Introduction to Assembly


intel

Today we’re going to start learning assembly language

Today we’re going to start learning assembly language

We’re going to use HERA, which is a synthetic assembly language

Today we’re going to start learning assembly language

We’re going to use HERA, which is a synthetic assembly language

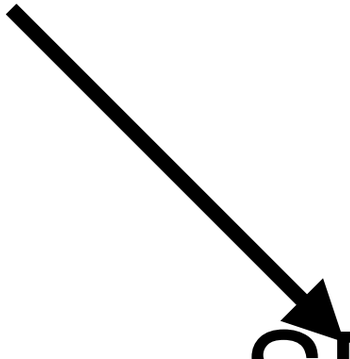
Same basic concepts as all other assembly languages. Later I’ll show you LLVM

Logistical notes

- HERA doesn't run on a "real" processor like an i7
- Instead, you'll use a simulator
- You write it within a C++ program
- A library (we wrote) interprets as an assembly program

```
1  // iterative factorial calculation just for 6! and 9! in HERA
3
4  #include <HERA.h>
5  #include <HERA-print.h>
6
7  void HERA_main()
8  {
9    // We use "single-precision" initially,
10   //  so we set the carry-block flag here.
11
12     SETCB()
13
14   // COMPUTE AND PRINT 6! AND 9!
15
16   print("\n***  Let's 6! and 9! as easily as we can, i.e., by straight-line sequences of steps \n")
17
18     SET(r1, 1)
19     SET(r2, 2)
20     MULT(r1, r1, r2)    // r1 = 1*2
21     SET(r2, 3)
22     MULT(r1, r1, r2)    // r1 = 1*2*3
23     SET(r2, 4)
24     MULT(r1, r1, r2)    // r1 = 1*2*3*4
25     SET(r2, 5)         // note: since r2 was 4, this could be INC(r2, 1)
26     MULT(r1, r1, r2)    // r1 = 1*2*3*4*5
27     SET(r2, 6)         // note: since r2 was 5, this could be INC(r2, 1)
28     MULT(r1, r1, r2)    // r1 = 1*2*3*4*5*6
29
30   print("6! is ")
31   print_reg(r1)
32
33     // now, try to find 9! in single precision (starting with 6! in r2)
34
35     INC(r2, 1)  // taking advantage of the observation above
36     MULT(r1, r1, r2)
37
38     INC(r2, 1)
39     MULT(r1, r1, r2)
40
41     INC(r2, 1)
42     MULT(r1, r1, r2)
43
44   print("9! computed in single precision is ")
```

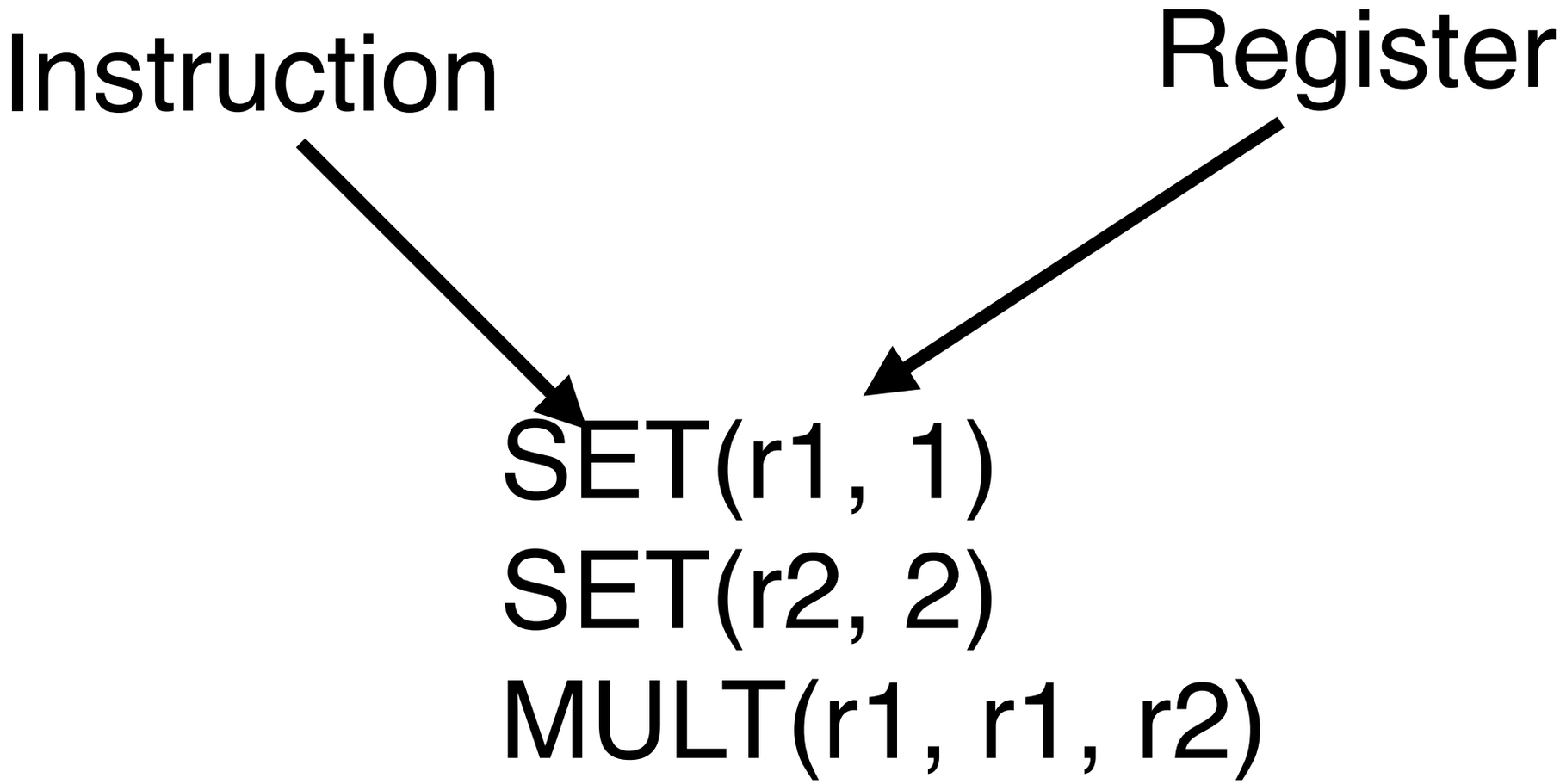
Instruction

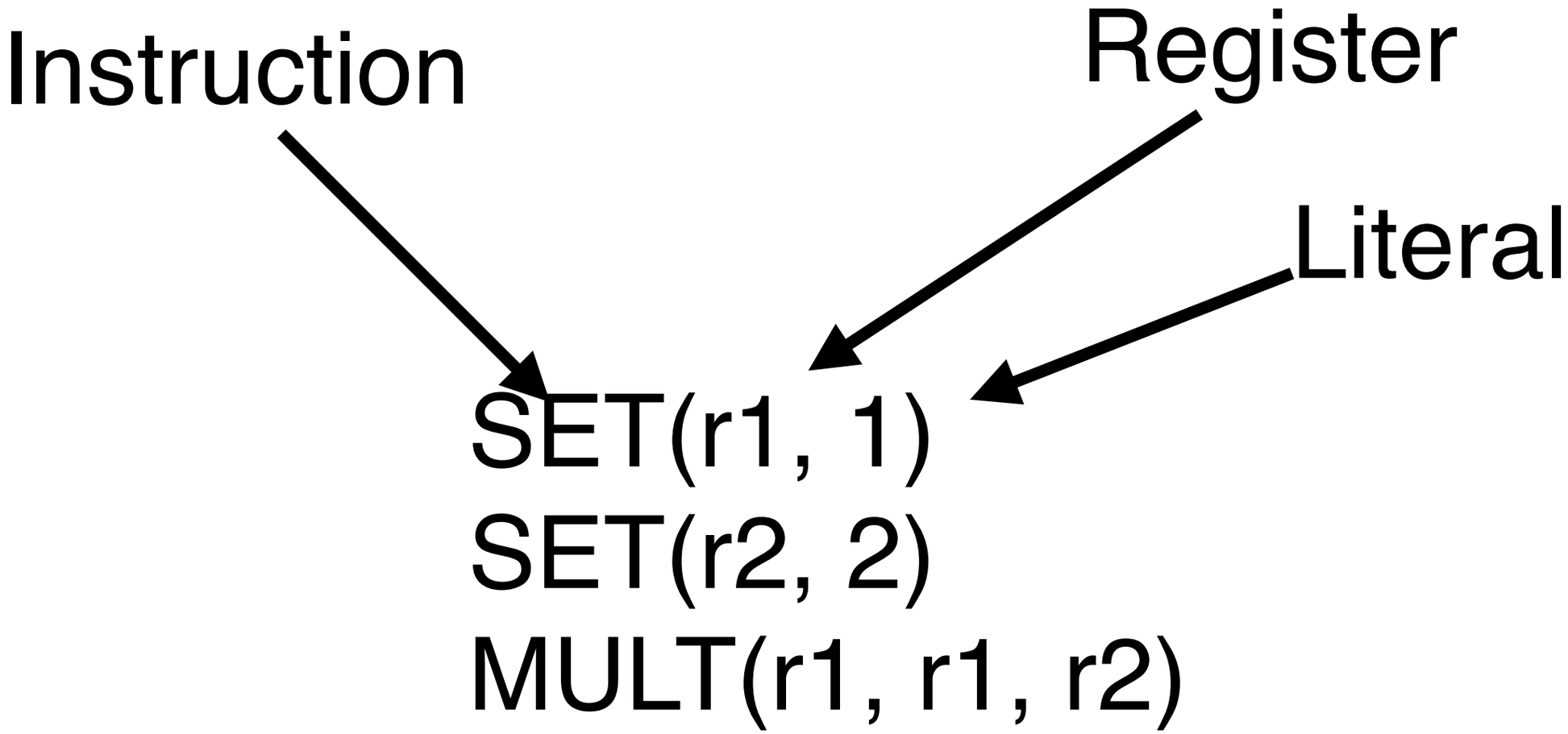


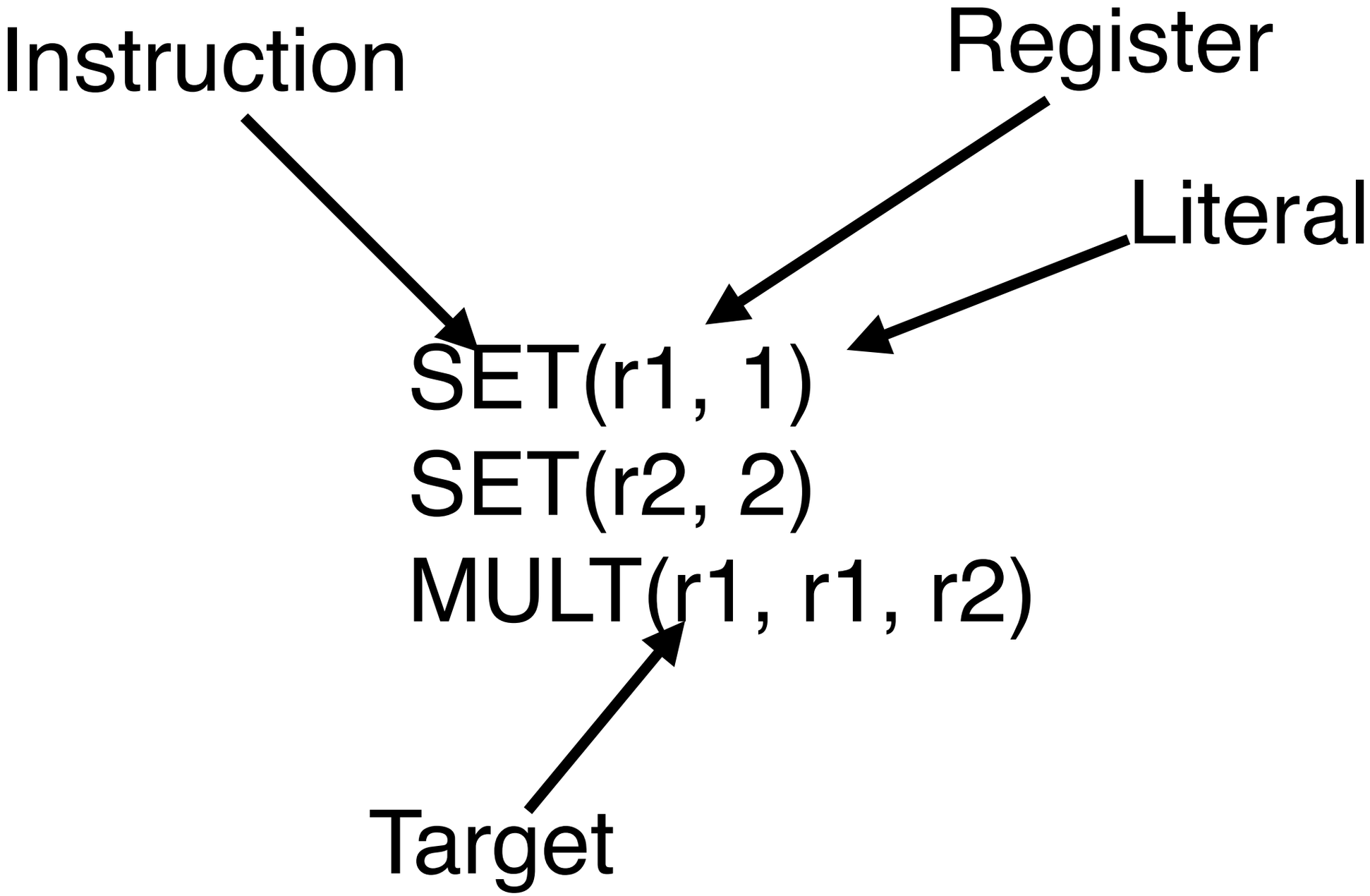
SET(r1, 1)

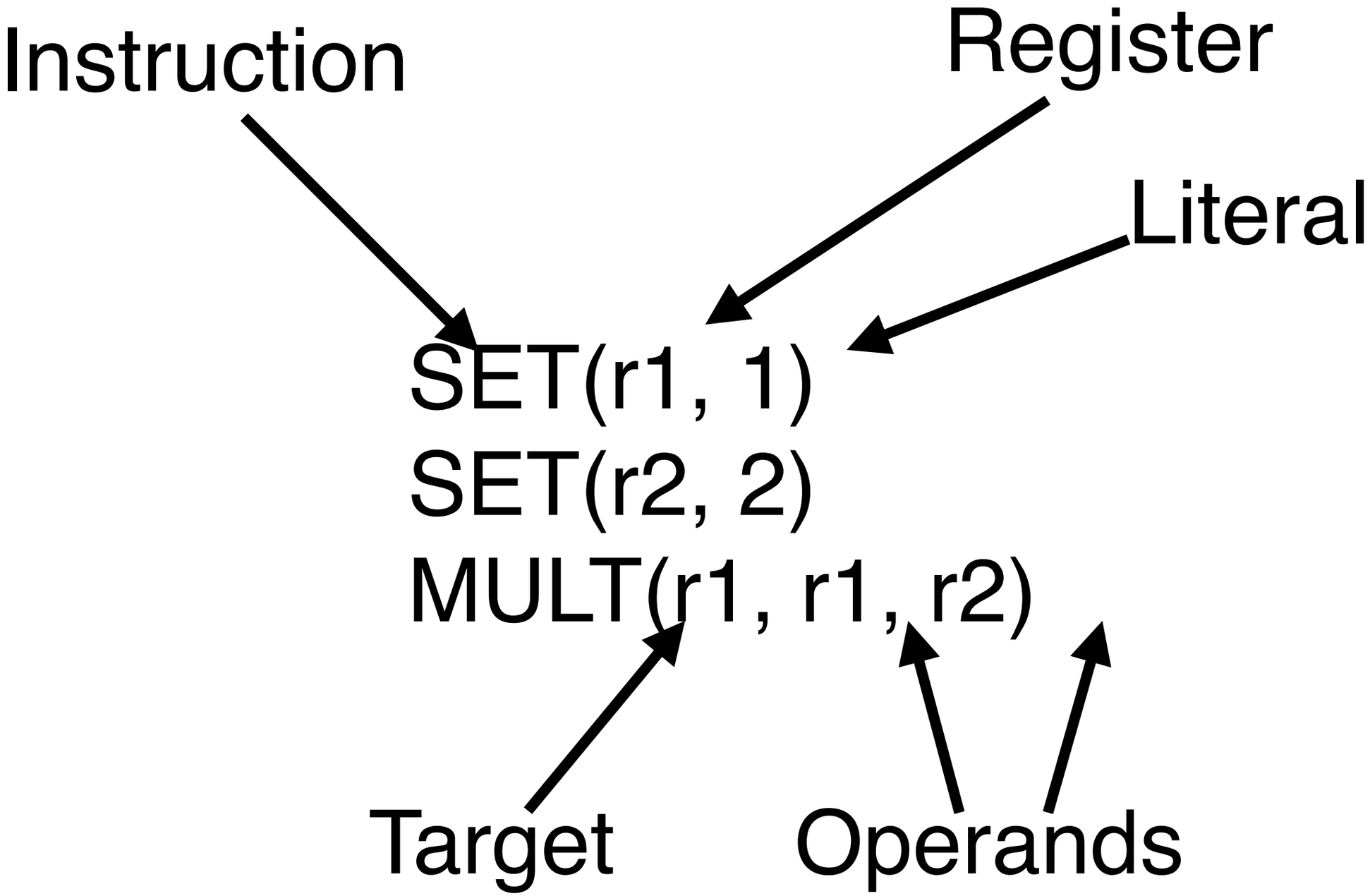
SET(r2, 2)

MULT(r1, r1, r2)


Instruction
Register
SET(r1, 1)
SET(r2, 2)
MULT(r1, r1, r2)


Instruction
Register
Literal
SET(r1, 1)
SET(r2, 2)
MULT(r1, r1, r2)

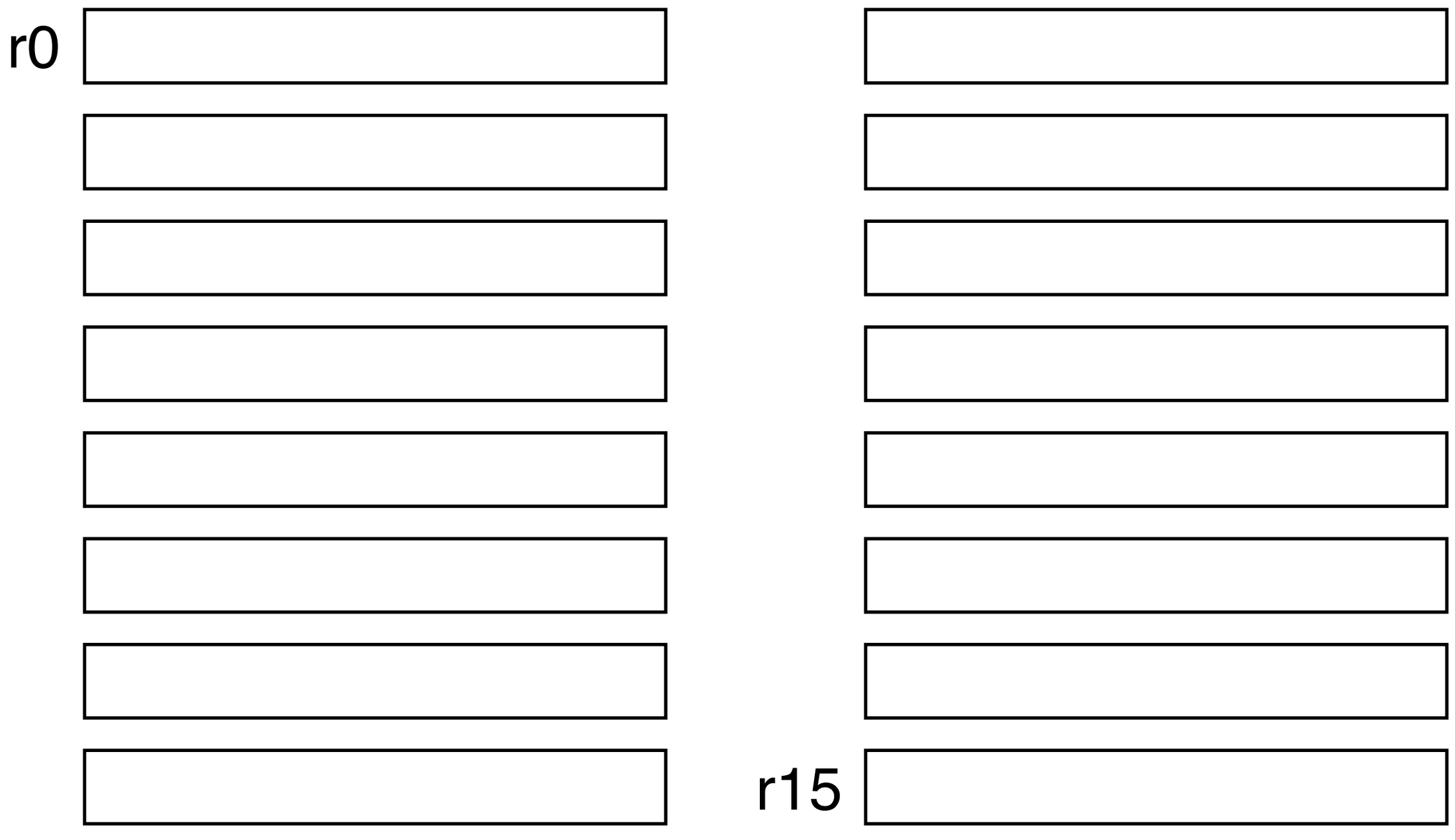

Instruction
Register
Literal
SET(r1, 1)
SET(r2, 2)
MULT(r1, r1, r2)
Target


Instruction
Register
Literal
SET(r1, 1)
SET(r2, 2)
MULT(r1, r1, r2)
Target
Operands

HERA has 16 16-bit general purpose registers

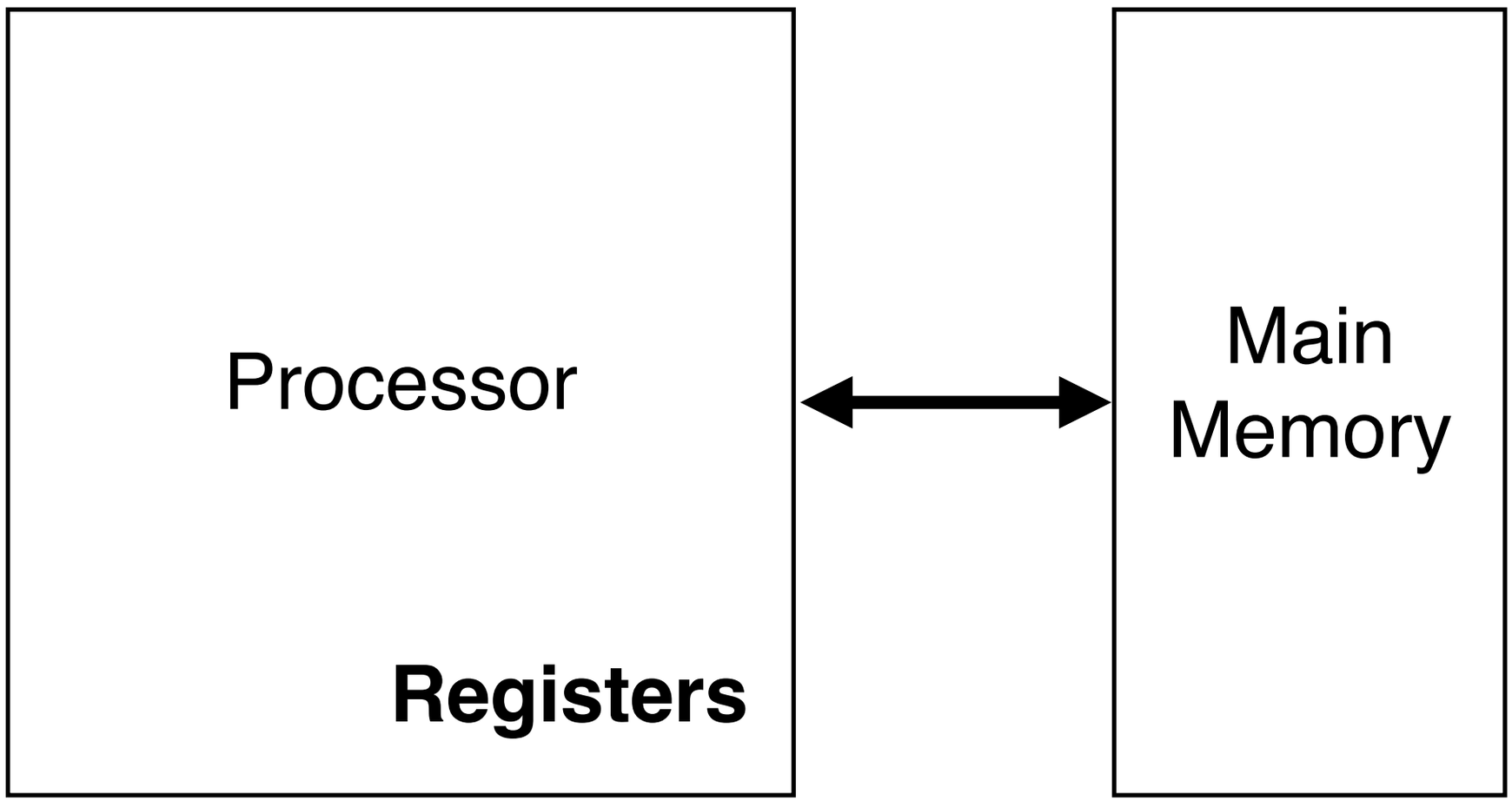
You use these like variables in C

16-bits wide



On a real processor, registers are the temporary variables

They are **not** the main memory



Most processors only have a *very small* number of registers (tens)

The Haverford Educational RISC Architecture

Table of contents

List of figures 2

1 Purpose and History 3

2 Architecture Overview: Registers, Flags, Memory Size 3

3 Instruction Set 4

3.1 Arithmetic, Shift, and Logical Instructions ($b_{15}=1$, $b_{15:12}=0011$) 4

3.1.1 SETLO and SETHI ($b_{15:13}=111$) 4

3.1.2 Three-address operations ($b_{15:13}=110$, 101, or 100) 4

3.1.3 Shifts, increments, and flag operations ($b_{15:12}=0011$) 5

Shifts 5

Set/clear flags 5

Save/restore flags 6

Increments 6

3.2 Memory Instructions ($b_{15:14}=01$) 6

3.3 Control-Flow and Other Instructions ($b_{15:14}=00$) 7

3.3.1 Branches, including jumps ($b_{15:13}=000$) 7

3.3.2 Function call and return; Interrupt processing ($b_{15:12}=0010$) 8

4 Assembly Language Conventions and Pseudo-Operations 9

5 Idioms 10

5.1 Single-Precision Arithmetic 10

5.2 Double-Precision or Mixed-Precision Arithmetic 10

5.3 Control Flow and Branch Instructions 11

5.4 Data Memory and Memory Instructions 12

5.4.1 Arrays and Address Arithmetic 12

5.4.2 Characters and Strings 15

5.5 Function Calls 16

5.5.1 Function Calls with Parameters in Registers, "Caller-Save" of Registers 18

Arithmetic

3.1.2 Three-address operations ($b_{15:13}$=110, 101, or 100)

The three-address HERA operations are

$b_{15:12}$	Mnemonic	Meaning	Notes
1000	AND(d, a,b)	$R_d(i) \leftarrow R_a(i) \wedge R_b(i)$	bit-wise logical and
1001	OR(d, a,b)	$R_d(i) \leftarrow R_a(i) \vee R_b(i)$	bit-wise logical or
1010	ADD(d, a,b)	$R_d \leftarrow R_a + R_b + (c \wedge F_4')$	use carry unless blocked
1011	SUB(d, a,b)	$R_d \leftarrow R_a - R_b - (c' \wedge F_4')$	use carry unless blocked
1100	MULT(d, a,b)	$R_d \leftarrow (R_a * R_b)_{15:0}$, $R_t \leftarrow (R_a * R_b)_{31:16}$	*signed* multiplication
1101	XOR(d, a,b)	$R_d \leftarrow R_a \oplus R_b$	bit-wise exclusive or

These operations all modify the zero flag (true if and only if the result of the operation was zero) and sign flag (true iff b_{15} of the result is true). Addition and subtraction modify the overflow flag and carry flag. Unless the carry-block flag is true, addition and subtraction use the carry flag as the incoming carry. If carry-block is true, addition and subtraction ignore the carry flag (carry-in is 0 for addition and 1 for subtraction).

Memory

(More on this later..)

Control Flow

3.3 Control-Flow and Other Instructions ($b_{15:14} = 00$)

Control-flow instructions include conditional branches, unconditional branches ("jump" instruction), and function and interrupt instructions.

3.3.1 Branches, including jumps ($b_{15:13} = 000$)

HERA provides the following branches that transfer to an address in a register (b) and relative branches that transfer to the current positions plus an 8-bit signed offset (o). Relative branches are distinguished in assembly language by an appended "R" in the operation name.

$b_{15:12}$	$b_{11:8}$	Mnemonic	Meaning
0001/0	0000	BR(b)/BRR(o)	Unconditional branch – $true$
0001/0	0001		(*unused*)
0001/0	0010	BL(b)/BLR(o)	Branch if signed result <0 — $(s \oplus v)$
0001/0	0011	BGE(b)/BGER(o)	Branch if signed result $\geqslant 0$ — $(s \oplus v)'$
0001/0	0100	BLE(b)/BLER(o)	Branch if signed result $\leqslant 0$ — $((s \oplus v) \vee z)$
0001/0	0101	BG(b)/BGR(o)	Branch if signed result >0 — $((s \oplus v) \vee z)'$
0001/0	0110	BULE(b)/BULER(o)	Branch if unsigned result $\leqslant 0$ — $(c' \vee z)$
0001/0	0111	BUG(b)/BUGR(o)	Branch if unsigned result >0 — $(c' \vee z)'$
0001/0	1000	BZ(b)/BZR(o)	Branch if zero — z (if CMP operands $=$)
0001/0	1001	BNZ(b)/BNZR(o)	Branch if not zero — z' (if operands $\neq$)
0001/0	1010	BC(b)/BCR(o)	Branch if carry — c (unsigned result $\geqslant 0$)
0001/0	1011	BNC(b)/BNCR(o)	Branch if not carry — c' (unsigned <0)
0001/0	1100	BS(b)/BSR(o)	Branch if sign (negative) — s
0001/0	1101	BNS(b)/BNSR(o)	Branch if not sign (non-negative) — s'
0001/0	1110	BV(b)/BVR(o)	Branch if overflow — v
0001/0	1111	BNV(b)/BNVR(o)	Branch if not overflow — v'