

## HERA Architecture and Assembly Language (version 0.99)

The HERA (Haverford educational RISC architecture) processor has seventeen 16-bit registers: a Program Counter (PC) which is only involved in control-flow operations; twelve general-purpose registers (numbered 1 - 12), a Stack Pointer (SP or 15), Frame Pointer (FP or 14), and Old Frame Pointer (OFP or 13) that are used for function calls (and may be used elsewhere); and a Zero register (number 0) that always has the value 0, letting us provide a comparison (CMP), negation (NEG) and other things with a minimal set of instructions.

Arithmetic operations may produce a value, and all but SAVEF set the arithmetic condition flags, which are 0 (or s, for sign), 1 (or z, for zero), 2 (or v, for overflow), and 3 (or c, for carry). These flags may then be used in a branch instruction, and the value of the carry flag may be used in subsequent arithmetic operations. There is an additional 4th flag known as "carry-block". When carry-block is set the carry is not used during arithmetic operations. This flag can be saved, restored, or explicitly modified, but is not affected by arithmetic and cannot be used in a branch.

A logical shift left by 1 bit sets all flags exactly as they would be set if we added a number to itself: c is set iff the number was negative (bit 15 was true), and v is set iff bit 14 xor bit 15 was true. A logical shift right by 1 bit sets c to the old bit 0 and shifts the carry into the leftmost position. For a shift of multiple positions, the resulting flags may be anything, and the carry is only shifted in once for LSL.

This architecture can address  $2^{16}$  16-bit bytes of memory, using the LOAD and STORE instructions. Memory transfer and control flow operations do not affect flags.

### Assembly Language Conventions

In the definitions below, the letters a, b, d, and t refer to four-bit register numbers, and the operation works on the values in the given registers: SUB(3,1,2) subtracts the contents of register 2 from that of register 1 and puts the result in register 3. The letter c refers to the carry bit (0 if unset, 1 if set);  $c^*$  is 1 iff carry is set and carry-block is clear. The letter f refers to the (unsigned) 4-bit flags register [cvzs]. Italicized letters such as *o*, *v*, *m*, and *u* are constants, and are treated as unsigned (positive) quantities during arithmetic: SETLO(1, 9) sets register 1 to the value 9, and INC(1,15) adds 15 to register 1 (rather than adding -1, which has the same 4-bit binary representation).

For the branching instructions (BR, BRN, and BR2), x, x1 and x2 refer to two-bit flag numbers; the terms flag\_s, flag\_z, flag\_v, and flag\_c may also be used.

True instructions (in normal font) each produce a single 16-bit instruction; pseudo-instructions (italicized) are converted by the assembler into a sequence of machine-language instructions.

## Typical Stack Frame Layout (lower addresses lower on page)

SP --> Space above frame for constructing new frames for function calls  
Local variables, temporaries, and saved registers  
Parameters  
Return value  
Static link, if used: 1 word  
Control link (saved old fp): 1 word  
FP --> Return address: 1 word

## Idioms

Single-Precision Arithmetic: (e.g. make R1 the single-precision sum of R2...R4)  
SETCB() // no need for carry flag in single-precision  
ADD(1,0,2) // R1 = R2  
ADD(1,1,3) // R1 = R1 (i.e. value of R2) + R3  
ADD(1,1,4)

Double-Precision Arithmetic: (e.g. Make [R1 R2] the sum of [R3 R4] and [R5 R6])  
CLCCB() // we need add-with-carry, and carry initially false  
ADD(2,4,6) // R2 = R4+R6, carry set if necessary  
ADD(1,3,5) // R1 = R3+R5 plus carry, if set

Function Call with parameters and return value on stack, "callee-save" of registers:  
// Call the function "times2" with R1 as a single-precision param., add 1 to result  
STORE(4,SP,1) // Store R1 at SP+4 (conventional location of 1st param.)  
CALL(12,5,times2) // Call times2, frame size initially 5, using R12  
LOAD(3,SP,2) // Load R2 from SP+3 (conventional return area)  
INC(2,1) // Increment R2 -- now R2 == 2\*R1+1

Function Body (as above):

```
// Body of the "times2" function: 1 single-precision param, 1 local var, 1 saved reg.  
// Assumes we normally run in carry-blocked (single-precision) mode  
LABEL(times2)  
STORE(1,FP,OFFP) // Store OFFP at FP+1 (not needed if no further calls)  
INC(SP,2) // Make space for 1 local, 1 saved reg  
STORE(6,FP,1) // Store R1 at FP+6  
LOAD(4,FP,1) // R1 = 1st parameter (FP+4)  
STORE(5,FP,1) // Save R1 in local variable at FP+5 (no real reason)  
ADD(1,1,1) // Double R1  
STORE(3,FP,1) // Save R1 in return area (FP+3)  
LOAD(6,FP,1) // Get back the value of R1 that we saved 5 lines ago  
LOAD(1,FP,OFFP) // Get back our old frame pointer before doing the return  
RETURN()
```

## Arithmetic (& Logic) Instructions

Name	Args	Meaning (RTL)	Notes
SETLO	(d,v)	$d = v \& 0xff$	Sets high 8 bits to 0, low 8 to v
SETHI	(d,v)	$d = d   (v \ll 8)$	Leaves low 8 bits alone
SET	(d,v)	<i>set d to value v using SETLO(d, v &amp; 0xff); SETHI(d, v &gt;&gt; 8)</i>	
ADD	(d,a,b)	$d = a + b + c^*$	Add (with carry (if not blocked))
SUB	(d,a,b)	$d = a - b - c^*$	Subtract with Carry (borrow)
CMP	(a,b)	<i>set flags for a-b-c*, discard result: SUB(0,a,b)</i>	
NEG	(d)	<i>negate d: SUB(d,0,d)</i>	
UMULLO	(d,a,b)	$d = (a * b) \& 0xffff$	Low 16 bits of a*b (unsigned)
UMULHI	(d,a,b)	$d = (a * b) \ll 16$	High 16 bits of a*b (unsigned)
AND	(d,a)	$d = d \& a$	
OR	(d,a)	$d = d   a$	
NOT	(d)	$d = \text{not } d$	
XOR	(d,a)	$d = d \text{ xor } a$	
NAND	(d,a)	$d = d \text{ nand } a$	
ZERO	(d)	<i>set d to 0: AND(d,0,0)</i>	
SETF	(m,v)	$f = (f \& \text{not } m)   (v \& m)$	Set/Clear flags with mask m
CLCCB	()	<i>clear carry &amp; carry-block: SETF(16+8,0)</i>	
SETCB	()	<i>set carry-block flag: SETF(16,16)</i>	
SAVEF	(d)	$d = f$	Save flag values in register d
RSTRF	(a)	$f = a \& 7$	Restore flags saved in a
FLAGS	(a)	<i>set flags for a+0, discard result: SETF(8,0); ADD(0,0,a)</i>	
INC	(d,u)	$d = d + u + c^*$	(note: u is 4-bit unsigned (0-15))
DEC	(d,u)	$d = d - u - c^*$	
LSL	(d,u)	$c \ d = [d \ c^*] \ll u$	Logical shift left u bits w/ carry
LSR	(d,u)	$d \ c = [c^* \ d] \gg u$	Logical shift right u bits w/ carry

## Memory & Register Transfer Instructions

Name	Args	Meaning (RTL)	Notes
LOAD	(o,a,d)	$d = M[a+o]$	Load a word from memory (note: o is 5-bit unsigned (0-31))
STORE	(o,a,b)	$M[a+o] = b$	Store a word into memory

## Control Flow Instructions

Name	Args	Meaning (RTL)	Notes
BR	(x,a)	x: PC = a else: PC = PC+1	Branch to a iff flag x is true
BRN	(x,a)	!x: PC = PC else: PC = PC+1	Branch to a iff flag x is false
BR2	(x1,c2,a)	x1 or !x2: PC = a else: PC = PC+1	Branch to a iff flag x1 or ! flag x2 (note: x1==x2 always branches)
CAL	(o,a)	M[sp] = PC+1, PC = a ofp = fp fp = sp sp = sp + o	Call function at address a  (note: ofp is overwritten)  (note: o is 8-bit unsigned (0-255))
RETURN	()	PC=M[fp] fp = ofp sp = fp	Return from a function (note: may need to fix ofp first)
SWI	()		Software interrupt (sys. stack/reg)
RTI	()		Return from interrupt
NOP	()		Do nothing
HALT	()	PC = PC	Halt the processor
<i>LABEL</i>	<i>(l)</i>		<i>Define label l for branch</i>
<i>JUMP</i>	<i>(t,l)</i>		<i>Jump to l unconditionally: SET(t,l); BR2(0,0,t)</i>
<i>BREQ</i>	<i>(t,l)</i>		<i>Branch if a=b in last CMP: SET(t,l); BR(flag_z,t)</i>
<i>BRNE</i>	<i>(t,l)</i>		<i>Branch if a!=b in last CMP: SET(t,l); BRN(flag_z,t)</i>
<i>BRLT</i>	<i>(t,l)</i>		<i>Branch if a&lt;b in last CMP: SET(t,l); BR(flag_n,t)</i>
<i>BRGE</i>	<i>(t,l)</i>		<i>Branch if a&gt;=b in last CMP: SET(t,l); BR2(flag_z,flag_n,t)</i>
<i>CALL</i>	<i>(t,o,l)</i>		<i>Call func. at label l: SET(t,l); CAL(o,t)</i>