

Tema-3-parte-1-Estructura-de-com...



atmuntenas



Fundamentos de Informática



1º Grado en Ingeniería de las Tecnologías de Telecomunicación



**Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada**



[Accede al documento original](#)

Tema 3 - Conceptos básicos y elementos constitutivos de un procesador

INTRODUCCIÓN

Un ordenador es un sistema de gran complejidad. Para analizarlo consideramos distintos niveles de estudio, organizados por su complejidad, permitiendo su comprensión de cada uno de ellos sin recurrir a niveles de mayor complejidad al estudiado.

NIVEL DE DISPOSITIVOS Y CIRCUITOS ELECTRÓNICOS

Con componentes como transistores, condensadores y resistencias. La información se representa con tensiones eléctricas, corrientes eléctricas, estados de magnetización, etc.

NIVEL DE LÓGICA DIGITAL

La mayoría de las operaciones digitales se describen en el nivel de lógica digital, y sus elementos básicos, las puertas lógicas, pueden almacenar, manipular y transmitir datos representables en códigos binarios (0,1), haciendo abstracción de la magnitud física que los sustenta.

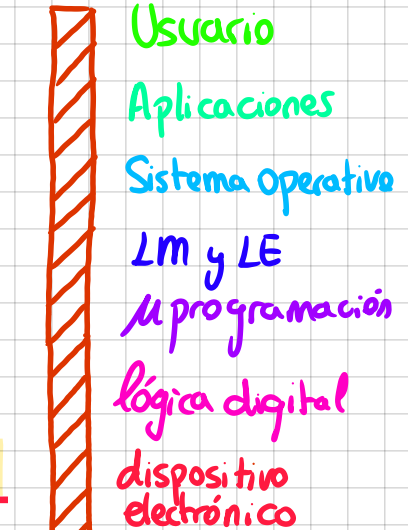
μprogramación

Los módulos que utiliza el nivel de μoperaciones son registros, multiplexores, bancos de registros, módulos de memoria RAM, etc. construidos con puertas lógicas. Las señales de control son para realizar operaciones tales como carga de un registro, efectuar una operación concreta en la ALU, escribir en la memoria, etc.

LENGUAJES MÁQUINA Y ENSAMBLADOR (LM Y LE)

Es el genuino a la arquitectura del computador ya que el diseño de un computador parte de la especificación del lenguaje máquina.

Poco Complejo



Muy complejo

El lenguaje ensamblador es igual al lenguaje máquina, salvo que utiliza una terminología distinta para describir los módulos o instrucciones que se definen en este nivel.

Cada instrucción máquina se descompone en una secuencia ordenada de señales de control.

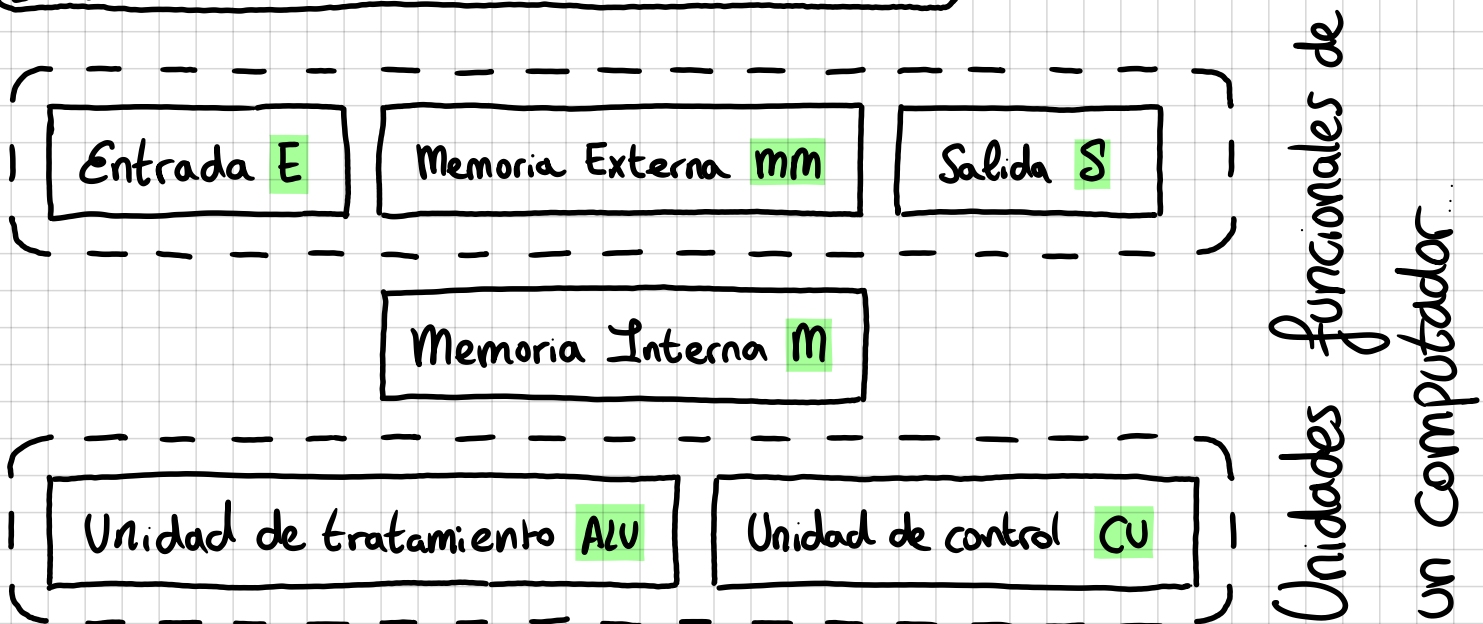
SISTEMA OPERATIVO • máquina operativa

Se puede considerar como la interfaz entre el Hardware y el Software. Añade una capa para facilitar el uso del Hardware desde el punto de vista de los usuarios y de los programas de aplicación.

PROGRAMAS DE APLICACIÓN

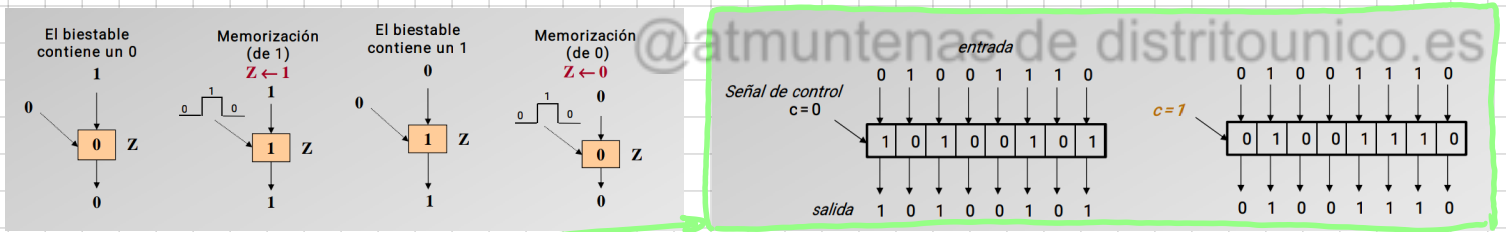
Es el que realmente ven los programadores de aplicaciones y los usuarios. Si bien está concebido, el usuario no necesitará saber nada de los niveles inferiores, y sus instrucciones son simbólicas, debiendo ser lo más próximo a las personas.

ESQUEMA DE FUNCIONAMIENTO DE UN COMPUTADOR



Las distintas unidades de un ordenador están formadas por una serie de elementos básicos:

→ **BIESTABLES** es un elemento de memoria que almacena un bit de información. Dar a un biestable Z un valor (ej: 1), se escribe así: $Z \leftarrow 1$



→ **REGISTRO** es un circuito hecho para almacenar temporalmente un dato, una instrucción o una dirección de un número determinado de n bits. Dispone de n entradas, por donde, a través de un bus de ancho n , llega el dato o instrucción de n salidas donde constantemente aparece la información que está grabada en el registro, y una entrada de 1 bit para una señal de control c .

Si $c=0$, el registro no cambia la información almacenada, aunque hayan datos en la entrada. Pero en el momento que $c=1$, la información que hoy en las entradas quedan almacenadas.

Dar a un registro (ej: $R7$) un valor (ej: Rm) se escribe así: $R7 \leftarrow Rm$.

Los registros suelen ser de 8, 16, 32, 64 ó 128 bits.

→ **CONTADORES** son circuitos cuyas salidas presentan una secuencia repetitiva de valores, al actuar una señal de control.

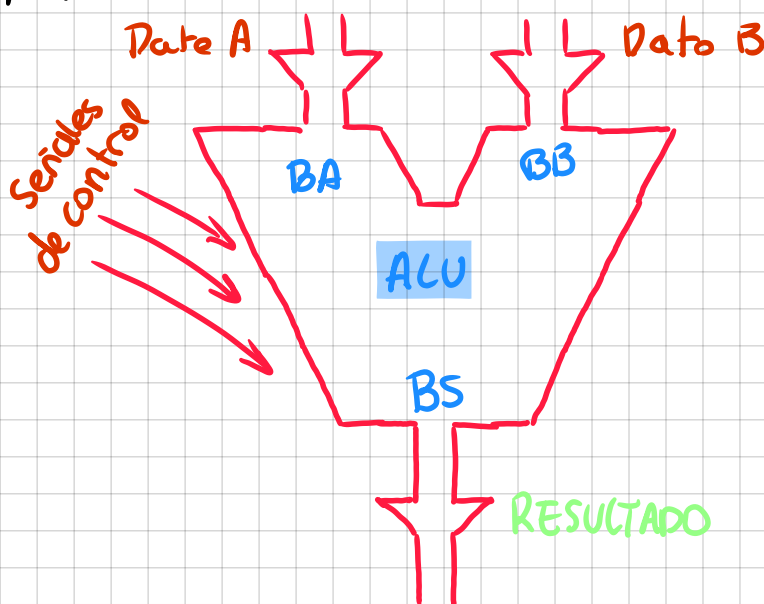
Consideraremos los contadores binarios ascendentes y con la posibilidad de carga en paralelo de 8 bits.

Definimos el contador PC , cuyo incremento se describe como $PC \leftarrow PC + 1$. La carga en paralelo de PC con el contenido (salida), por ejemplo, del registro DR , se denota como: $PC \leftarrow DR$.

→ **UNIDAD DE TRATAMIENTO (AU)** está formada por la unidad aritmético-lógica (ALU) y otros elementos auxiliares por los que se transmiten o se almacenan temporalmente los datos para poder operar con ellos. Contiene un **BANCO DE REGISTROS**, usualmente de 8 a

256, donde se almacenan los datos y resultados parciales más utilizados para evitar accesos a la memoria principal, que son siempre más lentos.

Normalmente la ALU opera en paralelo con datos, de una determinada longitud, que se proporcionan a través de dos buses de entrada, y proporciona el resultado a través de un bus de salida. La ALU dispone de



otras entradas a través de las cuales la unidad de control especifica la operación concreta que debe realizar con los datos de entrada en un momento dado.

Por ejemplo, una operación de suma en la ALU se representa así:

$$BS \leftarrow BA + BB$$

→ **MEMORIA CENTRAL**. La memoria está dividida en posiciones o **palabras de memoria**, de un determinado número de bits n , que es donde se almacena o memoriza la información.

La E/S en la memoria se puede hacer con un bus bidireccional, que se puede utilizar para ambas finalidades, o con buses específicos para el grabado o lectura de datos. Los buses son de n bits de forma que en un instante dado se leen los bits de una posición completa de memoria.

Además la memoria dispone de un **bus de direcciones** de m bits, pudiendo direccionar en total 2^m posiciones distintas de memoria.

La capacidad máxima de una memoria viene dada por la expresión:

$$C_{mp} = 2^m \text{ palabras} = \frac{2^m \cdot n}{8} \text{ bytes}$$

Para escribir un dato, usaremos la expresión $M[AR] \leftarrow DR$, donde AR es el registro que contiene la dirección de memoria a la que se accederá y

* **DE CONTROL** lleva las señales de control y de estado, indicando la dirección de la transferencia de datos, coordinando la temporización de eventos durante la transferencia, transmitiendo señales de interrupción, etcétera.

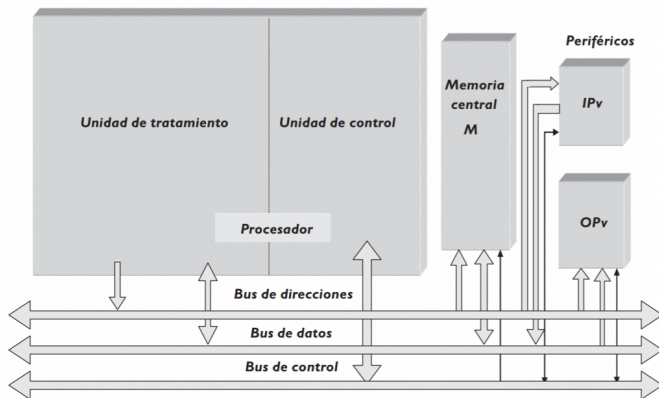


Figura 7.6. Interconexión de las distintas unidades funcionales de un computador con un bus único.

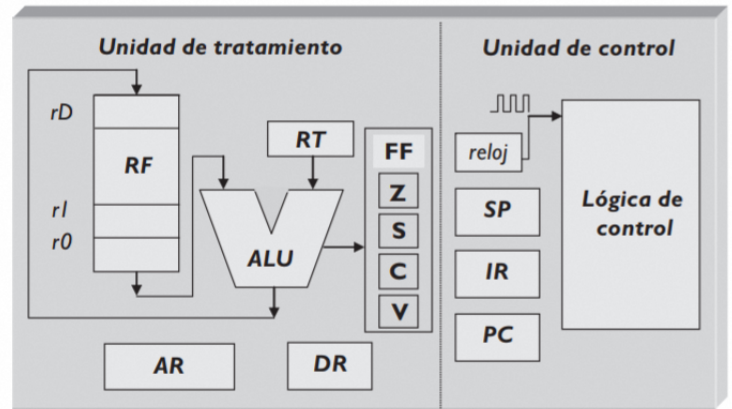


Figura 7.9. Elementos básicos de un procesador.

Ejemplo de interconexión de las distintas unidades funcionales.

ESTRUCTURA DE UN PROCESADOR

El procesador determina el funcionamiento de un computador, y está compuesto por la Unidad de Control y Bloque de Tratamiento. Este último conformado por:

→ **BLOQUE DE REGISTROS** o **Archivo de registros** o **Register File** que se usan como almacén temporal de los datos con los que va a operar la ALU/ de resultados intermedios/direcciones de memoria. La longitud de estos registros suelen ser de una palabra, aunque también se pueden encontrar de media y doble en un computador.

A veces los lenguajes máquina incluyen instrucciones para operar entre un registro significativo (ACUMULADOR) y otro registro o una posición de la memoria, almacenando el resultado en el acumulador.

En otros casos también se usa un **REGISTRO TEMPORAL** para almacenar uno de los operandos, con la ALU operando entre un registro de RF y RT, guardando el resultado en un registro de RF. *Almacena un dato.*

→ **BIESTABLES INDICADORES / DE CONDICIÓN** (flip flops en inglés) valen 0 ó 1 en función de la última operación realizada por la ALU.

Existen biestables como:

C: Acarreo

S: Indicador de signo ($S=1$, negativo)

Z: Indicador de cero ($Z=1$, resultado 0)

V: Indicador de desbordamiento

P: Indicador de paridad

Los periféricos se conectan a la memoria principal a través de los buses de dirección y de datos. Para que el procesador funcione bien, hay:

→ **REGISTRO DE DIRECCIÓN (AR)** donde debe ubicarse la dirección del dato/instrucción a leer o escribir.

→ **REGISTRO DE DATOS (DR)** donde se almacenará el dato a escribir en la memoria o en un puerto de salida, o la información leída de la memoria o un puerto de entrada, dependiendo del caso.

Respecto de la Unidad de Control, lo conforma:

→ **LÓGICA DE CONTROL** constituido por los circuitos que generan las distintas señales de control

→ **RELOJ**, generador de pulsos, con los que se sincronizan las microoperaciones que implican la ejecución de las distintas instrucciones máquina.

→ **REGISTRO DE INSTRUCCIÓN (IR, Instruction Register)** que memoriza la instrucción que está ejecutando.

→ **CONTADOR DE PROGRAMA (PC, Program Counter)** es un registro-contador que actúa como contador ascendente binario, con posibilidad de carga paralela y que contiene la dirección de memoria donde se encuentra la próxima instrucción a ejecutar.

→ **PUNTERO DE PILA (SP, Stack Pointer)** contiene una dirección de memoria para gestionar las llamadas y retornos de subrutinas

EL PROCESO DE EJECUCIÓN DE INSTRUCCIONES

Para iniciar la ejecución de un programa:

Ubicamos en el PC la dirección de memoria donde comienza dicho programa. El contenido pasa del PC a la AR, $AR \leftarrow PC$, y la Unidad de Control (CU/Control Unit) da la orden de leer.

Después de un tiempo determinado (**Tiempo de acceso a la memoria**), en el bus de E/S de memoria, aparecerá la información contenida en la posición de dirección PC, la primera instrucción del programa, cargándose en el registro DR (Data Registry), $DR \leftarrow M(AR)$.

La información en la DR, es decir, la instrucción se carga al IR (Instruction Registry), $IR \leftarrow DR$, e incrementamos el PC un punto, $PC \leftarrow PC + 1$, para apuntar a la siguiente instrucción.

FASE DE CAPTACIÓN DE INSTRUCCIÓN

Una vez cargada la información en IR, se decodifica y ejecuta bajo la supervisión de la Unidad de Control (CU).

Después de haber ejecutado la instrucción, la CU repite todo el proceso explicado.

FASE DE EJECUCIÓN

En resumen, la **FASE DE CAPTACIÓN DE INSTRUCCIONES** y la **FASE DE EJECUCIÓN** realizará los operaciones específicas correspondientes al codop (Código de Operación) que tenga la instrucción almacenada en la IR.

1. $AR \leftarrow PC$
2. $DR \leftarrow M(AR)$
3. $IR \leftarrow DR$
4. $PC \leftarrow PC + 1$

Al conjunto de operaciones detalladas previamente son las acciones más elementales que puede hacer el computador, y reciben el nombre de **MICROOPERACIONES**.

Nota editor

* Estamos asumiendo que cada instrucción ocupa una sola posición en la memoria.

* La ejecución de una instrucción

conlleva por lo menos un acceso a la memoria, pudiéndose realizar accesos adicionales (ej: memorización de resultados).

* Las μ operaciones están sincronizadas con el reloj, consumiendo uno o varios ciclos de reloj.

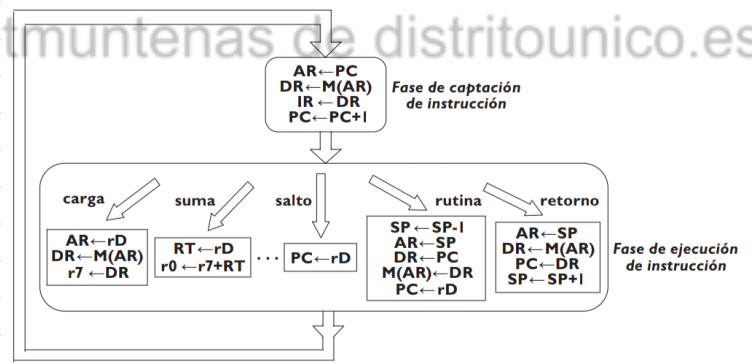


Figura 7.13. Diagrama mostrando las microoperaciones que tienen lugar cuando se ejecutan las instrucciones de carga de registro, suma, salto, llamada a subrutina y retorno de subrutina.

Unas instrucciones tardan más en ejecutarse (consumen más ciclos), lo podemos medir con la siguiente expresión $t_{ei} = \frac{\text{número de ciclos}}{\text{frecuencia del reloj}}$

GESTIÓN DE LAS INSTRUCCIONES DE CONTROL

Con el esquema anterior vemos que el procesador ejecuta secuencialmente las instrucciones del programa que se encuentran en posiciones consecutivas de memoria.

Las instrucciones de control permiten alterar el orden de ejecución de un programa, saltando a la ejecución de una instrucción ubicada en una dirección de memoria arbitraria (**ds**), fijada por el programador.

Una vez ejecutada la instrucción de salto, el programa ejecuta la instrucción contenida en la posición **ds** y las que se encuentren sucesivamente a partir de ella.

Hay distintos tipos de instrucción de control:

→ **Salto (jump, branch):** Bifurcaciones de salto condicional/incondicional.

En la fase de ejecución el procesador cambia el contenido del contador

de programa por **ds**, $PC \leftarrow ds$, de forma que, en la siguiente fase de captación de instrucciones, se capta la instrucción almacenada en **ds**, y se produce el salto.

En los saltos condicionales el salto sólo se produce si se cumple alguna o algunas condiciones, establecidas por los biestables indicadores o algún registro.

→ **Llamadas a subrutinas (call)** condicionales o incondicionales.

Es una ruptura de la secuencia normal de ejecución de un programa, de forma que tras la ejecución de la **call**, se ejecuta otro programa (subrutina/subprograma), y terminada su ejecución, a través de una llamada de retorno, **return**, se vuelve al programa desde el que se hizo la **call**, siguiendo su ejecución.

✳️ Esto es posible gracias a las memorias pila o LIFO (Last Input First Output)

Las direcciones de retorno necesitamos almacenarlas temporalmente durante la ejecución de las instrucciones **call**. Estas direcciones se extraerán de la memoria al ejecutarse las instrucciones **return**.

Entonces, al igual que se produce el anidamiento de subrutinas, las direcciones de retorno hay que extraerlas de la memoria en el orden inverso al que se introdujeron, es decir, recorro del revés el camino del desiño.

Por ejemplo, si voy por la autovía a Granada, si me desvió a Almería y luego a Murcia, para volver a Granada seguiré el camino a la inversa, de Murcia a Almería y de Almería a Granada.

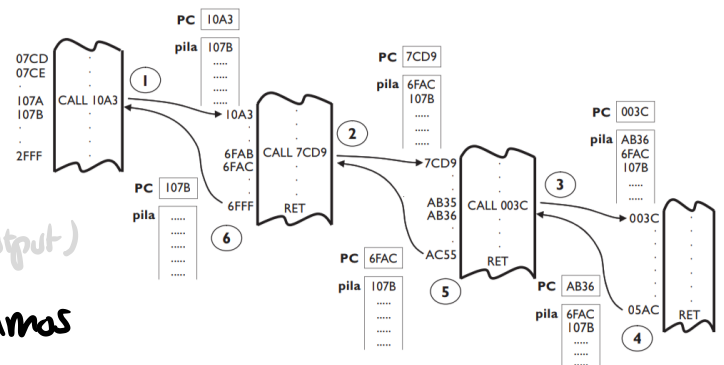
1. Granada
2. Almería
3. Murcia

ORDEN DE
REGISTRO

⇒

1. Murcia
2. Almería
3. Granada

Orden devuelto por
la memoria LIFO.



✓ Entonces, las operaciones de la llamada serán: $Pila \leftarrow PC$, $PC \leftarrow ds$, y el retorno $PC \leftarrow Pila$.

Las memorias LIFO pueden ser físicas (Hardware) o virtuales (Software), el último almacenando las direcciones en la MP (Memoria Principal).

La pila Software se implementa así:

$SP \leftarrow SP + 1$

$AR \leftarrow SP$

$AR \leftarrow SP$

$DR \leftarrow M(AR)$

$DR \leftarrow PC$

$PC \leftarrow DR$

$M(AR) \leftarrow DR$

$SP \leftarrow SP + 1$

$PC \leftarrow rs$

RETORNOS

LLAMADAS

DISEÑO DE LA CONTROL UNIT

Antes de abordar CODE 2, estudiaremos la UNIDAD DE CONTROL.

La Control Unit se encarga de detectar **señales de estado** procedentes de las distintas unidades, Capta de la memoria una a una las **instrucciones máquina** del programa, y genera las **señales de control** dirigidas a todas las unidades, monitorizando las operaciones que implican la ejecución de la instrucción. Todos los procesos sincronizados por el reloj.

Por tanto, es el subsistema que toma como **entradas** el contenido del registro IR, el estado actual del procesador y la señal periódica del reloj, para generar como **salidas** las señales de control (microórdenes) convenientemente secuenciadas en el tiempo.

La UC genera para cada ciclo de reloj dentro de una instrucción ($c_1, c_2, \dots$) un conjunto de X valores binarios que monitorizan las acciones de todos los elementos del computador.

La UC puede ser implementada por medio de funciones de conmutación (puertas lógicas, cableada) o microprogramada (en memoria de control, los X valores de señales de control a generar a cada instrucción/ciclo).

INTEGRADOS

Como conclusión a todo este bloque introductorio, decir que gracias a los avances científicos actuales, se puede condensar todo el conglomerado de componentes de un procesador en un único circuito integrado.

Adicionalmente, con el tiempo se han creado procesadores específicos para una finalidad, como las GPUs, DSP, etc.

CODE 2

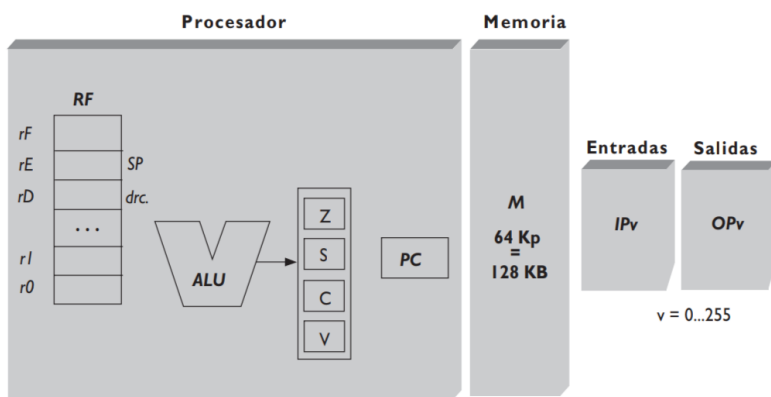
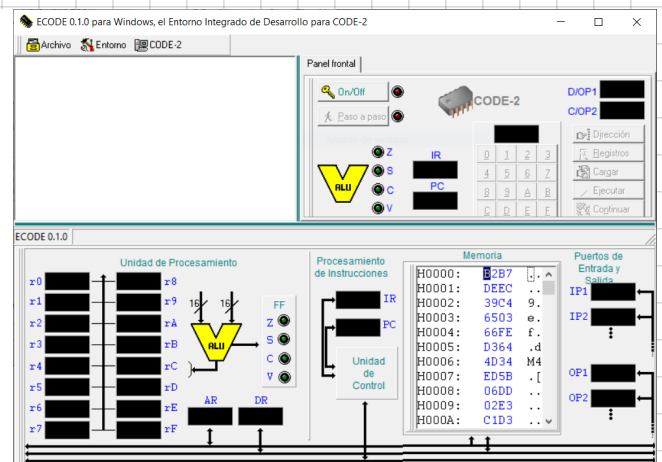


Figura 8.11. Esquema simplificado del Computador Didáctico Elemental (CODE-2).



CODE-2 es un computador de tipo von Neumann que contiene: entradas, salidas, unidad de control y de procesamiento y memoria de datos.

CODE-2 es de 16 bits de longitud de palabra y tiene tan sólo 16 instrucciones máquina. Vamos a describir su funcionamiento al **nivel de máquina convencional**:

ELEMENTOS A LOS QUE SE TIENE ACCESO DESDE EL LENGUAJE MÁQUINA

- **BANCO DE REGISTROS (RF)** compuesto de 16 registros r0, ..., rF. El registro rE se utiliza como puntero de pila (SP) y el rD como registro de dirección. Aún así, el programador puede usar ambos registros para otros fines.
- **UNIDAD ARITMÉTICO LÓGICA (ALU)** con la que se pueden realizar las operaciones de suma y resta, operación lógica NAND, y desplazamientos lógicos del acumulador a derecha a izquierda y desplazamiento aritmético a la derecha.

→ **Biestables indicadores (FF)** Se activan después de hacer operaciones con la ALU. Se incluyen los biestables de cero (Z), signo (S), acarreo (C) y desbordamiento (V).

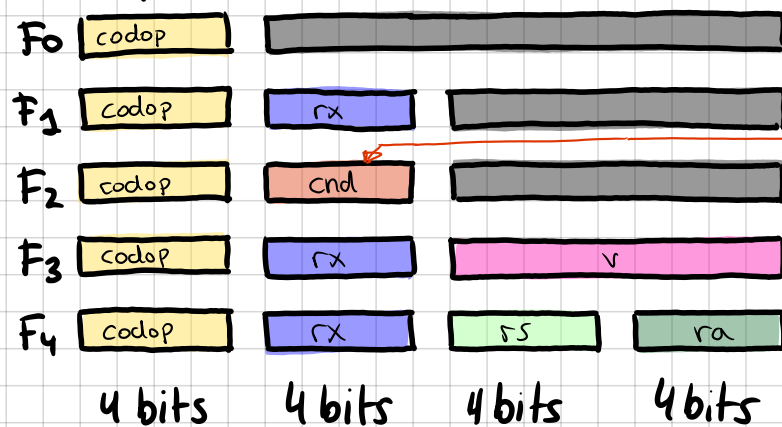
→ **Memoria principal (M)** de $2^{16} = 64K$ palabras de 16 bits (128 KBytes).

→ **Puertos de entrada (IP)** Hasta 256 (IP00 a IPFF) de periféricos.

→ **Puertos de salida (OP)** Hasta 256 (OP00 a OPFF) de periféricos.

FORMATOS DE INSTRUCCIONES Y DATOS REPERTORIO DE INSTR. MÁQUINA

El repertorio de instrucciones de CODE-2 utiliza cinco tipos de formatos:



En CODE-2 trabajamos en HEXADECIMAL. Una instrucción CODE-2 estará formado por 4 caracteres hexadecimales o 16 valores binarios.

Estudiemos las 16 instrucciones que nos ofrece CODE-2:

- 0 **Cargar** un registro con un dato de la memoria (*) cuya dirección se obtiene sumando el contenido del registro rD con el valor del campo v de la propia instrucción
- 1 **Almacenar** el contenido de un registro en la memoria (*)

Tabla 8.3. Valores que toman los biestables indicadores en función del resultado de las instrucciones que se indican.

Instrucción	Biestable			
	Z	S	C	V
ADDS	*	*	*	*
SUBS	*	*	*	*
NAND	*	*	—	—
SHL	*	*	*	—
SHR	*	0	*	—
SHRA	*	*	*	—

Notas:

1. Se indican sólo las instrucciones que actúan sobre los biestables indicadores.
2. * significa que el biestable se modifica de acuerdo con el resultado de la operación.
3. — significa que el biestable no se modifica.

Tabla 8.4. Tipos de saltos y llamadas a subrutinas, y códigos utilizados para especificar cada uno de ellos.

Tipo de salto o llamada	HEX	—	end
Incondicional	0	R	0000
Condional, si biestable de cero es 1	1	Z	0001
Condional, si biestable de signo negativo es 1	2	S	0010
Condional, si biestable de acarreo es 1	3	C	0011
Condional, si biestable de desbordamiento es 1	4	V	0100

Verificado con el emulador

Tabla 8.5. Repertorio de instrucciones de CODE-2.

Codop	Nombre	Nemónico	Parámetros	Formato	Explicación	Nº ciclos
Binario	Hex					
0000	0	Cargar	LD rx,[v]	F3	rx ← M(rD + v)	9
0001	1	Almacenar	ST [v],rx	F3	M(rD + v) ← rx	9
0010	2	Carga inmediata baja	LLI rx,v	F3	rx(15:8) ← H'00; rx(7:0) ← v	6
0011	3	Carga inmediata alta	LHI rx,v	F3	rx(15:8) ← v	8
0100	4	Entrada	IN rx,IPv	F3	rx ← IPv	8
0101	5	Salida	OUT OPv,rx	F3	OPv ← rx	8
0110	6	Suma	ADDS rx,rs,ra	F4	rx ← rs + ra	7
0111	7	Resta	SUBS rx,rs,ra	F4	rx ← rs - ra	7
1000	8	NAND	NAND rx,rs,ra	F4	rx ← (rs · ra)'	7
1001	9	Desplaza izquierda	SHL rx	F1	C ← rx(15), rx(i) ← rx(i + 1), i = 15,...,1; rx(0) ← 0	6
1010	A	Desplaza derecha	SHR rx	F1	C ← rx(0), rx(i) ← rx(i + 1), i = 0,...,14; rx(15) ← 0	6
1011	B	Desplaza arit. dcha.	SHRA rx	F1	C ← rx(0), rx(i) ← rx(i + 1), i = 0,...,14	6
1100	C	Salto	B- cnd	F2	Si cnd se cumple, PC ← rD	6
1101	D	Subrutina	CALL- cnd	F2	Si cnd se cumple, rE ← rE - 1, M(rE) ← PC, PC ← rD	6/10
1110	E	Retorno	RET -	F0	PC ← M(rE); rE ← rE + 1	8
1111	F	Parar	HALT -	F0	Parar	6

- [2] **Carga Inmediata Baja** carga inmediatamente el byte menos significativo de un registro rx , con el valor de v . El byte menos significativo se pone a $H'00$.
- [3] **Carga Inmediata alta** carga inmediatamente el byte un registro rx , con el valor del campo v de la propia instrucción. El byte menos significativo conserva su valor previo.
- [4] **Entrada**, capta el contenido de un puerto de entrada. Lleva la información de un puerto IP_v (donde v se especifica en la instrucción) y lo lleva al registro rx .
- [5] **Salida**, lleva el contenido del registro rx al puerto de salida OP_v .
- [6] **Suma** (*) del contenido de 2 registros, rs y ra , depositando el resultado en rx . Los cuatro biestables indicadores (Z , S , C y V) se activan de acuerdo con el resultado de la operación.
- [7] **Resta** (*) $\rightarrow$ [o uno u otro, pero no los dos]
- [8] **NAND**, operación lógica NAND bit a bit (*) Se activan biestables indicadores z y s de acuerdo con el resultado de la operación.
- [9] **Desplazamiento a la Izquierda** desplaza los bits del registro rx una posición a la izquierda. El bit más significativo pasa al biestable de acarreo (C) y el bit menos significativo se hace cero. El biestable de signo (S) pasa a tomar el valor del nuevo bit más significativo, y el de cero (Z) pasa a valer 1, si el nuevo valor de registro es $0000_{H'}$.
- [A] **Desplazamiento a la Derecha** desplaza los bits del registro rx una posición a la derecha. El bit menos significativo pasa al biestable de acarreo (C) y el bit más significativo se hace cero. El biestable de signo (S) pasa a tomar el valor del nuevo bit más significativo, y el de cero (Z) pasa a valer 1, si el nuevo valor de registro es $0000_{H'}$.

Tabla 8.7. Operación lógica NAND: $c = (a \cdot b)'$.

a	b	$c = (a \cdot b)'$
0	0	1
0	1	1
1	0	1
1	1	0

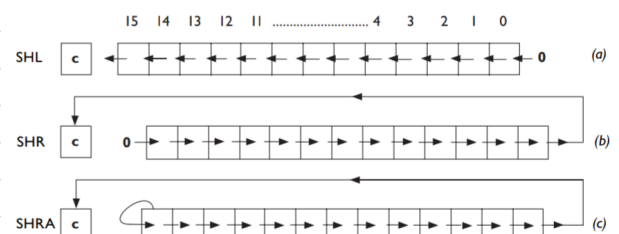


Figura 8.24. Desplazamiento de los bits de un registro provocado por las instrucciones: (a) SHL, (b) SHR, y (c) SHRA.

- [B] **Desplazamiento aritmético a la Derecha** desplaza los bits del registro rx una posición a la derecha. El bit menos significativo pasa al biestable de acarreo (C) y el bit más significativo conserva su valor. El biestable de signo (S) pasa a tomar el valor del nuevo bit más significativo, y el de cero (Z) pasa a valer 1, si el nuevo valor de registro es 0000_H .
- [C] **Salto**, ruptura de secuencia, provoca un salto a la instrucción cuya dirección se encuentra en rD, si se cumple la condición cnd. Es decir, dependiendo de si el valor del biestable indicador especificado por cnd es 1 después de la última operación de la ALU, a no ser que sea un salto incondicional, en cuyo caso siempre lo realizará.
- [D] **Subrutina** provoca una llamada a la subrutina cuya dirección se encuentra en el registro rD, si se cumple la condición cnd. Es decir, dependiendo de si el valor del biestable indicador especificado por cnd es 1 después de la última operación de la ALU, a no ser que sea una llamada incondicional, en cuyo caso siempre lo realizará. Recuerda guardar en el PC la dirección de retorno antes de hacer la llamada. En CODE-2 se hace así: $rE \leftarrow rE - 1, m(rE) \leftarrow PC$.
- [E] **Retorno** se incluye al final de la subrutina y provoca un retorno al último programa que lo llamó, la instrucción siguiente a la de llamada. Recuerda restaurar el valor del PC desde la pila y actualizar el puntero de pila. En CODE-2 se hace así: $PC \leftarrow m(rE), rE \leftarrow rE + 1$.
- [F] **Parar** hace que CODE-2 entre en un estado de espera, deteniendo su funcionamiento antes de la siguiente fase de captación de la siguiente instrucción. Esto se utiliza para dar por finalizado un programa o poner una pausa en CODE-2 para comprobar resultados. Se puede resumir la ejecución pulsando la tecla CONTINUAR.

PROCESO DE CREACIÓN DE UN PROGRAMA

De forma resumida:

- ① **Planteamiento del problema** → saber al detalle lo que hay que hacer.
- ② **Análisis del problema** → se exploran distintas alternativas para solucionar

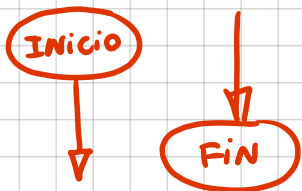
el problema, y determinar los datos E/S que se necesitan y su formato más adecuado.

- ③ Diseño del algoritmo que resuelve el problema
- ④ Implementación del algoritmo: **Programación** → representar el algoritmo en un lenguaje de programación adecuado, traducir el programa a código máquina y ejecutarlo.
- ⑤ **Prueba y optimización** → comprobar si el programa es eficiente y que funciona correctamente y búsquedas de otras opciones que resuelva mejor el problema o partes de él.

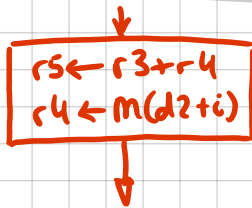
ORGANIGRAMAS

Para diseñar con facilidad la estructura lógica de los programas podemos recurrir a los organigramas. A través de la siguiente simbología podemos crear estructuras lógicas complejas:

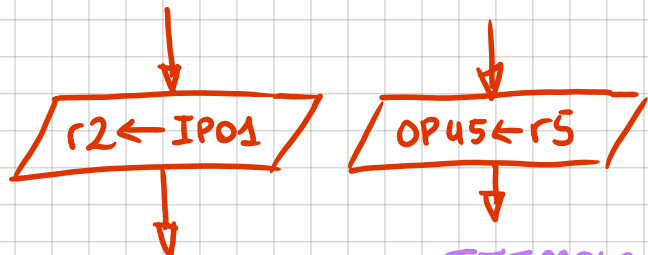
INICIO / FIN



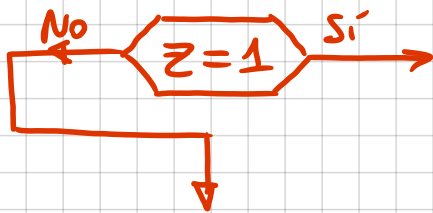
ASIGNACIONES



E/S



SALTOS / BIFURCACIONES

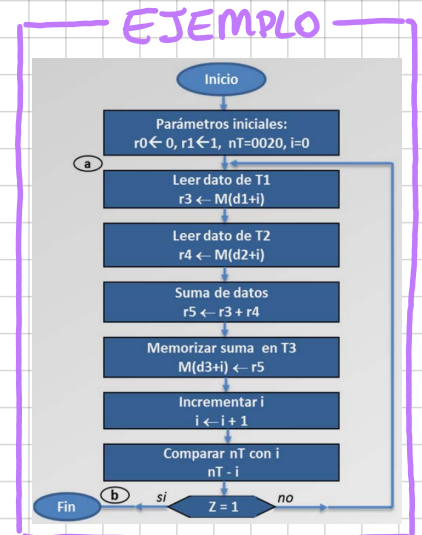


CONEXIONES



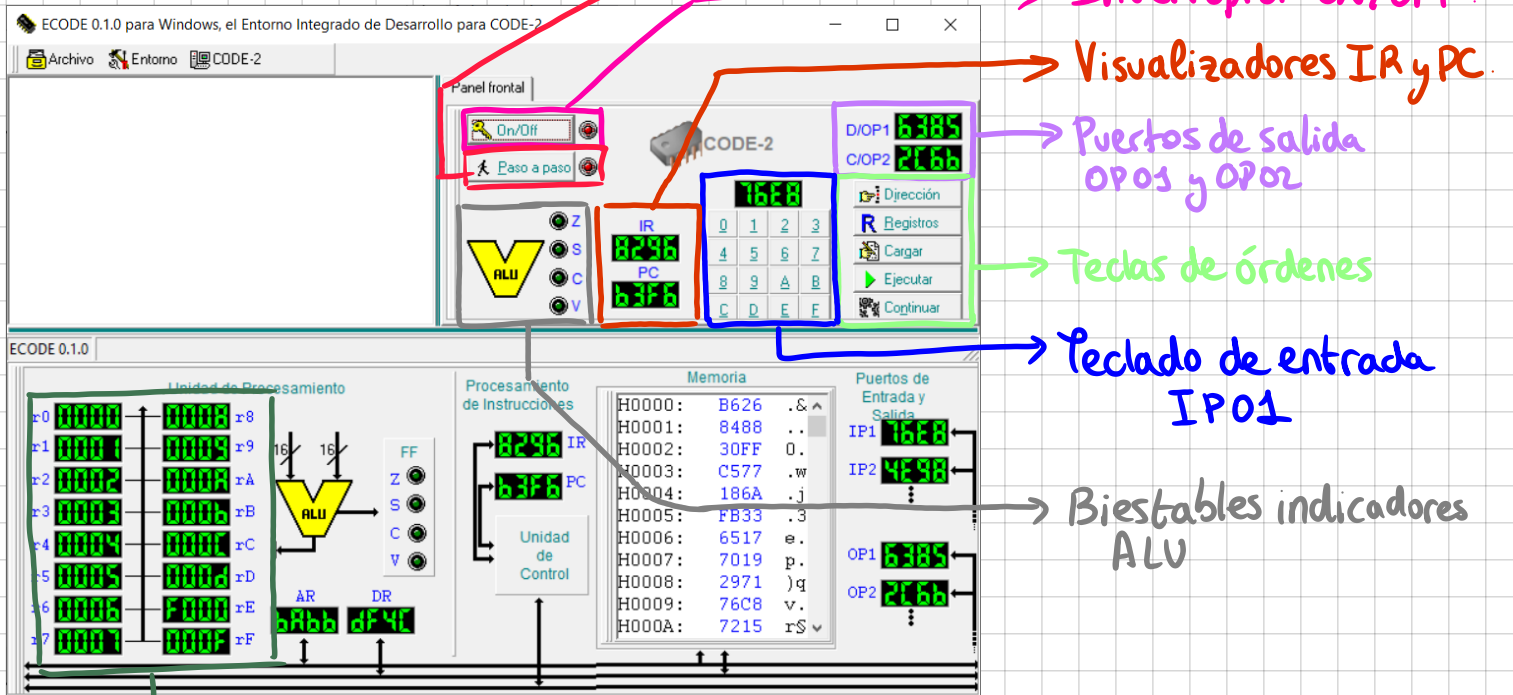
METODOLOGÍA PARA HACER PROGRAMAS EN LM.

- ① Describir el algoritmo a implementar por medio de un organigrama o con pseudocódigo
- ② Asignación de memoria → De registros y posiciones de memoria para los parámetros y variables utilizadas en el programa y de la dirección de inicio del programa.
- ③ Redactar el programa en nemónico



- ④ Codificar las instrucciones en código máquina (HEXADECIMAL o BINARIO)
- ⑤ Documentar el programa (comentarios para su uso).

INTERFAZ DE CODE-2



Una vez activado el interruptor ON/OFF pueden realizarse las siguientes acciones:

- **Seleccionar una posición de memoria** → Se teclea la dirección hexadecimal deseada y se pulsa **"DIRECCIÓN"**. En los puertos OP01 y OP02 aparecerán la dirección seleccionada y su contenido. Al presionar **"CONTINUAR"** se avanza a la siguiente dirección de memoria y se muestra su contenido.
- **Cargar información en una posición de memoria** → Tras seleccionar una dirección, se introduce la información en el teclado hexadecimal y se pulsa **"CARGAR"**. Esto almacena la información en la dirección seleccionada y avanza a la siguiente dirección de memoria, permitiendo cargar datos o programas de forma secuencial.
- **Seleccionar los registros** → Al pulsar **"REGISTROS"**, OP01 muestra el código del primer registro (0000) y su contenido. Con continuar se muestran los códigos de los siguientes registros (0001, 0002, ..., 000F) y sus contenidos.
- **Cargar información en un registro** → Después de seleccionar un registro, se introduce la información a almacenar y se pulsa **"CARGAR"**. Guarda la información en el registro y avanza al siguiente.

→ **Ejecutar un programa** → Se selecciona la dirección de memoria de la primera instrucción del programa y se pulsa "EJECUTAR"

DESCRIPCIÓN DE UN ODE A NIVEL DE LENGUAJE DE ENSAMBLADOR

COMPARACIÓN
LM y LE

El lenguaje máquina tiene dos importantes ventajas:

- Es directamente interpretable por el procesador
- Los programas en lenguaje máquina se ajustan a los componentes del sistema, permitiendo ser más eficientes en términos de tiempo de ejecución y consumo de memoria.

Sin embargo, presenta varios inconvenientes:

- Las instrucciones se escriben en binario, octal o hexadecimal, dificultando la comprensión del programa, ya que es una secuencia de números sin un significado a primera vista.
- El programador debe decidir dónde colocar los distintos elementos del programa, como el código y los datos, complicando su organización.

Para superar estos inconvenientes, se utiliza el lenguaje ensamblador. Este lenguaje mantiene las ventajas del lenguaje máquina, pero con las siguientes mejoras:

- Las instrucciones se representan con palabras clave que describen la acción a realizar, **mnemónicos**.
- Se pueden usar nombres simbólicos para referirse a variables y direcciones de memoria.

Es decir, hacer los programas más legibles. Pero el lenguaje ensamblador necesita ser traducido a lenguaje máquina, mediante un ensamblador, programa que convierte las instrucciones en ensamblador en su equivalente en máquina.

El proceso de traducción puede realizarse en el mismo sistema o mediante un ensamblador cruzado. Un ejemplo de esto es el ensamblador de CODE-2, en el que el programa resultante en lenguaje máquina se carga y ejecuta en el

sistema CODE-2 o un emulador.

PROCESADORES CISC y RISC ☒ Ampliado!!

Los procesadores CISC se caracterizan por tener un conjunto amplio y complejo de instrucciones que permite la realización de operaciones complejas entre operandos situados en la memoria o en los registros internos. Son fáciles de programar y utilizan la memoria de manera más eficiente, pero su rendimiento se ralentiza porque el tiempo que toman las diferentes instrucciones no es el mismo. Además, son más grandes y requieren de más transistores. Por ejemplo, los procesadores Intel Core.

Por otro lado, los procesadores RISC tienen instrucciones más simples y reducidas, lo que permite que las instrucciones se ejecuten más rápidamente, pues se necesitan menos ciclos de reloj por instrucción. Además, utiliza un sistema de direcciones no destructivas en RAM, reduciendo la ejecución de nuevas operaciones. Como desventaja, el rendimiento del procesador depende del programador o compilador, disponen de grandes memorias caché en el propio chip, y requieren que el hardware del chip se re programe continuamente. El CODE-2 es un procesador de este tipo.