To Design 16-Bit ALU

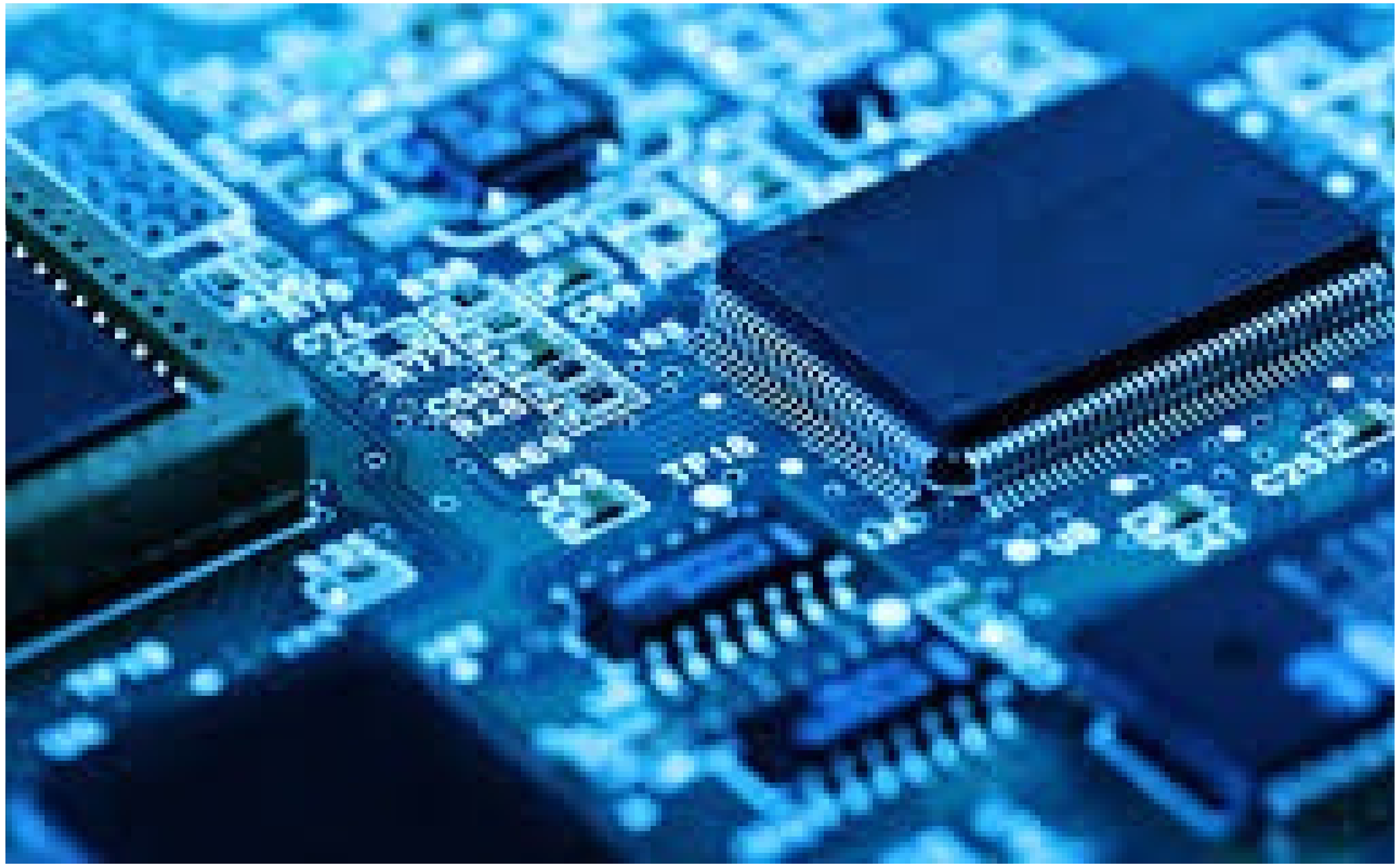
Using 8086 Emulator

CONTENTS

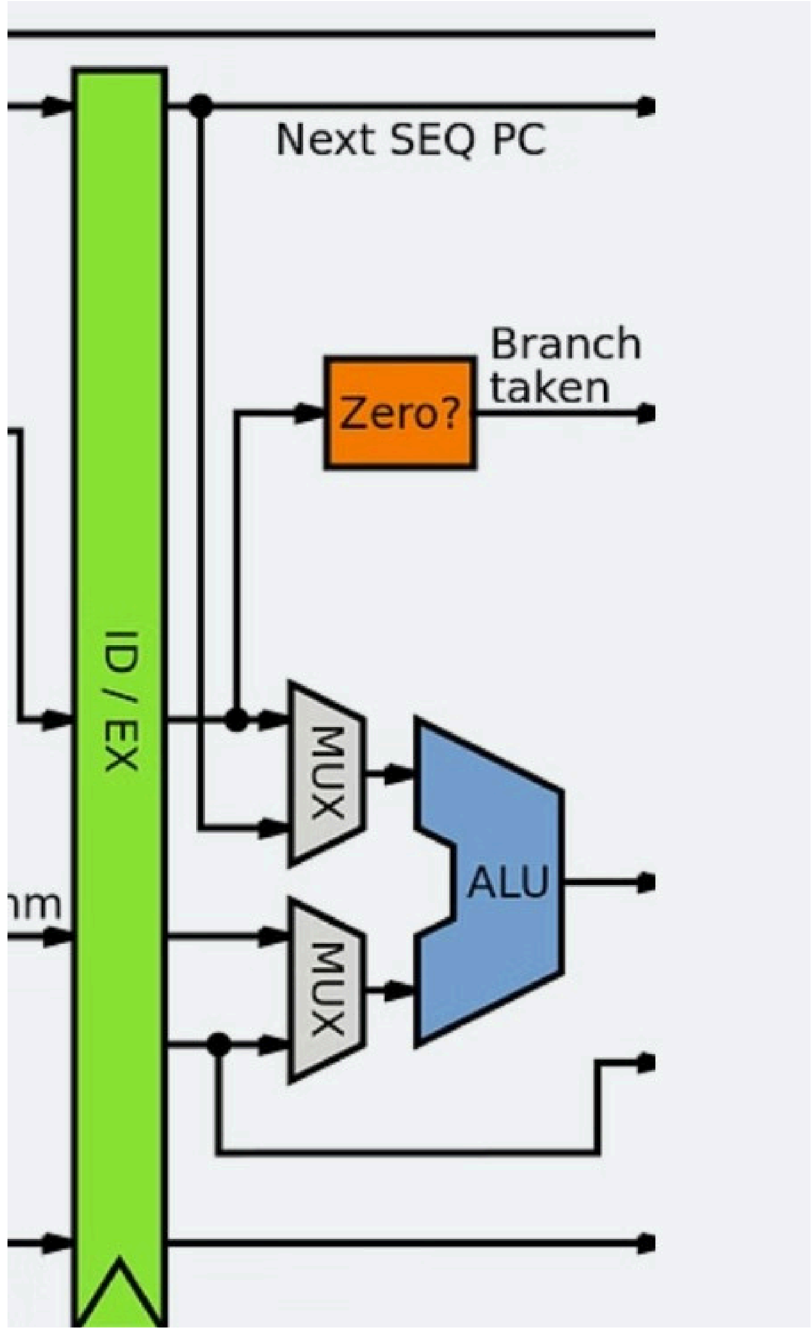
- Introduction
- Architectural Design
- Functional Units & Operations
- Flowchart
- Interface
- Limitations

INTRODUCTION

The 16-bit Arithmetic Logic Unit (ALU) is a fundamental building block of modern computer systems. It performs essential arithmetic and logical operations that power a wide range of applications, from embedded systems to high-performance computing.



Architectural Design of the 16-bit ALU



1.Register File

The register file stores the operands and results of ALU operations, providing fast access to data.

2.Arithmetic Unit

The arithmetic unit performs basic operations like addition, subtraction and multiplication on 16-bit data.

3.Logic Unit

The logic unit handles bitwise operations such as AND, OR, and XOR enabling complex logical computations. bit of body text.

FUNCTIONAL UNITS AND OPERATIONS

Arithmetic Operations:

The ALU supports standard arithmetic operations like addition, subtraction, multiplication, and division.

Shift Operations

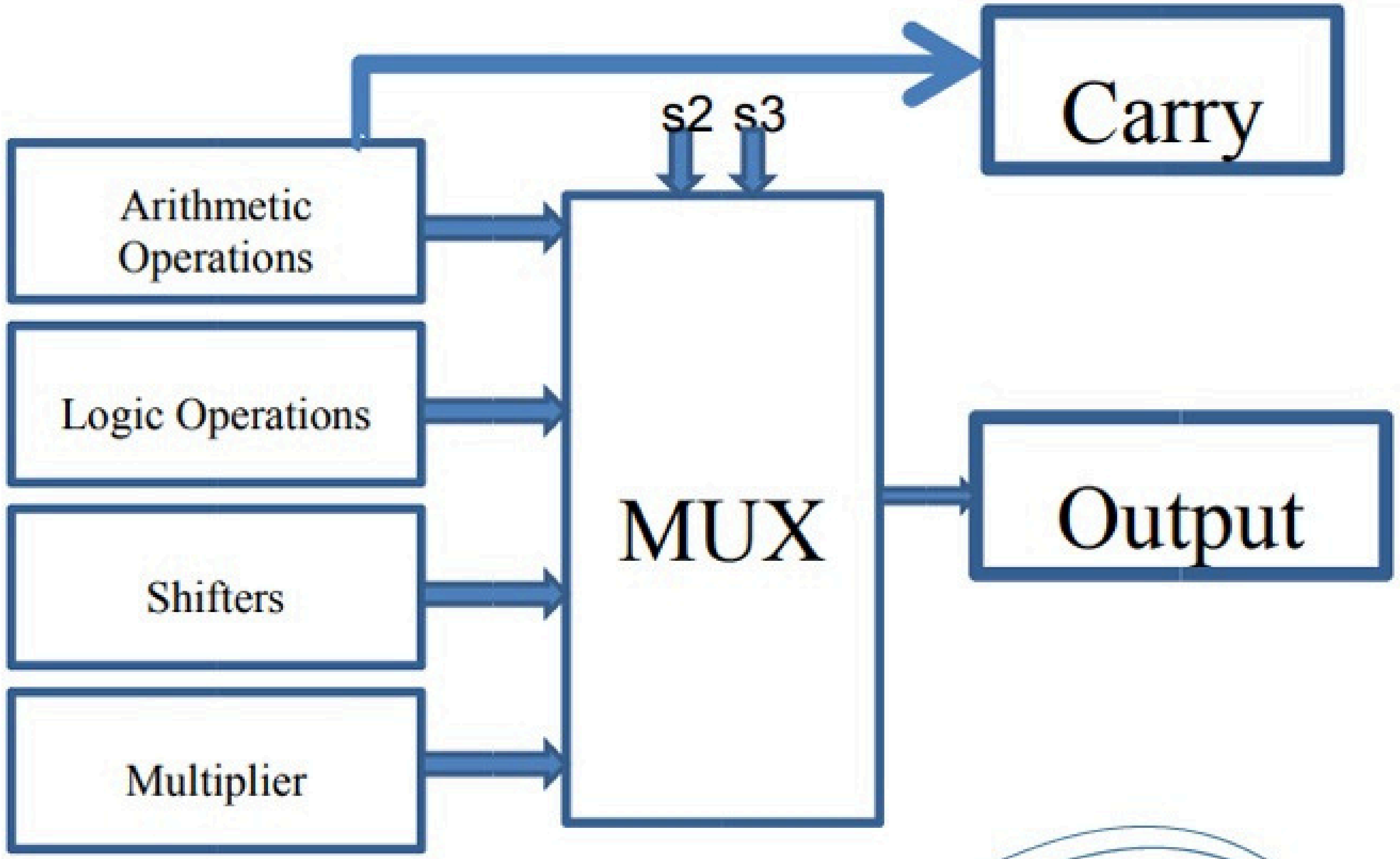
Logical and arithmetic shift operations enable efficient bit manipulation and number scaling.

Comparison Operations. The ALU can perform various comparison operations,

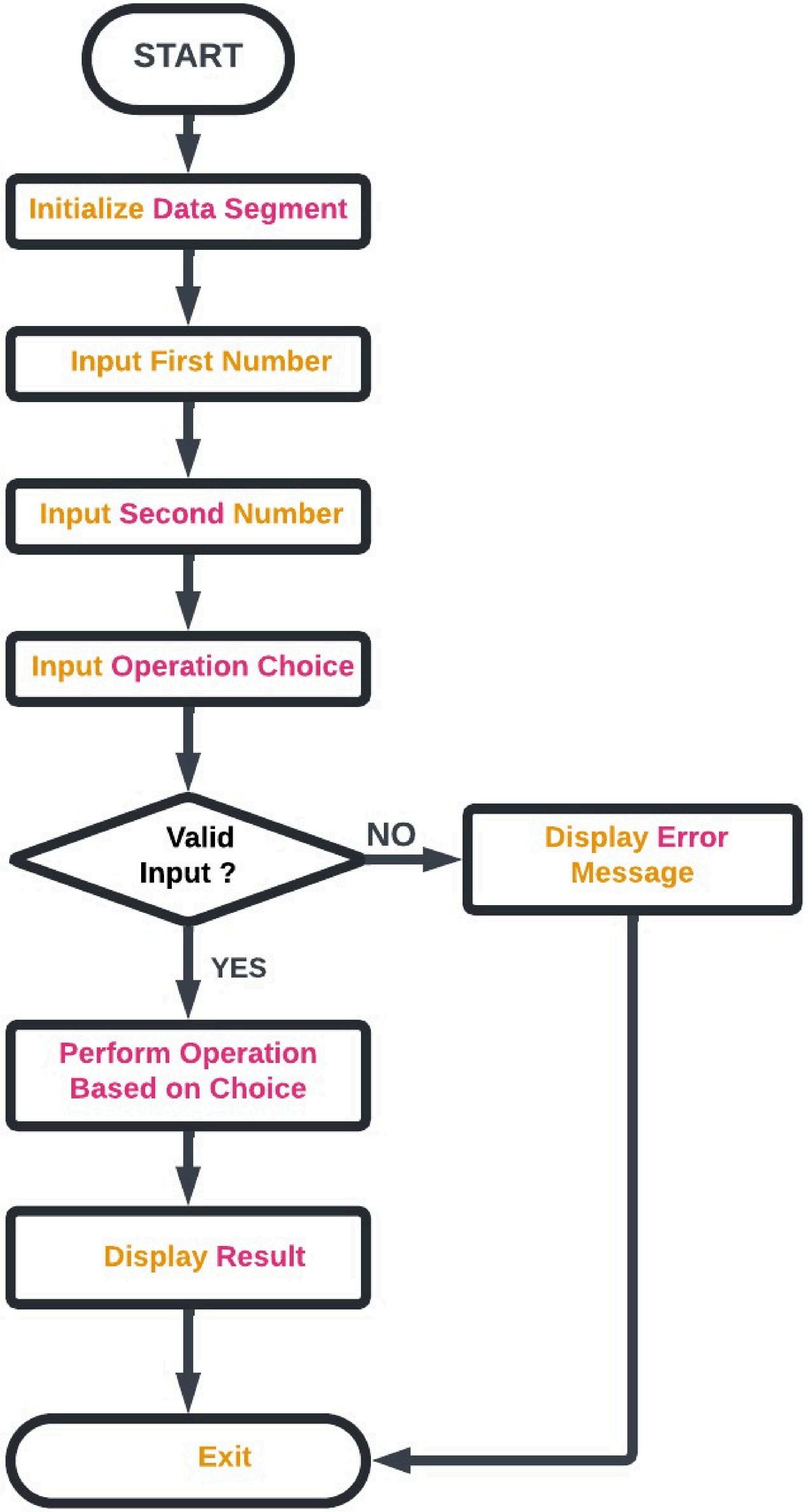
including equality, greater than, and less than.

Logical Operations

Bitwise operations such as AND, OR, NOT, and XOR are provided for Boolean logic and bit manipulation.



FLOW CHART



START

Initialize Data Segment

Input First Number

Input Second Number

Input Operation Choice

Valid Input ?

NO

Display Error Message

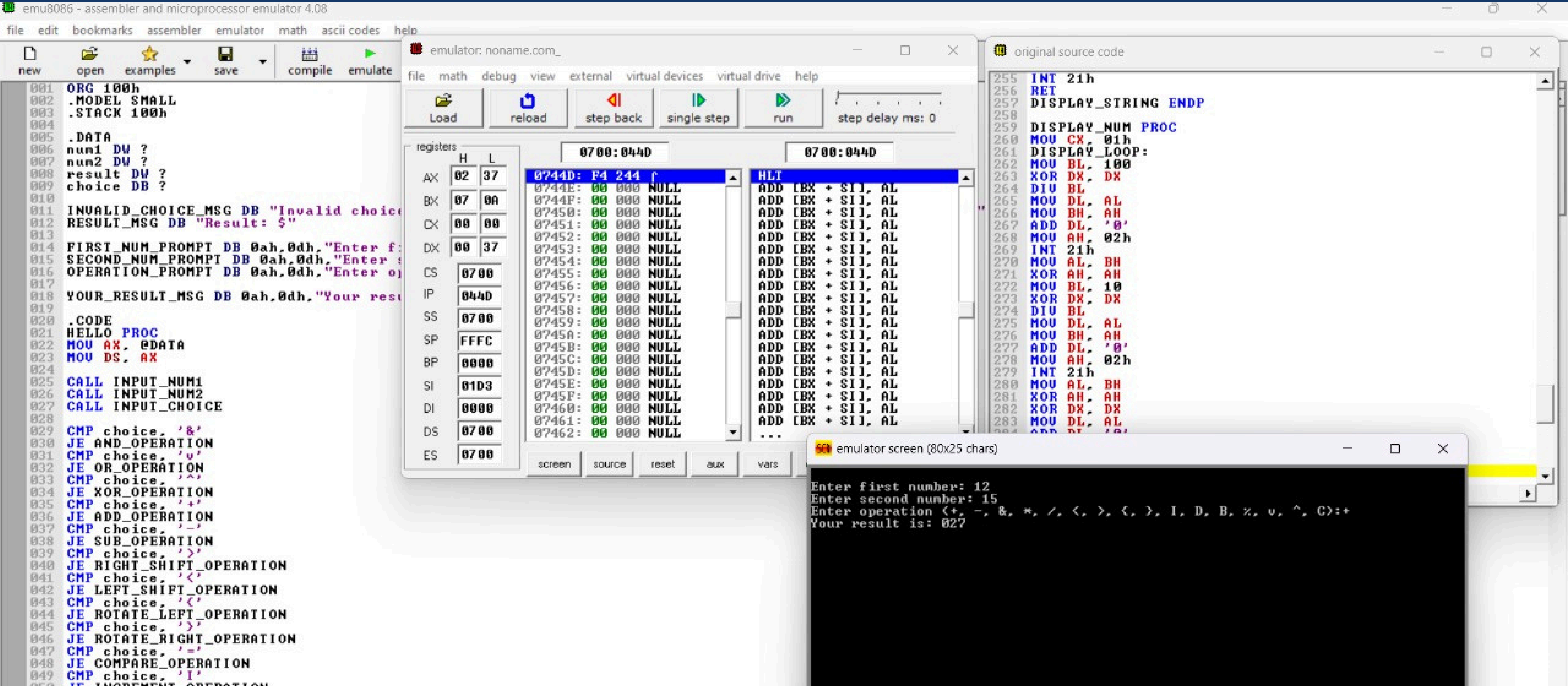
YES

Perform Operation Based on Choice

Display Result

Exit

INTERFACE


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new
open
examples
save
compile
emulate
emulator: noname.com_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
screen
source
reset
aux
vars
original source code
emulator screen (80x25 chars)

```
ORG 100h
.MODEL SMALL
.STACK 100h

.DATA
num1 DW ?
num2 DW ?
result DW ?
choice DB ?

INVALID_CHOICE_MSG DB "Invalid choic
RESULT_MSG DB "Result: $"

FIRST_NUM_PROMPT DB 0ah,0dh,"Enter f:
SECOND_NUM_PROMPT DB 0ah,0dh,"Enter :
OPERATION_PROMPT DB 0ah,0dh,"Enter o]

YOUR_RESULT_MSG DB 0ah,0dh,"Your resu

.CODE
HELLO PROC
MOV AX, @DATA
MOV DS, AX

CALL INPUT_NUM1
CALL INPUT_NUM2
CALL INPUT_CHOICE

CMP choice, '&'
JE AND_OPERATION
CMP choice, 'v'
JE OR_OPERATION
CMP choice, '^'
JE XOR_OPERATION
CMP choice, '+'
JE ADD_OPERATION
CMP choice, '-'
JE SUB_OPERATION
CMP choice, '>'
JE RIGHT_SHIFT_OPERATION
CMP choice, '<'
JE LEFT_SHIFT_OPERATION
CMP choice, '{'
JE ROTATE_LEFT_OPERATION
CMP choice, '}'
JE ROTATE_RIGHT_OPERATION
CMP choice, '='
JE COMPARE_OPERATION
CMP choice, 'I'
JE INCREMENT_OPERATION
```

	H	L
AX	02	37
BX	07	0A
CX	00	00
DX	00	37
CS	0700	
IP	044D	
SS	0700	
SP	FFFC	
BP	0000	
SI	01D3	
DI	0000	
DS	0700	
ES	0700	

```
0700:044D
0744D: F4 244 ⌠
0744E: 00 000 NULL
0744F: 00 000 NULL
07450: 00 000 NULL
07451: 00 000 NULL
07452: 00 000 NULL
07453: 00 000 NULL
07454: 00 000 NULL
07455: 00 000 NULL
07456: 00 000 NULL
07457: 00 000 NULL
07458: 00 000 NULL
07459: 00 000 NULL
0745A: 00 000 NULL
0745B: 00 000 NULL
0745C: 00 000 NULL
0745D: 00 000 NULL
0745E: 00 000 NULL
0745F: 00 000 NULL
07460: 00 000 NULL
07461: 00 000 NULL
07462: 00 000 NULL
```

```
0700:044D
HLT
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

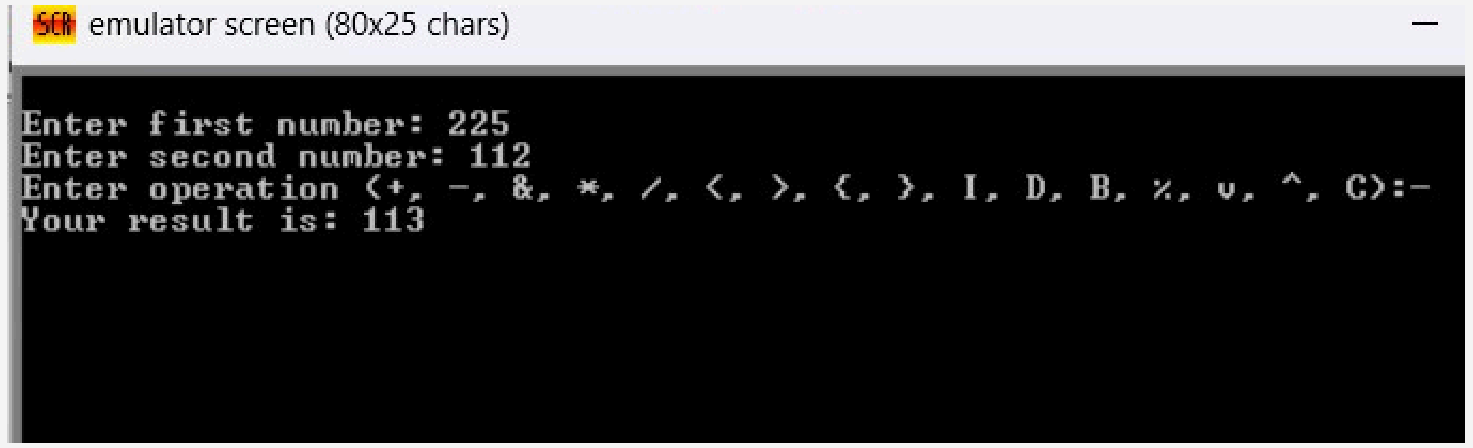
```
INT 21h
RET
DISPLAY_STRING ENDP

DISPLAY_NUM PROC
MOV CX, 01h
DISPLAY_LOOP:
MOV BL, 100
XOR DX, DX
DIV BL
MOV DL, AL
MOV BH, AH
ADD DL, '0'
MOV AH, 02h
INT 21h
MOV AL, BH
XOR AH, AH
MOV BL, 10
XOR DX, DX
DIV BL
MOV DL, AL
MOV BH, AH
ADD DL, '0'
MOV AH, 02h
INT 21h
MOV AL, BH
XOR AH, AH
XOR DX, DX
MOV DL, AL
ADD DL, '0'
```

```
Enter first number: 12
Enter second number: 15
Enter operation (+, -, &, *, /, <, >, {, }, I, D, B, %, v, ^, C):+
Your result is: 027
```

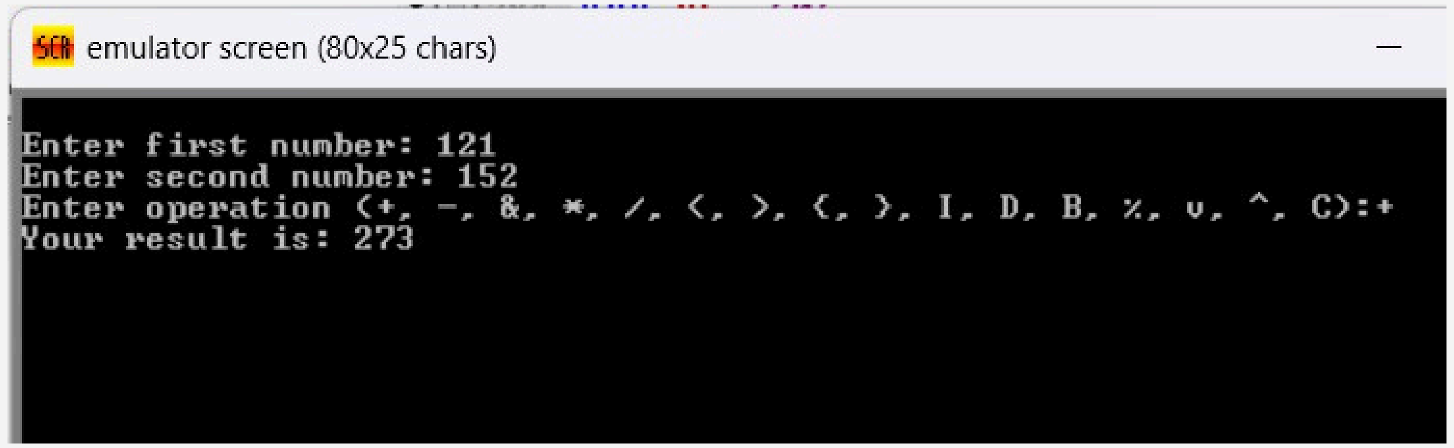
ARITHMETIC INSTRUCTIONS

Subtraction



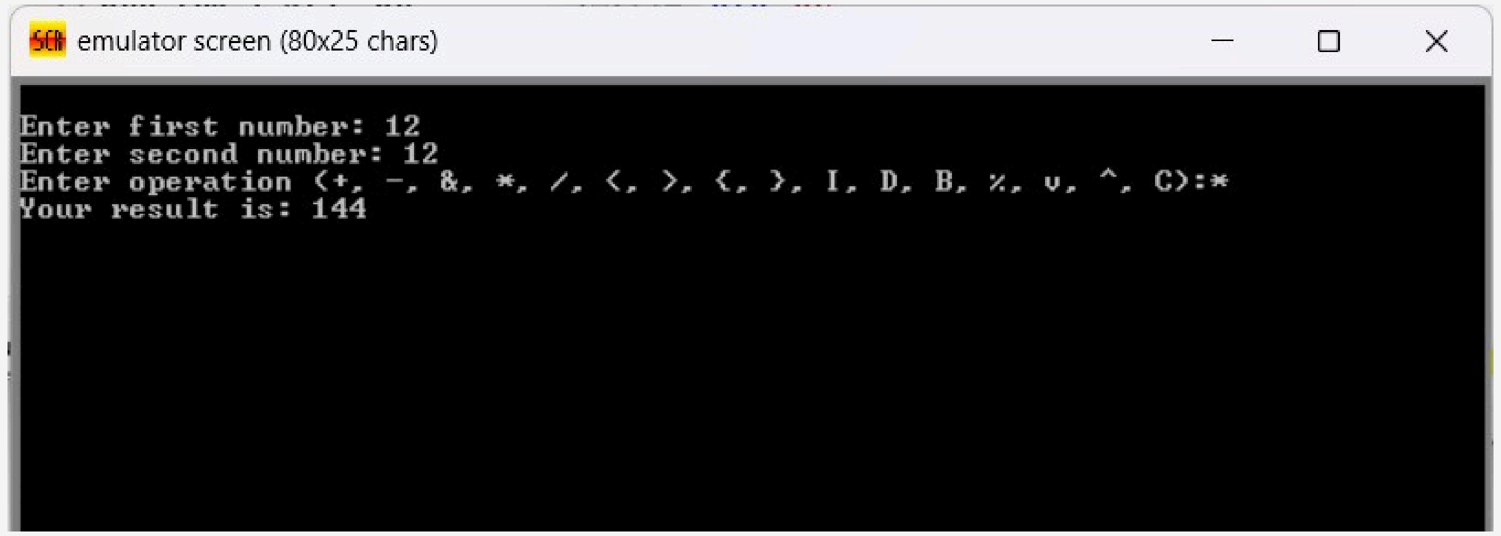
```
Enter first number: 225
Enter second number: 112
Enter operation (+, -, &, *, /, <, >, {, }, I, D, B, %, v, ^, C):-
Your result is: 113
```

Addition



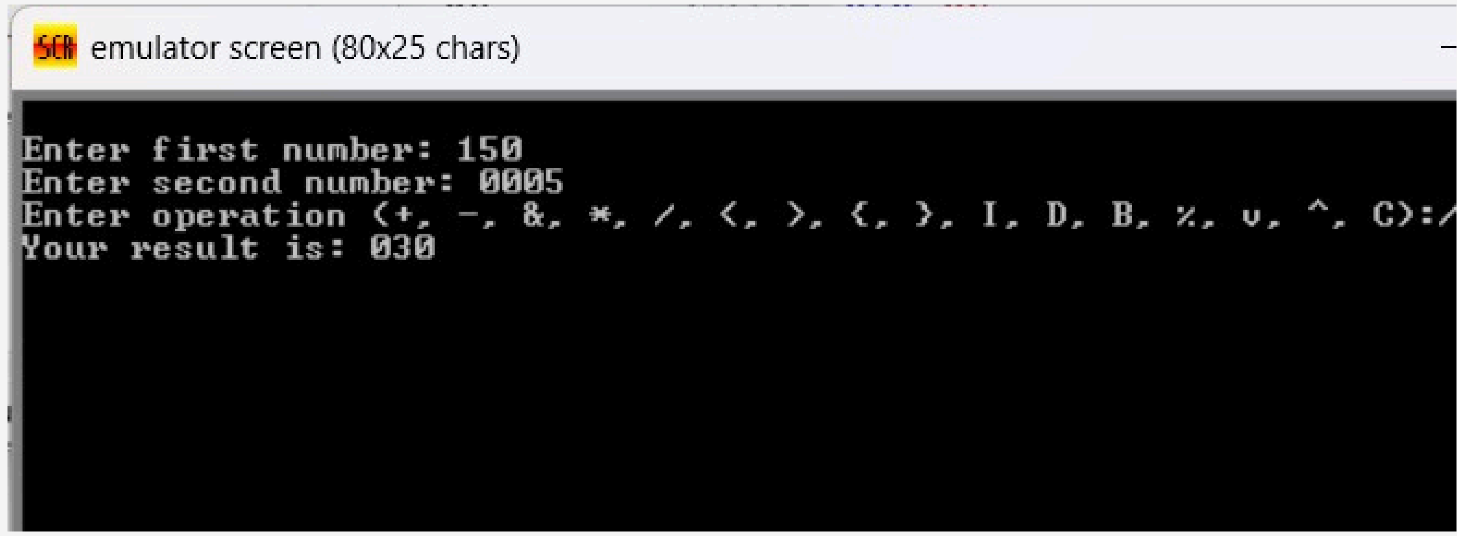
```
Enter first number: 121
Enter second number: 152
Enter operation (+, -, &, *, /, <, >, {, }, I, D, B, %, v, ^, C):+
Your result is: 273
```

Multiply



```
Enter first number: 12
Enter second number: 12
Enter operation (+, -, &, *, /, <, >, {, }, I, D, B, %, v, ^, C):*
Your result is: 144
```

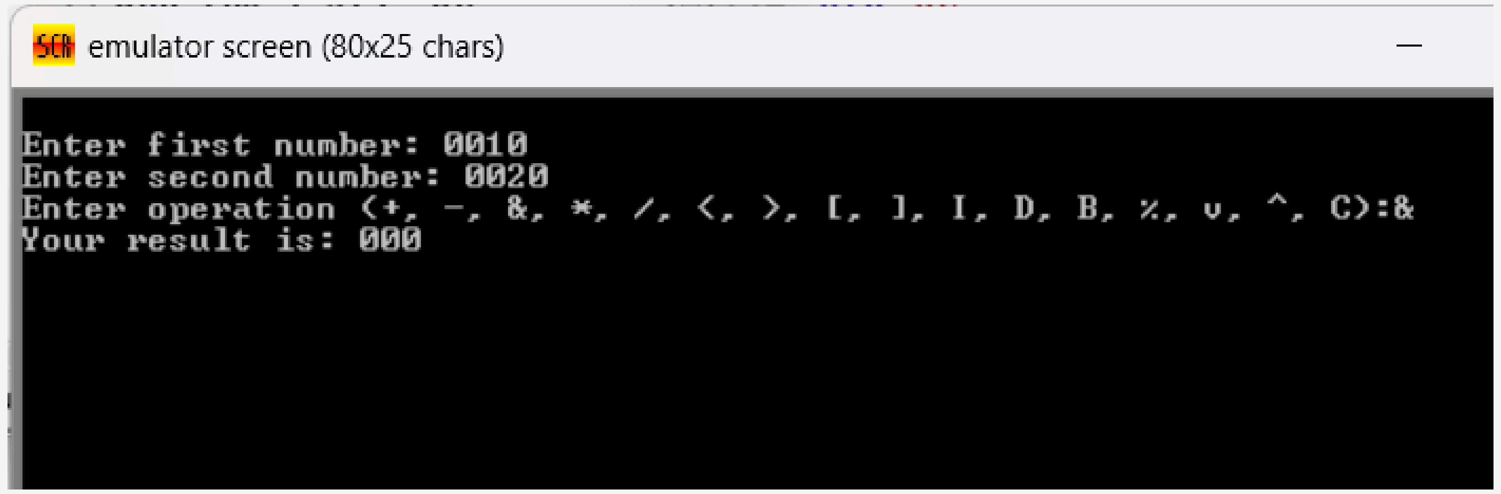
Division



```
Enter first number: 150
Enter second number: 0005
Enter operation (+, -, &, *, /, <, >, {, }, I, D, B, %, v, ^, C):/
Your result is: 030
```

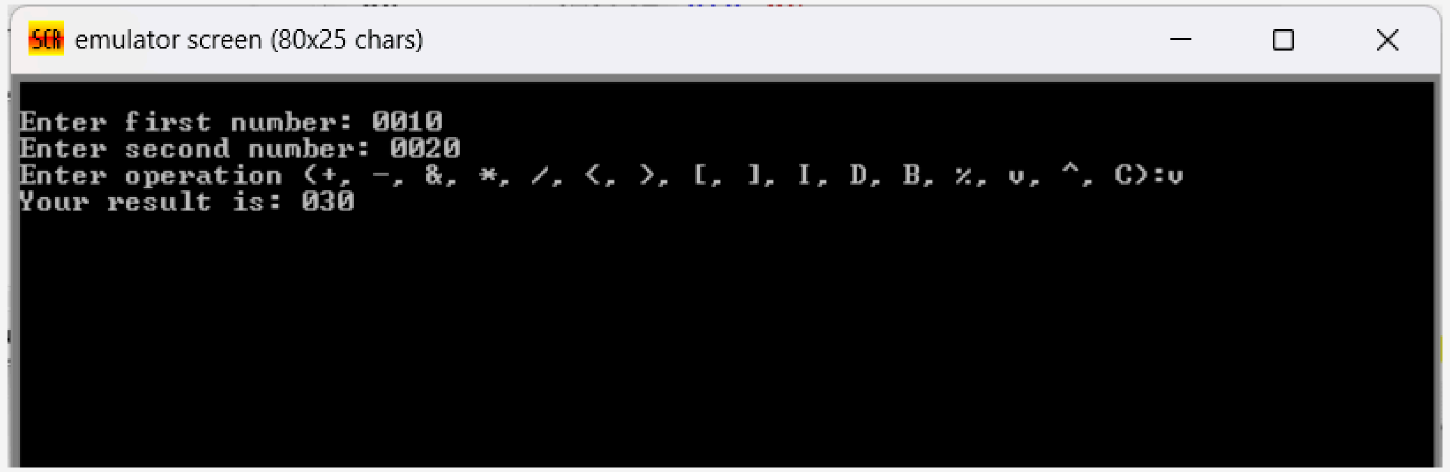
LOGICAL OPERATIONS

BITWISE AND



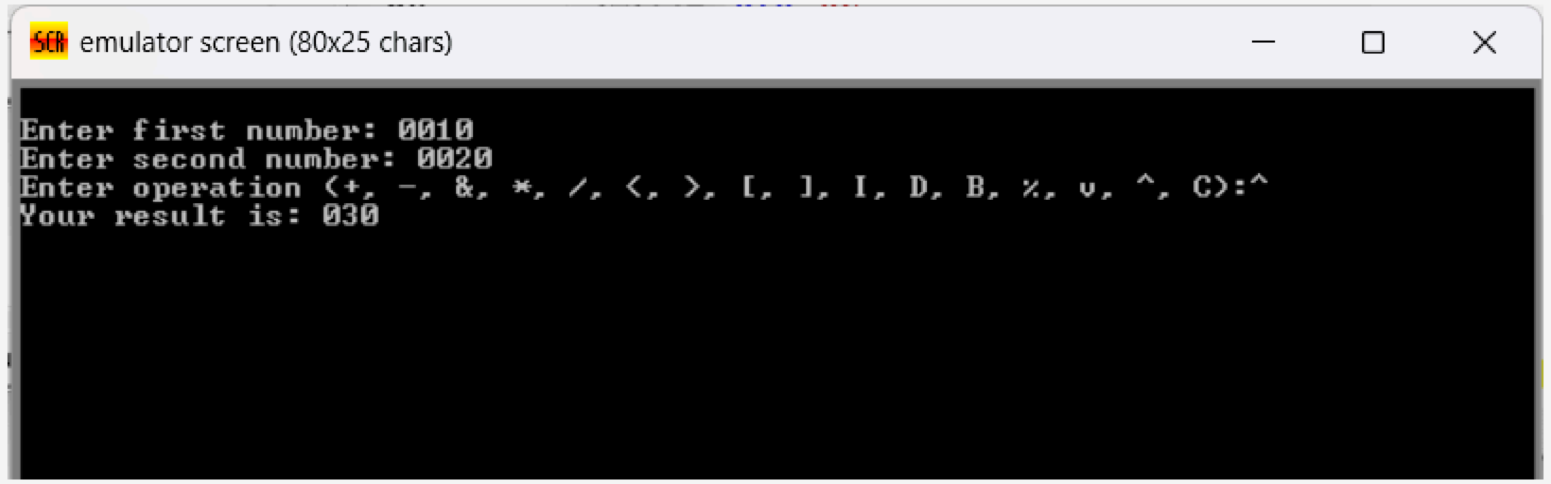
```
Enter first number: 0010
Enter second number: 0020
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):&
Your result is: 000
```

BITWISE OR



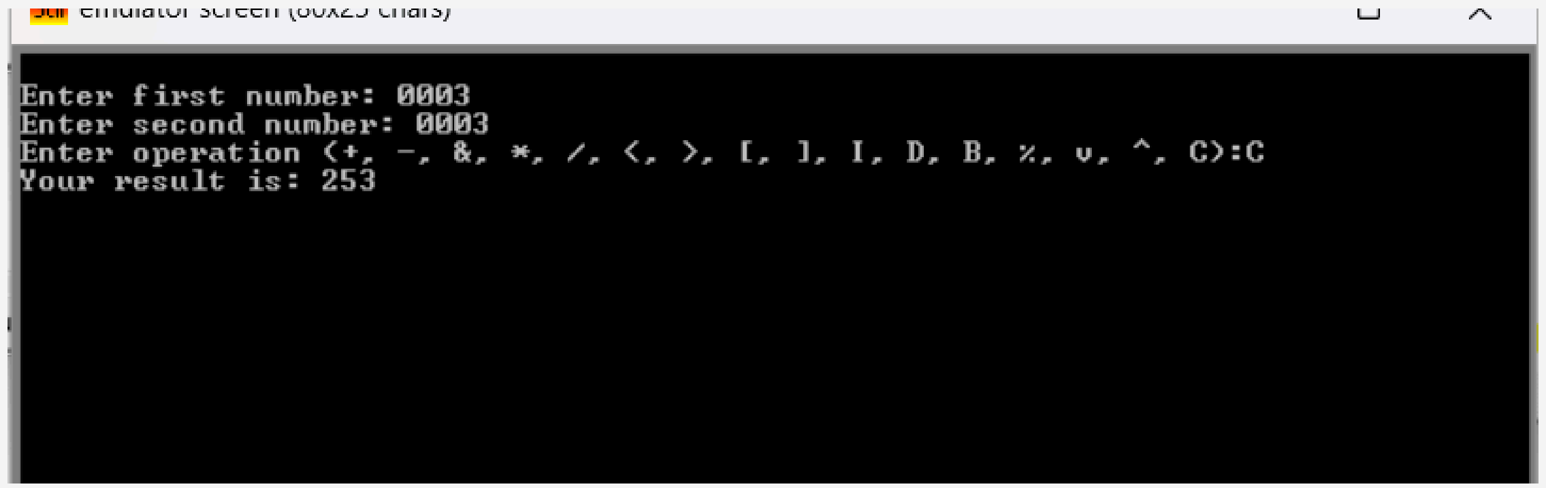
```
Enter first number: 0010
Enter second number: 0020
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):v
Your result is: 030
```

BITWISE XOR



```
Enter first number: 0010
Enter second number: 0020
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):^
Your result is: 030
```

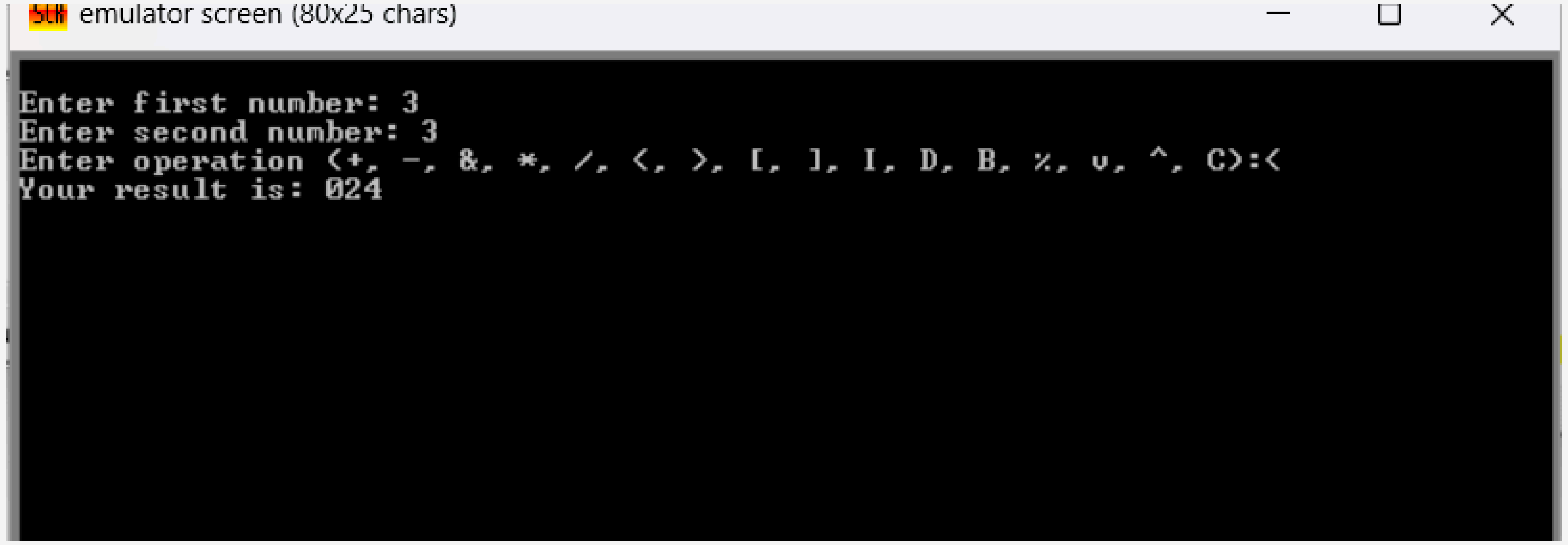
COMPLEMENT



```
Enter first number: 0003
Enter second number: 0003
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):C
Your result is: 253
```

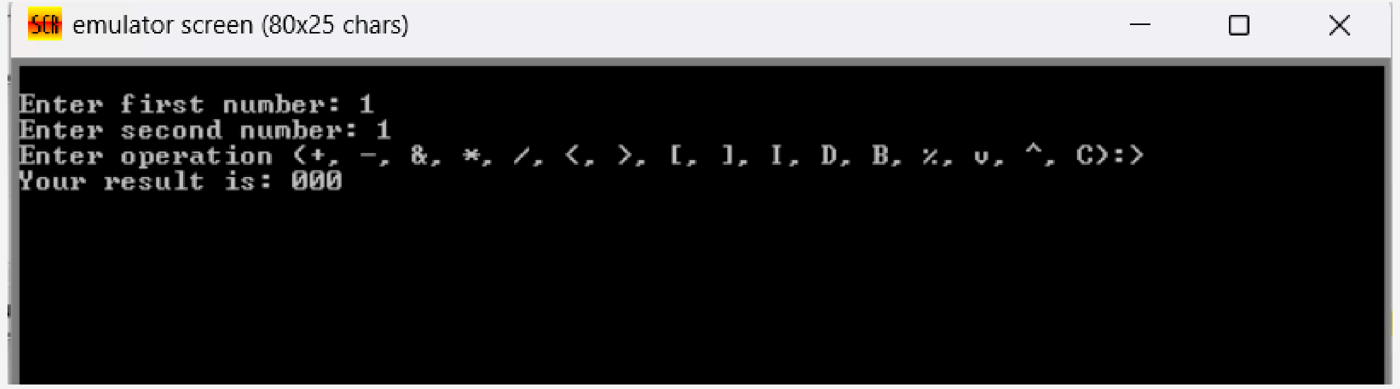
SHIFT/ROTATE OPERATIONS

SHIFT LEFT



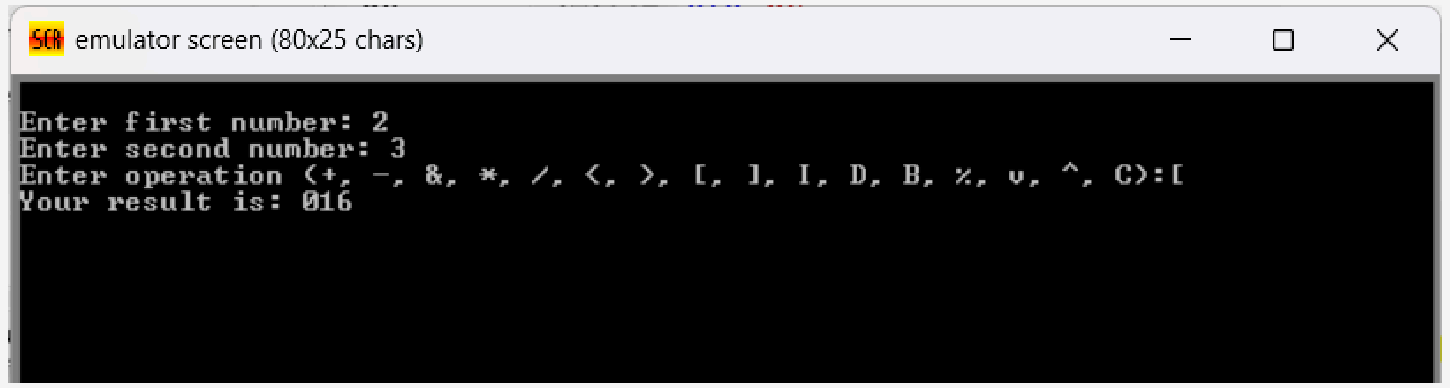
```
Enter first number: 3
Enter second number: 3
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):<
Your result is: 024
```

SHIFT RIGHT



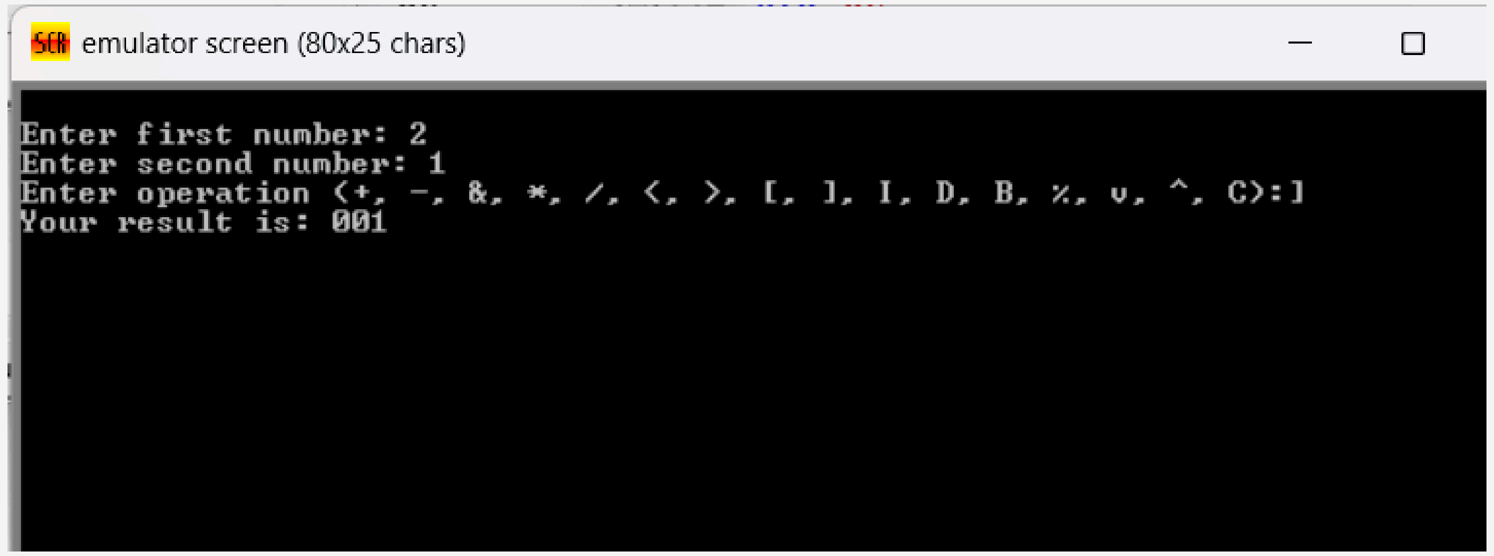
```
Enter first number: 1
Enter second number: 1
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):>
Your result is: 000
```

ROTATE LEFT



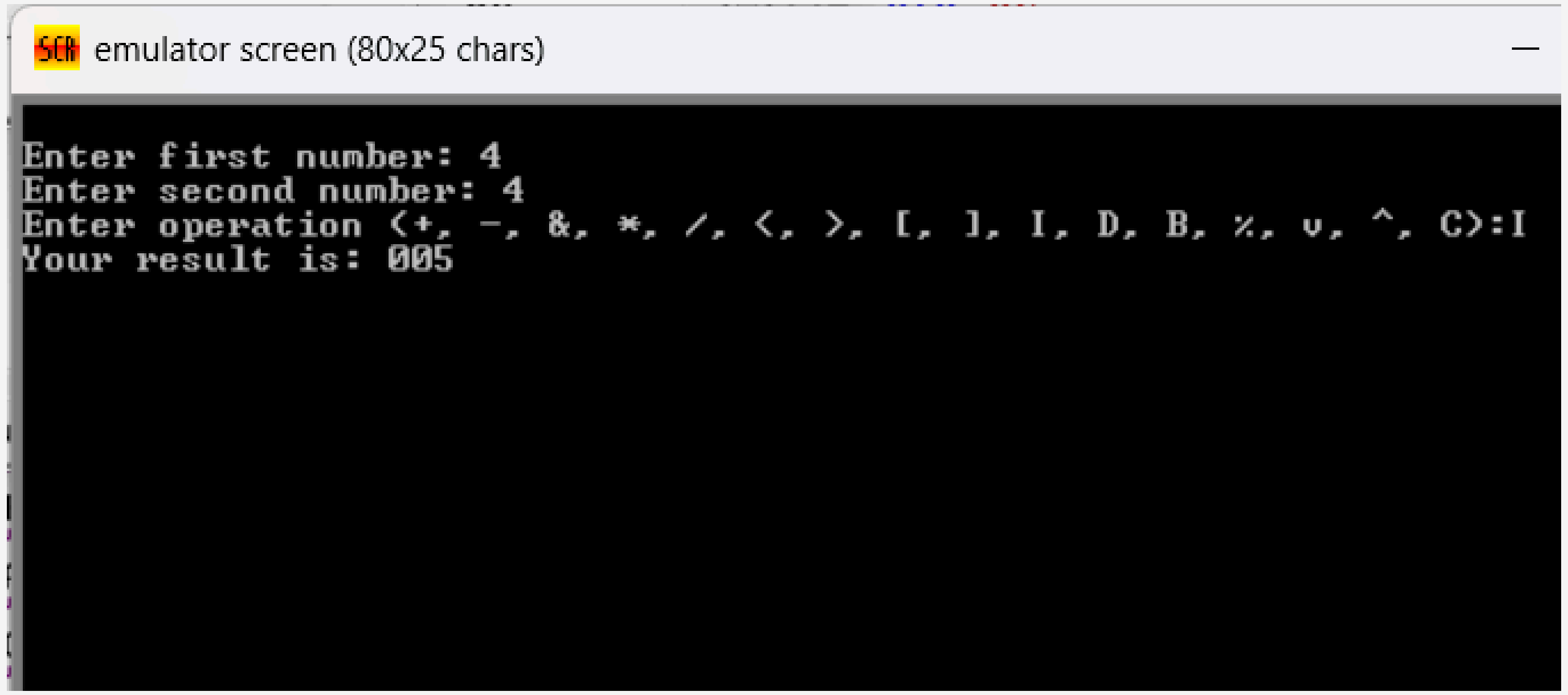
```
Enter first number: 2
Enter second number: 3
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):[
Your result is: 016
```

ROTATE RIGHT



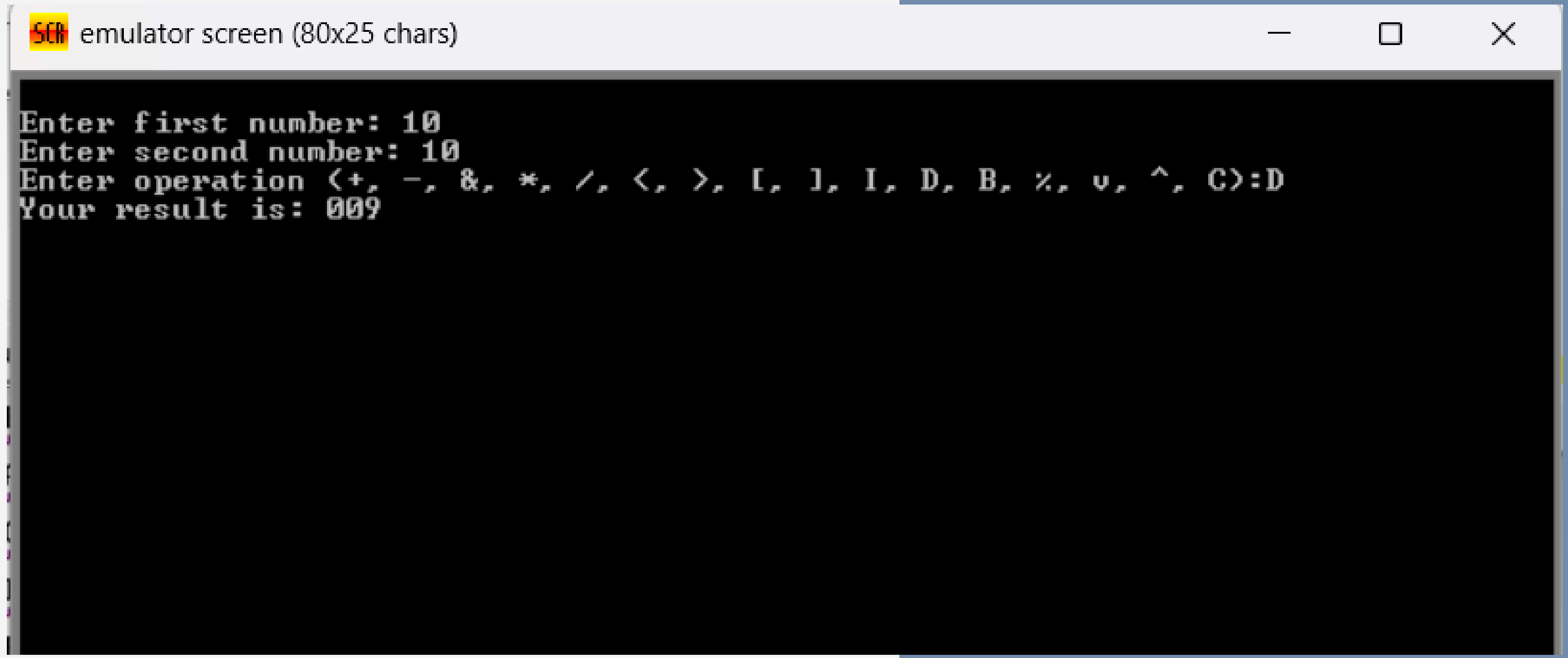
```
Enter first number: 2
Enter second number: 1
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):]
Your result is: 001
```

INCREMENT



```
Enter first number: 4
Enter second number: 4
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):I
Your result is: 005
```

DECREMENT



```
Enter first number: 10
Enter second number: 10
Enter operation (+, -, &, *, /, <, >, [, ], I, D, B, %, v, ^, C):D
Your result is: 009
```

LIMITATIONS

- **Limited Error Handling:**The code only provides a single error message for an invalid choice. It does not handle other potential errors such as division by zero or overflow errors. As a result, the program's robustness and reliability are compromised.

- **Limited Operand Size:** The code assumes that the operands (num1 and num2) are 16-bit integers (DW). This limits the range of values that can be processed by the program. For larger numbers, the program may produce incorrect results or overflow.

- **Limited User Interface:** The program's user interface is text-based and lacks interactivity

Thank you !