Processors

Chapter 5

Processor Types And Instruction Sets

Topics

- Introduction
- Mathematical power, convenience, and cost
- Instruction set and representation
- Opcodes, Operands and results
- Typical instruction format
- Variable-length vs fixed-length instructions
- General-purpose registers
- Floating point registers and register identification
- Programming with registers

Topics

- Register banks
- Complex and reduced instructions sets
- RISC design and the execution popeline
- Pipelines and instruction stalls
- Other causes of pipeline stalls
- Consequence for programmers
- Types of operations
- An example instruction set
- Minimalistic instruction set

Topics

- The principle of orthogonality
- Programming using conditional branching
- Summary

Introduction

- In this chapter we discuss
  - the set of operations a processor can perform
  - the different architect approaches to instruction sets and tradeoffs

What Operations Should A Processor Offer And The Tradeoffs

- What operations should a processor offer? At least three views based on
  - mathematical power: powerful instructions
  - convenience: easy to program
  - cost: less hardware
- Should a processor offer many, or a few basic operations
  - Architect wants a small operation set to reduce hardware
  - Programmer wants more for convenience

What Operations Should A Processor Offer And The Tradeoffs

- *The set of operations a processor provides represents a tradeoff among the cost of the hardware, the convenience for a programmer, and engineering considerations such as power consumption.*

Instruction Set And Representation

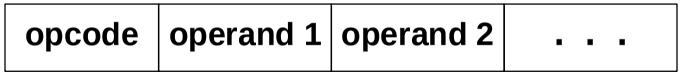
- An architect of a programmable processor decides
  - the set of operations supported by the hardware (instruction set)
  - the operation representation (instruction format)
- An instruction defi nition includes
  - an exact defi nition (opcode)
  - values (operands) and corresponding result
  - exceptions
- Instruction format
  - binary representation
  - defi nes hardware-software boundary

Opcodes and Operands

- Opcode
  - Unique number assigned to denote the operation.
- Operand
  - Values on which operation is performed.
  - The number and type of operand is specifi ed for each operation.

Typical Instruction Format

- Instruction
  - represented as a binary string
- Instruction fi elds
  - Opcode, followed by operand(s)



Hardware vs Software battle continues ..

- Variable-length vs. fi xed-length instructions
- Should one instruction be shorter than another ?
  - Programmer says variable-length optimizes memory
  - Architect says hardware is less complex for fi xed-length instruction

Variable-length vs. fixed-length instructions ..

- Hardware camp wins
- Fixed-length instruction set
  - Extra fi elds are left unused
  - Unused bits are ignored
  - Leaving bits unused is needed for hardware optimization

General purpose registers

- High-speed hardware device with fi xed size
- Temporary storage device
- Supports fetch and store operations
- Semantics same as memory

Floating point registers and register identification

- Separate registers to hold floating point values
- Numbering same as GP register.
- So which register is #X ?
  - Processor interprets from opcode if it needs to fetch operand from GP or FP register

Programming with registers

- Move operands to register
- Execute instruction
- Results may be stored in register
- Registers are few: optimize allocation.
- Double precision result. Storage?
  - Hardware: registers are consecutive; use 2
  - Programmers: plan accordingly.

Register Banks

- New complication to register allocation- register banks !
- What are they ?
  - Register group with separate physical access
- Why do we need them ?
  - Allows simultaneous access, e.g. one cycle to obtain both operands
- So what is the problem!
  - Programmers can’t permanently assign data to registers
  - Conflict occurs (both operands end up on the same bank); solution,
    * Reassign registers
    * Insert copy instruction

Register Banks

- Are banks absolutely necessary ?
  - Yes, performance !!
- Last word on registers
  - *Registers are expensive. Use judiciously for improving performance*

Complex and reduced instruction sets

- Instruction set categories
  - CISC
  - RISC
- CISC
  - Instruction set large
  - Instructions may do complex computation
- RISC
  - Instruction set minimized
  - Basic computations
  - Instruction execution constant: 1 clock cycle

Execution pipeline in RISC

- RISC designed to complete instruction/cycle.
- How?
  - Processor divides fetch-execute cycle into multiples steps
  - Hardware a multistage pipeline
  - Each stage completes one step/cycle, throughput 1 ins/cycle.

Pipelines & instruction stalls

- Advantage
  - Pipeline transparent to programmers
- Disadvantage
  - programmer can inadvertently introduce ineffi ciencies
  - e.g. Pipeline stalls

Causes of pipelines stalls

- Waiting for operands
- The processor
  - Accesses external storage
  - Invokes a coprocessor
  - Branches to new location
  - Calls a subroutine
- Hardware solution
  - Duplicate copies in pipeline for a branch, compute both results, discard one.

Consequences for programmers

- Avoid
  - Branch instructions
  - After result is computed, delay reference to the result register
- Separate references from computation

Types of operations

- Arithmetic
- Logical
- Data access and transfer
- Branch (conditional and unconditional)
- Floating point
- Processor control

MIPS processor instruction set

Instruction	Meaning
Arithmetic	
add	**integer addition**
subtract	**integer subtraction**
add immediate	**integer addition (register + constant)**
add unsigned	**unsigned integer addition**
subtract unsigned	**unsigned integer subtraction**
add immediate unsigned	**unsigned addition with a constant**
move from coprocessor	**access coprocessor register**
multiply	**integer multiplication**
multiply unsigned	**unsigned integer multiplication**
divide	**integer division**
divide unsigned	**unsigned integer division**
move from Hi	**access high-order register**
move from Lo	**access low-order register**
Logical (Boolean)	
and	**logical AND (two registers)**
or	**logical OR (two registers)**
and immediate	**AND of register and constant**
or immediate	**OR of register and constant**
shift left logical	**Shift register left N bits**
shift right logical	**Shift register right N bits**

Data Transfer

load word	**load register from memory**
store word	**store register into memory**
load upper immediate	**place constant in upper sixteen bits of register**
move from coproc. register	**obtain a value from a coprocessor**

Conditional Branch

branch equal	**branch if two registers equal**
branch not equal	**branch if two registers unequal**
set on less than	**compare two registers**
set less than immediate	**compare register and constant**
set less than unsigned	**compare unsigned registers**
set less than immediate	**compare unsigned register and constant**

Unconditional Branch

jump	**go to target address**
jump register	**go to address in register**
jump and link	**procedure call**

Floating point instructions defined by MIPS architectur

Instruction	Meaning
Arithmetic	
FP add	**floating point addition**
FP subtract	**floating point subtraction**
FP multiply	**floating point multiplication**
FP divide	**floating point division**
FP add double	**double-precision addition**
FP subtract double	**double-precision subtraction**
FP multiply double	**double-precision multiplication**
FP divide double	**double-precision division**
Data Transfer	
load word coprocessor	**load value into FP register**
store word coprocessor	**store FP register to memory**
Conditional Branch	
branch FP true	**branch if FP condition is true**
branch FP false	**branch if FP condition is false**
FP compare single	**compare two FP registers**
FP compare double	**compare two double precision values**

Minimalistic instruction set

- Design objective
  - Speed (1 ins/cycle)
  - Minimalistic (fewest # of instructions)
- Example of speed: fast access to 0. Design?
  - Reserve register 0 to contain 0 always

Principle of Orthogonality

- Each instruction should perform a unique task without duplicating or overlapping the functionality of other instructions
- Advantages
  - Understanding ease
  - Elegance

Conditional Branching

- Analogous to if-then-else

```
if (A > B) {
    Q;
    } else {
    R;
}
```

Conditional code mechanism to implement branching

```
sub A B          # compute A - B and set condition code
bnz Label1       # branch to Label1 if condition
jmp Label2       # jump to Label2
Label1:...code for R        # instructions that implement R go here
Label2:...       # program continues at this point
```

Wrapup

- Each processor defi nes an instruction set it supports
- Register help improve performance
- Instructions
  - RISC
    - Speed, simple, minimal instructions
  - CISC
    - Many, complex