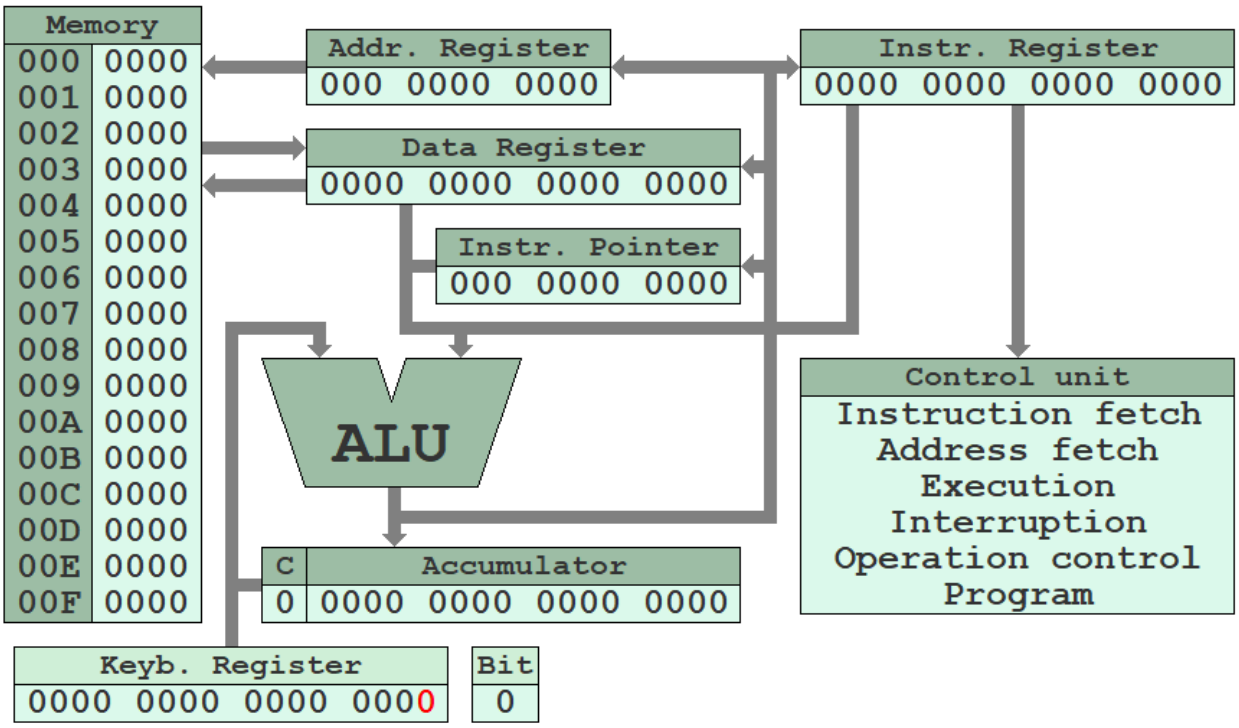
ITMO UNIVERSITY

**D.B. Afanasev, I.A. Bessmertny, S.V. Bykovskii, A.G. Ilina, S.V. Klimenkov, J.A. Koroleva**

# Basic Computer

## Study guide

## Part 1




**Saint Petersburg**
**2020**

**MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION**
**ITMO UNIVERSITY**

**D.B. Afanasev, I.A. Bessmertny, S.V. Bykovskii, A.G. Ilina, S.V. Klimenkov, J.A. Koroleva**

# Basic Computer
# Study guide
# Part 1

**RECOMMENDED AS A STUDY GUIDE**

**09.03.01 «Informatics and Computer Engineering»**
**ITMO University and Hangzhou Dianzi University joint Bachelor's program**
**«Computer Science and Technology»**


**Saint Petersburg**
**2020**

D.B. Afanasev, I.A. Bessmertny, S.V. Bykovskii, A.G. Ilina, S.V. Klimenkov, J.A. Koroleva. Basic Computer study guide. Part 1 – Saint Petersburg: ITMO University, 2020. - 112 p.

Reviewer: Polyakov V.I., associate professor, Faculty of Software Engineering and Computer Systems, ITMO University.

The study guide contains theoretical and practical materials to conduct a laboratory work on «Computer Basics». It covers such topics as computer architecture, data representation in computers, processor instruction formats, principles of low-level program execution, concepts of program flow, subprograms and machine cycles.

The study guide explains how a computer executes programs and how it handles data. For practice the students are offered to use a simplified Basic Computer Model, which was designed at ITMO and successfully tested by generations of students. The model allows rapid development of skills to interact with internal features of computers, such as registers, internal memory, control and arithmetic units. It helps to create knowledge background to study more complex concepts in programming and computer architecture.

The study guide is for foreign computer science students of 09.03.01 «Informatics and Computer Engineering» of ITMO University and Hangzhou Dianzi University joint bachelor's program «Computer Science and Technology».

ITMO UNIVERSITY

**ITMO University** - is the leading Russian university in the field of information and photonic technologies, one of the Russian universities with the status of the national research university granted in 2009. Since 2013 ITMO University has been a participant of the Russian universities' competitiveness raising program among the world's leading academic centers known as "5 to 100".

# Contents

# Glossary

**A**

| | |
|---|---|
| Abacus | a simple mechanical device that was used to perform arithmetical operations in the ancient Near East, Europe, China, and Russia. |
| Access time | a time period that is needed to reach a resource like a memory, a computer etc. |
| Accumulator | a register used by the processor to store results of arithmetic and logic operations from ALU. |
| Addition | adding numbers or amounts together. |
| Address bus | signal lines (wires) that are used in a computer to transfer the address value to the memory or to peripheral devices to define a place of data reading and writing. |
| Address register | a register that holds the address value. |
| ALU | Arithmetic and Logic Unit. It is a part of the processor. |
| Analog computation | a kind of computation with data in analog representation. |
| Analog computer | a computer which works with analog signals. |
| Analog signal | a continuous signal. |
| Arithmetic operation | an operation with numbers or amounts such as addition, subtraction, multiplication and division. |
| Arithmometer | the first digital mechanical calculator. |
| Application | a computer program that runs on a computer. |
| Assembler | a computer program that converts a mnemonic program code to a machine code. Assembler is also a language that is used to write a mnemonic program code. |
| Assembly operation | an operation that represents a mnemonic code in a machine code. |

**B**

| | |
|---|---|
| Bit | a minimal unit to represent information in digital computers. |
| Bit rate | how many bits per second can be transmitted using predefined channels. |
| Binary system | a system that uses only two symbols (0 and 1) to represent data. |
| Binary number | a number in the binary system. |
| Branch (in program) | a point in a computer program, where the algorithm can be changed depending on a condition. |
| Byte | a unit that is used to measure the amount of digital information. It equals 8 bits. |

**C**

| | |
|---|---|
| Calculator | a device that performs mathematic operations with numbers. |
| Capacity (for memory) | amount of data that can be stored in the memory. |
| Circuit design | a process of creating digital electronic circuits. |
| CISC | Complex Instruction Set Computer. |
| Cluster computing | computing on a set of tightly connected computers by high speed communication channels. For users it looks like a one hardware resource. |
| Command line | an interface that is used to enter commands for an operating system. |

| | |
|---|---|
| Compiler | a computer program translating program commands which are written in a programming language into sets of machine codes. |
| Complement code | a code that is used to represent negative numbers. |
| Computer | a machine that can handle data and perform calculations automatically according to the predefined program. |
| Computer architecture | a set of rules and methods that describe the functionality, organization, and implementation of computer systems. |
| Computer basics | basic (fundamental) knowledge about the way a computer works. |
| Clock signal | a signal that defines the time in a computer. |
| Computer data | data which a computer operates with. They could be numbers, symbols, and computer commands. |
| Computer memory | a device that stores data. |
| Control bus | signal lines (wires) that are used in a computer to transfer control signals (like read/write signals) between its units such as ALU, the memory, registers etc. |
| Control flow | the order in which program instructions or function calls are executed. |
| Control path | a scheme of control signals transmission between functional units such as ALU, the memory, registers etc. |
| CPU | Central Processing Unit. |
| Cyclic shift | the operation of moving bits in a register. In the left cyclic shift, the last bit goes to the place of the first bit, i.e. the bits go to the left. In the right cyclic shift, the bits go to the right (another direction) and the first bit goes to the place of the last bit. |

**D**

| | |
|---|---|
| Data bus | signal lines (wires) that are used in a computer to transfer data between its units such as ALU, the memory, registers etc. |
| Data register | a register in a processor that stores data. |
| Data representation | a format of data storing or exchanging. |
| Data storage | a storage for data, for example, a hard disk. |
| Datapath | a scheme of data signals transmission between functional units such as ALU, the memory, registers etc. |
| Data word | a byte of data equals to eight bits. |
| Digital computer | a computer in which data are represented as digital signals. |
| Digital design | designing, developing and debugging electronic schemes using logic gates. Circuit design |
| Digital signal | a signal that uses data represented as a sequence of discrete values. |
| Distributed computing | a field of computer science that studies computer performance improvement and scalability using calculation on several computers which are located in different places. |
| Display | an output device for representing graphical information. |
| Discretization | measuring values in discrete time moments and storing them as a sequence of discrete numbers. |
| Division | an arithmetic operation that shows how many times a number is contained in a larger number. |

**F**

| | |
|---|---|
| Fan or cooler | a device for producing airflow, often for cooling. |
| File system | a system or a computer program that defines and controls how data is stored and fetched. |
| Firmware | a class of computer programs that provides low-level control for computer hardware. |
| Fixed point numbers | a real data type for a number that has a fixed number of digits after the decimal point. |
| Float point numbers | a real data type for a number that has a float position of the decimal point. This position is indicated as the exponent component. |

**G**

| | |
|---|---|
| Gate | a circuit part which controls outputs and inputs according to the truth table. |

**H**

| | |
|---|---|
| Hard disk | an electro-mechanical data storage device that uses magnetic storage to maintain and retrieve digital information. |
| Hardware | physical parts of a computer. |
| Hexadecimal (hex) numbers | a system that uses the hexadecimal format (sixteen symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) to represent data. |
| Human-machine interface (HMI) | an interface that is used to communicate between humans and computers. |

**I**

| | |
|---|---|
| Instruction set | a set of instructions that a processor can execute. |
| Integrated circuit | a set of electronic circuits on one small piece of semiconductor material. |
| Instruction pointer | a processor register that stores the address of the next instruction to execute. |
| Instruction register | a processor register that stores the value of instruction that is being executed. |
| I/O System | an input/output system of a computer. |

**L**

| | |
|---|---|
| Layout (in digital circuits design) | an arrangement of elements in digital circuits. |
| Logical operation | a boolean logic operation. |

**M**

| | |
|---|---|
| Machine code | data and instruction representation that can be handled by a computer (machine). |
| Mainframe | big computers to process large amounts of data. |
| Memory | a device for storing data. |
| Microprogram | a program that implements a higher-level processor instruction. |

| | |
|---|---|
| Mnemonic code | a program code which represents every machine operation (instruction) in a readable format for the user. For example, mnemonic ADD stands for addition. |
| Motherboard | the main printed circuit board found in general purpose computers. It holds CPU and the memory allowing communication between them, and provides connectors for peripheral devices. |
| Multiprogramming | a feature that enables to run several programs simultaneously. |
| Multiplication | an arithmetic operation of adding a number to itself a particular number of times. |

**N**

| | |
|---|---|
| Natural language interface | a type of computer human interface where verbs, phrases and clauses act as control commands for creating and modifying data in computers. |
| Negative numbers | a real number that is less than zero. |
| Numbers range | a set of numbers. |

**P**

| | |
|---|---|
| Performance | an amount of instructions that a computer performs in a unit time. |
| PC | personal computer. |
| Parallel data transfer | transfer of several data bits simultaneously. |
| Personal computer | a multi-purpose computer with the size, capabilities, and price feasible for an individual user. |
| Pipeline | a technique for implementing instruction-level parallelism within a single processor. |
| Pop (memory operation) | fetching data from a predefined place. |
| Printing | a way to have a hardcopy of data stored in a computer which are computing results. It is a way to transfer data outside the computer. |
| Processor | a part of a computer that executes program. |
| Processor instruction | a command of a processor to do some manipulations with data. |
| Program | a sequence of instructions that make a computer perform an action or a particular type of work |
| Program model of processor | a representation of a computer from a programmer's point of view. |
| Programmer | a person who writes programs. |
| Programming | a process of creating a program. |
| Programming language | a language that is used for writing a program. |
| Push (memory operation) | storing data to a predefined place. |

**R**

| | |
|---|---|
| Register | a memory element that stores one word of data. |
| RISC | Reduced Instruction Set Computer |
| ROM | Read Only Memory |

**S**

| | |
|---|---|
| Sequential data transfer | bit by bit data transfer. |
| Shift | an operation of moving bits in one direction inside the register. |
| Signal | an electrical signal – a time-varying physical value (for example, electric current, voltage, etc.) that contains information. |
| Stack (memory) | a kind of memory that is based on the principle of last-in-first-out pattern. |
| Software | a program or set of programs that tell the computer how to work. |
| Subtraction | an arithmetical operation of taking a number or an amount from another number or an amount. |
| Super-computer | a computer with a high level of performance. |
| System bus | signal lines (wires) that are used to connect CPU and the memory. |

**T**

| | |
|---|---|
| Terminal | a set of primary input and output devices for a computer |
| Transistor | a semiconductor device used to amplify or switch electronic signals and electrical power. It is a base of modern integrated circuits. |
| Trigger | a device that can be in one stable state in every moment of time. If an event occurs, it changes its state. It can store one bit of data. |
| Truth table | a table that shows the dependence between input and output values in digital circuits. |

**U**

| | |
|---|---|
| User interface | space where humans and machines communicate. |

# Introduction

This study guide discusses the basic concepts of computer organization and functioning. The study guide is divided into two chapters.

The first part provides information about data representation in a computer, how a computer executes program on a low level, the theory about the basic computer architecture, instruction formats and how a computer performs operations. The basic concepts are explained using the Basic Computer Model developed at ITMO University. This is a simplified model that has typical features of the widely used digital computers.

The second chapter contents a set of laboratory works that can be used to build up knowledge and develop skills of understanding a program execution process using the Basic Computer Model. Each laboratory work contents the step-by-step guides that help to successfully complete the task.

The assignments for the laboratory works cover such topics as:

- basic principles of program execution in computer;
- low level instruction execution;
- program control flow;
- subprograms.

To work with the book the students need:

- The Basic Computer Model that can be downloaded by this link: https://se.ifmo.ru/web/guest/bcomp.jar
- JRE (Java Runtime Environment) Version 8 Update 221 or newer. It can be downloaded using this link: https://java.com/ru/download/.

# 1. The Basic Computer model

## 1.1 Purpose and description of the Basic Computer model

Basic Computer is a simple hypothetical machine that has the typical features of many specific computers. Awareness what parts a basic computer has and how it functions is quite useful for the development of microprocessor systems of any type. This is the reason why it is called a basic computer. It is highly advisable to start studying computers with a low-level machine.

Figure 1 shows the simple structure of a basic computer. It is a unicast (single-address) computer of accumulator type, it works with 16-bit words. It implements two types of addresses: direct and indirect. The smallest addressable unit is a word that consists of eight bits or one byte.

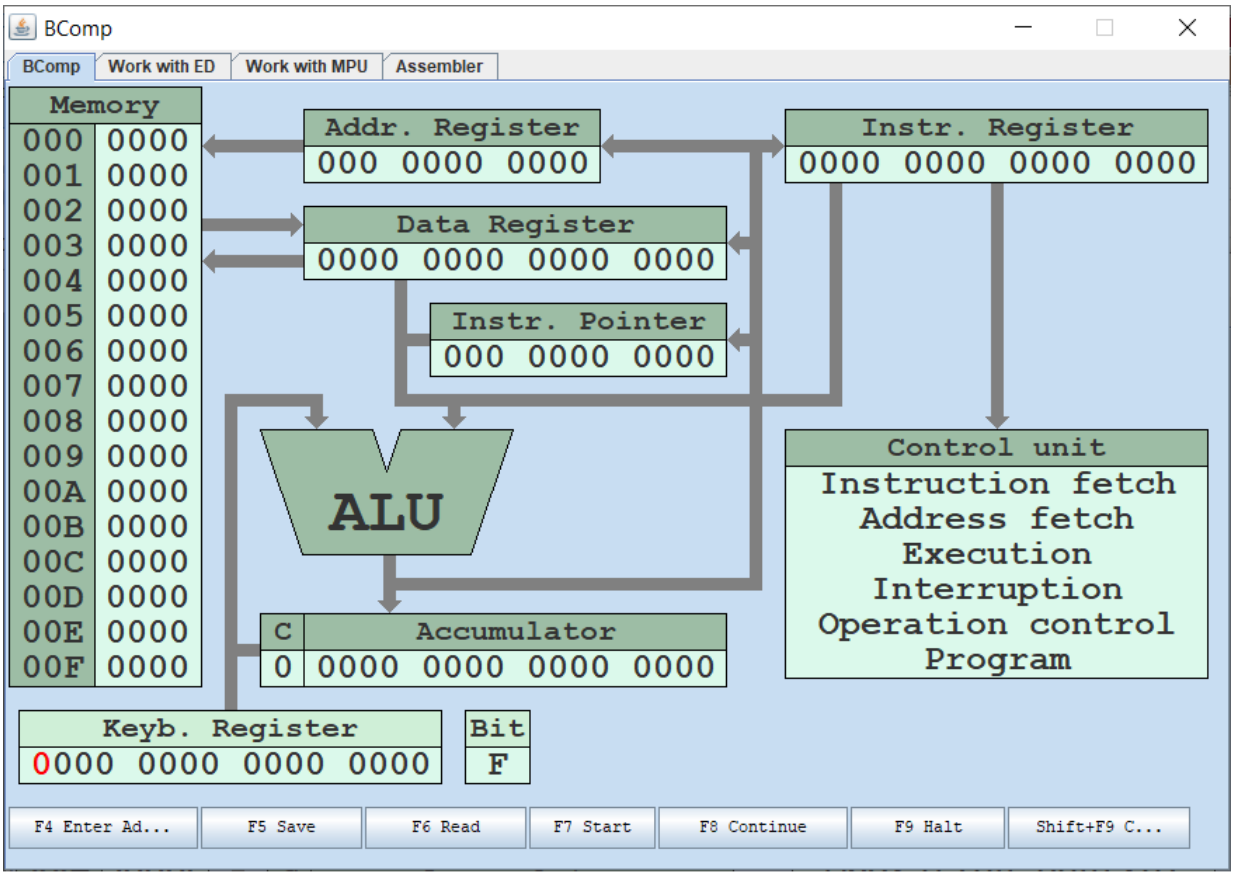



*Figure 1. The Basic computer model*

Consider the components of Basic Computer:

- **Memory** consists of 2048 cells (1 cell = 16 bit). These cells have addresses from 0 to 2047. A hex address range 008 to 00F are index cells. They are intended for addressing mode organization, usually, they are useful in loops.
- **The Processor** consists of several registers, an arithmetic and logic unit, and a control unit.
- **Arithmetic and logic unit** (ALU) usually performs arithmetic operations, such as addition and addition-with-carry. A carry can be obtained as a result of the previous operation, which is the operation of logical multiplication and inversion. All transfers between registers are also performed via the ALU.
- **Instruction Pointer** (IP) is a register, that stores the address of a memory cell, which contains the address of the next command to be executed. The length of IP is 11 bit, so it

allows to address 2048 memory cells. In literature this register can be also referred as the program counter.

- **Address register** (AR) also has 11-bit length. It contains the address of the memory cell accessed by the processor.
- **Instruction register** (IR) has 16-bit length. It stores the code of the currently executing command.
- **Data register** (DR) has 16-bita length. It is used for temporary storage of 16-bit words during the exchange of data between the memory and the processor.
- **Buffer register** (BR) has 17-bit length. It is used for temporary storage of calculation results from ALU. This register is also used when performing shifts.
- **Accumulator** (A) is a 16 -bit register. One operand for arithmetic and logic operations is in the accumulator; the second one is in the data register. The intermediate result is stored in the accumulator register.
- **One-bit registers (flags)** are useful for both arithmetic and logic operations:
    - **Carry flag** (C) is a flag register used to indicate when an accumulator register is overflowed.
    - **Zero flag** (Z) is a single-bit register. It is set to 1 or true, if an arithmetic result in the accumulator register is zero or otherwise it resets.
    - **Sign flag** (N) stores a number sign value from the accumulator register, it is a duplicate of the fifteenth bit of this value.

## 1.2 Basic Computer instructions

The basic computer can execute the predefined instruction set. While writing a program, a user is limited by these instructions. The basic computer instructions can be divided into three groups depending on blocks they refer to.

- address instructions;
- no-address instructions;
- input/output instructions.

Address instructions command the machine to perform actions with a memory location which address is specified in the address part of the instruction.

No-address instructions perform various actions without reference to a memory location.

For example, the CLA instruction (Table 1) commands the basic computer to clean the accumulator register (write a zero-code into A). This is an instruction to process an operand, which is located in a specific place. The machine “knows” the place, so the address is not needed. Another example of no-address instruction is the HLT instruction.

I/O instructions control the exchange of data between the processor and external computer devices.

A complete list of basic computer instructions is given in Table 1.

*Instruction formats and addressing mode.*

The developers of the computer choose three formats of 16-bit (single-word) commands with a 4-bit operation code (figure 2). In memory access instructions, 11 bits are allocated to the address. Therefore, it is possible to directly address $2^{11}$= 2048 memory cells, i.e. to the entire memory of the basic computer, (which is direct addressing). In this case, the address type bit should contain 0. If 1 is set in this bit, then the address located in the address part of the command indicates the cell that contains the operand address; this is indirect addressing.

Note that when mnemonically writing instructions, indirect addressing is indicated by enclosing the address in brackets. For example, the instruction ADD (25) means to add the contents of A to the content of the cell which address is stored in cell 25 (indirect addressing).

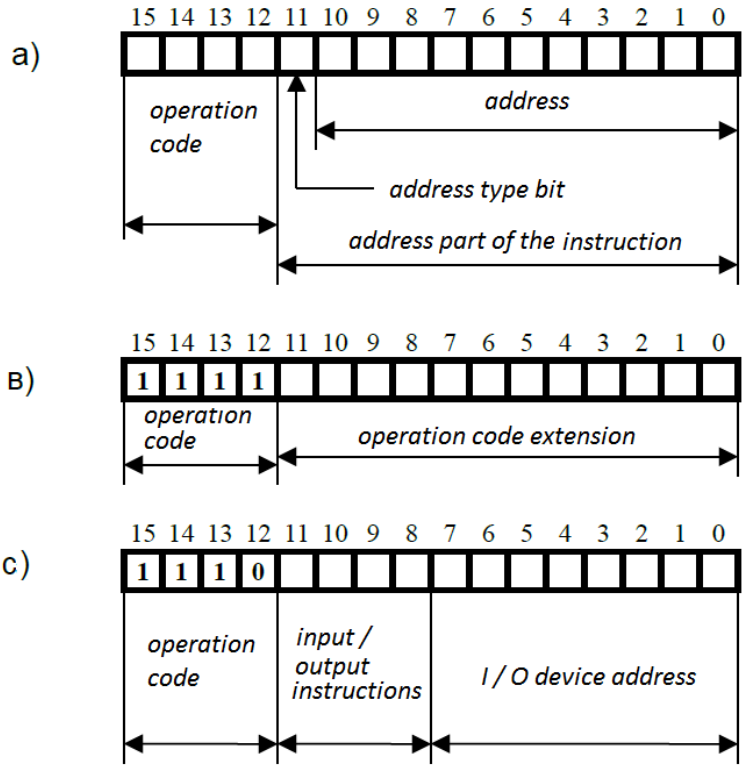



*Figure 2. Instruction formats: a – address; b – no-address; c – i/o instructions*

## 1.3 Representation of integers in Basic Computer

Unsigned binary integers can be used to represent zero and positive integers. Placing such numbers in one 16-bit word, they can vary from $(0000\ 0000\ 0000\ 0000)_2 = (0000)_{16} = 0$ to $(1111\ 1111\ 1111\ 1111)_2 = (FFFF)_{16} = 2_{16} - 1 = 65535$. This value representation is called a direct number code.

Such values (signed binary numbers) are fixed-point numbers that separate the integer and fractional parts of a number. In the numbers used in the basic computer, the position of the decimal point is strictly fixed after the least significant bit of the word.

The signed binary numbers are used when distinguishing between positive and negative numbers is necessary. In modern computers, an additional code represents signed integers. The most significant bit of the format determines the sign of a number: 0 for positive numbers and 1 for negative ones. Obviously, a twos-complement form of positive number matches its direct code. To represent a negative number in the twos-complement form, the direct code of the number module should be inverted (obtaining the reverse number code) and add 1 to the result. The same operation is used while changing the sign of a number represented in the twos-complement form.

(M), (A), (IP), (C), (B) – the content of a cell, where the address is memory, accumulator, the instruction pointer, carry register, data register and I / O devices with address B.

XXX – memory cell address.

XX – I / O device address.

*Table 1. Basic computer instructions*

**Address Instructions**

| Name | Mnemonic name | Code | Description |
|---|---|---|---|
| Conjunction | `AND M` | 1XXX | (M) & (A) ➔ A |
| Move | `MOV M` | 3XXX | (A) ➔M |
| Addition | `ADD M` | 4XXX | (M) + (A) ➔ A |
| Addition with carry | `ADC M` | 5XXX | (M) + (A) + (C) ➔ A |
| Substraction | `SUB M` | 6XXX | (A) – (M) ➔ A |
| Branch if carry is set | `BCS M` | 8XXX | If (C) = 1, then M ➔ IP |
| Branch if plus | `BPL M` | 9XXX | If (N)= 0, then M ➔ IP |
| Branch if minus | `BMI M` | AXXX | If (N) = 1, then M ➔ IP |
| Branch if zero | `BEQ M` | BXXX | If (Z) = 1, if M ➔ IP |
| Unconditional branch | `BR M` | CXXX | M ➔ IP |
| Increment and skip | `ISZ M` | 0XXX | M + 1 ➔ M, if (M) >= 0, then (IP) + 1 ➔ IP |
| Subprogram call | `JSR M` | 2XXX | (IP) ➔ M, M + 1 ➔ IP |

**No-address Instructions**

| Name | Mnemonic name | Code | Description |
|---|---|---|---|
| Clear accumulator register | `CLA` | F200 | 0 ➔ A |
| Clear carry register | `CLC` | F300 | 0 ➔ C |
| Invert accumulator register | `CMA` | F400 | (!A) ➔ A |
| Invert carry register | `CMC` | F500 | (!C) ➔ C |
| Left cyclic shift by 1 bit | `ROL` | F600 | A & C content moves left, A(15) ➔ C, C ➔ A(0) |
| Right cyclic shift by 1 bit | `ROR` | F700 | A & C content moves right, A(0) ➔ C, C ➔ A(15) |
| Increment accumulator register | `INC` | F800 | (A) + 1 ➔ A |
| Decrement accumulator register | `DEC` | F900 | (A) – 1 ➔ A |
| Halt | `HLT` | F000 | |
| No operation | `NOP` | F100 | |
| Enable interruptions | `EI` | FA00 | |
| Disable interruptions | `DI` | FB00 | |

**I/O Instructions**

| Name | Mnemonic name | Code | Description |
|---|---|---|---|
| Clear flag | `CLF B` | E0XX | 0 ➔ device flag |
| Check flag | `TSF B` | E1XX | If (device flag B) = 1, then (IP) + 1 ➔ IP |
| Input | `IN B` | E2XX | (B) ➔ A |
| Output | `OUT B` | E3XX | (A) ➔ B |

To represent the number -70910 in twos-complement form one needs:

1. Write the direct code of the module of a given number
$0\ 000\ 0010\ 1100\ 0101_2 = |709_{10}|$

2. Invert it (all 0 must be changed to 1, and all 1 must be changed to 0)
1 111 1101 0011 1010

3. Add 1 to the result

1 111 1101 0011 1010
+ 1
_____
1 111 1101 0011 1011 This is the -709 in twos-complement form

Check the correctness of the calculations. Let's add two numbers $709_{10}$ and $-709_{10}$ :

0 000 0010 1100 0101 it is 709
+ 1 111 1101 0011 1011 it is -709
_____
0 000 0000 0000 0000 the result is 0

Since the carrying from the high order extends beyond the boundaries of the bit grid, it is not considered. The rest 16-bit sum is zero, so it confirms the correctness of the conversion.

The use of twos-complement form simplifies computer designing, it allows to avoid subtracting a smaller module number from a larger one adding two numbers with different signs. After it, the sign of a larger number is assigned to the result. In addition, the same adder circuit can be used to perform operations on the sign and unsigned representation of a number. An indication of going beyond the boundaries of the bit grid for an unsigned representation of a number is the transfer to the most significant bit (C bit means carry). An indication of an overflow of the bit grid for the sign representation is the overflow bit. In the basic computer, unfortunately, this feature of the result is not implemented. Consider the occurrence of these situations in the example of the numbers representation in a four-digit grid (figures 3a and 3b).

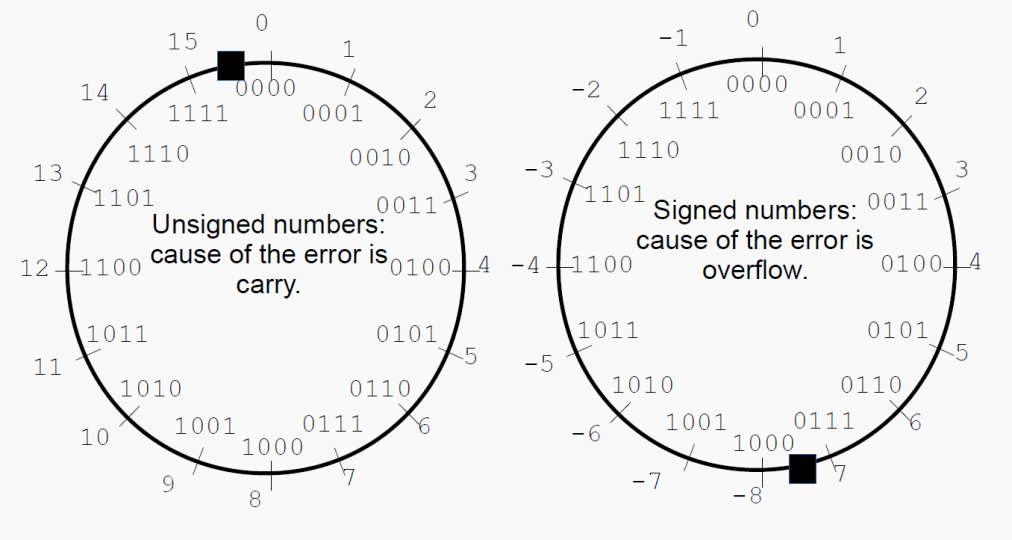



*Figure 3a. Overflow and carry examples*

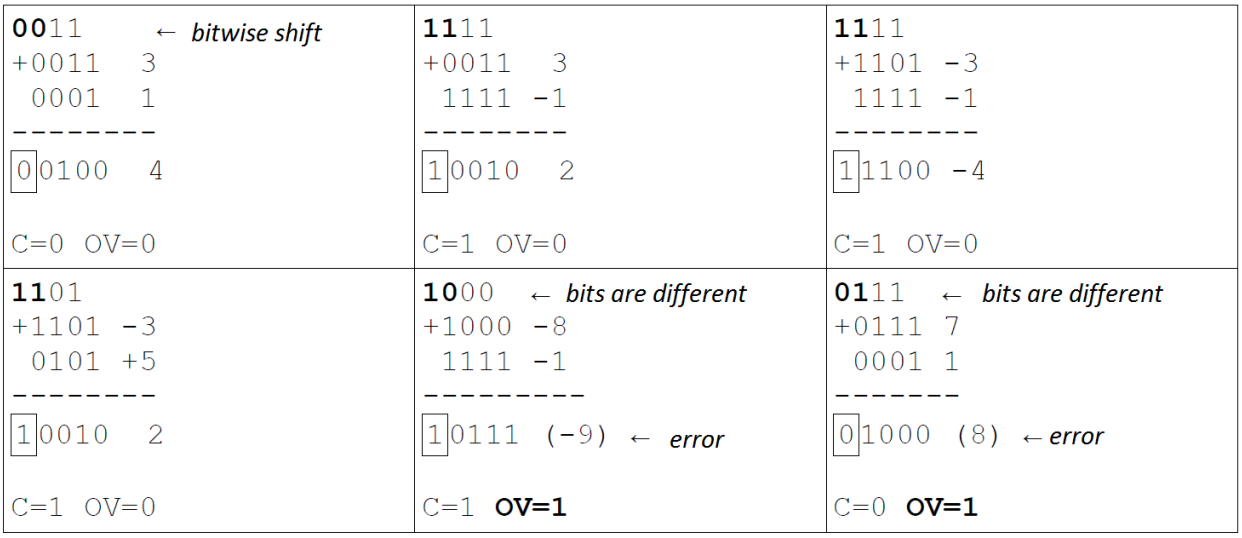



*Figure 3b. Overflow and carry examples*

The processor determines overflow by the following rule: if bitwise transfers in the sign and in the most significant bit are simultaneously absent or present, then there is no overflow; if bitwise transfers are present in only one of them, then there is an overflow of the sign bit grid.

## 1.4 Arithmetic operations

In the basic computer, ADD instruction is used for the addition of binary integers, both signed and unsigned.

To increase the value in the accumulator by one, the accumulator uses INC instruction, while DEC instruction subtracts 1 from the value stored in the accumulator. If a transfer from the most significant bit of A occurs, then one is written into the carry register, otherwise 0 is written into it.

Subtraction (X-Y) can be performed by changing the sign of the subtracted (CLA, ADD Y, CMA, INC) and subsequent adding to the other argument (ADD X). Obviously, this requires the execution of several commands. In the basic computer, to reduce programs and subtraction execution time, the SUB Y (CLA, ADD X, SUB Y) instruction is provided, it implements the same actions taking less time.

In the basic computer there are no commands for performing multiplication and division (ALU does not perform such operations). Therefore, composition and quotient are obtained as a result of a program.

## 1.5 Shifts and logical operations

In any computer, bitwise data processing is performed by logical multiplication and cyclic shift instructions, as well as the instructions for inverting and cleaning of the accumulator and carry register.

Instruction AND M performs a logical multiplication operation on each bit of the accumulator content and the content of a memory cell M. The result of executing the instruction for each pair of operand bits equals one only when both bits are equal to one, and the result bit is zero in other cases.

Instructions ROL (cyclic left shift by one bit) and ROR (cyclic right shift by one bit) encircle the accumulator and the carry register into a ring and shift all bits of the ring on one bit to the left or right (figure 4). The operations of multiplication or division by two (one shift), by four (two shifts), by eight (three shifts), etc. can be realized by shifting the number to the left or to the right.

The CMA instruction (accumulator invert) inverts the contents of the accumulator bit by bit.

The CMC (invert carry flag) and CLC (reset carry flag) instructions invert and reset the state of the carry flag, respectively.

| | C flag | Accumulator |
|---|---|---|
| Before shift | 0 | 1011100000101011 |
| After left shift | 1 | 0111000001010110 |
| After right shift | 1 | 0101110000010101 |

*Figure 4. cyclic shifts*

## 1.6 Computational process control

In any computer, the task of managing the computing process, i.e. the required sequence of instruction execution, is solved by using the transition instructions (BCS, BPL, BMI, BEQ, BR), the instructions "Increment and skip" (ISZ) and "Stop" (HLT). All these instructions (except HLT) are addressable, i.e. they contain the address of a memory cell from which the next program instruction should be selected when one or another condition is true. If the conditions are not true, then the instruction is located after this control instruction is executed. Moreover, like in other address commands, indirect addressing can be used here. Jump instructions do not change the state of the accumulator and the carry register. These instructions can only change the contents of the instruction pointer by placing in it the address, determined by the address part of the instruction.

Instruction BCS M determines the jump to the instruction located in the memory cell with the address M, if the content of the carry register is 1.

Instruction BPL M defines the jump to the instruction, located in the memory cell with the address M, if the sign of the result N is 0. Therefore, it means the result is positive, i.e. it is greater than or equal to zero.

Instruction BMI M determines the jump to the instruction, located in the memory cell with address M, if the sign of result N is 1, so it means, the result is negative.

Instruction BEQ M determines the jump to the instruction, located in the memory cell with the address M, if the sign Z is 0 (zero result).

Instruction BR M jumps to the instruction, located in the memory cell with the address M.

Jump instructions are useful for organization of cyclic programs, which are used in the cases where it is necessary to perform a set of identical actions several times with different data. The basic computer has a number of tools to simplify cyclic programs. The feasibility of introducing these funds will be considered using the following examples.

Example 1. Get the result of Z = Y * 50.

Since in the Basic Computer command system there is no multiplication command, multiplication by a constant can be performed by combining the operations of shift and addition. To illustrate this, let's consider the simplest but non- optimal way: Y will be added Y value 50 times using the program shown in Table 2. Because in this program the accumulator is used not only to accumulate the composition, but also to change the number of completed cycles and compare them with the value of the multiplier, the intermediate results Z and C have to be stored in the memory. Obviously, this program can be simplified by calculating the number of completed cycles and checking the termination condition of a cyclic program that does not affect the content of the accumulator. This means ISZ instruction (increment and skip). Each time the ISZ M instruction is executed, 1 is added to the content of the memory cell with address M. If

the result is less than zero, then the instruction following ISZ M is executed. The instruction is located in the memory cell with address A + 1. Otherwise, this instruction is skipped, that is, the instruction located at address A + 2 is executed. The example of a program using the ISZ instruction is given in Table 3.

*Table 2. First example of program for getting Z = Y * 50*

| **Address** | **Content** | | **Comments** |
|---|---|---|---|
| | **Code** | **Mnemonics** | |
| 5 | 0078 | Y | Multiplicand (in decimal it is 120) |
| 6 | 0000 | Z | Memory cell for result |
| 7 | 0032 | M | Multiplicand 50 = (32)16 |
| 8 | 0000 | C | The cell used to accumulate the number of completed cycles - cycle counter |
| ... | ... | | |
| 10 | F200 | CLA | Clear accumulator |
| 11 | 4006 | ADD 6 | To the intermediate result located in cell 6, one more value of the Y is added |
| 12 | 4005 | ADD 5 | |
| 13 | 3006 | MOV 6 | |
| 14 | F200 | CLA | |
| 15 | 4008 | ADD 8 | Content of the cycle counter is incremented by 1, and its copy is stored in accumulator |
| 16 | F800 | INC | |
| 17 | 3008 | MOV 8 | |
| 18 | 6007 | SUB 7 | If the content of the cycle counter is less than the multiplier value, it goes to the instructions performing a new summation of Y with intermediate Z value |
| 19 | A010 | BMI 10 | |
| 1A | F000 | HLT | Stop. The result is in the memory cell 6 |

*Table 3. Second example of program for getting Z = Y * 50*

| **Address** | **Content** | | **Comments** |
|---|---|---|---|
| | **Code** | **Mnemonics** | |
| 5 | 0078 | Y | Multiplicand |
| 6 | 0000 | Z | Memory cell for result |
| 7 | FFCE | M | Negative value of multiplicand (-50) |
| ... | ... | | |
| 10 | F200 | CLA | Clear accumulator |
| 11 | 4005 | ADD 5 | Add Y to accumulator content |
| 12 | 0007 | ISZ 7 | Increment M on 1 and execute instruction BR 11 if M below Zero. If M = 0 then instruction BR 11 skipped. |
| 13 | C011 | BR 11 | |
| 14 | 3006 | MOV 6 | Result of 50 addition sets to cell 6 |
| 15 | F000 | HLT | Stop computer |

Example 2. Get sum of 32 array elements in memory cell 005. All array elements are stored in the memory cell with addresses from 010 to 02F.

In the previous task, the content of the same memory cell (Y) is summarized many times, so in this example, one needs to summarize the content of different memory cells. If the computer instructions allowed only direct addressing of memory cells, then a program for solving the task would either have to use 32 addition instructions (4010, 4011, ..., 402E, 402F), or apply the modification address part of the addition instruction. The latter is implemented in the program that is described in Table 4.

Usually, modern computers do not use instruction modifications. To ensure the ability to work with uploaded programs in read-only memory devices (instructions can only be read), special addressing tools were developed, the one of these tools is indirect addressing.

*Table 4. First example of summarize array elements program*

| Address | Content | | Comments |
|---|---|---|---|
| | Code | Mnemonics | |
| 5 | 0000 | | Memory cell for result |
| 6 | FFE0 | | Negative count of array elements |
| ... | | | |
| 10 | | | Number values of array elements |
| . | | | |
| . | | | |
| . | | | |
| 2F | | | |
| 30 | F200 | CLA | The temporary result (cell 5) is added to the content of the array element which address is located in the address part of the instruction in cell 32 (first this address is 10, and then it increment by 1 by instructions from 34 to 37 on each loop iteration) |
| 31 | 4005 | ADD 5 | |
| 32 | 4010 | ADD 10 | |
| 33 | 3005 | MOV 5 | |
| 34 | F200 | CLA | Move instruction into accumulator that located in memory cell 32, increment its content by 1 and save modified instruction in old place (memory cell 32) |
| 35 | 4032 | ADD 32 | |
| 36 | F800 | INC | |
| 37 | 3032 | MOV 32 | |
| 38 | 0006 | ISZ 6 | Increment array elements counter by 1 and go to instruction 30 while counter is less than 0 |
| 39 | C030 | BR 30 | |
| 3A | F000 | HLT | Stop computer |

To use indirect addressing, one needs to select in the computer memory a cell (for example, 007), write the address of the first element of the summed array (address 010), replace in the program Table 4 instruction 4010 to instruction 4807 (cell 32) and replace the modification instructions with the calculation instructions of the current address of the array element to be summed. You can get a more compact program if you use instruction ISZ 7 (without changing the content of the accumulator) to calculate the current address of the array element to sum. The example of the program is described in Table 5.

*Table 5. Second example of summarize array elements program*

| Address | Content | | Comments |
|---|---|---|---|
| | Code | Mnemonics | |
| 5 | 0000 | | Memory cell for result |
| 6 | FFE0 | | Negative count of array elements |
| 7 | 0010 | | Current address of array element |
| ... | | | |
| 10 | | | Number values of array elements |
| . | | | |
| . | | | |
| 2F | | | |
| 30 | F200 | CLA | Clear accumulator |
| 31 | 4807 | ADD (7) | Summarize next array element |
| 32 | 0007 | ISZ 7 | Increment current address of array element by 1 |
| 33 | F100 | NOP | Instruction “No operation” |
| 34 | 0006 | ISZ 6 | Increment elements counter by 1 and go to instruction 31 |

| | | | |
|---|---|---|---|
| 35 | C031 | BR 31 | while elements counter is less than zero |
| 36 | 3005 | MOV 5 | Set result into memory cell 5 |
| 37 | F000 | HLT | Stop computer |

*Table 6. Third example of summarize array elements program*

| **Address** | **Content** | | **Comments** |
|---|---|---|---|
| | **Code** | **Mnemonics** | |
| 5 | 0000 | | Memory cell for result |
| 6 | FFE0 | | Negative count of array elements |
| ... | | | |
| F | 0010 | | Address of the first element in the array |
| 10 | | | Number values of array elements |
| . | | | |
| . | | | |
| . | | | |
| 2F | | | |
| 30 | F200 | CLA | Clear accumulator |
| 31 | 480F | ADD (F) | Add the next array element. In the first iteration the index memory cell F contains the address of the first element in the array (10). After the first execution of this instruction the content of memory cell F will be incremented by 1 and will be pointed to the second element of the array, after the second execution on the third element, etc. |
| 32 | 0006 | ISZ 6 | Increment the elements counter by 1 and go to instruction 31 while elements counter is less than zero |
| 33 | C031 | BR 31 | |
| 34 | 3005 | MOV 5 | |
| 35 | F000 | HLT | Stop computer |

Since the ISZ 7 instruction (Table 5) increment value (address), then after its execution, the instruction counter will be pointed to the instruction 34 (instruction at address 33 will be skipped). Therefore, in the cell 33 the instruction "No operation" is placed, but only a number could be placed. Finally, consider another tool to simplify cyclic basic computer programs - index cells (cells with addresses from 008 to 00F). If you indirectly address any of these cells, then firstly their content will be used as the address of the operand, and then will be automatically incremented by 1. Index cells are not changed by indirect addressing, as their content can only be changed only in the case of writing new information in the cell. The described condition of index cells allows creating optimal program to summarize the array elements (Table 6).

## 1.7 Subprograms

Usually, different parts of a program must execute the same actions related to data processing, for example, calculation of trigonometric function. In such cases, it is reasonable to move repeated parts of a program into a subprogram and to replace appropriated places of the same code with instruction that execute this subprogram. The instruction JSR is used for this purpose in the basic computer. The part of the main program that contains two instructions JSR 300 that jumps to subprogram execution, described on figure 5.

Instruction JSR 300 located in cell 25 writes number 25 + 1 = 26, which is the instruction pointer value after executing the instruction selection cycle, in the memory cell with address 300 and writes number 300 + 1 = 301 into the instruction pointer, which is the first instruction address of the subprogram. This is the way of jumping to the subprogram. Further subprogram instructions execute before the instruction BR (300) located in the memory cell 326.

The instruction of an unconditional jump with indirect addressing prescribes a jump to the instruction located at the address stored in the cell 300. Since the number 26 is previously written in this cell, it executes the instruction located in cell 26, i.e. following the call to the subprogram. Similarly, the instruction JSR 300 will be executed. This instruction is located in the memory cell 72;a jump to memory cell 73 will happen after subprogram instructions are executed.

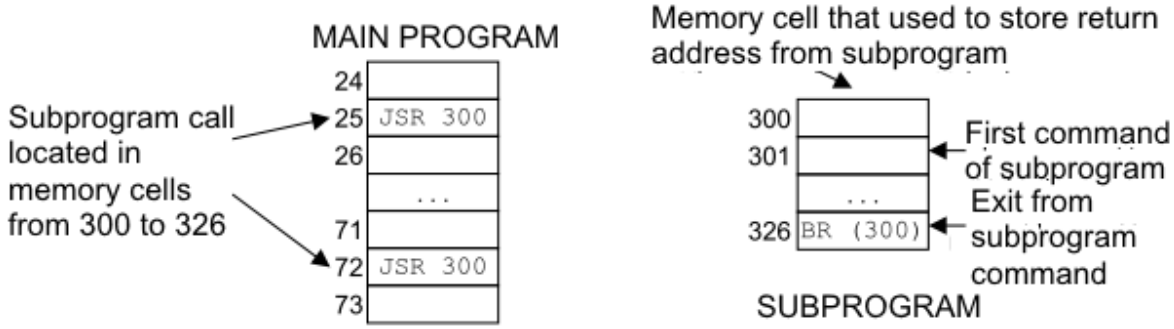



*Figure 5. Subprogram call and return from it*

Thus, when coding a subprogram before its first instruction you should place a cell that contains return address from the subprogram. The subprogram call instruction contains the address of this cell, for example, the address M in the instruction JSR M. To return from the subprogram you can use any jump instruction indirectly addressed to cell M, e.g. BR (M). It executes the jump to the instruction which address is stored in the first cell of the subprogram.

## 1.8 Machine instruction execution

Executing instructions, the computer control unit analyses and transfers an instruction, its individual parts (operation code, addressing mark and address) or operand from one computer register to its other register, ALU, memory or input / output device. These actions (microoperations) are coordinated and occur in defined time sequence. To ensure such a coordination, the computer uses clock generator cycles. The control device contains a sequence of actions for executing instructions and elementary operation called cycles.

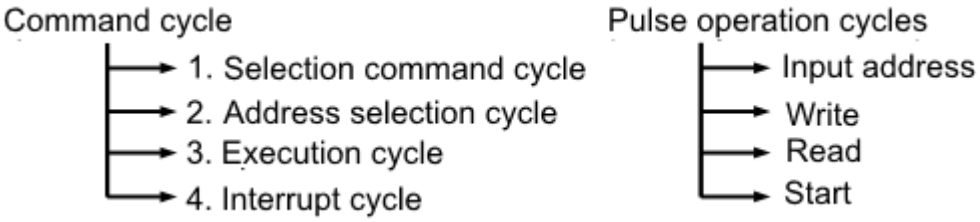



*Figure 6. Control unit cycles*

Instruction cycle. Many actions need to implement one instruction; each of these actions is initiated by one clock cycle. The total number of clock cycles required to execute an instruction determines the execution time, called the *instruction cycle*. The instruction cycle includes several *machine cycles*: instruction fetching, address fetching, execution and interruption. The main actions performed by the computer during each machine cycles are illustrated and described below.

*Instruction fetching*. In this machine, the cycle is executed by reading instruction from the memory and its partial decoding.

1. The instruction pointer content is written into the buffer register via ALU.
2. The buffer register's content is written into the address register (figure 7).
3. The memory cell content that is pointed in the address register is read from the memory to data register (figure 8) and the instruction pointer content goes to ALU where it is incremented by 1. The result is stored into the buffer register.
4. The buffer register's content is stored into the instruction pointer.
5. The data register is sent through ALU into the buffer register
6. The buffer register is stored into the instruction register.
7. The code of the instruction that is located in the instruction register partially decodes to understand the type of instruction (address, without address, input/output) and type of addressing for address instruction (direct, indirect) (figure 9)

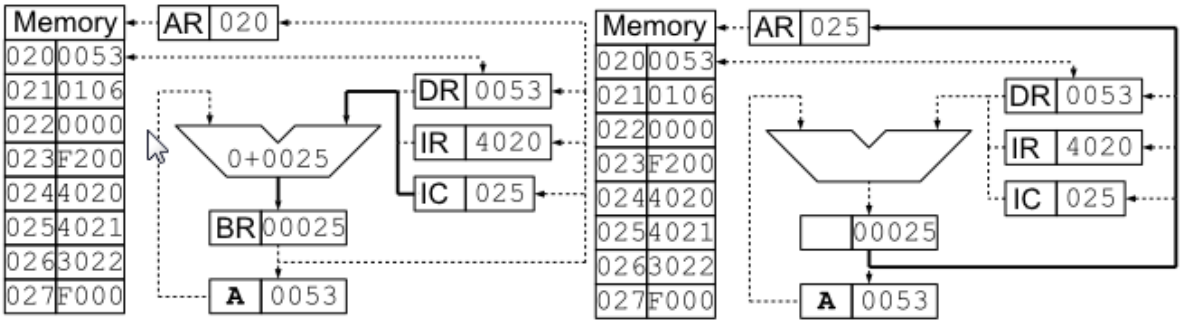



*Figure 7. Transfer IC into data register (pulses 1 and 2 of the instruction fetching)*

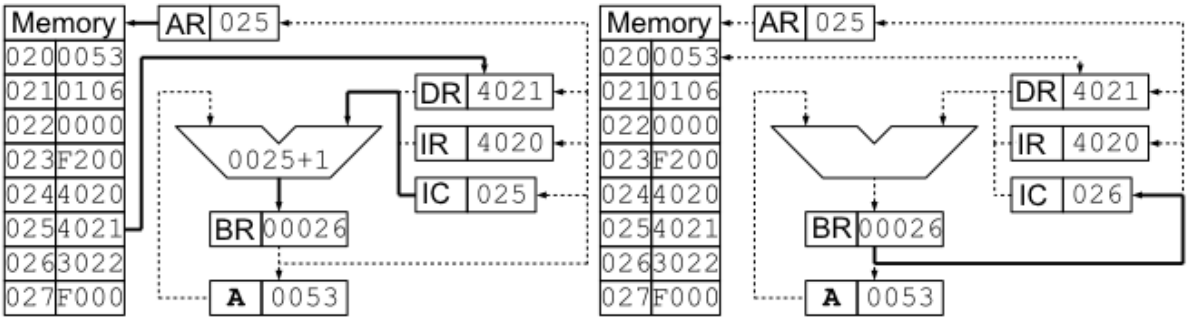



*Figure 8. Instruction fetching with the IC incrementing at the same time (pulses 3 and 4 of the instruction fetching)*

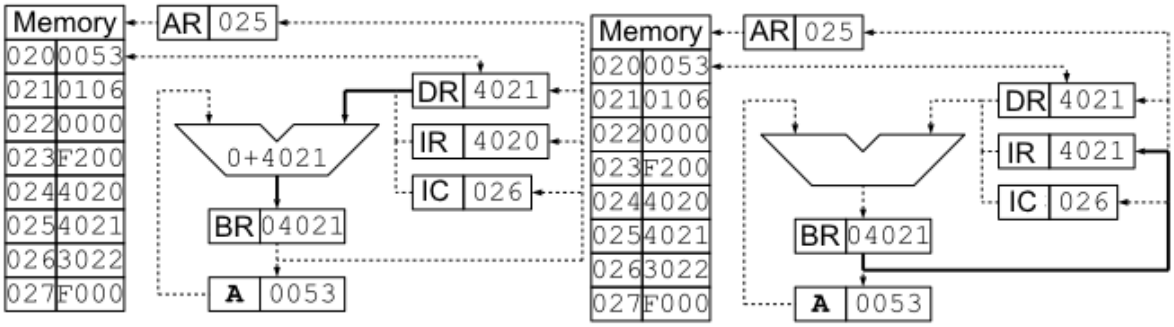



*Figure 9. Writing instruction code into IR (pulses 5,6 and 7 of the instruction fetching)*

Address fetching. This machine cycles are executed after instruction fetching only *for address instructions with indirect addressing*, i.e the instruction where the type addressing bit equals 1.

This cycle is used for reading an operand address from the memory, resulting and jumping. It consists of several steps:

1. The data register content is sent into the buffer register through ALU.
2. The address (lower 11 bits) of content is stored in the buffer, register, which is stored into the address register.
3. The memory cell content that points to the address register is read into the data register. Now, this register contains an operand address or a result address, or the jump address that will be used in an execution instruction cycle. If one of the index cells (addresses 8...F) is addressed indirectly, then the sampling cycle of the operand address (result) continues, otherwise, the machine cycle finishes.
4. The data register content goes to ALU where is incremented by 1. Then the result is stored into the buffer register.
5. The buffer register content is stored into the data register.
6. The changed content of the data register is transferred into the memory cell located at the address that is pointed in the address register.
7. The register content is transferred into ALU where it is decremented by 1 and sent into the buffer register.
8. The buffer register content is stored into the data register.

After the last operation, the data register restores the address value that was in the index cell before step 3. The index cell content is incremented by 1. The next time it is accessed and a new address will be selected.

Execution. The execution instruction determines the sequence of actions in this machine cycle.

1. For instructions that require fetching an operand from the memory (AND, ADD, ADC, SUB, ISZ), the execution cycle reads the operand into the data register and execution of the operation indicated by the instruction operation code. An example of the execution cycle of the ADD 21 instruction is given in figure 10.
2. By the transfer instruction (MOV), this machine cycle is written in the content of the accumulator into a memory cell with an address stored in the data register. To do this, the content of the data register is sent to the address register and the content of the accumulator in the data register and then in the memory cell, which is pointed by the address register.
3. When transition instructions (BCS, BPL, BMI, BEQ) are executed, the corresponding condition (1 - in the transfer register, 0 - in the N flag, etc.) is checked and the address from the data register will be transferred to the instruction pointer when fulfilling this condition. Otherwise, that instruction will be selected, which follows the transition instruction. When executing an unconditional jump instruction (BR) the transfer address is forwarded to the instruction pointer without any check.
4. During this machine cycle the subprogram call (JSR) instruction transfers the content of the instruction pointer to the memory cell, the address of which is contained in the data register, and stores the data register content incremented by one in the instruction pointer.

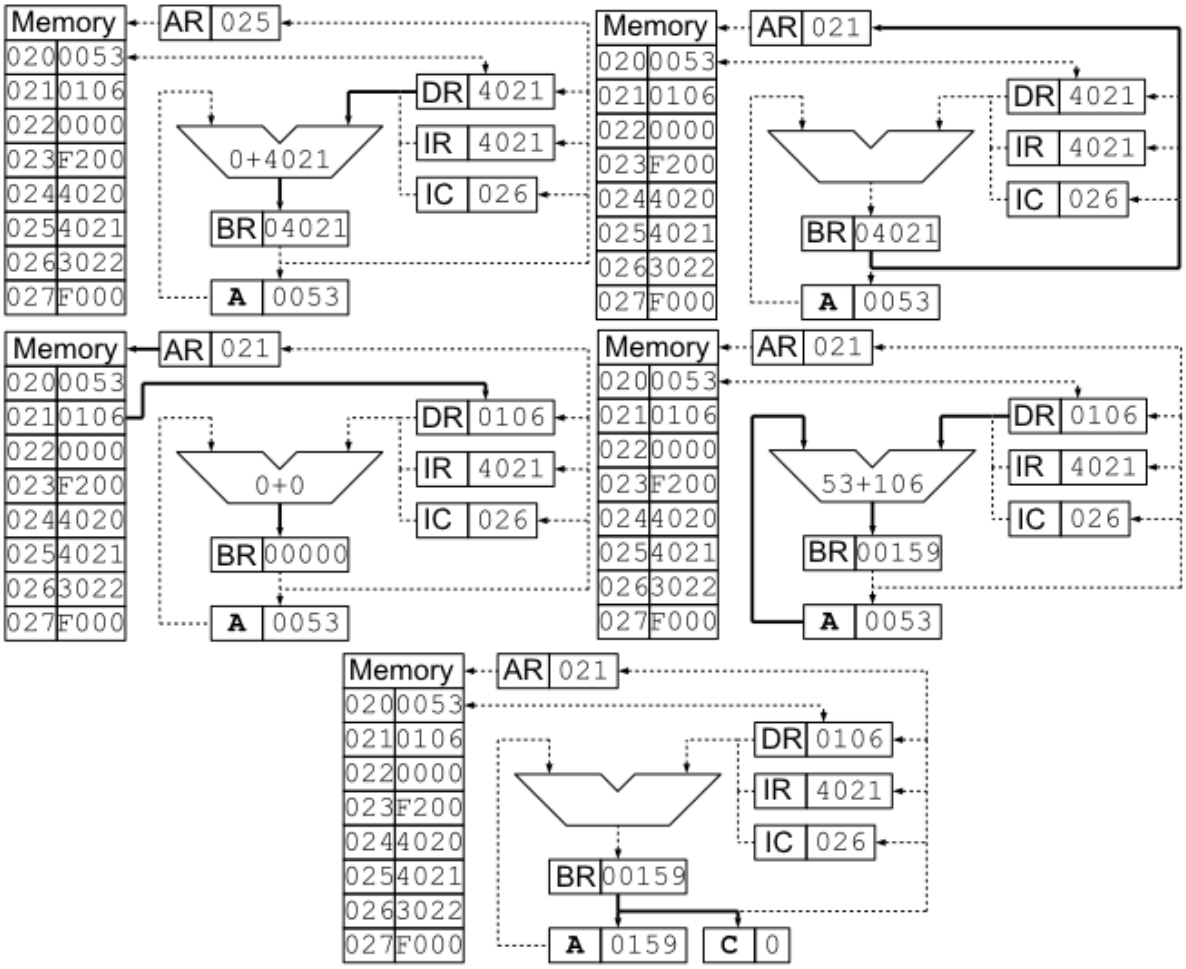



*Figure 10. Cycle "Execution" of the instruction ADD 21*

Control panel operation cycle includes the following actions: input address, write, read, start.

Control panel operation "Input address" writes the content of keyboard registry in the instruction pointer.

Control panel operation "Write" writes the content of keyboard registry in a memory cell, which address is pointed in the instruction pointer. After that, the content of the instruction pointer is incremented by one and goes to the next cell.

Control panel operation "Read" reads into the data register the content of a memory cell , which address is in the instruction pointer. After that, it increments by one the instruction pointer content and goes to the next cell.

Control panel operation "Start" resets the accumulator content, overflow flag, availability of all external devices, restricts all interrupts. If mode "Work" is set, Control panel operation "Start" goes to the instruction, which address is stored in the instruction pointer.

# 2. Laboratory works

## 2.1 Laboratory work 1. Program execution in Basic Computer

### 2.1.1 Overview

Laboratory work 1 is aimed at understanding computer programs written in machine codes and how a computer program can be executed. The work is performed using the Basic Computer model. This simplified computer model represents the basic principles of program execution on a low level. All programs written in modern programming languages are translated into machine codes. The machine codes can be different in different computers, but the basic principles of program execution are the same. This lab work equips students with the knowledge how any program is executed in computers.

After completing this lab work students will know the architecture of Basic Computer; the basic principles of program execution in computer. They will be able to translate the machine code of a program to mnemonic; to understand the content of the program; to upload and execute the program.

### 2.1.2 Lab work task

1. Read your variant.
2. Translate the given machine code in a hexadecimal system into a mnemonic code and fill in Table 7.

*3. Table 7. The answer table for instruction 2 of laboratory work 1*

| ADDRESS | MACHINE CODE | MNEMONIC CODE |
|---|---|---|
| | | |

4. Define the operands **XXXX**, **YYYY**, **ZZZZ** in your variant with any proper numbers from the domain of their definition except 0 (zero). All operands and the results should be represented by signed 16-bit digits. Write the operands in hexadecimal format in Table 8.

*Table 8. The answer table for instruction 3 of laboratory work 1*

| OPERAND | VALUE |
|---|---|
| XXXX | |
| YYYY | |
| ZZZZ | |

5. Execute your program virtually in your head and decide what the memory in Basic Computer would contain after the program execution. Fill in Table 9 below with your ideas.

*Table 9. The answer table for instruction 2 of laboratory work 1*

| ADDRESS | MACHINE CODE |
|---|---|
| | |

6. Enter the program code into Basic Computer Memory
7. Execute your program in Basic Computer Model

8. Fill in Table 10 with the content of the memory after the program execution.

*Table 10. Template for answer table for instruction 7 of laboratory work 1*

| ADDRESS | MACHINE CODE |
|---|---|
| | |

9. Compare the result obtained with the result in point 4. Check yourself – the table contents should be the same. If the table contents are not the same, you should find and correct the mistakes. The mistakes can be in both tables. Be careful.

### 2.1.3 Lab work guidance

To understand how to do the laboratory work we will do the sample variant step by step.

1. ***Read your variant.***

For example, your variant defines a program that is listed in Table 11.

*Table 11. The sample of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 0DD | XXXX | |
| 0DE | 0000 | |
| 0DF | ZZZZ | |
| 0E0 | 0000 | |
| 0E1 | F200 | + |
| 0E2 | 40EA | |
| 0E3 | 10DD | |
| 0E4 | 30DE | |
| 0E5 | F200 | |
| 0E6 | 60DF | |
| 0E7 | 60DE | |
| 0E8 | 30E0 | |
| 0E9 | F000 | |
| 0EA | YYYY | |

This variant contains the list of machine codes of program data and instructions. The program is stored into the memory of the Basic Computer Model. Each code is kept in the address, which you can see in the corresponding line of the column «ADDRESS». The first instruction of your program (the entry point) is marked with symbol «+».

All machine codes are presented as hexadecimal values.

2. ***Translate the given machine code in a hexadecimal system into a mnemonic code and fill in the table below***.

To translate the machine code, you can use the table with the instruction set of Basic Computer. The instruction set for Basic Computer is presented in Appendix A. Note, some memory cells can have a data code instead of instructions. Basic Computer has the von Neumann architecture. It means the data and instructions are stored in the same memory. The program starts from the cell that is identified as **ENTRY POINT** and ends with instruction code F000 (HALT instruction).

The first instruction is **F200**. It is the code of CLA (Table 12) instruction according to the instruction set table.

*Table 12. CLA instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Clear accumulator register* | *CLA* | *F200* | *0 -> A* |

The next instruction is **40EA**. It is the addition (ADD) (Table 13). The content of the memory cell with address 0EA is added to the accumulator register. The result is stored in the accumulator register.

*Table 13. ADD instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Addition* | *ADD 0EA* | *40EA* | *(0EA) + (A) -> A* |

The next instruction is **10DD**. It is conjunction (AND) (Table 14). The conjunction is calculated with the content of the memory cell with address 0DD and the accumulator register. The result is stored in the accumulator register.

*Table 14. AND instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Conjunction* | *AND 0DD* | *10DD* | *(0DD) & (A) -> A* |

The next instruction is **30DE**. It is a move instruction (MOV) (Table 15). The content of the accumulator registers is stored in the memory cell with address 0DE.

*Table 15. MOV instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Move* | *MOV 0DE* | *30DE* | *(A) -> 0DE* |

The next instruction is **F200** again. It clears the accumulator register.

The next instruction is **60DF**. It is subtraction (SUB) (Table 16). The content of the memory cell with address 0DF is subtracted from the accumulator register. The result is stored in the accumulator register.

*Table 16. SUB instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Subtraction* | *SUB 0DF* | *60DF* | *(A) – (0DF) -> (A)* |

The next instruction is **60DE**. It is also subtraction but with the content of the memory cell with address 0DE. The result is also stored in the accumulator register.

The next instruction is **30E0**. It is a move instruction again, but it stores the accumulator register content in the memory cell with address 0E0.

The next instruction is **F000**. It is a halt instruction (HALT) (Table 17). It indicates the end of the program.

*Table 17. HLT instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Halt* | *HLT* | *F000* | *Stop program execution* |

*The* rest memory cells with addresses 0DD, 0DE, 0DF, 0E0, 0EA have the machine codes of program data. The rest memory cells are used as a data source and destination place in the memory of the Basic Computer.

Thus, the correct answer at this point is presented in Table 18.

*Table 18. The answer of instruction 2 of laboratory work 1*

| ADDRESS | MACHINE CODE | MNEMONIC CODE |
|---|---|---|
| 0DD | XXXX | |
| 0DE | 0000 | |
| 0DF | ZZZZ | |
| 0E0 | 0000 | |
| 0E1 | F200 | CLA |
| 0E2 | 40EA | ADD 0EA |
| 0E3 | 10DD | AND 0DD |
| 0E4 | 30DE | MOV 0DE |
| 0E5 | F200 | CLA |
| 0E6 | 60DF | SUB 0DF |
| 0E7 | 60DE | SUB 0DE |
| 0E8 | 30E0 | MOV 0E0 |
| 0E9 | F000 | HLT |
| 0EA | YYYY | |

3. ***Define the operands XXXX, YYYY, ZZZZ in your variant with any proper numbers from the domain of their definition except 0 (zero). All operands and the results should be represented by signed 16-bit digits.***

In this case any 16-bit number would be correct. For example, in this sample it would be A, B, C in the hexadecimal system (Table 19).

*Table 19. The answer of instruction 2 of laboratory work 1*

| OPERAND | VALUE |
|---|---|
| XXXX | 000A |
| YYYY | 000B |
| ZZZZ | 000C |

4. ***Execute your program virtually in your head and decide what the memory in Basic Computer would contain after the program execution.***

Let’s perform all the instructions in our heads. The first instruction clears the accumulator register.

A = 0

The next instruction adds the content of the cell memory with address 0EA to the accumulator register. The cell 0EA contains YYYY operand with value 000B.

A = 000B

The next instruction calculates the conjunction between the accumulator and the value of the cell 0DD with operand XXXX.

A = 000B & 000A = 000A

The next instruction moves the value of the accumulator to the memory cell with address **0DE**. At this step we have the memory content presented in Table 20.

*Table 20. The partial answer of instruction 3 of laboratory work 1*

| ADDRESS | MACHINE CODE |
|---|---|
| 0DD | 000A |
| 0DE | **000A** |
| 0DF | 000C |
| 0E0 | 0000 |
| 0E1 | F200 |
| 0E2 | 40EA |
| 0E3 | 10DD |
| 0E4 | 30DE |
| 0E5 | F200 |
| 0E6 | 60DF |
| 0E7 | 60DE |
| 0E8 | 30E0 |
| 0E9 | F000 |
| 0EA | 000B |

The next instruction clears the accumulator again.

A = 0

The next instruction subtracts the value of cell 0DF from the accumulator.

A = 0 – 000C = FFF4

The next instruction subtracts the value of cell 0DE from the accumulator.

A = FFF4 – 000A = FFEA

The next instruction moves the accumulator value to cell 0E0. After this instruction, the memory is changed as presented in Table 21.

*Table 21. The answer of instruction 3 of laboratory work 1*

| ADDRESS | MACHINE CODE |
|---|---|
| 0DD | 000A |
| 0DE | 000A |
| 0DF | 000C |
| 0E0 | **FFEA** |
| 0E1 | F200 |
| 0E2 | 40EA |
| 0E3 | 10DD |
| 0E4 | 30DE |
| 0E5 | F200 |
| 0E6 | 60DF |
| 0E7 | 60DE |
| 0E8 | 30E0 |
| 0E9 | F000 |
| 0EA | 000B |

The next instruction is a HALT instruction. It is the end of the program. The table above contains the correct answer of this task.

### *5. Enter the program code into the Basic Computer Memory*

To enter a program code into Basic Computer Memory you should open Basic Computer Model. It can be done by using double click on the file **bcomp.jar**. After that, you can see the structure of the processor of Basic Computer presented in figure 11.

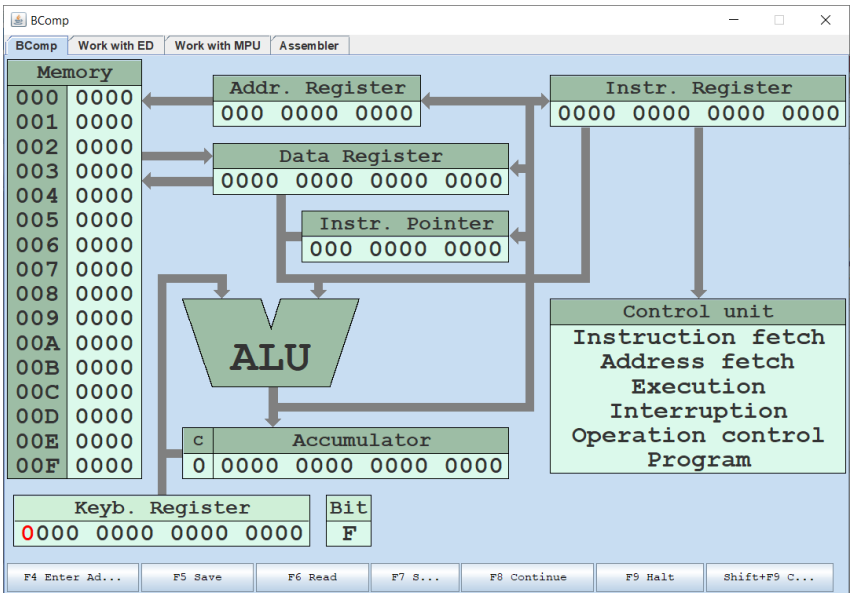



*Figure 11. The main window of the Basic Computer Model*

Let's enter your program into Basic Computer memory. To enter each machine code into the memory you need to define the address using **F4** key and the memory cell value using **F5** key. If you enter the codes sequentially, you should only define the address of the first instruction. After storing the first instruction, the **Instr. Pointer** will be autoincremented and you can enter the second code immediately. For example, let's enter the first instruction with code **F200**.

Use **Up** arrow on your keyboard to change the value of each bit in the **Keyb. Register**. Use arrows **Left** and **Right** to move from one bit to another. Set the address of the first command in the **Keyb. Register** (figure 12). In our sample, the address of the first command is 0E1 (0000 0000 1110 0001 in a binary system). The **Keyb. Register** is used as the intermediate register to input data to the Basic Computer Memory. It looks like a simple user console.

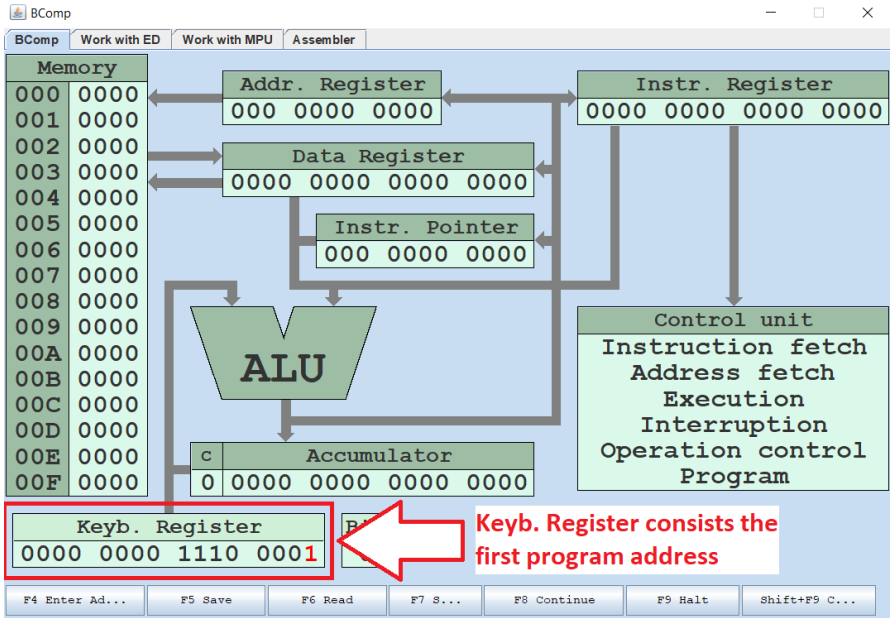



*Figure 12. The new value of the Keyb. Register*

Press **F4** to store this address in the **Instr. Pointer** (figure 13).

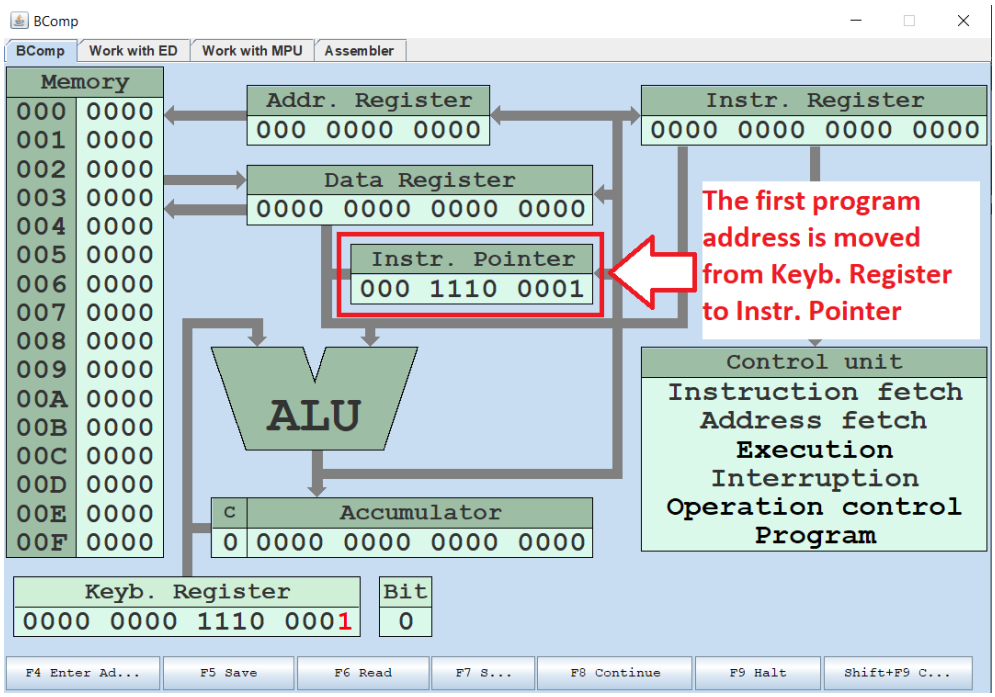



*Figure 13. The new value of the Instr. Pointer*

Enter the value of machine code **F200** (1111 0010 0000 0000 in binary system) into **Keyb. Register** (figure 14).

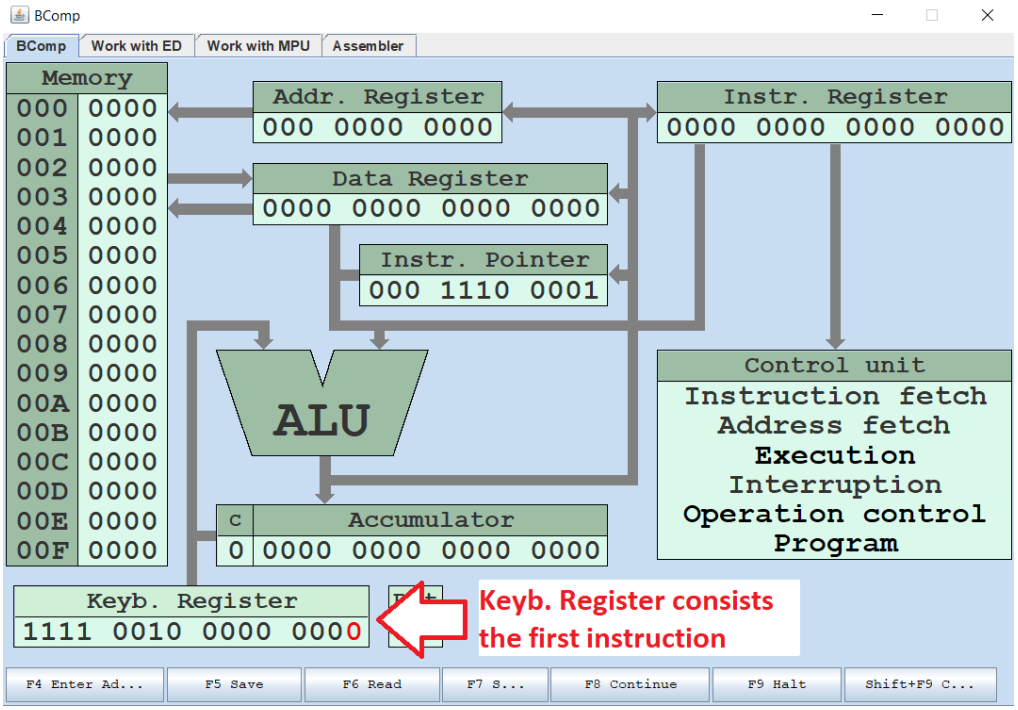



*Figure 14. The first instruction in the Keyb. Register*

Press **F5** to store the value of the **Keyb. Register** in the memory in the cell with 0E1 address (figure 15).

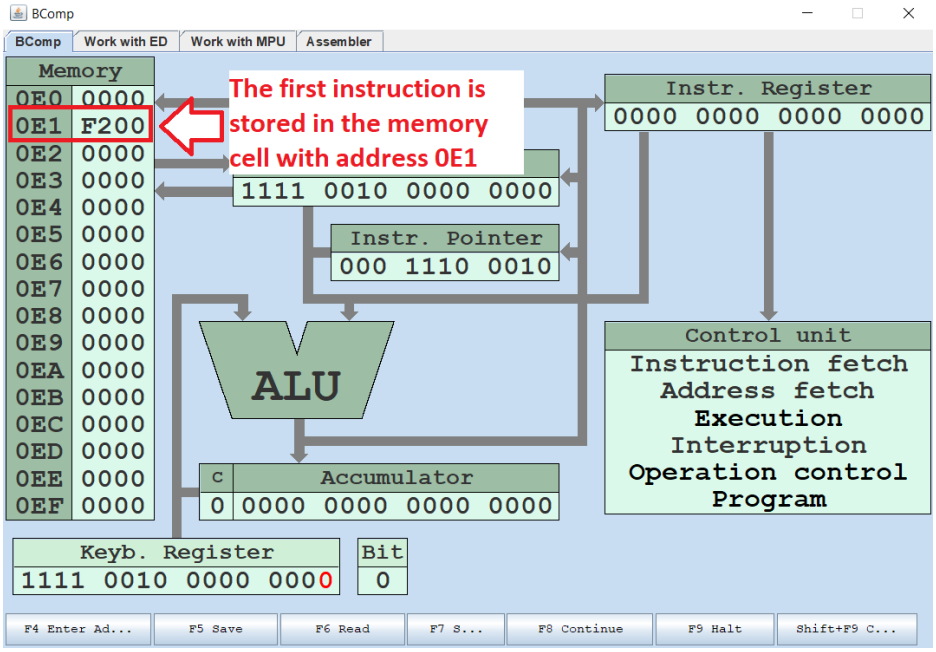



*Figure 15. The first instruction in the memory cell*

Then, enter the rest instructions of your program. For the second instruction and other ones you do not need to enter the instruction address again, as the content of Instr. Pointer register will be autoincremented after pressing **F5**.

Similarly, enter your operands XXXX, YYYY, ZZZZ.

### *6. Execute your program in Basic Computer Model*

Before starting the execution, you need to define the first program address, i.e. the program entry point. Enter the first address of your program in the **Instr. Pointer** using the **Keyb. Register** and **F4** as before.

Then make sure that the mode of the program execution is **Halt**, as presented in figure 16.

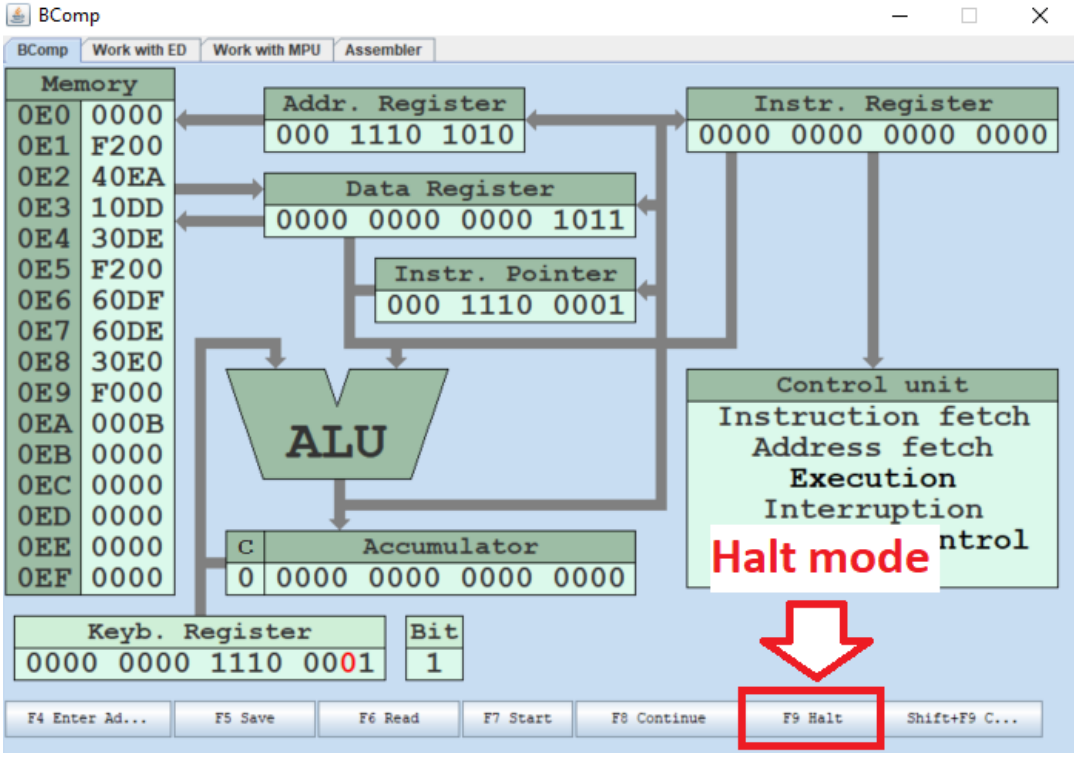



*Figure 16. Halt mode button*

If it does not work, you should press key **F9** to change the mode. The **Halt** mode allows executing a program sequentially – step by step.

Press **F7** to start execution. Then press **F8** to continue execution till the last instruction **F000** with address **0E9**.

If you have done all the instructions correctly, you will see the window presented in figure 17.

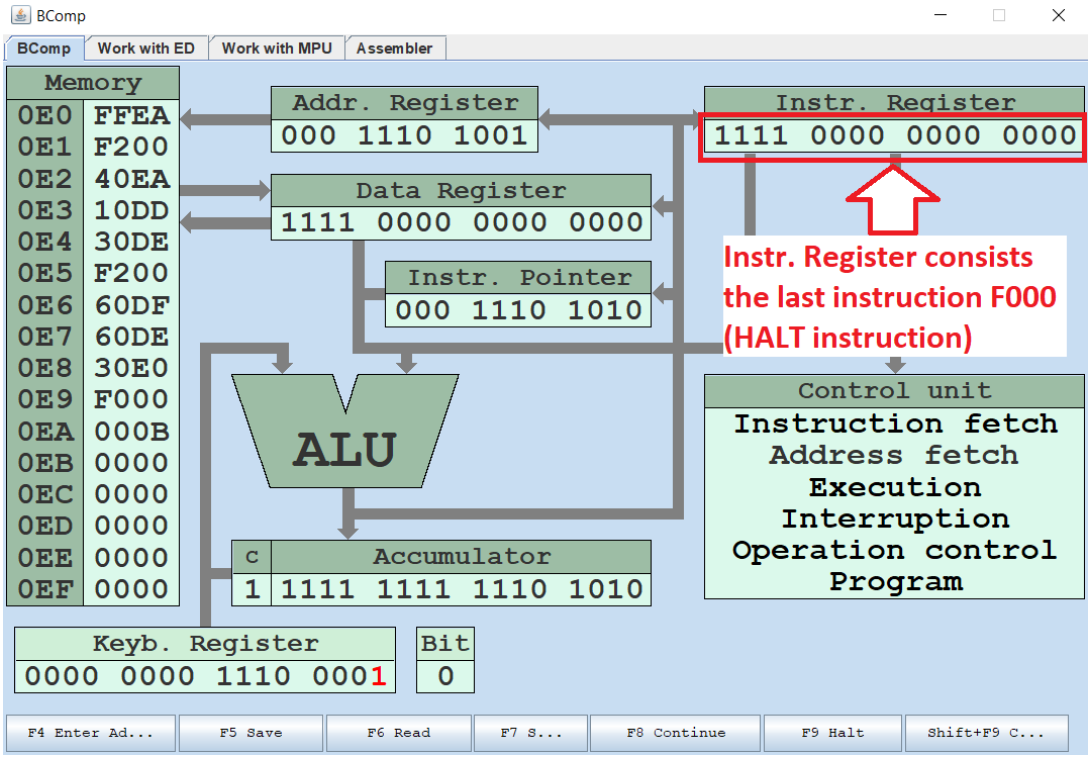



*Figure 17. Halt instruction in the Instr. Register*

The **Inst. Register** has the last instruction **F000**. It is HALT instruction. It is the end of the program.

On the left side of the window, we can see the **Memory** content. We need to use this information in the next step of our task.

### *7. Fill in the table with the content of the memory after the program execution.*

After program execution we can see only a part of memory content on the left side of the Basic Computer window. To see the top part of the memory we can enter the address of the first data cell 0DD and press **F4** to enter the address and then press **F6** to read from this address. After this, you can see the top part of the memory content as presented in Table 18.

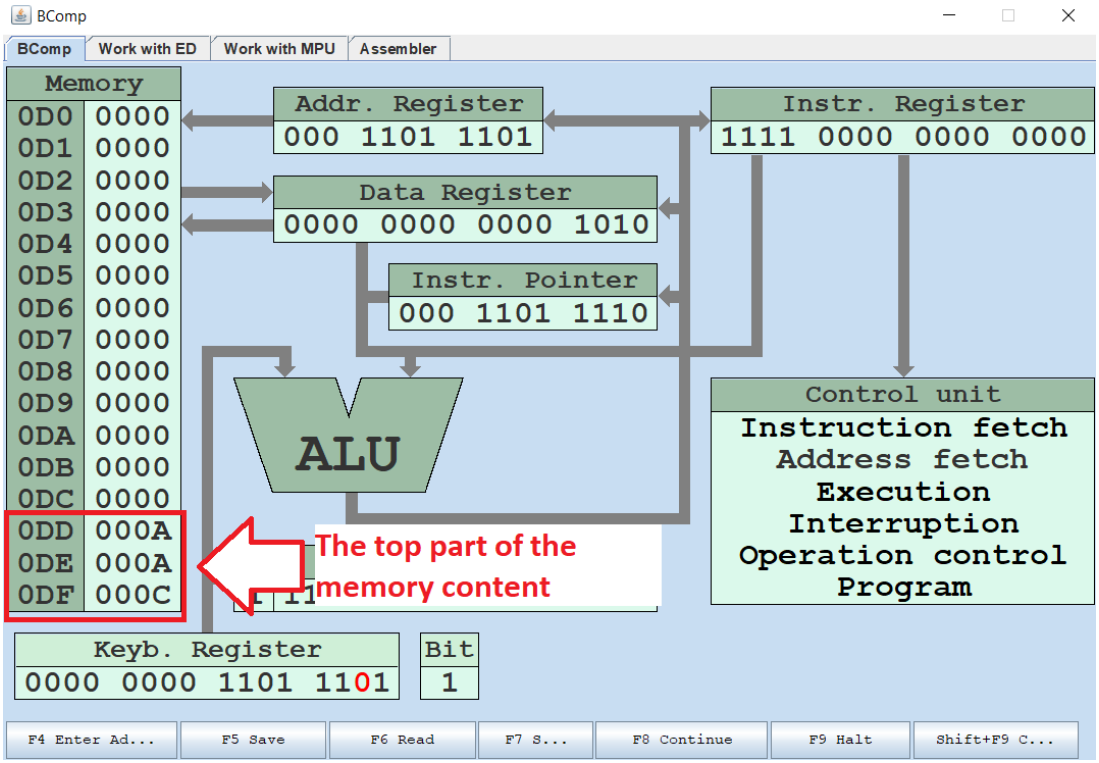



*Figure 18. The top part of the memory content*

Now we can fill in the answer table as presented in Table 22.

*Table 22. The answer of the instruction 7 of laboratory work 1*

| ADDRESS | MACHINE CODE |
|---|---|
| 0DD | 000A |
| 0DE | 000A |
| 0DF | 000C |
| 0E0 | FFEA |
| 0E1 | F200 |
| 0E2 | 40EA |
| 0E3 | 10DD |
| 0E4 | 30DE |
| 0E5 | F200 |
| 0E6 | 60DF |
| 0E7 | 60DE |
| 0E8 | 30E0 |
| 0E9 | F000 |
| 0EA | 000B |

8. ***Compare the result obtained with the result in point 4. Check yourself – the tables content should be the same. If the tables content is not the same, you should find and correct the mistakes. The mistakes can be in both tables. Be careful.***

We can see that both of our tables are the same. It means we have done the task successfully.

### 2.1.4 Lab work variants

Variant 1 of laboratory work 1 is presented in Table 23.

*Table 23. Variant 1 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 0B3 | XXXX | |
| 0B4 | YYYY | |
| 0B5 | ZZZZ | |
| 0B6 | 0000 | |
| 0B7 | F200 | + |
| 0B8 | 40B5 | |
| 0B9 | 10B3 | |
| 0BA | 30C0 | |
| 0BB | F200 | |
| 0BC | 60B4 | |
| 0BD | 60C0 | |
| 0BE | 30B6 | |
| 0BF | F000 | |
| 0C0 | 0000 | |

Variant 2 of laboratory work 1 is presented in Table 24.

*Table 24. Variant 2 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 038 | 0000 | |
| 039 | 0000 | |
| 03A | YYYY | |
| 03B | F200 | + |
| 03C | 4044 | |
| 03D | 4045 | |
| 03E | 3039 | |
| 03F | F200 | |
| 040 | 403A | |
| 041 | 1039 | |
| 042 | 3038 | |
| 043 | F000 | |
| 044 | XXXX | |
| 045 | ZZZZ | |

Variant 3 of laboratory work 1 is presented in Table 25.

*Table 25. Variant 3 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 0A0 | F200 | + |
| 0A1 | 40AC | |
| 0A2 | 10AB | |
| 0A3 | 30AA | |
| 0A4 | F200 | |
| 0A5 | 60AD | |
| 0A6 | 40AA | |
| 0A7 | 30A9 | |
| 0A8 | F000 | |
| 0A9 | 0000 | |
| 0AA | 0000 | |
| 0AB | YYYY | |
| 0AC | ZZZZ | |
| 0AD | XXXX | |

Variant 4 of laboratory work 1 is presented in Table 26.

*Table 26. Variant 4 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 042 | 0000 | |
| 043 | F200 | + |
| 044 | 404E | |
| 045 | 104F | |
| 046 | 3042 | |
| 047 | F200 | |
| 048 | 404C | |
| 049 | 6042 | |
| 04A | 304D | |
| 04B | F000 | |
| 04C | ZZZZ | |
| 04D | 0000 | |
| 04E | YYYY | |
| 04F | XXXX | |

Variant 5 of laboratory work 1 is presented in Table 27.

*Table 27. Variant 5 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 09F | 0000 | |
| 0A0 | F200 | + |
| 0A1 | 40A9 | |
| 0A2 | 10AC | |
| 0A3 | 309F | |
| 0A4 | F200 | |
| 0A5 | 40AB | |
| 0A6 | 409F | |
| 0A7 | 30AA | |
| 0A8 | F000 | |
| 0A9 | YYYY | |
| 0AA | 0000 | |
| 0AB | ZZZZ | |
| 0AC | XXXX | |

Variant 6 of laboratory work 1 is presented in Table 28.

*Table 28. Variant 6 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 041 | 0000 | |
| 042 | ZZZZ | |
| 043 | 0000 | |
| 044 | XXXX | |
| 045 | F200 | + |
| 046 | 6044 | |
| 047 | 4042 | |
| 048 | 3043 | |
| 049 | F200 | |
| 04A | 404E | |
| 04B | 1043 | |
| 04C | 3041 | |
| 04D | F000 | |
| 04E | YYYY | |

Variant 7 of  laboratory work 1 is presented in Table 29.

*Table 29. Variant 7 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 0A9 | 0000 | |
| 0AA | ZZZZ | |
| 0AB | F200 | + |
| 0AC | 60B5 | |
| 0AD | 60AA | |
| 0AE | 30A9 | |
| 0AF | F200 | |
| 0B0 | 40B4 | |
| 0B1 | 10A9 | |
| 0B2 | 30B6 | |
| 0B3 | F000 | |
| 0B4 | XXXX | |
| 0B5 | YYYY | |
| 0B6 | 0000 | |

Variant 8 of  laboratory work 1 is presented in Table 30.

*Table 30. Variant 8 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 04B | F200 | + |
| 04C | 4056 | |
| 04D | 4055 | |
| 04E | 3054 | |
| 04F | F200 | |
| 050 | 4057 | |
| 051 | 1054 | |
| 052 | 3058 | |
| 053 | F000 | |
| 054 | 0000 | |
| 055 | YYYY | |
| 056 | XXXX | |
| 057 | ZZZZ | |
| 058 | 0000 | |

Variant 9 of laboratory work 1 is presented in Table 31.

*Table 31. Variant 9 of laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 0A8 | F200 | + |
| 0A9 | 60B4 | |
| 0AA | 40B3 | |
| 0AB | 30B1 | |
| 0AC | F200 | |
| 0AD | 40B5 | |
| 0AE | 10B1 | |
| 0AF | 30B2 | |
| 0B0 | F000 | |
| 0B1 | 0000 | |
| 0B2 | 0000 | |
| 0B3 | XXXX | |
| 0B4 | ZZZZ | |
| 0B5 | YYYY | |

Variant 10 of  laboratory work 1 is presented in Table 32.

*Table 32. Variant 10 of  laboratory work 1*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 049 | ZZZZ | |
| 04A | 0000 | |
| 04B | YYYY | |
| 04C | 0000 | |
| 04D | F200 | + |
| 04E | 6049 | |
| 04F | 4056 | |
| 050 | 304A | |
| 051 | F200 | |
| 052 | 404B | |
| 053 | 104A | |
| 054 | 304C | |
| 055 | F000 | |
| 056 | XXXX | |

## 2.2 Laboratory work 2. Low level instruction execution

### 2.2.1 Overview

This lab work is aimed at understanding the basic principles of instruction execution on a low level. The lab gives basic skills of tracing program and knowledge about low-level steps of each instruction, known as machine cycles.

After completing this lab work, students will know the basic principles of instruction execution on a low level; know the concept of machine cycles. They will be able to decompose instruction execution into machine cycles; to trace a program.

### 2.2.2 Lab work task

1. Read your variant.
2. Translate the given machine code in a hexadecimal system into a mnemonic code and fill Table 33.

*Table 33. Template for answer table for instruction 2 of laboratory work 2*

| ADDRESS | MACHINE CODE | MNEMONIC CODE |
|---|---|---|
| | | |

3. Execute your program virtually in your head and fill in trace Table 34 with your ideas. In the trace table, you need to define the internal register content after each instruction execution, the address and the value of the memory cell that have been changed.

*Table 34. The answer table for instruction 3 of laboratory work 2*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| | | | | | | | | | |

4. Enter the program code into Basic Computer Memory
5. Execute your program in the Basic Computer Model step by step and fill in trace table 35. You should write the content of each register after the execution of each instruction.

*Table 35. The answer table for instruction 5 of laboratory work 2*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| | | | | | | | | | |

6. Compare the obtained result with the result in point 3. Check yourself – the tables content should be the same. If the tables content is not the same, you should find and correct the mistakes. The mistakes can be in both tables. Be careful.

### 2.2.3 Lab work guidance

To understand how to complete the laboratory work, we will do the sample variant step by step.

1. ***Read your variant***

For example, the variant defines a program that is listed in Table 36.

*Table 36 The* sample *variant of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 091 | 0001 | |
| 092 | 001F | |
| 093 | 0347 | |
| 094 | 0000 | |
| 095 | F200 | + |
| 096 | 6092 | |
| 097 | 4091 | |
| 098 | 3094 | |
| 099 | F200 | |
| 09A | 4093 | |
| 09B | 1094 | |
| 09C | 309E | |
| 09D | F000 | |
| 09E | 0000 | |

This sample contains the list of machine codes for program data and instructions. The program is stored in the memory of the Basic Computer Model. Each code is kept in the address, which you can see in the corresponding line of the column «ADDRESS». The first instruction of your program is marked with symbol «+».

All machine codes are presented as a hexadecimal value.

2. ***Translate the given machine code in hexadecimal system into a mnemonic code and fill in the answer table.***

To translate the given machine code, you can use the table with the instruction set of Basic Computer. The Instruction set of the Basic Computer is presented in Appendix A. Note, some memory cells can have a data code instead of instructions. Basic Computer has the von Neumann architecture. It means the data and instructions are stored in the same memory. The program starts from the cell that is identified as **ENTRY POINT** and ends with instruction code F000 (HALT instruction).

The first instruction is **F200**. It is a code of CLA instruction (Table 37) according to the instruction set table.

*Table 37 CLA instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Clear accumulator register* | *CLA* | *F200* | *0 -> A* |

The next instruction is **6092**. It is subtraction (Table 38). The content of the memory cell with address 092 is subtracted from the accumulator register. The result is stored in the accumulator register.

*Table 38 SUB instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Subtraction* | *SUB 092* | *6092* | *(A) – (092) -> (A)* |

The next instruction is **4091**. It is an addition (Table 39). The content of the memory cell with address 091 is added to the accumulator register. The result is stored in the accumulator register.

*Table 39 ADD instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Addition* | *ADD 091* | *4091* | *(091) + (A) -> A* |

The next instruction is **3094**. It is a move instruction (Table 40). The content of the accumulator register is stored in the memory cell with address 094.

*Table 40. MOV instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Move* | *MOV 094* | *3094* | *(A) -> 094* |

The next instruction is **F200** again. It clears the accumulator register.

The next instruction is **4093**. It is an addition. The content of the memory cell with address 093 is added to the accumulator register. The result is stored in the accumulator register.

The next instruction is **1094**. It is conjunction (Table 41). The conjunction is calculated with the content of the memory cell with address 094 and the accumulator register. The result is stored in the accumulator register.

*Table 41. AND instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Conjunction* | *AND 094* | *1094* | *(094) & (A) -> A* |

The next instruction is **309E**. It is a move instruction. The content of the accumulator register is stored in the memory cell with address 09E.

The next instruction is **F000**. It is halt instruction (Table 42). It indicates the end of the program.

*Table 42. HLT instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Halt* | *HLT* | *F000* | *Stop program execution* |

Thus, the correct answer at this point is presented in Table 43.

*Table 43. The answer for the instruction 2 of the* sample *variant*

| ADDRESS | MACHINE CODE | MNEMONIC CODE |
|---|---|---|
| 091 | 0001 | |
| 092 | 001F | |
| 093 | 0347 | |
| 094 | 0000 | |
| 095 | F200 | CLA |
| 096 | 6092 | SUB 092 |
| 097 | 4091 | ADD 091 |
| 098 | 3094 | MOV 094 |
| 099 | F200 | CLA |
| 09A | 4093 | ADD 093 |
| 09B | 1094 | AND 094 |
| 09C | 309E | MOV 09E |
| 09D | F000 | HLT |
| 09E | 0000 | |

***3. Execute your program virtually in your head and fill in the trace table with your ideas. In the trace table you need to define the internal register content after each instruction execution, the address and the value of the memory cell that has been changed.***

The Basic Computer model has five internal registers: the address register (AR), the data register (DR), the instruction register (IR), the instruction pointer (IP) and the accumulator register (A). The accumulator register stores intermediate results while instruction executing.

The **address register** is an 11-bit register that stores the address of data or instruction to access. The **data register** is a 16-bit register to store data during the execution of an instruction that moves data from or to the memory. **The instruction register** is a 16-bit register that stores the code of the current instruction. The **instruction pointer** is an 11-bit register that stores the address of the next instruction. The **accumulator register** is a 16-bit register that stores the result of ALU operation.

Each instruction is executed in Basic Computer in several steps. The number of steps is also known as machine cycles, which depend on the type of an instruction. In this lab work we have instructions only with direct addressing, for example ADD instruction, and without addressing, for example CLA instruction.

In many cases the instruction execution is divided into three stages. They are the instruction fetch, the address fetch and the execution. The address fetch stage is presented only in the instruction with indirect addressing. The additional information about stages of instruction execution you can see in chapter 1.

Let's start from the first instruction. The first instruction is **F200**. To perform this instruction Basic Computer does several machine cycles:

1) The content of the instruction pointer 095 goes to the address register. 095 is the address of the first instruction (figure 19). It is necessary for the instruction pointer to have this value before the program execution.

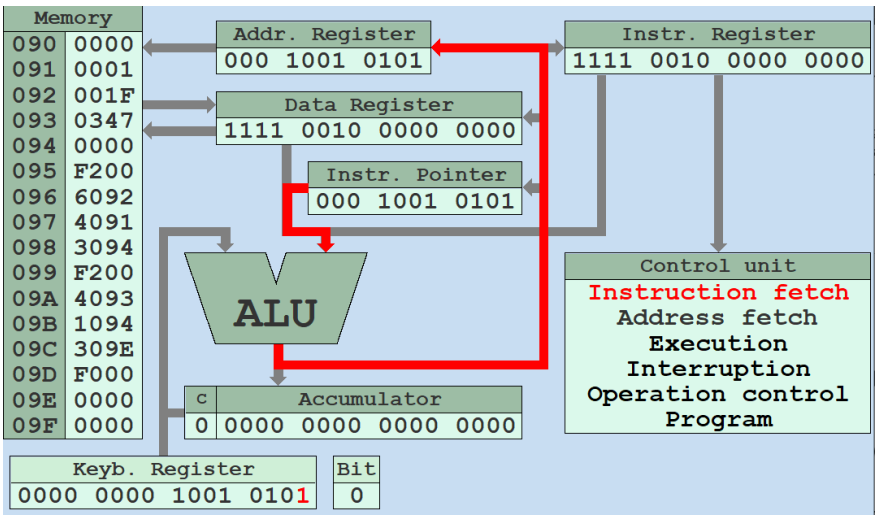



*Figure 19. The first cycle of the F200 instruction*

2) The memory cell value F200 with address 095 that is stored in the address register moves to the data register (figure 20).

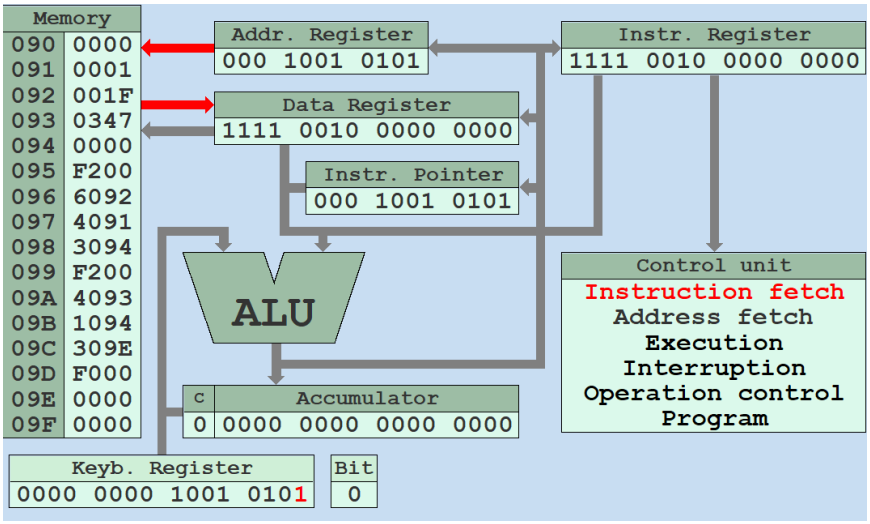



*Figure 20. The second cycle of the F200 instruction*

3) The instruction pointer is incremented by 1 and becomes 096 (figure 21).

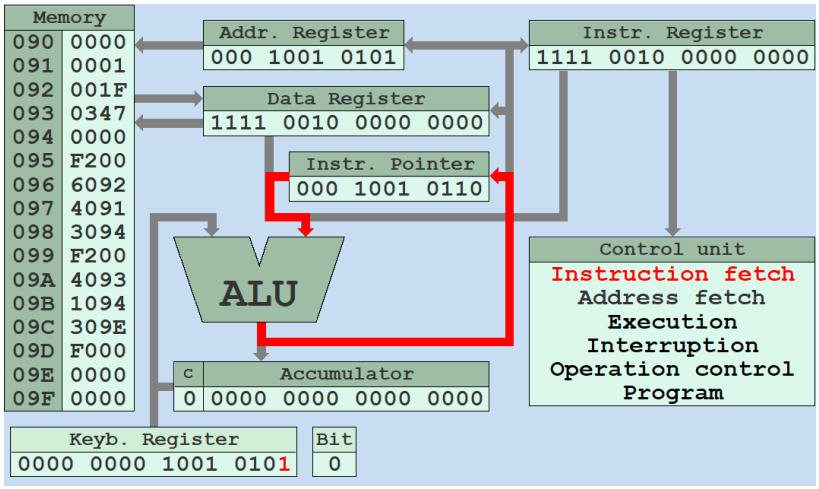



*Figure 21. The third cycle of the F200 instruction*

4) The content F000 of the data register is moved to the instruction register and is decoded (figure 22).

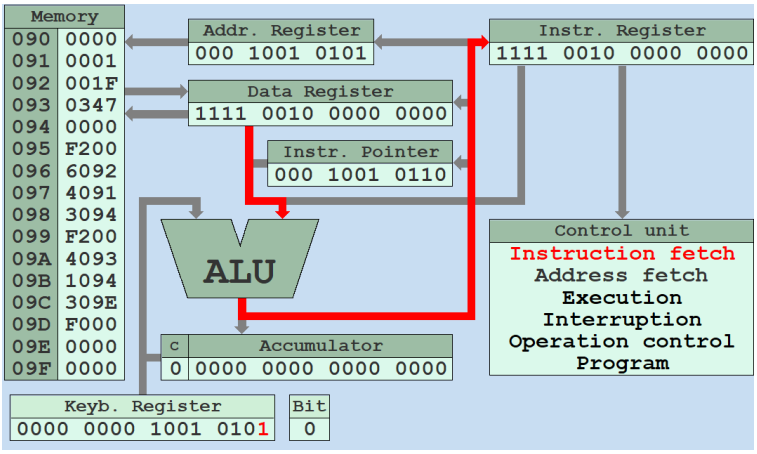



*Figure 22. The fourth cycle of the F200 instruction*

5) The accumulator register is cleared (figure 23).

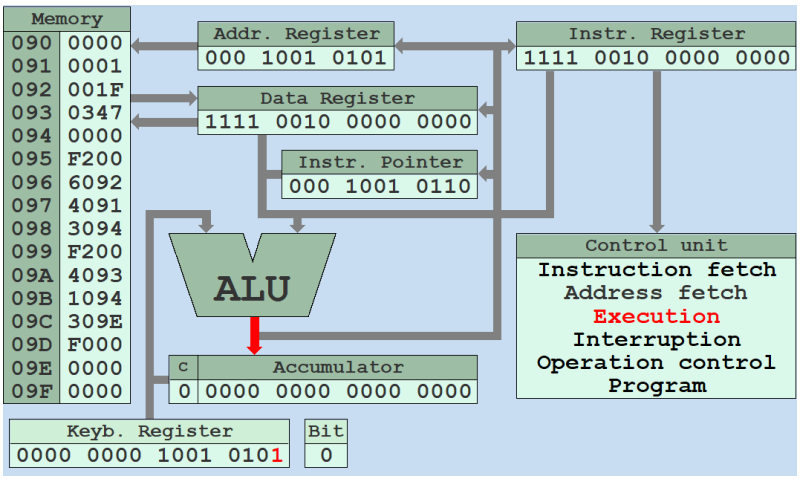



*Figure 23. The fifth cycle of the F200 instruction*

The instruction fetch stage consists of the machine cycles from 1 to 4. The machine cycle 5 is the execution stage of this instruction. Thus, after the instruction execution the content of the internal registers will be as presented in Table 44.

*Table 44. The register content after the F200 instruction execution*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| 095 | F200 | 095 | F200 | F200 | 096 | 0000 | 0 | | |

The next instruction is **6092**. To perform this instruction Basic Computer does several machine cycles:

1) The content of the instruction pointer 096 goes to the address register (figure 24).

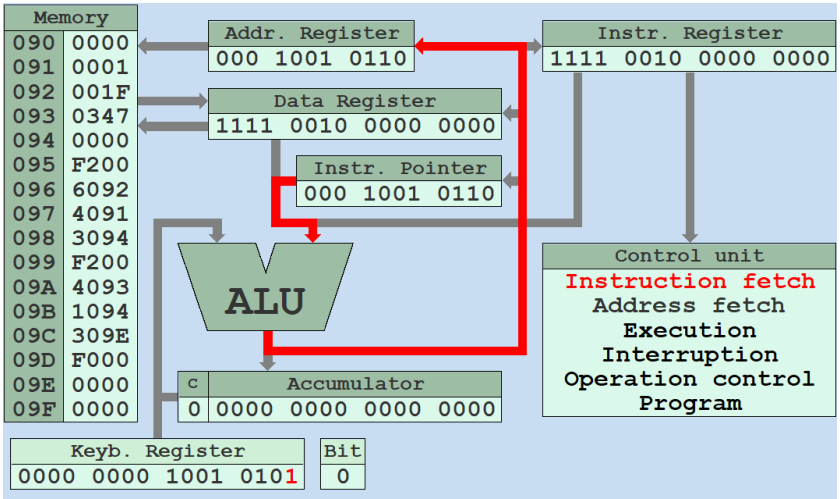



*Figure 24. The first cycle of the 6092 instruction*

2) The memory cell value 6092 with address 096 that is stored in the address register moves to the data register (figure 25).

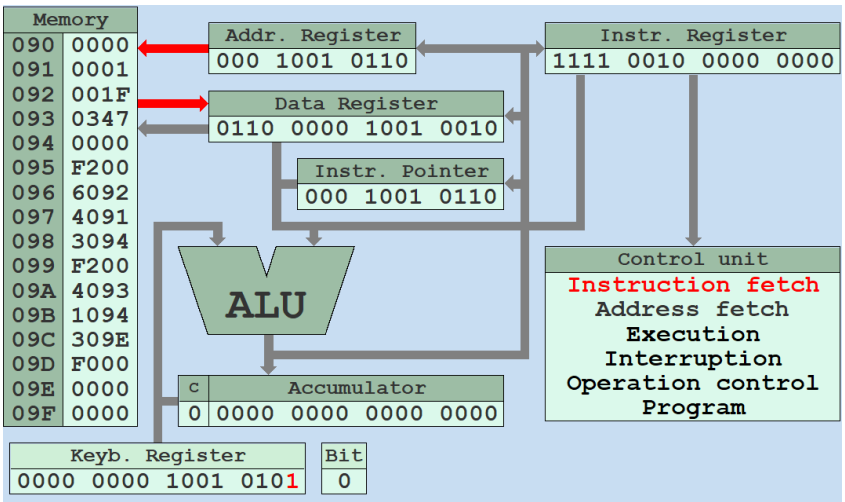



*Figure 25. The second cycle of the 6092 instruction*

The instruction pointer is incremented by 1 and becomes 097 (figure 26).

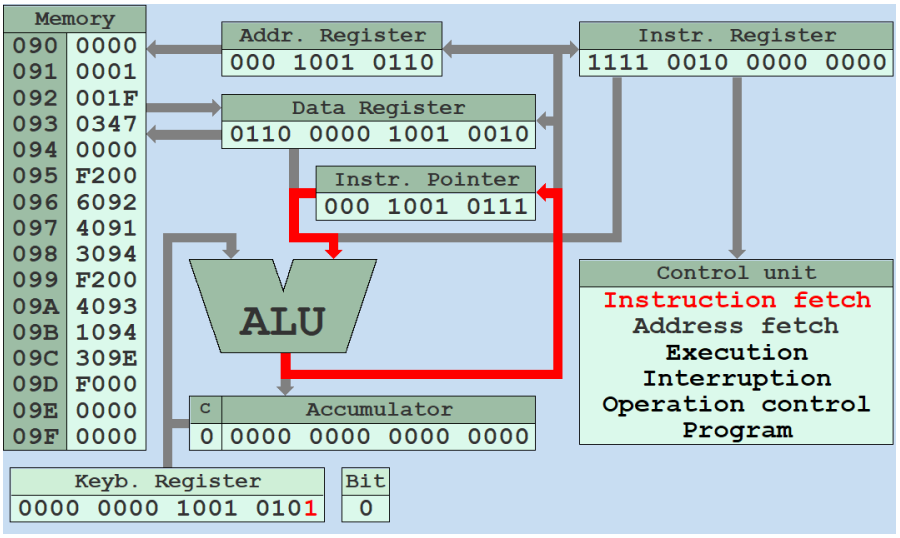



*Figure 26. The third cycle of the 6092 instruction*

3) The content 6092 of the data register is moved to the instruction register and decoded (figure 27).

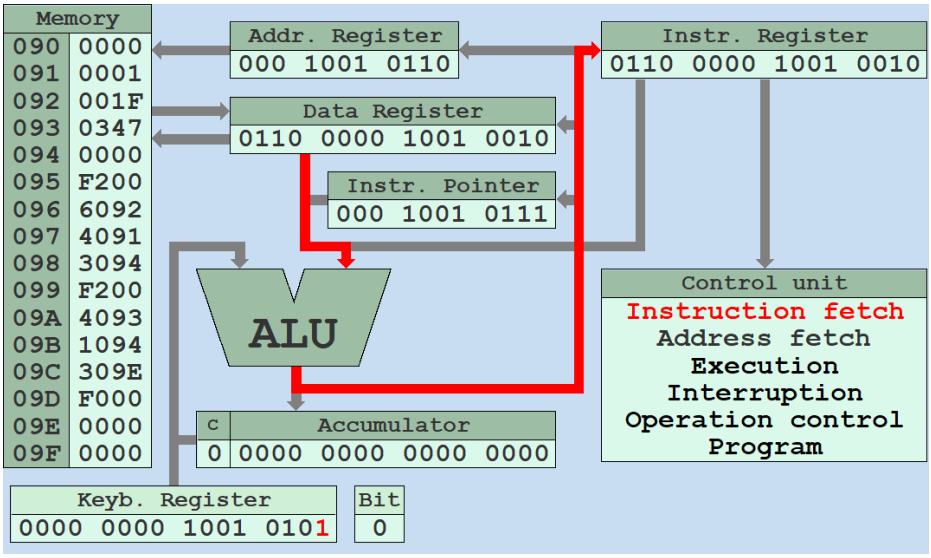



*Figure 27. The fourth cycle of the 6092 instruction*

4) The address part of the data register is moved to the address register (figure 28).

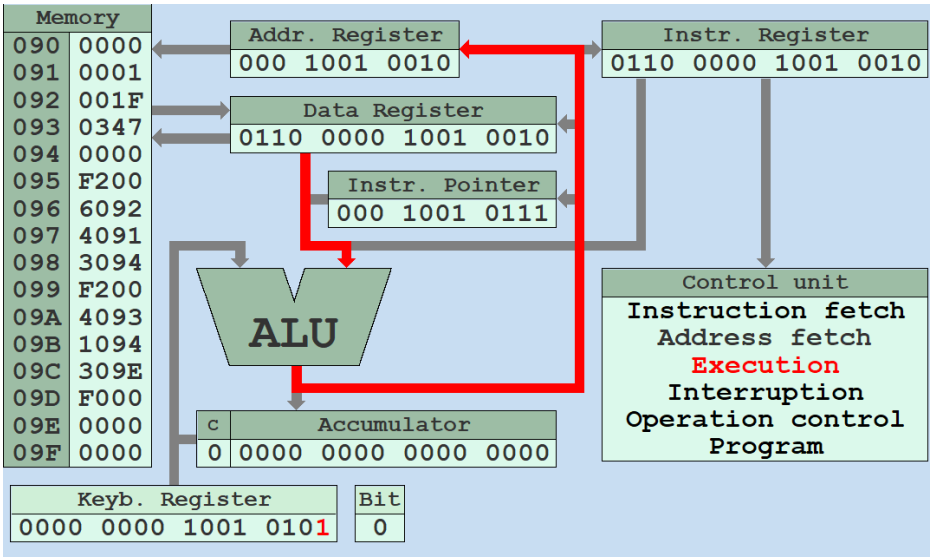



*Figure 28. The fifth cycle of the 6092 instruction*

5) The content of the memory cell with address 092 is moved to the data register (figure 29). 092 is the operand address for 6092 instruction (SUB 092).

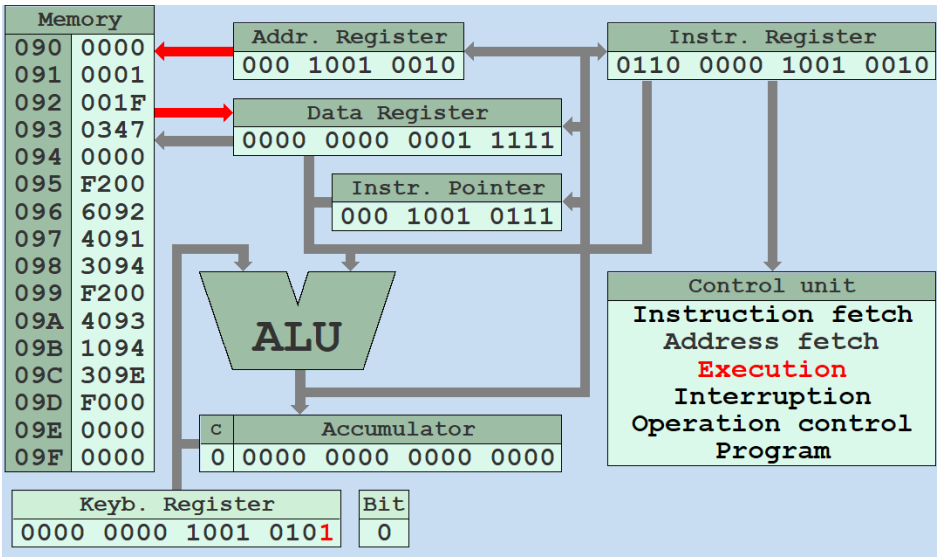



*Figure 29. The sixth cycle of the 6092 instruction*

6) The operand from the data register is subtracted from the accumulator register (figure 30). The result is stored in the accumulator register.

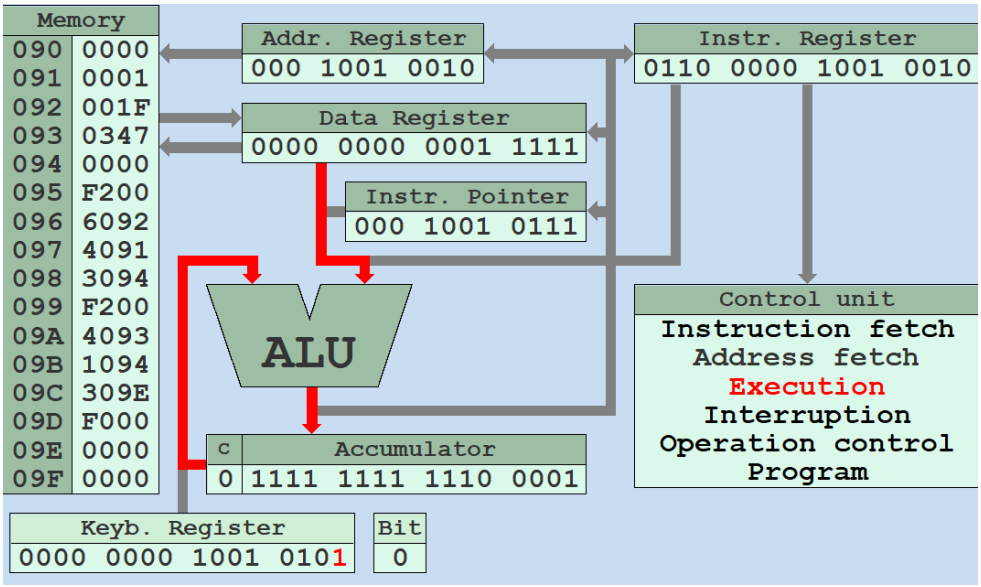



*Figure 30. The seventh cycle of the 6092 instruction*

The instruction fetch stage consists of the machine cycles from 1 to 4. The machine cycles from 5 to 7 are the execution stage of this instruction. Thus, after the instruction execution the content of the internal registers will be the same as presented in Table 45.

*Table 45. The register content after the 6092 instruction execution*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| 096 | 6092 | 092 | 001F | 6092 | 097 | FFE1 | 0 | | |

The next instruction is **4091**. To perform this instruction Basic Computer does several machine cycles:

1) The content of the instruction pointer 097 goes to the address register.
2) The memory cell value 4091 with address 097 that is stored in the address register moves to the data register.
3) The instruction pointer is incremented by 1and becomes 098.
4) The content 4091 of the data register is moved to the instruction register and decoded.
5) The address part 091 of the data register is moved to the address register.
6) The content of the memory cell with address 091 is moved to the data register. 091 is the operand address for the 4091 instruction (ADD 091).
7) The operand from the data register is added to the accumulator register. The result is stored in the accumulator register.

Thus, after the instruction execution the content of the internal registers will be the same as presented in Table 46.

*Table 46. The register content after the 4091 instruction execution*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| 097 | 4091 | 091 | 0001 | 4091 | 098 | FFE2 | 0 | | |

The next instruction is **3094**. To perform this instruction Basic Computer does several machine cycles:

1) The content of the instruction pointer 098 goes to the address register.
2) The memory cell value 3094 with address 098 that is stored in the address register moves to the data register.
3) The instruction pointer is incremented by 1 and becomes 099.
4) The content 3094 of the data register is moved to the instruction register and is decoded.
5) The address part 094 of the data register is moved to the address register.
6) The content of the accumulator register is moved to the data register.
7) The content of the data register is stored in the memory cell with address 094.

Thus, after the instruction execution the content of the internal registers will be the same as presented in Table 47.

*Table 47 The register content after the 3094 instruction execution*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| 098 | 3094 | 094 | FFE2 | 3094 | 099 | FFE2 | 0 | 094 | FFE2 |

Similarly, we need to decompose other instructions. The result trace table will be the same as presented in Table 48.

*Table 48 The registers content during the program execution*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| 095 | F200 | 095 | F200 | F200 | 096 | 0000 | 0 | | |
| 096 | 6092 | 092 | 001F | 6092 | 097 | FFE1 | 0 | | |
| 097 | 4091 | 091 | 0001 | 4091 | 098 | FFE2 | 0 | | |
| 098 | 3094 | 094 | FFE2 | 3094 | 099 | FFE2 | 0 | 094 | FFE2 |
| 099 | F200 | 099 | F200 | F200 | 09A | 0000 | 0 | | |
| 09A | 4093 | 093 | 0347 | 4093 | 09B | 0347 | 0 | | |
| 09B | 1094 | 094 | FFE2 | 1094 | 09C | 0342 | 0 | | |
| 09C | 309E | 09E | 0342 | 309E | 09D | 0342 | 0 | 09E | 0342 |
| 09D | F000 | 09D | F000 | F000 | 09E | 0342 | 0 | | |

### *4. Enter the program code into Basic Computer Memory*

To enter a program code into the Basic Computer Memory, you should open Basic Computer Model. It can be done by using double click on the file **bcomp.jar**. After that you can see the structure of the processor of Basic Computer presented in figure 31.

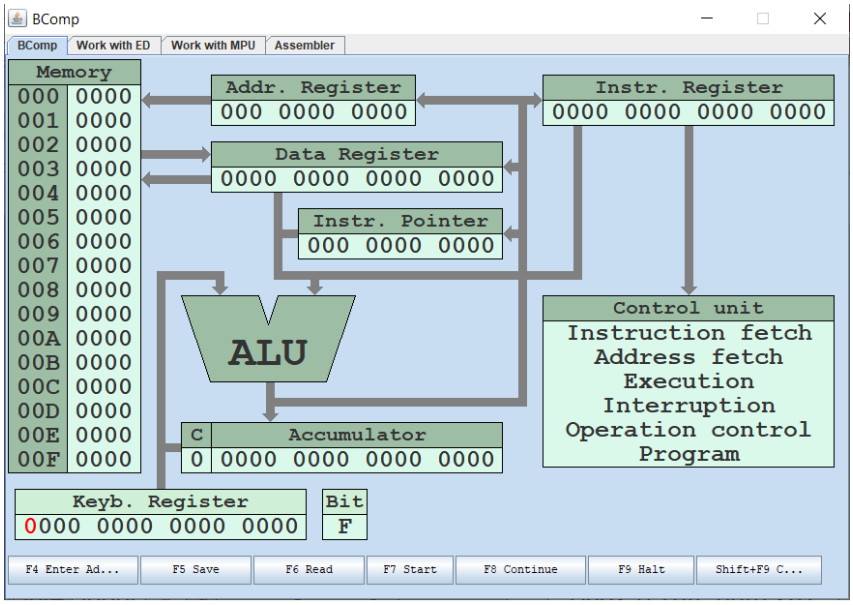



*Figure 31. The main window of Basic Computer Model*

Let's enter your program into the Basic Computer memory. To enter each machine code into the memory, you need to define the address using **F4** key and memory cell value using **F5** key. If you enter the codes sequentially, you should define only the address of the first code. After storing, the first code the **Instr. Pointer** will be autoincremented, and you can enter the second code immediately. For example, let's enter the first code **0001** in address **091**.

Use **Up** arrow on your keyboard to change the value of each bit in the **Keyb. Register**. Use arrows **Left** and **Right** to move from one bit to another. Set the address of the first code in the **Keyb. Register** (figure 32).

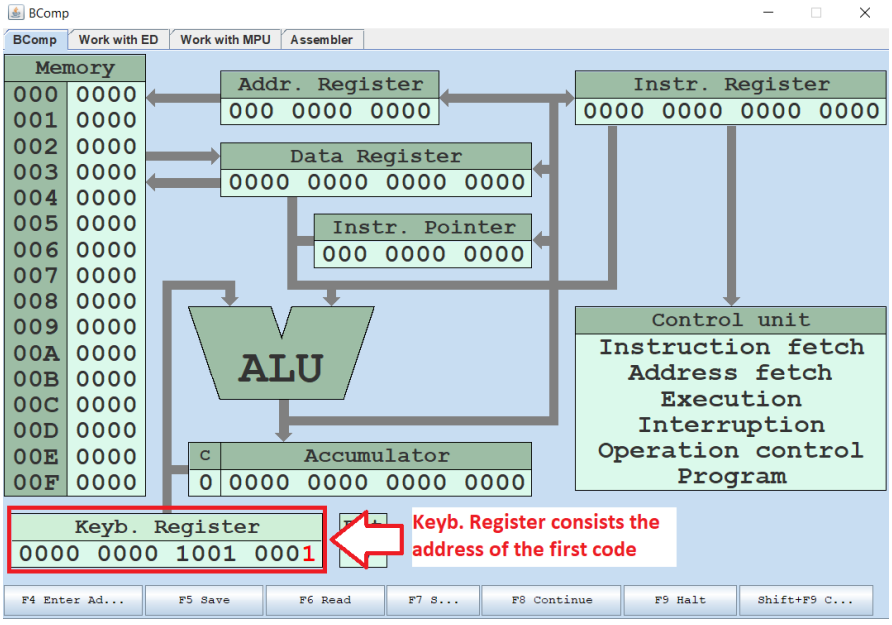



*Figure 32. The content of the Keyb. Register*

In this sample, the address of the first code is 091 (0000 0000 1001 0001 in binary system). The Keyb. Register is used as the intermediate register to input data to the Basic Computer Memory. It looks like a simple user console.

Press **F4** to store this address in the **Instr. Pointer** (figure 33).

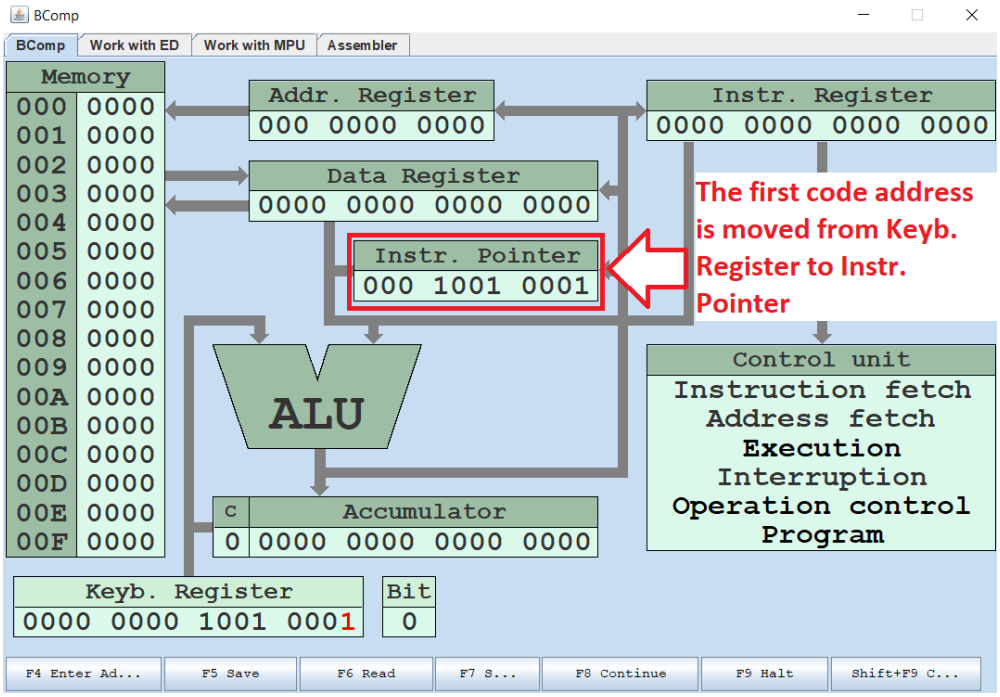



*Figure 33. The first code address in the Instr. Pointer*

Enter the value of machine code **0001** (0000 0000 0000 0001 in binary system) into the **Keyb. Register** (figure 34).

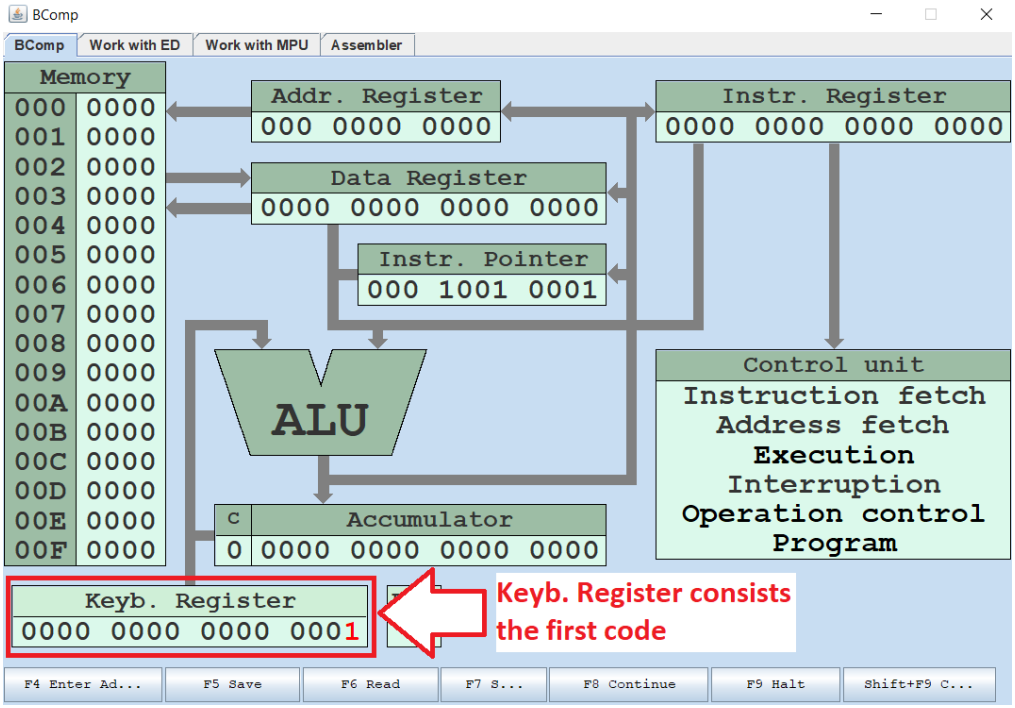



*Figure 34. The first code in Keyb. Register*

Press **F5** to store the value of the **Keyb. Register** in the memory cell with 091 address (figure 35).

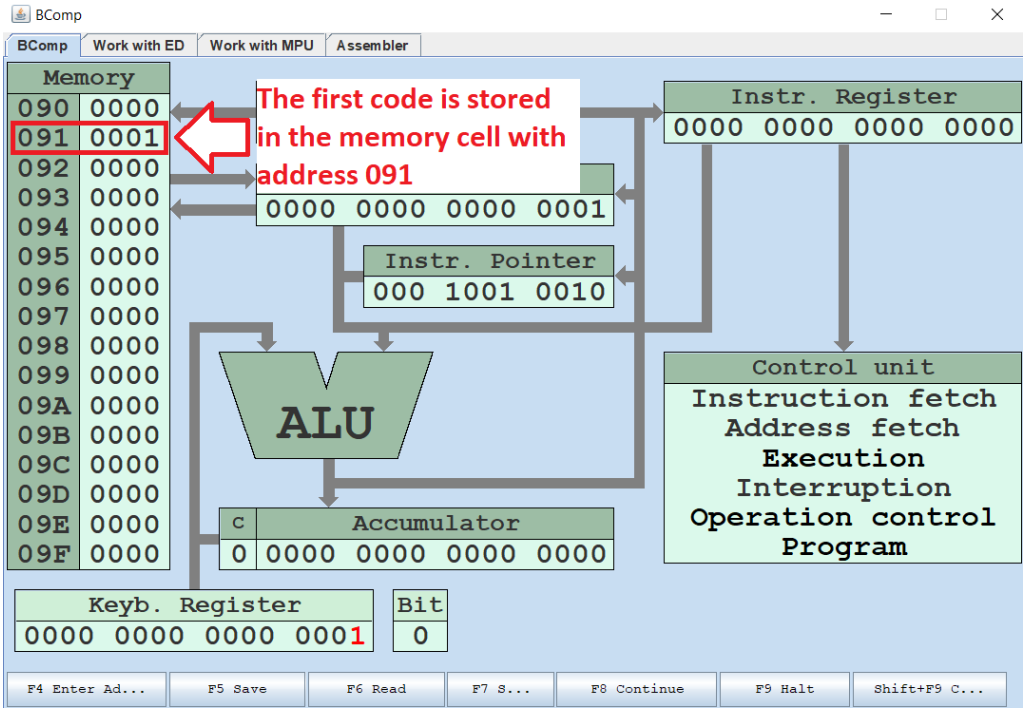



*Figure 35. The first code in the memory*

Then, enter the rest codes of your program. For the second code and other ones you do not need to enter the code address again, as the content of the Instr. Pointer register will be autoincremented after pressing **F5**.

5. ***Execute your program in Basic Computer Model step by step and fill in the trace table below. You should write the content of each register after the execution of each instruction.***

Before starting the execution, you need to define the first program address, i.e. the program entry point. Enter the first address **095** of your program in the **Instr. Pointer** using Keyb. Register and **F4** as before.

Then make sure that the mode of program execution is **Halt** (figure 36).

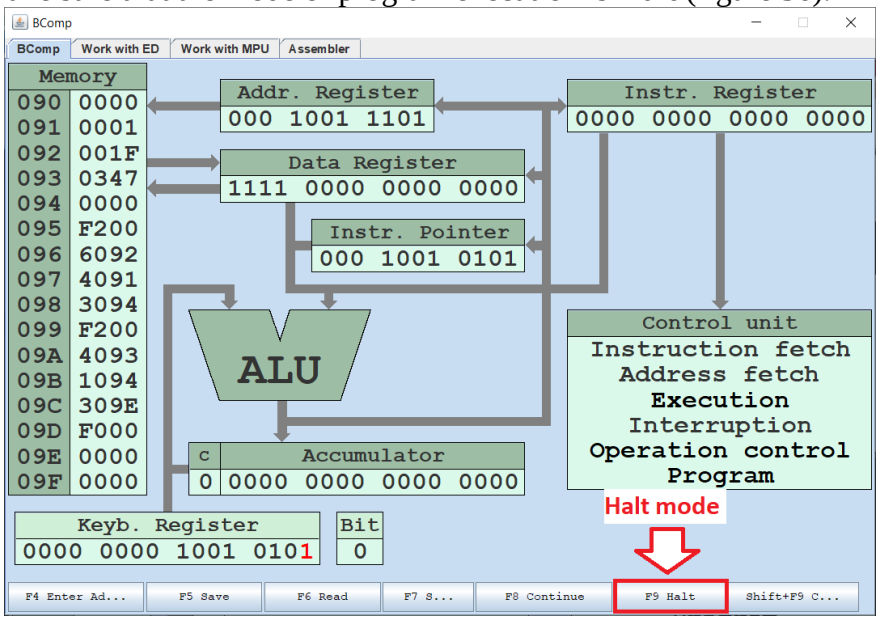



*Figure 36. The halt mode button*

If it does not work, you should press key **F9** to change the mode. The **Halt** mode allows executing program sequentially – step by step.

Press **F7** to start execution. Then press **F8** to execute the first instruction. After the instruction execution, you can see the content of the internal registers: the Addr. Register (**AR**), the Data Register (**DR**), the Instr. Register (**IR**), the Instr. Pointer (**IP**), the Accumulator (**A**) and the carry flag (**C**). It is depictured in figure 37.

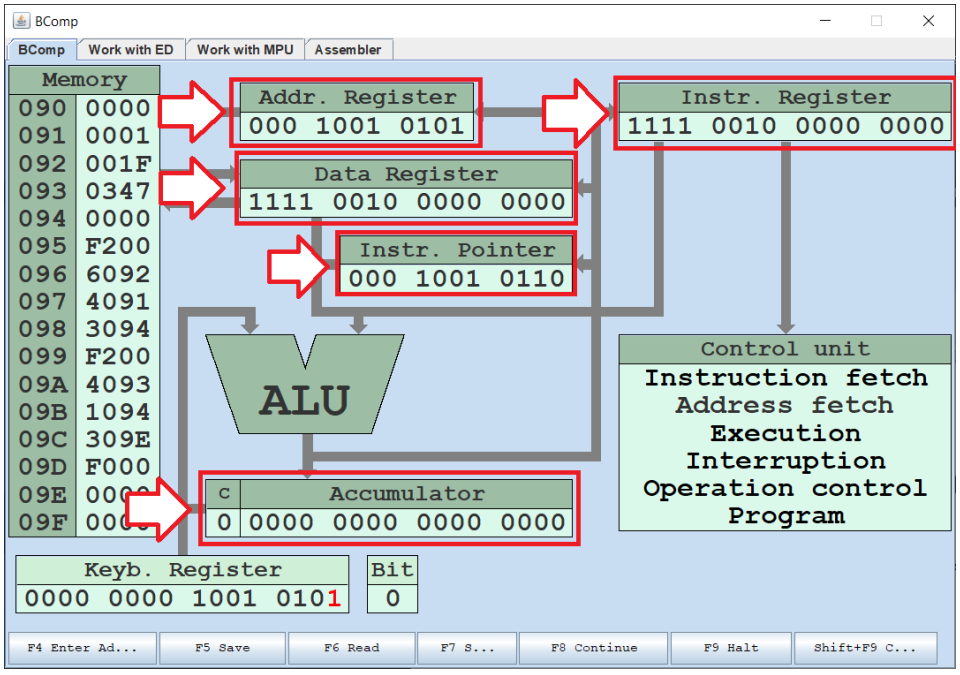



*Figure 37. The content of the internal registers*

Then you should write the content of the internal registers in the answer table as presented in Table 49.

*Table 49. The register content after the F200 instruction execution*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| 095 | F200 | 095 | F200 | F200 | 096 | 0000 | 0 | | |

Continue the program execution step by step till the last instruction **F000** with address **09D** and continue filling the answer table.

The correct answers can be found in Table 50.

*Table 50. The registers content during the program execution*

| ADDRESS | MACHINE CODE | REGISTER CONTENT | | | | | | The cell of the memory that has been changed after instruction execution | |
|---|---|---|---|---|---|---|---|---|---|
| | | AR | DR | IR | IP | A | C | ADDRESS | NEW MACHINE CODE |
| 095 | F200 | 095 | F200 | F200 | 096 | 0000 | 0 | | |
| 096 | 6092 | 092 | 001F | 6092 | 097 | FFE1 | 0 | | |
| 097 | 4091 | 091 | 0001 | 4091 | 098 | FFE2 | 0 | | |
| 098 | 3094 | 094 | FFE2 | 3094 | 099 | FFE2 | 0 | 094 | FFE2 |
| 099 | F200 | 099 | F200 | F200 | 09A | 0000 | 0 | | |
| 09A | 4093 | 093 | 0347 | 4093 | 09B | 0347 | 0 | | |
| 09B | 1094 | 094 | FFE2 | 1094 | 09C | 0342 | 0 | | |
| 09C | 309E | 09E | 0342 | 309E | 09D | 0342 | 0 | 09E | 0342 |
| 09D | F000 | 09D | F000 | F000 | 09E | 0342 | 0 | | |

6. ***Compare the obtained result with the result in point 3. Check yourself – the tables content should be the same. If the tables content are not the same, you should find and correct the mistakes. The mistakes can be in both tables. Be careful.***

We can see the both tables are the same. It means we have done the task successfully.

### 2.2.4 Lab work variants

Variant 1 of laboratory work 2 is presented in Table 51.

*Table 51. Variant 1 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 0D8 | 0045 | |
| 0D9 | 0F13 | |
| 0DA | 0000 | |
| 0DB | F200 | + |
| 0DC | 40E5 | |
| 0DD | 40D8 | |
| 0DE | 30E4 | |
| 0DF | F200 | |
| 0E0 | 40D9 | |
| 0E1 | 10E4 | |
| 0E2 | 30DA | |
| 0E3 | F000 | |
| 0E4 | 0000 | |
| 0E5 | 0007 | |

Variant 2 of laboratory work 2 is presented in Table 52.

*Table 52. Variant 2 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 071 | F200 | + |
| 072 | 407A | |
| 073 | 407C | |
| 074 | 307D | |
| 075 | F200 | |
| 076 | 407B | |
| 077 | 107D | |
| 078 | 307E | |
| 079 | F000 | |
| 07A | 0001 | |
| 07B | 000F | |
| 07C | A001 | |
| 07D | 0000 | |
| 07E | 0000 | |

Variant 3 of laboratory work 2 is presented in Table 53.

*Table 53. Variant 3 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 013 | 0005 | |
| 014 | F200 | + |
| 015 | 6020 | |
| 016 | 401F | |
| 017 | 301D | |
| 018 | F200 | |
| 019 | 4013 | |
| 01A | 101D | |
| 01B | 301E | |
| 01C | F000 | |
| 01D | 0000 | |
| 01E | 0000 | |
| 01F | FF02 | |
| 020 | 0012 | |

Variant 4 of laboratory work 2 is presented in Table 54.

*Table 54. Variant 4 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 070 | 0000 | |
| 071 | 0000 | |
| 072 | 0890 | |
| 073 | 0056 | |
| 074 | F200 | + |
| 075 | 4072 | |
| 076 | 407D | |

| | | |
|---|---|---|
| 077 | 3070 | |
| 078 | F200 | |
| 079 | 4073 | |
| 07A | 1070 | |
| 07B | 3071 | |
| 07C | F000 | |
| 07D | 0A56 | |

Variant 5 of laboratory work 2 is presented in Table 55.

*Table 55. Variant 5 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 012 | 0000 | |
| 013 | 0055 | |
| 014 | 0066 | |
| 015 | 0177 | |
| 016 | F200 | + |
| 017 | 4014 | |
| 018 | 1015 | |
| 019 | 3012 | |
| 01A | F200 | |
| 01B | 4013 | |
| 01C | 4012 | |
| 01D | 301F | |
| 01E | F000 | |
| 01F | 0000 | |

Variant 6 of laboratory work 2 is presented in Table 56.

*Table 56. Variant 6 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 07A | 0000 | |
| 07B | 0456 | |
| 07C | F200 | + |
| 07D | 4085 | |
| 07E | 107B | |
| 07F | 3086 | |
| 080 | F200 | |
| 081 | 4087 | |
| 082 | 4086 | |
| 083 | 307A | |
| 084 | F000 | |
| 085 | 0001 | |
| 086 | 0000 | |
| 087 | 0CCC | |

Variant 7 of laboratory work 2 is presented in Table 57.

*Table 57. Variant 7 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 01C | F200 | + |
| 01D | 4029 | |
| 01E | 1028 | |
| 01F | 3026 | |
| 020 | F200 | |
| 021 | 6027 | |
| 022 | 4026 | |
| 023 | 3025 | |
| 024 | F000 | |
| 025 | 0000 | |
| 026 | 0000 | |
| 027 | 0007 | |
| 028 | 0008 | |
| 029 | F561 | |

Variant 8 of laboratory work 2 is presented in Table 58.

*Table 58. Variant 8 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 078 | F200 | + |
| 079 | 4082 | |
| 07A | 4085 | |
| 07B | 3084 | |
| 07C | F200 | |
| 07D | 4083 | |
| 07E | 1084 | |
| 07F | 3081 | |
| 080 | F000 | |
| 081 | 0000 | |
| 082 | 0123 | |
| 083 | 0A98 | |
| 084 | 0000 | |
| 085 | 0003 | |

Variant 9 of laboratory work 2 is presented in Table 59.

*Table 59. Variant 9 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 01A | 0006 | |
| 01B | 0007 | |
| 01C | 0001 | |
| 01D | 0000 | |
| 01E | F200 | + |
| 01F | 401B | |
| 020 | 101C | |
| 021 | 301D | |
| 022 | F200 | |
| 023 | 601A | |
| 024 | 601D | |
| 025 | 3027 | |
| 026 | F000 | |
| 027 | 0000 | |

Variant 10 of laboratory work 2 is presented in Table 60.

*Table 60. Variant 10 of laboratory work 2*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 083 | AFCB | |
| 084 | 0000 | |
| 085 | 0000 | |
| 086 | F200 | + |
| 087 | 608F | |
| 088 | 6083 | |
| 089 | 3085 | |
| 08A | F200 | |
| 08B | 4090 | |
| 08C | 1085 | |
| 08D | 3084 | |
| 08E | F000 | |
| 08F | 0036 | |
| 090 | 01A8 | |

## 2.3 Laboratory work 3. Program control flow

### 2.3.1 Overview

This lab is aimed at understanding how a program behavior can be changed depending on conditions. The lab gives knowledge about branch instructions and how they are executed in a computer. Different types of addressing are introduced. The lab also gives basic skills of organizing loops in programs and visualizing the algorithm with a flow chart.

After performing the lab task students will be able to analyze the program control flow and control the program behavior using branch instructions and loops.

### 2.3.2 Lab work task

1. Read your variant.
2. Translate the given machine code in a hexadecimal system into a mnemonic code and fill Table 61. Write the type of addressing (direct or indirect) used in the instructions.

*Table 61. Template for answer table for instruction 2 of laboratory work 3*

| ADDRESS | MACHINE CODE | MNEMONIC CODE | TYPE OF ADDRESSING |
|---|---|---|---|
| | | | |

3. Draw the flow chart of your program
4. Enter the program code into the Basic Computer memory
5. Execute your program in the Basic Computer model
6. Fill Table 62 with the content of the memory after the program execution. Write the number of memory accesses for each memory cell during program execution.

*Table 62. The answer table for instruction 6 of laboratory work 3*

| ADDRESS | MACHINE CODE | NUMBER OF MEMORY ACCESSES |
|---|---|---|
| | | |

### 2.3.3 Lab work guidance

To understand how to do the laboratory work we will do the sample variant step by step. Let's start from the first instruction.

***1. Read your variant***

This variant contains the list of machine codes of program data and instructions. The program is stored in the memory of the Basic Computer Model. Each code is kept in address, that you can see in the corresponding line of the column «ADDRESS». The first instruction of your program is marked by symbol «+».

All machine codes are presented as a hexadecimal value.

For example, your variant defines a program, which is in Table 63.

*Table 63. The* sample *variant of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 3FA | 0412 | |
| 3FB | 0004 | |
| 3FC | F500 | |
| 3FD | F200 | + |
| 3FE | 43FB | |
| 3FF | F400 | |
| 400 | F800 | |
| 401 | 300D | |
| 402 | F200 | |
| 403 | 33FC | |
| 404 | 43FA | |
| 405 | 300E | |
| 406 | F200 | |
| 407 | 480E | |
| 408 | F700 | |
| 409 | 840F | |
| 40A | F600 | |
| 40B | 63FC | |
| 40C | 940F | |
| 40D | 43FC | |
| 40E | 33FC | |
| 40F | 000D | |
| 410 | C406 | |
| 411 | F000 | |
| 412 | F501 | |
| 413 | FA01 | |
| 414 | F700 | |
| 415 | E101 | |

2. ***Translate the given machine code in a hexadecimal system into a mnemonic code and fill in the table. Please write down the type of addressing (direct or indirect) that is used in instructions.***

To translate the machine code, you can use the table with the instruction set of Basic Computer. The instruction set of Basic Computer is presented in Appendix A. Note, some memory cells can have a data code instead of an instruction. Basic Computer has the von Neumann architecture. It means data and instructions are stored in the same memory. The program starts from the cell that is identified as **ENTRY POINT** and ends with instruction code F000 (HALT instruction).

The first instruction is **F200**. It is a code of CLA instruction according to the instruction set table (Table 64).

*Table 64. CLA instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Clear accumulator register* | *CLA* | *F200* | *0 -> A* |

The next instruction is **43FB**. It is an addition. The content of the memory cell with address 3FB is added to the accumulator register content. The result is stored in the accumulator register (Table 65).

*Table 65. ADD instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Addition* | *ADD 3FB* | *43FB* | *(3FB) + (A) -> A* |

The next instruction is **F400**. This instruction inverts the value of the accumulator register (Table 66).

*Table 66. CMA instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Invert accumulator register* | *CMA* | *F400* | *(!A) -> A* |

The instruction is **F800**. This instruction increments the accumulator, i.e. it adds to the accumulator value 1 (Table 67).

*Table 67. INC instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Increment accumulator register* | *INC* | *F800* | *(A) + 1 -> A* |

The next instruction is **300D**. It is a move instruction (Table 68). The content of the accumulator registers is stored in the memory cell with address 00D.

*Table 68. MOV instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Move* | *MOV 00D* | *300D* | *(A) -> 00D* |

The next instruction is **F200** again. It clears the accumulator register.

The next instruction is **33FC**. It is a move instruction. The content of the accumulator registers is stored in the memory cell with address 3FC.

The next instruction is **43FA**. It is an addition. The content of the memory cell with address 3FA is added to the accumulator register. The result is stored in the accumulator register.

The next instruction is **300E**. It is a move instruction. The content of the accumulator registers is stored in the memory cell with address 00E.

The next instruction is **F200** again. It clears the accumulator register.

The next instruction is **480E**. It is an addition. This instruction uses **INDIRECT** addressing because of the most significant bit in the address 80E (1000 0000 1110 in the binary system) equals 1. The content of the memory cell with the address, which is stored in the cell with address 00E, is added to the accumulator register. The result is stored in the accumulator register. **Attention!** The memory cells with addresses from 008 to 00F are index registers. The content of such a cell is autoincremented when the program writes or reads from it using **INDIRECT** addressing. In this case, after the execution of this instruction, **the content of the cell 00E will be autoincremented!**

The next instruction is **F700**. This instruction makes the right cyclic shift of the value of the carry flag in the accumulator register by 1 bit. The cyclic shift means that the least significant bit is not lost and goes to the place of the carry flag «C». The value of the carry flag goes to the place of 15th bit of the accumulator as depictured below (figure 38 and Table 69).

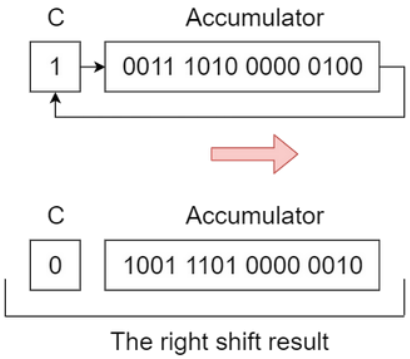



*Figure 38. The scheme of ROR instruction execution*

*Table 69. ROR instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Right cyclic shift by 1 bit* | *ROR* | *F700* | *A & C content moves right, A(0) -> C, C -> A(15)* |

The next instruction is **840F**. It is a branch instruction. The program goes to the address 40F if the carry flag is set. In this case, the address 40F is stored in Instruction Pointer (**IP**) register (Table 70). If the carry flag is not set, the program goes further and the Basic Computer Model executes the next instruction in address 40A.

*Table 70. BCS instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Branch if carry flag is set* | *BCS 40F* | *840F* | *If (C)=1, then 40F -> IP* |

The next instruction is **F600**. This instruction makes the left cyclic shift of the value of the carry flag in the accumulator register by 1 bit. The cyclic shift means that the most significant bit is not lost and goes to the place of the carry flag «C». The value of the carry flag goes to the place of 0 (zero) bit of the accumulator as depictured below (figure 39 and Table 71).

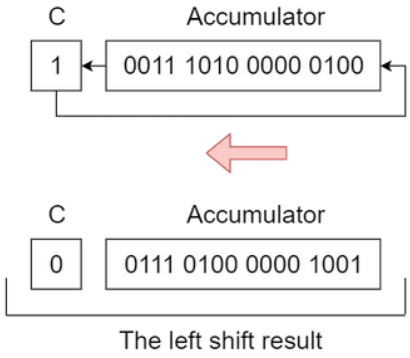



*Figure 39. The scheme of ROL instruction execution*

*Table 71. ROL instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Left cyclic shift by 1 bit* | *ROL* | *F600* | *A & C content moves left, A(15) -> C, C -> A(0)* |

The next instruction is **63FC**. It is a subtraction. The content of the memory cell with address 3FC is subtracted from the accumulator register. The result is stored in the accumulator register (Table 72).

*Table 72. SUB instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Subtraction* | *SUB 3FC* | *63FC* | *(B) – (3FC) -> (A)* |

The next instruction is **940F**. It is a branch instruction. If the accumulator value is greater or equal to zero, Basic Computer goes to the address 40F. In this case, the address 40F will be stored in Instruction Pointer (**IP**) register. If the condition is not satisfied, the program goes further and the Basic Computer executes the next instruction in address 40D (Table 73).

*Table 73. BPL instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Branch if the accumulator is positive* | *BPL 40F* | *940F* | *If (A) >= 0, then 40F -> IP* |

The next instruction is **43FC**. It is an addition. The content of the memory cell with address 3FC is added to the accumulator register. The result is stored in the accumulator register.

The next instruction is **33FC**. It is a move instruction. The content of the accumulator registers is stored in the memory cell with address 3FC.

The next instruction is **000D**. This instruction increments the value in the address 000D and if the result is a positive number, the instruction pointer is incremented also. If the result is negative, Basic Computer performs the next instruction without jumping through one instruction (Table 74).

*Table 74. ISZ instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Increment and skip* | *ISZ 00D* | *000D* | *(00D) + 1 -> 00D<br>If (00D) >= 0 then<br>(IP)+1 -> IP* |

The next instruction is **C406**. It is an unconditional branch. After performing this instruction Basic Computer jumps to the address 406. It changes the content of instruction pointer (IP) to 406 (Table 75).

*Table 75. BR instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Unconditional branch* | *BR 406* | *C406* | *406 -> IP* |

The next instruction is **F000**. It is a halt instruction. It indicates the end of the program (Table 76).

*Table 76. ROL instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Halt* | *HLT* | *F000* | *Stop program execution* |

Thus, the correct answer at this point is presented below in Table 77.

*Table 77. The answer for instruction 2 of the* sample *variant*

| ADDRESS | MACHINE CODE | MNEMONIC CODE | TYPE OF ADDRESSING |
|---|---|---|---|
| 3FA | 0412 | | |
| 3FB | 0004 | | |
| 3FC | F500 | | |
| 3FD | F200 | CLA | |
| 3FE | 43FB | ADD 3FB | DIRECT |
| 3FF | F400 | CMA | |
| 400 | F800 | INC | |
| 401 | 300D | MOV 00D | DIRECT |
| 402 | F200 | CLA | |
| 403 | 33FC | MOV 3FC | DIRECT |
| 404 | 43FA | ADD 3FA | DIRECT |
| 405 | 300E | MOV 00E | DIRECT |
| 406 | F200 | CLA | |
| 407 | 480E | ADD 80E | INDIRECT |
| 408 | F700 | ROR | |
| 409 | 840F | BCS 40F | DIRECT |
| 40A | F600 | ROL | |
| 40B | 63FC | SUB 3FC | DIRECT |
| 40C | 940F | BPL 40F | DIRECT |
| 40D | 43FC | ADD 3FC | DIRECT |
| 40E | 33FC | MOV 3FC | DIRECT |
| 40F | 000D | ISZ 00D | DIRECT |
| 410 | C406 | BR 406 | DIRECT |
| 411 | F000 | HLT | |
| 412 | F501 | | |
| 413 | FA01 | | |
| 414 | F700 | | |
| 415 | E101 | | |

### *3. Draw the flow chart of your program.*

The flow chart helps to understand the algorithm that is realized in the program. It also helps to understand the control flow of the program, that contains branches and loops.

To draw the flow chart in this lab we need three figures.

The first is an ellipse. It is presented in figure 40 and defines the start and endpoint of a program.

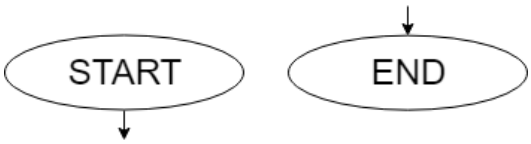



*Figure 40. The start and end point of the program*

The next figure is a rectangle. It is used to represent actions in a program. For example, the first instruction in our program is F200 (CLA). It sets zero value in the accumulator. We can draw this action in the flow chart as presented in figure 41.

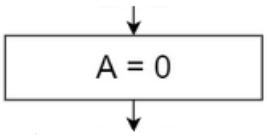



*Figure 41. The CLA instruction in the flow chart*

The last necessary figure is a rhombus. It represents branches. For example, the first branch in our program is BCS 40F (840F) in address 409. It checks the carry flag. We can show it in the flow chart as presented in figure 42.

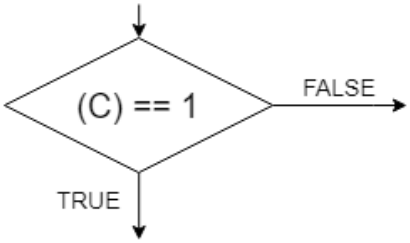



*Figure 42. The BCS instruction in the flow chart*

The rhombus has two branches illustrating what the program does if the condition is TRUE or FALSE. The condition is written in the center of the rhombus.

To draw the flow chart for our program we need to convert mnemonic codes of instructions to program actions. We use the parentheses to show that the value of the accumulator or the memory cell is used in the instruction.

For example, for 43FB instruction Basic Computer does:

A = (A) + (3FB),

i.e. the value of the memory cell 3FB is added to the accumulator. For a cyclic shift we use such symbols as >>> (right cycle shift) and <<< (left cycle shift). For the inversion, we use the symbol «!» (exclamation mark).

We need to draw the flow chart step by step. The arrows in the flow chart show the sequence of instruction execution in the program.

The right program chart for our program is presented in figure 43.

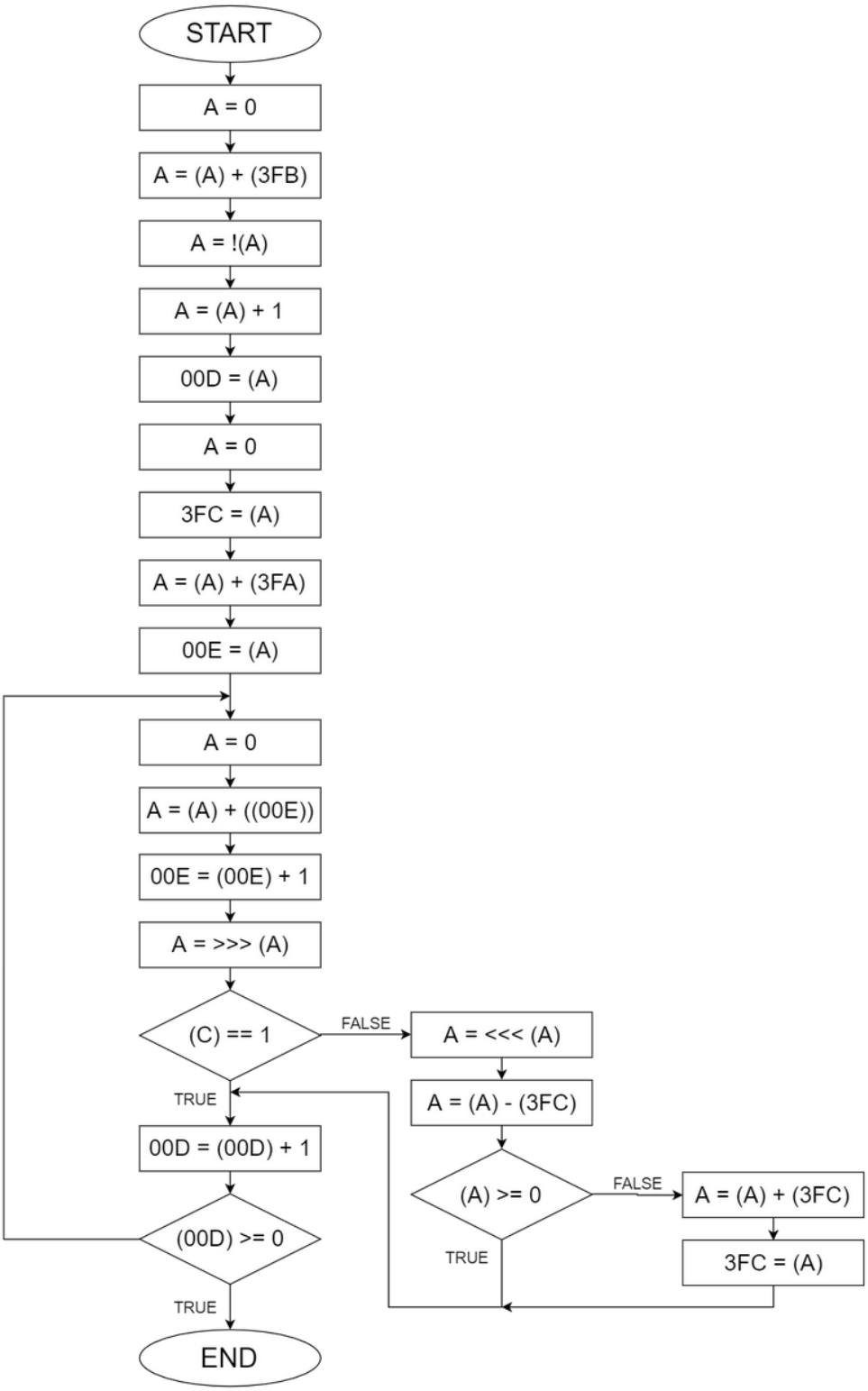



*Figure 43. The flow chart of the program*

If we analyze this flow chart, we can find the three branches in our program. One of these branches defines a loop, i.e. repeating part of the code. The loops are used to repeat the same operations with different data, for example with elements of data arrays.

In this case, the instruction 000D defines the loop. The loop ends when the content of the memory cell with address 00D becomes a positive number.

### *4. Enter the program code into the Basic Computer Memory*

To enter a program code into Basic Computer Memory you should open the Basic Computer Model. It can be done by double click on the file **bcomp.jar**. After that you can see the structure of the processor of Basic Computer presented in figure 44.

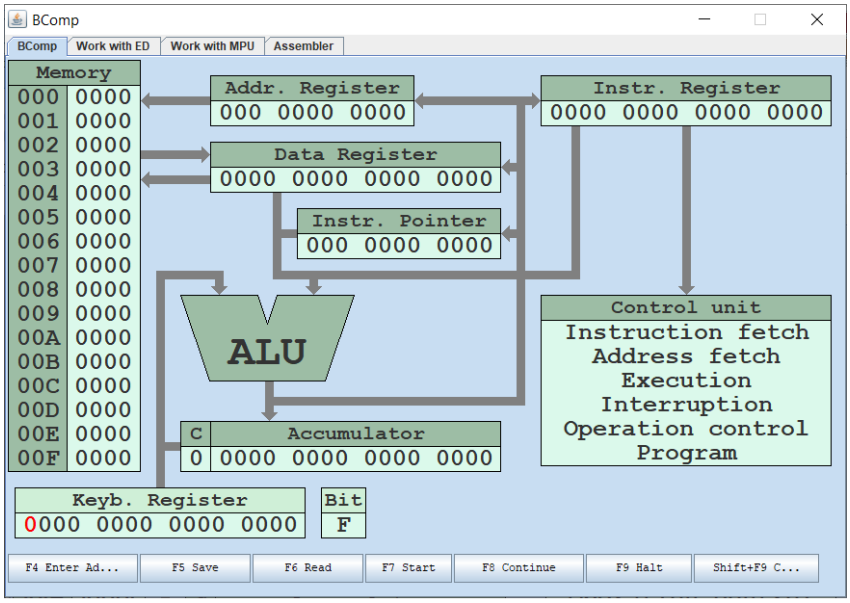



*Figure 44. The main window of the Basic Computer Model*

Let's enter your program into Basic Computer memory. To enter each machine code into the memory you need to define the address using **F4** key and memory cell value using **F5** key. If you enter the codes sequentially, you should define only the address of the first code. After storing the first code, the **Instr. Pointer** will be autoincremented and you can enter the second code immediately. For example, let's enter the first code **0412** in address **3FA**:

Use **Up** arrow on your keyboard to change the value of each bit in **Keyb. Register**. Use arrows **Left** and **Right** to move from one bit to another. Set the address of the first code in the **Keyb. Register** (figure 45). In our sample, the address of the first code is 3FA (0000 0011 1111 1010 in a binary system). The Keyb. Register is used as an intermediate register to input data to the Basic Computer Memory. It looks like a simple user console.

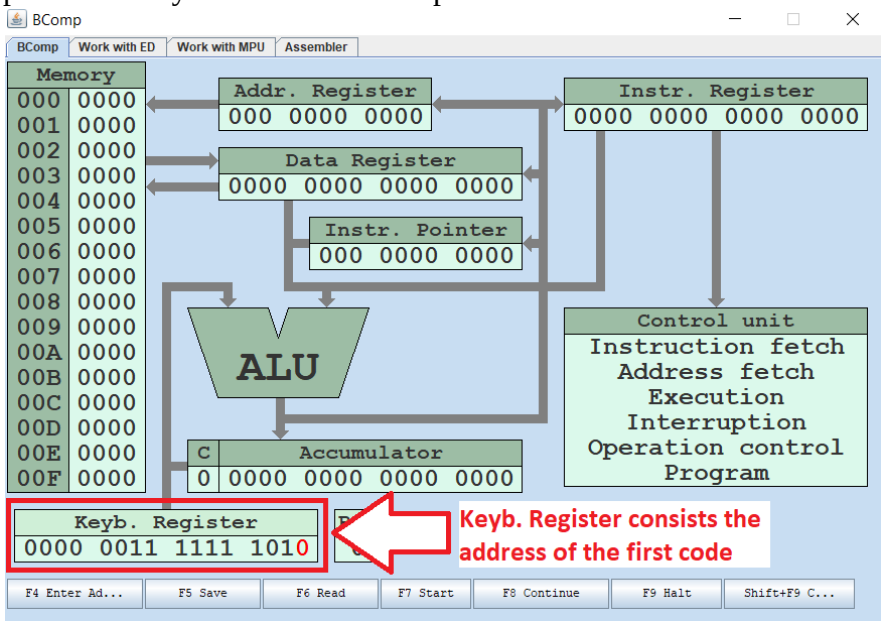



*Figure 45. The flow chart of the program*

Press **F4** to store this address in **Instr. Pointer** (figure 46).

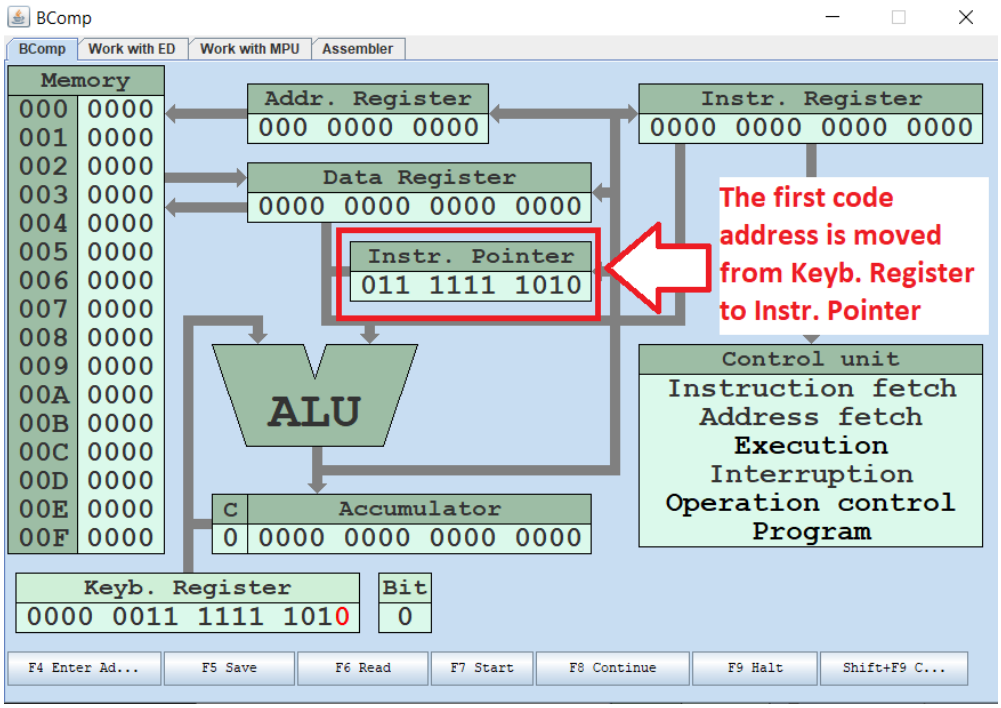



*Figure 46. The first code address in Instr. Pointer*

Enter the value of machine code **0412** (0000 0100 0001 0010 in binary system) into **Keyb. Register** (figure 47).

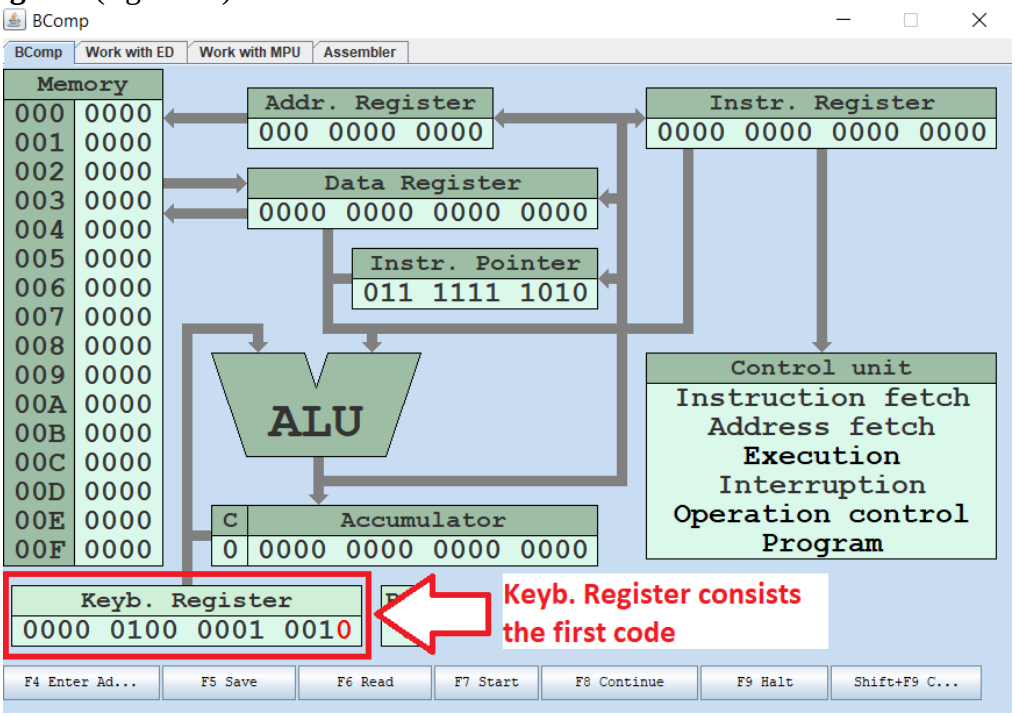



*Figure 47. The first code in Keyb. Register*

Press **F5** to store the value of **Keyb. Register** in the memory in the cell with 3FA address (figure 48).

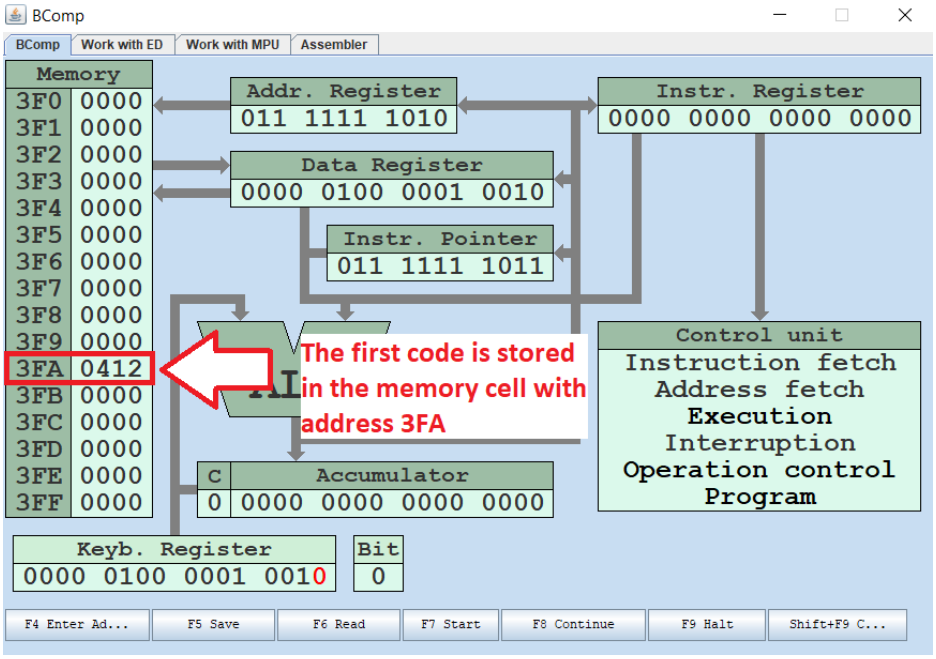



*Figure 48. The first code in the memory*

Then enter the rest codes of your program. For the second code and other you do not need to enter the code address again as the content of Instr. Pointer register will be autoincremented after pressing **F5**.

### 5. *Execute your program in Basic Computer Model*

Before starting the execution, you need to define the first program address, i.e. the program entry point. Enter the first address **3FD** of your program in **Instr. Pointer** using Keyb. Register and **F4** as before.

Then make sure that the mode of program execution is **Halt**, as presented in figure 49.

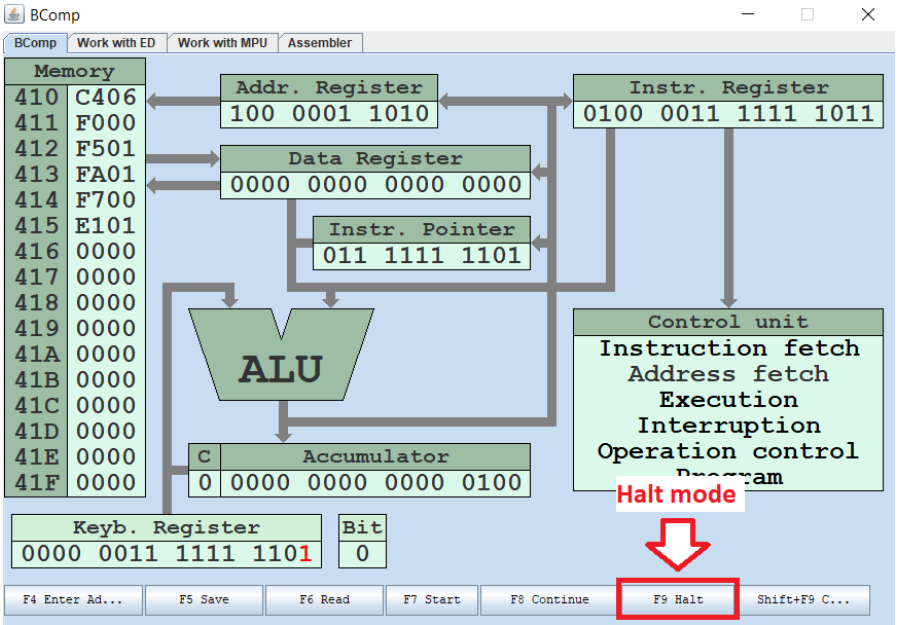



*Figure 49. The halt mode button*

If it is not so, you should press the key **F9** to change the mode. The **Halt** mode allows executing program sequentially – step by step.

Press **F7** to start execution. Then press **F8** to continue execution till the last instruction **F000** with address **411**. During execution you need **to count the number of memory accesses**. A memory access can be a reading instruction from the memory or reading and writing data into memory.

If you have done all instructions correctly, you will see the window presented in figure 50.

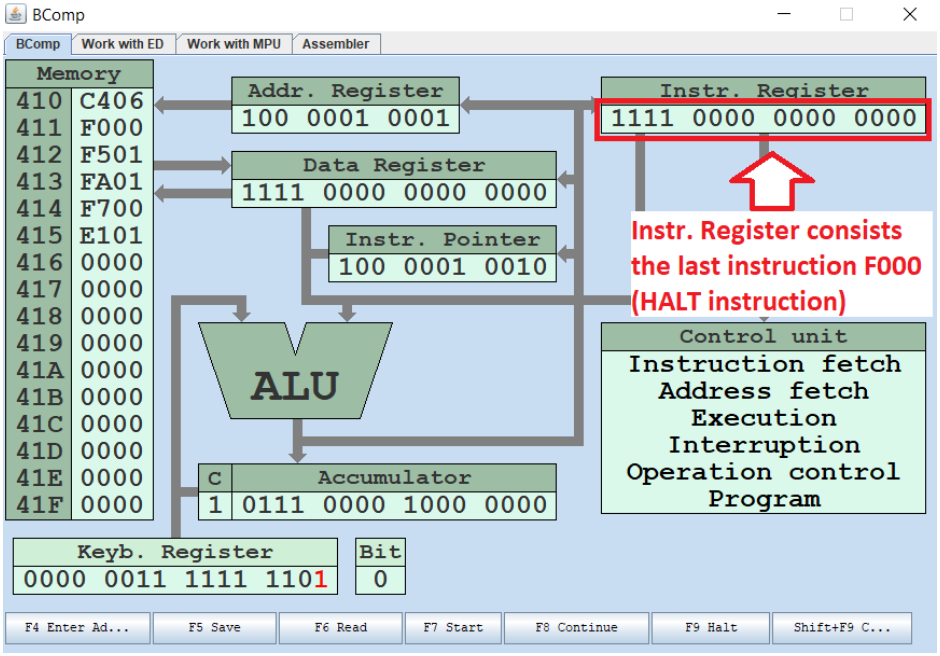



*Figure 50. The HALT instruction in Instr. Register*

The **Inst. Register** has the last instruction **F000**. It is HALT instruction. It is the end of the program.

In the left side of the window we can see the **Memory** content. We need to use this information in the next step of our task.

6. ***Fill in the table with the content of the memory after the program execution. Please write down the number of memory accesses for each memory cell during program execution.***

After program execution we can see only a part of memory content in the left side of the Basic Computer window. To see the top part of the memory we can enter the address of the first data cell 3FA and press F4 to enter address and then press F6 to read from this address. After this you can see the top part of the memory content as presented in Table 51.

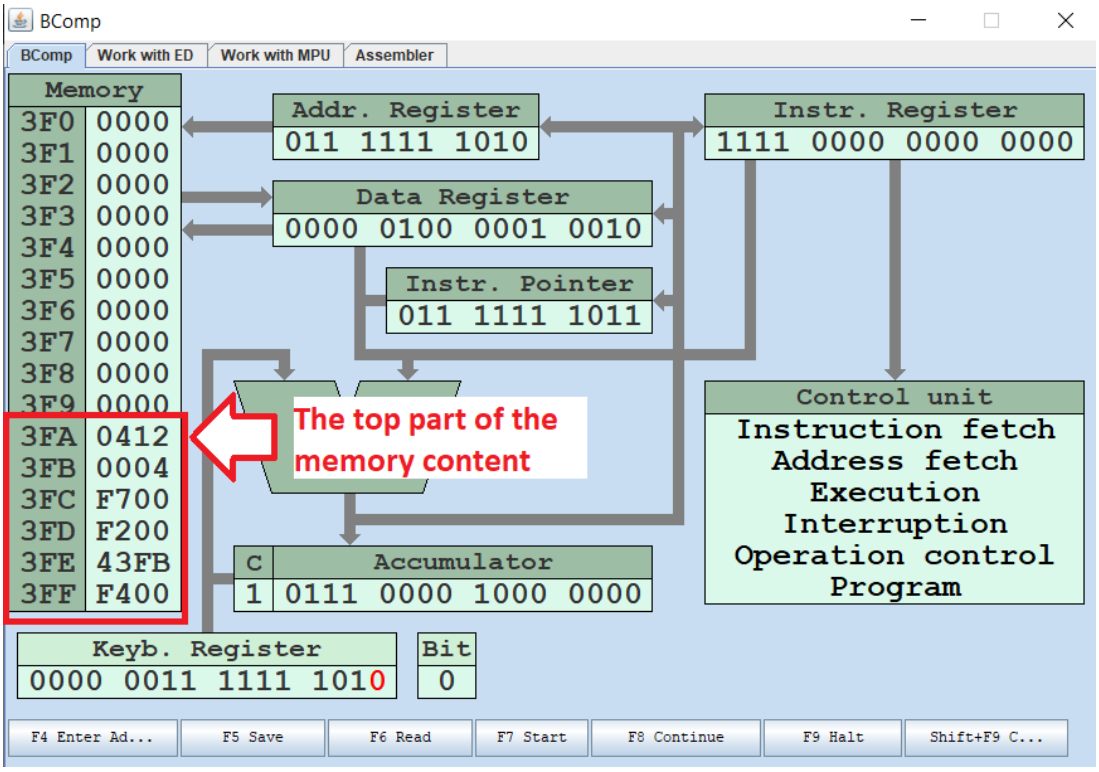



*Figure 51. The top part of the memory content*

If you enter the program correctly, the Basic Computer memory will have the content presented in Table 78. In the table you can see the number of memory accesses.

*Table 78. The answer for instruction 6 of the* sample *variant*

| ADDRESS | MACHINE CODE | NUMBER OF MEMORY ACCESSES |
|---|---|---|
| 3FA | 0412 | 1 |
| 3FB | 0004 | 1 |
| 3FC | F700 | 4 |
| 3FD | F200 | 1 |
| 3FE | 43FB | 1 |
| 3FF | F400 | 1 |
| 400 | F800 | 1 |
| 401 | 300D | 1 |
| 402 | F200 | 1 |
| 403 | 33FC | 1 |
| 404 | 43FA | 1 |
| 405 | 300E | 1 |
| 406 | F200 | 4 |
| 407 | 480E | 4 |
| 408 | F700 | 4 |
| 409 | 840F | 4 |
| 40A | F600 | 1 |
| 40B | 63FC | 1 |
| 40C | 940F | 1 |
| 40D | 43FC | 1 |
| 40E | 33FC | 1 |
| 40F | 000D | 4 |
| 410 | C406 | 3 |
| 411 | F000 | 1 |
| 412 | F501 | 1 |
| 413 | FA01 | 1 |
| 414 | F700 | 1 |
| 415 | E101 | 1 |

If we follow up the program execution, we can understand that in the data array from address 412 to address 415 the program finds the last negative element, which has the least significant bit equal to zero, and stores it in the memory cell with address 3FC.

### 2.3.4 Lab work variants

Variant 1 of laboratory work 3 is presented in Table 79.

*Table 79. Variant 1 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 332 | 034B | |
| 333 | 0004 | |
| 334 | F500 | |
| 335 | F200 | + |
| 336 | 4333 | |
| 337 | F400 | |
| 338 | F800 | |
| 339 | 300C | |
| 33A | F200 | |
| 33B | 3334 | |
| 33C | 4332 | |
| 33D | 300B | |
| 33E | F200 | |
| 33F | 480B | |
| 340 | F700 | |
| 341 | 8348 | |
| 342 | F700 | |
| 343 | 8348 | |
| 344 | F600 | |
| 345 | F600 | |
| 346 | 4334 | |
| 347 | 3334 | |
| 348 | 000C | |
| 349 | C33E | |
| 34A | F000 | |
| 34B | 533D | |
| 34C | F700 | |
| 34D | F402 | |
| 34E | 233D | |

Variant 2 of laboratory work 3 is presented in Table 80.

*Table 80. Variant 2 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 36D | 0385 | |
| 36E | 0004 | |
| 36F | F500 | |
| 370 | F200 | + |
| 371 | 436D | |
| 372 | 300E | |
| 373 | F200 | |
| 374 | 336F | |
| 375 | 636E | |
| 376 | 300D | |
| 377 | F200 | |
| 378 | 480E | |
| 379 | F700 | |
| 37A | F500 | |
| 37B | 8382 | |
| 37C | F500 | |
| 37D | F600 | |
| 37E | F200 | |
| 37F | 436F | |
| 380 | F800 | |
| 381 | 336F | |
| 382 | 000D | |
| 383 | C377 | |
| 384 | F000 | |
| 385 | 0378 | |
| 386 | C378 | |
| 387 | 8378 | |
| 388 | C36C | |

Variant 3 of laboratory work 3 is presented in Table 81.

*Table 81. Variant 3 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 2FD | 0311 | |
| 2FE | 0003 | |
| 2FF | F300 | |
| 300 | F200 | + |
| 301 | 32FF | |
| 302 | 42FE | |
| 303 | F400 | |
| 304 | F800 | |
| 305 | 300C | |
| 306 | F200 | |
| 307 | 42FD | |
| 308 | 300D | |
| 309 | F200 | |
| 30A | 480D | |
| 30B | A30E | |
| 30C | 02FF | |
| 30D | F100 | |
| 30E | 000C | |
| 30F | C309 | |
| 310 | F000 | |
| 311 | 230C | |
| 312 | F000 | |
| 313 | 12FE | |

Variant 4 of laboratory work 3 is presented in Table 82.

*Table 82. Variant 4 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 2F8 | 030F | |
| 2F9 | 0005 | |
| 2FA | F300 | |
| 2FB | F200 | + |
| 2FC | 32FA | |
| 2FD | 42F8 | |
| 2FE | 300E | |
| 2FF | F200 | |
| 300 | 62F9 | |
| 301 | 300D | |
| 302 | F200 | |
| 303 | 480E | |
| 304 | F700 | |
| 305 | 830C | |
| 306 | F700 | |
| 307 | 830C | |
| 308 | F600 | |
| 309 | F600 | |
| 30A | 62FA | |
| 30B | 32FA | |
| 30C | 000D | |
| 30D | C302 | |
| 30E | F000 | |
| 30F | F501 | |
| 310 | F300 | |
| 311 | 6309 | |
| 312 | 6305 | |
| 313 | F903 | |

Variant 5 of laboratory work 3 is presented in Table 83.

*Table 83. Variant 5 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 3DD | 03F6 | |
| 3DE | 0005 | |
| 3DF | F500 | |
| 3E0 | F200 | + |
| 3E1 | 33DF | |
| 3E2 | 43DD | |
| 3E3 | 3009 | |
| 3E4 | F200 | |
| 3E5 | 43DE | |
| 3E6 | F400 | |
| 3E7 | F800 | |
| 3E8 | 300A | |
| 3E9 | F200 | |
| 3EA | 4809 | |
| 3EB | F700 | |
| 3EC | 83EE | |
| 3ED | C3F3 | |
| 3EE | F600 | |
| 3EF | 63DF | |
| 3F0 | A3F3 | |
| 3F1 | 43DF | |
| 3F2 | 33DF | |
| 3F3 | 000A | |
| 3F4 | C3E9 | |
| 3F5 | F000 | |
| 3F6 | F301 | |
| 3F7 | B3EB | |
| 3F8 | 13EB | |
| 3F9 | FA00 | |
| 3FA | FB00 | |

Variant 6 of laboratory work 3 is presented in Table 84.

*Table 84. Variant 6 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 417 | 042D | |
| 418 | 0004 | |
| 419 | F300 | |
| 41A | F200 | + |
| 41B | 3419 | |
| 41C | 6418 | |
| 41D | 3009 | |
| 41E | F200 | |
| 41F | 4417 | |
| 420 | 300A | |
| 421 | F200 | |
| 422 | 480A | |
| 423 | F700 | |
| 424 | F500 | |
| 425 | F200 | |
| 426 | F400 | |
| 427 | 1419 | |
| 428 | F600 | |
| 429 | 3419 | |
| 42A | 0009 | |
| 42B | C421 | |
| 42C | F000 | |
| 42D | E300 | |
| 42E | 1423 | |
| 42F | F400 | |
| 430 | 3422 | |

Variant 7 of laboratory work 3 is presented in Table 85.

*Table 85. Variant 7 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 3A8 | 03C0 | |
| 3A9 | 0003 | |
| 3AA | F300 | |
| 3AB | F200 | + |
| 3AC | 43A8 | |
| 3AD | 300C | |
| 3AE | F200 | |
| 3AF | 33AA | |
| 3B0 | 43A9 | |
| 3B1 | F400 | |
| 3B2 | F800 | |
| 3B3 | 300D | |
| 3B4 | F200 | |
| 3B5 | 480C | |
| 3B6 | F700 | |
| 3B7 | F500 | |
| 3B8 | 83BD | |
| 3B9 | F500 | |
| 3BA | F600 | |
| 3BB | 43AA | |
| 3BC | 33AA | |
| 3BD | 000D | |
| 3BE | C3B4 | |
| 3BF | F000 | |
| 3C0 | E000 | |
| 3C1 | 13B5 | |
| 3C2 | F501 | |

Variant 8 of laboratory work 3 is presented in Table 86.

*Table 86. Variant 8 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 3E2 | 03F7 | |
| 3E3 | 0005 | |
| 3E4 | F500 | |
| 3E5 | F200 | + |
| 3E6 | 63E3 | |
| 3E7 | 3009 | |
| 3E8 | F200 | |
| 3E9 | 33E4 | |
| 3EA | 43E2 | |
| 3EB | 300A | |
| 3EC | F200 | |
| 3ED | 480A | |
| 3EE | F700 | |
| 3EF | F200 | |
| 3F0 | F400 | |
| 3F1 | 13E4 | |
| 3F2 | F600 | |
| 3F3 | 33E4 | |
| 3F4 | 0009 | |
| 3F5 | C3EC | |
| 3F6 | F000 | |
| 3F7 | 93F4 | |
| 3F8 | 33EA | |
| 3F9 | B3E3 | |
| 3FA | E301 | |
| 3FB | A3E7 | |

Variant 9 of laboratory work 3 is presented in Table 87.

*Table 87. Variant 9 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 4C7 | 04D9 | |
| 4C8 | 0005 | |
| 4C9 | F500 | |
| 4CA | F200 | + |
| 4CB | 64C8 | |
| 4CC | 300B | |
| 4CD | F200 | |
| 4CE | 34C9 | |
| 4CF | 44C7 | |
| 4D0 | 300C | |
| 4D1 | F200 | |
| 4D2 | 480C | |
| 4D3 | 94D6 | |
| 4D4 | 04C9 | |
| 4D5 | F100 | |
| 4D6 | 000B | |
| 4D7 | C4D1 | |
| 4D8 | F000 | |
| 4D9 | F200 | |
| 4DA | 0000 | |
| 4DB | 0000 | |
| 4DC | 24C8 | |
| 4DD | F100 | |

Variant 10 of laboratory work 3 is presented in Table 88.

*Table 88. Variant 10 of laboratory work 3*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 502 | 0518 | |
| 503 | 0003 | |
| 504 | F300 | |
| 505 | F200 | |
| 506 | 3504 | |
| 507 | 4502 | |
| 508 | 300B | |
| 509 | F200 | |
| 50A | 4503 | |
| 50B | F400 | |
| 50C | F800 | |
| 50D | 300C | |
| 50E | F200 | |
| 50F | 480B | |
| 510 | 9515 | |
| 511 | 6504 | |
| 512 | 9515 | |
| 513 | 4504 | |
| 514 | 3504 | |
| 515 | 000C | |
| 516 | C50E | |
| 517 | F000 | |
| 518 | 0000 | |
| 519 | 0000 | |
| 51A | 0000 | |

## 2.4 Laboratory work 4. Subprograms

### 2.4.1 Overview

This lab is dedicated to understanding the principles of reusing code in the program. The lesson observes how to encapsulate code in subprograms and reuse it. The students will know what is a program context and how it can be stored and restored, how subprograms are executed on a low level in a computer. This task develops the skill of analyzing the behavior of modular program with subprograms and how to write it.

After completing this lab work student will know what program context is. They will be able to encapsulate code in subprograms; to store and restore program context; to write programs with subprograms.

### 2.4.2 Lab work task

1. Read your variant
2. Translate the given hexadecimal machine code into mnemonic code and fill Table 89. Please write down the type of special points in the code: the entry point of the main program (EPMP), the entry point of subprogram (EPSP), the end point of the main program (ENDMP), the endpoint of a subprogram (ENDSP), the place of the return address (RA).

*Table 89. Template for answer table for instruction 2 of laboratory work 4*

| ADDRESS | MACHINE CODE | MNEMONIC CODE | CODE REGION |
|---|---|---|---|
| | | | |

3. Draw the flow chart of your program.
4. Enter the program code into the Basic Computer Memory
5. Execute your program in the Basic Computer Model
6. Fill in Table 90 with the content of the memory after the program execution. Please write down the number of memory accesses for each memory cell.

*Table 90. The answer table for instruction 6 of laboratory work 4*

| ADDRESS | MACHINE CODE | NUMBER OF MEMORY ACCESSES |
|---|---|---|
| | | |

### 2.4.3 Lab work guidance

To understand how to do the laboratory work we will do the sample variant step by step. Let's start from the first instruction.

1. ***Read your variant***

   For example, the variant defines a program that is listed in Table 91.

*Table 91. The* sample *variant of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 5DB | F200 | + |
| 5DC | 35F0 | |
| 5DD | 45ED | |
| 5DE | 26A0 | |
| 5DF | 65F0 | |
| 5E0 | 35F0 | |
| 5E1 | F200 | |
| 5E2 | 45EE | |
| 5E3 | F900 | |
| 5E4 | 26A0 | |
| 5E5 | 45F0 | |
| 5E6 | 35F0 | |
| 5E7 | F200 | |
| 5E8 | 45EF | |
| 5E9 | 26A0 | |
| 5EA | 45F0 | |
| 5EB | 35F0 | |
| 5EC | F000 | |
| 5ED | 0045 | |
| 5EE | F567 | |
| 5EF | 0707 | |
| 5F0 | 0000 | |
| | | |
| 6A0 | 0000 | |
| 6A1 | A6A6 | |
| 6A2 | 66AF | |
| 6A3 | 96A5 | |
| 6A4 | C6AB | |
| 6A5 | 46AF | |
| 6A6 | 36AE | |
| 6A7 | F300 | |
| 6A8 | F600 | |
| 6A9 | 66B0 | |
| 6AA | CEA0 | |
| 6AB | F200 | |
| 6AC | 46AF | |
| 6AD | CEA0 | |
| 6AE | 0000 | |
| 6AF | 00AC | |
| 6B0 | 0081 | |

2. ***Translate the given hexadecimal machine code into the mnemonic code and fill in the table. Please write down the type of the special points in the code: the entry point of the main program (EPMP), the entry point of a subprogram (EPSP), the endpoint of the main program (ENDMP), the end point of subprogram (ENDSP), the place of the return address (RA).***

To translate the machine code, you can use the table with the instruction set of Basic Computer. The instruction set of Basic Computer is presented in appendix A. Note, some memory cells can have data code instead of instructions. Basic Computer has the von Neumann architecture. It means data and instructions are stored in the same memory. The program starts from the cell that is identified as **ENTRY POINT** and ends with instruction code F000 (HALT instruction).

The first instruction is **F200**. It is a code of CLA instruction according to the instruction set table (Table 92).

*Table 92. CLA instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Clear accumulator register* | *CLA* | *F200* | *0 -> A* |

The next instruction is **35F0**. It is a move instruction. The content of the accumulator register is stored in the memory cell with address 5F0 (Table 93).

*Table 93. MOV instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Move* | *MOV 5F0* | *35F0* | *(A) -> 5F0* |

The next instruction is **45ED**. It is Addition. The content of the memory cell with address 5ED is added to the accumulator register. The result is stored in the accumulator register (Table 94).

*Table 94. ADD instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Addition* | *ADD 5ED* | *45ED* | *(5ED) + (A) -> A* |

The next instruction is **26A0**. It is a subprogram call. This instruction stores the content of the instruction pointer at the address **6A0** and change the instruction pointer value to the 6A0+1 = **6A1**. The subprogram starts execution from address **6A1**. The memory cell with address **6A0** keeps the program context, i.e. the return address (Table 95).

*Table 95. JSR instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Subprogram call* | *JSR 6A0* | *26A0* | *(IP) -> A, 6A0 + 1 -> IP* |

In some cases, it is necessary to store the accumulator value in the memory at the begin of the subprogram because it can change it in an unpredictable way. In this case, we need also restore the accumulator value in the end of the subprogram.

The next instruction is **65F0**. It is a subtraction. The content of the memory cell with address 5F0 is subtracted from the accumulator register. The result is stored in the accumulator register (Table 96).

*Table 96. SUB instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Subtraction* | *SUB 5F0* | *65F0* | *(C) – (5F0) -> (A)* |

The next instruction is **35F0**. It is a move instruction. The content of the accumulator registers is stored in the memory cell with address 5F0.

The next instruction is **F200** again. It clears the accumulator register.

The next instruction is **45EE**. It is an addition. The content of the memory cell with address 5EE is added to the accumulator register. The result is stored in the accumulator register.

The instruction is **F900**. This instruction decrements the accumulator, i.e. it subtracts 1 from the accumulator value (Table 97).

*Table 97. DEC instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Increment accumulator register* | *DEC* | *F900* | *(B) - 1 -> A* |

The next instruction is **26A0** again. It is a subprogram call.

The next instruction is **45F0**. It is an addition. The content of the memory cell with address 5F0 is added to the accumulator register. The result is stored in the accumulator register.

The next instruction is **35F0**. It is a move instruction. The content of the accumulator registers is stored in the memory cell with address 5F0.

The next instruction is **F200** again. It clears the accumulator register.

The next instruction is **45EF**. It is an addition. The content of the memory cell with address 5EF is added to the accumulator register. The result is stored in the accumulator register.

The next instruction is **26A0** again. It is a subprogram call.

The next instruction is **45F0**. It is an addition. The content of the memory cell with address 5F0 is added to the accumulator register. The result is stored in the accumulator register.

The next instruction is **35F0**. It is a move instruction. The content of the accumulator registers is stored in the memory cell with address 5F0.

The next instruction is **F000**. It is halt instruction (Table 98). It indicates the end of the program.

*Table 98. HLT instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Halt* | *HLT* | *F000* | *Stop program execution* |

We have decoded all the instructions in the main program. Then we need to decode the instruction of the subprogram that starts from address 6A0.

The first instruction of the subprogram is **A6A6**. It is a branch instruction. If the accumulator value less than zero, Basic Computer will go to the address 6A6. In this case, the address 6A6 will be stored in Instruction Pointer (**IP**) register (Table 99). If the condition is not satisfied, the program goes further and the Basic Computer will execute the next instruction in address 6A2.

*Table 99. BMI instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Branch if the accumulator is negative* | *BMI 6A6* | *A6A6* | *If (A) < 0, then 6A6 -> IP* |

The next instruction is **66AF**. It is Subtraction. The content of the memory cell with address 6AF is subtracted from the accumulator register. The result is stored in the accumulator register.

The next instruction is **96A5**. It is a branch instruction. If the accumulator value is greater or equal to zero, the Basic Computer will go to the address 6A5. In this case, the address 6A5 will be stored in Instruction Pointer (**IP**) register (Table 100). If the condition is not satisfied, the program goes further and the Basic Computer will execute the next instruction in address 6A4.

*Table 100. BPL instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Branch if the accumulator is positive* | *BPL 6A5* | *96A5* | *If (A) >= 0, then 6A5 -> IP* |

The next instruction is **C6AB**. It is an unconditional branch. After performing this instruction Basic Computer jumps to the address 6AB. It changes the content of instruction pointer (IP) to 6AB (Table 101).

*Table 101. BR instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Unconditional branch* | *BR 6AB* | *C6AB* | *6AB -> IP* |

The next instruction is **46AF**. It is an addition. The content of the memory cell with address 6AF is added to the accumulator register. The result is stored in the accumulator register.

The next instruction is **36AE**. It is a move instruction. The content of the accumulator registers is stored in the memory cell with address 6AE.

The next instruction is **F300**. This instruction clears the carry register C (Table 102).

*Table 102. CLC instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Clear carry register* | *F300* | *CLC* | *0 -> C* |

The next instruction is F600. This instruction does the left cyclic shift of the value of the carry flag and the accumulator register by 1 bit. The cyclic shift means that the most significant bit is not lost and goes to the place of the carry flag «C». The value of the carry flag goes to the place of 0 (zero) bit of the accumulator as depictured below (figure 52 and Table 103).

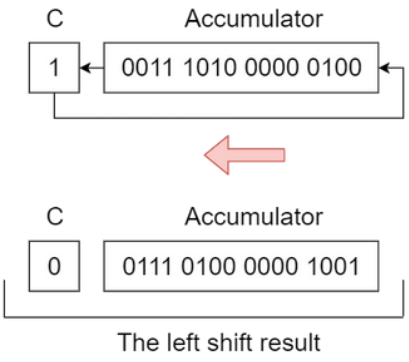



*Figure 52. The scheme of ROL instruction execution*

*Table 103. ROL instruction description*

| ***Name*** | ***Mnemonic code*** | ***Machine code*** | ***Description*** |
|---|---|---|---|
| *Left cyclic shift by 1 bit* | *ROL* | *F600* | *A & C content moves left, A(15) -> C, C -> A(0)* |

The next instruction is **66B0**. It is Subtraction. The content of the memory cell with address 6B0 is subtracted from the accumulator register. The result is stored in the accumulator register.

The next instruction is **CEA0**. It is an unconditional branch with indirect addressing. The code EA0 (1110 1010 0000) has 1 in the most significant bit. Thus, after performing this instruction the Basic Computer jumps to the address, that was stored in 6A0. It changes the content of the instruction pointer (IP) to the content of 6A0 memory cell. This cell stores the return address of the subprogram.

The next instruction is **F200**. It clears the accumulator register.

The next instruction is **46AF**. It is addition. The content of the memory cell with address 6AF is added to the accumulator register. The result is stored in the accumulator register.

The next instruction is **CEA0**. It is an unconditional branch with indirect addressing again.

The rest memory cells are data cells because the program does not use it as an instruction. The value of these cells is used as operands in the arithmetic instruction (addition and subtraction).

Summing up, the correct answer of this part of the task is presented in Table 104.

*Table 104. The answer for instruction 2 of laboratory work 4*

| ADDRESS | MACHINE CODE | MNEMONIC CODE | TYPE OF THE CODE POINTS |
|---|---|---|---|
| 5DB | F200 | CLA | EPMP |
| 5DC | 35F0 | MOV 5F0 | |
| 5DD | 45ED | ADD 5ED | |
| 5DE | 26A0 | JSR 6A0 | |
| 5DF | 65F0 | SUB 5F0 | |
| 5E0 | 35F0 | MOV 5F0 | |
| 5E1 | F200 | CLA | |
| 5E2 | 45EE | ADD 5EE | |
| 5E3 | F900 | DEC | |
| 5E4 | 26A0 | JSR 6A0 | |
| 5E5 | 45F0 | ADD 5F0 | |
| 5E6 | 35F0 | MOV 5F0 | |
| 5E7 | F200 | CLA | |
| 5E8 | 45EF | ADD 5EF | |
| 5E9 | 26A0 | JSR 6A0 | |
| 5EA | 45F0 | ADD 5F0 | |
| 5EB | 35F0 | MOV 5F0 | |
| 5EC | F000 | HLT | ENDMP |
| 5ED | 0045 | | |
| 5EE | F567 | | |
| 5EF | 0707 | | |
| 5F0 | 0000 | | |
| | | | |
| 6A0 | 0000 | | RA |
| 6A1 | A6A6 | BMI 6A6 | EPSP |
| 6A2 | 66AF | SUB 6AF | |
| 6A3 | 96A5 | BPL 6A5 | |

| | | | |
|---|---|---|---|
| 6A4 | C6AB | BR 6AB | |
| 6A5 | 46AF | ADD 6AF | |
| 6A6 | 36AE | MOV 6AE | |
| 6A7 | F300 | CLC | |
| 6A8 | F600 | ROL | |
| 6A9 | 66B0 | SUB 6B0 | |
| 6AA | CEA0 | BR EA0 | ENDSP |
| 6AB | F200 | CLA | |
| 6AC | 46AF | ADD 6AF | |
| 6AD | CEA0 | BR EA0 | ENDSP |
| 6AE | 0000 | | |
| 6AF | 00AC | | |
| 6B0 | 0081 | | |

### *3. Draw the flow chart of your program.*

The flow chart helps to understand the algorithm that was realized in the program. It also helps to understand the control flow of the program.

To draw the flow chart in this lab we need four figures.

The first is an ellipse, it is presented below. It is used to define the start and the end points of a program (figure 53).

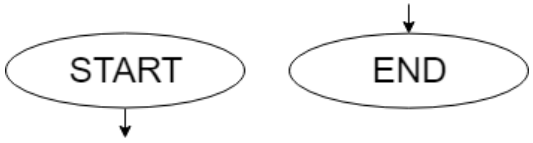



*Figure 53. The start and end points of the program*

The next figure is a rectangle. It is used to represent actions in a program. For example, the first instruction in our program is F200 (CLA). It sets the accumulator value as zero. We can draw this action in the flow chart as presented in figure 54.

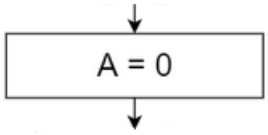



*Figure 54. The CLA instruction in the flow chart*

The next necessary figure is a rhombus. It is used to represent branches. For example, the first branch in our program is BMI 6A6 (A6A6) in the address 6A1. It checks the content of the accumulator. We can show it in the flow chart as presented in figure 55. The rhombus has two branches that show what the program does if the condition is TRUE or FALSE. The condition is written in the center of the rhombus.

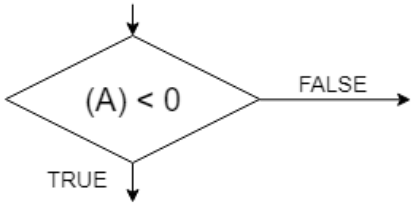



*Figure 55. The BMI instruction in the flow chart*

The last necessary figure is presented in figure 56. This figure is used to show the subprogram call by JSR instruction. The number after the «SUBPROG» is a number of the subprogram.

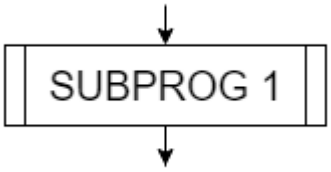



*Figure 56. The subprogram symbol in the flow chart*

To draw the flow chart for our program we need to convert the mnemonic codes of the instructions to the program actions. We will use the parentheses to show that the accumulator value or the memory cell is used in the instruction.

For example, for 45ED instruction the Basic Computer does:

$$A = (A) + (5ED),$$

i.e. the value of the memory cell 5ED is added to the accumulator. For a cyclic shift we will use such symbol as >>> (right cycle shift) and <<< (left cycle shift). For inversion, we will use the symbol «!» (exclamation mark).

We need to draw the flow chart step by step. The arrows in the flow chart show the sequence of instruction execution in the program.

The correct program chart for our program is presented in figure 57.

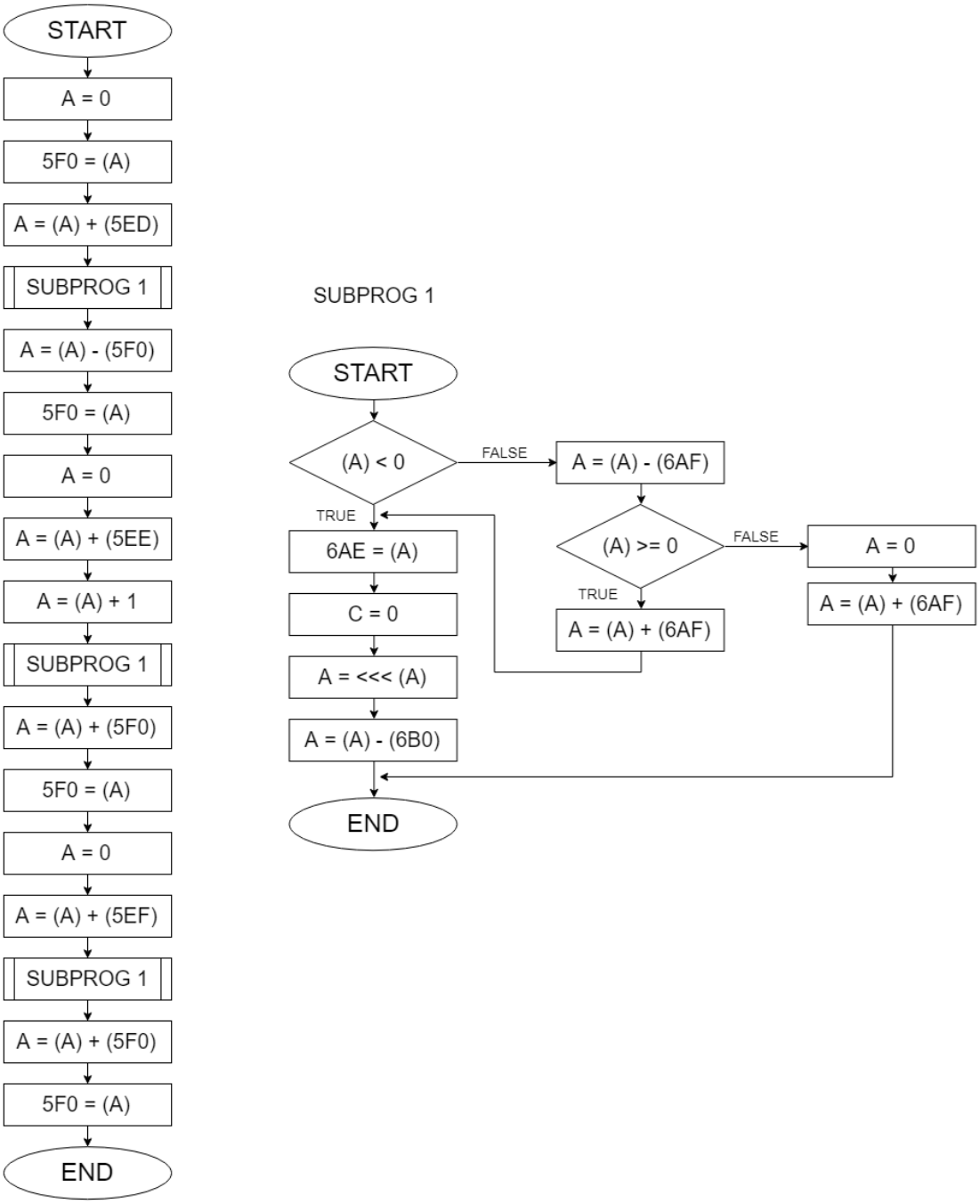



*Figure 57. The flow chart of the program*

### *4. Enter the program code into the Basic Computer Memory.*

To enter program code into the Basic Computer Memory you should open the Basic Computer Model. It can be done by using double click on the file **bcomp.jar**. After that you can see the Basic Computer processor structure (figure 58).

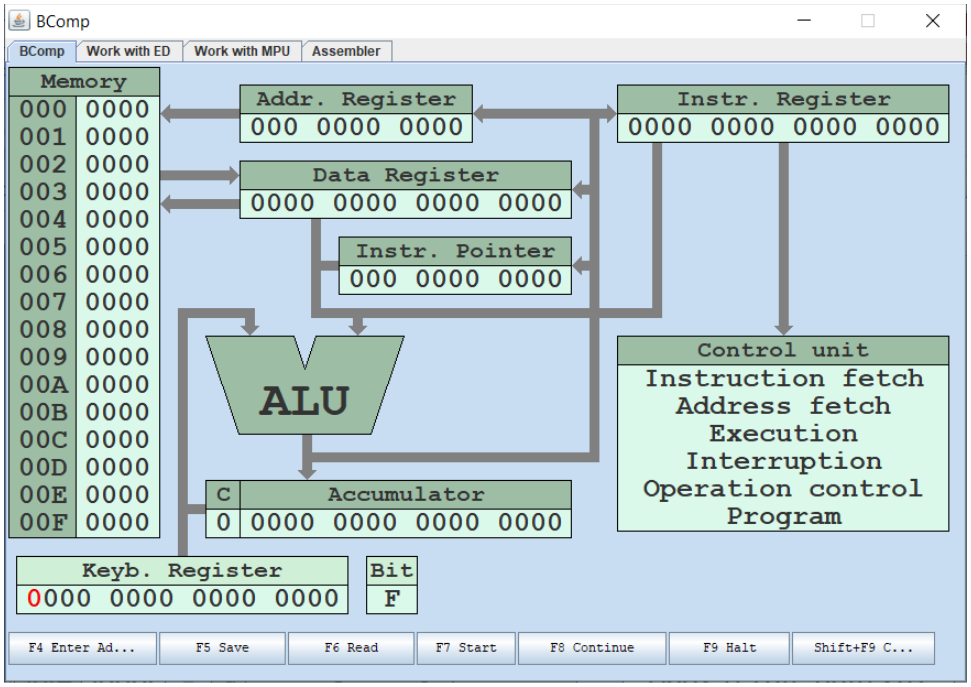



*Figure 58. The main window of Basic Computer Model*

Let's enter your program code into the Basic Computer memory. To enter every machine code into the memory you need to define the address using **F4** key and the memory cell value using **F5** key. If you enter the codes sequentially, you should only define the address of the first instruction. After storing the first instruction, the **Instr. Pointer** will be autoincremented and you can enter the second code immediately. For example, let's enter the first instruction **F200** in address **5DB**:

Use **Up** arrow on your keyboard to change the value of each bit in **Keyb. Register**. Use arrows **Left** and **Right** to move from one bit to another. Set the address of the first command in the **Keyb. Register** (figure 59). In our sample, the address of the first command is 5DB (0000 0101 1101 1011 in a binary system). The Keyb. Register is used as the intermediate register to input data to the Basic Computer Memory. It looks like a simple user console.

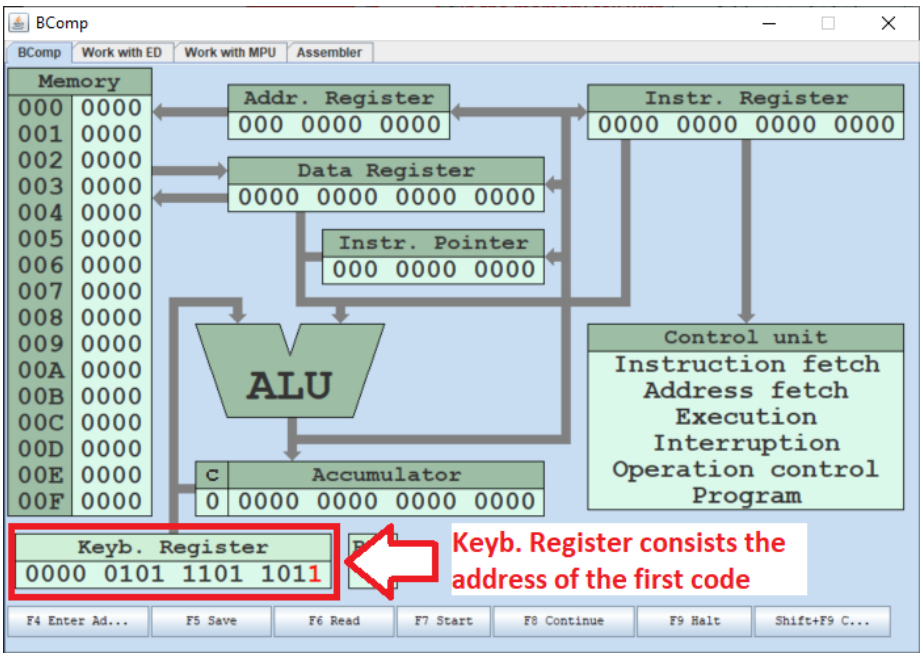



*Figure 59. The first code address in Keyb. Register*

Press **F4** to store this address in **Instr. Pointer** (figure 60).

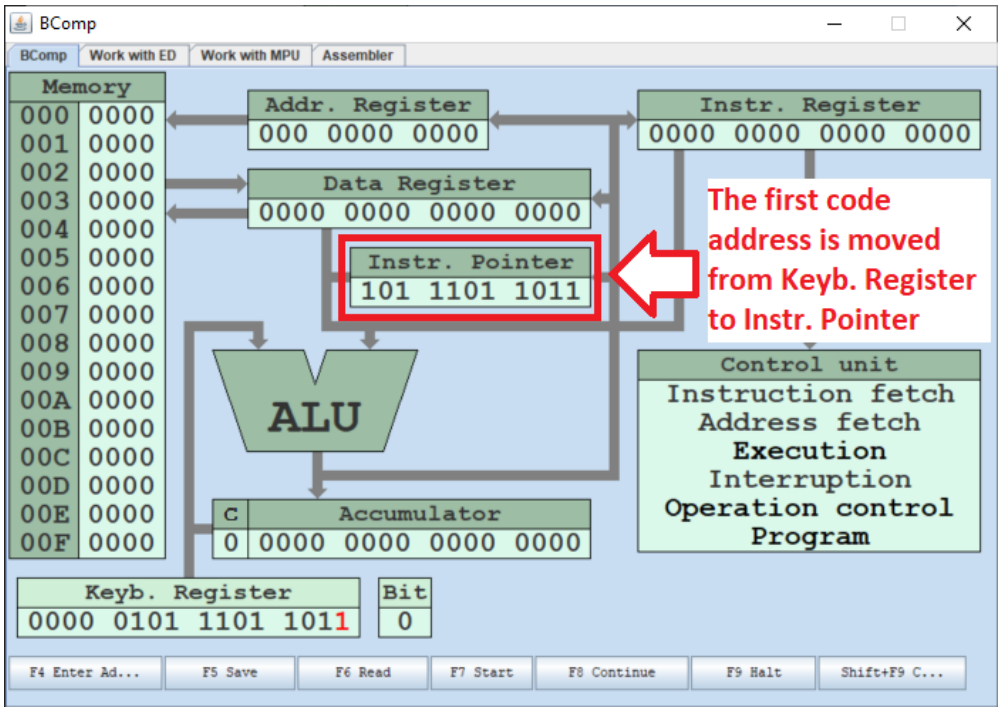



*Figure 60. The first code address in Instr. Pointer*

Enter the value of machine code **F200** (1111 0010 0000 0000 in binary system) into **Keyb. Register** (figure 61).

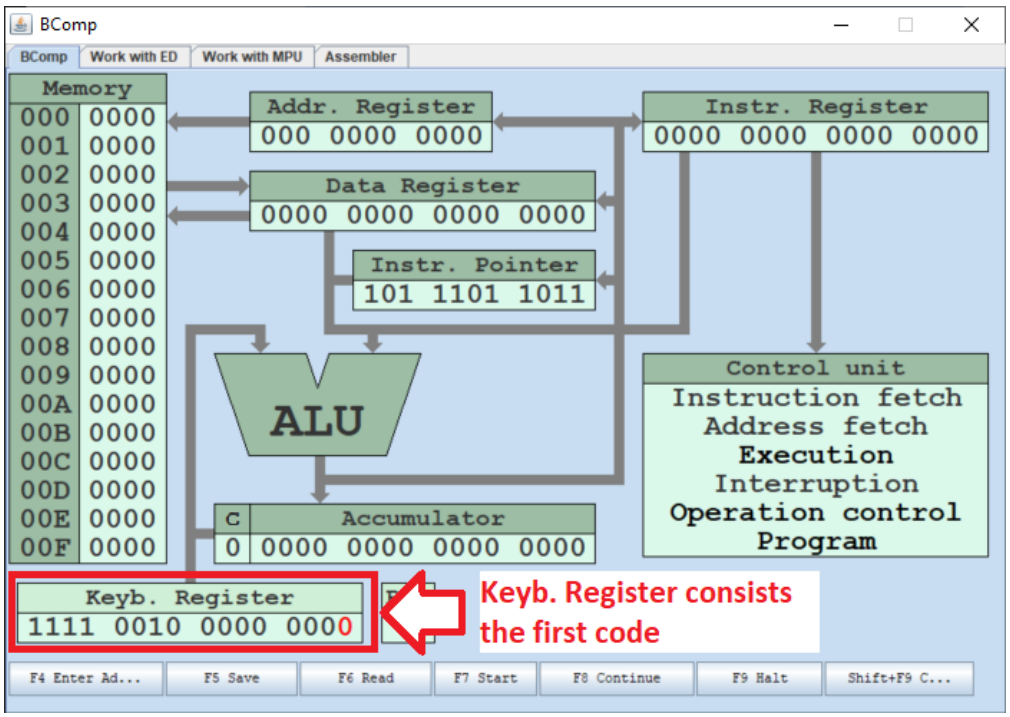



*Figure 61. The first code in Keyb. Register*

Press **F5** to store the value of **Keyb. Register** in the memory in the cell with 5DB address (figure 62).

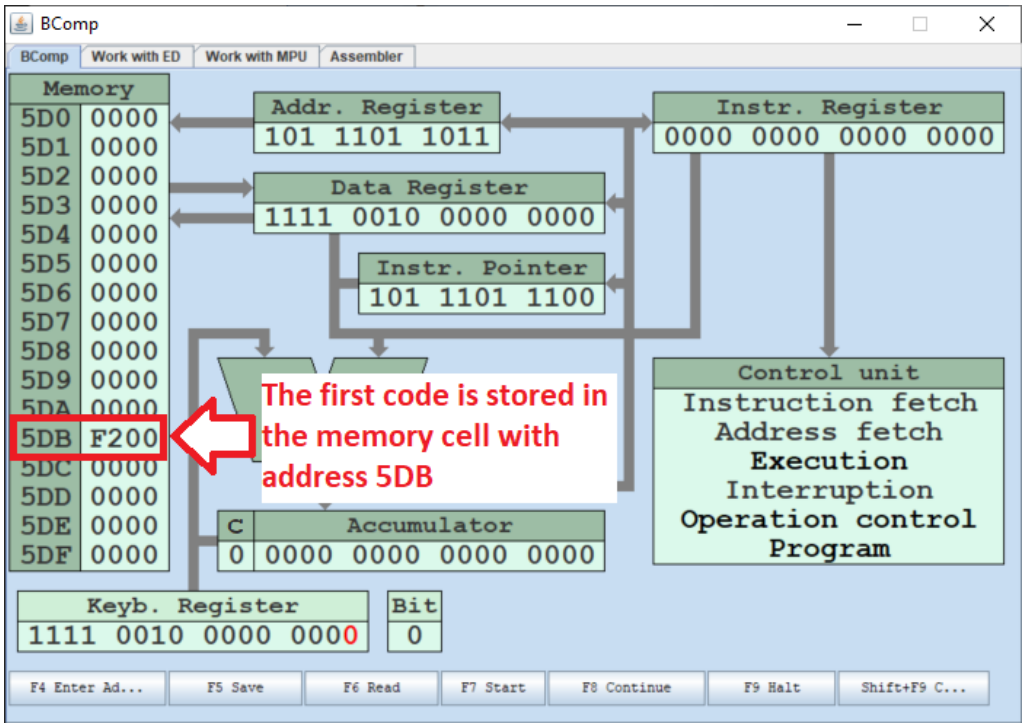



*Figure 62. The first code in the memory*

Then enter the rest instructions of your program. For the second instruction and other ones you do not need to enter the instruction address, as the content of Instr. Pointer register will be autoincremented after pressing **F5**.

### 5. *Execute your program in Basic Computer Model*

Before starting the execution, you need to define the first program address, i.e. the program entry point. Enter the first address **5DB** of your program in **Instr. Pointer** using Keyb. Register and **F4** as before.

Then make sure that the mode of program execution is **Halt**, as presented in figure 63.

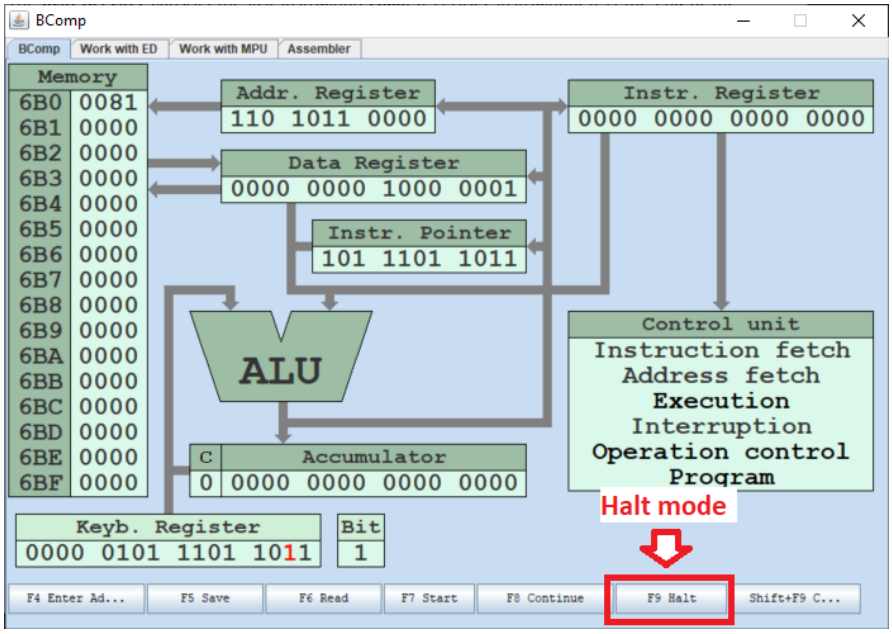



*Figure 63. The halt mode button*

If it does not work, you should press key **F9** to change the mode. The **Halt** mode allows executing program sequentially – step by step.

Press **F7** to start execution. Then press **F8** to continue execution till the last instruction **F000** with address **5EC**. During execution you need **to count the number of memory accesses**. Memory access can be a reading instruction from the memory or reading and writing data into memory.

If you have done all instructions correctly, you will see the window presented in figure 64.

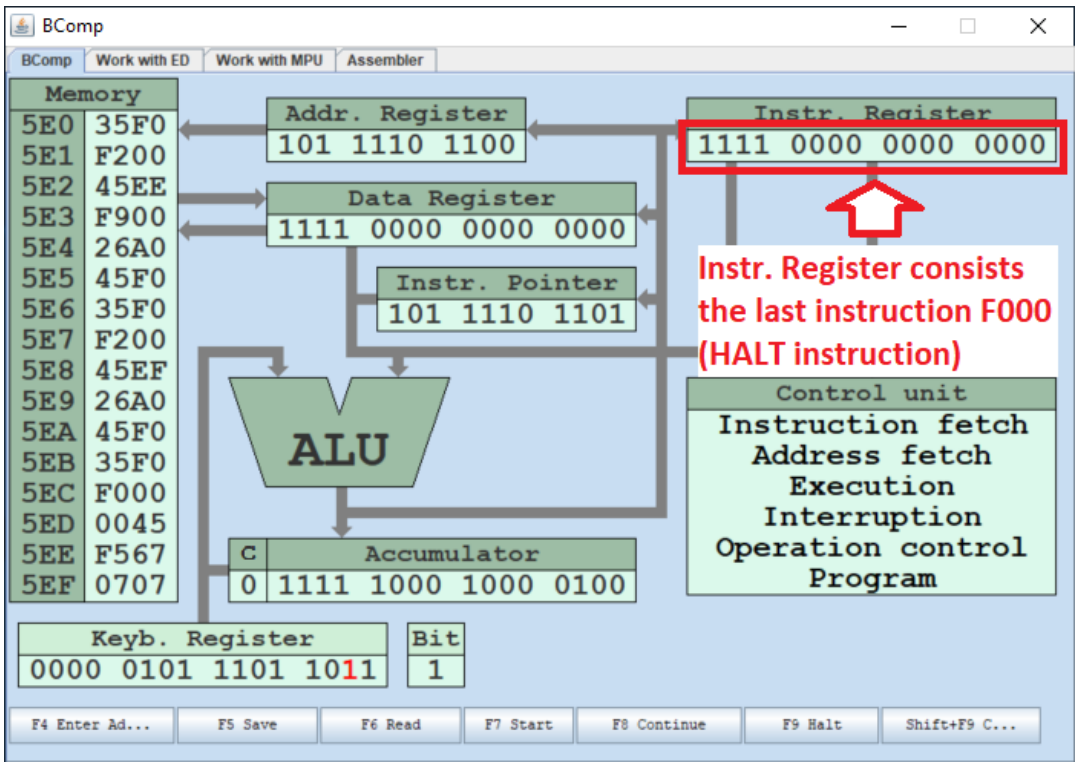



*Figure 64. The HALT instruction in Instr. Register*

The **Inst. Register** contains the last instruction **F000**. It is HALT instruction. It is the end of the program.

On the left side of the window, we can see the **Memory** content. We need to use this information in the next step of our task.

6. ***Fill in the table with the content of the memory after the program execution. Please write down the number of memory accesses for each memory cell.***

After program execution we can see only a part of memory content on the left side of the Basic Computer window. To see the top part of the memory we can enter the address of the data cell into Keyb. Register and press **F4** and then press **F6** to read from this address. After this, you can see the top part of the memory content.

If you have entered the program correctly the Basic Computer memory will have the content presented in Table 105. In the table you can see also the number of memory accesses.

*Table 105. The answer for instruction 6 of laboratory work 4*

| ADDRESS | MACHINE CODE | NUMBER OF MEMORY ACCESSES |
|---|---|---|
| 5DB | F200 | 1 |
| 5DC | 35F0 | 1 |
| 5DD | 45ED | 1 |
| 5DE | 26A0 | 1 |
| 5DF | 65F0 | 1 |
| 5E0 | 35F0 | 1 |
| 5E1 | F200 | 1 |
| 5E2 | 45EE | 1 |
| 5E3 | F900 | 1 |
| 5E4 | 26A0 | 1 |
| 5E5 | 45F0 | 1 |
| 5E6 | 35F0 | 1 |
| 5E7 | F200 | 1 |
| 5E8 | 45EF | 1 |
| 5E9 | 26A0 | 1 |
| 5EA | 45F0 | 1 |
| 5EB | 35F0 | 1 |
| 5EC | F000 | 1 |
| 5ED | 0045 | 1 |
| 5EE | F567 | 1 |
| 5EF | 0707 | 1 |
| 5F0 | F884 | 7 |
| | | |
| 6A0 | 05EA | 6 |
| 6A1 | A6A6 | 3 |
| 6A2 | 66AF | 2 |
| 6A3 | 96A5 | 2 |
| 6A4 | C6AB | 1 |
| 6A5 | 46AF | 1 |
| 6A6 | 36AE | 2 |
| 6A7 | F300 | 2 |
| 6A8 | F600 | 2 |
| 6A9 | 66B0 | 2 |
| 6AA | CEA0 | 2 |
| 6AB | F200 | 1 |
| 6AC | 46AF | 1 |
| 6AD | CEA0 | 1 |
| 6AE | 0707 | 2 |
| 6AF | 00AC | 4 |
| 6B0 | 0081 | 2 |

### 2.4.4 Lab work variants

Variant 1 of laboratory work 4 is presented in Table 106.

*Table 106. Variant 1 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 5AB | F200 | + |
| 5AC | 35C2 | |
| 5AD | 45BF | |
| 5AE | 26CB | |
| 5AF | F800 | |
| 5B0 | 45C2 | |
| 5B1 | 35C2 | |
| 5B2 | F200 | |
| 5B3 | 45C1 | |
| 5B4 | 26CB | |
| 5B5 | F900 | |
| 5B6 | 65C2 | |
| 5B7 | 35C2 | |
| 5B8 | F200 | |
| 5B9 | 45C0 | |
| 5BA | 26CB | |
| 5BB | F900 | |
| 5BC | 45C2 | |
| 5BD | 35C2 | |
| 5BE | F000 | |
| 5BF | 120F | |
| 5C0 | 0056 | |
| 5C1 | 0089 | |
| 5C2 | 0000 | |
| | | |
| 6CB | 0000 | |
| 6CC | A6D1 | |
| 6CD | 66DA | |
| 6CE | 96D0 | |
| 6CF | C6D6 | |
| 6D0 | 46DA | |
| 6D1 | 36D9 | |
| 6D2 | F300 | |
| 6D3 | F600 | |
| 6D4 | 66DB | |
| 6D5 | CECB | |
| 6D6 | F200 | |
| 6D7 | 46DA | |
| 6D8 | CECB | |
| 6D9 | 0000 | |
| 6DA | 0F38 | |
| 6DB | 00A3 | |

Variant 2 of laboratory work 4 is presented in Table 107.

*Table 107. Variant 2 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 27B | F200 | + |
| 27C | 328F | |
| 27D | 428E | |
| 27E | 2749 | |
| 27F | 428F | |
| 280 | 328F | |
| 281 | F200 | |
| 282 | 428C | |
| 283 | 2749 | |
| 284 | 628F | |
| 285 | 328F | |
| 286 | F200 | |
| 287 | 428D | |
| 288 | 2749 | |
| 289 | 428F | |
| 28A | 328F | |
| 28B | F000 | |
| 28C | 0009 | |
| 28D | 0035 | |
| 28E | F631 | |
| 28F | 0000 | |
| | | |
| 749 | 0000 | |
| 74A | A754 | |
| 74B | B754 | |
| 74C | 6758 | |
| 74D | 9754 | |
| 74E | 4758 | |
| 74F | 3757 | |
| 750 | F300 | |
| 751 | F600 | |
| 752 | 4759 | |
| 753 | CF49 | |
| 754 | F200 | |
| 755 | 4758 | |
| 756 | CF49 | |
| 757 | 0000 | |
| 758 | 075D | |
| 759 | 0023 | |

Variant 3 of laboratory work 4 is presented in Table 108.

*Table 108. Variant 3 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 24B | F200 | + |
| 24C | 3261 | |
| 24D | 425E | |
| 24E | F800 | |
| 24F | 2674 | |
| 250 | 6261 | |
| 251 | 3261 | |
| 252 | F200 | |
| 253 | 425F | |
| 254 | F800 | |
| 255 | 2674 | |
| 256 | 4261 | |
| 257 | 3261 | |
| 258 | F200 | |
| 259 | 4260 | |
| 25A | 2674 | |
| 25B | 4261 | |
| 25C | 3261 | |
| 25D | F000 | |
| 25E | 0001 | |
| 25F | 0026 | |
| 260 | 3024 | |
| 261 | 0000 | |
| | | |
| 674 | 0000 | |
| 675 | B677 | |
| 676 | 967B | |
| 677 | 6684 | |
| 678 | A67A | |
| 679 | C680 | |
| 67A | 4684 | |
| 67B | 3683 | |
| 67C | F300 | |
| 67D | F600 | |
| 67E | 6685 | |
| 67F | CE74 | |
| 680 | F200 | |
| 681 | 4684 | |
| 682 | CE74 | |
| 683 | 0000 | |
| 684 | FBFB | |
| 685 | 00CA | |

Variant 4 of laboratory work 4 is presented in Table 109.

*Table 109. Variant 4 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 31B | F200 | + |
| 31C | 3331 | |
| 31D | 432E | |
| 31E | 274D | |
| 31F | 4331 | |
| 320 | 3331 | |
| 321 | F200 | |
| 322 | 432F | |
| 323 | F800 | |
| 324 | 274D | |
| 325 | F900 | |
| 326 | 6331 | |
| 327 | 3331 | |
| 328 | F200 | |
| 329 | 4330 | |
| 32A | 274D | |
| 32B | 4331 | |
| 32C | 3331 | |
| 32D | F000 | |
| 32E | 0001 | |
| 32F | 0074 | |
| 330 | 0458 | |
| 331 | 0000 | |
| | | |
| 74D | 0000 | |
| 74E | B750 | |
| 74F | 9754 | |
| 750 | 675C | |
| 751 | A753 | |
| 752 | C758 | |
| 753 | 475C | |
| 754 | 375B | |
| 755 | 475B | |
| 756 | 675D | |
| 757 | CF4D | |
| 758 | F200 | |
| 759 | 475C | |
| 75A | CF4D | |
| 75B | 0000 | |
| 75C | F92F | |
| 75D | 0016 | |

Variant 5 of laboratory work 4 is presented in Table 110.

*Table 110. Variant 5 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 2EB | F200 | + |
| 2EC | 3302 | |
| 2ED | 4300 | |
| 2EE | 2679 | |
| 2EF | 6302 | |
| 2F0 | 3302 | |
| 2F1 | F200 | |
| 2F2 | 4301 | |
| 2F3 | 2679 | |
| 2F4 | F800 | |
| 2F5 | 6302 | |
| 2F6 | 332 | |
| 2F7 | F200 | |
| 2F8 | 42FF | |
| 2F9 | F900 | |
| 2FA | 2679 | |
| 2FB | F800 | |
| 2FC | 6302 | |
| 2FD | 3302 | |
| 2FE | F000 | |
| 2FF | 0070 | |
| 300 | 0125 | |
| 301 | F012 | |
| 302 | 0000 | |
| | | |
| 679 | 0000 | |
| 67A | A684 | |
| 67B | B684 | |
| 67C | 6688 | |
| 67D | 9684 | |
| 67E | 4688 | |
| 67F | 3687 | |
| 680 | 4687 | |
| 681 | 4687 | |
| 682 | 4689 | |
| 683 | CE79 | |
| 684 | F200 | |
| 685 | 4688 | |
| 686 | CE79 | |
| 687 | 0000 | |
| 688 | 00CA | |
| 689 | 00F2 | |

Variant 6 of laboratory work 4 is presented in Table 111.

*Table 111. Variant 6 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 57B | F200 | + |
| 57C | 3592 | |
| 57D | 4590 | |
| 57E | 26F6 | |
| 57F | F800 | |
| 580 | 4592 | |
| 581 | 3592 | |
| 582 | F200 | |
| 583 | 4591 | |
| 584 | F800 | |
| 585 | 26F6 | |
| 586 | F900 | |
| 587 | 4592 | |
| 588 | 3592 | |
| 589 | F200 | |
| 58A | 458F | |
| 58B | 26F6 | |
| 58C | 6592 | |
| 58D | 3592 | |
| 58E | F000 | |
| 58F | 0005 | |
| 590 | 0756 | |
| 591 | 8912 | |
| 592 | 0000 | |
| | | |
| 6F6 | 0000 | |
| 6F7 | A6FD | |
| 6F8 | B6FD | |
| 6F9 | 6705 | |
| 6FA | 96FC | |
| 6FB | C701 | |
| 6FC | 4705 | |
| 6FD | 3704 | |
| 6FE | 4704 | |
| 6FF | 6706 | |
| 700 | CEF6 | |
| 701 | F200 | |
| 702 | 4705 | |
| 703 | CEF6 | |
| 704 | 0000 | |
| 705 | 0A70 | |
| 706 | 002D | |

Variant 7 of laboratory work 4 is presented in Table 112.

*Table 112. Variant 7 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 54B | F200 | + |
| 54C | 3561 | |
| 54D | 455F | |
| 54E | 2722 | |
| 54F | F900 | |
| 550 | 4561 | |
| 551 | 3561 | |
| 552 | F200 | |
| 553 | 4560 | |
| 554 | 2722 | |
| 555 | 6561 | |
| 556 | 3561 | |
| 557 | F200 | |
| 558 | 455E | |
| 559 | F900 | |
| 55A | 2722 | |
| 55B | 6561 | |
| 55C | 3561 | |
| 55D | F000 | |
| 55E | 0001 | |
| 55F | 0020 | |
| 560 | 0560 | |
| 561 | 0000 | |
| | | |
| 722 | 0000 | |
| 723 | A729 | |
| 724 | B729 | |
| 725 | 6733 | |
| 726 | 9728 | |
| 727 | C72F | |
| 728 | 4733 | |
| 729 | 3732 | |
| 72A | F300 | |
| 72B | F600 | |
| 72C | 4732 | |
| 72D | 6734 | |
| 72E | CF22 | |
| 72F | F200 | |
| 730 | 4733 | |
| 731 | CF22 | |
| 732 | 0000 | |
| 733 | 06FF | |
| 734 | 0066 | |

Variant 8 of laboratory work 4 is presented in Table 113.

*Table 113. Variant 8 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 05A | F200 | + |
| 05B | 3072 | |
| 05C | 4071 | |
| 05D | F800 | |
| 05E | 26FB | |
| 05F | F800 | |
| 060 | 6072 | |
| 061 | 3072 | |
| 062 | F200 | |
| 063 | 4070 | |
| 064 | F900 | |
| 065 | 26FB | |
| 066 | F900 | |
| 067 | 6072 | |
| 068 | 3072 | |
| 069 | F200 | |
| 06A | 406F | |
| 06B | 26FB | |
| 06C | 6072 | |
| 06D | 3072 | |
| 06E | F000 | |
| 06F | 0004 | |
| 070 | 0023 | |
| 071 | F511 | |
| 072 | 0000 | |
| | | |
| 6FB | 0000 | |
| 6FC | 9701 | |
| 6FD | 670B | |
| 6FE | A700 | |
| 6FF | C707 | |
| 700 | 470B | |
| 701 | 370A | |
| 702 | F300 | |
| 703 | F600 | |
| 704 | 470A | |
| 705 | 470C | |
| 706 | CEFB | |
| 707 | F200 | |
| 708 | 470B | |
| 709 | CEFB | |
| 70A | 0000 | |
| 70B | F94F | |
| 70C | 0015 | |

Variant 9 of laboratory work 4 is presented in Table 114.

*Table 114. Variant 9 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 02A | F200 | + |
| 02B | 3040 | |
| 02C | 403E | |
| 02D | 2726 | |
| 02E | 6040 | |
| 02F | 3040 | |
| 030 | F200 | |
| 031 | 403D | |
| 032 | 2726 | |
| 033 | 4040 | |
| 034 | 3040 | |
| 035 | F200 | |
| 036 | 403F | |
| 037 | F900 | |
| 038 | 2726 | |
| 039 | F800 | |
| 03A | 4040 | |
| 03B | 3040 | |
| 03C | F000 | |
| 03D | F005 | |
| 03E | 1245 | |
| 03F | 0056 | |
| 040 | 0000 | |
| | | |
| 726 | 0000 | |
| 727 | B729 | |
| 728 | 972D | |
| 729 | 6736 | |
| 72A | A72C | |
| 72B | C732 | |
| 72C | 4736 | |
| 72D | 3735 | |
| 72E | F300 | |
| 72F | F600 | |
| 730 | 6737 | |
| 731 | CF26 | |
| 732 | F200 | |
| 733 | 4736 | |
| 734 | CF26 | |
| 735 | 0000 | |
| 736 | F1C8 | |
| 737 | 0090 | |

Variant 10 of laboratory work 4 is presented in Table 115.

*Table 115. Variant 10 of laboratory work 4*

| ADDRESS | MACHINE CODE | ENTRY POINT |
|---|---|---|
| 2BB | F200 | + |
| 2BC | 32D0 | |
| 2BD | 42CF | |
| 2BE | F800 | |
| 2BF | 26A4 | |
| 2C0 | 42D0 | |
| 2C1 | 32D0 | |
| 2C2 | F200 | |
| 2C3 | 42CE | |
| 2C4 | 26A4 | |
| 2C5 | 62D0 | |
| 2C6 | 32D0 | |
| 2C7 | F200 | |
| 2C8 | 42CD | |
| 2C9 | 26A4 | |
| 2CA | 42D0 | |
| 2CB | 32D0 | |
| 2CC | F000 | |
| 2CD | 0001 | |
| 2CE | 0025 | |
| 2CF | 0106 | |
| 2D0 | 0000 | |
| | | |
| 6A4 | 0000 | |
| 6A5 | 96AF | |
| 6A6 | 66B3 | |
| 6A7 | A6AF | |
| 6A8 | B6AF | |
| 6A9 | 46B3 | |
| 6AA | 36B2 | |
| 6AB | 46B2 | |
| 6AC | 46B2 | |
| 6AD | 46B4 | |
| 6AE | CEA4 | |
| 6AF | F200 | |
| 6B0 | 46B3 | |
| 6B1 | CEA4 | |
| 6B2 | 0000 | |
| 6B3 | FD5D | |
| 6B4 | 00F3 | |

# Appendix A. Instruction Set of Basic Computer

## Address Instructions

| Name | Mnemonic name | Code | Description |
|---|---|---|---|
| Conjunction | AND M | 1XXX | (M) & (A) → A |
| Move | MOV M | 3XXX | (A) →M |
| Addition | ADD M | 4XXX | (M) + (A) → A |
| Addition with carry | ADC M | 5XXX | (M) + (A) + (C) → A |
| Substraction | SUB M | 6XXX | (A) – (M) → A |
| Branch if carry is set | BCS M | 8XXX | If (C) = 1, then M → IP |
| Branch if plus | BPL M | 9XXX | If (N)= 0, then M → IP |
| Branch if minus | BMI M | AXXX | If (N) = 1, then M → IP |
| Branch if zero | BEQ M | BXXX | If (Z) = 1, if M → IP |
| Unconditional branch | BR M | CXXX | M → IP |
| Increment and skip | ISZ M | 0XXX | M + 1 → M, if (M) >= 0, then (IP) + 1 → IP |
| Subprogram call | JSR M | 2XXX | (IP) → M, M + 1 → IP |

## No-address Instructions

| Name | Mnemonic name | Code | Description |
|---|---|---|---|
| Clear accumulator register | CLA | F200 | 0 → A |
| Clear carry register | CLC | F300 | 0 → C |
| Invert accumulator register | CMA | F400 | (!A) → A |
| Invert carry register | CMC | F500 | (!C) → C |
| Left cyclic shift by 1 bit | ROL | F600 | A & C content moves left, A(15) → C, C → A(0) |
| Right cyclic shift by 1 bit | ROR | F700 | A & C content moves right, A(0) → C, C → A(15) |
| Increment accumulator register | INC | F800 | (A) + 1 → A |
| Decrement accumulator register | DEC | F900 | (A) – 1 → A |
| Halt | HLT | F000 | |
| No operation | NOP | F100 | |
| Enable interruptions | EI | FA00 | |
| Disable interruptions | DI | FB00 | |

## I/O Instructions

| Name | Mnemonic name | Code | Description |
|---|---|---|---|
| Clear flag | CLF B | E0XX | 0 → device flag |
| Check flag | TSF B | E1XX | If (device flag B) = 1, then (IP) + 1 → IP |
| Input | IN B | E2XX | (B) → A |
| Output | OUT B | E3XX | (A) → B |

# Appendix B. Basic Computer Hot Keys

**General commands**

| | |
|---|---|
| F4 | Enter address. Move the content of Keyb. Register to Instr. Pointer |
| F5 | Write data. Data of Keyb. Register is moved to the memory cell with the address from Instr. Pointer. Instr. Pointer is incremented. |
| F6 | Read data. Move data from the memory cell with address from Instr. Pointer to Data Register. |
| F7 | Start program execution. This command clears accumulator, carry register, ready flags from peripheral devices, deny all interrupts. If the Run mode is active the program execution is started from the address that stored in Instr. Register. |
| F8 | Continue execution. If the Run mode is active the program execution will be continued. If the Halt mode is active the next instructions will be executed. |
| F9 | Change modes of program execution. There are two modes: Run and Halt. |
| F10 | Exit |

**Commands for Keyb. Register modification**

| | |
|---|---|
| RIGHT | Pressing RIGHT arrow on the keyboard moves pointer one position to the right in Keyb. Register. |
| LEFT | Pressing LEFT arrow on the keyboard moves pointer one position to the left in Keyb. Register |
| UP | Inversion of the bit value in Keyb. Register |
| 1 | Set 1 in the selected bit in Keyb. Register |
| 0 | Set 0 in the selected bit in Keyb. Register |

D.B. Afanasev
I.A. Bessmertny
S.V. Bykovskii
A.G. Ilina
S.V. Klimenkov
J.A. Koroleva

# Basic Computer
# Study guide
# Part 1
# Educational and methodological guide

Edition
ITMO University's Editorial and Publishing Department
Department head Gusarova N.F.
Order №
Edition copy
Printed on risograph

**ITMO University's**
**Editorial and Publishing Department**
197101, Saint Petersburg, 49 Kronverksky Pr.