**ASMCalc64**

**64-bit calculator in assembly language**

**Understanding a 64-bit Assembly Calculator**

**A Deep Dive into x86\_64 Assembly Programming**

**Program Overview**

**Our calculator supports:**

* **Basic arithmetic operations (+, -, \*, /)**
* **Integer input/output**
* **Error handling**
* **Continuous operation loop**
* **Clean exit functionality**

**Data Section Breakdown**

**section .data**

**prompt\_msg db "Enter operation (+ - \* / q to quit): ", 0**

**prompt\_len equ $ - prompt\_msg**

* **.data section contains initialized data**
* **db defines bytes**
* **equ $ - prompt\_msg calculates length automatically**
* **0 represents null terminator**

**Key Teaching Point: Assembly requires explicit memory management!**

**BSS Section**

**section .bss**

**operation resb 2 ; Space for operator char + newline**

**num1 resb 32 ; Buffer for first number string**

**num2 resb 32 ; Buffer for second number string**

**num1\_val resq 1 ; 64-bit storage for first number**

**num2\_val resq 1 ; 64-bit storage for second number**

* **.bss is for uninitialized data**
* **resb reserves bytes**
* **resq reserves quadwords (64 bits)**
* **Numbers are stored twice:**
  + **As strings (for input/output)**
  + **As binary values (for calculations)**

**Register Usage in x86\_64**

**Important registers we use:**

* **rax - Accumulator, syscall number, arithmetic**
* **rdi - First argument to syscalls**
* **rsi - Second argument to syscalls**
* **rdx - Third argument to syscalls**
* **rbx - General purpose (we use for second operand)**
* **rcx - Counter in loops**

**Key Difference from 32-bit: Registers are now 64-bit (prefix 'r' instead of 'e')**

**System Calls**

**; Print message**

**mov rax, 1 ; sys\_write**

**mov rdi, 1 ; stdout**

**mov rsi, prompt\_msg ; buffer**

**mov rdx, prompt\_len ; length**

**syscall**

**Modern syscall convention:**

1. **rax holds syscall number**
2. **Arguments in rdi, rsi, rdx**
3. **Use syscall instruction (not int 0x80)**

**Common syscalls:**

* **0 = read**
* **1 = write**
* **60 = exit**

**String to Integer Conversion**

**string\_to\_int:**

**xor rax, rax ; Clear result**

**xor rcx, rcx ; Clear sign flag**

**.process\_digits:**

**movzx rbx, byte [rsi] ; Get character**

**sub bl, '0' ; Convert ASCII to number**

**imul rax, 10 ; Multiply by 10**

**add rax, rbx ; Add new digit**

**Key concepts:**

* **ASCII to integer conversion**
* **Building number digit by digit**
* **Handling negative numbers**
* **Error checking**

**Arithmetic Operations**

**do\_add:**

**add rax, rbx**

**jmp print\_result**

**do\_multiply:**

**imul rbx**

**jmp print\_result**

**do\_divide:**

**test rbx, rbx ; Check for division by zero**

**jz division\_by\_zero**

**cqo ; Sign-extend RAX into RDX:RAX**

**idiv rbx**

**Important points:**

* **add/sub for basic arithmetic**
* **imul for signed multiplication**
* **idiv for signed division**
* **cqo extends sign for 64-bit division**

**Error Handling**

**Our program handles:**

1. **Division by zero**
2. **Invalid operations**
3. **Input validation**
4. **Buffer overflows (via fixed buffer sizes)**

**Example:**

**division\_by\_zero:**

**mov rax, 1**

**mov rdi, 1**

**mov rsi, div\_zero\_msg**

**mov rdx, div\_zero\_len**

**syscall**

**Program Flow**

1. **Display prompt**
2. **Read operation**
3. **Read first number**
4. **Convert to integer**
5. **Read second number**
6. **Convert to integer**
7. **Perform operation**
8. **Convert result to string**
9. **Display result**
10. **Loop or exit**

**Debug Features**

**push rax**

**mov rax, 1**

**mov rdi, 1**

**mov rsi, debug\_num1**

**mov rdx, debug\_num1\_len**

**syscall**

**pop rax**

* **Shows intermediate values**
* **Helps verify conversion**
* **Useful for understanding program flow**
* **Essential for development and teaching**

**Key Learning Points**

1. **Memory Segments:**
   * **.data for initialized data**
   * **.bss for uninitialized data**
   * **.text for code**
2. **64-bit Specifics:**
   * **Larger registers (64-bit)**
   * **Different syscall interface**
   * **More available registers**
3. **Number Handling:**
   * **String ↔ Integer conversion**
   * **Signed arithmetic**
   * **Buffer management**
4. **Program Structure:**
   * **Input/Output loops**
   * **Error handling**
   * **Code organization**

**Common Student Questions**

**Q: Why use Assembly instead of a high-level language? A: Understanding computer architecture, system calls, and memory management**

**Q: Why 64-bit instead of 32-bit? A: Modern architecture, larger numbers, more registers**

**Q: How does error handling work? A: Compare operations, conditional jumps, system calls for messages**

**Practice Exercises**

1. **Add new operations:**
   * **Modulo**
   * **Power**
   * **Square root**
2. **Enhance functionality:**
   * **Decimal number support**
   * **Multiple operations**
   * **Memory functions**
3. **Improve error handling:**
   * **Better input validation**
   * **Overflow checking**
   * **More user feedback**

**End Notes**

**This calculator demonstrates:**

* **Basic assembly structures**
* **System interaction**
* **Number handling**
* **Input/Output operations**
* **Error management**
* **Program flow control**

**Remember: Assembly provides direct hardware access but requires careful memory and register management!**