Instruction Usage
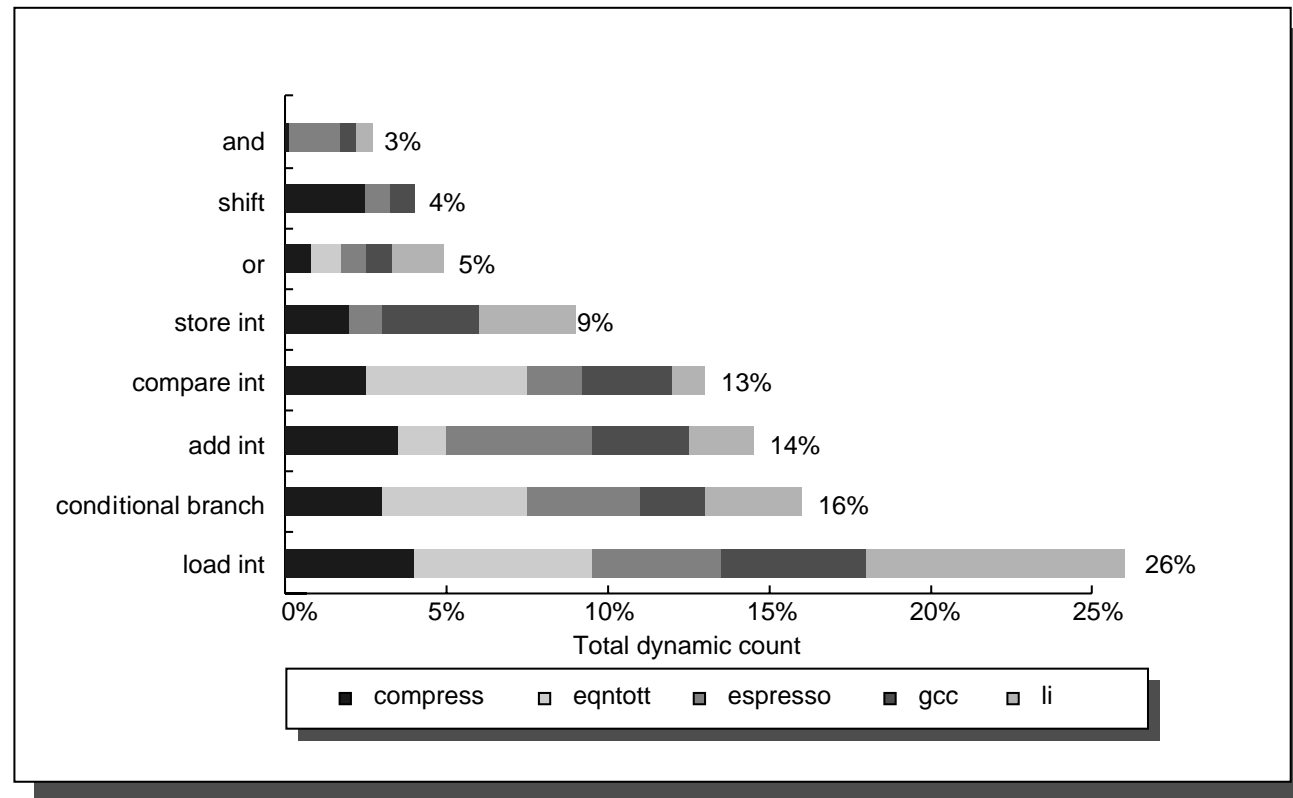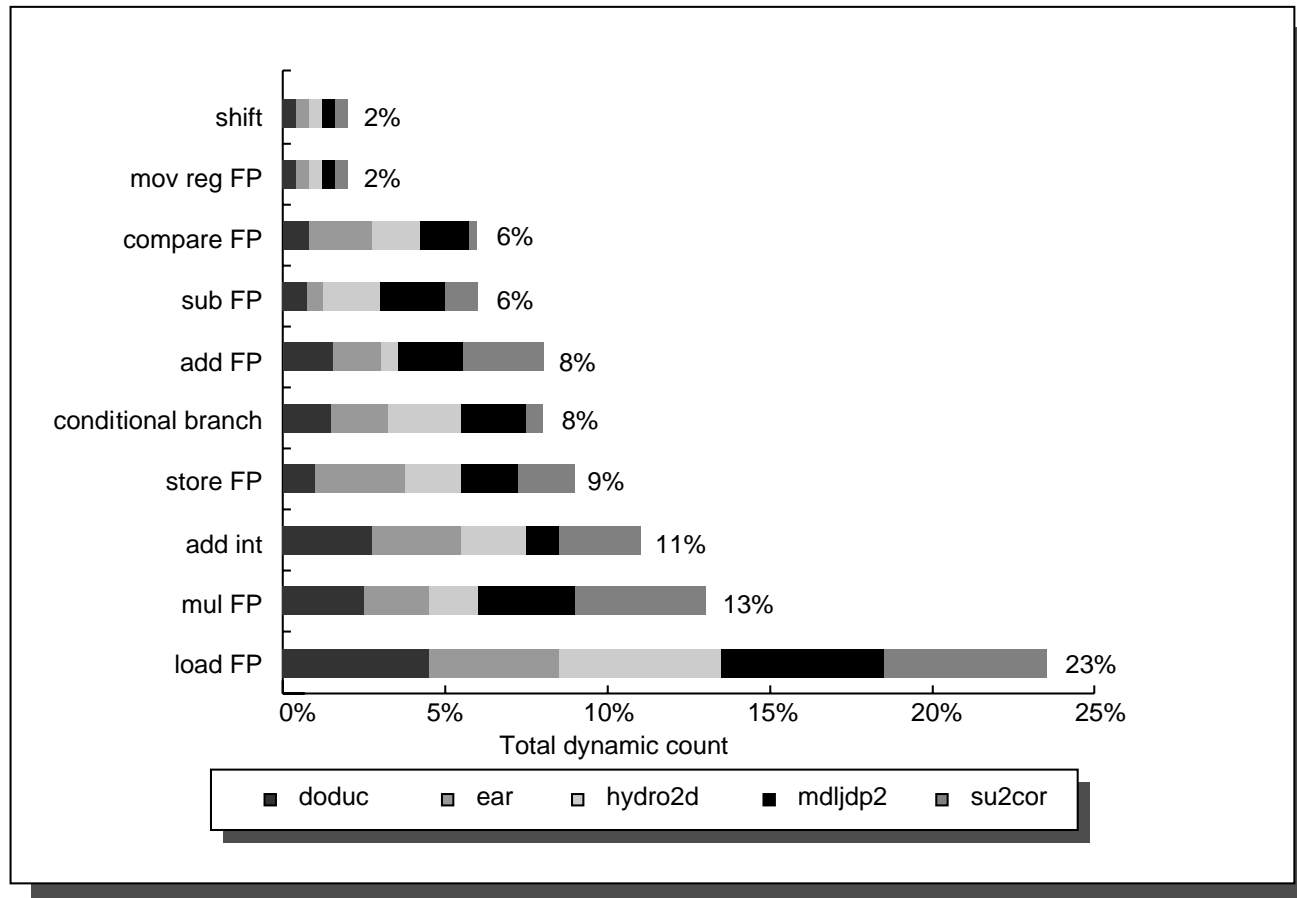
Usage of DLX Instructions By SPEC92 Integer Code

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

```
beq  r1,$0 SKIP  ! If r1 = 0 goto SKIP
nop
add  r2, r2, r5
SKIP:
add  r2, r3, r4
```

Control transfer (control flow) instructions (CTIs) . . .

. . . may cause next instruction to be fetched from . . .

. . . somewhere other than $PC + 4$ (assuming 4-byte instructions).

Programs would be trivial without them.

Occur frequently in programs.

Often a performance bottleneck.

Names used below are common:

- *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.

C Code   CTI Examples

```
          if( t2 == t3 ) goto TARGET1;
          t4 = t5 + t6;  goto TARGET2;
TARGET1:      t4 = t5 * t6;
TARGET2:      PROCNAME();    return;
```

MIPS Assembler Code

```
 sub  $t1, $t2, $t3     ! $t1 = $t2 - $t3
 beq  $t1, $0 TARGET1  ! Branch if $t1 = 0.
 nop
 add  $t4, $t5, $t6   ! $t4 = $t5 + $t6
 j    TARGET2             ! Jump unconditionally.
 nop
TARGET1:                 ! Label. (Assembler and linker find address.)
 mult $t5, $t6            ! {hi,lo} = $t5 * $t6
 mflo $t4
TARGET2:                 ! Another label.
 sw   16($t7), $ra    ! Save return address (of caller to this proc.)
 jal  PROCNAME         ! Call PROCNAME, store this return addr in $ra.
 nop
 lw   $ra, 16($t7)     ! Restore return address (overwritten by jal).
 jr $ra                  ! Return (from this procedure).
 nop
```

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

```
 sub  $t1, $t2, $t3    ! $t1 = $t2 - $t3
 beq  $t1, $0 TARGET1  ! Branch if $t1 = 0.
 nop
 add  $t4, $t5, $t6    ! $t4 = $t5 + $t6
 j    TARGET2          ! Jump unconditionally.
 nop
TARGET1:               ! Label. (Assembler and linker find address.)
 mult $t5, $t6         ! {hi,lo} = $t5 * $t6
 mflo $t4
TARGET2:               ! Another label.
 sw   16($t7), $ra     ! Save return address (of caller to this proc.)
 jal  PROCNAME         ! Call PROCNAME, store this return addr in $ra.
 nop
 lw   $ra, 16($t7)     ! Restore return address (overwritten by jal).
 jr $ra                ! Return (from this procedure).
 nop
```
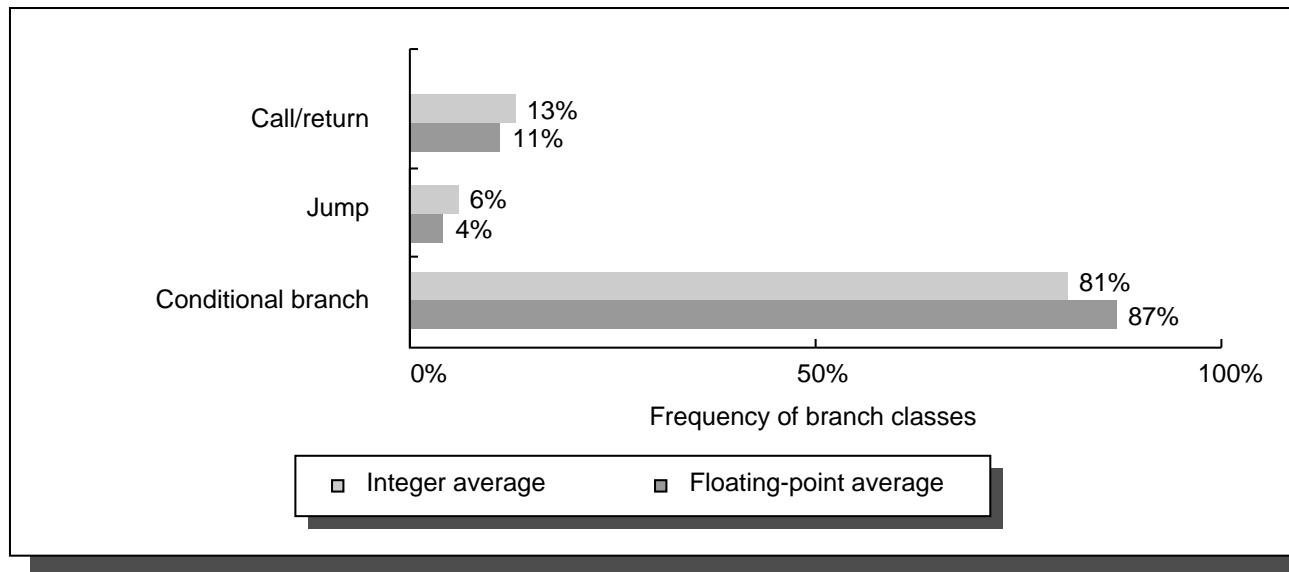
Branch instruction: beq.

Jump instruction: j.

Call instruction: jal.

Return instruction: jr.

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

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

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.

  Not used in MIPS or SPARC.

- *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.

  Used in branch instructions by most ISAs, including MIPS and SPARC.

- *Register Indirect*

  Destination in register.

  Used for subroutine return addresses.

  Used for function address passed as parameters, as with `qsort`.

  Also used by ISAs in which immediates smaller than addresses.

  Used in MIPS (`jr` and `jalr`) and SPARC (`jmpl`).

- *Indexed*

  Destination is sum of two registers.

  Useful for C `switch` and similar statements.

  Used in SPARC `jmpl` (jump and link), not used in MIPS.

- *Region*

  Destination is high few bits of PC concatenated with immediate.

  Used in MIPS `j` and `jal` instructions.

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

Destination Address Examples

Absolute and PC-Relative Examples (Generic ISA)

```
 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.

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

Destination Address Examples

Jump and link SPARC instruction.

First two operands specify address: a register plus immediate.

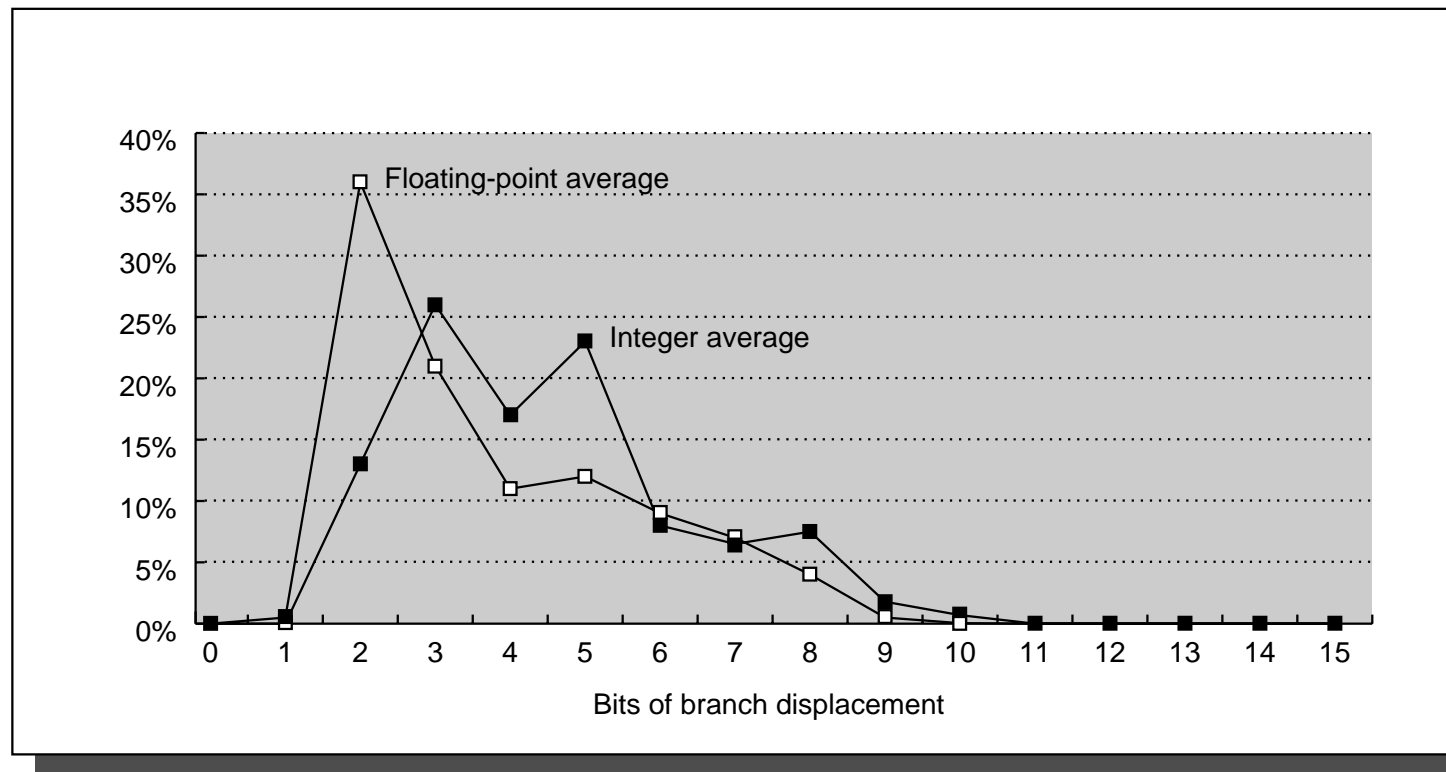Last operand is register to save PC in.

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
```

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

Branch Displacement Size

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

How branch condition might be specified:

- Test value of general-purpose register (GPR).

- Test value of special-purpose *condition code register* (CCR).

- Test value of special-purpose *loop-counter register*.

- 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 (MIPS)

MIPS 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.*, `c.gt.d`) write result into special *FP status* (condition-code) *register*.

```
slt  r1, r2, r3      ! Compare r2 and r3, set r1=1 if r2 < r3.
beq  r1, $0 TARGET1 ! Branch if r1 = 0. (r2 >= r3).
nop
c.gt.d  f0, f2       ! Compare f0 and f2, set FP CCR true if f0 > f2
bc1t TARGET1         ! Branch if FP CCR is true.
....
TARGET1:
```

MIPS Mnemonics:

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

`beq`: Branch if (op1) equal (to op2).

`c.gt.d`: Greater-than double: Set FP CCR if op1 > op2.

Branch Condition Example (SPARC)

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:
```

Branch Condition Example, Itanium (IA-64)

Itanium has special *counted loop branches* and a *loop count* register (LC).

If LC register non-zero counted loop branch is taken and register decremented.

```
 mov lc = 10               ! Set loop count to 10.
LOOP:
 add r100 = r100, r101  ! There are 128 64-bit GPRs.
 br.cloop LOOP;;           ! if( lc > 0 ) { lc--; goto LOOP; }
```

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.)

⇒Test loop counter.

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

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).

$\Rightarrow$ ABI determines how "Additional Support" provided.

04-19

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

04-19

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.

Implementation of Call and Return Steps

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 1 instruction after CTI.

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

```
1  j TARGET        ! Jump to TARGET.
2  add r1,r2,r3    ! Executed after jump.
3  sub r4,r5,r6


TARGET:
```

Without delayed CT, instruction 2 not executed.

With delayed CT instruction 2 is executed, but not 3.

Speeds simple implementations, but complicates modern designs.

Delayed control transfers used on many RISC ISAs, including MIPS, SPARC.

Not included on Alpha.

CTI Variations

Branch Instructions with *Prediction Hints*

Programmer indicates (hints) whether ...

... branch is expected to be taken most of the time or ...

... not taken most of the time.

Hint only affects execution speed, not what the program does.

If programmer correct, execution may be faster ...

... if programmer wrong execution is correct but slower.

A low-cost implementation might ignore the hint.

Example: **Hypothetical** prediction hints added to MIPS.

Hint indicated by following instruction mnemonic with . . .

. . . ,pt (predict taken) or . . .

. . . ,pn (predict not taken).

```
bltz,pn  r1, LINE1   ! Programmer expects r1 to be nonnegative most of the time.
nop
bgtz,pt  r2, LINE2   ! Programmer expects r2 positive most of the time.
nop
add      r3, r4, r5
LINE1:
addi     r3, r3, 1
LINE2:
add      r3, r6, r7
```

Predicated Execution

Predicated Instructions:

Arithmetic or register move instructions along with a condition.

Results written **only if** condition true.

If condition false instruction may still be executed ...
... though results not written.

When predicated execution used fewer branches needed ...
... though total number of instructions may be greater.

Because branches can slow implementations ...
... execution can be faster with predication even though more instructions executed.

Example: Add predication to MIPS (hypothetical). (Pay Attention)

Predicate indicated in parenthesis before mnemonic.

($\langle$**rp**$\rangle$) means write results if $\langle$**rp**$\rangle$ nonzero . . .

. . . (!$\langle$**rp**$\rangle$) means write results if $\langle$**rp**$\rangle$ is zero.

```
! High level code: if( r1 == 0 ) r2 = r2 + r5 else r2 = r2 + r4;
! Without predicated execution.
       beq   r1,$0 LINE1
       nop
       add   r2, r2, r4
       j     LINE2
       nop
LINE1: add   r2, r2, r5
LINE2: ...

! With predicated execution.
(r1)  add   r2, r2, r4  ! Execute if r1 nonzero.
(!r1) add   r2, r2, r5  ! Execute if r1 zero.
```

Is it a coincidence that the destination and first source operands were the same in the adds?

ISA Classification

Consider the way cars are classified:

Criteria: performance, comfort, off-road capabilities.

Classifications: sports cars, luxury cars, SUVs.

A classification may be based on one or more criteria . . .
. . . for example sports cars are high performance and comfortable (excluding ride).

Processors are also classified based on multiple criteria.

Processor Classification: Three Criteria

Address space size: 32-bit, 64-bit, etc.

Register use by instructions: Load/store, memory/memory, stack, etc.

Instruction format: Fixed length, variable length, *bundled* (VLIW).

Criterion: Address Space Size

Describes address space size.

n-bit processor indicates. . .
. . . n-bit address space and fast n-bit integer instructions for pointer arithmetic.

Terminology widely adopted but not universal.

Temptation to inflate n, *e.g.*, 128-bit processor (may refer to bus width).

Criterion: Register Use

Describes how instructions can use registers and addressing modes.

Load/Store, General Purpose Register

Memory only accessed by special load and store instructions.

Few special purpose registers.

Arithmetic and logic instructions use register or immediates.

Examples include most ISAs developed in 1980s and 1990s.

Memory/Memory, General Purpose Register

Memory can be read and written by arithmetic and logic instructions.

Arithmetic and logic instructions can also use register and immediate operands.

Few special purpose registers.

Characteristic of older ISAs.

Criterion: Register Use

Accumulator

Arithmetic instructions implicitly use accumulator.

May include other special-purpose registers.

Stack

Instructions refer to stack.

Used in several old and new machines.

Criterion: Instruction Format

Describes allowed instruction sizes and something called bundling.

Fixed Size

All instructions same size. (*E.g.*, 32 bits).

All instructions do about same amount of work.

Eases implementation.

Characteristic of newer ISAs.

Variable Size

Instruction size depends on instruction and operands.

Potential for compact code.

Implementation difficult.

Characteristic of older and special-purpose ISAs.

Criterion: Instruction Format

Bundled (VLIW,EPIC)

Instructions grouped into bundles, to be executed as a group.

Bundles also contain information to speed execution.

Used in Itanium ISA and older experimental machines.

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

ISA Classifications

Common Classifications and Status

RISC (Reduced Instruction Set Computing)...
... Current General Purpose.

CISC (Complex Instruction Set Computing)...
... Former and Legacy (still used) General Purpose.

VLIW (Very Long Instruction Word)...
... Future General Purpose?

Stack...
... Embedded (built in).

Accumulator...
... Past General Purpose.

CISC (Complex Instruction Set Computing)

Criteria

Register Use: Memory/Memory, General Purpose Register

Instruction Format: Variable Instruction Size

Characteristics

Programmer friendly, lots of instruction choices.

Programmer un-friendly, may contain many almost-useful special purpose instructions.

Average code size.

Implementation cost inflated by ISA complexity.

From mid 1980s to late 1990s ISA complexity slowed implementations but . . .
. . . *fin-de-siècle*[1] CISC processors nearly as fast as RISC processors.

Examples: VAX, 68000, arguably IA-32 (popularly known as 80x86).

---

[1] End of century, 20th century here.

04-34

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

04-34

RISC (Reduced Instruction Set Computing)

Criteria

Register Use: Load/Store, General Purpose Register

Instruction Format: Fixed Instruction Sizes

Characteristics

Work performed by each instruction balanced to ease implementation.

Few special purpose, rarely used instructions.

Code size large. (Because of fixed instruction size.)

RISC Characteristics, continued.

Instructions designed to speed implementation . . .

. . . though sometimes design biased towards contemporary technology . . .

. . . ignoring future technology that might be used in later implementations.

Some implementations rely on compiler to arrange (schedule) instruction for fast execution (discussed later).

Most major ISAs since 1985 are RISC types.

Examples: SPARC V9, Alpha, MIPS, HP PA (Precision Architecture), Power4, PowerPC.

VLIW (Very Long Instruction Word)

Special case: EPIC (Explicitly Parallel Instruction Computing)

Criteria

Instruction Format: Bundled.

Characteristics

Extra information in bundles allows higher speed implementations.

Examples: Cray MTA (supercomputer), TI 320C6 (meant for signal processing), . . .
. . . the new Intel Itanium ISA (implemented by the Itanium processor).

Discussed further later in the course.

04-37

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.

04-37

Stack

Criteria

Register Use: Stack

Instruction Format: Variable Width

Characteristics

Small code size. Important for *embedded* [in cell phone, VCR, automobile engine, etc.] processors.

Limited speed because stack limits execution overlap opportunities.

Examples: Burroughs Corp. machines (1961-1977), INMOS Transputer (1983-).

EE 4720 Lecture Transparency. Formatted 10:07, 4 October 2004 from lsli04.