RISC Instruction Set Architecture

Simple load/store instruction set

- data transfer, arithmetic/logical, control
- general purpose registers
- use instructions in combination for more complex operations

Easily decoded (in parallel) instruction set

- fixed length instructions
- few instruction formats
- may fields in common, different fields in the same bit locations
- addressing modes implicit in opcode

Designed for pipeline efficiency

- primitive instructions
- instructions require the same amount of work
- few addressing modes (register, immediate, displacement/ indexed)
- fast versions of some instructions

Instruction Set Architecture (RISCs)

Still some issues

- condition codes (S) vs. condition registers (A)
- GPR organization
    - register windows (S)
    - flat file (everyone else)
- sizes of immediates
- move between integer and FP registers
- support for integer divide & FP operations
- how CISCy do we get?

New RISC Architectures

A: Alpha AXP: **21164**

S: SPARC V9: **UltraSPARC**

M: MIPS-IV: **R10000**

P: PowerPC: **620**

- 64 bit architectures
  - 64b registers (32 I, 32 FP), datapath
  - 64b addresses (ld,st,br)
  - linear address space (no segmentation)
  - software not allowed to use unimplemented high address bits
- 4-way superscalar

X: x86: **Pentium Pro** (P6)

- 32 bit architecture

Backwards Compatibility

General approach

- start from scratch (A)
  - some 32b arithmetic instructions for backwards compatibility for translators for other DEC architectures
- 2 instruction subsets, mode for each (M)
  - 64b instructions: ld/st, arithmetic, shift, cond br
  - illegal-instruction trap on 64b instructions in 32 bit mode
- hybrid (S)
  - no changes to existing instruction set
  - extra 64b-operand instructions
  - all instructions available for all programs but a mode bit to truncate to 32 bit virtual addresses

Backwards Compatibility

ld/st

- 64b values in 1 instruction (all)
- 32b ld, zero extended, for backwards compatibility (S)
- operand sizes
  - byte, halfword, word, double (S)
  - ld/st 32/64 only (A)
    extract, insert for smaller operands
    cycle time considerations

shift right

- specify operand width (S)
  so sign/zero extend from correct bit (31/63)

FP registers

- 1 set holds single & double-precision operands (A)
- mode bit to enable 16 new double-precision registers (M)
- "clever register addressing scheme" (S)

Backwards Compatibility

conditions

Alpha

- condition registers
  no overflow condition --> trap on overflow
  so separate 64b/32b integer add & subtract instructions
- FP CCs

SPARC

- 64- bit & 32-bit integer condition codes
  1 set of arithmetic instructions
  conditional branch & overflow instructions test either CC set
- 4 sets of FP CCs (3 new: the wide issue width problem)

delayed branches

- require more instructions in a superscalar (S)

New Instructions

Purpose:

- for better performance
  - to better match new implementations (e.g., wide issue)
  - to take advantage of new compiler optimizations
  - to support new, compute-intensive applications

New Instructions

data prefetch (A S)

- increases chance of a cache hit
- caveats:
  - may displace data still being used
  - may saturate a multiprocessor bus
  - an extra instruction
- characteristics
  - hint
  - nonblocking (uses a **lockup-free cache**)
  - nonfaulting

speculative load (S)

- **renaming registers** for loaded data

instruction prefetch (S)

- hint to prefetch way before the branch

Eliminate load or fetch latency

New Instructions

branch prediction (static)

- branch prediction bit (S)
- based on the sign of the branch displacement (A)
- on an indirect branch the displacement field is a hint for the target address (A)

  L1 cache access starts before instruction dispatch

conditional move instruction (A S)

also called **predicated execution**

- test condition & move if true (cheap if simple code sequences)
- replaces conditional branch & move (expensive)
- **pluses & minuses?**

Both eliminate branch latency

New Instructions

loop support

- scaled add/subtract/compare (A)
- branch on count (P)

Eliminates instructions for a common programming idiom

New Instructions

multimedia instructions (implementation-dependent) (S:VIS)

- targeted for graphics, audio and video data
  - partitioned arithmetic
  - `expand`, `fpack`, partial store, logical ops on FP unit & registers
  - special: `sad` (`pdist`), `pmerge`, `edge`, `array`
- ramifications on the architecture
  - new instructions
  - new formats
- surprisingly small proportion of die
- how used? assembly language library routines

Support an important compute-intensive application with small hardware cost

New Instructions

additional register window support (S)

purpose:

- fast context switching for fine-grain multithreaded applications (transactions processing applications)
  - second set of global registers for another thread
  - windows hold registers for two threads; **processor state** marks the boundaries
- fast register clearing after process termination in a secure environment (financial or business applications)
  - **clean-window trap** rather than save-window trap for a window that belongs to a terminated process

x86 History

8080

- 8b
- accumulator machine

8086 (1978)

- 16b
- special-purpose registers

8087 (1980)

- floating point coprocessor
- extended stack architecture
- some register-memory instructions

80286 (1982)

- 24b
- memory-mapping & protection model

80386 (1985)

- 32b
- GPR instructions
- added 2-level paging to segmented addressing

80486, Pentium, Pentium Pro (P6)

Today's x86 (relative to RISC)

Register-memory architecture

7 data memory addressing modes + multiple sizes for displacements

- restrictions on what registers can be used with what modes

Special-purpose registers

Hybrid stack architecture for floating point

- has been extended with addressing modes

More opcodes:

- different opcodes for intra/inter segment transfers of control
- transcendental functions, string manipulation

Instruction encoding

- variable length instructions
  (different numbers of operands, different operand sizes,
  prefixes, postbytes to specify addressing mode, etc.)
- lots of formats, tight encoding

Segmentation with paging

Backwards Compatibility

Each register has a special function

3 data sizes

- mode bit for 8b & one larger; prefix to change to the other larger
- allows instructions for all data sizes

Special categories of instructions

Real addressing, segmentation without paging, segmentation with paging

Stack model for floating point when most programs use arbitrary memory operand addresses

Backwards Compatibility

Special problem for P6 when using 8b & 16b codes

Implemented low-performance solutions

- **unaligned loads**
  data might span 2 cache blocks
  $\Rightarrow$ flush store buffer
- **partial register ops**
  data in the reorder buffer can't be accessed on byte boundaries
  $\Rightarrow$ wait until data is written to physical registers
- **segment register writes**
  large code & data spaces don't fit into one 64KB segment (16b address)
  must drain the pipeline so instructions in this segment complete before the segment changes
  $\Rightarrow$ acts like a barrier between code in different segments

P6

New instructions

- conditional move based on condition codes
- static branch prediction based on the direction of the branch
- MMX + KNI for multimedia
    - MMX registers mapped onto FP registers (use in a stack-based address model)
        - only 8, so can't use *register-based* 4x4 digital filter (a 3-bit register specifier)
    - 8 new KNI FP registers, directly addressed
    - KNI: `sad`, so for real-time MPEG-1 encoding (motion estimation)
    - MMX: some saturating operations to avoid overflow & underflow traps
    - KNI: streaming store to bypass the caches
    - KNI: data prefetch to arbitrary cache destinations
    - other instructions

RISC vs. CISC

Advantage of RISC depends on (among other things):

- chip technology
- processor complexity

Pre 1990: chip density **low** -- RISC processors **simple**

- single-chip CPUs (1986)
- onchip caches
- instruction decoding "large" part of execution cycle for CISCs

Post 1990: chip density **high** -- processors **complex**

- both RISC & CISC implementations fit on a chip with big L1 caches
- instruction decoding smaller time component:
    - multiple instruction issue
    - out-of-order execution
    - speculative execution

Other Important Factors

Clock rate

- all HW manipulates primitive instructions
- architecture: superpiplining
- good process technology (.25 micron) --> small feature size

Compiler technology

- architecture features that help compilation
  - orthogonal architecture, simple architecture
  - primitive operations
  - lots of general purpose registers
  - no side effects

Ability of the design team

New/old architecture

- historical legacy takes time (whether RISC or CISC)

$$$

Wrap-up

What RISC ISAs look like today

- the new instructions
  - what they do
  - why they’re used
- differences from the original model

64b architectures

- backwards compatibility to old “word” sizes
- makes you realize how pervasive the “word” size is -- not just the addressable memory space

x86 features

RISC vs. CISC is not the simplistic debate it used to be

The machines are there to illustrate the concepts