Computer architecture

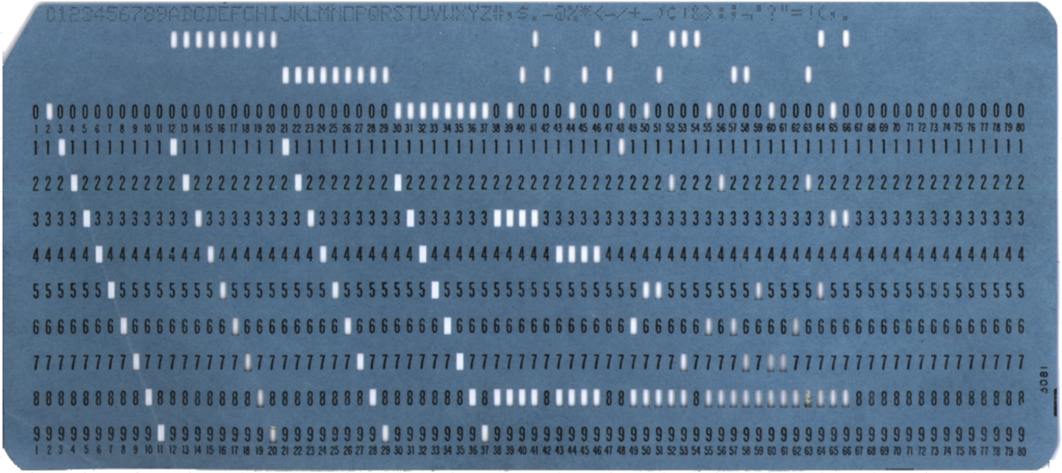
ver. 17 z drobnymi modyfikacjami!

Wojciech Myszka

2023-10-30 16:11:55 +0100

Wrocław University
of Science and Technology

Punched cards

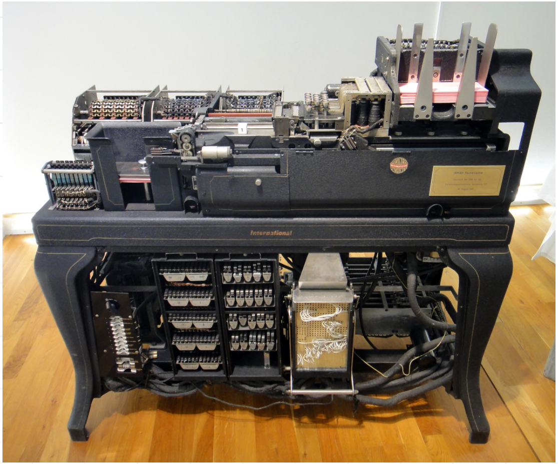


Some history

In the first half of the 20th Century, IBM's flagship product was the Tabulator, which is pretty much just a big adding machine. Beginning about 1930, IBM began to produce machines that could also multiply, and eventually divide, as well as add and subtract. These were called Calculators and they were used primarily for engineering and scientific applications.

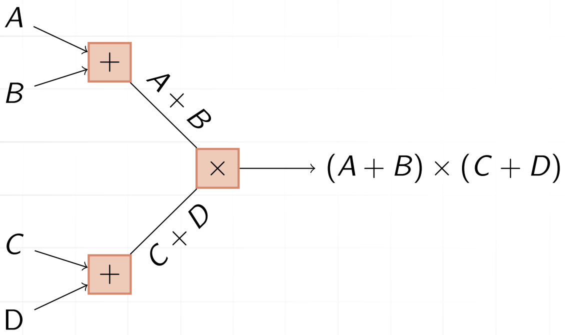
IBM 601 Calculator

First IBM (electro-mechanical) calculator that could multiply.



1931

The idea of computing



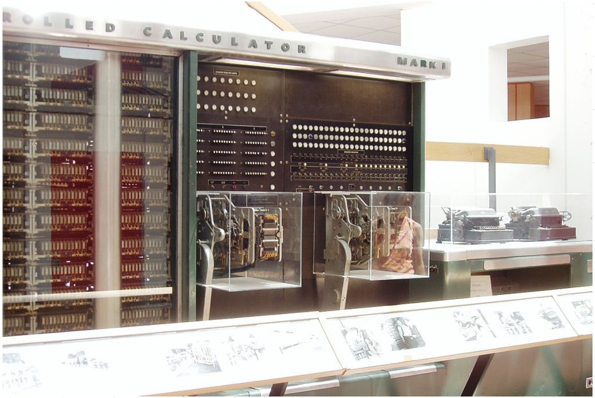
- A, B, C, D — data: "stream" (file/deck) of punched cards
- $A + B,\ C + D$ — partial result: (file/deck) of punched cards
- $(A + B)(C + D)$ — result: the stream of punched cards

IBM 601 Calculator, cont.

The IBM 601 Multiplying Punch read two factors up to eight decimal digits in length from a card and punched their product onto a blank field of the same card. It could subtract and add as well as multiply. It had no printing capacity, so was generally used as an offline assistant for a tabulator or accounting machine. The 601 that was delivered to Eckert's lab in 1933 was a special model "capable of doing the direct interpolation, a very unusual feature, specially designed for Eckert by one of IBM's top engineers at Endicott [NY]". Eckert went a step further by connecting the 601 to a Type 285 Tabulator and a Type 016 Duplicating Punch through a calculation control switch of his own design, forming the first machine to perform complex scientific computations automatically.

http://www.columbia.edu/cu/computinghistory/601.html

Early digital (electro-mechanical) computers. . .



. . . follow this idea, as well.

Project Manhattan

Richard P. Feynman used to administer the group of “human computers” used for performing computations for the Manhattan Project. He assisted in establishing a system for using IBM punched cards for computation.
You can find this story on-line, described by Feynman.
This was described in the book *Surely You’re Joking, Mr. Feynman!* by Richard Feynman and Ralph Leighton.

John von Neumann

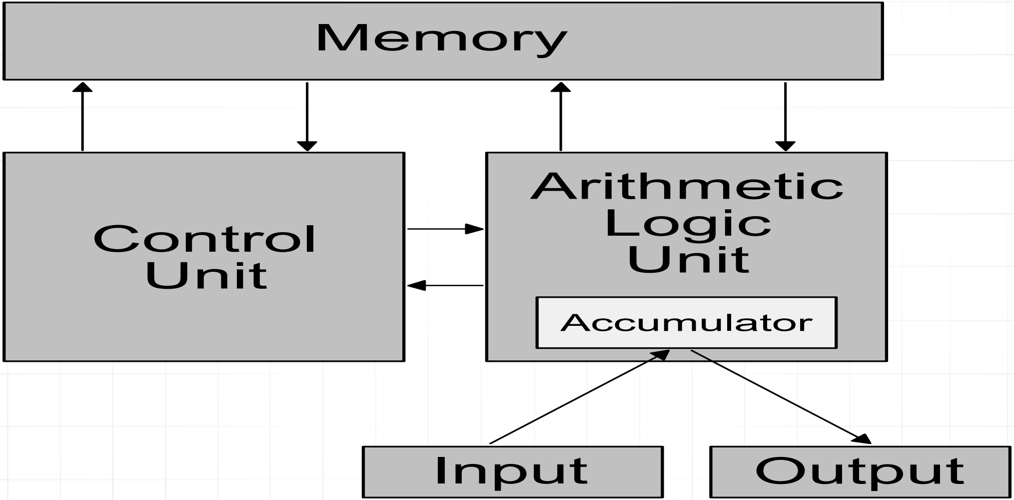
1. One of the most important computational projects in the forties (20th century) were calculations for the atomic bomb.
2. John von Neumann, a brilliant mathematician, and physicist (who was also involved in quantum mechanics, game theory, computer science, functional analysis ...) were engaged in this work.

von Neumann architecture

So-called “von Neumann architecture” (presented in First Draft of a Report on the EDVAC) describes the architecture for a computer with subdivision of a processing unit, a control unit, a memory, external mass storage, and input and output mechanisms.

- processing unit has an *arithmetic logic unit* and *processor registers* (kind of a local storage),
- memory was used to store both *data* and *instructions*,
- control unit has an *instruction register* and *program counter*,

von Neumann architecture



von Neumann architecture

Computer system build according to von Neuman's architecture allows for:

- inputting a program from an external source to the computer's memory,
- inputting data and changing them easily.

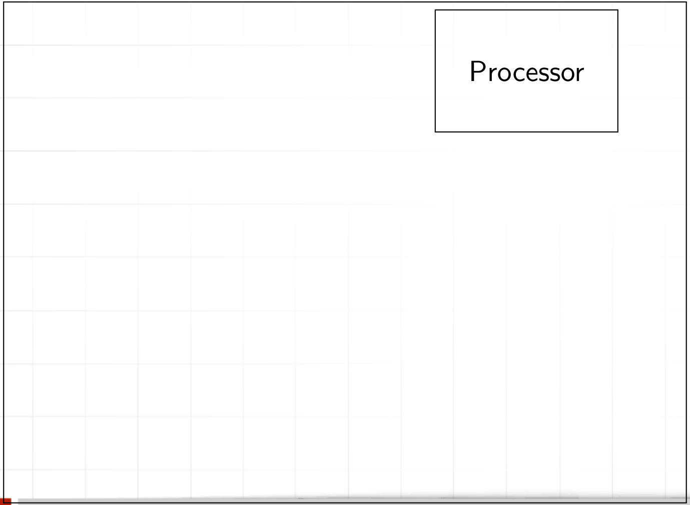
- In general, the data and the program residing in teh computer's memory are indistinguishable.
- Processor or strictly speaking, computer program can modify itself.
- The information is processed by sequential execution of program instructions.

von Neumann architecture

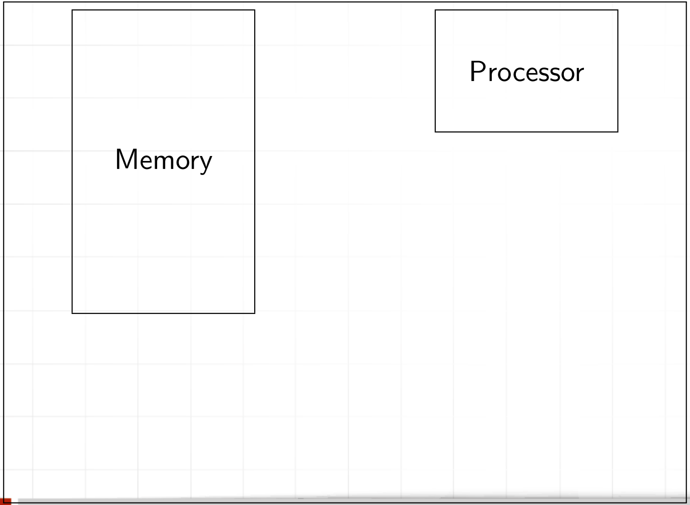
- These conditions allow for switching from the execution of one task (program) to another without physical intervention in the structure of the system and thus ensure its versatility.
- Von Neumann's computer system does not have separate memory for storing data and instructions. Instructions and data are encoded in the form of numbers. Without an analysis of the program is difficult to determine whether the area of memory contains data or instructions.
- The executed program can modify itself treating its own instructions (code) as data: changing them and then executing.

Simplified diagram of a computer

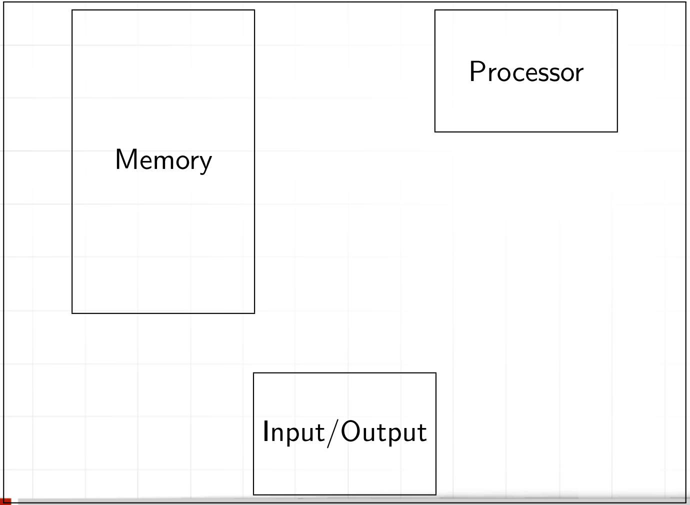
Simplified diagram of a computer



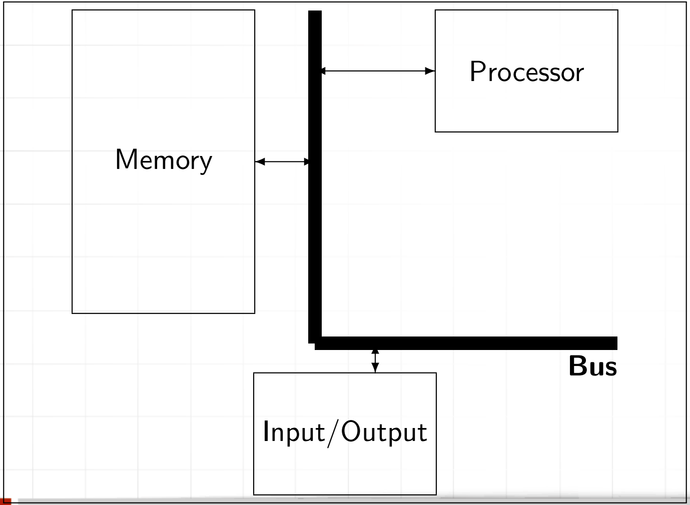
Simplified diagram of a computer



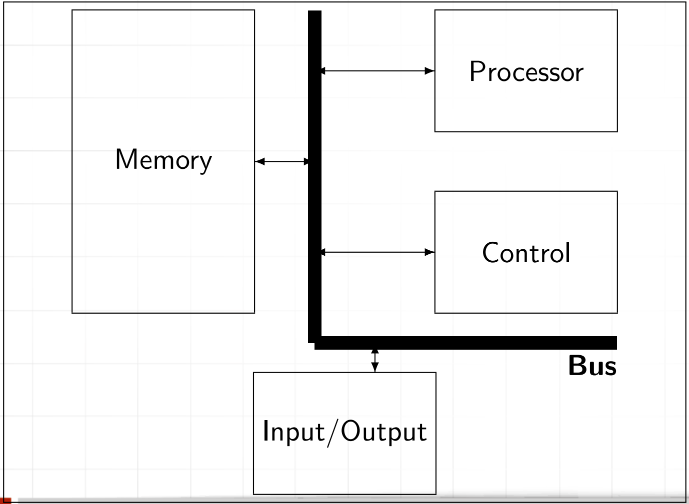
Simplified diagram of a computer



Simplified diagram of a computer



Simplified diagram of a computer



Simplified diagram of a computer

In this model:

- Processor.
- Memory (all kinds of it: RAM, ROM, Disks, floppies, external disks, and so on).
- All input and output devices allowing for communication with "external word": keyboard, mouse, touch-pad, printer, graphic card...
- Control: all electronic devices allowing for fetching and executing program's instructions from the memory, and transferring instructions and data through the bus.
- Bus: all connections allowing for a flow of data and control instructions.

Calculator

Display			
1	2	3	$+$
4	5	6	$-$
7	8	9	$*$
0	C	$=$	$\div$

Calculator

Display			
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is the simplest calculator: four operations only.

Calculator

Display			
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is the simplest calculator: four operations only.
- Only keyboard (digits, operations) and display.

Calculator

Display			
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is the simplest calculator: four operations only.
- Only keyboard (digits, operations) and display.
- After pressing digit keys (123) data is copied to the display.

Calculator

			1
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is the simplest calculator: four operations only.
- Only keyboard (digits, operations) and display.
- After pressing digit keys (123) data is copied to the display.

Calculator

12			
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is the simplest calculator: four operations only.
- Only keyboard (digits, operations) and display.
- After pressing digit keys (123) data is copied to the display.

Calculator

			123
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷

- It is the simplest calculator: four operations only.
- Only keyboard (digits, operations) and display.
- After pressing digit keys (123) data is copied to the display.

Calculator

			123
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is the simplest calculator: four operations only.
- Only keyboard (digits, operations) and display.
- After pressing digit keys (123) data is copied to the display.
- There should be some kind of a memory for storing a keyed number. This memory on one side is connected to the keyboard (data input) on the other to the display (data output).

Calculator

			123
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is the simplest calculator: four operations only.
- Only keyboard (digits, operations) and display.
- After pressing digit keys (123) data is copied to the display.
- There should be some kind of a memory for storing a keyed number. This memory on one side is connected to the keyboard (data input) on the other to the display (data output).
- Let's modify our diagram.

Calculator

Display			
Accumulator			
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷

- It is a simplest calculator: four operations only: keyboard (digits, operations) and display

Calculator

Display			
Accumulator			
1	2	3	+
4	5	6	−
7	8	9	$*$
0	C	=	÷

- It is a simplest calculator: four operations only: keyboard (digits, operations) and display
- You can not see memory called accumulator, but probably it exists.

Calculator

Display			
Accumulator			
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is a simplest calculator: four operations only: keyboard (digits, operations) and display
- You can not see memory called accumulator, but probably it exists.
- After pressing digit keys (123) data is stored in the memory and displayed on the display.

Calculator

			1
			1
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷

- It is a simplest calculator: four operations only: keyboard (digits, operations) and display
- You can not see memory called accumulator, but probably it exists.
- After pressing digit keys (123) data is stored in the memory and displayed on the display.

Calculator

			12
			12
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is a simplest calculator: four operations only: keyboard (digits, operations) and display
- You can not see memory called accumulator, but probably it exists.
- After pressing digit keys (123) data is stored in the memory and displayed on the display.

Calculator

			123
			123
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is a simplest calculator: four operations only: keyboard (digits, operations) and display
- You can not see memory called accumulator, but probably it exists.
- After pressing digit keys (123) data is stored in the memory and displayed on the display.

Calculator

			123
			123
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷

- It is a simplest calculator: four operations only: keyboard (digits, operations) and display
- You can not see memory called accumulator, but probably it exists.
- After pressing digit keys (123) data is stored in the memory and displayed on the display.
- Let's press operation key (+)

Calculator

			123
			123
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷

- It is a simplest calculator: four operations only: keyboard (digits, operations) and display
- You can not see memory called accumulator, but probably it exists.
- After pressing digit keys (123) data is stored in the memory and displayed on the display.
- Let's press operation key (+)

Calculator

			123
			123
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷

- The display has not changed but pressing the digit key erases the display and the new value appears.

Calculator

			123
			123
1	2	3	$+$
4	5	6	$-$
7	8	9	$*$
0	C	$=$	$\div$

- The display has not changed but pressing the digit key erases the display and the new value appears.
- The first value does not disappear; there must be another memory.

Calculator

			123
			123
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷

- The display has not changed but pressing the digit key erases the display and the new value appears.
- The first value does not disappear; there must be another memory.
- All arithmetic operations have two arguments.

Calculator

			123
			123
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷

- The first value does not disappear; there must be another memory.
- All arithmetic operations have two arguments.
- Additional memory stores the second argument.

Calculator

Display			
Accumulator			
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷
Memory			

Now, probably all key elements are shown.

Calculator

Display			
Accumulator			
1	2	3	$+$
4	5	6	$-$
7	8	9	$*$
0	C	$=$	$\div$
Memory			

Operations

Calculator

			123
			123
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷
			Memory

Operations

123

Calculator

123			
123			
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷
123			

Operations

123

+

Calculator

			55
			55
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷
			123

Operations

123

+

55

Calculator

			178
			178
1	2	3	+
4	5	6	−
7	8	9	∗
0	C	=	÷
			123

Operations

123

+

55

+

Calculator

			22
			22
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷
			178

Operations

123

+

55

+

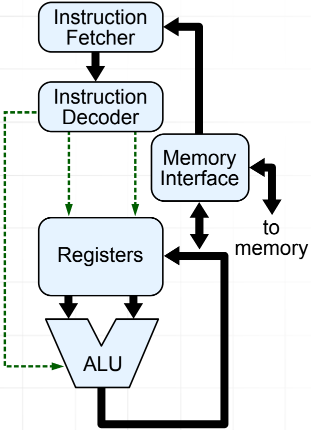
22

Calculator

			200
			200
1	2	3	+
4	5	6	−
7	8	9	*
0	C	=	÷
			178

Operations
123
+
55
+
22
=

Processor



Instruction Fetcher
Instruction Decoder
Memory Interface
Registers
ALU
to memory

Basic operations

Arithmetic

- `Load` <*memory address*> copies the data from a particular cell of a RAM into the accumulator.
- `Store` <*memory address*> copies the content of the accumulator to the memory.
- `Load indirect` <*number*> copies the number into the accumulator.
- `Add` <*memory address*> adds content of memory cell to the current content of the accumulator. (We can assume that there are also operations for subtracting, multiplying, and dividing data; but this is not always true.)
  Execution of each operation, changing the register (accumulator) sets the indicators (zero, overflow, negative).

Basic operations

Performed on bits

- `Nagation` Changes the sign of the number in the accumulator.
- `And` <*memory address*> logical AND (bitwise)
- `Or` <*memory address*>
- `Xor` <*memory address*> — Exclusive OR
- `Shift_left`
- `Shift_right`

Control operations

- `Jump` <*memory address*> the next instruction will be taken from the given address
- `Jump_if_zero` <*memory address*>
- `Jump_if_negative` <*memory address*>
- `Jump_if_overflow` <*memory address*>
- `Jump_to_subroutine` <*memory address*> very similar to the normal jump instruction, but also saves the current state of the processor in a dedicated memory (allowing for restoration of the state later on).

Assembler

Very simple operation:

`A=B+C`

Assembler

Very simple operation:

`A=B+C`

We have to put the result of the sum of values stored in the memory addresses `B` and `C` in the memory cell, having the address `A`.

Assembler

Very simple operation:

`A=B+C`

We have to put the result of the sum of values stored in the memory addresses `B` and `C` in the memory cell, having the address `A`.

Computer realisation:

```
Load B
Add C
Store A
```

Assembler

More complicated example

$$Z = \frac{[(A + B)(C + D)]}{W}$$

Assembler

More complicated example

$$Z = \frac{[(A+B)(C+D)]}{W}$$

$$T1 = A + B$$

Assembler

More complicated example

$$Z = \frac{[(A + B)(C + D)]}{W}$$

$$T1 = A + B$$
$$T2 = C + D$$

Assembler

More complicated example

$$Z = \frac{[(A + B)(C + D)]}{W}$$

$$T1 = A + B$$
$$T2 = C + D$$
$$T3 = T1 * T2$$

Assembler

More complicated example

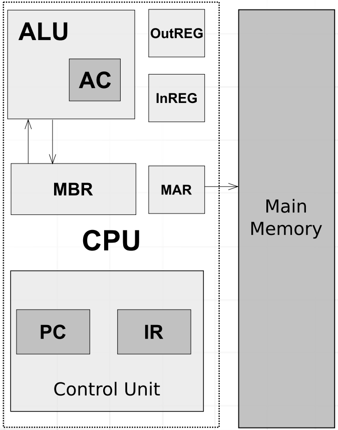
$$Z = \frac{[(A + B)(C + D)]}{W}$$

$$T1 = A + B$$
$$T2 = C + D$$
$$T3 = T1 * T2$$
$$Z = T3/W$$

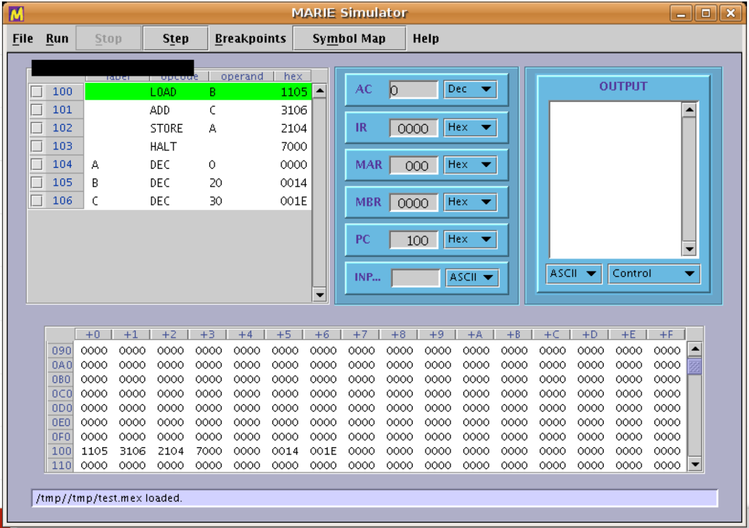
MARIE

MARIE — A Machine Architecture that is Really Intuitive and Easy

- base two positional system (2's complement)
- constant word size
- addressing of words
- 4Ki bytesof main memory (12 bits)
- 16-bit data (16-bit word)
- 16-bit instruction (4-bit operation code + 12-bit address)
- 16-bit accumulator (AC)
- 16-bit instruction register (IR)
- 16-bit memory buffer register (MBR)
- 12-bit program counter (PC)
- 12-bit memory address register (MAR)
- 8-bit Input register (InREG)
- 8-bit output register (OutREG)



MARIE Simulator



Homework

Simple program for MARIE computer simulator (for example multiplication of two numbers).

Additional resources:

MARIE.
https://pl.wikipedia.org/wiki/MARIE.
Only in Polish.

Student resources – essentials of computer organization and architecture, second edition.
http://samples.jbpub.com/9781284123036/9781284136852_FMxx_Print_Final.pdf.
Only Table of Contents and Preface.

Kuo-pao Yang.
MARIE: An introduction to a simple computer.
https://www2.southeastern.edu/Academics/Faculty/kyang/2013/Spring/CMPS375/ClassNotes/CMPS375ClassNotesChap04.pdf, 2013.

Linda Null and Julia Lobur.
The essentials of computer organization and architecture.
Jones and Bartlett Publishers, Sudbury, Mass., 2006.

Linda Null and Julia Lobur.
MARIE: an introduction to a simple computer.
In *The essentials of computer organization and architecture*. 2006.
http://samples.jbpub.com/9781449600068/00068_CH04_Null3e.pdf.

Linda Null and Julia Lobur.
A guide to the MARIE machine simulator environment, 2010.
https://cs.msutexas.edu/~simpson/wordpress/wp-content/uploads/2012/12/MarieGuide.pdf.

Reverse Polish Notation (RPN)

Let's see the operation

$$3 + 7 \times 5$$

What is the result?

Reverse Polish Notation (RPN)

Let's see the operation

$$3 + 7 \times 5$$

What is the result?
50 or 38?

Reverse Polish Notation (RPN)

Let's see the operation

$$3 + 7 \times 5$$

What is the result?

50 or 38?

Which one is correct?

Reverse Polish Notation (RPN)

Let's see the operation

$$3 + 7 \times 5$$

What is the result?

50 or 38?

Which one is correct?

But, still, there is a lot of simple calculators that are "bad?"

“Priority” of arithmetic operations

1. power
2. multiplying and dividing
3. summing and subtracting

We can use the parentheses for changing the order of operations.

“Priority” of arithmetic operations

1. power
2. multiplying and dividing
3. summing and subtracting

We can use the parentheses for changing the order of operations.

BTW, What about changing the sign?

Do there exist unambiguous notation?

Polish logician, Łukasiewicz, introduced “prefix notation”. Instead of writing $z = x + y$ he proposed notation:

$$+xy$$

Do there exist unambiguous notation?

Polish logician, Łukasiewicz, introduced “prefix notation”. Instead of writing $z = x + y$ he proposed notation:

$$+xy$$

Let's note that it is very similar to writing a function of two variables:

$$z = f(x, y)$$

Do there exist unambiguous notation?

Polish logician, Łukasiewicz, introduced "prefix notation". Instead of writing $z = x + y$ he proposed notation:

$$+xy$$

Let's note that it is very similar to writing a function of two variables:

$$z = f(x, y)$$

The sum function $(+)$ has two arguments:

$$z = +(x, y)$$

Polish Notation

Operation $3 + 7 \times 5$ mening $3 + (7 \times 5)$ we can note:

$$\underbrace{+\underbrace{\times 7\ 5}_{35}\ 3}_{38}$$

Reverse Polish Notation

For some practical reasons we use "postfix notation," writing the operation (operator) **after** its arguments:

$$xy+$$

We call this notation Reverse Polish Notation (RPN)
So we can write this operation like this:

$$7\ 5 \times 3+$$

and a more complicated example, as:

$$A\ B\ +\ C\ D\ + \times W\ /$$

Reverse Polish Notation: Stack

Practical realization of this operation

$$A\ B\ +\ C\ D\ +\times W\ /$$

requires a stack and additional operations in internal language.

- `Push` writes accumulator's content to the stack.
- `Pop` takes value from the top of the stack and puts it in the accumulator.

Reverse Polish Notation: Stack

Practical realization of this operation

$$A\ B\ +\ C\ D\ +\times W\ /$$

requires a stack and additional operations in internal language.

- `Push` writes accumulator's content to the stack.
- Pop takes value from the top of the stack and puts it in the accumulator.

The stack is, sometimes called LIFO (Last In First Out) memory...

Reverse Polish Notation: Stack

Practical realization of this operation

$$A\ B\ +\ C\ D\ + \times W\ /$$

requires a stack and additional operations in internal language.

- `Push` writes accumulator's content to the stack.
- Pop takes value from the top of the stack and puts it in the accumulator.

The stack is, sometimes called LIFO (Last In First Out) memory...

The queue (aka "line") is called FIFO (First IN First Out) memory...

The idea of a stack