TEMA2. DISEÑO DEL REPERTORIO DE INSTRUCCIONES

2.1 Taxonomía de las arquitecturas a nivel lenguaje máquina.

2.2 Las máquinas de registros de propósito general (GPR).

2.3 Direccionamiento de la memoria
- a. Ordenación de los bytes y alineamiento
- b. Modos de direccionamiento e índices de utilización.
- c. Modo de direccionamiento desplazamiento
- d. Modo de direccionamiento inmediato
- e. Codificación de los modos de direccionamiento

2.4 Repertorio de instrucciones
- a. Tipos de operaciones
- b. Repertorio de instrucciones M-M frente a R-R
- c. Instrucciones de control

2.5 Tipo y tamaño de los operandos

2.6 La arquitectura como objeto del compilador

2.7 Algunos repertorios de instrucciones

TEMA 2. DISEÑO DEL REPERTORIO DE INSTRUCCIONES

2.1. Taxonomía de las arquitecturas a nivel lenguaje máquina.

Clasificación de los repertorios de instrucciones atendiendo a diferentes criterios:

Almacenamiento de operandos en la CPU

Se refiere al lugar de almacenamiento de los operandos en la CPU (además de en memoria). Es el factor distintivo más importante entre las arquitecturas a nivel lenguaje máquina. Todas las arquitectura, prácticamente, tienen algún almacenamiento temporal en CPU.

Operandos explícitos

Número de operandos explícitos en las instrucciones. El número de operandos explícitos por instrucción está condicionado por el tipo de almacenamiento temporal en CPU.

Posición del operando

Posibilidad de expresar operandos de instrucciones ALU en memoria o exclusivamente en registros de la CPU. Modos de especificación de los operandos en memoria. Este es un factor importante entre los repertorios recientes.

Operaciones

Tipo de operaciones proporcionadas por el repertorio de instrucciones. Este factor tiene poca interacción con otros factores de la arquitectura.

Tipo y tamaño de los operandos

Tipo y tamaño de cada operando y modo de especificarlo. Este factor es bastante independiente de otras elecciones.

Tipo de almacenamiento interno de la CPU

Es una de las cuestiones más importantes. Las alternativas fundamentales son las siguientes:

Arquitectura de pila

Los operandos en una arquitectura de pila están implícitamente en la cima de la pila.

Arquitectura de acumulador

En una arquitectura de acumulador, un operando está implícitamente en el acumulador.

Arquitectura de registros de propósito general

Las arquitecturas de registros de propósito general tienen sólo operandos explícitos, en registros o en posiciones de memoria.

Ejemplos de almacenamiento de operandos en la CPU.

Consideramos una instrucción de dos operandos fuente y uno resultado.

Tipo de almacenamiento temporal	Ejemplos	Operandos explícitos por instrucción	Destino del resultado	Acceso a operandos explícitos
Pila	B5500, HP3000/70	0	Pila	Apilar y desapilar
Acumulador	PDP-8 Motorola 6809	1	Acumulador	Cargar/ almacenar acumulador
GPR	IBM 360 DEC VAX	2 o 3	Registros o memoria	Cargar/almacenar registros o memoria

Tres ejemplos: Secuencia de código C=A+B en las tres clases de repertorios de instrucciones.

Pila	Acumulador	Registro
Push a	Load a	Load R1, a
Push b	Add b	Add R1, b
Add	Store c	Store c, R1
Pop c		

(a, b, c están en memoria).

Tendencia actual, GPR

Las máquinas más antiguas utilizaban arquitecturas de pila y acumulador.

En la última década se han utilizado frecuentemente las arquitecturas GPR, por dos razones:

Los registros tienen acceso más rápido que la memoria

Los registros son más fáciles de utilizar por los compiladores.

Nos centraremos en las arquitecturas GPR.

2.2 Las máquinas de registro de propósito general. GPR.

La ventaja más importante de las arquitecturas GPR es la **utilización efectiva de los registros** por parte del compilador.

Los registros permiten una **ordenación más flexible** que las pilas o acumuladores a la hora de **evaluar expresiones**.

Los **registros** pueden **utilizarse** para que **contengan variables**.

Esto **reduce el tráfico de memoria y acelera el programa** (los registros son más rápidos que la memoria).

Mejora la densidad de código (un registro se nombra con menos bits que una posición de memoria).

Los escritores de compiladores prefieren que todos los **registros** sean **no reservados** para **ubicar las variables de forma más flexible**.

El **número de registros** necesario **depende** del uso del **compilador**, reservando registros para:

Evaluar expresiones.
Paso de parámetros.
Ubicar variables. Según algoritmo de ubicación utilizado.

Clasificación de las arquitecturas GPR.

Atendiendo al **número de operandos** de instrucciones ALU

Tres operandos
Un operando resultado y dos operandos fuente

Dos operandos
Un operando es fuente y destino

Atendiendo al **número de operandos** que se pueden direccionar **en memoria** en instrucciones ALU. (De cero a tres).

Registro-registro (carga almacenamiento).
Máquinas sin referencia a memoria para las instrucciones ALU. Las operaciones ALU sólo presentan operandos como registros de la CPU.

Registro memoria
Se permite un sólo operando referenciando la memoria.

Memoria memoria
Se permite más de un operando referenciando la memoria.

Ventajas y desventajas de las arquitecturas GPR

Tipo	Ventajas	Desventajas
Registro-registro	•Codificación simple, instrucciones de longitud fija. •Las instrucciones emplean números de ciclos similares para ejecutarse.	•Mayor recuento de instrucciones que las arquitecturas con referencias a memoria.
Registro-memoria	•Los datos pueden ser accedidos sin cargarlos primero.	•Se destruye un operando fuente. •Codificar un número de registro y una dirección de memoria en cada instrucción puede restringir el número de registros. •Los ciclos de instrucción varían según los operandos.
Memoria-memoria	•No se emplean registros para temporales. •Código más compacto.	•Gran variación en el tamaño de las instrucciones. •Gran variación en el trabajo por instrucción. •Los accesos a memoria crean cuellos de botella en memoria.

En general:

Máquinas R-R con **menos alternativas (más simples)**, son más fáciles de implementar pero incrementan los recuentos de instrucciones.
Máquinas R-M con **más alternativas** y formatos más complejos, incrementan la complejidad de implementación pero decrementan el recuento de instrucciones.

Ejemplos de máquinas con diferentes alternativas:

Número de direcciones de memoria por instrucción ALU	Número de operandos por instrucción ALU	Ejemplos
0	2 3	IBM RT-PC SPARC, MIPS, HP Precision Architecture
1	2 3	PDP-10, Motorola 68000, Parte del IBM 360 (instrucciones RS)
2	2 3	PDP-11, National 32x32 Parte del IBM 360 (instrucciones SS)
3	3	VAX (también tiene formatos de dos operandos)

2.3. Direccionamiento de la memoria.

Las arquitecturas deben definir cuantas direcciones de memoria son interpretadas y como se especifican.

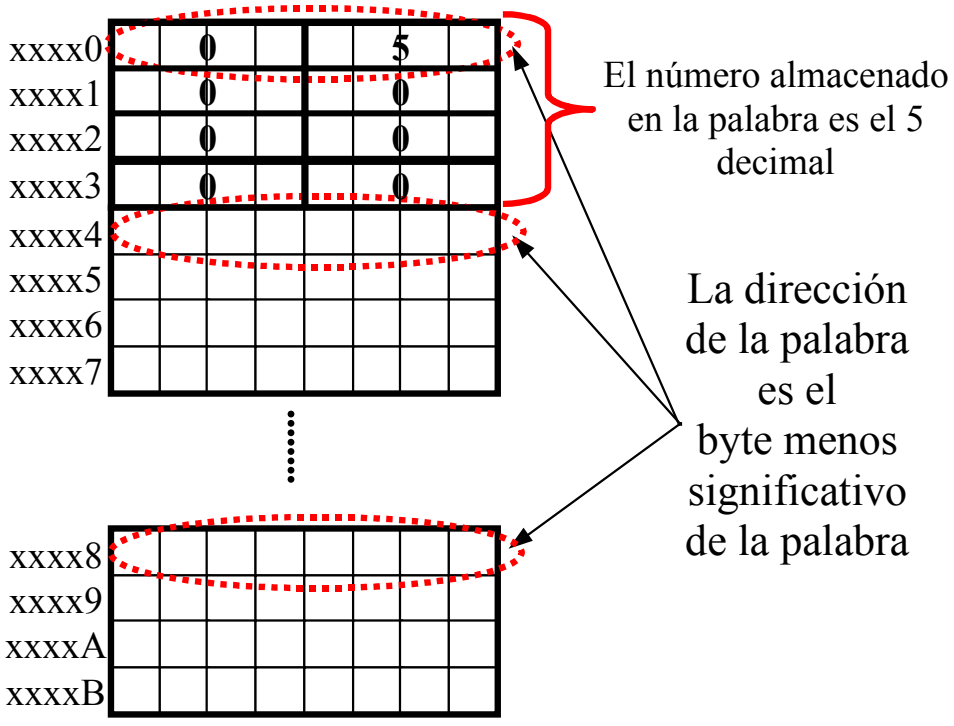
Muchas arquitecturas direccionan por bytes y proporcionan acceso a bytes (8 bits), medias palabras (16 bits), palabras (32 bits) y dobles palabras (64 bits).

a. Ordenación de los bytes y alineamiento

La ordenación de los bytes de las palabras

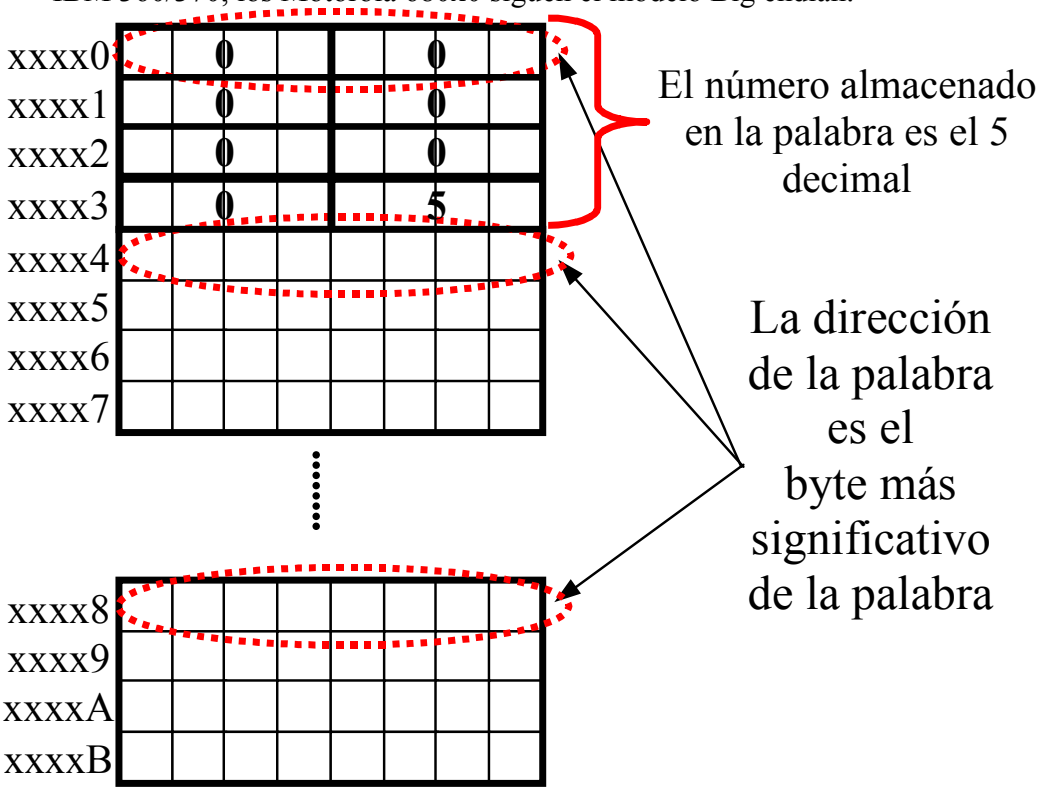
Existen dos convenios utilizados para ordenar los bytes de una palabra. Ordenación "Little Endian" y ordenación "Big Endian". Estos términos provienen de un famoso artículo de Cohen[1981] que establece una analogía entre la discusión sobre por que extremo de byte comenzar y la discusión de los Viajes de Gulliver sobre que extremo del huevo abrir.

La ordenación Little endian (extremos pequeño)
La dirección de un dato es la dirección del byte menos significativo.
DEC PDP11, VAX y 80x86 siguen el modelo Little endian.



La ordenación Big endian (extremo grande)

La dirección de un dato es la dirección del byte más significativo
IBM 360/370, los Motorola 680x0 siguen el modelo Big endian.



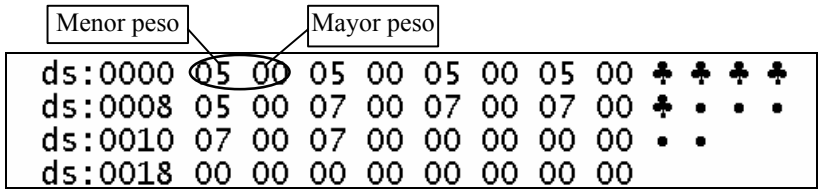
La ordenación se puede aplicar también a los bits, pero muy pocas arquitecturas proporcionan instrucciones para acceder a los bits por su dirección.
La ordenación de los bytes puede ser un problema cuando se intercambian datos entre máquinas con diferentes ordenaciones.

Por ejemplo el 80x86 tiene ordenación little endian

Reservando 5 palabras (16 bits) inicializadas a 5

```
N EQU 5
DATOS1 SEGMENT
   VECT1 DW N DUP(5)
   VECT2 DW N DUP(7)
DATOS1 ENDS
```

El segmento de datos contiene



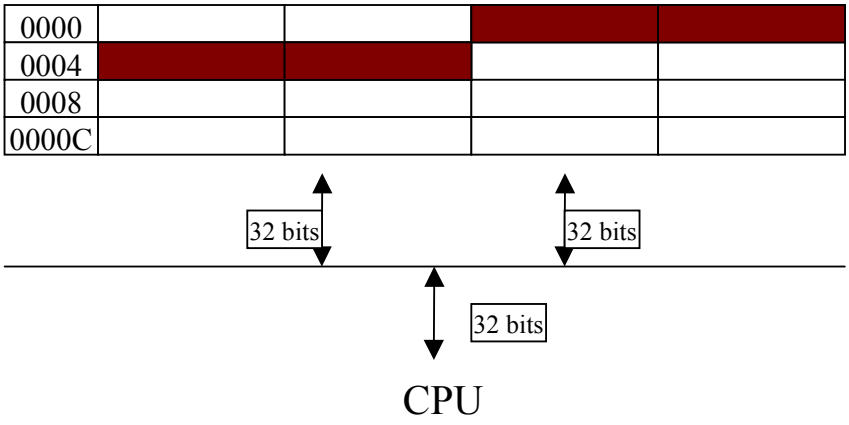
Alineamiento de los accesos a los objetos de memoria.

En algunas máquinas los accesos a los objetos mayores de un byte deben estar alineados.

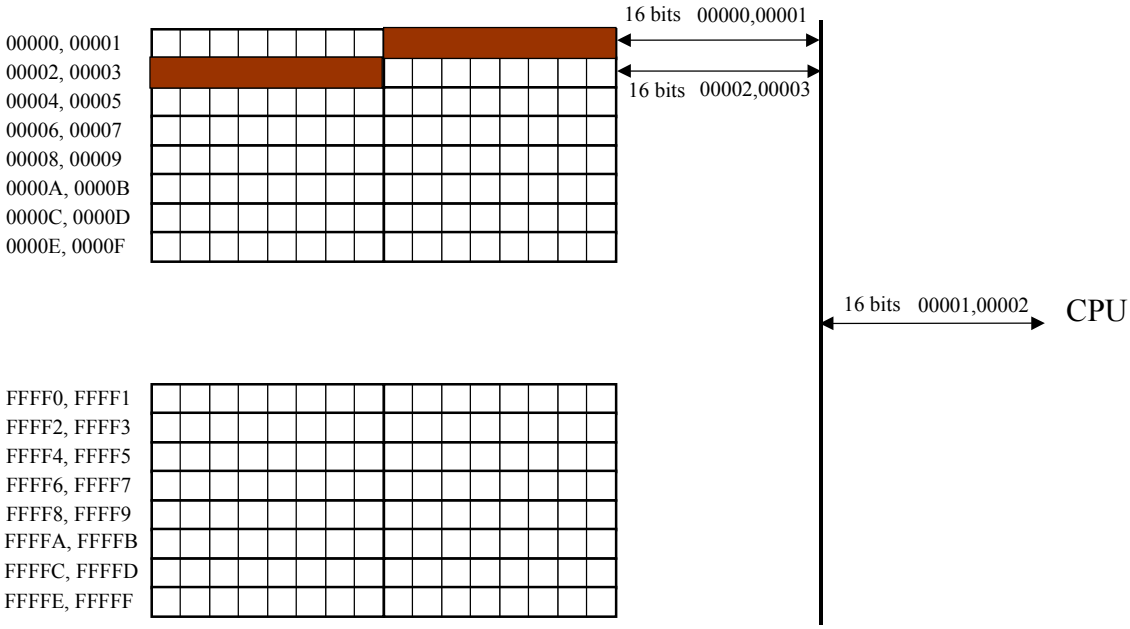
Un acceso a un objeto mayor de un byte en la dirección A y en una memoria de tamaño n bytes (ancho de palabra) en su bus de datos, esta alineado, si la dirección A mod n = 0

El acceso no alineado a los datos puede empeorar el tiempo de ejecución del programa debido a la necesidad de realizar varios accesos a memoria para completar un acceso.

Ejemplo: Que ocurre en un sistema con un bus de datos de 32 bits al acceder a una palabra no alineada.



Ejemplo: Que ocurre en el 80x86 cuando se realiza un acceso a una palabra no alineada (sistema con un bus de 16 bits a memoria).



b. Modos de direccionamiento

Los modos de direccionamiento se refieren a la forma en que las arquitecturas especifican la dirección de un objeto.

En las arquitecturas GPR un modo de direccionamiento puede especificar una constante, un registro o una posición de memoria.

En caso de ser una posición de memoria, la dirección real especificada por el modo de direccionamiento se denomina *dirección efectiva.*

Los nombres de los modos de direccionamiento de la tabla pueden diferir entre arquitecturas

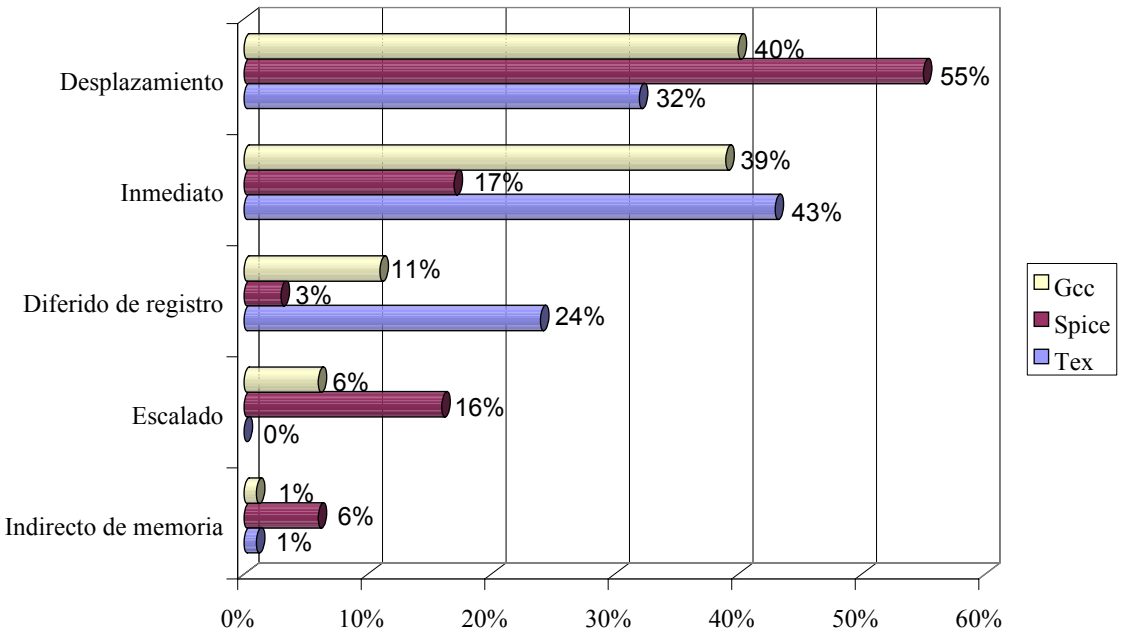
Modo de direccionamiento	**Ejemplo**	**Significado**	**Cuando se usa**
Registro	Add R4, R3	R4←R4+R3	Cuando un valor está en un registro
Inmediato o literal	Add R4, #3	R4←R4+3	Para constantes. En algunas máquinas, literal e inmediato son dos modos diferentes de direccionamiento
Desplazamiento	Add R4, 100(R1)	R4←R4+M[100+R1]	Acceso a variables locales
Registro diferido o indirecto	Add R4, (R1)	R4←R4+ M[R1]	Acceso utilizando un puntero o una dirección calculada
Indexado	Add R3, (R1+R2)	R3←R3+ M[R1+R2]	A veces útil en direccionamiento de arrays- R1 base del array; R2 índice.
Directo o absoluto	Add R1, (1001)	R1←R1+M[1001]	A veces útil para acceder a datos estáticos; la constante que especifica la dirección puede necesitar ser grande
Indirecto o diferido de memoria	Add R1, @(R3)	R1←R1+M[M[R3]]	Si R3 es la dirección de un puntero p, entonces el modo obtiene *p
Autoincremento	Add R1, (R2)+	R1←R1+M[R2] R2←R2+d	Util para recorridos de arrays en un bucle. R2 apunta al principio del array; cada referencia incrementa R2 en el tamaño de un elemento, d.
Autodecremento	Add R1, -(R2)	R2←R2-d R1←R1+M[R2]	El mismo uso que autoincremento. Autoincremento/decremento también puede utilizarse para realizar una pila mediante introducir y sacar (push y pop)
Escalado o índice	Add R1, 100 (R2)[R3]	R1←R1+M[100+R2+ R3*d]	Usado para acceder a arrays por índice. Puede aplicarse a cualquier modo de direccionamiento básico en algunas máquinas.

Los modos de direccionamiento reducen significativamente el recuento de instrucciones, pero hacen más compleja la construcción de una máquina, pudiendo incrementar el CPI medio.

El arquitecto de computadores debe elegir que modos de direccionamiento incluir en el computador. Para tomar estas decisiones será útil estudiar la frecuencia con que se utilizan los diferentes modos de direccionamiento.

En la figura vemos los resultados de medir la frecuencia de utilización de los modos de direccionamiento en los benchmark SPEC (gcc, spice, Tex) en el VAX que soporta todos los modos mostrados. Estas medidas son bastante independientes de la arquitectura.

Observamos que el direccionamiento inmediato y desplazamiento dominan la utilización de los modos de direccionamiento.

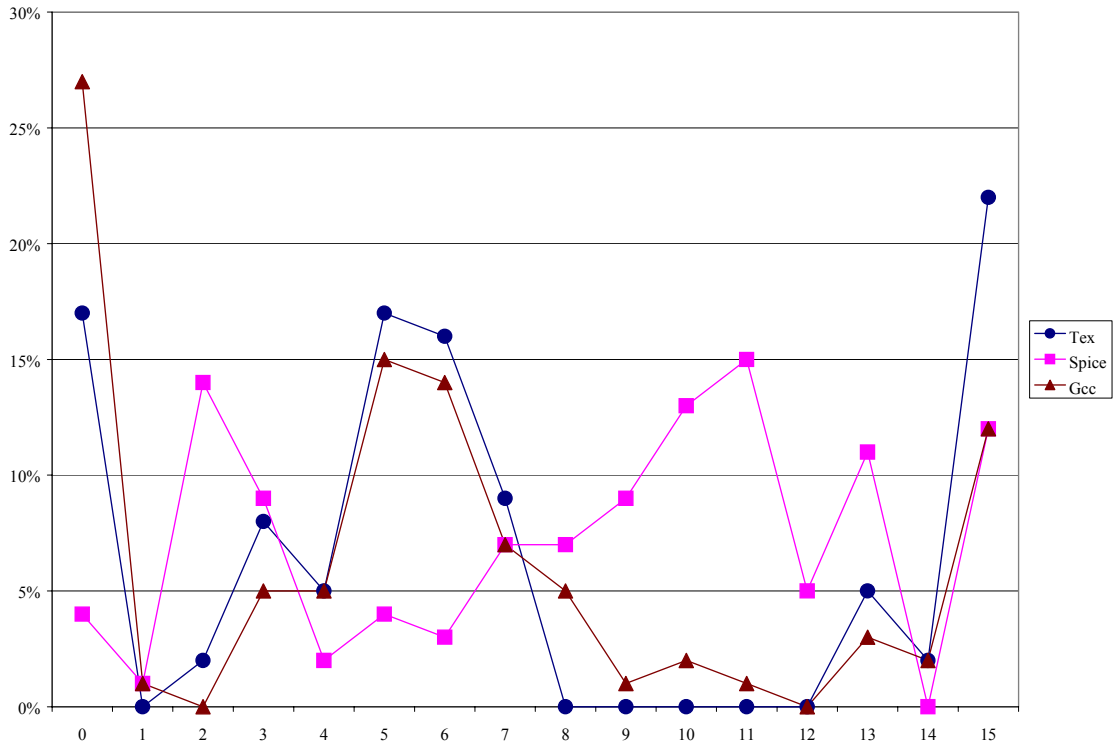


c. Modo de direccionamiento desplazamiento

¿Cuál es el rango más frecuente de desplazamientos en este modo de direccionamiento?

La respuesta a esta pregunta indicará que tamaño soportar. Escoger el tamaño del campo de desplazamiento es importante porque afecta directamente a la longitud de la instrucción.

Se observa que los valores de los desplazamientos están ampliamente distribuidos.



El eje x representa $\log_2$ del desplazamiento, es decir, el tamaño del campo requerido para representar la magnitud del desplazamiento.

Aunque hay un gran número de valores pequeños también hay un número razonable de valores grandes. La amplia distribución de los valores de desplazamiento se debe a múltiples áreas de almacenamiento para las variables y diferentes desplazamientos utilizados para accederlas.

Estos datos se tomaron en la arquitectura MIPS, mostrando la media de cinco programas de SPECint92 (compress, espresso, eqntott, gcc, li) y la media de cinco programas SPECfp92 (dudoc, ear, hydro2d, mdljdp2, su2cor).

Desplazamientos en bits para diferentes arquitecturas: VAX (8,16,32); IBM360 (12) ; DLX (16); 80x86 (8,16).

d. Modo de direccionamiento literal o inmediato

Los inmediatos se utilizan frecuentemente en:

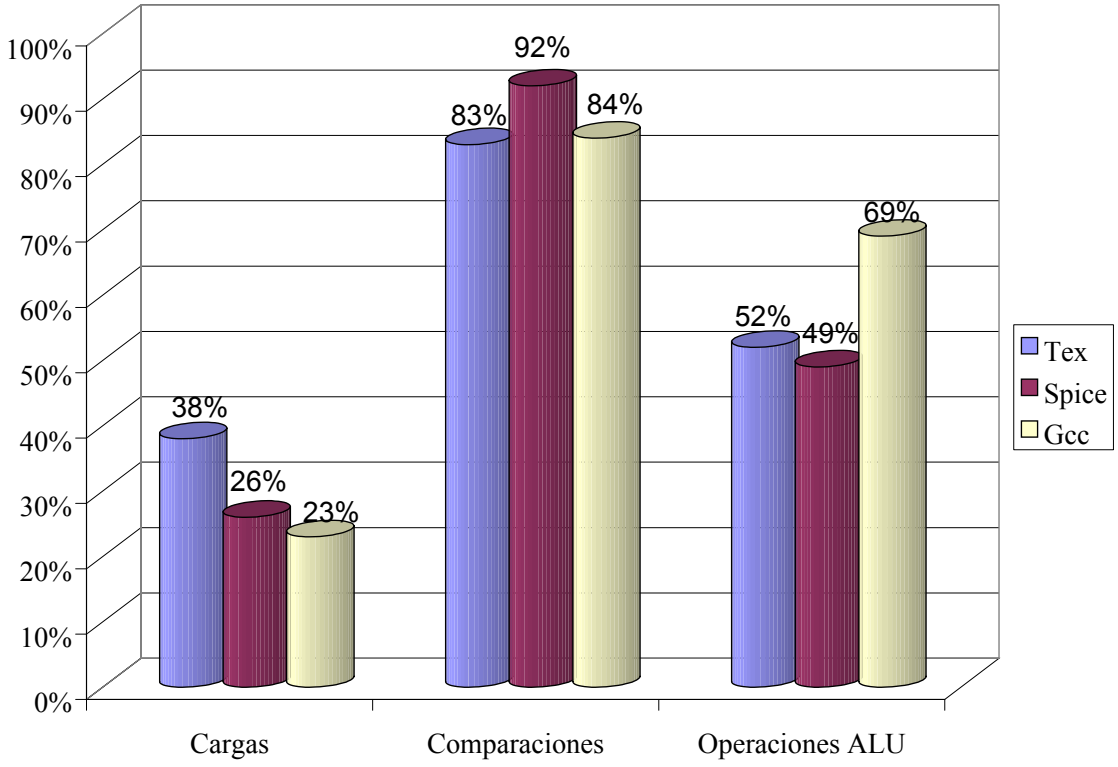
- Operaciones aritméticas
- Comparaciones (principalmente para saltos)
- Transferencias para poner una constante en un registro. Este caso se presenta para:
  - Constantes escritas en el código que tienden a ser pequeñas.
  - Constantes de direcciones que pueden ser grandes.

Para el uso de los inmediatos es importante plantearse dos cuestiones:

- ¿Que operaciones necesitan soportar inmediatos?
- ¿Qué rango de valores es necesario para los inmediatos?

Operaciones con inmediatos

En el siguiente gráfico vemos la frecuencia de los inmediatos para distintos tipos de operaciones.



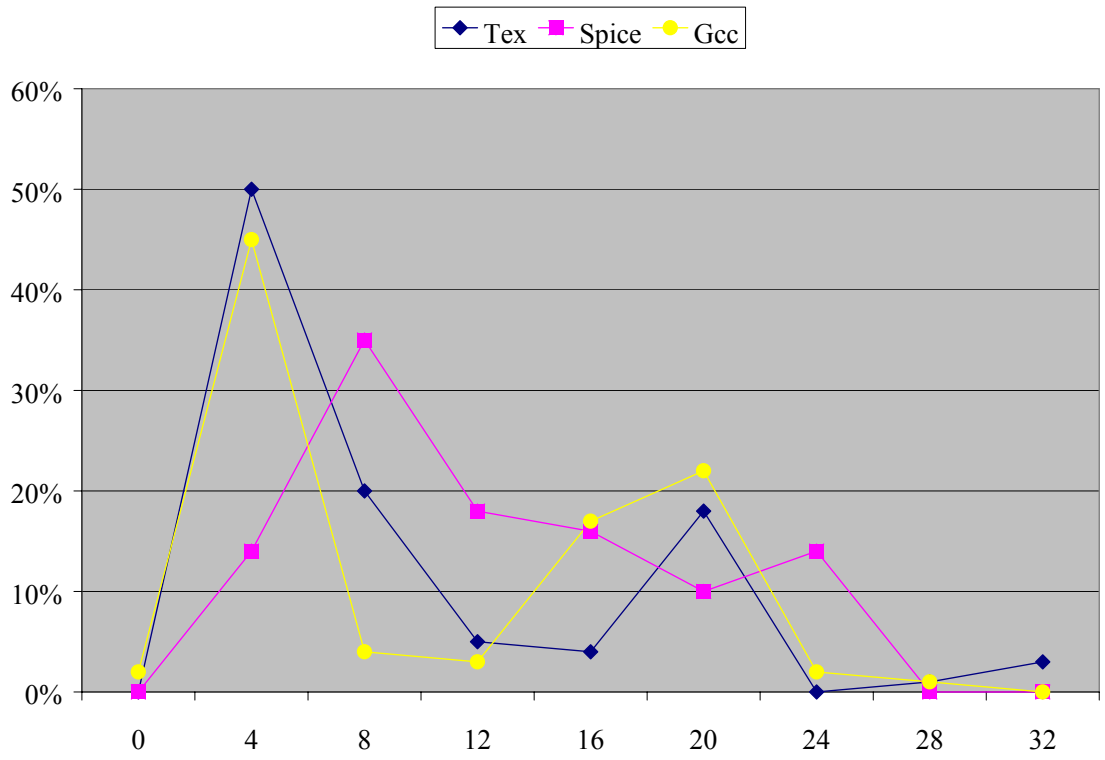
Aproximadamente la mitad de las operaciones de la ALU tienen un operando inmediato. Más del 85% de las operaciones de comparación utilizan un operando inmediato. Aproximadamente un 30% de las operaciones de carga utilizan un operando inmediato (carga inmediata).

Las medidas se tomaron en una arquitectura MIPS R2000.

Rango de valores de los inmediatos

El tamaño de los valores inmediatos afecta a la longitud de la instrucción.

El siguiente gráfico muestra la distribución de valores inmediatos: Los valores inmediatos pequeños son los más intensamente utilizados, aunque se usan inmediatos grandes en el cálculo de direcciones.



El eje x muestra el número de bits necesarios para representar la magnitud de un valor inmediato.

La inmensa mayoría de los valores inmediatos son positivos menos un 6% aproximadamente negativos.

Los datos se tomaron en un VAX.

Dimensión de los inmediatos en bits para diferentes arquitecturas: VAX (8,16,32); IBM360 (8) ; DLX (16); 80x86 (8,16).

e. Codificación de los modos de direccionamiento

La codificación de los modos de direccionamiento puede realizarse de varias formas:

Codificación incluida en el código de operación: Para un pequeño número de modos de direccionamiento o combinaciones modo de direccionamiento/código de operación, el modo de direccionamiento puede codificarse en el código de operación. Por ejemplo el IBM 360 tiene 5 modos de direccionamiento y la mayoría de las operaciones utilizan sólo uno o dos modos.

Especificador de direcciones separado para cada operación: En muchas ocasiones se necesita este especificador para indicar el modo de direccionamiento que esta usando cada operando.

Cuando se codifican las instrucciones, el número de registros y el de modos de direccionamiento tienen un impacto significativo.

El arquitecto debe equilibrar diversas tendencias al codificar el repertorio de instrucciones:

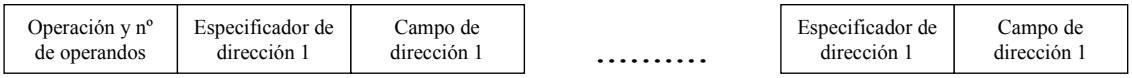
El interés de disponer del mayor número posible de registros y modos de direccionamiento.

El impacto del tamaño de los campos de los registros y de los modos de direccionamiento en el tamaño medio de la instrucción.

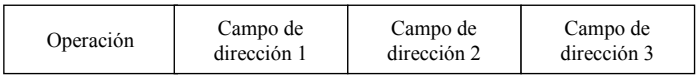
El interés de tener instrucciones codificadas en longitudes fáciles de manejar e implementar.

Tres opciones populares para codificar los modos de direccionamiento

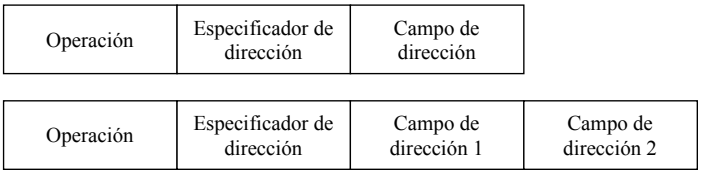
a) Variable (VAX)



b) Fijo (DLX, MIPS, PowerPC, Precision Architecture, SPARC,)



c) Hibrido (IBM 360,370)



Variable:

Permite cualquier modo de direccionamiento con cualquier operador. Interesante con un número alto de modos de direccionamiento y operaciones. Consigue un menor recuento de instrucciones en los programas pero las instrucciones individuales varían en talla y cantidad de trabajo. Ejemplo VAX.

Fija:

Combina la operación y el modo de direccionamiento en el código de operación. Frecuentemente tiene un tamaño único para todas las instrucciones. Interesante con un número reducido de modos de direccionamiento y operaciones. Son más fáciles de decodificar e implementar pero conducen a recuentos de instrucciones altos. Ejemplo DLX.

Híbrida:

Esta alternativa reduce la variabilidad en talla y trabajo proporcionando varias longitudes de instrucción. Es una alternativa intermedia que persigue las ventajas de las anteriores: reducir recuento de instrucciones y formato sencillo de fácil implementación. Ejemplo IBM 360.

2.4. Repertorio de instrucciones.

a. Tipos de operaciones

Estudiamos en este apartado los tipos de operaciones más frecuentes, presentes en los repertorios de instrucciones.

Tipo de operación	Ejemplo
Aritmético y lógico	Operaciones lógicas y aritméticas enteras: suma, and, resta, or...
Transferencias de datos	Cargas y almacenamientos
Control	Salto, bifurcación, llamada y retorno de procedimiento, traps
Sistema	Llamada al sistema operativo, instrucciones de gestión de memoria virtual
Punto flotante	Operaciones de punto flotante: suma, multiplicación
Decimal	Suma decimal, multiplicación decimal, conversiones de decimal a caracteres
Cadenas	Transferencia de cadenas, comparación de cadenas, búsqueda de cadenas
Gráficos	Operaciones sobre pixels, operaciones de compresión descompresión

Todas las máquinas proporcionan, generalmente, un repertorio completo de operaciones para las tres primeras categorías. El soporte para las funciones del sistema varía entre arquitecturas. La incorporación de operación en punto flotante es muy frecuente incluso en repertorios reducidos.

Las tres últimas categorías pueden no estar presentes en algunas arquitecturas. Otras arquitecturas, de repertorio más extenso, pueden contener un amplio repertorio en las tres categorías.

Una regla de comportamiento común a todas las arquitecturas es que las instrucciones utilizadas más extensamente de un conjunto de instrucciones son las operaciones simples. Por ejemplo vemos 10 instrucciones simples del 80x86 que contabilizan el 96% de las instrucciones ejecutadas. El diseñador debe esforzarse en hacer rápidas estas instrucciones:

	Instrucciones 80x86	Promedio
1	Load	22%
2	Salto condicional	20%
3	Comparación	16%
4	Store	12%
5	Add	8%
6	And	6%
7	Sub	5%
8	Move reg-reg	4%
9	Call	1%
10	Return	1%
	Total	96%

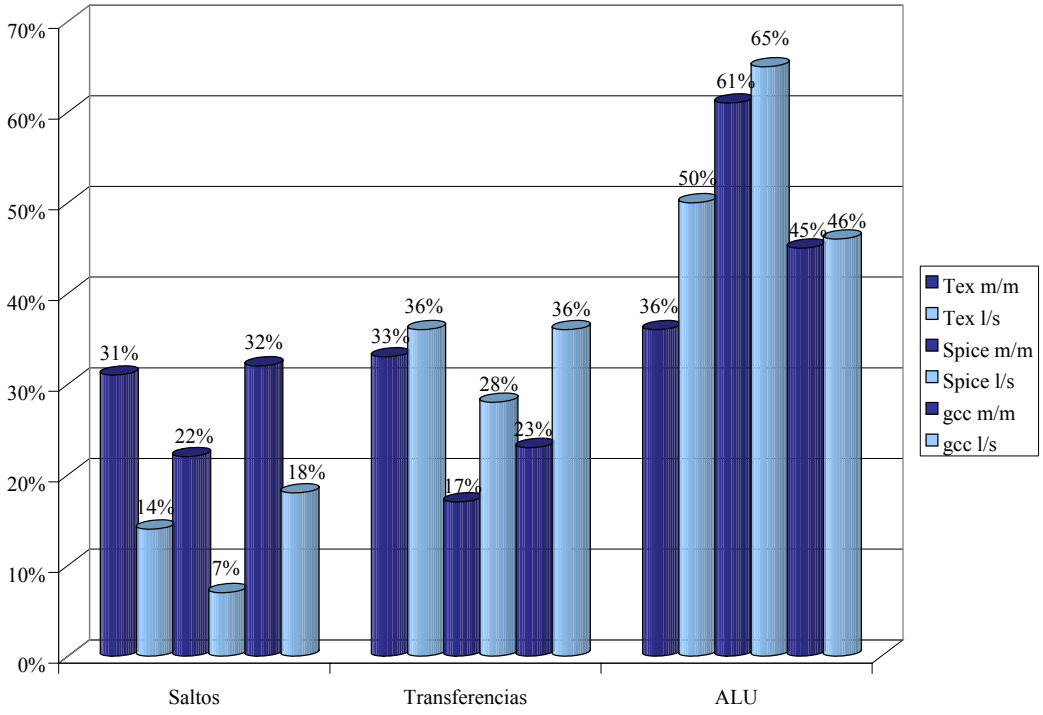
b. Repertorios de instrucciones memoria memoria frente a carga almacenamiento.

Vamos a estudiar el comportamiento de los benchmarks ejecutados en diferentes arquitecturas (carga-almacenamiento y memoria-memoria). La tecnología de compiladores es diferente para estas arquitecturas afectando a las medidas globales.

Estudiamos la frecuencia de tres operaciones básicas:

Referencias a memoria
Operaciones de la ALU
Instrucciones de flujo de control (saltos y bifurcaciones)

Observamos en la figura las frecuencias para una arquitectura carga/almacenamiento (DLX) y para una arquitectura memoria/memoria (VAX).



En la máquina de carga almacenamiento, los movimientos de datos son cargas o almacenamientos. En la máquina memoria-memoria los movimientos de datos incluyen transferencias entre dos posiciones (según los operandos registros o posiciones de memoria). La mayoría de los movimientos de datos involucra un registro y una posición de memoria.

La máquina de carga almacenamiento muestra un mayor porcentaje de movimientos de datos ya que para operar con los datos deben transferirse a los registros.

La frecuencia relativa más baja para los saltos de la arquitectura carga/almacenamiento es consecuencia del uso de más instrucciones de los otros tipos.

Observamos en la figura para las mismas máquinas y programas:

El recuento absoluto de instrucciones ejecutadas

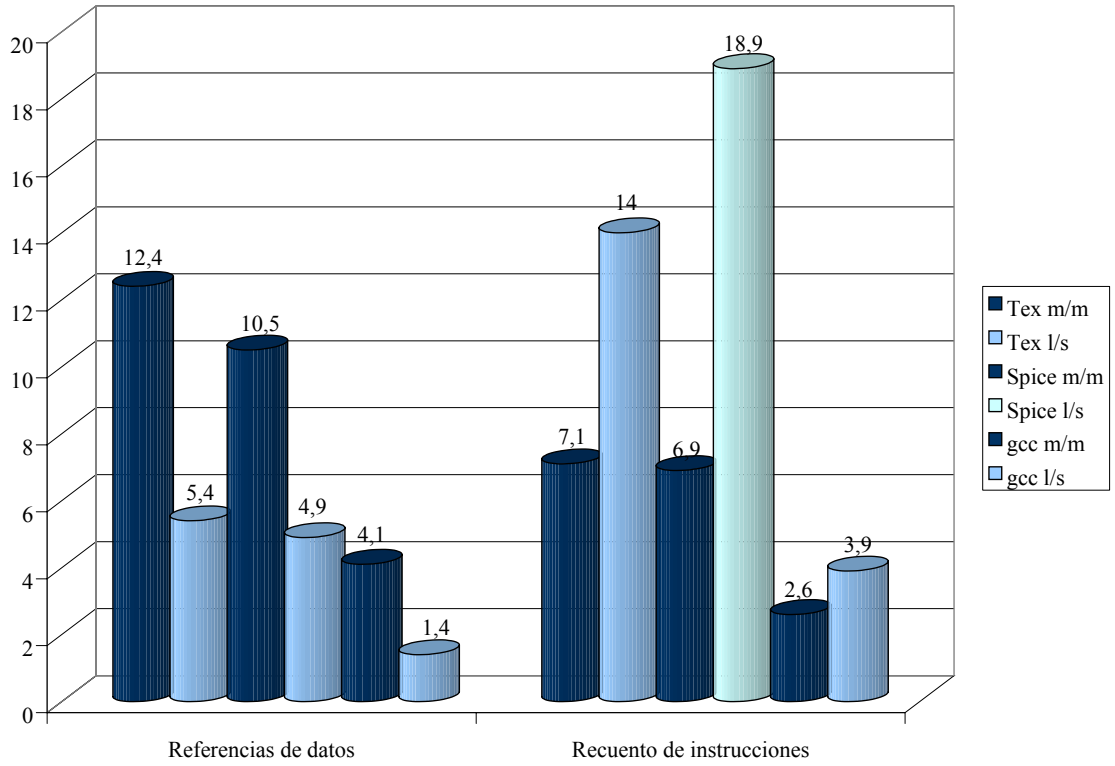
Referencias a datos en memoria (cargas, almacenamientos, ALU mem)

Se observa que la máquina carga almacenamiento requiere más instrucciones. Esto no implica nada respecto al rendimiento de estas arquitecturas.

Podríamos deducir de los recuentos globales de instrucciones que el número de referencias a datos realizadas por las arquitecturas carga/almacenamiento es mayor. Sin embargo los datos de la figura indican lo contrario (más referencias a datos en mem/mem que en l/s).

En las arquitecturas mem/mem se realizan referencias a datos no sólo con operaciones de transferencia sino con las ALU. La diferencia en las referencias a los datos surge de las mejores posibilidades de ubicación de registros de las arquitecturas l/s.

Las diferencias entre las referencias a datos de las arquitecturas mem-mem y las l/s equilibra la diferencia entre referencias a instrucciones.



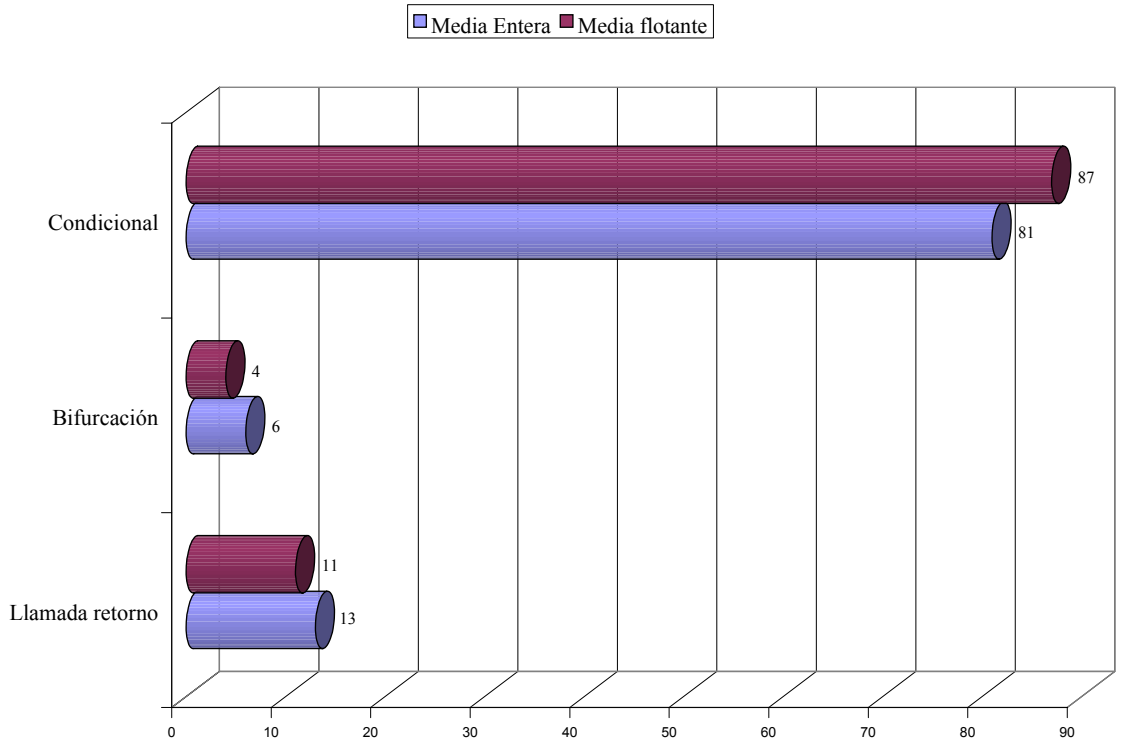
c. Instrucciones de control

Utilizaremos el término bifurcación (jump) para los cambios en el control que sean incondicionales y salto (branch) cuando sean condicionales.

Distinguimos cuatro tipos de cambios del flujo de control:

1. Saltos condicionales
2. Bifurcaciones
3. Llamadas a procedimientos
4. Retornos de procedimiento

Vemos en el siguiente gráfico la frecuencia de las instrucciones de flujo de control para una máquina de carga almacenamiento:



Los saltos condicionales son los que más se utilizan.

Formas de especificar el destino del salto:

La dirección destino del salto se especifica siempre. Esta dirección, en la mayoría de los casos, se especifica explícitamente, siendo el retorno de procedimiento la excepción más importante (el destino no se conoce en tiempo de compilación ya que el procedimiento se puede llamar desde varios puntos y en consecuencia los retornos pueden ser a varios puntos).

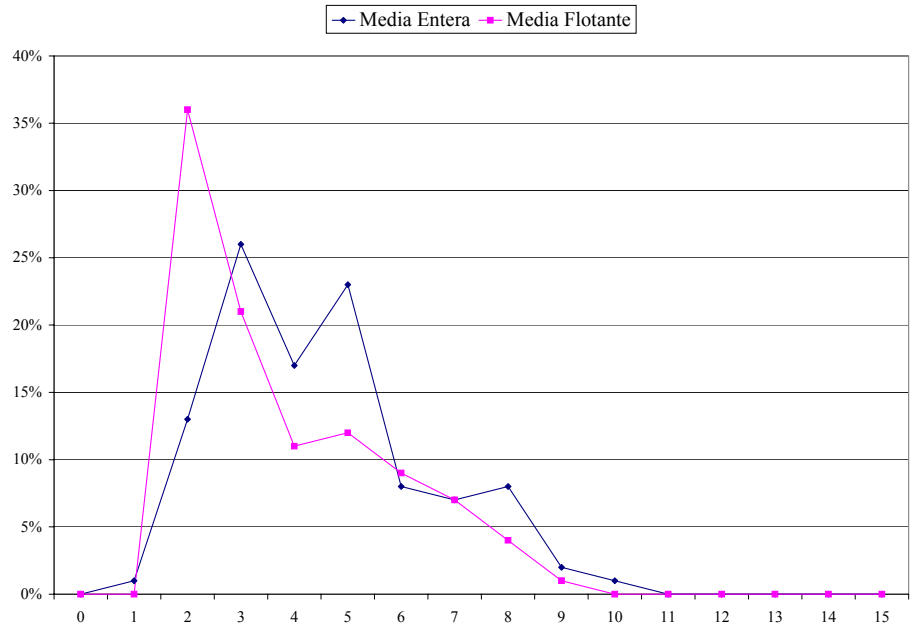
Saltos relativos al PC

La dirección se especifica mediante un desplazamiento que se suma al contador de programa o PC. Normalmente la posición destino del salto es cercana a la actual y especificar el salto requiere pocos bits.

Saltos no relativos al PC

Para implementar retornos y saltos a direcciones concretas en los que el destino del salto no se conoce en tiempo de compilación y es necesario especificarlo dinámicamente. Para ello se puede nombrar un registro que contenga la dirección del destino; alternativamente, el salto puede permitir que se utilice cualquier modo de direccionamiento.

El diseñador debe conocer la magnitud de los desplazamientos para ver como afecta a la longitud y codificación de la instrucción. Vemos en el siguiente gráfico las distancias de los saltos relativos al PC, en función del número de instrucciones entre el destino y la instrucción de salto.



Los saltos más frecuentes en los programas enteros están entre 3 y 5 instrucciones, aproximadamente un 25%. En los programas en punto flotante los saltos de 2 instrucciones son los más frecuentes.

Esto indica que los campos de desplazamientos cortos son suficientes con frecuencia. 8 bits cubren el 93% de los casos.

Estas mediadas fueron tomadas en una arquitectura carga almacenamiento (DLX).

Formas de especificar la condición del salto:

Otra cuestión importante es como especificar la condición de salto. Las tres técnicas principales son:

Código de condición

Los saltos examinan bits especiales inicializados por las operaciones de la ALU.

Ejemplo:

```
SUB R1, R2, R3;     R1=R2-R3
CMP R1, #0;         Si R1=0 el indicador z=1
BEQ eti;            Salta si z=1
```

Ventaja: Las comparaciones pueden eliminarse en algún caso.

Inconveniente: Problemas en máquinas segmentadas derivados de la posible utilización simultanea de z desde varias instrucciones.

Registro de condición

Los saltos examinan registros arbitrarios con el resultado de una comparación.

Ejemplo:

```
SUB R1, R2, R3;         R1=R2-R3
SEQ R10, R1, #0;        Si R1=0 se actualiza R10 con un 1
BNEZ R10,eti;           Salta si R10<>0
```

Ventaja: Independencia entre la operación y el registro implicado.

Inconveniente: Se consume un registro.

Comparación y salto

La comparación es parte del salto, permitiendo saltar con una sola instrucción, si bien puede ser demasiado trabajo por instrucción.

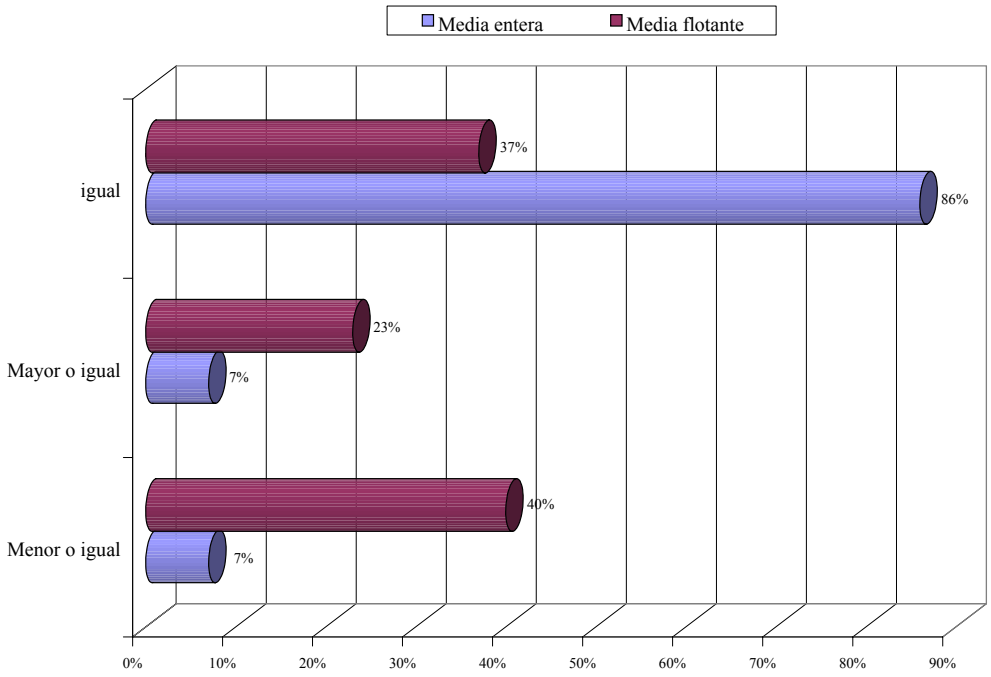
Ejemplo:

```
SUB R1, R2, R3;         R1=R2-R3
C&B R1, #0, eti;        Si R1=0 salta a etiqueta.
```

Ventaja: Reducción del recuento de instrucciones.

Inconveniente: Puede ser demasiado trabajo para una instrucción, aumentando el CPI o el clk.

La mayor parte de las comparaciones son test de igualdad desigualdad y un gran número son comparaciones con 0 (aproximadamente un 50% son test de igualdad con 0).



Saltos efectivos y no efectivos

Un salto es efectivo si se realiza, es decir si la condición de salto es verdadera, siendo no efectivo en caso contrario.

25% de saltos son hacia atrás (bucles), donde el 90% son efectivos.
75% de saltos son hacia delante, donde el 50% son efectivos.
0,25*0,9+0,75*0,5=0,6 El 60% de los saltos son efectivos.

Esta afirmación podrá ser utilizada en las técnicas de predicción de salto que se estudiarán en la segmentación.

2.5 Tipo y tamaño de los operandos

Un problema fundamental es la forma de designar el tipo de operando. Hay dos alternativas importantes:

En el código de operación:

El tipo de operando se expresa en el código de operación. Es el método utilizado con más frecuencia

Datos identificados o autodefinidos:

El dato se anota con identificadores que especifican el tipo de cada operando y que son interpretados por el hardware. Son extremadamente raras. Arquitecturas de Burroughs. Symbolics para implementaciones LISP.

Tamaños más comunes de los operandos:

Byte (8 bits)
Media palabra (16 bits)
Palabra (32 bits)
Doble palabra (64 bits)

Codificaciones más comunes de los operandos:

Los caracteres se presentan como:
EBCDIC: utilizado por las arquitecturas de grandes computadores IBM.
ASCII: (128 ASCII estandar y 128 ASCII extendido). Muy difundido.

Los enteros: Representación en complemento a 2 muy difundida.

Punto flotante: El estándar más difundido en la actualidad es el 754 de IEEE.

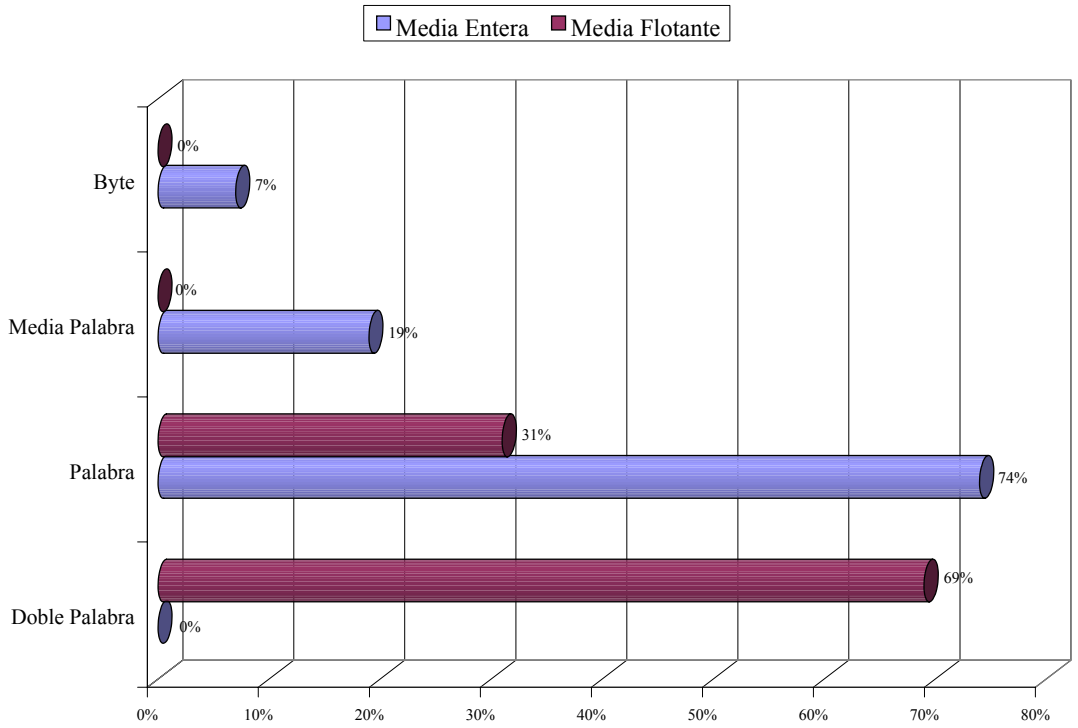
Precisión simple: 32 bits (1+8+23 signo, exponente, mantisa).
Precisión doble: 64 bits (1+11+52 signo, exponente, mantisa).
Precisión simple extendida:
Precisión doble extendida:
Los formatos extendidos se utilizan para evitar errores y desbordamientos en operaciones intermedias aumentando el número de bits de mantisa y exponente. Dependen de implementaciones

Cadenas de caracteres: Algunas arquitecturas soportan operaciones sobre cadenas de caracteres ASCII (comparaciones, desplazamientos...)

Decimales: Algunas arquitecturas soportan un formato denominado habitualmente decimal empaquetado. Se utilizan 4 bits para codificar los valores 0-9, y en cada byte se empaquetan dos dígitos decimales.

Distribución de los accesos a los datos por tamaños:

Los accesos a los tipos principales de datos (palabra y doble palabra) dominan claramente.



Se observa el predominio de operandos enteros de 32 bits y operandos en coma flotante de 64 bits (IEEE 754).

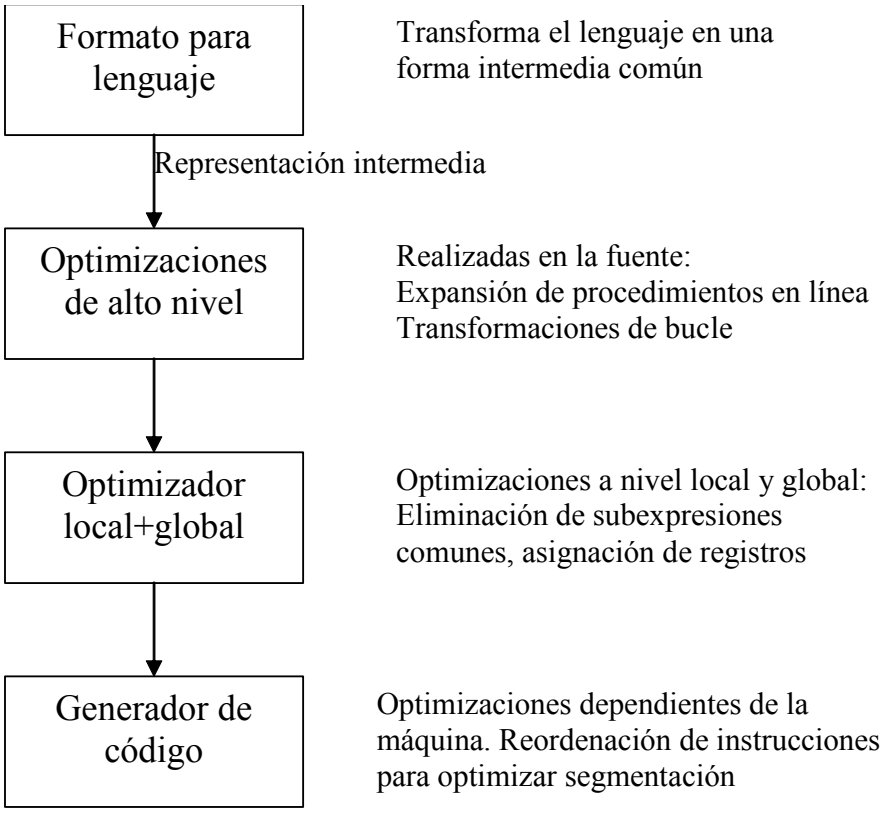
2.6 La arquitectura como objeto del compilador

La mayor parte de la programación se realiza en lenguajes de alto nivel. Por lo tanto, la mayoría de las instrucciones ejecutadas son la salida de un compilador.

Una arquitectura a nivel lenguaje máquina es esencialmente un objeto del compilador. Por lo tanto, la comprensión de la tecnología de compiladores es fundamental para el arquitecto de computadores.

Estructura de los compiladores recientes

Los compiladores actuales constan de varios pasos o fases que transforman representaciones de más alto nivel en representaciones progresivamente de más bajo nivel, alcanzando el repertorio de instrucciones.



Dos preguntas importantes para el arquitecto de computadores

¿Cuantos registros se necesitan para ubicar las variables?

La óptima ubicación de las variables en los registros depende del número de registros de propósito general y de la estrategia de ubicación del compilador.

La ubicación de registros influye tanto en la aceleración del código (acceso a registros frente a acceso a memoria) como en hacer útiles otras optimizaciones. Por ejemplo, en la eliminación global de subexpresiones comunes (encontrar dos instancias de una expresión que calculan en mismo valor y guardar el valor de forma temporal), para que la optimización sea efectiva el valor temporal debe almacenarse en un registro, el coste de almacenar en memoria y recargarlo puede anular el afecto de la mejora.

Los algoritmos de ubicación de registros están basados en una técnica denominada coloreado de grafos, cuyo funcionamiento mejora cuando al menos existen 16 registros (preferiblemente más) de propósito general, disponibles para ubicación de variables enteras y algunos adicionales para las variables de punto flotante. Por ejemplo DLX proporciona 32 enteros y 32 para trabajo en punto flotante.

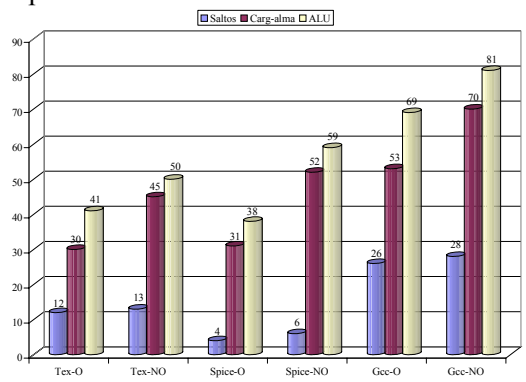
Por otro lado el disponer de un número de registros elevado puede complicar la arquitectura y el repertorio de instrucciones.

¿Cómo influyen las técnicas de optimización en la mezcla de instrucciones?

El gráfico nos puede ayudar a analizar esta pregunta. Los datos se tomaron en una arquitectura carga-almacenamiento.

El efecto más inmediato de la optimización es la reducción del recuento global de instrucciones.

La frecuencia de los saltos disminuye más lentamente que las referencias a memoria y las operaciones ALU. Las estructuras de control son las más difíciles de reducir y hacer más rápidas por parte del compilador.



Observamos los millones de instrucciones ejecutadas para gcc, Tex y spice (spice decenas de millones). Los programas no optimizados ejecutan el 21%, 58% y 30% más instrucciones para gcc, spice y Tex respectivamente.

Propiedades de los repertorios de instrucciones que ayudan al escritor de compiladores:

Ortogonalidad:

Los tres componentes principales de un repertorio de instrucciones, operaciones, tipos de datos y modos de direccionamiento deben ser ortogonales. Dos aspectos de una arquitectura se dice que son ortogonales si son independientes.

Por ejemplo, las operaciones y los modos de direccionamiento son ortogonales si para cada operación (a la que se le pueda aplicar un cierto modo de direccionamiento) son aplicables todos los modos de direccionamiento.

La ortogonalidad ayuda a simplificar la generación de código.

Un buen contraejemplo es restringir los registros que se pueden utilizar para un cierto tipo de instrucción. Esto puede dar lugar a que el compilador tenga muchos registros disponibles pero ninguno del tipo correcto.

Proporcionar primitivas y no soluciones:

Intentos de soportar lenguajes de alto nivel no han tenido éxito.

Proporcionar información de las secuencias alternativas de código de rendimiento óptimo:

Una de las tareas del escritor de compiladores es imaginar la secuencia de instrucciones óptima para cada segmento de código.
Con las últimas técnicas de mejora del rendimiento (caches, segmentación...) mediadas como el número de instrucciones o el tamaño del código no son representativas. El arquitecto debe ayudar al diseñador de compiladores a realizar esta tarea.

2.7 Algunos repertorios de instrucciones.

Vamos a examinar **algunas arquitecturas específicas y medidas detalladas** de estas.

Hemos escogido **cuatro máquinas** para examinarlas:

- **VAX de DEC** (ha durado 10 años, década de los 80, y cientos de miles de unidades).

- **IBM 360** (ha durado 25 años decadas 70 y 80, y cientos de miles de unidades).

- **Intel 8086** Es el computador de propósito general más popular del mundo. (más de 10 millones de máquinas). Decada de los 80 y 90.

- **DLX** Maquina genérica de carga almacenamiento muy popular desde finales de los 80.

2.7.1 La arquitectura VAX de DEC

Los procesadores de **la familia VAX (Virtual Address Extension)** tienen una **arquitectura de 32 bits**, complementada con el **sistema operativo VAX/VMS**, que **permite** el trabajo con **memoria virtual**.

Los **tres sistemas iniciales**

1. **VAX 11/730:** Es el modelo más barato.
2. **VAX 11/750:** Es el modelo medio.
3. **VAX 11/780:** Es el modelo más potente, destinado al manejo de grandes bases de datos y ejecución de procesos de altos requerimientos.

El primer modelo, el **VAX-11/780**, **apareció en 1977** como una **extensión de 32 bits del PDP-11**; soportando una sintaxis similar para el lenguaje ensamblador, los mismos tipos de datos y soporte de emulación del PDP-11.

Uno de los **objetivos** fue **facilitar** la **tarea** de **escritura** de **compiladores y sistemas operativos** proporcionando una **arquitectura altamente ortogonales**.

Las demás arquitecturas que estudiaremos son **subconjuntos del VAX** en términos de instrucciones y modos de direccionamiento, por eso empezamos con el VAX.

El **VAX** es una **máquina** de **registros de propósito general** con un **repertorio** de instrucciones **muy ortogonal.**

Los registros

El VAX **tiene 16 registros de propósito general de 32 bits**, aunque **cuatro** de ellos son **utilizados por la arquitectura**.

R_0 , R_{11}	**Registros de propósito general**
R_{12}	**(AP) Puntero de argumento, dirección de comienzo de lista argumentos de un proc.**
R_{13}	**(FP) Puntero de región, empleado en las llamadas a procedimientos.**
R_{14}	**Es el puntero de pila.**
R_{15}	**Es el PC (contador de programa).**

La doble palabra de estado del procesador

El VAX dispone de una **palabra de estado de 32 bits**. Los **16 bits de menor peso** forman la **palabra de estado del procesador**. Los **16 bits de mayor peso** tienen un **carácter privilegiado**.

El VAX puede **trabajar en varios modos** de privilegio: modo **núcleo** (nivel núcleo del S.O. máxima prioridad); modo **ejecutivo** (usado por las llamadas del sistema operativo); modo **supervisor** (nivel de interpretación de comandos...); modo **usuario** (utilidades compiladores, depuradores, menor privilegio).

Se utilizan **códigos de condición para los saltos** que son **inicializados por** las **operaciones aritméticas, lógicas y de transferencia**. Son los tres bits de menor peso de la palabra de estado del procesador.

La memoria

El **bus** de direcciones de la CPU es **de 32 líneas**, por lo tanto direcciones de hasta **4Gygabytes** (2^{32}) de **memoria virtual**.

La memoria sigue una ordenación **little endian**.

Una instrucción como la move **transfiere datos entre dos posiciones** direccionables **cualesquiera**:

Cargas:	reg-mem
Almacenamientos:	mem-reg
Transferencias r-r:	reg-reg
Transferencias m-m:	mem-mem

Observamos en la tabla los **tipos de datos soportados**:

Bits	Tipo de dato	Nuestro nombre	Nombre de DEC
8	Entero	Byte	Byte (B)
16	Entero	Media palabra	Palabra (W)
32	Entero	Palabra	Palabra larga (L)
64	Entero	Doble palabra	Cuad palabra (Q)
128	Entero	Cuad palabra	Octa-palabra (O)
32	Punto flotante	Simple precisión	F_flotante (F)
64	Punto flotante	Doble precisión	D_flotante, G_flotante (D,G)
128	Punto flotante	Huge (Enorme)	H-flotante (H)
4n	Decimal	Empaquetado	Empaquetado (P)
8n	Cadena numérica	Desempaquetado	Cadenas numéricas (S)
8n	Cadenas de caracteres	Caracter	Caracter (C)

La inicial del tipo de dato **se utiliza** con frecuencia **para completar un nombre de código de operación.** Por **ejemplo** cada instrucción mov transfiere un operando del tipo de dato indicado:

MOVB, MOVW, MOVL, MOVQ, MOVO, MOVF, MOVG, MOVD, MOVH, MOVC3, MOVP (No hay diferencia entre mover cadenas numéricas y de caracteres).

El **VAX utiliza** el nombre **palabra para referenciar cantidades de 16 bits**.

Modos de direccionamiento

Los **modos de direccionamiento** utilizados por el VAX son:

Modo de direccionamiento	Sintaxis
Literal	#valor
Inmediato	#valor
Registro	R_n
Registro diferido	(R_n)
Desplazamiento de byte/palabra/largo	Desplazamiento (R_n)
Desplazamiento diferido de byte/palabra/largo	@Desplazamiento (R_n)
Escalado (indexado)	Modo base [R_x]
Autoincremento	(R_n)+
Autodecremento	-(R_n)
Autoincremento diferido	@(R_n)+

El **modo literal sólo** admite operandos de **6 bits** mientras que **el inmediato** permite **más bits**.

Una instrucción VAX de tres operandos **puede incluir desde 0 a tres referencias a memoria**, cada una de las cuales puede utilizar **cualquier modo de direccionamiento**.

Modos de direccionamiento con el PC

Cuando se utilizan los **modos de direccionamiento con el PC** (R_{15}) los modos son los siguientes:

Inmediato: El valor inmediato expresado en el programa incrementa el PC.
Absoluto: El flujo de instrucciones expresa una dirección absoluta de 32 bits.
Desplazamiento de byte/ palabra / largo: la base de desplazamiento es el PC
Desplazamiento diferido de byte/ palabra / largo: la base de desplazamiento es el PC

Codificación de las operaciones VAX

Las **instrucciones VAX constan** de un **código de operación seguido** por **cero o más especificadores de operandos.**

El **código de operación** casi siempre es **un solo byte** que **especifica**: la **operación**, **el tipo de dato** y el **número de operandos**.

Las **operaciones** son **ortogonales** con **respecto** a los **modos de direccionamiento**.

La longitud de los **especificadores de operando puede variar** desde un byte a muchos.

El primer byte de cada especificador de operando consta de **dos campos de 4 bits:**

> **4 bits: Tipo** del **especificador** de direcciones
> **4 bits**: **Registro** que es **parte del modo** de direccionamiento
> **(Si se requiere especificar** un **desplazamiento**, **registros adicionales o un valor inmediato,** se incrementa mediante **aumentos de 1 byte)**

La longitud de cada modo de direccionamiento es **1 byte más la longitud de cualquier desplazamiento o campo inmediato** que este en el modo.

Modo de direccionamiento	Sintaxis	Longitud en bytes
Literal	#valor	1 byte
Inmediato	#valor	1 + longitud del inmediato
Registro	R_n	1
Registro diferido	(R_n)	1
Desplazamiento de byte/palabra/largo	Desplazamiento (R_n)	1 + longitud del desplazamiento
Desplazamiento de byte/palabra/largo	@Desplazamiento (R_n)	1 + longitud del desplazamiento
Escalado (indexado)	Modo base [R_x]	1 + longitud del modo de direccionamiento base
Autoincremento	(R_n)+	1
Autodecremento	-(R_n)	1
Autoincremento diferido	@(R_n)+	1

En consecuencia **el tamaño total de las instrucciones** se puede calcular **sumando 1 byte** (raramente 2) **del código de operación al tamaño de los especificadores** de operando.

Ejemplo: ¿Que longitud tiene la siguiente instrucción?

ADDL3 R_1, 737(R_2), #456

1 + 1 + (1+2) + (1+4)

Operaciones del VAX

Las operaciones del VAX pueden dividirse en clases: (* significa múltiples tipos de datos)

Tipo	Ejemplo (*)	Significado de la instrucción
Transferencias de datos		**Transferencia de datos entre operandos de byte, media palabra, palabra o doble palabra**
	MOV *	Transferencia entre dos operandos
	MOVZB *	Transfiere un byte a una o media palabra extendiéndolo con ceros
	MOVA *	Transfiere dirección de operando
	PUSH *	Introduce operando en pila
Aritmética lógica		**Operaciones sobre bytes, enteros o lógicos, medias palabras (16 bits), palabras (32 bits)**
	ADD *	**Suma** con dos o tres operandos
	CMP *	**Compara** e inicializa códigos de condición
	TST *	**Compara con cero** e inicializa códigos de condición
	ASH *	**Desplazamiento** aritmético
	CLR *	**Pone a cero**
	CVTB *	Byte de signo extendido para tamaño de tipo de datos
Control		**Saltos condicionales e incondicionales**
	BEQL, BNEQ	**Salta igual/ no igual**
	BCS, BCC	**Salta acarreo a 1, salta acarreo a 0**
	BRB, BRW	**Salto incondicional** con un desplazamiento de 8 o 16 bits
	JMP	**Bifurcación** utilizando **cualquier modo de direccionamiento**
	AOBLEQ	Suma uno al operando y salta si resultado ≤ que 2º operando
	CASE	Salto basado en el selector case
Procedimiento		**Llamada/retorno de procedimiento**
	CALLS	**Llama a procedimiento** con argumentos en pila
	CALLG	**Llama a procedimiento** con lista de **parámetros** estilo FORTRAN
	JSB	Salta a subrutina guardando dirección de vuelta
	RET	Retorno de llamada de procedimiento
Caracter decimal de campo de bits		**Opera sobre campos de bits de longitud variable, cadenas de caracteres y cadenas decimales, ambas en formato de caracteres y BCD**
	EXTV	Extrae un campo de bits (longitud variable) en palabra de 32 bits
	MOVC3	**Transfiere una cadena** de caracteres de longitud dada
	CMPC3	**Compara dos cadenas** de caracteres de longitud dada
	MOVC5	Transfiere cadena de caracteres con truncación o ajuste
	ADDP4	**Suma cadena decimal** de longitud indicada
	CVTPT	**Convierte cadena decimal** empaquetada **en cadena de caracteres**
Punto flotante		**Operaciones de punto flotante sobre formatos, D, F, G y H**
	ADDD	**Suma números flotantes** en formato D en doble precisión
	SUBD	**Resta números flotantes** en formato D en doble precisión
	MULF	**Multiplica en punto flotante** en formato F en simple precisión
	POLYF	**Evalúa un polinomio** utilizando una tabla de coeficientes form F
Sistema		**Cambio a modo de sistema, modifica registros protegidos**
	CHMK, CHME	**Cambia modo a kernel** (núcleo) **ejecutivo**
	REI	**Retorno de excepción** o interrupción
Otros		**Operaciones especiales**
	CRC	Calcula comprobación de redundancia cíclica
	INSQUE	Inserta una entrada de cola en una cola

2.7.2 La arquitectura 360/370

Objetivos oficiales del 360

1. **Explotar la memoria**: gran memoria principal, jerarquías de memoria (ROM para microcódigo).
2. **Soportar E/S concurrente**
3. Crear una **máquina de propósito general** con **muchos tipos de datos** y **facilidades para** los **sistemas operativos**.
4. **Compatibilidad** del lenguaje máquina

El **sistema 370**, es completamente **compatible** con el **360**. Las **principales extensiones**:

- **Memoria virtual** y **traducción dinámica** de direcciones
- **Intrusiones nuevas**: cadenas largas, manipular bytes en registros, instrucciones decimales.
- **Eliminación** de **requerimientos de alineación** de datos.

Arquitectura a nivel lenguaje máquina del 360/370

El **IBM 360** es una **máquina de 32 bits** con **direccionamiento por bytes** que **soporta diversos tipos de datos**:

> **Byte**, **media palabra** (16 bits), **palabra** (32 bits), **doble palabra** (doble precisión real), **decimal empaquetado** y **cadenas de caracteres desempaquetados**.

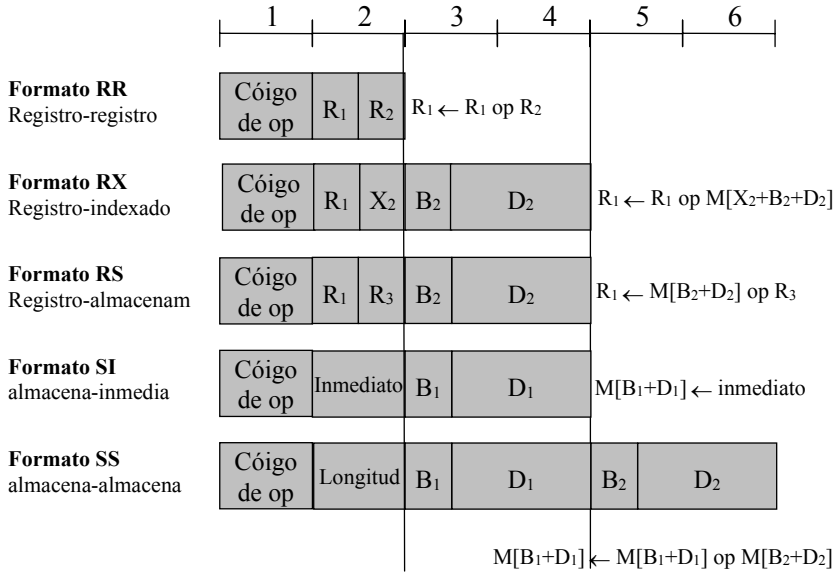
El modelo **360 tiene restricciones de alineación eliminadas en** el modelo **370**.

Componentes:

- **Dieciséis registros de propósito general de 32 bits**.
- **Cuatro registros de punto flotante de doble precisión** (64 bits)
- La **palabra de estado** de programa (**PSW**) **contiene** el **PC**, algunos **señalizadores** y **códigos de condición**.

Modos de direccionamiento y formatos de instrucción

El **IBM 360/370** tiene **cinco formatos de instrucción**. **Cada formato asociado a un modo de direccionamiento** y tiene un conjunto de operaciones definidas para ese formato. Los formatos de instrucción son los siguientes:



RR (Registro-registro). **Ambos operandos** son el **contenido de los registros**. El **primer operando fuente** es **también destino**.

RX (Registro-indexado).

El **primer operando** (fuente y destino**)** es **un registro**.

El **segundo operando** es el contenido de **la posición de memoria dada por la suma** de los campos:

D_2 desplazamiento de 12 bits

B_2 contenido del registro B_2

X_2 contenido del registro X_2

RS (Registro-memoria).

El **primer operando es el** registro **destino**.

El **tercer operando** es un registro que se utiliza como **segunda fuente**.

El **segundo operando** es el contenido de la **posición de memoria dada por la suma** de:

D_2: campo **desplazamiento** de 12 bits

B_2: contenido del registro B_2.

El modo RS se diferencia del RX en que soporta forma de tres operandos, pero elimina el registro índice.

SI (memoria-inmediato).

El destino es un operando de memoria dado por la suma de:

B_1: contenido del registro B_1

D_1: valor del desplazamiento D_1.

El **segundo operando**, un **campo inmediato de 8 bits** es la fuente.

SS (memoria-memoria). **Las direcciones de los dos operandos de memoria son la suma** del contenido de un **registro base B_i** y un **desplazamiento D_i**. El **primer operando** es **fuente y destino**.

Operaciones del 360/370

Las instrucciones pueden dividirse en cutro clases.

1. Operaciones lógicas sobre bits, cadanas de caracteres y cadenas fijas. (RR, RX y algunas RS).
2. Operaciones decimales o de caracteres sobre cadenas de caracteres o dígitos decimales. (SS).
3. Aritmética binaria de punto fijo. (RR y RX).
4. Aritmética de punto flotante. (RR y RX).

Clase o instrucción	Formato	Significado
Control		**Cambia el PC**
BC_	RS, RR	**Examina la condición y salta** condicionalmente
BAL_	RS, RR	Salta y enlaza (Dirección de la siguiente inst en R_{15})
Aritmético, lógica		**Operaciones aritméticas y lógicas**
A_	RX, RR	**Suma**
S_	RX, RR	**Resta**
SLL_	RS	**Desplazamiento lógico** a la izquierda
LA_	RX	Carga dirección
CLI_	SI	**Compara byte de memoria con inmediato**
NI	SI	**AND** inmediato en byte de memoria
C_	RX, RR	Compara y modifica los códigos de condición
TM	RS	Test bajo máscara
MH	RX	**Multiplica media palabra**
Transferencia de datos		**Transferencia entre registros y registro memoria**
L_	RX, RR	**Carga un registro desde memoria u otro registro**
MVI	SI	**Almacena un byte inmediato en memoria**
ST	RX	**Almacena un registro**
LD	RX	**Carga un registro de punto flotante** de doble precis
STD	RX	**Almacena un registro de punto flotante** de doble
LPDR	RR	**Transfiere un registro de punto flotante** de doble
LH	RX	Carga media palabra desde memoria en un regsitro
IC	RX	Inserta un byte de memoria en el de orden inferior de un registro
LTR	RS	Carga un registro e inicializa códigos de condición
Punto flotante		**Operaciones de punto flotante**
AD_	RS, RR	**Suma en punto flotante** y doble precisión
MD_	RS, RR	**Multiplica FP** doble precisión
Cadena decimal		**Operaciones sobre decimales y cadenas de caracteres**
MVC	SS	**Transfiere caracteres**
AP	SS	**Suma cadenas decimales** empaquetadas
ZAP	SS	Pone a cero y suma empaquetada
CVD	RX	**Convierte una palabra binaria en doble palabra** deci
MP	SS	Multiplica dos cadenas decimales empaquetadas
CLC	SS	Compara dos cadenas de caracteres
CP	SS	Compara dos cadenas de decimales empaquetados
ED	SS	Edita convierte decimal empaquetado en cadena de caracteres

El subrayado significa que el código de operación son dos códigos de operación distintos uno RX y otro RR.
Se utilizan códigos de operación separados para especificar el formato de una instrucción.

2.7.3 La arquitectura 8086

Acumulador o GPR

La arquitectura 8086 salió al mercado como una extensión del 8088.
El 8080 era una máquina de acumulador. El 8086 amplió el banco de registros, sin llegar a ser una arquitectura de registros de propósito general ya que practicamente cada registros tiene un uso dedicado.

Arquitectura de 16 bits y memoria segmentada

El 8086 es una arquitectura de 16 bits; con registros internos de 16 bits.
Los diseñadores lograron un espacio de direcciones de 20 bits mediante la segmentación de la memoria en segmentos de 64Kb.

La familia 80x86

Los 80186, 80286, 80386, 80486 y el pentium son extensiones compatibles del 8086.

- El 80186 extendió el repertorio original. Sistema de 16 bits.

- El 80286 amplió el espacio de direcciones a 24 bits. Multitarea y memoria virtual.

- El 80386 se introdujo en 1985 como una verdadera máquina de 32 bits, con registros de 32 bits y espacio de direcciones de 32 bits. Existe un modo virtual que proporciona, en la memoria de 80386, multiples particiones de direcciones 8086 de 20 bits. Además añadio un nuevo conjunto de modos de direccionamiento y de operaciones. Todo esto hace del 80386 una arquitectura practicamente GPR. Para la mayoria de las operaciones se puede utilizar cualquier registro como operando.

- El 80486 se introdujo en el 1989 con más instrucciones y un incremento sustancial del rendimiento. Segmentación del cauce (5 etapas) y cache más sofisticada.

- Pentium. Introducción de técnicas superescalares, varias instrucciones en parlelo. (varias unidades de ejecución).

- Pentium Pro (1995): Profundiza sobre técnicas superescalares.

- Pentium II: Incorpora tecnología MMX (procesamiento eficiente de video audio y gráficos). Se utilizan los registros de la pila del coprocesador.
  La arquitectura del Pentium II (similar a la del Pentium Pro) consta de una envoltura CISC con un núcleo RISC:
  1. El procesador capta instrucciones de memoria
  2. Cada instrucción se traduce en una o más instrucciones RISC de tamaño fijo (microops)
  3. El procesador ejecuta las microops con una organización superescalar
  4. Los datos calculados se escriben en el banco de registros en el orden establecido por el programa

- Pentium III: Instrucciones adicionales en punto flotante para procesamiento eficiente de gráficos 3D. SSE (Streaming SIMD Extension) 8 nuevos registros de 128 bits.

Registros

El 8086 **soporta** tipos de **datos byte y palabra** (16 bits).
Tiene un total de **14 registros divididos** en cuatro grupos:

Clase	Registro	Propósito
Dato		**Usado para que contenga y opere sobre datos**
	AX	Usado para multiplicar dividir y E/S
	BX	Tambien puede usarse como registro de dirección base
	CX	Para operaciones de cadena e instrucciones de bucle
	DX	Para multiplicar dividir y E/S
Dirección		**Usado para formar direcciones efectivas**
	SP	Puntero de pila
	BP	Registro base, en modo de direccion basado
	SI	Registro índice
	DI	Registro índice
Segmento		**Usado para formar direcciones efectivas**
	CS	Segmento de código
	SS	Segmento de pila
	DS	Segmento de datos
	ES	Segmento extra
Control		**Usado para control y estado del programa**
	IP	Puntero de instrucción
	FLAGS	Seis bits de código de condición

En el Pentium

Unidad de enteros

Tipo	Número	Longitud	Propósito
General	8	32	Registros de usuario de uso general
De segmento	6	16	Contienen selectores de segmento
Indicadores	1	32	Bits de estado y control
Puntero de instruc	1	32	Puntero de instrucción

Unidad de punto flotante

Tipo	Número	Longitud	Propósito
Numérico	8	80	Contienen números en punto flotante
Control	1	16	Bits de control
Estado	1	16	Bits de estado
Palabra de etiquetas	1	16	Especifica contenidos registros numéricos
Puntero de instruc	1	48	Apunta a la instrucción interrumpida
Puntero de dato	1	48	Apunta al operando interrumpido

Sucesores del pentium: profundiza en las técnicas superescalares.

Modos de direccionamiento

Las **instrucciones ALU y** de **transferencia** de datos son **de dos operandos** con las combinaciones siguientes:

Tipo de operando fuente/ destino	Segundo operando fuente
Registro	Registro
Registro	Inmediato
Registro	Memoria
Memoria	Registro
Memoria	Inmediato

Los **inmediatos** pueden ser **desde 8 hasta 16 bits de longitud**. Ausencia del modo memoria memoria.

Los modos de direccionamiento son:

	Modo	**Operando**	**Registro**	**Ejemplo**
Registro	Registro	Registro		mov AX, BX
inmediato	Valor	Valor		mov AX, 500
Inmediato	Directo	Variable	DS	mov AX, TABLA

Indirecto	Indirecto mediante registro	[BX]
		[BP]
		[DI]
		[SI]
			DS
			SS
			DS
			DS
				Mov AX, [BX]
				Mov AX, [BP]
				Mov AX, [DI]
				Mov AX, [SI]

Desplaza	Relativo a base	[BX]+desp [BP]+desp	DS SS	mov AX, [BX]+4 mov AX, [BP]+4
Desplaza	Directo indexado	[DI]+desp [SI]+desp	DS DS	mov AL, TABLA[DI] mov AL, TABLA[SI]
Indexado	Indexado a base	[BX][SI]+desp [BX][DI]+desp [BP][SI]+desp [BP][DI]+desp	DS DS SS SS	mov AX, TABLA[BX] [SI] mov AX, TABLA[BX] [DI] mov AX, TABLA[BP] [SI] mov AX, TABLA[BP] [DI]

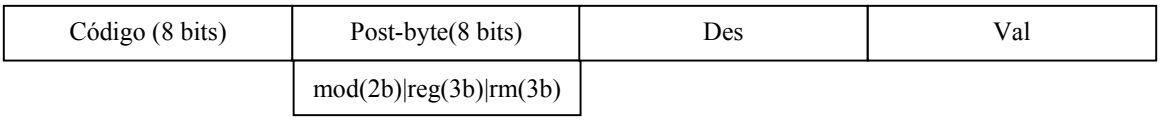
Operaciones del 8086

Hay **cuatro tipos** de instrucción

1. Instrucciones **de transferencia**. Incluyen transferencia introducir y sacar. (move, push, pop).
2. Instrucciones **aritméticas y lógicas**. Operaciones lógicas, de test, desplazamientos y aritméticas.
3. Instrucciones **de control** del flujo. Saltos condicionales, incondicionales, llamadas y retornos.
4. Instrucciones **de cadena**. Transferencia y comparación de cadenas.

Instrucción	Significado
Control	**Saltos condicionales e incondicionales**
JNZ, JZ	**Salta si condición a IP+desplazamiento de 8 bits**
JMP, JMPF	**Bifurcación incondicional**, **intrasegmento y intersegmento**
CALL, CALLF	**Llamada a subrutina**, **intrasegmento y intersegmento**
RET, RETF	**Saca dirección de retorno de la pila**, **intrasegmento y intersegmento**
LOOP	**Salto de bucle, decrementa CX**
Transferencia de datos	**Transferencia de datos entre registros o registros memoria**
MOV	**Transferencia entre dos registros o registro y memoria**
PUSH	**Introduce operando en la pila**
POP	**Saca operando de la cabeza de la pila** o registro
LES	Carga ES y uno de los GPR desde memoria
Aritmético lógicas	**Operaciones aritméticas y lógicas utilizando los registros de datos y memoria**
ADD	**Suma** fuente a destino, formato registro memoria
SUB	**Resta** fuente de destino, formato registro memoria
CMP	**Compara** fuente y destino, formato registro memoria
SHL	**Desplazamiento** a la izquierda
SHR	Desplazamiento lógico a la derecha
RCR	**Rotación** a la derecha con acarreo como relleno
CBW	**Convierte byte** de AL a palabra en AX
TEST	**AND** lógica de fuente y destino modifica señalizadores
INC	**Incrementa** destino; formato registro memoria
DEC	**Decrementa** destino; formato registro memoria
OR	**OR** lógica; formato registro memoria
XOR	**OR** exclusiva; formato registro memoria
Instrucciones de cadena	**Operaciones con cadenas de caracteres**
MOVS	**Copia de la cadena fuente en la destino**; puede repetirse
LODS	Carga un byte o palabra de una cadena en el registro A

Formatos de instrucción



Código: 1er byte, es el único que existe siempre, el resto pueden aparecer o no.

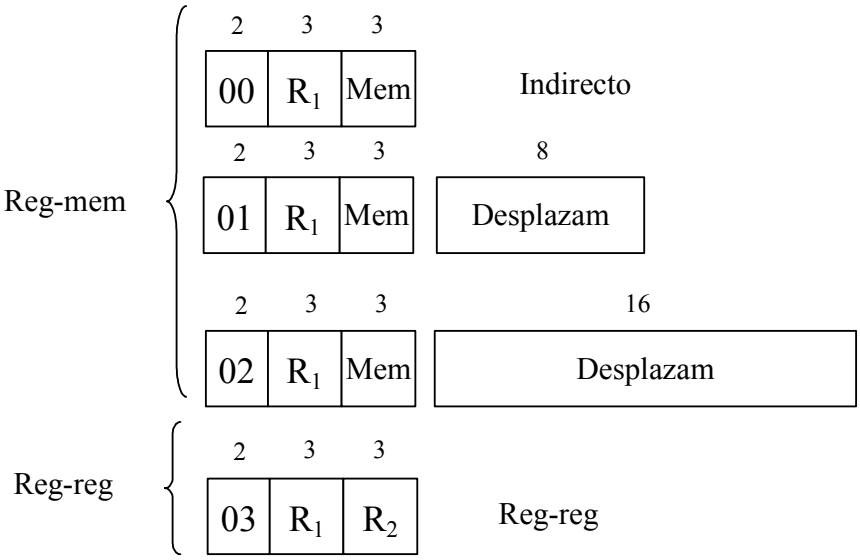
Post-byte: Refleja los operandos de la instrucción

1er operando: mediante mod y rm. Puede ser un registro o una posición de memoria. mod=tipo de direccionamiento. rm=registro de direccionamiento.
2º operando mediante reg: Debe ser un registro.

Des: componente desplazamiento de una dirección de memoria. 1 o 2 bytes.

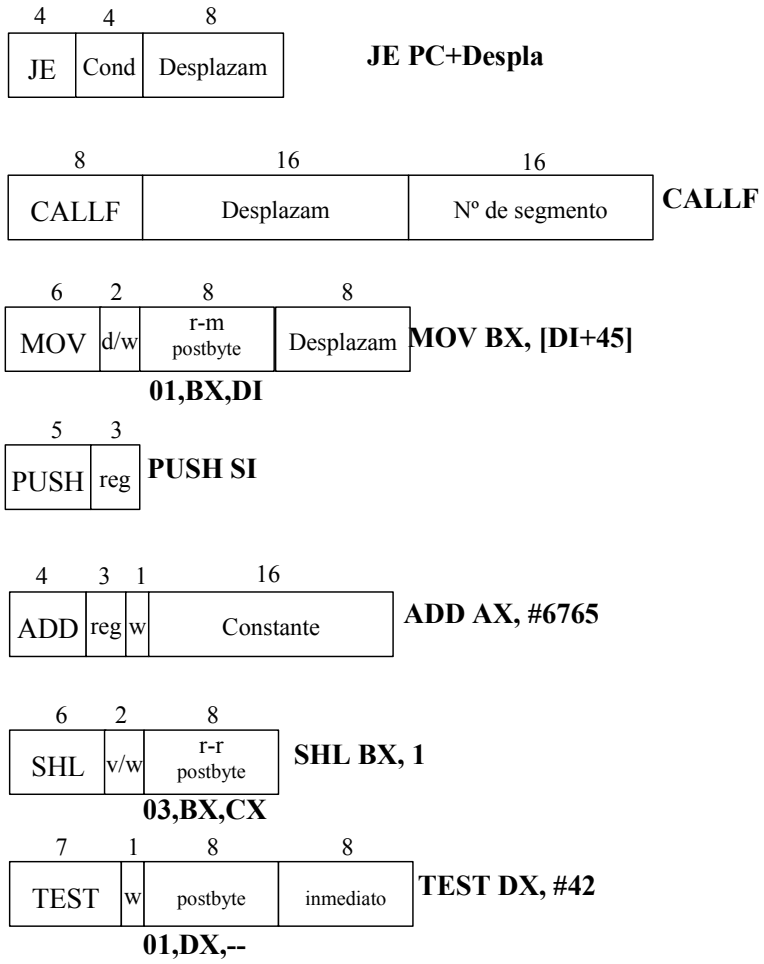
Val: valor inmediato. 1 o 2 bytes.

Hay cuatro codificaciones posibles para el postbyte:



Las **tres primeras** indican una instrucción **registro-memoria** donde **mem es el registro base**, la cuarta forma es registro registro.

Ejemplos:
La **codificación** de las instrucciones del **8086** es **compleja** con **muchos formatos** diferentes. La **instrucción puede variar** desde **1 byte hasta 6 bytes**.



En la figura **vemos algunas instrucciones** ejemplo.

El byte del código de operación **contiene habitualmente un bit w** que **indica** si la instrucción es de **palabra o de byte**.

d: en mov para instrucciones mem-reg o reg-mem indicando la dirección de la transferencia

v: en SHL indica un desplazamiento de longitud variable, indicando un registro que tiene la cuenta del desplazamiento.

2.7.4 La arquitectura DLX

DLX es una **sencilla arquitectura de carga almacenamiento**.

Se obtuvo el nombre para la máquina **del promedio de una serie de máquinas próximas** a **DLX**, expresado **en números romanos**.

AMD 29K, DECstation 3100, HP 850, IBM 801, Intel i860, MPS M/102ª, MIPS M/1000, Motorola 88K, RISC I, SGI 4D/60, SPARCstation-1, Sun-4/110, Sun-4/260.

La arquitectura **DLX se escogió basándose** en las **observaciones** sobre **las primitivas más frecuentes utilizadas** en los programas. Las funciones más sofisticadas se implementaban a nivel software con múltiples instrucciones.

DLX hace énfasis en:

- Un **sencillo repertorio de instrucciones** de carga almacenamiento
- **Diseño de segmentación eficiente** (pipelining)
- Un repertorio de **instrucciones fácilmente decodificables**
- **Eficiencia como objeto del compilador**

Características

Los registros

La arquitectura tiene **32 registros de propósito general GPR de 32 bits**; el valor de **R0 siempre es 0**.
Registros de **punto flotante** (**FPR**), se pueden utilizar como **32 registros de simple precisión (32 bits),** o como **parejas de doble precisión F_0 , F_2, ..., F_{28} , F_{30}** .
Hay también unos pocos **registros especiales** para acceder a la **información** sobre el **estado**, que se pueden transferir a y desde registros enteros (ej. Registro de estado de punto flotante).

La memoria

La memoria es **direccionable por bytes** en el modo <<**Big Endian**>> con una **dirección de 32 bits**. Todas **las referencias a memoria** se realizan **a través de cargas o almacenamientos entre memoria y** los **GPR o FPR**.
Los accesos que involucran a los GPR pueden realizarse **a un byte**, a **media palabra** y a **una palabra**.
Los accesos que involucran a los FPR pueden realizarse **a palabras en simple o doble precisión.**

Los accesos a memoria deben estar **alineados**.

Todas las instrucciones son **de 32 bits** y deben estar **alineadas**.

Operaciones:

- **Cargas y almacenamientos**
- **Operaciones ALU**
- **Saltos y bifurcaciones**
- Operaciones en **punto flotante**.

Cualquiera de los **registros GPR o FPR se puede cargar o almacenar** (**excepto R0**).

Modo único de direccionamiento: **registro base + desplazamiento de 16 bits con signo**.

El **formato de punto flotante** es el **IEEE 754**.

Tipo de instrucción. Cód de oper	**Significado de la instrucción**
Transferencia de datos	**Transfieren datos entre registros y memoria, o entre registros enteros y FP o registros especiales.**
LB, LBU, SB	**Carga byte**, carga byte sin signo, **almacena** byte
LH, LHU, SH	**Carga med pal**, carga med pal sin signo, **almacena** med pal
LW, SW	**Carga palabra**, **almacena** palabra
LF, LD, SF,SD	**Carga punto flotante** SP, carga punto flotante DP, almacena punto flotante SP, almacena punto flotante DP
MOVI2S, MOVS2I	**Transfiere** desde/ a GPR a/ desde un registro especial
MOVF, MOVD	Copia un registro de punto flotante a un par en DP
MOVFP2I, MOVI2FP	Transfiere 32 bits desde/a registros FP a/ desde registros enteros
Aritmético-lógicas	**Operaciones sobre datos enteros o lógicos en GPR.**
ADD, ADDI, ADDU, ADDUI	**Suma, suma inmediato** (todos los inmediatos son de 16 bits)
SUB, SUBI, SUBU, SUBUI	**Resta, resta inmediato** con y sin signo
MULT, MULTU, DIV, DIVU	**Multiplica y divide**, con signo y sin signo, los operandos deben estar en registros de punto flotante
AND, ANDI	**And**, and inmediato
OR, ORI, XOR, XORI	**Or**, or inmediato, or exclusiva, or exclusiva inmediata
LHI	Carga inmediato superior, carga la mitad superior de registro con inmediato
SLL, SRL, SRA, SLLI, SRLI, SRAI	**Desplazamientos**, **lógicos** dere izqu, aritméticos derecha
Control	**Saltos y bifurcaciones condicionales; relativos al PC o mediante registros.**
BEQZ, BNEZ	**Salto GPR igual/no igual a cero**, despla 16 bits
BFPT, BFPF	**Test de bit de comparación** reg estado FP y salto, despla 16
J, JR	**Bifurcaciones**: desplazamiento de 26 bits
JAL, JALR	Bifurcación y enlace
TRAP	Transfiere a S.O. a una dirección vectorizada
RFE	Volver a código de usuario desde una excepción
Punto flotante	**Operaciones en punto flotante en formatos DP y SP**
ADDD, ADDF	**Suma números** DP, SP
SUBD, SUBF	**Resta números** DP, SP
MULTD, MULTF	**Multiplica punto flotante** DP, SP
DIVD, DIVF	**Divide punto flotante** DP, SP
CVTF2D, CVTF2I, CVTD2F, CVTD2I, CVTI2F, CVTI2D	Convierte instrucciones
_____D, _____F	Compara DP, SP

Todas las **instrucciones de la ALU son** instrucciones **registro-registro**, incluyendo:

Aritméticas
Lógicas (suma, resta, and, or, xor).
Desplazamientos

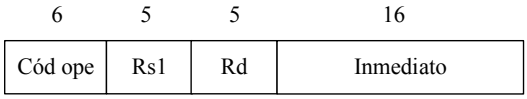
Se proporcionan las formas inmediatas de todas estas instrucciones.

Las instrucciones de **punto flotante manipulan los registros FPR** e **indican si** la operación **es de simple o doble precisión**. **Las de simple precisión se pueden aplicar a todos los registros** mientras que **las de doble** se aplican **a parejas par impar** (designadas por el número de registro par).

Formatos de instrucción

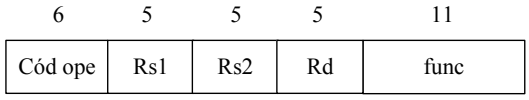
Todas las **instrucciones son de 32 bits** con un **código de operación de 6 bits.**

Instrucción tipo I



- Cargas y almacenamientos (byte, media palabra, palabra)
- ALUs con operandos inmediatos
- Instrucciones de salto condicional (BEQZ, BNEZ)
  Rs1 registro implicado Rd no se utiliza
- Saltos a registro
  Rd=0; Inmediato=0; Rs1=destino

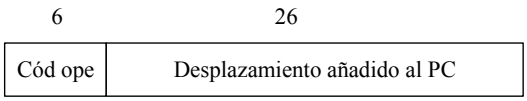
Instrucción tipo R



- Aritméticas y lógicas entre registros

  Rs1= fuente1
  Rs2= fuente2
  Rd= Registro destino
  Fun.= operación del flujo de datos

Instrucción tipo J



- Instrucciones de salto
  - Desplazamiento 26 bits con signo añadido al PC
    JAL Salto incondicional y enlace (R31)
    J Salto incondicional
    Trap Interrupciones