Instruction Set Architecture (ISA) - Part IV

CS6133 - Computer Architecture I

Vikram Padman

Polytechnic Institute of New York University

vikram@poly.edu

Agenda

1. **Introduction** - Part I
2. **Classifying ISA** - Part I, II, III and IV
   - Encoding Instruction Set - Part IV

Encoding Instruction Set

Representation

Many instructions were shown in past lectures using mnemonics such as: ld, sw, bnez ... etc. How does the CPU understand these mnemonics?

Encoding Instruction Set

Representation

Many instructions were shown in past lectures using mnemonics such as: ld, sw, bnez ... etc. How does the CPU understand these mnemonics?

- CPU does not understand mnemonics!
- Instructions are encoded in binary format (1's & 0's)
- A binary number is assigned to every operation, register and other CPU resources
- An encoded binary instruction could fixed or variable size
  - MIPS instructions are fixed to 32-bit, while x86 instructions are variable size
  - We will concentrate on MIPS.
- Instruction size effect the application size, and CPU's implementation complexity

Encoding Instruction Set

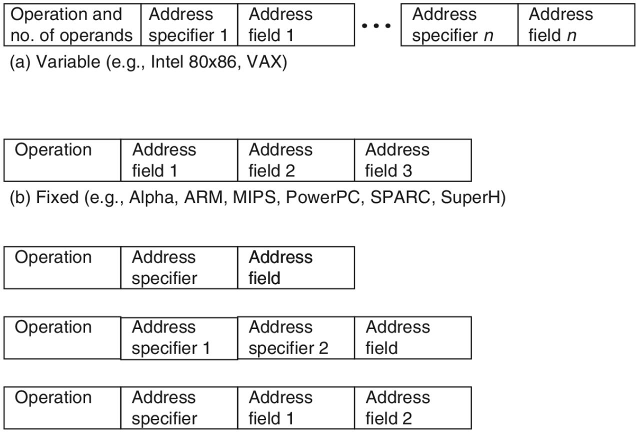
Instruction Length and Format

- Software and architectural requirements drive instruction encoding, format and length.
- For example: Let say a CPU should supports:
  1. 10 Instructions - 4 bits
  2. 6 Addressing modes - 3 bits
  3. 64 Register - 6 bits
  4. 1 to 5 operands per instruction - 30 bits
- Instruction size should be 37 bits - fixed format and 13 bits - variable format
- Number of registers, addressing modes, and operands has a huge impact on the size!

Encoding Instruction Set

Instruction Length and Format

Three popular encoding styles are:



(c) Hybrid (e.g., IBM 360/370, MIPS16, Thumb, TI TMS320C54x)

From "Computer Architecture – A Quantitative Approach" page A-22

Encoding Instruction Set

MIPS Instruction Set

- MIPS CPU is used as an example in this lecture.
- MIPS is popular architecture in high-end and embedded processor market.
- MIPS is a RISC CPU and has a fixed 32-bit instruction format.

Encoding Instruction Set

MIPS - Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination
    add a, b, c # a gets b + c
- All arithmetic operations have this form. For example:
  - C Code: f = (f + h) - (i - j);
  - Compile MIPS code:
    add \$t0, \$s1, \$s2
    sub \$t1, \$s3, \$s4
    sub \$s0, \$t0, \$t1
- Arithmetic instructions use register operands
- MIPS has a 32 32-bit register file, numbered 0 to 31.
- Register mnemonics are:
  \$t0, \$t1, , \$t9 for temporary values
  \$s0, \$s1, , \$s7 for saved variables

Encoding Instruction Set

MIPS - Memory Operands

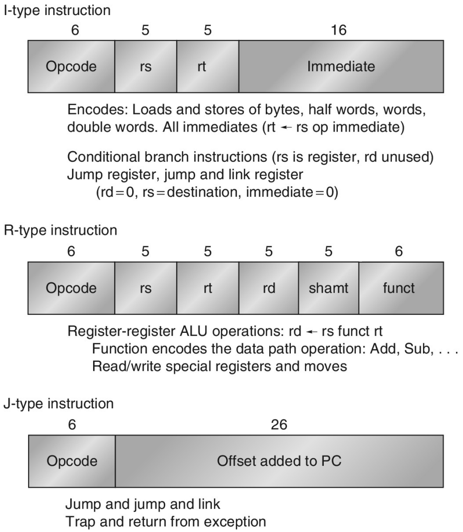
- Main memory used for larger data types (i.e arrays, structures, dynamic data)
- To apply arithmetic operations on large data:
  - Load values from memory into registers
  - Store result from register to memory
- Memory is byte addressed. Each address identifies a byte (8-bits)
- MIPS only supports word (32-bit) aligned memory access in Big Endian packing format.
- C code: int g,h,A[16]; g = h + A[8];
- Compiled MIPS code (Assume that g, h and A's base are already loaded in $s1,$s2, and $s3):
  lw $t0, 64($s3); # load word
  add $s1, $s2, $t0;

Encoding Instruction Set
MIPS - Immediate Operands

- Constant data specified in an instruction: addi \$s3, \$s3, 4
- No subtract immediate instruction: addi \$s2, \$s1, -1
- MIPS register 0 (\$zero) is the constant 0. Used to move data between registers: add \$t2, \$s1, \$zero

Encoding Instruction Set

MIPS - Instruction Formats


I-type instruction
6
5
5
16
Opcode
rs
rt
Immediate
Encodes: Loads and stores of bytes, half words, words, double words. All immediates (rt ← rs op immediate)
Conditional branch instructions (rs is register, rd unused)
Jump register, jump and link register
(rd = 0, rs = destination, immediate = 0)
R-type instruction
6
5
5
5
5
6
Opcode
rs
rt
rd
shamt
funct
Register-register ALU operations: rd ← rs funct rt
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves
J-type instruction
6
26
Opcode
Offset added to PC
Jump and jump and link
Trap and return from exception

From "Computer Architecture – A Quantitative Approach" page A-35