Computer Architecture
(CSC-3501)
Lecture 16
(25 Mar 2008)

Seung-Jong Park (Jay)
http://www.csc.lsu.edu/~sjpark

Announcement

Chapter 5 Objectives

- Understand the factors involved in instruction s et architecture design.
- Gain familiarity with memory addressing mode s.
- Understand the concepts of instruction-level pi pelining and its affect upon execution performa nce.

5.1 Introduction

- This chapter builds upon the ideas in Chapter 4.
- We present a detailed look at different instructio n formats, operand types, and memory access methods.
- We will see the interrelation between machine o rganization and instruction formats.
- This leads to a deeper understanding of comput er architecture in general.

5.2 Instruction Formats

Instruction sets are differentiated by the following:

- Number of bits per instruction.
- Stack-based or register-based.
- Number of explicit operands per instruction.
- Operand location.
- Types of operations.
- Type and size of operands.

5.2 Instruction Formats

Instruction set architectures are measured acc ording to:

- Main memory space occupied by a program.
- Instruction complexity.
- Instruction length (in bits).
- Total number of instructions in the instruction set.

5.2 Instruction Formats

In designing an instruction set, consideration is given to:

- Instruction length.
  - Whether short, long, or variable.
- Number of operands.
- Number of addressable registers.
- Memory organization.
  - Whether byte- or word addressable.
- Addressing modes.
  - Choose any or all: direct, indirect or indexed.

5.2 Instruction Formats

- Byte ordering, or *endianness*, is another major arc hitectural consideration.
- If we have a two-byte integer, the integer may be stored so that the least significant byte is followed by the most significant byte or vice versa.
  - In *little endian* machines, the least significant byte i s followed by the most significant byte.
  - *Big endian* machines store the most significant byte first (at the lower address).

5.2 Instruction Formats

- As an example, suppose we have the hexadecima l number 12345678.
- The big endian and small endian arrangements of the bytes are shown below.

Address ⟶	00	01	10	11
Big Endian	12	34	56	78
Little Endian	78	56	34	12

5.2 Instruction Formats

- Big endian:
  - Is more natural.
  - The sign of the number can be determined by lookin g at the byte at address offset 0.
  - Strings and integers are stored in the same order.
- Little endian:
  - Makes it easier to place values on non-word bounda ries.
  - Conversion from a 16-bit integer address to a 32-bit integer address does not require any arithmetic.

Standard…What Standard?

- Pentium (80x86), VAX are little-endian
- IBM 370, Moterola 680x0 (Mac), and most RISC are big-endi an
- Internet is big-endian
  - Makes writing Internet programs on PC more awkward!
  - WinSock provides htoi and itoh (Host to Internet & Internet to Host) func tions to convert

5.2 Instruction Formats

- The next consideration for architecture design con cerns how the CPU will store data.
- We have three choices:
  1. A stack architecture
  2. An accumulator architecture
  3. A general purpose register architecture.
- In choosing one over the other, the tradeoffs are si mplicity (and cost) of hardware design with executi on speed and ease of use.

5.2 Instruction Formats

- In a stack architecture, instructions and operands are implicitly taken from the stack.
  - A stack cannot be accessed randomly.
- In an accumulator architecture, one operand of a bin ary operation is implicitly in the accumulator.
  - One operand is in memory, creating lots of bus traffic.
- In a general purpose register (GPR) architecture, reg isters can be used instead of memory.
  - Faster than accumulator architecture.
  - Efficient implementation for compilers.
  - Results in longer instructions.

5.2 Instruction Formats

- Most systems today are GPR systems.
- There are three types:
  - Memory-memory where two or three operands may be in memory.
  - Register-memory where at least one operand must be in a register.
  - Load-store where no operands may be in memory.
- The number of operands and the number of availa ble registers has a direct affect on instruction lengt h.

5.2 Instruction Formats

- Stack machines use one- and zero-operand instru ctions.
- **LOAD** and **STORE** instructions require a single mem ory address operand.
- Other instructions use operands from the stack im plicitly.
- **PUSH** and **POP** operations involve only the stack's t op element.
- Binary instructions (e.g., **ADD**, **MULT**) use the top tw o items on the stack.

5.2 Instruction Formats

- Stack architectures require us to think about arithm etic expressions a little differently.
- We are accustomed to writing expressions using *in fix* notation, such as: Z = X + Y.
- Stack arithmetic requires that we use *postfix* notati on: Z = XY+.
  - This is also called *reverse Polish notation*, (somewh at) in honor of its Polish inventor, Jan Lukasiewicz (1878 - 1956).

5.2 Instruction Formats

- The principal advantage of postfix notation is th at parentheses are not used.
- For example, the infix expression,

  Z = (X × Y) + (W × U),

  becomes:

  Z = X Y × W U × +

  in postfix notation.

5.2 Instruction Formats

- In a stack ISA, the postfix expression,

  Z = X Y × W U × +

  might look like this:

```
PUSH X
PUSH Y
MULT
PUSH W
PUSH U
MULT
ADD
PUSH Z
```

Note: The result of a binary opera tion is implicitly s tored on the top of the stack!

5.2 Instruction Formats

- In a one-address ISA, like MARIE, the infix expression,

  Z = X × Y + W × U

  looks like this:

```
LOAD X
MULT Y
STORE TEMP
LOAD W
MULT U
ADD TEMP
STORE Z
```


5.2 Instruction Formats

- In a two-address ISA, (e.g.,Intel, Motorola), the infix expression,

  Z = X × Y + W × U

  might look like this:

```
LOAD R1,X
MULT R1,Y
LOAD R2,W
MULT R2,U
ADD  R1,R2
STORE Z,R1
```

Note: One-address ISAs usually require one operand to be a register.

5.2 Instruction Formats

- With a three-address ISA, (e.g.,mainframes), the infix expression,

  Z = X × Y + W × U

  might look like this:

```
MULT R1,X,Y
MULT R2,W,U
ADD  Z,R1,R2
```

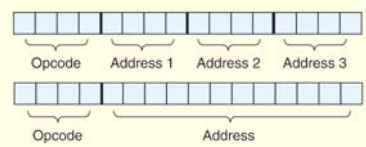
Would this program execute faster than the corresponding (longer) program that we saw in the stack-based ISA?

5.2 Instruction Formats

- We have seen how instruction length is affected by the number of operands supported by the ISA.
- In any instruction set, not all instructions require the same number of operands.
- Operations that require no operands, such as **HALT**, necessarily waste some space when fixed-length instructions are used.
- One way to recover some of this space is to use expanding opcodes.

5.2 Instruction Formats

- A system has 16 registers and 4K of memory.
- We need 4 bits to access one of the registers. We also need 12 bits for a memory address.
- If the system is to have 16-bit instructions, we have two choices for our instructions:



5.2 Instruction Formats

- If we allow the length of the opcode to vary, we could create a very rich instruction set:

```
0000 R1   R2   R3      } 15 3-address codes
1110 R1   R2   R3      }

1111 0000 R1   R2      } 14 2-address codes
1111 1101 R1   R2      }

1111 1110 0000 R1      } 31 1-address codes
1111 1111 1110 R1      }

1111 1111 1111 0000    } 16 0-address codes
1111 1111 1111 1111    }
```