ECG 700 Advanced Computer System Architecture Fall 2012

Appendix A – Instruction Set Principles

Mei Yang

Adapted from David Patterson's slides on graduate computer architecture

Outline

- Classification of Instruction Set Architectures
- GPR Architectures
- Memory Addressing
- Type and Size of Operands
- Operations in the Instruction Set
- Instructions for Control Flow
- The Role of Compilers
- The MIPS Architecture

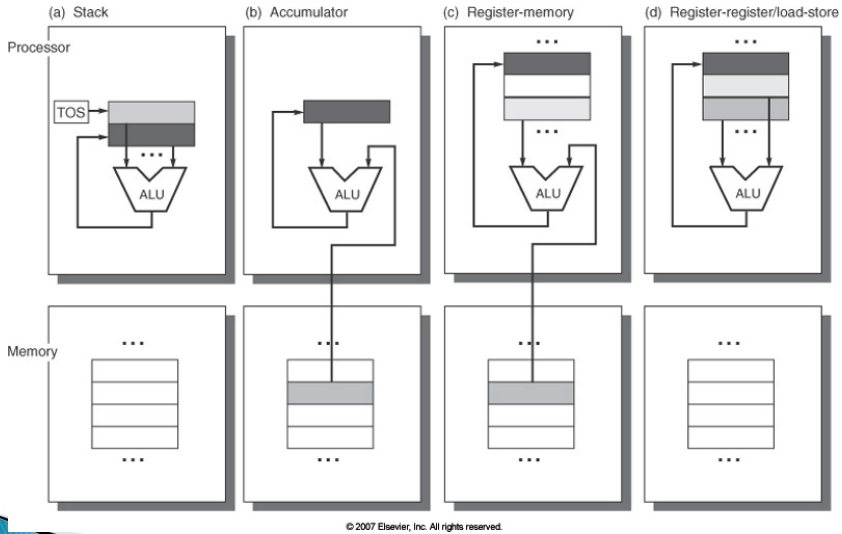
2

Classification of Instruction Set Architectures

- **Type of internal storage**
  - **Stack, accumulator, and a set of registers**
- **Stack architecture**
  - **Operands are implicitly on the top of the stack**
- **Accumulator architecture**
  - **One operand is implicitly the accumulator**
- **General-purpose register (GPR) architectures**
  - **Register-memory architecture**
    - » **One input operand is a register, one is in memory, and the result goes to register**
  - **Register-register/load-store architecture**
    - » **All operands are registers**
  - **Memory-memory architecture**

3

Classification of Instruction Set Architectures



4

GPR Architectures

- Two major instruction set characteristics divide GPR architectures
  - Whether an ALU instruction has two or three operands
    - Two operands: Sub R1, R2
    - Three operands: Add R3,R1,B
  - How many of the operands may be memory addresses in ALU instructions

5

GPR Architectures

Number of memory addresses	Maximum number of operands allowed	Type of architecture	Examples
0	3	Load-store	Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, TM32
1	2	Register-memory	IBM 360/370, Intel 80x86, Motorola 68000, TI TMS320C54x
2	2	memory-memory	VAX
3	3	memory-memory	VAX

6

Memory Addressing

- Interpreting memory addresses
  - All ISAs are byte addressed and provide access to bytes, half words, and words
  - Ordering of bytes
    - Little Endian: 7 6 5 4 3 2 1 0
    - Big Endian: 0 1 2 3 4 5 6 7
  - Accesses to objects larger than a byte must be aligned

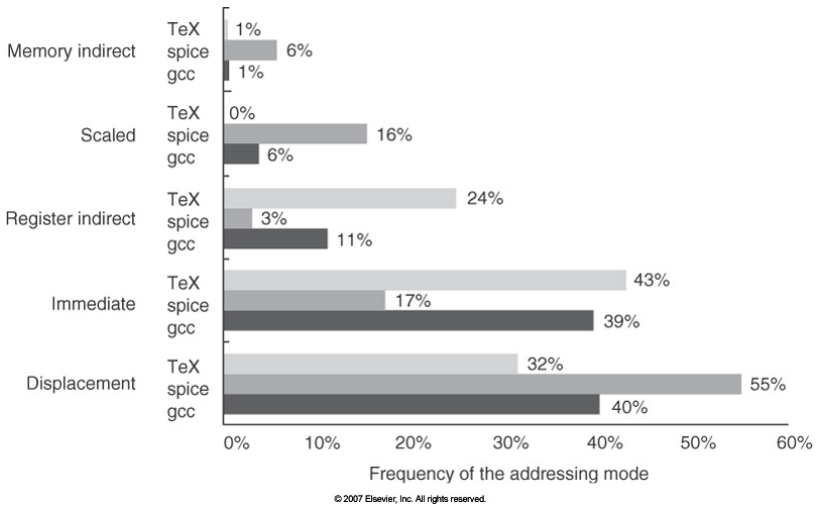
7

Addressing Modes

Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4] ← Regs[R4] + 3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100+Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes).
Register indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
Indexed	Add R3,(R1+R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1]+Regs[R2]]	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute	Add R1,(1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect	Add R1,@(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer *p*, then mode yields **p*.
Autoincrement	Add R1,(R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + *d*	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, *d*.
Autodecrement	Add R1,–(R2)	Regs[R2] ← Regs[R2] – *d* Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/-increment can also act as push/pop to implement a stack.
Scaled	Add R1,100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100+Regs[R2] + Regs[R3]*d]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

8

Addressing Modes



9

Type and Size of Operands

- Common operand types include
  - Character (8 bits)
  - Half word (16 bits)
  - Word (32 bits)
  - Single-precision floating point (1 word)
  - Double-precision floating point (2 words)

10

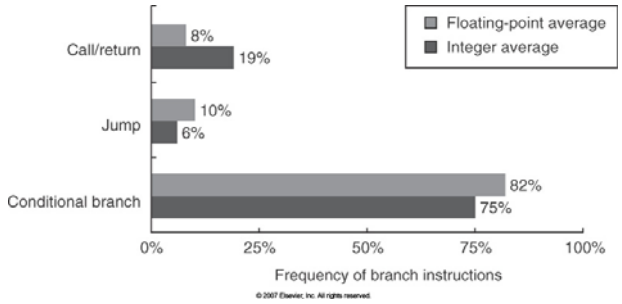
Operations in the Instruction Set

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%

11

Instructions for Control Flow

- Conditional branches
- Jumps
- Procedure calls
- Procedure returns



12

Instructions for Control Flow

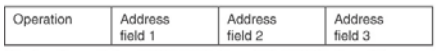
- Addressing modes
  - PC-relative
  - Register indirect for jump instructions to support returns
  - Branch displacement at least 8 bits
- Procedure invocation options
  - Caller saving
  - Callee saving

13

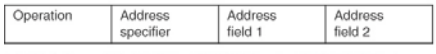
Encoding an Instruction Set

- Three types:
  - Variable suitable for code size preferred
  - Fixed suitable for performance preferred (RISC)
  - Hybrid

(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)

14

The Role of Compilers

- A new instruction set architecture to have at least 16 general-purpose registers—not counting separate registers for floating-point numbers.
- All supported addressing modes apply to all instructions that transfer data.
- Provide primitives instead of solutions, simplify trade-offs between alternatives, don't bind constants at run time.

15

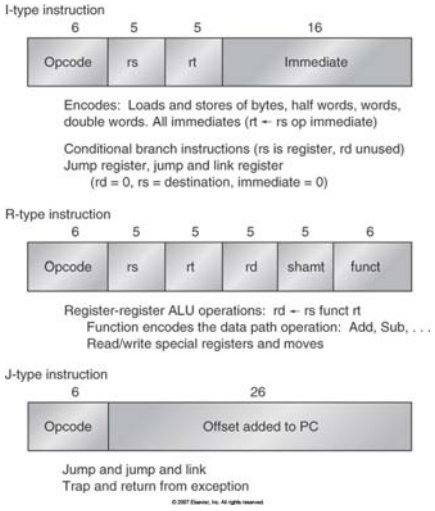
The MIPS Architecture

- MIPS64
  - Registers for MIPS
    - 32 64-bit general-purpose registers (GPRs), named R0(0), R1, ... , R31
    - 32 64-bit floating-point registers (FPRs), named F0, F1, . . . , F31
  - Data Types for MIPS
    - 8-bit bytes, 16-bit half words, 32-bit words, and 64-bit double words for integer data
    - 32-bit single precision and 64-bit double precision for floating point
  - Addressing Modes for MIPS Data Transfers
    - Immediate and displacement, both with 16-bit fields
    - Register indirect and absolute addressing are supported implicitly
    - Memory is byte addressable with a 64-bit address

16

The MIPS Architecture

- MIPS instruction format (32-bit)



Example

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]$_{56..63}$

Example

Example instruction	Instruction name	Meaning
DADDU R1,R2,R3	Add unsigned	Regs[R1]←Regs[R2]+Regs[R3]
DADDIU R1,R2,#3	Add immediate unsigned	Regs[R1]←Regs[R2]+3
LUI R1,#42	Load upper immediate	Regs[R1]←0^{32}##42##0^{16}
DSLL R1,R2,#5	Shift left logical	Regs[R1]←Regs[R2]<<5
SLT R1,R2,R3	Set less than	if (Regs[R2]<Regs[R3]) Regs[R1]←1 else Regs[R1]←0

19

MIPS Control Flow Instructions

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

20