VM & ISA

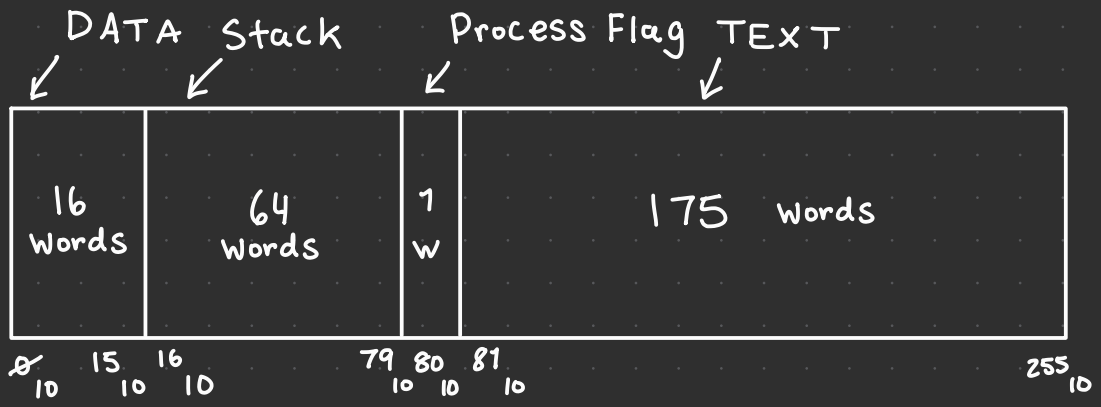
Documentation
& user manual

Virtual Hardware

The hardware's smallest (and largest) unit of memory is a word. A word is 8 bits long.

There consists 4 Registers all 1 word in size. Of these registers there are 2 General Purpose (R0, R1), 1 Stack Pointer (SP), and 1 Program counter (PC).

The memory is 256 words long. It is laid out below:



Instructions

Name	Args	C-Like Effect
MOV	Ra Rb	Ra = Rb
CMP	Ra Rb	Mem(80) = Ra − Rb
SHL	Ra Rb	Ra = Ra << Rb
SHR	Ra Rb	Ra = Ra >> Rb
ADD	Ra Rb	Ra = Ra + Rb
SUB	Ra Rb	Ra = Ra − Rb
AND	Ra Rb	Ra = Ra & Rb
ORR	Ra Rb	Ra = Ra | Rb
NOT	Ra	Ra = ~Ra
PSH	Ra	Mem[SP] = Ra, SP++
POP	Ra	SP--, Ra = Mem[SP]
SYS	Ra	Syscall(Ra, argc, argv)
JMP	mask, Addr	if mem(80) & mask : PC = Addr
LDI	Ra, Imm	Ra = Imm
LDA	Ra, Imm	Ra = mem(Imm)
STA	Ra, Imm	mem(Imm) = Ra

See prev page

LT EQ GT
100 010 001

N = mem[SP] mem[SP−1 .. −N]

JMP is Special

See prev page

Note: Ra, Rb, mask, Imm Serve as placeholders

Instruction Groupings

Type	op code	name
X	0 0 0 0	MOV
	0 0 0 1	CMP
	0 0 1 0	SHL
	0 0 1 1	SHR
X	0 1 0 0	ADD
	0 1 0 1	SUB
	0 1 1 0	AND
	0 1 1 1	ORR
Y	1 0 0 0	NOT
	1 0 0 1	PSH
	1 0 1 0	POP
	1 0 1 1	SYS
Z_J →	1 1 0 0	JMP
Z	1 1 0 1	LDI
	1 1 1 0	LDA
	1 1 1 1	STA

Instruction Types

X: O_3 is $\emptyset$

opcode				R_a		R_b	
O_3	O_2	O_1	$O_\emptyset$	a_1	$a_\emptyset$	b_1	$b_\emptyset$

Y: O_3 is 1 & O_2 is $\emptyset$

opcode				unused		R_a	
O_3	O_2	O_1	$O_\emptyset$	$\emptyset$	$\emptyset$	a_1	$a_\emptyset$

Z: O_3 is 1 & O_2 is 1 BUT: Z_J: Z & $O_1 = O_\emptyset$ is $\emptyset$

opcode				unused		R_a	
O_3	O_2	O_1	$O_\emptyset$	$\emptyset$	$\emptyset$	a_1	$a_\emptyset$

$\emptyset\emptyset R_a$ becomes:

mask			
$\emptyset$	m_2	m_1	$m_\emptyset$

Immmediate

I_7	I_6	I_5	I_4	I_3	I_2	I_1	$I_\emptyset$

more explicitly Z_J: O_3 is 1 & O_2 is 1 & O_1 is $\emptyset$ & $O_\emptyset$ is $\emptyset$

opcode				unused	mask			+ 8-bit Immediate
O_3	O_2	O_1	$O_\emptyset$	$\emptyset$	m_2	m_1	$m_\emptyset$	

Assembler Syntax

asm → data text $

data → ".data" dataList
| λ

dataList → dataItem dataList
| λ

dataItem → [ident] "=" [immediate]

text → ".text" textList
| λ

textList → [ident] ":" textList
| Xinstruction textList
| Yinstruction textList
| Zinstruction textList
| λ

Xinstruction → [XCMD] [Reg] [Reg]

Yinstruction → [YCMD] [Reg]

Zinstruction → [ZCMD] [Reg] "," Zitem
| ZJinstruction

ZJinstruction → [ZJCMD] [Mask] "," Zitem

Zitem → [immediate]
| [ident]

System Calls

0_{16} Sys_Exit

1_{16} sys_sleep

Note about sys_Exit & sys_sleep:

2_{16} sys_Rand

3_{16} sys_out_int

4_{16} sys_out

System Calls Examples

Sys_Exit :

```
LDI R0, 0_10
PSH R0
```

} argv[0] = 0_{10}

```
LDI R0, 0_10   ← Sys_Exit
Sys R0         ← Syscall(R0, ~~argc~~, argv)
```

Dont need to set since exit Takes const Argc of 1

Sys_Exit(ExitCode)

Sys_SLEEP :

```
LDI R0, 50_10
PSH R0
```

} argv[0] = 50_{10}

```
LDI R0, 1_10   ← Sys_sleep
Sys R0         ← Syscall(R0, ~~argc~~, argv)
```

Sys_sleep(TimeMS)

System Calls Examples Pt. 2

Sys_OUT:

```
LDI R0, 21₁₀
PSH R0
```
} argv[2] = 21_{10} (!)

```
LDI R0, 105₁₀
PSH R0
```
} argv[1] = 105_{10} (i)

```
LDI R0, 72₁₀
PSH R0
```
} argv[0] = 72_{10} (H)

```
LDI R0, 3₁₀
PSH R0
```
} argc = 3_{10}

```
LDI R0, 2₁₀
Sys R0
```

← Sys_OUT

← Syscall(R0, argc, argv)

SYS_OUT(Len, STR)

Sys_Rand:

```
LDI R0, 3₁₀
Sys R0
Pop R0
```

← Sys_Rand

Sys_Rand(void)

← Syscall(R0, ~~argc~~, ~~argv~~)

← move Return into R0