# Arquitectura y Organización de Computadores

Universidad Francisco de Vitoria
UFV Madrid

# TEMA 2
# Arquitectura del Repertorio de Instrucciones

# Objetivos

## Generales

- Conocer las características básicas de un lenguaje ensamblador y su correspondiente lenguaje máquina
- Conocer el repertorio de instrucciones de un procesador real y las metodologías para la programación de bajo nivel con dicho repertorio.

## Específicos

- Enumerar las características de un repertorio de instrucciones
- Calcular la ubicación de un dato para un modo de direccionamiento dado
- Escribir secuencias de instrucciones en lenguaje ensamblador de diferentes máquinas
- Explicar los diferentes campos que aparecen en el formato máquina de una determinada instrucción
- Interpretar el significado y simular la ejecución de programas en ensamblador del Z80
- Enunciar las diferencias entre los enfoques CISC y RISC para el diseño de repertorios de instrucciones
- Clasificar un repertorio de instrucciones como CISC o RISC

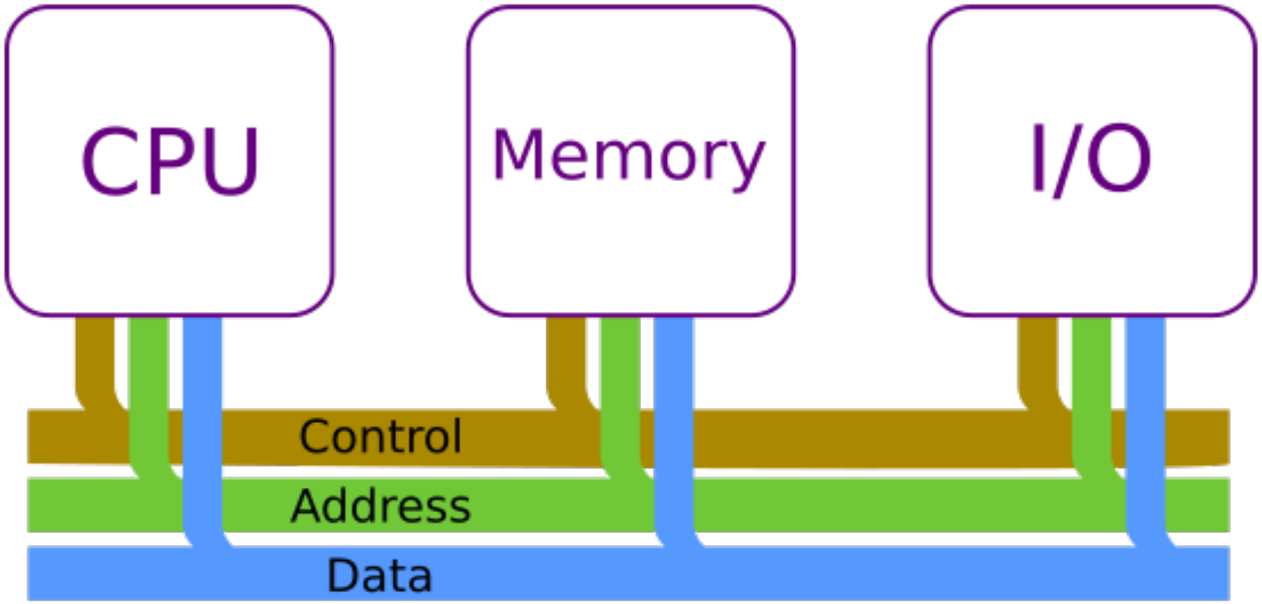
# Contenidos

- Características de las instrucciones máquina
- Cuestiones de diseño en los repertorios de instrucciones
- Tipos de operandos
- Tipos de operaciones
- Modos de direccionamiento
- Número de direcciones
- Formato máquina
- Arquitecturas CISC y RISC

# Bibliografía

- Capítulos 10 y 11 del libro "**Organización y Arquitectura de Computadores**", de W. Stallings
- Capítulo 6 del libro "**Fundamentos de computadores**", Volumen 1, de R. Hermida
- Capítulo 3 del libro " **Arquitectura de computadores, un enfoque cuantitativo**", de Hennessy – Patterson.
- Documentación Z80 del aula virtual

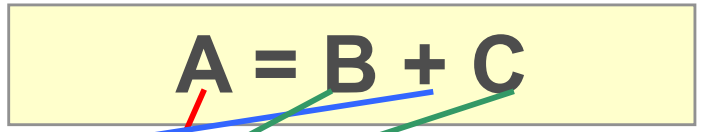
# Arquitectura Von Neumann




Adelanto de arquitectura Von-Neumann

# Características de las instrucciones máquina

- **Instrucción máquina**: instrucción directamente ejecutable por un procesador
- **Repertorio de instrucciones**: conjunto de todas las instrucciones distintas que puede ejecutar el procesador
- **Elementos de una instrucción máquina**:




  - **Código de operación**
  - **Referencia a operandos fuente**
  - **Referencia al operando resultado**
  - **Referencia a la siguiente instrucción**: está implícito para la mayoría de instrucciones, y la siguiente instrucción es la almacenada en la siguiente posición de memoria

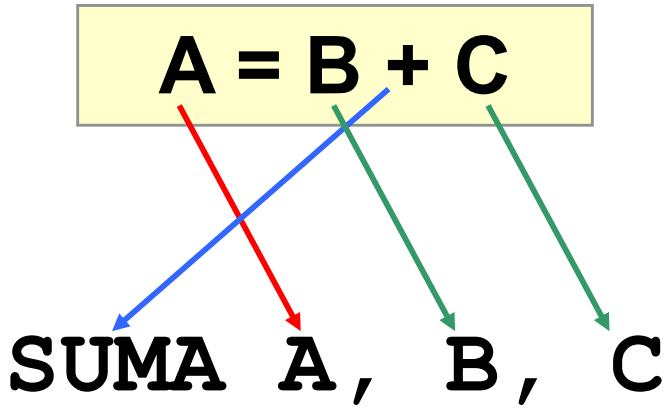
**PC** (**Program Counter**): registro interno del procesador utilizado para gestionar el secuenciamiento implícito de instrucciones. Se inicializa con la dirección de la primera instrucción y se incrementa cada vez que se accede a memoria para leer una nueva instrucción.

# Características de las instrucciones máquina

- **Codificación de las instrucciones:** secuencia de bits, donde ciertos grupos (campos) representan cada uno de los elementos anteriores.
- **Formato de instrucción:** división de la secuencia de bits en campos. En la mayoría de los repertorios de instrucciones hay más de un formato.
- **Representaciones simbólicas de las instrucciones máquinas:** los códigos de operación y los operandos se representan mediante símbolos denominados nemotécnicos
- **Lenguaje máquina:** conjunto de instrucciones máquina en binario
- **Lenguaje ensamblador**: conjunto de instrucciones máquina en representación simbólica

# Características de las instrucciones máquina

- Una instrucción máquina debe indicar, por tanto, qué operación realiza, sobre qué datos la realiza y dónde se almacena el resultado

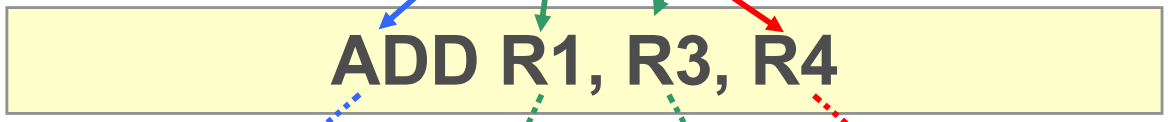



- Para eso existen múltiples alternativas, que varían en
  - la cantidad de memoria requerida
  - en el tiempo de ejecución del programa

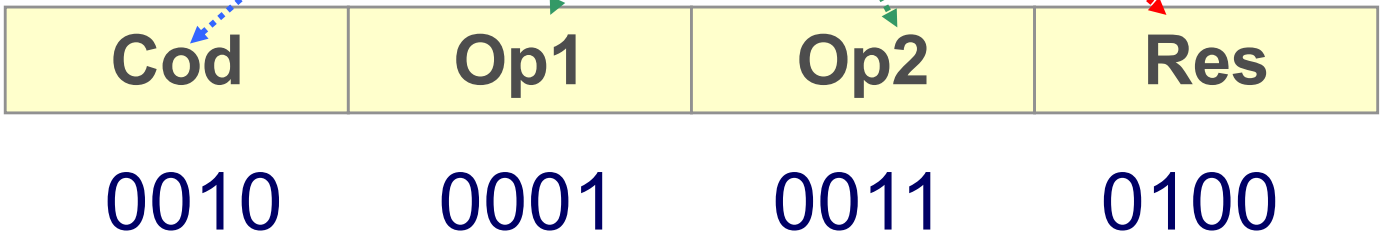
# Características de las instrucciones máquina

- Ejemplo 1:
  - **Instrucción en un lenguaje de alto nivel**


  - **Notación simbólica**

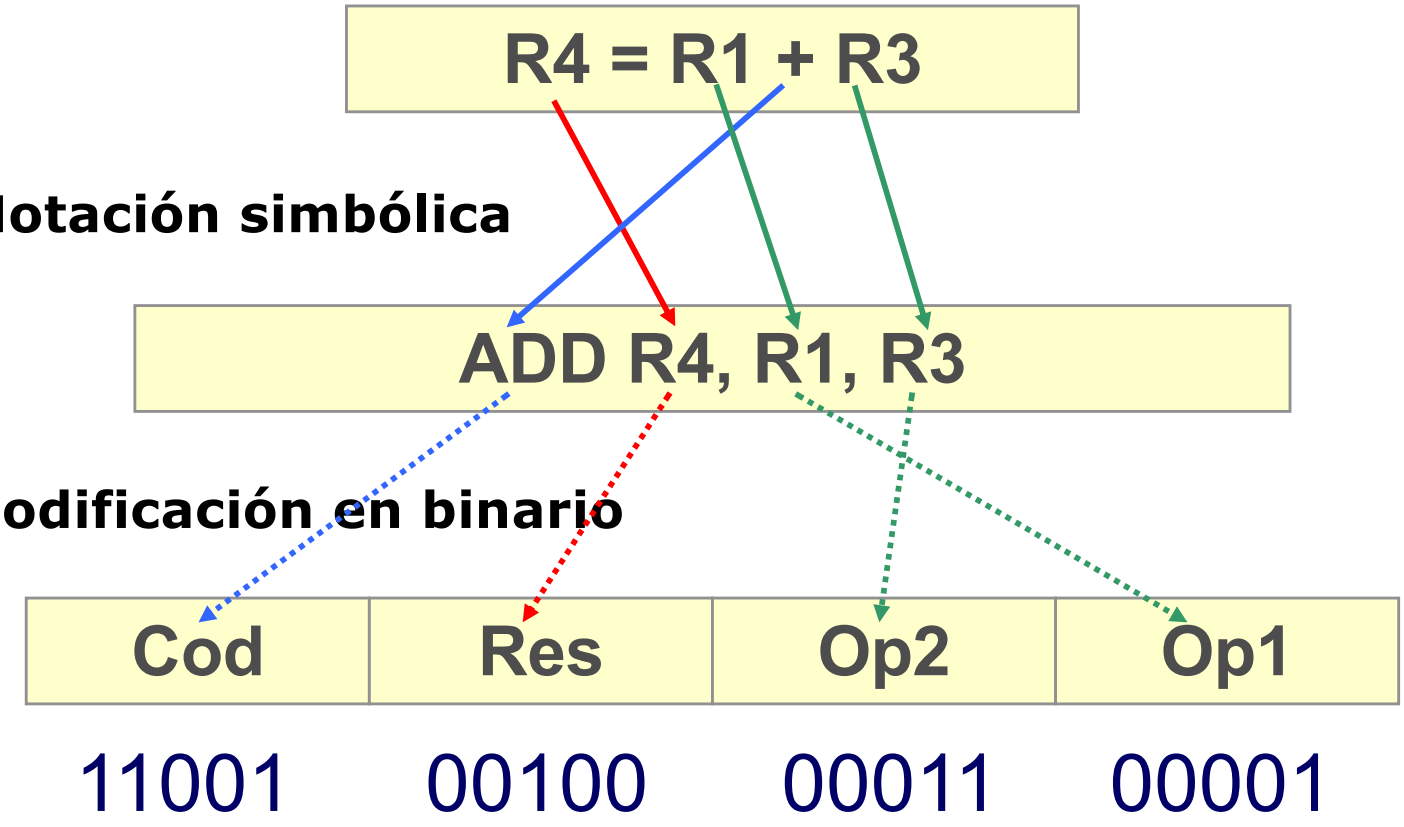



  - **Codificación en binario (Formato máquina)**



# Características de las instrucciones máquina

- Ejemplo 2:
  - **Instrucción en un lenguaje de alto nivel**



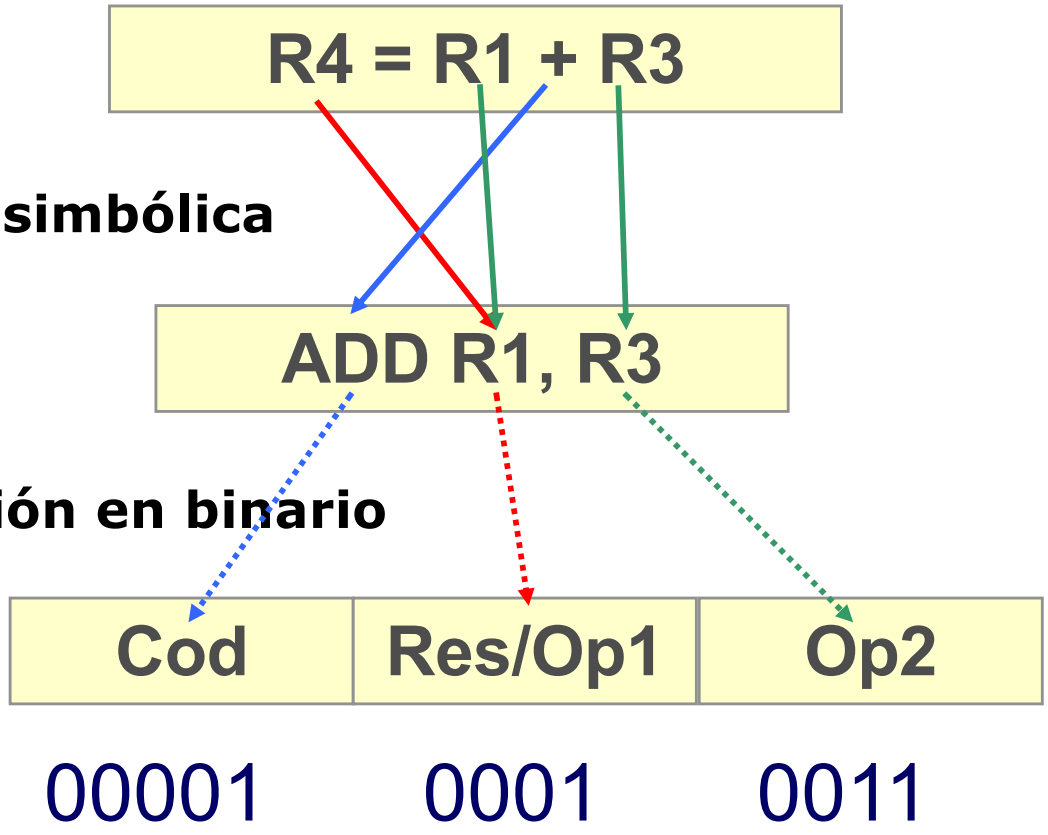

  - **Notación simbólica**




  - **Codificación en binario**

# Características de las instrucciones máquina

- Ejemplo 3:
  - **Instrucción en un lenguaje de alto nivel**
  - **Notación simbólica**
  - **Codificación en binario**

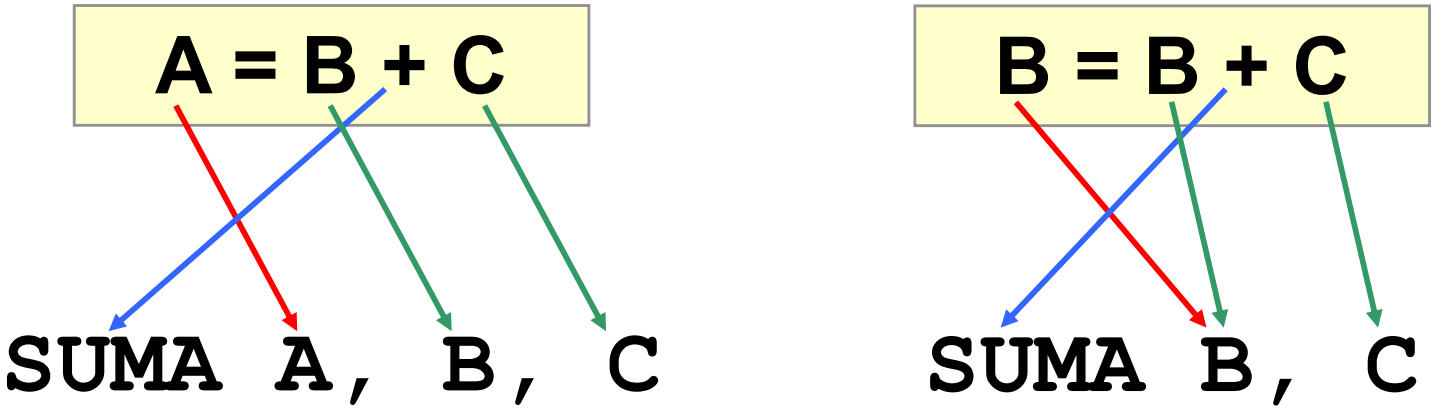


# Cuestiones de diseño de un repertorio de instrucciones

- Existen **seis aspectos** básicos (decisiones que se toman a la hora de diseñar un lenguaje máquina) que diferencian a unos repertorios de instrucciones de otros:
  1. Número de direcciones
  2. Ubicación de los datos
  3. Modos de direccionamiento
  4. Número y tipos de instrucciones
  5. Tamaño y tipos de datos
  6. Formato máquina

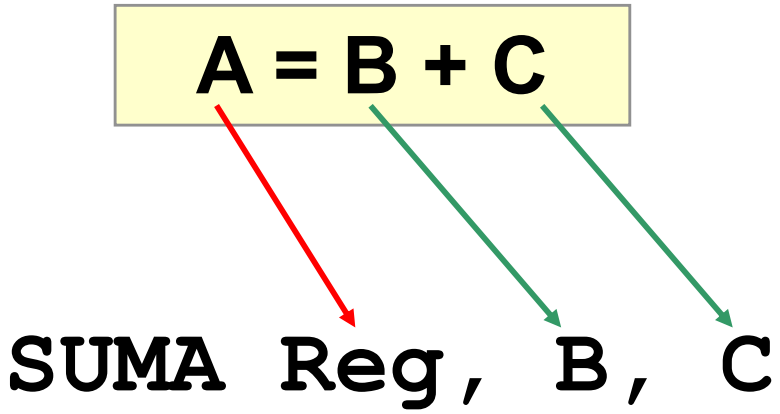
# Cuestiones de diseño de un repertorio de instrucciones

- Cuestión de Diseño 1:
  - **Número de direcciones**: número de referencias explícitas a operandos en las operaciones aritméticas o lógicas
  - Pueden ser de 0, 1, 2 ó 3 direcciones
  - Ejemplo: 3 y 2 direcciones



# Cuestiones de diseño de un repertorio de instrucciones

- Cuestión de Diseño 2:
  - **Ubicación de operandos**: los datos sobre los que se opera pueden estar en memoria, pero también en registros internos del procesador.
  - La ubicación del dato se especifica mediante un **modo de direccionamiento**.



# Cuestiones de diseño de un repertorio de instrucciones

- Cuestión de Diseño 3:
  - La ubicación del dato se especifica mediante un **modo de direccionamiento**.
  - Para el acceso a datos en memoria existen múltiples modos de direccionamiento, que difieren en el tamaño que se necesitan para codificarse en la instrucción y el tiempo que tardan en obtener el dato.

- Cuestión de Diseño 4:
  - **Número y tipo de instrucciones**:
    - Instrucciones artiméticas: ADD, SUB, MULT, DIV, ...
    - Instrucciones lógicas: AND, OR, COMP, ...
    - Instrucciones de transferencia de datos: LOAD, STORE, MOVE,
    - Instrucciones de control de flujo: JUMP, BEQ, BNE, ...
    - Instrucciones de E/S
    - ...

# Cuestiones de diseño de un repertorio de instrucciones

- Cuestión de Diseño 5:
  - **Tipos y tamaños de los datos**:
    - Enteros de 8, 16, 32, o 64 bits
    - Reales en coma flotante de 32, 64 ó 128 bits
    - Caracteres....

- Cuestión de Diseño 6:
  - Las instrucciones del repertorio deben codificarse en binario en, lo que se denomina, **el formato máquina**, que puede ser
    - De tamaño fijo
    - De tamaño variable

# Número de direcciones (Cuestión de Diseño 1)

- **Instrucciones de 3 direcciones**
  - La instrucción especifica de manera explícita la ubicación de los 2 operandos y del resultado, que pueden por tanto estar en **tres sitios diferentes**.
  - Ejemplos:

| INSTRUCCIÓN | SIGNIFICADO |
|---|---|
| `ADD R, A, B` | `R= A + B` |
| `SUB R, A, B` | `R= A - B` |
| `MUL R, A, B` | `R= A * B` |
| `DIV R, A, B` | `R= A / B` |

# Número de direcciones (Cuestión de Diseño 1)

- **Instrucciones de 2 direcciones**:
  - La instrucción especifica dos operandos de manera explícita, uno de ellos, además, actúa al mismo tiempo como resultado.
  - Intrucciones más cortas, pero programas menos compactos, dado que en cada operación uno de los operandos se pierde y puede ser necesario salvarlo previamente.
  - Ejemplos:

| INSTRUCCIÓN | SIGNIFICADO |
|---|---|
| `ADD A, B` | `A= A + B` |
| `SUB A, B` | `A= A - B` |
| `MUL A, B` | `A= A * B` |
| `DIV A, B` | `A= A / B` |

# Número de direcciones (Cuestión de Diseño 1)

- **Instrucciones de 1 dirección**:
  - Utiliza un registro especial llamado acumulador (AC).
  - La instrucción especifica de manera explícita la ubicación de un operando; el otro operando y el resultado se sitúan en el acumulador.
  - Intrucciones más cortas pero programas menos compactos. Dado que el acumulador puede almacenar un único dato, esta arquitectura obliga a muchos movimientos de datos para realizar las operaciones.
  - Ejemplos:

| INSTRUCCIÓN | SIGNIFICADO |
|---|---|
| ADD B | AC= AC + B |
| SUB B | AC= AC - B |
| XOR B | AC= AC XOR B (xor bit a bit) |
| AND B | AC= AC AND B (and bit a bit) |

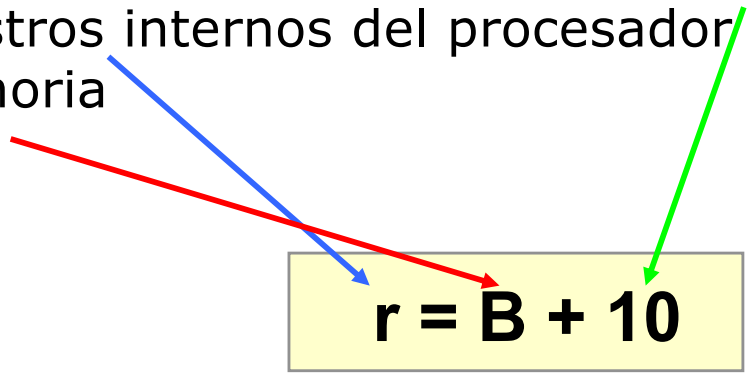
# Número de direcciones (Cuestión de Diseño 1)

- **Instrucciones de 0 direcciones:**
  - Utiliza una PILA para almacenar temporalmente los datos.
  - Los 3 operandos están implícitos en la PILA, en el orden en que han de ser utilizados para la operación.
  - Es poco flexible, y no permite reutilizar operandos, sólo cálculos intermedios.
  - Ejemplos:

| INSTRUCCIÓN | SIGNIFICADO |
|---|---|
| ADD | TOS = TOS + (TOS-1) |
| SUB | TOS = TOS – (TOS-1) |
| MUL | TOS = TOS * (TOS-1) |
| DIV | TOS = TOS / (TOS-1) |

# Ubicación de los datos (Cuestión de diseño 2)

- Los datos pueden encontrarse en tres sitios diferentes
  - En la propia instrucción (datos constantes)
  - En registros internos del procesador
  - En memoria



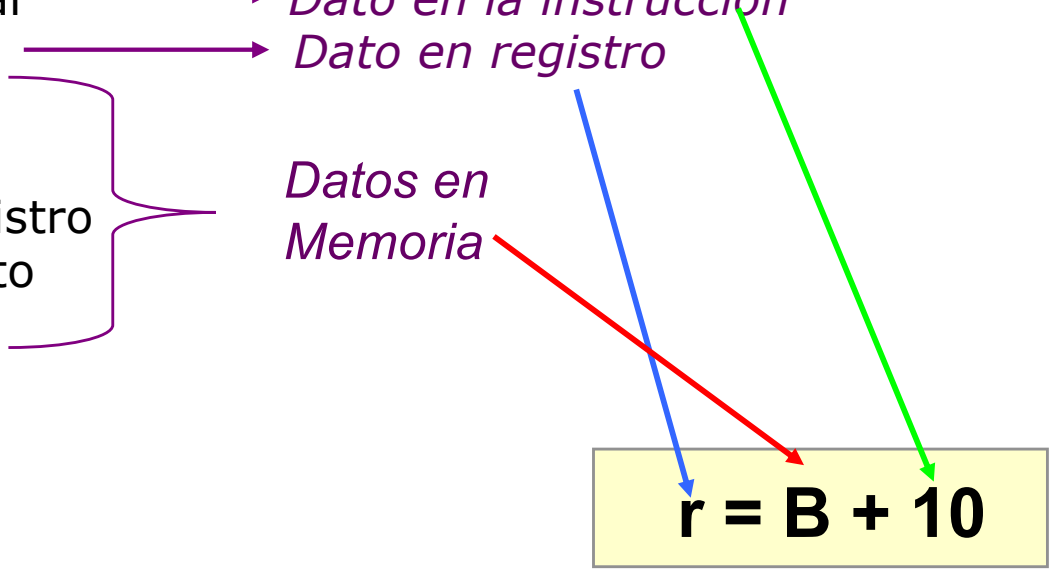

- Todos los lenguajes manejan datos en cualquiera de estas tres ubicaciones. La diferencia de unos lenguajes a otros estriba en las ubicaciones posibles para los datos sobre los que se realizan operaciones aritméticas:
  - Algunos lenguajes sólo permiten operar con datos que están almacenados en los registros internos del procesador (deben traerse previamente de memoria)
  - Otros lenguajes permiten operar con datos que se encuentran almacenados en memoria (se traen de memoria justo cuando se necesitan para la operación).

# Modos de direccionamiento (Cuestión de diseño 3)

- **Modo de direccionamiento**: modo en el que se indica la ***dirección efectiva*** de un operando, es decir, su ubicación

- **Modos**:
  - Inmediato o literal → *Dato en la instrucción*
  - Registro → *Dato en registro*
  - Directo
  - Indirecto
  - Indirecto con registro
  - De desplazamiento
  - Etc.

  (Directo, Indirecto, Indirecto con registro, De desplazamiento, Etc. → *Datos en Memoria*)



# Modos de direccionamiento (Cuestión de diseño 3)

- **Inmediato**:
  - el dato aparece en la propia instrucción y, por tanto, es constante
  - no requiere acceso a memoria adicional para obtener el dato
  - el rango de datos representables depende de la longitud del campo correspondiente en la instrucción, y del tipo de dato.
  - **Cuestión de diseño: ¿Cuántos bits destinar a este campo?**
    - Rango de números enteros con n bits: $[-2^{n-1}, 2^{n-1}-1]$
    - Rango de números sin signo con n bits: $[0, 2^{n}-1]$
  - Ejemplos:

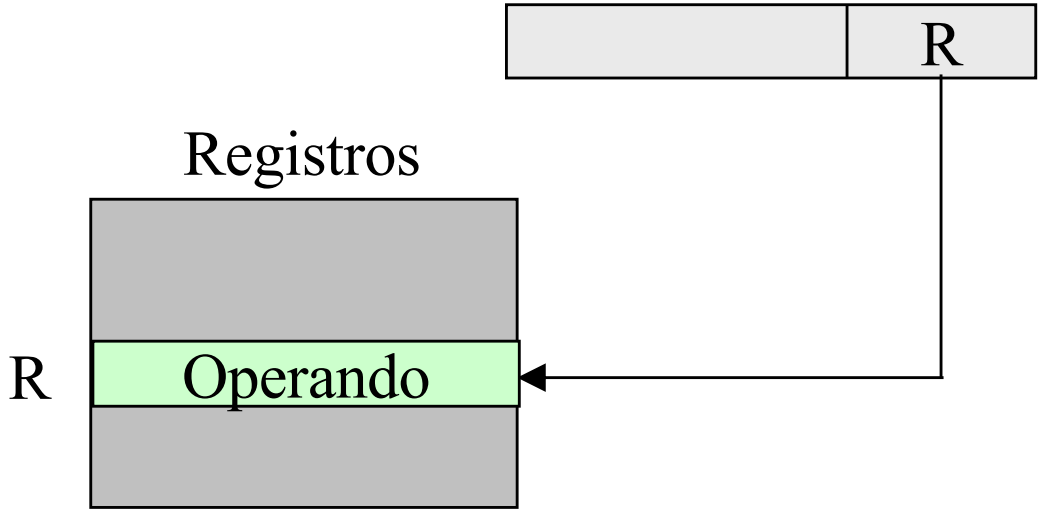
    LD A, 0xF3

    JP 0x80BF

    AND 0x07

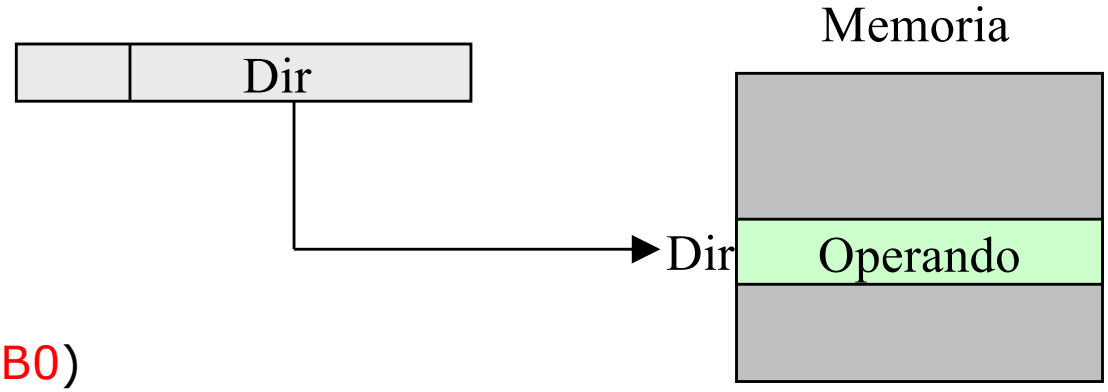
# Modos de direccionamiento (Cuestión de diseño 3)

- **Registro**:
  - el dato se encuentra almacenado en un registro
  - la instrucción incluye el número de registro correspondiente (normalmente se necesitan pocos bits para codificarlo)
  - no requiere acceso a memoria para obtener el dato
  - acceso muy rápido al dato
  - Ejemplos:
    LD A, C
    JP (HL)
    AND B



# Modos de direccionamiento (Cuestión de diseño 3)

- **Directo** (o absoluto):
  - sólo requiere un acceso a memoria
  - la instrucción contiene la dirección de la posición de memoria donde se encuentra el dato
  - rango de memoria alcanzable: $2^k$, $k$ es la longitud en bits del campo Dir

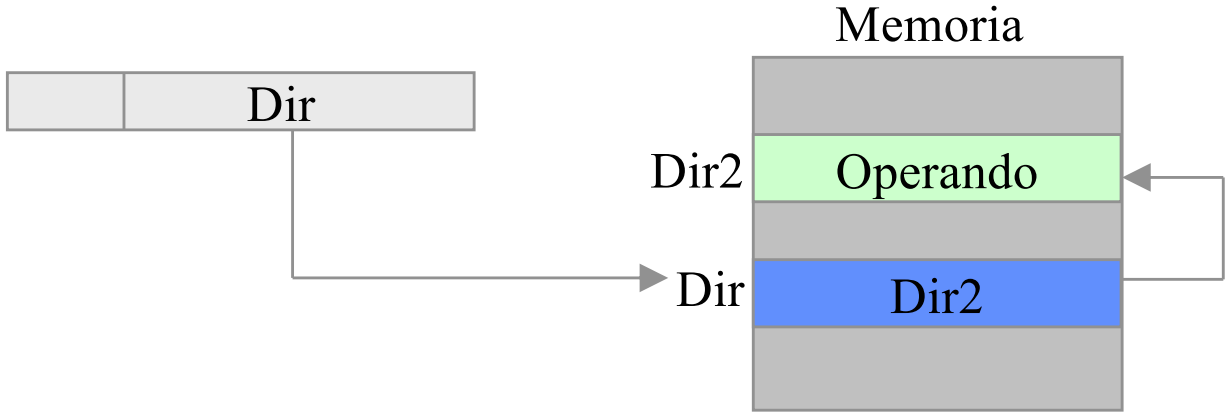



  - Ejemplo:

    LD A, (0x41B0)

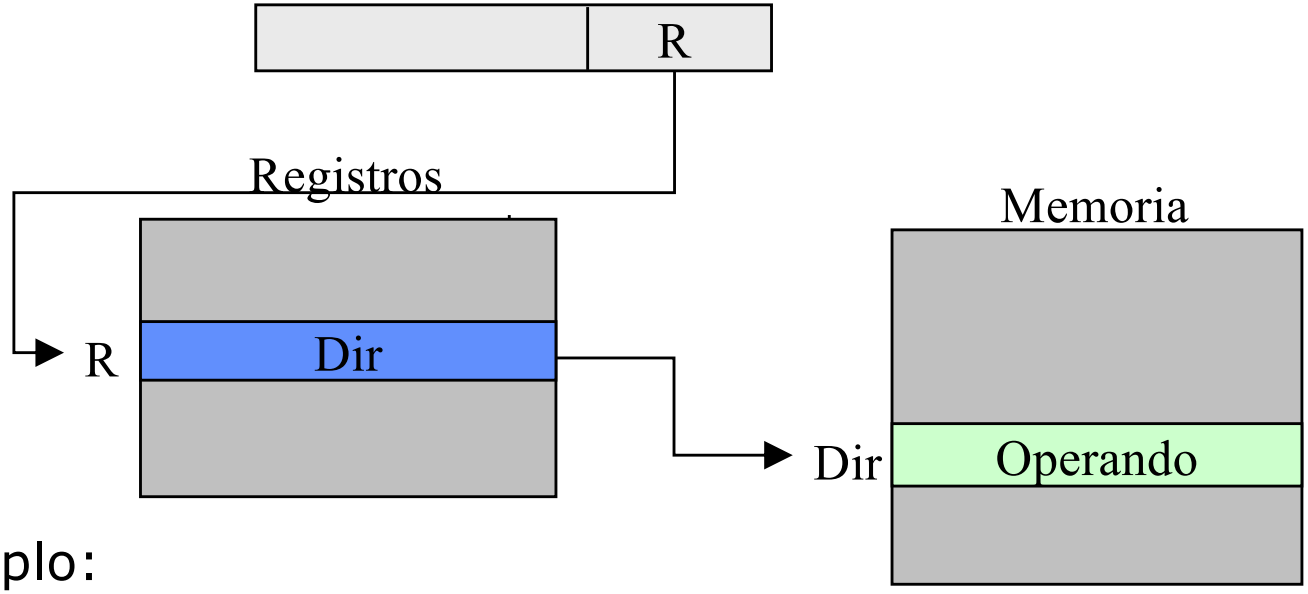
# Modos de direccionamiento (Cuestión de diseño 3)

- **Indirecto**:
  - requiere dos accesos a memoria
  - la instrucción contiene la dirección de la posición de memoria donde se encuentra la dirección del dato
  - rango de memoria donde pueden situarse los punteros: $2^k$, $k$ es la longitud en bits del campo Dir



# Modos de direccionamiento (Cuestión de diseño 3)

- **Indirecto con registro**:
  - requiere un acceso a registro y otro a memoria
  - la instrucción contiene el número de registro donde se encuentra la dirección del dato
  - rango de memoria alcanzable: $2^N$, $N$ es la anchura en bits del registro

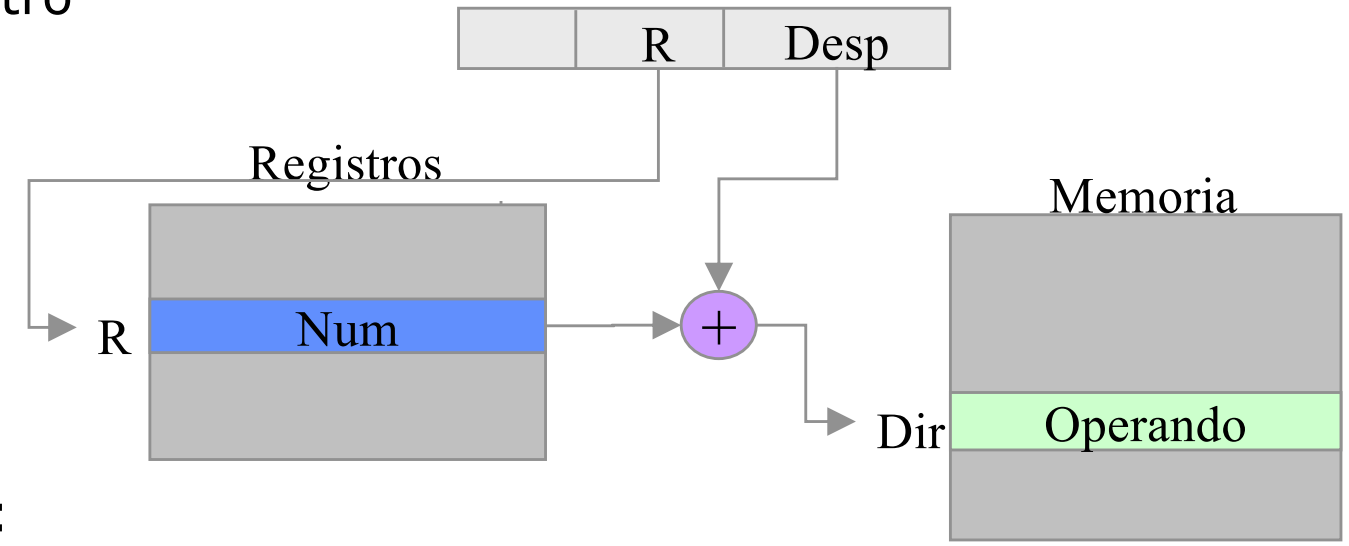



  - Ejemplo:

    LD A, (BC)

# Modos de direccionamiento (Cuestión de diseño 3)

- **De desplazamiento**:
  - la instrucción contiene dos campos: un número de registro y una constante llamada desplazamiento. La dirección de memoria del dato se calcula como la suma de dicha constante más el contenido del registro.
  - requiere un acceso a registro y otro a memoria
  - rango de memoria alcanzable: $2^N$, $N$ es la anchura en bits del registro



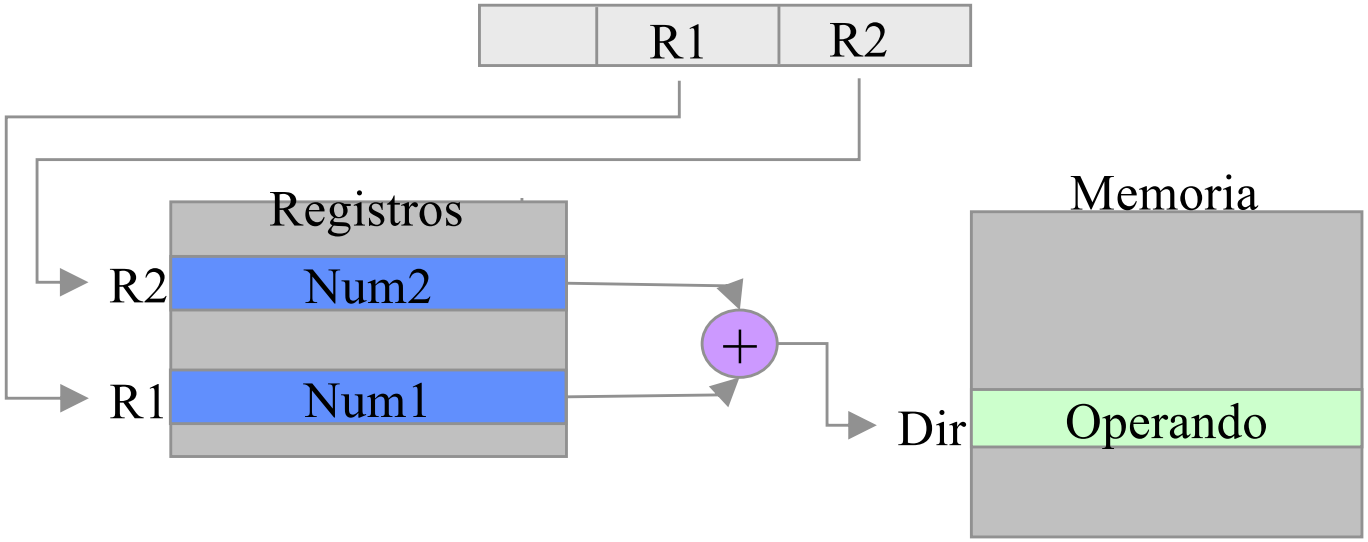

  - Ejemplo:

    LD A, **(IX+34)**

# Modos de direccionamiento (Cuestión de diseño 3)

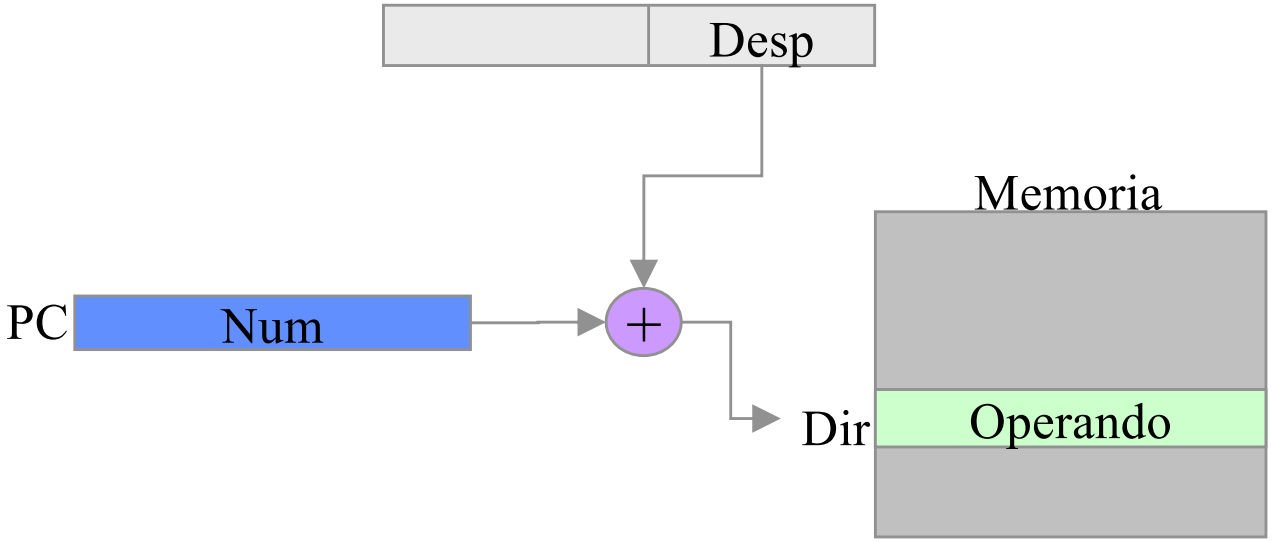
- **Indexado**:
  - la instrucción contiene dos campos registro. La dirección de memoria del dato se calcula como la suma del contenido de ambos registros.
  - requiere dos accesos a registros
  - Ejemplo:

    Add R3, **(R1+R2)**



# Modos de direccionamiento (Cuestión de diseño 3)

- **Relativo a PC**:
  - la instrucción contiene un campo con una constante llamada desplazamiento. La dirección de memoria del dato se calcula como la suma de dicha constante más el contenido del PC (contador de programa).
  - requiere un acceso a un registro específico (PC)
  - Es importante para hacer código reubicable




  - Ejemplos:

    JR +45

    DJNZ -10

# Tipo de instrucciones (Cuestión de diseño 4)

Tipos comunes de instrucciones en la mayoría de lenguajes

- **Transferencia de datos**: modifican la ubicación de un dato, es decir, mueven datos de un lugar de almacenamiento a otro
- **Aritméticas**: suma, resta, multiplicación y división para números naturales, enteros en punto fijo, reales en coma flotante, y decimales (en BCD)
- **Lógicas**: AND, OR, NOT, XOR, operaciones de desplazamiento y rotación de bits
- **De E/S**: para transferencia de datos entre la CPU y los dispositivos periféricos
- **De control de flujo**: permiten alterar el secuenciamiento implícito
  - Salto condicional
  - Salto incondicional
  - Salto a subrutina

# Tipo de instrucciones (Cuestión de diseño 4)

- Ejemplo: Algunos códigos nemotécnicos comunes

  - **Transferencia de datos**

    MOVE STORE LOAD

    CLEAR SET

  - **Aritméticas**

    ADD SUB MULT DIV

  - **Lógicas**

    AND OR NOT XOR

    CMP SHIFT ROTATE

  - **Control de flujo**

    JUMP BEQ BNQ BGT

    BGE BLT BLE

# Tipos de datos (Cuestión de diseño 5)

Las instrucciones máquina suelen manejar

- **Direcciones de memoria**: números enteros sin signo
- **Datos**
  - **Numéricos**:
    - Enteros de varios tamaños: 8 bits, 16 bits, 32 bits, …
      - Sin signo (*unsigned*): números naturales en binario
      - Con signo (*signed*): números enteros en complemento a dos
    - Reales en coma flotante (formato IEEE 754), de 32 bits, 64 bits, …
    - Decimales en BCD
  - **Caracteres: letras, números y otros símbolos representados mediante un código** (ASCII, Unicode, …)
- **Datos lógicos**
  - Cualquier tipo de dato de los anteriores manipulado bit a bit

# Formato máquina
# (Cuestión de diseño 6)

Alternativas en la codificación (formato máquina)

- Instrucciones de **tamaño variable**:
  - Se utilizan cuando existen muchos modos de direccionamiento y muchas instrucciones diferentes

| Op + nº dir | Modo dir. 1 | Operando 1 | ••• | Modo dir. 1 | Operando 1 |
|---|---|---|---|---|---|

- Instrucciones de **tamaño fijo**
  - Se utilizan cuando existen pocas instrucciones y modos de direccionamiento

| Op + nº dir + modo | Operando 1 | Operando 2 | Operando 3 |
|---|---|---|---|

# Arquitecturas actuales

- Desde los años 80 los procesadores de propósito general (utilizados en PCs, servidores, etc.) sólo implementan arquitecturas de 2 y 3 direcciones, llamadas Arquitecturas con **Registros de Propósito General** (GPR)

- Los primeros computadores (EDSAC, UNIVAC I, etc) tenían arquitecturas de tipo **acumulador**, es decir, de 1 dirección. Hoy en día se utilizan arquitecturas de acumulador en los microcontroladores utilizados en sistemas empotrados.

- En los años 60 y 70 existieron varios computadores con arquitectura de **pila** (0 direcciones). La mayoría de procesadores incorporan una pila e instrucciones (de 0 direcciones) para acceder a ella.

# Arquitecturas actuales

- **3 direcciones**
  - Operan con datos en registros exclusivamente
  - Ejemplos: RISC-V, ARM7, PowerPC de IBM-Apple-Motorola, 88000 de Motorola, etc.
- **2 direcciones**
  - Operan con datos en registros y con datos en memoria
  - En cada operación, se permite un acceso máximo a memoria
  - Ejemplo: Z80, de Zilog, x86 de Intel, 68000 de Motorola, etc.

| Number of memory addresses | Maximum number of operands allowed | Type of architecture | Examples |
|---|---|---|---|
| 0 | 3 | Register-register | Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, Trimedia TM5200 |
| 1 | 2 | Register-memory | IBM 360/370, Intel 80x86, Motorola 68000, TI TMS320C54x |
| 2 | 2 | Memory-memory | VAX (also has three-operand formats) |
| 3 | 3 | Memory-memory | VAX (also has two-operand formats) |

# Arquitecturas CISC vs. RISC

- Existen dos filosofías opuestas para la definición de las arquitecturas de repertorios de instrucciones (ISA) de los procesadores: **CISC** (Complex Instruction Set Computer) y **RISC** (Reduce Instruction Set Computer).

- Las arquitecturas CISC proporcionan un lenguaje máquina muy extenso y rico, en tipos de instrucciones diferentes y en modos de direccionamiento para acceso a datos.

- Las arquitecturas RISC, en cambio, proporcionan un lenguaje máquina mucho más simple. Aparecieron mucho después que las CISC (años 80), con el objetivo de simplificar las microarquitecturas de los procesadores e incrementar así su rendimiento.

# Arquitecturas CISC vs. RISC

## Tecnología CISC

- Instrucciones de formato variable
- Muchas instrucciones y modos de direccionamiento
- Cualquier instrucción puede referenciar a la memoria
- Número reducido de registros
- La complejidad reside en el microprograma

IBM 360, IA32, IA64, M68k, Z80

## Tecnología RISC

- Instrucciones de formato fijo
- Pocas instrucciones y modos de direccionamiento
- Sólo carga/almacenamiento pueden referenciar a memoria
- Varios conjuntos de registros
- La complejidad reside en el compilador

PA-RISC, PowerPC, Sparc, M88k, MIPS, ARMv7, RISC-V

¿Qué significa esto?

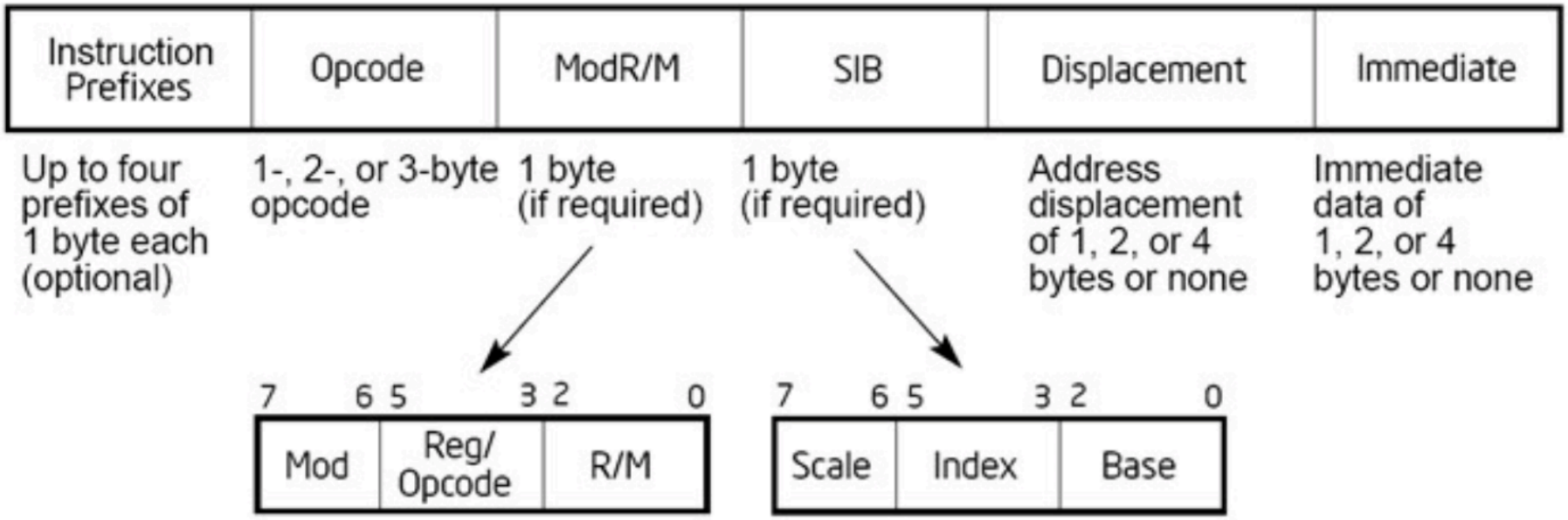
# Arquitecturas CISC vs. RISC

- Ejemplo: código máquina y ensamblador IA-32

```
8048374:    55                      push   %ebp
8048375:    89 e5                   mov    %esp,%ebp
8048377:    83 ec 08                sub    $0x8,%esp
804837a:    83 e4 f0                and    $0xfffffff0,%esp
804837d:    b8 00 00 00 00          mov    $0x0,%eax
8048382:    29 c4                   sub    %eax,%esp
8048384:    c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffc(%ebp)
804838b:    83 7d fc 09             cmpl   $0x9,0xfffffffc(%ebp)
804838f:    7e 02                   jle    8048393 <main+0x1f>
8048391:    eb 13                   jmp    80483a6 <main+0x32>
8048393:    c7 04 24 84 84 04 08    movl   $0x8048484,(%esp)
804839a:    e8 01 ff ff ff          call   80482a0 <printf@plt>
804839f:    8d 45 fc                lea    0xfffffffc(%ebp),%eax
80483a2:    ff 00                   incl   (%eax)
80483a4:    eb e5                   jmp    804838b <main+0x17>
80483a6:    c9                      leave
80483a7:    c3                      ret
80483a8:    90                      nop
80483a9:    90                      nop
80483aa:    90                      nop
```

# Arquitecturas CISC vs. RISC

- Ejemplo: formato máquina IA-32



# Arquitecturas CISC vs. RISC

- Ejemplo: código ensamblador ARM

```
        AREA    ARMex, CODE, READONLY
                                ; Name this block of code ARMex
        ENTRY                   ; Mark first instruction to execute
start
        MOV     r0, #10         ; Set up parameters
        MOV     r1, #3
        ADD     r0, r0, r1      ; r0 = r0 + r1
stop
        MOV     r0, #0x18       ; angel_SWIreason_ReportException
        LDR     r1, =0x20026    ; ADP_Stopped_ApplicationExit
        SVC     #0x123456       ; ARM semihosting (formerly SWI)
        END                     ; Mark end of file
```

# Arquitecturas CISC vs. RISC

- Ejemplo MIPS 16: código y máquina

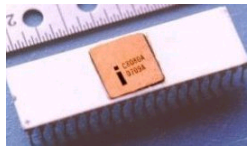
```
.data:00000000 3155          dsll s1,v0,5
.data:00000002 89d2          lh a2,36(s1)
.data:00000004 8be5          lh a3,10(v1)
.data:00000006 0845          la s0,0x00000118
.data:00000008 8b56          lh v0,44(v1)
.data:0000000a 0c75          la a0,0x000001dc
.data:0000000c 8d53          lh v0,38(a1)
.data:0000000e ff58          daddiu v0,sp,96
.data:00000010 b60f          lw a2,0x0000004c
.data:00000012 160c          b loc_00000000
.data:00000014 4c88          addiu a0,-120
.data:00000016 0113          addiu s1,sp,76
.data:00000018 c283          sb a0,3(v0)
.data:0000001a 8401          lb s0,1(a0)
.data:0000001c 75c9          cmpi a1,201
.data:0000001e 5bf1          sltiu v1,241
.data:00000020 5d5e          sltiu a1,94
.data:00000022 20c3          beqz loc_00000000
```

# Evolución de la arquitectura de Intel

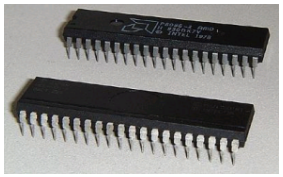
Evolución de la arquitectura y microarquitectura

- Ejemplo: microprocesadores de Intel

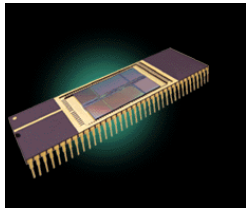
1971: **4004**
4 bits



1974: **8080**
8 bits



1978: **8086**
16 bits

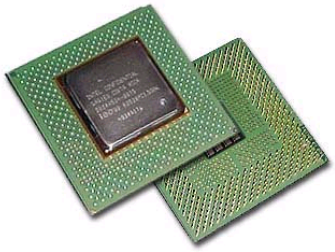


1982: **80286**
16 bits


1985: **80386**
32 bits (IA32)


2001: **Itanium**

IA-64



2004: **Pentium 4, Xeon**

64 bits (EM64T)