

T eaching L ondon C omputing

CAS London CPD Day 2016

Little Man Computer



William Marsh
School of Electronic Engineering and Computer Science
Queen Mary University of London

Overview and Aims

- LMC is a computer simulator
 - ... understanding how a computer work
 - To program the LMC, must understand:
 - Memory addresses
 - Instructions
 - Fetch-execute cycle
 - *Practical exercises*
 - What we can learn from LMC
-

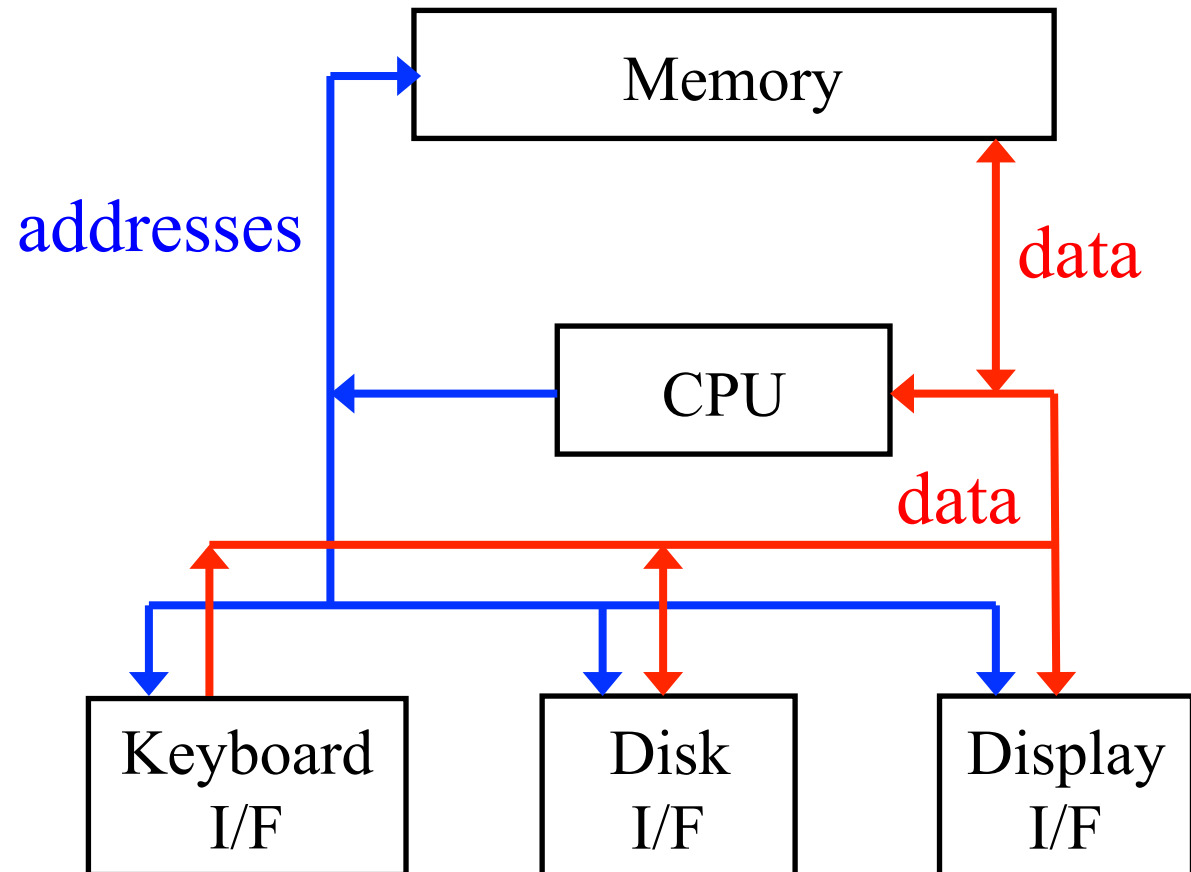


What is in a Computer?

- Memory
 - CPU
 - I/O
-

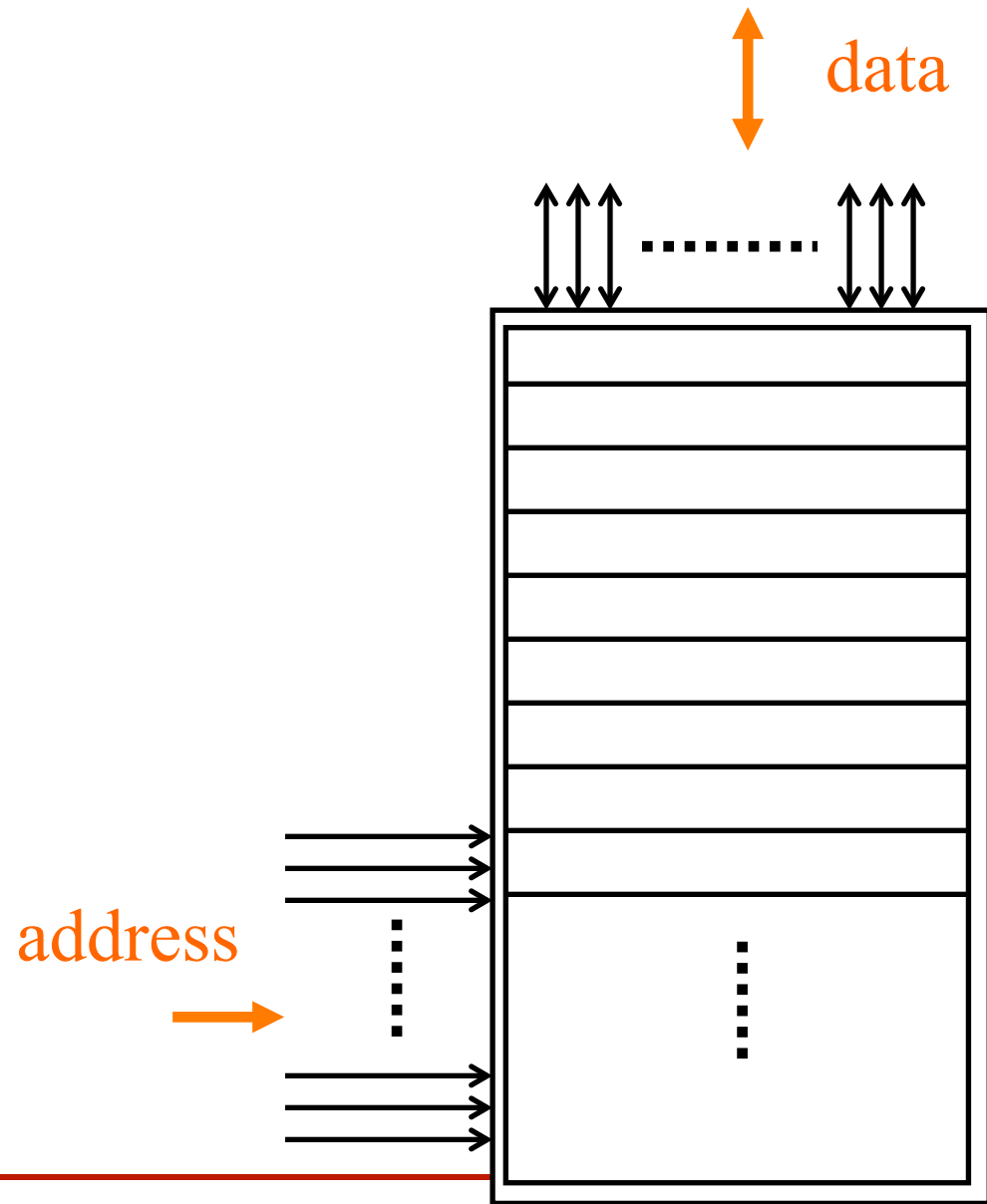
Simple Computer

- Processor
 - CPU
- Memory
 - Data
 - Program instructions
- I/O
 - Keyboard
 - Display
 - Disk



Memory

- Each location
 - has an address
 - hold a value
- Two interfaces
 - address – which location?
 - data – what value?

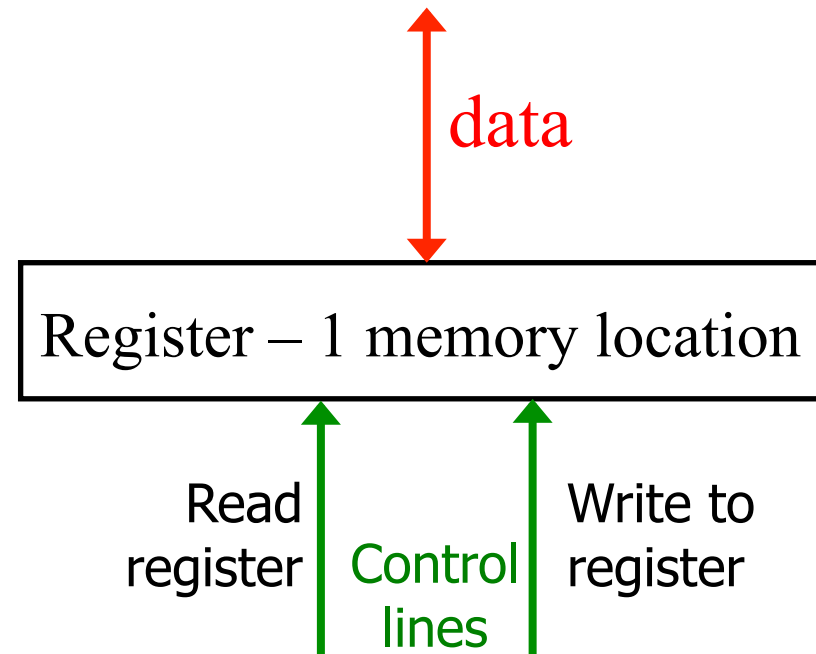




Quiz – What is the Memory?

Registers (or Accumulators)

- A storage area inside the CPU
- VERY FAST
- Used for arguments and results to one calculation step



Assembly Language Code

```
INP          00 INP
STA 10       01 STA 10
INP          02 INP
ADD 10       03 ADD 10
STA 11       04 STA 11
OUT          05 OUT
            06
HALT
```

Write a program here

OUTPUT

110

I/O

CPU

CPU

PROGRAM COUNTER 07

INSTRUCTION REGISTER 0

ADDRESS REGISTER 00

ACCUMULATOR 110

ARITHMETIC UNIT

INPUT

11

I/O

V1.3 Little Man Computer

RAM

0	1	2	3	4	5	6	7	8	9	
901	310	901	110	311	902	000	000	000	000	
10	11						16	17	18	19
099	110						00	000	000	000
20	21	22	23	24	25	26	27	28	29	
000	000	000	000	000	000	000	000	000	000	
30	31	32	33	34	35	36	37	38	39	
000	000	000	000	000	000	000	000	000	000	
40	41	42	43	44	45	46	47	48	49	
000	000	000	000	000	000	000	000	000	000	
50	51	52	53	54	55	56	57	58	59	
000	000	000	000	000	000	000	000	000	000	
60	61	62	63	64	65	66	67	68	69	
000	000	000	000	000	000	000	000	000	000	
70	71	72	73	74	75	76	77	78	79	
000	000	000	000	000	000	000	000	000	000	
80	81	82	83	84	85	86	87	88	89	
000	000	000	000	000	000	000	000	000	000	
90	91	92	93	94	95	96	97	98	99	
000	000	000	000	000	000	000	000	000	000	

Memory

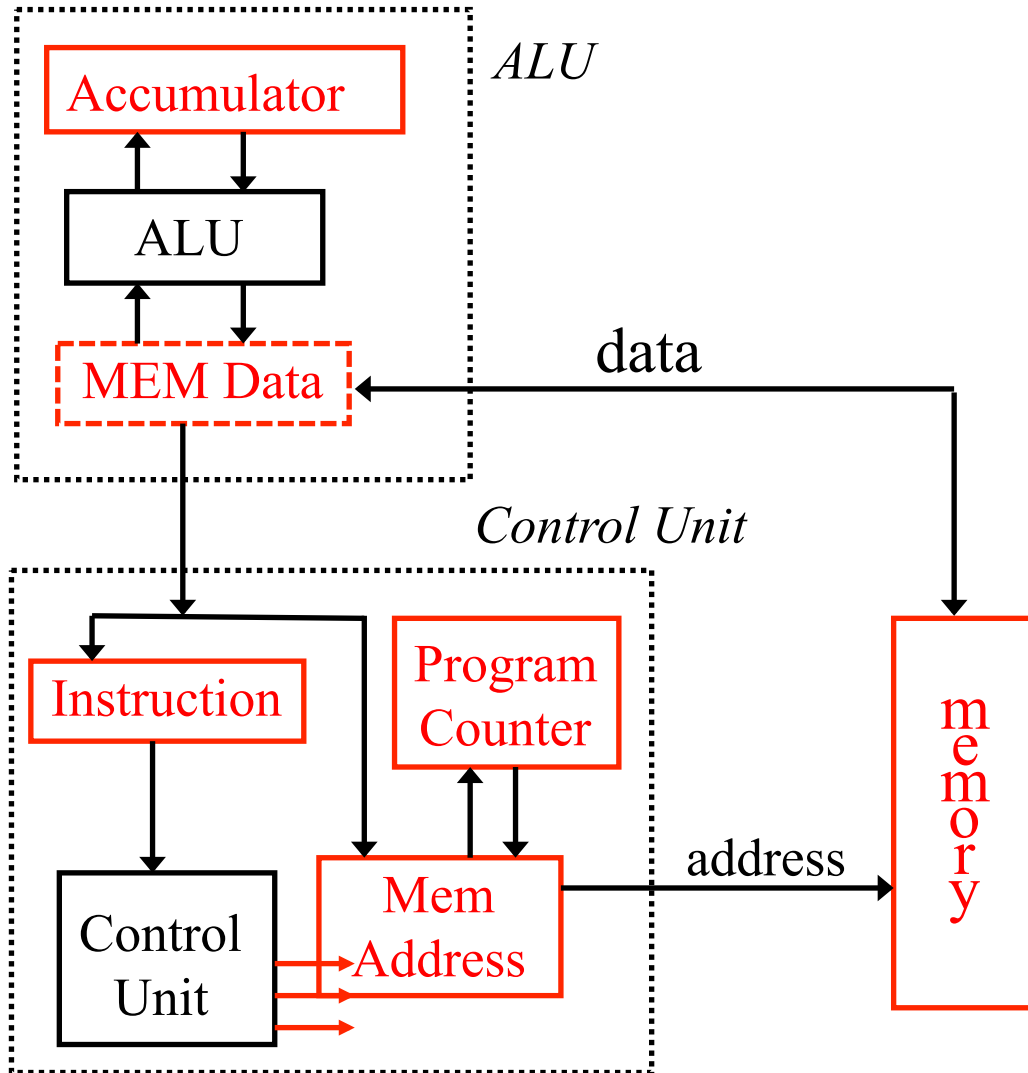
ASSEMBLE INTO RAM RUN STEP

RESET LOAD HELP SELECT

Program HALTED, RESET, LOAD, SELECT or alter memory

OPTIONS ©GCSEcomputing.org.uk and Peter Higginson

LMC CPU Structure



- Visible registers shown in red
- Accumulators
 - Data for calculation
- *Data*
 - *Word to/from memory*
- PC
 - Address of next instruction
- Instruction
- Address
 - For memory access



Instructions

The primitive language of a computer

Instructions

OpCode	Address
--------	---------

- Instruction
 - What to do: Opcode
 - Where: memory address
- Instructions for arithmetic
 - Add, Multiply, Subtract
- Memory instructions
 - LOAD value from memory
 - STORE value in memory

- The instructions are very simple
- Each make of computer has different instructions
- Programs in a high-level language can work on all computers

Instructions

OpCode	Address
--------	---------

- Opcode: 1 decimal digit
- Address: two decimal digits – xx
- Binary versus decimal

Code	Name	Description
000	HLT	Halt
1xx	ADD	Add: acc + memory → acc
2xx	SUB	Subtract: acc – memory → acc
3xx	STA	Store: acc → memory
5xx	LDA	Load: memory → acc
6xx	BR	Branch always
7xx	BRZ	Branch is acc zero
8xx	BRP	Branch if acc > 0
901	IN	Input
902	OUT	Output

Add and Subtract Instruction

ADD	Address
SUB	Address

- One address and accumulator (ACC)
 - Value at address combined with accumulator value
 - Accumulator changed
 - **Add:** $ACC \leftarrow ACC + \text{Memory}[\text{Address}]$
 - **Subtract:** $ACC \leftarrow ACC - \text{Memory}[\text{Address}]$
-

Load and Store Instruction

LDA	Address
STA	Address

- Move data between memory and accumulator (ACC)
 - **Load:** $ACC \leftarrow \text{Memory}[\text{Address}]$
 - **Store:** $\text{Memory}[\text{Address}] \leftarrow ACC$
-

Input and Output

INP	1 (<i>Address</i>)
OUT	2 (<i>Address</i>)

- **Input:** $ACC \leftarrow \text{input value}$
 - **output:** $\text{output area} \leftarrow ACC$

 - It is more usual for I/O to use special memory addresses
-

Branch Instructions

BR	Address
----	---------

- Changes program counter
 - May depend on accumulator (ACC) value
 - **BR**: $PC \leftarrow \text{Address}$
 - **BRZ**: if $ACC == 0$ then $PC \leftarrow \text{Address}$
 - **BRP**: if $ACC > 0$ then $PC \leftarrow \text{Address}$
-

Assembly Code

- Instructions in text
- Instruction name: STA, LDA
- Address: name using DAT

Numbers

- Memory holds numbers
- Opcode: 0 to 9
- Address: 00 to 99

Line

1	INP
2	STA x
3	INP
4	STA y
5	HLT
6	x DAT
7	y DAT

ASSEMBLE



Location

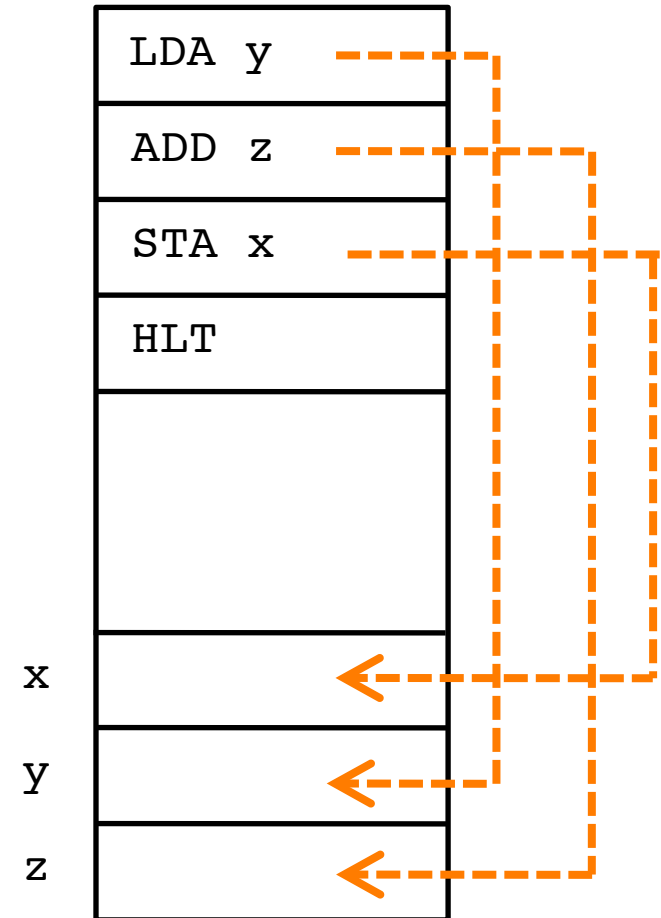
00	9 01
01	3 05
02	9 01
03	3 06
04	0 00
05	<i>(used for x)</i>
06	<i>(used for y)</i>



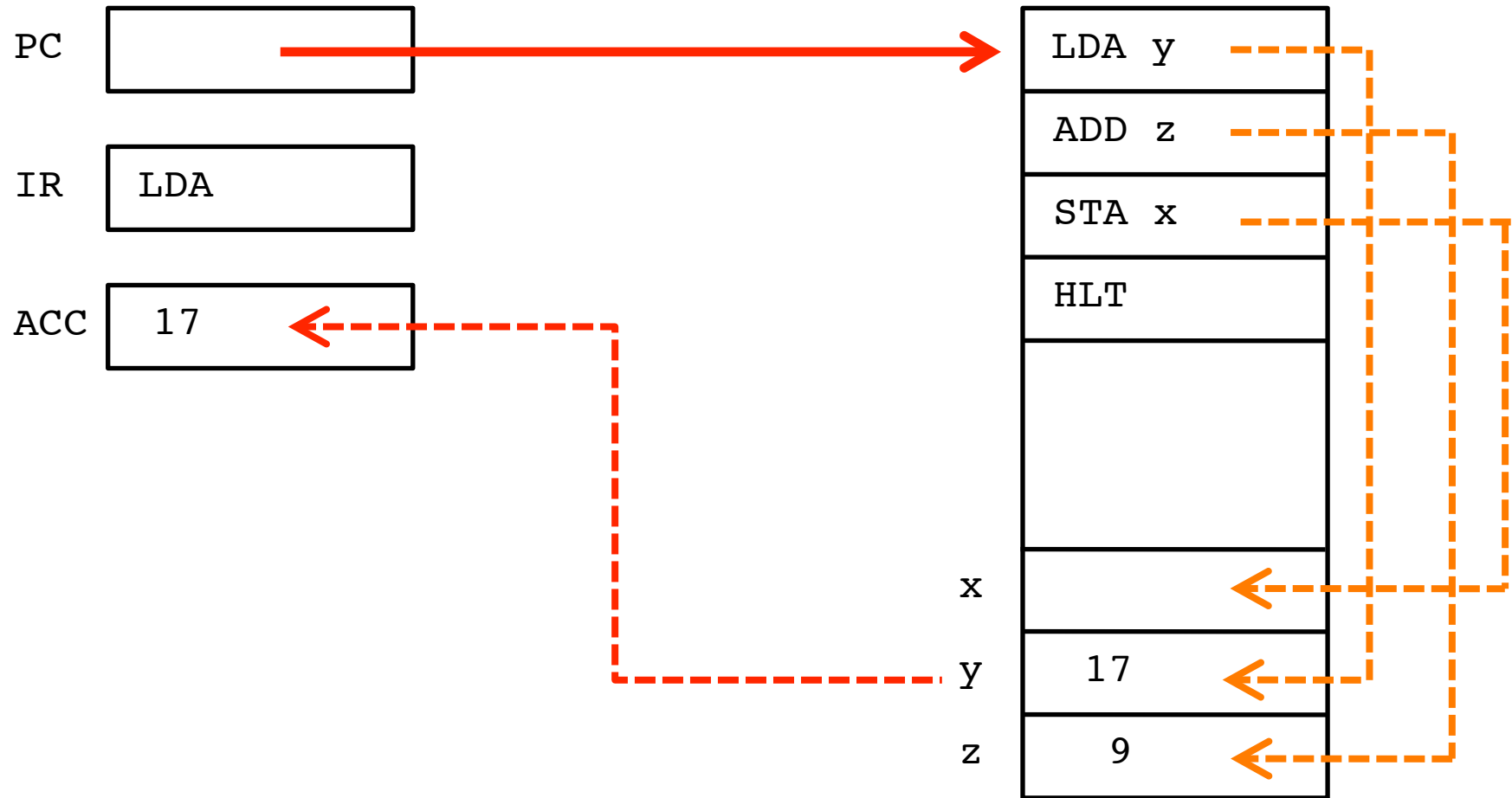
LMC Example

Simple Program

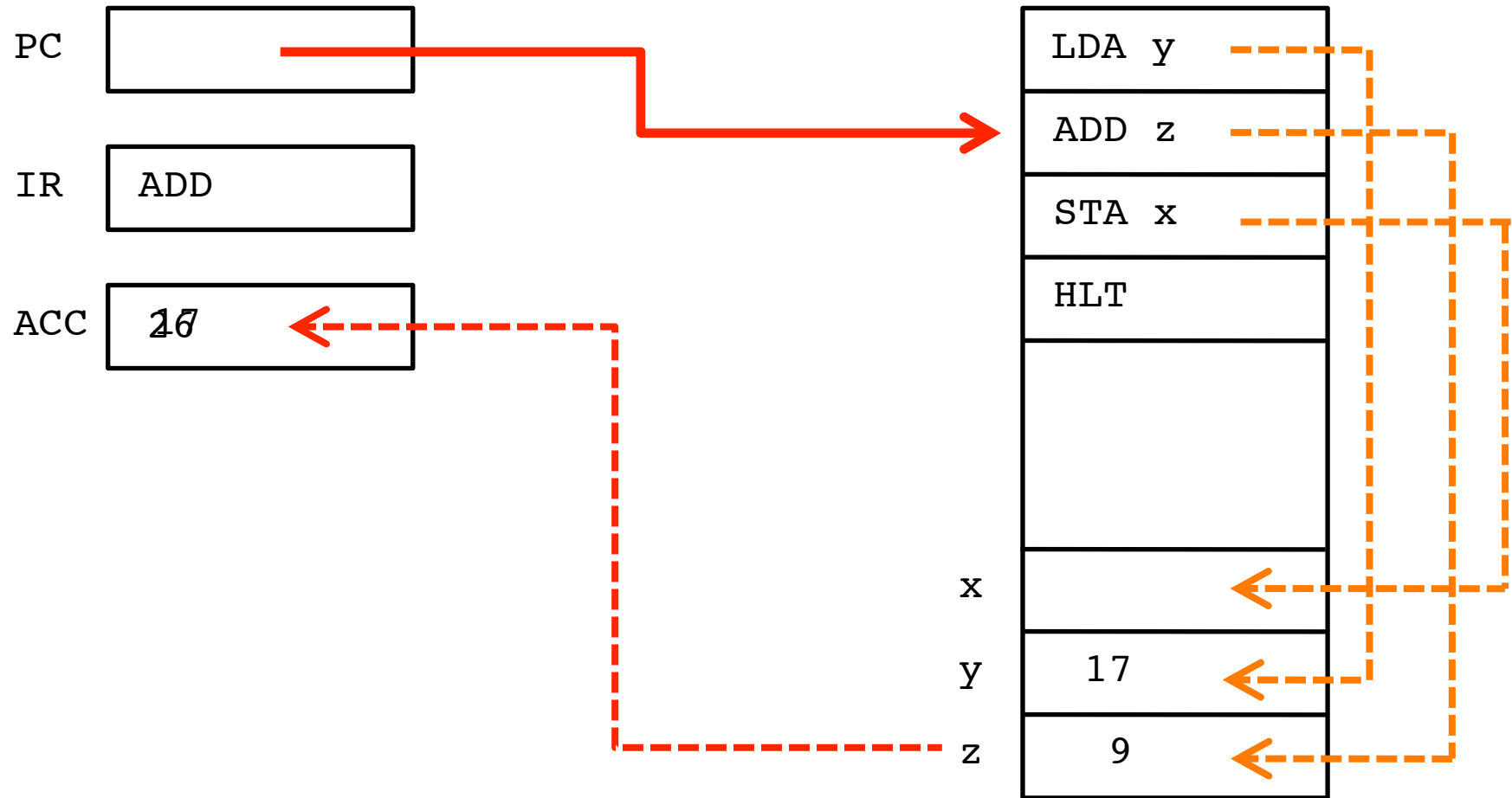
- $x = y + z$



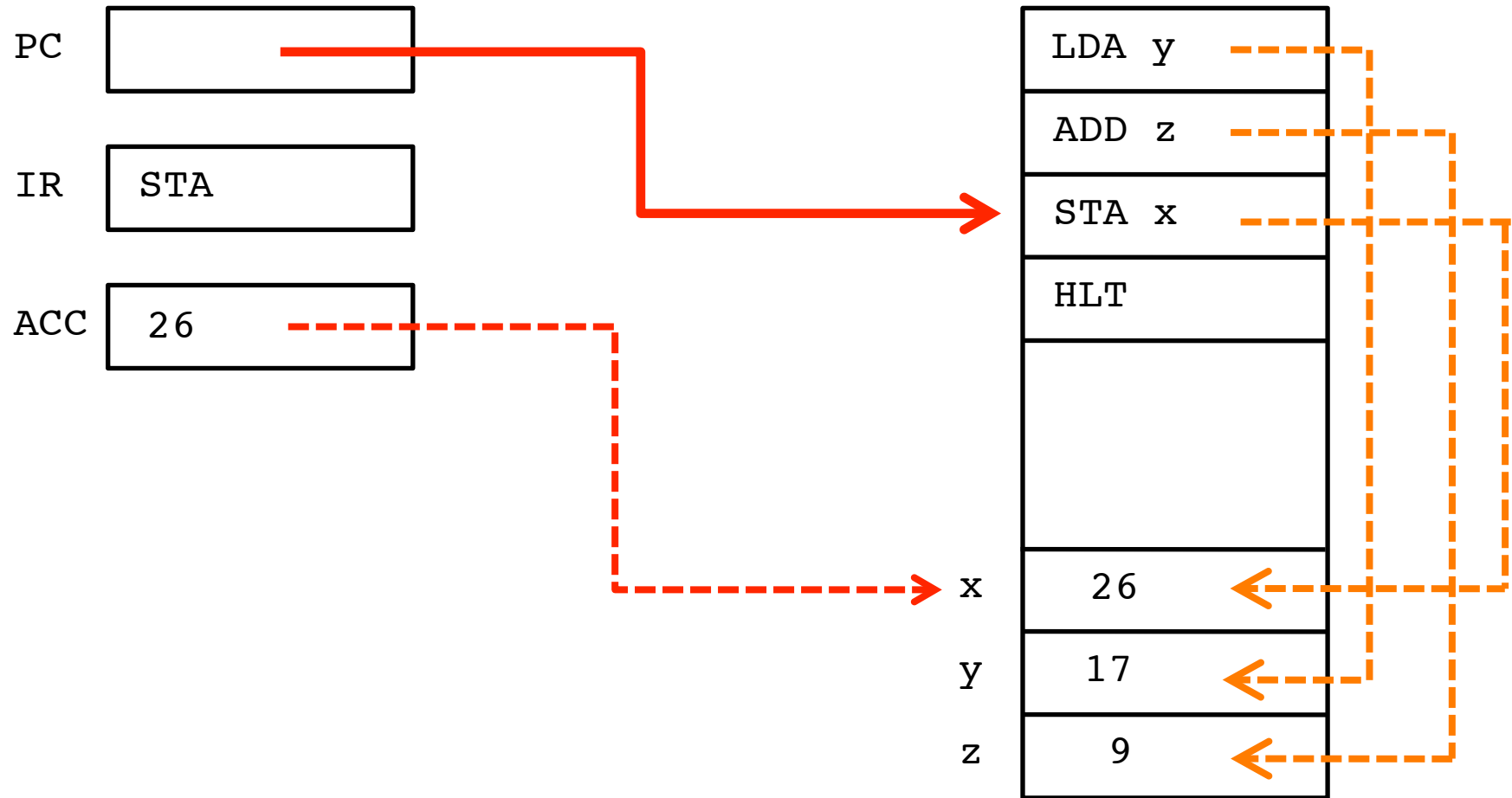
Running the Simple Program



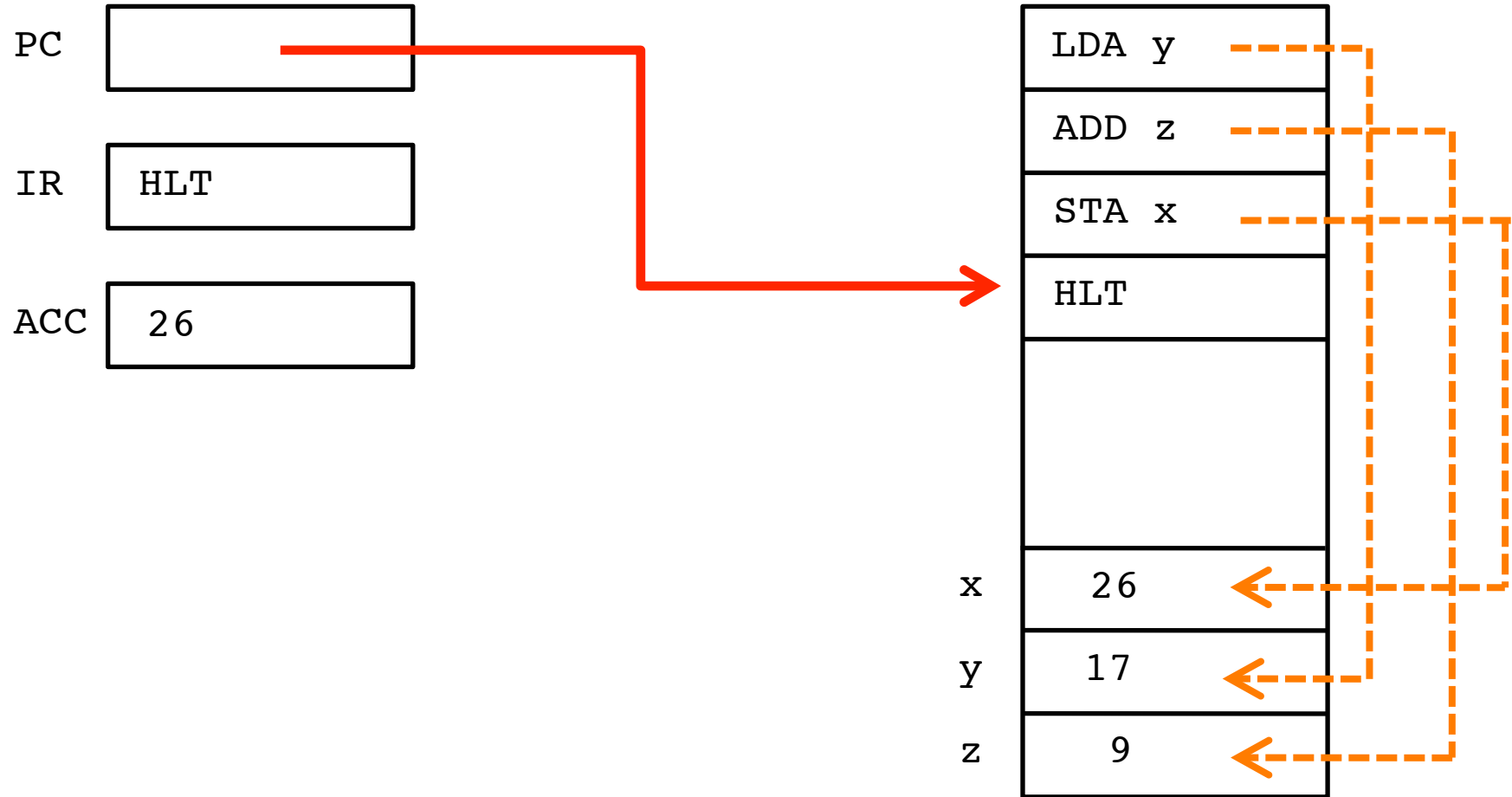
Running the Simple Program



Running the Simple Program



Running the Simple Program



Practice Exercises

- Try the first three exercises on the practical sheet
-

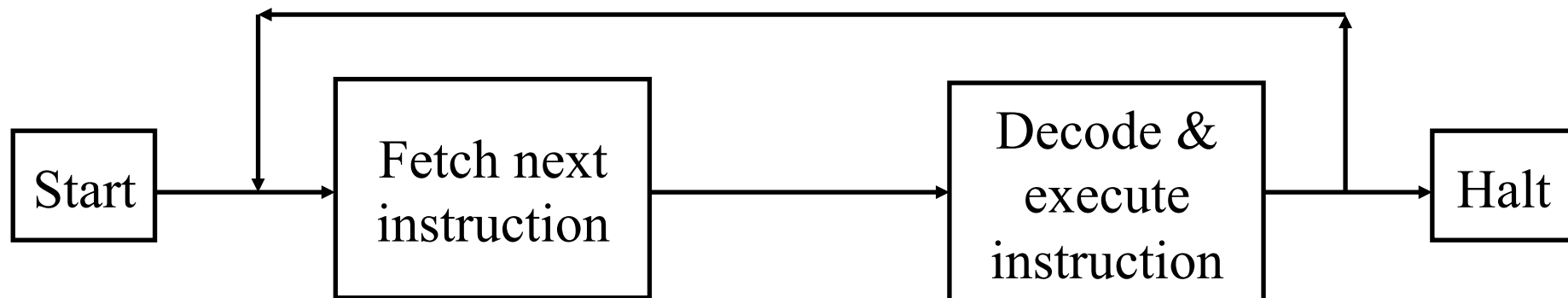


Fetch-Execute Cycle

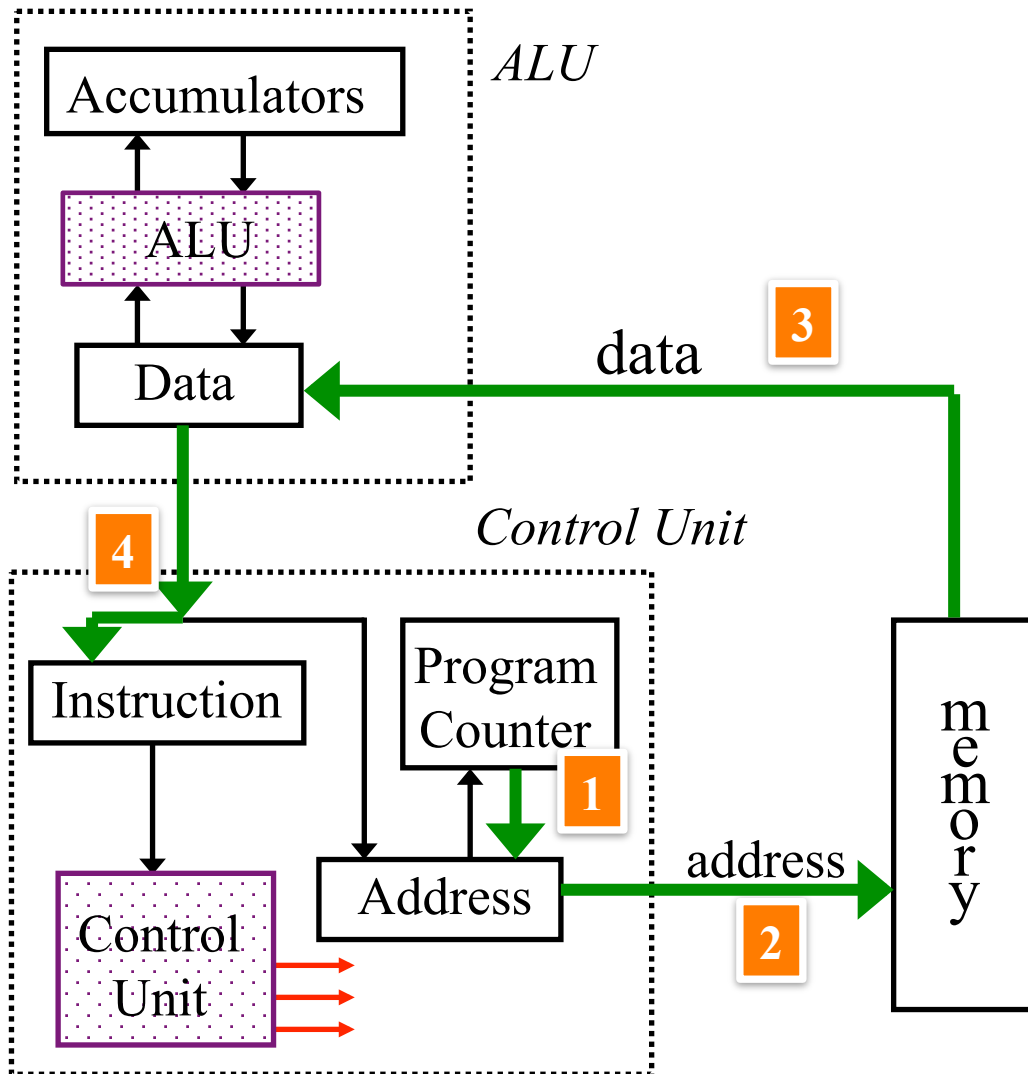
How the Computer Processes Instructions

Fetch-Execute

- Each instruction cycle consists on two subcycles
- Fetch cycle
 - Load the next instruction (Opcode + address)
 - Use Program Counter
- Execute cycle
 - Control unit interprets the opcode
 - ... an operation to be executed on the data by the ALU

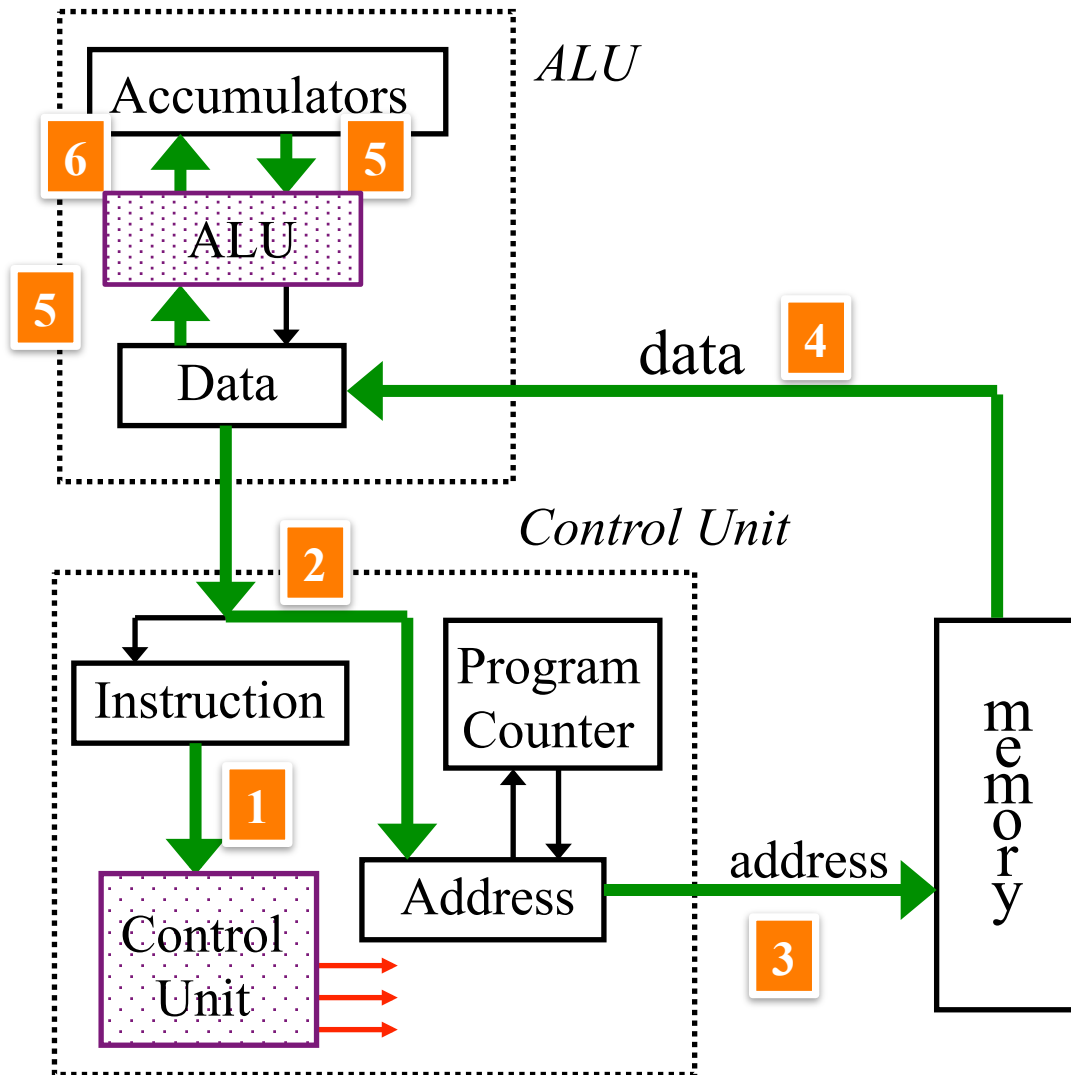


Fetch Instruction



1. Program counter to address register
2. Read memory at address
3. Memory data to 'Data'
4. 'Data' to instruction register
5. Advance program counter

Execute Instruction



1. Decode instruction
2. Address from instruction to 'address register'
3. Access memory
4. Data from memory to 'data register'
5. Add (e.g.) data and accumulator value
6. Update accumulator



What We Can Learn from LMC

1. How programming language work
 2. What a compiler does
 3. Why we need an OS
-

Understanding Variables and Assignment

- What is a variable?
- What is on the left hand side of:

$$x = x + 1$$

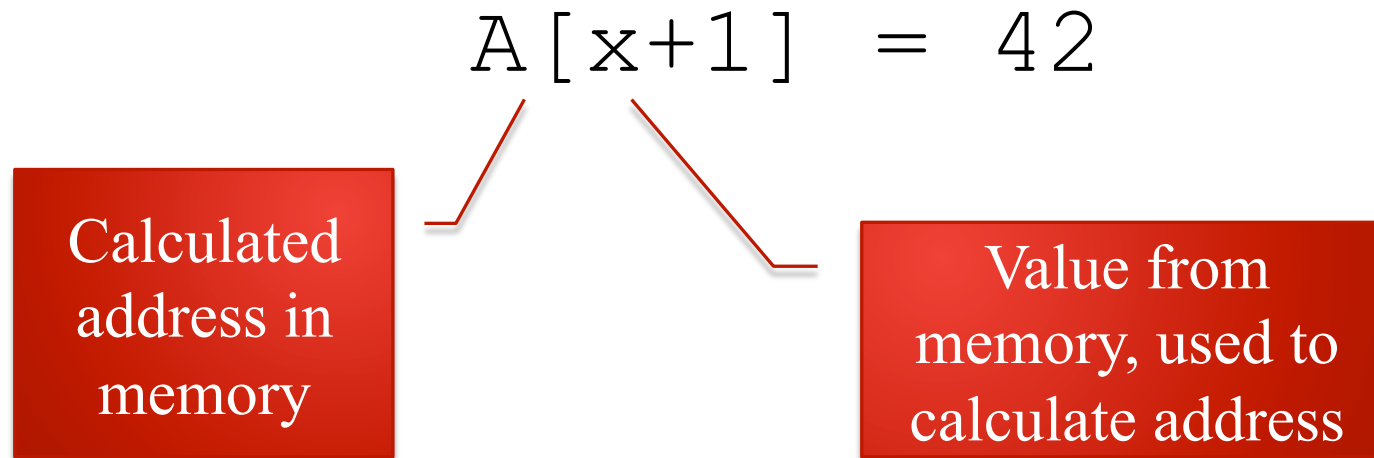
Variable is
an address in
memory

Value from
memory
(at address x)



Understanding Variables and Assignment

- What is a variable?
- What is on the left hand side of:



Understanding If and Loops

- Calculate the address of the next instruction

```
if x > 42:
```

Choose PC (L1 or L2)
from comparison

Instructions
at address L1

```
    large = large + 1
```

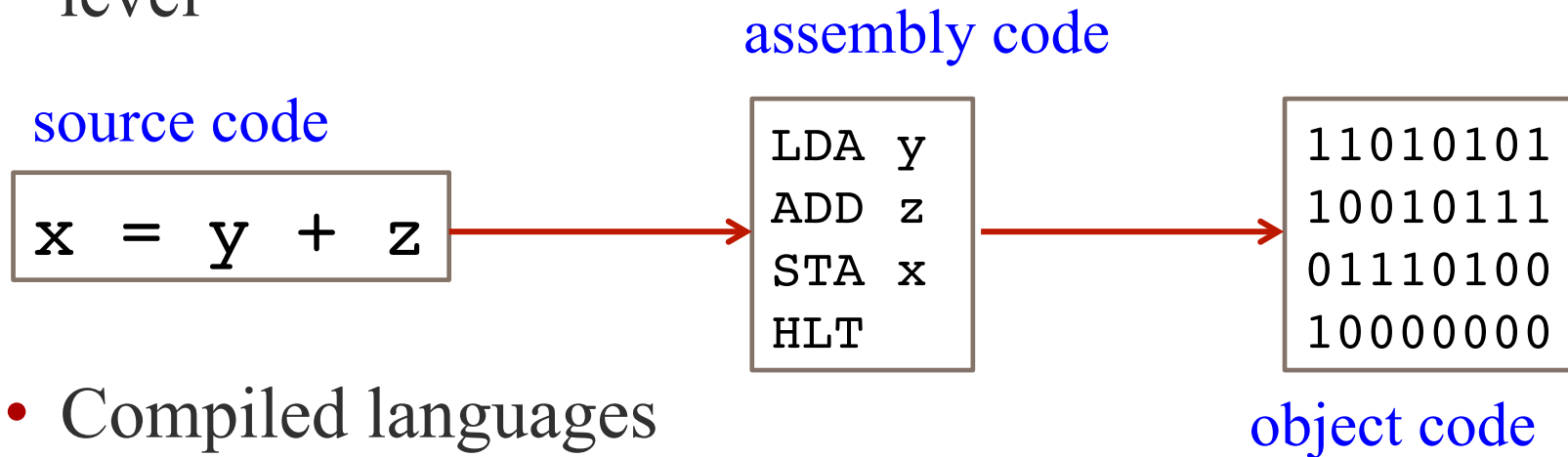
```
else:
```

Instructions at
address L2

```
    small = small + 1
```

Compiler

- Compiler **translates** high level program to low level



- Compiled languages
 - Statically typed
 - Close to machine
 - Examples: C, C++, (Java)
 - Compiler for each CPU
-

Why We Need An OS

LMC

- Only one program
- Program at fixed place in memory
- No
 - Disk
 - Screen
 - ...

Real Computer

- Many programs at once
 - Program goes anywhere in memory
 - Complex I/O
-

Summary of CPU Architecture

- Memory contains data and program
 - Program counter: address of next instruction
 - Instructions represented in binary
 - Each instruction has an ‘opcode’
 - Instructions contain addresses
 - Addresses used to access data
 - Computer does ‘fetch-execute’
 - ‘Execute’ depends on opcode
 - Computer can be built from $< 10,000$ electronic switches (transistors)
-



Project: Writing an LMC Interpreter

Write a Simple LMC Emulator

```
def readMem(memory):  
    global mdr  
    mdr = memory[mar]
```

```
acc = 0  
mdr = 0  
mar = 0  
pc = 0  
memory = [504, 105, 306, 0,  
          11, 17, ...]
```

State of the
LMC: registers
and memory

```
def execute(memory, opcode, arg):  
    global acc, mar, mdr, pc  
    if opcode == ADD:  
        mar = arg  
        readMem(memory)  
        acc = acc + mdr  
    elif opcode == SUB:  
        mar = arg  
        readMem(memory)  
        acc = acc - mdr  
    ...
```

```
def fetch(memory):  
    global pc, mar  
    mar = pc  
    pc = pc + 1  
    readMem(memory)
```

Update state
following rules