Módulo 3: Arquitectura del repertorio de instrucciones

Curso 2013-2014

Índice

- Introducción
- Modos de direccionamiento
- Tipos y tamaños de operandos
- Tipos de operaciones
  - Instrucciones de control de flujo
- Codificación del repertorio de instrucciones
- Ejemplo: La arquitectura MIPS

- **Bibliografía:**
  - John L. Hennessy & David A. Patterson , "Computer Architecture: A Quantitative Approach", Morgan Kaufmann, Apéndice B 4ed (2007), Apéndice A 5ed (2012).
    - "Computer Architecture and Amdahl's Law", Gene M. Amdahl, IEEE Computer, pp38-46. DOI: 10.1109/MC.2013.418

Introducción

- ¿Qué es?
  - *La parte del computador visible al programador o al compilador. (Hennessy and Patterson, "Computer Architecture: A Quantitative Approach")*
- Algunas cuestiones:
  - ¿Qué decisiones son relevantes para diseñar una Arquitectura del Repertorio de Instrucciones (ISA, *Instruction Set Architecture*)?
  - ¿Qué importancia tiene el área de aplicación en el diseño del ISA?
  - Relación ISA - compilador
  - ¿Cómo se toman las decisiones con respecto a número de registros, modos de direccionamiento y operaciones?

Introducción

- Decisiones relevantes:
  - Tipos de instrucciones
  - Modos de direccionamiento
  - Número de operandos explícitos
  - Tipos de datos
  - Elementos de almacenamiento accesibles al programador:
    - Registros de propósito específico
    - Banco de registros
    - Memorias

Introducción

- Áreas de aplicación:
  - **Escritorio** (Desktop): Se centra en el rendimiento de programas que usan tipos de datos enteros o en coma flotante, con poco énfasis en el tamaño del programa
  - **Servidor**: Se usan principalmente bases de datos, servidores de ficheros y aplicaciones web. La coma flotante es menos importante que los enteros o las cadenas
  - **Tecnología móvil y dispositivos empotrados**: Se centran en coste y energía, por lo que el tamaño de los programas es importante (menos memoria y energía). Tipos de datos como coma flotante pueden llegar a ser opcionales

Introducción

- Relación ISA - compilador
  - El desarrollo de aplicaciones, en su mayor parte, no se desarrolla ya en lenguaje máquina
  - El papel del compilador es crítico en la determinación del conjunto de instrucciones que usa mayoritariamente la CPU
  - Inversamente, la elección de una arquitectura impacta directamente en la complejidad del compilador y en el código resultante

Introducción

- ¿Cómo se toman las decisiones con respecto a número de registros, modos de direccionamiento y operandos?
  - Objetivos: Rendimiento, coste, consumo, etc

    Tej = N * CPI * Tciclo
  - Compromiso: Más variedad implica mayor coste
  - Estudio de la frecuencia de uso
  - Ley de Amdahl

Introducción

- Ley de Amdahl
  - La mejora de rendimiento obtenida al usar algún modo de ejecución más rápido está limitada por la fracción de tiempo en que este modo de ejecución puede ser usado.

Speedup=Tiempo de ejecución sin mejora/Tiempo de ejecución con mejora

$$T_{ej\_new} = T_{ej\_old} * ((1 - fracción_{mejorada}) + \frac{fracción_{mejorada}}{speed_{mejorado}}$$

$fracción_{mejorada}$ es la parte (fracción) del tiempo de cálculo del computador original que puede ser mejorada.

$speedup_{mejorado}$ indica cuánto más rápida es la parte mejorada respecto al original

Introducción

- Ejemplo de aplicación de la ley de Amdahl:
  - Supongamos que un programa se ejecuta en 100 ns en un computador, de los cuales 80 ns se dedican a las multiplicaciones. ¿Cuánto debo mejorar la velocidad de la multiplicación si quiero que mi programa se ejecute cinco veces más rápido?
- Respuesta:

$$T_{ej\_new} = T_{ej\_old} * ((1 - fracción_{mejorada}) + \frac{fracción_{mejorada}}{speed_{mejorado}}$$

20 ns = 100 * ((1-0,8)+0,8/x) =20 + 80/x

0 = 80/x

No es posible conseguirlo

Modos de direccionamiento

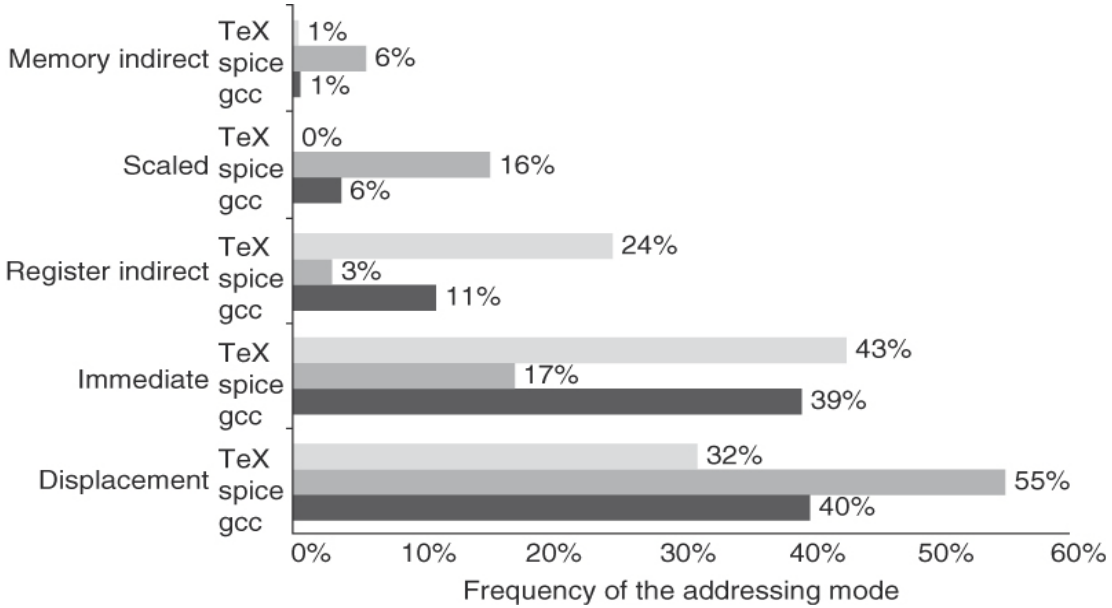
- Indican cómo se define la dirección donde se encuentra un operando
- Están disponibles en una gran variedad
- Características de los modos de direccionamiento:
  - Espacio que ocupan en la instrucción (codificación)
  - Capacidad de direccionamiento
  - Velocidad de acceso a los datos
  - Frecuencia de uso

Modos de direccionamiento

Modo	Ejemplo	Significado	Uso
Registro	Add R4, R3	Regs[R4] ← Regs[R4] + Regs[R3]	Cuando el valor está en un registro
Inmediato	Add R4, #3	Regs[R4] ← Regs[R4] + 3	Para constantes
Con desplazamiento	Add R4, 100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]	Acceso a variables locales
Indirecto de registro	Add R4, (R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Acceso usando punteros
Indexado	Add R3, (R1+R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]	Indexado de arrays, con R1 dir. base del array
Directo o absoluto	Add R1, (1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Datos estáticos (la constante de desplazamiento puede ser muy grande)
Indirecto a memoria	Add R1, @(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	Si R3 es la dirección de un puntero, tenemos *p
Autoincremento	Add R1, (R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + d	Paso secuencial en arrays en un bucle. 'd' es el tamaño de paso. Útil para implementar push/pop en pila
Autodecremento	Add R1, -(R2)	Regs[R2] ← Regs[R2] - d Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	
Escalado	Add R1, 100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100 + Regs[R2] + Regs[R3]*d]	Indexado de arrays (implementa cualquier modo indexado)

Modos de direccionamiento

- ¿Algunos de ellos son más relevantes o útiles?
  - Los modos de direccionamiento complejos tienen el potencial de reducir el número de instrucciones en un programa pero incrementan el CPI
  - Recordar: alineamiento y ordenamiento (little-big endian)



Uso de distintos modos de direccionamiento en tres programas en la arquitectura VAX.

Modos de direccionamiento

- La elección del **rango de desplazamiento** o del **tamaño del campo inmediato** se realiza tras estudios de uso en programas tipo.
  - Esta elección es importante porque afecta a la longitud de las instrucciones

- En el diseño de una arquitectura del repertorio de instrucciones cabe esperar:
  - Soportar al menos los modos de direccionamiento:
    - Registro
    - Con desplazamiento
    - Inmediato
    - Indirecto de registro
      - Estos modos representan del 75% al 99% de los modos de direccionamiento
  - Tamaño del desplazamiento debería ser de 12 a 16 bits
    - Estos tamaños capturan del 75% al 99% de los desplazamientos
  - Tamaño del valor inmediato de 8 a 16 bits
    - El 80% de los valores inmediatos se capturan con 16 bits y el 50% con 8 bits

Tipos y tamaños de operandos

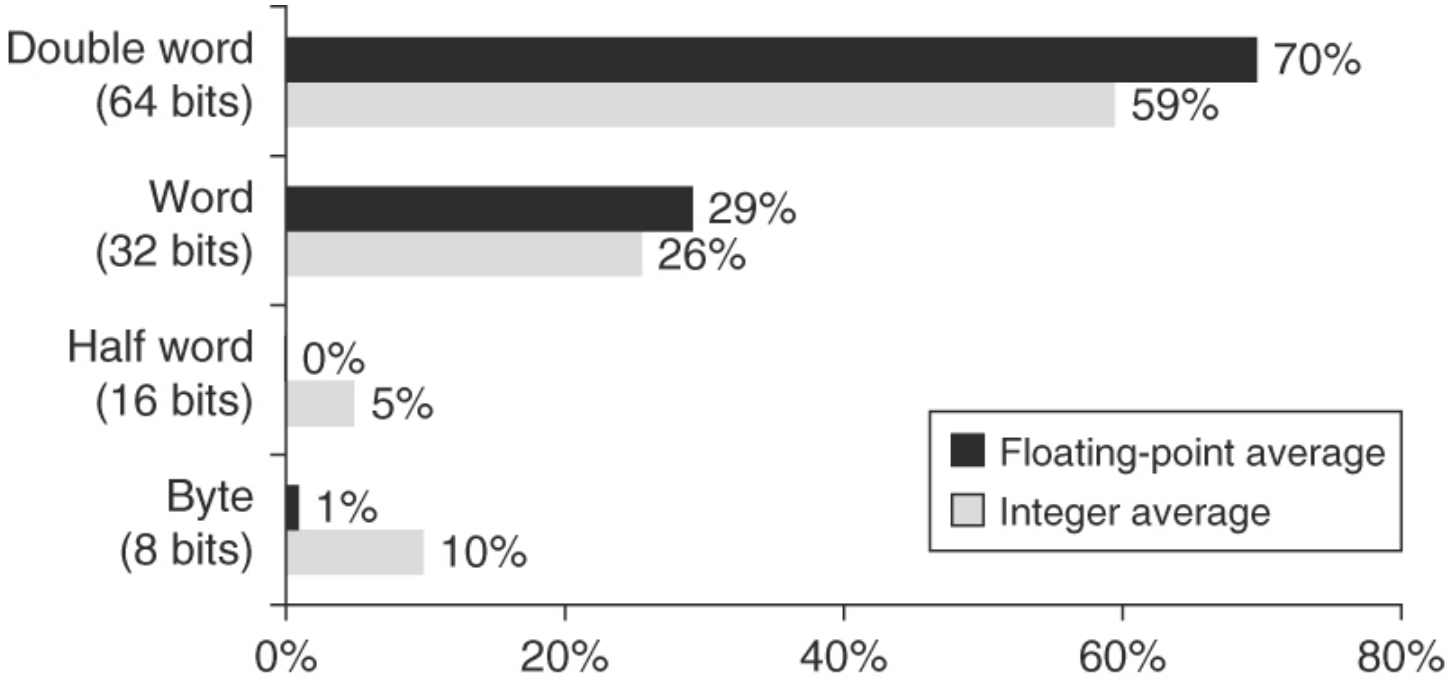
- El tipo de operando se suele codificar en el código de operación.

- Aplicaciones de propósito general:
  - Normalmente el tipo de dato (entero, coma flotante simple precisión, carácter, etc.) define explícitamente su tamaño
    - Carácter: 8 bits
    - Media palabra: 16 bits
    - Palabra: 32 bits
    - Coma flotante, simple precisión: palabra, 32 bits
    - Coma flotante, doble precisión: 2 palabras, 64 bits
  - Enteros: casi universalmente representados en C2
  - Caracteres usualmente en ASCII, pero el Unicode de 16 bits (usado en Java) está ganando popularidad
  - Coma flotante
    - Hasta los 80s cada empresa usaba su formato
    - A partir de ahí casi todos siguen el estándar IEEE 754

Tipos y tamaños de operandos

- Aplicaciones de negocios
  - Algunas arquitecturas dan soporte al formato decimal, comúnmente llamado BCD (binary coded decimal)
  - Se usan 4 bits para codificar un valor de 0 a 9 y cada byte almacena dos dígitos
  - ¿Por qué usar operandos decimales?
    - Hay fracciones que no tienen una representación exacta en binario
      - $0.10_{10} = 0.0001100110011_2$ ….
    - Cálculos que son exactos en decimal son inexactos en binario, lo que supone un problema en transacciones financieras

Tipos y tamaños de operandos



Distribución de acceso a datos por tamaño para los SPEC CPU 2000 ALPHA

Como la doble palabra se usa para direcciones, el porcentaje en los SPECint es elevado

Tipos de operaciones

- **Aritmético-lógicas**: Aritmética entera y operaciones lógicas: suma, resta, y lógica, o lógica, multiplicación, división
- **Transferencia:** Carga, almacenamiento e instrucciones de movimiento de datos en computadores memoria-memoria
- **Control:** Bifurcaciones condicionales, saltos, llamadas a procedimientos, retorno de procedimientos, excepciones
- **Sistema:** Llamadas al sistema operativo, instrucciones de gestión de memoria virtual
- **Coma flotante:** Operaciones en coma flotante: suma, multiplicación, división, comparación
- **Decimal:** suma decimal, multiplicación decimal
- **Cadena:** Movimiento, comparación, búsqueda
- **Gráficos:** Operaciones con píxeles, operaciones vectoriales, operaciones de compresión/descompresión

  – Todos los computadores soportan al menos las tres primeras categorías (y algunas instrucciones mínimas de llamada al sistema)

Tipos de operaciones

Posición	Instrucción 80x86	Promedio
1	Carga	**22%**
2	Salto condicional	**20%**
3	Comparación	16%
4	Almacenamiento	13%
5	Suma	8%
6	Y	6%
7	Resta	5%
8	Transferencia registro-registro	4%
9	Llamada a procedimiento	1%
10	Retorno de procedimiento	1%
Total		**96%**

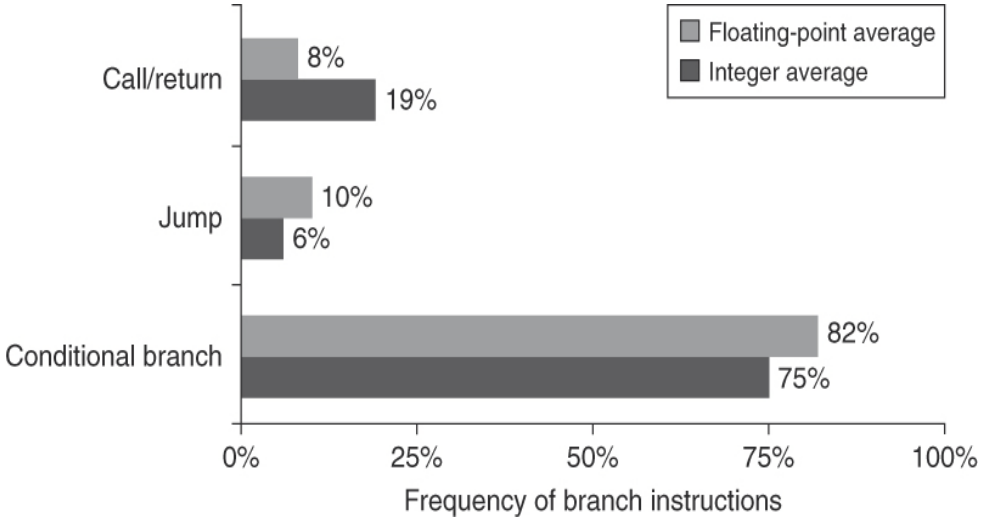
Resultados de los SPECint92

Programas de aritmética entera ejecutados en una arquitectura Intel 80x86.

El 96% del código está compuesto por 10 instrucciones "simples"

Instrucciones de control de flujo

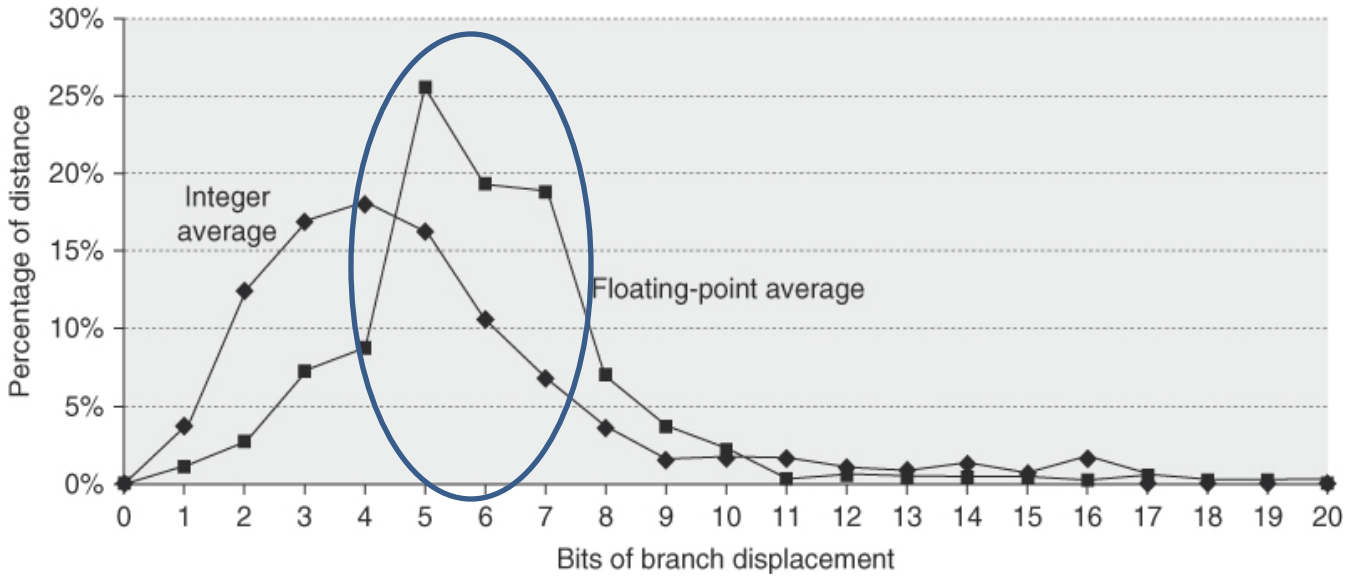
- Tres categorías básicas
  - Salto condicional
  - Salto
  - Llamada y retorno de procedimientos / subrutinas
- ¿Cuál es la frecuencia relativa de estas tres categorías en las instrucciones de control?



SPEC CPU 2000 en arquitectura Alpha

Instrucciones de control de flujo

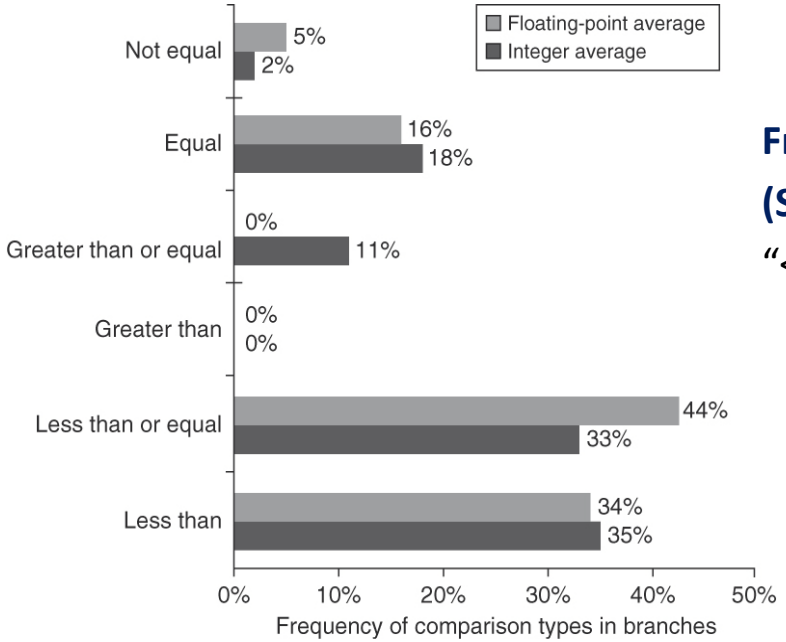
- Modos de direccionamiento en las instrucciones de salto condicional
  - Direccionamiento relativo al PC
    - Seguramente se salte a una posición cercana al PC
    - ¿Cuántos bits son necesarios para indicar la dirección relativa?



SPEC CPU 2000 en arquitectura Alpha (load-store)

Instrucciones de control de flujo

- Saltos condicionales
  - Al ser los más frecuentes deberían ser los más rápidos
    - Es importante obtener cuanto antes la "solución" del test
    - Generación de la condición
      - Registro de estado
      - Registro de propósito general
      - Comparar y saltar
    - ¿Cuántos tipos de condiciones son necesarias?



Frecuencia de condiciones en los saltos condicionales (SPEC CPU 2000 en Alpha)

"<", "<=" y "==" son las dominantes

Codificación del repertorio de instrucciones

- Desde una perspectiva de compilador obtenemos las siguientes conclusiones en cuanto a los elementos más interesantes para un ISA:
  - Arquitectura registro-registro
  - Modos de direccionamiento: con desplazamiento, inmediato e indirecto de registro
  - Operaciones: Operaciones “simples”, saltos condicionales y llamadas y retornos de procedimientos
  - El ancho de los datos en 8-, 16-, 32- y 64- bits para unidad entera y 32- y 64- para coma flotante

- Obviamente, estas decisiones afectarán directamente a la codificación de las instrucciones

Codificación del repertorio de instrucciones

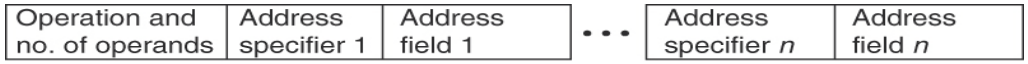
- La codificación afecta a:
  - Tamaño del programa compilado
  - Implementación del propio procesador, que debe decodificar y ejecutar estas instrucciones lo más rápidamente posible
- La operación a realizar se codifica en un campo llamado código de operación (opcode)
- La decisión más relevante se toma al codificar los modos de direccionamiento
  - Esta decisión depende del número de modos de direccionamiento diferentes
  - … y del grado de dependencia entre operaciones y modos
    - Algunos computadores antiguos llegan a tener hasta 5 operandos y 10 MDs posibles por operando ⇒ Un campo específico por operando para indicar el MD usado para ese operando
    - En el otro extremo se encuentran las arquitecturas load-store con sólo uno o dos MDs (en este caso el modo suele estar implícito en el opdoce)

Codificación del repertorio de instrucciones

- La arquitectura debe equilibrar distintas tendencias a la hora de codificar un ISA:
  - El deseo de tener tantos registros y modos de direccionamiento como puedan ser necesarios
  - El impacto del tamaño del campo registro y campo modo de direccionamiento en el tamaño medio de la instrucción y de aquí en el tamaño medio de los programas
  - El deseo de tener tamaños de instrucción fácilmente manejables en un procesador segmentado
  - Las arquitecturas actuales se decantan por usar un tamaño de instrucción fijo que proporciona implementaciones mas eficientes en términos de rendimiento y costo, sacrificando en alguna medida el tamaño medio de los programas

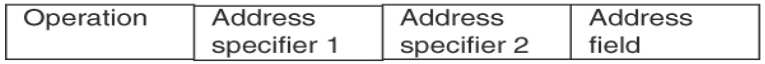
Codificación del repertorio de instrucciones

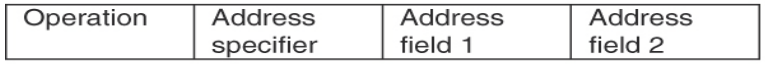
- Tres variaciones básicas en la codificación de instrucciones.
  - Longitud variable: Soporta cualquier número de operandos y MDs. Menor tamaño medio de programas.
  - Longitud fija: Mínimo número de operandos y MDs (normalmente el MD no se codifica, viene implícito en el OPCODE). Programas más largos.
  - Híbrido: El que tenga longitud fija o variable depende generalmente del OPCODE.



(a) Variable (e.g., Intel 80x86, VAX)

(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)





(c) Hybrid (e.g., IBM 360/370, MIPS16, Thumb, TI TMS320C54x)

Codificación del repertorio de instrucciones

- RISC: Reduced-Instruction Set Computer
  - Término acuñado por Patterson al principio de los 80
  - Primeros diseños:
    - Berkeley RISC-I (Patterson)
    - Stanford MIPS (Hennessy)
    - IBM 801 (Cocke)
  - Manifiesto RISC: “Crear ISAs que …”
    - Simplifiquen el diseño de los procesadores
    - Faciliten la optimización del compilador
    - Repertorio de instrucciones relativamente simple
    - Modos de direccionamiento asociados a la instrucción
    - Formatos de instrucción reducidos
    - Hacer rápido lo más usado
  - Ejemplos: PowerPC, ARM, SPARC, Alpha, PA-RISC

- CISC: Complex-Instruction Set Computer
  - El término no existía hasta que apareció “RISC”
  - Ejemplos: x86, VAX, Motorola 68000, IBM 360/ 370...

Codificación del repertorio de instrucciones

- CISC/RISC: Ventajas e inconvenientes
  - Ecuación de rendimiento:
    - $T_{CPU} = N * CPI * t$
  - CISC
    - Reduce N usando instrucciones complejas, de tamaño variable
    - Aumenta CPI
    - Aumenta t
      - Instrucciones más complejas tienden utilizar mayor tiempo de ciclo
  - RISC
    - Reduce CPI
      - Instrucciones más sencillas permiten reducir el tiempo de ciclo
    - Reduce t
    - Aumenta N
      - Nota: Los compiladores tienen una labor más sencilla ⇒ Mejor optimización de código

Ejemplo: La arquitectura MIPS

- MIPS: Arquitectura tipo load-store sencilla de 64 bits
  - Primer procesador MIPS en 1985
  - Describimos un subconjunto llamado MIPS64
    - Un repertorio de instrucciones simple tipo load-store
    - Diseñado para obtener una segmentación eficiente
    - Facilidad de optimización por el compilador

Ejemplo: La arquitectura MIPS

- Banco de registros
  - 32 registros / 64 bits de propósito general (GPRs)
    - R0, R1, …, R31, el valor de R0 es siempre 0
    - También llamados registros “enteros”
  - 32 registros / 64 bits para aritmética flotante (FPRs)
    - F0, F1, …, F31
    - Simple precisión (32 bits) o doble precisión (64 bits)
    - Soporte para operaciones de simple y doble precisión
- Memoria
  - Memoria direccionable por bytes con dirección de 64 bits
    - Tiene un bit de modo para seleccionar el alineamiento (little o big endian)
    - Las palabras en memoria están alineadas

Ejemplo: La arquitectura MIPS

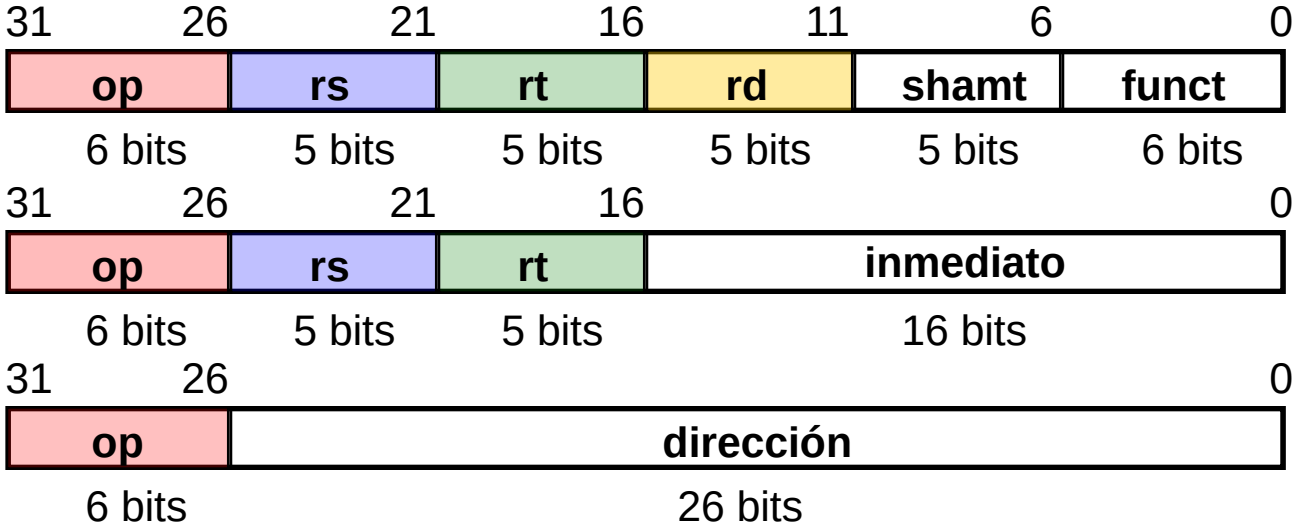
- Tipos de datos
  - 8-, 16-, 32- y 64-bits para aritmética entera
  - 32- y 64-bits para coma flotante (IEEE 754)
  - Las operaciones en MIPS64 operan con enteros de 64 bits o en coma flotante de 32 y 64 bits
    - Operandos enteros de 8-, 16- y 32-bits se rellenan con ceros o con el bit de signo para completar los 64 bits y operar

Ejemplo: La arquitectura MIPS

- Modos de direccionamiento
  - Sólo tres modos de direccionamiento
    - De registro (campo de 5 bits indicando el registro)
    - Inmediato (campo de 16 bits en la instrucción)
    - Con desplazamiento (desplazamiento de 16 bits en instrucción)
      - MD indirecto de registro se implementa escribiendo “0” en el campo desplazamiento
      - MD absoluto se implementa usando el registro R0 e indicando la dirección absoluta en el campo desplazamiento

Ejemplo: La arquitectura MIPS

- Formatos de instrucción:



  - Pocos modos de direccionamiento ⇒ van codificados en el OPCODE
  - Instrucciones de 32 bits con un OPCODE de 6 bits

Ejemplo: La arquitectura MIPS

- Operaciones
  - Cuatro grandes grupos
    - Load/store
    - Operaciones en ALU
    - Saltos/bifurcaciones
    - Coma flotante
      - Todos los registros pueden ser utilizados en load/store, salvo el R0 (load sin efecto alguno).

Ejemplo: La arquitectura MIPS

- Operaciones Load/store del MIPS64

Example instruction	Instruction name	Meaning
LD R1,30(R2)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[30+Regs[R2]]
LD R1,1000(R0)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[1000+0]
LW R1,60(R2)	Load word	Regs[R1] $\leftarrow_{64}$ (Mem[60+Regs[R2]]$_0$)^{32} ## Mem[60+Regs[R2]]
LB R1,40(R3)	Load byte	Regs[R1] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{56} ## Mem[40+Regs[R3]]
LBU R1,40(R3)	Load byte unsigned	Regs[R1] $\leftarrow_{64}$ 0^{56} ## Mem[40+Regs[R3]]
LH R1,40(R3)	Load half word	Regs[R1] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{48} ## Mem[40+Regs[R3]] ## Mem[41+Regs[R3]]
L.S F0,50(R3)	Load FP single	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D F0,50(R2)	Load FP double	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R2]]
SD R3,500(R4)	Store double word	Mem[500+Regs[R4]] $\leftarrow_{64}$ Regs[R3]
SW R3,500(R4)	Store word	Mem[500+Regs[R4]] $\leftarrow_{32}$ Regs[R3]$_{32..63}$
S.S F0,40(R3)	Store FP single	Mem[40+Regs[R3]] $\leftarrow_{32}$ Regs[F0]$_{0..31}$
S.D F0,40(R3)	Store FP double	Mem[40+Regs[R3]] $\leftarrow_{64}$ Regs[F0]
SH R3,502(R2)	Store half	Mem[502+Regs[R2]] $\leftarrow_{16}$ Regs[R3]$_{48..63}$
SB R2,41(R3)	Store byte	Mem[41+Regs[R3]] $\leftarrow_{8}$ Regs[R2]$_{56..63}$

Ejemplo: La arquitectura MIPS

- Operaciones en ALU
  - Todas son instrucciones registro-registro
  - El registro R0 se usa “como comodín” para implementar algunas instrucciones a partir de otras: Contiene “cero”
    - Ej: Para implementar MOVER R1 a R2 podemos usar: ADD R2, R1, R0
- Instrucciones en coma flotante
  - Los datos en coma flotante / simple precisión ocupan medio registro.
  - El formato usado es el IEEE 754
  - Trabajan con los registros de coma flotante
  - Indican cuándo operar en simple o doble precisión
    - Ej.: MOV.S (simple) MOV.D (doble)
  - Para mejorar el rendimiento de rutinas gráficas, MIPS64 implementa instrucciones para realizar dos operaciones simultáneas en precisión simple
    - Cada una en una parte de los registros en coma flotante de 64 bits

Ejemplo: La arquitectura MIPS

- Instrucciones de salto y salto condicional
  - La condición de salto se indica en la propia instrucción

Example instruction		Instruction name	Meaning
J	name	Jump	$PC_{36..63} \leftarrow$ name
JAL	name	Jump and link	Regs[R31]←PC+8; $PC_{36..63} \leftarrow$ name; $((PC + 4)-2^{27}) \leq$ name $< ((PC+4)+2^{27})$
JALR	R2	Jump and link register	Regs[R31]←PC+8; PC←Regs[R2]
JR	R3	Jump register	PC←Regs[R3]
BEQZ	R4,name	Branch equal zero	if (Regs[R4]==0) PC←name; $((PC+4)-2^{17}) \leq$ name $< ((PC+4)+2^{17})$
BNE	R3,R4,name	Branch not equal zero	if (Regs[R3]!= Regs[R4]) PC←name; $((PC+4)-2^{17}) \leq$ name $< ((PC+4)+2^{17})$
MOVZ	R1,R2,R3	Conditional move if zero	if (Regs[R3]==0) Regs[R1]←Regs[R2]