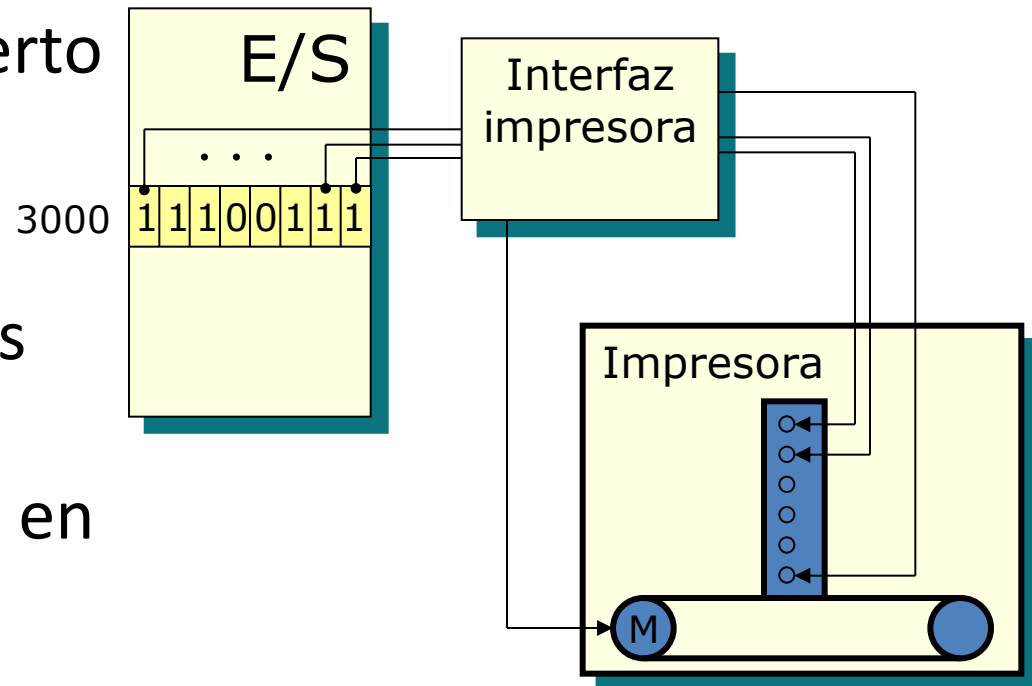
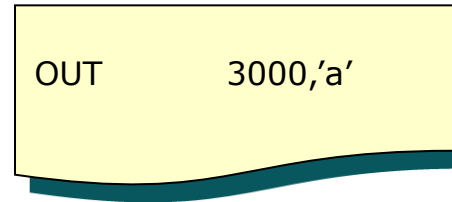
- Puertos de E/S:

- Lugares de almacenamiento de información que, cuando son leídos o escritos, provocan acciones en los dispositivos de E/S
- Organizados en direcciones



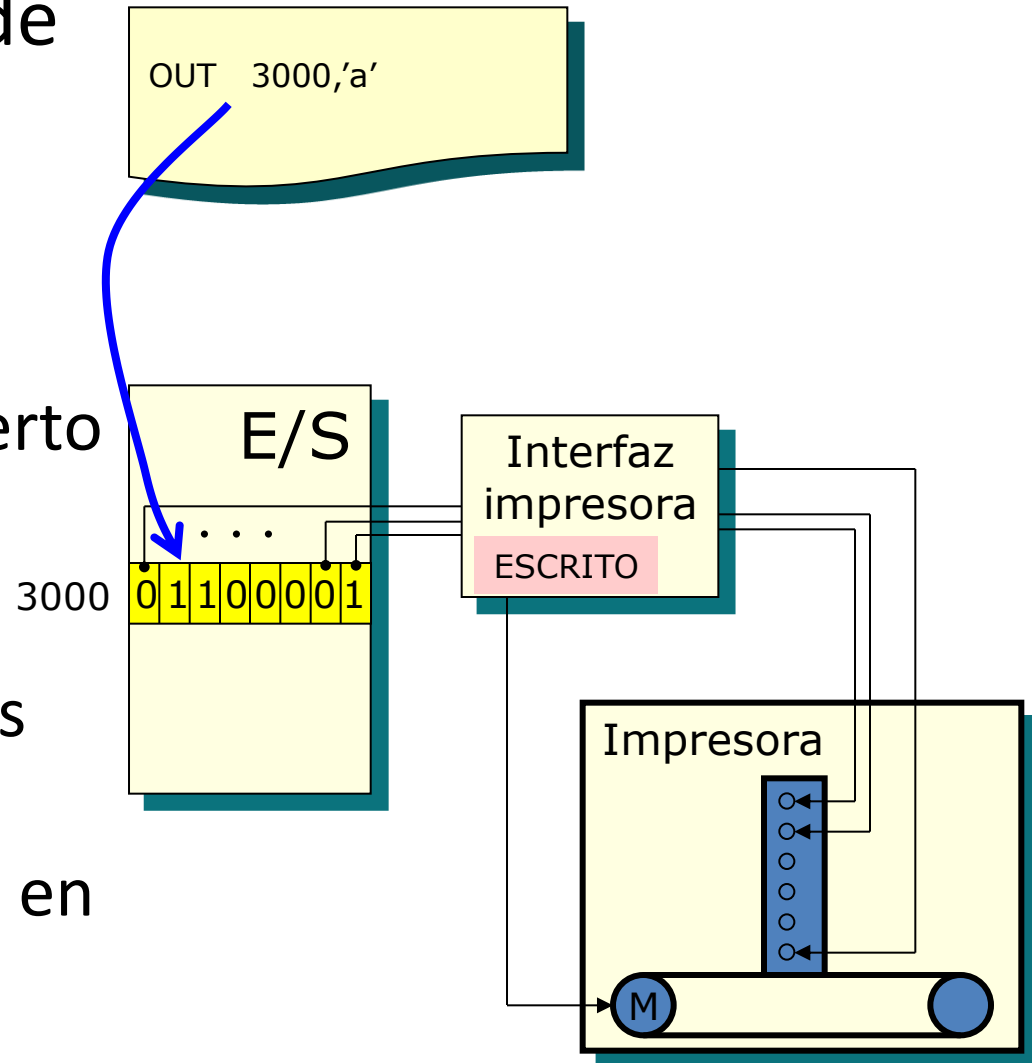
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo



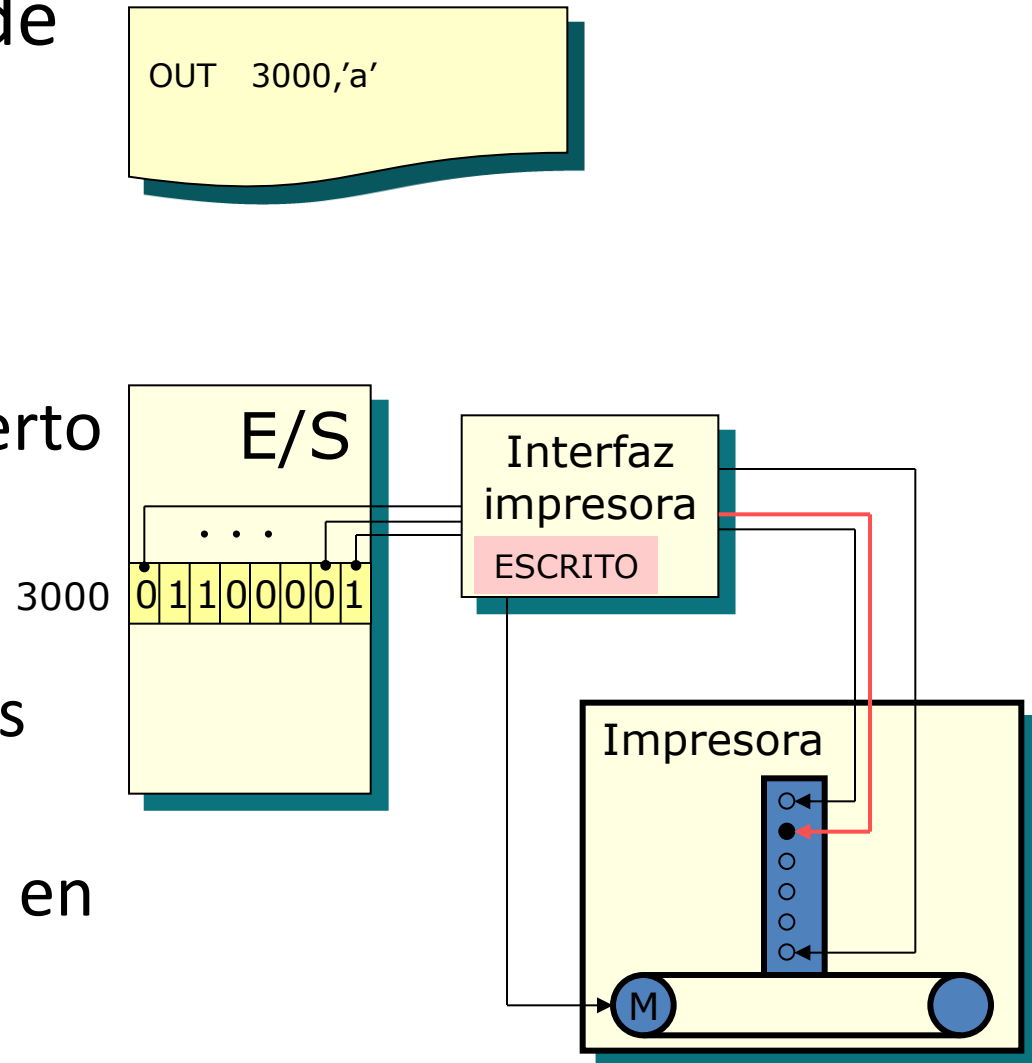
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo



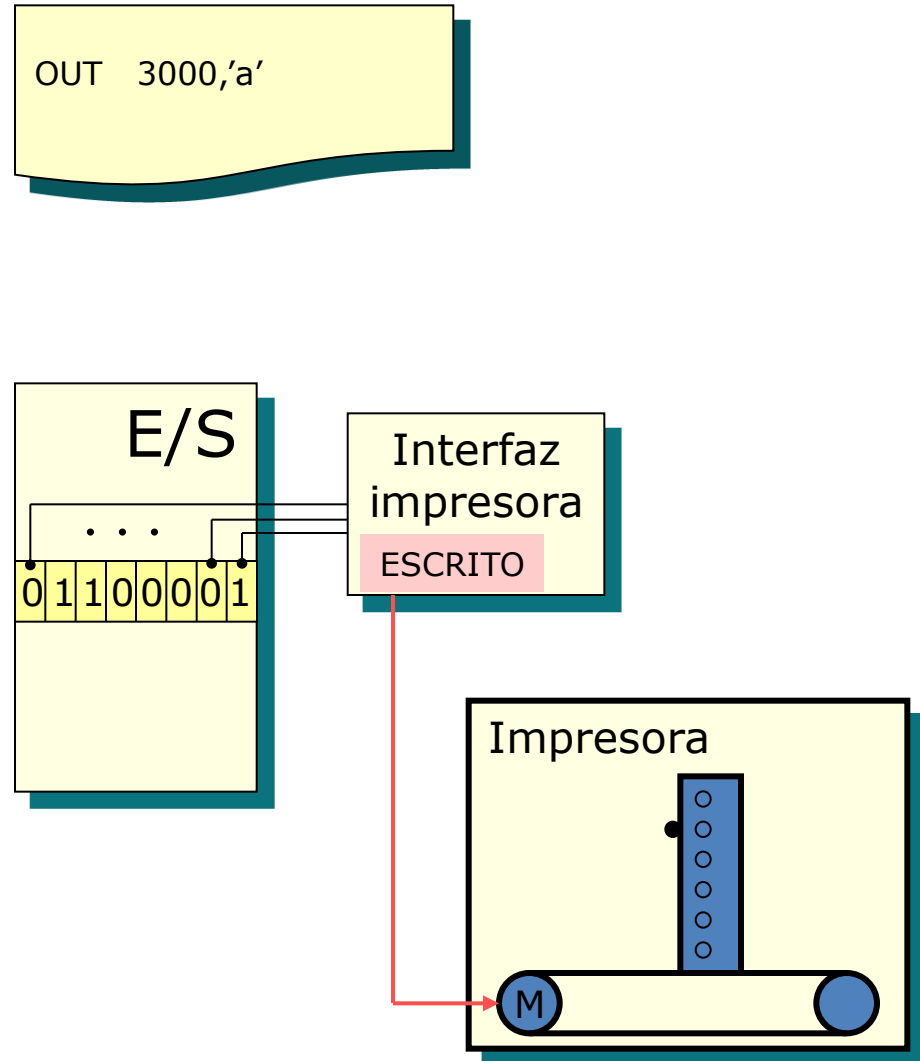
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo



Funcionamiento de dispositivos

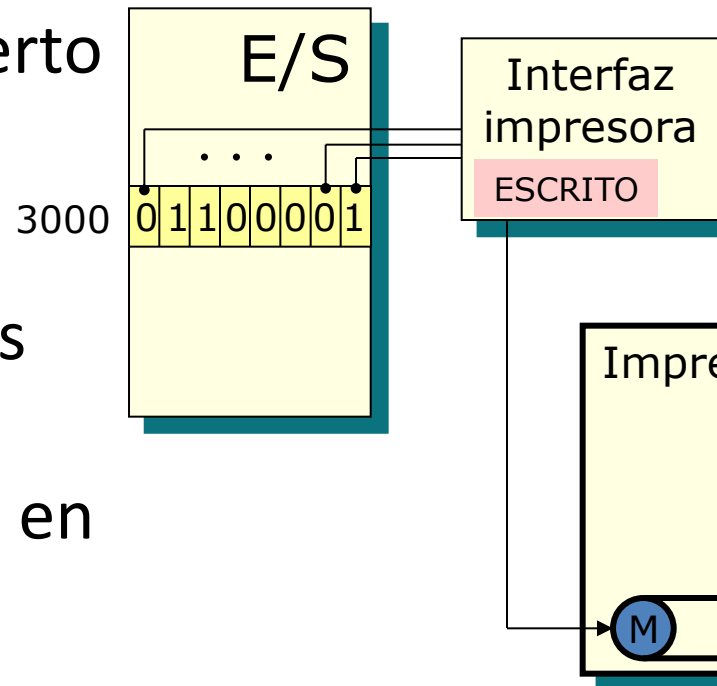
- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo



Funcionamiento de dispositivos

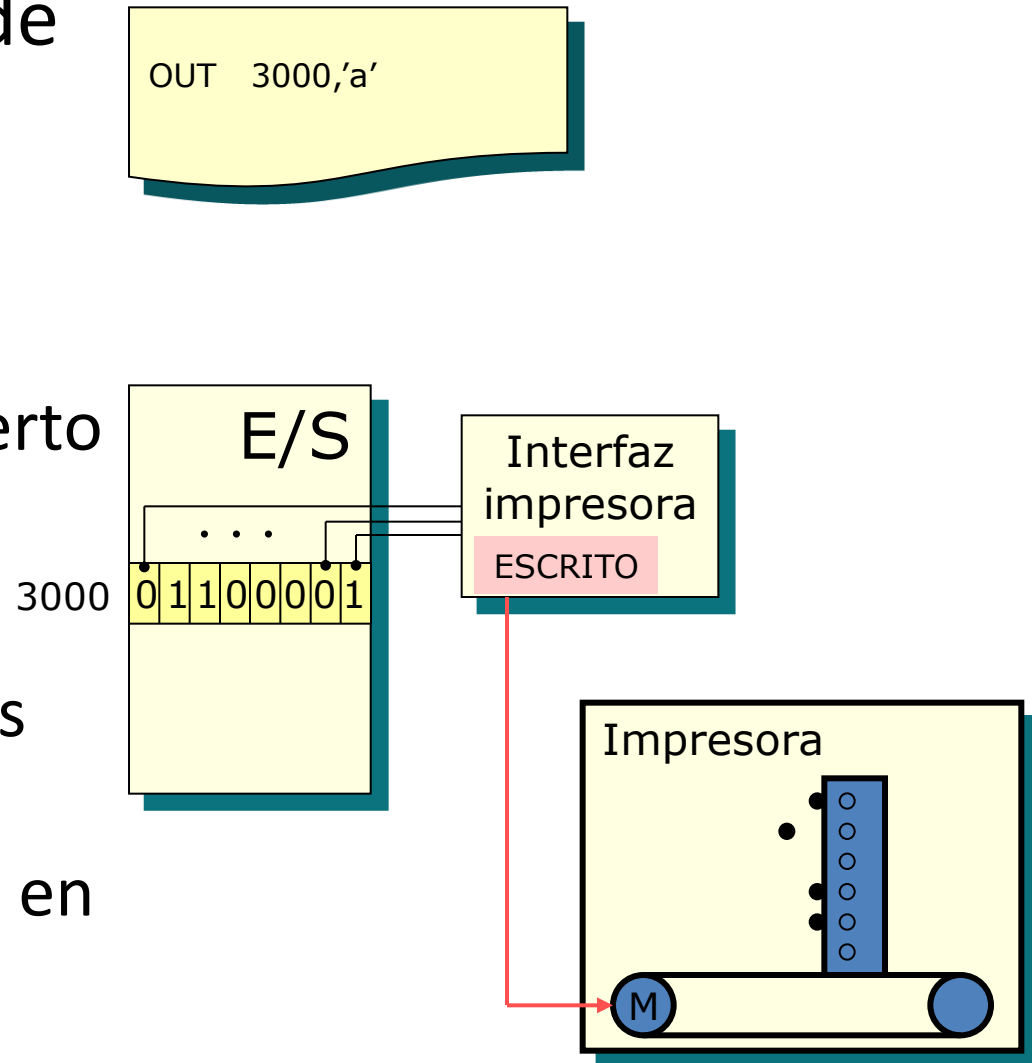
- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo

OUT 3000,'a'



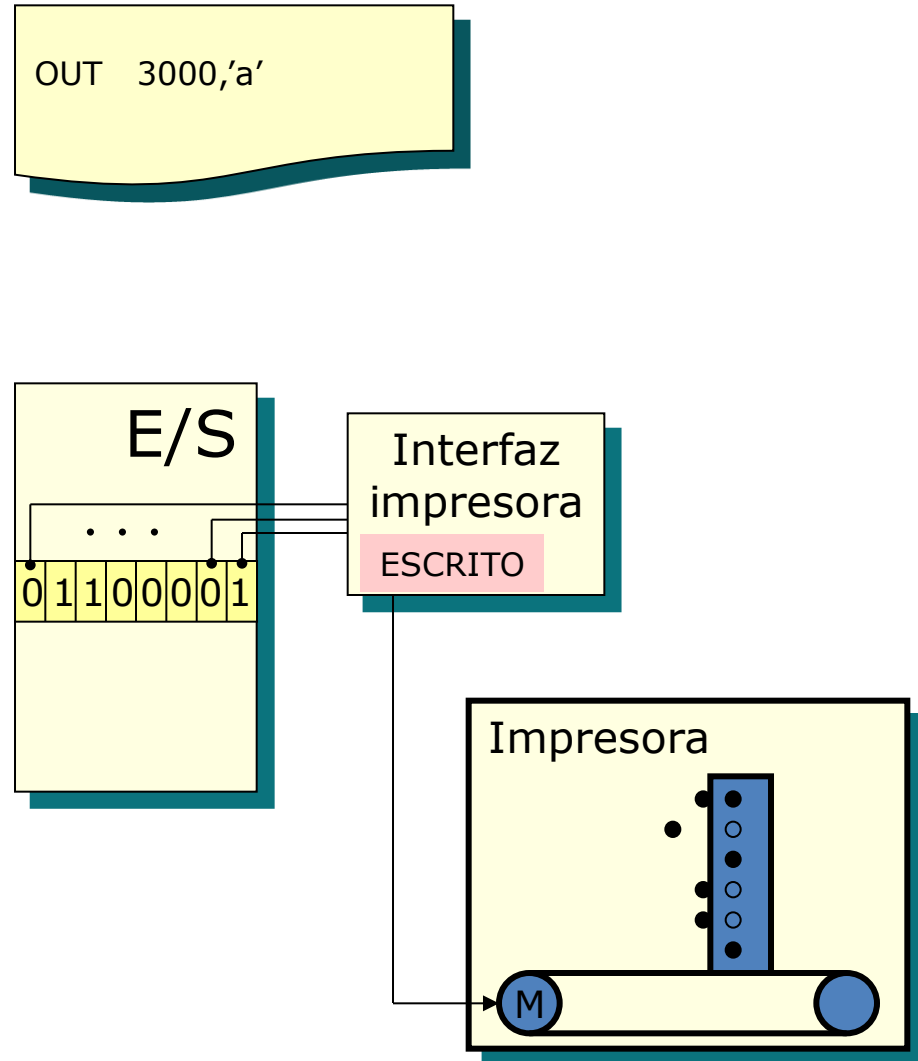
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo



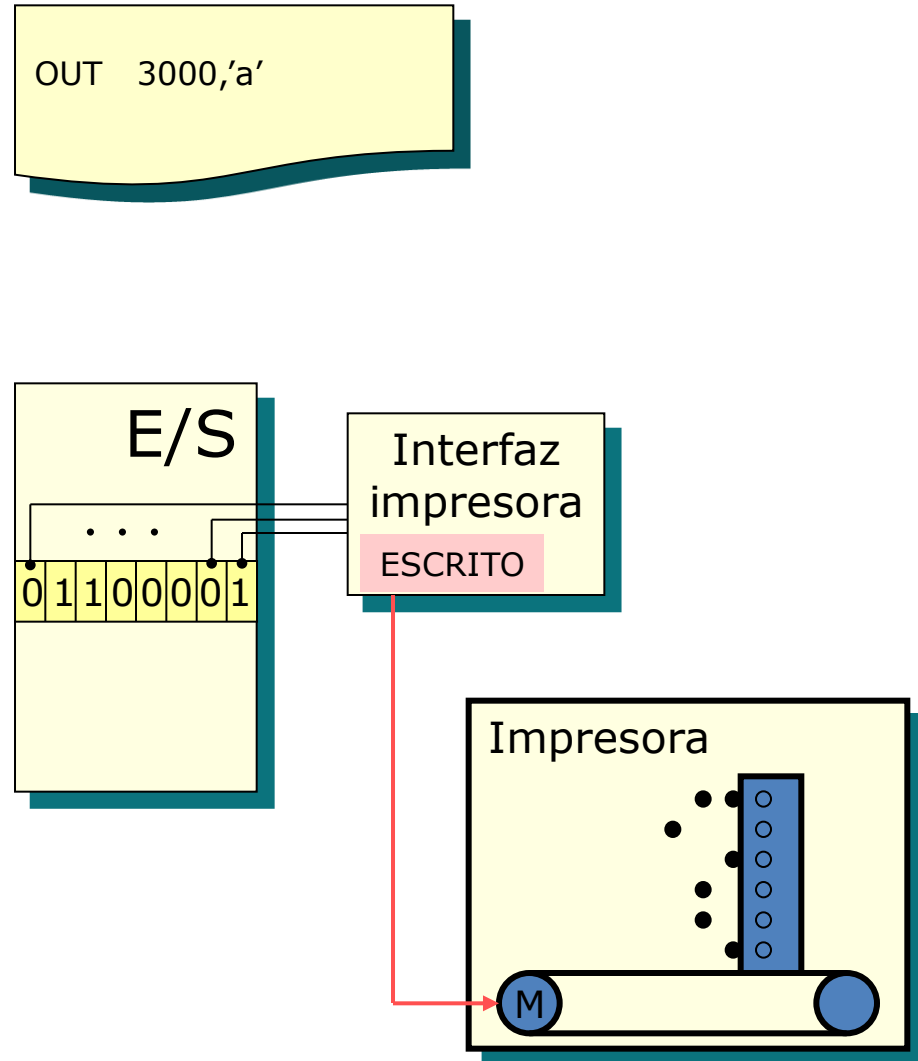
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo



Funcionamiento de dispositivos

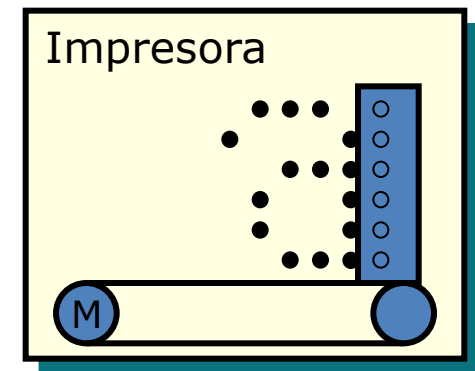
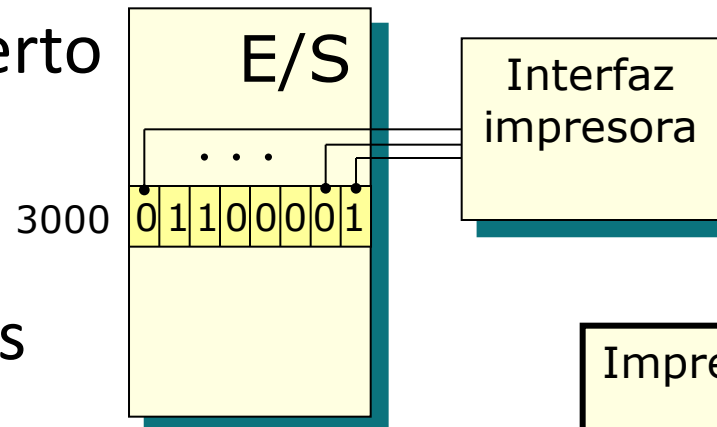
- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo



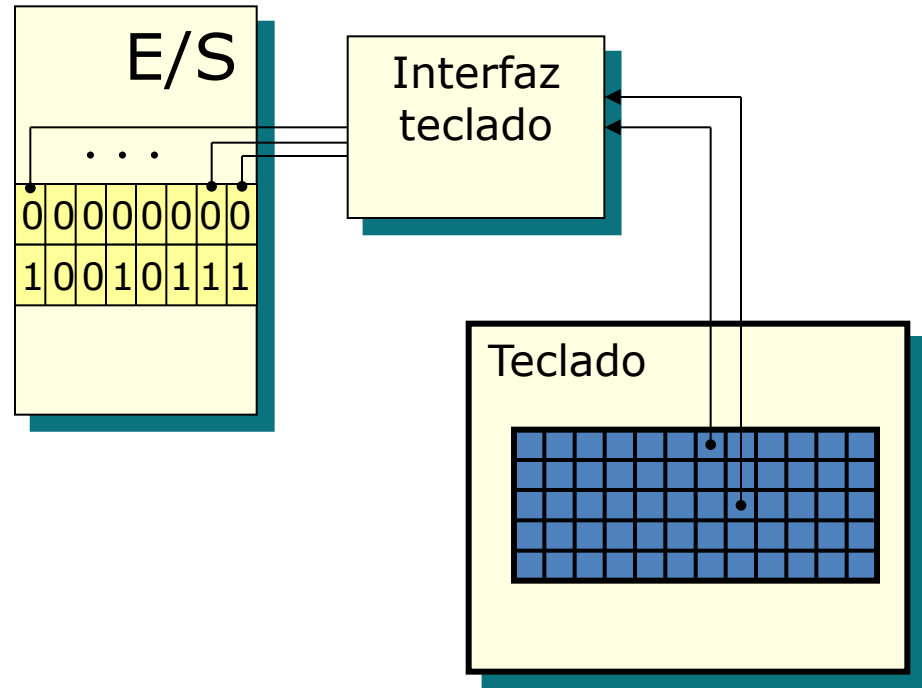
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Salida: impresora
 - El programador escribe en un puerto del dispositivo
 - La electrónica del interfaz genera las acciones correspondientes en el dispositivo

OUT 3000,'a'



- La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos
- El programador lee del/los puertos del dispositivo



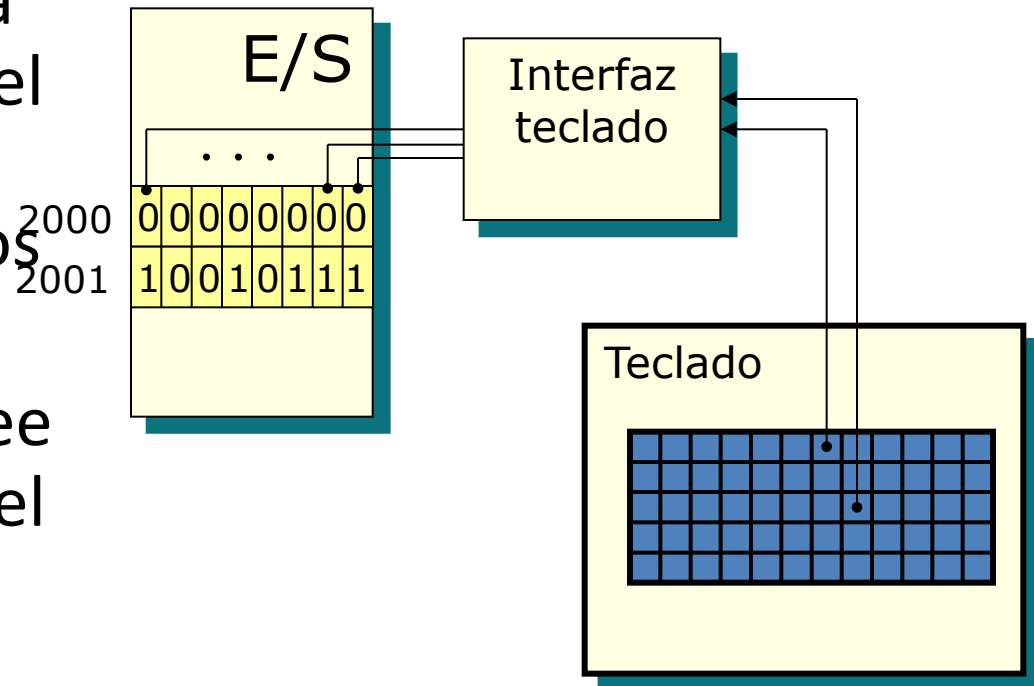
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Entrada: teclado

- La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos

- El programador lee del/los puertos del dispositivo

```
ESPERA:
  TEST 2000,000000001b
  JZ    ESPERA
  IN    R0,2001
```



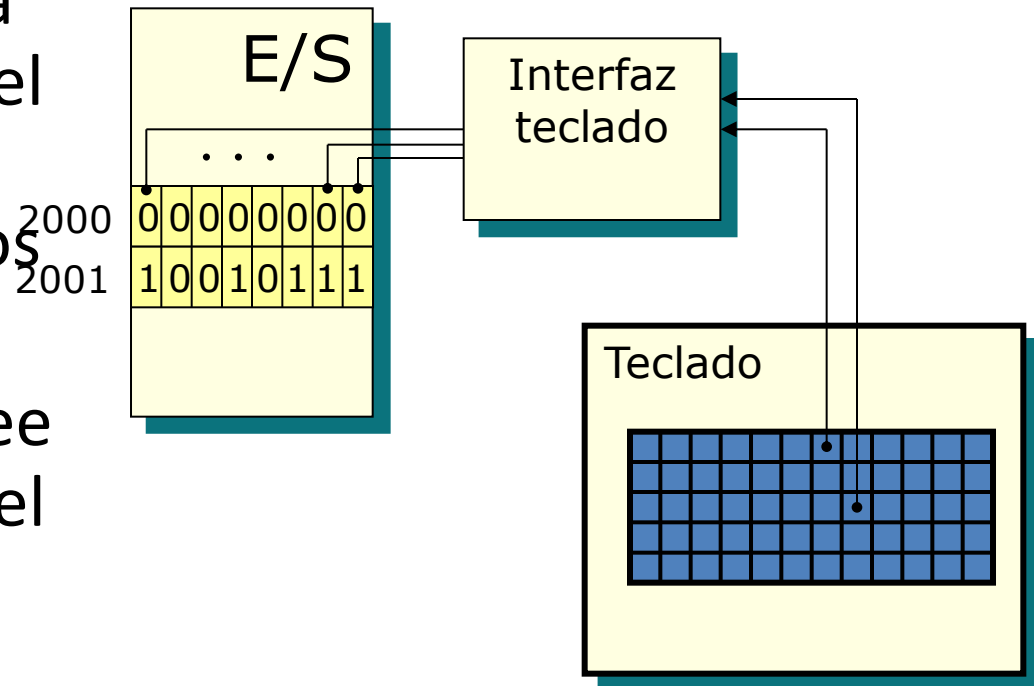
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Entrada: teclado

- La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos

- El programador lee del/los puertos del dispositivo

```
ESPERA:  
TEST 2000,000000001b  
JZ   ESPERA  
IN   R0,2001
```



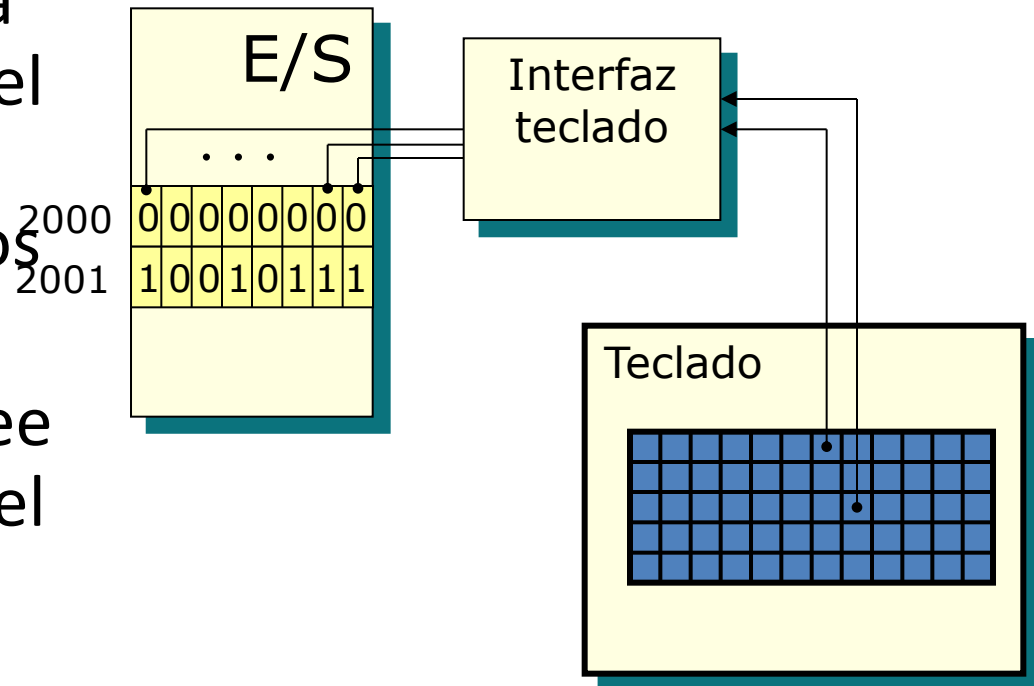
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Entrada: teclado

- La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos

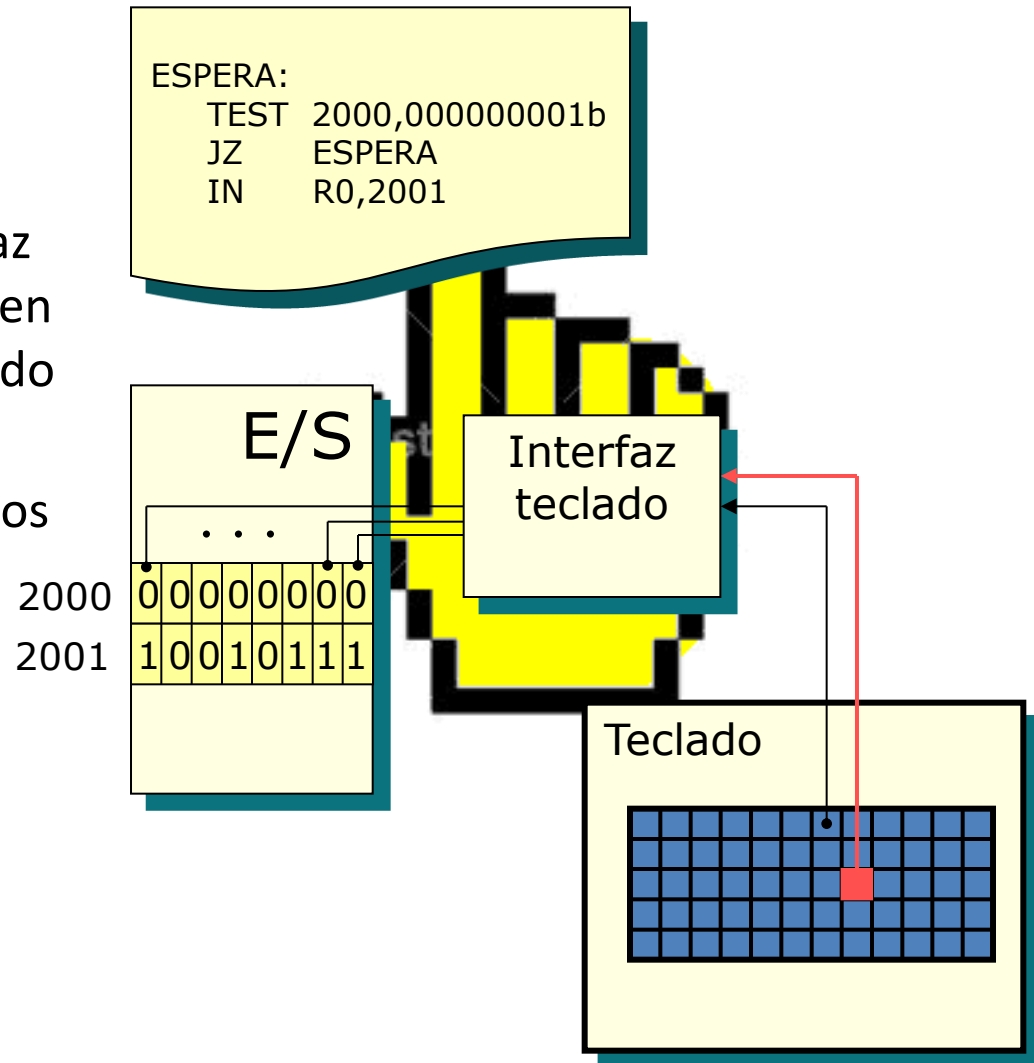
- El programador lee del/los puertos del dispositivo

```
ESPERA:
  TEST 2000,000000001b
  JZ   ESPERA
  IN    R0,2001
```



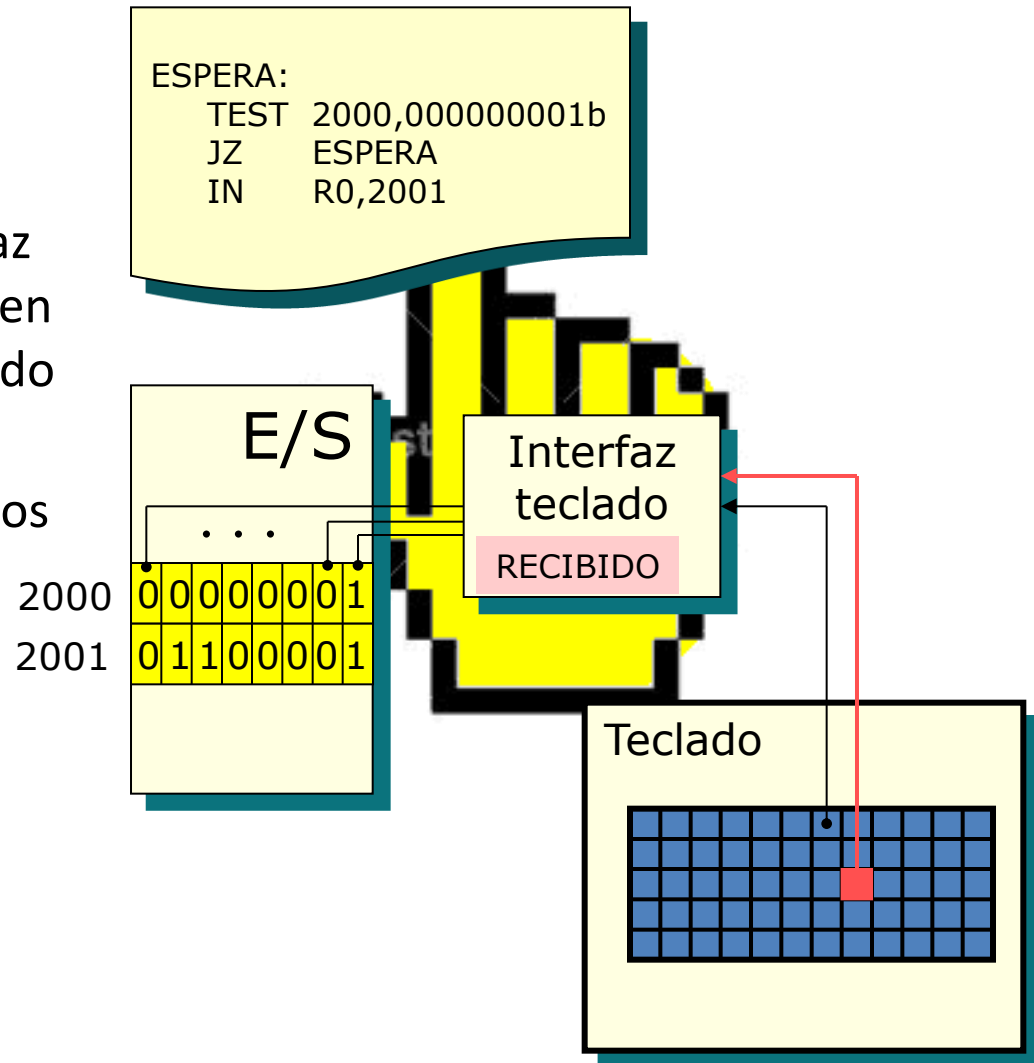
Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Entrada: teclado
 - La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos
 - El programador lee del/los puertos del dispositivo



Funcionamiento de dispositivos

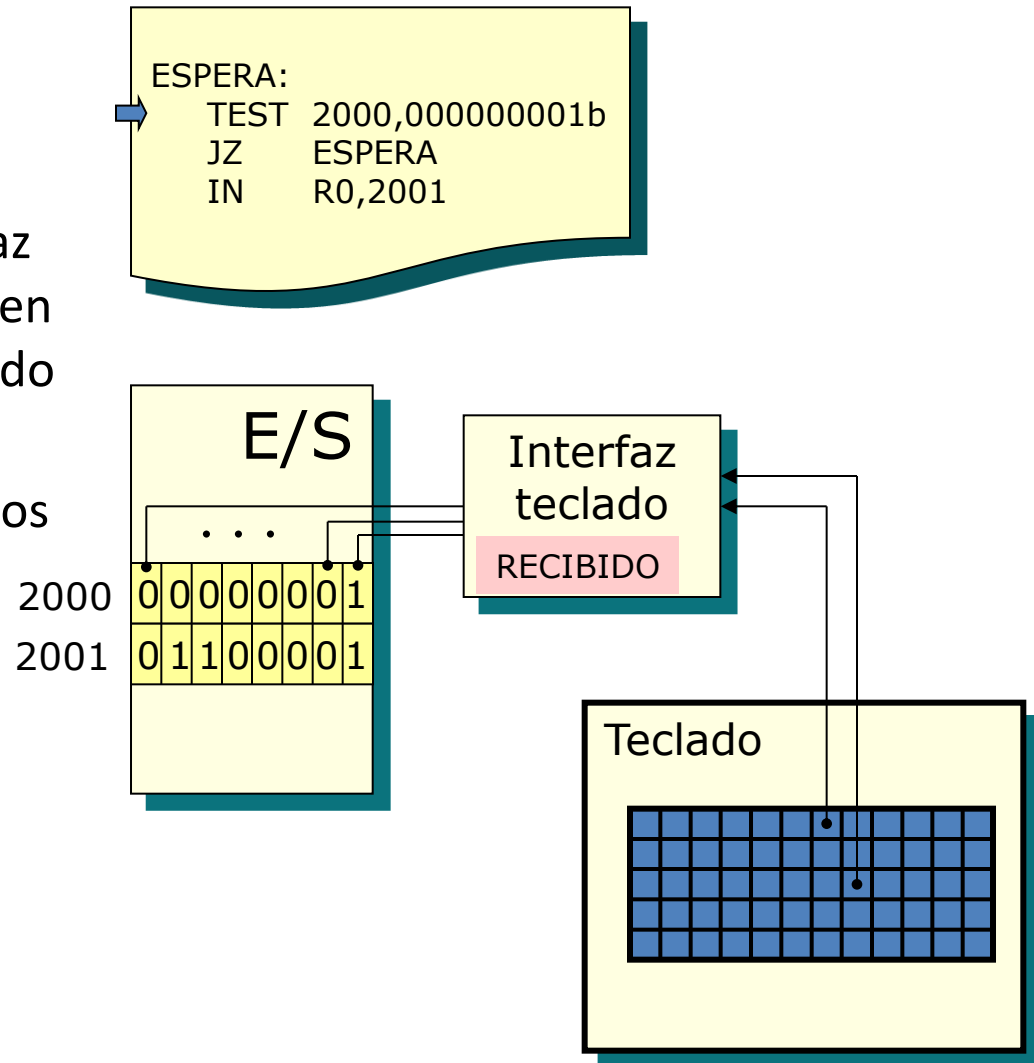
- Funcionamiento de un dispositivo de Entrada: teclado
 - La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos
 - El programador lee del/los puertos del dispositivo



Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Entrada: teclado

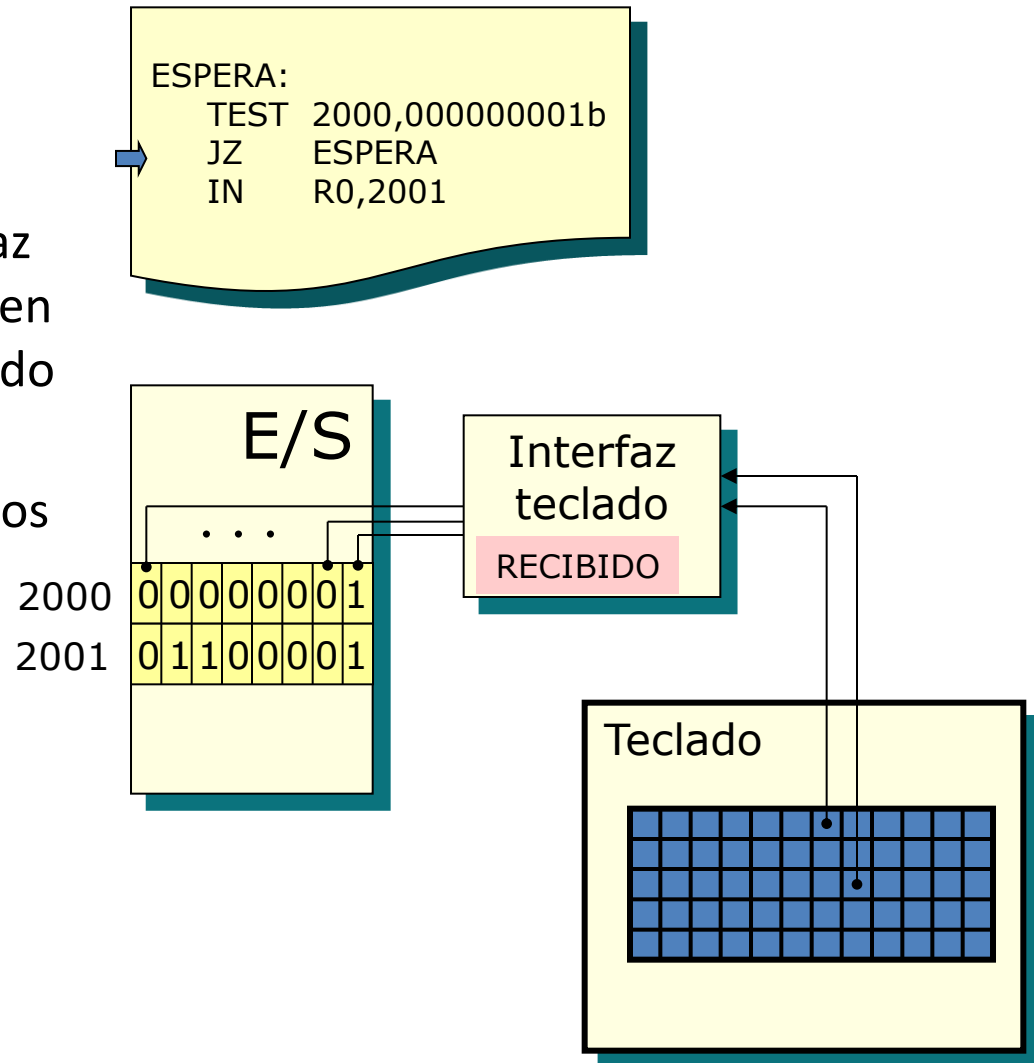
- La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos
- El programador lee del/los puertos del dispositivo



Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Entrada: teclado

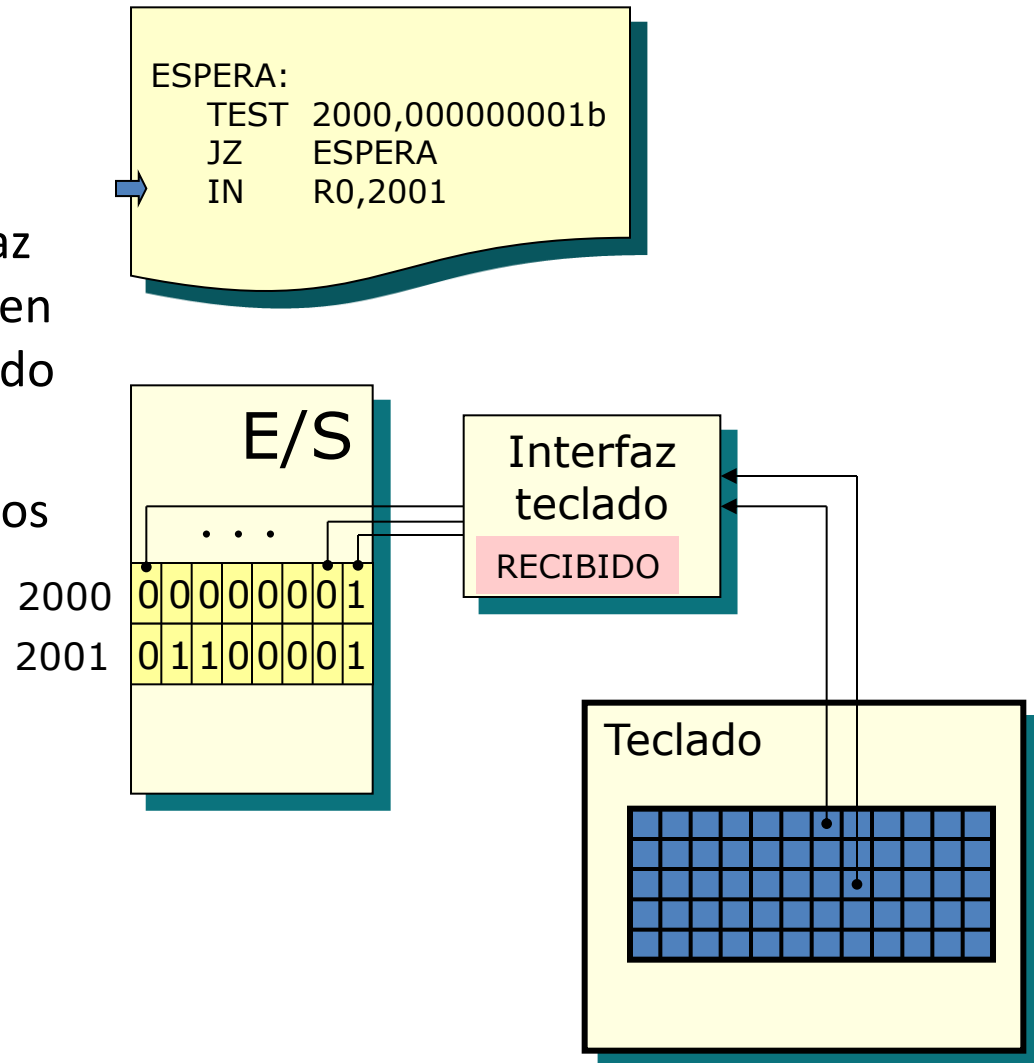
- La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos
- El programador lee del/los puertos del dispositivo



Funcionamiento de dispositivos

- Funcionamiento de un dispositivo de Entrada: teclado

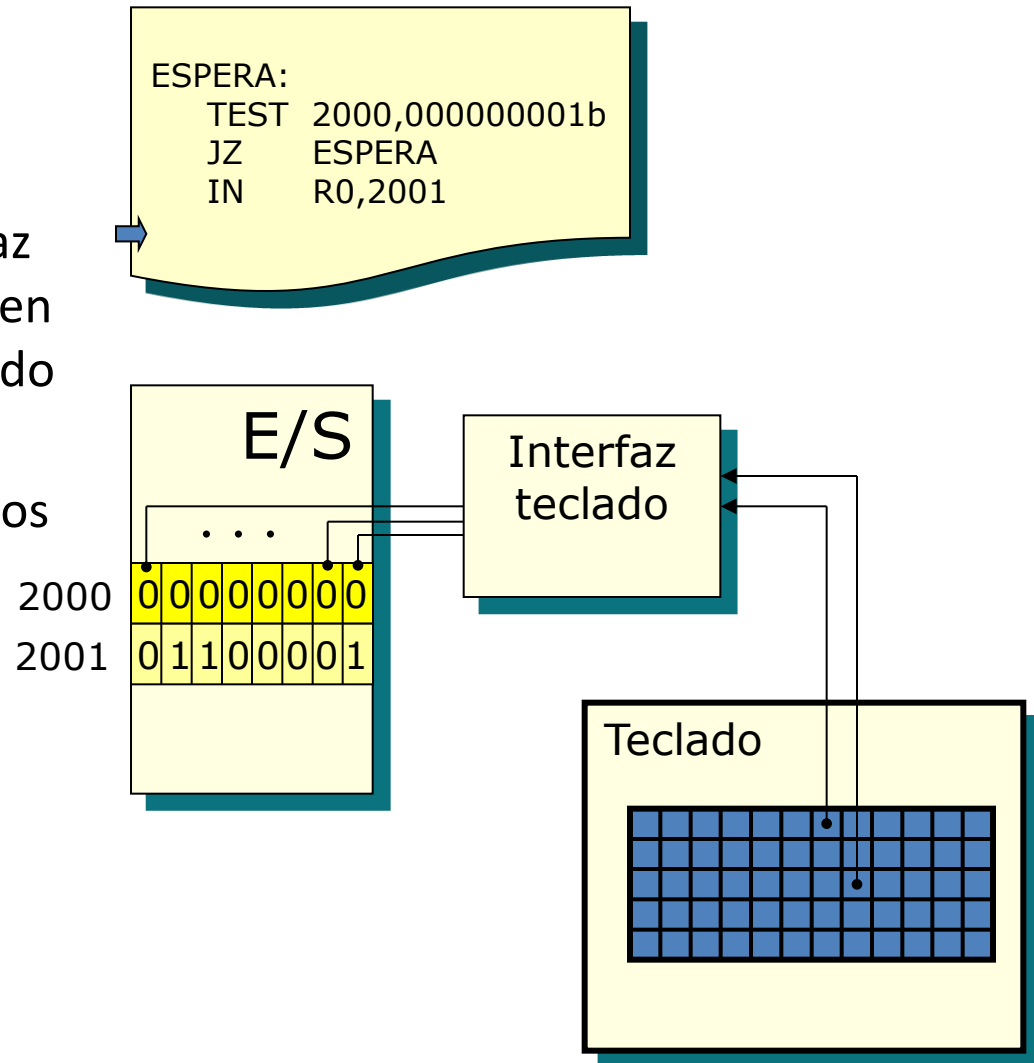
- La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos
- El programador lee del/los puertos del dispositivo



Funcionamiento de dispositivos

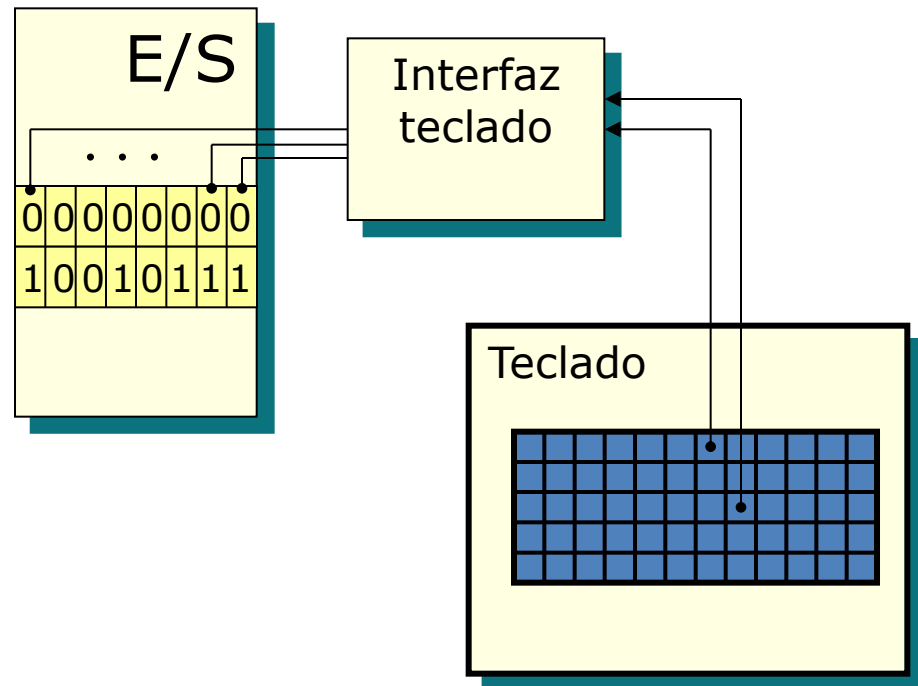
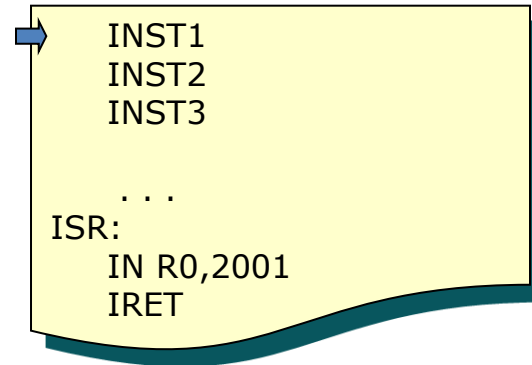
- Funcionamiento de un dispositivo de Entrada: teclado

- La electrónica del interfaz reacciona ante cambios en el dispositivo, modificando el/los puertos
- El programador lee del/los puertos del dispositivo



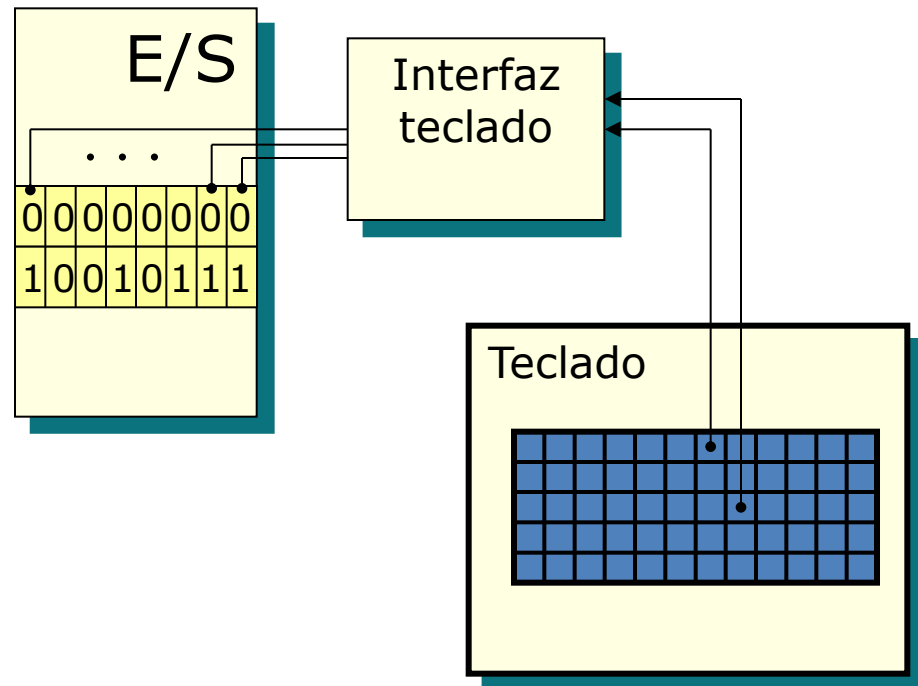
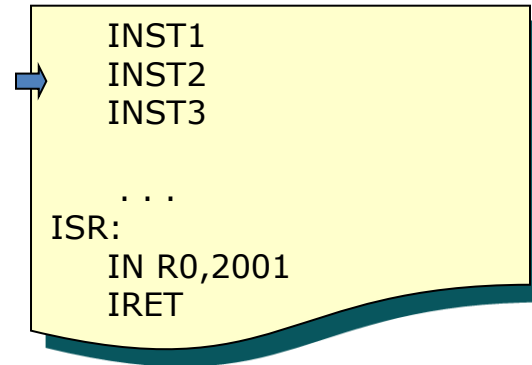
Sincronización de E/S

- Muestreo o polling:
 - El programa comprueba el estado del dispositivo leyendo los puertos
 - Ocupa mucho tiempo de CPU
- Interrupciones:
 - La CPU está ejecutando otro programa
 - Cuando el dispositivo avisa de algún evento, se abandona temporalmente la ejecución del programa y se pasa a ejecutar una ISR. Se retoma el programa al terminar la ISR.



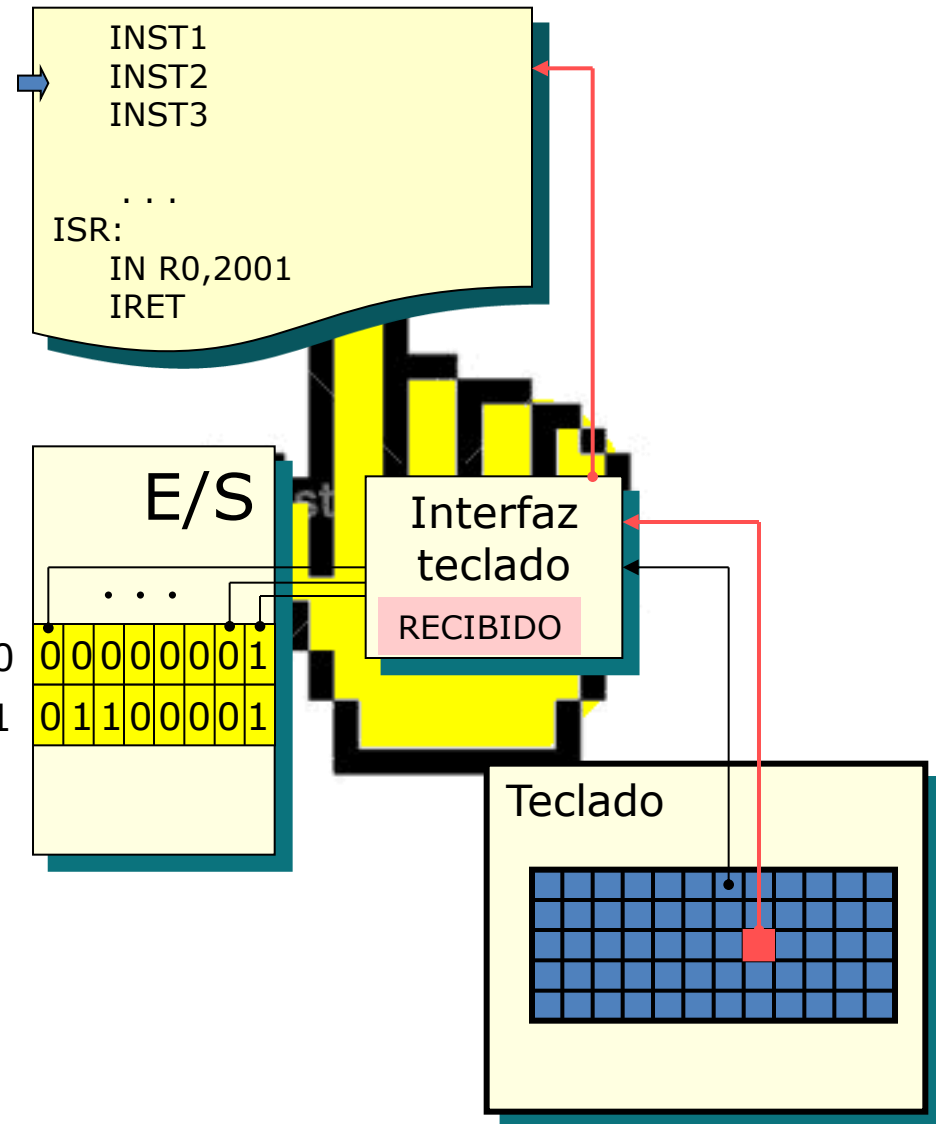
Sincronización de E/S

- Muestreo o polling:
 - El programa comprueba el estado del dispositivo leyendo los puertos
 - Ocupa mucho tiempo de CPU
- Interrupciones:
 - La CPU está ejecutando otro programa
 - Cuando el dispositivo avisa de algún evento, se abandona temporalmente la ejecución del programa y se pasa a ejecutar una ISR. Se retoma el programa al terminar la ISR.



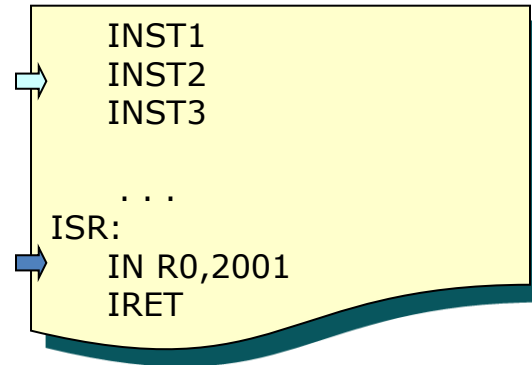
Sincronización de E/S

- Muestreo o polling:
 - El programa comprueba el estado del dispositivo leyendo los puertos
 - Ocupa mucho tiempo de CPU
- Interrupciones:
 - La CPU está ejecutando otro programa
 - Cuando el dispositivo avisa de algún evento, se abandona temporalmente la ejecución del programa y se pasa a ejecutar una ISR. Se retoma el programa al terminar la ISR.

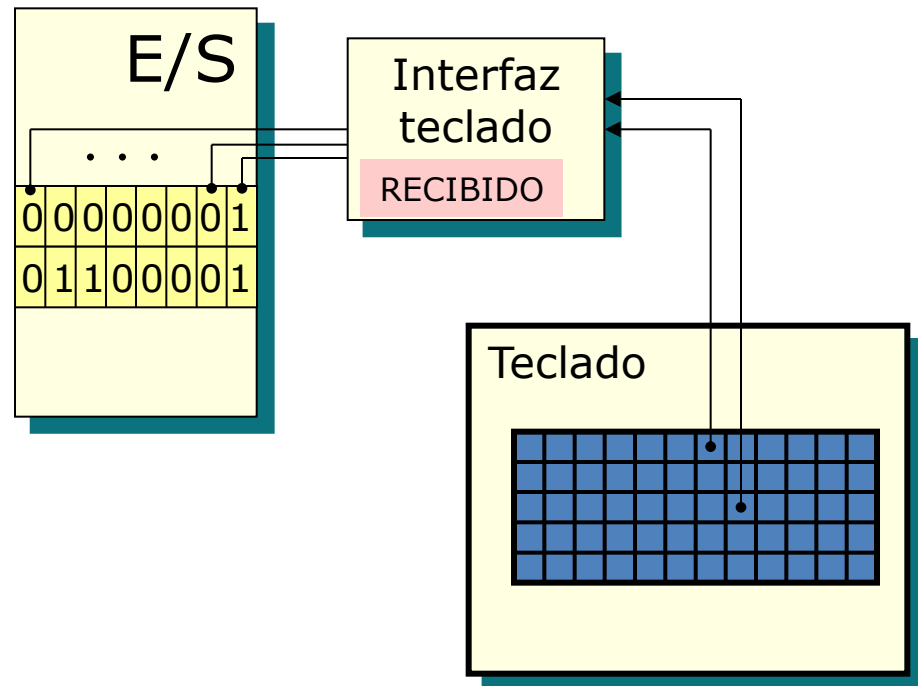


Sincronización de E/S

- Muestreo o polling:
 - El programa comprueba el estado del dispositivo leyendo los puertos
 - Ocupa mucho tiempo de CPU

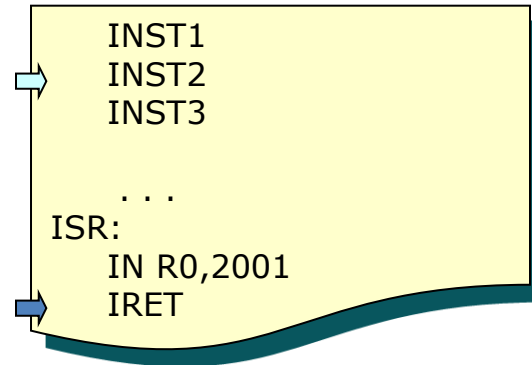


- Interrupciones:
 - La CPU está ejecutando otro programa
 - Cuando el dispositivo avisa de algún evento, se abandona temporalmente la ejecución del programa y se pasa a ejecutar una ISR. Se retoma el programa al terminar la ISR.

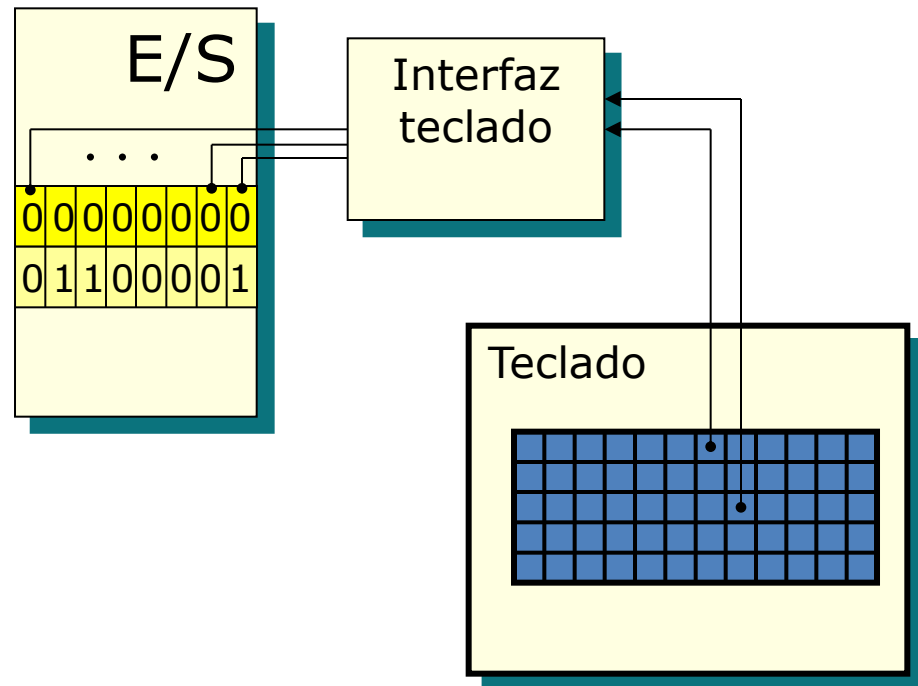


Sincronización de E/S

- Muestreo o polling:
 - El programa comprueba el estado del dispositivo leyendo los puertos
 - Ocupa mucho tiempo de CPU



- Interrupciones:
 - La CPU está ejecutando otro programa
 - Cuando el dispositivo avisa de algún evento, se abandona temporalmente la ejecución del programa y se pasa a ejecutar una ISR. Se retoma el programa al terminar la ISR.



Sincronización de E/S

- Muestreo o polling:
 - El programa comprueba el estado del dispositivo leyendo los puertos
 - Ocupa mucho tiempo de CPU
- Interrupciones:
 - La CPU está ejecutando otro programa
 - Cuando el dispositivo avisa de algún evento, se abandona temporalmente la ejecución del programa y se pasa a ejecutar una ISR. Se retoma el programa al terminar la ISR.

