

Multiplication Table Program Project

Soha Rehman (CSC-18F-116)

Sauda Paracha (CSC-18F-142)

INTRODUCTION:

An assembly (or assembler) language, often abbreviated asm, is a [low-level programming language](https://en.wikipedia.org/wiki/Low-level_programming_language) for a [computer](https://en.wikipedia.org/wiki/Computer), or other programmable device, in which there is a very strong (but often not [one-to-one](https://en.wikipedia.org/wiki/Bijection)) correspondence between the language and the [architecture's](https://en.wikipedia.org/wiki/Computer_architecture) [machine code](https://en.wikipedia.org/wiki/Machine_code) [instructions](https://en.wikipedia.org/wiki/Instruction_set_architecture). Each assembly language is specific to a particular computer architecture. In contrast, most [high-level programming languages](https://en.wikipedia.org/wiki/High-level_programming_language) are generally [portable](https://en.wikipedia.org/wiki/Porting) across multiple architectures but require [interpreting](https://en.wikipedia.org/wiki/Interpreter_(computing)) or [compiling](https://en.wikipedia.org/wiki/Compiler). Assembly language may also be called symbolic machine code.

Assembly language is converted into executable machine code by a [utility program](https://en.wikipedia.org/wiki/Utility_software) referred to as an [assembler](https://en.wikipedia.org/wiki/Assembly_language#Assembler). The conversion process is referred to as assembly, or assembling the [source code](https://en.wikipedia.org/wiki/Source_code). Assembly time is the computational step where an assembler is run.

Assembly language uses a [mnemonic](https://en.wikipedia.org/wiki/Mnemonic) to represent each low-level [machine instruction](https://en.wikipedia.org/wiki/Machine_code) or [opcode](https://en.wikipedia.org/wiki/Opcode), typically also each [architectural register](https://en.wikipedia.org/wiki/Register_(computing)#ARCHITECTURAL), [flag](https://en.wikipedia.org/wiki/Bit_field), etc. Many operations require one or more [operands](https://en.wikipedia.org/wiki/Operand#Computer_science) in order to form a complete instruction. Most assemblers can take [expressions](https://en.wikipedia.org/wiki/Expression_(computer_science)) of numbers, named constants, registers, and [labels](https://en.wikipedia.org/wiki/Label_(computer_science)) as operands. Thus, the programmers are freed from tedious repetitive calculations. Depending on the architecture, these elements may also be combined for specific instructions or [addressing modes](https://en.wikipedia.org/wiki/Addressing_mode) using [offsets](https://en.wikipedia.org/wiki/Offset_(computer_science)) or other data as well as fixed addresses. Many assemblers offer additional mechanisms to facilitate program development, to control the assembly process, and to aid [debugging](https://en.wikipedia.org/wiki/Debugging).

SCOPE:

A very long time ago, when dinosaurs still roamed the Earth, I designed a computer (using the fire-blackened end of a spear on a cave's wall.) Still in high school, I was greedily-somewhat competent with digital circuits but knew nothing about computer architecture.

But I had learned Fortran and naively assumed some parallel between that language and how computers worked. The result: a machine that would have been a complete failure, and which used a greatly subsetted and compressed form of Fortran as its native instruction set.

College gave me grade-destroying access to a Univac 1108 and I quickly learned its assembly language. Suddenly computer architecture became crystal clear. The one-to-one mapping of machine instructions to simple logic circuits was beautiful; the stored program that substituted instructions in memory for massive amounts of hardware breathtaking.

Since then I've read many books about computer design but feel none would reveal a fundamental insight into CPU architecture without relying heavily on the essentials of assembly language. The ALU, program counter and stack pointers are dead lifeless things, capable of nothing till animated like Frankenstein's monster with instructions stored in memory.

Assembly is both the basis of all computers and the name of a class of languages. Often used to specify a particular variant (e.g,. "8051 assembly"), it oddly doesn't even get a capitalized first letter as all other languages do. Or, did, until grammar died a horrible death at the hands of clowns sporting marketing degrees. Proper nouns like Fortran, Ada, C, and Pascal gave way to iPhone, dBASE, and eEverything. The nuns at St. Camillus would have beaten us senseless for peppering our writing with stuDLycAps, yet today that affectation is not only common, one is relieved when at the very least the spelling is correct.

In the early days of microprocessors all programs were written in assembly. No C- code compilers existed for the minimal CPUs of the day and memory was so expensive and processors so slow that no one dreamed of sacrificing any form of efficiency for reduced development costs. All firmware folk were experts in at least one assembly language. Usually several.

Perhaps the two greatest gifts to the embedded world were C and IDEs. Though I still think assembly is more fun than using a high-level language, C reduces development costs so much I'd never dream of cranking much assembly code anymore.

**CODE:**

**.MODEL SMALL**

**.DATA**

**MSG DB 0AH,0DH,'ENTER A NUMBER: ',0AH,'$'**

**NEW DB 0AH,0DH,24H**

**INP DB 10,0,10 DUP('$')**

**MULT DB 01H**

**MSG2 DB 0AH,0DH,'MULTIPLICATION TABLE:',0AH,'$'**

**.CODE**

**START:**

**MOV AX,@DATA**

**MOV DS,AX**

**LEA DX,MSG**

**MOV AH,09H**

**INT 21H**

**LEA DX,INP**

**MOV AH,0AH**

**INT 21H**

**ADD DX,02H**

**MOV SI,DX**

**LEA DX,MSG2**

**MOV AH,09H**

**INT 21H**

**LEA DI,MULT**

**MOV AH,00H**

**MOV CX,000AH**

**INC DI**

**MOV [DI],CX**

**DEC DI**

**MOV BL,[DI]**

**MOV BH,00H**

**AGAIN:**

**MOV [DI],BL**

**INC DI**

**MOV [DI],CL**

**DEC DI**

**MOV DL,[SI]**

**MOV AH,02H**

**INT 21H**

**MOV DL,'\*'**

**MOV AH,02H**

**INT 21H**

**MOV DL,[DI]**

**CMP DL,0AH**

**JNE TEN**

**MOV DL,31H**

**MOV AH,02H**

**INT 21H**

**MOV DL,30H**

**MOV AH,02H**

**INT 21H**

**JMP EQUAL**

**TEN:ADD DL,30H**

**MOV AH,02H**

**INT 21H**

**EQUAL:MOV DL,'='**

**MOV AH,02H**

**INT 21H**

**MOV AL,[SI]**

**SUB AL,30H**

**MOV AH,00H**

**MUL BX**

**CALL DISP**

**MOV BL,[DI]**

**MOV BH,00H**

**INC BX**

**INC DI**

**MOV CL,[DI]**

**MOV CH,00H**

**DEC DI**

**LOOP AGAIN**

**.EXIT**

**DISP PROC**

**MOV BX, 000AH**

**MOV DX, 0000H**

**MOV CX, 0000H**

**L1: MOV DX, 0000H**

**DIV BX**

**PUSH DX**

**INC CX**

**CMP AX, 0000H**

**JNE L1**

**L2: POP DX**

**ADD DX, 30H**

**MOV AH, 02H**

**INT 21H**

**LOOP L2**

**LEA DX,NEW**

**MOV AH,09H**

**INT 21H**

**RET**

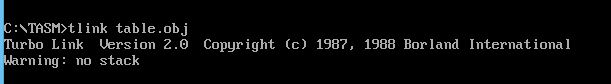
**DISP ENDP**

**END START;**

**ASSEMBLE:**



**LINKER:**



**OUTPUT:**

