INSTRUCCIONES Y REPERTORIOS

Principios de Computadores

1. **Formatos y codificación de instrucciones**
2. **Criterios de diseño de formatos de instrucciones**
3. **Operandos y número de direcciones**
4. **Tipos de instrucciones**
5. **Propiedades de los repertorios de instrucciones. RISC frente a CISC**

1. Formatos y codificación de instrucciones

1. Formatos y codificación de instrucciones

- El propósito de una **instrucción** es *especificar una operación a realizar y los operandos que deben ser utilizados.*

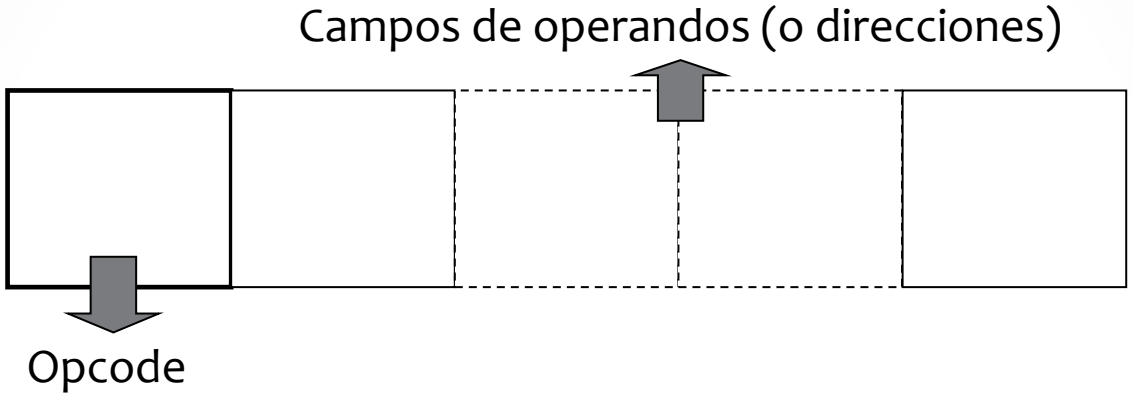
- Tanto la operación como los operandos son *descritos en campos específicos de la palabra de instrucción.*

1) La operación se especifica en un el **campo OPCODE** (**código de operación**).

2) El **campo de operando** contiene la *dirección de la posición de memoria principal o el registro del procesador que contiene al operando.*

1. Formatos y codificación de instrucciones

- Formato típico:



- Las instrucciones pueden ser de longitud menor, igual o mayor que la longitud de palabra.

- En algunas máquinas de nivel 2 todas las instrucciones tienen la misma longitud. En otras puede haber 2 o 3 longitudes distintas.

1. Formatos y codificación de instrucciones MIPS

En MIPS, sólo hay tres formatos distintos de instrucciones:

1) Instrucciones tipo R (instrucciones aritméticas y lógicas):

Op - 6	Rs - 5	Rt - 5	Rd - 5	shant - 5	funct - 6

2) Instrucciones tipo I (transferencia de datos y aritméticas inmediatas):

Op - 6	Rs - 5	Rt - 5	Dirección - 16

3) Instrucciones tipo J (salto incondicional):

Op - 6	Dirección - 26

2. Criterios de diseño de formatos de instrucciones

2. Criterios de diseño de formatos de instrucciones

1. **Longitud de la instrucción (nº de bits):**

   ¿Las instrucciones cortas son mejor que las largas?

   Ej: un programa de n instrucciones de 16 bits ocupa menos espacio en memoria que un programa de n instrucciones de 32 bits. ¿Se optimiza la memoria?

2. **Velocidad de transferencia de la memoria** (caudal = nº bits/seg que pueden ser leídos de ella):

   ¿Se leen/procesan más rápido las instrucciones largas o cortas? ¿la velocidad de ejecución de las instrucciones depende de su longitud?

2. Criterios de diseño de formatos de instrucciones

3. **Longitud del opcode (nº de bits):** ¿Cuántos bits se necesitan en el campo opcode? Es necesario dejar suficiente espacio en la instrucción para expresar todas las operaciones deseadas.

   Ej: si la máquina tiene un opcode de longitud n bits, será capaz de hacer 2^n operaciones.

4. **Nº de palabras que ocupa la instrucción**: es conveniente que la instrucción ocupe un número entero de palabras o que un número entero de instrucciones ocupe una palabra.

2. Criterios de diseño de formatos de instrucciones

5. **Número de operandos (direcciones):** ¿Cuántos operandos conviene incluir en una instrucción?

   Menos operandos: instrucciones más cortas, más primitivas, mayor longitud del programa (mayor tiempo de ejecución) pero requieren circuitos de decodificación más simples.

   Más operandos: requieren circuitos de decodificación y procesamiento más complejos.

3. Operandos y número de direcciones

3. Operandos y número de direcciones

Máquina de **4 direcciones**

ADD A B C D

Sumar el contenido de la dirección especificada en A con el contenido de la dirección especificada en B, poner el resultado en la dirección especificada en C y encontrar la siguiente instrucción en la dirección especificada en D.

Código de operación	Dirección del operando A	Dirección del operando B	Dirección del resultado C	Dirección de la siguiente instrucción

¿Problema?

3. Operandos y número de direcciones

Máquina de **4 direcciones**

ADD A B C D

La instrucción es demasiado larga. Ej: si la memoria principal tiene 2^{15} celdas, 4 campos de dirección requieren 60 bits, más los bits del opcode -> INVIABLE

SOLUCIÓN: eliminar el campo que especifica la dirección de la siguiente instrucción.

Código de operación	Dirección del operando A	Dirección del operando B	Dirección del resultado C	Dirección de la siguiente instrucción

3. Operandos y número de direcciones

Máquina de **3 direcciones**

ADD A B C

Sumar el contenido de la dirección especificada en A con el contenido de la dirección especificada en B y poner el resultado en la dirección especificada en C.

Ej:
Honeywell 8200, CDC6600 , MIPS32

Código de operación	Dirección del operando A	Dirección del operando B	Dirección del resultado C

¿Cómo sabemos ahora donde está la siguiente instrucción?

3. Operandos y número de direcciones

Máquina de **3 direcciones**

Por ejemplo, en MIPS:

```
add $s0, $s1, $s2

sub $s0, $s1, $s2
```

Atención:
Ponemos como ejemplo una instrucción de MIPS de tres operandos, pero hay que tener en cuenta que los operandos no siempre son registros.

En general, los datos estarán en la memoria del ordenador y los operandos serán direcciones de memoria

3. Operandos y número de direcciones

Máquina de **2 direcciones**

ADD A B

Sumar el contenido de la dirección especificada en A con el contenido de la dirección especificada en B y poner el resultado en la dirección B (destruye uno de los sumandos).

Ej: PDP-11, IBM 1620, IBM 370

Código de operación	Dirección del operando A	Dirección del operando B

3. Operandos y número de direcciones

Máquina de **2 direcciones** (variante 1)

ADD A B

Sumar el contenido de la dirección especificada en A con el contenido de la dirección especificada en B y poner el resultado en el **acumulador** (registro de la ALU). No se destruye uno de los sumandos).

Código de operación	Dirección del operando A	Dirección del operando B

3. Operandos y número de direcciones

Máquina de **2 direcciones**

Por ejemplo, en MIPS:

- mult rs, rt

El resultado se guarda en (HI, LO).
Está implícito, ya que no se indica en la instrucción.

Atención:
Ponemos como ejemplo una instrucción de MIPS de dos operandos, pero hay que tener en cuenta que los operandos no siempre son registros.

En general, los datos estarán en la memoria del ordenador y los operandos serán direcciones de memoria

3. Operandos y número de direcciones

Máquina de **2 direcciones** (variante 2)

La primera dirección es la localización del operando y la segunda es la dirección de la próxima instrucción. Esta forma se ha utilizado muy poco en la historia, porque la fórmula de que las instrucciones se ejecuten secuencialmente es la preponderante.

Ej: IBM 650

Código de operación	Dirección del operando A	Dirección de la siguiente instrucción

3. Operandos y número de direcciones

Máquina de **1 dirección**

La única dirección es la localización del operando A.

Código de operación	Dirección del operando A

¿Cómo sabemos ahora donde está la siguiente instrucción?

¿Cómo sabemos ahora donde está el otro operando?

3. Operandos y número de direcciones

Máquina de 1 dirección

Por ejemplo, en MIPS:

jr $st5

Atención:
Ponemos como ejemplo una instrucción de MIPS de un operando, pero hay que tener en cuenta que los operandos no siempre son registros.

En general, los datos estarán en la memoria del ordenador y los operandos serán direcciones de memoria

3. Operandos y número de direcciones

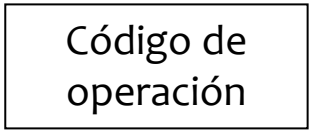
Máquina de **0 direcciones**

No hay un tamaño de instrucción más pequeño. También se llaman "máquinas de stack" ("de pila").

Todas las direcciones están implícitas.

- La localización de la siguiente instrucción está en el PC (contador de programa).
- Los operandos están en la el tope de la pila.
- El resultado se coloca en el tope de dicha pila.

Ej: KDF-9, B5500



3. Operandos y número de direcciones

- Hay 3 tipos de operandos:

1) **Registro del procesador**: cada registro tiene asociada una determinada posición (normalmente un número entero). Se denotan con "R" y el número de registro. Ej: R2 es el segundo registro. En MIPS se escribiría así: $t2.

2) **Posición de memoria:** se suele denotar con el prefijo 0x seguido de la posición de memoria hexadecimal. Ej: 0x0F41 es la posición en binario 0000111101000001 (3905 en decimal).

3) **Dispositivo de E/S:** cada dispositivo de E/S tiene un espacio de memoria para almacenar los datos transferidos con la CPU. Dependiendo del tipo de dispositivo, este espacio de almacenamiento puede tener un identificador asociado, o se puede referenciar como una posición del espacio de memoria principal.

4. Tipos de instrucciones

4. Tipos de instrucciones

¿Qué requerimientos debe satisfacer un juego de instrucciones de lenguaje máquina?

1. Debe ser **COMPLETO**: capaz de construir un programa en lenguaje máquina que evalúe cualquier función computable usando una cantidad de memoria razonable.
2. Debe ser **EFICIENTE**: las funciones requeridas más frecuentemente se deben ejecutar rápidamente usando pocas instrucciones.
3. Debe ser **REGULAR**: debe contener códigos de operación y modos de direccionamiento esperados.
4. Debe ser **COMPATIBLE**: con los juegos de instrucciones de máquinas ya existentes.

4. Tipos de instrucciones

- Las instrucciones de nivel de máquina convencional (nivel de lenguaje máquina) se dividen en dos:

1) Instrucciones de propósito general.
2) Instrucciones de propósito especial.

- Veremos las instrucciones de propósito general más importantes:

1) Movimiento de datos
2) Operaciones con dos operandos
3) Operaciones con un operando
4) Comparaciones y saltos condicionales
5) Control de iteraciones
6) Entrada/Salida

1. Instrucciones de movimiento de datos

- Son las instrucciones más fundamentales.

- Mover datos de un lugar a otro = hacer una copia (creación de un nuevo objeto con la misma secuencia de bits que el original).
¡Atención!: MOVER = HACER COPIA. El objeto original que se copia sigue existiendo en su sitio original.

- Los datos se almacenan en:
  - **1) memoria principal**
  - **2) registro**
  - **3) pila (implementada en registros especiales o en memoria)**

- Se accede a los datos de modo distinto dependiendo de dónde estén guardados.
Ej: un acceso a memoria requiere la especificación de una dirección, el acceso a una pila no requiere direccionamiento explícito.

- Estas instrucciones deben indicar la cantidad de datos que se va a mover (1 bit, una palabra, media palabra, etc).

1. Instrucciones de movimiento de datos

Por ejemplo, en MIPS:

```
lw $st0, 400($s1)

sw $st0, 400($s1)
```

2. Instrucciones para realizar operaciones con dos operandos (operaciones binarias)

- Combinan dos operandos para obtener un resultado. Dos tipos:

1) **Instrucciones aritméticas** (suma, resta, multiplicación, división)

2) **Instrucciones booleanas o lógicas**. Existen 16 funciones booleanas distintas de dos variables, pero las máquinas no tienen instrucciones para todas. Las más comunes: AND, OR, XOR.

- Ej: AND se calcula bit a bit entre dos palabras dando como resultado una nueva palabra. A veces también hay instrucciones para medias palabras, dobles palabras, etc.

- Uso común de la AND: extracción bits de las palabras.

Uso común de la OR: empaquetamiento de bits en una palabra (acción contraria a la anterior).

2. Instrucciones para realizar operaciones con dos operandos (operaciones binarias)

Por ejemplo, en MIPS:

```
add $s0, $s1, $s2
sub $s0, $s1, $s2

and $s0, $s1, $s2
or $s0, $s1, $s2
```

3. Instrucciones para realizar operaciones con un operando (operaciones unarias)

- Operaciones que aceptan un operando y devuelven un resultado. Como estas instrucciones tienen una dirección menos que especificar a veces son más cortas que las instrucciones de dos operandos.

- Operaciones unarias básicas: DESPLAZAMIENTOS y ROTACIONES.

- **DESPLAZAMIENTO**: movimiento de los bits de una palabra hacia la izquierda o hacia la derecha. Se pierden los bits que salen de la palabra (*desplazamiento lineal*).

- **ROTACIÓN**: movimiento de los bits de una palabra hacia la izquierda o hacia la derecha. Los bits que salen de la palabra por un lado aparecen por el otro (*desplazamiento circular*).

3. Instrucciones para realizar operaciones con un operando (operaciones unarias)

- Los desplazamientos se pueden hacer sin o con extensión de signo.

- Un uso importante del desplazamiento es la multiplicación y división de potencias de 2:

- Un entero positivo se desplaza k bits a la izquierda: el resultado es el nº original x 2^k

- Un entero positivo se desplaza k bits a la derecha: el resultado es el nº original x 2^{-k} (dividido por 2^k).

¡Atención! Esto no se cumple con los números negativos:
- Desplazar a la izquierda en C1 no multiplica por 2, pero desplazar a la derecha sí se corresponde a dividir por 2
- Desplazar a la derecha en C2 no divide por 2, pero desplazar a la izquierda sí se corresponde a multiplicar por 2.

3. Instrucciones para realizar operaciones con un operando (operaciones unarias)

- Los desplazamientos son útiles para empaquetamiento y extracción de bits en una palabra.

- Todas las máquinas tienen una amplia variedad de instrucciones de desplazamiento/rotación en ambas direcciones.

- Ciertas operaciones de dos operandos ocurren con tanta frecuencia con el mismo operando que muchas máquinas tienen instrucciones unarias que las hacen rápidamente:

- Mover un 0 a una palabra de memoria o a un registro al comienzo de un cálculo: con una instrucción "borrar" que acepta una única dirección (la de la posición a "borrar", es decir, a poner a 0).
- Sumar 1 a un registro o a una posición de memoria, para contar: con una operación unaria (INC, DEC)
- Operación "negar": hacer la resta binaria 0-X = -X

3. Instrucciones para realizar operaciones con un operando (operaciones unarias)

Por ejemplo, en MIPS:

```
sll $st0, $st1, 8

srl $st0, $st1, 8
```

4. Instrucciones para realizar comparaciones y saltos condicionales

- Estas instrucciones permiten al programa examinar datos y alterar el flujo de control en función de los resultados. Ej: raíz cuadrada, si el argumento es negativo se debe dar un mensaje de error, y si no, se debe realizar la operación.

- **Las instrucciones de salto condicional** (o de bifurcación) comprueban alguna condición y saltan a una dirección determinada si se cumple la condición.

- Cuando se cumple una condición, normalmente se salta o se omite la siguiente instrucción. Se pueden saltar varias instrucciones en vez de una sola (el número de instrucciones a saltar viene especificado en la propia instrucción).

- El ***salto incondicional*** es un caso especial del *salto condicional* en el que la condición siempre se cumple.

4. Instrucciones para realizar comparaciones y saltos condicionales

- Ejemplos:

1) La condición más normal es verificar si un bit particular es 0 o no:

- Bit de signo de un número en C2: "*salta a LUGAR si es 1*" (si el número es negativo)
- "*salta a LUGAR si el bit es 0*". Esto es problemático si se usa C1 (hay +0 y -0), y fácil si se usa C2 (un único código para el 0).

2) Instrucción para comprobar el último bit de la derecha (saber si el número es par o impar)

3) Comprobación de si una palabra está a 0 (iteraciones, etc): es necesario comprobar todos los bits de la palabra (con una OR).

4) Comparación de dos palabras para ver si son iguales o cuál es la mayor (ordenamiento).

4. Instrucciones para realizar comparaciones y saltos condicionales

Por ejemplo, en MIPS:

```
j 1000

jr $sra
```

5. Instrucciones de control de iteraciones

- Estas instrucciones facilitan la ejecución de un grupo de instrucciones un número fijo de veces (número de iteraciones).

- Se requiere de un contador que se incremente o decremente en alguna constante cada vez que pasa por una iteración. Si se verifica alguna condición de parada, la iteración termina.

- Método de ***comprobación al final***: se inicializa el contador, se ejecuta el código de la iteración, se actualiza el contador, y si la condición de parada no se cumple se salta a la 1ª instrucción del lazo.

- Método de ***comprobación al inicio***: se inicializa el contador, si la condición de parada no se cumple se ejecuta la iteración, la última instrucción hace el salto a la nueva comprobación de la condición de parada.

6. Instrucciones de entrada/salida

- Las instrucciones de entrada/salida son simples instrucciones de carga o almacenamiento entre CPU y el puerto de E/S (similares a las de lectura/escritura en memoria). Posibilidades:

1) Parte de las direcciones de la memoria principal están asignadas a los puertos de E/S (E/S mapeada en memoria): **INSTRUCCIONES DE E/S IMPLÍCITAS** (las mismas que se usan para escribir y leer en memoria).

2) Si el espacio de direcciones de memoria y el espacio de direcciones de E/S están separadas: **INSTRUCCIONES DE E/S EXPLÍCITAS** (instrucciones específicas). Ej:
- IN X: una palabra se transmite desde el puerto de E/S X al acumulador (registro de la ALU, dentro de la CPU).
- OUT X: una palabra se transmite desde el acumulador al puerto de E/S X.

5. Propiedades de los repertorios de instrucciones. RISC frente a CISC

5. Propiedades de los repertorios de instrucciones. RISC frente a CISC

- No hay acuerdo general sobre cual es el tamaño adecuado para un juego de instrucciones de propósito general.

- Dos tendencias:

1) **RISC = Reduced Instruction Set Computers**: instrucciones sencillas que requieren la ejecución de un número pequeño de pasos básicos. Para llevar a cabo una tarea se necesitan muchas de estas instrucciones sencillas. Ej: ARM.

2) **CISC = Complex Instruction Set Computers**: instrucciones complejas que implican un gran número de pasos. Para hacer una tarea se necesitan menos instrucciones. Ej: Motorola, Intel.

5. Propiedades de los repertorios de instrucciones. RISC frente a CISC

- **RISC**: Se necesitan más instrucciones para ejecutar una tarea determinada. Pero las instrucciones están más adaptadas al hardware de la máquina y proporcionan velocidades de ejecución elevadas.

- **CISC**: Se necesitan menos instrucciones para ejecutar una tarea determinada. Pero la ejecución de estas instrucciones es más compleja. Aunque se necesiten menos instrucciones, éstas tardan más en ejecutarse.

5. Propiedades de los repertorios de instrucciones. RISC frente a CISC

- Características RISC:

- **Pocos tipos** de instrucciones y de modos de direccionamiento.

- Formatos de instrucción **fijos y fácilmente decodificables**

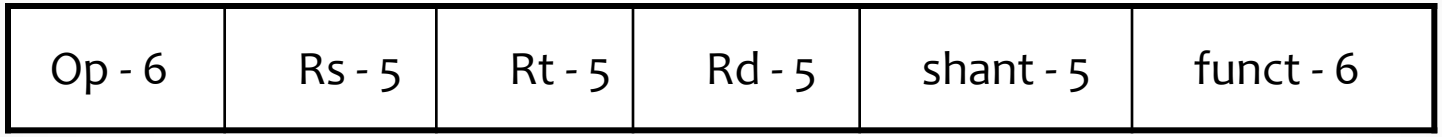
- **Ejecución rápida de instrucciones** en un solo ciclo: el tamaño pequeño y la regularidad del juego de instrucciones simplifica el diseño de la unidad de control y facilita la ejecución en un solo ciclo.

- La mayoría de las instrucciones hacen operaciones de **registro a registro** internos de la CPU. Para acceder a memoria hay muy pocas.

Ejemplo de instrucciones RISC: MIPS

En MIPS, sólo hay tres formatos distintos de instrucciones:

1) Instrucciones tipo R (tipo registro):



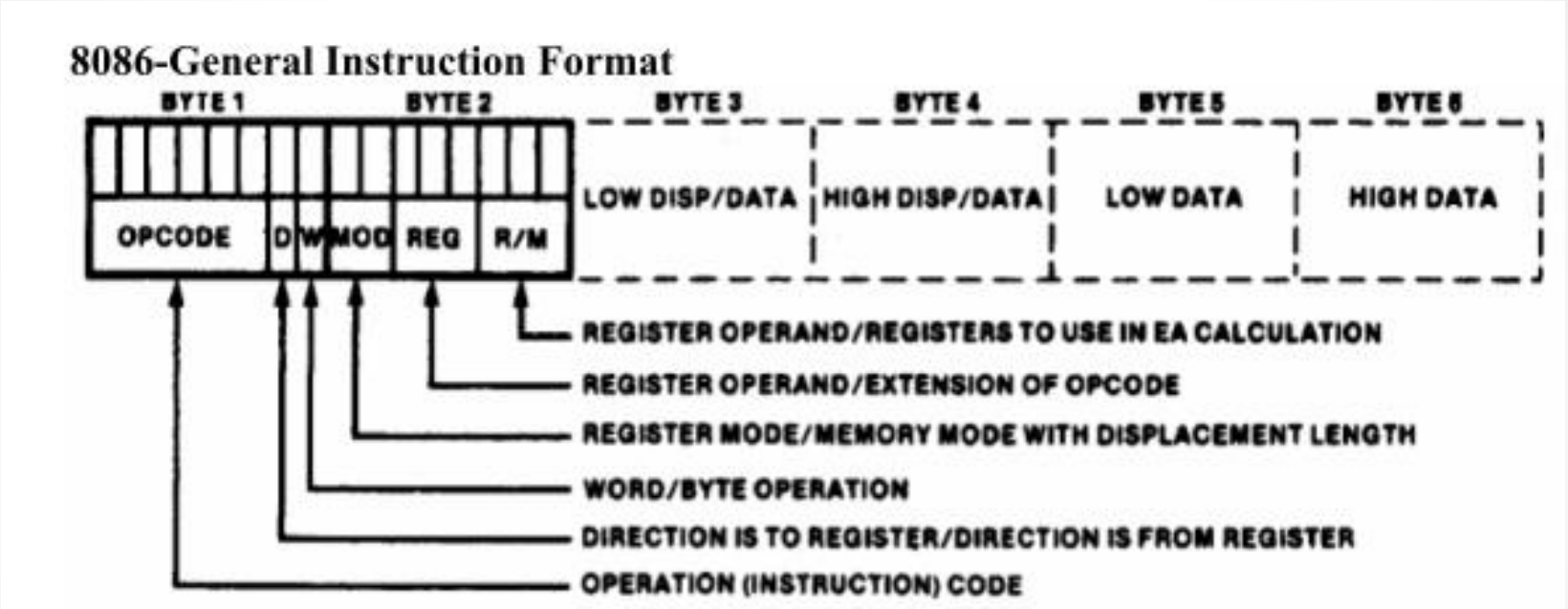
2) Instrucciones tipo I (transferencia de datos):

3) Instrucciones tipo J (salto):

Salto incondicional

Salto condicional

Ejemplo de instrucciones CISC: 8086



Ejemplo de instrucciones CISC: 8086