1 The Virtual Machine

This is a simplistic, purely register-based RISC[1] Virtual Machine implemented in C++.

1.1 Design & Overview

- Data & Instructions are stored in RAM[2], which is an array of bytes.
- In every cycle, the CU[3] reads the byte specified by the instruction pointer from the RAM and interprets it as an instruction. Depending on the instruction, it also loads up to two additional parameters from the RAM.
- There are mainly three types of instructions:
  - jump instructions, affecting only the instruction counter. Some of them depend on the status of the ALU-flags.
  - ALU instructions (see below),
  - data moving instruction, i. e. instructions that copy data from RAM into a register (or vice-versa) or write a specific value into a register.

  In Addition, there are two special instructions: `NOP` (0x10), which does nothing, and `STP` (0x00) which stops the execution of the VM.
- The ALU[4] can perform the following basic arithmetical and logical operations:
  - add,
  - substract,
  - multiply,
  - integer divide,
  - bitwise shift left,
  - bitwise shift right,
  - bitwise logical and,
  - bitwise logical or.

  For each operations, the operand(s) have to be in register A (and register B, if two operands are needed). The result is stored in register C. The operation does not modify registers A and B.
- The VM is executed strictly sequential and not pipelined.

Figure 1 shows all possible paths of data and instructions.

[1] Reduced Instruction Set Computer
[2] Random Acess Memory
[3] Controll Unit
[4] Arithmetical Logical Unit

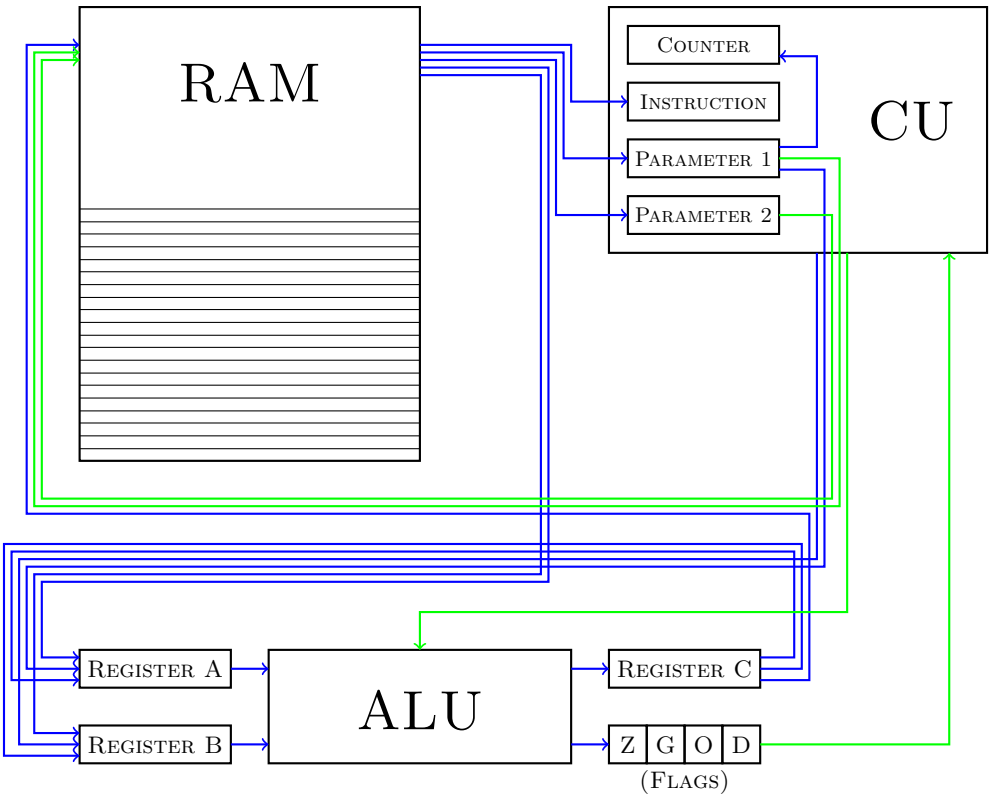

Figure 1: Schematic view of VM. Blue: data, green: control, adresses

1.2 Implementation

The important classes of the implementation are:

- VirtualMachine: handles instances of ram and cpu, and loads programms from txt-files. Also handles the main loop of the simulation.
- Ram: handles the radom-access-memory (which is implemented as a byte-array).
- CentralProcessingUnit: has instances of CU and ALU and handles the computation cycles.
- ControllUnit: handles execution of commands and instruction counter.
- ArithmeticalLogicalUnit: executes arithmetical operations and flags.

1.3 Instruction Set

See table 1.

Table 1: Overview of all Operations supported by the Virtual Machine

Machine Code	**Assembly Command**	**Number of Paramters**	**Describtion**
0x00	STP	0	Stops the execution
0x01	JMP	1	Unconditional jump
0x02	JGZ	1	Jump if C > 0
0x03	JOF	1	Jump if last Operation caused Overflow
0x04	ADD	0	Compute C = A + B
0x05	SUB	0	Compute C = A - B
0x06	AND	0	Compute C = A & B (logical bitwise and)
0x07	BOR	0	Compute C = A | B (logical bitwise or)
0x08	SHL	0	Shift A one bit to the left, stores in C
0x09	SHR	0	Shift A one bit to the right, stores in C
0x0A	LDA	1	Load value from adress (parameter) to A
0x0B	LDB	1	Load value from adress (parameter) to B
0x0C	LDC	1	Load constant value (parameter) to C
0x0D	LD0	0	Store 0 in B
0x0E	STR	1	Store value from C in RAM
0x0F	MOV	2	Copy value from first adress to second adress
0x10	NOP	0	No operation
0x11	–	–	Currently not used
0x12	JEZ	1	Jump if C = 0
0x13	JNO	1	Jump if last operation caused no overflow
0x14	MUL	0	Computes C = A $\cdot$ B
0x15	DIV	0	Computes C = A `div` B (integer division)
0x16	–	–	Currently not used
0x17	–	–	Currently not used
0x18	–	–	Currently not used
0x19	–	–	Currently not used
0x1A	RLA	0	Reload value fom C into A
0x1B	RLB	0	Reload value fom C into B
0x1C	LDM	0	Load maximum value into B
0x1D	LD1	0	Load 1 into B
0x1E	–	–	Currently not used
0x1F	–	–	Currently not used

Table 2: Overview of additional Operations supported by Assembler

Assembly Command	**Number of Paramters**	**Describtion**
EAD	2	C = sum of both parameters
ESU	2	C = first parameter - second parameter
EMU	2	C = product of both parameters
EDI	2	C = first parameter `div` second parameter (integer division)
STC	2	store first parameter at variable given by second parameter in RAM
VAR	1	declares a variable
BEGIN	0	defines the beginning of code (can be used as a jump-label)
LABEL	1	defines a label that can be used as destination for a jump instruction
//	–	comment, lines beginning with // will be ignored by assembler

2 The Assembler

2.1 Implementation

2.2 Instruction Set

See table 2.

2.3 Example Program

Example program `Prim.txt`

```
1  // a test program that checkes if the value in test is prim
2  // writes 1 to res if test is prim
3  // writes 0 to res otherwise
4  VAR test
5  VAR max
6  VAR counter
7  VAR res
8  BEGIN
9  STC 13 test
10 STC 2 counter
11 LDA test
12 SHR
13 // max = test/2, maximum number that needs to be checked
14 STR max
15 LABEL start
16 // check if test/counter has remainder:
17 LDA test
18 LDB counter
19 DIV
20 RLA
21 MUL
22 RLA
23 LDB test
24 SUB
25 // if 0, no remainder: test is not prim
26 JEZ notprim
27 // otherwise increase counter
28 LDA counter
29 LD1
30 ADD
31 STR counter
32 RLB
33 // check if max is reached:
34 LDA max
35 SUB
36 // counter < max: goto start
37 JGZ start
38 // otherwise: test is prim
39 STC 1 res
40 JMP end
41 LABEL notprim
42 STC 0 res
43 LABEL end
44 STP
```

Execution output of VM of program Prim:

Loading program... Starting Virtual Machine...

cycle	ic	instr	src	dest	val	FLAGS	reg A	reg B	reg C
1	0	JMP	–	0XB	–	–––––	62	0	0
2	11	LDC	–	–	0XD	–––––	13	0	0
3	14	LD0	–	–	–	–––––	13	0	0
4	15	ADD	–	–	–	–G–––	13	0	13
5	16	STR	–	0X3	0XD	–G–––	13	0	13
6	19	LDC	–	–	0X2	–G–––	2	0	13
7	22	LD0	–	–	–	–G–––	2	0	13
8	23	ADD	–	–	–	–G–––	2	0	2
9	24	STR	–	0X7	0X2	–G–––	2	0	2
10	27	LDA	0X3	–	0XD	–G–––	13	0	2
11	30	SHR	–	–	–	–G–––	13	0	6
12	31	STR	–	0X5	0X6	–G–––	13	0	6
13	34	LDA	0X3	–	0XD	–G–––	13	0	6
14	37	LDB	0X7	–	0X2	–G–––	13	2	6
15	40	DIV	–	–	–	–G–––	13	2	6
16	41	RLA	–	–	–	–G–––	6	2	6
17	42	MUL	–	–	–	–G–––	6	2	12
18	43	RLA	–	–	–	–G–––	12	2	12
19	44	LDB	0X3	–	0XD	–G–––	12	13	12
20	47	SUB	–	–	–	–GO––	12	13	−1
21	48	JEZ	–	0X4E	–	–GO––	12	13	−1
22	51	LDA	0X7	–	0X2	–GO––	2	13	−1
23	54	LD1	–	–	–	–GO––	2	1	−1
24	55	ADD	–	–	–	–G–––	2	1	3
25	56	STR	–	0X7	0X3	–G–––	2	1	3
26	59	RLB	–	–	–	–G–––	2	3	3
27	60	LDA	0X5	–	0X6	–G–––	6	3	3
28	63	SUB	–	–	–	–G–––	6	3	3
29	64	JGZ	–	0X22	–	–G–––	6	3	3
30	34	LDA	0X3	–	0XD	–G–––	13	3	3
31	37	LDB	0X7	–	0X3	–G–––	13	3	3
32	40	DIV	–	–	–	–G–––	13	3	4
33	41	RLA	–	–	–	–G–––	4	3	4
34	42	MUL	–	–	–	–G–––	4	3	12
35	43	RLA	–	–	–	–G–––	12	3	12
36	44	LDB	0X3	–	0XD	–G–––	12	13	12
37	47	SUB	–	–	–	–GO––	12	13	−1
38	48	JEZ	–	0X4E	–	–GO––	12	13	−1

39	51	LDA	0X7	–	0X3	–GO–	3	13	−1
40	54	LD1	–	–	–	–GO–	3	1	−1
41	55	ADD	–	–	–	–G––	3	1	4
42	56	STR	–	0X7	0X4	–G––	3	1	4
43	59	RLB	–	–	–	–G––	3	4	4
44	60	LDA	0X5	–	0X6	–G––	6	4	4
45	63	SUB	–	–	–	–G––	6	4	2
46	64	JGZ	–	0X22	–	–G––	6	4	2
47	34	LDA	0X3	–	0XD	–G––	13	4	2
48	37	LDB	0X7	–	0X4	–G––	13	4	2
49	40	DIV	–	–	–	–G––	13	4	3
50	41	RLA	–	–	–	–G––	3	4	3
51	42	MUL	–	–	–	–G––	3	4	12
52	43	RLA	–	–	–	–G––	12	4	12
53	44	LDB	0X3	–	0XD	–G––	12	13	12
54	47	SUB	–	–	–	–GO–	12	13	−1
55	48	JEZ	–	0X4E	–	–GO–	12	13	−1
56	51	LDA	0X7	–	0X4	–GO–	4	13	−1
57	54	LD1	–	–	–	–GO–	4	1	−1
58	55	ADD	–	–	–	–G––	4	1	5
59	56	STR	–	0X7	0X5	–G––	4	1	5
60	59	RLB	–	–	–	–G––	4	5	5
61	60	LDA	0X5	–	0X6	–G––	6	5	5
62	63	SUB	–	–	–	–G––	6	5	1
63	64	JGZ	–	0X22	–	–G––	6	5	1
64	34	LDA	0X3	–	0XD	–G––	13	5	1
65	37	LDB	0X7	–	0X5	–G––	13	5	1
66	40	DIV	–	–	–	–G––	13	5	2
67	41	RLA	–	–	–	–G––	2	5	2
68	42	MUL	–	–	–	–G––	2	5	10
69	43	RLA	–	–	–	–G––	10	5	10
70	44	LDB	0X3	–	0XD	–G––	10	13	10
71	47	SUB	–	–	–	–GO–	10	13	−3
72	48	JEZ	–	0X4E	–	–GO–	10	13	−3
73	51	LDA	0X7	–	0X5	–GO–	5	13	−3
74	54	LD1	–	–	–	–GO–	5	1	−3
75	55	ADD	–	–	–	–G––	5	1	6
76	56	STR	–	0X7	0X6	–G––	5	1	6
77	59	RLB	–	–	–	–G––	5	6	6
78	60	LDA	0X5	–	0X6	–G––	6	6	6
79	63	SUB	–	–	–	Z–––	6	6	0
80	64	JGZ	–	0X22	–	Z–––	6	6	0

```
     81 |       67 |   LDC        –          –         0X1 |  Z——  |      1     6     0
     82 |       70 |   LD0        –          –           – |  Z——  |      1     0     0
     83 |       71 |   ADD        –          –           – |  –G—  |      1     0     1
     84 |       72 |   STR        –        0X9         0X1 |  –G—  |      1     0     1
     85 |       75 |   JMP        –       0X56           – |  –G—  |      1     0     1
     86 |       86 |   STP        –          –           – |  –G—  |      1     0     1
Machine terminated!

### STATISTICS ###

 INSTRUCTION  | #CALLS
———————————————————————
      0 (STP) |      1
    0X1 (JMP) |      2
    0X2 (JGZ) |      4
    0X3 (JOF) |      0
    0X4 (ADD) |      7
    0X5 (SUB) |      8
    0X6 (AND) |      0
    0X7 (BOR) |      0
    0X8 (SHL) |      0
    0X9 (SHR) |      1
    0XA (LDA) |     13
    0XB (LDB) |      8
    0XC (LDC) |      3
    0XD (LD0) |      3
    0XE (STR) |      8
    0XF (MOV) |      0
   0X10 (NOP) |      0
   0X12 (JEZ) |      4
   0X13 (JNO) |      0
   0X14 (MUL) |      4
   0X15 (DIV) |      4
   0X1A (RLA) |      8
   0X1B (RLB) |      4
   0X1C (LDM) |      0
   0X1D (LD1) |      4
———————————————————————
        TOTAL |     86

### END ###
```