RISC - MIPS and SPARC

Machine Language

The **big** idea: Stored program computer

- Instructions are binary numbers
- Instructions stored in memory
- Programs are interpreted: fetch and execute in sequence (unless branch)
- Contrast with non-stored-program computer

All MIPS and SPARC instructions are 32 bits.

Data words and instructions are the same length.

Fields:

- names,
- length, (5 bits specify 32 registers)
- function

Field name	Op	rs	rt	rd	shamt	funct
Field width	6	5	5	5	5	6

Specifying the Operand

The different ways of specifying an operand's location are called addressing modes.

Addressing modes by operation type

- arithmetic and logical (Boolean)
  - register
  - immediate (constant)
- data transfer (load/store)
  - displacement, i.e., `register + constant`
  - index, i.e., `register + register` (SPARC only)

Instruction Format

- All MIPS instructions use 1 of 3 formats

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	immediate		
J	op	address				

- Arithmetic operations use R format
- `lw` and `sw` use the I format
- J format for control flow
- Processor uses the first field to recognize formats

Principle: Good design demands compromise

Example: Larger addresses or constants versus the complexity of supporting multiple instruction sizes.

Implementing Flow Control

Conditional branches

- condition codes – separate compare and branch

```
cmp     %o0,%o1
bne     L77004
```

- combined compare and branch

```
bne     $4,$5,L33       # MIPS
```

Unconditional branches – *jump*, e.g.,

```
j       L23             # MIPS
j       $31             # MIPS
```

`if` Statements

C fragment:

```
int h, i, j; ...

if (i == j) h = i + j;
```

MIPS assembly code:

```
        bne     $4,$5,L1    # i != j branch
        add     $3,$4,$5    # h = i + j
L1:
```

SPARC assembly code:

```
        cmp     %o0,%o1
        bne     L77004      ! i != j branch
        add     %o0,%o1,%o2 ! h = i + j
L77004:
```

C fragment:

```
int h, i, j; ...

if (i != j) h = i + j;
else        h = i - j;
```

MIPS assembly code:

```
        beq     $4,$5,L1
        add     $3,$4,$5
        j       L2
L1:     sub     $3,$4,$5
L2:
```

SPARC assembly code:

```
        cmp     %i0,%i1
        be      L77003
        add     %i0,%i1,%i0
        b       L77005
L77003: sub     %i0,%i1,%i0
L77005:
```

Loop Statements

C fragment:

```
int h, i, j; ...

do { h += i;
     i++;
} while (i != j);
```

MIPS assembly language

```
L1:     add     $6,$6,$4
        add     $4,$4,1
        bne     $4,$5,L1
```

SPARC assembly language:

```
L1:     add     %o2,%o0,%o2
        inc     %o0          ! add %o0,1,%o0
        cmp     %o0,%o1
        bne     L1
```

C fragment:

```
int h, i, j; ...

for (i = 1; i != 10; i++)
    h += i;
```

MIPS assembly language:

```
        li      $5, 1
        beq     $5,10,L2
L1:     add     $4,$4,$5
        add     $5,$5, 1
        bne     $5,10,L1
L2:
```

SPARC assembly language:

```
        mov     1,%i5           ! synthetic instr
        b       L2
L1:     add     %i0,%i5,%i0
        inc     %i5             ! synthetic instr
L2:     cmp     %i5,10
        bne     L1
```

Other Conditional Branches – MIPS

Use the `slt` instruction

C fragment:

```
int h, i, j; ...

if (i >= j) h = i + j;
```

MIPS assembly code:

```
        slt     $1,$4,$5 # set $1 if $4 < $5
        bne     $1,$0,L1
        add     $6,$4,$5
L1:
```

Can simulate all relational operators with `slt`, `bne`, and `beq`.

Other Conditional Branches – SPARC

A full set of `b`*icc* instructions

- `ba` (or simply `b`) "always", `bn` "never"
- `be` "equal", `bne` "not equal"
- `bg` "greater", `ble` "less or equal"
- `bge` "greater or equal", `bl` "less"

SPARC assembly code:

```
        cmp     %o0,%o1
        bl      L77004
        add     %o0,%o1,%o2
L77004:
```

Delay Slots

A (conditional) branch or (unconditional) jump doesn't take effect until the instruction after the branch or the jump has executed.

- The MIPS assembler hides this detail. It fills the delay slot with either a `nop` or an instruction from before or after the branch or jump in the assembly language program.
- The SPARC assembler requires the compiler/programmer to fill the delay slot.
  - You can't move the `cmp` instruction into the delay slot.

Example

C fragment:

```
int h, i, j;
        h++;
        if (i >= j) h = i + j;
```

Obvious fill:

```
        inc     %o0
        cmp     %o1,%o2
        bl      L77004
        nop
        add     %o1,%o2,%o0
L77004:
```

Clever fill:

```
        cmp     %o1,%o2
        bl      L77004
        inc     %o0
        add     %o1,%o2,%o0
L77004:
```

Loading Absolute/Immediate Values

Load Address - 0x40004000 into a register

Cannot be loaded in a single instruction (immediate value too large)

MIPS -

```
lui $5, 0x4000        # load higher order 16 bits
ori $5, $5, 0x4000   # or in the lower 16 bits in
```

SPARC -

```
sethi %hi(0x40004000), %o5   ! load high order 22 bits
or %o5, %lo(0x40004000), %o5 ! or in lower 10 bits
```