18-447 Computer Architecture Lecture 4: More ISA Tradeoffs

Prof. Onur Mutlu
Carnegie Mellon University
Spring 2012, 1/23/2012

Homework 0

Due now

Reminder: Homeworks for Next Two Weeks

Homework 1

- Due Monday Jan 28, midnight
- Turn in via AFS (hand-in directories) or box outside CIC 4th floor
- MIPS warmup, ISA concepts, basic performance evaluation

Homework 2

Will be assigned next week. Stay tuned...

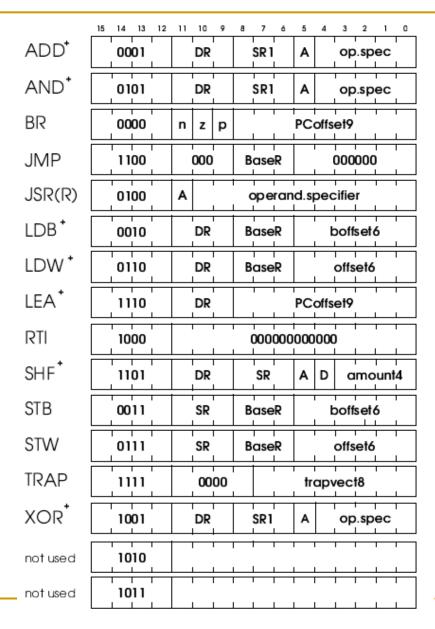
Reminder: Lab Assignment 1

- Due next Friday (Feb 1), at the end of Friday lab
- A functional C-level simulator for a subset of the MIPS ISA
- Study the MIPS ISA Tutorial
 - □TAs will cover this in Lab Sessions this week

Review of Last Lecture

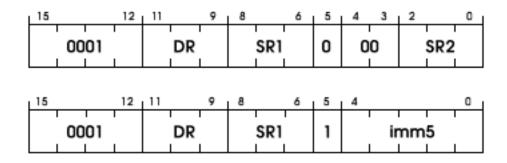
- ISA Principles and Tradeoffs
- Elements of the ISA
 - Sequencing model, instruction processing style
 - Instructions, data types, memory organization, registers, addressing modes, orthogonality, I/O device interfacing ...
- What is the benefit of autoincrement addressing mode?
- What is the downside of having an autoincrement addressing mode?
- Is the LC-3b ISA orthogonal?
 - Can all addressing modes be used with all instructions?

Is the LC-3b ISA Orthogonal?



LC-3b: Addressing Modes of ADD

Encodings

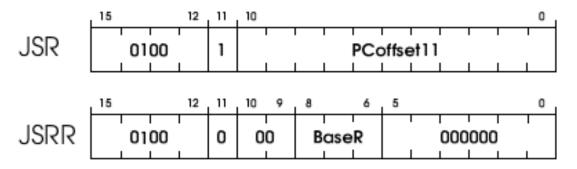


Operation

```
if (bit[5] == 0)
   DR = SR1 + SR2;
else
   DR = SR1 + SEXT(imm5);
setcc();
```

LC-3b: Addressing Modes of of JSR(R)

Encodings



Operation

```
R7 = PC<sup>†</sup>;
if (bit[11] == 0)
PC = BaseR;
else
PC = PC<sup>†</sup> + LSHF(SEXT(PCoffset11), 1);
```

Description

First, the incremented PC is saved in R7. This is the linkage back to the calling routine. Then, the PC is loaded with the address of the first instruction of the subroutine, causing an unconditional jump to that address. The address of the subroutine is obtained from the base register (if bit[11] is 0), or the address is computed by sign-extending bits [10:0] to 16 bits, left-shifting the result one bit, and then adding this value to the incremented PC (if bit[11] is 1).

Another Question

Does the LC-3b ISA contain complex instructions?

Complex vs. Simple Instructions

- Complex instruction: An instruction does a lot of work, e.g. many operations
 - Insert in a doubly linked list
 - Compute FFT
 - String copy
- Simple instruction: An instruction does small amount of work, it is a primitive using which complex operations can be built
 - Add
 - XOR
 - Multiply

Complex vs. Simple Instructions

- Advantages of Complex instructions
 - + Denser encoding → smaller code size → better memory utilization, saves off-chip bandwidth, better cache hit rate (better packing of instructions)
 - + Simpler compiler: no need to optimize small instructions as much
- Disadvantages of Complex Instructions
 - Larger chunks of work → compiler has less opportunity to optimize (limited in fine-grained optimizations it can do)
 - More complex hardware → translation from a high level to control signals and optimization needs to be done by hardware

ISA-level Tradeoffs: Semantic Gap

- Where to place the ISA? Semantic gap
 - □ Closer to high-level language (HLL) → Small semantic gap, complex instructions
 - □ Closer to hardware control signals? → Large semantic gap, simple instructions
- RISC vs. CISC machines
 - RISC: Reduced instruction set computer
 - CISC: Complex instruction set computer
 - FFT, QUICKSORT, POLY, FP instructions?
 - VAX INDEX instruction (array access with bounds checking)

ISA-level Tradeoffs: Semantic Gap

- Some tradeoffs (for you to think about)
- Simple compiler, complex hardware vs. complex compiler, simple hardware
 - Caveat: Translation (indirection) can change the tradeoff!
- Burden of backward compatibility
- Performance?
 - Optimization opportunity: Example of VAX INDEX instruction: who (compiler vs. hardware) puts more effort into optimization?
 - Instruction size, code size

X86: Small Semantic Gap: String Operations

- An instruction operates on a string
 - Move one string of arbitrary length to another location
 - Compare two strings
- Enabled by the ability to specify repeated execution of an instruction (in the ISA)
 - Using a "prefix" called REP prefix
- Example: REP MOVS instruction
 - Only two bytes: REP prefix byte and MOVS opcode byte (F2 A4)
 - Implicit source and destination registers pointing to the two strings (ESI, EDI)
 - Implicit count register (ECX) specifies how long the string is

X86: Small Semantic Gap: String Operations

```
REP MOVS (DEST SRC)
                                                                                                         DEST \leftarrow SRC:
                                                                                                         IF (Byte move)
                                                                                                            THEN IF DF = 0
                                                                                                                THEN
                                                                                                                    (R|E)SI \leftarrow (R|E)SI + 1;
IF AddressSize = 16
                                                                                                                    (R|E)DI \leftarrow (R|E)DI + 1;
     THEN
                                                                                                                ELSE
                                                                                                                     (R|E)SI \leftarrow (R|E)SI - 1;
           Use CX for CountReg;
                                                                                                                    (R|E)DI \leftarrow (R|E)DI - 1;
                                                                                                                FI:
     ELSE IF AddressSize = 64 and REX.W used
                                                                                                            ELSE IF (Word move)
           THEN Use RCX for CountReg; FI;
                                                                                                                THEN IF DF = 0
                                                                                                                    (R|E)SI \leftarrow (R|E)SI + 2;
     ELSE
                                                                                                                    (R|E)DI \leftarrow (R|E)DI + 2;
           Use ECX for CountReg;
                                                                                                                ELSE
FI:
                                                                                                                     (R|E)SI \leftarrow (R|E)SI - 2;
                                                                                                                     (R|E)DI \leftarrow (R|E)DI - 2;
WHILE CountReg \neq 0
     D0
                                                                                                            ELSE IF (Doubleword move)
                                                                                                                THEN IF DF = 0
           Service pending interrupts (if any);
                                                                                                                    (R|E)SI \leftarrow (R|E)SI + 4;
                                                                                                                    (R|E)DI \leftarrow (R|E)DI + 4;
           Execute associated string instruction;
                                                                                                                    FI:
                                                                                                                ELSE
           CountReg \leftarrow (CountReg - 1);
                                                                                                                    (R|E)SI \leftarrow (R|E)SI - 4;
           IF CountReq = 0
                                                                                                                    (R|E)DI \leftarrow (R|E)DI - 4;
                  THEN exit WHILE loop; FI;
                                                                                                            ELSE IF (Quadword move)
           IF (Repeat prefix is REPZ or REPE) and (ZF = 0)
                                                                                                                THEN IF DF = 0
                                                                                                                    (R|E)SI \leftarrow (R|E)SI + 8;
           or (Repeat prefix is REPNZ or REPNE) and (ZF = 1)
                                                                                                                    (R|E)DI \leftarrow (R|E)DI + 8;
                                                                                                                    FI:
                  THEN exit WHILE loop; FI;
                                                                                                                ELSE
     OD;
                                                                                                                    (R|E)SI \leftarrow (R|E)SI - 8;
                                                                                                                    (R|E)DI \leftarrow (R|E)DI - 8;
                                                                                                                FI;
                                                                                                        FI;
   How many instructions does this take in MIPS?
```

Small Semantic Gap Examples in VAX

- FIND FIRST
 - Find the first set bit in a bit field
 - Helps OS resource allocation operations
- SAVE CONTEXT, LOAD CONTEXT
 - Special context switching instructions
- INSQUEUE, REMQUEUE
 - Operations on doubly linked list
- INDEX
 - Array access with bounds checking
- STRING Operations
 - Compare strings, find substrings, ...
- Cyclic Redundancy Check Instruction
- EDITPC
 - Implements editing functions to display fixed format output
- Digital Equipment Corp., "VAX11 780 Architecture Handbook," 1977-78.

Small versus Large Semantic Gap

CISC vs. RISC

- □ Complex instruction set computer → complex instructions
 - Initially motivated by "not good enough" code generation
- □ Reduced instruction set computer → simple instructions
 - John Cocke, mid 1970s, IBM 801
 - Goal: enable better compiler control and optimization

RISC motivated by

- Memory stalls (no work done in a complex instruction when there is a memory stall?)
 - When is this correct?
- □ Simplifying the hardware → lower cost, higher frequency
- Enabling the compiler to optimize the code better
 - Find fine-grained parallelism to reduce stalls

How High or Low Can You Go?

Very large semantic gap

- Each instruction specifies the complete set of control signals in the machine
- Compiler generates control signals
- Open microcode (John Cocke, circa 1970s)
 - Gave way to optimizing compilers

Very small semantic gap

- ISA is (almost) the same as high-level language
- Java machines, LISP machines, object-oriented machines, capability-based machines

A Note on ISA Evolution

ISAs have evolved to reflect/satisfy the concerns of the day

Examples:

- Limited on-chip and off-chip memory size
- Limited compiler optimization technology
- Limited memory bandwidth
- Need for specialization in important applications (e.g., MMX)
- Use of translation (in HW and SW) enabled underlying implementations to be similar, regardless of the ISA
 - Concept of dynamic/static interface
 - Contrast it with hardware/software interface

Effect of Translation

 One can translate from one ISA to another ISA to change the semantic gap tradeoffs

Examples

- Intel's and AMD's x86 implementations translate x86 instructions into programmer-invisible microoperations (simple instructions) in hardware
- Transmeta's x86 implementations translated x86 instructions into "secret" VLIW instructions in software (code morphing software)
- Think about the tradeoffs

ISA-level Tradeoffs: Instruction Length

- Fixed length: Length of all instructions the same
 - + Easier to decode single instruction in hardware
 - + Easier to decode multiple instructions concurrently
 - -- Wasted bits in instructions (Why is this bad?)
 - -- Harder-to-extend ISA (how to add new instructions?)
- Variable length: Length of instructions different (determined by opcode and sub-opcode)
 - + Compact encoding (Why is this good?)
 Intel 432: Huffman encoding (sort of). 6 to 321 bit instructions. How?
 - -- More logic to decode a single instruction
 - -- Harder to decode multiple instructions concurrently

Tradeoffs

- Code size (memory space, bandwidth, latency) vs. hardware complexity
- ISA extensibility and expressiveness
- Performance? Smaller code vs. imperfect decode

ISA-level Tradeoffs: Uniform Decode

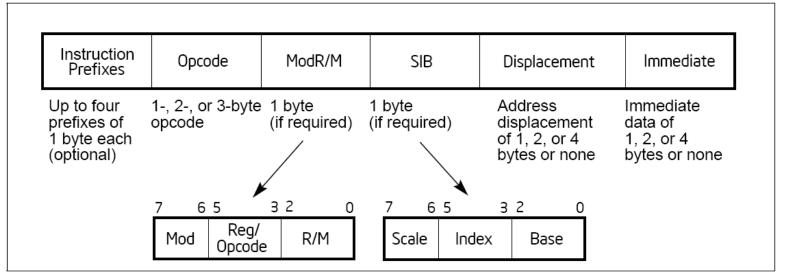
- Uniform decode: Same bits in each instruction correspond to the same meaning
 - Opcode is always in the same location
 - Ditto operand specifiers, immediate values, ...
 - Many "RISC" ISAs: Alpha, MIPS, SPARC
 - + Easier decode, simpler hardware
 - + Enables parallelism: generate target address before knowing the instruction is a branch
 - -- Restricts instruction format (fewer instructions?) or wastes space

Non-uniform decode

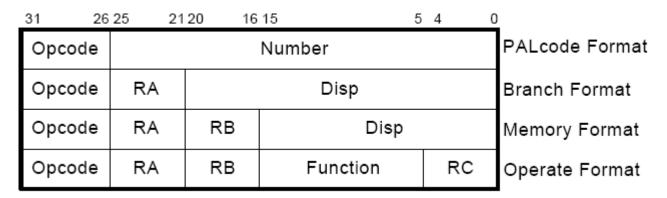
- E.g., opcode can be the 1st-7th byte in x86
- + More compact and powerful instruction format
- -- More complex decode logic

x86 vs. Alpha Instruction Formats

x86:

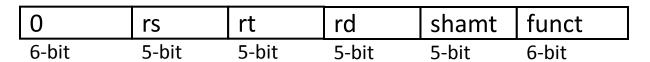


Alpha:



MIPS Instruction Format

R-type, 3 register operands



I-type, 2 register operands and 16-bit immediate operand

opcode	rs	rt	immediate	I-type
6-hit	5-hit	5-hit	16-hit	

J-type, 26-bit immediate operand

opcode	immediate	J-type
6-hit	26-bit	

- Simple Decoding
 - 4 bytes per instruction, regardless of format
 - must be 4-byte aligned (2 lsb of PC must be 2b'00)
 - format and fields easy to extract in hardware

R-type

A Note on Length and Uniformity

- Uniform decode usually goes with fixed length
- In a variable length ISA, uniform decode can be a property of instructions of the same length
 - It is hard to think of it as a property of instructions of different lengths

A Note on RISC vs. CISC

Usually, ...

RISC

- Simple instructions
- Fixed length
- Uniform decode
- Few addressing modes

CISC

- Complex instructions
- Variable length
- Non-uniform decode
- Many addressing modes

ISA-level Tradeoffs: Number of Registers

Affects:

- Number of bits used for encoding register address
- Number of values kept in fast storage (register file)
- (uarch) Size, access time, power consumption of register file

Large number of registers:

- + Enables better register allocation (and optimizations) by compiler → fewer saves/restores
- -- Larger instruction size
- -- Larger register file size