Tiny Machine ISA

1 Tiny Machine VM Architecture

The Tiny Machine is a von Neumann machine with a single register, the accumulator (ACCUM) and a single memory. It is word-addressed and each instruction occupies a single word.

1.1 Registers

The VM has only one built-in user-accessible register, the accumulator (ACCUM).

1.2 Instruction Format

The Instruction Set Architecture (ISA) has instructions that may have two components, which are input and output as integers (i.e., they have the C type int) named as follows:

OP	is the operation code
ADDR	an address in the memory

The list of instructions and details on their execution appears in Appendix A.

1.3 VM Cycles

The VM instruction cycle conceptually does the following for each instruction:

1. Let *IR* be the instruction in the memory at the location given by PC. (Note that *IR* could be considered to be the contents of a register.)

2. The PC is made to point to the next location in memory (it is incremented by 1).

3. The instruction *IR* is executed The OP component of this instruction (*IR*.OP) indicates the operation to be executed. For example, if *IR*.OP encodes the instruction ADD, then the machine adds the memory value at the address given (by the *IR*.ADDR field of the instruction) to the accumulator. Note that arithmetic overflows and underflows happen as in C **int** arithmetic.

This instruction cycle is repeated endlessly until the machine halts by executing a HLT instruction.

1.4 VM Initial/Default Values

When the VM starts execution, PC and ACCUM are all 0. This means that execution starts with the instruction at memory element 0. Similarly, the initial values in memory are all zero (0), before the program is loaded.

1.5 Size Limits

The following constants define the size limitations of the VM.

- `MAX_ADDR` is 65536

1.6 Invariants

The VM enforces the following invariant and will halt with an error message (written to stderr) if violated:

- $0 \leq \mathsf{PC} \wedge \mathsf{PC} < \texttt{MAX\_ADDR}$

A Appendix A: Instructions

In the following table, italicized names (such as a) are meta-variables that refer to integers. If an instruction's ADDR field is notated as $-$, then its value does not matter (we use `0` as a placeholder for such values in examples).

OP Code Num.	OP Mnemonic	M	Comment (Explanation)
0	LIT	n	Literal n put in accumulator: $\mathsf{ACCUM} \leftarrow n$
1	LOD	a	Load from address a: $\mathsf{ACCUM} \leftarrow \text{memory}[a]$
2	STO	a	Store to address a: $\text{memory}[a] \leftarrow \mathsf{ACCUM}$
3	ADD	a	Add value in a to accumulator: $\mathsf{ACCUM} \leftarrow \mathsf{ACCUM} + \text{memory}[a]$
4	SUB	a	Subtract value in a from accumulator: $\mathsf{ACCUM} \leftarrow \mathsf{ACCUM} - \text{memory}[a]$
5	CIN	$-$	Read a character from stdin, with -1 representing EOF or error: $\mathsf{ACCUM} \leftarrow \textbf{getc}()$
6	COU	$-$	Print a character to stdout: $\textbf{putc}(\mathsf{ACCUM})$
7	HLT		Halt the program's execution
8	JMP	a	Jump to the given address: $\mathsf{PC} \leftarrow a$
9	SKZ	a	Skip if zero: $\textbf{if}\ \mathsf{ACCUM} = 0\ \textbf{then}\ \{\mathsf{PC} \leftarrow \mathsf{PC} + 1\}$
10	SKG	a	Skip if greater than zero: $\textbf{if}\ \mathsf{ACCUM} > 0\ \textbf{then}\ \{\mathsf{PC} \leftarrow \mathsf{PC} + 1\}$
11	SKL	a	Skip if less than zero: $\textbf{if}\ \mathsf{ACCUM} < 0\ \textbf{then}\ \{\mathsf{PC} \leftarrow \mathsf{PC} + 1\}$
12	OR	a	Bitwise-or the value in a to accumulator: $\mathsf{ACCUM} \leftarrow \mathsf{ACCUM} \vee \text{memory}[a]$
13	AND	a	Bitwise-and the value in a to accumulator: $\mathsf{ACCUM} \leftarrow \mathsf{ACCUM} \wedge \text{memory}[a]$
14	NOT	$-$	Bitwise complement the value in the accumulator: $\mathsf{ACCUM} \leftarrow \neg\mathsf{ACCUM}$
15	NDB	$-$	Stop printing debugging output

B Appendix B: Examples

B.1 A Simple Example Showing Output Formatting

The following very simple example shows the expected formatting. Suppose the input is the following file (`tm-test0.txt`, the name of this file is passed to the VM on the Unix command line):

```
7 0
```

Running the VM with the above input produces the following output (written to stdout). Note that there are two parts to the output: (1) a listing of the instructions in the program one per line, following a header, with mnemonics for each instruction and (2) a trace of the program's execution, following the line `Tracing ...` (all on standard output). The trace of execution shows the state of the built-in registers (PC and ACCUM) and the memory's values, the first 100 locations with values printed in hexadecimal notation (as that is more convenient for reading instructions) and the rest of the locations with values printed in decimal. However, only the first of a run on zero values are shown, as most locations hold zero (since that is the initial value). After this output of the program state, there is a line printed (on stdout) that starts with `==> addr:` followed by: (a) the address of the instruction being executed, then (b) the instruction with its mnemonic and ADDR value, then after showing the instruction being executed (and after the instruction's execution by the VM) the state is again shown (in the same format as before). The output of the instruction and the resulting state are show after is each instruction executed.

```
Addr  OP    ADDR
0     HLT   0
Tracing ...
PC: 0 ACCUM: 0
memory: 0: 0x7000000 1: 0x0 ...
100: 0 ...
==> addr: 0     HLT   0
PC: 1 ACCUM: 0
memory: 0: 0x7000000 1: 0x0 ...
100: 0 ...
```

B.2 A More Involved Example

The following example has a more complex program and shows some of the details of the machine's execution.

B.2.1 Input File

The following is the contents of the file `tm-test1.txt`:

```
0 5
2 105
0 7
3 105
2 106
0 12
4 106
9 0
8 12
0 89
2 107
8 14
0 78
2 107
15 0
0 10
2 108
1 107
```

```
6 0
1 108
6 0
7 0
```

B.2.2 Output (To Stdout)

Running the VM with the above input produces the following output (written to stdout).

```
Addr  OP    ADDR
0     LIT   5
1     STO   105
2     LIT   7
3     ADD   105
4     STO   106
5     LIT   12
6     SUB   106
7     SKZ   0
8     JMP   12
9     LIT   89
10    STO   107
11    JMP   14
12    LIT   78
13    STO   107
14    NDB   0
15    LIT   10
16    STO   108
17    LOD   107
18    COU   0
19    LOD   108
20    COU   0
21    HLT   0
Tracing ...
PC: 0 ACCUM: 0
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ...
==> addr: 0     LIT   5
PC: 1 ACCUM: 5
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ...
==> addr: 1     STO   105
PC: 2 ACCUM: 5
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
```

```
22: 0x0 ...
100: 0 ... 105: 5 106: 0 ...
==> addr: 2     LIT   7
PC: 3 ACCUM: 7
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 0 ...
==> addr: 3     ADD   105
PC: 4 ACCUM: 12
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 0 ...
==> addr: 4     STO   106
PC: 5 ACCUM: 12
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 12 107: 0 ...
==> addr: 5     LIT   12
PC: 6 ACCUM: 12
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 12 107: 0 ...
==> addr: 6     SUB   106
PC: 7 ACCUM: 0
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 12 107: 0 ...
==> addr: 7     SKZ   0
PC: 9 ACCUM: 0
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 12 107: 0 ...
==> addr: 9     LIT   89
PC: 10 ACCUM: 89
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
```

```
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 12 107: 0 ...
==> addr: 10    STO   107
PC: 11 ACCUM: 89
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 12 107: 89 108: 0 ...
==> addr: 11    JMP   14
PC: 14 ACCUM: 89
memory: 0: 0x5 1: 0x2000069 2: 0x7 3: 0x3000069 4: 0x200006a 5: 0xc
6: 0x400006a 7: 0x9000000 8: 0x800000c 9: 0x59 10: 0x200006b
11: 0x800000e 12: 0x4e 13: 0x200006b 14: 0xf000000 15: 0xa 16: 0x200006c
17: 0x100006b 18: 0x6000000 19: 0x100006c 20: 0x6000000 21: 0x7000000
22: 0x0 ...
100: 0 ... 105: 5 106: 12 107: 89 108: 0 ...
==> addr: 14    NDB   0
Y
```