

Moon-1 (I)

CS31

Pascal Van Hentenryck



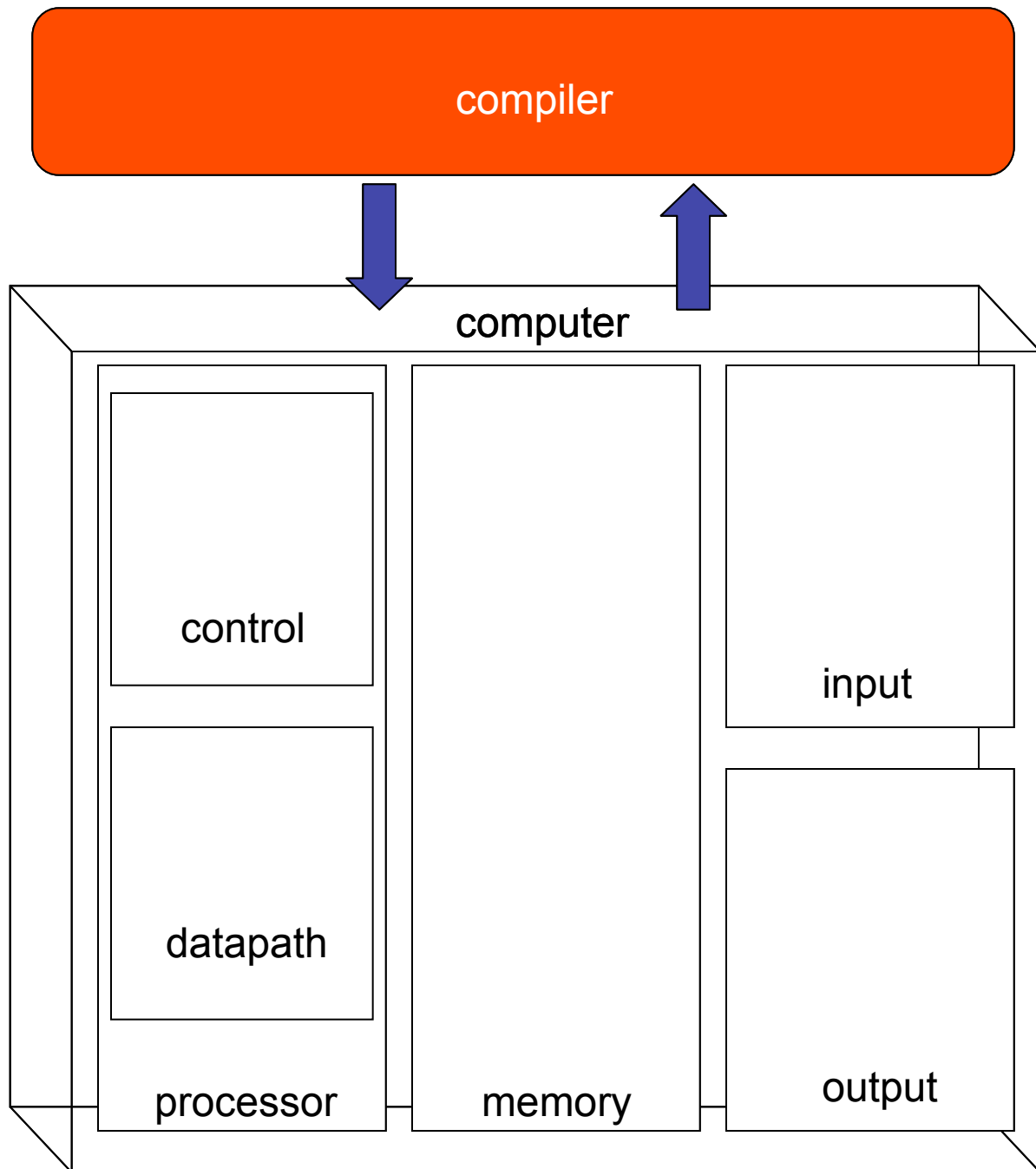
Overview

Moon I (I)

- A simple computer model
- A simple assembly language

Assembler

The Big Picture



Abstraction Hierarchy

Programming Language

Assembly Language

Machine Language

Sequential Circuit

Combinational Circuit

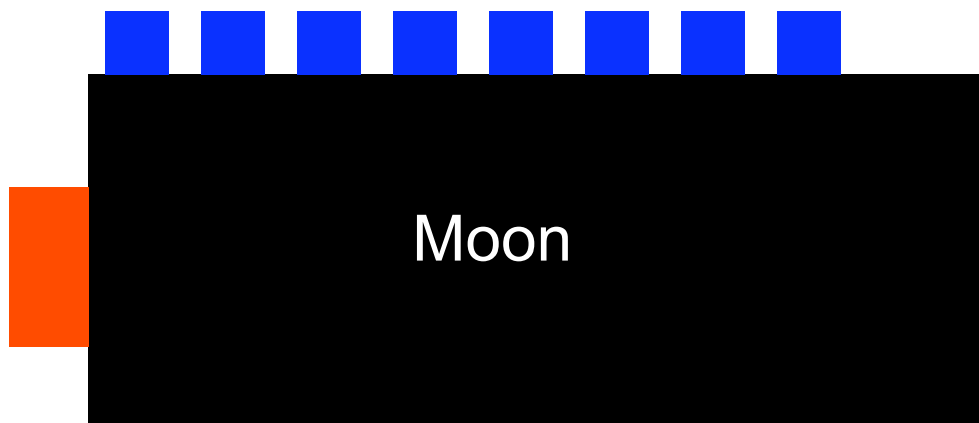
Binary Value

Voltage

Moon

Functionality

- 4 8-bit words of RAM
- 1 8-bit register called the accumulator
- 8 lights (output)
- An input unit (to load your program)



Moon Assembly Language

Assembly Language Program

- A sequence of instructions

```
<inst_1>  
<inst_2>  
<inst_3>  
...  
<inst_n>
```

Control Flow

- Unless specified otherwise, instructions are executed sequentially

Four types of instructions

- Load/store instructions
- Arithmetic instructions
- Input/output instructions
- Control instructions

Load/Store Instructions

Goals

- Read from memory
- Write to memory

Moon Instructions

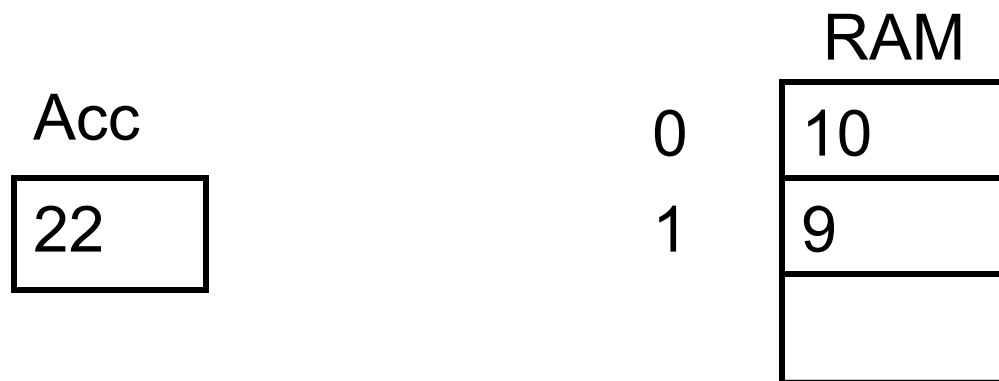
LAD a: load the accumulator
 with the data stored
 at address a in RAM

ACC := RAM[a];

SAD a: store the value of
 the accumulator into
 the RAM at address a

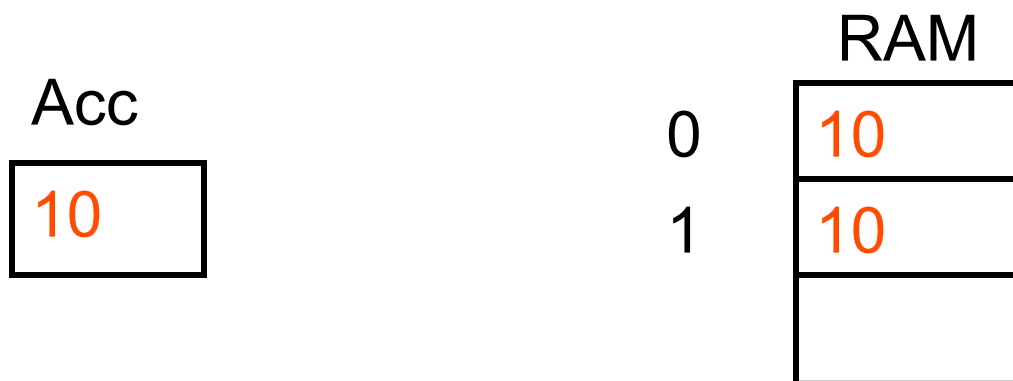
RAM[a] := ACC;

Load/Store Instruction



Program

LAD 0
SAD 1



Load Immediate

Goal

- Initialize the accumulator with a value

Moon Instruction

LIM value:

store value in the accumulator

```
ACC := value;
```

Arithmetic Instructions

Moon Instructions

ADD a:

add the value stored at
address a in the RAM to the
value of the accumulator
and put the result in the
accumulator

```
ACC := ACC + RAM[a];
```

SUB a:

Subtract the data stored at
address a in the RAM from
the value of the
accumulator and put the
result in the accumulator

```
ACC := ACC - RAM[a]
```

Arithmetic Instructions

Acc

22

Program

```
LIM 0
ADD 0
ADD 1
ADD 2
SAD 3
```

Acc

6

RAM

0	1
1	2
2	3
3	123

RAM

0	1
1	2
2	3
3	6

I/O Instructions

Goals

- Communication with the environment

Moon Instructions

SOU:

Store the value in the accumulator into the output register (turning on the appropriate lights)

```
OUTPUT := ACC;
```

More on Assembly Languages

Memory Labels

- We use symbolic names for the memory location in the RAM; not their addresses

Instruction Labels

- We can use labels to name the instructions as well

Example

```
start:      LIM    10
            SAD    limit
            LIM    1
            SAD    one
Loop:      LAD    0
            SUB    one
            SAD    limit
```

Control Instructions

Goals

- Modifying the flow of control

Moon Instructions

JZ label:

if the accumulator has the value
zero, execute instruction labeled
label. Otherwise execute the next
instruction

Example: Print 1 if the accumulator does not
contain zero; print 0 otherwise

```

                                JZ     zero_c
                                LIM     1     -
                                SOU
                                LIM     0
                                JZ     endif
zero_c:                        SOU
endif:                          ...
```

A Complete Program

```
start:      LIM    10
            SAD    limit
            LIM    0
            SAD    count
            LIM    1
            SAD    one
            LAD    limit
            SAD    count
loop:       LAD    count
            SOU
            JZ     start
            SUB    one
            SAD    count
            LIM    0
            JZ     loop
```

Moon Machine Language

Remember

- A program is a sequence of numbers
- These numbers are stored in memory

The problem to solve

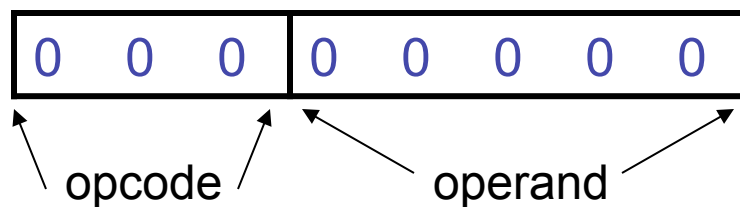
- How to move from assembly language to machine language

Key Operations

- Replace memory labels
- Replace instruction labels
- Replace opcodes

Opcodes: Instruction Format

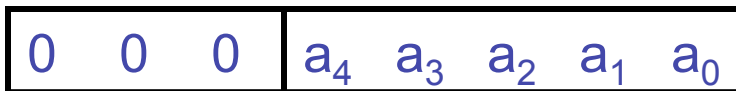
Instructions are 8 bits, with a 3-bit *opcode* and a 5-bit *operand*



- Opcode tells us what to do
- Operand tells us what to do to it

Opcodes: Instruction Format

LAD addr



LIM value



SAD addr



SOU



SUB addr



JZ addr



Removing Memory Labels

Simple scheme here

- Replace each label by an actual address

LIM 10	LIM 10
SAD limit	SAD 0
LIM 1	LIM 1
SAD one	SAD 1

Remove the opcodes

Binary	Hexa	Assembly
01001010	4A	LIM 10
01100000	60	SAD 0
01000001	41	LIM 1
01100001	61	SAD 1

Dealing with Jumps

In Moon the program will be stored in memory starting at address 0

0	Instruction 1
1	Instruction 2
2	Instruction 3

Dealing with Jumps

```

        JZ     zero_c
        LIM   1
        SOU
        LIM   0
        JZ     endif
zero_c: SOU
endif:  ...

```

Binary	Hexa	Assembly
11000101	C5	JZ zero_c
01000001	41	LIM 1
10000000	80	SOU
01000000	40	LIM 0
11000110	C6	JZ endif
10000000	80	zero_c: SOU