```
$$$$$$$\                                                     $$$$$$\  $$$$$$$\  $$\   $$\ 
$$  __$$\                                                   $$  __$$\ $$  __$$\ $$ |  $$ |
$$ |  $$ | $$$$$$\   $$$$$$\  $$\   $$\  $$$$$$$\  $$$$$$$\ $$ /  \__|$$ |  $$ |$$ |  $$ |
$$$$$$$\ |$$  __$$\ $$  __$$\ $$ |  $$ |$$  _____|$$  _____|$$ |      $$$$$$$  |$$ |  $$ |
$$  __$$\ $$ /  $$ |$$ |  \__|$$ |  $$ |\$$$$$$\  \$$$$$$\  $$ |      $$  ____/ $$ |  $$ |
$$ |  $$ |$$ |  $$ |$$ |      $$ |  $$ | \____$$\  \____$$\ $$ |  $$\ $$ |      $$ |  $$ |
$$$$$$$  |\$$$$$$  |$$ |      \$$$$$$  |$$$$$$$  |$$$$$$$  |\$$$$$$  |$$ |      \$$$$$$  |
\_______/  \______/ \__|       \______/ \_______/ \_______/  \______/ \__|       \______/ 
```

®

Boruss® CPU Architecture Software Developer’s Manual

Volume:
1

NOTE: This document contains Boruss® CPU Architecture Software Developer's Manual.

October 2025

Notices & Disclaimers

Boruss CPU technology require use hardware

CHAPTER 1
ABOUT THIS MANUAL

The Boruss CPU Architecture Software Developer's Manual, Volume 1 describes the architecture and programming environment of Boruss® CPU architecture processor.

1.1 BORUSS® CPU PROCESSORS COVERED IN THIS MANUAL

- BorussCPU "Laibach"

1.2 OVERVIEW OF VOLUME 1: BASIC ARCHITECTURE

A description of this manual's content follows:

Chapter 1 – About This Manual. Gives an overview of the BorussCPU.

Chapter 2 - Boruss® Architecture. Introduces the Boruss CPU architecture and gives overview of the features.

Chapter 3 – Basic Execution Environment. Introduces the model of memory organization and describes the register set used by applications.

Chapter 4 – Instruction Set Summary. List all BorussCPU instructions

1.3 NOTATIONAL CONVENTION

This manual uses typical notation described below.

1.3.1 Bit and Byte Order

- **Bit order** (in byte) specifies how bits are arranged within a single byte
  - **MSB** (Most Significant Bit) leftmost bit is the most significant (bit 7)
  - **LSB** (Least Significant Bit first) leftmost bit is the least significant (bit 0)

```
MSB0:  b7 b6 b5 b4 b3 b2 b1 b0   (bits numbered 7 down to 0, left to right)
LSB0:  b0 b1 b2 b3 b4 b5 b6 b7   (bits numbered 0 up to 7, left to right)
       |------------8 bits-----------|
```

- **Byte Order** (Endianness)
  - **Big-endian:** Most significant byte first (higher address = less significant)
  - **Little-endian:** Least significant byte first (lower address = less significant)

```
Memory addresses:    0      1      2      3
Big-endian:        [0x12] [0x34] [0x56] [0x78]
Little-endian:     [0x78] [0x56] [0x34] [0x12]
```

CHAPTER 2
BorussCPU® Architecture

2.1 BORUSS® CPU ARCHITECTURE

This chapter describes all BorussCPU components

2.1.1 Arithmetic-Logic Unit

Performs arithmetic (adding, subtracting) and logical(AND, OR, XOR) operations based on operation code received from CPU.

```
             +------------------+
             |                  |
    operand_a|                  | result [7:0]
    -------->|                  |---------->
             |                  |
             |    Boruss ALU    |
    operand_b|                  |
    -------->|                  |
             |                  |
             +------------------+
                  ^   | | |
                  |   | | +------------------> zero_flag
                  |   | +--------------------> carry_flag
                  |   +----------------------> negative_flag
                  |
                  |
                  |
          operation_code [7:0]
```

Figure 1 BorussCPU ALU inputs and outputs overview

2.1.1.1 ALU inputs and outputs

Name	Role	Description	Width
operand_a	Input	operand A	8 bits
operand_b	Input	operand B	8 bits
operation_code	Input	operation code	8 bits

Table 1 ALU inputs overview

Name	Role	Description	Width
result	Output	Operation result	8 bits
zero_flag	Output	Set when result is zero	1 bit
carry_flag	Output	Set when carry occurred	1 bit
negative_flag	Output	Set when result is negative (bit 7 is set to 1)	1 bit

Tabela 2 ALU outputs overview

2.1.1.2 ALU operation processing

```
                  START
                    |
                    v
Load operand_a, operand_b, operation_code
                    |
                    v
+--------------------------------+
|  CASE (operation_code):        |
|                                |
|  0x00: result = a + b          |
|        carry_flag = carry      |
|  0x01: result = a - b          |
|        carry_flag = borrow     |
|  0x02: result = a & b          |
|  0x03: result = a | b          |
|  0x04: result = a ^ b          |
|  0x05: result = ~a             |
|  0x06: result = a << 1         |
|        carry_flag = a[7]       |
|  0x07: result = a >> 1         |
|        carry_flag = a[0]       |
|  0x08-0x0E: result = b         | [JMP/JZ/JNZ/JC/JNC/JN/JP]
|  0x0F: result = a - b          | [CMP]
|        carry_flag = borrow     |
|  default: result = 0           |
+--------------------------------+
                   |
                   v
zero_flag     = (result == 0)
negative_flag = result[7]
                   |
                   v
Write: result, zero_flag, carry_flag, negative_flag
                   |
                   v
                 STOP
```

Figure 2 ALU operation processing flow chart

ALU gets the information about what operation is needed to be performed on the 8-bits wide input bus named “operation_code”. Based on that ALU selects and performs the operation and puts the outcome to the “result” 8-bits wide bus.

Operation name	Code	Used flags
ADD	0x0 (8'b00000000)	carry_flag

When operation code value is 0x0 that means the ADD operation is requested. Operands A and B are added and result can be longer (9 bits) that two input operands because the result can overflow. In that case the oldest bit (MSB) is stored in **carry_flag** and the rest written to the **result.**

Example:

operand_a = 8’11111111 (0xFF, 8`d255, 255)
operand_b = 8’00000001 (0x1, 8’d1, 1)

8’b11111111 + 8’b00000001 = 9’b**1**00000000 (0x100, 9’d256, 255 + 1 = 256)

result = 8’b00000000
carry_flag = 1’b1
zero_flag = 1’b0 (when result is zero)
negative_flag = 1’b0 (when result[7] set to 1)

Operation name	Code	Used flags
SUB	0x1 (8'b00000001)	carry_flag

When operation code value is 0x1 that means the SUB operation is requested. Operands A and B are subtracted and result can be longer (9 bits) that two input operands because the result can borrow. In that case the oldest bit (MSB) is stored in **carry_flag** and the rest written to the **result.**

Example:

operand_a = 8’00000101 (0x5, 8`d5, 5)
operand_b = 8’00001000 (0x8, 8’d8, 8)

8’00000101 - 8’00001000 = 9’b**1**11111101 (0xFFFD, -8’d3, 5 – 8 = -3)

result = 8’b11111101
carry_flag = 1’b1
zero_flag = 1’b1
negative_flag = 1’b1

Note: When **operand_a** < **operand_b** then carry_flag is set to 1.

Operation name	Code	Used flags
AND	0x2 (8'b00000010)	-

When operation code value is 0x2 that means the logical AND operation is requested.

Example:

operand_a = 8’00000101 (0x5, 8`d5, 5)
operand_b = 8’00001000 (0x8, 8’d8, 8)

8’00000101 & 8’00001000 = 8’b00000000 (0x0, 8’d0)

result = 8’b00000000
zero_flag = 1’b1
negative_flag = 1’b0

Operation name	Code	Used flags
OR	0x3 (8'b00000011)	-

When operation code value is 0x3 that means the logical OR operation is requested.

Example:

operand_a = 8’00000101 (0x5, 8`d5, 5)
operand_b = 8’00001000 (0x8, 8’d8, 8)

8’00000101 | 8’00001000 = 8’b00001101 (0xD, 8’d13)

result = 8’b00001101
zero_flag = 1’b0
negative_flag = 1’b0

Operation name	Code	Used flags
XOR	0x4 (8'b00000100)	-

When operation code value is 0x4 that means the logical XOR operation is requested.

Example:

operand_a = 8’00000101 (0x5, 8`d5, 5)
operand_b = 8’00001001 (0x9, 8’d9, 9)

8’00000101 ^ 8’00001000 = 8’b00001100 (0xD, 8’d12)

result = 8’b00001100
zero_flag = 1’b0
negative_flag = 1’b0

Operation name	Code	Used flags
NOT	0x5 (8'b00000101)	-

When operation code value is 0x5 that means the logical NOT operation is requested.

Example:

operand_a = 8’00000101 (0x5, 8`d5, 5)

~8’00000101 = 8’b11111010 (0xD, 8’d12)

result = 8’b11111010
zero_flag = 1’b0
negative_flag = 1’b0

Operation name	Code	Used flags
SHL	0x6 (8'b00000110)	carry_flag

When operation code value is 0x6 that means shift SHL left operation is requested.

Example:

operand_a = 8’00000101 (0x5, 8`d5, 5)

8’00000101<<1 = 8’b00001010 (0xA, 8’d10)

result = 8’b00001010
carry_flag = 1’b0
zero_flag = 1’b0
negative_flag = 1’b0

Operation name	Code	Used flags
SHR	0x7 (8'b00000111)	carry_flag

When operation code value is 0x6 that means shift right SHL operation is requested.

Example:

operand_a = 8’00000101 (0x5, 8`d5, 5)

8’00000101>>1 = 8’b0000010 (0xA, 8’d2)

result = 8’b00000010
carry_flag = 1’b1
zero_flag = 1’b0
negative_flag = 1’b0

Operations names	Codes	Used flags
JMP, JZ, JNZ, JC, JNC, JN, JP	0x8 -0xE (8'b00001000 – 8b’00001110)	carry_flag, zero_flag, negative_flag

When operation code value is 0x8-0xE that means jump xxx operation is requested.

Example:

operand_a - ignored
operand_b = 8’01010010 (0x50, 8`d82, 82)

Address	Instruction	Comment
0x10	JMP **0x50**	jump to address 0x50
0x11	(omitted)	this code is omitted
...
0x50 | ADD R1, R2 | Continue from here...

result = 01010010
carry_flag = 1’b0
zero_flag = 1’b0
negative_flag = 1’b0

Note: Jumps are not handled by ALU but always returns operand_b which is the jump address.

Operation name	Code	Used flags
CMP	0xF (8b’00001111)	carry_flag, zero_flag, negative_flag

When operation code value is 0xF that means CMP operation is requested.

Example:

operand_a = 8’00000101 (0x5, 8`d5, 5)
operand_b = 8’00001001 (0x9, 8’d9, 9)

operand_a - operand_b = 8’00000101 - 8’00001001 = 8b11111100

carry_flag = **1’b1**
zero_flag = 1’b0
negative_flag = 1’b1

Result: operand_a < operand_b

Flag status	Result	Condition
zero_flag = 1	operand_a == operand_b	Equal
zero_flag = 0	operand_a != operand_b	Not equal

carry_flag = 1	operand_a < operand_b	A less than B
carry_flag = 0	operand_a >= operand_b	A greater than B
negative_flag = 1	Negative value	-

Note: CMP result is not written to result register. CMP operation result can be read by flags status.