Instruction set design

Note Title

Topics:
1. Endianness
2. Number of operands
3. Expanding opcodes
4. Addressing modes

1. Endianness

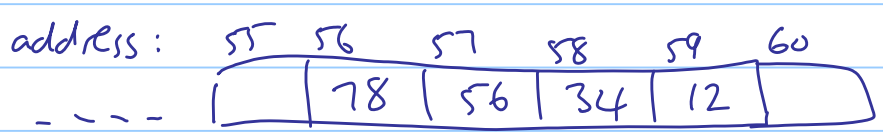
most significant byte

least significant byte

Suppose we want to store the 32-bit hexadecimal value 12345678 at location 56 on a byte addressable machine. There are two sensible ways of doing this:

← big endian (most sig. byte first)

OR



← little endian (least sig. byte first

Some computers do it one way; some the other.

e.g. Intel chips: little-endian; Motorola and Sun chips: big-endian
MIPS chips: can boot into either
network transmissions: big-endian

Hence, we often need to reverse the order of bytes in an (unsigned) integer e.g. to send data over the network from a little-endian machine.

For example, IA32 includes the BSWAP instruction
(refer to spec, link is on resources page)

Activity: download, compile & run endian.c.
Use this to determine the endianness of the lab machines and/or your own laptop.

Demo: Use wireshark to see source and dest values in an IP packet
(in network order – i.e. big endian – of course)

(3) Number of operands

When designing an instruction set, there's a tradeoff between

long instructions with many operands **reduces** instructions/prog

vs

short instructions with few operands **reduces** cycles/instruction

recall:

$$\text{program time} = \frac{\text{instructions}}{\text{prog}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

See textbook example 5.6, p 278 (2nd ed: example 5.1, p250)
There we see a particular computation ($z = xy + wu$) that requires

3 instructions with 3 operands
6 —"— 2 operands
7 —"— 1 operand

Exercise: How many instructions are required to compute $Z = (X+Y) \times (X+Y+W)$? when using 1-, 2-, or 3-operand instructions?

(3) Expanding opcodes

We can combine the advantages of differing numbers of operands by using expanding opcodes, in which certain prefixes imply longer opcodes:

Very simple example of expanding opcodes, and various addressing modes

6-bit instruction, 2-bit addresses, 2-bit word size, one accumulator

Machine language	mnemonic	meaning
00 A1 A2	ADD A1 A2	AC = M[A1]+M[A2]
01 A1 A2	SUB A1 A2	AC = M[A1]-M[A2]
10 A1 V	STOREIM A1 V	M[A1] = V
11 00 A1	LOAD A1	AC = M[A1]
11 01 A1	STORE A1	M[A1] = AC
11 10 A1	LOADI A1	AC = M[M[A1]]
11 11 00	SKIPZERO	Skip if AC==0
11 11 01	SKIPNEG	Skip if AC<0
11 11 10	CLEAR	AC=0
11 11 11	HALT	Halt

demo:

see IA32 spec vol 2 §2.1.2 – observe that first byte OF implies 2-byte opcode

activity:

design an expanding opcode for 9-bit instructions, with 3-bit addresses, containing 7 2-operand codes, 7 1-operand codes, and 8 0-operand codes.

④ Addressing modes

Instructions typically use one of 4 different addressing modes:

fictitious examples – don't exist in MARIE

description	example	pseudocode
1. immediate – operand is the parameter	LOADIM 5	AC = 5
2. direct – operand is the address of the parameter	LOAD 5	AC = M[5]
3. indirect – operand is the address of the address of the parameter	LOADI 5	AC = M[M[5]]
4. indexed – address of the parameter is the operand plus the value in a base or index register	LOADIND 5	AC = M[5+BR]

fictitious "Base register"