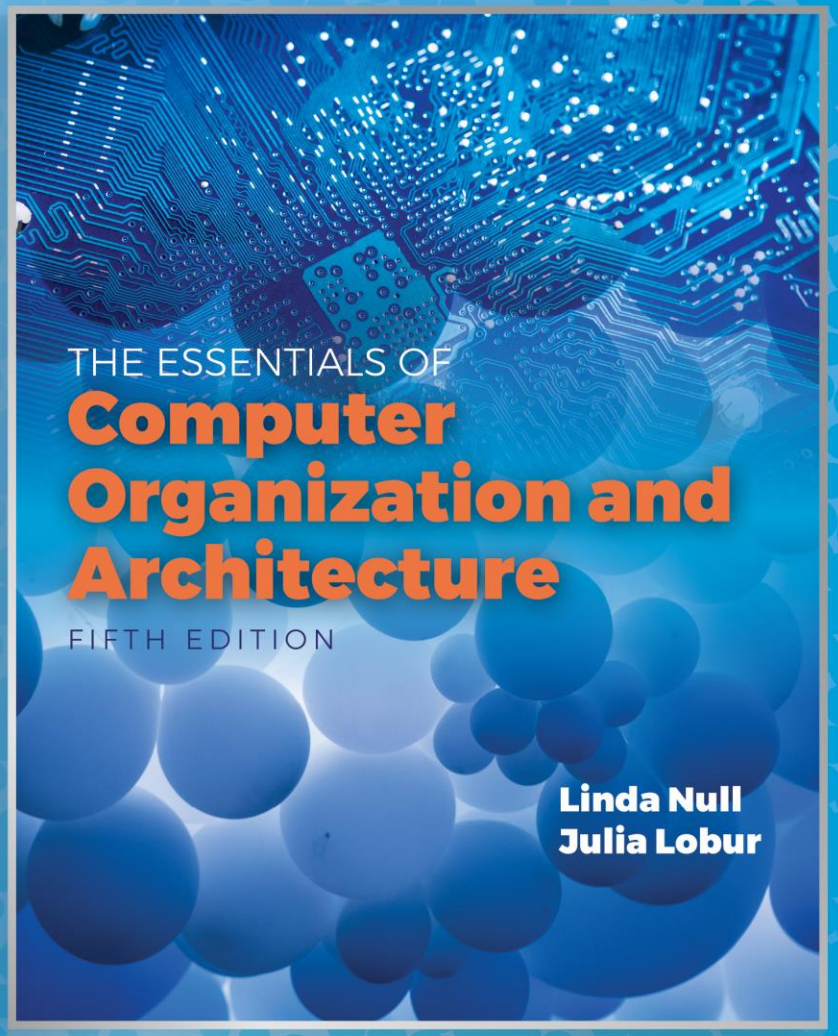
This is the second lecture of Chapter 5

Chapter 5

A Closer Look at Instruction Set Architectures



Quick review of last lecture

- Talk very briefly on chapter 4 topics
- Instruction formats
  - Instruction length (size)
  - Number of operands per instruction
  - Operand location
- Memory organizations
  - Byte or word addressable
  - Bib endian or little endian
- How the CPU will store data
  - A stack architecture
  - An accumulator architecture
  - A general purpose register architecture
- Stack machine and postfix notation

5.2 Instruction Formats (22 of 31)

- Let's see how to evaluate an infix expression using different instruction formats.
- 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
```

Redo it using load-store architecture

- With a three-address and load-store ISA (e.g., MIPS), the infix expression,

  Z = X × Y + W × U

- Might look like this:

```
LOAD   R1, X
LOAD   R2, Y
MULT   R3, R1, R2
LOAD   R1, W
LOAD   R2, U
MULT   R4, R1, R2
ADD    R5, R3, R4
STORE  Z, R5
```

5.2 Instruction Formats (23 of 31)

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

$$Z = X \times Y + W \times U$$

- Might look like this:

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

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

5.2 Instruction Formats (24 of 31)

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

  Z = X × Y + W × U

- Might looks like this:

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

5.2 Instruction Formats (25 of 31)

- In a stack ISA, the postfix expression,

  Z = X Y × W U × +

- might look like this:

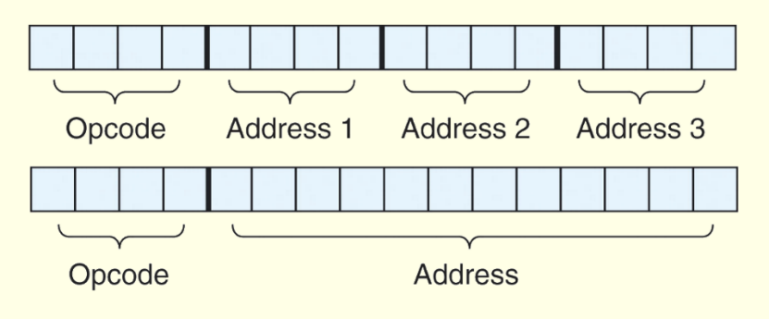
```
PUSH X
PUSH Y
MULT
PUSH W
PUSH U
MULT
ADD
POP Z
```

5.2 Instruction Formats (26 of 31)

- 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.

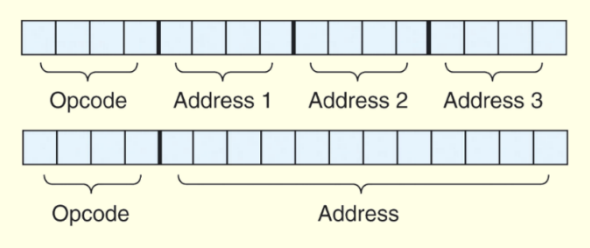
5.2 Instruction Formats (27 of 31)

- 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 (28 of 31)

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




```
0000 R1   R2   R3
...                          } 15 three-address codes
1110 R1   R2   R3
1111 - escape opcode

1111 0000 R1   R2
...                          } 14 two-address codes
1111 1101 R1   R2
1111 111  - escape opcode

1111 1110 0000 R1
...                          } 31 one-address codes
1111 1111 1110 R1
1111 1111 1111 - escape opcode

1111 1111 1111 0000
...                          } 16 zero-address codes
1111 1111 1111 1111
```

5.2 Instruction Formats (29 of 31)

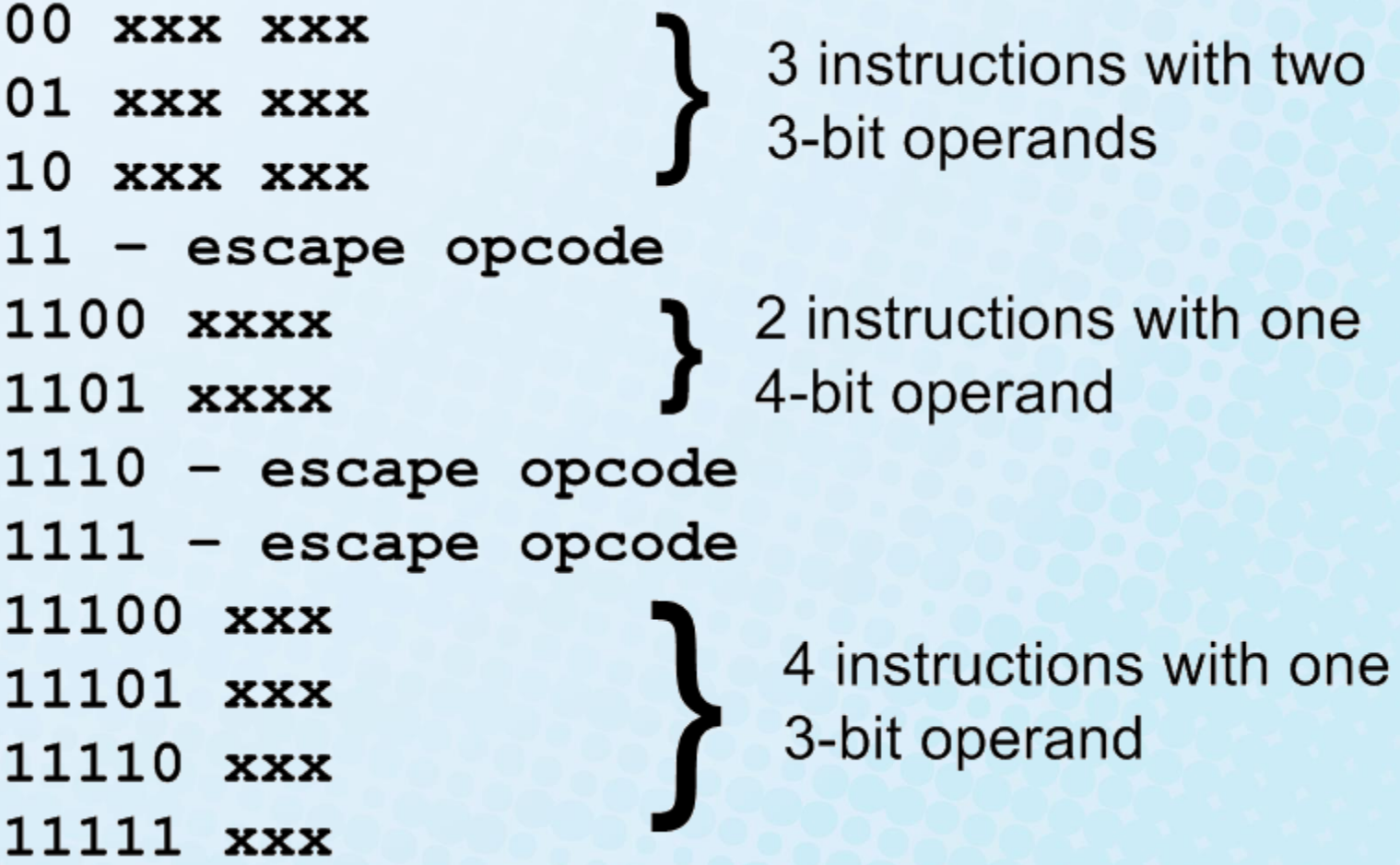
- Example: Given 8-bit instructions, is it possible to allow the following to be encoded?
  - 3 instructions with two 3-bit operands
  - 2 instructions with one 4-bit operand
  - 4 instructions with one 3-bit operand
- We need:
  - $3 \times 2^3 \times 2^3 = 192$ bits for the 3-bit operands
  - $2 \times 2^4 = 32$ bits for the 4-bit operands
  - $4 \times 2^3 = 32$ bits for the 3-bit operands
- Total: 256 bits.

5.2 Instruction Formats (30 of 31)

- With a total of 256 bits required, we can exactly encode our instruction set in 8 bits!
- We need:
  - $3 \times 2^3 \times 2^3 = 192$ bits for the 3-bit operands
  - $2 \times 2^4 = 32$ bits for the 4-bit operands
  - $4 \times 2^3 = 32$ bits for the 3-bit operands
- Total: 256 bits.

One such encoding is shown on the next slide.

5.2 Instruction Formats (31 of 31)



5.3 Instruction Types

- Instructions fall into several broad categories that you should be familiar with:
  - Data movement.
  - Arithmetic.
  - Boolean.
  - Bit manipulation.
  - I/O.
  - Control transfer.
  - Special purpose.

Can you think of some examples of each of these?