RASM v2.4

(different version of RASM mean that the hardware is different as well)
RASM has 8 bytes per instruction. The function of the bytes is as followed:

0-1. instruction (opcode).

2-3. source for the first operand (*src0*).

4-5. source for the second operand (*src1*).

6-7. Destination for the result (*dst*).

The notation for *src*/*dst* is as follows:

	source	destination
a	The value a	store at a
$<a>$	The value at register a	store at the value in register a (store at $<a>$)
$[a]$	Option to choose between a and $<a>$	Option to choose between a and $<a>$

Some instructions start with a, b and/or c (i.e., *abc01000*). This signifies where the inputs come from and where the result is stored.

a	=0	First operand is from the register where *src0* points. ($<src0>$)
	=1	First operand is an immediate with value *src0*.
b	=0	second operand is from the register where *src1* points. ($<src1>$)
	=1	second operand is an immediate with value *src1*.
c	=0	Result is saved at the register that is given by register *dst*. ($<<dst>>$)
	=1	Result is saved at the register *dst*. ($<dst>$)

Mnemonics aren't case sensitive, but variables assigned with *const* or *label* are.

There are 8 categories of opcodes (leading bits that don't matter are omitted and are otherwise denoted by '-'):

1. signed jumps (*xxx00xxx*).

2. (bit-wise) logic (*xxx01xxx*).

3. arithmetic (*xxx10xxx*).

4. bit shift (*xxx11xxx*).

5. unsigned jumps (*1 xxx00xxx*).

6. floating point arithmetic (*10 xxx10xxx*).

7. floating point conditionals (*10 xxx11xxx*).

8. *call*, *ret*, *copyptr*, *halt* & exit (*x1xx xxxxxxxx*).

9. stack (*xxx11xxx*).

All the following x, y and z are 16bit.
signed jumps

mnemonic	description	opcode
JumpGt [x] [y] [z]	If $x > y$, jump to z	*abc0 0001*
JumpEq [x] [y] [z]	If $x = y$, jump to z	*abc0 0010*
JumpGE [x] [y] [z]	If $x \geq y$, jump to z	*abc0 0011*
JumpLt [x] [y] [z]	If $x < y$, jump to z	*abc0 0100*
JumpNE [x] [y] [z]	If $x \neq y$, jump to z	*abc0 0101*
JumpLE [x] [y] [z]	If $x \leq y$, jump to z	*abc0 0110*
Jump [z]	jump to z	*abc0 0111*

abc0 0000 also exists but is useless, because the last three bits mean respectively: less than, equal and greater than. So *abc0 0000* would mean: jump if not less than AND not equal AND not greater than.

logic

mnemonic	description	opcode
and [x] [y] [z]	$x\&y \to z$	*abc0 1001*
or [x] [y] [z]	$x\|y \to z$	*abc0 1010*
nand [x] [y] [z]	$\neg(x\&y) \to z$	*abc0 1011*
nor [x] [y] [z]	$\neg(x\|y) \to z$	*abc0 1100*
xor [x] [y] [z]	Checks if the bit with index y in x is equal to 1. $x_y \to z$	*abc0 1101*
xnor [x] [y] [z]	If $x \leq y$, jump to z	*a0c0 1110*
test [z]	jump to z	*abc0 0111*
not [x] [z]	$\neg x \to y$	*a1c0 1111*
high [x] [z]	Stores the index of the highest bit in z. If there is no 1 then it stores 16 (0x10000) (Which is wrong because it should store 32).	*abc0 1111*

arithmetic

mnemonic	description	opcode
add [x] [y] [z]	$x + y \rightarrow z$ (set carry flag when overflowed)	*abc0 1001*
addc [x] [y] [z]	$x + y + carry \rightarrow z$ (set carry flag when overflowed)	*abc0 1010*
sub [x] [y] [z]	$x - y \rightarrow z$ (set carry flag when overflowed)	*abc0 1011*
subb [x] [y] [z]	$x - y - carry \rightarrow z$ (set carry flag when overflowed)	*abc0 1100*
mult [x] [y] [z]	$x * y \rightarrow z$ (set carry with overflow)	*abc0 1101*
div [x] [y] [z]	$x * y \rightarrow z$ ($x\ mod\ y \rightarrow carry$)	*a-c0 1110*
inc [z]	$x + 1 \rightarrow z$	*abc0 0111*
dec [x] [z]	$x - 1 \rightarrow z$	*a-c0 1111*

bit shift

mnemonic	description	opcode
shl [x] [y] [z]	Shift x, y places to the left $\rightarrow z$	*abc0 1001*
shlc [x] [y] [z]	Shift x,y places to the left and shift carry into the gap $\rightarrow z$	*abc0 1010*
shr [x] [y] [z]	Shift x, y places to the right $\rightarrow z$	*abc0 1011*
shrc [x] [y] [z]	Shift x,y places to the right and shift carry into the gap	*abc0 1100*
rotl [x] [y] [z]	Rotate x, y places to the left $\rightarrow z$	*abc0 1101*
rotr [x] [y] [z]	Rotate x, y places to the right $\rightarrow z$	*a-c0 1110*
copy [x] [z]	gets assembled to $x + 0 \rightarrow z$	*a-c1 0000*
exec [x]	NOT IMPLEMENTED	*a101 1111*

unsigned jumps (RASM v2.1)

mnemonic	description	opcode
uJumpGt [x] [y] [z]	If unsigned $x > y$, jump to z	*1 abc0 0001*
uJumpEq [x] [y] [z]	If unsigned $x = y$, jump to z	*1 abc0 0010*
uJumpGE [x] [y] [z]	If unsigned $x \geq y$, jump to z	*1 abc0 0011*
uJumpLt [x] [y] [z]	If unsigned $x < y$, jump to z	*1 abc0 0100*
uJumpNE [x] [y] [z]	If unsigned $x \neq y$, jump to z	*1 abc0 0101*
uJumpLE [x] [y] [z]	If unsigned $x \leq y$, jump to z	*1 abc0 0110*
uJump [x] [z]	jump to($X \| Z$?)	*1 a0c0 0111*

float arithmetic (RASM v2.2)

mnemonic	description	opcode
fadd [x] [y] [z]	$x + y \to z (SETSCARRYATOVERFLOW?)(float)$	*10 abc1 0000*
fsub [x] [y] [z]	$x - y \to z (SETSCARRYATOVERFLOW?)(float)$	*10 abc1 0001*
fmult [x] [y] [z]	$x * y \to z (SETSCARRYATOVERFLOW?)(float)$	*10 abc1 0010*
fdiv [x] [y] [z]	$x/y \to z (SETSCARRYATOVERFLOW?)(float)$	*10 abc1 0011*
ftoint [x] [z]	$int(x) \to z$	*10 abc1 0100*
inttof [x] [z]	$float(x) \to z$	*10 abc1 0101*
fceil [x] [z]	$ceil(x) \to z$ (z *is an integer*)	*10 abc1 0110*
fround [x] [z]	$round(x) \to z$ (z *is an integer*)	*10 abc1 0111*

unsigned jumps (RASM v2.2)

mnemonic	description	opcode
fJumpGt [x] [y] [z]	(float) If $x > y$, jump to z	*10 abc0 0001*
fJumpEq [x] [y] [z]	(float) If $x = y$, jump to z	*10 abc0 0010*
fJumpGE [x] [y] [z]	(float) If $x \geq y$, jump to z	*10 abc0 0011*
fJumpLt [x] [y] [z]	(float) If $x < y$, jump to z	*10 abc0 0100*
fJumpNE [x] [y] [z]	(float) If $x \neq y$, jump to z	*10 abc0 0101*
fJumpLE [x] [y] [z]	(float) If $x \leq y$, jump to z	*10 abc0 0110*
fJump [x][z]	jump to ($X \| Z?$)	*10 a0c0 0111*
exit	stops the cpu	*110 — —0*
hlt [x] [y] [z]	stop until next interrupt	*110 — —1*
copyPtr [x][z]	copy <x> to y (thus copyPtr <x>is equivalent to copy <<x>> to y)	*10 a0c0 0111*

stack (RASM v2.3)

mnemonic	description	opcode
push [x]	Increment stack pointer and then push x to stack	*a001 1111*
pop [x]	Pop from stack to x and then decrement stack pointer	*a011 1111*
concat [x] [y] [z]	Concatenate the lowest 2bytes from [x] and [y] $\to$ z	*abc1 1110*

Due to how the computer is wired it is possible to make every comparison depend on the interrupt but that isn't implemented in the assembler.

addresses:

0-31. registers

32-34. input/output

35. carry/output

32768 (0x8000). read: read from where the RAM pointer is set to. write: set RAM pointer. (*ONLY* v2.1-v.2.3)

40960-49151 (0xA000-0xBFFF). write to RAM. (*ONLY* v2.1-v2.3)

37-16777216 (0x25-0xFFFFFF). read/write to RAM (v2.4)

49152 (0xC000). (write only) counter (v2.1-v2.3)

36 (0x24). (write only?) counter (v2.4)

it's important to remember that immediate values only go up to 16bit so accessing RAM should be done via a pointer.

Const {name} {value}: Every .ovt program can start with a block of lines that start with 'const'. The assembler will replace every occurence of name with value.

Label {name}: assigns the instruction number of the next instruction to {name}.

everything after ';' is seen as a comment (until a new line).