Engineering 9859
CoE Fundamentals — Computer Architecture
Instruction Set Principles

Dennis Peters[1]

Fall 2007

[1] Based on notes from Dr. R. Venkatesan

RISC vs. CISC

- *Complex Instruction Set Computers* (CISC) have 1000s of instructions.
- *Reduced Instruction Set Computers* (RISC) designed to efficiently execute about 100 instructions.
- Most modern processors have features of both RISC and CISC.
- Architects/designers should remember Amdahl's law:
  - Implement most frequently executed instructions efficiently in hardware
  - Some rarely executed instructions can even be microcoded.
- Simplicity and uniformity (consistency and orthogonality) in architectural choices will lead to hardware that has a short critical path (worst-case propagation delay), resulting in high performance.

Classifying IS Architectures

Classify based on type of internal storage in processor:

Stack Operands implicitly on top of stack.

Accumulator One operand implicitly in the accumulator.

Register-memory Only explicit operands, one may be in memory.

Register-register/load-store Only explicity register operands.

Load-store is used for almost all new architectures since 1980.

Instruction Types

- Arithmetic & Logic (ALU) instructions
  - Only register operands
  - Register operands plus one immediate operand (with instn.)
- Memory access instructions: Load & Store
- Transfer of control instructions:
  - branches and jumps that are conditional or unconditional,
  - procedure calls, returns,
  - interrupts,
  - exceptions, traps, etc.
- Special instructions: flag setting, etc.
- Floating point instructions
- Graphic, digital signal processing (DSP) instructions
- Input / Output instructions: Memory-mapped I/O?
- Other instructions

Instruction Occurrence

ALU (incl. ALU immediates)	25–40%
Loads	20–30%
Stores	10–15%
Conditional branches	15–30%
others	$> 2\%$

- Data processing applications use loads and stores more frequently than ALU instructions;
- numeric (scientific & engineering) applications use more ALU.
- Graphic processors, network processors, DSP, vector coprocessors will have a different mix.

Word and Instruction Size

- General purpose CPUs are most commonly 32- or 64-bit word based.
- Microcontrollers with 8-bit and 16-bit words are still common.
- Instruction size is variable or fixed.
  - Fixed is common in high-performance processors.
  - May be smaller than, equal to or larger than word size.
- Most GP CPUs use 64-bit words and 32-bit instructions.
- Very large instruction word (VLIW) processors are emerging.
- Memory access, especially writing, needs to be done one byte (8 bits) at a time due to character data type — memory is almost always byte-addressable even with larger word size.

Memory Organization

- Which bit is bit 0, lsb or msb? only a convention
- Every byte has unique address, so a word spans several addresses.
  - Example: 64-bit computer, word spans 8 addresses.
  - Must be consecutive addresses.
- Which byte is LSB?
  - *Little endian* — LSB is lowest address (little end).
  - *Big endian* — LSB is highest address (big end).
  - Also only a convention.
- Aligned or non-aligned memory access:
  - Aligned: gaps in memory but fast and simple hardware.
  - Non-aligned: efficient storage but complex and slow hardware.
- Most (but not all) modern processors restrict that the size of all operands in ALU instructions be equal to the word size again, for fast and simple hardware.

MIPS-64 Instruction Set

General Purpose Register (GPR), load-store architecture.

- 64-bit word, 32-bit instructions.
- 32 64-bit GPRs
  - R0 is read-only contains 0.
  - R31 is used with procedure calls.
- 32 32-bit Floating Point Registers (FPR) — accessed in pairs as 64-bit entities..
- Instruction formats:

  R-type 6-bit opcode, 3x5-bit register ids, 6-bit function.
  - Used by ALU reg-reg instructions

  I-type 6-bit opcode, 2x5-bit register ids, 16-bit disp./imm.
  - Used by ALU reg-imm., load, store, conditional branch.

  J-type used only for J and JAL

MIPS-64 Instruction Set (cont'd)

Load, store byte (8 bits), half word (16), word (32), double word (64).

ALU add, subtract, AND, OR, XOR, shifts (all register-register).

FP single- or double-precision

- move, add, subtract, multiply, divide
- converstion to/from integer
- compare, test with branch (true/false)
- paired single — operate on two single-precision operands in a single double-precision register.

Branch e.g.: BEQ, BNE, BNEQZ (always conditional).

Jump to name or using register, with or without link (return address) in R31.

Sample Code

```
      DADDI   R1, R0, #8000      ;1000 words in array
here: SUBDI   R1, R1, #8         ;Word has 8 bytes
      LD      R2, 30000(R1)      ;Array starts at 30000
      DADD    R2, R2, R2         ;Double contents
      SD      30000(R1), R2      ;Store back in array
      BNEZ    R1, here           ;if not done, repeat
```

A 1000-word array, starting at the memory location 30000, is accessed word-by-word (64-bit words), and the value of each word is doubled.

Memory Access (for sample code)

- Approximately, 40% of all instructions executed access memory.
- Without cache, memory is accessed $2 * 1000$ times during the execution of the code.
- If L1 data cache is at least 64kB large, then there would be only compulsory misses of the cache during the execution of the code.

Compilers and Architecture

- Architectures must be developed in such a manner that compilers can be easily developed.
- Architects should be aware of how compilers work.
- Compiler designers can design good products if they are aware of the details of the architecture.
- Simple compilers perform table look-up.
- Optimizing compilers can come up with machine code that is up to 50 times faster in execution.

Optimization

Optimization is done by compilers at various levels:

- Front-end: transform to common intermediate form
- High-level: loop transformations and procedure in-lining
- Global optimizer: global and local optimizations; register allocation
- Code generator: machine-dependent optimizations

Example: compiler & architecture

- A program execution entails running 10 million instructions.
- 45% of these are ALU instructions.
- An optimizing compiler removes a third of them.
- The processor runs at 1 GHz clock speed.
- Instruction execution times are:
  - 1 cc for ALU,
  - 2 cc for load & store that comprise 25% of all operations, and
  - 3 cc for conditional branches.
- Compute MIPS ratings and CPU times before and after optimization.

Example solution

Before optimization:

$$\begin{aligned} \mathrm{CPI}_{\mathrm{avg}} &= \\ \text{MIPS} &= \\ \text{CPU time} &= \end{aligned}$$

After optimization:

$$\begin{aligned} \mathrm{CPI}_{\mathrm{avg}} &= \\ \text{MIPS} &= \\ \text{CPU time} &= \end{aligned}$$