Classifying ISAs

Concepts Introduced in Appendix A

- classifying instruction set architectures
- memory addressing
- operands
- operations
- interaction between compilers and architecture
- ISA design

Instruction Set Types

- stack architecture (zero address machine)
- accumulator architecture (one address machine)
- general-purpose register architecture (two and three address machines)
  - register-register (load-store)
  - register-memory
  - memory-memory

Instruction Set Types

- stack architecture (zero address machine)
- accumulator architecture (one address machine)
- general-purpose register architecture (two and three address machines)
  - register-register (load-store)
  - register-memory
  - memory-memory

Code Sequences for C = A + B

- stack
  
  Push A  --  M[SP]=M[A]; SP=SP+1;
  Push B  --  M[SP]=M[B]; SP=SP+1;
  Add     --  M[SP-2]=M[SP-1]+M[SP-2]; SP=SP-1;
  Pop C   --  C=M[SP-1]; SP=SP-1;

- accumulator
  
  Load A  --  AC=M[A];
  Add B   --  AC=AC+M[B];
  Store C --  M[C]=AC;

- register (register-memory)
  
  Load R1,A  --  r[1]=M[A];
  Store R3,C  --  M[C]=r[3];

- register (load-store)
  
  Load R1,A  --  r[1]=M[A];
  Load R2,B  --  r[2]=M[B];
  Store R3,C  --  M[C]=r[3];

Addressing Issues

- byte addressable
- number of bytes in a word
- byte order (little endian or big endian)
- alignment requirements
- extension of bytes and halfwords
DSP and other special-purpose processors sometimes have special purpose addressing modes to support specific types of applications.

- Modulo or circular addressing to support continuous streams of data.
- Bit reverse addressing to support fast fourier transforms.

Media and signal processor processors sometimes have special operations to support specific types of applications.

- partitioned arithmetic operations
- saturating arithmetic
- special instructions tailored for applications (e.g. multiply-accumulate)

Types and Sizes of Operands

- general-purpose operand types and sizes
  - char or unsigned char, 1 byte, two’s complement or unsigned
  - short or unsigned short, 2 bytes, two’s complement or unsigned
  - int or unsigned int, 4 bytes, two’s complement or unsigned
  - float, 4 bytes, IEEE FPS
  - long int or unsigned long int, 4 or 8 bytes, two’s complement or unsigned
  - pointer is 4 or 8 bytes
  - double, 8 bytes, IEEE FPS
  - long long int or unsigned long long int, 8 bytes, two’s complement or unsigned

- DSP types
  - fixed point, fractions between -1 and +1

Transfers of Control

- conditional branches (pc-relative)
- unconditional jumps (pc-relative or immediate)
- direct procedure calls (immediate)
- returns (register)
- indirect procedure calls (register)
- large switch statements (register)
Evaluating Branch Conditions

- condition codes (e.g. SPARC)
  \[ \text{IC} = r[2] \ ? \ r[3]; \]
  \[ \text{PC} = \text{IC} < 0, L1; \]
- condition register (e.g. MIPS)
  \[ r[1] = (r[2] < r[3]) \ ? \ 1 : 0; \]
  \[ \text{PC} = r[1] \neq r[0], L1; \]
- compare and branch (e.g. VAX)
  \[ \text{PC} = r[2] < r[3], L1; \]

Calling Conventions

- Identifying registers for special purposes.
  - stack pointer
  - return address
- Preserving the values of registers across calls.
  - caller save
  - callee save
- Transferring values between functions.
  - Passing argument values through registers and on the stack.
  - Returning a value.

Compiler Optimizations

- compiler optimization goals
  - Preserve semantic behavior.
  - Reduce execution time.
  - Decrease code size.
  - Reduce energy usage.
- levels of compiler optimizations
  - high-level (close to the source code level)
  - low-level (close to the machine code level)
  - local (within a basic block)
  - global (across basic blocks)
  - interprocedural (across function boundaries)

Register Allocation

- two levels of allocating registers:
  - register assignment - assigning temporaries to registers
  - register allocation - assigning live ranges of variables to registers
- areas of memory (most register allocation is performed for items on the stack)
  - run-time stack (some registers are dedicated to management of the stack)
  - static data (sometimes there is a dedicated register that points to the global data to reduce the cost of global data accesses)
  - heap
  - code (one register, the PC, is dedicated to point to the current instruction)
Tradeoffs When Encoding an Instruction Set

- Benefit of additional registers versus extra bits for referencing them.
- Simple formats simplify decoding and complex formats require less space.
- Fixed sized instructions simplify the fetch unit and pipelining and variable size instructions require less space.

Instruction Set Properties

- A well designed instruction set can make life much easier for the compiler writer.
  - regularity
  - provide primitives, not solutions
  - simplify tradeoffs

Computer Architecture Periods

- 1960s
  - stack architectures
- 1970s
  - high-level (CISC) architectures
- 1980s
  - load-store (RISC) architectures
- 1990s
  - doubling the address size
  - use of conditional execution
  - prefetch instructions
  - support for multimedia and signal processing
- 2000s
  - run-time translation into micro-ops
  - virtual machines and JIT compilation

Fallacies and Pitfalls

- Pitfall: Designing high-level instruction set features to support a high-level language structure.
- Fallacy: An architecture with flaws cannot be successful.
- Fallacy: You can design a flawless architecture.