

The Haverford Educational RISC Architecture (HERA)

Quick Guide to Instructions for Version 2.2.3

Arithmetic/Logical/Shift Instructions

Mnemonic	Meaning	Op. Code	Notes
SETLO(d, v)	$R_d \leftarrow v$	E $d v v$	set R_d to signed quantity v (8-bits)
SETHI(d, v)	$(R_d)_{15:8} \leftarrow v$	F $d v v$	set high eight bits
AND(d, a, b)	$R_d(i) \leftarrow R_a(i) \wedge R_b(i)$	8 $d a b$	bitwise logical and
OR(d, a, b)	$R_d(i) \leftarrow R_a(i) \vee R_b(i)$	9 $d a b$	bitwise logical or
XOR(d, a, b)	$R_d(i) \leftarrow R_a(i) \oplus R_b(i)$	D $d a b$	bitwise logical exclusive or
ADD(d, a, b)	$R_d \leftarrow R_a + R_b + (c \wedge F_4')$	A $d a b$	add, use carry unless blocked
SUB(d, a, b)	$R_d \leftarrow R_a - R_b - (c' \wedge F_4')$	B $d a b$	subtract, use carry unless blocked
MULT(d, a, b)	$R_d \leftarrow (R_a * R_b)_{15:0},$ $R_t \leftarrow (R_a * R_b)_{31:16}$	C $d a b$	<i>signed</i> multiplication
INC(d, δ)	$R_d \leftarrow R_d + \delta$	3 $d 10\epsilon\epsilon \epsilon\epsilon\epsilon$	increment R_d by δ (where $\epsilon = \delta - 1$)
DEC(d, δ)	$R_d \leftarrow R_d - \delta$	3 $d 11\epsilon\epsilon \epsilon\epsilon\epsilon$	decrement R_d by δ (where $\epsilon = \delta - 1$)
LSL(d, b)	$R_d \leftarrow \text{shl/rolc}(R_b)$	3 $d 0 b$	logical shift left, possibly with carry
LSR(d, b)	$R_d \leftarrow \text{shr/rorc}(R_b)$	3 $d 1 b$	logical shift right, possibly with carry
LSL8(d, b)	$R_d \leftarrow \text{shl8}(R_b)$	3 $d 2 b$	logical shift left 8 bits
LSR8(d, b)	$R_d \leftarrow \text{shr8}(R_b)$	3 $d 3 b$	logical shift right 8 bits
ASL(d, b)	$R_d \leftarrow \text{asl/aslc}(R_b)$	3 $d 4 b$	arithmetic shift left, possibly with carry
ASR(d, b)	$R_d \leftarrow \text{asr}(R_b)$	3 $d 5 b$	arithmetic shift right

Flag Manipulation Instructions

Mnemonic	Meaning	Op. Code	Notes
SETF(v)	$F \leftarrow F \vee v$	3 000 v 6 $vvvv$	set flags for which v is true
CLRF(v)	$F \leftarrow F \wedge v'$	3 100 v 6 $vvvv$	clear flags for which v is true
SAVEF(d)	$R_d \leftarrow F$	3 $d 7 0$	save flags to R_d
RSTRF(d)	$F \leftarrow R_d$	3 $d 7 8$	restore flags from R_d

Memory Access Instructions

Mnemonic	Meaning	Op. Code	Notes
LOAD(d, o, b)	$R_d \leftarrow M[R_b + o]$	0100 $d oooo b$	load from $R_b + o$ (o is 5-bit <i>unsigned</i>)
STORE(d, o, b)	$M[R_b + o] \leftarrow R_d$	0110 $d oooo b$	store to $R_b + o$ (o is 5-bit <i>unsigned</i>)

Branch Instructions (see Mano, Ch. 9-8)

Mnemonic	Meaning	Op. Code	Notes
BR(<i>b</i>)	$PC \leftarrow R_b$	1 0 0 <i>b</i>	Unconditional branch – <i>true</i>
BL(<i>b</i>)	$PC \leftarrow R_b$ if $(s \oplus v)$	1 2 0 <i>b</i>	Branch if signed result < 0
BGE(<i>b</i>)	$PC \leftarrow R_b$ if $(s \oplus v)'$	1 3 0 <i>b</i>	Branch if signed result ≥ 0
BLE(<i>b</i>)	$PC \leftarrow R_b$ if $((s \oplus v) \vee z)$	1 4 0 <i>b</i>	Branch if signed result ≤ 0
BG(<i>b</i>)	$PC \leftarrow R_b$ if $((s \oplus v) \vee z)'$	1 5 0 <i>b</i>	Branch if signed result > 0
BULE(<i>b</i>)	$PC \leftarrow R_b$ if $(c' \vee z)$	1 6 0 <i>b</i>	Branch if unsigned result ≤ 0
BUG(<i>b</i>)	$PC \leftarrow R_b$ if $(c' \vee z)'$	1 7 0 <i>b</i>	Branch if unsigned result > 0
BZ(<i>b</i>)	$PC \leftarrow R_b$ if z	1 8 0 <i>b</i>	Branch if zero/if equal
BNZ(<i>b</i>)	$PC \leftarrow R_b$ if z'	1 9 0 <i>b</i>	Branch if not zero/not equal
BC(<i>b</i>)	$PC \leftarrow R_b$ if c	1 A 0 <i>b</i>	Branch if carry/unsigned result ≥ 0
BNC(<i>b</i>)	$PC \leftarrow R_b$ if c'	1 B 0 <i>b</i>	Branch if not carry/unsigned result < 0
BS(<i>b</i>)	$PC \leftarrow R_b$ if s	1 C 0 <i>b</i>	Branch if sign (negative)
BNS(<i>b</i>)	$PC \leftarrow R_b$ if s'	1 D 0 <i>b</i>	Branch if not sign (non-negative)
BV(<i>b</i>)	$PC \leftarrow R_b$ if v	1 E 0 <i>b</i>	Branch if overflow
BNV(<i>b</i>)	$PC \leftarrow R_b$ if v'	1 F 0 <i>b</i>	Branch if not overflow
BRR(<i>r</i>)	$PC \leftarrow PC + r$	0 0 <i>r r</i>	Relative branch by <i>r</i> (<i>r</i> is 8-bit <i>signed</i>)
...			
BZR(<i>r</i>)	$PC \leftarrow PC + r$ if z	0 8 <i>r r</i>	Relative branch if zero
BNZR(<i>r</i>)	$PC \leftarrow PC + r$ if z'	0 9 <i>r r</i>	Relative branch if not zero
...			(All branches can also be relative)

Function/Interrupt Instructions

Mnemonic	Meaning	Op. Code	Notes
CALL(<i>s, b</i>)	$M[SP] \leftarrow PC + 1, PC \leftarrow R_b,$ $R_t \leftarrow FP, FP \leftarrow SP, SP \leftarrow SP + s$	2 <i>s s b</i>	Call function at R_b with stack size <i>s</i> (<i>s</i> is 8-bit <i>unsigned</i>)
RETURN()	$PC \leftarrow M[FP],$ $FP \leftarrow R_t, SP \leftarrow FP$	1 1 1 1	Return from function call
SWI(<i>i</i>)		1 1 0 <i>i</i>	Software interrupt # <i>i</i>
RTI()		1 1 1 0	Return from interrupt

Pseudo-Instructions

Mnemonic	Definition	Notes
SET(<i>d, l</i>)	SETLO(<i>d, l</i> &0xff); SETHI(<i>d, l</i> \gg 8)	$R_d \leftarrow l$ (set R_d to 16-bit value <i>l</i>)
CMP(<i>a, b</i>)	SETC(); SUB(0, <i>a, b</i>)	Set flags for $a - b$
NEG(<i>d, b</i>)	SETC(); SUB(<i>d, 0, b</i>)	Set $R_d \leftarrow -R_b$
NOT(<i>d, b</i>)	SET($R_{12}, 0x\text{fff}$); XOR(<i>d, R₁₂, b</i>)	Bitwise complement
HALT()	BRR(0)	Halt the program
NOP()	BRR(1)	Do nothing (“No operation”)
SETC()	SETF(0x08)	Set the carry flag
CLRC()	CLRF(0x08)	Clear the carry flag
SETCB()	SETF(0x10)	Set the carry-block flag
CLCCB()	CLRF(0x18)	Clear carry and carry-block flags
FLAGS(<i>a</i>)	CLRC(); ADD(0, <i>a, 0</i>)	Set flags for R_a
LABEL(<i>L</i>)	(no machine language generated)	Define a label L^*
INTEGER(<i>i</i>)	<i>i</i>	Put <i>i</i> in the current memory cell
TIGER_STRING(<i>s</i>)	<i>s</i>	Put string <i>s</i> in memory for Tiger