

Lecture 3

Instruction Set Principles (Continued)

Instructor: Jun Yang

Review

- ISA classification
- Address Modes
- Operand type and size
- Types of operations
- Non-data related program control instructions (Branches)

Outline

- Non-data related program control instructions (Branches)
- Instruction Format
- Role of Compiler
- RISC vs. CISC
- MIPS

Control Operations

- Types
 - Conditional branches
 - Jumps
 - Procedure calls and returns
- Issues
 - Taken or not?
 - Where is the target?
 - How is the target specified?
 - Link return address?
 - Save registers?

Branch Condition

• Branch Condition Test (fig 2.21)

1. Implicitly set condition code bits:
 - o Restrict code scheduling since information is passed from one instruction to the branch.
 - o Extra state that is not user visible.
2. Explicitly set condition codes
 - o Extra state (condition registers).
 - o Decouples condition setting from branch. [Condition Code](#)
3. Compare and branch instruction
 - o One instruction instead of two.
 - o No direct dependencies.
 - o Requires an ALU operation for comparison.
4. Condition in GPR
 - o No special "hidden state."
 - o Uses up a register.
 - o Separates condition setting and branch.
- Mixed Instructions
 - Experimental data:
 - o in > 50% of conditional branches, the conditions is tested against 0.
 - MIPS ISA:
 - o Conditional branch for == and <> 0 only (1,2,4).
 - o Compare and branch otherwise (3).

Addressing Modes for Control Flow Instructions

• Target Address Options

- Arbitrary target address
 - (+) Most general case, orthogonal.
 - (-) Uses more bits in instruction.
- PC relative with immediate
 - (+) Position independence.
 - (+) Short immediate: 47% use < 4 bits, 94% < 8 bits.
 - (-) Statically known target address, no long jumps.
- Register
 - (+) Uses few bits.
 - (+) Supports dynamically specified target addresses, used in:
 - case or switch statements
 - dynamically shared libraries: loaded only when invoked by program
- virtual functions: different routines execute based on type of data.
- Function pointers: allow function to be passed as a parameter.
 - (-) Requires extra instruction to load register.
- In Practice
 - Conditional branches: PC relative
 - Jumps: PC relative + Register
 - Procedure calls: PC relative, register
 - Returns: register
 - Calling conventions---read the book on page 116.

Outline

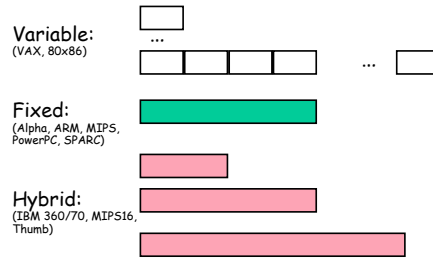
- Non-data related program control instructions (Branches)
- **Instruction Format**
- Role of Compiler
- RISC vs. CISC
- MIPS

Oct. 3, 2002

Lec. 3

7

Generic Examples of Instruction Format Widths



Oct. 3, 2002

Lec. 3

8

Effects of Instruction Formats on Code Size

- If code size is most important, use variable length instructions
- If performance is most important, use fixed length instructions
- Recent embedded machines (ARM, MIPS) added optional mode to execute subset of 16-bit wide instructions (Thumb, MIPS16); per procedure decide performance or density
- Some architectures actually explore on-the-fly decompression for more density.
 - PowerPC (IBM) compresses code in memory, decompresses it in instruction cache using special hardware.
 - Branch targets are handled through an address mapping table for uncompressed and compressed code.

Oct. 3, 2002

Lec. 3

9

Outline

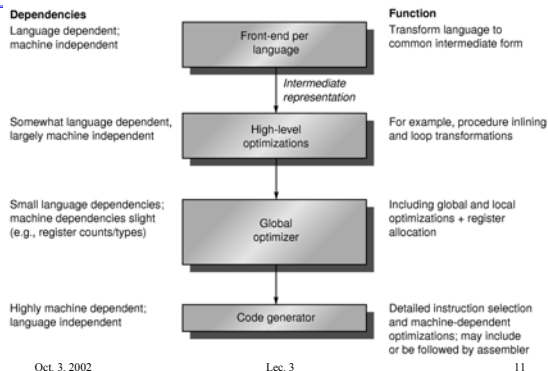
- Non-data related program control instructions (Branches)
- **Instruction Format**
- **Role of Compiler**
- RISC vs. CISC
- MIPS

Oct. 3, 2002

Lec. 3

10

Structure of Compilers



Oct. 3, 2002

Lec. 3

11

Compiler Optimizations

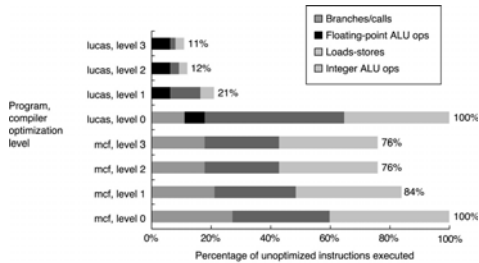
- **Goals:**
 - (1) correctness, (2) speed.
- **Optimizations**
 - High level: at source code level
 - o procedure integration
 - Local: within a basic block
 - o common sub-expression elimination (cse)
 - o constant propagation
 - o stack height reduction
 - Global: across basic blocks
 - o global cse
 - o copy propagation
 - o invariant code motion
 - o induction variable elimination
- **Impact of optimizations**
 - Machine dependent
 - o strength reduction
 - o pipeline scheduling
 - o branch delay slot filling
 - Dramatic impact on fp code, less so on integer code.
 - Most reduction in integer and load/store operations.
 - Some reduction in branches in lucas.

Oct. 3, 2002

Lec. 3

12

Effects of Compiler Optimizations



Level 1: local optimizations, code scheduling, and local register allocation
 Level 2: global optimizations, loop transformations (software pipelining), global register allocation
 Level 3: procedure integration

Oct. 3, 2002

Lec. 3

13

Compiler Issues

- Data allocation areas:
 - Stack: used for local variables, mostly scalars, SP-relative addressing.
 - Global data area: statically declared data structures.
 - Heap: non-scalar dynamic objects, pointer addressing.
- Register allocation:
 - Objective: which variable should be in a register?
 - In general, the most important optimization.
 - Uses graph coloring algorithm heuristics, works best with more than 16 registers.
 - Register allocation is:
 - o easy for stack variables
 - o difficult for global area variables
 - o impossible for heap based variables

Oct. 3, 2002

Lec. 3

14

Compiler Issues

- Phase ordering problem:
 - in what order should the optimizations be applied?
 - what can the compiler assume about capability of later phases?
 - examples of phase ordering:
 - o Global cse replaces an expression with a temporary variable.
 - If variable must be stored and re-loaded from memory it might be faster to evaluate the expression!
 - But register allocation is not done till the very end!
- The aliasing problem

<code>p = &a</code>	<code>p</code> holds the address of <code>a</code>
<code>a = ...</code>	<code>a</code> is assigned to directly
<code>*p = ...</code>	uses <code>p</code> to assign to <code>a</code>
<code>... = ... a</code>	uses <code>a</code>

variable `a` cannot be register allocated past the third statement.

Oct. 3, 2002

Lec. 3

15

Compilers and ISA

- How ISA design can help compilers
 - Regularity:
 - o Keep operations, data types and addressing modes orthogonal whenever reasonable (simplifies code generation).
 - Provide primitives not solutions:
 - o do not attempt to match high-level language constructs in the ISA.
 - Simplify trade-off among alternatives:
 - o make it easier to select best code sequence.
 - Allow the binding of compile time known values.

Oct. 3, 2002

Lec. 3

16

Outline

- Non-data related program control instructions (Branches)
- Instruction Format
- Role of Compiler
- RISC vs. CISC
- MIPS

Oct. 3, 2002

Lec. 3

17

RISC vs. CISC Instruction Set Design

- The historical background:
 - In first 25 years (1945-70) performance came from both technology and design.
 - Design constraints:
 - o small and slow memories: compact programs are fast.
 - o small no. of registers: memory operands.
 - o attempts to bridge the semantic gap: model high level language features in instructions.
 - o no need for portability: same vendor application, OS and hardware.
 - o backward compatibility: every new ISA must carry the good and bad of all past ones.
 - Result: powerful and complex instructions that are rarely used.
 - IC technology and microprocessors in 1970s: lower costs, low power consumption, higher clock rates, cheaper and larger memories.

Oct. 3, 2002

Lec. 3

18

RISC vs. CISC (2)

- Emergence of RISC
 - Very large scale integration (processor on a chip): silicon real-estate at a premium. Micro-store occupies about 70% of chip area: replace micro-store with registers ==> load/store ISA.
 - Increased difference between CPU and memory speeds.
 - Complex instructions where not used by new compilers.
 - Software changes:
 - standardized vendor independent OS (Unix) became very popular in some market segments (academia and research).
 - Early RISC projects: IBM 801 (America), Berkeley SPUR, RISC I and RISC II and Stanford MIPS.

RISC vs. CISC (3)

- Characteristics of ISAs

CISC	RISC
Variable length instruction	Single word instruction
Variable format	Fixed-field decoding
Memory operands	Load/store architecture
Complex operations	Simple operations

Outline

- Non-data related program control instructions (Branches)
- Instruction Format
- Role of Compiler
- RISC vs. CISC
- MIPS - next time!