جامعة ألأنبار

كلية علوم الحاسوب وتكنولوجيا المعلومات

قسم أنظمة شبكات الحاسوب

المرحلة الثانيه

Computer Architecture

التدريسي: أ.م.د عمر منذر حسين

A Closer Look at Instruction Set Architectures

5.1 Introduction.

- Machine instructions consist of opcodes and operands.
- The opcodes specify the operations to be executed; the operands specify register or memory locations of data.

5.2 Instruction Formats

instruction sets are differentiated by the following features:

- Operand storage (data can be stored in a stack structure or in registers or both).
- Number of explicit operands per instruction (zero, one, two, and three being the most common).
- Operand location (instructions can be classified as register-to-register, register-to-memory, or memory-to-memory, which simply refer to the combinations of operands allowed per instruction).
- Operations (including not only types of operations but also which instructions can access memory and which cannot).
- Type and size of operands (operands can be addresses, numbers, or even characters).

Instruction set architectures are measured according to:

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

In designing an instruction set, consideration is given to:

- Short instructions are typically better because they take up less space in memory and can be fetched quickly.

- Instructions of a fixed length are easier to decode but waste space.
- Memory organization affects instruction format. If memory has, for example, 16- or 32-bit words and is not byte-addressable, it is difficult to access a single character. For this reason, even machines that have 16- 32- or 64-bit words are often byteaddressable, meaning every byte has a unique address even though words are longer than 1 byte.
- A fixed length instruction does not necessarily imply a fixed number of operands. We could design an ISA with fixed overall instruction length, but allow the number of bits in the operand field to vary as necessary.
- Byte ordering, or endianness, is another major architectural 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 most significant byte is followed by the least significant byte.
    - Big endian machines store the least significant byte first.
- As an example, suppose we have the hexadecimal 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

FIGURE 5.1 The Hex Value 12345678 Stored in Both Big and Little Endian Formats

- Big endian:

  – Is more natural.

– The sign of the number can be determined by looking 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 boundaries.

– Conversion from a 16-bit integer address to a 32-bit integer address does not require any arithmetic.

5.2.1 Internal Storage in the CPU: Stacks Versus Registers

- Once byte ordering in memory is determined, the hardware designer must make some decisions about how the CPU should store data. This is the most basic means to differentiate ISAs. There are three choices:
  1. A stack architecture
  2. An accumulator architecture
  3. A general-purpose register (GPR) architecture
- In choosing one over the other, the tradeoffs are simplicity (and cost) of hardware design with execution speed and ease of use.
- 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 binary operation is implicitly in the accumulator.

– One operand is in memory, creating lots of bus traffic.

- In a general purpose register (GPR) architecture, registers can be used instead of memory.

– Faster than accumulator architecture.

– Efficient implementation for compilers.

– Results in longer instructions.

- Most systems today are GPR systems.
- The general-purpose architecture can be broken into three classifications, depending on where the operands are located. 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 available registers has a direct affect on instruction length.
- Stack machines use one - and zero-operand instructions.
- LOAD and STORE instructions require a single memory address operand.
- Other instructions use operands from the stack implicitly.
- PUSH and POP operations involve only the stack′s top element.
- Binary instructions (e.g., ADD, MULT) use the top two items on the stack.
- Stack architectures require us to think about arithmetic expressions a little differently.
- We are accustomed to writing expressions using infix notation, such as:

  Z = X + Y
- Stack arithmetic requires that we use postfix notation:

  Z = XY+
- This is also called Reverse Polish Notation (RPN), (somewhat) in honor of its Polish inventor, Jan Lukasiewicz (1878 - 1956).
- The principal advantage of postfix notation is that parentheses are not used.

- For example, the infix expression, Z = (X × Y) + (W × U) becomes:

  Z = X Y × W U × + in postfix notation.
- 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
STORE Z
```

- Note: The result of a binary operation is implicitly stored on the top of the stack!.
- 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
```

- 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.
- With a three-address ISA, (e.g., mainframes), the infix expression.
  $Z = X \times Y + W \times U$ might look like this:

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

- 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.