Chapter 6
Operand Addressing And Instruction Representation
Topics

- Introduction
- How Many Operands Are Needed In An Instruction?
- One Operand Per Instruction
- Two Operands Per Instruction
- Three Operands Per Instruction
- Operand Values
Topics

- Explicit And Implicit Operand Encoding
- Implicit Operand Encoding
- Explicit Operand Encoding
- Operands That Combine Multiple Values
- Register Offset
- Tradeoffs In The Choice Of Operands
- Addressing

CS250 -- Part II

Dr. Rajesh Subramanyan, 2005
• Previous chapter, we have seen

- processor types and instruction sets

- details of instruction representation

- ways to specify operands

In this chapter,

Introduction
How Many Operands Are Needed In An Instruction?

• Does it depend on the operation being performed, example add would need two, not would need one

• Problem with arbitrary number of operands

• Processor will run slower

• More fetching and decoding time

How Many Operands Are Needed In An Instruction?
How Many Operands Are Needed In An Instruction?

- Problem with many operands per instruction
  - Longer time to fetch and decode if done with single set of hardware
  - If parallel hardware used, more hardware on chip
  - More space needed on chip
  - More power consumed
  - Higher cost

What is the smallest number of operands that can be useful for general computation?
One Operand Per Instruction

- Is a single operand per instruction sufficient? How would `add` be performed?

- Special register called accumulator holds default value.
  - Extra load accumulator, move from accumulator
  - More programming instruction for the same operation.

- Less hardware, faster to decode Disadvantage
  - Less hardware, faster to decode Disadvantage

Advantage

- Example `add X => accumulator <- accumulator + X`
- Load accumulator, move from accumulator
Two Operands Per Instruction

- More flexible than using accumulator
- mulh

Common to have instructions to specify two values, e.g. add,
Three Operands Per Instruction

• Third operand can specify destination

• Example \( \text{add } R1 \ R2 \ R3, \text{ where } R3 = R1 + R2 \)

• Advantage
  
  – Program needs less instructions in than one or two operand instructions

• Disadvantage
  
  – More operands means more hardware, decoding, fewer bits for opcode or larger instruction length
An operand could denote a memory location.
- register number
- constant

Operands Values
In a 3-address processor, operands can denote the above:

- A memory location
- A pair of contiguous registers
- A single register
- Operands that specify a destination
- The value in a memory location
- The contents of a register
- An unsigned constant
- An signed constant
- Operands that specify a source

Example
Explicit and Implicit Operand Encoding

• How should operands be represented in an instruction

• Possibilities are immediate, register, and memory location

• Two possibilities

Explicit operand encoding

Implicit operand encoding

Explicit and Implicit Operand Encoding
**Implicit Operand Encoding**

Example of addition instructions for a processor that uses implicit operand encoding:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operands</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr memory</td>
<td>R1 + memory[M]</td>
<td>R1, M</td>
</tr>
<tr>
<td>Addr immediate unsigned</td>
<td>R1 + I</td>
<td>R1, UI</td>
</tr>
<tr>
<td>Addr immediate signed</td>
<td>R1 + I</td>
<td>R1, I</td>
</tr>
</tbody>
</table>

Multiple opcodes for each operation

- Each opcode corresponds to a unique combination of operands
- Each opcode corresponds to a unique combination of operands
- Multiple opcodes for each operation
- List of opcodes can become large
- Disadvantage of opcodes

---

**Example of addition instructions for a processor that uses implicit operand encoding:**

- Addr memory: R1 + memory[M] → R1, M
- Addr immediate unsigned: R1 + I → R1, UI
- Addr immediate signed: R1 + I → R1, I
- Addr register: R1 + R2 → R1, R2

---

**Implicit Operand Encoding**
explicit operand encoding

Example of operands in an architecture that uses explicit operand encoding:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Register 1</th>
<th>Register 2</th>
<th>Operand 1</th>
<th>Operand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>signed int</td>
<td>register</td>
<td>-93</td>
<td>register 2</td>
</tr>
<tr>
<td>op</td>
<td>operand</td>
<td>operand</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Fewer bits available for operand value
- Instruction set remains small
- Instruction provided by dividing operand held

Advantage
- Interpreted

Disadvantage
- Instruction set remains small
- Fewer bits available for operand value

Explicit Operand Encoding
Operands That Combine Multiple Values

- Combining operand values to obtain true value
denotes either a constant, register, or memory
So far we have interpreted operand value as a number that

Operands That Combine Multiple Values
Explicit encoding with operand subdivided into three fields, a register, an offset, and a field denoting operand type.

Example of an instruction in which each operand consists of a register plus an offset.

```
add 
opcode operand 1
    operand 2
    register 2
    offset 4
```

Add register and offset to get true value of operand. Add register, an offset and field denoting operand type.

Register Offset
Tradeoffs In The Choice Of Operands

- Ease of programming
  - Complex forms of operands make programming easier
- Fewer instructions
  - Increasing expressive power of operands reduces
  - Each instruction is larger
  - Instructions in a program
  - However goals below have to be sacrificed in a tradeoff
  - Register-offset, 3 operands per instruction
  - Complex forms of operands make programming easier
  - Ease of programming
Tradeoffs In The Choice Of Operands

- Smaller instructions
  - Limiting operands, set of operand types, maximum size of operands
  - Less hardware requirements
  - Decreases instructions in a program
  - Limiting operands, set of operand types, maximum size of operands

- Larger range of immediate values
  - Larger field allows larger values to be stored, but larger instruction
Tradeoffs In The Choice Of Operands

- Faster operand fetch and decode
  - Fewer operands allows hardware to operate faster
- Decreased hardware size
  - Limiting type and complexity of operand reduces size of circuits
- Faster operand fetch and decode
Processor architects have created a variety of operand styles. No single form is optimal for all processors because the choice represents a compromise among functionality, program size, performance, and ease of programming. Hardware required to fetch values, and ease of programming has created a variety of operand styles.
Addressing

• Immediate
  - operand value is a constant, actual value stored in register

• Direct
  - operand value is a constant

• Immediate

Addressing
Addressing

- Indirect:
  - Operand is a register, register value is a memory address, memory address has the actual value. This is a single level of indirection.
  - Example of two level indirection:
    - Obtain M, value of operand
    - Interpre M as a memory address, and fetch the value
    - Interpre A as another memory address, and fetch the value

- Indirect Indirect:
  - Operand value is a register, register value is a memory address, memory address has the actual value. This is a single level of indirection.

Addressing