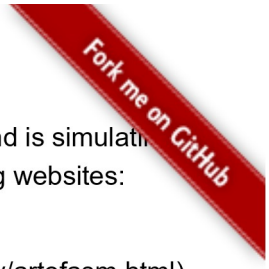
(https://github.com/Schweigi/assembler-simulator)



Introduction

This simulator provides a simplified assembler syntax (based on NASM (http://www.nasm.us)) and is simulati
x86 like cpu. In depth documentation and introduction to assembler can be found on the following websites:

- Assembly - Wikipedia (http://en.wikipedia.org/wiki/Assembly_language)
- The Art of Assembly Language Programming (http://cs.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html)
- NASM Language Documentation (http://www.nasm.us/xdoc/2.10.09/html/nasmdoc3.html)

The simulator consists of a 8-bit cpu and 256 bytes of memory. All instructions (code) and variables (data) needs to fit inside the memory. For simplicity every instruction (and operand) is 1 byte. Therefore a MOV instruction will use 3 bytes of memory. The simulator provides a console output which is memory mapped from 0xE8 to 0xFF. Memory mapped means that every value written to this memory block is visible on the console.

Syntax

The syntax is similar as most assemblers are using. Every instruction must be on their own line. Labels are optional and must either start with a letter or a dot (.) and end with a colon.

```
label: instruction operands     ; Comment
```

Valid number formats for constants are:

```
Decimal: 200
Decimal: 200d
Hex: 0xA4
Octal: 0o48
Binary: 101b
```

It is possible to define a number using a character or multiple numbers (see instruction *DB*) by using a string.

```
Character: 'A'
String: "Hello World!"
```

Operands can either be one of the four general purpose registers, stack pointer register, a memory address or a constant. Stack pointer register can only be used as operand in MOV, ADD, SUB, CMP, INC and DEC instructions. Instead of defining an address as a constant or by using a register you can use labels. The assembler will then replace the label with the corresponding constant.

```
General purpose (GP) register: A, B, C, D
Stack pointer register: SP
Address using a GP register: [A]
Address using a GP register and offset: [D-3]
Address using SP register and offset: [SP+2]
Address using a constant: [100]
Address using a label: label
Constant: Any number between 0..255 (8bit unsigned)
Offset for indirect addressing: Integer between -16..+15 (sign is mandatory)
```

MOV - Copy a value

Copies a value from *src* to *dest*. The MOV instruction is the only one able to directly modify the memory. SP can be

used as operand with MOV.

```
MOV reg, reg
MOV reg, address
MOV reg, constant
MOV address, reg
MOV address, constant
```

DB - Variable

Defines a variable. A variable can either be a single number, character or a string.

```
DB constant
```

Math operations

Addition and Subtraction
Adds two numbers together or subtract one number form another. This operations will modify the carry and zero flag. SP can be used as operand with ADD and SUB.

```
ADD reg, reg
ADD reg, address
ADD reg, constant
SUB reg, reg
SUB reg, address
SUB reg, constant
```

Increment and Decrement
Increments or decrements a register by one. This operations will modify the carry and zero flag. SP can be used as operand with INC and DEC.

```
INC reg
DEC reg
```

Multiplication and division
Multiplies or divides the *A* register with the given value. This operations will modify the carry and zero flag.

```
MUL reg
MUL address
MUL constant
DIV reg
DIV address
DIV constant
```

Logical instructions
The following logical instructions are supported: AND, OR, XOR, NOT. This operations will modify the carry and zero flag.

```
AND reg, reg
AND reg, address
AND reg, constant
OR reg, reg
OR reg, address
OR reg, constant
XOR reg, reg
XOR reg, address
XOR reg, constant
NOT reg
```

Shift instructions

The following shift instructions are supported: SHL/SAL and SHR/SAR. As this simulator only supports unsigned numbers SHR and SAR yield the same result. This operations will modify the carry and zero flag.

```
SHL reg, reg
SHL reg, address
SHL reg, constant
SHR reg, reg
SHR reg, address
SHR reg, constant
```

CMP - Compare

Compares two values and sets the zero flag to true if they are equal. SP can be used as operand with CMP. Use this instruction before a conditional jump.

```
CMP reg, reg
CMP reg, address
CMP reg, constant
```

Jumps

JMP - Unconditional jump

Let the instruction pointer do a unconditional jump to the defined address.

```
JMP address
```

Conditional jumps

Let the instruction pointer do a conditional jump to the defined address. See the table below for the available conditions.

Instruction	Description	Condition	Alternatives
JC	Jump if carry	Carry = TRUE	JB, JNAE
JNC	Jump if no carry	Carry = FALSE	JNB, JAE
JZ	Jump if zero	Zero = TRUE	JB, JE
JNZ	Jump if no zero	Zero = FALSE	JNE
JA	>	Carry = FALSE && Zero = FALSE	JNBE

Instruction	Description	Condition	Alternatives
JNBE	not <=	Carry = FALSE && Zero = FALSE	JA
JAE	>=	Carry = FALSE	JNC, JNB
JNB	not <	Carry = FALSE	JNC, JAE
JB	<	Carry = TRUE	JC, JNAE
JNAE	not >=	Carry = TRUE	JC, JB
JBE	<=	C = TRUE or Z = TRUE	JNA
JNA	not >	C = TRUE or Z = TRUE	JBE
JE	=	Z = TRUE	JZ
JNE	!=	Z = FALSE	JNZ

CALL - Function call
Call can be used to jump into a subroutine (function). Pushes the instruction address of the next instruction to the stack and jumps to the specified address.

```
CALL address
```

RET - Exit a subroutine
Exits a subroutines by popping the return address previously pushed by the CALL instruction. Make sure the SP is balanced before calling RET otherwise the instruction pointer will have an ambiguous value.

```
RET
```

Stack instructions

PUSH - Push to stack
Pushes a value to the stack. The stack grows down and the current position is available in the stack pointer register (SP). This instruction will decrease the SP.

```
PUSH reg
PUSH address
PUSH constant
```

POP - Pop from stack
Pops a value from the stack to a register. This instruction will increase the SP.

```
POP reg
```

Other instructions

HLT - Stops the processor.
Stops operation of the processor. Hit Reset button to reset IP before restarting.

```
HLT
```

by Marco Schweighauser (2015) | MIT License | Blog (https://www.mschweighauser.com/make-your-own-assembler-simulator-in-javascript-part1/)