8086 Assembly Calculator – Technical Report

1. Overview

This project implements a menu-driven calculator in Intel 8086 Assembly Language targeting the DOS environment and using INT 21H services for input/output operations.

The calculator supports:

- Addition
- Subtraction
- Multiplication
- Division (with quotient and remainder)
- Division-by-zero handling
- Input validation
- Interactive loop-based interface

Memory model used: `.MODEL SMALL`
Stack size: `100H` (256 bytes)

---

2. System Architecture

2.1 Memory Model

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
```

- SMALL model: One data segment (≤64 KB) and one code segment (≤64 KB).
- DS initialized using:

```
MOV AX, @DATA
MOV DS, AX
```

---

3. Functional Components

The program is modularized into:

Component	Type	Purpose
MAIN	Procedure	Program controller
READ_NUMBER	Procedure	Multi-digit numeric input
PRINT_NUMBER	Procedure	Decimal output conversion
CLEAR_SCREEN	Procedure	UI formatting
Arithmetic Blocks	Labels	Core operations

---

4. User Interface Flow

4.1 Execution Flow

```
START
  ↓
Initialize DS
  ↓
MAIN_LOOP
  ↓
Display Banner
  ↓
Display Menu
  ↓
Read Choice
  ↓
Validate Input
  ↓
Execute Operation
  ↓
Display Result
  ↓
Wait for Key
  ↓
Repeat
```

Exit occurs when user selects option 5.

---

5. Interrupt Usage (DOS INT 21H)

AH Value	Function	Usage
01H	Read character	Menu + digit input
02H	Print character	Digit output
09H	Print string	UI messages
4CH	Exit program	Termination

---

6. Arithmetic Implementation

6.1 Addition

```
MOV AX, num1
ADD AX, num2
```

- Result stored in `result`
- 16-bit arithmetic
- No overflow detection

6.2 Subtraction

```
MOV AX, num1
SUB AX, num2
```

Negative results are not formatted properly (will print unsigned representation).

6.3 Multiplication

```
MOV AX, num1
MOV BX, num2
MUL BX
```

- MUL produces a 32-bit result in DX:AX
- Only AX stored → high word ignored
- Potential overflow if result > 65535

6.4 Division

```
MOV DX, 0
DIV BX
```

Output: - Quotient → AX - Remainder → DX

Includes division-by-zero check:

```
CMP num2, 0
JE DIVISION_ERROR
```

---

7. Input Handling – READ_NUMBER Procedure

Algorithm

1. Read character
2. Validate digit
3. Convert ASCII → numeric
4. Multiply previous result by 10
5. Add new digit
6. Limit to 4 digits

Mathematical Logic

If number is 123:

```
0 * 10 + 1 = 1
1 * 10 + 2 = 12
12 * 10 + 3 = 123
```

Limitations

- Maximum: 4 digits
- No backspace handling
- No negative number support
- No overflow detection

---

8. Output Handling – PRINT_NUMBER Procedure

Conversion Algorithm

Repeated division by 10:

```
AX ÷ 10
Store remainder
Repeat
Print digits in reverse
```

Example (123):

```
123 ÷ 10 → 12 remainder 3
12 ÷ 10 → 1 remainder 2
1 ÷ 10 → 0 remainder 1
Print: 1 2 3
```

---

9. Error Handling

Implemented

- Invalid menu choice
- Division by zero

Not Implemented

- Arithmetic overflow
- Negative numbers
- Large multiplication overflow (DX ignored)

---

10. Strengths

- Modular structure (procedures)
- Clean UI formatting
- Proper stack preservation (PUSH/POP discipline)
- Multi-digit input support
- Loop-driven architecture
- Division remainder support

---

11. Technical Limitations

Issue	Explanation
No signed arithmetic	Uses unsigned instructions
No overflow detection	Flags ignored
Multiplication overflow ignored	DX not handled
No backspace support	Input loop ignores editing
No screen clear interrupt	Uses newline-based clearing

---

12. Time Complexity

- Input: O(n) per digit
- Print: $O(\log_{10} n)$
- Arithmetic: O(1)

Efficient for 16-bit architecture.

---

13. Security & Robustness Perspective

- Proper register preservation (good stack hygiene)
- No buffer overflow risks (digit-limited)
- Controlled user input loop
- Explicit validation

However:

- No bounds checking for arithmetic overflow
- Potential logical corruption for large multiplication

---

14. Suggested Improvements

1) Support Signed Arithmetic

Replace:

```
MUL → IMUL
DIV → IDIV
```

Add sign handling logic.

---

2) Handle 32-bit Multiplication Result

Store both DX and AX.

---

3) Add Overflow Detection

Check:

```
JO overflow_label
```

---

4) Improve Screen Clearing

Use BIOS interrupt INT 10H.

---

5) Add Backspace Support

Handle ASCII 08H in READ_NUMBER.

---

15. Conclusion

This project demonstrates:

- Strong understanding of 8086 architecture
- Effective use of DOS interrupts
- Structured assembly programming
- Modular procedural design
- Control flow mastery

With additional overflow handling and signed arithmetic support, this could be considered a near-production-quality educational calculator for DOS-based 8086 systems.