10.6 Instructions: Common extensions to MIPS core

(Original section[1])

Subsequent figures list instructions not found in COD Section D.3 (Instructions: The MIPS core subset) in the same four categories (data transfer, arithmetic/logical, control, and floating point). Instructions are put in these lists if they appear in more than one of the standard architectures. The instructions are defined using the hardware description language defined in COD Figure D.6.8 (Hardware description notation ...).

Although most of the categories are self-explanatory, a few bear comment:

- The "atomic swap" row means a primitive that can exchange a register with memory without interruption. This is useful for operating system semaphores in a uniprocessor as well as for multiprocessor synchronization (see COD Section 2.11 (Parallelism and instructions: Synchronization) in COD Chapter 2 (Instructions: Language of the Computer)).
- The 64-bit data transfer and operation rows show how MIPS, PowerPC, and SPARC define 64-bit addressing and integer operations. SPARC simply defines all register and addressing operations to be 64 bits, adding only special instructions for 64-bit shifts, data transfers, and branches. MIPS includes the same extensions, plus it adds separate 64-bit signed arithmetic instructions. PowerPC adds 64-bit right shift, load, store, divide, and compare and has a separate mode determining whether instructions are interpreted as 32- or 64-bit operations; 64-bit operations will not work in a machine that only supports 32-bit mode. PA-RISC is expanded to 64-bit addressing and operations in version 2.0.
- The "prefetch" instruction supplies an address and hint to the implementation about the data. Hints include whether the data are likely to be read or written soon, likely to be read or written only once, or likely to be read or written many times. Prefetch does not cause exceptions. MIPS has a version that adds two registers to get the address for floating-point programs, unlike nonfloating-point MIPS programs.
- In the "Endian" row, "Big/little" means there is a bit in the program status register that allows the processor to act either as big endian or little endian (see COD Appendix A (The Basics of Logic Design)). This can be accomplished by simply complementing some of the least significant bits of the address in data transfer instructions.
- The "shared memory synchronization" helps with cache-coherent multiprocessors: all loads and stores executed before the instruction must complete before loads and stores after it can start. (See COD Chapter 2 (Instructions: Language of the Computer).)
- The "coprocessor operations" row lists several categories that allow for the processor to be extended with special-purpose hardware.

Figure 10.6.1: Data transfer instructions not found in MIPS core but found in two or more of the five desktop architectures (COD Figure D.6.1).

The load linked/store conditional pair of instructions gives Alpha and MIPS atomic operations for semaphores, allowing data to be read from memory, modified, and stored without fear of interrupts or other machines accessing the data in a multiprocessor (see COD Chapter 2 (Instructions: Language of the Computer)). Prefetching in the Alpha to external caches is accomplished with `FETCH` and `FETCH_M`; on-chip cache prefetches use `LD_Q A`, `R31`, and `LD_Y A`. `F31` is used in the Alpha 21164 (see Bhandarkar [1995], p. 190).

Name	Definition	Alpha	MIPS-64	PA-RISC 2.0	PowerPC	SPARCv9
Atomic swap R/M (for locks and semaphores)	Temp<–Rd; Rd<–Mem[x]; Mem[x]<–Temp	LDL/Q_L; STL/Q_C	LL; SC	— (see D.8)	LWARX; STWCX	CASA, CASX
Load 64-bit integer	Rd<–$_{64}$ Mem[x]	LDQ	LD	LDD	LD	LDX
Store 64-bit integer	Mem[x]<–$_{64}$ Rd	STQ	SD	STD	STD	STX
Load 32-bit integer unsigned	$Rd_{32..63}$<–$_{32}$ Mem[x]; $Rd_{0..31}$<–$_{32}$ 0	LDL; EXTLL	LWU	LDW	LWZ	LDUW
Load 32-bit integer signed	$Rd_{32..63}$<–$_{32}$ Mem[x]; $Rd_{0..31}$<–$_{32}$ $Mem[x]_0$	LDL	LW	LDW; EXTRD,S 63, 8	LWA	LDSW
Prefetch	Cache[x]<–*hint*	FETCH, FETCH_M*	PREF, PREFX	LDD, r0 LDW, r0	DCBT, DCBTST	PRE-FETCH
Load coprocessor	Coprocessor<– Mem[x]	—	LWCi	CLDWX, CLDWS	—	—
Store coprocessor	Mem[x]<– Coprocessor	—	SWCi	CSTWX, CSTWS	—	—
Endian	(Big/little endian?)	Either	Either	Either	Either	Either
Cache flush	(Flush cache block at this address)	ECB	CPOop	FDC, FIC	DCBF	FLUSH
Shared memory synchronization	(All prior data transfers complete before next data transfer may start)	WMB	SYNC	SYNC	SYNC	MEMBAR

Figure 10.6.2: Arithmetic/logical instructions not found in MIPS core but found in two or more of the five desktop architectures (COD Figure D.6.2).

Name	Definition	Alpha	MIPS-64	PA-RISC 2.0	PowerPC	SPARCv9
64-bit integer arithmetic ops	Rd<–$_{64}$$Rs_1$ op_{64} Rs_2	ADD, SUB, MUL	DADD, DSUB DMULT, DDIV	ADD, SUB, SHLADD, DS	ADD, SUBF, MULLD, DIVD	ADD, SUB, MULX, S/UDIVX
64-bit integer logical ops	Rd<–$_{64}$$Rs_1$ op_{64} Rs_2	AND, OR, XOR	AND, OR, XOR	AND, OR, XOR	AND, OR, XOR	AND, OR, XOR
64-bit shifts	Rd<–$_{64}$$Rs_1$ op_{64} Rs_2	SLL, SRA, SRL	DSLL/V, DSRA/V, DSRL/V	DEPD,Z EXTRD,S EXTRD,U	SLD, SRAD, SRLD	SLLX, SRAX, SRLX
Conditional move	if (cond) Rd<–Rs	CMOV_	MOVN/Z	SUBc, n; ADD	—	MOVcc, MOVr
Support for multiword integer add	CarryOut, Rd <– Rs1 + Rs2 + OldCarryOut	—	ADU; SLTU; ADDU, DADU; SLTU; DADDU	ADDC	ADDC, ADDE	ADDcc
Support for multiword integer sub	CarryOut, Rd <– Rs1 Rs2 + OldCarryOut	—	SUBU; SLTU; SUBU, DSUBU; SLTU; DSUBU	SUBB	SUBFC, SUBFE	SUBcc
And not	Rd <– Rs1 & ~(Rs2)	BIC	—	ANDCM	ANDC	ANDN
Or not	Rd <– Rs1 | ~(Rs2)	ORNOT	—	—	ORC	ORN
Add high immediate	$Rd_{0..15}$<–$Rs1_{0..15}$ + (Const<<16);	—	—	ADDIL (R-I)	ADDIS (R-I)	—
Coprocessor operations	(Defined by coprocessor)	—	COPi	COPR,i	—	IMPDEPi

Figure 10.6.3: Control instructions not found in MIPS core but found in two or more of the five desktop architectures (COD Figure D.6.3).

Name	Definition	Alpha	MIPS-64	PA-RISC 2.0	PowerPC	SPARCv9
Optimized delayed branches	(Branch not always delayed)	—	BEQL, BNEL, B_ZL (<, >, <=, >=)	COMBT, n, COMBF, n	—	BPcc, A, FPBcc, A
Conditional trap	if (COND) {R31<–PC; PC <–0..0#i}	—	T_.,T_I (=, not=, <, >, <=, >=)	SUBc, n; BREAK	TW, TD, TWI, TDI	Tcc
No. control registers	Misc. regs (virtual memory, interrupts, . . .)	6	equiv. 12	32	33	29

Figure 10.6.4: Floating-point instructions not found in MIPS Core but found in two or more of the five desktop architectures (COD Figure D.6.4).

Name	Definition	Alpha	MIPS-64	PA-RISC 2.0	PowerPC	SPARCv9
Multiply and add	Fd <– (Fs1 × Fs2) + Fs3	—	MADD.S/D	FMPYFADD sgl/dbl	FMADD/S	
Multiply and sub	Fd <– (Fs1 × Fs2) – Fs3	—	MSUB.S/D		FMSUB/S	
Neg mult and add	Fd <– -((Fs1 × Fs2) + Fs3)	—	NMADD.S/D	FMPYFNEG sgl/dbl	FNMADD/S	
Neg mult and sub	Fd <– -((Fs1 × Fs2) – Fs3)	—	NMSUB.S/D		FNMSUB/S	
Square root	Fd <– SQRT(Fs)	SQRT_	SQRT.S/D	FSQRT sgl/dbl	FSQRT/S	FSQRTS/D
Conditional move	if (cond) Fd<–Fs	FCMOV_	MOVF/T, MOVF/T.S/D	FTESTFCPY	—	FMOVcc
Negate	Fd <– Fs ^ x80000000	CPYSN	NEG.S/D	FNEG sgl/dbl	FNEG	FNEGS/D/Q
Absolute value	Fd <– Fs & x7FFFFFFF	—	ABS.S/D	FABS/dbl	FABS	FABSS/D/Q

Figure 10.6.5: Data transfer instructions not found in MIPS Core but found in two or more of the five desktop architectures (COD Figure D.6.5).

We use —[1] to show sequences that are available in 32-bit mode but not 16-bit mode in Thumb or MIPS-16.

Name	Definition	ARMv4	Thumb	SuperH	M32R	MIPS-16
Atomic swap R/M (for semaphores)	Temp<–Rd; Rd<–Mem[x]; Mem[x]<–Temp	SWP, SWPB	—[1]	(see TAS)	LOCK; UNLOCK	—[1]
Memory management unit	Paged address translation	Via coprocessor instructions	—[1]	LDTLB		—[1]
Endian	(Big/little endian?)	Either	Either	Either	Big	Either

Figure 10.6.6: Arithmetic/logical instructions not found in MIPS Core but found in two or more of the five embedded architectures (COD Figure D.6.6).

We use —[1] to show sequences that are available in 32-bit mode but not in 16-bit mode in Thumb or MIPS-16. The superscript 2 shows new instructions found only in 16-bit mode of Thumb or MIPS-16, such as NEG[2].

Name	Definition	ARMv4	Thumb	SuperH	M32R	MIPS-16
Load immediate	Rd<–Imm	MOV	MOV	MOV, MOVA	LDI, LD24	LI
Support for multiword integer add	CarryOut, Rd <– Rd + Rs1 + OldCarryOut	ADCS	ADC	ADDC	ADDX	—[1]
Support for multiword integer sub	CarryOut, Rd <– Rd – Rs1 + OldCarryOut	SBCS	SBC	SUBC	SUBX	—[1]
Negate	Rd <– 0 – Rs1		NEG[2]	NEG	NEG	NEG
Not	Rd <– ~(Rs1)	MVN	MVN	NOT	NOT	NOT
Move	Rd <– Rs1	MOV	MOV	MOV	MV	MOVE
Rotate right	Rd <– Rs *i*, >> $Rd_{0\ldots i-1}$ <– $Rs_{31-i\ldots 31}$	ROR	ROR	ROTC		
And not	Rd <– Rs1 & ~(Rs2)	BIC	BIC			

Figure 10.6.7: Control information in the five embedded architectures (COD Figure D.6.7).

Name	Definition	ARMv4	Thumb	SuperH	M32R	MIPS-16
No. control registers	Misc. registers	21	29	9	5	36

Figure 10.6.8: Hardware description notation (and some standard C operators) (COD Figure D.6.8).

Notation	Meaning	Example	Meaning
<--	Data transfer. Length of transfer is given by the destination's length; the length is specified when not clear.	`Regs[R1]<--Regs[R2];`	Transfer contents of R2 to R1. Registers have a fixed length, so transfers shorter than the register size must indicate which bits are used.
M	Array of memory accessed in bytes. The starting address for a transfer is indicated as the index to the memory array.	`Regs[R1]<--M[x];`	Place contents of memory location x into `R1`. If a transfer starts at `M[i]` and requires 4 bytes, the transferred bytes are `M[i]`, `M[i+1]`, `M[i+2]`, and `M[i+3]`.
<--n	Transfer an *n*-bit field, used whenever length of transfer is not clear.	$M[y] \leftarrow\!\!-_{16} M[x]$;	Transfer 16 bits starting at memory location x to memory location y. The length of the two sides should match.
X_n	Subscript selects a bit.	$Regs[R1]_0 \leftarrow\!\!- 0$;	Change sign bit of `R1` to `0`. (Bits are numbered from MSB starting at 0.)
$X_{m..n}$	Subscript selects a field.	$Regs[R3]_{24..31} \leftarrow\!\!- M[x]$;	Moves contents of memory location x into low-order byte of R3.
X^n	Superscript replicates a bit field.	$Regs[R3]_{0..23} \leftarrow\!\!- 024$;	Sets high-order three bytes of `R3` to 0.
##	Concatenates two fields.	$Regs[R3] \leftarrow\!\!-^{24} 0 \#\# M[x]$; `F2##F3<--`$_{64}$`M[x];`	Moves contents of location x into low byte of `R3`; clears upper three bytes. Moves 64 bits from memory starting at location x; 1st 32 bits go into `F2`, 2nd 32 into `F3`.
*, &	Dereference a pointer; get the address of a variable.	`p*<--&x;`	Assign to object pointed to by p the address of the variable x.
<<, >>	C logical shifts (left, right).	`Regs[R1] << 5`	Shift `R1` left 5 bits.
==, !=, >, <, >=, <=	C relational operators; equal, not equal, greater, less, greater or equal, less or equal.	`(Regs[R1]== Regs[R2]) & (Regs[R3]!=Regs[R4])`	True if contents of `R1` equal the contents of `R2` and contents of `R3` do not equal the contents of `R4`.
&, |, ^, !	C bitwise logical operations: AND, OR, exclusive OR, and complement.	`(Regs[R1] & (Regs[R2]| Regs[R3]))`	Bitwise AND of `R1` and bitwise OR of `R2` and `R3`.

One difference that needs a longer explanation is the optimized branches. The figure below shows the options. The Alpha and PowerPC offer branches that take effect immediately, like branches on earlier architectures. To accelerate branches, these machines use branch prediction (see COD Chapter 4 (The Processor)). All the rest of the desktop RISCs offer delayed branches. The embedded RISCs generally do not support delayed branch, with the exception of SuperH, which has it as an option.

Figure 10.6.9: When the instruction following the branch is executed for three types of branches (COD Figure D.6.9).

	(Plain) branch	Delayed branch	Annulling delayed branch	
Found in architectures	Alpha, PowerPC, ARM, Thumb, SuperH, M32R, MIPS-16	MIPS-64, PA-RISC, SPARC, SuperH	MIPS-64, SPARC	PA-RISC
Execute following instruction	Only if branch *not* taken	Always	Only if branch taken	If forward branch *not* taken or backward branch taken

The other three desktop RISCs provide a version of delayed branch that makes it easier to fill the delay slot. The SPARC "annulling" branch executes the instruction in the delay slot only if the branch is taken; otherwise the instruction is annulled. This means the instruction at the target of the branch can safely be copied into the delay slot, since it will only be executed if the branch is taken. The restrictions are that the target is not another branch and that the target is known at compile time. (SPARC also offers a nondelayed jump because an unconditional branch with the annul bit set does not execute the following instruction.) Later versions of the MIPS architecture have added a branch likely instruction that also annuls the following instruction if the branch is not taken. PA-RISC allows almost any instruction to annul the next instruction, including branches. Its "nullifying" branch option will execute the next instruction depending on the direction of the branch and whether it is taken (i.e., if a forward branch is not taken or a backward branch is taken). Presumably this choice was made to optimize loops, allowing the instructions following the exit branch and the looping branch to execute in the common case.

Now that we have covered the similarities, we will focus on the unique features of each architecture. We first cover the desktop/server RISCs, ordering them by length of description of the unique features from shortest to longest, and then the embedded RISCs.

(*1) This section is in original form.