Principios del ISA
Lección 5

Prof.Ing. Fabián Zamora Ramírez

CE-4301 Arquitectura de Computadores I

Área de Ingeniería en Computadores

Instituto Tecnológico de Costa Rica

Agenda

Tipos de operaciones

Existen varios tipos de operaciones soportadas por las arquitecturas. Las más comunes son:

- Aritméticas y lógicas.
- Transferencia de datos (Load-Store)
- Control de flujo.

Algunas arquitecturas presentan operaciones avanzadas para uso de strings o imágenes (tratamientos sobre pixeles, etc)

Operaciones

Como regla general en una arquitectura, las instrucciones que se utilizan en mayor medida son las más simples y rápidas.

Rank	80x86 instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
Total		**96%**

Figure: Top 10 instrucciones más utilizadas en 8086

Resumen de tipo de operaciones

Operator type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, subtract, and, or, multiply, divide
Data transfer	Loads-stores (move instructions on computers with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating point	Floating-point operations: add, multiply, divide, compare
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel and vertex operations, compression/decompression operations

Instrucciones de control de flujo

Instrucciones pueden cambiar el flujo lineal de ejecución de un programa. Cuatro tipos principales:

- Branches Condicionales
- Saltos (Jumps)
- Llamadas a procedimientos
- Retornos de procedimientos

Modos de direccionamiento para Control de Flujo

La dirección destino de una instrucción de control de flujo **siempre** debe ser especificada: explícita o implícitamente.

Forma más común: **Dirección relativa al PC**.

- Requiere menos bits para especificar dirección.
- Dirección generalmente se encuentra cercana
- Permite que el código se ejecute independientemente de la plataforma.**

Modos de direccionamiento para Control de Flujo

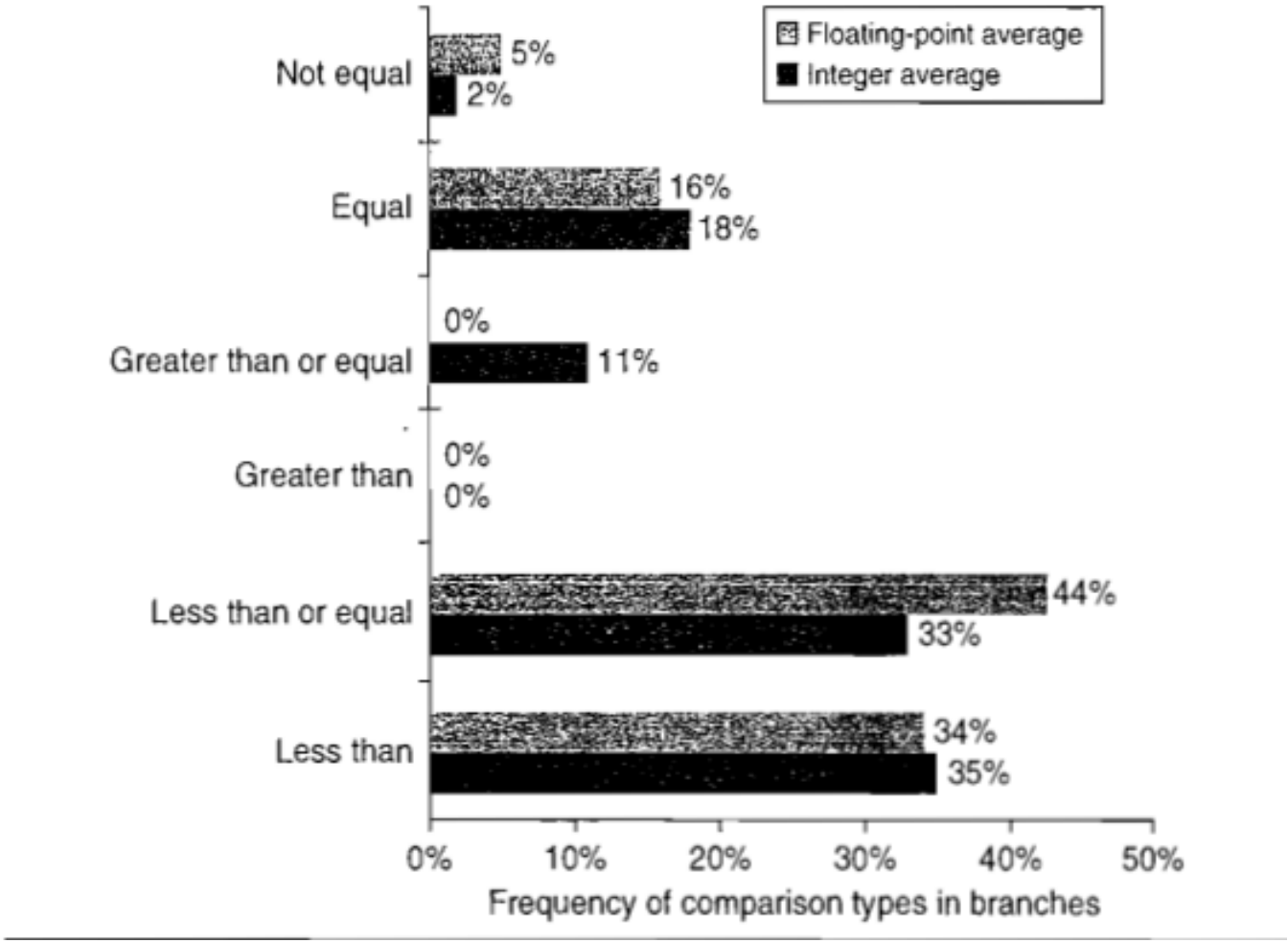
En algunos casos no es posible utilizar direcciones relativas al PC: La dirección de "salto" **NO es conocida** en tiempo de compilación.

- Saltos indirectos
- Retornos de llamadas a proc.

Se debe especificar dinámicamente la dirección: Uso de **registros** para almacenar la dirección.

Útil en caso de estructuras *case-switch*, punteros a funciones, bibliotecas dinámicas*.

Frecuencia en branches



Encodificación del set

Todos los aspectos mencionados anteriormente afectan en cómo se codifican finalmente las instrucciones a binario.

Las decisiones en encodificación afectarán parámetros de implementación del set (área, complejidad, potencia)

Encodificación

La mayoría de bits en una instrucción se aplican en modos de direccionamiento y nombre de registros.

La encodificación debe considerar:

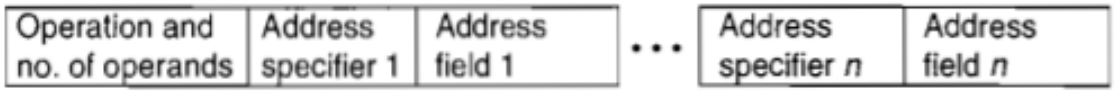
- Capacidad de contar con registros y modos de direccionamiento diferentes.
- El impacto del tamaño de los registros y de los modos en el tamaño de la instrucción y del programa.
- Instrucciones encodificadas con longitudes fijas o variables (pipeline, determinismo, etc).

Formas típicas de encodificación

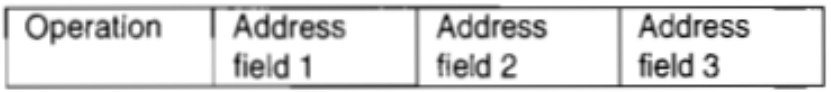
Comercialmente se implementan 3 formas de encodificar el set.

- **Longitud variable**: Común en arquitecturas CISC. Incluye múltiples métodos de direccionamiento y operaciones.
- **Longitud fija**: Arquitecturas RISC, combina modo de direccionamiento y operación en el *opcode*
- **Hibrido**: Múltiple tamaño de instrucción, pero reduciendo la cantidad de operaciones y modos de direccionamiento. Ventaja: **Reducir el código**

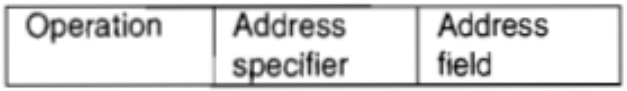
Ejemplos Tipos de encodificacíon

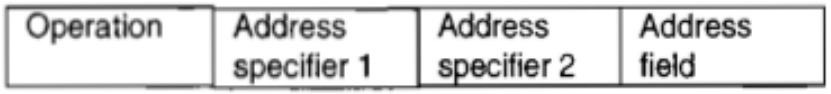


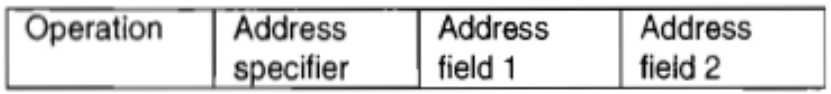
(a) Variable (e.g., Intel 80x86, VAX)



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







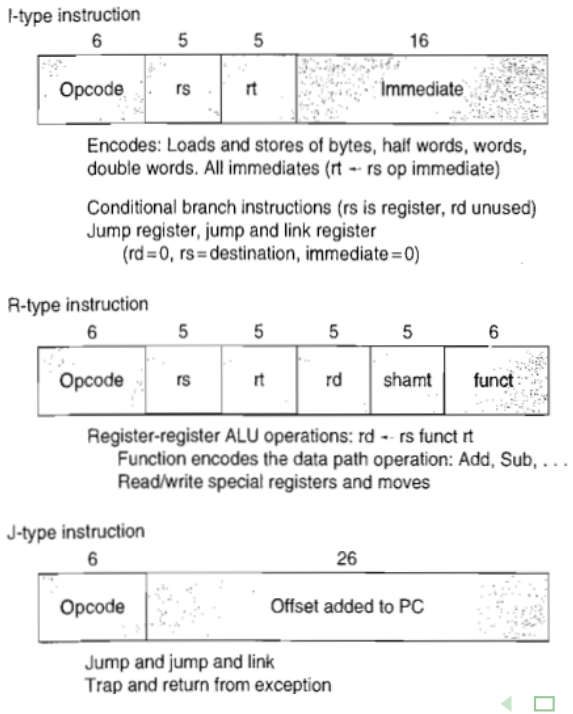
(c) Hybrid (e.g., IBM 360/370, MIPS16, Thumb, TI TMS320C54x)

MIPS Generalidades

- **Clase**: Arquitectura Load/Store con registros de propósito general.
- **Modos de dir**: Desplazamiento (offset de 12 a 16 bits), inmediato (8-16 bits) y registro indirecto.
- **Tipos de datos:** Enteros 8,16,32,64 bits, Flotantes IEEE 754
- **Operaciones**: Instrucciones simples (load,store,add,sub,move reg, shift)
- **Control de flujo:** Comparaciones igual, no igual, menor, branches (relativas al PC), saltos, llamadas y retornos.
- **Encodificación**: Longitud Fija, puede presentar hibrida (MIPS16)

MIPS

- 32 registros de propósito general: R0-R31. R0 tiene un valor de 0.
- Método de direccionamiento inmediato y desplazamiento (simula reg indirecto y directo)
- 3 Formatos de instrucción:



http://www.cs.cornell.edu/courses/cs316/2007fa/MIPS_Vol2.pdf

MIPS32® Instruction Set Quick Reference

R_D — Destination register
R_S, R_T — Source operand registers
R_A — Return address register (R31)
PC — Program counter
Acc — 64-bit accumulator
Lo, Hi — Accumulator low ($Acc_{31:0}$) and high ($Acc_{63:32}$) parts
± — Signed operand or sign extension
∅ — Unsigned operand or zero extension
:: — Concatenation of bit fields
R2 — MIPS32 Release 2 instruction
DOTTED — Assembler pseudo-instruction

Please refer to *"MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set"* for complete instruction set information.

Arithmetic Operations			
ADD	R_D, R_S, R_T	$R_D = R_S + R_T$	(overflow trap)
ADDI	R_D, R_S, const16	$R_D = R_S + const16^{\pm}$	(overflow trap)
ADDIU	R_D, R_S, const16	$R_D = R_S + const16^{\pm}$	
ADDU	R_D, R_S, R_T	$R_D = R_S + R_T$	
CLO	R_D, R_S	R_D = CountLeadingOnes(R_S)	
CLZ	R_D, R_S	R_D = CountLeadingZeros(R_S)	
LA	R_D, label	R_D = Address(label)	
LI	R_D, imm32	R_D = imm32	
LUI	R_D, const16	$R_D = const16 << 16$	
MOVE	R_D, R_S	$R_D = R_S$	
NEGU	R_D, R_S	$R_D = -R_S$	
SEB[R2]	R_D, R_S	$R_D = R_{S7:0}^{\pm}$	
SEH[R2]	R_D, R_S	$R_D = R_{S15:0}^{\pm}$	
SUB	R_D, R_S, R_T	$R_D = R_S - R_T$	(overflow trap)
SUBU	R_D, R_S, R_T	$R_D = R_S - R_T$	

Shift and Rotate Operations		
ROTR[R2]	R_D, R_S, bits5	$R_D = R_{S bits5-1:0} :: R_{S 31:bits5}$
ROTRV[R2]	R_D, R_S, R_T	$R_D = R_{S RT4:0-1:0} :: R_{S 31:RT4:0}$

Logical and Bit-Field Operations		
AND	R_D, R_S, R_T	$R_D = R_S \& R_T$
ANDI	R_D, R_S, const16	$R_D = R_S \& const16^{\varnothing}$
EXT[R2]	R_D, R_S, P, S	$R_S = R_{SP+S-1:P}^{\varnothing}$
INS[R2]	R_D, R_S, P, S	$R_{DP+S-1:P} = R_{S S-1:0}$
NOP		No-op
NOR	R_D, R_S, R_T	$R_D = \sim(R_S \mid R_T)$
NOT	R_D, R_S	$R_D = \sim R_S$
OR	R_D, R_S, R_T	$R_D = R_S \mid R_T$
ORI	R_D, R_S, const16	$R_D = R_S \mid const16^{\varnothing}$
WSBH[R2]	R_D, R_S	$R_D = R_{S23:16} :: R_{S31:24} :: R_{S7:0} :: R_{S15:8}$
XOR	R_D, R_S, R_T	$R_D = R_S \oplus R_T$
XORI	R_D, R_S, const16	$R_D = R_S \oplus const16^{\varnothing}$

Condition Testing and Conditional Move Operations		
MOVN	R_D, R_S, R_T	if $R_T \neq 0$, $R_D = R_S$
MOVZ	R_D, R_S, R_T	if $R_T = 0$, $R_D = R_S$
SLT	R_D, R_S, R_T	$R_D = (R_S^{\pm} < R_T^{\pm})$? 1 : 0
SLTI	R_D, R_S, const16	$R_D = (R_S^{\pm} < const16^{\pm})$? 1 : 0
SLTIU	R_D, R_S, const16	$R_D = (R_S^{\varnothing} < const16^{\varnothing})$? 1 : 0
SLTU	R_D, R_S, R_T	$R_D = (R_S^{\varnothing} < R_T^{\varnothing})$? 1 : 0

Multiply and Divide Operations		
DIV	R_S, R_T	$Lo = R_S^{\pm} / R_T^{\pm}$; $Hi = R_S^{\pm}$ mod $R_T^{\pm}$
DIVU	R_S, R_T	$Lo = R_S^{\varnothing} / R_T^{\varnothing}$; $Hi = R_S^{\varnothing}$ mod $R_T^{\varnothing}$
MADD	R_S, R_T	$Acc \mathrel{+}= R_S^{\pm} \times R_T^{\pm}$
MADDU	R_S, R_T	$Acc \mathrel{+}= R_S^{\varnothing} \times R_T^{\varnothing}$
MSUB	R_S, R_T	$Acc \mathrel{-}= R_S^{\pm} \times R_T^{\pm}$
MSUBU	R_S, R_T	$Acc \mathrel{-}= R_S^{\varnothing} \times R_T^{\varnothing}$
MUL	R_D, R_S, R_T	$R_D = R_S^{\pm} \times R_T^{\pm}$
MULT	R_S, R_T	$Acc = R_S^{\pm} \times R_T^{\pm}$
MULTU	R_S, R_T	$Acc = R_S^{\varnothing} \times R_T^{\varnothing}$

Jumps And Branches (Note: One Delay Slot)		
B	off18	$PC \mathrel{+}= off18^{\pm}$
BAL	off18	$R_A = PC + 8$, $PC \mathrel{+}= off18^{\pm}$
BEQ	R_S, R_T, off18	if $R_S = R_T$, $PC \mathrel{+}= off18^{\pm}$
BEQZ	R_S, off18	if $R_S = 0$, $PC \mathrel{+}= off18^{\pm}$
BGEZ	R_S, off18	if $R_S \geq 0$, $PC \mathrel{+}= off18^{\pm}$
BGEZAL	R_S, off18	$R_A = PC + 8$; if $R_S \geq 0$, $PC \mathrel{+}= off18^{\pm}$
BGTZ	R_S, off18	if $R_S > 0$, $PC \mathrel{+}= off18^{\pm}$
BLEZ	R_S, off18	if $R_S \leq 0$, $PC \mathrel{+}= off18^{\pm}$
BLTZ	R_S, off18	if $R_S < 0$, $PC \mathrel{+}= off18^{\pm}$
BLTZAL	R_S, off18	$R_A = PC + 8$; if $R_S < 0$, $PC \mathrel{+}= off18^{\pm}$
BNE	R_S, R_T, off18	if $R_S \neq R_T$, $PC \mathrel{+}= off18^{\pm}$
BNEZ	R_S, off18	if $R_S \neq 0$, $PC \mathrel{+}= off18^{\pm}$
J	addr28	$PC = PC_{31:28} :: addr28^{\varnothing}$
JAL	addr28	$R_A = PC + 8$; $PC = PC_{31:28} :: addr28^{\varnothing}$
JALR	R_D, R_S	$R_D = PC + 8$; $PC = R_S$
JR	R_S	$PC = R_S$

Load and Store Operations		
LB	R_D, off16(R_S)	R_D = mem8(R_S + off16)$^{\pm}$
LBU	R_D, off16(R_S)	R_D = mem8(R_S + off16)$^{\varnothing}$
LH	R_D, off16(R_S)	R_D = mem16(R_S + off16)$^{\pm}$
LHU	R_D, off16(R_S)	R_D = mem16(R_S + off16)$^{\varnothing}$
LW	R_D, off16(R_S)	R_D = mem32(R_S + off16)$^{\pm}$
LWL	R_D, off16(R_S)	R_D = LoadWordLeft(R_S + off16)$^{\pm}$
LWR	R_D, off16(R_S)	R_D = LoadWordRight(R_S + off16)$^{\pm}$
SB	R_S, off16(R_T)	mem8(R_T + off16)$^{\pm}$ = $R_{S7:0}$
SH	R_S, off16(R_T)	mem16(R_T + off16)$^{\pm}$ = $R_{S15:0}$
SW	R_S, off16(R_T)	mem32(R_T + off16)$^{\pm}$ = R_S
SWL	R_S, off16(R_T)	StoreWordLeft(R_T + off16$^{\pm}$, R_S)
SWR	R_S, off16(R_T)	StoreWordRight(R_T + off16$^{\pm}$, R_S)

Referencias

J Hennesy and David Patterson (2012)

Computer Architecture: A Quantitative Approach. 5th Edition. Elsevier – Morgan Kaufmann. [Appendix A, Appendix K]

Jeferson González G. (2017)

Material de clase: Arquitectura de computadores I.

Sitios web recomendados

- Alineamiento de memoria:
  https://www.ibm.com/developerworks/library/pa-dalign/
- Apéndices Hennesey:
  https://booksite.elsevier.com/9780123838728/references.php
- Byte addressing:
  https://en.wikipedia.org/wiki/Byte_addressing