## 10.11 Instructions unique to PA-RISC 2.0

(Original section[1])

PA-RISC was expanded slightly in 1990 with version 1.1 and changed significantly in 2.0 with 64-bit extensions in 1996. PA-RISC perhaps has the most unusual features of any desktop RISC machine. For example, it has the most addressing modes and instruction formats, and, as we shall see, several instructions that are really the combination of two simpler instructions.

### Nullification

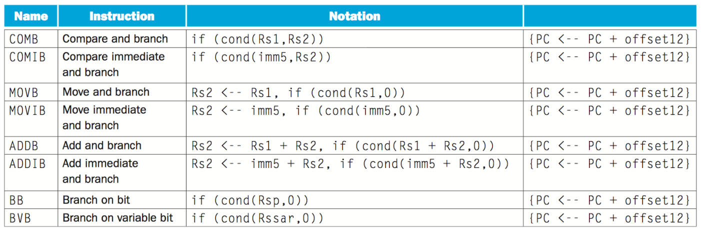
As shown in COD Figure D.6.9 (When the instruction following the branch is executed ...), several RISC machines can choose not to execute the instruction following a delayed branch to improve utilization of the branch slot. This is called *nullification* in PA-RISC, and it has been generalized to apply to any arithmetic/logical instruction as well as to all branches. Thus, an `add` instruction can add two operands, store the sum, and cause the following instruction to be skipped if the sum is zero. Like conditional move instructions, nullification allows PA-RISC to avoid branches in cases where there is just one instruction in the *then* part of an *if* statement.

### A cornucopia of conditional branches

Given nullification, PA-RISC did not need to have separate conditional branch instructions. The inventors could have recommended that nullifying instructions precede unconditional branches, thereby simplifying the instruction set. Instead, PA-RISC has the largest number of conditional branches of any RISC machine. The figure below shows the conditional branches of PA-RISC. As you can see, several are really combinations of two instructions.

Figure 10.11.1: The PA-RISC conditional branch instructions (COD Figure D.11.1).

The 12-bit offset is called `offset12` in this table, and the 5-bit immediate is called `imm5`. The 16 conditions are =, <, <=, odd, signed overflow, unsigned no overflow, zero or no overflow unsigned, never, and their respective complements. The BB instruction selects one of the 32 bits of the register and branches depending on whether its value is 0 or 1. The BVB selects the bit to branch using the shift amount register, a special-purpose register. The subscript notation specifies a bit field.



| Name | Instruction | Notation | |
|---|---|---|---|
| COMB | Compare and branch | if (cond(Rs1,Rs2)) | {PC <-- PC + offset12} |
| COMIB | Compare immediate and branch | if (cond(imm5,Rs2)) | {PC <-- PC + offset12} |
| MOVB | Move and branch | Rs2 <-- Rs1, if (cond(Rs1,0)) | {PC <-- PC + offset12} |
| MOVIB | Move immediate and branch | Rs2 <-- imm5, if (cond(imm5,0)) | {PC <-- PC + offset12} |
| ADDB | Add and branch | Rs2 <-- Rs1 + Rs2, if (cond(Rs1 + Rs2,0)) | {PC <-- PC + offset12} |
| ADDIB | Add immediate and branch | Rs2 <-- imm5 + Rs2, if (cond(imm5 + Rs2,0)) | {PC <-- PC + offset12} |
| BB | Branch on bit | if (cond(Rsp,0)) | {PC <-- PC + offset12} |
| BVB | Branch on variable bit | if (cond(Rssar,0)) | {PC <-- PC + offset12} |

### Synthesized multiply and divide

PA-RISC provides several primitives so that multiply and divide can be synthesized in software. Instructions that shift one operand 1, 2, or 3 bits and then add, trapping or not on overflow, are useful in multiplies. (Alpha also includes instructions that multiply the second operand of adds and subtracts by 4 or by 8: `S4ADD`, `S8ADD`, `S4SUB`, and `S8SUB`.) The divide step performs the critical step of nonrestoring divide, adding or subtracting depending on the sign of the prior result. Magenheimer et al. [1988] measured the size of operands in multiplies and divides to show how well the multiply step would work. Using these data for C programs, Muchnick [1988] found that by making special cases, the average multiply by a constant takes six clock cycles and the multiply of variables takes 24 clock cycles. PA-RISC has ten instructions for these operations.

The original SPARC architecture used similar optimizations, but with increasing numbers of transistors the instruction set was expanded to include full multiply and divide operations. PA-RISC gives some support along these lines by putting a full 32-bit integer multiply in the floating-point unit; however, the integer data must first be moved to floating-point registers.

### Decimal operations

COBOL programs will compute on decimal values, stored as 4 bits per digit, rather than converting back and forth between binary and decimal. PA-RISC has instructions that will convert the sum from a normal 32-bit add into proper decimal digits. It also provides logical and arithmetic operations that set the condition codes to test for carries of digits, bytes, or halfwords. These operations also test whether bytes or halfwords are zero. These operations would be useful in arithmetic on 8-bit ASCII characters. Five PA-RISC instructions provide decimal support.

### Remaining instructions

Here are some remaining PA-RISC instructions:

- *Branch vectored* shifts an index register left 3 bits, adds it to a base register, and then branches to the calculated address. It is used for *case* statements.
- *Extract* and *deposit* instructions allow arbitrary bit fields to be selected from or inserted into registers. Variations include whether the extracted field is sign-extended, whether the bit field is specified directly in the instruction or indirectly in another register, and whether the rest of the register is set to zero or left unchanged. PA-RISC has 12 such instructions.
- To simplify use of 32-bit address constants, PA-RISC includes `ADDIL`, which adds a left-adjusted 21-bit constant to a register and places the result in register 1. The following data transfer instruction uses offset addressing to add the lower 11 bits of the address to register 1. This pair of instructions allows PA-RISC to add a 32-bit constant to a base register, at the cost of changing register 1.
- PA-RISC has nine debug instructions that can set breakpoints on instruction or data addresses and return the trapped addresses.
- *Load* and *clear* instructions provide a semaphore or lock that reads a value from memory and then writes zero.
- *Store bytes short* optimizes unaligned data moves, moving either the leftmost or the rightmost bytes in a word to the effective address, depending on the instruction options and condition code bits.
- Loads and stores work well with caches by having options that give hints about whether to load data into the cache if it's not already in the cache. For example, a load with a destination of register 0 is defined to be a software-controlled cache prefetch.

- PA-RISC 2.0 extended cache hints to stores to indicate block copies, recommending that the processor not load data into the cache if it's not already in the cache. It also can suggest that on loads and stores, there is spatial locality to prepare the cache for subsequent sequential accesses.
- PA-RISC 2.0 also provides an optional branch target stack to predict indirect jumps used on subroutine returns. Software can suggest which addresses get placed on and removed from the branch target stack, but hardware controls whether or not these are valid.
- *Multiply/add* and *multiply/subtract* are floating-point operations that can launch two independent floating-point operations in a single instruction in addition to the fused multiply/add and fused multiply/negate/add introduced in version 2.0 of PA-RISC.

(*1) This section is in original form.