Estructura de Computadores

Tema 2.- Arquitectura del repertorio de instrucciones

LECTURAS OBLIGATORIAS:

W. Stallings, “Organización y arquitectura de computadores”:
Apartados 9.1, 9.2 y 12.4

BIBLIOGRAFÍA DE CONSULTA DEL TEMA:

W. Stallings, “Organización y arquitectura de computadores”, capítulos 9, 10, 12 y 13

Arquitectura del repertorio de instrucciones (Instruction Set Architecture):
Especificación que detalla el conjunto o repertorio de instrucciones que puede ejecutar una CPU, así como las características de la CPU que son visibles al software.

Repertorio de instrucciones:

- Punto de encuentro entre el diseñador de la CPU y el programador del lenguaje ensamblador.
- Depende del computador, lo que crea incompatibilidades entre los computadores con lenguajes máquina diferentes.
- Afecta de forma **muy significativa** a la implementación de una CPU o procesador.

Objetivo principal del diseñador:

Encontrar repertorio de instrucciones que:

- Facilite diseño del HW y del compilador.
- Minimice el coste y maximice el rendimiento.

Aspectos a considerar durante el diseño del repertorio de instrucciones:

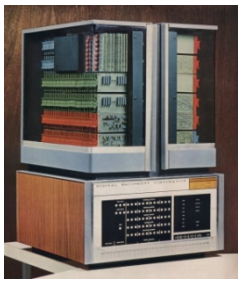
- **Conjunto de operaciones:** cuantas, de qué tipo y cuan complejas deben ser.
- **Tipos de datos que debe soportar:** enteros, coma flotante, arrays, caracteres,….
- **Formatos de las instrucciones:** longitud del formato, tamaño fijo, tamaño variable,….
- **Registros:** número de registros, uso de los mismos.
- **Modos de direccionamiento:** diferentes maneras de especificar las direcciones de los operandos.

Familia de computadores: misma arquitectura (ISA) con diferentes prestaciones (velocidad, precio, capacidad,…). Concepto introducido por IBM en 1964 para IBM 360 y adoptado el resto (Digital: PDP 8, PDP 11, Vax, Alpha, …).

PDP-8

Primer minicomputador

de la historia



Posible clasificación de ISA (atendiendo al almacenamiento de los operandos):

Es la distinción más básica…

Según el tipo de almacenamiento interno de la CPU:

- Operandos Implícitos:
  - Arquitecturas tipo PILA
  - Arquitecturas tipo ACUMULADOR
- Operandos Explícitos:
  - Arquitectura de registros de propósito general (GRP):
    - Registro – Memoria
    - Registro – Registro (Load – Store)
    - Memoria – Memoria

Arquitectura tipo PILA:

Modelo: B5500, HP 3000/70, ….
operandos explícitos: 0
Destino: Pila
Instr. de acceso a memoria: Push, Pop.

Ejemplo práctico: **C = A + B**

```
Push A
Push B
Add
Pop C
```

Ventajas:

- Modelo simple para evaluar expresiones
- Instrucciones cortas

Desventajas

- No permiten acceso aleatorio
- No generan código eficiente.

Arq. tipo ACUMULADOR:

Modelo: PDP-8 Motorola-6890
operandos explícitos: 1
Destino: Acumulador
Instr. de acceso a memoria: Load, Store.

Ejemplo práctico: **C = A + B**

```
Load A
Add B
Store C
```

Ventajas:

- Instrucciones más cortas
- Se minimizan los estados de la U.C

Desventajas

- Tráfico grande entre memoria y procesador

Arquitectura tipo GRP:

Modelo: IBM 370, VAX 11/780, MIPS,….
operandos explícitos: 2 ó 3
Destino: Registros o memoria
Instr. de acc. a mem.: Load,Store y Move.

Ejemplo práctico: **C = A + B**

```
Load R1, A
Load R2, B
Add R3, R1, R2
Store C, R3
```

Ventajas:

- Buen modelo para optimizar código. Mayor densidad de código.
- Menor tráfico de memoria (op. reg)

Desventajas

- Operandos explícitos (instrucciones más largas)

Tipo pila y Acumulador: en desuso desde 1975

Arquitecturas de tipo GPR:

Estas arquitecturas pueden a su vez clasificarse atendiendo a:

- Número de operandos explícitos por instrucción típica de la ALU.
  (sin diferenciar entre operandos de registro o de memoria)
  - 2 operandos: ADD R1, R2;
  - 3 operandos: ADD R1, R2, R3;
- Número de operandos direccionados en memoria por las instrucciones de la ALU.
  - de 0 a 3 operandos

Según esto, podemos encontrar las siguientes posibilidades:

- (0,3) → arquitecturas GPR de tipo LOAD/STORE (MIPS, SPARC, CDC 6600, …).
- (1,2) → arquitecturas GPR de tipo REG/MEM (PDP11, MOTOROLA 68000,…).
- (3,3) → arquitecturas GPR de tipo MEM/MEM (VAX11/780,…).

Arquitectura tipo LOAD/STORE: (0,3)

También conocidas como reg/reg

Ventajas:

- Codificación más simple
- Formatos de long. fija.
- Modelo simple de generación de código.
- Facilitan tarea compilador (menos decisiones).
- CPI similar en cada instrucción.

Desventajas:

- Mayor número de instrucciones.
- Algunas instrucciones desperdician bits del formato.

Ejemplo práctico: **C = A + B**

```
Load R1, A
Load R2, B
Add R3, R1, R2
Store C, R3
```

Arquitectura tipo REG/MEM: (1,2)

Ventajas:

- Acceso a los datos sin carga s previas
- Formatos de instrucción fáciles de codificar.
- Buena densidad de código.

Desventajas:

- Operandos no equivalentes (se destruye un operando fuente).
- Se restringe el número de bits para codificar registros.
- CPI variable.

Ejemplo práctico: **C = A + B**

```
Load R1, A
Add R1, B
Store C, R1
```

Arquitectura tipo MEM/MEM: (3,3)

Ventajas:

- Modelo más compacto
- No se gastan registros para almacenar variables intermedias.

Desventajas:

- Aumenta el tamaño de instrucción por el uso de instrucciones de tamaño variable. .
- Se originan cuellos de botella por accesos a memoria.
- CPI muy variable.
- Complican la tarea del compilador.

Ejemplo práctico: **C = A + B**

```
Mem[C]=Mem[A]+Mem[B]
```

Ejemplo: E = (A-B)*(C+D) mediante repertorios con diferente número de operandos explícitos por instrucción

E = (A – B)*(C + D)			
3 operandos	2 operandos	1 operando	0 operandos

ADD C, D, C
SUB A, B, A
MUL A, C, E
	ADD C, D
	SUB A, B
	MUL D, B
	MOV B, E
		LOAD A
		SUB B
		STORE A
		LOAD C
		ADD D
		MUL A
		STORE E
			(PUSH) LOAD A
			(PUSH) LOAD B
			SUB
			(PUSH) LOAD D
			(PUSH) LOAD C
			ADD
			MUL
			(PULL) STORE E

Ejemplo: E = (A+B) mediante repertorios con modo de almacenamiento de operandos en la CPU

C = A + B		
Pila	Acumulador	Conjunto de registros
PUSH A PUSH B ADD POP C	LOAD A ADD B STORE C	LOAD A, R1 ADD B, R1 STORE R1, C

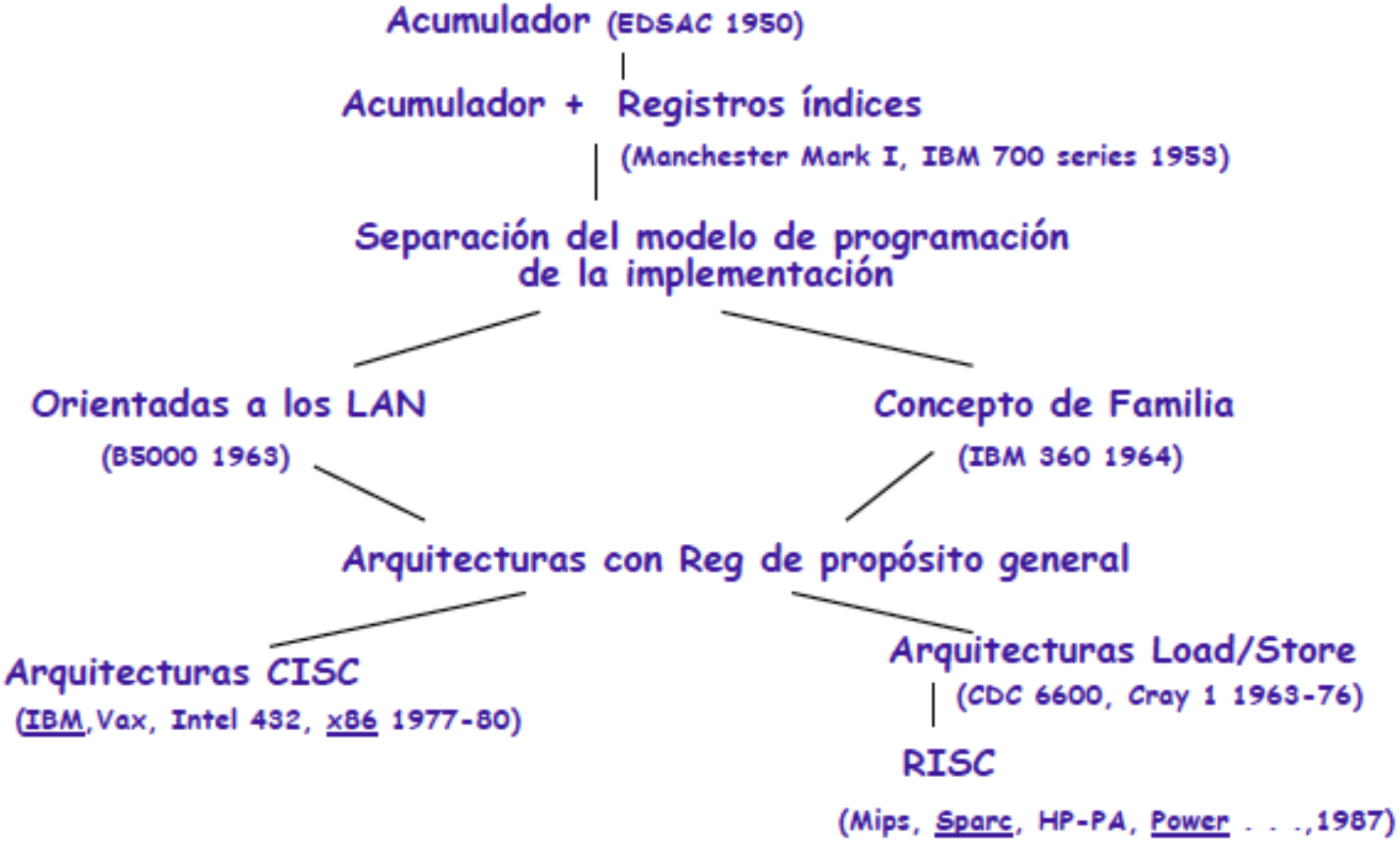
Conclusiones:

Las arquitecturas tipo PILA y ACUMULADOR han caído en desuso, por lo que casi todas las máquinas emplean conjuntos de registros de propósito general.

- Principales ventajas del uso de registros:
  - Los registros son más rápidos que la memoria.
  - Los registros son más fáciles de usar por el compilador.
  - Los registros pueden contener variables.
    - Se reduce así el tráfico de datos a/desde memoria: programas más rápidos.
    - Mejora la densidad de código (referenciar a un registro requiere menos bits que referenciar a una posición de memoria).

Las nuevas arquitecturas se basan y se basarán en el uso de registros de propósito general. La questión a determinar es el número de registros óptimo.

Evolución de los repertorios de instrucciones:



Evolución de los repertorios de instrucciones:

SISD: Single **I**nstruction **S**ingle **D**ata

SIMD: Single **I**nstruction **M**ultiple **D**ata (extensiones progresivamente acumuladas sobre repertorio x86)

MMX: Multimedia e**X**tension (1997, Pentium MMX)
3D Now! (1998, K6-2)
SSE: **S**treaming **SIMD E**xtensions (1999, Pentium **III**)
SSE2 (Pentium 4, Opteron y Athlon 64)**, SSE3** (P4 Prescott, Core 2 Duo y Xeon)**, SSE4 y SSE4a** (Core 2 Duo 45mm)
AVX: Advanced **V**ector e**X**tensions (2011 iX Sandy Bridge e **I**vy Bridge, AMD Bulldozer)
AVX2 (Intel Haswell, arquitectura sucesora de Ivy Bridge)
También llamadas **Haswell New Instructions**
AVX3 or AVX-512 (Knights Landing Xeon Phi, Skylake, Cannonlake, ...)

Evolución de los repertorios de instrucciones:

SIMD + AVX → "Paralelismo utilizando un único core"

Idea: si tenemos registros cuyo tamaño es múltiplo del tamaño de los datos con los que operamos, podemos acelerar la ejecución de algunas instrucciones sobre dichos registros.

Ejemplo: si tenemos registros de 256 bits, podemos realizar operaciones sobre 8 enteros de 32 bits utilizando una única instrucción, lo cual acelerará dichos cálculos.

MMX: Multimedia e**X**tension (1997, Pentium MMX)

8 op. enteras de 8-bits o 4 op. enteras de 16 bits

SSE: **S**treaming **SIMD E**xtensions (1999, Pentium **III**)

SSE2 (Pentium 4, Opteron y Athlon 64)**, SSE3** (P4 Prescott, Core 2 Duo y Xeon)**, SSE4 y SSE4a** (Core 2 Duo 45mm)

8 op. enteras de 16 bits

4 op. enteras/FP de 32 bits o 2 op. enteras/FP de 64 bits

AVX: Advanced **V**ector e**X**tensions (2011 iX Sandy Bridge e **I**vy Bridge, AMD Bulldozer)

4 op. enteras/FP de 64 bits

Los operandos deben ser consecutivos y en posiciones de memoria alineadas

Evolución de los repertorios de instrucciones:

Ejemplo: instrucciones AVX para arquitectura x86

4 operandos de 64 bits

AVX Instruction	Description
VADDPD	Add four packed double-precision operands
VSUBPD	Subtract four packed double-precision operands
VMULPD	Multiply four packed double-precision operands
VDIVPD	Divide four packed double-precision operands
VFMADDPD	Multiply and add four packed double-precision operands
VFMSUBPD	Multiply and subtract four packed double-precision operands
VCMPxx	Compare four packed double-precision operands for EQ, NEQ, LT, LE, GT, GE, ..
VMOVAPD	Move aligned four packed double-precision operands
VBROADCASTSD	Broadcast one double-precision operand to four locations in a 256-bit register

Evolución de los repertorios de instrucciones:

Principios que inspiran el diseño y evolución de las ISAs:

1.- La simplicidad favorece la regularidad: la regularidad motiva gran parte de las características de los repertorios de instrucciones: tratar de mantener el mismo tamaño para todas las instrucciones (o grupos de ellas), mismo número de operandos para cada grupo de instrucciones, regularidad en los formatos de instrucción,…)

2.- Más pequeño es más rápido: el deseo de velocidad es lo que ha hecho que la capacidad de integración haya evolucionado tanto. Ese mismo deseo ha hecho que las ISAs se basen en el uso de pequeños bancos de registros de propósito general.

3.- Un buen diseño exige un buen compromiso: en el sentido de encontrar soluciones de compromiso ante decisiones encontradas: proporcionar mayor capacidad de direccionamiento o diferente número de operandos y mantener todas las instrucciones con el mismo tamaño.

Las instrucciones: concretando…

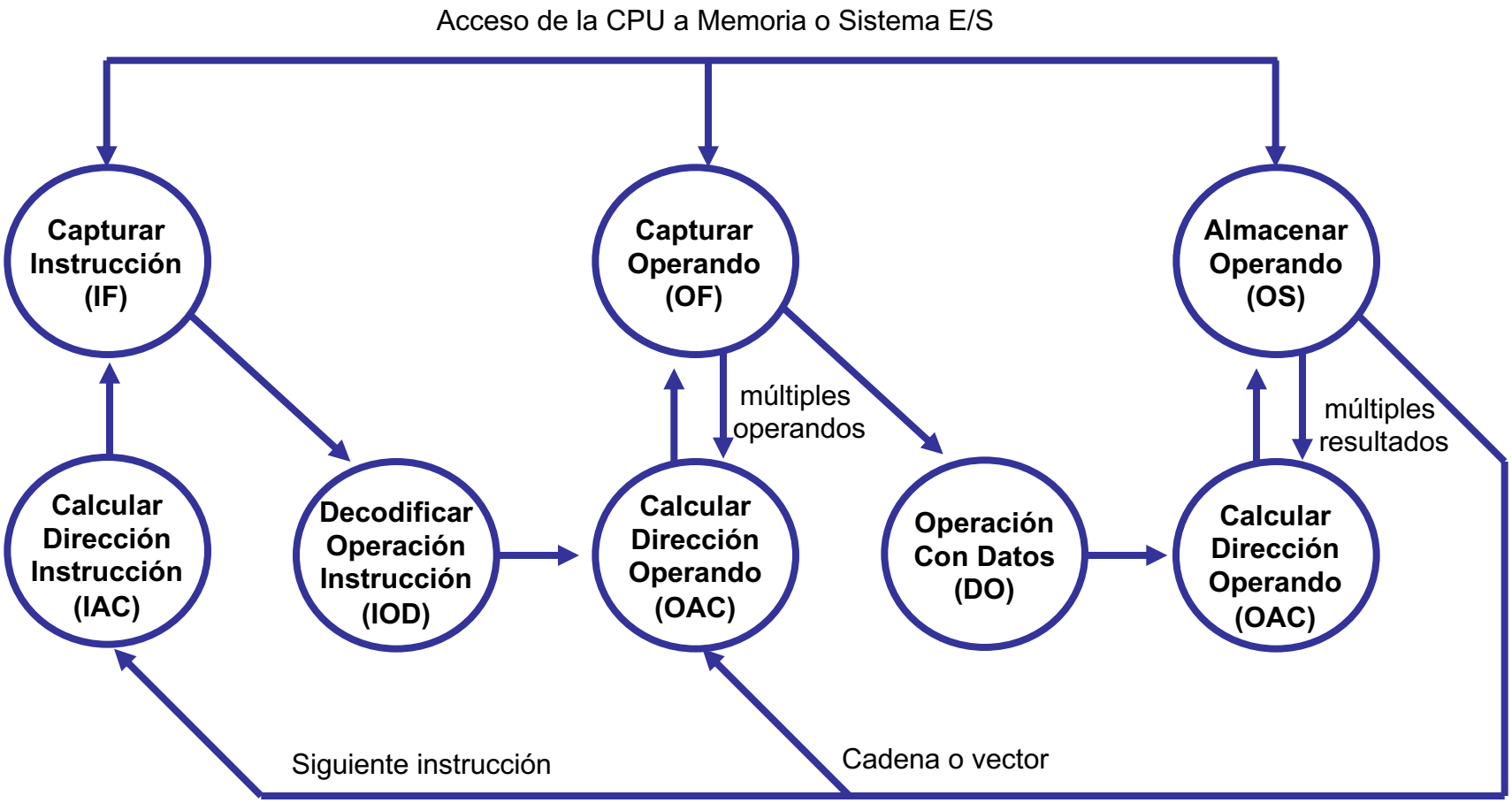
Formato de instrucción ←→ representación de la instrucción: especifica el significado de cada uno de los bits.

- Información que debe contener:
  - Operación a realizar: código de operación.
  - Direcciones de los operandos fuente y destino.
  - Dirección de la siguiente instrucción **(*)**.

(*) Solo en los casos en los que se necesiten los saltos (alterar secuencialidad).

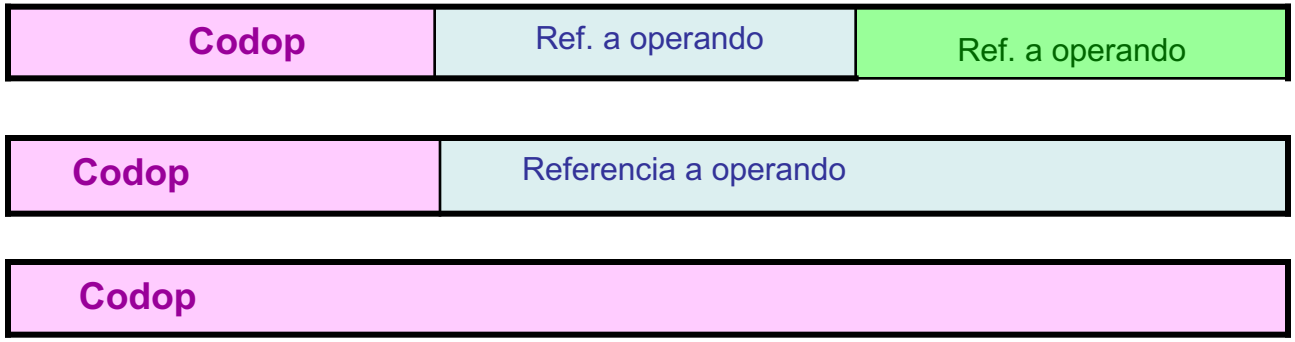
Cada tipo de información suele estar formada por grupos de bits llamados campos.

Diagrama de estados del ciclo de instrucción:



Las instrucciones: tipos básicos de campos.

Campos de dirección (hacen referencia a los operandos).



Ejemplos básicos de formatos de longitud fija (máximo 3 operandos).

Características del formato de instrucción:

•Longitud fija (mismo tamaño para todas las instrucciones). Mejora el rendimiento.

•Longitud variable (varía su tamaño según el tipo de instrucción). Disminuye el tamaño del código.

•Debe haber pocos formatos y deben ser sistemáticos.

•Instrucciones cortas mejor que largas (mayor rapidez de ejecución).

- No se suele poner la dirección de la siguiente instrucción.
- El tamaño de los formatos debe ser múltiplo o submúltiplo del tamaño de la palabra de la máquina.
- El tamaño de la palabra debería ser múltiplo de byte.

El tamaño del formato afecta de forma decisiva a la complejidad de la CPU

Las instrucciones: operaciones que pueden codificar.

- No todas las operaciones son siempre necesarias.
- Factores que influyen sobre la decisión de incluir o no una operación:
  - Frecuencia de las mismas y tamaño de la instrucción.
    - Longitud de la instrucción inversamente proporcional a la frecuencia de utilización de sus operaciones asociadas.
- Relación entre la cantidad de operaciones soportadas (codificadas) por las instrucciones y la organización de las mismas:
  - Tamaño fijo del COP ←→ formato de la instrucción.
  - Tamaño fijo de la instrucción ←→ cantidad de operaciones, número de operandos.
  - Compromiso = COP con extensión

- Extiende el COP para instrucciones con menor número de operandos.
- El COP extendido codifica la operación a realizar por la instrucción.
- En el COP básico no debe agotarse la capacidad máxima de codificación.

Las instrucciones: códigos de operación de longitud fija y variable

- Los bits del campo ref. a operando se pueden utilizar para extender el Codop de aquellas instrucciones con menor número de operandos

Ejemplo: partimos de una máquina con instrucciones de longitud fija de 24 bits y consideraremos los siguientes supuestos:

1) La máquina dispone de 16 registros generales

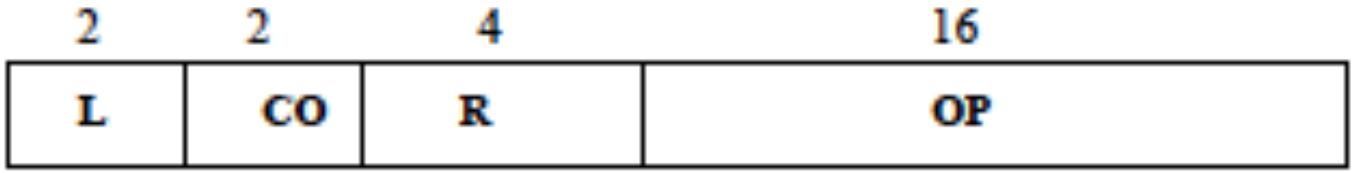
En este caso se pueden codificar 16 instrucciones de 2 operandos: uno en registro y otro en memoria

2) Si queremos extender el CO …

Las instrucciones: códigos de operación de longitud fija y variable

- Los bits del campo ref. a operando se pueden utilizar para extender el Codop de aquellas instrucciones con menor número de operandos

Ejemplo: partimos de una máquina con instrucciones de longitud fija de 24 bits y consideraremos los siguientes supuestos:



En este caso podemos codificar los siguientes grupos de instrucciones:

L = 00 → CO de 2 bits → 4 instrucciones de 2 operandos

L = 01 → CO de 6 bits → 64 instrucciones de 1 operando

L = 10 → CO de 22 bits → 4.194.304 instrucciones de 0 operandos

Las instrucciones: códigos de operación de longitud fija y variable

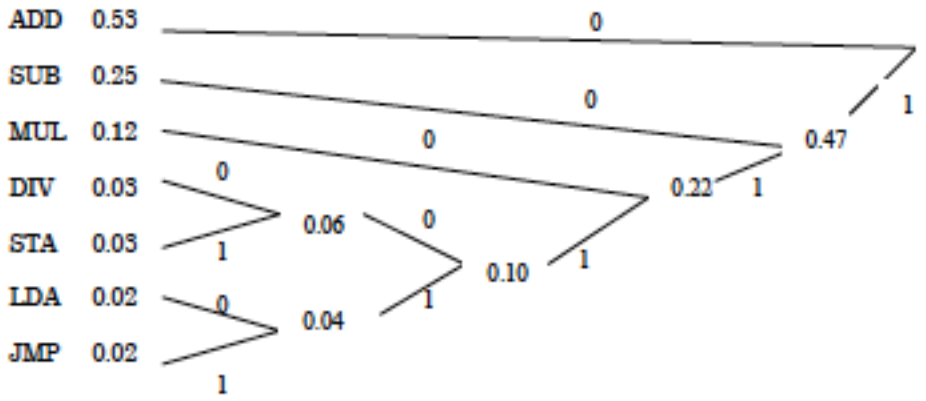
- Optimización del CO variable en función de la frecuencia de las instrucciones:

Una posibilidad a la hora de codificar las operaciones de un repertorio de instrucciones es utilizar algún criterio de óptimo. Alternativas:

Frecuencia de aparición en el programa → optimización de memoria
Frecuencia de ejecución en el programa → optimización tráfico CPU-Mem

Ej.: codificación de Huffman → código de longitud variable con la propiedad de no superposición de los CO resultantes (garantiza que el CO de una determinada instrucción no coincide con la subcadena inicial de bits del CO de otra instrucción). Decodificación de izquierda a derecha

Tipo de instrucciones	Frecuencia de ejecución
ADD	0.53
SUB	0.25
MUL	0.12
DIV	0.03
STA	0.03
LDA	0.02
JMP	0.02



Las instrucciones: operaciones que pueden codificar.

- Transferencia de datos.
- Aritméticas.
- Lógicas.
- Transferencia de control
- Entrada/Salida.
- Conversión (tipos de datos/direcciones).
- Control del sistema (privilegios, SO's,…).

- Requerimientos básicos:

Todas las máquinas generalmente proporcionan instrucciones de transferencia de datos, control, aritméticas y lógicas.

Todas las máquinas deben tener algún soporte básico para funciones del sistema (variando ampliamente entre diferentes arquitecturas).

Soporte para el manejo de ciertos tipos de datos (decimales, p. flotante y strings) que varían mucho entre arquitecturas distintas.

Las instrucciones de punto flotante en ocasiones forman parte de un repertorio opcional

Las instrucciones: operaciones más comunes del ISA.

Transferencia de datos	**Move** (Transferir)	Transfiere una palabra o bloque desde origen a destino
	Store (guardar en memoria)	Transfiere una palabra desde un registro a memoria
	Load (Cargar desde memoria)	Transfiere una palabra desde memoria a un registro
	Exchange (Intercambiar)	Intercambia los contenidos del origen y el destino
	Clear (Poner a 0)	Transfiere una palabra de ceros al destino
	Set (Poner a 1)	Transfiere una palabra de unos al destino
	Push (Apilar)	Transfiere una palabra desde un origen al tope de la pila
	Pop (Desapilar)	Transfiere una palabra desde el tope de la pila al destino
Aritméticas	**Add** (sumar)	Calcula la suma de dos operandos
	Substract (restar)	Calcula la resta de dos operandos
	Multiply (multiplicar)	Calcula la multiplicación de dos operandos
	Divide (dividir)	Calcula la división de dos operandos
	Absolute (valor absoluto)	Substituye el operando por su valor absoluto
	Negate (multiplicar x -1)	Cambia el signo del operando
	Increment (incrementar)	Suma 1 al operando
	Decrement(decrementar)	Resta 1 al operando

Las instrucciones: operaciones más comunes del ISA.

Control de Flujo	**Jump** (salto o bifurcación)	Transfiere el control. Carga el PC con dirección especificada
	Branch (salto condicional)	Carga el PC con nueva dirección si se cumple condición
	Jump & link (salto a subrutina)	Igual que Jump y además guarda dirección de retorno
	Return (retorno de subrutina)	Sustituye el contenido del PC con el valor guardado
	Execute (ejecutar)	Capta operando y lo ejecuta como una instrucción
	Skip (salto implícito)	Incrementa el PC para saltar a la instrucc. siguiente
	Skip cond. (salto impl. condicional)	Incrementa el PC si se cumple la condición
	Halt (parar)	Detiene la ejecución del programa
	Wait (esperar)	Detiene la ejecución del programa hasta cumplir condición
Lógicas	**And** (producto lógico)	Calcula la operación lógica Y de dos operandos
	Or (suma lógica)	Calcula la operación lógica O de dos operandos
	Not (complemento)	Calcula el complemento bit a bit de un operando
	Xor (or exclusiva)	Calcula la operación lógica O-exclusiva de dos operandos
	Test (comparar)	Comprueba la condición específica (modifica flags)
	Shift (desplazar)	Desplaza un operando a izquierda o derecha
	Rotate (rotar)	Desplaza un operando a izq. o der. y rota último bit
	Set control(fijar v. de control)	Fija control para protección,temporizador, interrupciones

Las instrucciones: operaciones más comunes del ISA.

Entrada/Salida	**Input** (Entrada)	Transfiere datos desde un puerto o dispositivo E/S al destino (memoria o registro)
	Output (Salida)	Transfiere datos desde memoria o registro a un puerto o dispositivo de E/S
	Start I/O (Iniciar E/S)	Transfiere instrucciones al procesador E/S para inciar las operaciones de E/S
	Test I/O (Comprobar E/S)	Transfiere la información de estado desde el sistema de E/S al destino especificado
Conversión	**Translate** (traducir)	Traduce valores de una sección de memoria basada en una tabla de correspondencia
	Convert (convertir)	Convierte el contenido de una palabra de un formato a otro (por ejemplo, de decimal empaquetado a binario)

Las instrucciones: concepto de ortogonalidad.

En general: **principio por el cual 2 variables son independientes una de la otra.**

En el contexto de **repertorio de instrucciones**, **un conjunto de instrucciones es ortogonal si cada operación puede realizarse con cualquier tipo de operando y con cualquier tipo de direccionamiento.**

Entonces se dice que el **Código de Operación** representa a **operaciones PURAS.**

Ventajas:

- Aporta claridad y simplicidad
- Simplifica la interpretación y decodificación de las instrucciones

Desventajas:

- Se codifican innecesariamente operaciones poco útiles

La tendencia en las computadoras CISC fue diseñar repertorios de instrucciones ortogonales (PDP11, VAX, …).

Las instrucciones: ejemplos MIPS R-2000

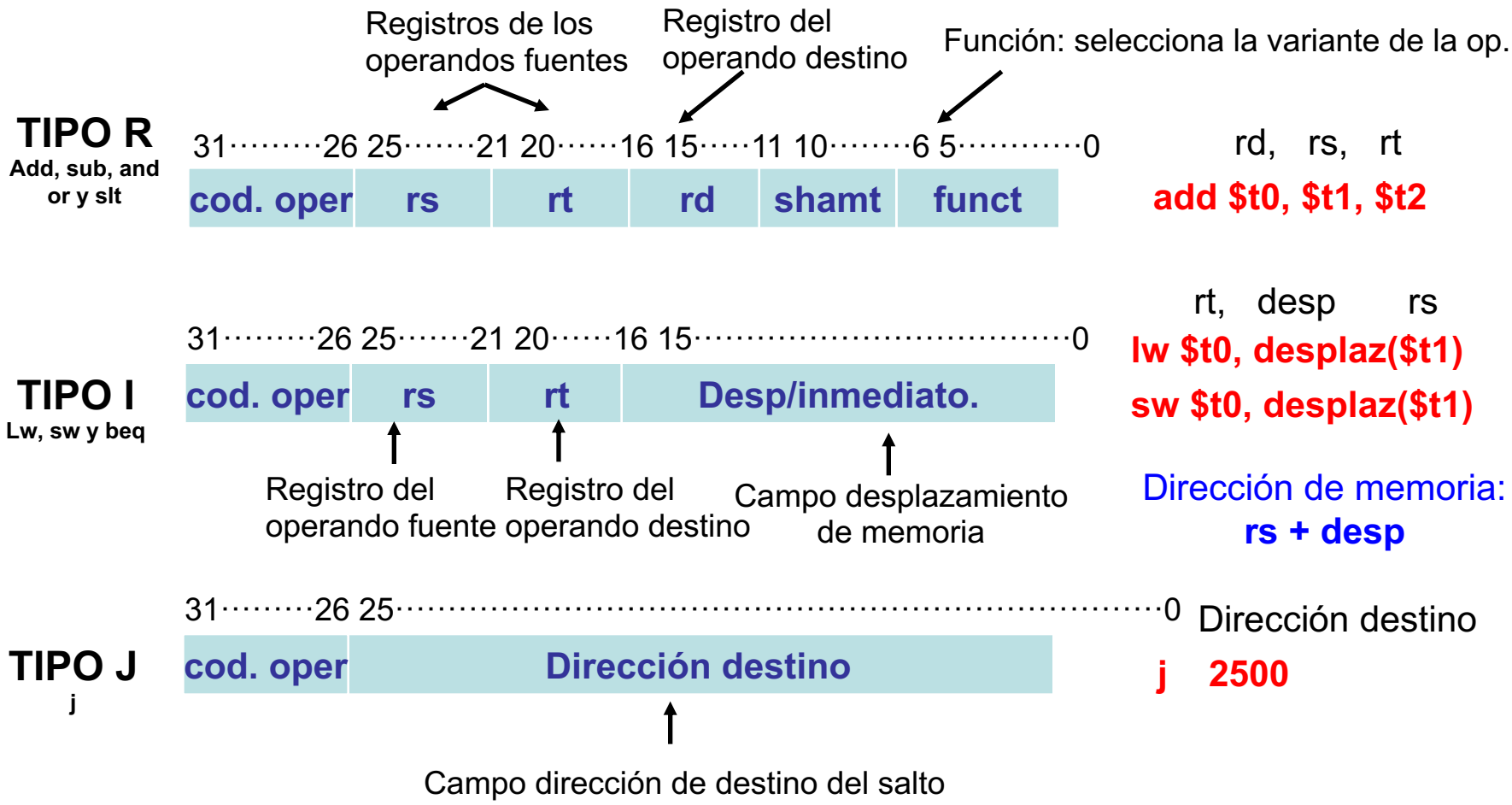
REPERTORIO DE INSTRUCCIONES

Operaciones que actúen sobre los posibles datos de la máquina

- Instrucciones de referencia a memoria:
  - Ej: MIPS R-2000: LW, SW (carga palabra, almacenar palabra)
- Instrucciones aritmético-lógicas:
  - Operaciones lógicas y de desplazamiento:
    - Ej: MIPS R-2000: SRL, SLL, AND, OR, XOR,…
  - Operaciones aritméticas:
    - Ej: MIPS R-2000: ADD, SUB, MULT, DIV,…
  - Operaciones de comparación:
    - Ej: MIPS R-2000: SLT, SGE, SGT, SNE,…
- Instrucciones de control de secuencia:
  - Ej: MIPS R-2000: BEQ, J, …
- Instrucciones de propósito específico.
- Instrucciones de E/S:
  - Ej: MIPS R-2000: manejo de interrupciones y excepciones,…

Las instrucciones: ejemplo

Ejemplo de formatos típicos de la arquitectura MIPS R-2000:



Las instrucciones: tipos de datos y operandos.

El conjunto de instrucciones de una máquina puede realizar operaciones sobre:.

- Direcciones.
- Números:
  - Punto fijo (enteros, punto fijo, complemento a 2).
  - Punto flotante: IEEE-754: estándar aprobado en los años 80:
    - Simple precisión: 32 bits
    - Doble precisión: 64 bits
    - Precisión extendida
  - Decimal:
    - Empaquetado: 4 bits/dígito (Código BCD).
    - Desempaquetado: 8 bits/dígito.
    - Carácter numérico en cadenas
- Caracteres:
  - ASCII: 128 caracteres imprimibles + char CONTROL + bit paridad)
  - EBDC: IMB 370 y grandes computadores,…)
- Datos lógicos: 4 (niblle), 8 (byte), 16 (media palabra), 32, 64, …, cualquier unidad lógica direccionable

Las instrucciones:

MIPS operands

Name	Example		Comments
32 registers	$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at		Fast locations for data. In MIPS, data must be in registers to perform arithmetic. register $zero always equals 0, and register $at is reserved by the assembler to
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

Las instrucciones: especificación de los operandos.

La mayoría de las máquinas direccionan la memoria por bytes:

BIG ENDIAN: La dirección del byte con valor numérico más bajo corresponde al **BYTE MAS SIGNIFICATIVO** del operando.

LITTLE ENDIAN: La dirección del byte con valor numérico más bajo corresponde al **BYTE MENOS SIGNIFICATIVO** del operando.

Ejemplo-1: $(3101)_{10} = 12*16^2 + 1*16^1 + 13*16^0 = (00\ 00\ 0C\ 1D)_{16}$

BIG ENDIAN

4n	**00**
4n +1	**00**
4n + 2	**0C**
4n + 3	**1D**

Ventajas:

Más rápido para ordenar cadenas de caracteres.

Las comparaciones con enteros son más rápidas

Orden coherente:
Los volcados de memoria se imprimen de izquierda a derecha sin causar confusión.

IBM, MIPS, Sun SPARC,…

LITTLE ENDIAN

4n	**1D**
4n +1	**0C**
4n + 2	**00**
4n + 3	**00**

Ventajas:

Más fácil las operaciones aritméticas de alta precisión.

Se ahorran sumas al convertir direcciones enteras de 32 bits a direcciones de 16 bits

Intel 80x86, Pentium,…

BI-ENDIAN: Permite a los ingenieros elegir uno u otro cuando portan aplicaciones o sistemas operativos (El S.O elige el modo).

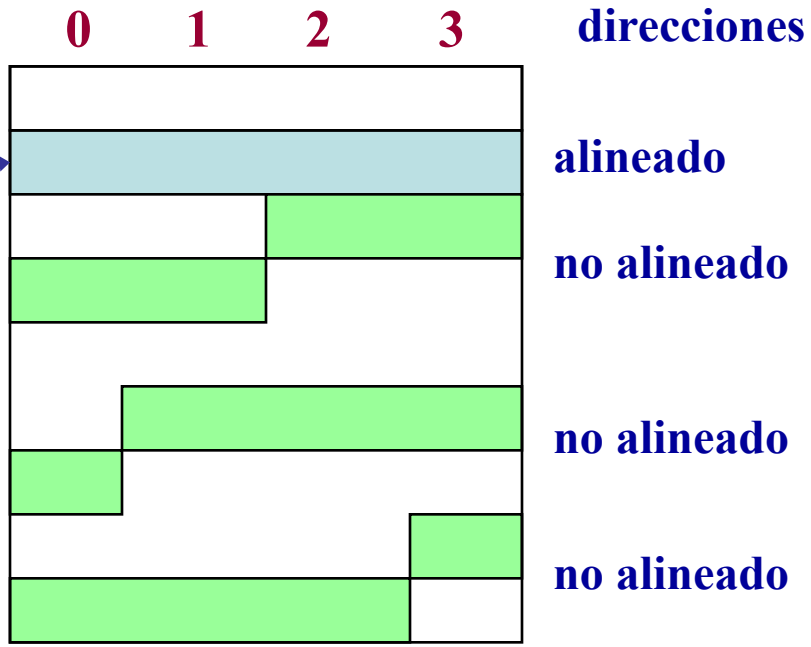
Las instrucciones: especificación de los operandos.

La mayoría de las máquinas requiere que los objetos a acceder estén alineados en memoria:

Un objeto (nstrucción o dato) de S bytes (siendo S potencia de 2) se encuentra alineado en memoria si se cumple la condición:

(Dirección objeto) MOD S = 0

Tamaño S = nº bytes	Alineado en las direcciones:
S = 2	0, 2, 4, 6, 8, 10, 12,...
S = 4	**0, 4, 8, 12, ...**
S = 8	0, 8, 16,...



Ventajas:

Reduce la complejidad del hardware

Reduce el tiempo de acceso a memoria (crucial)

Las instrucciones: especificación de los operandos.

- **Modos de direccionamiento:** sirven para calcular la dirección efectiva de un operando o instrucción. **Dirección efectiva = F (Información de la instrucción).**

- Los más comunes son:
  - Directo (Absoluto).
  - Registro.
  - Inmediato.
  - Directo + Registro (indexado).
  - Registro base + Desplazamiento.
  - Indirecto por memoria.
  - Indirecto por registro.
  - Registro base + registro índice.
  - Registro base + Desplazamiento + Registro índice.
  - Implícito.

Ventajas:

Reducen el número de instrucciones de la máquina

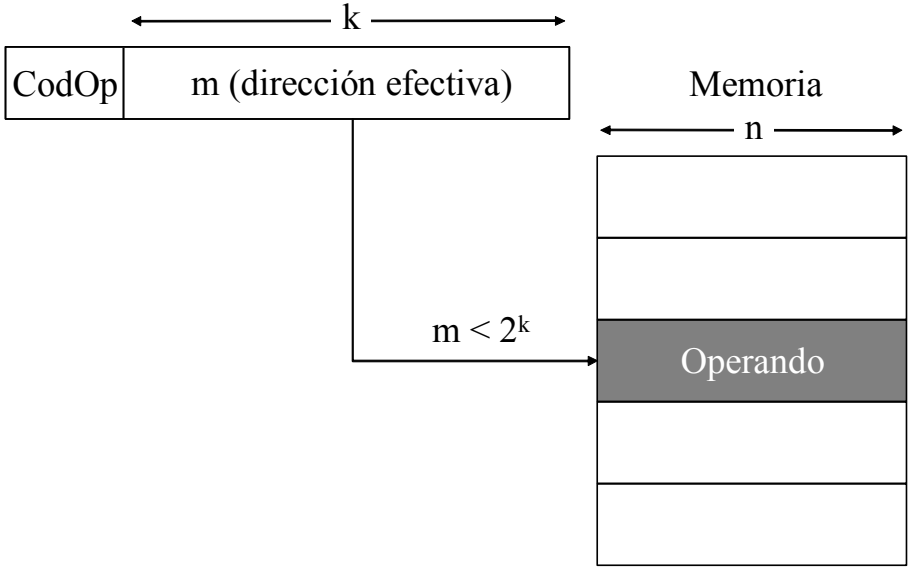
Amplían el rango de direccionamiento

Desventajas:

Añaden complejidad a la máquina

Las instrucciones: especificación de los operandos.

- **Direccionamiento Directo o Absoluto:**

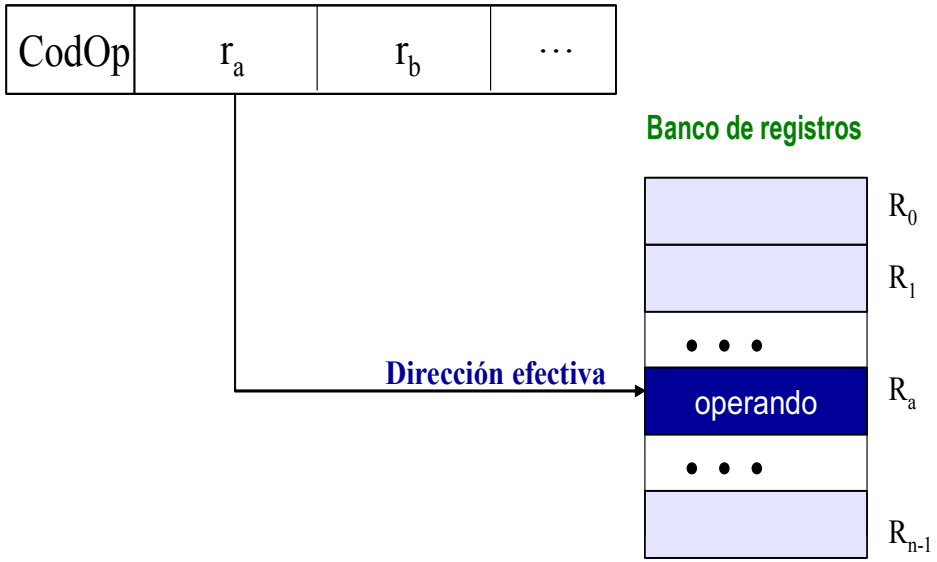


Suele usarse para acceder a datos estáticos.

Desventajas:

Espacio de direccionamiento reducido (1ª generación de computadores)

- **Direccionamiento por registro:**



Ventajas:

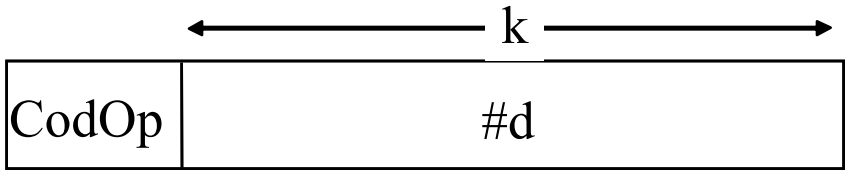
Campo de dirección pequeño.
Sin accesos a memoria.

Desventajas:

Espacio de direccionamiento limitado

Las instrucciones: especificación de los operandos.

- **Direccionamiento inmediato**



$d < 2^k$

Operando en la instrucción

Suele usarse **inicialización y manipulación de constantes** (op. Aritméticas, comparaciones, etc).

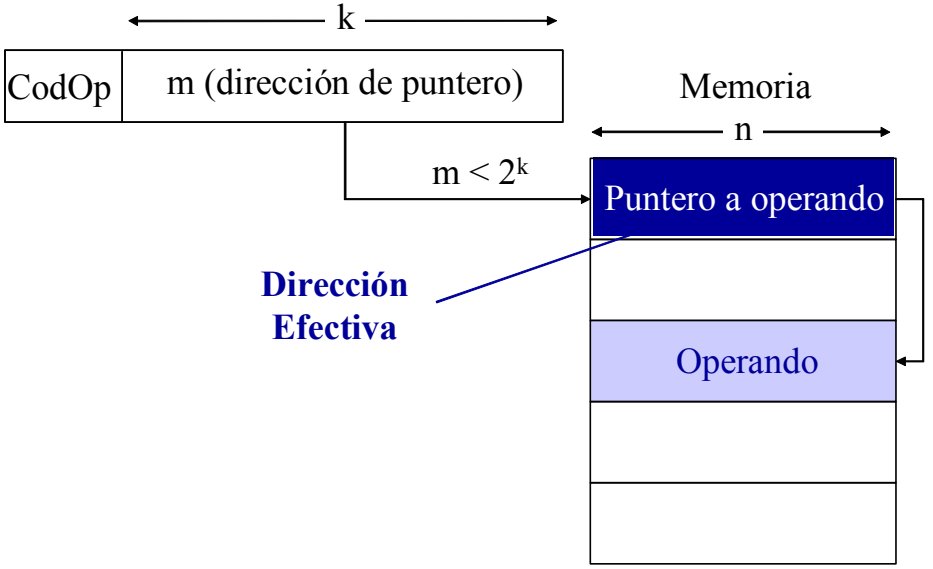
Ventajas:

No hay accesos a la memoria.

Desventajas:

Tamaño reducido de las constantes

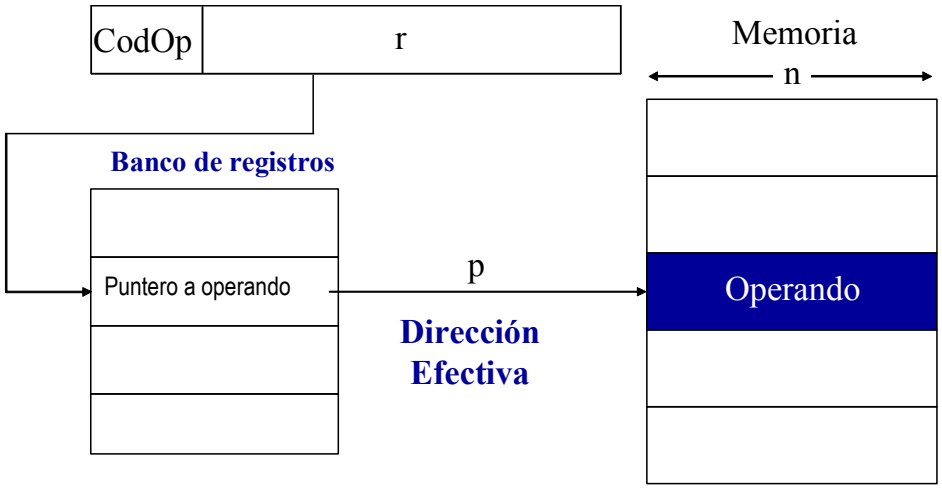
- **Direccionamiento indirecto por memoria:**



Suele usarse para acceder a **datos con punteros**.

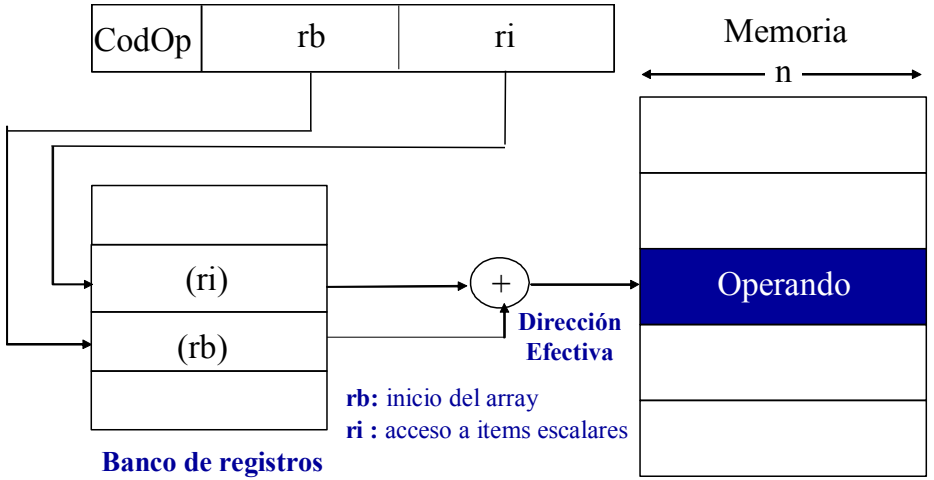
Las instrucciones: especificación de los operandos.

- **Direccionamiento indirecto por registro:**



Suele usarse para el **acceso de variables mediante punteros** (dirección efectiva: p).

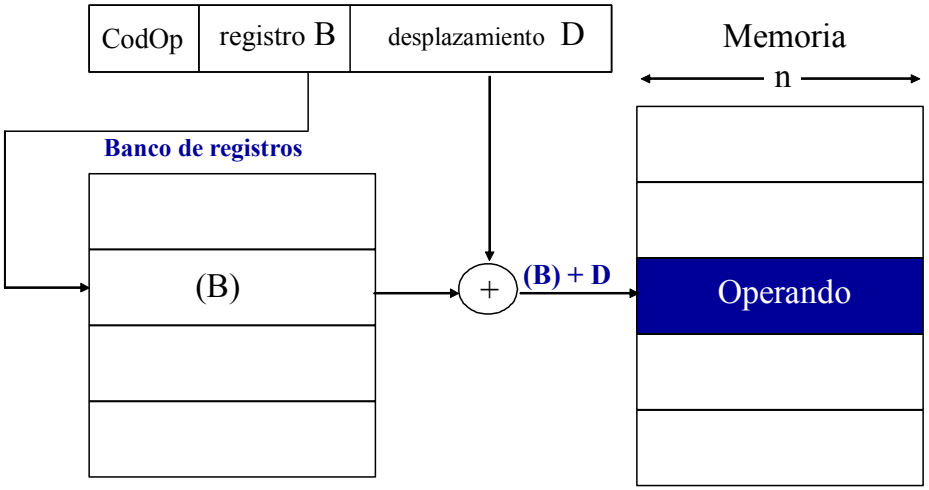
- **Direccionamiento base + índice:**



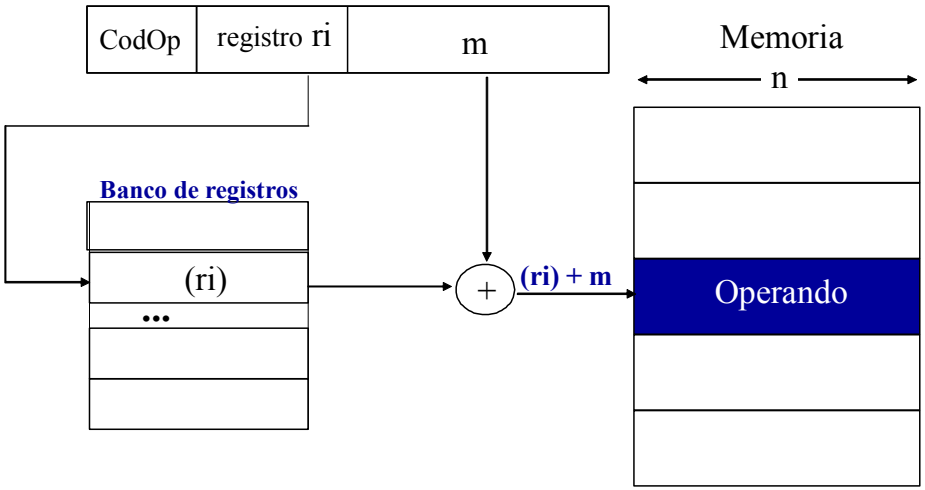
Suele usarse para el **direccionamiento de arrays** (dirección efectiva: (rb) + (ri)).

Las instrucciones: especificación de los operandos.

- **Direccionamiento Base + Desplazamiento:**



- **Direccionamiento Directo + Registro índice (indexado):**



Suele usarse para:

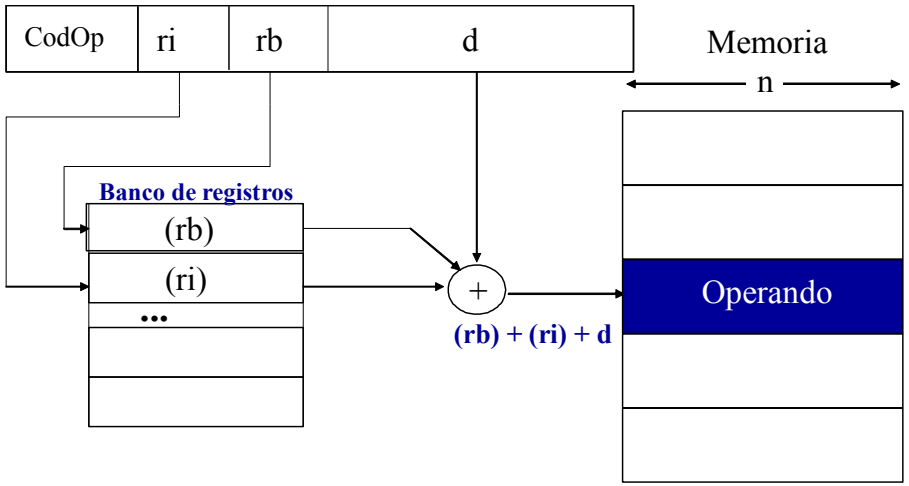
Acceso a variables locales.

Para implementar la segmentación.

Suele usarse en **operaciones iterativas** como **accesos a lo elementos de un array**.

Las instrucciones: especificación de los operandos.

Direccionamiento Base + Índice + Desplazamiento:



Suele usarse para:

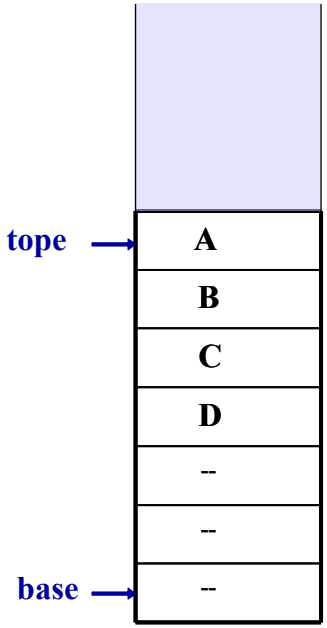
Acceso a estructuras multidemensionales (matrices,...)..

Accesos a campos de un array de records.

Direccionamiento Implícito (PILA):

PILA:

- Matriz lineal de posiciones de memoria que constituye un bloque reservado.
- Tiene asociado un puntero cuyo valor es el de la cabecera o tope de la pila.
- A veces los dos elementos primeros de la pila se almacenan en registros del procesador (más velocidad).
- El formato de instrucción no incluye referencias a memoria.
- A la memoria se accede implícitamente mediante el puntero de tope de pila guardado en un registro (SP)
- Las referencias a las posiciones de memoria en la pila son en realidad accesos indirectos por registro.



Direccionamientos en la MIPS

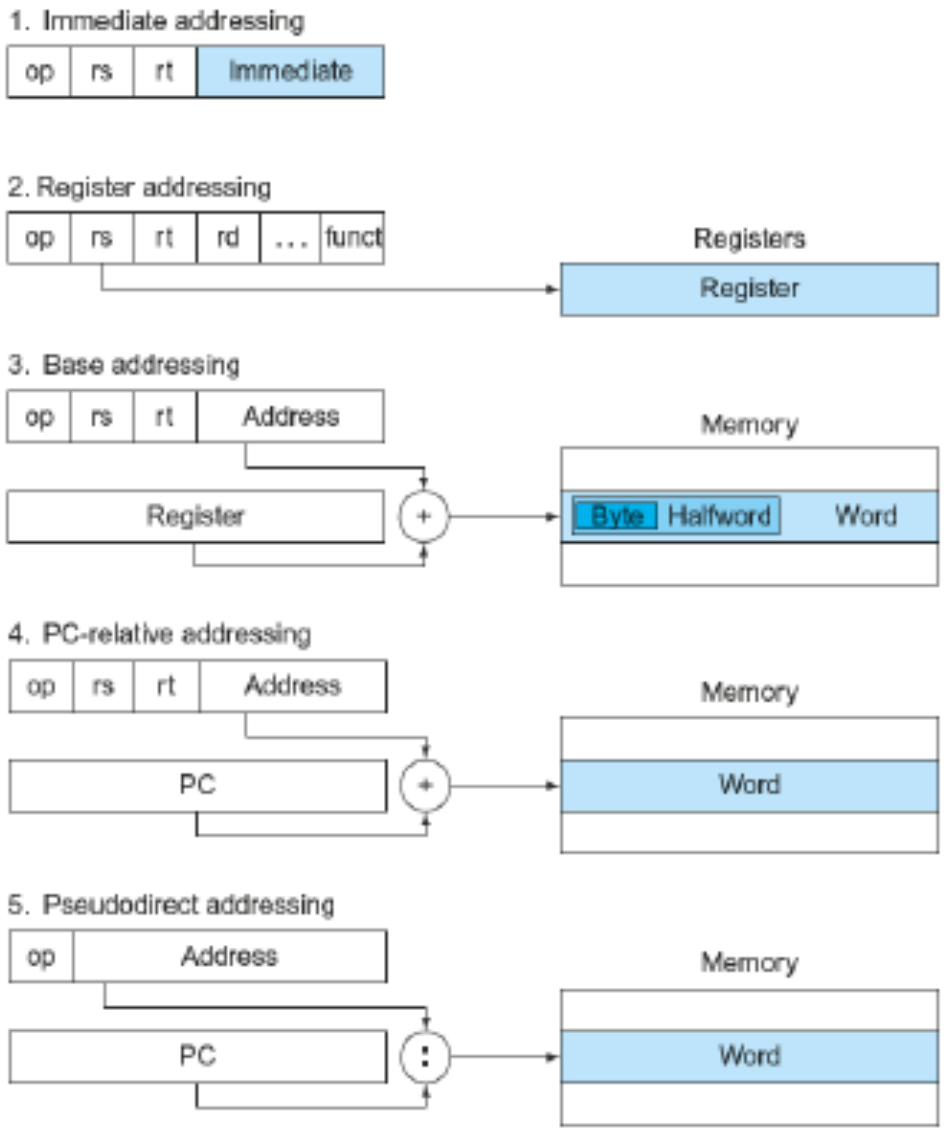
El operando está en la propia instrucción (campo inmediato)

El operando está en alguno de los registros de la máquina

El operando está en la posición de memoria apuntada por la suma del contenido de un registro base + una cantidad de desplazamiento

La dirección de sato se calcula como la suma del contenido del PC + una cantidad de desplazamiento

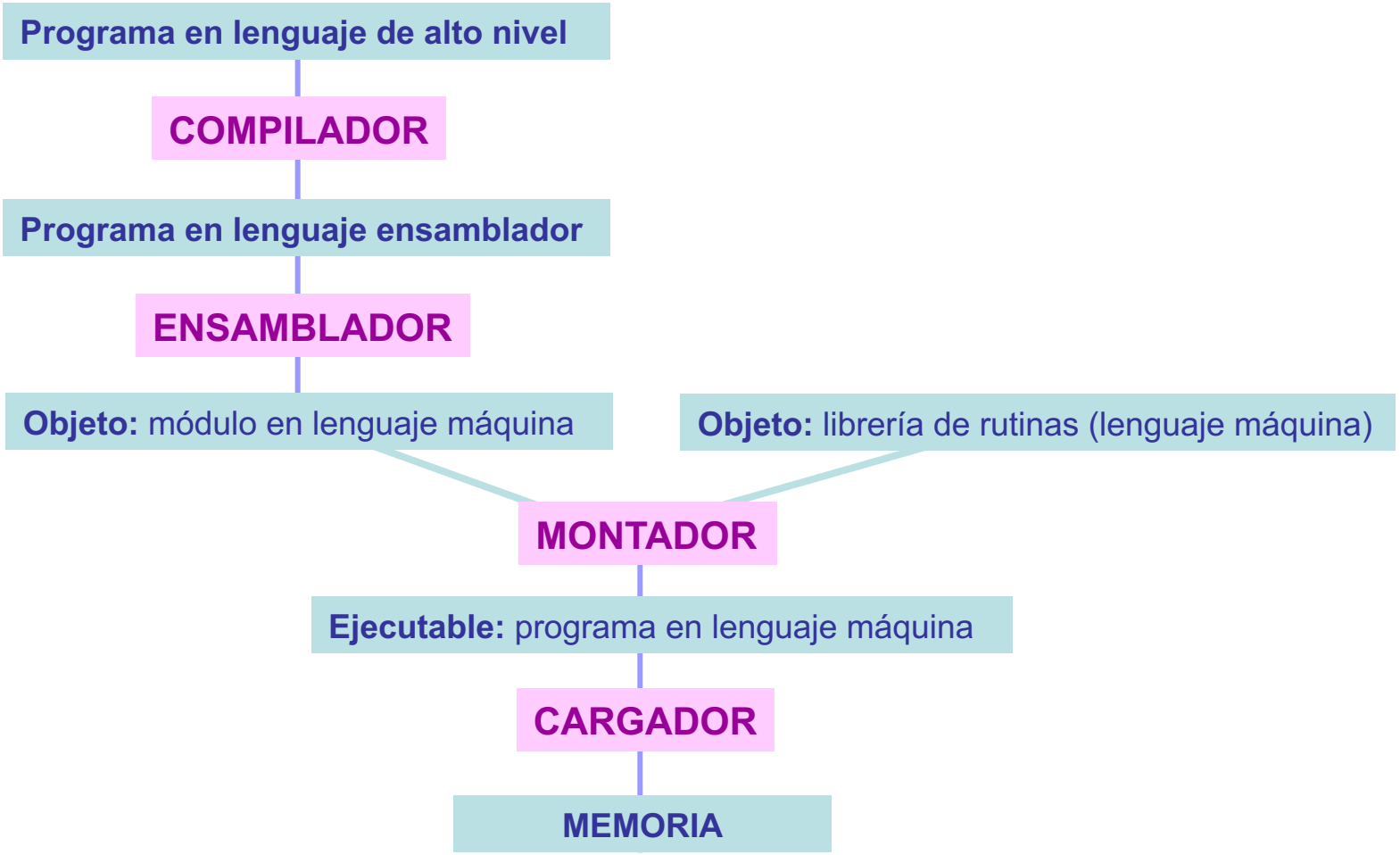
La dirección de salto se calcula como la concatenación de los 26 bits más significativos de la instrucción con los bits superiores del PC



Los compiladores y su papel dentro de ISA:

- Partiendo de la idea de que programar es usar un lenguaje de alto nivel, el nivel ISA es la máquina destino de un programa de alto nivel compilado.
- Diseño de un compilador ←→ diseño ISA.
  - Claves en el diseño e implementación de un repertorio de instrucciones.
  - Afecta a la calidad del código generado, y a la posibilidad de construir un buen compilador.
- Factores que pueden ayudar:
  - Diseño de instrucciones ortogonales.
  - No soportar estructuras específicas de los lenguajes de alto nivel.
  - Coste y rendimiento de las secuencias alternativas al código utilizado.
  - Diseño de instrucciones utilizables en contextos amplios.

Los compiladores y su papel:



TIPOS DE ARQUITECTURAS; ARQUITECTURA CISC:

- **CISC: Complex Instruction Set Computer**
- **Motivación:** salto semático
  - En los años 60's y 70's:
    - **Lenguajes de Alto Nivel** más potentes y complejos.
    - Se acrecienta la diferencia entre las operaciones proporcionadas por los lenguajes de alto nivel (L.A.N.) y operaciones del nivel ISA.
    - **Consecuencias:**
      - Tamaño excesivo del programa en lenguaje máquina.
      - Ineficiencia de la ejecución.
      - Complejidad de los compiladores.
    - Respuesta (años 70): arquitecturas con operaciones ISA muy complejas **(arquitecturas CISC)**.

TIPOS DE ARQUITECTURAS; ARQUITECTURA CISC:

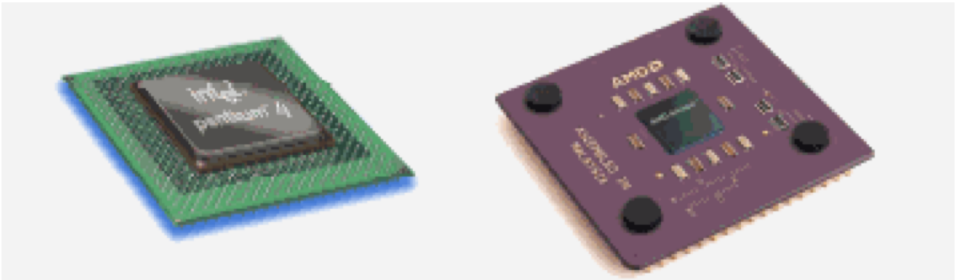
- **Objetivos:**
  - Compiladores sencillos y alta densidad de código (reducir el salto semántico).
- **Características típicas:**
  - Repertorios de instrucciones grandes.
  - Incluyen instrucciones complejas (algunas implementan directamente instrucciones LAN).
  - Unidad de control: microprogramada
    - Microprogramación: implementar una instrucción máquina mediante un conjunto de microinstrucciones, almacenadas en una memoria de control e interpretadas.
  - Formatos de instrucciones de longitud variable.
  - Muchos modos de direccionamiento (decenas).
  - Incluyen tipos de datos complejos.

TIPOS DE ARQUITECTURAS; ARQUITECTURA CISC:

Ventajas:

- Un amplio juego de instrucciones:
  - Facilita el diseño de compiladores.
  - Reduce el tamaño de los programas.
- Puede disponerse de un amplio juego de instrucciones con un coste reducido, debido a:
  - Tecnología de microprogramación.
  - Decreciente precio de las memorias.

- **Ejemplos:**
  - IBM370/168 (1973)
  - VAX 11/780 (1978)
  - Intel 80486 (1989) y todos los x86



TIPOS DE ARQUITECTURAS; ARQUITECTURA RISC:

- **RISC: Reduced Instruction Set Computer**
- **Motivación:**
  - Estudios estadísticos del comportamiento de la ejecución de los programas escritos en LAN's.:
    - Instrucciones más frecuentes:
      - Transferencias de datos
      - Control de flujo (if, loop)
      - Instrucciones aritméticas y lógicas simples
    - Instrucciones y modos de direccionamiento complejos: usados con muy poca frecuencia.
  - Resultado: arquitectura ISA compleja (más instrucciones máquina y más complejas) → Intérprete: microprograma complejo → implementación más lenta.
  - Microprogramación: difícil depuración y mantenimiento de microprogramas, alto coste.
- **Respuesta (años 80's):** arquitecturas con operaciones, modos de direccionamiento y tipos de datos simples.
  - Implementación más eficiente **(arquitecturas RISC)**

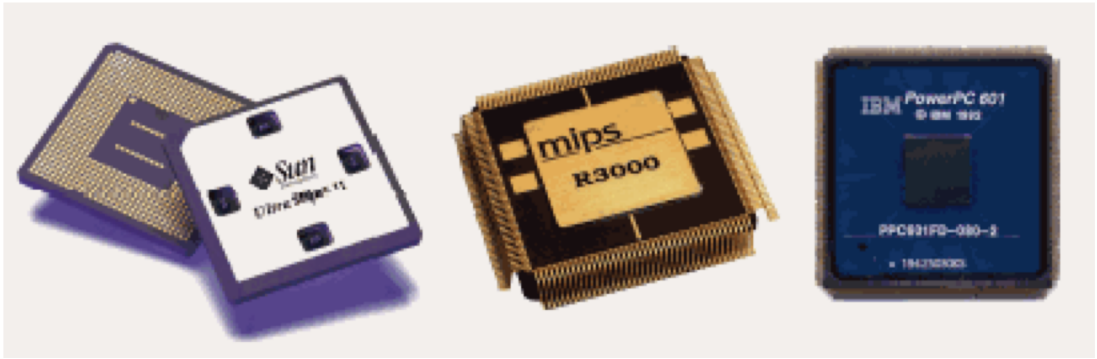
TIPOS DE ARQUITECTURAS; ARQUITECTURA RISC:

- **Características típicas:**
  - Repertorios de instrucciones limitado y sencillo. Una instrucción por ciclo.
  - Arquitecturas de carga/almacenamiento (registro - registro):
    - Simplifica el repertorio de instrucciones → la Unidad de Control.
    - Optimización por uso de registros (operandos frecuentemente accedidos).
  - Gran número de registros de uso general, o uso de tecnología de compiladores para optimizar el uso de registros.
  - Modos de direccionamiento sencillos.
  - Tipos de datos simples.
  - Énfasis en la optimización de la segmentación de instrucciones:
    - Simplicidad de la decodificación de las instrucciones.
    - Instrucción de formato único.
  - Unidad de control "cableada", en vez de microprogramada:
    - Instrucciones simples ejecutadas por una unidad de control directamente en HW → más rápido que implementación CISC microprogramada.
  - Compiladores optimizados, capaces de cubrir el salto semántico.

TIPOS DE ARQUITECTURAS; ARQUITECTURA RISC:

- **Especificaciones técnicas:**
  - Tamaño de instrucción único.
  - Pequeño número de modos de direccionamiento (<5).
  - No se usa direccionamiento indirecto → 2 accesos a memoria para obtener un operando.
  - No hay operaciones que combinen carga/almacenamiento con cálculos aritméticos/lógicos.
  - No hay más de 1 operando de memoria por instrucción.
  - Gran número de registros de uso general, o uso de tecnología de compiladores para optimizar el uso de registros .
  - Unidad de control “cableada”.

- **Ejemplos:**
  - RISC-I y RISC-II (1980)
  - MIPS R2000 (1986)
  - SUN SPARC (1987)
  - MIPS R4000 (1991)



TIPOS DE ARQUITECTURAS; ARQUITECTURA RISC:

- **Ejemplos modernos:**
  - MIPS Technologies Inc., en la mayoría de las Silicon Graphics hasta 2006, y que estuvo en las consolas Nintendo 64, PlayStation y PlayStation 2. Actualmente en la PSP y en algunos routers.
  - La serie IBM POWER, utilizada por IBM en Servidores y superordenadores
  - La versión PowerPC de Motorola e IBM, utilizada en los AmigaOne, Apple Macintosh (Imac, eMac, PowerMac y posteriores hasta 2006) Actualmente se utiliza en muchos sistemas empotrados en automóviles, routers, etc, así como en consolas como la PlayStation 3, Xbox 360 y Wii
  - SPARC y UltraSPARC de Sun Microsystems y Fijitsu, que se encuentra en sus últimos modelos de servidores (y hasta 2008 también en estaciones de trabajo)
  - El PA-RISC y el HP/PA de Hewlett-Packard (descatalogados)
  - El DEC Alpha en servidores HP AlphaServer y estaciones de trabajo AlphaStation (descatalogados)
  - El ARM – dominan en PALM, Nintendo DS, Game Boy Advance, PDA's, Apple Ipods, Apple Iphone, iPod Touch, Apple Ipad, Nintendo DS,…
  - El Atmel AVR usado en muchos productos (desde mandos de la Xbox a los coches de la BMW)
  - Procesadores XAP en muchos chips wireless de poco consumo (Bluetooth, wifi,….)

VENTAJAS DE LA ARQUITECTURA RISC:

- **Ventajas cualitativas:**
  - Simplicidad de instrucciones → compiladores generan códigos más eficientes .
  - Unidad de control HW construida expresamente para instrucciones simples → alta eficiencia.
  - Segmentación: puede aplicarse más eficazmente con un repertorio reducido de instrucciones simples.
  - Gestión más eficiente de las interrupciones (instrucciones más elementales).
  - Procesador más simple → diseño más sencillo del chip del procesador.
  - Gran número de registros de uso general, o uso de tecnología de compiladores para optimizar el uso de registros .
  - Unidad de control "cableada".
- **Ventajas cuantitativas:**
  - Comparación prestaciones y coste de procesadores CISC – RISC difícil:
    - No hay pareja de máquinas CISC – RISC comparables (coste, ciclo de vida, nivel de tecnología, complejidad, compilador, SO, etc.
    - No existe conjunto de programas de prueba definitivo (prestaciones varía según programa).
    - Difícil separar los efectos de HW de los debidos a la calidad del compilador.

RISC vs CISC:

Diseños RISC sacan provecho de algunas características CISC.
Los diseños CISC sacan provecho de algunas características CISC.

Diseños RISC recientes no son RISC puros.
Diseños CISC recientes no son CISC puros.

Características	CISC	RISC
Densidad de código	alta (NI menor)	baja (NI mayor, pero RAMS cada vez más baratas)
Coste	alto	bajo
CPU microprogramada	SI	NO
Formatos	Longitud variable	Longitud fija (32 bits)
Modos de direccionamiento	Muchos (combinados)	pocos (< 5)
Máquinas tipo Load/Store	NO	SI
Un sólo acceso a memoria por instrucción	NO	SI
Nº registros enteros y PF	< 32 (enteros) <16 (FP)	>= 32 (enteros) <=16 (FP)
Segmentación	Poco eficiente (*)	Muy eficiente

Resumen de los objetivos de ambas tecnologías:

- RISC supone una ruptura fundamental con la filosofía de diseño de computadores que siempre había existido.

Objetivo RISC: mejorar el rendimiento mediante la implementación de una **segmentación eficiente** apoyada por una **tecnología** de **diseño de compiladores** más sofisticada (conseguir **CPI → 1**)

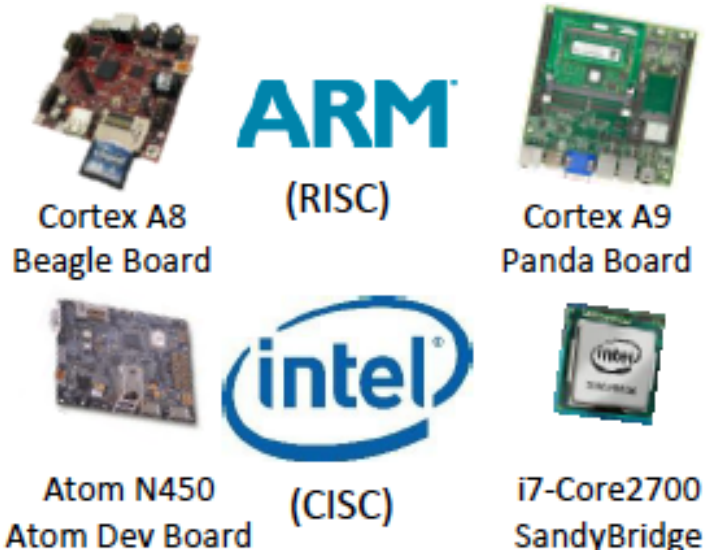
Objetivo CISC: reducir la **brecha semántica** existente entre los lenguajes de alto nivel y el lenguaje máquina **y simplificar así el diseño de compiladores.**

Table 13.8 Characteristics of Some Processors

Processor	Number of instruction sizes	Max instruction size in bytes	Number of addressing modes	Indirect addressing	Load/store combined with arithmetic	Max number of memory operands	Unaligned addressing allowed	Max Number of MMU uses	Number of bits for integer register specifier	Number of bits for FP register specifier
AMD29000	1	4	1	no	no	1	no	1	8	3[a]
MIPS R2000	1	4	1	no	no	1	no	1	5	4
SPARC	1	4	2	no	no	1	no	1	5	4
MC88000	1	4	3	no	no	1	no	1	5	4
HP PA	1	4	10[a]	no	no	1	no	1	5	4
IBM RT/PC	2[a]	4	1	no	no	1	no	1	4[a]	3[a]
IBM RS/6000	1	4	4	no	no	1	yes	1	5	5
Intel i860	1	4	4	no	no	1	no	1	5	4
IBM 3090	4	8	2[a]	no[a]	yes	2	yes	4	4	2
Intel 80486	12	12	15	no[a]	yes	2	yes	4	3	3
NSC 32016	21	21	23	yes	yes	2	yes	4	3	3
MC68040	11	22	44	yes	yes	2	yes	8	4	3
VAX	56	56	22	yes	yes	6	yes	24	4	0
Clipper	4[a]	8[a]	9[a]	no	no	1	0	2	4[a]	3[a]
Intel 80960	2[a]	8[a]	9[a]	no	no	1	yes[a]	—	5	3[a]

a RISC that does not conform to this characteristic.
b CISC that does not conform to this characteristic.

What's next:



"Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures"

19th IEEE International Symposium on High Performance Computer Architecture (HPCA 2013)