Ahf\_asm manual for ahf\_RISC521

By: Alexandre Hernandez Froio

Version 0.0.1

**Requirements to run ahf\_asm:**

* Python 3
* Commandline/terminal
* A .asm or similarly formatted assembly code file

**How to run ahf\_asm:**

1. Open your terminal/command line
2. Navigate to the folder where ahf\_asm.py is
3. Make sure the file you want to assemble is in the same folder.
4. Type in python ahf\_asm.py <assembly file name>
5. Rom1.mif and rom2.mif files are output when successful.

Experimental still:

python ahf\_asm.py <assembly file name> <OpCode Json file name>

**Current Error catches:**

* Checks for the size of inputs
* Checks if register referenced isn’t out of range
* Checks if too many variables are used on a certain line
* If it cannot parse a line, it quits and tells you what line caused the error

**Current Sections accepted:**

* Directives should be between .directive and .enddirective, everything in these should be denoted as .equ values, and outputs them to a ram.mif file that can be used to initialize the device’s RAM.
* Constants should be between .constants and .endconstants, everything in these should be denoted as .word values, currently saves these to the assembler memory, but not yet functional for use.
* Assembly code should be preceded with .code and ended with .endcode

**Addressing modes used in syntaxes:**

0xi - direct address, can go from 0000 to 3fff

i,j - constant, can also be expressed as 0xi, cannot be bigger than 19h

Ri/Rj/Rk - Register mode, only 32 registers available, from 0 to 31

label - any label declared by @<labelname>:

**Accepted commands and their syntaxes:**

LD - 0xi/i(Rk), Rj - takes indexed or direct addressing to load to Rj

ST - 0xi/i(Rk), Rj - takes indexed or direct addressing to store to Rj

CPY - Ri, Rj - Copy from Rj into Ri

SWAP - Ri, Rj - Swap values between Ri and Rj

ADD - Ri, Rj - Add Rj to Ri, store to Ri

SUB - Ri, Rj - Subtract Rj from Ri, store to Ri

ADDC - Ri, j - add j constant to Ri, store to Ri

SUBC - Ri, j - subtract j constant from Ri, store to Ri

NOT - Ri - logical not Ri, store to Ri

AND - Ri, Rj - Logical and between Ri and Rj, store to Ri

OR - Ri, Rj - Logical or between Ri and Rj, store to Ri

SHRA - Ri, j - shift Ri right by j (can be negative), store to Ri

ROTR - Ri, j - Rotate through carry Ri by j (can be negative), store to Ri

JMP - label - jump unconditionally to label

JMPC - label - jump if carry to label

JMPNC - label - jump if no carry to label

JMPN - label - jump if negative to label

JMPNN - label - jump if not negative to label

JMPV - label - jump if overflow to label

JMPNV - label - jump if no overflow to label

JMPZ - label - jump if Zero to label

JMPNZ - label - jump if not zero to label

@<labelname>: - reserves a label for use w/jumps, **do not** write commands after instantiating a label **on the same line.**

**Coding example:**

.code;

@start:

SUBC R2, 0x02;

SUBC 0x02, R5;

ST -20(R3), R2;

LD 0x04, R5;

ADDC R5, 0x19;

ADD R16, R15;

ADDC R18, 0x12;

SUB R5, R5;

JMPC uploop;

OR R5, R6;

@uploop:

NOT R5;

CPY R7, R5;

SWAP R7, R5;

NOT R7;

SHRA R5, 0x3;

ROTR R6, 0x1;

ROTR R6, 0x1;

CPY R8, R7;

**Planned improvements:**

* Remove most of the superfluous debug print messages in the code, make assembler output just what is being printed to the mif files
* Improve Regex filters on some of the mnemonics
* Enable user-made Opcodes inputs and implement emulated OpCodes.