10.16 Instructions unique to MIPS-16

(Original section[1])

MIPS-16 is not really a separate instruction set but a 16-bit extension of the full 32-bit MIPS architecture. It is compatible with any of the 32-bit address MIPS architectures (MIPS I, MIPS II) or 64-bit architectures (MIPS III, IV, V). The ISA mode bit determines the width of instructions: 0 means 32-bit-wide instructions and 1 means 16-bit-wide instructions. The new `JALX` instruction toggles the ISA mode bit to switch to the other ISA. `JR` and `JALR` have been redefined to set the ISA mode bit from the most significant bit of the register containing the branch address, and this bit is not considered part of the address. All jump-and-link instructions save the current mode bit as the most significant bit of the return address.

Hence, MIPS supports whole procedures containing either 16-bit or 32-bit instructions, but it does not support mixing the two lengths together in a single procedure. The one exception is the `JAL` and `JALX`: these two instructions need 32 bits even in the 16-bit mode, presumably to get a large enough address to branch to far procedures.

In picking this subset, MIPS decided to include opcodes for some three-operand instructions and to keep 16 opcodes for 64-bit operations. The combination of this many opcodes and operands in 16 bits led the architects to provide only eight easy-to-use registers—just like Thumb—whereas the other embedded RISCs offer about 16 registers. Since the hardware must include the full 32 registers of the 32-bit ISA mode, MIPS-16 includes move instructions to copy values between the eight MIPS-16 registers and the remaining 24 registers of the full MIPS architecture. To reduce pressure on the eight visible registers, the stack pointer is considered a separate register. MIPS-16 includes a variety of separate opcodes to do data transfers using SP as a base register and to increment SP: `LWSP`, `LDSP`, `SWSP`, `SDSP`, `ADJSP`, `DADJSP`, `ADDIUSPD`, and `DADDIUSP`.

To fit within the 16-bit limit, immediate fields have generally been shortened to 5 to 8 bits. MIPS-16 provides a way to extend its shorter immediates into the full width of immediates in the 32-bit mode. Borrowing a trick from the Intel 8086, the `EXTEND` instruction is really a 16-bit prefix that can be prepended to any MIPS-16 instruction with an address or immediate field. The prefix supplies enough bits to turn the 5-bit field of data transfers and 5- to 8-bit fields of arithmetic immediates into 16-bit constants. Alas, there are two exceptions. `ADDIU` and `DADDIU` start with 4-bit immediate fields, but since `EXTEND` can only supply 11 more bits, the wider immediate is limited to 15 bits. `EXTEND` also extends the 3-bit shift fields into 5-bit fields for shifts. (In case you were wondering, the `EXTEND` prefix does *not* need to start on a 32-bit boundary.)

To further address the supply of constants, MIPS-16 added a new addressing mode! PC-relative addressing for load word (`LWPC`) and load double (`LDPC`) shifts an 8-bit immediate field by 2 or 3 bits, respectively, adding it to the PC with the lower 2 or 3 bits cleared. The constant word or doubleword is then loaded into a register. Thus 32-bit or 64-bit constants can be included with MIPS-16 code, despite the loss of `LIU` to set the upper register bits. Given the new addressing mode, there is also an instruction (`ADDIUPC`) to calculate a PC-relative address and place it in a register.

MIPS-16 differs from the other embedded RISCs in that it can subset a 64-bit address architecture. As a result it has 16-bit instruction-length versions of 64-bit data operations: data transfer (`LD`, `SD`, `LWU`), arithmetic operations (`DADDU/IU`, `DSUBU`, `DMULT/U`, `DDIV/U`), and shifts (`DSLL/V`, `DSRA/V`, `DSRL/V`).

Since MIPS plays such a prominent role in this book, we show all the additional changes made from the MIPS core instructions in going to MIPS-16:

- *Drop of signed arithmetic instructions*—Arithmetic instructions that can trap were dropped to save opcode space: `ADD, ADDI, SUB, DADD, DADDI, DSUB`.
- *Drop of immediate logical instructions*—Logical immediates are gone too: `ANDI, ORI, XORI`.
- *Branch instructions pared down*—Comparing two registers and then branching did not fit, nor did all the other comparisons of a register to zero. Hence these instructions didn't make it either: `BEQ, BNE, BGEZ, BGTZ, BLEZ, and BLTZ`. As mentioned in COD Section D.3 (Instructions: The MIPS core subset), to help compensate MIPS-16 includes compare instructions to test if two registers are equal. Since compare and set on less than set the new T register, branches were added to test the T register.
- *Branch distance*—Since instructions are 16 bits wide, the branch address is shifted by one instead of by two.
- *Delayed branches disappear*—The branches take effect before the next instruction. Jumps still have a one-slot delay.
- *Extension and distance for data transfer offsets*—The 5-bit and 8-bit fields are zero-extended instead of sign-extended in 32-bit mode. To get greater range, the immediate fields are shifted left 1, 2, or 3 bits depending on whether the data are halfword, word, or doubleword. If the `EXTEND` prefix is prepended to these instructions, they use the conventional signed 16-bit immediate of the 32-bit mode.
- *Extension of arithmetic immediates*—The 5-bit and 8-bit fields are zero-extended for set on less than and compare instructions, for forming a PC-relative address, and for adding to SP and placing the result in a register (`ADDIUSP, DADDIUSP`). Once again, if the `EXTEND` prefix is prepended to these instructions, they use the conventional signed 16-bit immediate of the 32-bit mode. They are still sign-extended for general adds and for adding to SP and placing the result back in SP (`ADJSP, DADJSP`). Alas, code density and orthogonality are strange bedfellows in MIPS-16!
- *Redefining shift amount of 0*—MIPS-16 defines the value 0 in the 3-bit shift field to mean a shift of 8 bits.
- *New instructions added due to loss of register 0 as zero*—Load immediate, negate, and not were added, since these operations could no longer be synthesized from other instructions using r0 as a source.

(*1) This section is in original form.

Provide feedback on this section