

# Stupid-RISC Instruction Set

*Version: 0.0.3*



# Table of Contents

- General
- Number Systems
- Functions & Constants
- Comments
- Start Function
- ALU

# General

## INSTRUCTION ENCODING

Any instruction might have up to two arguments. The instruction code itself is 1 byte long. Each one of the arguments also has a 1 byte size. Though some instructions like jmp or halt might only require 0 or one argument/s, each of those is still 3 bytes long.

## CALLING RULES

For system instructions, both lowercase & uppercase can be used. For self-created functions, the capitalisation matters.

## REGISTERS

Any number in the assembly code can't be outside the range of  $0 \dots 2^7 - 1$ . The encoding is as following:

### Register:

Register?	0 bit	0 bit	0 bit	Reg select	Reg select	Reg select	Reg select
1	0	0	0	0	0	1	0

*This selects register 2. A register in assembly is represented with  $R_n$  where  $n$  is between  $0 \dots 11$ . Special registers are called by there names.*

### UInt7 Selection

Register?	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
0	0	0	0	0	0	1	0

*This outputs the number 2. To select negative numbers or other numbers outside this range, it'll need to be loaded from the data section or be created with arithmetic. A number in assembly is represented with  $N_n$ . Where  $n$  is within  $0 \dots 127$  in decimal &  $N$  is the number system.*

# Number Systems

## Overview

Number systems are marked with a starting character like x or no starting character if decimal. If uppercase versions of the number-system-selecting-character exist, those can be uppercase and lowercase.

All numbers will be converted to decimal during the assembly process.

## Limits

As the most-significant-bit is used for register definitions, the limit upper limit is  $2^7-1$  which is 127 (decimal) for numbers in the text section. Negative numbers can't be represented in the text section either, since only the least significant 7 bits can be set.

Define numbers out of bounds in the data section.

## Decimal

Decimal requires no prefix. In disassembly, „N“ or „n“ might be used to represent any number.

## Registers

Registers have a minimum value of 128, and are usually called with the r/R prefix, which will add 128 automatically. The value after the r is decimal.

## ASCII Characters

ASCII characters are supported with the prefix ' followed by the character and the same suffix. Only the first character in this sequence will have an effect. Whitespaces are currently not supported in the encoding process, so x20 or '\_' might be used instead.

## Other Number Systems

System Prefix	System
x	Hexadecimal (Base 16)
o	Octal (base 8)
b	Binary (base 2)

## Functions and constants

Constant definitions always start with a `.`, which is not a part of the constant name. The dot is followed by the name of the constant. There can't be any space between the dot and the name. The name is followed by the value separated by an in line whitespace character.

Here's an example constant definition:

```
.some_constant x0f
```

Later this might be used like:

```
add R0 some_constant
```

Functions are good for jumping somewhere in the code. They are defined by the functions name and a `:` afterwards. The function name will then point to the start of the codeblock afterwards. Functions usually have an indentation for the function blocks.

Example:

```
loop:
```

```
    jmp loop
```

*This loops forever.*

# Comments

Comments escape everything afterwards within the same line. They are defined with a hashtag (#) like in python.

Examples:

```
# This is a very interesting comment
```

```
inc R6 # increments i
```

## Start function

The start function is at the very top of the code and will be executed first (most of the time). It must also be defined with the `.global_start` label/constant, which will ensure the function is at the top if not the case already.



# ALU

The ALU is divided into three parts:

- Integer ALU (addi, etc) (having the ALU code 00)
- General Logic ALU (not, etc) (having the ALU code 01)
- Floating Point & Double ALU (addf, etc) (having the ALU code 10)

On the CPU level, arguments have 32 because they've already passed through the register selectors. In assembly, this are 8 bits per register (1 means register, selector follows afterwards).

An example might be: *add R1, R2* which would store  $R1 + R2$  in R1. Read the registers documentation for more information on this topic.

## INTEGER ALU PART INSTRUCTION SELECTORS:

Operation	ALU Part instr. select. bit code	Description	Impl. Yet	Sim Impl. y.
<b>add</b>	01000000	Adds two numbers and stores them in the selected output register.	0	1
<b>mul</b>	01000010	Multiplies two numbers and stores them in the selected output register.	0	1
<b>sub</b>	01000001	Subtracts the second number from the first one and stores the output in the selected output register.	0	1
<b>div</b>	01000011	Same as above with division.	0	1
<b>mod</b>	01000100	Same as above with modulo.	0	1

*The integer ALU part instruction selector is very simple, the other ones are more complex though.*

## GENERAL LOGIC ALU PART INSTRUCTION SELECTORS:

Operation	ALU Part instr. select. Bit code	Description	Impl. Yet	Sim Impl. y.
<b>or</b>	Not Defined	Performs a logical or.	0	0
<b>and</b>	Not Defined	Performs a logical and.	0	0
<b>xor</b>	Not Defined	Performs a logical exclusive or.	0	0

## FLOATING POINT AND DOUBLE ALU PART INSTRUCTION SELECTORS:

*This entire part is subject to change and will be reformed soon.*

Operation	ALU Part instr. select. Bit code	Description	Impl. Yet
<b>Tfl</b>	Not Defined	To float (converts an integer into a 64 bit floating point number). Subject to change.	0
<b>Tint</b>	Not Defined	To int (converts a float into an integer). Subject to change.	0
<b>Tintp</b>	Not Defined	To int with base (will put the $2^{10}$ bit at bit 32 in the integer with an offset of the given base, which is required to add, etc, as the result can be converted back later). Subject to change.	0
<b>Exp</b>	Not Defined	Gets the exponent of the float/double. Subject to change.	0
<b>Sep</b>	Not Defined	Set exponent (modifies the exponent). Subject to change.	0
<b>num</b>	Not Defined	Gets the value excluding the exponent. Subject to change.	0
<b>snum</b>	Not Defined	Sets the value excluding the exponent. Subject to change.	0
<b>intfltfl</b>	Not Defined	Int float to float converts an integer float from above back to a normal float where argument 0 is the int, argument 1 is the original offset and argument 2 is the destination. Subject to change.	0

*Math operations can't be done in the floating point / double state. The numbers would need to be converted for this. Example code adding two floats which are in R0 and R1 & storing the result in R0:*

*exp R0 R2;; Store R0's exponent in R2*

*tintp R0 R2 R0;; Store the value of R0 in R0 but as an integer as shown above.*

*tintp R1 R2 R1;; Store the value of R1 in R1 using an offset making it directly addable as shown above.*

*addi R0 R1 R0;; Perform the addition and store in R0*

*intfltfl R0 R2 R0;; Convert back to float using the original offset and store in R0;; done.*