Instructions

- load (ld)

  Can only be used to access memory.

  ld 4, R4→ copies value from memory address 4 to R4

  ld R3, R4→ copies value from memory address value of R3 to R4

- move (mov)

  Can only be used between registers or to store values in registers.

  mov 4, R4→ moves the value 4 to R4. mov R2, R4→ copies the value in R2 to R4. mov 1, [R1,4]→ this moves the value 1 to the space the Register 1 has on memory for arrays with an offset of 4 (because they are all integers this offset would mean 4 bits). mov 2, [R1,R2]→ this moves the value 2 to the space the Register 1 has on memory for arrays with an offset of the value saved in register 2. mov [R1, 8], R5→ This moves the value from the array at R1 with offset 8 to the register 5. mov [R1, R2], R6→ This moves the value from the array at R1 with offset as the value contained in R2 to the register 5.

- store (str)

  str 4, R4→ stores value from R4 to memory address 4.

  str R3, R4→ stores value from R4 to memory address at R3.

- add (add)

  Has 3 registers. First one and second one are the operands, the last one is where the result will be saved.

  add R4, R3, R4→ takes values from R4 and R3, sums them and saves the result in R4.

- subtract (sub)

  Uses 3 registers. First one and second one are the operands, the last one is where the result will be saved.

  sub R4, R3, R4→ takes values from R4 and R3, subtracts them and saves the result in R4.

- Multiply (mult)

  Uses 3 registers. First one and second one are the operands, the last one is where the result will be saved.

  mult R4, R3, R4→ takes values from R4 and R3, multiplies them and saves the result in R4.

- compare (cmp) Takes two registers and compares the values in them, sets a flag to the value after substracting the two.

cmp R2, R1→ takes values from R2 and R1, computes the subtraction (value in R2 - value in R1) and the result is saved on the flag.

- jump (jmp)

  Instruction that branches to a section.

```
loop:
    ...some code...
    jmp loop
```

- jump equal (je)

  Instruction that branches to a section only if the compare flag is set to 0.

```
loop:
    ...some code...
    cmp R2,R3
    je loop
```

- jump above (ja)

  Instruction that branches to a section only if the compare flag is set to a number greater than 0.

```
loop:
    ...some code...
    cmp R2,R3
    ja loop
```

- jump below (jb)

  Instruction that branches to a section only if the compare flag is set to a number smaller than 0.

```
loop:
    ...some code...
    cmp R2,R3
    jb loop
```

- call for recursive algorithms(call) This instruction is to make recursive algorithms. It's still on the works.

- end (end)

  Instruction set at the end of the code to let the simulator know all instructions have ended.