Haverford Educational Research Architecture (HERA)
Version 2 Update History and Plans/Errata

Changes to most significant version number (HERA 1 to HERA 2) indicate major changes in the instruction set; the next number (e.g., 2.3 to 2.4) indicates minor change to instruction semantics; the third generally indicates clarifications or minor corrections to documentation, with little/no impact on code semantics.

Errata for Current Version (2.4.0) … please add requests in GDocs suggest mode:

Corrections (✓checkmarks ⇒ corrected in the master file for 2.4.1, ask if you want PDF):

1. Correction/clarification: (Sec. 2.4) The description of ASR says it "always produc[e] a result that is half of the (signed) value", but does not specify how odd values should be rounded. They should be rounded down, so an ASR of the value 5 produces 2, and an ASR of -5 produces -3. (Thus, the corresponding "mod" will always give 0 or 1, not 0 or -1). ✓
2. Correction: (Sec. 2.2) The definition of the carry flag after the SUB operation is missing the words "or equal to" in the text "SUB sets *c* to true … (i.e., if the unsigned interpretation of R_a is greater than **or equal to** that of R_b). ✓
3. Clarification: The description of CALL/RETURN should explicitly state "The semantics of CALL and RETURN are undefined when a=b" (in Section 2.7.2, right after "Neither changes any flags"). ✓

Minor clarifications/resolutions of ambiguity:

- Idiom correction: in an example(s) in 7.4, the FP_alt register is not re-established even though it should be, before a second call from a function in which SP has changed.
- The description of the call idiom in 7.4 should probably say MOVE(FP_alt, SP), and define FP_alt *above* this point, rather than MOVE(R12, SP). The generated machine code is the same, though.

- Page 11/Section 1.3 states "HERA can address 2^{16} 16-bit words of memory, using the LOAD and STORE instructions." This should, perhaps, also mention that the *addresses* (i.e., program counter and branch targets) are also 16 bits (this is a consequence of being able to individually address each of the 2^{16} words, and is sort of implied by the branch section, but should be clear here, too).
- Page 14/Section 2.2 talks about setting s and z flags for "the result" … for multiplication, this is the 16-bit result, at least when CB is true… possibly in some future release, we'll want to record different information when working with (or producing) multiple-precision values by setting CB=false.
- Page 14/Section 2.2 uses the phrase "The latter set *v*…"; this means the group of the last *three* things after the word "but" in the previous sentence, i.e., ADD, SUB, and MUL. Perhaps it should be rewritten to be clearer.
- Page 14/Section 2.2 includes the text "...MUL produces produces the low word (bits 15-0)...". the word "produces" should only appear once in that phrase.
- Page 15/Section 2.4 includes the phrase "LSL, LSR, and ASL shift in the value ($c \wedge F_4'$)."". Some have found this phrasing a bit confusing. Perhaps this, and more details of shifting, could be clarified by instead starting this paragraph "These shift the bits of R_b by one bit (except for LSL8 and LSR8), and place the result in R_d. For arithmetic shifts, the sign bit of the R_d is always the sign of R_b. The value fed into the "open" end of LSL, LSR, and ASL (i.e., the leftmost bit of LSR, and the rightmost of LSL and ASL) is defined as ($c \wedge F_4'$)."
- Page 17's description of relative vs. register (absolute) branching is potentially confusing. J.D.'s statement is, hopefully, clearer: "Relative branches are indicated by an appended “R” in the assembly language name, and by b_{12}= 0 (vs. b_{12}= 1 for absolute branches) in machine code."

- Page 29/Section 5.1 and Page 30/Section 5.2 would be more helpful in understanding the role of the assembler if they emphasized the difference between the, specifically by emphasizing
    - the fact that instructions like SET(Rt, 0x0174) BR(Rt) would normally just be written BR(label), with the assembler figuring out that the label is on instruction 0x0174; and
    - the BGER/BGE options of Figure 5.1 by, e.g., boldfacing the part of the generated machine language that changes, i.e., **eb09 fb02 130b**.
- Page 47/Section 7.4 uses R10 and R11, and I've been trying to use R1...R9 for general computation; perhaps I'll revise that later…
- (Thanks to Dakotah for sharing thoughts based on experience teaching HERA) Quick Guide might be less confusing if
    - There was one column for each flag, showing whether it becomes zero or one or is left alone
    - At least one more relative Branch was listed, with ellipsis under each column, to help students notice the text if they are skimming down the set of numeric instructions…. Or..
    - Instead, have one line for all the register mode branches, and another for all of the relative branches, and then a table of the condition fields

Extensions under consideration:

- More specification for multiply (this would undercut some possible design activities in CS240)
- Reserve/specify some op codes as available for local extensions (as per RISC-V's "portions of the encoding space are guaranteed to never be used..." [as quoted in RISC-V tutorial at HiPEAC 2019])

NOTE FOR ANYONE WITH HERA 2_3 CODE THEY WANT TO RUN:

- Ask Dave W. about "HERA 2.3 compatibility mode" for HERA 2.4, and `#include <HERA_2_3_compatibility.hera>`

Errata/notes about Tools, e.g. Hera-C and Hassem

- Known *limitations* of HERA-C (there is no plan to fix these; I don't know how, given the approach)
    - Branches and calls in HERA-C **must use *labels*** rather than registers (or offset, for e.g BRR)
    - As per the C language convention, **numbers expressed with a leading 0 are treated as octal**, so 05 == 5, but 015 == 13 != 15 (I'd like to have Hassem warn about these.)
    - Forward-references to DLABELS are not allowed, so for example "node2" is an error in:

```
DLABEL(node1)  // doubly-linked-list-of-primes
 INTEGER(2)
 INTEGER(0)     // no previous node
 INTEGER(node2) // Not legal in HERA-C, don't know how to easily fix this…
DLABEL(node2)
 INTEGER(3)
 INTEGER(node1) // This is o.k.
 INTEGER(node3)
...
```

    - HERA-C accepts many things that are not legal HERA, so it is quite possible to get a program that seems to work in HERA-C but then won't assemble with Hassem (or, worse, assembles and then gives a different answer)

- Known bugs in Hassem (and recently-fixed bugs, still visible in pale text for reference)

    - Hassem sometimes does not produce error messages for certain mis-formed integer literals, e.g. SET(R1, 0ob00101010) (Note that HERA-C should complain about these)
    - Hassem does not (yet) give a warning for numbers that are written with leading 0's, though it does now interpret them in Octal, to be consistent with HERA-C, so 052 is 42, not 52.
    - I'd like to have octal and binary work by prefixing, so e.g. 0b1100 == 0x0c == 0o14 (I think the Chapel language does this, but not C, so it's not likely in HERA-C)
    - **Possibly?** (I've not yet checked) Hassem shares the same "no-forward-DLABEL-references"?

- Known bugs in HERA-C (and recently-fixed bugs, still visible in pale text for reference)
    - A bug exists in the MUL instruction when CB=false, for "HERA-C HERA 2.4.0 simulator, October 2018 edition". When CB=false, S=true, all other flags=false, any negative number multiplied by 1 (e.g., 1*-1) may leave the outgoing carry flag set incorrectly. (**still has to be confirmed, but be careful**)
    - Also when any two negative numbers are multiplied and all flags are off, the carry flag is set to true. But when any two negative numbers are multiplied and the sign flag is off, the carry flag is set to false. (**still has to be thought through and confirmed, but be careful**)

- If you are using Eclipse *for an older project (i.e., one created in our lab for HERA 2.3 or earlier)*, you'll need to adjust the include path for the libraries
    - right-click on the project in the Eclipse "Project Explorer" pane
    - select "properties" from the pop-up menu (at/near the bottom)
    - click the triangle by C/C++ build to open the sub-options
    - click on the word "settings" that is now visible below C/C++ build
    - if necessary, click the "tool settings" tab
    - under GCC C++ Compiler, select "Includes"
    - remove /home/courses/include
    - add /home/courses/include/HERA-C/lib (it should be below the /home/courses/include/HERA-C that's already there, and should remain)
    - click "apply and close"
    - if necessary, clean project and build project

- HERA-C-Run and Hassem should already recognize the compatibility file if you #include it as above
    - e.g., just type **`HERA_C_Run myfile.cc`** to *run* your program
    - e.g., just type **`Hassem myfile.cc`** to *assemble* your program (then see myfile.lcode for the logisim-formatted sequence of machine-language opcodes). Note that **`XHassem myfile.cc`** should assemble your program and open a window with the machine language.

HERA Version 2.4 to 2.5 CHANGES UNDER CONSIDERATION:

- Change Shifts, etc., so that $b_{11:8}$ are consistently the register to-be-written
-
-
- Explicitly reserve some operations for local extensions (as is done in RISC-V?)

HERA Version 2.3 to 2.4

A number of significant changes were made between 2.3.3 and 2.4 (the first two change binary semantics):

- `RETURN` now sets to the address of the instruction following the `RETURN`, is thus like `CALL`.
    - We can now think of these as either two instructions that each do a co-routine transfer, or as a traditional call/return pair.
    - Implementation is simplified because op codes 20** and 21** share semantics.

- The multiplication operation changed, simplifying coding (see below) and implementation:
    - A new `MUL` operation replaces the old `MULT`, i.e., op code `C`*`dab`* now encodes `MUL`(*d*, *a*, *b*). `MUL` produces a single 16-bit value, which (depending on flags) may be the low word ($b_{15:0}$) or high word ($b_{31:16}$) of the product, for either signed or unsigned multiplication:
        - If carry-block (F_4) is true, `MUL` produces the low word of *a*b*; thus, when it is known (or assumed) that all operations will use *and produce* results in the range [-2^{15} … $2^{15}-1$], all flags can be ignored if carry-block is true (I recommend requiring this functionality for 240, but making the rest of `MUL` optional).
        - If carry-block is false, `MUL` can compute various bits of either a*b (or, perhaps, some day, fused multiply-add a*b+c?), and is controlled by remaining flags … various values of (C,V,Z,S), listed in that order below, provide:
            - 0,0,0,0 produces the low word, as if carry-block had been set
            - 0,0,0,1 produces the high word of the signed result;
            - others are undefined in 2.4.0, but may be extended in 2.4.1 if pedagogically appropriate … possibly leaving these unspecified could make an interesting co-design challenge in CMSC 240, in which students might try to *design* extensions to allow some or all of the following
                - unsigned results
                - fused-multiply add (i.e., d=d+a*b or d=a+d*b, or some other)
                - something else that is the product of a co-design effort to create the fastest support for bignum's, where fastest could mean "fewest instructions executed" (or fewest nanoseconds?) for some blend of bignum operations including +, -, * (and maybe // and %, mostly in software, but part of this test?)... even designing the test is interesting.
    - `MUL` affects all flags, often differently from version 2.3:
        - sign and zero flags are now set based only on the requested word of the result, but if we have a request for a high word, or even CB=0, F=0000, possibly that should change the semantics so that 0x4000 * 0xC000 has s=T z=F rather than s=F z=T… greater investigation of multi-word multiplication will help to understand this;
        - the carry flag is now affected, being True iff $b_{31:16}$ of unsigned product aren't all zero;
        - the overflow flag is still true iff the sign-extended low word isn't the full signed result.
    - **For 2.4.1** (i.e., not yet), I expect to add Pseudo-ops for common cases of double-precision.

- Flag-setting operations have been renamed, so that the term "set" is now consistently used to mean "define a value that may be true or false" rather than "define as true", the latter being common in descriptions of one-bit systems but unfamiliar to folks new to hardware. Specifically, 3[01]6*v* and 3[89]6*v* have the same semantics, but are now said to "turn specific flags *on*" (or *off*), rather than to set and clear them, and the instructions are known as FON (vs SETF) and FOFF (vs CLRF); literal values can be *set* into the flag register with FSET5 (which includes carry-block) and FSET4 (which does not, and is more frequently useful).

- The “temporary” register is now only an assembler/coding convention, since (without the `MULT` op) it is only used in pseudo-ops and assembler-directed branches/calls. For the most part, assemblers should now use "R_t is R_{11}" in local branches and arithmetic and pseudo-ops, with function calls and returns using R_{12} (for function-start address and return address) and R_{13} (for the FP and old FP). Assemblers may, in principle, allow switches or pseudo-opo to designate which register should be used as R_t in these four distinct usage cases, but only if they allow a mechanism link only with libraries that are compatible.

- The document has been re-structured and significantly extended. The new version has more depth on some of the early idioms, especially single-precision arithmetic, which now discusses characters, and does a "from-the beginning" description more appropriate for those with no prior assembly-language experience. The new version better-separates the "assembly language programming idioms" (for both 245 and 240, as well as 350; this is meant as a teaching/explanatory document) from the "definition of what the processor and assembler provide" (for 240 and 356; Part I is still like a regular hardware definition manual, without much about "why") parts.

- The document's discussion of details has also been clarified, including:
    - Branches, including CALL/RETURN, do not change any flags
    - `LP_STRING` (pseudo-op for length-prefixed string) is now used consistently (rather than older `TIGER_STRING`, which is available for compatibility, but not mentioned in the spec.).
    - Examples of function calls (to library code) now appear before examples of func. definitions
    - Added hybrid register saving convention for calls (idioms section)

Updates within Version 2.3

Version 2.3.2 to 2.3.3

No changes to the HERA architecture itself; significant enhancements of the discussion of function call idioms in Section 5.5; the only changes before that point (Page 16) are a few corrections of typos/spelling mistakes and references to updated Section numbers at/after 5.

Version 2.3 and 2.3.1 to 2.3.2

Flags should now be defined in all cases; in particular, overflow should only come on when there is an overflow (loss of correctness within 16 signed bits).

HERA Version 2.2 to Version 2.3

- Function call now expects new frame between some register (above FP) and SP, and swaps that register with FP. This allows the maintenance of the invariant that only addresses below SP are in use in the stack, thus opening new options for handling interrupts, e.g., placing the interrupt stack adjacent to the regular stack, starting at SP.
- Furthermore, it saves the return address in the register that had provided the destination address. Thus, CALL and RETURN no longer need to access main memory; they use only registers. (Any remaining description of the CALL or RETURN instruction changing R_t, rather than whatever register is provided in the op code, are no longer accurate.)
- The flag settings for shift operations have been updated.
- Additionally, a standard MOVE pseudo-op is provided in Version 2.3.

Other minor corrections were made to the writeup for Version 2.3.

Updates within Version 2.3

Version 2.2.3 to Version 2.2.4

Corrected HERA documentation, including Quick Guide, to clarify things and remove inconsistencies:

- Changed the reference to the ‘‘negative" flag to correctly refer to the ‘‘sign" flag;
- Corrected the error in the description of the role of carry and carry-block in subtraction; main documentation now agrees with definition from quick guide: R_a-R_b-(c'∧F_4')
- Changed the example of SETF(0x15) to be 0x3865 rather than 0x3965.
- Changed uses of r for relative branch offset to be o (as elsewhere in the document).
- Added subsections to the ‘‘Idioms" section to give example uses of control flow and memory ops.
- Added a note that HERA-C requires all data stuff to come before any instructions.
- Swapped the order of two function call idiom sections, and removed one, to focus more on the way we do things in our compiler design class at Haverford (anyone who really wants that old section restored should contact the author).
- Added, to the now-first function call idiom section, enough code to make a complete runnable example and a diagram of the stack layout during the run of this complete example.
- Miscellaneous minor typos e.g. ‘‘in requires" --> ‘‘it requires".

Version 2.2.2 to Version 2.2.3

Fixed minor inconsistencies in HERA documentation and HERA-C, regarding the parameters for the NOT and NEG pseudo-ops — now both have distinct source and target registers.

Version 2.2.1 to Version 2.2.2

Various minor corrections/clarifications of inconsistencies (generally fixed in ‘‘obvious" ways):

- Fixed an inconsistency in the description of the binary instructions for flag set/clear, and corrected the totally wrong examples that had added to the confusion.
- Fixed incorrect examples of INC/DEC op. codes.
- Added Op. Codes column to Quick Guide.

Version 2.2 to Version 2.2.1

Corrected the definition of BUG and BULE to have C' rather than C (also cited Mano's definition for branch instructions).

Version 2.1 to Version 2.2

No binary operations were changed, but some flag definitions were changed to make a few things better:

- SETLO and SETHI no longer affect any flags. This allows assemblers to accept register-mode branch instructions with labels and use the temporary register (e.g. BRZ(label) becomes SET(tmp, label); BRZ(tmp)), which would not have worked in version 2.1 because the SETLO and SETHI would have killed the values of some flags.
- 8-bit shifts now shift in eight 0's — the other thing was goofy.

Also clarified the TIGER_STRING cannot contain control characters.

Version 2.0 to Version 2.1

Version 2.0 Alpha release 2 to Version 2.1

Several op codes have changed to make the following organizational changes:

- XOR is now a 3-address operation, like the other arithmetic operations.
- NOT is now a pseudo-op that sets R_t and uses XOR

Version 2.0 Alpha 1 to Version 2.0 Alpha 2

- Clarified that SETLO sign-extends a signed quantity
- Clarified that INC/DEC assembly language requires the value to be added/subtracted
- Retracted the unfortunate attempt to change the order op many operands

Version 1 to Version 2.0

Many, many elements of the HERA machine language have changed since Version 1. However, the assembly language is largely compatible with that of Version 1, except for the following

- For version 2.0 alpha 2 and later, most of the operands are in the same order, though there may be some exceptions (the 2.0 alpha 1 made changes to the order that were quickly retracted).
- most bitwise operations (except XOR) are now three-address rather than two-address (XOR is included in V.2.1)
- the bitwise NAND operation has been dropped
- logical shifts must be either 1-bit or 8-bit shifts (though now arithmetic shifts and rotates are supported)
- The LOAD operation sets the s and z flags; shift and bitwise operations no longer set c and v.
- The branch operations and pseudo-ops have changed significantly
- The ZERO pseudo-operation is no longer supported (as far as I know it was never used).
- The unconditional branch pseudo-op JUMP has been omitted (due to the new BR pseudo-op)
- The CALL and branch pseudo-operations no longer need a temporary register -- they use R_{13}

automatically.