

► **Agenda: Register Machines (RMs), i.e.,**

1. introduce associated 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,
noting that

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

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

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

-ve : can be very inefficient

Part 1: in theory (1)

Definition

An **instruction** is a description of a computational step.

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

label : 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 .

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.

Part 1: in theory (3)

► **Example:** roughly per [7, Chapter 11], consider a case where

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 available (abstract) instruction templates is

```
Li : Raddr ← Raddr + 1 then goto Li+1  
Li : Raddr ← Raddr - 1 then goto Li+1  
Li : if Raddr = 0 then goto Ltarget else goto Li+1  
Li : halt
```

► **Example:** now, given

1. a program comprised of (concrete) instruction instances 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.

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

► **Example:** now, given

1. a program comprised of (concrete) instruction instances 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:

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

► **Example:** now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

$$\begin{array}{lcl} C_{17} & = & (7, 0, 0, 1, 2) \\ L_7 & \leadsto & R_2 \leftarrow R_2 + 1 \text{ then goto } L_8 \\ C_{18} & = & (8, 0, 0, 2, 2) \end{array}$$

► **Example:** now, given

1. a program comprised of (concrete) instruction instances 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:

$$\begin{array}{lll} C_{18} & = & (8, 0, 0, 2, 2) \\ L_8 & \leadsto & \text{if } R_0 = 0 \text{ then goto } L_5 \text{ else goto } L_9 \\ C_{19} & = & (5, 0, 0, 2, 2) \end{array}$$

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

► **Example:** now, given

1. a program comprised of (concrete) instruction instances 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:

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

Part 1: in theory (4)

► Example: now, given

1. a program comprised of (concrete) instruction instances 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:

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

which demonstrates that the program copies R_2 into R_3 .

Part 1: in theory (5)

- ▶ **Question:** is this the *only* viable example?
- ▶ **Answer:** many **alternatives** exist.

Part 1: in theory (5)

- ▶ **Question:** is this the *only* viable example?
- ▶ **Answer:** many **alternatives** exist, stemming, e.g., from
 1. different models, e.g.,
 - ▶ **counter machine,**
 - ▶ **Random-Access Machine (RAM),**
 - ▶ **Random-Access Stored-Program (RASP) machine,**
 - ▶ ...

Part 1: in theory (5)

- ▶ **Question:** is this the *only* viable example?
- ▶ **Answer:** many **alternatives** exist, stemming, e.g., from

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

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

or “clear” instruction.

Part 1: in theory (5)

- ▶ **Question:** is this the *only* viable example?
- ▶ **Answer:** many **alternatives** exist, stemming, e.g., from

3. different instruction set format.

Part 1: in theory (5)

- ▶ **Question:** is this the *only* viable example?
- ▶ **Answer:** many **alternatives** exist, stemming, e.g., 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:

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

Part 1: in theory (5)

- ▶ **Question:** is this the *only* viable example?
- ▶ **Answer:** many **alternatives** exist, stemming, e.g., 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:

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

Part 1: in theory (5)

- ▶ **Question:** is this the *only* viable example?
- ▶ **Answer:** many **alternatives** exist, stemming, e.g., 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:

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

- ▶ **Question:** is this the *only* viable example?
- ▶ **Answer:** many **alternatives** exist, stemming, e.g., 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:

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

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

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

► Concept:

- One can view i.e.,

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

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

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

are different, valid interpretations of the same number.

► Concept:

► One can view

(human-readable) instruction $\approx$ abstraction of (machine-readable) control information.

► Concept:

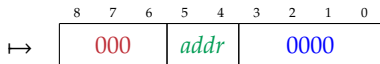
- These facts suggest a strategy: an RM is an FSM in disguise, in the sense that
 1. an RM configuration is an FSM state, and
 2. the RM program determines the FSM transition function,so we could therefore
 - use, e.g., Gödel encoding or variant thereof, to encode instructions into numerical **machine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

Part 2: in practice (2)

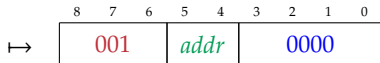
Design

- **Design:** specify an instruction encoding, e.g.,

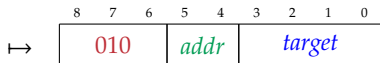
$L_i : R_{addr} \leftarrow R_{addr} + 1$ then goto L_{i+1}



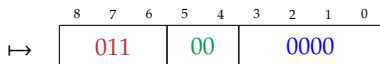
$L_i : R_{addr} \leftarrow R_{addr} - 1$ then goto L_{i+1}



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



$L_i : \text{halt}$



such that

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

► Observation(s):

- we opted to encode instructions as 9-element sequences of bits, noting that

$$000010000_{(2)} \equiv 010_{(16)} \equiv 16_{(10)}$$

and that

$$\begin{array}{ll} 000010000_{(2)} & \mapsto \text{“increment register one” if it represents an instruction} \\ 000010000_{(2)} & \mapsto \text{“the integer sixteen” if it represents some data} \end{array}$$

► Observation(s):

- we opted for a means of encoding that could be described as

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

so that decoding means extraction of contiguous bits from $\hat{x}$: i.e.,

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

- so this is *similar* to Gödel encoding, but *much* more efficient.

► Observation(s):

- this design clearly isn't the *only* one possible:
 - some aspects stem from *constraints*:
 - with $2^2 = 4$ registers, we need at least a 2-bit green field,
 - ...
 - some aspects stem from *choices*:
 - a 4-bit blue field limits our programs to $2^4 = 16$ instructions,
 - the assignment of values that identify instruction types is somewhat arbitrary,
 - the order and location of fields within the bit-sequence is somewhat arbitrary,
 - ...

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

such that

- we use

$$\text{MEM} = \langle 0A5_{(16)}, 060_{(16)}, \dots, 0C0_{(16)} \rangle,$$

a 10-element memory,

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

Definition

The **stored program** concept implies that a mutable program is stored in some form of memory. A **stored program computer** is said to **execute** a stored program, and so the instructions within it: it does so by loading (or fetching) those instructions, before decoding and executing them.

Definition

The **Program Counter (PC)** (or **instruction pointer**) is a special-purpose register that holds the address of the next instruction to be executed.

Definition

The **Instruction Register (IR)** is a special-purpose register that holds the instruction currently being executed.

Definition

An **instruction decoder** is typically a combinatorial component, either way tasked with taking an encoded instruction as input and producing a set of associated control signals as output; said control signals are used by the rest of the implementation to control execution of the instruction.

Definition

The **fetch-decode-execute cycle** (aka. **instruction cycle**) is a 3-stage process

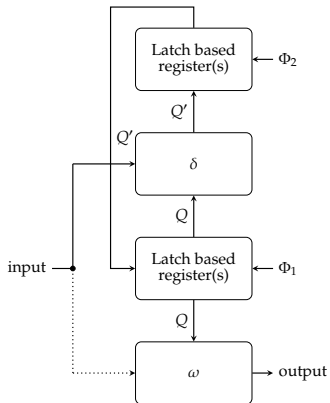
1. fetch stage : $\left\{ \begin{array}{ll} 1.a. & \text{load instruction into IR} \quad \mapsto \quad \text{IR} \leftarrow \text{MEM}[\text{PC}] \\ 1.b. & \text{increment PC} \quad \mapsto \quad \text{PC} \leftarrow \text{PC} + 1 \end{array} \right.$
2. decode stage : $\left\{ \begin{array}{l} \text{decide what instruction in IR means, i.e.,} \\ \text{translate IR into control signals which reflect instruction semantics} \end{array} \right.$
3. execute stage : $\left\{ \begin{array}{l} \text{do whatever instruction in IR means, i.e.,} \\ \text{apply instruction semantics} \end{array} \right.$

which describes execution of instructions; in some cases it makes sense to consider a 5-stage process by adding

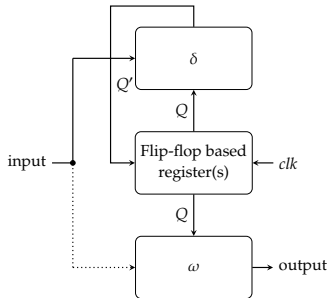
4. memory access stage : $\{ \text{perform any memory accesses (e.g., loads or stores) required} \}$
5. write-back (or commit) stage : $\{ \text{store result(s) stemming from instruction execution (e.g., computation)} \}$

i.e., expanding the execute stage to be more precise.

Recall



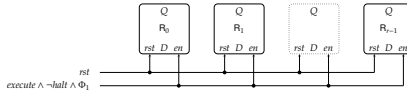
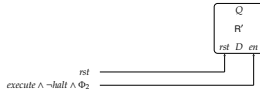
Recall



Part 2: in practice (9)

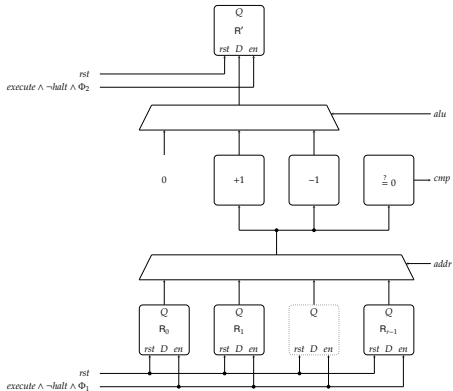
(High-level) implementation: data-path

Circuit



(High-level) implementation: data-path

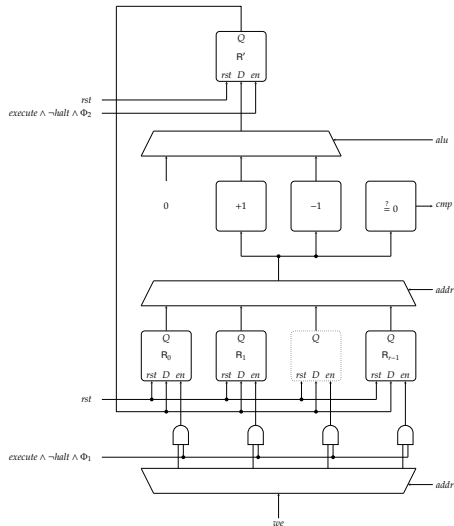
Circuit



Part 2: in practice (9)

(High-level) implementation: data-path

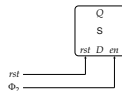
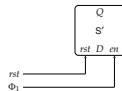
Circuit



Part 2: in practice (10)

(High-level) implementation: control-path

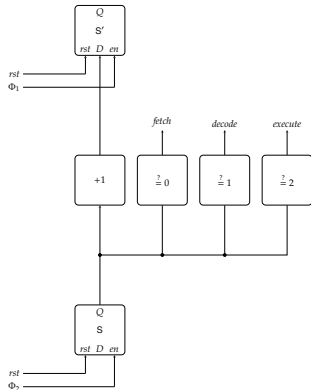
Circuit



Part 2: in practice (10)

(High-level) implementation: control-path

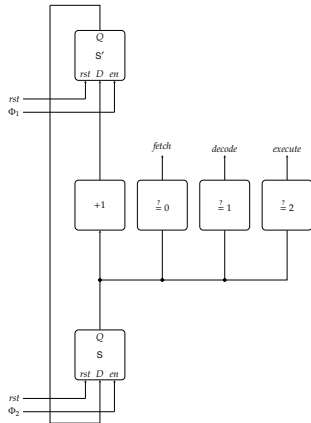
Circuit



Part 2: in practice (10)

(High-level) implementation: control-path

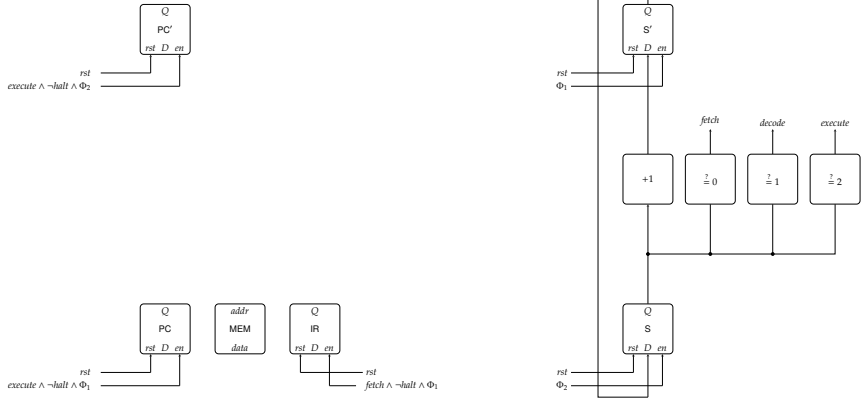
Circuit



Part 2: in practice (10)

(High-level) implementation: control-path

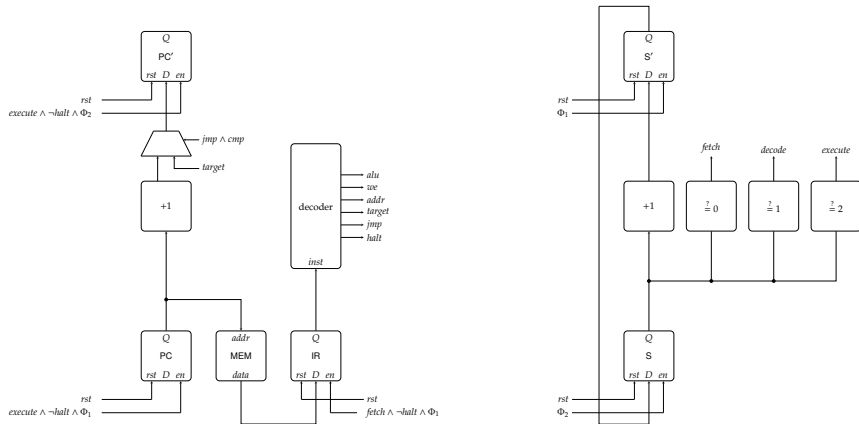
Circuit



Part 2: in practice (10)

(High-level) implementation: control-path

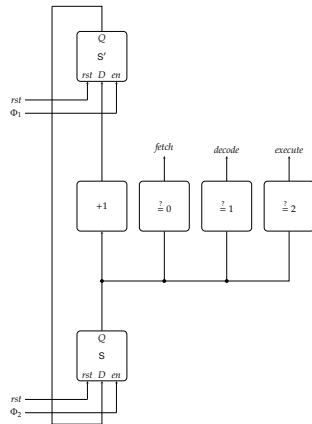
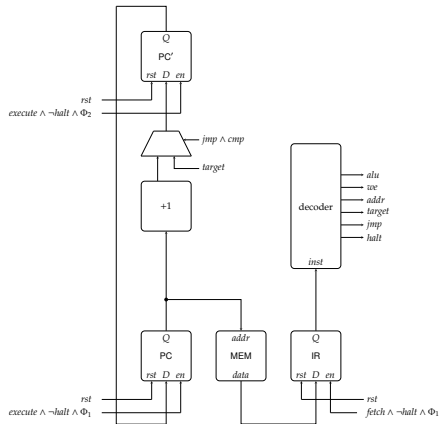
Circuit



Part 2: in practice (10)

(High-level) implementation: control-path

Circuit



Conclusions

► Take away points:

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

combinatorial logic	↷	fixed function, not stateful
sequential logic	↷	fixed function, stateful
FSM	↷	fixed function, stateful
RM	↷	not fixed function, stateful
	⋮	
micro-processor	↷	not fixed function, stateful

and although our register 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.

Additional Reading

- ▶ *Wikipedia: Register machine*. URL: https://en.wikipedia.org/wiki/Register_machine.
- ▶ *Wikipedia: Counter machine*. URL: https://en.wikipedia.org/wiki/Counter_machine.
- ▶ *Wikipedia: Random-access machine*. URL: 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.
- ▶ *Wikipedia: Gödel numbering*. URL: https://en.wikipedia.org/wiki/G%C3%B6del_numbering.
- ▶ *Wikipedia: Machine code*. URL: https://en.wikipedia.org/wiki/Machine_code.

References

- [1] *Wikipedia: Counter machine*. URL: https://en.wikipedia.org/wiki/Counter_machine (see p. 60).
- [2] *Wikipedia: Gödel numbering*. URL: https://en.wikipedia.org/wiki/G%C3%B6del_numbering (see p. 60).
- [3] *Wikipedia: Machine code*. URL: https://en.wikipedia.org/wiki/Machine_code (see p. 60).
- [4] *Wikipedia: Random-access machine*. URL: https://en.wikipedia.org/wiki/Random-access_machine (see p. 60).
- [5] *Wikipedia: Random-access stored-program machine*. URL: https://en.wikipedia.org/wiki/Random-access_stored-program_machine (see p. 60).
- [6] *Wikipedia: Register machine*. URL: https://en.wikipedia.org/wiki/Register_machine (see p. 60).
- [7] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967 (see pp. 1, 4).
- [8] E. Nagel and J.R. Newman. *Gödel's Proof*. Routledge, 1958 (see p. 36).