Topics for today:

A little more on stacks

Expanding opcodes

Addressing modes

Instruction Formats

- Design Decision: How will the CPU 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 simplicity (and cost) of hardware design with execution speed and ease of use.

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 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.

3

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 available registers has a direct affect on instruction length.

4

Recall: Stacks

A *stack* is a data structure in which items may be accessed or added only at one end, called the *top*.

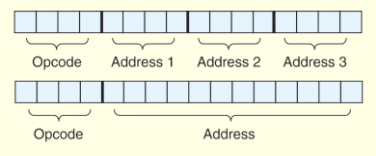
Zero-address instructions implicitly use a stack.

Expanding opcodes

An instruction set in which the opcode is of variable length is said to have an *expanding opcode*.

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:



7

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 three-address codes
1110 R1   R2   R3
1111 - escape opcode
1111 0000 R1   R2
...                          } 14 two-address codes
1111 1101 R1   R2
1111 1110 - 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
```

8

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.

9

Example (cont'd)

- 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.

10

Using escape opcodes

```
00 xxx xxx
01 xxx xxx
10 xxx xxx
11 – escape opcode
1100 xxxx
1101 xxxx
1110 – escape opcode
1111 – escape opcode
11100 xxx
11101 xxx
11110 xxx
11111 xxx
```

} 3 instructions with two 3-bit operands

} 2 instructions with one 4-bit operand

} 4 instructions with one 3-bit operand

11