COMMODORE 64 SOFTWARE

ASSEMBLER TUTOR

AST 6440 Diskette version

A comprehensive series of computerised lessons to aid in learning how to program in Assembly language. Supplements any book about 6502/6510 Assembly Language Programming.

commodore COMPUTER

ASSEMBLER TUTOR

AST 6440 Diskette version

A comprehensive series of computerised lessons to aid in learning how to program in Assembly language. Supplements any book about 6502/6510 Assembly Language Programming.

COMMODORE BUSINESS MACHINES (UK), LTD.
675 Ajax Avnue
Trading Estate
Slough, Berkshire SL1 4BG
ENGLAND

INTRODUCTION

This course shows you how to use Assembler on the COMMODORE 64. It assumes that you have already written programs in BASIC, and that you are conversant with the COMMODORE 64.

The aim is for you to understand how Assembler instructions work and their effect on the 651Ø registers and the COMMODORE 64, so that you can construct your own Assembler programs.

The course comprises three modules:

1. Information representation
2. Hardware and instructions
3. Software and programming

Each is relatively self-contained, and you can expect to spend 3Ø-4Ø hours to complete them all.

METHOD OF USE

Course modules are split into lessons, each designed to fit THE COMMODORE 64.

A self-test for each module enables you to determine which lessons to follow and to test your comprehension at the end. The programs should normally be run in sequence, and each will prompt with the number of the next. You can, of course, choose your own path.

If you are a beginner, you should follow the lessons in sequence.

If you already have a knowledge of an assembly-level language, you will find that you can skip some parts of the course, and need not study them in any detail. However, you do need to understand binary, and also the COMMODORE 64's hardware organisation. Do the self-test for the module and select your start point from its questions and from the lesson descriptions.

Text pages covering difficult concepts are normally followed by a demonstration, after which you can always go back and study them again. The ability to repeat a page and to back-track through previous pages in any one of the programs enables you to go at your own pace through the course.

When you finish a lesson, you may find it beneficial to repeat it; there may be points whose implications you did not appreciate, or understand, first time through. After finishing all of the lessons in a module, do the self-test to see if you should go back over any of them.

LOADING THE PROGRAMS

DISKETTE

To load the programs in the tutorial, do an 'absolute load' and type a question mark before the program name, e.g.:

```
LOAD"?n.n*",8,1
```

Where "n.n*" represents the program name, e.g. 1.2. Programs 3.5 and 3.6 do not need this absolute load. They may be copied for you own use only.

CASSETTE

To load the programs in the tutorial, press SHIFT and RUN/STOP together, this routine will load the first program found on the tape. Zero the counter on the cassette unit at the start of each tape. Having loaded a program note the counter reading. Use this reading to find the next program quickly or to refind a program previously loaded.

EXERCISES

Most lessons contain exercises to enable you to test your comprehension. If you repeat them, the program will usually vary them using different numbers. Have pencil and paper ready to use in the exercises. A calculator will also be useful, particularly in the first module (Information Representation).

When you try the programming exercises (Lesson 3.0) save your program before you run it; you may have to reset if you crash the 64. Be careful with the addresses you use and stay within the limits given.

BOOKS AND MANUALS

Many of the lessons introduce Assembler instructions but do not duplicate the contents of the manual. (You could not have a "programmed" manual in the 64 at the same time as entering a program!) Only a written manual gives you the detail you need; you must have one available whilst you study Modules 2 and 3. You may, however, need to make your own notes from some of the lessons - in particular where they give hints and instructions for using utility software.

The very readable book by R. Zaks "Programming the 65∅2" (Sybex) provides a useful work of reference, and contains many examples of Assembler programs. "65∅2 Programming" by Lance A. Leventhal (McGraw Hill) has more examples and hints on programming technique. The 651∅ uses the same instruction set as the 6502 and all but the most advanced programmers will find the differences inconsequential. The COMMODORE 64 Programmer's Reference Guide also contains details on assembly language programming.

MODULE 1 · INFORMATION REPRESENTATION

This module teaches you how data is held in binary, and how binary arithmetic and logical operations work.

1 Module instruction

1.0 Self-test

1.1 Binary numbers

This lesson introduces the concept of binary and describes numbering systems, what binary numbers are, bits and bytes, how to convert binary to decimal and how to convert decimal to binary.

1.2 Sign conventions

This lesson shows you how to express positive and negative values in binary. It explains the two's complement form of negative numbers, sign conventions in binary, how to convert a negative binary number into its positive form, and how to convert negative numbers to and from binary. It also introduces logical binary and the usage of each representation. Finally, it explains how to count in binary.

1.3 Binary arithmetic

This lesson explains the rules for adding and subtracting binary numbers, subtraction by adding two's complement, and carry from column to column. It introduces the concepts of carry and overflow, describing what carry signifies and when it can be ignored, why overflow occurs and conditions for carry and overflow to happen.

1.4 Logical operations

This lesson shows the diffference between logical operations on bit patterns and binary arithmetic by describing logical binary, types of logical operation, the workings of AND, inclusive OR (ORA), exclusive OR (EOR), how to simulate NOT and the purposes and usage of each.

1.5 Magnitude and fractions

This lesson explains some ways to handle large and small numbers by introducing the problem of number size, words and LO-HI format, and multi-byte operands. It shows how to do arithmetic on words, how bit shifting can cause precision loss, and explains fixed point binary representation, what a binary point is and the significance of the number of integral places in the operand.

1.6 Binary coded decimal

This lesson shows how to represent numbers in BCD format, and it describes the problem of precision loss, the inconvenience of binary fractions, BCD representation for precision, fixed point BCD, a convention for BCD operand format and how BCD arithmetic works.

1.7 Floating point

This lesson teaches the conventions in COMMODORE BASIC. It shows how a wide range of operands can be stored in a fixed number of bytes with 9-digit precision and enables you to use exponent/mantissa notation, rules for normalising a mantissa, floating point binary, signed numbers and exponents, conventions for exponent, mantissa, sign and the compacted format used for BASIC variables.

With this knowledge, you will be able to manipulate BASIC variables in Assembler. There are other conventions for floating-point, of course, and you can always invent your own.

1.8 Character codes

This lesson teaches you how to convert alpha-numeric characters and describes varieties of bit pattern, RAM, ASCII and screen code, how to convert from one to the other and how to convert to and from BCD.

1.9 Hexadecimal

This lesson teaches you how to use the hexadecimal conventions and describes the purpose of hexadecimal, reviews decimal numbering, introduces hexadecimal as a "shorthand" and explains numbering to base 16. It shows how to convert between hexadecimal and BCD, how to convert between hexadecimal and decimal and the usage of this notation for addresses.

MODULE 2 · HARDWARE AND INSTRUCTIONS

This module outlines the COMMODORE 64 architecture and the 651Ø MPU. It introduces the 651Ø instruction set. The aim is for you to understand the function of its parts and thus the purpose of the Assembler instructions which manipulate them. At the end, you should know what instructions are available, how they work and their correct format.

2 Module introduction

2.Ø Self-test

2.1 The COMMODORE 64

This lesson gives a simplified overview of the architecture of the COMMODORE 64. It describes the components of a computer system, types of memory (ROM, Video, RAM), processor chips (MPU, PIA, VIA), memory locations, address bus, data bus and control bus.

2.2 Memory

This lesson introduces the concept of a paged memory. It shows where Assembler programs may be sited and how they can be protected. It covers 16-bit addressing, paging, contents of pages Ø-3, page 4 for screen output, protection from BASIC, and pointer addresses in page Ø.

2.3 Stack

This lesson describes the 64's software stack, RAM page 1, the chronological nature of a stack, the software stack in page 1, the stack pointer, what the stack is used for, its cycle of operation and common pitfalls and mistakes to avoid.

2.4 651Ø microprocessor

This lesson introduces the registers in the MPU and gives an overview of their functions and relationships. It describes the program counter, status flags, accumulator, X and Y index registers and the stack pointer. It also points to instructions for data transfer between registers, data transfer to and from memory and for altering and testing status flags.

2.5 Accumulator and arithmetic

This lesson focuses on the accumulator and points to instructions for data manipulation. It describes the relationship between accumulator and ALU, loading of operands, the cycle of operation for addition, carry and word operands, increment and decrement, bit shifting and operand rotation instructions.

2.6 Addressing techniques

This lesson explains some of the address modes in 651∅ instructions and describes 651∅ instruction format, implicit addressing of registers, accumulator addressing for shifts. immediate addressing for constants and literals, absolute addressing – short and extended, relative addresses and branching.

2.7 Indexed and indirect addressing

This lesson completes the explanation of address modes, covering address calculation at run time, simple indexing, how to move a block of data, page ∅ indexing, indirect addressing, how to construct a 'tree jump', indexing for indirect addresses, differences in the use of X and Y index registers.

2.8 Interrupts

This lesson explains the concept of an interrupt. It describes program interruption, the interrupt sequence, 651∅ interrupts (RES NMI IRQ) IRQ occurences – clock, I/O, BRK, clock interrupt cycle, key addresses and RAM vectors.

MODULE 3 - SOFTWARE AND PROGRAMMING

The module is designed to get you started in developing and testing your own machine code routines and programs, using an assembly system. It assumes that you have studied and understood the earlier modules in the tutorial.

The aim is for you to understand what Assembler is, how to encode and decode the instructions, and how to link an Assembler routine with BASIC in the COMMODORE 64. At the end, you should be able to make well-structured programs, code them, and debug them.

The module has four lessons and also includes some simple software to help you to assemble and run your trial programs.

3 Module introduction

3.0 Programming exercises

Even if you know how to write Assembler it is difficult to decide what to write at first; the purpose of this lesson is to give ideas for small routines to enable you to practice and acquire skill in using the COMMODORE 64 in Assembler. They are carefully designed to exercise you in input and output routines, accessing and moving data and arithmetic. They mainly use the screen, so that you can see some results.

3.1 Assembler and Machine Code

This lesson describes the relationships and the differences between Machine Code and Assembler, covering language relationships, mnemonic opcodes, what Assembler is, binary opcode structure, how to disassemble Machine Code, addressing and symbolic addressing.

3.2 Program building and assembly

This lessons gives hints on creating and assembling a program by describing how to start with flowcharts, defining memory usage, the coding sequence, rules for symbolic names, program layout, how to use assembler directives, rules for EQUATE and .BYT, assembler expressions and how to get the program into the COMMODORE 64.

3.3 Linking with BASIC

This lesson shows you how to interface BASIC and Machine Code and covers entry and exit, how to use the screen, keyboard input, how to handle the keyboard buffer, the STOP key, accessing BASIC variables and accessing arrays.

3.4 Software aids

The lesson reviews some programming software and gives instruction in its use. It describes the need for programming aids, the small ASSEMBLER (3.5) provided and how it works, general instructions for its use, the Machine Language Monitor and its commands, and gives an overview of EXTRAMON.

3.5 Assembler

This is a BASIC program. You add Assembler source text to it as DATA statements, and it POKEs Machine Code into the memory. It uses the COMMODORE Assembler Development System conventions (lessons 3.1 and 3.2) for the source text, and a few of its directives for assembly.

Save the whole program, with your DATA statements, to make any subsequent changes easy to do. It is slow and only suitable for small Assembler routines (such as would fit into the cassette buffers).

3.6 Machine Code Saver/Loader

This program uses PEEK to extract your Machine Code and adds it to itself as DATA statements (hexadecimally coded). You can then save it and can add extra statements to it.

It incorporates a load routine which POKEs Machine Codes into the locations from which they were saved. You cannot use it to save Machine Code from the BASIC text area starting in page 8. If your Machine Code lies at the end of memory, make sure that after you alter the memory limit and start of string pointers **before** you run the program.

LIST OF PROGRAMS ON TAPES

TAPE 1

0.0 gen intro
1 module intro
1.0 self test
1.1 bin numbers
1.2 sign conv'n
1.3 bin arithmc
1.4 logical ops
1.5 magtude/frn
1.6 bcd
1.7 float point
1.8 char codes
1.9 hexadecimal

TAPE 2

2 module intr
2.0 self test
2.1 cbm 64
2.2 memory
2.3 stack
2.4 6510 mpu
2.5 accum/arith
2.6 addressing
2.7 indxd/indir
2.8 interrupts

TAPE 3

3 module intro
3.0 prog exc's
3.1 ass/mc code
3.2 prog bldg
3.3 link-basic
3.4 s'ware aids
3.5 assembler
3.6 saver/loader

DISK DIRECTORIES

Module 1

```
0    "assembler tutor " 01 2a
1       "data"                seq
43      ""0.0 gen intro       prg
43      ""1   module 1        prg
36      ""1.0 self test       prg
43      ""1.1 bin numbers     prg
40      ""1.2 sign convns     prg
40      ""1.3 bin arithmc     prg
40      ""1.4 logical ops     prg
41      ""1.5 magtude/frn     prg
38      ""1.6 bcd             prg
42      ""1.7 float point     prg
30      ""1.8 char codes      prg
39      ""1.9 hexadecimal     prg
28      " 3.5 assembler"      prg
16      " 3.6 save/load"      prg
0    blocks free.
```

Modules 2 & 3

```
0    "assembler tutor " 23 2a
1       "data"                seq
43      ""2   module 2        prg
41      ""2.0 self test       prg
40      ""2.1 commodore64     prg
48      ""2.2 memory          prg
40      ""2.3 stack           prg
39      ""2.4 6510 mpu        prg
40      ""2.5 accum/arith     prg
41      ""2.6 addressing      prg
41      ""2.7 indxd/indir     prg
40      ""2.8 interrupts      prg
43      ""3   module 3        prg
36      ""3.0 prog exercs     prg
40      ""3.1 ass/mc code     prg
41      ""3.2 prog bldg       prg
39      ""3.3 link-basic      prg
46      ""3.4 s'ware aids     prg
0    blocks free.
```

COMMODORE 64K SOFTWARE

Commodore Business Machines (UK) Ltd
675 Ajax Avenue, Slough
Berks, SL1 4BG.

Made in England.

commodore COMPUTER