Processors

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
- Opearand 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

Introduction

- Previous chapter, we have seen
  - processor types and instruction sets
- In this chapter,
  - details of instruction representation
  - ways to specify operands

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 arbitary number of operands
  - more fetching and decoding time
  - processor will run slower

How Many Operands Are Needed In An Instructior

- 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
- Example add X => accumulator <- accumulator + X
- Advantage
  - less hardware, faster to decode Disadvantage
  - more programming instruction for the same operation, extra load accumulator, move from accumulator instructions

Two Operands Per Instruction

- Common to have instructions to specify two values, e.g. *add*, *mult*
- More flexible than using accumulator

Three Operands Per Instruction

- Third operand can specify destination
- Example *add R1 R2 R3, 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

Operand Values

- An operand could denote a
  - constant
  - register number
  - memory location.

Example

Operand that specifies a source

- **A signed constant**
- **An unsigned constant**
- **The contents of a register**
- **The value in a memory location**

Operand that specifies a destination

- **A single register**
- **A pair of contiguous registers**
- **A memory location**

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

Explicit And Implicit Operand Encoding

- How should operands be represented in an instruction
- Possibilities are
  - immediate
  - register
  - memory location
- Two possibilities
  - implicit operand encoding
  - explicit operand encoding

Implicit Operand Encoding

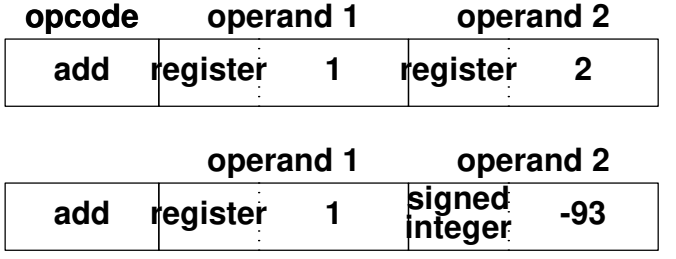
- Multiple opcodes for each operation
- Each opcode corresponds to a unique combination of operands
- Disadvantage
  - list of opcodes can become large

Opcode	Operands	Meaning
Add register	**R1 R2**	**R1 ← R1 + R2**
Add immediate signed	**R1 I**	**R1 ← R1 + I**
Add immediate unsigned	**R1 UI**	**R1 ← R1 + UI**
Add memory	**R1 M**	**R1 ← R1 + memory[M]**

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

Explicit Operand Encoding

- Additional information provided by dividing operand field and reserving some bits to specify how operand value is to be interpreted
- Advantage
  - instruction set remains small
- Disadvantage
  - fewer bits available for operand value



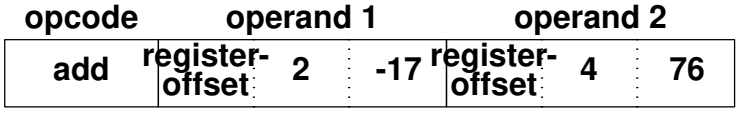
- Example of operands in an architecture that uses explicit operand encoding

Operands That Combine Multiple Values

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

Register Offset

- Explicit encoding with operand subdivided into three fields, a register, an offset and field denoting operand type
- Add register and offset to get true value of operand



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

Tradeoffs In The Choice Of Operands

- Ease of programming
  - complex forms of operands make programming easier, eg register-offset, 3 operands per instruction
  - however goals below have to be sacrificed in a tradeoff
- Fewer instructions
  - increasing expressive power of operands reduces instructions in a program
  - each instruction is larger

Tradeoffs In The Choice Of Operands

- Smaller instructions
  - limiting operands, set of operand types, maximum size of operands
  - less hardware requirements
  - increases instructions in a program
- 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

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, hardware required to fetch values, performance, and ease of programming

Addressing

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

Addressing

- 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.
  - Example of two level indirection
    * obtain M, value of operand
    * interpret M as a memory address, and fetch the value A from memory
    * interpret A as another memory address, and fetch the operand from memory