Contents

1 VSASM Architecture Reference

1.1 Overview

VSASM (Very Simple Assembly) is a simplified 8-bit instruction set architecture designed for educational purposes. It follows a Von Neumann architecture where both instructions and data share the same memory space.

1.2 System Specifications

- **Word Size**: 8-bit (1 Byte)
- **Memory**: 256 Bytes (Addresses 0x00 - 0xFF)
- **General Purpose Registers**: 14 known as a through n.
- **Special Registers**:
  - sp (Stack Pointer): Special register to use as the next free stack address;
  - bp (Base Pointer): Special register to use as current stack frame anchor;
  - pc (Program Counter): Holds the address of the next instruction;
  - FLAGS: Status register (Zero, Negative).

1.3 Registers

Register	Index	Description
a - n	0-13	General Purpose Registers (GPR). Used for arithmetic and data manipulation.
sp	14	Stack Pointer. Points to the top of the stack.

Register	Index	Description
bp	15	Base Pointer. Used to reference function arguments and local variables.

1.4 Memory Map

- **0x00 - ...**: Program code starts at address 0.
- **... - 0xFF**: Stack starts at the end of memory and grows towards 0.
- **Heap**: There is no dedicated heap region; the programmer must manage free memory between the code and stack.

1.5 Instruction Set Table

Arguments: `r` = register (e.g. `a`), `val` = immediate value (e.g. `10`), `addr` = memory address.

Opcode	Mnemonic	Arguments	Operation	Description
0	HALT	-	Stop	Stops the execution of the processor.
1	JMP	%reg	pc <- reg	Unconditional jump to address in register.
2	JMPIFNEG	%reg	if (neg)pc <- reg	Jump to address in register if Negative flag is set.
3	JMPIFZERO	%reg	if (zero)pc <- reg	Jump to address in register if Zero flag is set.
4	READ	%reg	reg <- Input	Read integer from Input Device into register. Updates Flags.
5	SHOW	%reg	Output <- reg	Write integer from register to Output Device.
6	ADD	%r1 %r2	r1 <- r1 + r2	Add value of r2 to r1. Updates Flags.

Opcode	Mnemonic	Arguments	Operation	Description
7	SUB	%r1 %r2	r1 <- r1 - r2	Subtract value of r2 from r1. Updates Flags.
8	NOT	%reg	reg <- ~reg	Bitwise NOT. Updates Flags.
9	AND	%r1 %r2	r1 <- r1 & r2	Bitwise AND. Updates Flags.
10	OR	%r1 %r2	r1 <- r1 | r2	Bitwise OR. Updates Flags.
11	XOR	%r1 %r2	r1 <- r1 ^ r2	Bitwise XOR. Updates Flags.
12	MOV	%r1 %r2	r1 <- r2	Copy value from r2 to r1. Updates Flags.
13	CMP	%r1 %r2	r1 - r2	Compare r1 and r2. Discards result, updates Flags only.
14	PUT	%reg val	reg <- val	Load immediate 8-bit value into register. Updates Flags.
15	LOAD	%reg addr	reg <- Mem[addr]	Load value from direct memory address. Updates Flags.
16	STORE	%reg addr	Mem[addr] <- reg	Store value from register to direct memory address.
17	LOADREG	%r1 %r2	r1 <- Mem[r2]	Load val from address stored in r2 into r1. Updates Flags.
18	STOREREG	%r1 %r2	Mem[r2] <- r1	Store val from r1 into address stored in r2.
19	OFFSET	%reg val	reg <- bp - val	Calculate address relative to BP. Useful for local vars/args. Updates Flags.

Opcode	Mnemonic	Arguments	Operation	Description
20	`LSHIFT`	`%reg val`	`reg <- reg << val`	Logical Left Shift. Updates Flags.
21	`RSHIFT`	`%reg val`	`reg <- reg >> val`	Logical Right Shift. Updates Flags.

1.6 Flags

- **Z (Zero)**: Set if the last value written to register or computed by the ALU is 0.
- **N (Negative)**: Set if the high-bit (bit 7) of the last value written to register or computed by the ALU is 1.

1.7 Addressing Modes

- **Immediate**: `PUT a 10`
- **Register**: `ADD a b`
- **Direct**: `LOAD a 50`
- **Indirect (Register)**: `LOADREG a b`

1.8 Visualizer Legend

- **Green Text**: Register/Memory Write
- **Blue Text**: Register/Memory Read
- **Yellow Background**: Stack Frame (Active function locals)
- **Cyan Border**: Stack Pointer (Top of stack)
- **Pink Background**: Stack contents
- **Blue Background**: Program Code (Static)