


BROWN

Computer System Design Lecture 5: Instruction Sets

Prof. R. Iris Bahar
EN164
February 5, 2007

Reading: Appendix B, section B.1-B.6




BROWN

Lab 1 Information

- Lab 1 handouts are now available
- Labs are due (i.e., need to be checked off by a TA by Thursday, February 15, by 10pm
- Lab write ups are due Friday, February 16, by 5pm
- TAs will have their lab hours posted on the web by tomorrow
- Lab is in room 196
 - The lab is normally locked; coordinate with TA hours to access the lab

EN164
Lecture 4-2



BROWN


Accessing Internal Storage

- How should memory be accessed?
 - Directly or through registers?
- How should ALU operations be expressed?
 - Implicit or explicit operands?
 - compact or flexible?
- Example: How should we represent $C = A + B$?

| Stack | Accumulator | Reg (reg-mem) | Reg (load-store) |
|--------|-------------|---------------|------------------|
| Push A | Load A | Load R1, A | Load R1, A |
| Push B | Add B | Add R3, R1, B | Load R2, B |
| Add | Store C | Store R3, C | Add R3, R1, R2 |
| Pop C | | | Store R3, C |

- Registers: fast, exploit locality, reduced memory traffic, easier to re-order

EN164
Lecture 4-4




BROWN

Register Architectures

| Type | Advantages | Disadvantages | Examples |
|--|--|--|----------------------------------|
| Register-Register (0 mem, 3 ops) RISC | Simple, fixed-length, simple code-generation, easy pipelining and parallelism extraction | High instr count and code size | Alpha, MIPS, ARM, PowerPC, SPARC |
| Register-Memory (1 mem, 2 ops) CISC | Can access data without doing a load, small code size | One of the operands is destroyed, instr latency is variable | Intel 80x86, Motorola 68000 |
| Memory-Memory (2 mem, 2 ops) or (3, 3) CISC | Most compact code size, doesn't waste registers | Variation in instr size (hard to decode), frequent memory accesses, variable instr latency | VAX |

EN164
Lecture 4-5




BROWN

RISC vs. CISC ISAs

- Reduced Instruction Set Computer (RISC): by using a few simple instruction primitives, the hardware is simpler
 - easy to extract parallelism
 - easy to effect high clock speeds
- Complex Instruction Set Computer (CISC): if you do it in hardware, it's fast → therefore, implement every functionality in hardware
 - Rich instruction set
 - Complex decoding
 - Complex analysis to identify dependences
 - Danger is a slower cycle time and/or a higher CPI
 - Goal is to reduce number of instruction executed
 - Why is this important??

EN164
Lecture 4-6




BROWN

Characteristics of RISC ISAs

- Common characteristics of all RISCs
 - Single cycle issue
 - Small number of fixed length instruction formats
 - Load/store architecture
 - Large number of registers
- Additional characteristics of most RISCs
 - Small number of instructions in the instruction set
 - Small number of addressing modes
 - Fast control unit
- Virtually all new instruction sets since 1982 have been RISC based

EN164
Lecture 4-7


 BROWN

Example Instruction Sets

- MIPS
 - Popular example of a RISC instruction set
 - Relatively easy to learn and use
- X86
 - CISC instruction set developed at Intel (1st version in 1978)
 - Why is it still popular today, despite the trend to RISC?

 - How is Intel able to incorporate RISC characteristics into its processor design while still implementing the X86 ISA?

EN164
Lecture 4-8

 BROWN

Addressing Modes for Memory

| Addressing mode | Example | Meaning |
|-------------------|-----------------|--|
| Register | Add R4, R3 | $\text{Regs}[R4] \leftarrow \text{Regs}[R4] + \text{Regs}[R3]$ |
| Immediate | Add R4, #3 | $\text{Regs}[R4] \leftarrow \text{Regs}[R4] + 3$ |
| Displacement | Add R4, 100(R1) | $\text{Regs}[R4] \leftarrow \text{Regs}[R4] + \text{Mem}[100 + \text{Regs}[R1]]$ |
| Register indirect | Add R4, (R1) | $\text{Regs}[R4] \leftarrow \text{Regs}[R4] + \text{Mem}[\text{Regs}[R1]]$ |
| Direct/absolute | Add R1, (1001) | $\text{Regs}[R1] \leftarrow \text{Regs}[R1] + \text{Mem}[1001]$ |
| Memory indirect | Add R1, @(R3) | $\text{Regs}[R1] \leftarrow \text{Regs}[R1] + \text{Mem}[\text{Mem}[\text{Regs}[R3]]]$ |
| Indexed | Add R3, (R1+R2) | $\text{Regs}[R3] \leftarrow \text{Regs}[R3] + \text{Mem}[\text{Regs}[R1] + \text{Regs}[R2]]$ |
| Autoincrement | Add R1, (R2)+ | $\text{Regs}[R1] \leftarrow \text{Regs}[R1] + \text{Mem}[\text{Regs}[R2]]$ $\text{Regs}[R2] \leftarrow \text{Regs}[R2] + d$ |

- More addressing modes → lower instruction counts, more complexity
- Most common modes: immediate and displacement
 - How do you simulate register indirect and direct addressing with displacement addressing mode?
 - Which would you more likely find in a CISC-like ISA?

EN164
Lecture 4-9