Chapter 5

Processors Types And Instruction Sets

Chapter 5

Processors
Topics

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

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

- The principle of orthogonality
- Programming using conditional branching
- The principle of orthogonality
- Summary
In this chapter we discuss:

- the set of operations a processor can perform
- the different architectural approaches to instruction sets and tradeoffs

In this chapter we discuss...
What operations should a processor offer? At least three views based on:

- Mathematica: powerful mathematical 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

The tradeoffs
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 programmers, and engineering considerations such as power consumption.

·
Instruction Set And Representation

A na rchitect of a programmable processor decides
- the set of operations supported by the hardware (instruction set)
- the operation representation (instruction format)

A ni nstruction definition includes
- an exact definition (opcode)
- values (operands) and corresponding result
- exceptions

Instruction Format
- binary representation
- defines hardware-software boundary

Exceptions
- values (operands) and corresponding result
- an exact definition (opcode)
- an instruction definition includes
  - the operation representation (instruction set)
  - the set of operations supported by the hardware
  - an architect of a programmable processor decides

Instruction Set And Representation
Opcodes and Operands

- **Opcodes**
  
  Unique number assigned to denote the operation.

- **Operands**
  
  The number and type of operand is specified for each operation.
  
  Values on which operation is performed.

- **Opcodes and Operands**
Typical Instruction Format

- Instruction
  - represented as a binary string
  - fields:
    - Opcode, followed by operand(s)
    - Instruction
      - represented as a binary string

Typical Instruction Format
Hardware vs. Software battle continues...

- Variable-length vs. fixed-length instructions

  - Programmer says variable-length optimizes memory
  - Architect says hardware is less complex for fixed-length

Should one instruction be shorter than another?

- Variable-length vs. fixed-length instructions

...
Variable-length vs. fixed-length instructions

- Hardware camp wins
- Extra fields are left unused
- Extra fields are ignored

- Unused bits are ignored
- Unused bits are needed for hardware optimization

- Fixed-length instruction set
- Hardware camp wins
- Variable-length vs. fixed-length instructions
General purpose registers

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

Dr. Rajesh Subramanayan, 2005
Floating point registers and register identification

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

- Move operands to register
- Execute instruction
- Results may be stored in register
- Double precision result? Store in register?

Hardware: registers are consecutive; use 2
- Registers are few; optimize allocation.
- Programmers: plan accordingly.
Register Banks

New complication to register allocation - register banks

- Why are they needed?
- Register group with separate physical access

- What do we need them for?
- Insert copy instruction

- Reassign registers

- Solution
- Conflicts occur (both operands end up in the same bank)

- Programmers cannot permanently assign data to registers

- Why do we need them?
- Allows simultaneous access, e.g., one cycle to obtain both operands

- What is the problem?
- Allows simultaneous access, e.g., one cycle to obtain both operands

• New complication to register allocation - register banks

Register Banks
Register Banks

- Are banks absolutely necessary?
- Yes, performance!

Last word on registers

- Registers are expensive. Use judiciously for improving performance.

Are banks absolutely necessary?
Complex and Reduced Instruction Sets

- Instruction set large
  - Instructions may do complex computation
  - Instruction set large

- Instruction set minimized
  - Basic computations
  - Instruction execution constant: 1 clock cycle

- RISC
  - Instruction set minimized

- CISC
  - Instruction set large

- RISC
  - Instruction set categories

- CISC
  - Instruction set categories

Dr. Rajesh Subramaniam, 2005
Execution pipeline in RISC

- RISC designed to complete instruction/cycle.

  - Processor divides fetch-execute cycle into multiple steps.
  - Each stage completes one step/cycle, throughput 1 ins/cycle.
  - Hardware a multistage pipeline.

How?

- RISC designed to complete instruction/cycle.
Advantage

- Pipeline transparent to programmers

Disadvantage

- Programmer can inadvertently introduce inefficiencies
- Pipeline stalls

Pipelines & Instruction Stalls
Causes of pipelines stalls

• Waiting for operands

• Hardware solution
  - Duplicates copies in pipeline for a branch, compute both results, discard one.

• Calls a subroutine

• Branches to new location
  - Invokes a coprocessor

• Accesses external storage

• The processor

Causes of pipelines stalls
Consequences for Programmers

• Separate references from computation
  - After result is computed, delay reference to the result
  - Branch instructions
  - Avoid
Types of operations

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic</strong></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>integer addition</td>
</tr>
<tr>
<td>sub</td>
<td>integer subtraction</td>
</tr>
<tr>
<td>addu</td>
<td>unsigned integer addition</td>
</tr>
<tr>
<td>subu</td>
<td>unsigned integer subtraction</td>
</tr>
<tr>
<td><strong>Logical (Boolean)</strong></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>logical AND (two registers)</td>
</tr>
<tr>
<td>or</td>
<td>logical OR (two registers)</td>
</tr>
<tr>
<td>andi</td>
<td>AND of register and constant</td>
</tr>
<tr>
<td>ori</td>
<td>OR of register and constant</td>
</tr>
<tr>
<td><strong>Shift (Register)</strong></td>
<td></td>
</tr>
<tr>
<td>slt</td>
<td>less than</td>
</tr>
<tr>
<td>sltu</td>
<td>unsigned less than</td>
</tr>
<tr>
<td><strong>Immediate (Integers)</strong></td>
<td></td>
</tr>
<tr>
<td>jal</td>
<td>jump and link</td>
</tr>
<tr>
<td>jalr</td>
<td>jump and link register</td>
</tr>
<tr>
<td><strong>Move (Register)</strong></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>load word</td>
</tr>
<tr>
<td>sw</td>
<td>store word</td>
</tr>
<tr>
<td><strong>Move (Coprocessor)</strong></td>
<td></td>
</tr>
<tr>
<td>move from coprocessor</td>
<td></td>
</tr>
<tr>
<td>move to coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
**Data Transfer**

- Load Word from Memory
- Store Word into Memory
- Load Upper Immediate into Memory
- Load Register from Memory
- Load Word
- Load Upper Immediate
- Load Register

**Conditional Branch**

- Branch if Two Registers Equal
- Branch if Two Registers Unequal
- Compare Two Registers and Constant
- Compare Two Registers
- Compare Unsigned Registers
- Compare Unsigned Register and Constant
- Compare Immediate and Register
- Compare Immediate and Unsigned Register

**Unconditional Branch**

- Jump to Address
- Jump Register
- Jump and Link

---

Dr. Rajesh Subramanayan, 2005
## Floating point instructions defined by MIPS architecture

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic</strong></td>
<td></td>
</tr>
<tr>
<td>FP add</td>
<td>floating point addition</td>
</tr>
<tr>
<td>FP subtract</td>
<td>floating point subtraction</td>
</tr>
<tr>
<td>FP multiply</td>
<td>floating point multiplication</td>
</tr>
<tr>
<td>FP divide</td>
<td>floating point division</td>
</tr>
<tr>
<td>FP add double</td>
<td>double-precision addition</td>
</tr>
<tr>
<td>FP subtract double</td>
<td>double-precision subtraction</td>
</tr>
<tr>
<td>FP multiply double</td>
<td>double-precision multiplication</td>
</tr>
<tr>
<td>FP divide double</td>
<td>double-precision division</td>
</tr>
<tr>
<td><strong>Data Transfer</strong></td>
<td></td>
</tr>
<tr>
<td>load word coprocessor</td>
<td>load value into FP register</td>
</tr>
<tr>
<td>store word coprocessor</td>
<td>store FP register to memory</td>
</tr>
<tr>
<td><strong>Conditional Branch</strong></td>
<td></td>
</tr>
<tr>
<td>branch FP true</td>
<td>branch if FP condition is true</td>
</tr>
<tr>
<td>branch FP false</td>
<td>branch if FP condition is false</td>
</tr>
<tr>
<td>FP compare single</td>
<td>compare two FP registers</td>
</tr>
<tr>
<td>FP compare double</td>
<td>compare two double precision values</td>
</tr>
</tbody>
</table>
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

- Minimalistic (fewest # of instructions)
- Speed (1 ins/cycle)

Design objective

Minimalistic instruction set
Principle of Orthogonality

Each instruction should perform a unique task without duplicating or overlapping the functionality of other instructions.

Advantages

- Elegance
- Understanding ease

Each instruction should perform a unique task without duplicating or overlapping the functionality of other instructions.
Conditional Branching

- Analogous to if-then-else:

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

Analogous to if-then-else.
Conditional code mechanism to implement branching

```asm
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 56
here
Label2:...code for R # instructions that implement R 56

jmp Label2 # jump to Label2
bz Label1 # branch to Label1 if condition
sub A B # compute A - B and set condition code
```

Conditional code mechanism to implement branching.
Each processor defines an instruction set it supports

**Wrapup**

- Many complex
- *CISC*
  - Speed, simple, minimal instructions
- *RISC*
  - Instructions
  - Register help improve performance