

# Computer Architecture

Daniel Page

Department of Computer Science,  
University Of Bristol,  
Merchant Venturers Building,  
Woodland Road,  
Bristol, BS8 1UB. UK.  
([csdsp@bristol.ac.uk](mailto:csdsp@bristol.ac.uk))

October 14, 2024

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
  - ▶ the associated notes page may be pre-populated with extra, written explanation of material covered in lecture(s), plus
  - ▶ anything with a “grey’ed out” header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

## ► Agenda:

1. introduce a theoretical computational model that is “*closely related to what happens in [practical] modern digital computers*” [7, Page 199], then
2. demonstrate how we implement said model in hardware.

Notes:

## Part 1: in theory (1)

## Definition

A **Register Machine (RM)** is specified by

- a finite number of registers, each of which can store an (infinite) natural number;  $R_i \in \mathbb{N}$  denotes the  $i$ -th such register for  $0 \leq i < r$ , and
- a program, consisting of a finite list of instructions of the form

$$\text{label} : \text{body}$$

such that the  $i$ -th instruction has label  $L_i$ .

## Definition

An RM **configuration** is a tuple

$$C = (l, v_0, v_1, \dots, v_{r-1})$$

where

- $l$  is the current label, and
- $v_j$  is the current value stored in register  $R_j$ .

+ve : theoretically attractive: can be proved equivalent to a Turing machine

+ve : practically attractive: has clear analogies with, e.g., a calculator

−ve : some aspects (e.g., infinite sized registers) cannot be realised in practice

−ve : can be very inefficient

Notes:

- Keep in mind item #54 of Perlis’ Epigrams on Programming [9]: “*[b]eware of the Turing tar-pit in which everything is possible but nothing of interest is easy*”.

## Part 1: in theory (2)

### Definition

A (finite or infinite) computation by some RM is captured by

$$\langle C_0, C_1, C_2, \dots \rangle$$

i.e., a sequence of configurations such that

- ▶ for  $i = 0$ ,

$$C_i = (0, v_0, v_1, \dots, v_{r-1})$$

is the **initial configuration** where  $v_j$  is the initial value stored in register  $R_j$ ,

- ▶ for  $i > 0$ ,  $C_i$  results from applying the instruction at label  $L_i$  to

$$C_{i-1} = (l, v_0, v_1, \dots, v_{r-1}).$$

### Definition

A finite computation by some RM is captured by

$$\langle C_0, C_1, C_2, \dots, C_{h-1} \rangle$$

such that in the **halting configuration**

$$C_{h-1} = (l, v_0, v_1, \dots, v_{r-1})$$

the instruction labelled  $L_l$  either

- ▶ explicitly, or intentionally forces computation to halt, i.e., is a halt instruction, or
- ▶ implicitly, or unintentionally forces computation to halt, e.g., causes an error condition.

Notes:

## Part 1: in theory (3)

- ▶ **Example:** roughly per [7, Chapter 11], consider a **counter machine** such that

1.  $r = 4$ , i.e., it has 4 registers,
2. each  $R_i \in \{0, 1, \dots, 2^4 - 1 = 15\}$ , i.e., the registers store (finite) 4-bit values,
3. the set of valid instructions is

$L_i : R_{addr} \leftarrow R_{addr} + 1$  then goto  $L_{i+1}$

$L_i : R_{addr} \leftarrow R_{addr} - 1$  then goto  $L_{i+1}$

$L_i : \text{if } R_{addr} = 0 \text{ then goto } L_{target} \text{ else goto } L_{i+1}$

$L_i : \text{halt}$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation.

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{array}{lcl} C_0 & = & (0, 0, 0, 2, 0) \\ L_0 & \rightsquigarrow & \text{if } R_2 = 0 \text{ then goto } L_5 \text{ else goto } L_1 \\ C_1 & = & (1, 0, 0, 2, 0) \end{array}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{array}{lcl} C_1 & = & (1, 0, 0, 2, 0) \\ L_1 & \rightsquigarrow & R_2 \leftarrow R_2 - 1 \text{ then goto } L_2 \\ C_2 & = & (2, 0, 0, 1, 0) \end{array}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{array}{lcl} C_2 & = & (2, 0, 0, 1, 0) \\ L_2 & \rightsquigarrow & R_3 \leftarrow R_3 + 1 \text{ then goto } L_3 \\ C_3 & = & (3, 0, 0, 1, 1) \end{array}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_3 &= (3, 0, 0, 1, 1) \\ L_3 &\leadsto R_1 \leftarrow R_1 + 1 \text{ then goto } L_4 \\ C_4 &= (4, 0, 1, 1, 1) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_4 &= (4, 0, 1, 1, 1) \\ L_4 &\leadsto \text{if } R_0 = 0 \text{ then goto } L_0 \text{ else goto } L_5 \\ C_5 &= (0, 0, 1, 1, 1) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_5 &= (0, 0, 1, 1, 1) \\ L_0 &\rightsquigarrow \text{if } R_2 = 0 \text{ then goto } L_5 \text{ else goto } L_1 \\ C_6 &= (1, 0, 1, 1, 1) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_6 &= (1, 0, 1, 1, 1) \\ L_1 &\rightsquigarrow R_2 \leftarrow R_2 - 1 \text{ then goto } L_2 \\ C_7 &= (2, 0, 1, 0, 1) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_7 &= (2, 0, 1, 0, 1) \\ L_2 &\leadsto R_3 \leftarrow R_3 + 1 \text{ then goto } L_3 \\ C_8 &= (3, 0, 1, 0, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_8 &= (3, 0, 1, 0, 2) \\ L_3 &\leadsto R_1 \leftarrow R_1 + 1 \text{ then goto } L_4 \\ C_9 &= (4, 0, 2, 0, 2) \end{aligned}$$

Notes:



## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_9 &= (4, 0, 2, 0, 2) \\ L_4 &\rightsquigarrow \text{if } R_0 = 0 \text{ then goto } L_0 \text{ else goto } L_5 \\ C_{10} &= (0, 0, 2, 0, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{10} &= (0, 0, 2, 0, 2) \\ L_0 &\rightsquigarrow \text{if } R_2 = 0 \text{ then goto } L_5 \text{ else goto } L_1 \\ C_{11} &= (5, 0, 2, 0, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{11} &= (5, 0, 2, 0, 2) \\ L_5 &\leadsto \text{if } R_1 = 0 \text{ then goto } L_9 \text{ else goto } L_6 \\ C_{12} &= (6, 0, 2, 0, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{12} &= (6, 0, 2, 0, 2) \\ L_6 &\leadsto R_1 \leftarrow R_1 - 1 \text{ then goto } L_7 \\ C_{13} &= (7, 0, 1, 0, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{13} &= (7, 0, 1, 1, 2) \\ L_7 &\leadsto R_2 \leftarrow R_2 + 1 \text{ then goto } L_8 \\ C_{14} &= (8, 0, 1, 1, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{14} &= (8, 0, 1, 1, 2) \\ L_8 &\leadsto \text{if } R_0 = 0 \text{ then goto } L_5 \text{ else goto } L_9 \\ C_{15} &= (5, 0, 1, 1, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{15} &= (5, 0, 1, 1, 2) \\ L_5 &\leadsto \text{if } R_1 = 0 \text{ then goto } L_9 \text{ else goto } L_6 \\ C_{16} &= (6, 0, 1, 1, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{16} &= (6, 0, 1, 1, 2) \\ L_6 &\leadsto R_1 \leftarrow R_1 - 1 \text{ then goto } L_7 \\ C_{17} &= (7, 0, 0, 1, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

```
C17 = (7, 0, 0, 1, 2)
L7  ~> R2 ← R2 + 1 then goto L8
C18 = (8, 0, 0, 2, 2)
```

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

```
C18 = (8, 0, 0, 2, 2)
L8  ~> if R0 = 0 then goto L5 else goto L9
C19 = (5, 0, 0, 2, 2)
```

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{19} &= (5, 0, 0, 2, 2) \\ L_5 &\leadsto \text{if } R_1 = 0 \text{ then goto } L_9 \text{ else goto } L_6 \\ C_{20} &= (9, 0, 0, 2, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

$$\begin{aligned} C_{20} &= (9, 0, 0, 2, 2) \\ L_9 &\leadsto \text{halt} \\ C_{21} &= (9, 0, 0, 2, 2) \end{aligned}$$

Notes:

## Part 1: in theory (4)

### ► Example: now, given

1. a program e.g.,

```
L0 : if R2 = 0 then goto L5 else goto L1
L1 : R2 ← R2 - 1 then goto L2
L2 : R3 ← R3 + 1 then goto L3
L3 : R1 ← R1 + 1 then goto L4
L4 : if R0 = 0 then goto L0 else goto L5
L5 : if R1 = 0 then goto L9 else goto L6
L6 : R1 ← R1 - 1 then goto L7
L7 : R2 ← R2 + 1 then goto L8
L8 : if R0 = 0 then goto L5 else goto L9
L9 : halt
```

2. an initial configuration, e.g.,

$$C_0 = (0, 0, 0, 2, 0)$$

we can produce a **trace** of computation:

```
C20 = (9, 0, 0, 2, 2)
L9  ~→ halt
C21 = (9, 0, 0, 2, 2)
```

which demonstrates that the program copies R<sub>2</sub> into R<sub>3</sub>.

Notes:

Aside: is this the *only* viable example?

### ► No: (many) **alternatives** exist, stemming from

1. different models, e.g.,

- counter machine,
- Random-Access Machine (RAM),
- Random-Access Stored-Program (RASP) machine,
- ...

Notes:

Aside: is this the *only* viable example?

- ▶ No: (many) **alternatives** exist, stemming from

2. different instruction set content, e.g., a counter machine with or without

$$L_i : R_{addr} \leftarrow 0$$

or “clear” instruction.

Notes:

Aside: is this the *only* viable example?

- ▶ No: (many) **alternatives** exist, stemming from

3. different instruction set format, e.g.,

- ▶ **register machine**  $\simeq$  3-operand model:

- $r$  registers,
- source and destination operands can be specified independently,
- (rough) example:

$$\begin{array}{llll} R_0 \leftarrow 10 & \rightsquigarrow & C_0 = ( & 0, \quad 0, \quad 0, \quad 0, \quad 0 \quad ) \\ R_1 \leftarrow 20 & \rightsquigarrow & C_1 = ( & 1, \quad 10, \quad 0, \quad 0, \quad 0 \quad ) \\ R_2 \leftarrow R_0 + R_1 & \rightsquigarrow & C_2 = ( & 2, \quad 10, \quad 20, \quad 0, \quad 0 \quad ) \\ & & C_3 = ( & 3, \quad 10, \quad 20, \quad 30, \quad 0 \quad ) \end{array}$$

Notes:



Aside: is this the *only* viable example?

- ▶ No: (many) **alternatives** exist, stemming from

3. different instruction set format, e.g.,

▶ **register machine**  $\simeq$  2-operand model:

- $r$  registers,
- operands may need to be reused as source *and* destination,
- (rough) example:

$$\begin{array}{lll} R_0 \leftarrow 10 & \rightsquigarrow & C_0 = ( \begin{array}{ccccc} 0, & 0, & 0, & 0, & 0 \end{array} ) \\ R_1 \leftarrow 20 & \rightsquigarrow & C_1 = ( \begin{array}{ccccc} 1, & 10, & 0, & 0, & 0 \end{array} ) \\ R_0 \leftarrow R_0 + R_1 & \rightsquigarrow & C_2 = ( \begin{array}{ccccc} 2, & 10, & 20, & 0, & 0 \end{array} ) \\ & & C_3 = ( \begin{array}{ccccc} 3, & 30, & 20, & 0, & 0 \end{array} ) \end{array}$$

Notes:

Aside: is this the *only* viable example?

- ▶ No: (many) **alternatives** exist, stemming from

3. different instruction set format, e.g.,

▶ **accumulator machine**  $\simeq$  1-operand model:

- may have  $r > 1$  register, but there is 1 special-purpose case termed the **accumulator**,
- operations implicitly use accumulator for source and/or destination operands,
- (rough) example:

$$\begin{array}{lll} A \leftarrow 10 & \rightsquigarrow & C_0 = ( \begin{array}{ccccc} 0, & 0, & 0, & 0, & 0 \end{array} ) \\ A \leftarrow A + 20 & \rightsquigarrow & C_1 = ( \begin{array}{ccccc} 1, & 10, & 0, & 0, & 0 \end{array} ) \\ & & C_2 = ( \begin{array}{ccccc} 2, & 30, & 0, & 0, & 0 \end{array} ) \end{array}$$

Notes:

Aside: is this the *only* viable example?

► No: (many) **alternatives** exist, stemming from

3. different instruction set format, e.g.,

► **stack machine**  $\simeq$  0-operand model:

- may have  $r > 1$  register, but managed per a **stack** (i.e., FILO-style) policy,
- operations implicitly use stack for source and/or destination operands,
- (rough) example:

	$\rightsquigarrow$	$C_0 = ( \quad 0, \quad 0, \quad 0, \quad 0, \quad 0 \quad )$
push 20	$\rightsquigarrow$	$C_1 = ( \quad 1, \quad 20, \quad 0, \quad 0, \quad 0 \quad )$
push 10	$\rightsquigarrow$	$C_2 = ( \quad 2, \quad 10, \quad 20, \quad 0, \quad 0 \quad )$
add	$\rightsquigarrow$	$C_3 = ( \quad 3, \quad 30, \quad 0, \quad 0, \quad 0 \quad )$
pop	$\rightsquigarrow$	$C_4 = ( \quad 4, \quad 0, \quad 0, \quad 0, \quad 0 \quad )$

Notes:

## Part 2: in practice (1)

Design

### Definition

Consider a sequence

$$x = \langle x_0, x_1, \dots, x_{n-1} \rangle$$

where, for each  $0 \leq i < n$  we have  $x_i \in \mathbb{N}$ . The associated **Gödel encoding** (or **Gödel numbering**) is

$$\hat{x} = \prod_{i=0}^{i < n} p_i^{x_i} = p_0^{x_0} \cdot p_1^{x_1} \cdots p_{n-1}^{x_{n-1}}$$

where  $p_i$  is the  $i$ -th prime, i.e.,  $p_0 = 2, p_1 = 3, p_2 = 5$ , and so on. Due to Euclid’s unique prime-factorisation theorem, factoring  $\hat{x}$  allows recovery of  $x$ .

$\therefore$  we can represent *anything* using elements of  $\mathbb{N}$ , e.g., per [8, Section VII.A],

- let 6 represent “0”,
- let 5 represent “=”, then
- the logical statement “ $0 = 0$ ” can be represented as

$$2^6 \cdot 3^5 \cdot 5^6 = 243,000,000.$$

Notes:

## Part 2: in practice (2)

Design

### ► Concept:

#### ► One can view

(human-readable) instruction  $\simeq$  abstraction of (machine-readable) control information,

i.e.,

instruction	=	information	$\mapsto$	what to do
data	=	information	$\mapsto$	what to do it on/with

Notes:

## Part 2: in practice (2)

Design

### ► Concept:

#### ► Gödel encoding allows numerical representation of *either* form of information, e.g.,

1	$\mapsto$	“the integer one” if it represents some data
1	$\mapsto$	“compute an addition” if it represents an instruction

are different, valid interpretations of the same number.

Notes:

► Concept:

- These facts suggest a strategy: an RM is an FSM in disguise, in the sense that
  1. an RM configuration is an FSM state, and
  2. the RM program determines the FSM transition function,
 so we could therefore
  - use, e.g., Gödel encoding or variant thereof, to encode instructions into numerical **machine code**,
  - store the machine code in memory,
  - have our implementation decode machine code into appropriate control signals.

Notes:

► Design: specify an instruction encoding, e.g.,

$L_i : R_{addr} \leftarrow R_{addr} + 1 \text{ then goto } L_{i+1}$	$\mapsto$	<table><tr><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="3">000</td><td colspan="2">addr</td><td colspan="3">0000</td><td></td></tr></table>	8	7	6	5	4	3	2	1	0	000			addr		0000			
8	7	6	5	4	3	2	1	0												
000			addr		0000															
$L_i : R_{addr} \leftarrow R_{addr} - 1 \text{ then goto } L_{i+1}$	$\mapsto$	<table><tr><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="3">001</td><td colspan="2">addr</td><td colspan="3">0000</td><td></td></tr></table>	8	7	6	5	4	3	2	1	0	001			addr		0000			
8	7	6	5	4	3	2	1	0												
001			addr		0000															
$L_i : \text{if } R_{addr} = 0 \text{ then goto } L_{target} \text{ else goto } L_{i+1}$	$\mapsto$	<table><tr><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="3">010</td><td colspan="2">addr</td><td colspan="3">target</td><td></td></tr></table>	8	7	6	5	4	3	2	1	0	010			addr		target			
8	7	6	5	4	3	2	1	0												
010			addr		target															
$L_i : \text{halt}$	$\mapsto$	<table><tr><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="3">011</td><td colspan="2">00</td><td colspan="3">0000</td><td></td></tr></table>	8	7	6	5	4	3	2	1	0	011			00		0000			
8	7	6	5	4	3	2	1	0												
011			00		0000															

such that

$$L_i : \text{if } R_2 = 0 \text{ then goto } L_5 \text{ else goto } L_{i+1} \mapsto 010100101_{(2)} = 0A5_{(16)}.$$

Notes:

## Part 2: in practice (4)

### Design

- **Design:** encode our original program

$L_0$ : if $R_2 = 0$ then goto $L_5$ else goto $L_1$	$\mapsto$	010100101 <sub>(2)</sub>	=	0A5 <sub>(16)</sub>
$L_1$ : $R_2 \leftarrow R_2 - 1$ then goto $L_2$	$\mapsto$	001100000 <sub>(2)</sub>	=	060 <sub>(16)</sub>
$L_2$ : $R_3 \leftarrow R_3 + 1$ then goto $L_3$	$\mapsto$	000110000 <sub>(2)</sub>	=	030 <sub>(16)</sub>
$L_3$ : $R_1 \leftarrow R_1 + 1$ then goto $L_4$	$\mapsto$	000010000 <sub>(2)</sub>	=	010 <sub>(16)</sub>
$L_4$ : if $R_0 = 0$ then goto $L_0$ else goto $L_5$	$\mapsto$	010000000 <sub>(2)</sub>	=	080 <sub>(16)</sub>
$L_5$ : if $R_1 = 0$ then goto $L_9$ else goto $L_6$	$\mapsto$	010011001 <sub>(2)</sub>	=	099 <sub>(16)</sub>
$L_6$ : $R_1 \leftarrow R_1 - 1$ then goto $L_7$	$\mapsto$	001010000 <sub>(2)</sub>	=	050 <sub>(16)</sub>
$L_7$ : $R_2 \leftarrow R_2 + 1$ then goto $L_8$	$\mapsto$	000100000 <sub>(2)</sub>	=	020 <sub>(16)</sub>
$L_8$ : if $R_0 = 0$ then goto $L_5$ else goto $L_9$	$\mapsto$	010000101 <sub>(2)</sub>	=	085 <sub>(16)</sub>
$L_9$ : halt	$\mapsto$	011000000 <sub>(2)</sub>	=	0C0 <sub>(16)</sub>

such that

- we use

$$\text{MEM} = \langle 0A5_{(16)}, 030_{(16)}, \dots, 0C0_{(16)} \rangle,$$

a 10-element memory,

- each  $\text{MEM}[i]$  is a 9-bit encoding of the instruction labelled  $L_i$ .

Notes:

## Part 2: in practice (5)

### Design

- **Translation:**

- we're encoding the instructions as 9-element sequence of bits,
- using a Gödel encoding is too inefficient, so we opt for

$$\begin{aligned}\hat{x} &= x_0 \parallel x_1 \parallel x_2 \\ &\equiv x_0 \cdot 2^0 + x_1 \cdot 2^4 + x_2 \cdot 2^6\end{aligned}$$

so decoding amounts to extraction of contiguous bits from  $\hat{x}$ , i.e.,

$$\begin{aligned}x_0 &= \hat{x}_{3..0} \equiv (\hat{x} \gg 0) \wedge F_{(16)} \\ x_1 &= \hat{x}_{5..4} \equiv (\hat{x} \gg 4) \wedge 3_{(16)} \\ x_2 &= \hat{x}_{8..6} \equiv (\hat{x} \gg 6) \wedge 7_{(16)}\end{aligned}$$

- where a field, e.g., *target*, is unused, we just use zero as a placeholder,
- this approach works, but clearly isn't the *only* one possible.

Notes:

Part 2: in practice (6)  
Design

Definition
The <b>Program Counter (PC)</b> is a special-purpose register that holds the address of the next instruction to be executed.

Definition
The <b>Instruction Register (IR)</b> is a special-purpose register that holds the instruction currently being executed.

Notes:

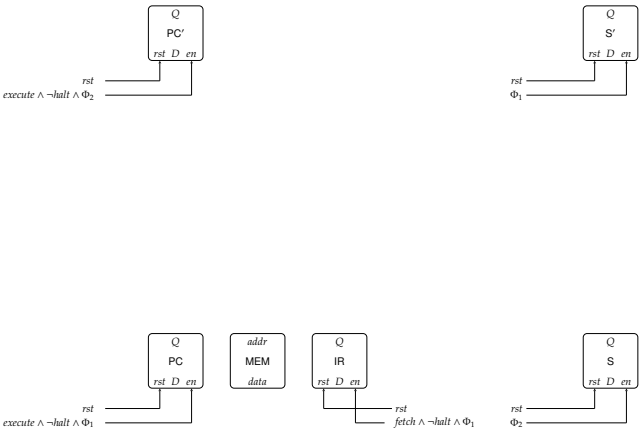
Part 2: in practice (7)  
Design

Definition
<p>The <b>fetch-decode-execute cycle</b> (aka. <b>instruction cycle</b>) is a 3-stage process</p> <div><div>1.</div><div>fetch stage :</div><div><div>1.a. load instruction into IR</div><div>1.b. increment PC</div><div><math>\mapsto</math></div><div><math>\mapsto</math></div><div><math>IR \leftarrow MEM[PC]</math></div><div><math>PC \leftarrow PC + 1</math></div></div></div> <div><div>2.</div><div>decode stage :</div><div><math>\left\{ \begin{array}{l} \textit{decide} \text{ what instruction in IR } \textit{means}, \text{ i.e.,} \\ \text{translate IR into control signals which reflect instruction semantics} \end{array} \right.</math></div></div> <div><div>3.</div><div>execute stage :</div><div><math>\left\{ \begin{array}{l} \textit{do} \text{ whatever instruction in IR } \textit{means}, \text{ i.e.,} \\ \text{apply instruction semantics} \end{array} \right.</math></div></div> <p>which describes execution of instructions; in some cases it makes sense to consider a 5-stage process by adding</p> <div><div>4.</div><div>memory access stage :</div><div>{ perform any memory accesses (e.g., loads or stores) required</div></div> <div><div>5.</div><div>write-back (or commit) stage :</div><div>{ store result(s) stemming from instruction execution (e.g., computation)</div></div> <p>i.e., expanding the execute stage to be more precise.</p>

Notes:

Part 2: in practice (8)  
Implementation: control-path

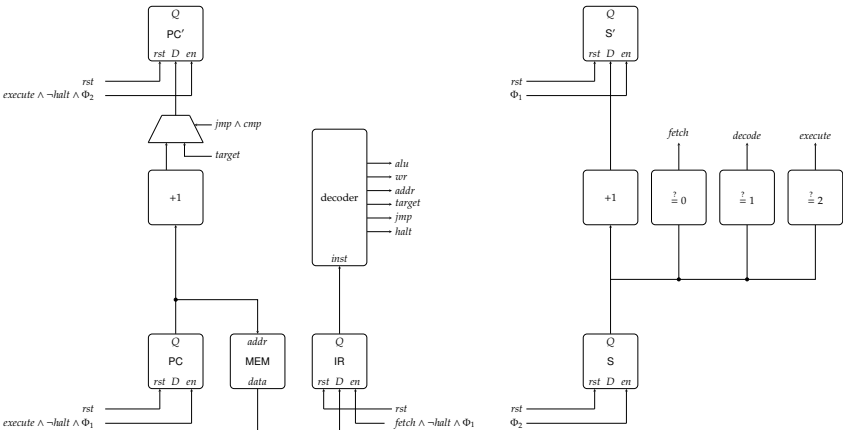
Circuit



Notes:

Part 2: in practice (8)  
Implementation: control-path

Circuit

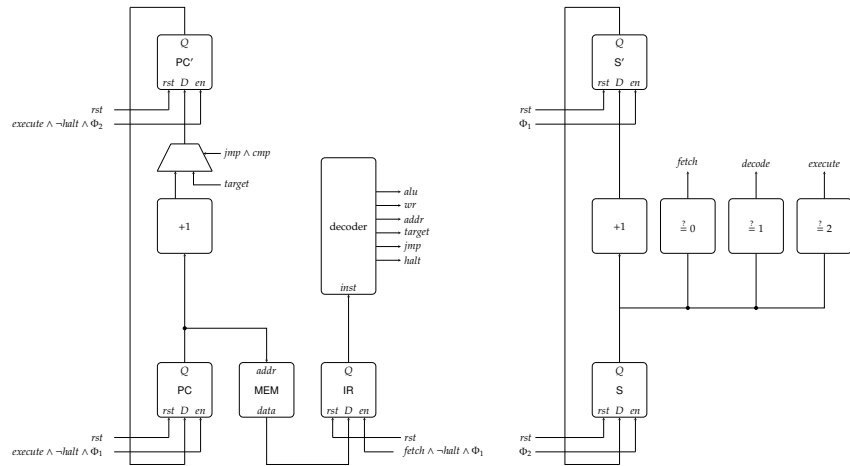


Notes:

## Part 2: in practice (8)

Implementation: control-path

### Circuit



Notes:

## Part 2: in practice (9)

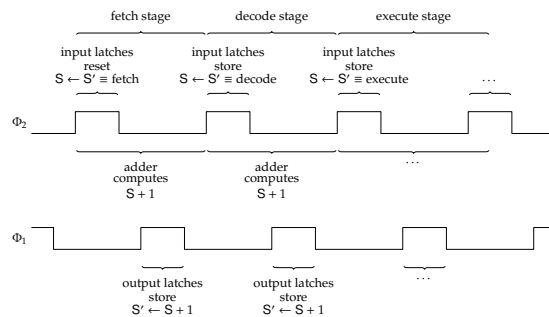
Implementation: control-path

### ► Caveat(s):

1. we're using latches (resp. flip-flops) with a dedicated *rst* input (in addition to *en*),
2. we're assuming a cyclic 2-bit counter, so the state *S* steps through values

$0, 1, 2, 3, 0, 1, 2, 3, \dots$ ,

3. we're not using  $S = 3$ , so in a sense this is an idle state,
4. we've reversed  $\Phi_1$  and  $\Phi_2$  for the sequencer, so we generate



Notes:

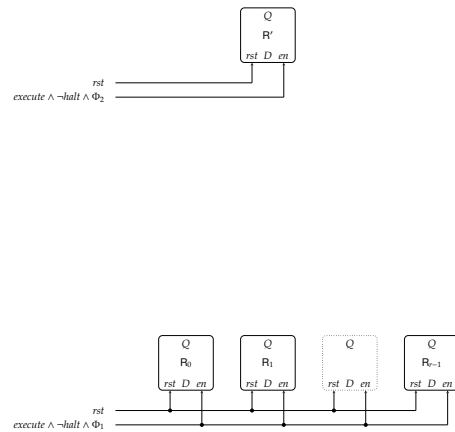
5. to correctly drive update to registers the data- and control-path,
5. we've deviated slightly from the fetch-decode-execute cycle, updating *PC* in the execute (rather than fetch) stage.



## Part 2: in practice (10)

Implementation: data-path

### Circuit

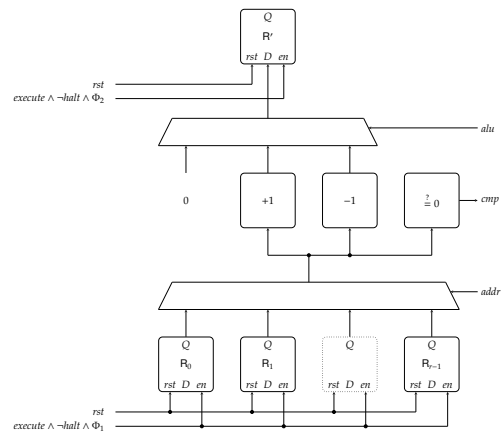


Notes:

## Part 2: in practice (10)

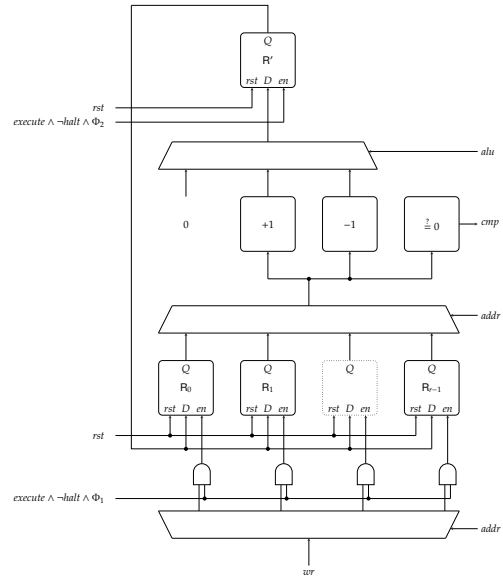
Implementation: data-path

### Circuit



Notes:

## Circuit



Notes:

## Conclusions

► Take away points:

1. A central aim here is to demonstrate that, per

combinatorial logic	$\leadsto$	fixed function, not	stateful
sequential logic	$\leadsto$	fixed function,	stateful
FSM	$\leadsto$	fixed function,	stateful
RM	$\leadsto$	not fixed function,	stateful
	$\vdots$		
micro-processor	$\leadsto$	not fixed function,	stateful

and although our counter machine is *still* limited,

- ▶ it has started to exhibit the characteristics of a *real* micro-processor, *and*
- ▶ we can *still* reason end-to-end about the implementation.

2. In doing so we've encountered some fundamental concepts, e.g.,

- ▶ instruction encoding and decoding,
- ▶ the fetch-decode-execute cycle,
- ▶ the role of PC and IR in supporting it,
- ▶ ...

which we'll revisit and refine.

Notes:

Additional Reading

- ▶ *Wikipedia: Register machine*. URL: [https://en.wikipedia.org/wiki/Register\\_machine](https://en.wikipedia.org/wiki/Register_machine).
- ▶ *Wikipedia: Counter machine*. URL: [https://en.wikipedia.org/wiki/Counter\\_machine](https://en.wikipedia.org/wiki/Counter_machine).
- ▶ *Wikipedia: Random-access machine*. URL: [https://en.wikipedia.org/wiki/Random-access\\_machine](https://en.wikipedia.org/wiki/Random-access_machine).
- ▶ *Wikipedia: Random-access stored-program machine*. URL: [https://en.wikipedia.org/wiki/Random-access\\_stored-program\\_machine](https://en.wikipedia.org/wiki/Random-access_stored-program_machine).
- ▶ *Wikipedia: Gödel numbering*. URL: [https://en.wikipedia.org/wiki/G%C3%B6del\\_numbering](https://en.wikipedia.org/wiki/G%C3%B6del_numbering).
- ▶ *Wikipedia: Machine code*. URL: [https://en.wikipedia.org/wiki/Machine\\_code](https://en.wikipedia.org/wiki/Machine_code).

Notes:

References

[1] *Wikipedia: Counter machine*. URL: [https://en.wikipedia.org/wiki/Counter\\_machine](https://en.wikipedia.org/wiki/Counter_machine) (see p. 105).

[2] *Wikipedia: Gödel numbering*. URL: [https://en.wikipedia.org/wiki/G%C3%B6del\\_numbering](https://en.wikipedia.org/wiki/G%C3%B6del_numbering) (see p. 105).

[3] *Wikipedia: Machine code*. URL: [https://en.wikipedia.org/wiki/Machine\\_code](https://en.wikipedia.org/wiki/Machine_code) (see p. 105).

[4] *Wikipedia: Random-access machine*. URL: [https://en.wikipedia.org/wiki/Random-access\\_machine](https://en.wikipedia.org/wiki/Random-access_machine) (see p. 105).

[5] *Wikipedia: Random-access stored-program machine*. URL: [https://en.wikipedia.org/wiki/Random-access\\_stored-program\\_machine](https://en.wikipedia.org/wiki/Random-access_stored-program_machine) (see p. 105).

[6] *Wikipedia: Register machine*. URL: [https://en.wikipedia.org/wiki/Register\\_machine](https://en.wikipedia.org/wiki/Register_machine) (see p. 105).

[7] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967 (see pp. 5, 11).

[8] E. Nagel and J.R. Newman. *Gödel's Proof*. Routledge, 1958 (see p. 71).

[9] A.J. Perlis. "Epigrams on programming". In: *ACM SIGPLAN Notices* 17.9 (1982), pp. 7–13 (see p. 8).

Notes: