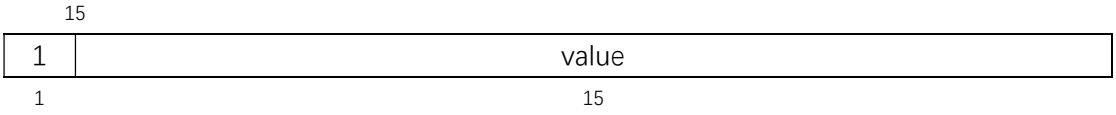
run-forth architecture

1. Assembly and Instruction Set

Run-forth uses a compact 16-bit instruction set that fits the structure of a stack machine. Types of instructions are shown below.

imm



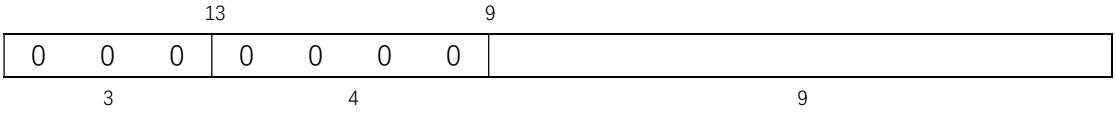
Example: imm 15

Push the immediate value into stack.

15-bit immediate value is supported and zero extended. To use full 16-bit value in run-forth extra instructions must be used.

Run-forth supports 16-bit integer only.

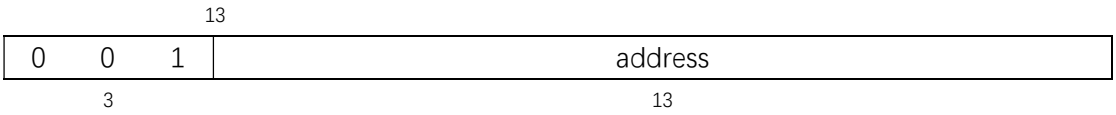
jr



Example: jr

Set PC to the address on the top of address stack, then pop from the address stack.

j

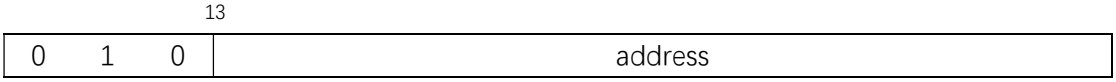


Example: j L0

Set PC to the address given.

An address space of at most 2^{13} instructions is supported.

jal

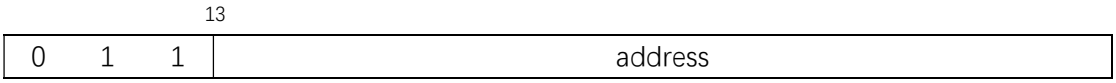


Example: jal L0

Push PC to address stack, then set PC to the address given.

An address space of at most 2^{13} instructions is supported.

jz

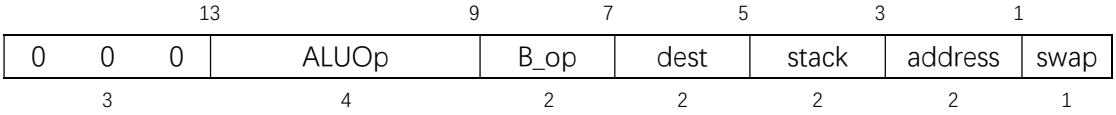


Example: jz L0

Set PC to the address given if TOS is zero.

An address space of at most 2^{13} instructions is supported.

ALU instruction



Example: movb _ T 1 0 1 #dup

Calculate the ALU result by TOS and a specified B value, move the stack pointer of both stacks according to the given offset, then store the result to the given destination.

Here we specify that the two operands are A and B: A is TOS and B is specified by B_op. Calculation is given in C-like representation.

Stack offset and address stack offset are both ranged between -1 and 1.

If swap field is set, the operands are swapped.

Calculation is specified by ALUOp field. Each kind of ALUOp is mapped to an assembly instruction.

Note that due to some sequential problem currently it is not allowed to write to a stack if the the instruction pop the stack as well and the value to be written depends on the value in the stack.

Meaning of B_op is given below.

Value	0	1	2	3
Position	PC+2	N	R	[T]

In the assembly, an underline can be used in B_op field if there's no need to specify the B. In this case, B_op will be N.

Meaning of dest is given below.

Value	0	1	2	3
Position	T	N	R	[T]

Meaning of ALUOp is given below.

Value	Calculation	Assembly
0	(reserved for jr)	
1	dest = A + B	add
2	dest = A - B	sub
3	dest = A * B	mul
4	dest = A / B	div
5	dest = A % B	mod
6	dest = A & B	and
7	dest = A | B	or
8	dest = A ^ B	xor
9	dest = ~B	not
10	dest = B	movb
11	dest = A << B	sll
12	dest = A >> B	srl
13	dest = A < B	slt
14	dest = A >= B	sge
15	dest = A == B	seq

2. Memory

Two stacks, both sized 256 * 2B are provided as data and return stack respectively. TOS, NOS and R can be accessed in the program.

Instructions and data use separated address spaces. Instructions are stored in an instruction memory of 4096 * 2B. The memory is further divided: the lower half is used to store the "text" and the entry point of the program is set to 0. The higher half, started from 2048, is used to store the dictionary of the program, i.e. definition of words. Data are stored in a data memory of 32768 * 2B. Both memory accesses will be aligned to 2B.

3. Compiler, Assembler and run-forth language

The compiler supports only a basic portion of Forth language. Some words are implemented inside the compiler, including word definition, integer literal, comments and:

```
>r, r>, r@, if, else, then, begin, until, do, loop
```

Some other words are implemented in run-forth assembly, including:

`dup, ., swap, drop, nip, rot, !, @, <, >, >=, <=, =, negate, abs, max, min, +, -, *, /, mod, and, or, xor, invert, lshift, rshift`

Memory access approaches including variables and arrays are not supported. To access memory, words `!, @` can be used. `!` stores NOS to [TOS], while `@` push [T] to data stack.

All words must be lower case.

Words except the compiler inherent ones can be redefined and shadowed. A call to the word will be dispatched to its nearest definition, even after it. Custom defined words should not be started with double underline ("__").

Compiler will translate the run-forth code to run-forth assembly. Each assembly is a sequence of .text sections and .dict sections. Then assembler will further translate the assembly to 2 files of machine code: .text file .dict file, each is the concatenation of the machine code of the corresponding section, which could be loaded by the processor.