ISA formats

Prof Rajeev Barua

E-A 001 -- Slide set 9

Instruction formats

The instruction format specify which bits of the instruction are what. That is called the ISA.

Instructions are composed of:

- Opcode -- bit pattern specifying instruction type.
- Location of operands (may be specified as register values, memory locations, or immediates)
  - Register specifiers are just their names (e.g., $3). Here, the constant 3 can be stored in the ISA instruction.
  - Constants may be numeric or address displacements.

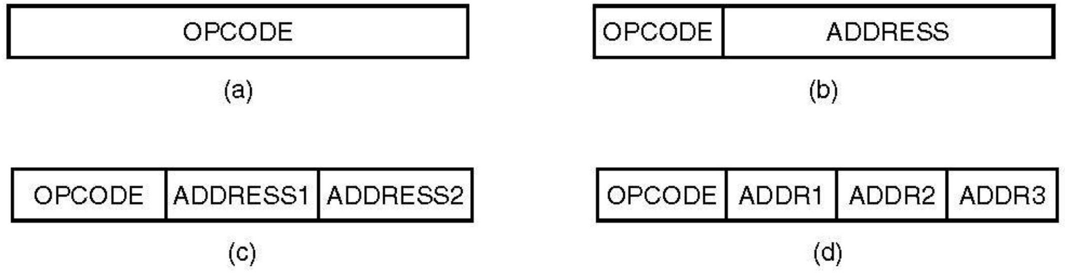

Figure 5-9. Four common instruction formats: (a) Zero-address instruction. (b) One-address instruction (c) Two-address instruction. (d) Three-address instruction.

Better to have fixed length opcode (should know from opcode which of the four formats on the left the current instruction is.)

How long are instructions?

- Fixed length: all instructions have the same length (e.g., MIPS32) OR
- Variable length: Some instructions are longer than others. (e.g., x86)

Advantage of fixed length instructions:

- Field interpretations can be opcode independent implying that we can interpret an operand field without waiting for the opcode to be decoded. If opcode reveals that operand is non-existent, then later in the pipeline, we can ignore and discard the field we read.
- Fixed fetch latency ⇒ easier to pipeline.

Downside:

- Might have immediate fields that are very short.
  - ➢ Not that serious since needing big immediates is rare.
  - ➢ In rare cases when needed, ISAs provide ways to load a long constant into a register using more than one instruction.

Design criteria for Instruction formats

Deciding exact format is difficult.

- Need to decide encoding of opcode, for example, for easy decoding.

Difficult because:

- Tradeoffs change over several decades long life of an ISA. E.g.:
  - ➢ CISC might be okay if no pipelining is used.

Short instructions are better because:

- Less time to read an instruction
- Less memory space (programs are growing very rapidly).
  - ➢ Memory speeds growing slower than proc (7% vs 60%).

Lower bound on instruction size:

- Must still fit all opcodes and operands.
- Must fit reasonable immediate sizes.

Other design criteria:

- Leave enough unused opcodes for future additions to ISA.

Expanding Opcodes

This is a way to fit in more opcodes into instructions.

Best explained through an example.

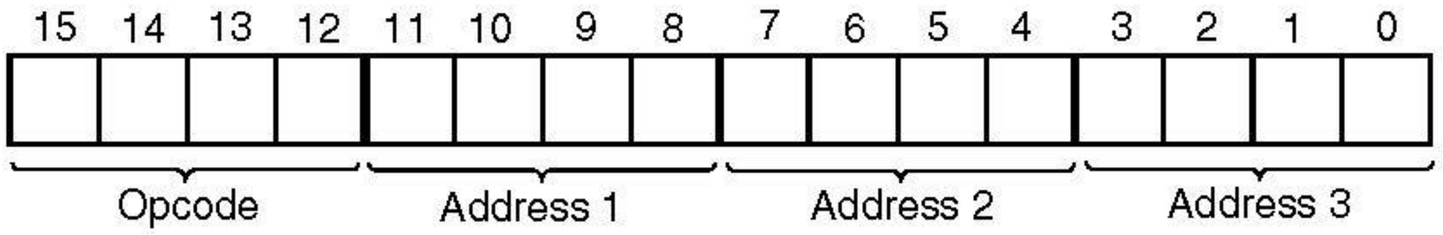

Figure 5-11. An instruction with a 4-bit opcode and three 4-bit address fields.

If all instructions have 4 bits of opcode ⇒ 16 instructions max.

But we can have more!

Idea to expand the number of opcodes:

For 3-address instructions, the opcode needs to be short.

- Here opcode has to fit in original opcode space in figure above,

For 2-address instructions, opcode can fit more bits. How many more? Either:

- 3-address opcode value has more than one reserved value, one for each possible opcode length. For those special values, the unused address fields can store part of the opcode.
- 3-address opcode value has just one reserved value. Then 2-address opcode has more reserved values for one-operand or zero-operand instructions. (Thereafter similar idea)

Advantage of former: faster decoding (two steps in decoding, instead of the original one step, but faster than the 3-step decoding of the latter.)

Advantage of latter: less reserved codes in 3-address instructions.

Expanding opcodes are used with care in today’s machines, because they increase the decoding time. Used in a manner that does not interfere with pipeline (E.g.: we will see this in MIP32)