Instruction Usage

Usage of DLX Instructions By SPEC92 Integer Code

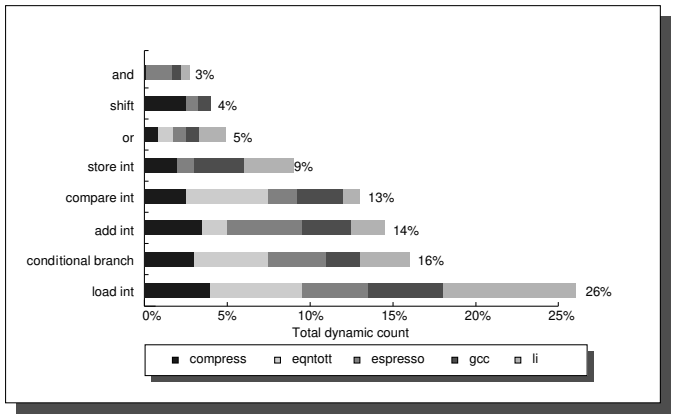

FIGURE 2.28 Graphical display of instructions executed of the five programs from SPECint92 in Figure 2.26.

Instruction Usage

Usage of DLX Instructions By SPEC92 Floating-Point Code

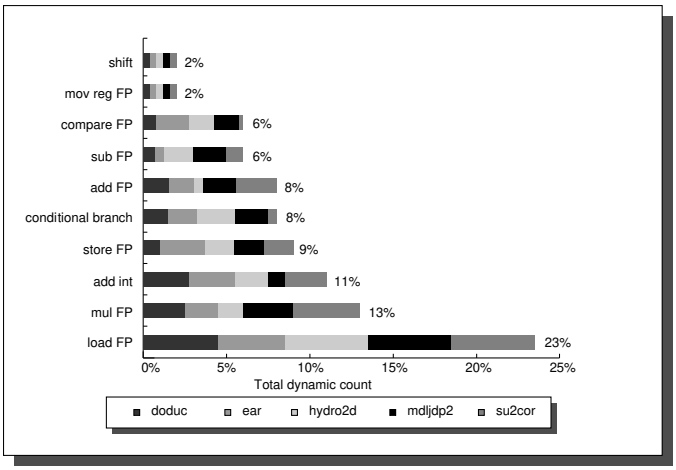

FIGURE 2.29 Graphical display of instructions executed of the five programs from SPECfp92 in Figure 2.27.

Control Transfer (Flow) Instructions

Control transfer instructions (CTIs) ...
... may cause next instruction to be fetched from ...
... somewhere other than PC + 4 (assuming 4-byte instructions).

(Called control flow instructions in book.)

Names used for CTIs vary by architecture.

Names used below are common, if not standard.

- *Branch*:
  Conditional control transfer.
- *Jump*:
  Unconditional control transfer.
  Sometimes special case of branch.
- *Jump and Link*:
  Unconditional, return address (PC) saved in register.
- *Call*:
  Unconditional control transfer, PC, etc. saved.
  Sometimes special case of jump and link.
- *Return*:
  Unconditional, PC, etc. from most recent call restored.
  Sometimes special case of jump and link.

CTI Examples

C Code

```
if( r2 == r3 ) goto TARGET1;
r4 = r5 + r6;
goto TARGET2;
TARGET1:
r4 = r5 * r6;
TARGET2:
PROCNAME();
return;
```

DLX Assembler Code

```
sub  r1, r2, r3    ! r1 = r2 - r3
beqz r1, TARGET1   ! Branch if r1 = 0.
add  r4, r5, r6    ! r4 = r5 + r6
j    TARGET2       ! Jump unconditionally.
TARGET1:           ! Label. (Assembler and linker find address.)
mult r4, r5, r6    ! r4 = r5 * r6
TARGET2:           ! Another label.
sw   16(r20),r31   ! Save return address (of caller to this proc.)
jal  PROCNAME      ! Call procedure PROCNAME.
lw   r31,16(r20)   ! Restore return address (overwritten by jal).
jalr r31           ! Return (from this procedure).
```

Branch instruction: `beqz`.

Jump instruction: `j`.

Call instruction: `jal`.

Return instruction: `jalr`.

Note: call and return are sometimes special case of *jump and link* instructions.

Usage of CTI By Type

CTI by type MIPS running SPEC92 benchmarks.

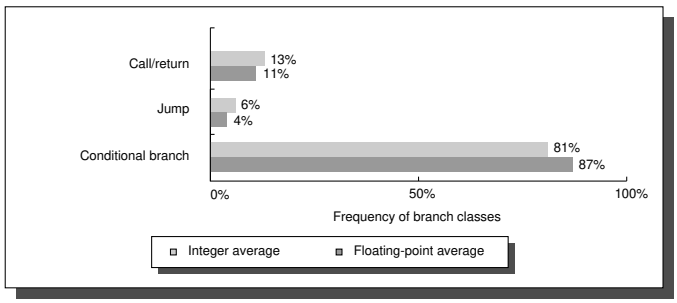

FIGURE 2.12 Breakdown of control flow instructions into three classes: calls or returns, jumps, and conditional branches.

Destination Address in CTIs

Any addressing mode *could* be used for destination.

Several are common:

- *Absolute*
  Destination address is an immediate.

  Best for procedure calls...
  ...because destination can be far away.

- *PC-Relative* or *Displacement*
  Destination is *displacement* added to program counter.

  ISA design choice: displacement size.

  Good for conditional branches...
  ...because destination usually close by...
  ...and so small immediates suffice.

- *Register Indirect*
  Destination in register.

  Used for subroutine return addresses.

  Also used by ISAs in which immediates smaller than addresses.

- *Indexed*
  Destination is sum of two registers.

  Useful for C `switch` and similar statements.

Destination Address Examples

Absolute and PC-Relative Examples

```
 jump_abs TARGET1    ! Instruction contains entire address, TARGET1.
 add r1, r2, r3
TARGET1:             ! When jump_rel executes, PC = TARGET1.
 jump_rel TARGET2    ! Instruction contains TARGET2-TARGET1.
 sub r4, r5, r6
TARGET2:
 and r7, r8, r9
```

To compute target address of `jump_rel` processor adds operand to PC.

Names `jump_abs` and `jump_rel` are made up.

In DLX the following CTIs use PC-relative addresses: `beqz` `bneq`, `j`, `jal`, `bfpt`, and `bfpf`

In DLX there are no CTIs that use absolute addressing.

Destination Address Examples

Jump and link SPARC instruction.

First two operands specify address, a register plus immediate.

Last operand is register to save PC in.

Register `g0` is a dummy register.

```
 jmpl %l0+0,%o7      ! Call address in %l0, PC saved in %o7
 nop
 ...

SUBROUTINE:
 add %l1, %l2, %l3   ! l3 = l2 + l1 (Note that l3 is desto.)
 jmpl %o7+0, %g0     ! Return using address in o7. Save PC in dummy reg.
 nop
```

Branch Displacement Size

04-9

Branch distances on DLX running SPEC92 programs.

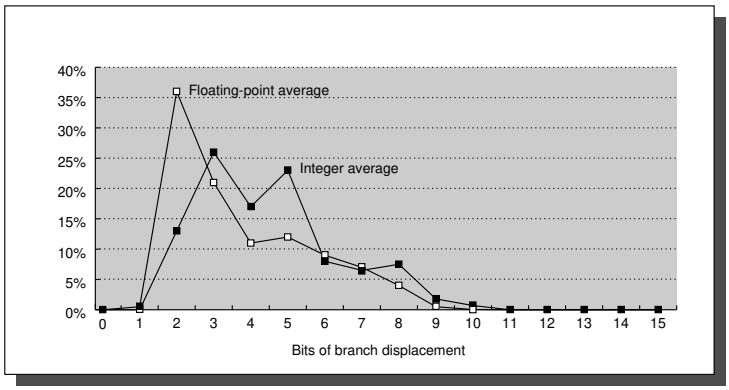

FIGURE 2.13 Branch distances in terms of number of instructions between the target and the branch instruction.

In DLX branch displacement limited to 16 bits.

Branch Conditions

04-10

How branch condition might be specified:

- Test value of general-purpose register (GPR).
- Test value of special-purpose *condition code register* (CCR).
- Comparison specified in branch instruction.

Note: *test* value means *test if value is zero ...*
*...*which is much faster than *test if value greater than constant*.

A typical ISA would use one or two of these methods.

Branch Condition Example (DLX)

04-11

DLX uses two methods: GPR and CCR:

Integer compare instructions (*e.g.*, `slt`) write result comparison into any GPR ...

... where it is read by a branch.

Floating-point compare instructions (*e.g.*, `gtd`) write result into special *FP status* (condition-code) *register*.

```
 slt  r1, r2, r3  ! Compare r2 and r3, set r1=1 if r2 < r3.
 beqz r1, TARGET1 ! Branch if r1 = 0. (r2 >= r3).
 gtd  f0, f2      ! Compare f0 and f2, set FP CCR true if f0 > f2
 bfpt TARGET1     ! Branch if FP CCR is true.
 ....
TARGET1:
```

DLX Mnemonics:

`slt`: Set (destination register if op1) less than (op2).

`beqz`: Branch if (op1) equal to zero.

`gtd`: Greater-than double: Set FP CCR if op1 > op2.

Branch Condition Example (Sparc)

04-12

Sparc uses an *integer condition codes* register (actually part of a larger status register).

ALU instructions can optionally set condition codes.

Register has several 1-bit fields which describe outcome: *negative*, *zero*, *overflow*, and *carry*.

Branch instruction can test for many combinations (*e.g.*, not negative and not zero, not negative, negative).

```
subcc %l1, %l2, %g0  ! %g0 = %l1 - %l2, discard difference but set codes.
sub   %l3, %l4, %l5  ! %l5 = %l3 - %l4. Doesn't set codes.
bg    TARGET         ! Branch if subcc result pos.. (g in bg is for >0.)
add   %l6, %l7, %l8  ! Branch delay slot: add always executed.
 ....
TARGET:
```

Tradeoffs between Methods to Specify Branch Condition

Factors

For compact code and programmer convenience:

⇒Comparison in branch instruction.

For fast implementation:

⇒Test GPR. (But may "waste" registers.)

⇒Test CCR. (Maybe limited to one condition at a time.)

(Faster because comparison made before branch instruction.)

Procedure Call and Return

Procedures (A.k.a., subroutines, functions.)

Fundamental part of every nontrivial program.

Requires careful support in ISA.

Mandatory ISA Support

Call instruction saves PC in register.

Return restores saved PC.

Additional Support, Provided by ISA or Software (*ABI*).

Save and restore registers.

Prepare *stack frame* of called procedure.

Application Binary Interface (ABI)

Rules for writing machine-language programs.

More restrictive than ISA ...
... but not enforced by hardware.

Code adhering to ABI rules called *compliant*.

Given an ABI, ...
... any compliant procedure ...
... can call any other compliant procedure ...
... (if call parameter and return value types match).

⇒ABI determines how "Additional Support" provided.

Implementation of Call and Return Steps

Mostly Hardware

Powerful call and return instructions do most of the work.

Call instruction ...
... saves program counter and other registers.

Return instruction ...
... adjusts stack and restores registers.

Mostly Software

Simple call and return only handle program counter.

Remainder done by general-purpose instructions ...
... using ABI guidelines.

Before call, using general-purpose instructions, ...
... procedure may save some registers.

Call instruction ...
... places return address in an ABI-specified register.

Called procedure, using general-purpose instructions, ...
... adjusts stack and may save registers.

Procedure return is similar.

CTI Variations

CTI Behaviors Chosen to Speed Implementation

- *Delayed Transfer*
  Control transfer occurs $d > 1$ instructions after CTI.

  E.g., consider execution of instruction 1 of DLX code:

```
1  j TARGET        ! Jump to TARGET.
2  add R1,R1,R1
3  add R2,R2,R2
4  add R3,R3,R3
5  add R4,R4,R4
...
TARGET:
```

  Normally, instruction 2 not executed.

  When $d = 2$ instruction 2 is executed, but not 3, 4, and 5.

  When $d = 3$ instruction 2 and 3 are executed, but not 4 and 5.

- Branch Instructions with Prediction Hints
  Programmer indicates whether branch is likely.

  If programmer correct, execution may be faster.

- Predicated Execution
  Non-CTI instructions that only execute if some condition true.

  E.g., `movg r1,r2`, meaning ...
  ... move `r1` to `r2` if greater-than condition true. (SPARC V9).