asm-simulator Documentation

Release 1.3.0

Pablo Parra

Oct 11, 2022

User documentation

A web-based simulator of a 16-bit CPU. This project has been developed for educational purposes to support the teaching of the Operating Systems course of the Degree in Computer Engineering of the University of Alcalá.

The simulator has the following features:

- A 16-bit big-endian CPU.
- Two modes of operation: supervisor & user. Each mode of operation has its own SP register.
- 4 general purpose registers, which can be accessed in word or byte modes.
- 1024 bytes of memory.
- A Memory Protection Unit (MPU).
- 16-bit input/output address map which can be accessed using IN/OUT instructions.
- An interrupt controller that supports up to 16 interrupt sources.
- A programmable 16-bit timer.
- Three input/output devices: * Visual display with a resolution of 16x16. * Textual display of 16 characters. * 10-keys numeric keypad.
- Inline memory editing.
- Execution breakpoints.

CHAPTER 1

Numbering formats

The assembler supports the following numbering formats:

- Decimal: 10, 2939d, etc.
- Octal: 0o237, 0o2332, etc.
- Binary: 0000000010001000b, 1111111101010101b, etc.
- Hexadecimal: 0x1000, 0x3FF, etc.

CHAPTER 2

Instruction Set

This section covers the complete set of instructions that are included in the simulator. Each instruction is identified by an *opcode* (operation code), a mnemonic and the type of its parameters. An instruction can have zero, one or two parameters. Two or more instructions of the same type can have the same mnemonic (e.g. `MOV`) but differ in their operation code, depending on the type of the operands that are involved. Thus, an instruction is always coded in memory as follows:

opcode	Operand 1	Operand 2
mandatory	*optional*	*optional*

The size of the operation code is always of **8 bits**, while the size of the operands can be **8 or 16 bits**, depending on their type.

2.1 Operand types

The type of operands or *addressing modes* supported by the simulator are the following. The table includes the code name of the operand type and the size of the instruction operand in memory.

Operand type	Description	Size
BYTE	8-bits immediate value	8 bits
WORD	16-bits immediate value	16 bits
ADDRESS	16-bits address	16 bits
REGISTER_8BITS	8-bits register	8 bits
REGISTER_16BITS	16-bits register	8 bits
REGADDRESS	Register addressing + offset	16 bits

The semantics of the operand types are the following:

- 8-bits immediate value: an operand of this type will define an unsigned 8-bits wide integer value.
- 16-bits immediate value: An operand of this type will define an unsigned 16-bits wide integer value.

- 16-bits address: an operand of this type will define an 16-bits memory address.
- 8-bits register: this operand will codify the reference number or index of one of the 8-bits registers that are implemented by the CPU. All the index values are expressed in decimal format:

Register Name	Description	Index
AH	MSB of Register A	9
AL	LSB of Register A	10
BH	MSB of Register B	11
BL	LSB of Register B	12
CH	MSB of Register C	13
CL	LSB of Register C	14
DH	MSB of Register D	15
DL	LSB of Register D	16

- 16-bits register: this operand will codify the reference number or index of one of the 16-bits registers that are implemented by the CPU. All the index values are expressed in decimal format:

Register Name	Description	Index
A	General Purpose Register A	0
B	General Purpose Register B	1
C	General Purpose Register C	2
D	General Purpose Register D	3
SP	Stack Pointer Register SP	4

- Register addressing + offset: this operand will codify on 1 byte the reference number of one of the 16-bits registers and, on the another byte the offset added to the value stored on the given register. The offset is codified using two's complement [-128, 127].

2.2 Instructions description

The assembler simulator supports the following instructions:

- *IN: read input/output register*
- *INC: increment 16-bits register*
- *INCB: increment 8-bits register*
- *IRET: return from ISR*
- *JA: jump if above*
- *JAE: jump if above or equal*
- *JB: jump if below*
- *JBE: jump if below or equal*
- *JC: jump if carry set*
- *JE: jump if equal*
- *JMP: jump to address*
- *JNA: jump if not above*
- *JNAE: jump if not above or equal*
- *JNB: jump if not below*
- *JNBE: jump if not below or equal*
- *JNC: jump if not carry set*
- *JNE: jump if not equal*
- *JNZ: jump if not zero*
- *JZ: jump if zero*
- *MOV: 16-bits copy*
- *MOVB: 8-bits copy*
- *MUL: 16-bits multiplication*
- *MULB: 8-bits multiplication*
- *NOT: 16-bits bitwise NOT*
- *NOTB: 8-bits bitwise NOT*
- *OR: 16-bits bitwise OR*
- *ORB: 8-bits bitwise OR*
- *OUT: write input/output register*
- *POP: pop 16-bits from stack*
- *POPB: pop 8-bits from stack*
- *PUSH: push 16-bits to stack*
- *PUSHB: push 8-bits to stack*
- *RET: return from subroutine*
- *SHL: 16-bits logical left shift*
- *SHLB: 8-bits logical left shift*
- *SHR: 16-bits logical right shift*

2.2.1 ADD: 16-bits addition

Performs an addition of two 16-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to a 16-bits register. The integer contained by the register will be added to the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

The operation will modify the values of the carry (C) and zero (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
17 (0x11)	*REGISTER_16BITS*	*REGISTER_16BITS*	`ADD A, B`
18 (0x12)	*REGISTER_16BITS*	*REGADDRESS*	`ADD C, [A-100]`
19 (0x13)	*REGISTER_16BITS*	*ADDRESS*	`ADD D, [0x1000]`
20 (0x14)	*REGISTER_16BITS*	*WORD*	`ADD B, 12345`

2.2.2 ADDB: 8-bits addition

Performs an addition of two 8-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to an 8-bits register. The integer contained by the register will be added to the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

The operation will modify the values of the carry (C) and zero (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
21 (0x15)	*REGISTER_8BITS*	*REGISTER_8BITS*	`ADDB AH, BH`
22 (0x16)	*REGISTER_8BITS*	*REGADDRESS*	`ADDB CL, [A-100]`
23 (0x17)	*REGISTER_8BITS*	*ADDRESS*	`ADDB DH, [0x100]`
24 (0x18)	*REGISTER_8BITS*	*BYTE*	`ADDB BL, 128`

2.2.3 AND: 16-bits bitwise AND

Performs a bitwise AND of two 16-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to a 16-bits register. A logic AND will be performed between the contents of the register and the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

Opcode	Operand 1	Operand 2	Example
88 (0x58)	*REGISTER_16BITS*	*REGISTER_16BITS*	`AND A, B`
89 (0x59)	*REGISTER_16BITS*	*REGADDRESS*	`AND C, [A-100]`
90 (0x5A)	*REGISTER_16BITS*	*ADDRESS*	`AND D, [0x1000]`
91 (0x5B)	*REGISTER_16BITS*	*WORD*	`AND B, 0x00FF`

2.2.4 ANDB: 8-bits bitwise AND

Performs a bitwise AND of two 8-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to an 8-bits register. A logic AND will be performed between the contents of the register and the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

Opcode	Operand 1	Operand 2	Example
92 (0x5C)	*REGISTER_8BITS*	*REGISTER_8BITS*	`ANDB AH, BL`
93 (0x5D)	*REGISTER_8BITS*	*REGADDRESS*	`ANDB CL, [A+30]`
94 (0x5E)	*REGISTER_8BITS*	*ADDRESS*	`ANDB DH, [0x30]`
95 (0x5F)	*REGISTER_8BITS*	*WORD*	`ANDB BL, 0x0F`

2.2.5 CALL: call to subroutine

Jumps to a subroutine that starts at the address referenced by Operand 1. The instruction will push to the stack the return address, i.e. the address of the instruction that follows the call.

Opcode	Operand 1	Operand 2	Example
69 (0x45)	*REGADDRESS*	*NONE*	`CALL [B-20]`
70 (0x46)	*WORD*	*NONE*	`CALL 0x1000`

2.2.6 CLI: clear interrupt mask

Clears the Interrupt Mask Bit of the Status Register. When the register is cleared, the CPU interrupts are masked and, thus, disabled. The instruction has no operands. This is a priviledged instruction that can only be called when in Supervisor mode.

Opcode	Operand 1	Operand 2	Example
130 (0x82)	*NONE*	*NONE*	`CLI`

2.2.7 CMP: 16-bits integer comparison

Compares two 16-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to a 16-bits register. The comparison will be done by substracting the value referenced by the second operand to the value contained by the register referenced by Operand 1. The result of the substraction will not be stored, but the **carry** (C) and **zero** (Z) flags of the Status Register will be modified as follows:

- Operand 1 == Operand 2 => C = 0, Z = 1
- Operand 1 > Operand 2 => C = 0, Z = 0
- Operand 1 < Operand 2 => C = 1, Z = 0

Opcode	Operand 1	Operand 2	Example
37 (0x25)	*REGISTER_16BITS*	*REGISTER_16BITS*	`CMP A, B`
38 (0x26)	*REGISTER_16BITS*	*REGADDRESS*	`CMP C, [A-100]`
39 (0x27)	*REGISTER_16BITS*	*ADDRESS*	`CMP D, [0x1000]`
40 (0x28)	*REGISTER_16BITS*	*WORD*	`CMP B, 12345`

2.2.8 CMPB: 8-bits integer comparison

Compares two 8-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to an 8-bits register. The comparison will be done by substracting the value referenced by the second operand to the value contained by the register referenced by Operand 1. The result of the substraction will not be stored, but the **carry** (C) and **zero** (Z) flags of the Status Register will be modified as follows:

- Operand 1 == Operand 2 => C = 0, Z = 1
- Operand 1 > Operand 2 => C = 0, Z = 0
- Operand 1 < Operand 2 => C = 1, Z = 0

Opcode	Operand 1	Operand 2	Example
41 (0x29)	*REGISTER_8BITS*	*REGISTER_8BITS*	`CMPB CH, CL`
42 (0x2A)	*REGISTER_8BITS*	*REGADDRESS*	`CMPB DL, [A-2]`
43 (0x2B)	*REGISTER_8BITS*	*ADDRESS*	`CMPB BH, [0x20]`
44 (0x2C)	*REGISTER_8BITS*	*BYTE*	`CMPB CH, 0x4`

2.2.9 DEC: decrement 16-bits register

Decrements the value of a 16-bits register by 1 unit. The result will be stored in the same register. The operation will modify the values of the **carry** (C) and **zero** (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
35 (0x23)	*REGISTER_16BITS*	*NONE*	`DEC B`

2.2.10 DECB: decrement 8-bits register

Decrements the value of an 8-bits register by 1 unit. The result will be stored in the same register. The operation will modify the values of the **carry** (C) and **zero** (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
36 (0x24)	*REGISTER_16BITS*	*NONE*	`DECB BL`

2.2.11 DIV: 16-bits division

Divides the value stored in Register A by the 16-bits value referred to by Operand 1. The result will be stored into Register A. The operation will modify the values of the **carry** (C) and **zero** (Z) flags of the Status Register. If the instruction executes a divison-by-zero, an exception will be triggered.

Opcode	Operand 1	Operand 2	Example
80 (0x50)	*REGISTER_16BITS*	*NONE*	`DIV B`
81 (0x51)	*REGADDRESS*	*NONE*	`DIV [A+100]`
82 (0x52)	*ADDRESS*	*NONE*	`DIV [0x1000]`
83 (0x53)	*WORD*	*NONE*	`DIV 0x2`

2.2.12 DIVB: 8-bits division

Divides the value stored in Register AL by the 8-bits value referred to by Operand 1. The result will be stored into Register AL. The operation will modify the values of the **carry** (C) and **zero** (Z) flags of the Status Register. If the instruction executes a divison-by-zero, an exception will be triggered.

Opcode	Operand 1	Operand 2	Example
84 (0x54)	*REGISTER_8BITS*	*NONE*	`DIVB BL`
85 (0x55)	*REGADDRESS*	*NONE*	`DIVB [A+100]`
86 (0x56)	*ADDRESS*	*NONE*	`DIVB [0x100]`
87 (0x57)	*BYTE*	*NONE*	`DIVB 0x2`

2.2.13 HLT: halt processor

Sets the CPU in halt mode. The **halt** (H) flag of the Status Register will be set and the processor will be stopped from executing further instructions. Interrupts can occur if they are properly enabled. If an interrupt occurs, the CPU will abandon halt mode (**halt** flag will be cleared) and the execution will resume from the instruction service routine.

Opcode	Operand 1	Operand 2	Example
0 (0x0)	*NONE*	*NONE*	`HLT`

2.2.14 IN: read input/output register

Reads the value of an input/output register. The address of the register to be read is obtained from the value of Operand 1. The result will be stored into Register A. This is a priviledged instruction that can only be called when in Supervisor mode.

Opcode	Operand 1	Operand 2	Example
135 (0x87)	*REGISTER_16BITS*	*NONE*	`IN B`
136 (0x88)	*REGADDRESS*	*NONE*	`IN [A+100]`
137 (0x89)	*ADDRESS*	*NONE*	`IN [0x1000]`
138 (0x8A)	*WORD*	*NONE*	`IN 0x2`

2.2.15 INC: increment 16-bits register

Increments the value of a 16-bits register by 1 unit. The result will be stored in the same register. The operation will modify the values of the **carry** (C) and **zero** (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
33 (0x21)	*REGISTER_16BITS*	*NONE*	`INC C`

2.2.16 INCB: increment 8-bits register

Increments the value of an 8-bits register by 1 unit. The result will be stored in the same register. The operation will modify the values of the **carry** (C) and **zero** (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
34 (0x22)	*REGISTER_16BITS*	*NONE*	`INCB DL`

2.2.17 IRET: return from ISR

Returns from an Interrupt Service Routiner (ISR). The execution of this instruction will recover the Instruction Pointer (IP), the Stack Pointer (SP) and the Status Register stored in the stack and jump to the IP address.

Opcode	Operand 1	Operand 2	Example
132 (0x84)	*NONE*	*NONE*	`IRET`

2.2.18 JA: jump if above

Jumps to a given address if the **carry** (C) and **zero** (Z) flags of the Status Register are zero (see *CMP: 16-bits integer comparison*). If the condition is met, the CPU will resume its execution from the address referenced by Operand 1. Otherwise, it will continue with the next instruction. The instruction has one mnemonic alias: `JNBE`.

Opcode	Operand 1	Operand 2	Example
55 (0x37)	*REGADDRESS*	*NONE*	`JA [C+20]`
56 (0x38)	*WORD*	*NONE*	`JA 0x1000`

2.2.19 JAE: jump if above or equal

See *JNC: jump if not carry set*.

2.2.20 JB: jump if below

See *JC: jump if carry set*.

2.2.21 JBE: jump if below or equal

See *JNA: jump if not above*.

2.2.22 JC: jump if carry set

Jumps to a given address if the **carry** (C) flag of the Status Register is set (see *CMP: 16-bits integer comparison*). If the condition is met, the CPU will resume its execution from the address referenced by Operand 1. Otherwise, it will continue with the next instruction. The instruction has two mnemonic aliases: `JBE` and `JNAE`.

Opcode	Operand 1	Operand 2	Example
47 (0x2F)	*REGADDRESS*	*NONE*	`JC [C+50]`
48 (0x30)	*WORD*	*NONE*	`JC 0x2000`

2.2.23 JE: jump if equal

See *JZ: jump if zero*.

2.2.24 JMP: jump to address

Inconditionally jumps to a given address. The CPU will resume its execution from the address referenced by Operand 1.

Opcode	Operand 1	Operand 2	Example
45 (0x2D)	*REGADDRESS*	*NONE*	`JMP [A+24]`
46 (0x2E)	*WORD*	*NONE*	`JMP 0x1200`

2.2.25 JNA: jump if not above

Jumps to a given address if the **carry** (C) or **zero** (Z) flags of the Status Register are set (see *CMP: 16-bits integer comparison*). If the condition is met, the CPU will resume its execution from the address referenced by Operand 1. Otherwise, it will continue with the next instruction. The instruction has one mnemonic alias: `JBE`.

Opcode	Operand 1	Operand 2	Example
57 (0x39)	*REGADDRESS*	*NONE*	`JNA [C+20]`
58 (0x3A)	*WORD*	*NONE*	`JNA 0x1000`

2.2.26 JNAE: jump if not above or equal

See *JC: jump if carry set*.

2.2.27 JNB: jump if not below

See *JNC: jump if not carry set*.

2.2.28 JNBE: jump if not below or equal

See *JNBE: jump if not below or equal*.

2.2.29 JNC: jump if not carry set

Jumps to a given address if the **carry** (C) flag of the Status Register is zero (see *CMP: 16-bits integer comparison*). If the condition is met, the CPU will resume its execution from the address referenced by Operand 1. Otherwise, it will continue with the next instruction. The instruction has two mnemonic aliases: `JNB` and `JAE`.

Opcode	Operand 1	Operand 2	Example
49 (0x31)	*REGADDRESS*	*NONE*	`JNC [C+2]`
50 (0x32)	*WORD*	*NONE*	`JNC 0x4000`

2.2.30 JNE: jump if not equal

See *JNZ: jump if not zero*.

2.2.31 JNZ: jump if not zero

Jumps to a given address if the **zero** (Z) flag of the Status Register is set (see *CMP: 16-bits integer comparison*). If the condition is met, the CPU will resume its execution from the address referenced by Operand 1. Otherwise, it will continue with the next instruction. The instruction has one mnemonic alias: `JNE`.

Opcode	Operand 1	Operand 2	Example
53 (0x35)	*REGADDRESS*	*NONE*	`JNZ [A+2]`
54 (0x36)	*WORD*	*NONE*	`JNZ 0x1000`

2.2.32 JZ: jump if zero

Jumps to a given address if the **zero** (Z) flag of the Status Register is zero (see *CMP: 16-bits integer comparison*). If the condition is met, the CPU will resume its execution from the address referenced by Operand 1. Otherwise, it will continue with the next instruction. The instruction has one mnemonic alias: `JE`.

Opcode	Operand 1	Operand 2	Example
51 (0x33)	*REGADDRESS*	*NONE*	`JZ [A+20]`
52 (0x34)	*WORD*	*NONE*	`JZ 0x1000`

2.2.33 MOV: 16-bits copy

Copies a 16-bits value, referenced by Operand 2, to the location referred to by Operand 1.

Opcode	Operand 1	Operand 2	Example
1 (0x01)	*REGISTER_16BITS*	*REGISTER_16BITS*	`MOV A, B`
2 (0x02)	*REGISTER_16BITS*	*REGADDRESS*	`MOV C, [A-100]`
3 (0x03)	*REGISTER_16BITS*	*ADDRESS*	`MOV D, [0x1000]`
4 (0x04)	*REGADDRESS*	*REGISTER_16BITS*	`MOV [B-2], A`
5 (0x05)	*ADDRESS*	*REGISTER_16BITS*	`MOV [0x100], D`
6 (0x06)	*REGISTER_16BITS*	*WORD*	`MOV A, 0x100`
7 (0x07)	*REGADDRESS*	*WORD*	`MOV [D-4], B`
8 (0x08)	*ADDRESS*	*WORD*	`MOV [0x200], C`

2.2.34 MOVB: 8-bits copy

Copies an 8-bits value, referenced by Operand 2, to the location referred to by Operand 1.

Opcode	Operand 1	Operand 2	Example
9 (0x09)	*REGISTER_8BITS*	*REGISTER_8BITS*	`MOVB AH, BL`
10 (0x0A)	*REGISTER_8BITS*	*REGADDRESS*	`MOVB BL, [A-100]`
11 (0x0B)	*REGISTER_8BITS*	*ADDRESS*	`MOVB DH, [0x1000]`
12 (0x0C)	*REGADDRESS*	*REGISTER_8BITS*	`MOVB [B-2], AH`
13 (0x0D)	*ADDRESS*	*REGISTER_8BITS*	`MOVB [0x100], CL`
14 (0x0E)	*REGISTER_8BITS*	*BYTE*	`MOVB AL, 0x80`
15 (0x0F)	*REGADDRESS*	*BYTE*	`MOVB [D-4], AL`
16 (0x10)	*ADDRESS*	*BYTE*	`MOVB [0x200], CH`

2.2.35 MUL: 16-bits multiplication

Multiplies the value stored in Register A by the 16-bits value referred to by Operand 1. The result will be stored into Register A. The operation will modify the values of the **carry** (C) and **zero** (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
72 (0x48)	*REGISTER_16BITS*	*NONE*	`MUL A`
73 (0x49)	*REGADDRESS*	*NONE*	`MUL [A+100]`
74 (0x4A)	*ADDRESS*	*NONE*	`MUL [0x2000]`
75 (0x4B)	*WORD*	*NONE*	`MUL 0x4`

2.2.36 MULB: 8-bits multiplication

Multiplies the value stored in Register AL by the 8-bits value referred to by Operand 1. The result will be stored into Register AL. The operation will modify the values of the **carry** (C) and **zero** (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
76 (0x4C)	*REGISTER_8BITS*	*NONE*	`MULB CL`
77 (0x4D)	*REGADDRESS*	*NONE*	`MULB [A+100]`
78 (0x4E)	*ADDRESS*	*NONE*	`MULB [0x400]`
79 (0x4F)	*BYTE*	*NONE*	`MULB 0x8`

2.2.37 NOT: 16-bits bitwise NOT

Performs a bitwise NOT on the bits of a 16-bits register, referenced by Operand 1. The result of the operation will be stored in the same register.

Opcode	Operand 1	Operand 2	Example
112 (0x70)	*REGISTER_16BITS*	*NONE*	`NOT A`

2.2.38 NOTB: 8-bits bitwise NOT

Performs a bitwise NOT on the bits of an 8-bits register, referenced by Operand 1. The result of the operation will be stored in the same register.

Opcode	Operand 1	Operand 2	Example
113 (0x71)	*REGISTER_8BITS*	*NONE*	`NOTB AL`

2.2.39 OR: 16-bits bitwise OR

Performs a bitwise OR of two 16-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to a 16-bits register. A logic OR will be performed between the contents of the register and the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

Opcode	Operand 1	Operand 2	Example
96 (0x60)	*REGISTER_16BITS*	*REGISTER_16BITS*	`OR C, B`
97 (0x61)	*REGISTER_16BITS*	*REGADDRESS*	`OR C, [B-100]`
98 (0x62)	*REGISTER_16BITS*	*ADDRESS*	`OR D, [0x1000]`
99 (0x63)	*REGISTER_16BITS*	*WORD*	`OR D, 0xA5A5`

2.2.40 ORB: 8-bits bitwise OR

Performs a bitwise OR of two 8-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to an 8-bits register. A logic OR will be performed between the contents of the register and the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

Opcode	Operand 1	Operand 2	Example
100 (0x64)	*REGISTER_8BITS*	*REGISTER_8BITS*	`ORB CH, BL`
101 (0x65)	*REGISTER_8BITS*	*REGADDRESS*	`ORB DL, [A+30]`
102 (0x66)	*REGISTER_8BITS*	*ADDRESS*	`ORB CH, [0x30]`
103 (0x67)	*REGISTER_8BITS*	*WORD*	`ORB BL, 0xA5`

2.2.41 OUT: write input/output register

Writes the contents of General Purpose Register A into an input/output register. The address of the register to be written is obtained from the value of Operand 1. This is a priviledged instruction that can only be called when in Supervisor mode.

Opcode	Operand 1	Operand 2	Example
139 (0x8B)	*REGISTER_16BITS*	*NONE*	`OUT C`
140 (0x8C)	*REGADDRESS*	*NONE*	`OUT [B+100]`
141 (0x8D)	*ADDRESS*	*NONE*	`OUT [0x1000]`
142 (0x8E)	*WORD*	*NONE*	`OUT 0x2`

2.2.42 POP: pop 16-bits from stack

Pops a 16-bits value from the top of the stack and stores it into the 16-bits register referenced by Operand 1. The instruction will update the Stack Pointer (SP), increasing it by 2 units.

Opcode	Operand 1	Operand 2	Example
67 (0x43)	*REGISTER_16BITS*	*NONE*	`POP A`

2.2.43 POPB: pop 8-bits from stack

Pops an 8-bits value from the top of the stack and stores it into the 8-bits register referenced by Operand 1. The instruction will update the Stack Pointer (SP), increasing it by 1 unit.

Opcode	Operand 1	Operand 2	Example
68 (0x44)	*REGISTER_8BITS*	*NONE*	`POPB AL`

2.2.44 PUSH: push 16-bits to stack

Pushes a 16-bits value, referenced by Operand 1, to the top of the stack. The instruction will update the Stack Pointer (SP), decreasing it by 2 units.

Opcode	Operand 1	Operand 2	Example
59 (0x3B)	*REGISTER_16BITS*	*NONE*	`PUSH C`
60 (0x3C)	*REGADDRESS*	*NONE*	`PUSH [B+100]`
61 (0x3D)	*ADDRESS*	*NONE*	`PUSH [0x1000]`
62 (0x3E)	*WORD*	*NONE*	`PUSH 0x2`

2.2.45 PUSHB: push 8-bits to stack

Pushes an 8-bits value, referenced by Operand 1, to the top of the stack. The instruction will update the Stack Pointer (SP), decreasing it by 1 units.

Opcode	Operand 1	Operand 2	Example
63 (0x3F)	*REGISTER_16BITS*	*NONE*	`PUSHB CL`
64 (0x40)	*REGADDRESS*	*NONE*	`PUSHB [B+100]`
65 (0x41)	*ADDRESS*	*NONE*	`PUSHB [0x400]`
66 (0x42)	*WORD*	*NONE*	`PUSHB 0x80`

2.2.46 RET: return from subroutine

Returns from a subroutine. The execution of this instruction will pop the Instruction Pointer (IP) stored in the stack and jump to the IP address. The instruction will update the Stack Pointer (SP).

Opcode	Operand 1	Operand 2	Example
71 (0x47)	*NONE*	*NONE*	`RET`

2.2.47 SHL: 16-bits logical left shift

Performs a logical left shift of the value of a 16-bits register. Every form of the instruction will have two operands. Operand 1 will always be a reference to a 16-bits register. Operand 2 will indicate the number of bit positions that the value shall be shifted. The result will be stored in the register referenced by Operand 1.

The operation will modify the values of the carry (C) and zero (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
114 (0x72)	*REGISTER_16BITS*	*REGISTER_16BITS*	`SHL A, B`
115 (0x73)	*REGISTER_16BITS*	*REGADDRESS*	`SHL C, [A-100]`
116 (0x74)	*REGISTER_16BITS*	*ADDRESS*	`SHL D, [0x1000]`
117 (0x75)	*REGISTER_16BITS*	*WORD*	`SHL B, 4`

2.2.48 SHLB: 8-bits logical left shift

Performs a logical left shift of the value of an 8-bits register. Every form of the instruction will have two operands. Operand 1 will always be a reference to an 8-bits register. Operand 2 will indicate the number of bit positions that the value shall be shifted. The result will be stored in the register referenced by Operand 1.

The operation will modify the values of the carry (C) and zero (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
118 (0x76)	*REGISTER_8BITS*	*REGISTER_8BITS*	`SHLB AH, BL`
119 (0x77)	*REGISTER_8BITS*	*REGADDRESS*	`SHLB CL, [A+30]`
120 (0x78)	*REGISTER_8BITS*	*ADDRESS*	`SHLB DH, [0x30]`
121 (0x79)	*REGISTER_8BITS*	*WORD*	`SHLB BL, 4`

2.2.49 SHR: 16-bits logical right shift

Performs a logical right shift of the value of a 16-bits register. Every form of the instruction will have two operands. Operand 1 will always be a reference to a 16-bits register. Operand 2 will indicate the number of bit positions that the value shall be shifted. The result will be stored in the register referenced by Operand 1.

The operation will modify the values of the carry (C) and zero (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
122 (0x7A)	*REGISTER_16BITS*	*REGISTER_16BITS*	`SHR A, B`
123 (0x7B)	*REGISTER_16BITS*	*REGADDRESS*	`SHR C, [A-100]`
124 (0x7C)	*REGISTER_16BITS*	*ADDRESS*	`SHR D, [0x1000]`
125 (0x7D)	*REGISTER_16BITS*	*WORD*	`SHR B, 4`

2.2.50 SHRB: 8-bits logical right shift

Performs a logical right shift of the value of an 8-bits register. Every form of the instruction will have two operands. Operand 1 will always be a reference to an 8-bits register. Operand 2 will indicate the number of bit positions that the value shall be shifted. The result will be stored in the register referenced by Operand 1.

The operation will modify the values of the carry (C) and zero (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
126 (0x7E)	*REGISTER_8BITS*	*REGISTER_8BITS*	`SHRB AH, BL`
127 (0x7F)	*REGISTER_8BITS*	*REGADDRESS*	`SHRB CL, [A+30]`
128 (0x80)	*REGISTER_8BITS*	*ADDRESS*	`SHRB DH, [0x30]`
129 (0x81)	*REGISTER_8BITS*	*WORD*	`SHRB BL, 4`

2.2.51 SRET: return from system call

Returns from an System Call (SVC). The execution of this instruction will recover the Instruction Pointer (IP) and the user Stack Pointer (SP) stored in the stack and jump to the IP address. This is a priviledged instruction that can only be called when in Supervisor mode. When executed, the CPU will be switched to User mode.

Opcode	Operand 1	Operand 2	Example
134 (0x86)	*NONE*	*NONE*	`SRET`

2.2.52 STI: set interrupt mask

Set the Interrupt Mask Bit of the Status Register. When the register is cleared, the CPU interrupts are unmasked and, thus, enabled. The instruction has no operands. This is a priviledged instruction that can only be called when in Supervisor mode.

Opcode	Operand 1	Operand 2	Example
129 (0x81)	*NONE*	*NONE*	`STI`

2.2.53 SUB: 16-bits substraction

Performs a substraction of two 16-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to a 16-bits register. The integer contained by the register will be substracted from the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

The operation will modify the values of the carry (C) and zero (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
25 (0x19)	*REGISTER_16BITS*	*REGISTER_16BITS*	`SUB A, B`
26 (0x1A)	*REGISTER_16BITS*	*REGADDRESS*	`SUB C, [A-100]`
27 (0x1B)	*REGISTER_16BITS*	*ADDRESS*	`SUB D, [0x1000]`
28 (0x1C)	*REGISTER_16BITS*	*WORD*	`SUB B, 12345`

2.2.54 SUBB: 8-bits substraction

Performs a substraction of two 8-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to an 8-bits register. The integer in the Operand 2 will be substracted from the value in the register specified in the Operand 1. The result will be stored in the register referenced by the Operand 1.

The operation will modify the values of the carry (C) and zero (Z) flags of the Status Register.

Opcode	Operand 1	Operand 2	Example
29 (0x1D)	*REGISTER_8BITS*	*REGISTER_8BITS*	`SUBB BH, DL`
30 (0x1E)	*REGISTER_8BITS*	*REGADDRESS*	`SUBB CH, [A-100]`
31 (0x1F)	*REGISTER_8BITS*	*ADDRESS*	`SUBB DL, [0x400]`
32 (0x20)	*REGISTER_8BITS*	*WORD*	`SUBB BL, 0x10`

2.2.55 SVC: system call

Performs a System Call (SVC). This instruction can only be executed when the CPU is in User mode. The execution of this instruction will: setup the Supervisor stack; push to it the Instruction Pointer (IP) and the user Stack Pointer (SP); switch the CPU to Supervisor mode; and jump to address 0x0006.

Opcode	Operand 1	Operand 2	Example
133 (0x85)	*NONE*	*NONE*	`SVC`

2.2.56 XOR: 16-bits bitwise XOR

Performs a bitwise XOR of two 16-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to a 16-bits register. A logic XOR will be performed between the contents of the register and the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

Opcode	Operand 1	Operand 2	Example
104 (0x68)	*REGISTER_16BITS*	*REGISTER_16BITS*	`XOR C, B`
105 (0x69)	*REGISTER_16BITS*	*REGADDRESS*	`XOR C, [B-100]`
106 (0x6A)	*REGISTER_16BITS*	*ADDRESS*	`XOR D, [0x400]`
107 (0x6B)	*REGISTER_16BITS*	*WORD*	`XOR D, 0xA5A5`

2.2.57 XORB: 8-bits bitwise XOR

Performs a bitwise XOR of two 8-bits integers. Every form of the instruction will have two operands. Operand 1 will always be a reference to an 8-bits register. A logic XOR will be performed between the contents of the register and the value referenced by Operand 2. The result will be stored in the register referenced by Operand 1.

Opcode	Operand 1	Operand 2	Example
108 (0x6C)	*REGISTER_8BITS*	*REGISTER_8BITS*	`XORB CH, BL`
109 (0x6D)	*REGISTER_8BITS*	*REGADDRESS*	`XORB DL, [A+30]`
110 (0x6E)	*REGISTER_8BITS*	*ADDRESS*	`XORB CH, [0x30]`
111 (0x6F)	*REGISTER_8BITS*	*WORD*	`XORB BL, 0xA5`

CHAPTER 3

Directives

This section covers the directives supported by the simulator. Specifically, the simulator supports the following directives:

- *DB: Define Byte*
- *DW: Define Word*
- *ORG: Advance Program Counter*
- *EQU: Define symbolic name*

3.1 DB: Define Byte

This directive allows you to reserve memory space for *one byte* (8 bits). The receive directive receives two operands. Operand 1 sets the name or identifier assigned to the space. Operand 2 sets the initialization value of the reserved space. This operand can be a numeric value, a character or an array of characters. In the case of a single character, the initial value stored will correspond to the ASCII code of the character encoded in single quotes (`''`). In the case of an array, the values corresponding to the ASCII codes of those encoded between double quotes (`""`` `) will be encoded. If you want to encode a numeric value explicitly within the array, you must do it using the escape code `\x`.

Directive	Result
`DB var1, 0x01`	Reserves one byte with an initial value of 0x01
`DB char1, '1'`	Reserves one byte with an initial value of 0x31
`DB string, "Hello"`	Reserves five bytes with values {0x48, 0x65, 0x6C, 0x6C, 0x6F}
`DB string, "\x01\x02\x03\x04"`	Reserves four bytes with values {0x01, 0x02, 0x03, 0x04}

3.2 DW: Define Word

This directive allows you to reserve memory space for *two bytes* (16 bits). The receive directive receives two operands. Operand 1 sets the name or identifier assigned to the space. Operand 2 sets the initialization value of the reserved space.

This operand can only be of numeric type.

Directive	Result
`DW var1, 2048`	Reserves two bytes with values {0x80, 0x00}
`DW var2, 0x1FFF`	Reserves two bytes with values {0x1F, 0xFF}

3.3 ORG: Advance Program Counter

This directive allows moving the program counter to a given address. The directive contains a single operand that sets the address from which subsequent instructions are to be encoded.

Directive	Result
`ORG 0x100`	The subsequent instructions will be encoded starting from address 0x100

3.4 EQU: Define symbolic name

The EQU directive gives a symbolic name or tag to an expression. This tag can be later used in subsequent lines of assembly code and will be substituted by the value given. The directive receives two operands. The first sets the label or symbolic name. The second defines the expression by which occurrences of the label will be replaced in the code. These occurrences will only be evaluated in lines of code following the definition of the label.

Directive	Result
`EOS EQU 255`	Any occurrence of `EOS` will be sustitued by `255`
`TMRPRELOAD EQU 0x0003`	Any occurrence of `TMRPRELOAD` will be sustitued by `0x0003`