Instruction Formats

op: operation of the instruction
rs: the first register source operand
rt: the second register source operand
rd: the register destination operand
shamt: shift amount
funct: function; variant of the operation in the op field

High-level Construct Support

Conditionals - if-then-else
Loops
Case statements - switch
Procedure calls
Stacks

Last-In-First-Out queue
Used to spill registers
Pointer to top of stack
Push and pop operations

Procedure Calls

Pass arguments to procedure
Allocate registers for program variables
Produce code for body of procedure
Preserve registers across procedure invocations

- Caller saves registers
- Callee saves registers
RISC Open Issues

Register usage

Setting of condition codes

Byte ordering

Hardware interlocks

Register Usage

Registers used in order to reduce memory traffic

Large % of memory traffic related to procedure calls

Larger number of registers implies more bits for op encoding
Solution: Overlapping register windows / Multiple register sets

- subset of registers visible at a time (say 32)
- current window pointer (CWP) points to active window
- circular buffer - advance CWP on procedure call
- register file overflow results in trap

Caches Versus Register Files

Caches hold instructions and data

No rigid split - adapts dynamically to usage pattern

Tag and valid bits add chip area overhead not used for data

Requires full memory address for access

Fast process switching but slower procedure calls
**SPARC Versus MIPS**

SPARC - Multiple register sets, versions of instructions that set condition codes, no mult/div instructions, big endian, hardware register interlocks for loads

MIPS - Single register set, big/little endian, mult/div instructions that take multiple cycles, no condition codes, no hardware interlocks, deeper pipeline stage

---

**Design Principles for RISC Machines**

1. Analyze the applications to find the key operations
2. Design a data path that is optimal for key operations
3. Design instructions that perform the key operations using the data path (registers+ALU)
4. Add new instructions only if they do not slow down the machine
5. Repeat this process for other resources
**RISC Versus CISC**

One instruction per data path cycle

Load/store architecture (only register addressing permitted for ordinary instructions)

Pipelining - one instruction issued each cycle

No microcode (hardwired for the common case) - reduced chip area

Increased memory usage

Fixed-format instructions

---

Reduced Instruction set

Put the complexity in the compiler - delayed loads, reduced addressing modes, optimize register usage

Multiple register sets