ITI 1121. Introduction to Computing II

Essential **computer architecture** concepts

by

Marcel Turcotte

Version January 6, 2020

Preambule

Preamble

Overview

Overview

Essential computer architecture concepts

We review the essential concepts of computer architecture: von Neumann's model, memory, and compilation. We simulate the execution of a machine language program using a didactic model of a microprocessor.

General objective :

- This week you will be able to describe the execution of the machine program in your own words.

Preamble

Learning objectives

Learning objectives

- **Explain** in your own words the concepts of memory, compilation, and variable.
- **Simulate** the execution of a simple machine program.

Readings:

- –

Preambule

Plan

Plan

Introduction

Prerequisites

You must master the following concepts:

- **Predefined data types** and **arrays**
- **Control structures**:
  such as `if`, `for`, `while`...;
- **Procedural abstraction** :
  i.e. how to decompose a problem into sub-problems.

Why so many programming languages?

Introduction

Computer architecture

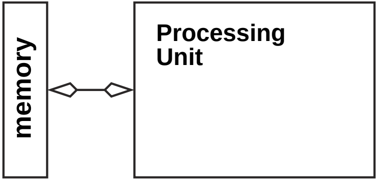
von Neumann

The **architecture** of modern computers is based on a model proposed by (John) **von Neumann** (1945).

memory: contains the **instructions** and the **data**

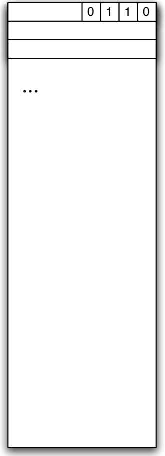
alu: **arithmetic** and **logic** unit

cu: the control unit **decodes the instructions**



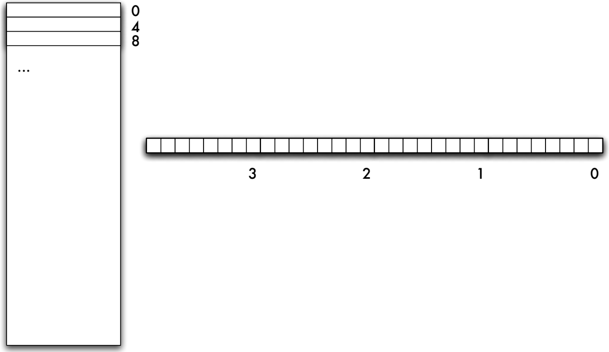
Memory model

- Can be seen as a **huge array**, each cell contains a zero or one (*binary digit* — bits);



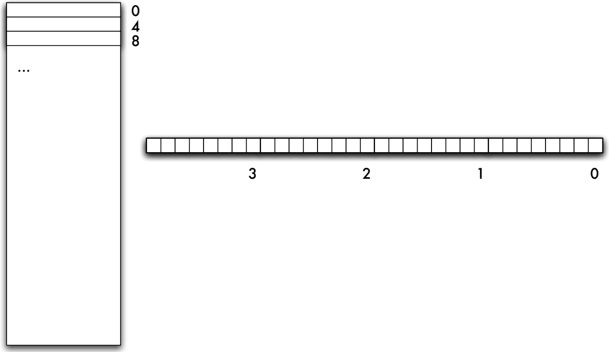
Memory model

- Each **byte** (group of 8 bits) has a **unique** (distinct) address;
- Bytes are grouped into **words**
- **Some types of data require more than one byte.**

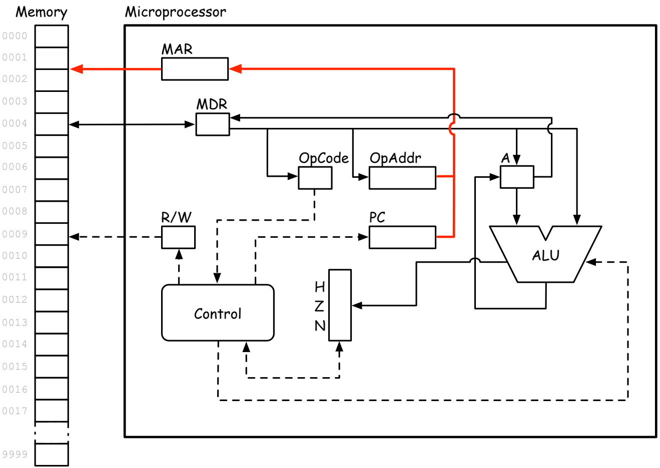


Memory model

- This type of memory is said to be direct access (*Random Access Memory*)
- **The access time to the memory cells is uniform and constant.,** On the order of 5 to 70 nanoseconds (nano $= 10^{-9}$)



Simplified model of a microprocessor (TC1101) and its assembly language.



Mnemonics, opCodes, description

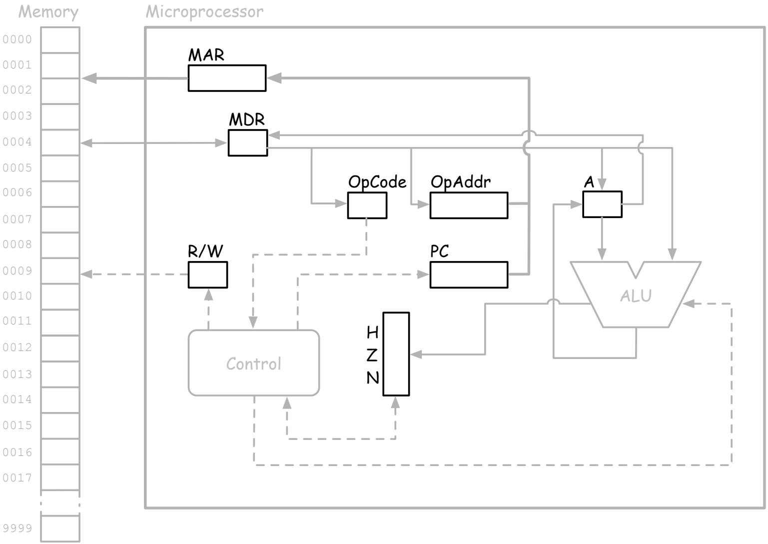
LDA	91	load x
STA	39	store x
CLA	08	clear (a=0, z=true, n=false)
INC	10	increment the accumulator (modifies z and n)
ADD	99	adds x to the accumulator (modifies z and n)
SUB	61	subtracts x from the accumulator (modifies z and n)
JMP	15	unconditional branching to x
JZ	17	branch to x if z==true
JN	19	branch to x if n==true
DSP	01	displays the stored at x
HLT	64	halt

TC1101 instructions

- This microprocessor supports **11 instructions**.
  - In the previous table, you'll find on the left side the **instruction name**, in the center the **machine code**, and on the right side the **instruction description**.
- Instructions with an **even** code have no parameters, whereas instructions with an **odd** have one.
- The term **operand** is used to name the parameter of an instruction.
- The operand is a **memory address**.

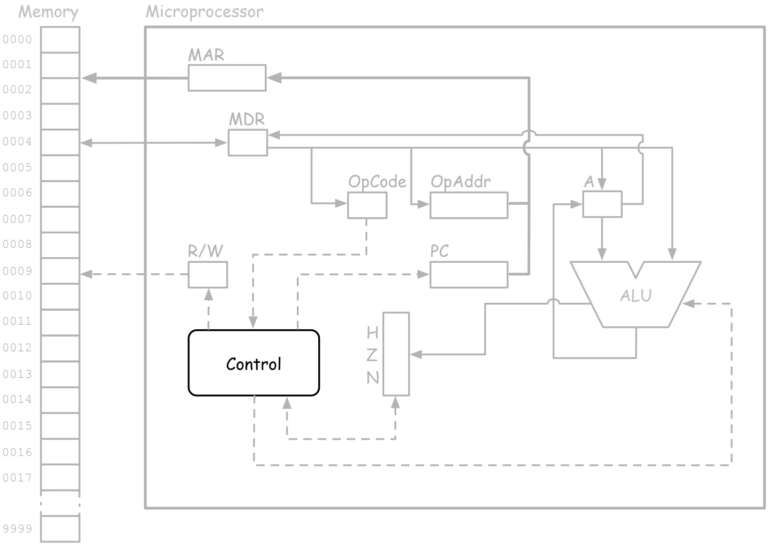
Registers

Registers are **specialized memory units**.



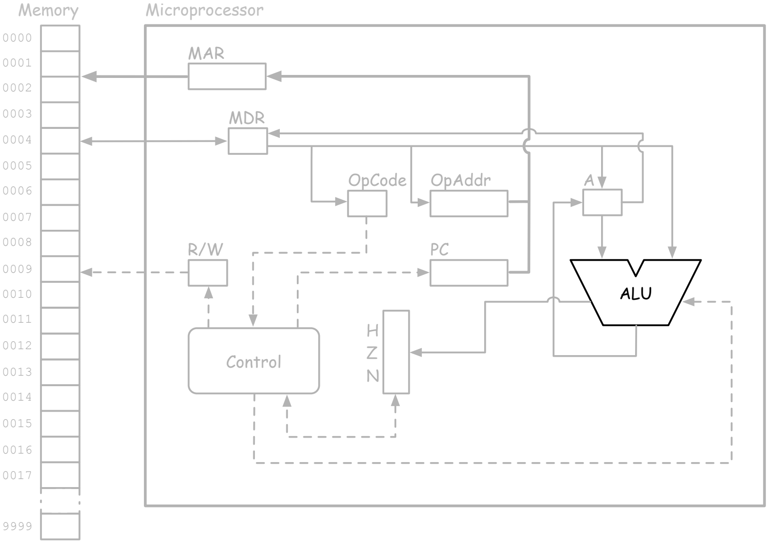
Control unit

The **control unit** orchestrates the execution of the instructions.



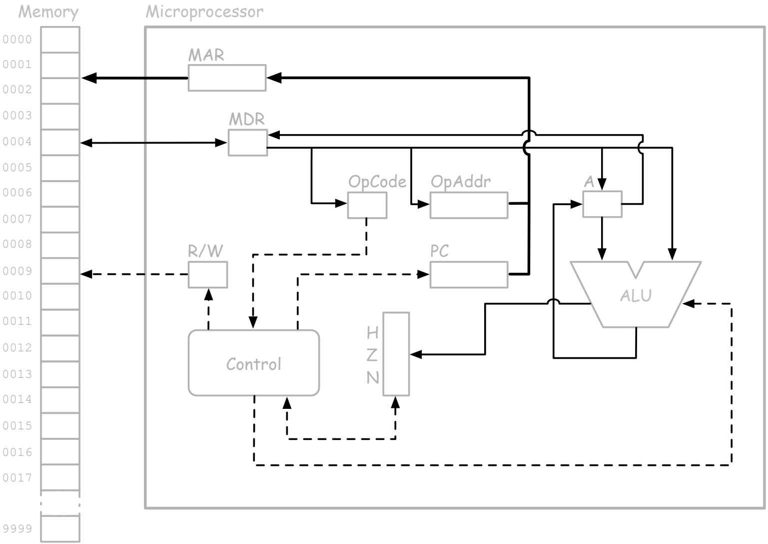
Arithmetic and logic unit (ALU)

The **Arithmetic and logic unit (ALU)** performs the calculations.

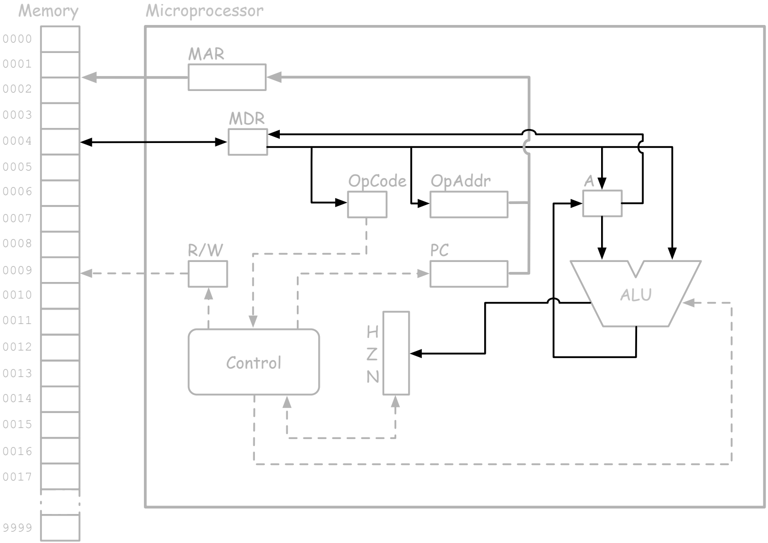


Bus

Information is transferred from one unit to another on buses.

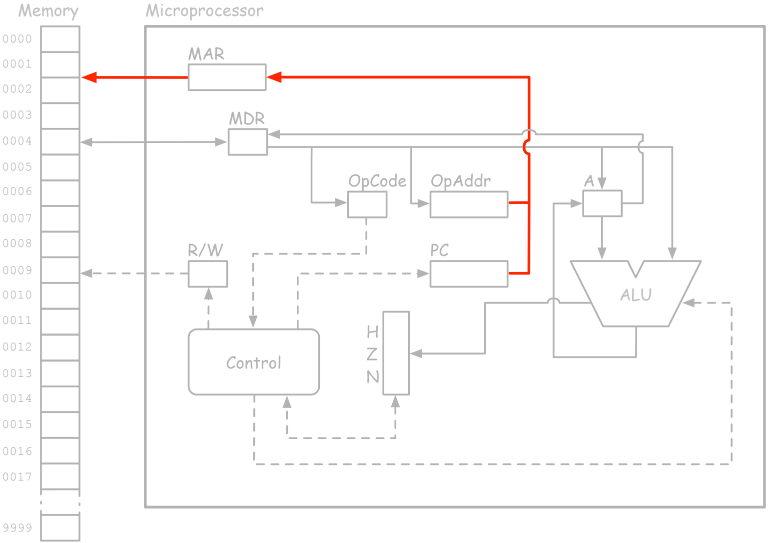


Data bus



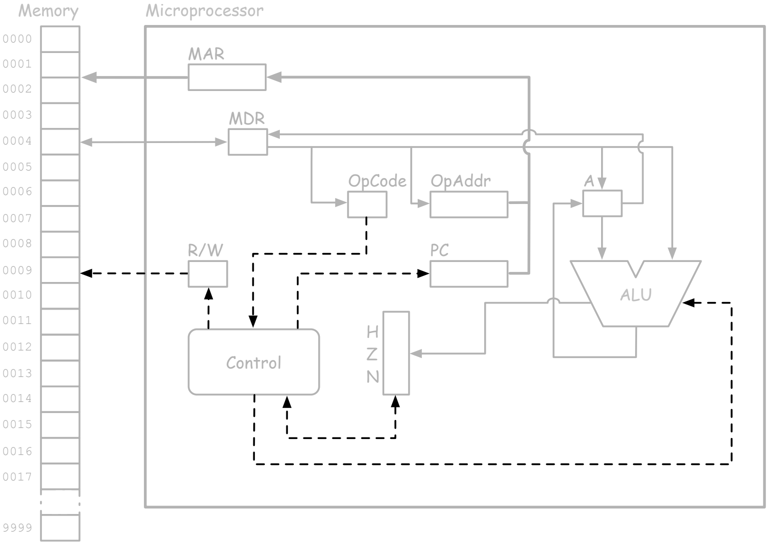
Memory
Microprocessor
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
MDR
OpCode
OpAddr
A
R/W
PC
ALU
H
Z
N
Control

Address bus



Memory
Microprocessor
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
MDR
OpCode
OpAddr
A
R/W
PC
ALU
Control
H
Z
N

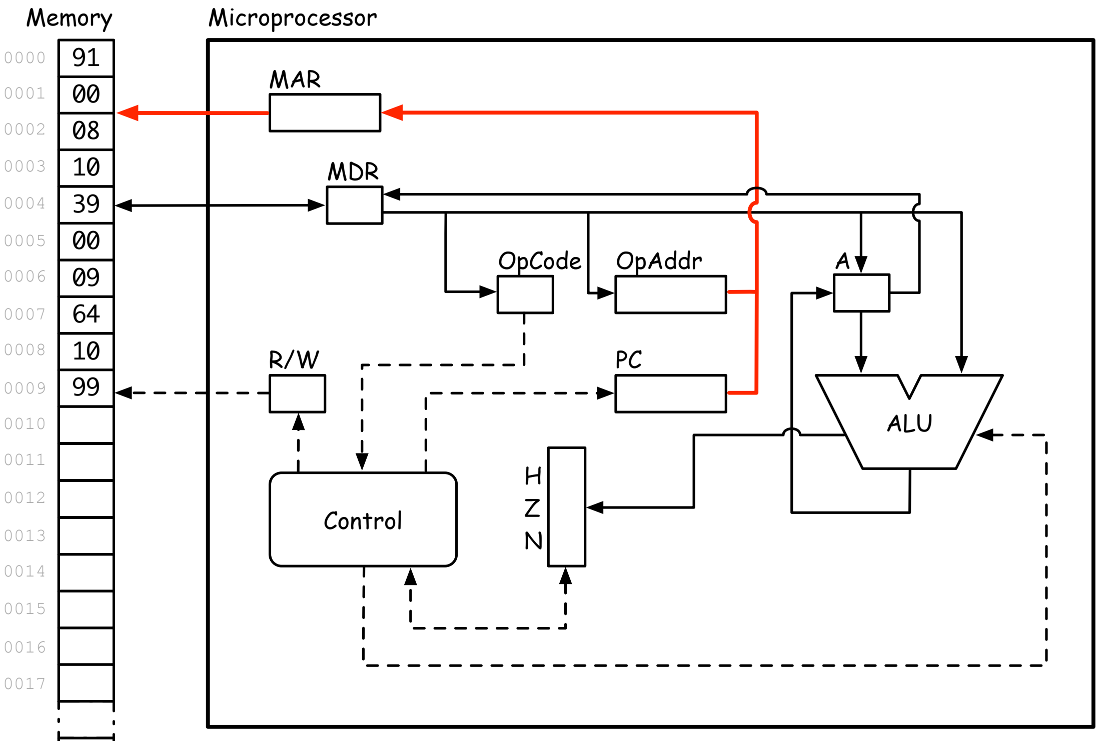
Control bus

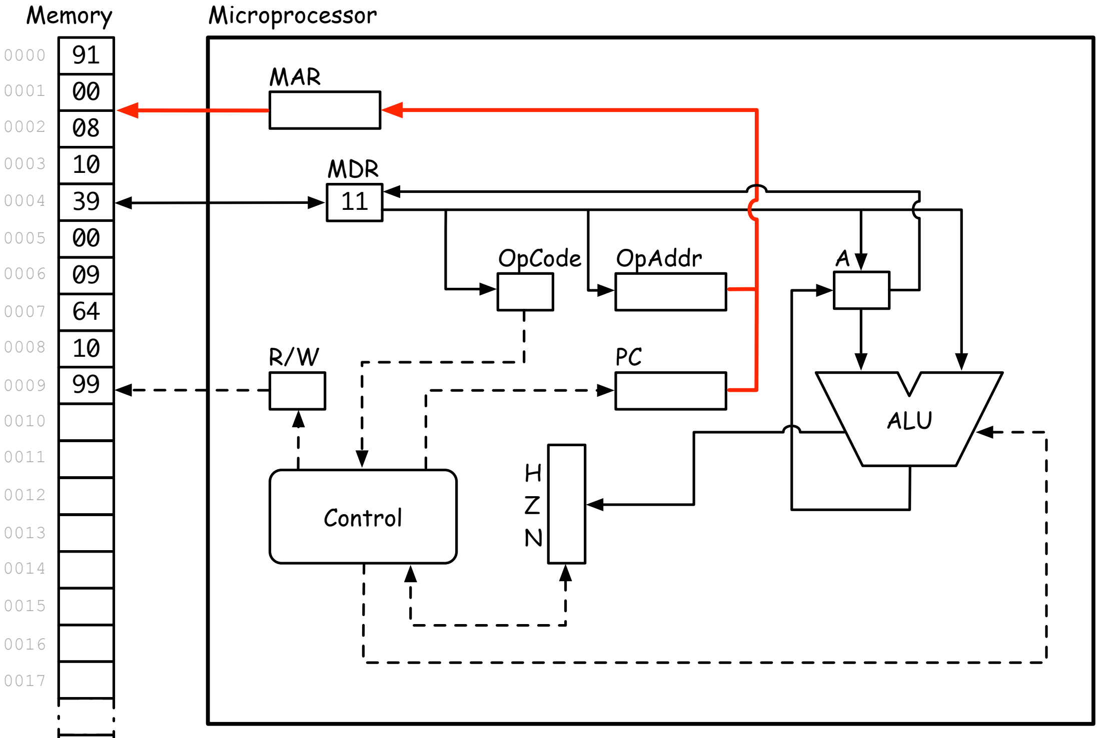

Memory
Microprocessor
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
MDR
OpCode
OpAddr
A
R/W
PC
ALU
Control
H
Z
N

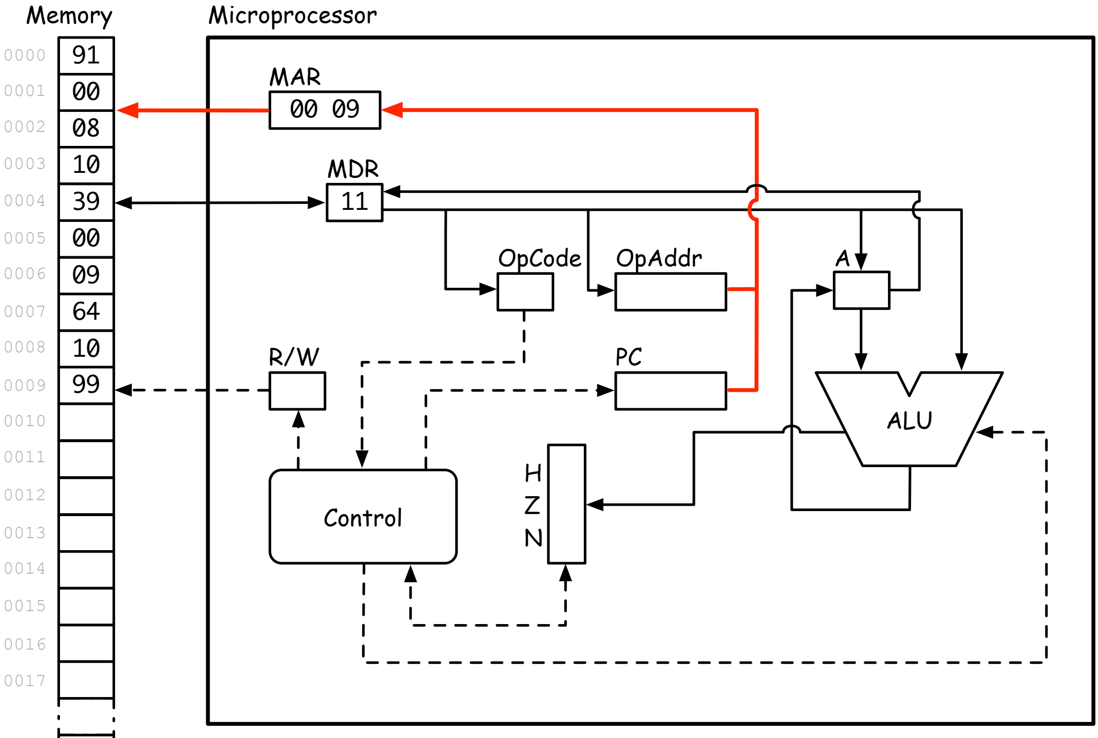
Transfer to the memory

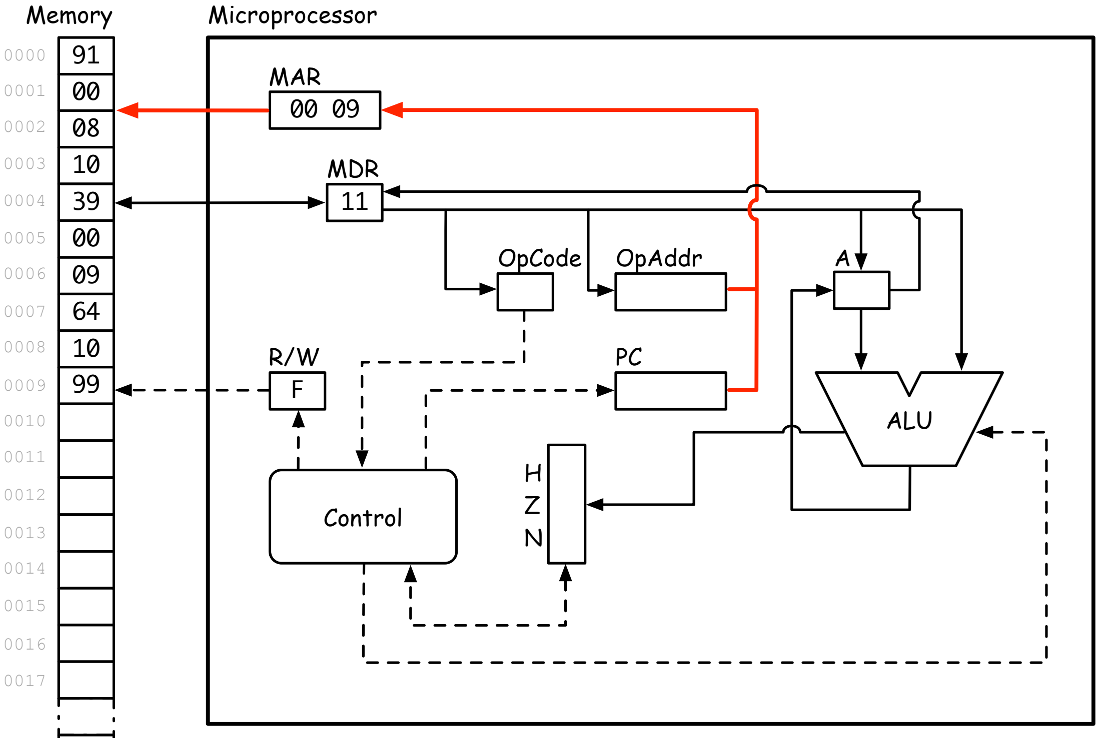
In order to transfer a **value** v from the microprocessor to the **memory address** x:

1. put v into the **memory data register** (MDR),
2. put x into the **memory address register** (MAR),
3. put **status bit RW to false**,
4. **activate the control line** «access_memory».

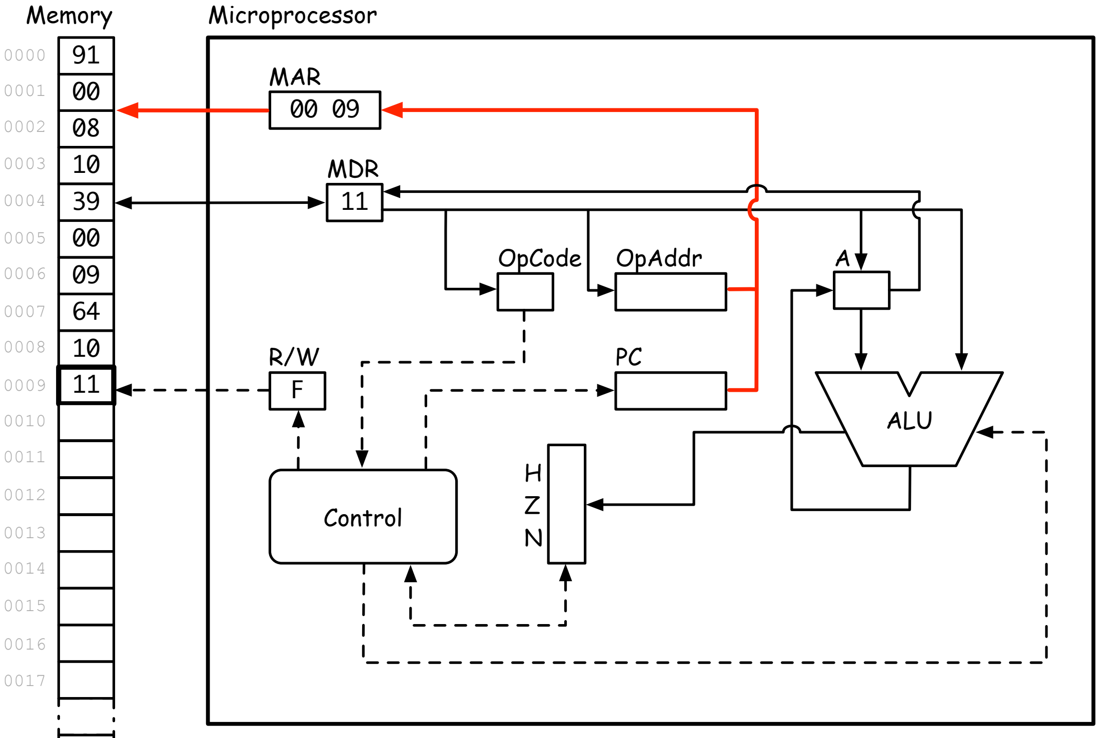

Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
MAR
MDR
OpCode
OpAddr
A
R/W
PC
ALU
H
Z
N
Control


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
MAR
MDR
11
OpCode
OpAddr
A
R/W
PC
ALU
H
Z
N
Control


Memory
Microprocessor
0000 91
0001 00
0002 08
0003 10
0004 39
0005 00
0006 09
0007 64
0008 10
0009 99
0010
0011
0012
0013
0014
0015
0016
0017
MAR
00 09
MDR
11
OpCode
OpAddr
A
R/W
PC
ALU
H
Z
N
Control



Memory
Microprocessor
0000 91
0001 00
0002 08
0003 10
0004 39
0005 00
0006 09
0007 64
0008 10
0009 99
0010
0011
0012
0013
0014
0015
0016
0017
MAR
00 09
MDR
11
OpCode
OpAddr
A
R/W
F
PC
ALU
H
Z
N
Control

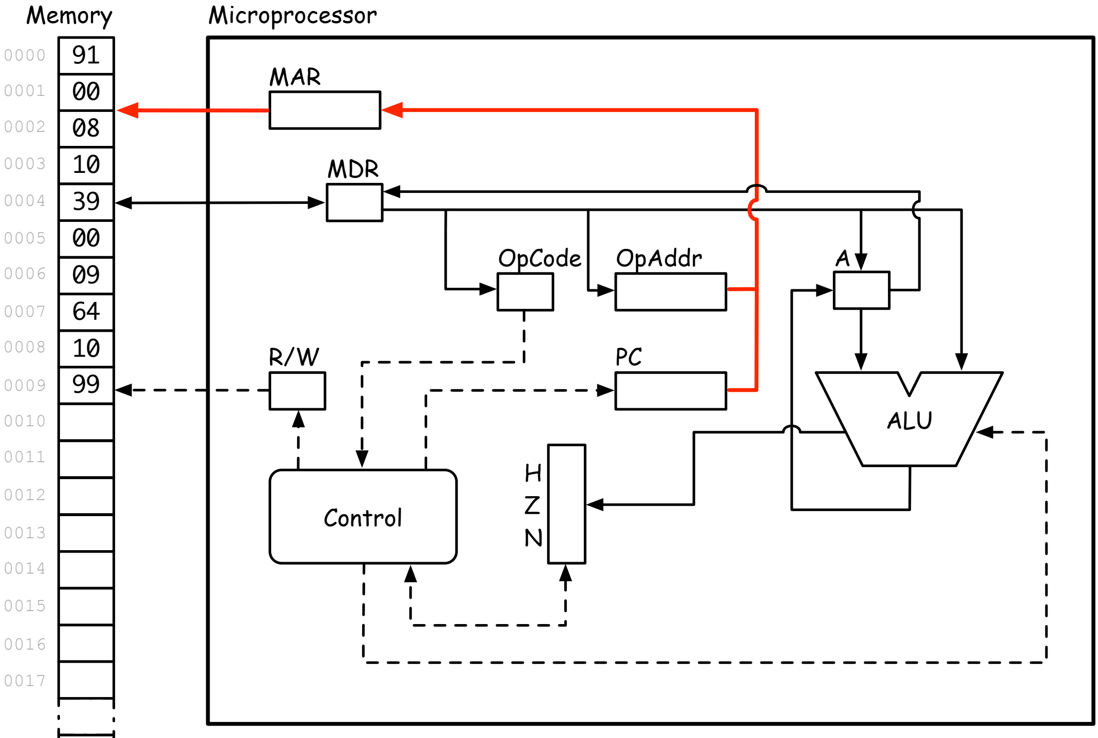


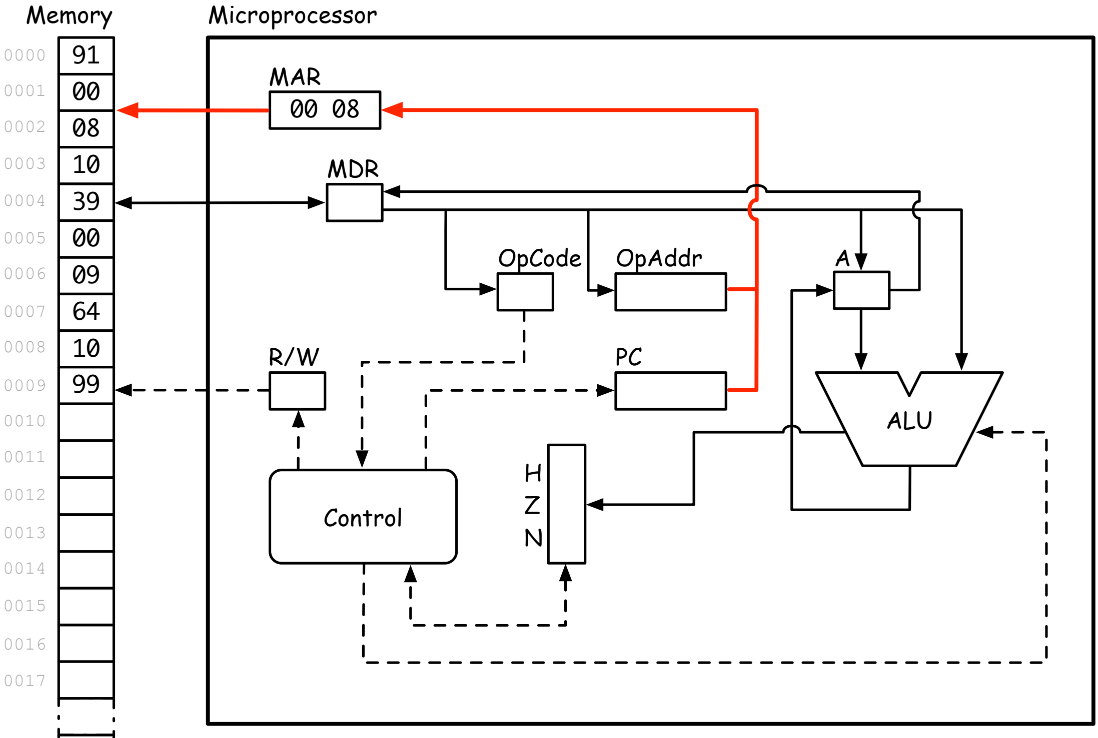
Memory
Microprocessor
0000 91
0001 00
0002 08
0003 10
0004 39
0005 00
0006 09
0007 64
0008 10
0009 11
0010
0011
0012
0013
0014
0015
0016
0017
MAR
00 09
MDR
11
OpCode
OpAddr
A
R/W
F
PC
ALU
Control
H
Z
N

Transfer from the memory

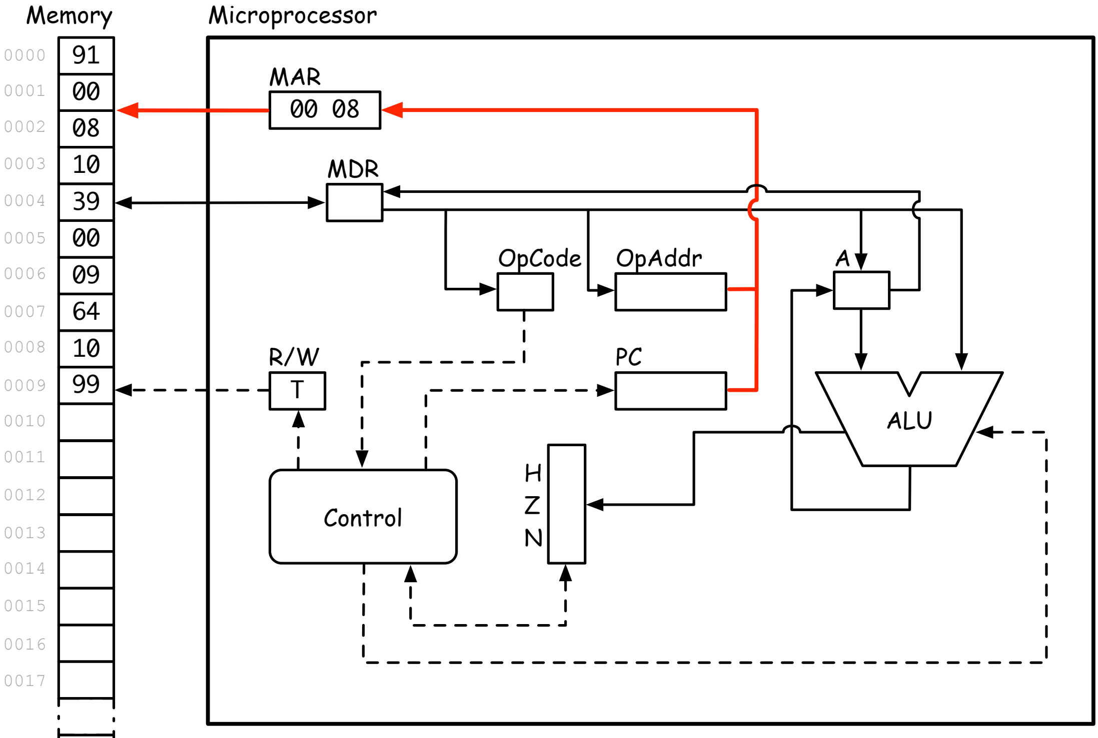
In order to transfer a value **from the (memory) address** x **to the microprocessor**:

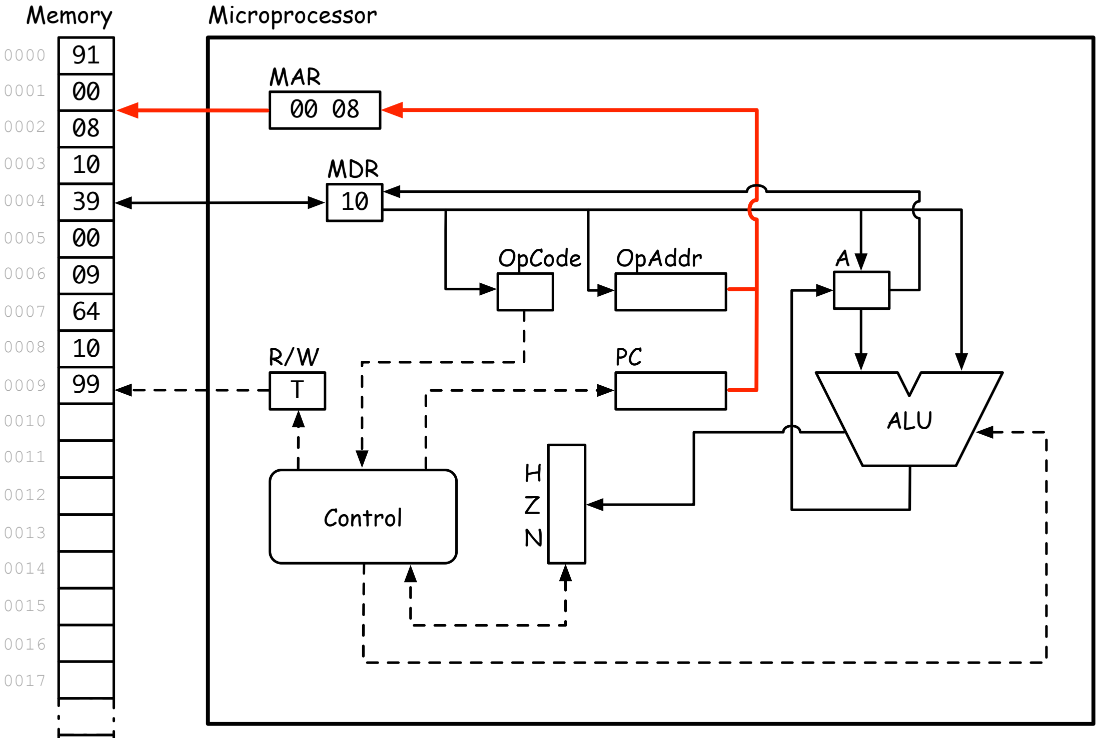
1. put the value x into the **memory address register** (MAR),
2. put the **status bt RW to true**,
3. **activate** the control line «access_memory»
4. the **memory data register** (MDR) now contains a **copy** of the value found at memory location x.


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
MAR
MDR
OpCode
OpAddr
A
R/W
PC
ALU
Control
H
Z
N



Memory
Microprocessor
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
91
00
08
10
39
00
09
64
10
99
MAR
00 08
MDR
OpCode
OpAddr
A
R/W
PC
ALU
H
Z
N
Control


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
MAR
00 08
MDR
OpCode
OpAddr
A
R/W
T
PC
ALU
H
Z
N
Control


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
MAR
00 08
MDR
10
OpCode
OpAddr
A
R/W
T
PC
ALU
H
Z
N
Control

Fetch-decode-execute cycle

1. transfer:
    1.1 transfer the OPCODE,
    1.2 increment PC,
2. based on OPCODE transfer the operand:
    2.1 transfer the first byte,
    2.2 increment PC,
    2.3 transfer the second byte,
    2.4 increment PC,
3. execute.

Compilation

The programs, sequences of statements from a high-level programming language, are translated (**compiled**) into a low-level language (assembler, machine code), directly interpretable by the hardware.
The expression $y = x + 1$ is translated to assembly :

```
LDA X
INC
STA Y
HLT
```

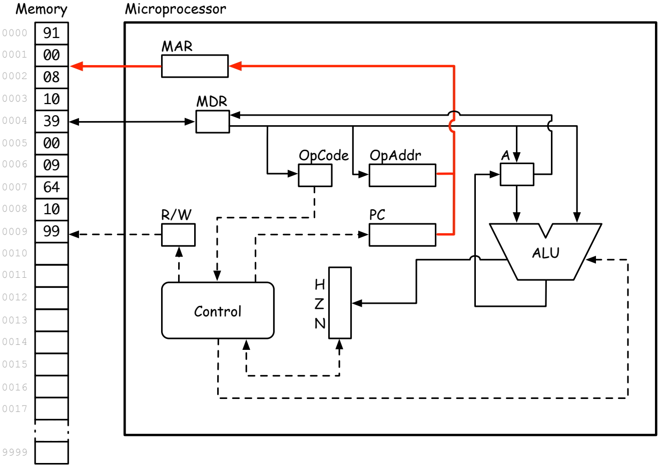
which is the translated to **machine code**:
| 91 00 08 | 10 | 39 00 09 | 64 | 10 | 99 |

The **expression** $y = x + 1$ is translated to **assembly** :

```
LDA X
INC
STA Y
HLT
```

which is then translated to **machine code** :

| 91 00 08 | 10 | 39 00 09 | 64 | 10 | 99 |



Functional Units of the TC-1101

PC (2 bytes): *Program Counter*, one 2 bytes register that contains the address of the next instruction to be executed;

opCode (byte): instruction register (sometimes called IR), contains the OPCODE of the current instruction;

opAddr (2 bytes): the operand of the current instruction. The operand is always an address. Some instructions necessitate the value found at the address designated by the operand — this value is not transferred by the basic cycle, but needs to be transferred during the execution of the instruction (see step 3 of the cycle and the description of each instruction below);

Functional Units of the TC-1101

MDR (byte): *Memory Data Register*. A value transferred (read/written) from the memory to the processor (or vice-versa) is always stored in this registered;

MAR (2 bytes) : *Memory Address Register*. This register contains the memory address of a value to be read or to be written;

A (byte): Accumulator. All the arithmetic operations use this register as an operand and also to store their result;

Functional Units of the TC-1101

H (bit): status bit “Halt”. This bit is set by the instruction halt (hlt). If the bit is true the processor stops at the end of this cycle;

N (bit): status bit “Negative”. Arithmetic operations set this bit to true whenever they produce a negative result. Some operations are not affecting the value of this bit, therefore its value does not always reflect the content of the accumulator;

Z (bit): status bit “Zero”. Arithmetic operations set the value of this bit to true whenever the result is zero. Some operations do not affect the content of this bit, therefore, its value does not always reflect the content of the accumulator;

Functional Units of the TC-1101

RW (bit): status bit "READ/WRITE". A value true means a value must be read (fetched) from the memory and transferred to MDR. A value false signifies that a value must be transferred from MDR to the memory.

Assembly language

- Assembly language is **not very expressive**.
- Each microprocessor has its own assembly language. Programs are thus **not portable** from one computer to another.

High-level programming languages

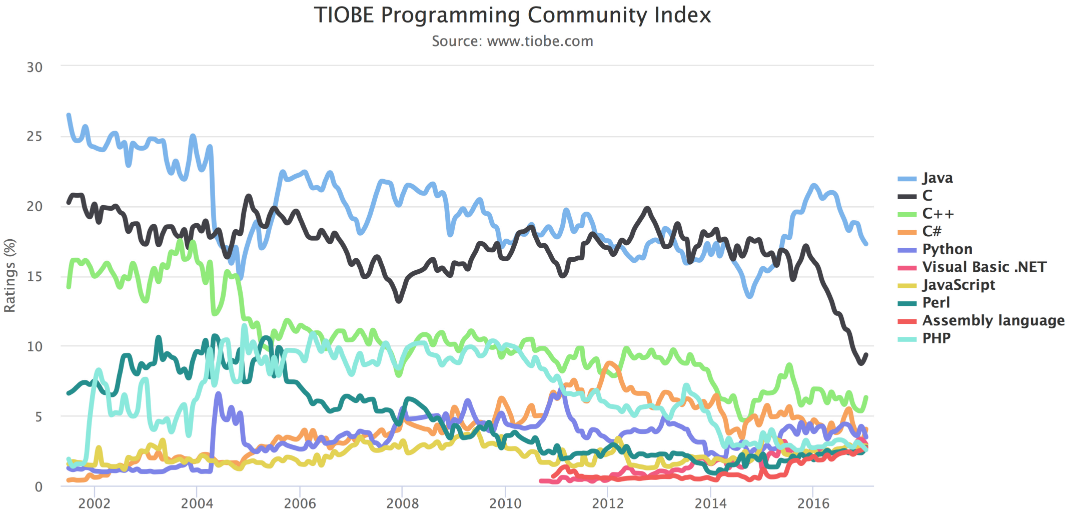
- High-level programming languages are **expressive**.
- Generally, a high-level programming language is also **portable**.

Paradigms

- **Imperative** or **procedural**
- **Object-oriented**
- **Declarative**
  - Functional
  - Logic
  - Constraint

Why Java?

Why Java?


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
Java
C
C++
C#
Python
Visual Basic .NET
JavaScript
Perl
Assembly language
PHP

Why Java?

1	Java	17%
2	C	9%
3	C++	6%
4	C#	4%
5	Python	4%
6	Basic	3%
7	JavaScript	3%
8	Perl	3%
9	Assembly	3%
10	PHP	3%

TIOBE Programming Community Index

Why Java?

1	JavaScript
2	Java
3	PHP
4	Python
5	C#
5	C++
5	Ruby
8	CSS
9	C
10	Objective-C

The **RedMonk** Programming Language Rankings: January 2016

Why Java?

Java is popular, but I don't know any application written in Java.

- The **server side** of many Web applications and services.
- **Mobile applications** (mobile phones)

2015 Top Trending Careers in Tech: Java Developer

- info.theladders.com/career-advice/top-market-trends-in-tech

Why Java?

« According to a report from NetApplications, which has measured browser usage data since 2004, Oracle's Java Mobile Edition has surpassed Android as **the #2 mobile OS** on the internet at 26.80%, with iOS at 46.57% and Android at 13.44%. And the trend appears to be growing. Java ME powers hundreds of millions of low-end 'feature phones' for budget buyers. In 2011, **feature phones made up 60% of the install base in the U.S.** »

Slashdot
3 janvier 2012
http://bit.ly/xSk5pN

Why Java?

- Writing programs using **C** requires discipline
  (managing memory, manipulating pointers, etc.)
- Java is an **excellent vehicle for teaching**
  (interface, single inheritance, generic types. . .)
- If you master Java, learning other object-oriented or imperative programming languages will be **simple**.

Why Java?

Source: `https://commons.wikimedia.org/wiki/File:Netflix_logo.svg`

`https://go.java/netflix.html`

Next module

- **Data types**

Division by successive subtractions

```
[1]   CLA
      STA Quot
[2]   LDA X
[3]   SUB Y
[4]   JN  [7]
[5]   STA Temp
      LDA Quot
      INC
      STA Quot
      LDA Temp
[6]   JMP [3]
[7]   ADD Y
[8]   STA Rem
[9]   DSP Quot
[10]  DSP Rem
[11]  HLT
X     BYTE 25
Y     BYTE 07
Quot  BYTE 00
Rem   BYTE 00
Temp  BYTE 00
```

Division: code machine

```
[1]   CLA              08
      STA Quot         39    00 44
[2]   LDA X            91    00 42
[3]   SUB Y            61    00 43
[4]   JN  [7]          19    00 29
[5]   STA Temp         39    00 46
      LDA Quot         91    00 44
      INC              10
      STA Quot         39    00 44
      LDA Temp         91    00 46
[6]   JMP [3]     ⇒    15    00 07
[7]   ADD Y            99    00 43
[8]   STA Rem          39    00 45
[9]   DSP Quot         01    00 44
[10]  DSP Rem          01    00 45
[11]  HLT              64
X     BYTE 25          25
Y     BYTE 07          07
Quot  BYTE 00          00
Rem   BYTE 00          00
Temp  BYTE 00          00
```

References I

- E. B. Koffman and Wolfgang P. A. T.
  ***Data Structures: Abstraction and Design Using Java*.**
  John Wiley & Sons, 3e edition, 2016.
- D. J. Barnes and M. Kölling.
  ***Objects First with Java: A Practical Introduction Using BlueJ*.**
  Prentice Hall, 4e edition, 2009.
- P. Sestoft.
  ***Java Precisely*.**
  The MIT Press, second edition edition, August 2005.

Marcel **Turcotte**
Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EECS)
University of Ottawa