Instrucciones del procesador

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


 C000                   ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START   LDS    #STACK

                ****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                ****************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from termin

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG I
 C014 24 FA             BCC    INCH      RECIEVE NOT READ
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                ****************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
 C020 81 30             CMP A  #'0       ZERO
 C022 2B 11             BMI    HEXERR    NOT HEX
 C024 81 39             CMP A  #'9       NINE
 C026 2F 0A             BLE    HEXRTS    GOOD HEX
 C028 81 41             CMP A  #'A
 C02A 2B 09             BMI    HEXERR    NOT HEX
 C02C 81 46             CMP A  #'F
 C02E 2E 05             BGT    HEXERR
 C030 80 07             SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                RTS

 C035 7E C0 AF  HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

Instrucciones del procesador

TEMAS:

- Qué es y qué hace una instrucción máquina
- Como se representan las instrucciones: formatos;
- Qué información contiene una instrucción: operaciones y operandos
- Qué tipos de instrucciones existen
- Qué tipos de operandos existen
- Cómo referencian la ubicación de los operandos: modos de direccionamiento
- Tipos de repertorios de instrucciones: RISC vs CISC

Instrucciones máquina (repaso)

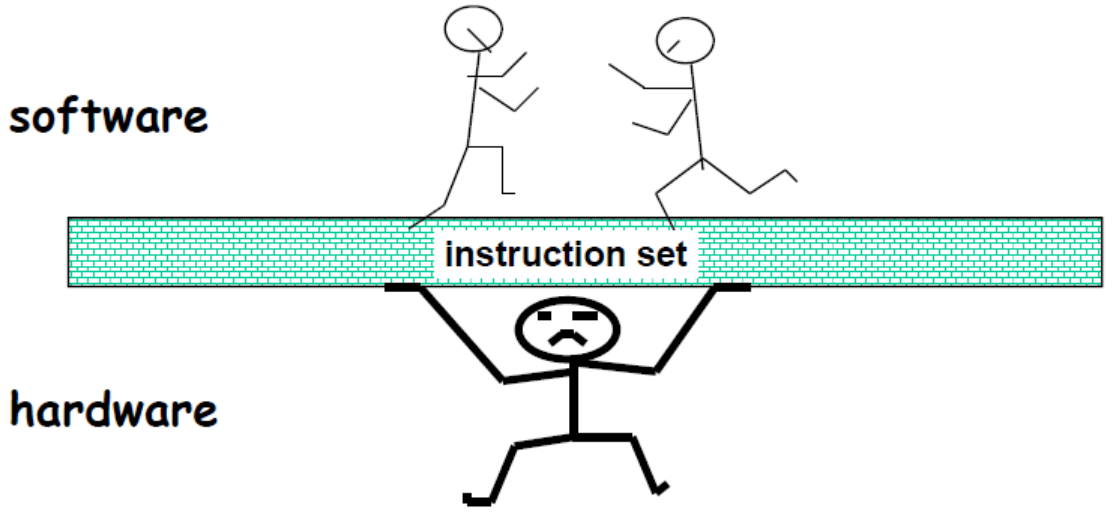
- Las operaciones posibles del circuito están determinadas por un conjunto finito y predefinido de instrucciones, denominado **repertorio de instrucciones máquina.**
- Una instrucción máquina es, formalmente, un código binario que se *interpreta* y *ejecuta* por parte del procesador mediante una secuencia de **microoperaciones.**
- Un conjunto de instrucciones máquina pueden combinarse libremente para, mediante algoritmos, crear un **programa**, que se almacena en una memoria (≡ ***software***)
- Una **unidad de control** del sistema debe proveer la secuencia de microoperaciones que realicen la búsqueda en la memoria de cada instrucción, la decodificación de su código de operación, y la generación de las microoperaciones sobre el camino de datos que ejecuten la operación indicada.

Arquitectura del repertorio de instrucciones (ISA)

La ISA (*instruction set architecture)* es una abstracción fundamental de la computadora, ya que es la interfaz que define cómo el software (el conjunto de instrucciones máquina) controla el hardware.

Sus principales funciones son especificar:

- La organización de la memoria
  - La cantidad de memoria direccionable y la organización de la información dentro de ella.
- El conjunto de registros
  - La cantidad, identificación, y el tipo de uso (general o específico) de los registros.

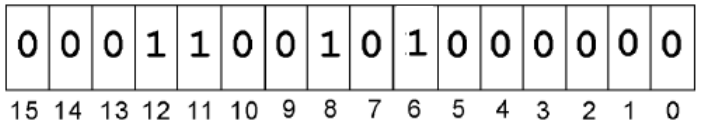


- El conjunto de instrucciones
  - La codificación de instrucciones; la descripción de las operaciones; la definición de tipos de datos y los modos de direccionamiento de la memoria.
- Los mecanismos de Entrada/Salida
  - Direcciones de E/S compartidas o no con la memoria. Mecanismos de interrupciones

Lenguaje ensamblador

- Cada instrucción máquina es una cadena de bits.
- Sin embargo, es difícil para el programador tratar con las representaciones binarias.

  Una simplificación es usar códigos hexadecimales, que agrupan 4 bits, reduciendo la longitud del valor y haciéndolo más “legible”:



$\equiv 1940_{16}$

- Una mejora adicional es utilizar una representación simbólica de la instrucción, que emplee símbolos más parecidos al lenguaje humano.

- Los **códigos de operación** (*CodOps*) se representan por medio de abreviaturas, llamadas *mnemónicos* que indican la operación. Por ejemplo:
  - ADD - adición (suma)
  - SUB - sustracción (resta)
  - MOV - movimiento de un dato
  - AND, OR, XOR - operaciones lógicas

Lenguaje ensamblador

- Los operandos también se pueden representar de manera simbólica:

  MOV reg1 , [dir1]

  La instrucción *copia (mueve)* el valor contenido en la posición de memoria llamada **dir1**, a un registro de la CPU denominado **reg1**.

- Esto da lugar al **Lenguaje Ensamblador** o ***Assembly:***
  - ✓ Textual (usa mnemónicos, no códigos numéricos)
  - ✓ Permite usar *etiquetas* y *directivas*, que simplifican la escritura

- Ejemplos:

```
Inicio: MOV R0, 0x0010
        MOV R1, 0x0001
        ADD R0, R1
        JMP Inicio
```

Aquí, **Inicio** es una *etiqueta*

```
V1: DW 0x001F
V2: DW 0x0FF0
        ADD R0, [V1]
        ADD R1, [V2]
```

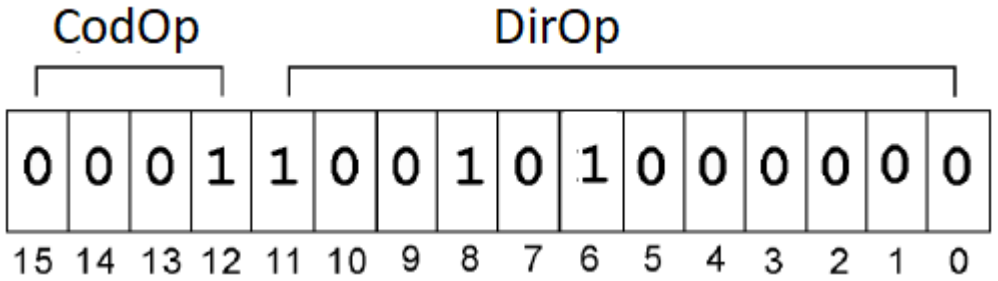
DW ("*Define Word*") es una *directiva* al ensamblador

Programa Ensamblador

- El programa llamado **Ensamblador** toma el código fuente (mnemónicos) y lo traduce al **código máquina** (bits).
- Para esto, realiza los siguientes pasos:
  - Resuelve las directivas (asigna elementos concretos del procesador).
  - Calcula el tamaño de cada una de las instrucciones, y en base a ello resuelve los valores de las etiquetas (calcula las longitudes de los saltos en el programa).
  - Traduce todas las instrucciones a ceros y unos de acuerdo al formato de la instrucción.

Formato de instrucciones

- Cada instrucción está estructurada en **campos**, que deben contener la información de:
  - La *operación específica* que debe realizarse (codificada en un *código de operación*).
  - La *localización* en la ruta de datos del *operando*
- En un caso simple:



- El código de operación (“CodOp”) dice ***qué se hace*** (“Cargar el registro R1”).
- La dirección (“DirOp”) indica ***donde está el operando*** (“el contenido de la dirección 940_{16}”).
- Este esquema se denomina **formato de instrucción.** En general, cada repertorio emplea más de un formato.

Formato general de instrucciones

- Además de la información referida a la operación que debe realizar, la instrucción debe contener la información referida a la *dirección* en la memoria de la siguiente instrucción del programa, en el caso en que sea distinta a la contenida por el Contador de Programa.
- Asimismo, en general las instrucciones pueden referenciar varios operandos a la vez para su procesamiento. Por ejemplo:

$$\text{ADD R1, A, B} \qquad R1 \leftarrow M[A] + M[B]$$

- Por lo tanto, un formato *general* de una instrucción es:

Código de Operación	Referencia a Operando fuente 1	Referencia a Operando fuente 2	Referencia a Operando resultado	Dirección próxima instrucción

Los operandos pueden ser un **registro**, una **posición de memoria**, o un **dispositivo de E/S**. Las referencias a cualquiera de ellos se denomina *dirección.*

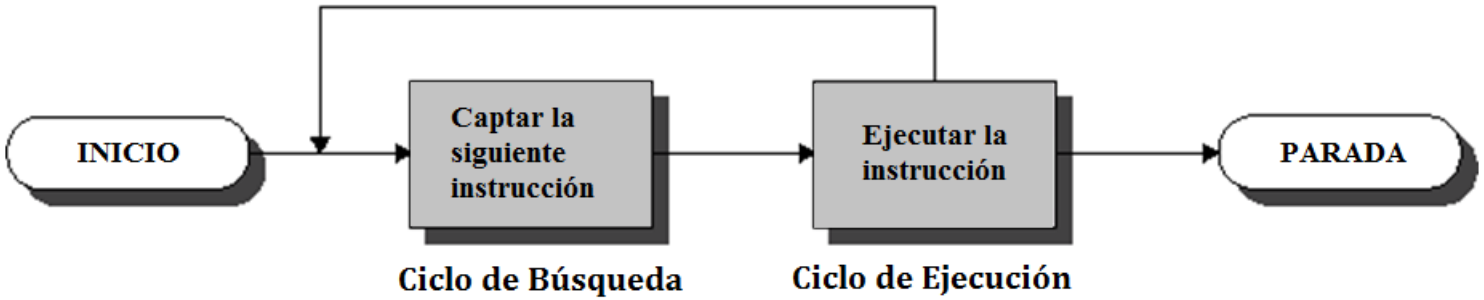
Ciclo básico de instrucción (repaso)

La CPU realiza una secuencia regular, que se repite hasta finalizar el mismo, denominada *ciclo de instrucción*:

- Traer la instrucción desde la memoria
- Realizar el procesamiento correspondiente a la instrucción específica.

Esto permite subdividir cada instrucción en al menos dos partes básicas:

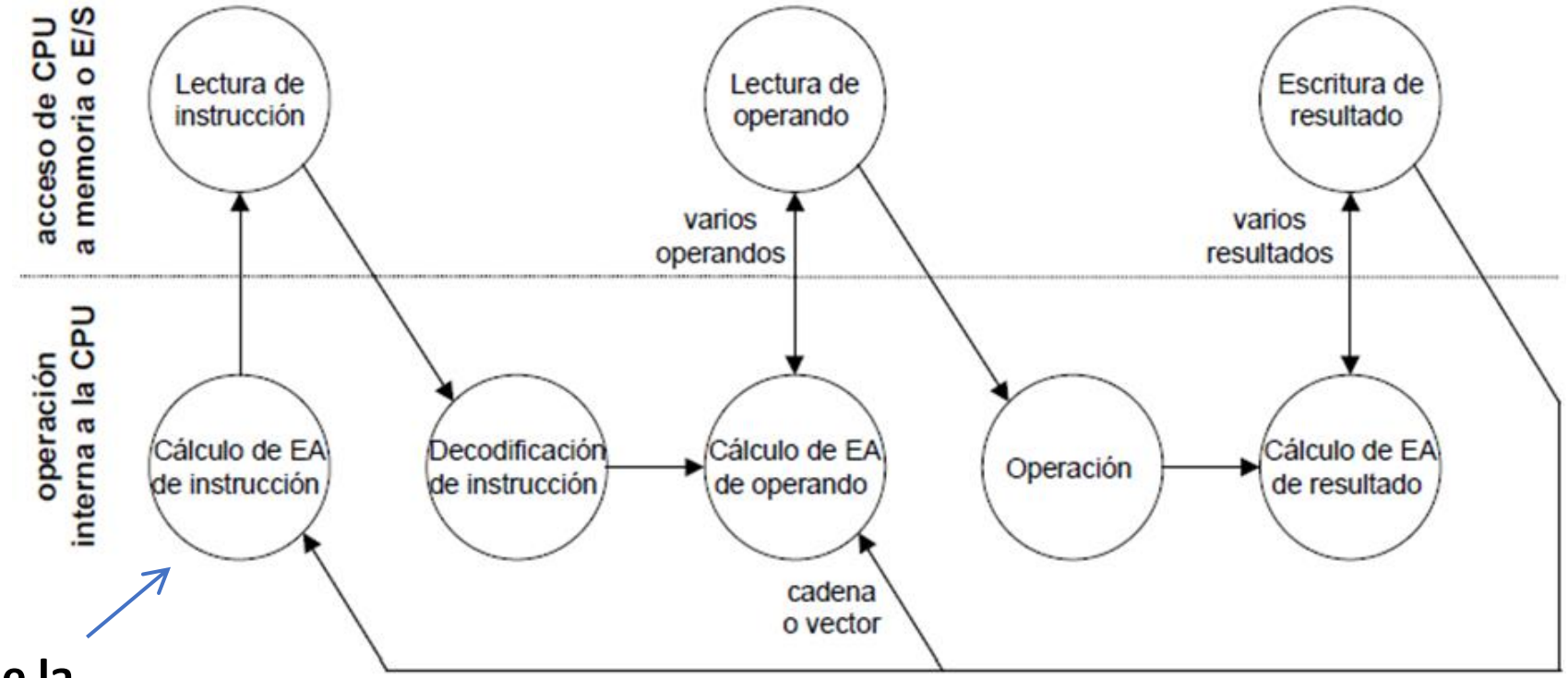
1. Un ciclo de *búsqueda* o *captación* (*fetch*).
2. Un ciclo de *ejecución*.



Ciclo de instrucción detallado

- Dado que en el caso general, una instrucción debe buscar y almacenar más de un operando, el proceso general de ejecución de cualquier instrucción puede representarse mediante el siguiente esquema:

EA = Dirección efectiva (*Effective adress*) en la memoria

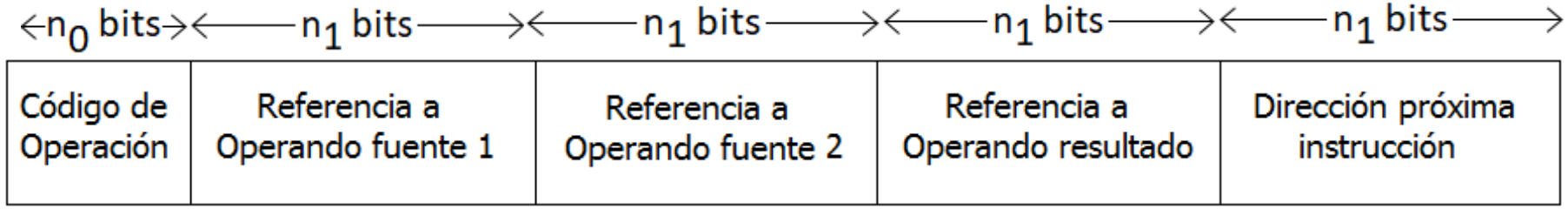


Elementos de una instrucción

- **Código de operación** (CodOp)
  Especifica la operación a realizar (suma, E/S, etc.)
- **Referencia a operandos fuente**
  La operación puede implicar uno o más operandos fuente, que son entradas para la instrucción.
- **Referencia al operando resultado:**
  La operación puede producir un resultado, salida de la instrucción.
- **Dirección de la siguiente instrucción**
  Indica a la CPU de donde captar la siguiente instrucción tras completarse la ejecución de la instrucción actual.

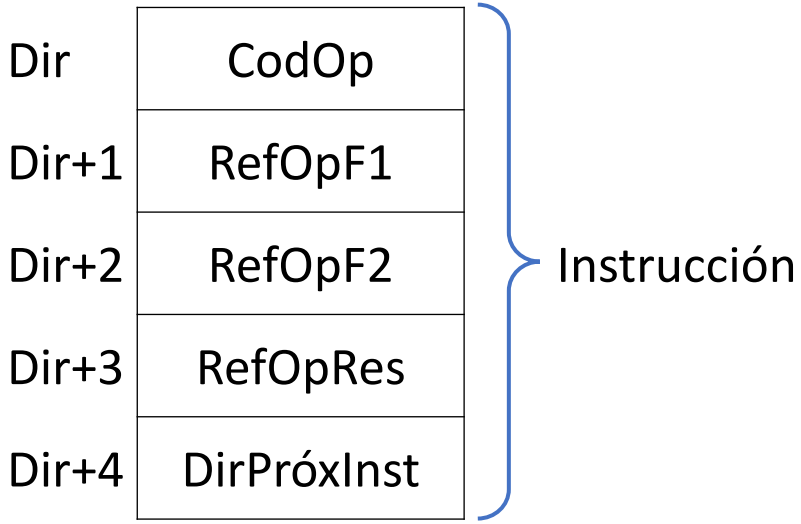
Código de Operación	Referencia a Operando fuente 1	Referencia a Operando fuente 2	Referencia a Operando resultado	Dirección próxima instrucción

Elementos de una instrucción



- En el esquema anterior, cada campo necesita un número determinado de bits, cuya suma dará la longitud total de la instrucción: N (bits) = $n_0 + 4 \times n_1$

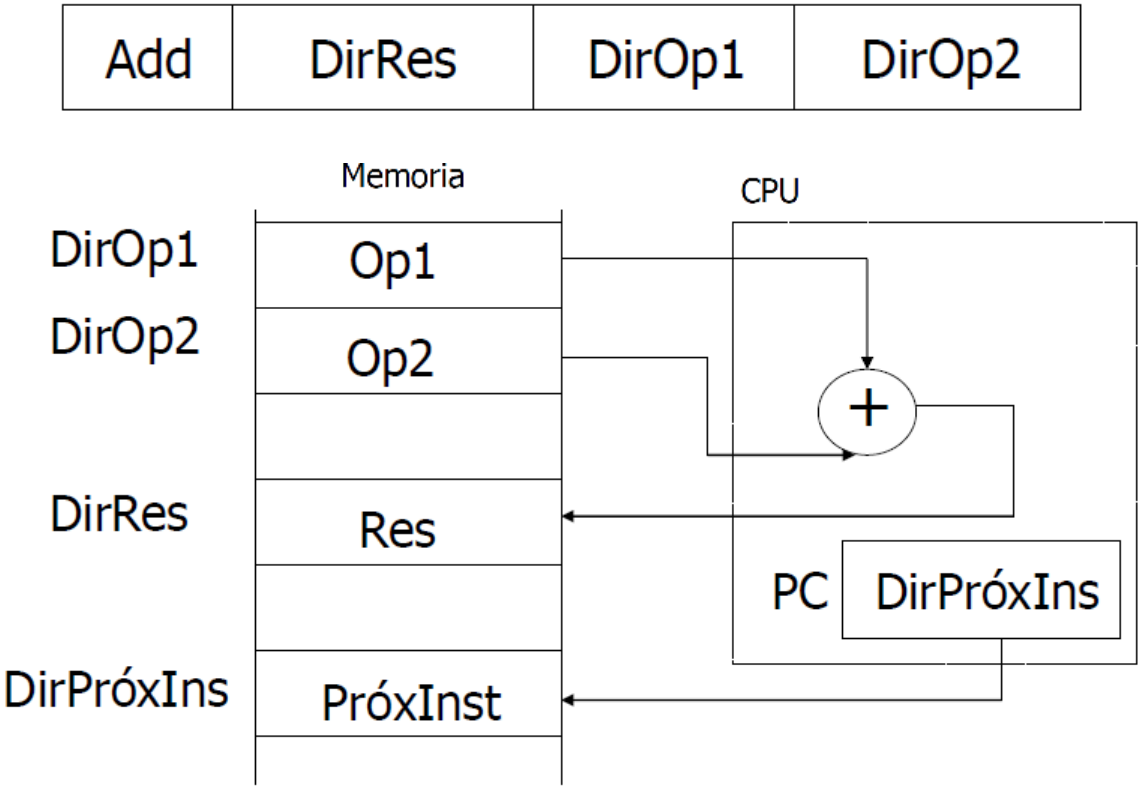
- Esto significa que la instrucción será más "larga" cuanto más campos explícitos tenga:
  - Ocupará más espacio en la memoria. Normalmente, el ancho de la palabra de memoria es menor que el largo de la instrucción → es necesario utilizar varias palabras para acomodar la instrucción.
  - Tardará más tiempo en ejecutarse, ya que necesitará mas operaciones de búsqueda en memoria.



Instrucción de tres direcciones

Add DirRes, DirOp1, DirOp2

ADD R1, A, B $\qquad R1 \leftarrow M[A] + M[B]$

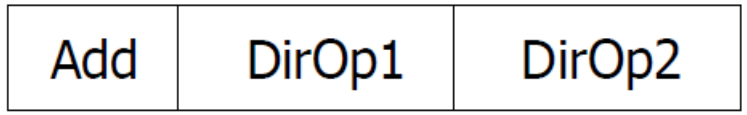


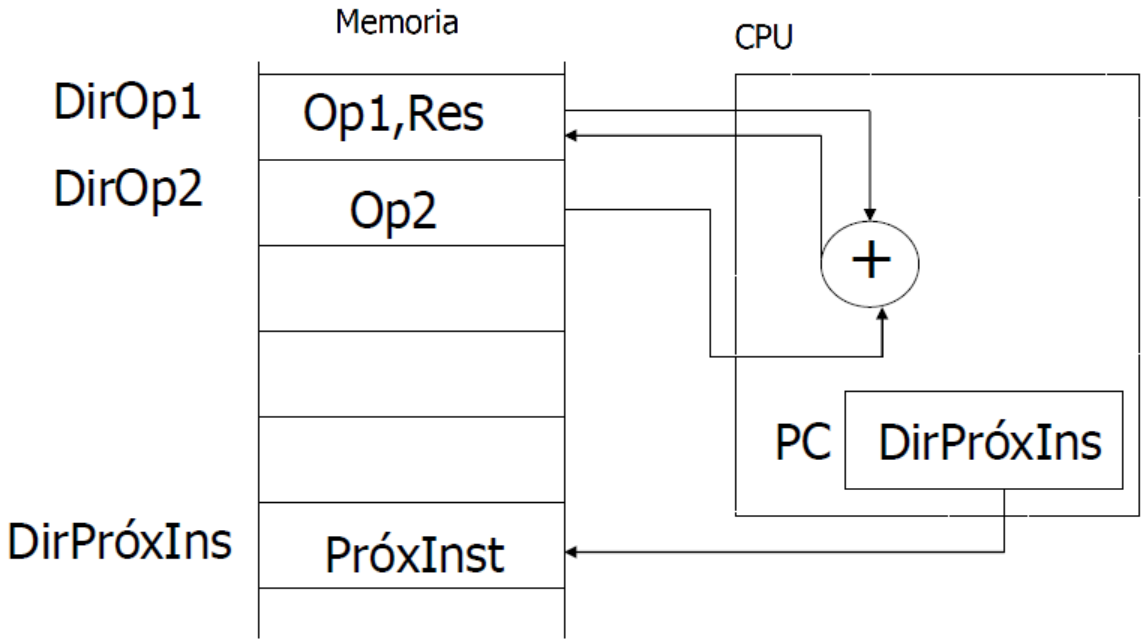
- Una primera opción es eliminar el campo de "dirección de la próxima instrucción", haciéndola implícita mediante el uso del registro Contador de Programa (PC).
- De esta manera, la referencia explícita a la próxima dirección sólo será necesaria en instrucciones de salto o bifurcación, o llamados a subrutinas.

Instrucción de dos direcciones

Add, DirOp1, DirOp2

ADD X, D $M[X] \leftarrow M[X] + M[D]$



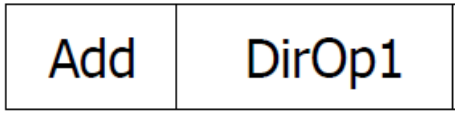


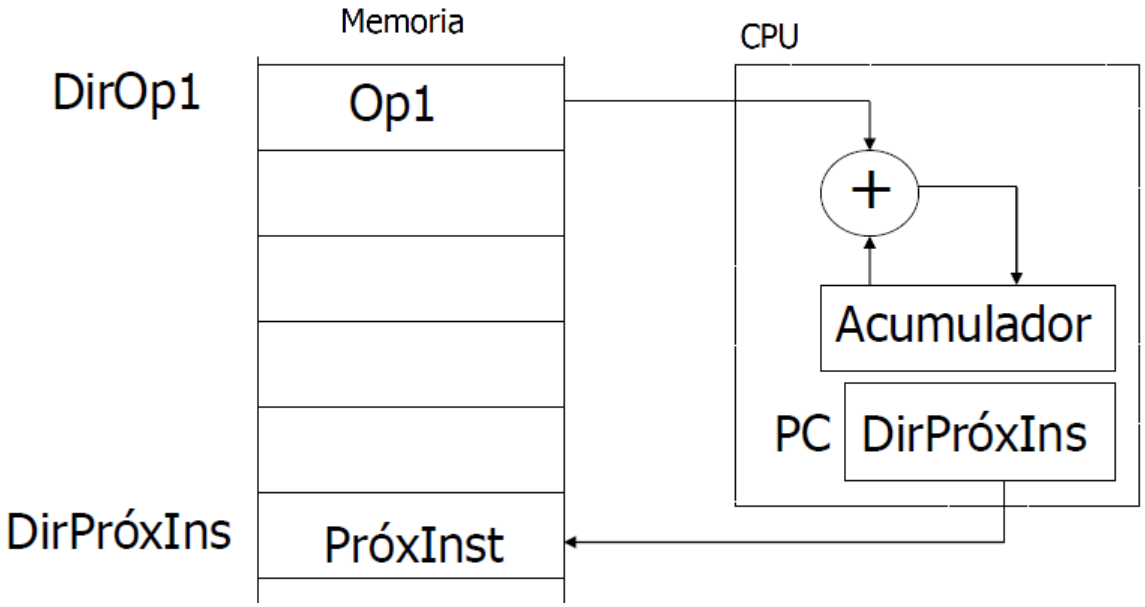
- Una de las direcciones debe hacer el servicio doble de uno de los operandos y del resultado.
- Requiere almacenamiento temporario para algún operando
- Reduce la longitud de la instrucción

Instrucción de una dirección

Add DirOp1

ADD B $ACC \leftarrow ACC + M[B]$





- Una segunda dirección debe estar implícita.
- Usualmente es el registro acumulador (ACC), que funciona como fuente de un operando y destino.

Instrucción de una dirección

- Todas las direcciones son implícitas
- Usan la pila, una estructura en memoria del tipo last-in-first-out (el último en entrar es primero en salir).
- Ejemplo: c = a + b

```
push a
push b
add
pop c
```

Tipos de instrucciones

Las instrucciones de máquina más comunes se clasifican en:

- **Transferencia de datos**: realizan movimientos de datos desde una posición a otra sin cambiar el contenido de la información binaria.
  - Carga, almacenamiento y movimiento de datos entre registros y memoria
  - Manejo de pila.
  - Operaciones de Entrada/Salida.
- **Procesamiento de Datos**: realizan operaciones con los datos.
  - Instrucciones aritméticas
  - Instrucciones lógicas y de manipulación de bits
  - Instrucciones de desplazamiento
- **Control de Programa**: evalúan resultados intermedios en función de ello pueden cambiar el camino tomado por el programa en ejecución.
  - Bifurcación (condicional e incondicional).
  - Llamadas y retorno de subrutinas.

Tipos de operandos

Las instrucciones máquina operan con datos. Las categorías más importantes son:

- **Números**

  Enteros, punto fijo/flotante

- **Caracteres**

  ASCII, BCD, etc.

- **Datos lógicos**

  Bits o flags: una palabra se considera una unidad de *n* elementos o datos de 1 bit, donde cada elemento vale 0 o 1. Permiten almacenar una matriz de datos binarios o booleanos, con lo que la memoria se utiliza más eficientemente, o bien permiten la manipulación de bits individuales de un dato.

- **Direcciones**

  En ocasiones debe realizarse algún cálculo sobre la referencia de un operando. Se consideran como números enteros sin signo.

Direccionamiento de operandos

- Como se vio, una instrucción utiliza un campo de bits para expresar el código de operación: lo que hace. Pero también necesita una importante cantidad de bits para especificar de dónde provienen los datos: **direccionarlos**.
- Los datos que maneja una instrucción máquina pueden estar ubicados en:
  - **En la propia instrucción**: el operando está contenido en un campo de la propia instrucción máquina.
  - **En un registro de la CPU**: Los registros internos de la CPU se pueden utilizar para almacenar temporalmente los datos.
  - **En la memoria del computador**: en cuyo caso será necesario especificar de algún modo la dirección de memoria dónde se halla el operando, denominada *dirección efectiva* (EA, "Effective Address") del operando.

Direccionamiento de operandos

- Dado que, en general, la cantidad de registros es reducida, su referencia puede estar dentro del mismo Código de Operación.
- En cambio, para los casos en que el operando está en la memoria, es muy importante reducir el tamaño de esa referencia.
- Hay dos métodos generales para lograrlo:

  1) Si un operando va a usarse varias veces puede colocarse en un registro.
     - el acceso es más rápido (la búsqueda del operando es una simple transferencia entre registros internos de la CPU; no hay que ir a buscarlo a la memoria externa, que demanda más ciclos de reloj)
     - la instrucción es más corta (la referencia estará en el Código de Operación, lo que no puede hacerse con una posición de memoria)

  2) Referenciar uno o más operandos en forma implícita.

Modos de direccionamiento

- Los **modos de direccionamiento** son distintas formas en que una misma operación puede ejecutarse para reducir el número de bits de la instrucción o hacer un manejo más eficiente de los datos. El programador puede utilizar estas posibilidades que le ofrezca el repertorio de instrucciones.
- Por lo tanto, una misma operación puede dar lugar a varias instrucciones con un modo de direccionamiento diferente
- La Unidad de Control determina el modo de direccionamiento que utiliza cada instrucción mediante:
  - CodOps diferentes, o bien,
  - Uno o más bits en un campo de modo, que indica que tipo de direccionamiento se emplea.

Modos de direccionamiento

- Los modos de direccionamiento utilizan el campo de direcciones de la instrucción para colocar:
  - Directamente el operando (modo inmediato)
  - Directamente la dirección efectiva del operando en la memoria (modos directos)
  - La referencia a un registro que contiene información sobre la dirección efectiva del operando en la memoria (modos de registro)
  - Una referencia a una posición de memoria que a su vez contiene información sobre la dirección efectiva del operando
  - Una combinación de los dos últimos.
- Cada uno de estos modos puede presentar variaciones, y es posible combinar dos o más de ellos para realizar modos más complejos.
- Se verán los modos de direccionamiento más comunes:

Modos de direccionamiento

Ubicación del operando

- Operando en la CPU
  1. Implícito
  2. Inmediato
  3. de Registro
- Operando en la Memoria
  4. Directo a memoria
     - 4.1 de página cero
     - 4.2 de página presente
  5. Indirecto a memoria
  6. Indirecto con registro
     - 6.1 Indirecto con pila
  7. Indirectos con desplazamiento
     - 7.1 Indexado
     - 7.2 de Registro base
     - 7.3 Relativo

Modos de direccionamiento

NOTACION

A = Contenido de un campo de dirección en la instrucción que referencia una palabra en la memoria.

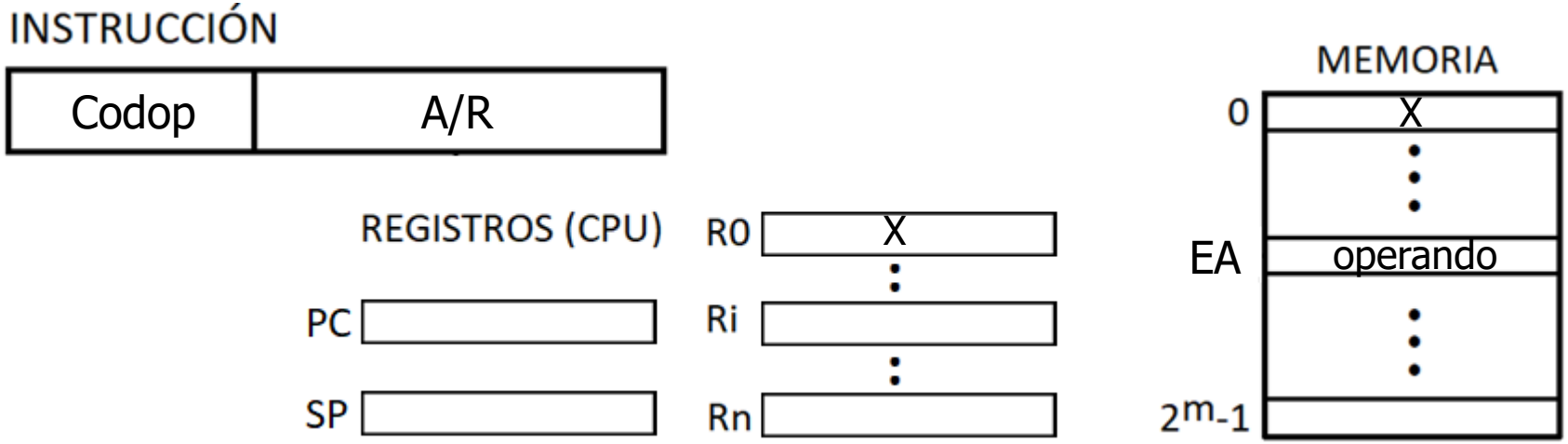
R = Contenido de un campo de dirección en la instrucción que referencia un registro.

[X] = Contenido del registro X o de la posición X de la memoria.

EA = Dirección real (efectiva) de la posición de memoria que contiene el operando que se referencia.

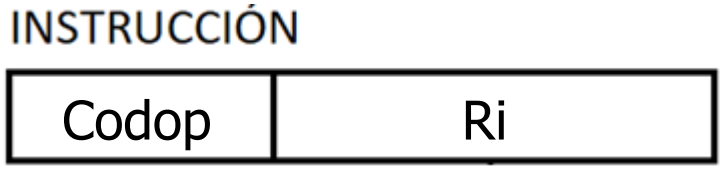
PC = Registro Contador de Programa

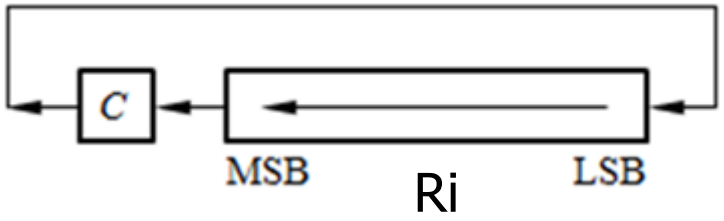
SP = Stack pointer, registro Puntero de la pila en memoria.



1. Direccionamiento implícito

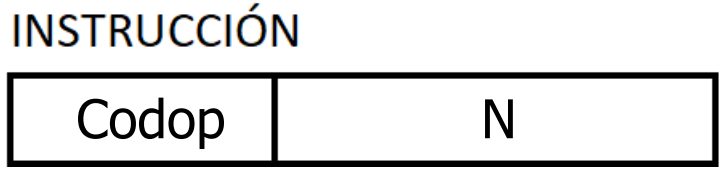
- El operando (contenido en un registro) se especifica implícitamente en la definición de la instrucción (se ejecuta directamente desde el IR)
- Está contenido en un campo de la propia instrucción → no posee campo de direcciones.
- Son instrucciones cortas (1 byte) y de rápida ejecución
- Ej.: RLC Ri (Rotar el registro Ri a la izquierda a través del acarreo)





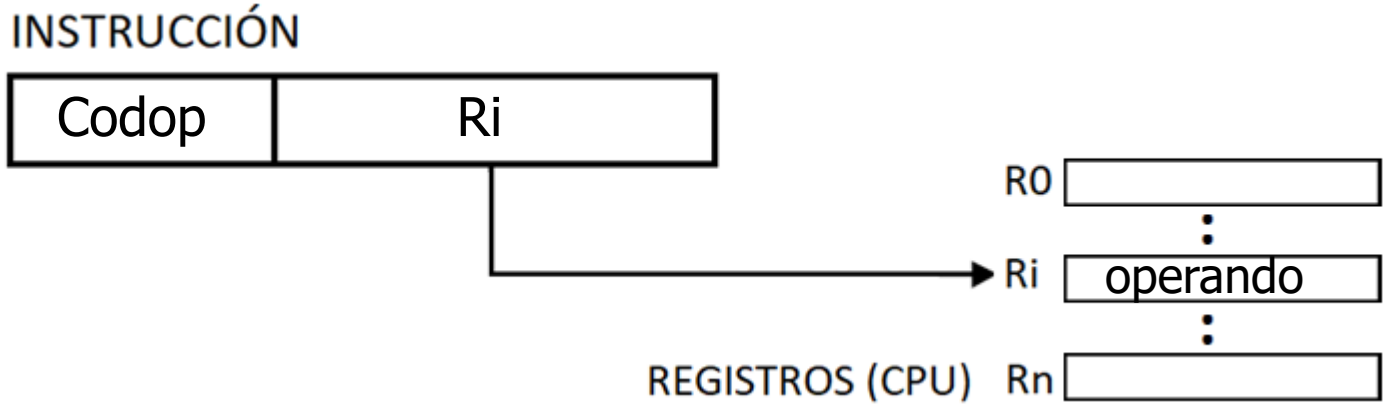
2. Direccionamiento inmediato

- El operando es el valor que se encuentra en el byte inmediato siguiente al código de operación: el campo de dirección contiene el operando (se lo obtiene directamente una vez que la instrucción se carga en el IR)
  - Operando = N (valor)
  - E.j. ADD 5
  - Suma 5 al contenido del acumulador
- Este modo se utiliza para definir constantes, o para fijar valores iniciales de variables.
- Normalmente el número se almacena en complemento a dos; el bit más a la izquierda del campo operando se utiliza como bit de signo.



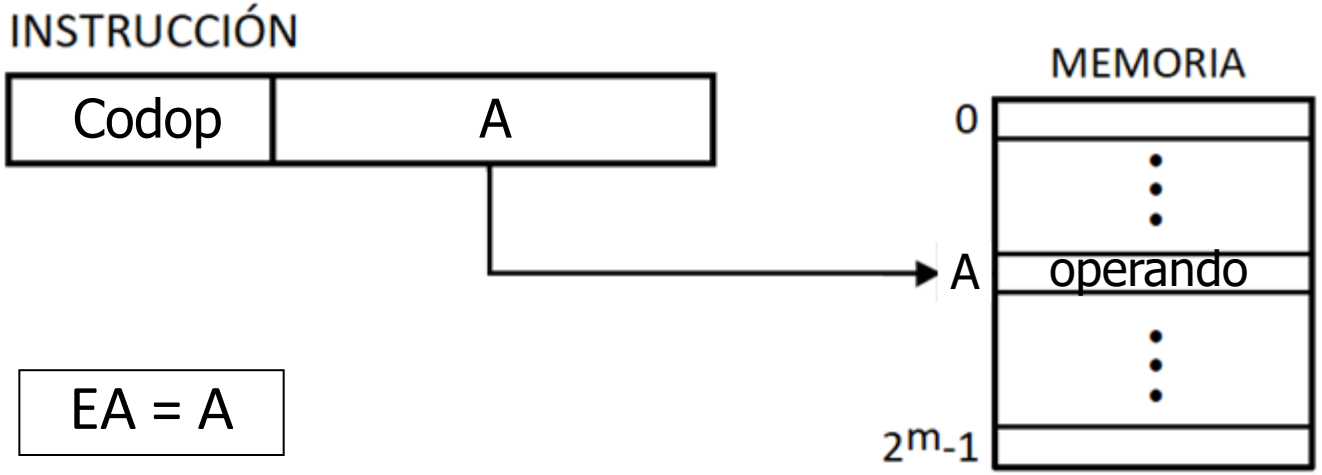
3. Direccionamiento de registro

- El campo de direcciones referencia un registro, en lugar de una dirección de memoria principal.
- El operando está contenido en un registro de la CPU, por lo que no es necesario acceder a la memoria.
- Son instrucciones cortas (1 byte) y de rápida ejecución



4. Direccionamiento directo

- El campo de direcciones *contiene la dirección efectiva del operando*
- EA = A
- P.ej., ADD A
  - Busca en memoria la dirección A para el operando



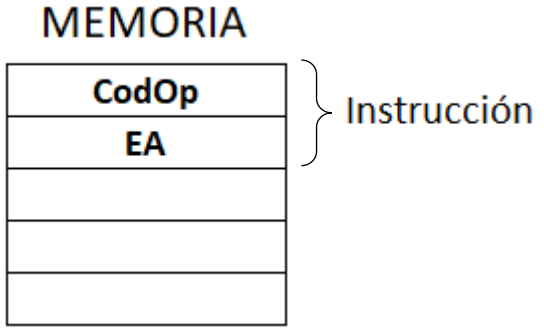
- Pueden darse dos situaciones, dependiendo del tamaño relativo del ancho de la palabra en relación al tamaño de la memoria:

4. Direccionamiento directo

Siendo n = **cantidad de bits** de la palabra

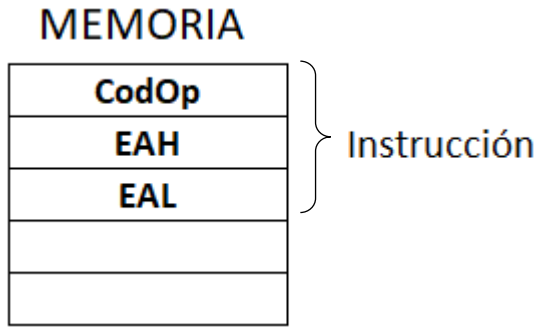
- Si 2^n = tamaño de la memoria →

  → es necesario un solo acceso a la memoria para encontrar la dirección efectiva.



- Si 2^n < tamaño de la memoria →

  → son necesarios al menos dos accesos a la memoria para encontrar la dirección efectiva, por ejemplo EAH y EAL (H=high, alto, bits de mayor peso de la palabra y L=low, bajo, bits de menor peso)

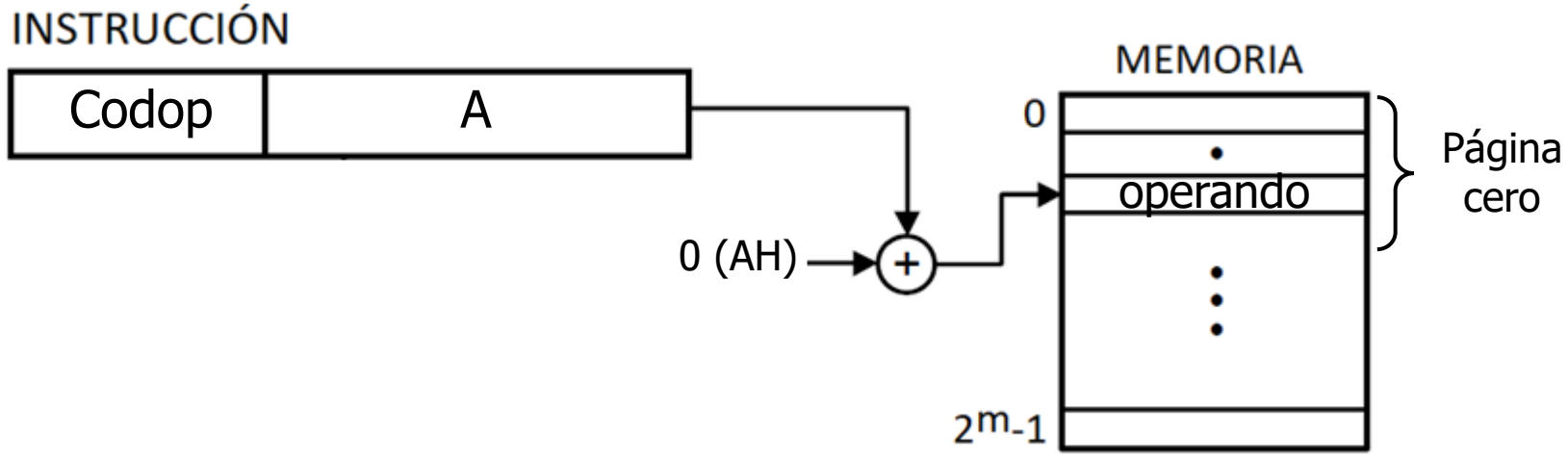


Para este último caso, existen dos opciones que permiten reducir la longitud del campo de direcciones:

- Direccionamiento de página cero
- Direccionamiento de página presente

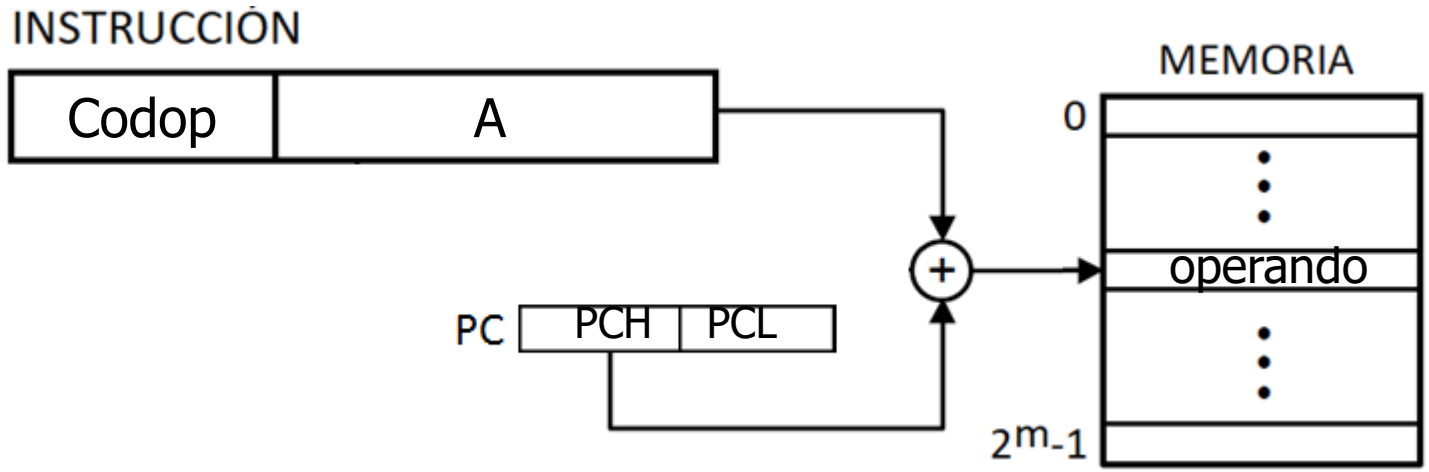
4.1. Direccionamiento de página cero

- El campo de direcciones *contiene los bits de menor peso de la dirección*
- Los bits de mayor peso (*página*) se asumen como ceros
- EA = 0;A
- El operando se encuentra en la parte más baja de la memoria.



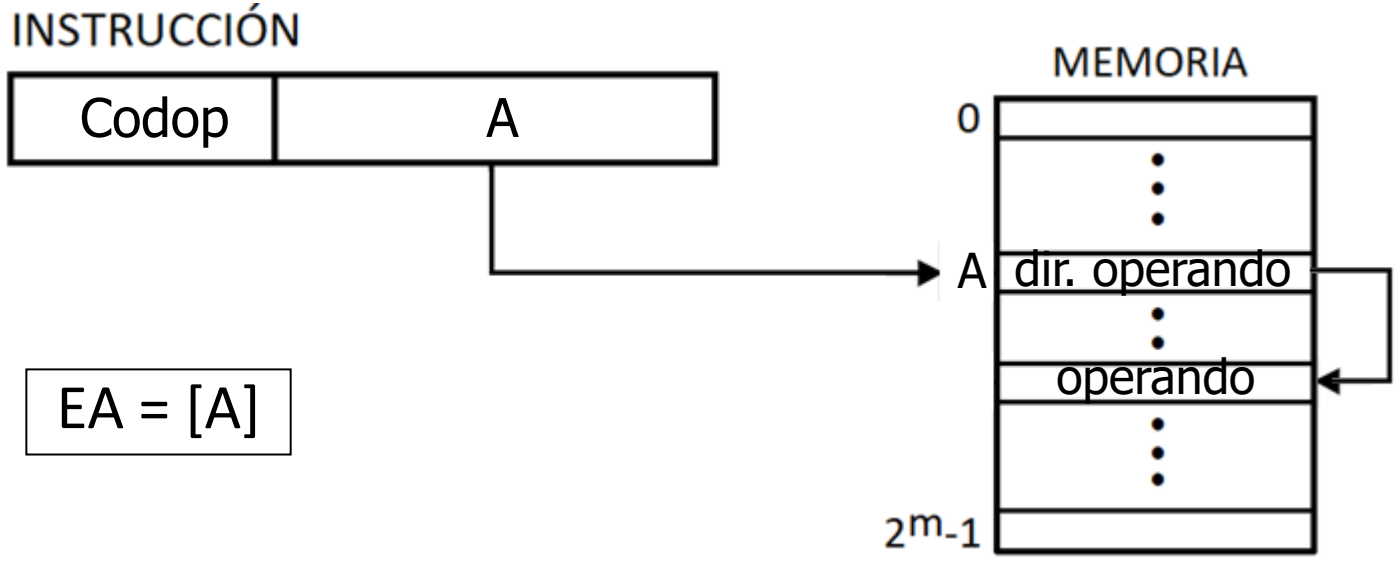
4.2. Direccionamiento de página presente

- El campo de direcciones *contiene los bits de menor peso de la dirección*
- Los bits de mayor peso (*página*) corresponden a la mitad superior (bits de mayor peso) del Contador de Programa (PC)
- EA = PCH;A
- El operando se encuentra en la página de la instrucción en curso.



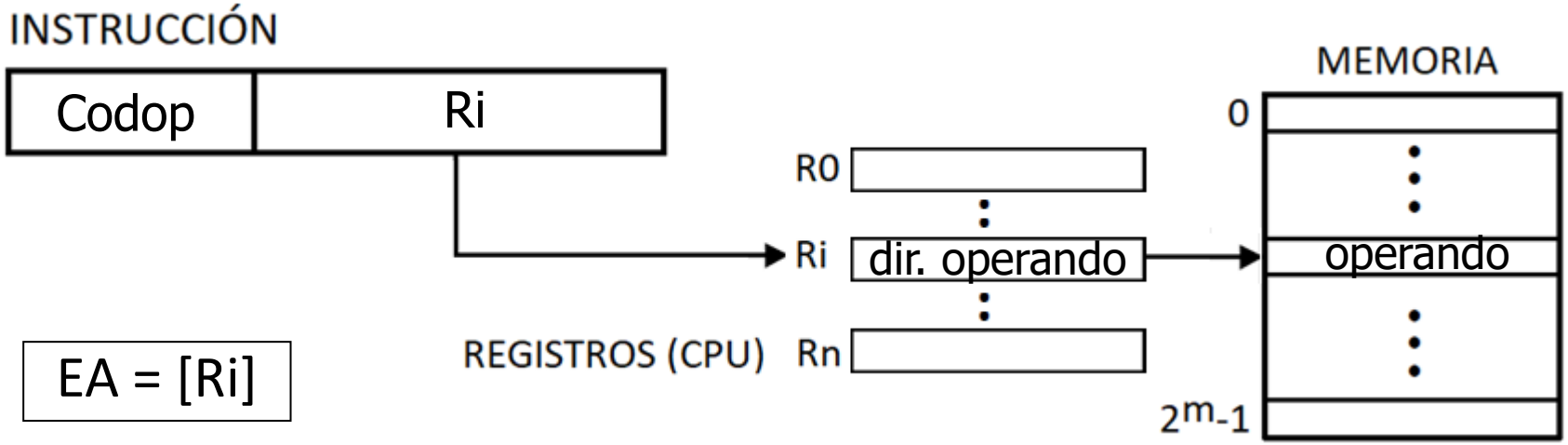
5. Direccionamiento indirecto

- El campo de direcciones contiene la dirección *en la que está almacenada la dirección efectiva del operando*
- EA = [A]
- P.ej., ADD [A]
  - Busca en memoria la dirección A y obtiene *la dirección* del operando



6. Direccionamiento indirecto con registro

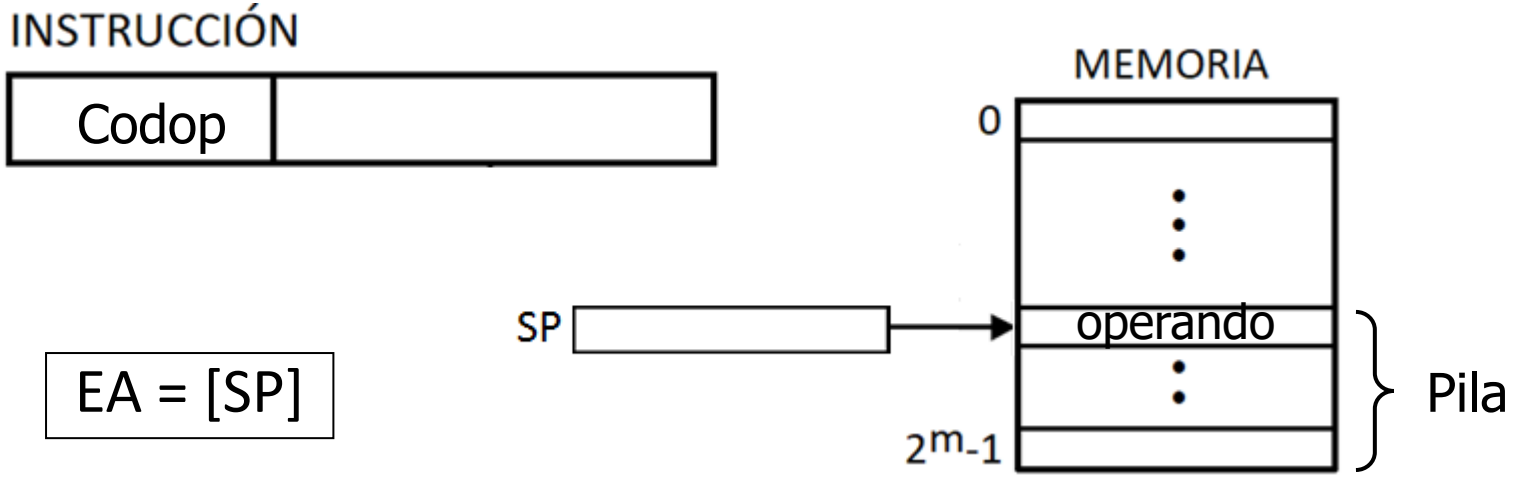
- El campo de direcciones contiene la referencia un registro *en el que está almacenada la dirección efectiva del operando*
- EA = [Ri]
- P.ej., ADD [Ri]
  - Busca el contenido del registro y obtiene *la dirección* del operando



- En los casos en los que el tamaño de los registros sea tal que no puedan referenciar toda la memoria, es común usar un par de registros como un registro doble, por ejemplo, H y L como [HL].

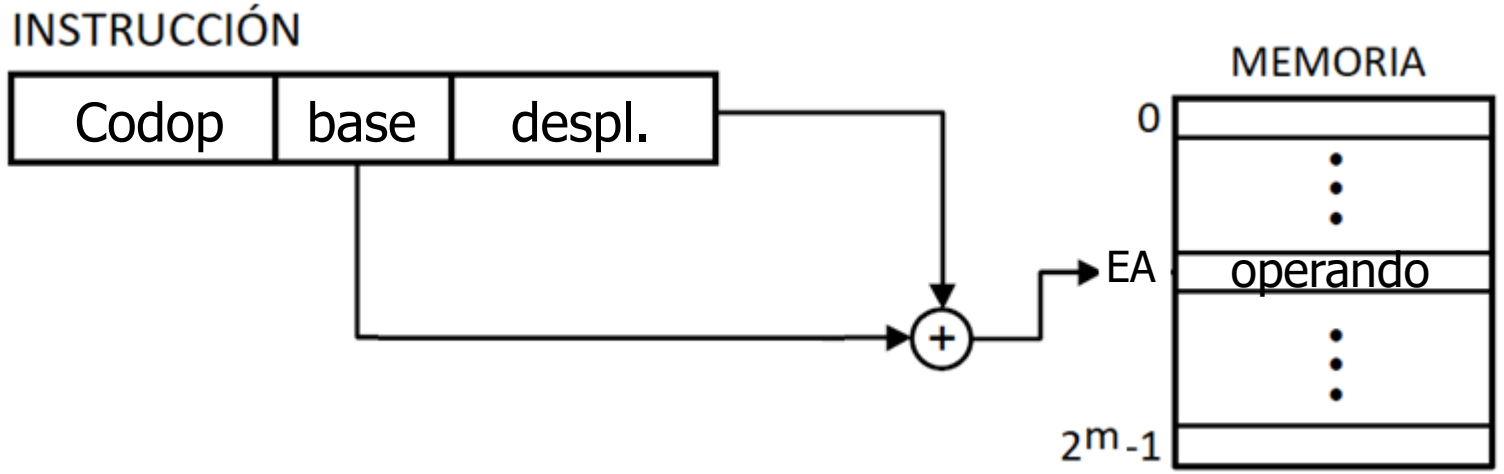
6.1. Direccionamiento indirecto con pila

- El operando está contenido en la memoria, en la cabecera de la pila apuntada por el Stack Pointer (SP)
- EA = [SP]
- Al estar implícito el registro (que a su vez apunta siempre a una dirección), no es necesario un campo de dirección.



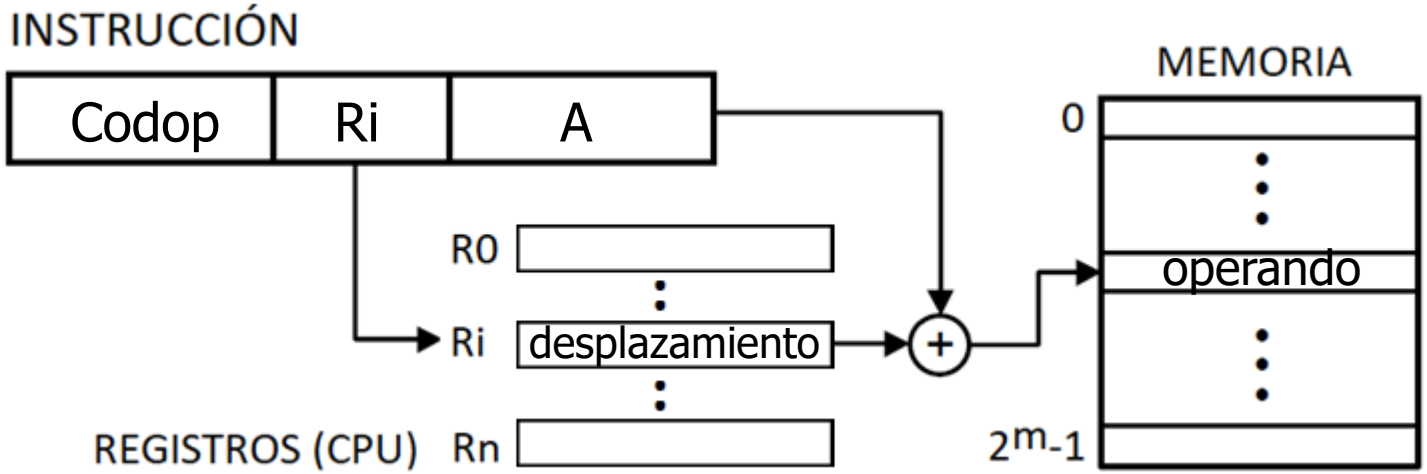
7. Modos de direccionamiento indirecto con desplazamiento

- Es un conjunto de modos de direccionamiento indirecto en los que la dirección efectiva del operando se calcula sumando dos cantidades: *base* y *desplazamiento*.
- El elemento en el que se ubiquen dichas cantidades determina el nombre del modo:
  - Indexado
  - De registro-base
  - Relativo



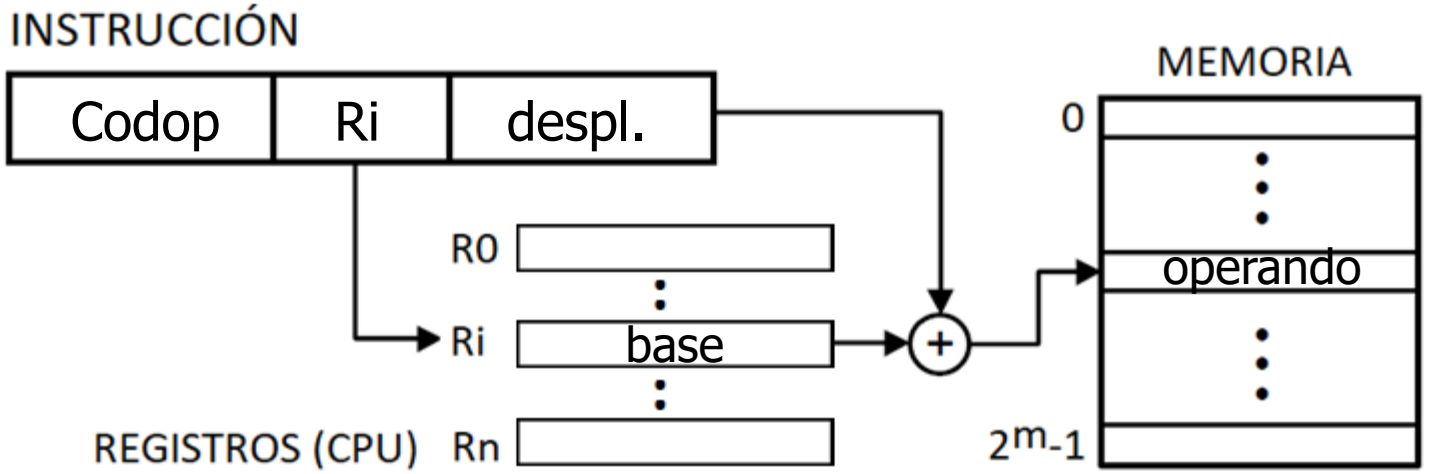
8.1. Direccionamiento indexado

- El contenido del campo de direcciones de la instrucción (A) constituye la base, y a ese valor se le suma el contenido de un registro, que constituye el desplazamiento.
- EA = A + [Ri]
- El registro puede ser un registro particular (X, o IX, *Index Register*), en cuyo caso está implícito en el CodOP, o un registro cualquiera del banco de Registros, en cuyo caso está explícito en el campo de direcciones.



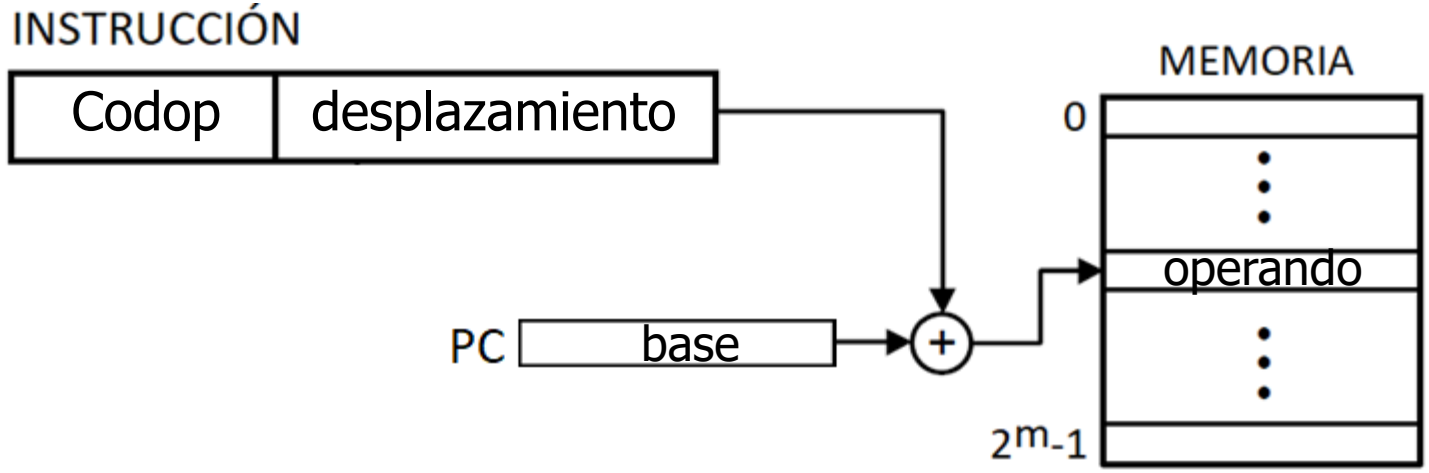
8.2. Direccionamiento de registro-base

- Es idéntico al anterior, solo que base y desplazamiento se invierten: el contenido del registro constituye la base, y a ese valor se le suma el contenido del campo de direcciones de la instrucción (A) que constituye el desplazamiento.
- EA = [Ri] + A
- En ambos casos, el desplazamiento es generalmente un número con signo (en complemento a dos), que permite recorrer una tabla a partir de la dirección base.



8.3. Direccionamiento relativo

- Es un direccionamiento de registro-base que utiliza al registro Contador de Programa (implícito) como base, y le suma el contenido del campo de direcciones como desplazamiento.
- EA = [PC] + A
- Al estar implícito el registro base, la instrucción requiere menos bits.
- Como en los casos anteriores, el desplazamiento es un número con signo.

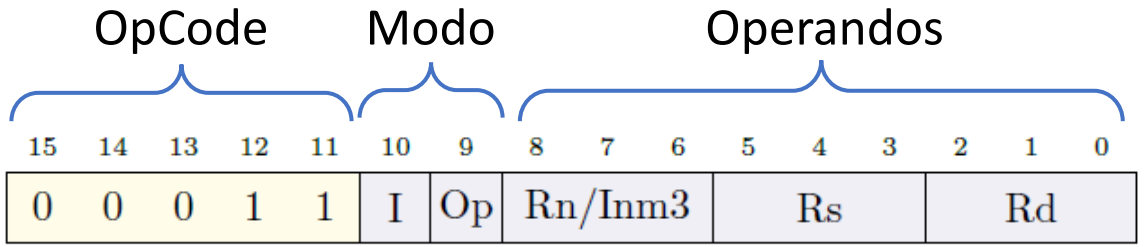


Codificación de instrucciones

La codificación del repertorio de instrucciones es un diseño propio de cada ISA. Dentro de ella, las codificación de los campos que la componen dependerá de:

- La cantidad y tamaño de los registros
- El espacio de memoria direccionable
- El tipo de instrucción
- El modo de direccionamiento

Por ejemplo, para una instrucción de suma en modo de direccionamiento directo a registro de una ISA de ARM



I Inmediato: 1, inmediato; 0, registro.

Op Tipo de operación: 1, resta; 0, suma.

Rn/Inm3 Registro o dato inmediato.

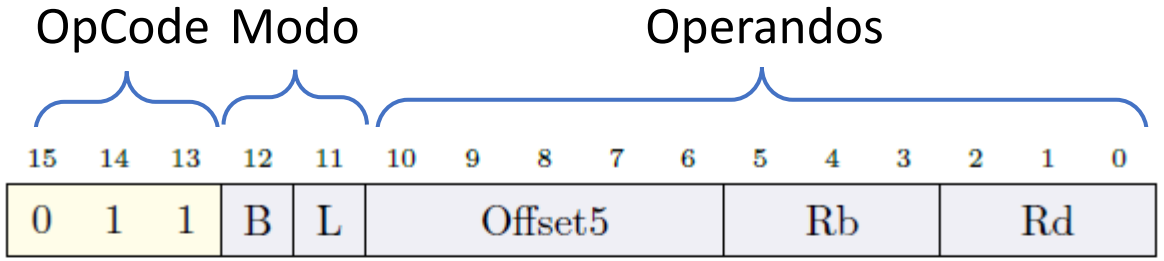
Rs Registro fuente.

Rd Registro destino.

Ensamblador	Máquina	Máquina en binario	Operación
`add r4, r5, r7`	19EC	0001 1001 1110 1100	Guarda en el registro r4 la suma de los contenidos de los registros r5 y r7.

Codificación de instrucciones

Formato de instrucción ARM de carga / almacenamiento de bytes y palabras con direccionamiento indirecto con desplazamiento.



B Byte/Word: 1, transferir byte; 0, palabra.

L Load/Store: 1, cargar; 0, almacenar.

Offset5 Dato inmediato.

Rb Registro base.

Rd Registro fuente/destino.

«**ldr** rd, [rb, #Offset5]», «**str** rd, [rb, #Offset5]», «**ldrb** rd, [rb, #Offset5]» y «**strb** rd, [rb, #Offset5]»

Ensamblador	Máquina	Máquina en binario	Operación
`ldr r5, [r0, #44]`	6AC5	0110 1010 1100 0101	Copia en el registro **r5** el contenido de la dirección de memoria formada sumando 44 al contenido del registro **r0**.

Arquitecturas del repertorio de instrucciones

Existen dos tipos de arquitecturas de conjunto de instrucciones que diferencian marcadamente la relación entre el hardware y el software:

- Procesadores de conjunto de instrucciones complejo (CISC, *Complex Instruction Set Computers*) que proporcionan un soporte hardware para operaciones de lenguajes de alto nivel y programas compactos;
- Procesadores de conjunto reducido de instrucciones (RISC, *Reduced Instruction Set Computers*) que se caracterizan por tener instrucciones sencillas y flexibles que, cuando se combinan, proporcionan un rendimiento más alto y mayor velocidad de ejecución.
- Los objetivos de una arquitectura RISC son conseguir un alto rendimiento y una alta velocidad de ejecución. El objetivo de la arquitectura CISC es ajustarse de forma más cercana a las operaciones empleadas en los lenguajes de programación.

Arquitecturas del repertorio de instrucciones

Estas dos arquitecturas se pueden distinguir considerando las propiedades que caracterizan a sus conjuntos de instrucciones.

RISC

1. Los accesos a memoria se restringen a las instrucciones de carga y almacenamiento.
2. Reducido número de modos de direcciona-mientos.
3. Los formatos de las instrucciones tienen la misma longitud.
4. Las instrucciones realizan ope-raciones básicas.

CISC

1. Los accesos a memoria están directamente dis-ponibles en casi todos los tipo de instrucciones.
2. Mayor número de modos de direccionamien-tos.
3. Los formatos de las instrucciones son de diferente longitud.
4. Las instrucciones realizan tanto operaciones elementales como complejas.

La mayoría de los procesadores actuales utilizan repertorios de instrucciones que resultan una combinación entre RISC y CISC.

Repaso conceptual

1. Enuncie el concepto de lenguaje ensamblador y de: mnemónicos, representación de operandos, etiquetas y directivas.
2. ¿Qué campos se encuentran en un formato general de instrucción?
3. ¿Cómo se puede referenciar la dirección de la próxima instrucción? ¿En qué casos debe ser explícita?
4. ¿Cuáles son las etapas de un ciclo de instrucción detallado?
5. ¿Qué tipos de instrucciones existen?
6. ¿Qué tipos de operandos existen?
7. ¿Qué son los modos de direccionamiento? ¿Cuáles son los principales?
8. Enuncie las principales características de las arquitecturas RISC y CISC.

Lecturas recomendadas

- Mano M., Kime, C. - Fundamentos de diseño lógico y de computadoras - 3º Ed. - Prentice Hall. Año 2000.

  → *Capítulos 10 - 11*

- Stallings, Williams - Organización y Arquitectura de Computadoras - 5º Ed. - Prentice Hall. Año 2000.

  → *Capítulos 10 - 11*