

# Informatics 1

## Lecture 2: Abstract machines

Wettl Ferenc

Budapest University of Technology and Economics

2016-09-13

- 1 Turing machine
- 2 RAM-machine (random access machine)

- A Turing machine can be defined by  $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ , where
- $Q$  is the non-empty set of "states",
- $\Gamma$  the finite, non-empty "tape alphabet",
- $b \in \Gamma$  the "blank symbol" (the only symbol allowed to occur on the tape infinitely often),
- $\Sigma \subseteq \Gamma \setminus \{b\}$  the set of "input symbols",
- $q_0 \in Q$  the "initial state"
- $F \subseteq Q$  the set of "final states" (this is when the machine stops),
- $\delta : (Q \setminus F) \times \Gamma \hookrightarrow Q \times \Gamma \times \{L, R\}$  is a partial function called the "transition function", where L is left shift, R is right shift (moves the tape)

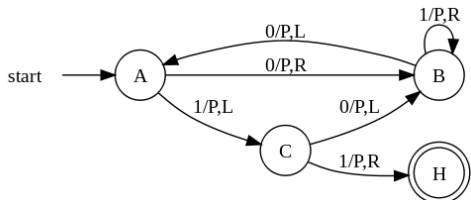


H *Church-Turing thesis*: Every formalizable problem, that can be solved with an algorithm can be solved with a Turing-machine.

- **Busy beaver** (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.

- $Q = \{A, B, C, \text{HALT}\}$
- $\Gamma = \{0, 1\}$
- $b = 0$  (empty symbol)
- $\Sigma = \{1\}$
- $q_0 = A$  (initial state)
- $F = \{\text{HALT}\}$
- $\delta$  table:

	A	B	C
0	1RB	1LA	1LB
1	1LC	1RB	1RH



1	A	0	0	0	0	0	0	0	0	0	0	0	0	0
2	B	0	0	0	0	0	0	1	0	0	0	0	0	0
3	A	0	0	0	0	1	1	0	0	0	0	0	0	0
4	C	0	0	0	1	1	0	0	0	0	0	0	0	0
5	B	0	0	1	1	1	0	0	0	0	0	0	0	0
6	A	0	1	1	1	1	0	0	0	0	0	0	0	0
7	B	0	0	1	1	1	1	1	0	0	0	0	0	0
8	B	0	0	0	1	1	1	1	1	0	0	0	0	0
9	B	0	0	0	0	1	1	1	1	1	0	0	0	0
10	B	0	0	0	0	0	1	1	1	1	1	0	0	0
11	B	0	0	0	0	0	0	1	1	1	1	1	1	0
12	A	0	0	0	0	1	1	1	1	1	1	0	0	0
13	C	0	0	0	1	1	1	1	1	1	0	0	0	0
14	H	0	0	0	1	1	1	1	1	1	0	0	0	0

- The RAM-machine consists of a  $p$  program register and an  $r$  data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell  $p_0$  and ends with an empty command.
- The contents of the  $i$ th cell of the data register ( $i \in \mathbb{N}_0$ ) is denoted by  $r[i]$  or  $r_i$ , these can only contain integers.
- These are the possible commands, where  $z \in \mathbb{Z}$ ,  $i, n \in \mathbb{N}_0$ :

$$r_i \leftarrow z$$

$$r_i \leftarrow r_n, r_i \leftarrow r_n \text{ (same as } r_i \leftarrow r[r[n]]\text{)},$$

$$r_i \leftarrow r_i \pm r_n, (r_i \leftarrow r_i * r_n, r_i \leftarrow r_i / r_n),$$

$p_n$ : jump to the  $n$ th program line,

if  $r_i = 0$   $p_n$ : jump to the  $n$ th program line if  $r_i = 0$ ,

if  $r_i > 0$   $p_n$ : jump to the  $n$ th program line if  $r_i > 0$ ,

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,
- every memory cell is 1 byte long, every program line is 2 bytes long, the first byte contains the command and the second byte contains the operand, i.e.

ADD 12 means:  $r_0 \leftarrow r_0 + r_{12}$

- every calculation is done with the 0th memory cell (and sometimes another one),
- we use mnemonikokkal for the commands, there are three types:
  - explicit: the operand  $n$  is a number (denoted by an = at the end of the expression)
  - direct: the operand  $n$  is a memory cell, the operation is done with the contents of  $r[n]$ ,
  - indirect: the operand  $n$  is the index of a memory cell, the operation is done with  $r[r[n]]$  (denoted by a \* at the end of the expression)

### Controller commands

JUMP	$n$	jump to the $n$ th command
JZERO	$n$	jump to the $n$ th command if $r_0 = 0$
JGTZ	$n$	jump to the $n$ th command if $r_0 > 0$
HALT		stop

### Arithmetic commands

	<i>direct</i>		<i>indirect</i>		<i>explicit op</i>			
ADD	$n$	$r_0 \leftarrow r_0 + r_n$	ADD*	$n$	$r_0 \leftarrow r_0 + r_{r_n}$	ADD=	$n$	$r_0 \leftarrow r_0 + n$
SUB	$n$	$r_0 \leftarrow r_0 - r_n$	SUB*	$n$	$r_0 \leftarrow r_0 - r_{r_n}$	SUB=	$n$	$r_0 \leftarrow r_0 - n$
MULT	$n$	$r_0 \leftarrow r_0 * r_n$	MULT*	$n$	$r_0 \leftarrow r_0 * r_{r_n}$	MULT=	$n$	$r_0 \leftarrow r_0 * n$
DIV	$n$	$r_0 \leftarrow r_0 / r_n$	DIV*	$n$	$r_0 \leftarrow r_0 / r_{r_n}$	DIV=	$n$	$r_0 \leftarrow r_0 / n$

### Data manipulation, IO

	<i>direct</i>		<i>indirect</i>		<i>explicit op</i>			
LOAD	$n$	$r_0 \leftarrow r_n$	LOAD*	$n$	$r_0 \leftarrow r_{r_n}$	LOAD=	$n$	$r_0 \leftarrow n$
STORE	$n$	$r_n \leftarrow r_0$	STORE*	$n$	$r_{r_n} \leftarrow r_0$			
READ	$n$	reads $n$ numbers from the input into $r_1, r_2, \dots, r_n$						
WRITE	$n$	writes $n$ numbers to the output from $r_1, r_2, \dots, r_n$						

Write a program to calculate  $(a, b)$  (greatest common divisor), where  $a, b \in \mathbb{N}_0!$

p	command	operand	notes
0	LOAD	= 12	
1	STORE	1	$r[1] \leftarrow a$
2	LOAD	= 16	
3	STORE	2	$r[2] \leftarrow b$
4	JZERO	17	
5	LOAD	1	$r[0] \leftarrow r[1]$
6	DIV	2	$r[0] \leftarrow \lfloor a/b \rfloor$
7	STORE	3	$r[3] \leftarrow \lfloor a/b \rfloor$
8	MULT	2	
9	STORE	4	$r[4] \leftarrow b \cdot \lfloor a/b \rfloor$
10	LOAD	1	
11	SUB	4	$r[0] \leftarrow a - b \cdot \lfloor a/b \rfloor = a \bmod b$
12	STORE	5	
13	LOAD	2	
14	STORE	1	$r[1] \leftarrow b$
15	LOAD	5	$b \leftarrow a \bmod b$
16	JUMP	3	
17	LOAD	1	
18	STORE	6	this is $(a, b)$
19	HALT	0	



A program for the Collatz-problem: let  $x \in \mathbb{N}^+$ , if  $x$  is even, then  $x \leftarrow x/2$ , if  $x$  is odd, then  $x \leftarrow 3x + 1$ . Is it true that starting from any number we eventually reach 1?

p	Assembly	op.	Machine code		$3x + 1$ (COLLATZ PROBLEM)
0	LOAD	= 33	10000011	00100001	load input value
1	STORE	2	10010000	00000010	store into cell 2
2	DIV	= 2	01110011	00000010	divide by 2
3	STORE	1	10010000	00000001	store into cell 1
4	MULT	= 2	01100011	00000010	multiply by 2
5	SUB	2	01010000	00000010	
6	JZERO	11	11100000	00001100	if it is even, jump
7	LOAD	2	10000000	00000010	
8	MULT	= 3	01100011	00000011	multiply by 3
9	ADD	= 1	01000011	00000001	plus 1
10	JUMP	1	11010000	00000010	jump to 1
11	LOAD	1	10000000	00000001	if it was even
12	STORE	2	10010000	00000010	
13	SUB	= 1	01010011	00000001	is it equal 1?
14	JZERO	17	11100000	00010010	if so, then stop
15	LOAD	1	10000000	00000001	if not, continue
16	JUMP	2	11010000	00000010	jump to 2
17	HALT		11000000	00000000	

# What about real machines?

- **Neumann-principle**: there is no difference between data and program.
- **Machine code**: binary sequence that the processor can directly interpret.
- Every code written in any programming language **compiles** into machine code.
- The closest language to machine code is **assembly**, it is a bit similar to what we described as an example to the RAM-machine. The compiler for assembly is called **assembler**.
- **Registers** are special CPU memories with extremely fast read-write speeds, but very limited capacity.

# Kérdések

- 1 What are the basics of a Turing-machine?
- 2 What does the "busy beaver" do?
- 3 What's the difference between the direct and the indirect commands?
- 4 What is the machine code, assembly and the assembler?
- 5 What are the contents of the memory after issuing these commands?

1	LOAD=	5
2	STORE	1
3	STORE*	1
4	JZERO	7
5	LOAD=	2
6	MUL	1
7	HALT	