Lecture 4: ISA Examples

Kunle Olukotun
Gates 302
kunle@ogun.stanford.edu

http://www-leland.stanford.edu/class/ee282h/

1

ISA case studies

- We'll look at examples of two instruction set architecture styles :
  - » DEC VAX
    - – conceptually elegant
    - – end-of-an-era CISC design popular from the late1970's
    - – much studied: many architectural measurements exist
  - » DLX
    - – based on the last decade of instruction-set measurement
    - – 3-operand load/store many-register architecture
  - » Intel 80x86
    - – Will not cover in class but read appendix D
    - – design influenced by generational history
    - – halfway between accumulator and general-register machine
    - – sells 100x any other ISA

2

Encoding an ISA

- Balance competing forces:
  - » Decode complexity
  - » cycle time
  - » maximizing what can be done early in execution
  - » compiler/programmer convenience: many registers and addressing modes
  - » minimizing instruction count or instruction size
- addressing modes are the hardest part
  - » For small number of modes, encode implicitly in the opcode
  - » For a large number of modes, use separate address specifier

3

The DEC VAX architecture

- Superset of many other ISAs, including IBM 360/370 and DEC PDP-11.
- VAX = “Virtual Address Extension: solve the problem of 16-bit addressing on the PDP-11, but be compatible:
  - » Same data types
  - » Same I/O bus (UNIBUS) plus others (MASSBUS, SBI)
  - » PDP-11 direct emulation mode
  - » Similar assembler syntax
- Large orthogonal instruction set, with support for:
  - » High-level language contructs (procedure call w/ arguments)
  - » Virtual memory management
  - » Rapid context switching (Process Control Block: Regs+state)
- First implementation: the VAX-11/780
  - » Internally microprogrammed, 96-bit word + writeable section

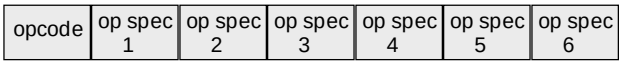
4

The VAX architecture

- General register machine: 16 32-bit registers
- Special register assignments:
  - » R15: program counter
  - » R14: current stack pointer
  - » R13: stack frame pointer
  - » R12: argument list pointer
  - » R0-R5: state information for long interruptible instructions
- 32-bit addressing

5

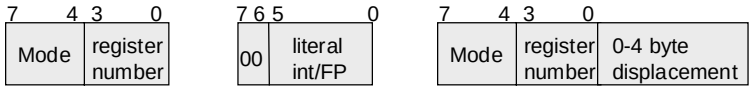
VAX Instruction Encoding



- 1-2 byte opcodes followed by 0-6 operand specifiers, each of which may be up to 5 bytes.
- About 300 opcodes, most 8-bit (integer, floating point, character string, bitfields, decimal, CRC, POLY)
- Opcode implies datatype and size, and number of operands (A=B+C vs A=A+B)
- Operand specifiers indicate addressing modes
- Orthogonality: any opcode with any operands of any modes
- How do you encode?

6

VAX operand addressing



- 6-bit Literal: 0-63 positive integers
  3-bit exp+3-bit frac floating point (integers 1-16, 16ths, more)
- register, register deferred, optional autoincrement/decrement
- 1, 2, 4-byte displacement from register, optionally deferred
- Indexed (scaled): R*operandsize + next operand specifier (for array indexing)
- Clever encoding:
  - » immediate = PC autoincrement
  - » absolute = PC autoincrement deferred

7

Characteristics of RISC architectures?

- RISC (“Reduced Instruction Set Computer”) architectures generally:
  - » have fixed-size 32-bit instructions
  - » are register-based with 32 or more registers
  - » have only load/store memory operand access instructions (no ALU memory ops)...
  - » No support complex data types
  - » try to avoid special-case instructions
    - – “orthogonality”: any operation with any register, etc.
  - » are willing to make architectural compromises to permit efficient pipelining

8

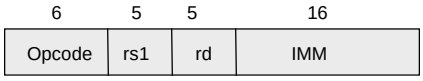
DLX

- DLX: a fictitious prototypical RISC machine
  - » Simple load/store instruction set
  - » Easily decoded and pipelined
  - » Good compiler target (few opcodes, many registers, one (1) addressing mode
- Overview
  - » 32 32-bit general-purpose registers; R0 is always 0
  - » 32 32-bit (single-precision) floating point registers, or 16 64-bit (double-precision) registers
  - » Memory byte addressable, 32-bit addresses
  - » Aligned data accesses
  - » All instructions are 32 bits long, in 4 classes:
    - – load and stores
    - – ALU ops
    - – branches and jumps
    - – floating point

9

DLX Instructions 1 of 3

I-type instructions

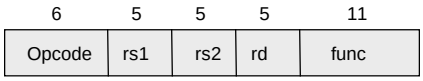


- Load and stores (byte/halfword/word, signed/unsigned)
  - » rd <=> Memory [rs1 + IMM]
- ALU operations with immediate operands
  - » rd <= rs1 op IMM
- Conditional branches: no condition code!
  - » if (rs1 == 0) PC += IMM
  - » Longer distances require an additional jump
- Jump [and link] register
  - » [r31 = PC+4] PC <= rs1
- Load immediate, load upper immediate
  - » Used together to load a 32-bit value

10

DLX Instructions
2 of 3

R-type instructions

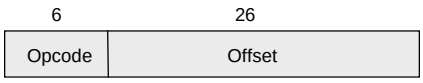


- Register-to-register instructions: 3-operand ALU ops
  - » rd <= rs1 func rs2
    - – add/sub/mul/divide, signed and unsigned
    - – and/or/xor
    - – shifts
    - – compares: if true destination is 1, else 0
    - – floating point add/sub/mul/divide, single/double precision
    - – floating/integer conversion instructions

11

DLX Instructions
3 of 3

J-type instructions



- Jump
  - » PC += Offset
- Jump and link
  - » r31 <= PC+4; PC += Offset

12

What DLX needs to do to emulate VAX addressing modes

- Assume:
  - » VAX displacements fit in the DLX displacement field
  - » The scale size is a power of 2

	VAX	DLX
register-deferred	add r1, (r2)	lw r3, 0(r2) add r1, r1, r3
displacement	add r1, 200(r2)	lw r3, 200(r2) add r1, r1, r3

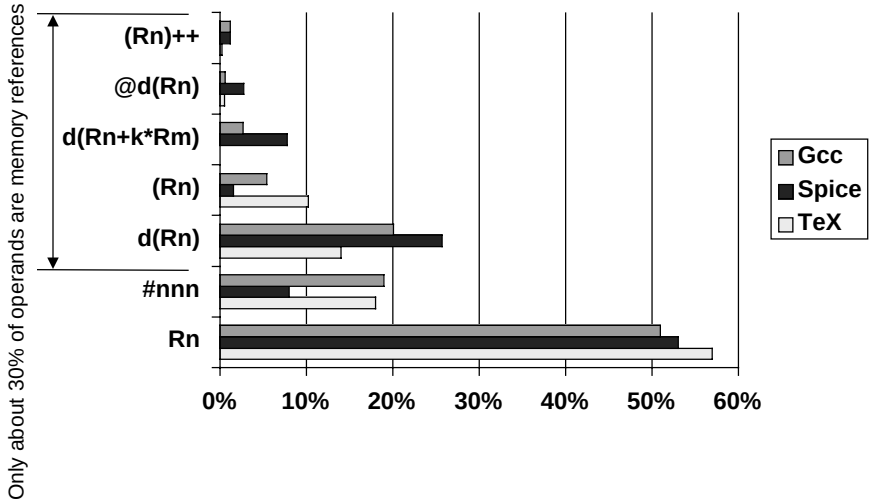
autoincrement	add r1, (r2)+	lw r3, 0(r2)
		addui r2, r2, opsize
		add r1, r1, r3

displacement deferred	add r1, @300(r2)	lw r3, 300(r2)
		lw r3, 0(r1)
		add r1, r1, r3

scaled	add r1, 100(r2)[r3]	sll r4, r3, $\log_2$opsize
		addu r4, r4, r2
		lw r5, 100(r4)
		add r1, r1, r5

13

What addressing modes are most common?



14

Computing the DLX penalty for not having VAX addressing modes

- For VAX there are about 1.8 operands per instruction
- For GCC, memory operand modes are used as follows:
  - » reg indirect 5.4% +1 instruction for DLX
  - » displacement 20% +1 instruction for DLX
  - » scaled 2.7% +3 instructions for DLX
  - » displ. indirect 0.6% +2 instructions for DLX
  - » autoincrement 1.2% +2 instructions for DLX
  - » register 51% +0 instructions for DLX
  - » immediate 19% +0 instructions for DLX
- Additional DLX instructions = 1.8 x
  (.054 x 1 + .20 x 1 + .027 x 3
  + .006 x 2 + .012 x 2)
  = 1.8 x .371 = 0.66

  Or, DLX/VAX of 1.66

15

DLX SPECint92 Instruction Mix

Instruction	compress	eqntott	espresso	gcc (cc1)	li	Integer average
load	19.8%	30.6%	20.9%	22.8%	31.3%	26%
store	5.6%	0.6%	5.1%	14.3%	16.7%	9%
add	14.4%	8.5%	23.8%	14.6%	11.1%	14%
sub	1.8%	0.3%		0.5%		0%
mul				0.1%		0%
div						0%
compare	15.4%	26.5%	8.3%	12.4%	5.4%	13%
load imm	8.1%	1.5%	1.3%	6.8%	2.4%	3%
cond. branch	17.4%	24.0%	15.0%	11.5%	14.6%	16%
uncond branch	1.5%	0.9%	0.5%	1.3%	1.8%	1%
call	0.1%	0.5%	0.4%	1.1%	3.1%	1%
return, jmp ind	0.1%	0.5%	0.5%	1.5%	3.5%	1%
shift	6.5%	0.3%	7.0%	6.2%	0.7%	4%
and	2.1%	0.1%	9.4%	1.6%	2.1%	3%
or	6.0%	5.5%	4.8%	4.2%	6.2%	5%
other (xor, not)	1.0%		2.0%	0.5%	0.1%	1%
load FP						0%
store FP						0%
add FP						0%
sub FP						0%
mul FP						0%
div FP						0%
compare FP						0%
mov reg-reg FP						0%
other FP						0%

FIGURE 2.26 DLX instruction mix for five SPECint92 programs. Note that integer register-register move instructions are included in the add instruction. Blank entries have the value 0.0%.

16

DLX SPECfp92 Instruction Mix

Instruction	doduc	ear	hydro2d	mdljdp2	su2cor	FP average
load	1.4%	0.2%	0.1%	1.1%	3.6%	1%
store	1.3%	0.1%		0.1%	1.3%	1%
add	13.6%	13.6%	10.9%	4.7%	9.7%	11%
sub	0.3%		0.2%		0.7%	0%
mul						0%
div						0%
compare	3.2%	3.1%	1.2%	0.3%	1.3%	2%
load imm	2.2%		0.2%	2.2%	0.9%	1%
cond. branch	8.0%	10.1%	11.7%	9.3%	2.6%	8%
uncond branch	0.9%	0.4%		0.4%	0.1%	0%
call	0.5%	1.9%			0.3%	1%
return, jmp ind	0.6%	1.9%			0.3%	1%
shift	2.0%	0.2%	2.4%	1.3%	2.3%	2%
and	0.4%	0.1%			0.3%	0%
or		0.2%	0.1%	0.1%	0.1%	0%
other (xor, not)						0%
load FP	23.3%	19.8%	24.1%	25.9%	21.6%	23%
store FP	5.7%	11.4%	9.9%	10.0%	9.8%	9%
add FP	8.8%	7.3%	3.6%	8.5%	12.4%	8%
sub FP	3.8%	3.2%	7.9%	10.4%	5.9%	6%
mul FP	12.0%	9.6%	9.4%	13.9%	21.6%	13%
div FP	2.3%		1.6%	0.9%	0.7%	1%
compare FP	4.2%	6.4%	10.4%	9.3%	0.8%	6%
mov reg-reg FP	2.1%	1.8%	5.2%	0.9%	1.9%	2%
other FP	2.4%	8.4%	0.2%	0.2%	1.2%	2%

FIGURE 2.27 DLX instruction mix for five programs from SPECfp92. Note that integer register-register move instructions are included in the add instruction. Blank entries have the value 0.0%.

17

ISA Design Example

- You propose new version of DLX called DLX-lite
- DLX-lite only has a register-indirect memory addressing mode.
- You claim that DLX-lite reduces the CPI of loads and stores from 2.0 to 1.0
- The CPI of all other instructions is 1.0 on both processors
- Is DLX-lite faster on integer programs?

18

Summary

- ISAs are shaped by implementations, compilers, and history
  - » Limited hardware + primitive compilers --> stack architectures
  - » HLL support + high memory cost --> orthogonal CISC
  - » Limited hardware + much assembly programming
    --> small register set designs
  - » Efficient pipelining + optimizing compilers + good caches
    --> RISC architectures
- What's coming?
  - » More support for multiple instruction issue
  - » Increased tolerance for long memory latency
  - » Continued "complexification" of RISC, within reason
  - » Even tighter compiler/ISA integration; compiler-generated hints for branches, memory accesses, etc.
  - » Support for multiple threads of control

19

Other RISC variations: PowerPC condition register

- Compromise between a global condition code later tested by branch instructions (hard to pipeline), and explicitly set compare results in registers (extra instructions):
  - » A condition register (CR) that has 8 independent 4-bit condition codes:
    - – one set implicitly by integer operations that request it "add" doesn't, "add." does
    - – one set implicitly by floating-point operations that request it
    - – six available to be set explicitly by compare instructions
  - » Branch instructions can test any of the 8 condition codes
- Allows the compiler to reschedule and separate them sufficiently for high performance without destroying branch conditions before they are used.
- For combined conditions to reduce branches: and/or/nand/nor/xor between any CR bits, results to CR bit.

20

Other RISC variations: DEC Alpha AXP (21064)

- Conditional register-move instruction
  - » if (Ra <cond> 0) Rc = Rb
- Scaled addition/subtraction for array indexing
  - » Rc = Ra + k x Rb, where k=4 or 8
- Byte manipulation instructions: compare, extract, insert, mask, zero
- PAL: Privileged Architecture Library
  - » disabled instruction sequences with access to internal machine state; almost main-memory microcode
  - » implementation-dependent
  - » used for interrupt dispatching, virtual memory control, etc.

21