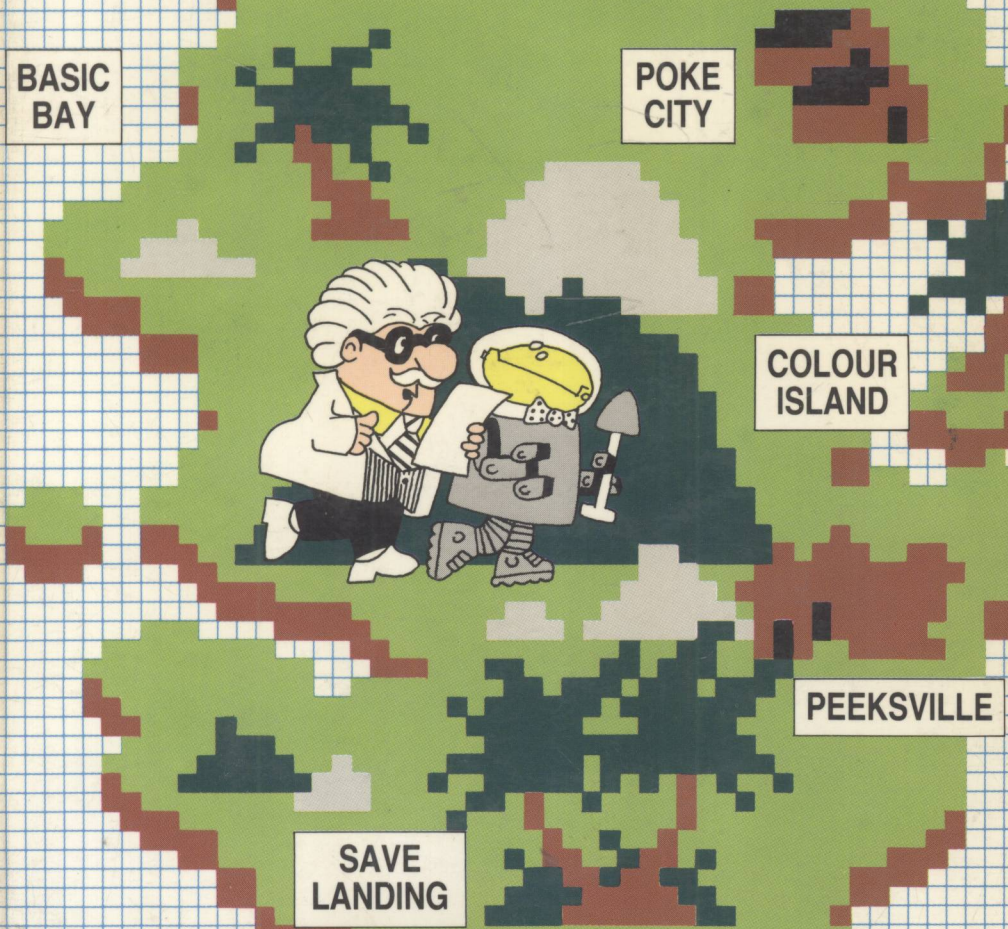
THE MASTER MEMORY MAP FOR THE COMMODORE 64

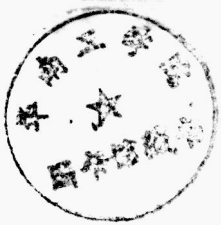


Paul Pavelko and Tim Kelly

MASTER MEMORY MAP FOR THE COMMODORE 64

A GUIDE TO THE INNER WORKING OF THE COMMODORE 64's BRAIN CELLS

by
PAUL PAVELKO
and
TIM KELLY



Englewood Cliffs, NJ London New Delhi Rio de Janerio
Singapore Sydney Tokyo Toronto Wellington

ISBN 0-13-574351-6

Commodore 64 is a trademark of Commodore Business Machines.
Professor von Chip and Prototype are trademarks of
Educational Software, inc.

10 9 8 7 6 5 4 3 2 1

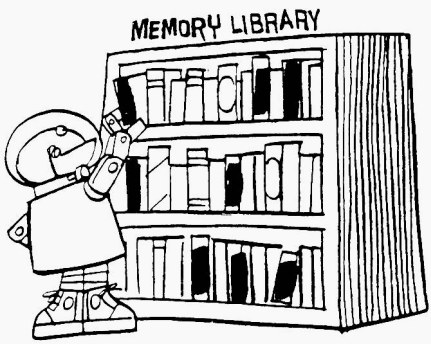
Printed in Great Britain by A. Wheaton & Co. Ltd., Exeter

TABLE OF CONTENTS

PRELUDE
TO THE COMMODORE 64 MASTER MEMORY MAP

Welcome all, beginning or expert programmer, to ESI's COMMODORE 64 MASTER MEMORY MAP! This book will be your guide into the inner working of the Commodore 64's 'brain cells'. This is truly a map, a guide to the special places inside the operating system of the computer. These places will help you add new features to the programs you write, making them really come alive!

Along the way, you have the humor of Professor Von Chip and the friendly alien Prototype to help make the journey a productive one.



The Master Memory Map is divided into sections to aid you. Each section deals with a particular part of the memory. There are lots of programming examples, too, because sometimes it's easier to understand an example when you see it on the screen instead of just reading it. Some of the programming examples add a useful utility to BASIC, like the RENUMBER routine. Others serve as useful programming 'tricks'. In every case, you should study the listings and play with the code to see what happens.

The appendices go into more detail, showing how to do something like create sprites or produce a sound. They give longer programming examples and show you some of the advanced things you can do with the Commodore 64.

This book really isn't a novel, so you can start reading anywhere. But sometime you should read it from cover to cover; sooner or later you'll see new ways to use the computer. This moment of enlightenment - a creative flash - is what makes working with a computer so much fun!

We've worked hard to make the Master Memory Map easy to read and use. Look in the upper right hand page corner for a guide that will show you which locations are covered on those pages. Just flip through the book until you come to the locations you need. The appendix sections are also marked in a similar way.

Prototype, sometimes just called Proto, will help you find locations and routines that the beginning or intermediate programmer will use most often. Look for him in the margin as you flip through the book.

A COPY OF THE PROGRAMS

The programs in this book are used as illustrations for techniques and ideas. You will gain a lot of knowledge if you type in the programs yourself. But if you don't want to tire your fingers, send $9.95 to:

Educational Software, inc.
4565 Cherryvale Ave.
Soquel, CA 95073

A BONUS!

If you discover a new, unpublished use for one of the memory locations send it to Educational Software. In return, we'll send you some software, free.

SOURCES

A few of the program examples in the Master Memory Map come from other sources. The source is identified in the text by using these symbols.

C=: COMMODORE 64 PROGRAMMER'S REFERENCE GUIDE, Commodore Business Machines, Inc., Computer Systems Division, 487 Devon Park Drive, Wayne, PA 19087.

C!: COMPUTE! Magazines, Copyright 1982, Small System Service, Inc., Reprinted by permission from COMPUTE! Magazine, P.O. Box 5406, Greensboro NC 25403, 12 Issue, Subscription $20.00.

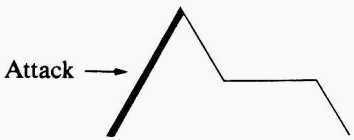
GLOSSARY

ASCII: The American Standard Code for Information Interchange. This is one standard for assigning numbers to the letters and characters on the keyboard. Commodore computers do not follow a true ASCII (pronounced 'ASK KEY') but have their own code instead.

Accumulator: The results of logic and arithmetic operations are stored here temporarily. It acts as a busy bus stop, nobody stays here long!

Address: The number of a given location. It's just like a street address.

Attack: The rate a note or sound changes from 'off' to its highest volume.



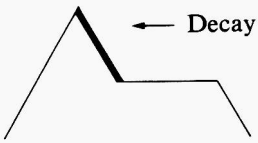
Baud: This is the rate of transmittion of information conveyed over a line . This rate is determined by the bits per second that are being transfered. You encounter this term if you are using a modem or some device that requires special interfacing (RS-232).

Bit: The smallest piece of information the computer can handle. There are eight bits in a byte.

Buffer: A storage place. For example, the keyboard buffer stores your keystrokes and allows you to type faster. The information in a buffer eventually goes somewhere else to be acted upon.

Bus: A bus is a system of electrical lines shared by all devices that are connected to it. This is a convenient way for these devices to share addressing and data. It works just like a party line telephone.

Decay: A musical term meaning the rate of change from the highest level to the sustaining level of sound.



Default: The beginning value of a memory location especially when the power is turned on or other operations are done.

Disable: Turn off. By disabling the RUN/STOP key, you can prevent anyone from accidentally stopping your program.

Enable: To turn on; the opposite of disable.

Flag: A signal that something has happened. Flags can be used in your own programs. For example:

```
If A$ = "ouch" then B = 1
```

B is the flag in that statement.

Floating Point: Arithmetic operations using decimal numbers.

Immediate Mode: Using the computer without running a program. For example:

```
10 PRINT 3 + 2
```

is a program and must be run to get an answer.

```
PRINT 3 + 2
```

will answer "5" when you press RETURN.

Jump: To go from one location to another. In BASIC, the equivalent terms are GOTO and GOSUB.

KERNAL: This is Commodore's word for a series of machine language subroutines that operate the computer. See the Appendix for more information.

Nybble: Pronounced 'nibble'. A nybble is half a byte. Really. The low nybble is composed of bits 0 to 3. The high nybble has bits 4 - 7.

Operating System: Sometimes this is referred to as the OS. Part of this is the KERNAL described above. Its job is to make the computer run.

Page: A page is 256 bytes of memory. The computer often keeps track of different blocks of memory in terms of pages since it is easier for the computer to store.

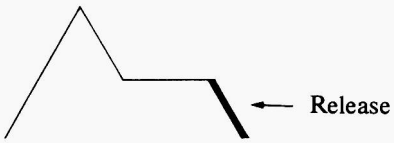
Pointer: It does just what it says, it acts as a signpost, telling the computer where to look for information.

RAM: Random Access Memory. This type of memory can be easily changed. Your programs are stored in RAM, and when the computer is turned off any information in RAM is lost.

ROM: Read Only Memory. This type of memory does not change when the power to the computer is turned off. Examples of ROM memory include BASIC and the KERNAL.

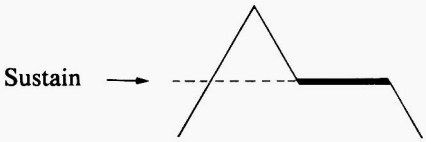
Register: Another name for memory location. Registers can be more than one byte long.

Release: A musical term describing the rate of fall from the sustain level to zero volume.



Reserved Word: Letters that can't be part of your program. Examples of reserved words are the status word, ST, and the time words, TI and TI$. To save yourself from trouble, don't use any variables in your programs that have the same starting letters of any BASIC command or have BASIC words in them.

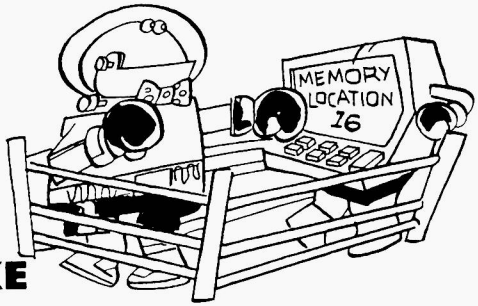
Sustain: Another musical term used with the sound capabilities of the computer. This extends the sound, like the pedals of a piano.



Waveform: This term is a description of the type of sound produced. The computer has four different waveforms; triangle, sawtooth, pulse and noise. Each type produces a different kind of sound.

Vector: This is another kind of pointer. It refers to the starting address of a routine. The computer needs to know where to look for things.

How to PEEK and POKE



This part is for those who have yet to learn how to use a memory map. Basically, a memory map is a list of valuable locations within the computer (in this case a COMMODORE 64), that you can directly use for various purposes. These locations are called bytes (memory locations) of memory at a specific place. With 64K of memory, there are 64*1024 memory locations that you can work with. Although some of these bytes are used for the computer's Operating System, most of them are blank for you to use in your programs. This manual will tell you about the ones that you can do something with.

For example, you can quickly look down this list to find that location 650 controls the repeating of certain keys, like the space bar. By following the included hints, you can change the "normal" value in that location, so that when you press *any* key it will repeat as long as you hold it down. Please note that any of the changes that you make are only temporary and will go away when the computer is turned off.

Now to explain how to make changes from BASIC. Say you look down the list and decide to change location 650 (all numbers are decimal unless marked with a $ symbol, which denotes a HEX-adecimal number, or in a column marked "hex"). 650 is called RPTFLG by Commodore. Its function is to decide which keys on the keyboard to repeat. So, if you would like all the keys to repeat as long as you hold the key down, you simply look up the correct value to POKE into location 650.

In this example, the memory map says to use the number 255 to repeat all keys. The BASIC instruction to put a number into memory is called "POKE". After all this long-winded explanation, you can now see how simple it is to make this change:

POKE 650,255

— HINT —

Always use the decimal numbers with a POKE statement. This means that sometimes you will have to convert between binary, hexadecimal and decimal. Also, any one memory location can only hold a number up to 255. Why?...remember that the COMMODORE uses eight bits per word (memory location), and eight bits in binary counts from 0 to 255 (internally, the machine uses binary). You may want to study the next section of the manual, "Bytes and Bits" to learn about binary. Because of this limitation, sometimes you must POKE numbers into two locations in a row.

For example, look at memory locations 643 & 644 which are called MEMSIZ. These locations hold a number that corresponds to the top of your available memory (called RAM - Random Access Memory). Since the top of memory can be up to 40960, a number well above the limit of 256 for any one memory location, the computer will need two locations to store the value. Yes, I know 256 for the first location plus 256 for the second doesn't seem to add up to a large enough number to hold 64K, but the computer takes each number in the second location and multiplies it by 256. Examples:

```
 11 stored in the low byte
+1 stored in the high byte
---
267
```

The computer "sees" 256*1+11 which equals 267.

Another example:

```
 121 in the low byte
  +7 in the high byte
----
1913
```

Since 7*256 +121=1913.

Sometimes it is desirable to fool the COMMODORE 64 into thinking that the top of memory is lower than it actually is, perhaps to keep it from using the last thousand bytes of memory, thus reserving them for Sprites. You do the same type of POKE here as in the first example, except that you have to do it twice; once for the "LOW" part of the number and once for the "HIGH" part.

I said the LOW part of the number is placed in the first memory location and the HIGH part is next. Although it seems backwards, this is really not hard to understand. The COMMODORE (and most other micro-computers) store multiple part numbers this way. Occasionally this rule is broken, so please don't call me up if you find an exception.

Here's what you do. We want to change the value of MEMSIZ to be 1K less than it currently is.

1) Find current value...

```
10 A = PEEK(643)+PEEK(644) * 256
       LOW Part      HIGH Part
```

This number will be 40960, which is the value of MEMSIZ, when the 64 is turned on.

2) Subtract 1K from this value...

```
20 A = A-(1 * 1024)
```

Remember that one K is actually 1024 bytes.
(You don't have to use 1, you can can change the size by any number.)

3) Break the new value up into LOW and HIGH parts...

```
30 B = INT(A/256): C = A-(B * 256)
```

B would = 156, C would = 0

What this does is make C the LOW part of the number and B the HIGH part.

EX: 40960-1K = 39936

Line 30, when run, will give you 156 for the HIGH part and 0 for the LOW part.

4) POKE these values into memory...

```
40 POKE 643,C: POKE 644,B (#'s in decimal!!)
```

EX: POKE 643,0:POKE 644,156

!! FINAL WORDS OF WISDOM !!

1) Feel free to POKE and PEEK all you want, trying out ideas or testing the effects mentioned in the Master Memory Map. The explanations are only the most basic part of how to do the various effects possible on the COMMODORE. Watch for EDUCATIONAL SOFTWARE'S series of TRICKY TUTORIALS™ that will take you step by step through Sprites, Page Flipping, Sound, Animation and other uses of the computer. These are the techniques that the best programs use, and all of our Tutorials are done in BASIC, although we do sometimes include a machine language subroutine to offer you some advantage like speed.

2) Remember that two numbers are required to tell the computer the value for some locations, and these are stored LOW part first, HIGH part second. This is opposite of what you might think.

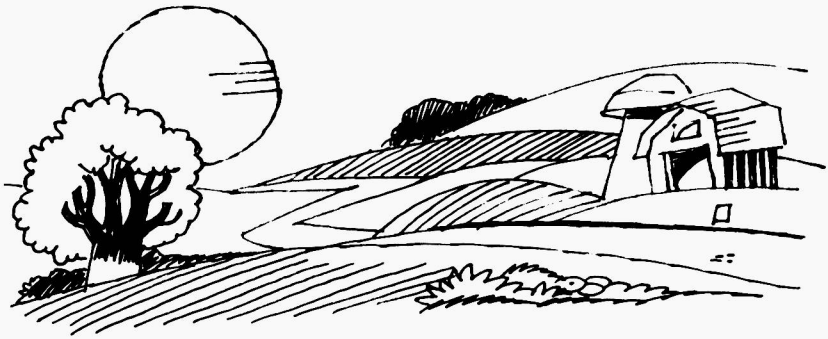
3) All of the memory locations are here, but many are for advanced users only. Don't feel bad if you have no idea what they are for. The idea is to experiment and learn.

4) You can usually press Run/Stop and Restore if trouble occurs. This will restore the original (default) values of many locations.

5) Some locations in the Master Memory Map are used to read from only; that is, you can PEEK to see what is there, but you can't POKE your own number in. This is because part of memory is a type called "ROM" which means "READ ONLY MEMORY". This type is permanent and can't be changed by POKEing, but Commodore has thoughtfully provided a way for you to put a copy of the ROM into memory so you can change it if you wish.

Go back and re-read the last section at least a few hundred times. There are only four lines in the program that both read the old value of MEMSIZ and store a new value. These lines don't have to be part of a program. You could enter them directly.

BYTES and BITS



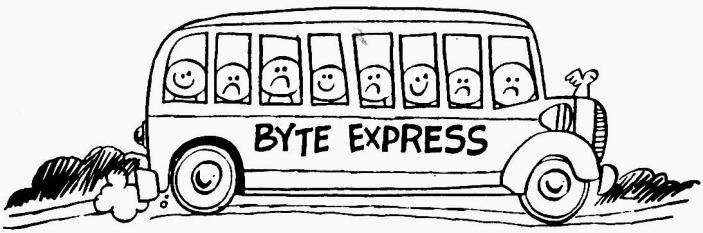
To PEEK & POKE you need to understand what a byte is and how it is structured. It isn't too hard to understand - and you can use the Master Memory Map without learning too much about them - but the more you know about Bytes and Bits the more you'll learn about controlling your Commodore.

A BYTE is really not complicated at all. A BYTE is simply a group of eight BITS. When eight BITS are structured into a BYTE then each of those BITS have special significance. You look puzzled! What, you say, is a BIT?

A BIT is the smallest piece of information a computer can deal with. In fact, the computer only deals with BITs at the most fundamental level. It may be helpful to imagine the microprocessor as a bus station. This bus station has only one single lane road attached to it. That means a bus can only travel in one direction at a time as there is not enough room for two busses to pass each other. Therefore, a bus may either be arriving at the station or departing. The microprocessor, or bus station, can schedule its bus with a signal light that says "I AM ACCEPTING ARRIVALS" or "I AM SENDING DEPARTURES".

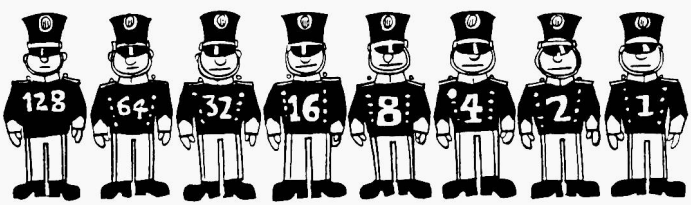
In fact, in real computer hardware architecture, the wires that carry information to and from a microprocessor are called the DATA BUSSES. We don't need eight separate INPUT and eight separate OUTPUT wires because, like the single lane road connected to the bus station, the wires are bi-directional, that is, information can either be arriving (INPUT) or departing (OUTPUT). The microprocessor also has a signal of its own that determines whether it will receive (INPUT) or send (OUTPUT) information.

WHO'S RIDING THE BUS?



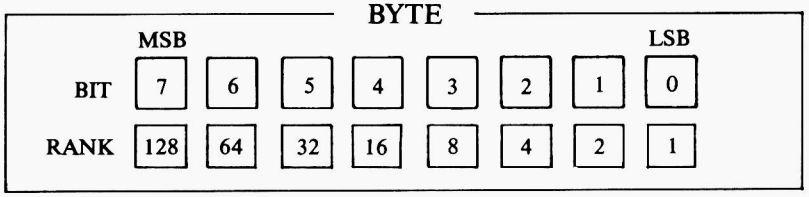
Let's take a closer look at that bus. It is known as the BYTE express, has eight seats, and always carries eight passengers. Those passengers are little messengers known as BITs, and, as a group, they are known as a BYTE. These messengers, or BITs, are rather moody. They are either turned "ON" or they are turned "OFF". That is called BINARY as they are BI-STATE signals, ON being a "1" state or OFF being a "0" state. Their vocabulary is just as limited...the only thing they are willing to tell you is their mood. Now how do we get any meaningful information out of a group of eight little monsters standing in front of us, each screaming "ON" or "OFF" at one time?

Well, when the bus arrives, we could have the whole BYTE stand in front of us and count everyone who is turned "ON". That would give us the capability of counting to eight. Seems pretty limited, doesn't it? Hmmm, the group really needs a leader. That leader will be the first BIT on the left. We'll call that BIT the Most Significant Bit, or the MSB. The last BIT on the right will be the Least Significant Bit, or LSB. Terrific! Now that we have a group leader and group follower, then all the BITs should have a rank.



Handing out ranks is a serious matter and much thought should be given to it. We can start with the LSB and assign that BIT the rank of "1", since it is the Least Significant Bit. We can be easy on everyone if

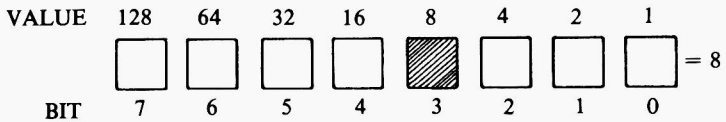
we just double that rank for the next BIT in line. So, why not just keep doubling the rank for the next BIT in line and so on until we get to the MSB or Most Significant Bit. Now out BYTE looks something like this:



MESSAGES WITH MEANING!

What have we gained? More than meets the eye! When the BYTE gets off the BYTE express and each BIT starts telling us what their mood for the day is, we can make a different and more meaningful interpretation out of the ignorant little beasts. If everyone is turned "OFF" except the fourth BIT from the right we can check the rank of that BIT and find it is eight (8). Unknown to the BITs, they have brought us a message and the message is "8".

Computers are very efficient and do not like to waste information, therefore, computer related numbers usually start at BASE ZERO (0) since zero is unique. We usually like to start counting with one (1) for the convenience of our thinking process. That way, the number we arrive at when we have counted the last item actually represents the number of items we counted. Normally, we would count the BITs as one (1) through eight (8). The computer thinks a little more efficiently than mortals and sees BITs zero through seven as representing eight (8) individual BITs.



If you SET or turn "ON" only the fourth BIT (BIT #3) from the right you will observe a value of eight (8) in the value box. That was the message we received!

Aha, we now want the BITs to get on the bus and carry a message back to the sender. We want them to tell the sender "9". Oops, a small