02/04/95
ASSEMBLY LANGUAGE/MACHINE LANGUAGE/CODING TUTORIAL -- Part ONE
BY SCATT

INDEX

PREFIX

Hello Everyone..

This is my attempt to catalog everything I learn about Machine Language (referred also as Machine Language and Coding) and put it into a simple format for everyone who is interested in learning to try it out for themselves. Every program that I have seen has been a bit too much for me to comprehend, and too far advanced for me. So this is my attempt to teach Assembly Language for the Commodore 64. Good luck, and if you want to reach me for questions, please contact me at ex240@cleveland.freenet.edu or as327@freenet.buffalo.edu

I AM NOT A PROGRAMMER! ALL I KNOW AS OF THIS VERY MOMENT
IS A SMALL AMOUNT OF BASIC, so please, Don't assume I know what I am

talking about. Let's just hope that the sources that I took all of this data from were accurate. If you have something to dispute about this, please e-mail me, and I will try to make updates. If you learn anything new, that is not documented within the scope of this document, please, write to me, and we'll see what we can find out TOGETHER!

Regards, SCATT

PS: If you see a number in parenthesis after a quote (i.e. "text"(4)), this means that the preceding text was taken from another source. Look at the Bibliography in the end of this text file for the source.

Part ONE : The Basics - An Introduction.

THE MAIN REASON for learning Assembly or ML is this: It is FASTER, and SMALLER (memory-wise) then BASIC programs (which stands for Beginners All-purpose Symbolic Instruction Code), and (ML Programs) give you an insight to how the computer operates. And best of all, It brings us CLOSER to the computer (which is every computer geek's goal!) haha..

"THE BEST WAY TO LEARN ANY PROGRAMMING LANGUAGE IS TO PROGRAM IN THAT LANGUAGE."(7)

"BASIC might be compared to a reliable, comfortable car. It will take you where you want to go. Machine language is like a sleek racing car - you get there with lots of time to spare. When programming involves large amounts of data, music, graphics, or games - speed can become the single most important factor."(2)

"So, which language is best? (BASIC or ML) They are both best - but for different purposes. Many programmers, after learning ML, find that they continue to construct programs in BASIC, and then add ML modules where speed is important. But perhaps the best reason of all for learning ML is that it is fascinating and fun."(2) :)

OK let me tell you one other thing before we start. I assume (making an ASS out of U and ME) that you understand how your computer basically works. I am not going to attempt to "take a quick tour of the computers internal parts," so please go get a book about this, ok? :)

There are definitions all over this thing (so TAKE NOTES!!) to explain some of the terms but that's as far as I'm gonna go with it. An example of what you should already know is like what exactly memory is! What is memory? It is actually little switches and each one can have two states: on or off! Did you know that? IF NOT, then this is not for you! Well, not yet that is! Do you know what I/O, ROM, RAM, etc is? IF NOT, again, this is not for you YET! You need to start out elsewhere! I don't mean to be rude, but we all have our starting points! OK? Now SMILE! And do what must be done in order to get up to this point. Machine Language programming is not something to rush into...

There are A LOT of books around this wide planet, so whether you get your information from comp.cbm or a library, or whatever, ask people! Visit your library! GO! GO NOW! Don't wait another minute or else it's gonna be too late!!!!!!!!!   :)

First Steps

I would recommend that you either get a Commodore 64 (If you don't already have one) or a good emulator program. One emulator I recommend is C64S. Ask around, especially on IRC #c-64. They should all know where to get it.

Once you have your C64 or emulator, I recommend you get an Assembler. Again, ask around. You will have one in no time.

One other thing: "Many of the first home computerists in the 1970's learned ML before they learned BASIC. This is because an average version of the BASIC language used in microcomputers takes up around 12,000 bytes of memory, and early personal computers (KIM, AIM, etc.) were severely restricted by containing only a small amount of available memory. These early machines were unable to offer BASIC, so everyone programmed in ML."(2) So hey! ML is not more difficult to understand than BASIC. (But sometimes more of a challenge to debug) But it's not too far beyond BASIC. So DIG IN ALREADY!

Processors

Another thing: I'm not sure which processor is in the different versions of the C=64. I have seen 6502, and 6510. When I figure it out, I will update this again! As of this point, I am not sure that all of the commands in this book will work on the C=64. We will learn together though, won't we!

Well, I found some more info on the CPU. "The heart of your machine (C=64) is the 40-pin chip just to the left of the RF modulator can. (He is talking about the old-style case) This is the 6510A microprocessor."(4)
He also states that "This 40-pin custom chip operates like a 6502 MPU (also known as CPU) except the 6510 has a built-in 6-bit peripheral I/O port that controls memory management and cassette I/O."

Bits and Bytes!

"It's interesting that the word "bit" is frequently explained as a shortening of the phrase BInary digiT. In fact, the word bit goes back several centuries. There was a coin which was soft enough to be cut with a knife into eight pieces. Hence, pieces of eight. A single piece of this coin was called a bit and, as with computer memories, it meant that you couldn't slice it any further. We still use the word bit today as in the phrase "two bits" meaning 25 cents."(2)

A byte is 8 bits of data that may be loaded together into a register. A register holds 1 byte. The 6502 can only affect 1 byte in one operation. Because the 6502 can perform hundreds of thousands of operations a second, it can affect 100's of 1000's of bytes per second. In fact, "the Commodore 64 can handle about 500,000 of these steps each second." This is from the C-64 Troubleshooting & Repair Guide by Robert C. Brenner.

Number Systems

DECimal Numbers: We all know what these are, like 0,1,2,3 etc. These are base 10 numbers. ML can be accomplished in Decimal, but very rarely seen.

*BINary Numbers: Binary numbers are base 2 numbers. All we have to remember in Binary numbers is 0's and 1's. It's supposedly how the computer "thinks". What I take this as is that it's the way the processor sends and receives data internally (through it's 8-bit channel.) with 1's (or positive voltage) and 0's or a lack of voltage. All digits and numbers are converted to BIN. The easiest way to convert DECimal numbers to Binary is this:

```
 Place      0 0 0 0    Here we have 1's place, 2's place,
  Holder->  8 4 2 1    4's place and 8's place and so on..
            -------
 Bin Num->  0 0 0 0    Here's the binary number..
```

So, if we have a binary number of let's say, 0101, then we just add up the place's numbers and see what decimal number we get.. So we have a 1 in the 4's place, so that's decimal #4. We have no 8's or 2's and we have 1 in the 1's place. So if we add the 4 to the 1, we get a decimal of 5. So, if we had let's say a decimal number of like 12, we would know that there is at least one 8, and a 4, and we come up with 1100(bin)=12(dec)! Try some on your own and get familiar converting these back and fourth.....

BINARY	DECIMAL	BINARY	DECIMAL
0000	0	0110	6
0001	1	0111	7
0010	2	1000	8
0011	3	1001	9
0100	4	1010	10
0101	5	1011	11

The Bit significance and the byte..

Bit Number:	b7	b6	b5	b4	b3	b2	b1	b0
Bit Significance:	128	64	32	16	8	4	2	1
Binary Number:	0	0	0	0	0	0	0	0

This would be an 8-BIT Binary number. Often written as 0000 0000. Understood? Kool. So the Decimal number "25" would convert to what? Yup, you got it, 0001 1001 !!!

The rightmost Bit=Bit 0 (Tells us whether we have a 1 in our byte) The next to the left (Bit 1) tells us whether we have a two, etc..

And we go ON!

*HEX Numbers: Hexadecimal Numbers are Base 16. "HEX" for 6, and DECI for 10, so when you add them, 6+10=16!!! :) Kool. That is, multiples of 16. 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f. When we program (or the new word seems to be "code" or shall I say the "in" word haha..) So when we CoDe, we use a "$" to represent HEX numbers. Remember this. Put it into your ROM and KEEP IT THERE! It is important!

"See how hex $10 (see the dollar-sign?) looks like binary? If you split a hex number into two parts, 1 and 0, and the binary (it's an eight-bit group, a byte) into two parts, 0001 and 0000 - you can see the relationship."(2)

Remember when I did this: 0000 0000? Well, some people consider one of those sets of 4 bits to be a "nybble". To represent a byte (8-bits) in HEX notation, divide the 8-bit byte into two 4-bit units (yup, that's a nybble). Each of the 4-bit units (or nybbles) has a value of from 0 to 15 (decimal) which we express with a single hexadecimal digit! So you can use just ONE hexadecimal digit to represent 1 nybble (4-bits)! Isn't that kool! Now you remembered that the "$" represents the HEX notation, right? Well, check out this chart:

HEX		DECIMAL
---		----
$0	=	0
$01	=	1
$02	=	2
$03	=	3
$04	=	4
$05	=	5
$06	=	6
$07	=	7
$08	=	8 (gee this gets boring..)
$09	=	9
$0A	=	10 (what's this? WO! an "A"!!!)
$0B	=	11
$0C	=	12
$0D	=	13
$0E	=	14
$0F	=	15
$10	=	16
$11	=	17
$12	=	18
$13	=	19
etc		
etc		

So there we have it..

Here's another way to put it:

"   DECIMAL   0 1 2 3 4 5 6 7 8 9   then you start over with 10

      HEX   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F then you start over with 10"(2)

Let me go and see if I can find some text on how to mathematically convert decimal to hex.. I'll be right back..

Well, I didn't find what I was looking for, but I found this little charm..

"Microsoft Hex-Decimal Converter"(2)

```
1     HE$="0123456789ABCDEF"
2     ?"{CLEAR}{03 DOWN}PLEASE CHOOSE:
4     ?"{03 DOWN}{03 RIGHT}1-INPUT HEX & GET DECIMAL BACK.
5     REM NEW LINE HERE
6     ?"{02 DOWN}{03 RIGHT}2-INPUT DECIMAL TO GET HEX BACK.
7     GET K:IF K=0 THEN GOTO 7
9     ?"{CLEAR}":ON K GOTO 200,400
100   H$="":FOR M=3 TO 0 STEP -1:N%=DE/(16^M):
      DE=DE-N%*16^M:H$=H$+MID$(HE$,N%+1,1):NEXT
101   RETURN
102   D=0:Q=3:FOR M=1 TO 4:FOR W=0 TO 15:
      IF MID$(H$,M,1)=MID$(HE$,W+1,1) THEN GOTO 104
103   NEXT W
104   D1=W*(16^(Q)):D=D+D1:Q=Q-1:NEXT M
105   DE=INT(D):RETURN
200   INPUT"{02 DOWN}HEX";H$:GOSUB 102:
      PRINT SPC(11)"{UP}= {REV}"DE"{LEFT} "
210   GOTO 200
400   INPUT"{02 DOWN}DECIMAL";DE:GOSUB 100:
      PRINT SPC(14)"{UP}= {REV} "H$" "
410   GOTO 400
```

Something useful: "To figure out a HEX number, multiply the second column by 16 and add the other number to it. So, $1A would be one times 16 plus 10 (Recall that A stands for ten)."(2)

Well, since I sent in my $$ to register "The PC Assembler Tutor" and never got
anything back from the guy, I will ASSUME (ASS-U-ME) that Mr. Nelson won't mind me reproducing this next goody without his consent. (Although I did mention his name to keep him happy! :)

```
                HEX              CONVERT       BINARY
                ---              -------       -------
                "3         ->    2 + 1     ->   0011
                 9         ->    8 + 1     ->   1001
                 F = 15    ->    8+4+2+1   ->   1111
```

All computers operate on binary data, so why do we use hex numbers? Take a test. Copy these two binary numbers:

```
1011 1000 0110 1010 1001 0101 0111 1010
0111 1100 0100 1100 0101 0110 1111 0011
```

Now copy these two hex numbers:

```
B86A957A
7C4C56F3
```

As you can see, you recognize hex numbers faster and you make fewer mistakes in transcription with hex numbers.

ADDITION AND SUBTRACTION

The rules for binary addition are easy:

```
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0  (carry 1 to the next digit left)
```

similarly for binary subtraction:

```
0 - 0 = 0
0 - 1 = 1  (borrow 1 from the next digit left)
1 - 0 = 1
1 - 1 = 0" (8)
```

OK.. I hope that clears some stuff up.. Well, for now, I can't find much on converting Decimal numbers to Hex, so as the book states "Even the sketchiest understanding of hexadecimal math, however, should be sufficient for you to follow and use (assembly)"(1)

and...

"You need not memorize (HEX NUMBERS) beyond learning to count from 1 to 16 - learning the symbols. Be able to count from 00 up to 0F. (By convention, even the smallest hex number is listed as two digits as in 03 or 0B."(2)

So, what I would recommend you do (and what I will be doing before not too long) is copying a DEC to HEX table from somewhere (or just make your own) and tape it in front of you, avoiding the monitor you are using for a billboard, and you will then know how to convert DEC to HEX or visa versa.

As I've heard somewhere before, and also very useful, "Most ML programming
involves working with hex numbers only between 0 and 255. This is because a single byte (8-bits) can hold no number larger than 255. Manipulating numbers larger than 255 is no real importance in ML

programming until you are ready to work with more advanced ML programs. For example, all 6502 ML instructions are coded into one byte, all the "flags" are held in one byte, and many "addressing modes" use one byte to hold their argument."(2)

A little on Computer MEMORY

I'm sorry to use so many quotes, but everything I've found seems so useful, and I am learning so much from all of this info, I just can't stop! And all the typing is very good for my fingers..

"THE CITY OF BYTES

Imagine a city with a single long row of houses. It's night. Each house has a peculiar Christmas display: on the roof is a line of eight lights. The houses represent bytes; each light is a single bit. If we fly over the city of bytes, at first we see only darkness. Each byte contains nothing (zero), so all eight of its bulbs are off. (On the horizon we can see a glow, however, because the computer has memory up there, called ROM memory, which is very active and contains built-in programs.) But we are down in RAM, our free user-memory, and there are no programs now in RAM, so every house is dark. Let's observe what happens to an individual byte when different numbers are stored there; we can randomly choose byte 1504. We hover over that house to see what information is "contained" in the light display.

```
     ____.____.____.____.____.____.____.____.____(“.”=off, “o”=on)
```

Like all the rest, this byte is dark. Each bulb is off. Observing this, we know that the byte here is "holding" or representing a zero. If someone at the computer types in POKE 1504,1 - suddenly the rightmost light bulb goes on and the byte holds a one instead of a zero:

```
     ____.____.____.____.____.____.____.____o____
```

This rightmost bulb is in the 1's column (just as it would be in our usual way of counting by ten's, our familiar decimal system). But the next bulb is in a 2's column, so POKE 1504,2 would be:

```
     ____.____.____.____.____.____.____o____.____
```

And three would be one and two:

```
     ____.____.____.____.____.____.____o____o____
```

In this way - by checking which bits are turned on and then adding them together - the computer can look at a byte and know what number is there. Each light bulb, each BIT, is in its own special position in the row of eight and has a value twice the value of the one just before it:

```
     ____o____o____o____o____o____o____o____o____ = 255!
       128's 64's 32's 16's 8's  4's  2's  1's
```

65535 is an interesting number because it represents the limit of our computer's memories. In special cases, with additional hardware, memory can be expanded beyond this. But this is the normal upper limit because the 6502 chip is designed to be able to address (put bytes in or take them out of memory cells) up to $FFFF."(2)

ASCII

"Instead of a number from 0 to 255, an 8-bit byte can be used to represent an upper or lower case letter of the alphabet, a punctuation mark, or a printer-control character such as a carriage return."(1) ASCII-American Standard Code for Information Interchange. You've heard it a million times, and will hear it a million more. It is the "closest thing the industry has to a standard set of character codes."(1) So, "Whether a given byte is interpreted as a number, an ASCII character, or something else depends entirely on the program using that byte."(1)

REGISTERS

A register is a special area in memory for storing the data upon which the program is operating.

Three Registers in the 6502 Processor:
A- Accumulator - Can add or subtract any number up to 255

X, and Y - These can either be used to add one or subtract one digit.

" The "A" register is often called the accumulator which indicates its function: all math and logical manipulations are done to the "A" register (from here on out it will be referred to as .A).
 There are two other registers inside the 6502 processor, specifically .X and .Y. These registers help act as counters and indexes into memory (sort of like mem[x] in pascal but not quite...)."(7)

The 6502 can set one register equal to any other register.

Instruction Cycle
----

*The 6502 only knows 151 instructions called opcodes. (I'm not sure if this has changed in the C=64, but I will find out. and update this) Each opcode is 1-byte (8-bits) long. Opcodes tell the processor what to do. The processor gets the first opcode, preforms the specified operation, gets the next opcode, preforms the operation, etc.

So where does the processor get the list of opcodes? You got it, from the program. The 6502 has a PC (Program Counter) that tells it where to get the next opcode from in memory. The PC stores the address of some location in memory. When the processor starts it's instruction cycle, it looks at the PC, gets the memory location for the first op- code, goes there, and preforms the operation specified by that opcode. When it's done with the first one, it MAKES the PC point to the next opcode. So the processor uses the PC as sort of a MAP. Then, it again looks at the PC and gets the memory location back and goes there and starts over again.

Here's a cool flowchart:

```
                [---------------------------]
                [  Fetch opcode pointed to  ]
                [  by the PC (Program       ]<-----\
                [  counter.                 ]      |
                [---------------------------]      |
                              |                    |
                             \|/                   |
                [---------------------------]      |
                [  Perform operation        ]      |
                [  specified by opcode      ]      |
                [---------------------------]      |
                              |                    |
                             \|/                   |
                [---------------------------]      |
                [  Make PC (Program Counter)]      |
                [  point to next opcode in  ]      |
                [        memory             ]      |
                [---------------------------]      |
                              |                    |
                              |____________________|
```

Cool, eh?
This is the 6502 Instruction Cycle.

MACHINE LANGUAGE

Machine Language program is nothing more then a series of ML instructions stored in memory. Each ML instruction is stored in memory as a 1-byte (8-bit) long opcode which may be followed by 1 or 2 bytes of operand. ML is usually in hexadecimal format. So, here is a short ML program: A9 05 20 02 04 A2 F5 60 Yup. Just a bunch of numbers! cool.

ASSEMBLERS

"To make it easier to write programs in machine language (called "ML" from here on), most programmers use a special program called an assembler. This is where the term "assembly language" comes from. ML and assembly language programs are both essentially the same thing. Using an assembler to create ML programs is far easier than being forced to look up and then POKE each byte into RAM memory. That's the way it used to be done, when there was too little memory in computers to hold languages (like BASIC or Assemblers) at the same time as programs created by those languages. That old style hand-programming was very laborious."(2)

"Program (which) takes source code in basic form or from a file and writes to memory or a file the resulting executable. Allows higher flexibility than a monitor (see below) due to use of labels etc and not having to keep track of each address within the program.

Monitor - A program, resident in memory, invoked by a SYS call from basic or by hitting the restore key that will let you disassemble, assemble and examine areas of memory and execute programs directly from the monitor. Useful for debugging programs and for writing short programs."(7)

One monitor that I've seen is the MLX monitor.

Object Code: is a series of 6502 machine language instructions to be stored in memory and executed.

Source Code: An assembly language source program consists of one or more lines of assembly language source code. These consist of 4 fields:

LABEL ---- MNEMONIC ---- OPERAND ---- COMMENT

Label is a name given to the instruction. Similar to BASIC line numbers.

Mnemonic is a cool word! It is the 3-letter name that suggests a function of a given ML instruction. (Easy! -- like LDA, LDX, or LDY... we'll get into these later.)

Operand would be the action of the Mnemonic. It's like this:

```
      LDA   $0300 <---operand... in this case we're loading
                       the accumulator with $0300..
```

LABEL- This is an optional field. This is where you put your comments. You separate the Label from the rest of the instruction with a ";" (semicolon).. This makes the source code more understandable.

Here's another cool flowchart:

```
      Source of input-->     PROGRAMMER
                              |
                             \|/
      What he/she inputs-->  SOURCE CODE
```

```
                             |
                            \|/
Program that converts    --->     ASSEMBLER
      Source code to ML      |
                            / \
                           /   \
                          /     \
                         /       \
            Output:       Assembler    Object
                      listing  code
                        |        |
      Intended for the:    Programmer   Processor
```

OK! Now if any of this is a bit confusing, look it over, and get used to it! You will be responsible for having this stuff in the back of your head at ALL TIMES!!! Good luck.. Next up is some Mnemonics!
See you all then!

APPENDIX A

COMMODORE 64 MEMORY MAP ROM/RAM

```
; Data types in headers (for reassembler):
;
;     DATA        Misc data
;     TEXT        String terminated with 00
;     WORD        Vectors in LO/HI byte pairs
;     CHIP        I/O Area
;     EMPTY       ROM containing FF's or AA's
;

HEX   DECIMAL           BITS  DESCRIPTION

0000  0           7-0   MOS 6510 Data Direction
                              Register (xx101111)
                              Bit= 1: Output, Bit=0:
                              Input, x=Don't Care

0001  1                 MOS 6510 Micro-Processor
                              On-Chip I/O Port
                  0     /LORAM Signal (0=Switch      BASIC ROM Out)
                  1     /HIRAM Signal (0=Switch Kernal ROM Out)
                  2     /CHAREN Signal (O=Swith Char. ROM In)
                  3     Cassette Data Output Line
                  4     Cassette Switch Sense: 1 = Switch Closed
                  5     Cassette Motor Control
                        O = ON, 1 = OFF
                  6-7   Undefined

D6510     0000        0       6510 On-chip Data Direction Register.
R6510     0001        1       6510 On-chip 8-bit Input/Output Register.
TEMP      0002        2       Unused. Free for user programs.
```

ADRAY1	0003-0004	3	Jump Vector: Convert FAC to Integer in (A/Y) ($B1AA).
ADRAY2	0005-0006	5	Jump Vector: Convert Integer in (A/Y) to Floating point in (FAC); ($B391).
CHARAC	0007	7	Search Character/Temporary Integer during INT.
ENDCHR	0008	8	Flag: Scan for Quote at end of String.
INTEGR	0007-0008	7	Temporary Integer during OR/AND.
TRMPOS	0009	9	Screen Column for last TAB.
VERCK	000A	10	Flag: 0 = Load, 1 = Verify.
COUNT	000B	11	Input Buffer Pointer/Number of Subscripts.
DIMFLG	000C	12	Flag: Default Array dimension.
VALTYP	000D	13	Data type Flag: $00 = Numeric, $FF = String.
INTFLG	000E	14	Data type Flag: $00 = Floating point, $80 = Integer.
GARBFL	000F	15	Flag: DATA scan/List Quote/Garbage collection.
SUBFLG	0010	16	Flag: Subscript reference/User Function call.
INPFLG	0011	17	Input Flag: $00 = INPUT, $40 = GET, $98 = READ.
TANSGN	0012	18	Flag: TAN sign/Comparative result.
CHANNL	0013	19	File number of current Input Device.
LINNUM	0014-0015	20	Temporary: Integer value.
TEMPPT	0016	22	Pointer: Temporary String Stack.
LASTPT	0017-0018	23	Last temporary String Address.
TEMPST	0019-0021	25	Stack for temporary Strings.
INDEX	0022-0025	34	Utility Pointer Area.
INDEX1	0022-0023	34	First Utility Pointer.
INDEX2	0024-0025	36	Secong Utility Pointer.
RESHO	0026-002A	38	Floating point product of Multiply and Divide.
TXTTAB	002B-002C	43	Pointer: Start of BASIC Text Area ($0801).
VARTAB	002D-002E	45	Pointer: Start of BASIC Variables.
ARYTAB	002F-0030	47	Pointer: Start of BASIC Arrays.
STREND	0031-0032	49	Pointer: End of BASIC Arrays + 1.
FRETOP	0033-0034	51	Pointer: Bottom of String space.
FRESPC	0035-0036	53	Utility String Pointer.
MEMSIZ	0037-0038	55	Pointer: Highest Address available to BASIC ($A000).
CURLIN	0039-003A	57	Current BASIC Line number.
OLDLIN	003B-003C	59	Previous BASIC Line number.
OLDTXT	003D-003E	61	Pointer: BASIC Statement for CONT.
DATLIN	003F-0040	63	Current DATA Line number.
DATPTR	0041-0042	65	Pointer: Used by READ - current DATA Item Address.
INPPTR	0043-0044	67	Pointer: Temporary storage of Pointer during INPUT Routine.
VARNAM	0045-0046	69	Name of Variable being sought in Variable Table.

```
VARPNT    0047-0048   71      Pointer: to value of (VARNAM) if Integer,
to
                              descriptor if String.
FORPNT    0049-004A   73      Pointer: Index Variable for FOR/NEXT loop.
VARTXT    004B-004C   75      Temporary storage for TXTPTR during READ,
                              INPUT and GET.
OPMASK    004D        77      Mask used during FRMEVL.
TEMPF3    004E-0052   78      Temporary storage for FLPT value.
FOUR6     0053        83      Length of String Variable during Garbege
                              collection.
JMPER     0054-0056   84      Jump Vector used in Function Evaluation -
                              JMP followed by Address ($4C,$LB,$MB).
TEMPF1    0057-005B   87      Temporary storage for FLPT value.
TEMPF2    005C-0060   92      Temporary storage for FLPT value.
FAC       0061-0066   97      Main Floating point Accumulator.
FACEXP    0061        97      FAC Exponent.
FACHO     0062-0065   98      FAC Mantissa.
FACSGN    0066        102     FAC Sign.
SGNFLG    0067        103     Pointer: Series Evaluation Constant.
BITS      0068        104     Bit Overflow Area during normalisation
                              Routine.
AFAC      0069-006E   105     Auxiliary Floating point Accumulator.
ARGEXP    0069        105     AFAC Exponent.
ARGHO     006A-006D   106     AFAC Mantissa.
ARGSGN    006E        110     AFAC Sign.
ARISGN    006F        111     Sign of result of Arithmetic Evaluation.
FACOV     0070        112     FAC low-order rounding.
FBUFPT    0071-0072   113     Pointer: Used during CRUNCH/ASCII
conversion.
CHRGET    0073-008A   115     Subroutine: Get next Byte of BASIC Text.
            ,0073  INC $7A        ,0082  BEQ $0073
            ,0075  BNE $0079      ,0084  SEC
            ,0077  INC $7B        ,0085  SBC #$30
          ! ,0079  LDA $0801      ,0087  SEC
            ,007C  CMP #$3A       ,0088  SBC #$D0
            ,007E  BCS $008A      ,008A  RTS
            ,0080  CMP #$20
CHRGOT    0079        121     Entry to Get same Byte again.
TXTPTR    007A-007B   122     Pointer: Current Byte of BASIC Text.
RNDX      008B-008F   139     Floating RND Function Seed Value.
STATUS    0090        144     Kernal I/O Status Word  ST.
STKEY     0091        145     Flag: $7F = STOP key.
SVXT      0092        146     Timing Constant for Tape.
VERCKK    0093        147     Flag: 0 = Load, 1 = Verify.
C3PO      0094        148     Flag: Serial Bus - Output Character
buffered.
BSOUR     0095        149     Buffered Character for Serial Bus.
SYNO      0096        150     Cassette Sync. number.
TEMPX     0097        151     Temporary storage of X Register during
CHRIN.
TEMPY     0097        151     Temporary storage of Y Register during
RS232
                              fetch.
LDTND     0098        152     Number of Open Files/Index to File Table.
```

DFLTN	0099	153	Default Input Device (0).
DFLTO	009A	154	Default Output Device (3).
PRTY	009B	155	Parity of Byte Output to Tape.
DPSW	009C	156	Flag: Byte received from Tape.
MSGFLG	009D	157	Flag: $00 = Program mode: Suppress Error Messages, $40 = Kernal Error Messages only, $80 = Direct mode: Full Error Messages.
FNMIDX	009E	158	Index to Cassette File name/Header ID for Tape write.
PTR1	009E	158	Tape Error log pass 1.
PTR2	009F	159	Tape Error log pass 2.
TIME	00A0-00A2	160	Real-time jiffy Clock (Updated by IRQ Interrupt approx. every 1/60 of Second); Update Routine: UDTIMK ($F69B).
TSFCNT	00A3	163	Bit Counter Tape Read or Write/Serial Bus EOI (End Of Input) Flag.
TBTCNT	00A4	164	Pulse Counter Tape Read or Write/Serial Bus shift Counter.
CNTDN	00A5	165	Tape Synchronising count down.
BUFPNT	00A6	166	Pointer: Tape I/O buffer.
INBIT	00A7	167	RS232 temporary for received Bit/Tape temporary.
BITC1	00A8	168	RS232 Input Bit count/Tape temporary.
RINONE	00A9	169	RS232 Flag: Start Bit check/Tape temporary.
RIDATA	00AA	170	RS232 Input Byte Buffer/Tape temporary.
RIPRTY	00AB	171	RS232 Input parity/Tape temporary.
SAL	00AC-00AD	172	Pointer: Tape Buffer/Screen scrolling.
EAL	00AE-00AF	174	Tape End Address/End of Program.
CMPO	00B0-00B1	176	Tape timing Constants.
TAPE1	00B2-00B3	178	Pointer: Start Address of Tape Buffer ($033C).
BITTS	00B4	180	RS232 Write bit count/Tape Read timing Flag.
NXTBIT	00B5	181	RS232 Next Bit to send/Tape Read - End of Tape.
RODATA	00B6	182	RS232 Output Byte Buffer/Tape Read Error Flag.
FNLEN	00B7	183	Number of Characters in Filename.
LA	00B8	184	Current File - Logical File number.
SA	00B9	185	Current File - Secondary Address.
FA	00BA	186	Current File - First Address (Device number). OPEN LA,FA,SA; OPEN 1,8,15,"I0":CLOSE 1
FNADR	00BB-00BC	187	Pointer: Current File name Address.
ROPRTY	00BD	189	RS232 Output Parity/Tape Byte to be Input or Output.
FSBLK	00BE	190	Tape Input/Output Block count.
MYCH	00BF	191	Serial Word Buffer.
CAS1	00C0	192	Tape Motor Switch.
STAL	00C1-00C2	193	Start Address for LOAD and Cassette Write.
MEMUSS	00C3-00C4	195	Pointer: Type 3 Tape LOAD and general use.
LSTX	00C5	197	Matrix value of last Key pressed; No Key = $40.

NDX	00C6	198	Number of Characters in Keyboard Buffer queue.
RVS	00C7	199	Flag: Reverse On/Off; On = $01, Off = $00.
INDX	00C8	200	Pointer: End of Line for Input (Used to suppress trailing spaces).
LXSP	00C9-00CA	201	Cursor X/Y (Line/Column) position at start of Input.
SFDX	00CB	203	Flag: Print shifted Characters.
BLNSW	00CC	204	Flag: Cursor blink; $00 = Enabled, $01 = Disabled.
BLNCT	00CD	205	Timer: Count down for Cursor blink toggle.
GDBLN	00CE	206	Character under Cursor while Cursor Inverted.
BLNON	00CF	207	Flag: Cursor Status; $00 = Off, $01 = On.
CRSW	00D0	208	Flag: Input from Screen = $03, or Keyboard = $00.
PNT	00D1-00D2	209	Pointer: Current Screen Line Address.
PNTR	00D3	211	Cursor Column on current Line, including Wrap-round Line, if any.
QTSW	00D4	212	Flag: Editor in Quote Mode; $00 = Not.
LNMX	00D5	213	Current logical Line length: 39 or 79.
TBLX	00D6	214	Current Screen Line number of Cursor.
SCHAR	00D7	215	Screen value of current Input Character/Last Character Output.
INSRT	00D8	216	Count of number of inserts outstanding.
LDTB1	00D9-00F2	217	Screen Line link Table/Editor temporaries. High Byte of Line Screen Memory Location.
USER	00F3-00F4	243	Pointer: Current Colour RAM Location.
KEYTAB	00F5-00F6	245	Vector: Current Keyboard decoding Table. ($EB81)
RIBUF	00F7-00F8	247	RS232 Input Buffer Pointer.
ROBUF	00F9-00FA	249	RS232 Output Buffer Pointer.
FREKZP	00FB-00FE	251	Free Zero Page space for User Programs.
BASZPT	00FF	255	BASIC temporary Data Area.
ASCWRK	00FF-010A	255	Assembly Area for Floating point to ASCII conversion.
BAD	0100-013E	256	Tape Input Error log.
STACK	0100-01FF	256	6510 Hardware Stack Area.
BSTACK	013F-01FF	319	BASIC Stack Area.
BUF	0200-0258	512	BASIC Input Buffer (Input Line from Screen).
LAT	0259-0262	601	Kernal Table: Active logical File numbers.
FAT	0263-026C	611	Kernal Table: Active File First Addresses (Device numbers).
SAT	026D-0276	621	Kernal Table: Active File Secondary Addresses.
KEYD	0277-0280	631	Keyboard Buffer Queue (FIFO).
MEMSTR	0281-0282	641	Pointer: Bottom of Memory for Operating System ($0800).
MEMSIZ	0283-0284	643	Pointer: Top of Memory for Operating System ($A000).

TIMOUT	0285	645	Serial IEEE Bus timeout defeat Flag.
COLOR	0286	646	Current Character Colour code.
GDCOL	0287	647	Background Colour under Cursor.
HIBASE	0288	648	High Byte of Screen Memory Address ($04).
XMAX	0289	649	Maximum number of Bytes in Keyboard Buffer ($0A).
RPTFLG	028A	650	Flag: Repeat keys; $00 = Cursors, INST/DEL & Space repeat, $40 no Keys repeat, $80 all Keys repeat ($00).
KOUNT	028B	651	Repeat Key: Speed Counter ($04).
DELAY	028C	652	Repeat Key: First repeat delay Counter ($10).
SHFLAG	028D	653	Flag: Shift Keys: Bit 1 = Shift, Bit 2 = CBM, Bit 3 = CTRL; ($00 = None, $01 = Shift, etc.).
LSTSHF	028E	654	Last Shift Key used for debouncing.
KEYLOG	028F-0290	655	Vector: Routine to determine Keyboard table to use based on Shift Key Pattern ($EB48).
MODE	0291	657	Flag: Upper/Lower Case change: $00 = Disabled, $80 = Enabled ($00).
AUTODN	0292	658	Flag: Auto scroll down: $00 = Disabled ($00).
M51CTR	0293	659	RS232 Pseudo 6551 control Register Image.
M51CDR	0294	660	RS232 Pseudo 6551 command Register Image.
M51AJB	0295-0296	661	RS232 Non-standard Bits/Second.
RSSTAT	0297	663	RS232 Pseudo 6551 Status Register Image.
BITNUM	0298	664	RS232 Number of Bits left to send.
BAUDOF	0299-029A	665	RS232 Baud Rate; Full Bit time microseconds.
RIDBE	029B	667	RS232 Index to End of Input Buffer.
RIDBS	029C	668	RS232 Pointer: High Byte of Address of Input Buffer.
RODBS	029D	669	RS232 Pointer: High Byte of Address of Output Buffer.
RODBE	029E	670	RS232 Index to End of Output Buffer.
IRQTMP	029F-02A0	671	Temporary store for IRQ Vector during Tape operations.
ENABL	02A1	673	RS232 Enables.
TODSNS	02A2	674	TOD sense during Tape I/O.
TRDTMP	02A3	675	Temporary storage during Tape READ.
TD1IRQ	02A4	676	Temporary D1IRQ Indicator during Tape READ.
TLNIDX	02A5	677	Temporary for Line Index.
TVSFLG	02A6	678	Flag: TV Standard: $00 = NTSC, $01 = PAL.
TEMP	02A7-02FF	679	Unused.
SPR11	02C0-02FE	704	Sprite #11 Data Area. (SCREEN + $03F8 + SPR number) POKE 1024+1016+0,11 to use Sprite#0 DATA from ($02C0-$02FE).
IERROR	0300-0301	768	Vector: Indirect entry to BASIC Error

			Message, (X) points to Message ($E38B).
IMAIN	0302-0303	770	Vector: Indirect entry to BASIC Input Line and Decode ($A483).
ICRNCH	0304-0305	772	Vector: Indirect entry to BASIC Tokenise Routine ($A57C).
IQPLOP	0306-0307	774	Vector: Indirect entry to BASIC LIST Routine ($A71A).
IGONE	0308-0309	776	Vector: Indirect entry to BASIC Character dispatch Routine ($A7E4).
IEVAL	030A-030B	778	Vector: Indirect entry to BASIC Token evaluation ($AE86).
SAREG	030C	780	Storage for 6510 Accumulator during SYS.
SXREG	030D	781	Storage for 6510 X-Register during SYS.
SYREG	030E	782	Storage for 6510 Y-Register during SYS.
SPREG	030F	783	Storage for 6510 Status Register during SYS.
USRPOK	0310	784	USR Function JMP Instruction ($4C).
USRADD	0311-0312	785	USR Address ($LB,$MB).
TEMP	0313	787	Unused.
CINV	0314-0315	788	Vector: Hardware IRQ Interrupt Address ($EA31).
CNBINV	0316-0317	790	Vector: BRK Instruction Interrupt Address ($FE66).
NMINV	0318-0319	792	Vector: Hardware NMI Interrupt Address ($FE47).
IOPEN	031A-031B	794	Vector: Indirect entry to Kernal OPEN Routine ($F34A).
ICLOSE	031C-031D	796	Vector: Indirect entry to Kernal CLOSE Routine ($F291).
ICHKIN	031E-031F	798	Vector: Indirect entry to Kernal CHKIN Routine ($F20E).
ICKOUT	0320-0321	800	Vector: Indirect entry to Kernal CHKOUT Routine ($F250).
ICLRCH	0322-0323	802	Vector: Indirect entry to Kernal CLRCHN Routine ($F333).
IBASIN	0324-0325	804	Vector: Indirect entry to Kernal CHRIN Routine ($F157).
IBSOUT	0326-0327	806	Vector: Indirect entry to Kernal CHROUT Routine ($F1CA).
ISTOP	0328-0329	808	Vector: Indirect entry to Kernal STOP Routine ($F6ED).
IGETIN	032A-032B	810	Vector: Indirect entry to Kernal GETIN Routine ($F13E).
ICLALL	032C-032D	812	Vector: Indirect entry to Kernal CLALL Routine ($F32F).
USRCMD	032E-032F	814	User Defined Vector ($FE66).
ILOAD	0330-0331	816	Vector: Indirect entry to Kernal LOAD Routine ($F4A5).
ISAVE	0332-0333	818	Vector: Indirect entry to Kernal SAVE Routine ($F5ED).
TEMP	0334-033B	820	Unused.
TBUFFR	033C-03FB	828	Tape I/O Buffer.
SPR13	0340-037E	832	Sprite #13.
SPR14	0380-03BE	896	Sprite #14.

```
SPR15     03C0-03FE   960     Sprite #15.
TEMP      03FC-03FF   1020    Unused.
VICSCN    0400-07E7   1024    Default Screen Video Matrix.
TEMP      07E8-07F7   2024    Unused.
SPNTRS    07F8-07FF   2040    Default Sprite Data Pointers.


          0800-9FFF   2048    Normal BASIC Program space.
          8000-9FFF   32768   Optional Cartridge ROM space.
          A000-BFFF   40960   BASIC ROM (Part) or 8 KB RAM.

a000  40960 -     Restart Vectors                    WORD
a00c  40972 stmdsp      BASIC Command Vectors              WORD
a052  41042 fundsp      BASIC Function Vectors             WORD
a080  41088 optab BASIC Operator Vectors             WORD
a09e  41118 reslst      BASIC Command Keyword Table        DATA
a129  41257 msclst      BASIC Misc. Keyword Table          DATA
a140  41280 oplist      BASIC Operator Keyword Table       DATA
a14d  41293 funlst      BASIC Function Keyword Table       DATA
a19e  41374 errtab      Error Message Table                DATA
a328  41768 errptr      Error Message Pointers             WORD
a364  41828 okk   Misc. Messages                     TEXT
a38a  41866 fndfor      Find FOR/GOSUB Entry on Stack
a3b8  41912 bltu  Open Space in Memory
a3fb  41979 getstk      Check Stack Depth
a408  41992 reason      Check Memory Overlap
a435  42037 omerr Output ?OUT OF MEMORY Error
a437  42039 error Error Routine
a469  42089 errfin      Break Entry
a474  42100 ready Restart BASIC
a480  42112 main  Input & Identify BASIC Line
a49c  42140 main1 Get Line Number & Tokenise Text
a4a2  42146 inslin      Insert BASIC Text
a533  42291 linkprg     Rechain Lines
a560  42336 inlin Input Line Into Buffer
a579  42361 crunch      Tokenise Input Buffer
a613  42515 fndlin      Search for Line Number
a642  42562 scrtch      Perform [new]
a65e  42590 clear Perform [clr]
a68e  42638 stxpt Reset TXTPTR
a69c  42652 list  Perform [list]
a717  42775 qplop Handle LIST Character
a742  42818 for   Perform [for]
a7ae  42926 newstt      BASIC Warm Start
a7c4  42948 ckeol Check End of Program
a7e1  42977 gone  Prepare to execute statement
a7ed  42989 gone3 Perform BASIC Keyword
a81d  43037 restor      Perform [restore]
a82c  43052 stop  Perform [stop], [end], break
a857  43095 cont  Perform [cont]
a871  43121 run   Perform [run]
a883  43139 gosub Perform [gosub]
a8a0  43168 goto  Perform [goto]
a8d2  43218 return      Perform [return]
```

```
a8f8  43256 data  Perform [data]
a906  43270 datan Search for Next Statement / Line
a928  43304 if    Perform [if]
a93b  43323 rem   Perform [rem]
a94b  43339 ongoto      Perform [on]
a96b  43371 linget      Fetch linnum From BASIC
a9a5  43429 let   Perform [let]
a9c4  43460 putint      Assign Integer
a9d6  43478 ptflpt      Assign Floating Point
a9d9  43481 putstr      Assign String
a9e3  43491 puttim      Assign TI$
aa2c  43564 getspt      Add Digit to FAC#1
aa80  43648 printn      Perform [print]#
aa86  43654 cmd   Perform [cmd]
aa9a  43674 strdon      Print String From Memory
aaa0  43680 print Perform [print]
aab8  43704 varop Output Variable
aad7  43735 crdo  Output CR/LF
aae8  43752 comprt      Handle comma, TAB(, SPC(
ab1e  43806 strout      Output String
ab3b  43835 outspc      Output Format Character
ab4d  43853 doagin      Handle Bad Data
ab7b  43899 get   Perform [get]
aba5  43941 inputn      Perform [input#]
abbf  43967 input Perform [input]
abea  44010 bufful      Read Input Buffer
abf9  44025 qinlin      Do Input Prompt
ac06  44038 read  Perform [read]
ac35  44085 rdget General Purpose Read Routine
acfc  44284 exint Input Error Messages               TEXT
ad1e  44318 next  Perform [next]
ad61  44385 donext      Check Valid Loop
ad8a  44426 frmnum      Confirm Result
ad9e  44446 frmevl      Evaluate Expression in Text
ae83  44675 eval  Evaluate Single Term
aea8  44712 pival Constant - pi                      DATA
aead  44717 qdot  Continue Expression
aef1  44785 parchk      Expression in Brackets
aef7  44791 chkcls      Confirm Character
aef7  44791 -     -test ')'-
aefa  44794 -     -test '('-
aefd  44797 -     -test comma-
af08  44808 synerr      Output ?SYNTAX Error
af0d  44813 domin Set up NOT Function
af14  44820 rsvvar      Identify Reserved Variable
af28  44840 isvar Search for Variable
af48  44872 tisasc      Convert TI to ASCII String
afa7  44967 isfun Identify Function Type
afb1  44977 strfun      Evaluate String Function
afd1  45009 numfun      Evaluate Numeric Function
afe6  45030 orop  Perform [or], [and]
b016  45078 dorel Perform <, =, >
b01b  45083 numrel      Numeric Comparison
b02e  45102 strrel      String Comparison
```

```
b07e  45182 dim   Perform [dim]
b08b  45195 ptrget      Identify Variable
b0e7  45287 ordvar      Locate Ordinary Variable
b11d  45341 notfns      Create New Variable
b128  45352 notevl      Create Variable
b194  45460 aryget      Allocate Array Pointer Space
b1a5  45477 n32768      Constant 32768 in Flpt            DATA
b1aa  45482 facinx      FAC#1 to Integer in (AC/YR)
b1b2  45490 intidx      Evaluate Text for Integer
b1bf  45503 ayint FAC#1 to Positive Integer
b1d1  45521 isary Get Array Parameters
b218  45592 fndary      Find Array
b245  45637 bserr ?BAD SUBSCRIPT/?ILLEGAL QUANTITY
b261  45665 notfdd      Create Array
b30e  45838 inlpn2      Locate Element in Array
b34c  45900 umult Number of Bytes in Subscript
b37d  45949 fre   Perform [fre]
b391  45969 givayf      Convert Integer in (AC/YR) to Flpt
b39e  45982 pos   Perform [pos]
b3a6  45990 errdir      Confirm Program Mode
b3e1  46049 getfnm      Check Syntax of FN
b3f4  46068 fndoer      Perform [fn]
b465  46181 strd  Perform [str$]
b487  46215 strlit      Set Up String
b4d5  46293 putnw1      Save String Descriptor
b4f4  46324 getspa      Allocate Space for String
b526  46374 garbag      Garbage Collection
b5bd  46525 dvars Search for Next String
b606  46598 grbpas      Collect a String
b63d  46653 cat   Concatenate Two Strings
b67a  46714 movins      Store String in High RAM
b6a3  46755 frestr      Perform String Housekeeping
b6db  46811 frefac      Clean Descriptor Stack
b6ec  46828 chrd  Perform [chr$]
b700  46848 leftd Perform [left$]
b72c  46892 rightd      Perform [right$]
b737  46903 midd  Perform [mid$]
b761  46945 pream Pull sTring Parameters
b77c  46972 len   Perform [len]
b782  46978 len1  Exit String Mode
b78b  46987 asc   Perform [asc]
b79b  47003 gtbytc      Evaluate Text to 1 Byte in XR
b7ad  47021 val   Perform [val]
b7b5  47029 strval      Convert ASCII String to Flpt
b7eb  47083 getnum      Get parameters for POKE/WAIT
b7f7  47095 getadr      Convert FAC#1 to Integer in LINNUM
b80d  47117 peek  Perform [peek]
b824  47140 poke  Perform [poke]
b82d  47149 wait  Perform [wait]
b849  47177 faddh Add 0.5 to FAC#1
b850  47184 fsub  Perform Subtraction
b862  47202 fadd5 Normalise Addition
b867  47207 fadd  Perform Addition
b947  47431 negfac      2's Complement FAC#1
```

```
b97e  47486 overr Output ?OVERFLOW Error
b983  47491 mulshf      Multiply by Zero Byte
b9bc  47548 fone  Table of Flpt Constants                 DATA
b9ea  47594 log   Perform [log]
ba28  47656 fmult Perform Multiply
ba59  47705 mulply      Multiply by a Byte
ba8c  47756 conupk      Load FAC#2 From Memory
bab7  47799 muldiv      Test Both Accumulators
bad4  47828 mldvex      Overflow / Underflow
bae2  47842 mul10 Multiply FAC#1 by 10
baf9  47865 tenc  Constant 10 in Flpt               DATA
bafe  47870 div10 Divide FAC#1 by 10
bb07  47879 fdiv  Divide FAC#2 by Flpt at (AC/YR)
bb0f  47887 fdivt Divide FAC#2 by FAC#1
bba2  48034 movfm Load FAC#1 From Memory
bbc7  48071 mov2f Store FAC#1 in Memory
bbfc  48124 movfa Copy FAC#2 into FAC#1
bc0c  48140 movaf Copy FAC#1 into FAC#2
bc1b  48155 round Round FAC#1
bc2b  48171 sign  Check Sign of FAC#1
bc39  48185 sgn   Perform [sgn]
bc58  48216 abs   Perform [abs]
bc5b  48219 fcomp Compare FAC#1 With Memory
bc9b  48283 qint  Convert FAC#1 to Integer
bccc  48332 int   Perform [int]
bcf3  48371 fin   Convert ASCII String to a Number in FAC#1
bdb3  48563 n0999 String Conversion Constants       DATA
bdc2  48578 inprt Output 'IN' and Line Number
bddd  48605 fout  Convert FAC#1 to ASCII String
be68  48744 foutim      Convert TI to String
bf11  48913 fhalf Table of Constants                DATA
bf71  49009 sqr   Perform [sqr]
bf7b  49019 fpwrt Perform power ($)
bfb4  49076 negop Negate FAC#1
bfbf  49087 logeb2      Table of Constants                DATA
bfed  49133 exp   Perform [exp]


          C000-CFFF   49152   4 KB RAM.
          D000-DFFF   53248   Input/Output Devices and Colour RAM or
                              4 KB RAM or Character ROM.
          D000-D02E   53248   6566 Video Interface Chip, VIC II.

D000-D02E   53248-54271 MOS 6566 VIDEO INTERFACE CONTROLLER (VIC)

D000        53248       Sprite O X Pos
D001        53249       Sprite O Y Pos
D002        53250       Sprite 1 X Pos
D003        53251       Sprite 1 Y Pos
D004        53252       Sprite 2 X Pos
D005        53253       Sprite 2 Y Pos
D006        53254       Sprite 3 X Pos
D007        53255       Sprite 3 Y Pos
D008        53256       Sprite 4 X Pos
```

```
D009        53257       Sprite 4 Y Pos
D00A        53258       Sprite 5 X Pos
D00B        53259       Sprite 5 Y Pos
D00C        53260       Sprite 6 X Pos
D00D        53261       Sprite 6 Y Pos
D00E        53262       Sprite 7 X Pos
D00F        53263       Sprite 7 Y Pos
D010        53264       Sprites 0-7 X Pos (msb of X coord.)

D011        53265       VIC Control Register
                  7     Raster Compare: (Bit 8)      See 53266
                  6     Extended Color Text Mode 1 = Enable
                  5     Bit Map Mode. 1 = Enable
                  4     Blank Screen to Border Color: O = Blank
                  3     Select 24/25 Row Text Display: 1 = 25 Rows
                  2-0   Smooth Scroll to Y Dot-Position (0-7)

D012  53266             Read Raster / Write Raster Value for Compare IRQ
D013  53267             Light-Pen Latch X Pos
D014  53268             Light-Pen Latch Y Pos
D015  53269             Sprite display Enable: 1 = Enable

D016  53270             VIC Control Register
                  7-6   Unused
                  5     ALWAYS SET THIS BIT TO 0 !
                  4     Multi-Color Mode: 1 = Enable (Text or Bit-Map)
                  3     Select 38/40 Column Text Display: 1 = 40 Cols
                  2-0   Smooth Scroll to X Pos

D017  53271             Sprites O-7 Expand 2x Vertical (Y)

D018  53272             VIC Memory Control Register
                  7-4   Video Matrix Base Address (inside VIC)
                  3-1   Character Dot-Data Base      Address (inside VIC)
                  0     Select upper/lower Character Set

D019  53273             VIC Interrupt Flag Register (Bit = 1: IRQ
Occurred)
                  7     Set on Any Enabled VIC IRQ Condition
                  3     Light-Pen Triggered IRQ Flag
                  2     Sprite to Sprite Collision IRQ Flag
                  1     Sprite to Background Collision IRQ Flag
                  0     Raster Compare IRQ Flag

D01A  53274             IRQ Mask Register: 1 = Interrupt Enabled
D01B  53275             Sprite to Background Display Priority: 1 = Sprite
D01C  53276             Sprites O-7 Multi-Color Mode Select: 1 = M.C.M.
D01D  53277             Sprites 0-7 Expand 2x Horizontal (X)

D01E  53278             Sprite to Sprite Collision Detect
D01F  53279             Sprite to Background Collision Detect
D020  53280             Border Color
D021  53281             Background Color O
D022  53282             Background Color 1
```

```
D023  53283            Background Color 2
D024  53284            Background Color 3
D025  53285            Sprite Multi-Color Register 0
D026  53286            Sprite Multi-Color Register 1

D027  53287            Sprite O Color
D028  53288            Sprite 1 Color
D029  53289            Sprite 2 Color
D02A  53290            Sprite 3 Color
D02B  53291            Sprite 4 Color
D02C  53292            Sprite 5 Color
D02D  53293            Sprite 6 Color
D02E  53294            Sprite 7 Color


          D400-D41C   54272   6581 Sound Interface Device, SID.

D400-D7FF   54272-55295 MOS 6581 SOUND INTERFACE DEVICE (SID)

D400  54272            Voice 1: Frequency Control - Low-Byte
D401  54273            Voice 1: Frequency Control - High-Byte
D402  54274            Voice 1: Pulse Waveform     Width - Low-Byte
D403  54275      7-4   Unused
                 3-0   Voice 1: Pulse Waveform Width - High-Nybble
D404  54276            Voice 1: Control Register
                 7     Select Random Noise Waveform, 1 = On
                 6     Select Pulse Waveform, 1 = On
                 5     Select Sawtooth Waveform, 1 = On
                 4     Select Triangle Waveform, 1 = On
                 3     Test Bit: 1 = Disable Oscillator 1
                 2     Ring Modulate Osc. 1 with Osc. 3 Output, 1 = On
                 1     Synchronize Osc. 1 with Osc. 3 Frequency, 1 = On
                 0     Gate Bit: 1 = Start Att/Dec/Sus, 0 = Start Release

D405  54277            Envelope Generator 1: Attack / Decay Cycle Control
                 7-4   Select Attack Cycle Duration: O-15
                 3-0   Select Decay Cycle Duration: 0-15

D406  54278            Envelope Generator 1: Sustain / Release Cycle
Control
                 7-4   Select Sustain Cycle Duration: O-15
                 3-0   Select Release Cycle Duration: O-15

D407  54279            Voice 2: Frequency Control - Low-Byte
D408  54280            Voice 2: Frequency Control - High-Byte
D409  54281            Voice 2: Pulse Waveform Width - Low-Byte

D40A  54282      7-4   Unused
                 3-0   Voice 2: Pulse Waveform Width - High-Nybble

D40B  54283            Voice 2: Control Register
                 7     Select Random Noise Waveform, 1 = On
                 6     Select Pulse Waveform, 1 = On
                 5     Select Sawtooth Waveform, 1 = On
```

```
                4     Select Triangle Waveform, 1 = On
                3     Test Bit: 1 = Disable Oscillator 1
                2     Ring Modulate Osc. 2 with Osc. 1 Output, 1 = On
                1     Synchronize Osc. 2 with Osc. 1 Frequency, 1 = On
                0     Gate Bit: 1 = Start Att/Dec/Sus, 0 = Start Release

D40C  54284           Envelope Generator 2: Attack / Decay Cycle Control
                7-4   Select Attack Cycle Duration: O-15
                3-0   Select Decay Cycle Duration: 0-15

D40D  54285           Envelope Generator 2: Sustain / Release Cycle
Control
                7-4   Select Sustain Cycle Duration: O-15
                3-0   Select Release Cycle Duration: O-15

D40E  54286           Voice 3: Frequency Control - Low-Byte
D40F  54287           Voice 3: Frequency Control - High-Byte
D410  54288           Voice 3: Pulse Waveform Width - Low-Byte
D411  54289     7-4   Unused
                3-0   Voice 3: Pulse Waveform Width - High-Nybble
D412  54290           Voice 3: Control Register
                7     Select Random Noise Waveform, 1 = On
                6     Select Pulse Waveform, 1 = On
                5     Select Sawtooth Waveform, 1 = On
                4     Select Triangle Waveform, 1 = On
                3     Test Bit: 1 = Disable Oscillator 1
                2     Ring Modulate Osc. 3 with Osc. 2 Output, 1 = On
                1     Synchronize Osc. 3 with Osc. 2 Frequency, 1 = On
                0     Gate Bit: 1 = Start Att/Dec/Sus, 0 = Start Release

D413  54291     Envelope Generator 3: Attac/Decay Cycle Control
                7-4   Select Attack Cycle Duration: O-15
                3-0   Select Decay Cycle Duration: 0-15

D414  54285           Envelope Generator 3: Sustain / Release Cycle
Control
                7-4   Select Sustain Cycle Duration: O-15
                3-0   Select Release Cycle Duration: O-15


D415  54293           Filter Cutoff Frequency: Low-Nybble (Bits 2-O)
D416  54294           Filter Cutoff Frequency: High-Byte
D417  54295           Filter Resonance Control / Voice Input Control
                7-4   Select Filter Resonance: 0-15
                3     Filter External Input: 1 = Yes, 0 = No
                2     Filter Voice 3 Output: 1 = Yes, 0 = No
                      Filter Voice 2 Output: 1 = Yes, 0 = No
                0     Filter Voice 1 Output: 1 = Yes, 0 = No

D418  54296           Select Filter Mode and Volume
                7     Cut-Off Voice 3 Output: 1 = Off, O = On

                6     Select Filter High-Pass Mode: 1 = On
                5     Select Filter Band-Pass Mode: 1 = On
```

```
                 4     Select Filter Low-Pass Mode: 1 = On
                 3-0   Select Output Volume: 0-15

D419  54297            Analog/Digital Converter: Game Paddle 1 (O-255)
D41A  54298            Analog/Digital Converter Game Paddle 2 (O-255)
D41B  54299            Oscillator 3 Random Number Generator
D41C  54230            Envelope Generator 3 Output


          D500-D7FF   54528   SID Images.
          D800-DBE7   55296   Colour RAM (Nybbles = 4 Bit RAM, LSB).
          DBE8-DBFF   56296   Unused Nybbles.
          DC00-DC0F   56320   6526 Complex Interface Adaptor, CIA.

DC00  56320            Data Port A (Keyboard, Joystick, Paddles, Light-
Pen)

                 7-0   Write Keyboard Column Values for Keyboard Scan
                 7-6   Read Paddles on Port A / B (01 = Port A, 10 = Port
B)
                 4     Joystick A Fire Button: 1 = Fire
                 3-2   Paddle Fire Buttons
                 3-0   Joystick A Direction (0-15)

DC01  56321            Data Port B (Keyboard, Joystick, Paddles): Game
Port 1
                 7-0   Read Keyboard Row Values for Keyboard Scan

                 7     Timer B Toggle/Pulse Output
                 6     Timer A: Toggle/Pulse Output

                 4     Joystick 1 Fire Button: 1 = Fire
                 3-2   Paddle Fire Buttons
                 3-0   Joystick 1 Direction

DC02  56322            Data Direction Register - Port A (56320)
DC03  56323            Data Direction Register - Port B (56321)
DC04  56324            Timer A: Low-Byte
DC05  56325            Timer A: High-Byte
DC06  56326            Timer B: Low-Byte
DC07  56327            Timer B: High-Byte

DC08  56328            Time-of-Day Clock: 1/10 Seconds
DC09  56329            Time-of-Day Clock: Seconds
DC0A  56330            Time-of-Day Clock: Minutes
DC0B  56331            Time-of-Day Clock: Hours + AM/PM Flag (Bit 7)

DC0C  56332            Synchronous Serial I/O Data Buffer
DC0D  56333            CIA Interrupt Control Register (Read IRQs/Write
Mask)

                 7     IRQ Flag (1 = IRQ Occurred) / Set-Clear Flag
                 4     FLAG1 IRQ (Cassette Read / Serial Bus SRQ Input)
                 3     Serial Port Interrupt
```

```
                   2     Time-of-Day Clock Alarm Interrupt
                   1     Timer B Interrupt
                   0     Timer A Interrupt

DC0E  56334              CIA Control Register A
                   7     Time-of-Day Clock Frequency: 1 = 50 Hz, 0 = 60 Hz
                   6     Serial Port I/O Mode Output, 0 = Input
                   5     Timer A Counts: 1 = CNT Signals, 0 = System 02
Clock

                   4     Force Load Timer A: 1 = Yes
                   3     Timer A Run Mode: 1 = One-Shot, 0 = Continuous
                   2     Timer A Output Mode to PB6: 1 = Toggle, 0 = Pulse
                   1     Timer A Output on PB6: 1 = Yes, 0 = No
                   0     Start/Stop Timer A: 1 = Start, 0 = Stop

DC0F  56335              CIA Control Register B
                   7     Set Alarm/TOD-Clock: 1 = Alarm, 0 = Clock
                   6-5   Timer B Mode Select:
                               00 = Count System 02 Clock Pulses
                               01 = Count Positive CNT Transitions
                               10 = Count Timer A Underflow Pulses
                               11 = Count Timer A Underflows While CNT
Positive
                   4-0   Same as CIA Control Reg. A - for Timer B



DC00-DCFF   56320-56575 MOS 6526 Complex Interface Adapter (CIA) #1

DD00-DDFF   56576-56831 MOS 6526 Complex Interface Adapter (CIA) #2

DD00  56576              Data Port A (Serial Bus, RS-232, VIC Memory
Control)
                   7     Serial Bus Data Input
                   6     Serial Bus Clock Pulse Input
                   5     Serial Bus Data Output
                   4     Serial Bus Clock Pulse Output
                   3     Serial Bus ATN Signal Output
                   2     RS-232 Data Output (User Port)
                   1-O   VIC Chip System Memory Bank Select (Default = 11)

DD01  56577        Data Port B (User Port, RS-232)
                   7     User / RS-232 Data Set Ready
                   6     User / RS-232 Clear to Send
                   5     User
                   4     User / RS-232 Carrier Detect
                   3     User / RS-232 Ring Indicator
                   2     User / RS-232 Data Terminal Ready
                   1     User / RS-232 Request to Send
                   0     User / RS-232 Received Data

DD02  56578              Data Direction Register - Port A
```

```
DD03  56579           Data Direction Register - Port B
DD04  56580           Timer A: Low-Byte
DD05  56581           Timer A: High-Byte
DD06  56582           Timer B: Low-Byte
DD07  56583           Timer B: High-Byte

DD08  56584           Time-of-Day Clock: 1/10 Seconds
DD09  56585           Time-of-Day Clock: Seconds
DD0A  56586           Time-of-Day Clock: Minutes
DD0B  56587           Time-of-Day Clock: Hours + AM/PM Flag (Bit 7)
DD0C  56588           Synchronous Serial I/O Data Buffer
DD0D  56589           CIA Interrupt Control Register (Read NMls/Write
Mask)
                 7     NMI Flag (1 = NMI Occurred) / Set-Clear Flag
                 4     FLAG1 NMI (User/RS-232 Received Data Input)
                 3     Serial Port Interrupt

                 1     Timer B Interrupt
                 0     Timer A Interrupt

DD0E  56590           CIA Control Register A

                 7     Time-of-Day Clock Frequency: 1 = 50 Hz, 0 = 60 Hz
                 6     Serial Port I/O Mode Output, 0 = Input
                 5     Timer A Counts: 1 = CNT Signals, 0 = System 02
Clock
                 4     Force Load Timer A: 1 = Yes
                 3     Timer A Run Mode: 1 = One-Shot, 0 = Continuous
                 2     Timer A Output Mode to PB6: 1 = Toggle, 0 = Pulse
                 1     Timer A Output on PB6: 1 = Yes, 0 = No
                 0     Start/Stop Timer A: 1 = Start, 0 = Stop

DD0F  56591           CIA Control Register B
                 7     Set Alarm/TOD-Clock: 1 = Alarm, 0 = Clock
                 6-5   Timer B Mode Select:
                             00 = Count System 02 Clock Pulses
                             01 = Count Positive CNT Transitions
                             10 = Count Timer A Underflow Pulses
                             11 = Count Timer A Underflows While CNT
Positive
                 4-0   Same as CIA Control Reg. A - for Timer B


DEOO-DEFF   56832-57087 Reserved for Future I/O Expansion
DFOO-DFFF   57088-57343 Reserved for Future I/O Expansion


          E000-FFFF   57344   BASIC (Part)/Kernal ROM or 8 KB RAM.
          E000-E4FF   57344   BASIC ROM (Part) or RAM.

e000  57344 (exp continues)  EXP continued From BASIC ROM
e043  57411 polyx Series Evaluation
e08d  57485 rmulc Constants for RND           DATA
e097  57495 rnd   Perform [rnd]
```

```
e0f9  57593 bioerr      Handle I/O Error in BASIC
e10c  57612 bchout      Output Character
e112  57618 bchin Input Character
e118  57624 bckout      Set Up For Output
e11e  57630 bckin Set Up For Input
e124  57636 bgetin      Get One Character
e12a  57642 sys   Perform [sys]
e156  57686 savet Perform [save]
e165  57701 verfyt      Perform [verify / load]
e1be  57790 opent Perform [open]
e1c7  57799 closet      Perform [close]
e1d4  57812 slpara      Get Parameters For LOAD/SAVE
e200  57856 combyt      Get Next One Byte Parameter
e206  57862 deflt Check Default Parameters
e20e  57870 cmmerr      Check For Comma
e219  57881 ocpara      Get Parameters For OPEN/CLOSE
e264  57956 cos   Perform [cos]
e26b  57963 sin   Perform [sin]
e2b4  58036 tan   Perform [tan]
e2e0  58080 pi2   Table of Trig Constants              DATA
;e2e0 1.570796327 pi/2
;e2e5 6.28318531  pi*2
;e2ea 0.25

;e2ef #05   (counter)
;e2f0 -14.3813907
;e2f5 42.0077971
;e2fa -76.7041703
;e2ff 81.6052237
;e304 -41.3417021
;e309 6.28318531

e30e  58126 atn   Perform [atn]
e33e  58174 atncon      Table of ATN Constants         DATA

;e33e #0b   (counter)
;e3ef -0.000684793912
;e344  0.00485094216
;e349 -0.161117018
;e34e  0.034209638
;e353 -0.0542791328
;e358  0.0724571965
;e35d -0.0898023954
;e362  0.110932413
;e367 -0.142839808
;e36c  0.19999912
;e371 -0.333333316
;e376  1.00

e37b  58235 bassft      BASIC Warm Start [RUNSTOP-RESTORE]
e394  58260 init  BASIC Cold Start
e3a2  58274 initat      CHRGET For Zero-page
e3ba  58298 rndsed      RND Seed For zero-page         DATA
;e3b2 0.811635157
```

```
e3bf  58303 initcz      Initialize BASIC RAM
e422  58402 initms      Output Power-Up Message
e447  58439 bvtrs Table of BASIC Vectors (for 0300)  WORD
e453  58451 initv Initialize Vectors
e45f  58463 words Power-Up Message             DATA
e4ad  58541 -     Patch for BASIC Call to CHKOUT
e4b7  58551 -     Unused Bytes For Future Patches          EMPTY
e4da  58586 -     Reset Character Colour
e4e0  58592 -     Pause After Finding Tape File
e4ec  58604 -     RS-232 Timing Table -- PAL         DATA

           E500-FFFF   58624   Kernal ROM or RAM.
e500  58624 iobase      Get I/O Address
e505  58629 screen      Get Screen Size
e50a  58634 plot  Put / Get Row And Column
e518  58648 cint1 Initialize I/O
e544  58692 -     Clear Screen
e566  58726 -     Home Cursor
e56c  58732 -     Set Screen Pointers
e59a  58778 -     Set I/O Defaults (Unused Entry)
e5a0  58784 -     Set I/O Defaults
e5b4  58804 lp2   Get Character From Keyboard Buffer
e5ca  58826 -     Input From Keyboard
e632  58930 -     Input From Screen or Keyboard
e684  59012 -     Quotes Test
e691  59025 -     Set Up Screen Print
e6b6  59062 -     Advance Cursor
e6ed  59117 -     Retreat Cursor
e701  59137 -     Back on to Previous Line
e716  59158 -     Output to Screen
e72a  59178 -     -unshifted characters-
e7d4  59348 -     -shifted characters-
e87c  59516 -     Go to Next Line
e891  59537 -     Output <CR>
e8a1  59553 -     Check Line Decrement
e8b3  59571 -     Check Line Increment
e8cb  59595 -     Set Colour Code
e8da  59610 -     Colour Code Table
e8ea  59626 -     Scroll Screen
e965  59749 -     Open A Space On The Screen
e9c8  59848 -     Move A Screen Line
e9e0  59872 -     Syncronise Colour Transfer
e9f0  59888 -     Set Start of Line
e9ff  59903 -     Clear Screen Line
ea13  59923 -     Print To Screen
ea24  59940 -     Syncronise Colour Pointer
ea31  59953 -     Main IRQ Entry Point
ea87  60039 scnkey      Scan Keyboard
eadd  60125 -     Process Key Image
eb79  60281 -     Pointers to Keyboard decoding tables    WORD
eb81  60289 -     Keyboard 1 -- unshifted                 DATA
ebc2  60354 -     Keyboard 2 -- Shifted             DATA
ec03  60419 -     Keyboard 3 -- Commodore                 DATA
ec44  60484 -     Graphics/Text Control
```

```
ec78  60536 -     Keyboard 4 -- Control              DATA
ecb9  60601 -     Video Chip Setup Table             DATA
ece7  60647 -     Shift-Run Equivalent
ecf0  60656 -     Low Byte Screen Line Addresses          DATA
ed09  60681 talk  Send TALK Command on Serial Bus
ed0c  60684 listn Send LISTEN Command on Serial Bus
ed40  60736 -     Send Data On Serial Bus
edad  60845 -     Flag Errors
edad  60845 -     Status #80 - device not present
edb0  60848 -     Status #03 - write timeout
edb9  60857 second      Send LISTEN Secondary Address
edbe  60862 -     Clear ATN
edc7  60871 tksa  Send TALK Secondary Address
edcc  60876 -     Wait For Clock
eddd  60893 ciout Send Serial Deferred
edef  60911 untlk Send UNTALK / UNLISTEN
ee13  60947 acptr Receive From Serial Bus
ee85  61061 -     Serial Clock On
ee8e  61070 -     Serial Clock Off
ee97  61079 -     Serial Output 1
eea0  61088 -     Serial Output 0
eea9  61097 -     Get Serial Data And Clock In
eeb3  61107 -     Delay 1 ms
eebb  61115 -     RS-232 Send
ef06  61190 -     Send New RS-232 Byte
ef2e  61230 -     'No DSR' / 'No CTS' Error
ef39  61241 -     Disable Timer
ef4a  61258 -     Compute Bit Count
ef59  61273 -     RS-232 Receive
ef7e  61310 -     Set Up To Receive
ef90  61328 -     Process RS-232 Byte
efe1  61409 -     Submit to RS-232
f00d  61453 -     No DSR (Data Set Ready) Error
f017  61463 -     Send to RS-232 Buffer
f04d  61517 -     Input From RS-232
f086  61574 -     Get From RS-232
f0a4  61604 -     Serial Bus Idle
f0bd  61629 -     Table of Kernal I/O Messages       DATA
f12b  61739 -     Print Message if Direct
f12f  61743 -     Print Message
f13e  61758 getin Get a byte
f157  61783 chrin Input a byte
f199  61849 -     Get From Tape / Serial / RS-232
f1ca  61898 chrout      Output One Character
f20e  61966 chkin Set Input Device
f250  62032 chkout      Set Output Device
f291  62097 close Close File
f30f  62223 -     Find File
f31f  62239 -     Set File values
f32f  62255 clall Abort All Files
f333  62259 clrchn      Restore Default I/O
f34a  62282 open  Open File
f3d5  62421 -     Send Secondary Address
f409  62473 -     Open RS-232
```

```
f49e  62622 load  Load RAM
f4b8  62648 -     Load File From Serial Bus
f533  62771 -     Load File From Tape
f5af  62927 -     Print "SEARCHING"
f5c1  62913 -     Print Filename
f5d2  62930 -     Print "LOADING / VERIFYING"
f5dd  62941 save  Save RAM
f5fa  62970 -     Save to Serial Bus
f659  63065 -     Save to Tape
f68f  63119 -     Print "SAVING"
f69b  63131 udtim Bump Clock
f6dd  63197 rdtim Get Time
f6e4  63204 settim      Set Time
f6ed  63213 stop  Check STOP Key
f6fb  63227 -     Output I/O Error Messages
f6fb  63227 -     'too many files'
f6fe  63230 -     'file open'
f701  63233 -     'file not open'
f704  63236 -     'file not found'
f707  63239 -     'device not present'
f70a  63242 -     'not input file'
f70d  63245 -     'not output file'
f710  63248 -     'missing filename'
f713  63251 -     'illegal device number'
f72d  63277 -     Find Any Tape Header
f76a  63338 -     Write Tape Header
f7d0  63440 -     Get Buffer Address
f7d7  63447 -     Set Buffer Stat / End Pointers
f7ea  63466 -     Find Specific Tape Header
f80d  63501 -     Bump Tape Pointer
f817  63511 -     Print "PRESS PLAY ON TAPE"
f82e  63534 -     Check Tape Status
f838  63544 -     Print "PRESS RECORD..."
f841  63553 -     Initiate Tape Read
f864  63588 -     Initiate Tape Write
f875  63605 -     Common Tape Code
f8d0  63696 -     Check Tape Stop
f8e2  63714 -     Set Read Timing
f92c  63788 -     Read Tape Bits
fa60  64096 -     Store Tape Characters
fb8e  64398 -     Reset Tape Pointer
fb97  64407 -     New Character Setup
fba6  64422 -     Send Tone to Tape
fbc8  64456 -     Write Data to Tape
fbcd  64461 -     IRQ Entry Point
fc57  64599 -     Write Tape Leader
fc93  64659 -     Restore Normal IRQ
fcb8  64696 -     Set IRQ Vector
fcca  64714 -     Kill Tape Motor
fcd1  64721 -     Check Read / Write Pointer
fcdb  64731 -     Bump Read / Write Pointer
fce2  64738 -     Power-Up RESET Entry
fd02  64770 -     Check For 8-ROM
fd12  64786 -     8-ROM Mask '80CBM'                DATA
```

```
fd15  64789 restor      Restore Kernal Vectors (at 0314)
fd1a  64794 vector      Change Vectors For User
fd30  64816 -     Kernal Reset Vectors               WORD
fd50  64848 ramtas      Initialise System Constants
fd9b  64923 -     IRQ Vectors For Tape I/O           WORD
fda3  64931 ioinit      Initialise I/O
fddd  64989 -     Enable Timer
fdf9  65017 setnam      Set Filename
fe00  65024 setlfs      Set Logical File Parameters
fe07  65031 readst      Get I/O Status Word
fe18  65048 setmsg      Control OS Messages
fe21  65057 settmo      Set IEEE Timeout
fe25  65061 memtop      Read / Set Top of Memory
fe34  65076 membot      Read / Set Bottom of Memory
fe43  65091 -     NMI Transfer Entry
fe66  65126 -     Warm Start Basic [BRK]
febc  65212 -     Exit Interrupt
fec2  65218 -     RS-232 Timing Table - NTSC   DATA
fed6  65238 -     NMI RS-232 In
ff07  65287 -     NMI RS-232 Out
ff43  65347 -     Fake IRQ Entry
ff48  65352 -     IRQ Entry
ff5b  65371 cint  Initialize screen editor
ff80  65408 -     Kernal Version Number [03]   DATA
```

APPENDIX B

```
; ---<FROM FILE C64rom.lib>---
;
;     Commodore 64 ROM Memory Map
;
; BASIC interpreter ROM ($A000 - $BFFF)
;
; label       address   type  comments
restart     = $a000
stmdsp      = $a00c
fundsp      = $a052
optab = $a080
reslst      = $a09e
msclst      = $a129
oplist      = $a140
funlst      = $a14d
errtab      = $a19e
errptr      = $a328
okk   = $a364
fndfor      = $a38a
bltu  = $a3b8
getstk      = $a3fb
reason      = $a408
omerr = $a435
error = $a437
```

```
errfin = $a469
ready = $a474
main = $a480
main1 = $a49c
inslin = $a4a2
linkprg = $a533
inlin = $a560
crunch = $a579
fndlin = $a613
scrtch = $a642
clear = $a65e
stxpt = $a68e
list = $a69c
qplop = $a717
for = $a742
newstt = $a7ae
ckeol = $a7c4
gone = $a7e1
gone3 = $a7ed
restor = $a81d
stop = $a82c
cont = $a857
run = $a871
gosub = $a883
goto = $a8a0
return = $a8d2
data = $a8f8
datan = $a906
if = $a928
rem = $a93b
ongoto = $a94b
linget = $a96b
let = $a9a5
putint = $a9c4
ptflpt = $a9d6
putstr = $a9d9
puttim = $a9e3
getspt = $aa2c
printn = $aa80
cmd = $aa86
strdon = $aa9a
print = $aaa0
varop = $aab8
crdo = $aad7
comprt = $aae8
strout = $ab1e
outspc = $ab3b
doagin = $ab4d
get = $ab7b
inputn = $aba5
input = $abbf
bufful = $abea
qinlin = $abf9
read = $ac06
```

```
rdget = $ac35
exint = $acfc
next  = $ad1e
donext      = $ad61
frmnum      = $ad8a
frmevl      = $ad9e
eval  = $ae83
pival = $aea8
qdot  = $aead
parchk      = $aef1
chkcls      = $aef7
synerr      = $af08
domin = $af0d
rsvvar      = $af14
isvar = $af28
tisasc      = $af48
isfun = $afa7
strfun      = $afb1
numfun      = $afd1
orop  = $afe6
dorel = $b016
numrel      = $b01b
strrel      = $b02e
dim   = $b07e
ptrget      = $b08b
ordvar      = $b0e7
isletc      = $b113
notfns      = $b11d
notevl      = $b128
aryget      = $b194
n32768      = $b1a5           data
facinx      = $b1aa
intidx      = $b1b2
ayint = $b1bf
isary = $b1d1
fndary      = $b218
bserr = $b245
notfdd      = $b261
inlpn2      = $b30e
umult = $b34c
fre   = $b37d
givayf      = $b391
pos   = $b39e
errdir      = $b3a6
def   = $b3b3
getfnm      = $b3e1
fndoer      = $b3f4
strd  = $b465
strlit      = $b487
putnw1      = $b4d5
getspa      = $b4f4
garbag      = $b526
dvars = $b5bd
grbpas      = $b606
```

```
cat	= $b63d
movins	= $b67a
frestr	= $b6a3
frefac	= $b6db
chrd	= $b6ec
leftd	= $b700
rightd	= $b72c
midd	= $b737
pream	= $b761
len	= $b77c
len1	= $b782
asc	= $b78b
gtbytc	= $b79b
val	= $b7ad
strval	= $b7b5
getnum	= $b7eb
getadr	= $b7f7
peek	= $b80d
poke	= $b824
wait	= $b82d
faddh	= $b849
fsub	= $b850
fadd5	= $b862
fadd	= $b867
negfac	= $b947
overr	= $b97e
mulshf	= $b983
fone	= $b9bc		data
log	= $b9ea
fmult	= $ba28
mulply	= $ba59
conupk	= $ba8c
muldiv	= $bab7
mldvex	= $bad4
mul10	= $bae2
tenc	= $baf9		data
div10	= $bafe
fdiv	= $bb07
fdivt	= $bb0f
movfm	= $bba2
mov2f	= $bbc7
movfa	= $bbfc
movaf	= $bc0c
round	= $bc1b
sign	= $bc2b
sgn	= $bc39
abs	= $bc58
fcomp	= $bc5b
qint	= $bc9b
int	= $bccc
fin	= $bcf3
n0999	= $bdb3		data
inprt	= $bdc2
fout	= $bddd
```

```
foutim      = $be68
fhalf = $bf11          data
sqr   = $bf71
fpwrt = $bf7b
negop = $bfb4
logeb2      = $bfbf          data
exp   = $bfed
;
;
;     C64 KERNEL ROM
;
(exp  = $e000
polyx = $e043
rmulc = $e08d          data
rnd   = $e097
bioerr      = $e0f9
bchout      = $e10c
bchin = $e112
bckout      = $e118
bckin = $e11e
bgetin      = $e124
sys   = $e12a
savet = $e156
verfyt      = $e165
opent = $e1be
closet      = $e1c7
slpara      = $e1d4
combyt      = $e200
deflt = $e206
cmmerr      = $e20e
ocpara      = $e219
cos   = $e264
sin   = $e26b
tan   = $e2b4
pi2   = $e2e0          data
atn   = $e30e
atncon      = $e33e          data
bassft      = $e37b
init  = $e394
initat      = $e3a2
rndsed      = $e3ba
initcz      = $e3bf
initms      = $e422
bvtrs = $e447          data
initv = $e453
words = $e45f
-     = $e4ad
-     = $e4b7          illegal
-     = $e4da
-     = $e4e0
-     = $e4ec          data
iobase      = $e500
screen      = $e505
plot  = $e50a
```

```
cint1 = $e518
-     = $e544
-     = $e566
-     = $e56c
      = ;
-     = $e59a
lp2   = $e5b4
-     = $e5ca
-     = $e632
-     = $e684
-     = $e691
-     = $e6b6
-     = $e6ed
-     = $e701
-     = $e716
-     = $e87c
-     = $e891
-     = $e8a1
-     = $eacb
-     = $e8da
-     = $e8ea
-     = $e965
-     = $e9c8
-     = $e9e0
-     = $e9f0
-     = $e9ff
-     = $ea13
-     = $ea24
-     = $ea31
scnkey      = $ea87
-     = $eadd          data
-     = $eb79          data
-     = $eb81          data
-     = $ebc2          data
-     = $ec03
-     = $ec44          data
-     = $ec78          data
-     = $ecb9
-     = $ece7          data
-     = $ecf0
talk  = $ed09
-     = $ed40
-     = $edad
second      = $edb9
-     = $edbe
tksa  = $edc7
-     = $edcc
ciout = $eddd
untlk = $edef
acptr = $ee13
-     = $ee85
-     = $ee8e
-     = $ee97
-     = $eea0
```

```
-     = $eea9
-     = $eeb3
-     = $eebb
-     = $ef06
-     = $ef2e
-     = $ef39
-     = $ef4a
-     = $ef59
-     = $ef7e
-     = $ef90
-     = $efe1
-     = $f00d
-     = $f017
-     = $f04d
-     = $f086
-     = $f0a4
-     = $f0bd
-     = $f128
getin = $f13e
chrin = $f157
-     = $f199
chrout      = $f1ca
chkin = $f20e
chkout      = $f250
close = $f291
-     = $f30f
-     = $f31f
clall = $f32f
clrchn      = $f333
open  = $f34a
-     = $f3d5
-     = $f409
load  = $f49e
;
;--------------
;
save  = $f5dd
udtim = $f69b
rdtim = $f6dd
settim      = $f6e4
stop  = $f6ed
restor      = $fd15
vector      = $fd1a
ramtas      = $fd50
ioinit      = $fda3
setnam      = $fdf9
setlfs      = $fe00
readst      = $fe07
setmsg      = $fe18
settmo      = $fe21
memtop      = $fe25
membot      = $fe34
cint  = $fe58
```

APPENDIX C

```
;
;
; C64 KERNEL call addresses
;
acptr = $ffa5
chkin = $ffc6
chkout      = $ffc9
chrin = $ffcf
chrout      = $ffd2
ciout = $ffa8
cint  = $ff81
clall = $ffe7
close = $ffc3
clrchn      = $ffcc
getin = $ffe4
iobase      = $fff3
ioinit      = $ff84
listen      = $ffb1
load  = $ffd5
membot      = $ff9c
memtop      = $ff99
open  = $ffc0
plot  = $fff0
ramtas      = $ff87
rdtim = $ffde
readst      = $ffb7
restor      = $ff8a
save  = $ffd8
scnkey      = $ff9f
screen      = $ffed
second      = $ff93
setlfs      = $ffba
setmsg      = $ff90
setnam      = $ffbd
settim      = $ffdb
settmo      = $ffa2
stop  = $ffe1
talk  = $ffb4
tksa  = $ff96
udtim = $ffea
unlsn = $ffae
untlk = $ffab
vector      = $ff8d
;
```

APPENDIX D

OPCODES::

REPRODUCED FROM C=HACKING MAGAZINE..

6502 Opcodes and Quasi-Opcodes.
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

The following table lists all of the available opcodes on the 65xx line of micro-processors (such as the 6510 on the C=64 and the 8502 on the C=128)

Std	Mnemonic	Hex Value	Description	Addressing Mode	Bytes/Time
*	BRK	$00	Stack <- PC, PC <- ($fffe)	(Immediate)	1/7
*	ORA	$01	A <- (A) V M	(Ind,X)	6/2
	JAM	$02	[locks up machine]	(Implied)	1/-
	SLO	$03	M <- (M >> 1) + A + C	(Ind,X)	2/8
	NOP	$04	[no operation]	(Z-Page)	2/3
*	ORA	$05	A <- (A) V M	(Z-Page)	2/3
*	ASL	$06	C <- A7, A <- (A) << 1	(Z-Page)	2/5
	SLO	$07	M <- (M >> 1) + A + C	(Z-Page)	2/5
*	PHP	$08	Stack <- (P)	(Implied)	1/3
*	ORA	$09	A <- (A) V M	(Immediate)	2/2
*	ASL	$0A	C <- A7, A <- (A) << 1	(Accumalator)	1/2
	ANC	$0B	A <- A /\ M, C=~A7	(Immediate)	1/2
	NOP	$0C	[no operation]	(Absolute)	3/4
*	ORA	$0D	A <- (A) V M	(Absolute)	3/4
*	ASL	$0E	C <- A7, A <- (A) << 1	(Absolute)	3/6
	SLO	$0F	M <- (M >> 1) + A + C	(Absolute)	3/6
*	BPL	$10	if N=0, PC = PC + offset	(Relative)	2/2'2
*	ORA	$11	A <- (A) V M	((Ind),Y)	2/5'1
	JAM	$12	[locks up machine]	(Implied)	1/-
	SLO	$13	M <- (M >. 1) + A + C	((Ind),Y)	2/8'5
	NOP	$14	[no operation]	(Z-Page,X)	2/4
*	ORA	$15	A <- (A) V M	(Z-Page,X)	2/4
*	ASL	$16	C <- A7, A <- (A) << 1	(Z-Page,X)	2/6
	SLO	$17	M <- (M >> 1) + A + C	(Z-Page,X)	2/6
*	CLC	$18	C <- 0	(Implied)	1/2
*	ORA	$19	A <- (A) V M	(Absolute,Y)	3/4'1
	NOP	$1A	[no operation]	(Implied)	1/2
	SLO	$1B	M <- (M >> 1) + A + C	(Absolute,Y)	3/7
	NOP	$1C	[no operation]	(Absolute,X)	2/4'1
*	ORA	$1D	A <- (A) V M	(Absolute,X)	3/4'1
*	ASL	$1E	C <- A7, A <- (A) << 1	(Absolute,X)	3/7
	SLO	$1F	M <- (M >> 1) + A + C	(Absolute,X)	3/7
*	JSR	$20	Stack <- PC, PC <- Address	(Absolute)	3/6
*	AND	$21	A <- (A) /\ M	(Ind,X)	2/6
	JAM	$22	[locks up machine]	(Implied)	1/-
	RLA	$23	M <- (M << 1) /\ (A)	(Ind,X)	2/8
*	BIT	$24	Z <- ~(A /\ M) N<-M7 V<-M6	(Z-Page)	2/3
*	AND	$25	A <- (A) /\ M	(Z-Page)	2/3

*	ROL	$26	C <- A7 & A <- A << 1 + C	(Z-Page)	2/5
	RLA	$27	M <- (M << 1) /\ (A)	(Z-Page)	2/5'5
*	PLP	$28	A <- (Stack)	(Implied)	1/4
*	AND	$29	A <- (A) /\ M	(Immediate)	2/2
*	ROL	$2A	C <- A7 & A <- A << 1 + C	(Accumalator)	1/2
	ANC	$2B	A <- A /\ M, C <- ~A7	(Immediate9	1/2
*	BIT	$2C	Z <- ~(A /\ M) N<-M7 V<-M6	(Absolute)	3/4
*	AND	$2D	A <- (A) /\ M	(Absolute)	3/4
*	ROL	$2E	C <- A7 & A <- A << 1 + C	(Absolute)	3/6
	RLA	$2F	M <- (M << 1) /\ (A)	(Absolute)	3/6'5
*	BMI	$30	if N=1, PC = PC + offset	(Relative)	2/2'2
*	AND	$31	A <- (A) /\ M	((Ind),Y)	2/5'1
	JAM	$32	[locks up machine]	(Implied)	1/-
	RLA	$33	M <- (M << 1) /\ (A)	((Ind),Y)	2/8'5
	NOP	$34	[no operation]	(Z-Page,X)	2/4
*	AND	$35	A <- (A) /\ M	(Z-Page,X)	2/4
*	ROL	$36	C <- A7 & A <- A << 1 + C	(Z-Page,X)	2/6
	RLA	$37	M <- (M << 1) /\ (A)	(Z-Page,X)	2/6'5
*	SEC	$38	C <- 1	(Implied)	1/2
*	AND	$39	A <- (A) /\ M	(Absolute,Y)	3/4'1
	NOP	$3A	[no operation]	(Implied)	1/2
	RLA	$3B	M <- (M << 1) /\ (A)	(Absolute,Y)	3/7'5
	NOP	$3C	[no operation]	(Absolute,X)	3/4'1
*	AND	$3D	A <- (A) /\ M	(Absolute,X)	3/4'1
*	ROL	$3E	C <- A7 & A <- A << 1 + C	(Absolute,X)	3/7
	RLA	$3F	M <- (M << 1) /\ (A)	(Absolute,X)	3/7'5
*	RTI	$40	P <- (Stack), PC <-(Stack)	(Implied)	1/6
*	EOR	$41	A <- (A) \-/ M	(Ind,X)	2/6
	JAM	$42	[locks up machine]	(Implied)	1/-
	SRE	$43	M <- (M >> 1) \-/ A	(Ind,X)	2/8
	NOP	$44	[no operation]	(Z-Page)	2/3
*	EOR	$45	A <- (A) \-/ M	(Z-Page)	2/3
*	LSR	$46	C <- A0, A <- (A) >> 1	(Absolute,X)	3/7
	SRE	$47	M <- (M >> 1) \-/ A	(Z-Page)	2/5
*	PHA	$48	Stack <- (A)	(Implied)	1/3
*	EOR	$49	A <- (A) \-/ M	(Immediate)	2/2
*	LSR	$4A	C <- A0, A <- (A) >> 1	(Accumalator)	1/2
	ASR	$4B	A <- [(A /\ M) >> 1]	(Immediate)	1/2
*	JMP	$4C	PC <- Address	(Absolute)	3/3
*	EOR	$4D	A <- (A) \-/ M	(Absolute)	3/4
*	LSR	$4E	C <- A0, A <- (A) >> 1	(Absolute)	3/6
	SRE	$4F	M <- (M >> 1) \-/ A	(Absolute)	3/6
*	BVC	$50	if V=0, PC = PC + offset	(Relative)	2/2'2
*	EOR	$51	A <- (A) \-/ M	((Ind),Y)	2/5'1
	JAM	$52	[locks up machine]	(Implied)	1/-
	SRE	$53	M <- (M >> 1) \-/ A	((Ind),Y)	2/8
	NOP	$54	[no operation]	(Z-Page,X)	2/4
*	EOR	$55	A <- (A) \-/ M	(Z-Page,X)	2/4
*	LSR	$56	C <- A0, A <- (A) >> 1	(Z-Page,X)	2/6
	SRE	$57	M <- (M >> 1) \-/ A	(Z-Page,X)	2/6
*	CLI	$58	I <- 0	(Implied)	1/2
*	EOR	$59	A <- (A) \-/ M	(Absolute,Y)	3/4'1
	NOP	$5A	[no operation]	(Implied)	1/2
	SRE	$5B	M <- (M >> 1) \-/ A	(Absolute,Y)	3/7

	NOP	$5C	[no operation]	(Absolute,X)	3/4'1
*	EOR	$5D	A <- (A) \-/ M	(Absolute,X)	3/4'1
	SRE	$5F	M <- (M >> 1) \-/ A	(Absolute,X)	3/7
*	RTS	$60	PC <- (Stack)	(Implied)	1/6
*	ADC	$61	A <- (A) + M + C	(Ind,X)	2/6
	JAM	$62	[locks up machine]	(Implied)	1/-
	RRA	$63	M <- (M >> 1) + (A) + C	(Ind,X)	2/8'5
	NOP	$64	[no operation]	(Z-Page)	2/3
*	ADC	$65	A <- (A) + M + C	(Z-Page)	2/3
*	ROR	$66	C<-A0 & A<- (A7=C + A>>1)	(Z-Page)	2/5
	RRA	$67	M <- (M >> 1) + (A) + C	(Z-Page)	2/5'5
*	PLA	$68	A <- (Stack)	(Implied)	1/4
*	ADC	$69	A <- (A) + M + C	(Immediate)	2/2
*	ROR	$6A	C<-A0 & A<- (A7=C + A>>1)	(Accumulator)	1/2
	ARR	$6B	A <- [(A /\ M) >> 1]	(Immediate)	1/2'5
*	JMP	$6C	PC <- Address	(Indirect)	3/5
*	ADC	$6D	A <- (A) + M + C	(Absolute)	3/4
*	ROR	$6E	C<-A0 & A<- (A7=C + A>>1)	(Absolute)	3/6
	RRA	$6F	M <- (M >> 1) + (A) + C	(Absolute)	3/6'5
*	BVS	$70	if V=1, PC = PC + offset	(Relative)	2/2'2
*	ADC	$71	A <- (A) + M + C	((Ind),Y)	2/5'1
	JAM	$72	[locks up machine]	(Implied)	1/-
	RRA	$73	M <- (M >> 1) + (A) + C	((Ind),Y)	2/8'5
	NOP	$74	[no operation]	(Z-Page,X)	2/4
*	ADC	$75	A <- (A) + M + C	(Z-Page,X)	2/4
*	ROR	$76	C<-A0 & A<- (A7=C + A>>1)	(Z-Page,X)	2/6
	RRA	$77	M <- (M >> 1) + (A) + C	(Z-Page,X)	2/6'5
*	SEI	$78	I <- 1	(Implied)	1/2
*	ADC	$79	A <- (A) + M + C	(Absolute,Y)	3/4'1
	NOP	$7A	[no operation]	(Implied)	1/2
	RRA	$7B	M <- (M >> 1) + (A) + C	(Absolute,Y)	3/7'5
	NOP	$7C	[no operation]	(Absolute,X)	3/4'1
*	ADC	$7D	A <- (A) + M + C	(Absolute,X)	3/4'1
*	ROR	$7E	C<-A0 & A<- (A7=C + A>>1)	(Absolute,X)	3/7
	RRA	$7F	M <- (M >> 1) + (A) + C	(Absolute,X)	3/7'5
	NOP	$80	[no operation]	(Immediate)	2/2
*	STA	$81	M <- (A)	(Ind,X)	2/6
	NOP	$82	[no operation]	(Immediate)	2/2
	SAX	$83	M <- (A) /\ (X)	(Ind,X)	2/6
*	STY	$84	M <- (Y)	(Z-Page)	2/3
*	STA	$85	M <- (A)	(Z-Page)	2/3
*	STX	$86	M <- (X)	(Z-Page)	2/3
	SAX	$87	M <- (A) /\ (X)	(Z-Page)	2/3
*	DEY	$88	Y <- (Y) - 1	(Implied)	1/2
	NOP	$89	[no operation]	(Immediate)	2/2
*	TXA	$8A	A <- (X)	(Implied)	1/2
	ANE	$8B	M <-[(A)\/$EE] /\ (X)/\(M)	(Immediate)	2/2^4
*	STY	$8C	M <- (Y)	(Absolute)	3/4
*	STA	$8D	M <- (A)	(Absolute)	3/4
*	STX	$8E	M <- (X)	(Absolute)	3/4
	SAX	$8F	M <- (A) /\ (X)	(Absolute)	3/4
*	BCC	$90	if C=0, PC = PC + offset	(Relative)	2/2'2
*	STA	$91	M <- (A)	((Ind),Y)	2/6
	JAM	$92	[locks up machine]	(Implied)	1/-

	SHA	$93	M <- (A) /\ (X) /\ (PCH+1)	(Absolute,X)	3/6'3
*	STY	$94	M <- (Y)	(Z-Page,X)	2/4
*	STA	$95	M <- (A)	(Z-Page,X)	2/4
	SAX	$97	M <- (A) /\ (X)	(Z-Page,Y)	2/4
*	STX	$96	M <- (X)	(Z-Page,Y)	2/4
*	TYA	$98	A <- (Y)	(Implied)	1/2
*	STA	$99	M <- (A)	(Absolute,Y)	3/5
*	TXS	$9A	S <- (X)	(Implied)	1/2
	SHS	$9B	X <- (A) /\ (X), S <- (X)	(Absolute,Y)	3/5
			M <- (X) /\ (PCH+1)		
	SHY	$9C	M <- (Y) /\ (PCH+1)	(Absolute,Y)	3/5'3
*	STA	$9D	M <- (A)	(Absolute,X)	3/5
	SHX	$9E	M <- (X) /\ (PCH+1)	(Absolute,X)	3/5'3
	SHA	$9F	M <- (A) /\ (X) /\ (PCH+1)	(Absolute,Y)	3/5'3
*	LDY	$A0	Y <- M	(Immediate)	2/2
*	LDA	$A1	A <- M	(Ind,X)	2/6
*	LDX	$A2	X <- M	(Immediate)	2/2
	LAX	$A3	A <- M, X <- M	(Ind,X)	2/6
*	LDY	$A4	Y <- M	(Z-Page)	2/3
*	LDA	$A5	A <- M	(Z-Page)	2/3
*	LDX	$A6	X <- M	(Z-Page)	2/3
	LAX	$A7	A <- M, X <- M	(Z-Page)	2/3
*	TAY	$A8	Y <- (A)	(Implied)	1/2
*	LDA	$A9	A <- M	(Immediate)	2/2
*	TAX	$AA	X <- (A)	(Implied)	1/2
	LXA	$AB	X04 <- (X04) /\ M04	(Immediate)	1/2
			A04 <- (A04) /\ M04		
*	LDY	$AC	Y <- M	(Absolute)	3/4
*	LDA	$AD	A <- M	(Absolute)	3/4
*	LDX	$AE	X <- M	(Absolute)	3/4
	LAX	$AF	A <- M, X <- M	(Absolute)	3/4
*	BCS	$B0	if C=1, PC = PC + offset	(Relative)	2/2'2
*	LDA	$B1	A <- M	((Ind),Y)	2/5'1
	JAM	$B2	[locks up machine]	(Implied)	1/-
	LAX	$B3	A <- M, X <- M	((Ind),Y)	2/5'1
*	LDY	$B4	Y <- M	(Z-Page,X)	2/4
*	LDA	$B5	A <- M	(Z-Page,X)	2/4
*	LDX	$B6	X <- M	(Z-Page,Y)	2/4
	LAX	$B7	A <- M, X <- M	(Z-Page,Y)	2/4
*	CLV	$B8	V <- 0	(Implied)	1/2
*	LDA	$B9	A <- M	(Absolute,Y)	3/4'1
*	TSX	$BA	X <- (S)	(Implied)	1/2
	LAE	$BB	X,S,A <- (S /\ M)	(Absolute,Y)	3/4'1
*	LDY	$BC	Y <- M	(Absolute,X)	3/4'1
*	LDA	$BD	A <- M	(Absolute,X)	3/4'1
*	LDX	$BE	X <- M	(Absolute,Y)	3/4'1
	LAX	$BF	A <- M, X <- M	(Absolute,Y)	3/4'1
*	CPY	$C0	(Y - M) -> NZC	(Immediate)	2/2
*	CMP	$C1	(A - M) -> NZC	(Ind,X)	2/6
	NOP	$C2	[no operation]	(Immediate)	2/2
	DCP	$C3	M <- (M)-1, (A-M) -> NZC	(Ind,X)	2/8
*	CPY	$C4	(Y - M) -> NZC	(Z-Page)	2/3
*	CMP	$C5	(A - M) -> NZC	(Z-Page)	2/3
*	DEC	$C6	M <- (M) - 1	(Z-Page)	2/5

	DCP	$C7	M <- (M)-1, (A-M) -> NZC	(Z-Page)	2/5
*	INY	$C8	Y <- (Y) + 1	(Implied)	1/2
*	CMP	$C9	(A - M) -> NZC	(Immediate)	2/2
*	DEX	$CA	X <- (X) - 1	(Implied)	1/2
	SBX	$CB	X <- (X)/\(A) - M	(Immediate)	2/2
*	CPY	$CC	(Y - M) -> NZC	(Absolute)	3/4
*	CMP	$CD	(A - M) -> NZC	(Absolute)	3/4
*	DEC	$CE	M <- (M) - 1	(Absolute)	3/6
	DCP	$CF	M <- (M)-1, (A-M) -> NZC	(Absolute)	3/6
*	BNE	$D0	if Z=0, PC = PC + offset	(Relative)	2/2'2
*	CMP	$D1	(A - M) -> NZC	((Ind),Y)	2/5'1
	JAM	$D2	[locks up machine]	(Implied)	1/-
	DCP	$D3	M <- (M)-1, (A-M) -> NZC	((Ind),Y)	2/8
	NOP	$D4	[no operation]	(Z-Page,X)	2/4
*	CMP	$D5	(A - M) -> NZC	(Z-Page,X)	2/4
*	DEC	$D6	M <- (M) - 1	(Z-Page,X)	2/6
	DCP	$D7	M <- (M)-1, (A-M) -> NZC	(Z-Page,X)	2/6
*	CLD	$D8	D <- 0	(Implied)	1/2
*	CMP	$D9	(A - M) -> NZC	(Absolute,Y)	3/4'1
	NOP	$DA	[no operation]	(Implied)	1/2
	DCP	$DB	M <- (M)-1, (A-M) -> NZC	(Absolute,Y)	3/7
	NOP	$DC	[no operation]	(Absolute,X)	3/4'1
*	CMP	$DD	(A - M) -> NZC	(Absolute,X)	3/4'1
*	DEC	$DE	M <- (M) - 1	(Absolute,X)	3/7
	DCP	$DF	M <- (M)-1, (A-M) -> NZC	(Absolute,X)	3/7
*	CPX	$E0	(X - M) -> NZC	(Immediate)	2/2
*	SBC	$E1	A <- (A) - M - ~C	(Ind,X)	2/6
	NOP	$E2	[no operation]	(Immediate)	2/2
	ISB	$E3	M <- (M) - 1,A <- (A)-M-~C	(Ind,X)	3/8'1
*	CPX	$E4	(X - M) -> NZC	(Z-Page)	2/3
*	SBC	$E5	A <- (A) - M - ~C	(Z-Page)	2/3
*	INC	$E6	M <- (M) + 1	(Z-Page)	2/5
	ISB	$E7	M <- (M) - 1,A <- (A)-M-~C	(Z-Page)	2/5
*	INX	$E8	X <- (X) +1	(Implied)	1/2
*	SBC	$E9	A <- (A) - M - ~C	(Immediate)	2/2
*	NOP	$EA	[no operation]	(Implied)	1/2
	SBC	$EB	A <- (A) - M - ~C	(Immediate)	1/2
*	SBC	$ED	A <- (A) - M - ~C	(Absolute)	3/4
*	CPX	$EC	(X - M) -> NZC	(Absolute)	3/4
*	INC	$EE	M <- (M) + 1	(Absolute)	3/6
	ISB	$EF	M <- (M) - 1,A <- (A)-M-~C	(Absolute)	3/6
*	BEQ	$F0	if Z=1, PC = PC + offset	(Relative)	2/2'2
*	SBC	$F1	A <- (A) - M - ~C	((Ind),Y)	2/5'1
	JAM	$F2	[locks up machine]	(Implied)	1/-
	ISB	$F3	M <- (M) - 1,A <- (A)-M-~C	((Ind),Y)	2/8
	NOP	$F4	[no operation]	(Z-Page,X)	2/4
*	SBC	$F5	A <- (A) - M - ~C	(Z-Page,X)	2/4
*	INC	$F6	M <- (M) + 1	(Z-Page,X)	2/6
	ISB	$F7	M <- (M) - 1,A <- (A)-M-~C	(Z-Page,X)	2/6
*	SED	$F8	D <- 1	(Implied)	1/2
*	SBC	$F9	A <- (A) - M - ~C	(Absolute,Y)	3/4'1
	NOP	$FA	[no operation]	(Implied)	1/2
	ISB	$FB	M <- (M) - 1,A <- (A)-M-~C	(Absolute,Y)	3/7
	NOP	$FC	[no operation]	(Absolute,X)	3/4'1

```
*   SBC      $FD       A <- (A) - M - ~C          (Absolute,X)     3/4'1
*   INC      $FE       M <- (M) + 1               (Absolute,X)     3/7
    ISB      $FF       M <- (M) - 1,A <- (A)-M-~C (Absolute,X)     3/7
```

'1 - Add one if address crosses a page boundry.
'2 - Add 1 if branch succeeds, or 2 if into another page.
'3 - If page boundry crossed then PCH+1 is just PCH
'4 - Sources disputed on exact operation, or sometimes does not work.
'5 - Full eight bit rotation (with carry)

Sources:
  Programming the 6502, Rodney Zaks, (c) 1983 Sybex
  Paul Ojala, Post to Comp.Sys.Cbm (po87553@cs.tut.fi / albert@cc.tut.fi)
  D John Mckenna, Post to Comp.Sys.Cbm (gudjm@uniwa.uwa.oz.au)

Compiled by Craig Taylor (duck@pembvax1.pembroke.edu)

APPENDIX E

```
; C64 Kernal Jump Table
;
ff81  jmp $ff5b   cint        Init Editor & Video Chips
ff84  jmp $fd23   ioinit            Init I/O Devices, Ports & Timers
ff87  jmp $fd50   ramtas            Init Ram & Buffers
ff8a  jmp $fd15   restor            Restore Vectors
ff8d  jmp $fd1a   vector            Change Vectors For User
ff90  jmp $fe18   setmsg            Control OS Messages
ff93  jmp $edb9   secnd       Send SA After Listen
ff96  jmp $edc7   tksa        Send SA After Talk
ff99  jmp $fe25   memtop            Set/Read System RAM Top
ff9c  jmp $fe34   membot            Set/Read System RAM Bottom
ff9f  jmp $ea87   scnkey            Scan Keyboard
ffa2  jmp $fe21   settmo            Set Timeout In IEEE
ffa5  jmp $ee13   acptr       Handshake Serial Byte In
ffa8  jmp $eddd   ciout       Handshake Serial Byte Out
ffab  jmp $edef   untalk            Command Serial Bus UNTALK
ffae  jmp $edfe   unlsn       Command Serial Bus UNLISTEN
ffb1  jmp $ed0c   listn       Command Serial Bus LISTEN
ffb4  jmp $ed09   talk        Command Serial Bus TALK
ffb7  jmp $fe07   readss            Read I/O Status Word
ffba  jmp $fe00   setlfs            Set Logical File Parameters
ffbd  jmp $fdf9   setnam            Set Filename
ffc0  jmp ($031a) (iopen)           Open Vector [f34a]
ffc3  jmp ($031c) (iclose)    Close Vector [f291]
ffc6  jmp ($031e) (ichkin)    Set Input [f20e]
ffc9  jmp ($0320) (ichkout)   Set Output [f250]
ffcc  jmp ($0322) (iclrch)    Restore I/O Vector [f333]
ffcf  jmp ($0324) (ichrin)    Input Vector, chrin [f157]
ffd2  jmp ($0326) (ichrout)   Output Vector, chrout [f1ca]
ffd5  jmp $f49e   load        Load RAM From Device
ffd8  jmp $f5dd   save        Save RAM To Device
ffdb  jmp $f6e4   settim            Set Real-Time Clock
ffde  jmp $f6dd   rdtim       Read Real-Time Clock
```

```
ffe1  jmp ($0328) (istop)          Test-Stop Vector [f6ed]
ffe4  jmp ($032a) (igetin)    Get From Keyboad [f13e]
ffe7  jmp ($032c) (iclall)    Close All Channels And Files [f32f]
ffea  jmp $f69b   udtim       Increment Real-Time Clock
ffed  jmp $e505   screen           Return Screen Organization
fff0  jmp $e50a   plot        Read / Set Cursor X/Y Position
fff3  jmp $e500   iobase           Return I/O Base Address

;fff6 Vectors

fff6  [5252]           -
fff8  [5942]           SYSTEM

;fffa Transfer Vectors
fffa  [fe43]           NMI
fffc  [fce2]           RESET
fffe  [ff48]           IRQ
```

APPENDIX F

BASIC KEYWORDS

COMMODORE BASIC KEYWORDS

Common Keywords (Tokens 80 - CB)

Tokens 80 to A2 represent action keywords, while codes B4 trough CA are function keywords. AA - B3 are BASIC operators.

```
Token Keyword

80    end
81    for
82    next
83    data
84    input#
85    input
86    dim
87    read

88    let
89    goto
8a    run
8b    if
8c    restore
8d    gosub
8e    return
8f    rem

90    stop
91    on
92    wait
```

```
93    load
94    save
95    verify
96    def
97    poke

98    print#
99    print
9a    cont
9b    list
9c    clr
9d    cmd
9e    sys
9f    open

a0    close
a1    get
a2    new
------------------
a3    tab(
a4    to
a5    fn
a6    spc(
a7    then

a8    not
a9    step
------------------
aa    +
ab    -
ac    *
ad    /
ae    ^
af    and

b0    or
b1    >
b2    =
b3    <
------------------
b4    sgn
b5    int
b6    abs
b7    usr

b8    fre
b9    pos
ba    sqr
bb    rnd
bc    log
bd    exp
be    cos
bf    sin
```

c0	tan
c1	atn
c2	peek
c3	len
c4	str$
c5	val
c6	asc
c7	chr$
c8	left$
c9	right$
ca	mid$
------------------	
cb	go
ff	pi

Extension Keywords (Tokens CC - FE)

The following codes are defined differently in each Basic version. The leftmost column shows VIC Super Expander commands (CC trough DD). Basic 3.5 and 7.0 differ in codes CE and FE, which are prefixes in 7.0, whereas in 3.5 CE = rlum and FE is unused.

Codes CC to D4 (3.5, 7.0 and 10.0) are function keywords, and D5 trough FA are action keywords.

Token	Keyword			
	2.0 Super	4.0	3.5/7.0	10.0
cc	key	concat	rgr	rgr 2)
cd	graphic	dopen	rclr	rclr 2)
ce	scnclr	dclose	rlum/*prefix*	*prefix*
cf	circle	record	joy	joy
d0	draw	header	rdot	rdot 2)
d1	region	collect	dec	dec
d2	color	backup	hex$	hex$
d3	point	copy	err$	err$
d4	sound	append	instr	instr
d5	char	dsave	else	else
d6	paint	dload	resume	resume
d7	rpot	catalog	trap	trap
d8	rpen	rename	tron	tron
d9	rsnd	scratch	troff	troff
da	rcolr	directory	sound	sound
db	rgr		vol	vol
dc	rjoy		auto	auto
dd	rdot		pudef	pudef
de			graphic	graphic

df	paint	paint 2)
e0	char	char
e1	box	box
e2	circle	circle
e3	gshape	paste 2)
e4	sshape	cut 2)
e5	draw	line
e6	locate	locate 2)
e7	color	color
e8	scnclr	scnclr
e9	scale	scale 2)
ea	help	help
eb	do	do
ec	loop	loop
ed	exit	exit
ee	directory	dir
ef	dsave	dsave
f0	dload	dload
f1	header	header
f2	scratch	scratch
f3	collect	collect
f4	copy	copy
f5	rename	rename
f6	backup	backup
f7	delete	delete
f8	renumber	renumber
f9	key	key
fa	monitor	monitor
	--------------------------	
fb	using	using
fc	until	until
fd	while	while
fe	*prefix*	*prefix*

Prefixed Extension Keywords (Tokens CE02 - CE0A)

The following codes implement function keywords. Basics 7.0 and 10.0 only.

Token	Keyword
ce00	
ce01	
ce02	pot
ce03	bump
ce04	pen
ce05	rspos

```
ce06  rsprite
ce07  rspcolor

ce08  xor
ce09  rwindow
ce0a  pointer
```

Prefixed Extension Keywords (Tokens FE02 - FE26)

The following codes are for 7.0 and 10.0 only. Keywords in the middle are commom.

```
Token       Keyword
       7.0           10.0

fe00
fe01
fe02        bank
fe03        filter
fe04        play
fe05        tempo
fe06        movspr
fe07        sprite

fe08        sprcolor
fe09        rreg
fe0a        envelope
fe0b        sleep
fe0c        catalog
fe0d        dopen
fe0e        append
fe0f        dclose

fe10        bsave
fe11        bload
fe12        record
fe13        concat
fe14        dverify
fe15        dclear
fe16        sprsav
fe17        collision

fe18        begin
fe19        bend
fe1a        window
fe1b        boot
fe1c        width  2)
fe1d        sprdef 2)
fe1e        quit 1) 2)
fe1f   stash            dma
```

```
fe20
fe21   fetch            dma
fe22
fe23   swap       dma
fe24        off 1) 2)
fe25        fast
fe26        slow
fe27              type

fe28              bverify
fe29              ectory (diRectorY)
fe2a              erase
fe2b              find
fe2c              change
fe2d              set  3)
fe2e              screen
fe2f              polygon

fe30              ellipse
fe31              viewport 2)
fe32              gcopy 2)
fe33              pen
fe34              palette
fe35              dmode
fe36              dpat
fe37              pic  2)

fe38              genlock
fe39              foreground
fe3a
fe3b              background
fe3c              border
fe3d              highlight
```

Notes:

1) Gives "unimplemented command error" on BASIC 7.0
2) Gives "unimplemented command error" on BASIC 10.0 v0.9
3) Only 'set def' is implemented.

APPENDIX G

REU'S

The following is based on the Commodore 1764 user's manual (german version)

Contents:

1) _External RAM Access With REUs_

The REUs provide additional RAM for the C64/128. Three types of REUs have been produced by Commodore. These are the 1700, 1764 and 1750 with 128, 256 and 512 KBytes built in RAM. However they can be extended up to several MBytes.   The external memory can not be addressed directly by the C64 with it's 16-bit address space. It has to be transferred from an to the main memory of the C64. For that purpose there is a built in RAM Expansion Controller (REC) which transfers memory between the C64 and the REU using Direct Memory Access (DMA). It can also be used for other purposes.

REU means Ram Expansion Unit. There are several different ones. The official Commodore REU's are the 1700, 1764 and 1750 which are respectively 128, 256 and 512Kb of memory (not directly addressable of course). There seem to be hacks to expand these to 1Mb or even 2Mb. I myself have recently made 512K in the 256K cartridge without any difficulties. CLD, an american company makes clones of the 1750 and maybe others. These clones are smaller than the originals but probably not as expandable. I have a 1750 Clone (512Kb) and it seems to be 100% compatible (no, not 99.9% but really 100%).

Furthermore there is the Georam expansion. This cartridge is ugly as hell and only works with GEOS. I believe it's also 512K. In my opinion, the real REU is better in every respect. (W. Lamee)

---

2) _RAM Expansion Controller (REC) Registers_

The REC is programmed by accessing it's registers, that appear memory mapped in the I/O-area between $DF00 and $DF0A when a REU is connected through the expansion port of the C64. They can be read and written to like VIC- and SID-registers.

```
$DF00: STATUS REGISTER
       various information can be obtained (read only)

Bit 7:     INTERRUPT PENDING  (1 = interrupt waiting to be served)
           unnecessary
Bit 6:     END OF BLOCK  (1 = transfer complete)
```

```
              unnecessary
Bit 5:        FAULT  (1 = block verify error)
              Set if a difference between C64- and REU-memory areas was
found
              during a compare-command.
Bit 4:        SIZE  (1 = 256 KB)
              Seems to indicate the size of the RAM-chips. It is set on 1764
              and 1750 and clear on 1700.
Bits 3..0: VERSION
              Contains 0 on my REU.

$DF01: COMMAND REGISTER
       By writing to this register RAM transfer or comparision can be
       executed.

Bit 7:        EXECUTE  (1 = transfer per current configuration)
              This bit must be set to execute a command.
Bit 6:        reserved  (normally 0)
Bit 5:        LOAD  (1 = enable autoload option)
              With autoload enabled the address and length registers (see
              below) will be unchanged after a command execution.
              Otherwise the address registers will be counted up to the
              address off the last accessed byte of a DMA + 1,
              and the length register will be changed (normally to 1).
Bit 4:        FF00
              If this bit is set command execution starts immediately
              after setting the command register.
              Otherwise command execution is delayed until write access to
              memory position $FF00
Bits 3..2: reserved  (normally 0)
Bits 1..0: TRANSFER TYPE
              00 = transfer C64 -> REU
              01 = transfer REU -> C64
              10 = swap C64 <-> REU
              11 = compare C64 - REU

$DF02..$DF03: C64 BASE ADDRESS
              A 16-bit C64 - base address in low/high order.

$DF04..$DF06: REU BASE ADDRESS
              This is a three byte address consisting of a low and
              high byte and an expansion bank number.
              Normally only bits 2..0 of the expansion bank are valid
              (for a maximum of 512 KByte), the other bits are always
              set. This must be different if more than 512 KByte are
              installed.

$DF07..$DF08: TRANSFER LENGTH
              This is a 16-bit value containing the number of bytes to
              transfer or compare.
              The value 0 stands for 64 Kbytes.
              If the transfer length plus the C64 base address exceeds
              64K the C64 address will overflow and cause C64 memory
              from 0 on to be accessed.
```

```
              If the transfer length plus the REU base address exceeds
              512K the REU address will overflow and cause REU memory
              from 0 on to be accessed.

$DF09: INTERRUPT MASK REGISTER
       unnecessary

Bit 7:     INTERRUPT ENABLE  (1 = interrupt enabled)
Bit 6:     END OF BLOCK MASK  (1 = interrupt on end)
Bit 5:     VERIFY ERROR  (1 = interrupt on verify error)
Bits 4..0: unused (normally all set)

$DF0A: ADDRESS CONTROL REGISTER
       Controlls the address counting during DMA.
       If an address is fixed, not a memory block but always the same
       byte addressed by the base address register is used for DMA.

Bit 7:     C64 ADDRESS CONTROL  (1 = fix C64 address)
Bit 6:     REU ADDRESS CONTROL  (1 = fix REU address)
Bits 5..0: unused (normally all set)
```

To access the REU-registers in assembly language it is convenient to define labels something like this:

```
    status   = $DF00
    command  = $DF01
    c64base  = $DF02
    reubase  = $DF04
    translen = $DF07
    irqmask  = $DF09
    control  = $DF0A
```

3) _How To Recognize The REU_

Normally the addresses between $DF00 and $DF0A are unused. So normally if values are stored there they get lost. So if you write e.g. the values 1,2,3,... to $DF02..$DF08 and they don't stay there you can be sure that no REU is connected. However if the values are there it could be because another kind of module is connected that also uses these addresses. Another problem is the recognition of the number of RAM banks (64 KByte units) installed. The SIZE bit only tells that there are at least 2 (1700) or 4 (1764, 1750) banks installed. By trying to access & verify bytes in as many RAM banks as possible the real size can be determined. This can be seen in the source to "Dynamic memory allocation for the 128" in Commodore Hacking Issue 2. (He) personally prefer(s) to let the user choose if and which REU banks shall be used.

4) _Simple RAM Transfer_

Very little options of the REU are necessary for the main purposes of RAM expanding.

Just set the base addresses, transfer length and then the command register.

The following code transfers one KByte containing the screen memory ($0400..$07FF) to address 0 in the REU:

```
lda #0
sta control ; to make sure both addresses are counted up
lda #<$0400
sta c64base
lda #>$0400
sta c64base + 1
lda #0
sta reubase
sta reubase + 1
sta reubase + 2
lda #<$0400
sta translen
lda #>$0400
sta translen + 1
lda #%10010000;  c64 -> REU with immediate execution
sta command
```

To transfer the memory back to the C64 replace "lda #%10010000" by "lda #%10010001".

I think that this subset of 17xx functions would be enough for a reasonable RAM expansion. However if full compatibility with 17xx REUs is desired also the more complicated functions have to be implemented.

5) _Additional Features_

Swapping Memory

With the swap-command memory between 17xx and C64 is exchanged. The programming is the same as in simple RAM transfer.

Comparing Memory

No RAM is transferred but the number of bytes specified in the transfer length register is compared. If there are differences the FAULT-bit of the status register is set. This bit is cleared by reading the status register which has to be done before comparing to get valid information.

Using All C64 Memory

C64 memory is accessed by the REU normally in the memory configuration existing during writing to the command register. However in order to be able to write to the command register the I/O-area has to be active.

  If RAM between $D000 and $DFFF or character ROM shall be used it is possible to delay the execution of the command by storing a command byte with bit 4 ("FF00") cleared. The command will then be executed by writing any value to address $FF00.

Example:

```
    < Set base addresses and transfer length >
    lda #%10000000 ; transfer C64 RAM -> REU delayed
    sta command
    sei
    lda $01
    and #$30
    sta $01 ; switch on 64 KByte RAM
    lda $FF00 ; to not change the contents of $FF00
    sta $FF00 ; execute DMA
    lda $01
    ora #$37
    sta $01 ; switch on normal configuration
    cli
```

6) _Transfer Speed_

  During DMA the CPU is halted and the memory access cycles normally available for the CPU are now used to access one byte each. So with screen and sprites switched off in every clock cycle (985248 per second on PAL machines) a byte is transferred. If screen is on or sprites are enabled transfer is a bit slower, as the VIC exclusively accesses RAM sometimes. An exact description of those "missing cycles" can be found in Commodore Hacking Issue 3.

  Comparing memory areas is as fast as transfers. (Comparison is stopped once the first difference is found.)

  Swapping memory is only half as fast, as for every bytes two C64 memory accesses (read & write) are necessary.

7) _Interrupts_

  By setting certain bits in the interrupt mask register IRQs at the end of a DMA can be selected. However as the CPU is halted during DMA it will always be finished after the store instruction into the command register or $FF00. So there is no need to check for an "END OF BLOCK" (bit 6 of status register) or to enable an interrupt.

8) _Executing Code In Expanded Memory_

  Code in external memory has always to be copied into C64 memory to be executed. This is a disadvantage against bank switching systems. However bank switching can be simulated by the SWAP command. This is done e.g. in RAMDOS where only 256 bytes of C64 memory are occupied, the 6 KByte RAM disk driver is swapped in whenever needed. Probably too much swapping is the reason for RAMDOS to be not really fast at sequential file access.

9) _Other Useful Applications Of The REU_

The REC is not only useful for RAM transfer and comparison.

One other application (used in GEOS) is to copy C64 RAM areas by first transferring it to the REU and then transferring it back into the desired position in C64 memory. Due to the fast DMA this is about 5 times faster than copying memory with machine language instructions.

Interesting things can be done by fixing base addresses. Large C64 areas can be filled very fast with a single byte value by fixing the REU base address. Thus it is also possible to find the end of an area containing equal bytes very fast e.g. for data compression.

Fixing the C64 base address is interesting if an I/O-port is used, as data can be written out faster than normally possible.
It would be possible to use real bitmap graphics in the upper and lower screen border by changing the "magic byte" (highest by the VIC addressed byte) in every clock cycle during the border switched off.

Generally the REC could be used as graphics accelerator e.g. to copy bitmap areas or to copy data fast into the VIC-addressable 16 KByte area.

10) _Comparision Of Bank Switching and DMA_

When comparing bank switching and DMA for memory expansion I think DMA is the more comfortable methode to program and also is faster in most cases. The disadvantage with code execution not possible in external memory could be minimized by copying only the necessary parts into C64 memory. Executing the code will take much more time than copying it into C64 memory.

APPENDIX H

ABOUT THE PROCESSOR CHIP

C= Commodore Semiconductor Group

Microprocessors

Description
The 6500/8500 Series family includes a range of software compatible micropro-
cessors which provide a selection of addressable memory range, interrupt input
options and on-chip oscillators and drivers. All of the microprocessors within
the group are directly bus compatible with the MC6800 series IC's.

The family includes ten microprocessors with on-board clock oscillators and seven microprocessors driven by external clocks. The on-chip clock versions are aimed at high performance, low cost applications where single phase crystal or RC inputs provide the time base. The external clock versions are geared for multiprocessor system applications where maximum timing control is mandatory.

Features

- Single +5 volt supply
- N channel, silicon gate, depletion load technology
- Tri-state address bus, data bus and R/W controlled by AEC input
- Direct memory access capability
- "Ready" input (for single cycle execution)
- 56 Instructions with 13 addressing modes
- 8 bit parallel processing
- Decimal and binary arithmetic
- True indexing capability
- 8 bit Bi-directional Data Bus
- Programmable Stack Pointer

Available Microprocessors

Device	*Clocks	Pins	IRQ	NMI	RDY	Port	Address	AEC	Sync	Speed (MHz)
6502	O	40	X	X	X	-	64K	-	X	1,2,3,4
65CE02	O	40	X	X	X	-	64K	-	X	0 - 10
6503	O	28	X	X	-	-	4K	-	-	1,2,3,4
6504	O	28	X	-	-	-	8K	-	-	1,2,3,4
6505	O	28	X	-	X	-	4K	-	-	1,2,3,4
6506	O	28	X	-	-	-	4K	-	-	1,2,3,4
6507	O	28	-	-	X	-	8K	-	-	1,2,3,4
6508	E	40	X	-	-	8	64K	X	-	1,2,3
6509	E	40	X	X	X	**	1 M	X	X	1,2,3
6510	O,E	40	X	X	X	6,8	64K	X	-	1,2,3,4
6512	E	40	X	X	X	-	64K	-	X	1,2,3,4
6513	E	28	X	X	-	-	4K	-	-	1,2,3,4
6514	E	28	X	-	-	-	8K	-	-	1,2,3,4
6515	E	28	X	-	X	-	4K	-	-	1,2,3,4
8501	O	40	X	-	X	7	64K	X	-	1,2,3
8502	O	40	X	X	X	7	64K	X	-	1,2,3,4
8503	O	40	X	-	-	8	64K	X	-	1,2,3,4

* O - On chip clocks, E - External Clocks

** Four extended address pins expand memory capacity to one megabyte.

Pinout

Pin	6502	6510/8500	8502
1	Vss	Phi0 in	Phi0 in
2	RDY	RDY	RDY
3	Phi1 out	/IRQ	/IRQ
4	/IRQ	/NMI	/NMI
5	NC	AEC	AEC
6	/NMI	Vcc	Vcc
7	Sync	A0	A0
8	Vcc	A1	A1
9	AB0	A2	A2
10	AB1	A3	A3
11	AB2	A4	A4
12	AB3	A5	A5
13	AB4	A6	A6
14	AB5	A7	A7
15	AB6	A8	A8
16	AB7	A9	A9
17	AB8	A10	A10
18	AB9	A11	A11
19	AB10	A12	A12
20	AB11	A13	A13
21	Vss	GND	GND
22	AB12	A14	A14
23	AB13	A15	A15
24	AB14	P5	P6
25	AB15	P4	P5
26	D7	P3	P4
27	D6	P2	P3
28	D5	P1	P2
29	D4	P0	P1
30	D3	D7	P0
31	D2	D6	D7
32	D1	D5	D6
33	D0	D4	D5
34	R/W	D3	D4
35	NC	D2	D3
36	NC	D1	D2
37	Phi0 in	D0	D1
38	SO	R/W	D0
39	Phi2 out	Phi2 out	R/W
40	/RES	/RES	/RES

APPENDIX I

DIFFERENCES IN PROCESSORS

I told you that I'd come back with something like this, so here it is!

This is taken from CHacking..

"Q $03F) Now, for those into 6502 machine language. What instruction was not available on the first 6502 chips?

A $03F) ROR (ROtate Right) was not available until after June, 1976. However,
all Commodore VICs and C64s should have this instruction. Some people
gave instructions that are found on the 65c02, designed by Western
Design Center, and licensed to many companies. However, the 65c02
itself occurs in two flavors, and neither are used in any stock
Commodore product I know of."

Here's another interesting tidbit (from CHACKING)

It seems that the "6510 internal registers were grafted onto a 6502 core processor."

64 KERNAL ROM DIFFERENCES
Date: Fri Jun 17 16:38:46 1994
Received: from funet.fi by oulu.fi (4.1/SMI-4.1)

6.2 Commodore 64 KERNAL ROM versions.

Below is information on differences between the Commodore 64 KERNAL revisions R1, R2, R3 and the Commodore SX-64 and the Commodore 4064 ROMs. The chronological order must be R1, R2, 4064, R3 and SX-64.

The KERNAL ROM R1 was obviously used only in early NTSC systems. It lacks the PAL/NTSC detection, and always uses white color while clearing the screen. The white color feature is from the VIC-20 ROM, but the VIC had a white background by default. Thus, this feature can be listed as a bug. The CIA 1 timer A will always divide the system clock through $411C == 16668. The other ROMs use the values $4026 an $4296, depending on the system version (PAL/NTSC), so their interrupt frequency is 985248 Hz / 16422 == 59.996 Hz or 1022727 Hz / 17046 == 59.998 Hz. Note that both clock divisor values differ from the value used in the KERNAL R1.

The PAL/NTSC flag ($2A6) affects the RS-232 timer settings as well. It seems that the new RS-232 tables for the PAL have been created on the upper BASIC interpreter area ($E000--$E4FF), from the address $E4EC on. Surprisingly also the original NTSC tables have been changed. Very probably the units running the KERNAL R1 had a slower clock frequency. Extrapolating from the interrupt timer values, they ran at 1.0000 MHz. Now this makes sense, since the first (NTSC) video chips had 262 lines per frame and 64 cycles per line. The frame rate was thus 1 MHz / 262 / 64 == 59.637 Hz. The newer NTSC units run at 1022727 Hz and draw 263 lines per frame and use 65 cycles per line. This produces a frame rate of 59.826 Hz. Well, now it is very obvious that there has been at least one mother board type that has only been used on NTSC units. Probably the processor clock was created from a 8 MHz chrystal frequency, which served as the dot clock. The latter NTSC units generate the processor clock by dividing the chrystal frequency of 14318181 Hz by 14, and the dot clock will be generated by octacoupling the processor clock.

The PAL systems have been developed later, and they always run at the same clock frequency, 17734472 Hz / 18. The frame rate has always been 17734472 Hz / 312 / 63 == 50.125 Hz on those puppies.

The changes in the latter ROM revisions were mainly cosmetical. There were some bugs corrected in the R3 revision, though.

Format for list:

```
Address: 901227-01 (Commodore 64 KERNAL R1, $FF80 content $AA)
         901227-02 (Commodore 64 KERNAL R2, $FF80 content $00)
         901227-03 (Commodore 64 KERNAL R3, $FF80 content $03)
         ??????-?? (SX-64 or DX-64 KERNAL, $FF80 content $43)
         ??????-?? (4064 aka PET 64 aka Educator 64, $FF80 content $64)

E119:  C9, FF
       AD, E4
       AD, E4
       AD, E4
       AD, E4

E42D:  20, 1E, AB
       20, 1E, AB
       20, 1E, AB
       20, 1E, AB
       4C, 41, E4

E477:  20, 20, 2A, 2A, 2A, 2A, 20, 43, 4F, 4D, 4D, 4F, 44, 4F, 52, 45,
       20, 20, 2A, 2A, 2A, 2A, 20, 43, 4F, 4D, 4D, 4F, 44, 4F, 52, 45,
       20, 20, 2A, 2A, 2A, 2A, 20, 43, 4F, 4D, 4D, 4F, 44, 4F, 52, 45,
       20, 20, 20, 2A, 2A, 2A, 2A, 2A, 20, 20, 53, 58, 2D, 36, 34, 20,
       2A, 2A, 2A, 2A, 20, 43, 4F, 4D, 4D, 4F, 44, 4F, 52, 45, 20, 34,

-:     20, 36, 34, 20, 42, 41, 53, 49, 43, 20, 56, 32, 20, 2A, 2A, 2A,
       20, 36, 34, 20, 42, 41, 53, 49, 43, 20, 56, 32, 20, 2A, 2A, 2A,
       20, 36, 34, 20, 42, 41, 53, 49, 43, 20, 56, 32, 20, 2A, 2A, 2A,
       42, 41, 53, 49, 43, 20, 56, 32, 2E, 30, 20, 20, 2A, 2A, 2A, 2A,
       30, 36, 34, 20, 20, 42, 41, 53, 49, 43, 20, 56, 32, 2E, 30, 20,

-:     2A, 0D, 0D, 20, 36, 34, 4B, 20, 52, 41, 4D, 20, 53, 59, 53, 54,
       2A, 0D, 0D, 20, 36, 34, 4B, 20, 52, 41, 4D, 20, 53, 59, 53, 54,
       2A, 0D, 0D, 20, 36, 34, 4B, 20, 52, 41, 4D, 20, 53, 59, 53, 54,
       2A, 0D, 0D, 20, 36, 34, 4B, 20, 52, 41, 4D, 20, 53, 59, 53, 54,
       2A, 2A, 2A, 2A, 0D, 0D, 00, 20, 20, 20, 20, 20, 20, 20, 20, 20,

-:     45, 4D, 20, 20, 00, 2B
       45, 4D, 20, 20, 00, 5C
       45, 4D, 20, 20, 00, 81
       45, 4D, 20, 20, 00, B3
```

```
        20, 20, 20, 20, 20, 63

E4AD:   AA, AA, AA, AA, AA, AA, AA, AA, AA, AA
        48, 20, C9, FF, AA, 68, 90, 01, 8A, 60
        48, 20, C9, FF, AA, 68, 90, 01, 8A, 60
        48, 20, C9, FF, AA, 68, 90, 01, 8A, 60
        48, 20, C9, FF, AA, 68, 90, 01, 8A, 60

E4C8:   AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA,
        AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA,
        AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, 85, A9, A9, 01, 85,
        AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, 85, A9, A9, 01, 85,
        2C, 86, 02, 30, 0A, A9, 00, A2, 0E, 9D, 20, D0, CA, 10, FA, 4C,

-:      AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA,
        AA, AA, AD, 21, D0, 91, F3, 60, 69, 02, A4, 91, C8, D0, 04, C5,
        AB, 60, AD, 86, 02, 91, F3, 60, 69, 02, A4, 91, C8, D0, 04, C5,
        AB, 60, AD, 86, 02, 91, F3, 60, 69, 02, A4, 91, C8, D0, 04, C5,
        87, EA, AD, 21, D0, 91, F3, 60, 69, 02, A4, 91, C8, D0, 04, C5,

-:      AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA,
        A1, D0, F7, 60, 19, 26, 44, 19, 1A, 11, E8, 0D, 70, 0C, 06, 06,
        A1, D0, F7, 60, 19, 26, 44, 19, 1A, 11, E8, 0D, 70, 0C, 06, 06,
        A1, D0, F7, 60, 19, 26, 44, 19, 1A, 11, E8, 0D, 70, 0C, 06, 06,
        A1, D0, F7, 60, 19, 26, 44, 19, 1A, 11, E8, 0D, 70, 0C, 06, 06,

-:      AA, AA, AA, AA, AA, AA, AA, AA
        D1, 02, 37, 01, AE, 00, 69, 00
        D1, 02, 37, 01, AE, 00, 69, 00
        D1, 02, 37, 01, AE, 00, 69, 00
        D1, 02, 37, 01, AE, 00, 69, 00

E535:   0E
        0E
        0E
        06
        01

E57C:   B5, D9, 29, 03, 0D, 88, 02, 85, D2, BD, F0, EC, 85, D1, A9, 27,
        B5, D9, 29, 03, 0D, 88, 02, 85, D2, BD, F0, EC, 85, D1, A9, 27,
        20, F0, E9, A9, 27, E8, B4, D9, 30, 06, 18, 69, 28, E8, 10, F6,
        20, F0, E9, A9, 27, E8, B4, D9, 30, 06, 18, 69, 28, E8, 10, F6,
        20, F0, E9, A9, 27, E8, B4, D9, 30, 06, 18, 69, 28, E8, 10, F6,

-:      E8, B4, D9, 30, 06, 18, 69, 28, E8, 10, F6, 85, D5, 60
        E8, B4, D9, 30, 06, 18, 69, 28, E8, 10, F6, 85, D5, 60
        85, D5, 4C, 24, EA, E4, C9, F0, 03, 4C, ED, E6, 60, EA
        85, D5, 4C, 24, EA, E4, C9, F0, 03, 4C, ED, E6, 60, EA
        85, D5, 4C, 24, EA, E4, C9, F0, 03, 4C, ED, E6, 60, EA

E5EF:   09
        09
        09
        0F
```

```
       09

E5F4:  E6, EC
       E6, EC
       E6, EC
       D7, F0
       E6, EC

E622:  ED, E6
       ED, E6
       91, E5
       91, E5
       91, E5

EA07:  A9, 20, 91, D1, A9, 01, 91, F3, 88, 10, F5, 60
       A9, 20, 91, D1, 20, DA, E4, EA, 88, 10, F5, 60
       20, DA, E4, A9, 20, 91, D1, 88, 10, F6, 60, EA
       20, DA, E4, A9, 20, 91, D1, 88, 10, F6, 60, EA
       A9, 20, 91, D1, 20, DA, E4, EA, 88, 10, F5, 60

ECCA:  1B, 00
       9B, 37
       9B, 37
       9B, 37
       9B, 37

ECD2:  00
       0F
       0F
       0F
       0F

ECD9:  0E, 06, 01, 02, 03, 04, 00, 01, 02, 03, 04, 05, 06, 07
       0E, 06, 01, 02, 03, 04, 00, 01, 02, 03, 04, 05, 06, 07
       0E, 06, 01, 02, 03, 04, 00, 01, 02, 03, 04, 05, 06, 07
       03, 01, 01, 02, 03, 04, 00, 01, 02, 03, 04, 05, 06, 07
       00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00

EF94:  85, A9, 60
       85, A9, 60
       4C, D3, E4
       4C, D3, E4
       85, A9, 60

F0D8:  0D, 50, 52, 45, 53, 53, 20, 50, 4C, 41, 59, 20, 4F, 4E, 20
       0D, 50, 52, 45, 53, 53, 20, 50, 4C ,41, 59, 20, 4F, 4E, 20
       0D, 50, 52, 45, 53, 53, 20, 50, 4C ,41, 59, 20, 4F, 4E, 20
       4C, 4F, 41, 44, 22, 3A, 2A, 22, 2C, 38, 0D, 52, 55, 4E, 0D
       0D, 50, 52, 45, 53, 53, 20, 50, 4C ,41, 59, 20, 4F, 4E, 20

F387:  03
       03
       03
       08
```

```
        03

F428:   D0, 0B, AD, 95, 02, 0A, A8, AD, 96, 02, 4C, 3F, F4, 0A, AA, BD,
        F0, 1C, 0A, AA, AD, A6, 02, D0, 09, BC, C1, FE, BD, C0, FE, 4C,
        F0, 1C, 0A, AA, AD, A6, 02, D0, 09, BC, C1, FE, BD, C0, FE, 4C,
        F0, 1C, 0A, AA, AD, A6, 02, D0, 09, BC, C1, FE, BD, C0, FE, 4C,
        F0, 1C, 0A, AA, AD, A6, 02, D0, 09, BC, C1, FE, BD, C0, FE, 4C,

-:      C0, FE, 0A, A8, BD, C1, FE, 2A, 48, 98, 69, C8, 8D, 99, 02, 68,
        40, F4, BC, EB, E4, BD, EA, E4, 8C, 96, 02, 8D, 95, 02, AD, 95,
        40, F4, BC, EB, E4, BD, EA, E4, 8C, 96, 02, 8D, 95, 02, AD, 95,
        40, F4, BC, EB, E4, BD, EA, E4, 8C, 96, 02, 8D, 95, 02, AD, 95,
        40, F4, BC, EB, E4, BD, EA, E4, 8C, 96, 02, 8D, 95, 02, AD, 95,

-:      69, 00, 8D, 9A, 02
        02, 0A, 20, 2E, FF
        02, 0A, 20, 2E, FF
        02, 0A, 20, 2E, FF
        02, 0A, 20, 2E, FF

F459:   4C
        20
        20
        20
        20

F4B7:   7B
        7B
        7B
        F7
        7B

F5F9:   5F
        5F
        5F
        F7
        5F

F762:   91, C9, FF, F0, FA
        A1, 20, E0, E4, EA
        A1, 20, E0, E4, EA
        A1, 20, E0, E4, EA
        A1, 20, E0, E4, EA

F81F:   2F
        2F
        2F
        2F
        2B

F82C:   2F
        2F
        2F
        2F
```

```
        2B

FCFC:   18, E5
        5B, FF
        5B, FF
        5B, FF
        5B, FF

FDDD:   A9, 1B, 8D, 04, DC, A9, 41, 8D, 05, DC, A9, 81, 8D, 0D, DC, AD,
        AD, A6, 02, F0, 0A, A9, 25, 8D, 04, DC, A9, 40, 4C, F3, FD, A9,
        AD, A6, 02, F0, 0A, A9, 25, 8D, 04, DC, A9, 40, 4C, F3, FD, A9,
        AD, A6, 02, F0, 0A, A9, 25, 8D, 04, DC, A9, 40, 4C, F3, FD, A9,
        AD, A6, 02, F0, 0A, A9, 25, 8D, 04, DC, A9, 40, 4C, F3, FD, A9,

-:      0E, DC, 29, 80, 09, 11, 8D, 0E, DC, 4C, 8E, EE
        95, 8D, 04, DC, A9, 42, 8D, 05, DC, 4C, 6E, FF
        95, 8D, 04, DC, A9, 42, 8D, 05, DC, 4C, 6E, FF
        95, 8D, 04, DC, A9, 42, 8D, 05, DC, 4C, 6E, FF
        95, 8D, 04, DC, A9, 42, 8D, 05, DC, 4C, 6E, FF

FEC2:   AC, 26, A7, 19, 5D, 11, 1F, 0E, A1, 0C, 1F, 06, DD, 02, 3D, 01,
        C1, 27, 3E, 1A, C5, 11, 74, 0E, ED, 0C, 45, 06, F0, 02, 46, 01,
        C1, 27, 3E, 1A, C5, 11, 74, 0E, ED, 0C, 45, 06, F0, 02, 46, 01,
        C1, 27, 3E, 1A, C5, 11, 74, 0E, ED, 0C, 45, 06, F0, 02, 46, 01,
        C1, 27, 3E, 1A, C5, 11, 74, 0E, ED, 0C, 45, 06, F0, 02, 46, 01,

-:      B2, 00, 6C
        B8, 00, 71
        B8, 00, 71
        B8, 00, 71
        B8, 00, 71

FF08:   93, 02, 29, 0F, D0, 0C, AD, 95, 02, 8D, 06, DD, AD, 96, 02, 4C,
        95, 02, 8D, 06, DD, AD, 96, 02, 8D, 07, DD, A9, 11, 8D, 0F, DD,
        95, 02, 8D, 06, DD, AD, 96, 02, 8D, 07, DD, A9, 11, 8D, 0F, DD,
        95, 02, 8D, 06, DD, AD, 96, 02, 8D, 07, DD, A9, 11, 8D, 0F, DD,
        95, 02, 8D, 06, DD, AD, 96, 02, 8D, 07, DD, A9, 11, 8D, 0F, DD,

-:      25, FF, 0A, AA, BD, C0, FE, 8D, 06, DD, BD, C1, FE, 8D, 07, DD,
        A9, 12, 4D, A1, 02, 8D, A1, 02, A9, FF, 8D, 06, DD, 8D, 07, DD,
        A9, 12, 4D, A1, 02, 8D, A1, 02, A9, FF, 8D, 06, DD, 8D, 07, DD,
        A9, 12, 4D, A1, 02, 8D, A1, 02, A9, FF, 8D, 06, DD, 8D, 07, DD,
        A9, 12, 4D, A1, 02, 8D, A1, 02, A9, FF, 8D, 06, DD, 8D, 07, DD,

-:      A9, 11, 8D, 0F, DD, A9, 12, 4D, A1, 02, 8D, A1, 02, A9, FF, 8D,
        AE, 98, 02, 86, A8, 60, AA, AD, 96, 02, 2A, A8, 8A, 69, C8, 8D,
        AE, 98, 02, 86, A8, 60, AA, AD, 96, 02, 2A, A8, 8A, 69, C8, 8D,
        AE, 98, 02, 86, A8, 60, AA, AD, 96, 02, 2A, A8, 8A, 69, C8, 8D,
        AE, 98, 02, 86, A8, 60, AA, AD, 96, 02, 2A, A8, 8A, 69, C8, 8D,

-:      06, DD, 8D, 07, DD, AE, 98, 02, 86, A8, 60
        99, 02, 98, 69, 00, 8D, 9A, 02, 60, EA, EA
        99, 02, 98, 69, 00, 8D, 9A, 02, 60, EA, EA
        99, 02, 98, 69, 00, 8D, 9A, 02, 60, EA, EA
```

```
        99, 02, 98, 69, 00, 8D, 9A, 02, 60, EA, EA

  FF5B: AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA,
        20, 18, E5, AD, 12, D0, D0, FB, AD, 19, D0, 29, 01, 8D, A6, 02,
        20, 18, E5, AD, 12, D0, D0, FB, AD, 19, D0, 29, 01, 8D, A6, 02,
        20, 18, E5, AD, 12, D0, D0, FB, AD, 19, D0, 29, 01, 8D, A6, 02,
        20, 18, E5, AD, 12, D0, D0, FB, AD, 19, D0, 29, 01, 8D, A6, 02,

  -:    AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA, AA,
        4C, DD, FD, A9, 81, 8D, 0D, DC, AD, 0E, DC, 29, 80, 09, 11, 8D,
        4C, DD, FD, A9, 81, 8D, 0D, DC, AD, 0E, DC, 29, 80, 09, 11, 8D,
        4C, DD, FD, A9, 81, 8D, 0D, DC, AD, 0E, DC, 29, 80, 09, 11, 8D,
        4C, DD, FD, A9, 81, 8D, 0D, DC, AD, 0E, DC, 29, 80, 09, 11, 8D,

  -:    AA, AA, AA, AA, AA
        0E, DC, 4C, 8E, EE
        0E, DC, 4C, 8E, EE
        0E, DC, 4C, 8E, EE
        0E, DC, 4C, 8E, EE

  FF80: AA
        00
        03
        43
        64

  FF82: 18, E5
        53, FF
        53, FF
        53, FF
        53, FF

  FFF8: 42, 59
        42, 59
        42, 59
        42, 59
        00, 00
```

APPENDIX J

CHIP INFORMATION CHART

IC'S

LOCATION	IC NUMBER	DESCRIPTION
U1	6526 CIA #1	COMPLEX INTERFACE ADAPTER
U2	6526 CIA #2	"
U3	901226-01	NMOS 8192X8 STATIC BASIC ROM
U4	901227-XX	NMOS 8192X8 STATIC KERNAL ROM
U5	901225-01	NMOS 4096X8 STATIC CHARACTER ROM
U6	2114-30L/MCM2114P20	NMOS 1024X8 STATIC RAM
U7	6510	NMOS MPU (CPU)

```
U8          7406N/M53206P          QUAD OPERATIONAL AMPLIFIER
U9          4164-2/MK4564N-20NMOS 65536X1-BIT DYNAMIC RAM
U10         4164-2/MK4564N-20NMOS 65536X1-BIT DYNAMIC RAM
U11         4164-2/MK4564N-20NMOS 65536X1-BIT DYNAMIC RAM
U12         4164-2/MK4564N-20NMOS 65536X1-BIT DYNAMIC RAM
U13         74LS257                QUAD 2-INPUT TRI-STATE MULTIPLEXER
U14         74LS258                TTL DIGITAL MULTIPLEXER
U15         74LS139                DUAL 2/4 DECODER DEMULTIPLEXER
U16         4066             CMOS QUAD ANALOG SWITCH
U17         82S100                 FIELD PROGRAMMABLE PLA
U18         6581 SID         SOUND INTERFACE DEVICE
U19         6567 VIC         VIDEO INTERFACE CHIP
U20         556/MC3456       DUAL 555 TIMER
U21         4164-2 RAM       NMOS 65536X1-BIT DYNAMIC RAM
U22         4164-2 RAM       NMOS 65536X1-BIT DYNAMIC RAM
U23         4164-2 RAM       NMOS 65536X1-BIT DYNAMIC RAM
U24         4164-2 RAM       NMOS 65536X1-BIT DYNAMIC RAM
U25         74LS257                QUAD 2-INPUT TRI-STATE MULTIPLEXER
U26         74LS373                8-BIT TRANSPARENT LATCH
U27         75LS08                 QUAD 2-INPUT AND
U28         4066             CMOS ANALOG SWITCH
U29         74LS74                 QUAD D FLIP-FLOP
U30         74LS193                BINARY UP/DOWN COUNTER
U31         74LS629N         DUAL VOLTAGE CONTROLLER OSCILLATOR
U32         MC4044                 TTL PHASE FREQUENCY DETECTOR

OTHER COMPONENTS:

LOCATION    DEVICE                 DESCRIPTION
--------    ------                 -----------
CR1         1N4371                 2.7-VOLT ZENER DIODE
CR2         1N755            7.5-VOLT ZENER DIODE
CR3         1N914            SIGNAL DIODE
CR4         VM08 (P/S)       BRIDGE RECTIFIER DIODE
CR5         1N4001 (P/S)           POWER DIODE
CR6         1N4001 (P/S)           POWER DIODE
Q1          2N4401                 TRANSISTOR
Q2          2N3904                 "
Q3          TP29B            "
Q4          PN2222                 "
Q5          PN2222                 "
Q6          PN2222                 "
Q7          PN2222A                "
Q8          PN2222                 "
VR1         MD7812CT/UA7812UCFIXED POSITIVE LINEAR VOLTAGE REG.
VR2         MC7805CT         " WITH 1500 mA OUTPUT


APPENDIX K
----------
SPECIFICATIONS OF THE COMMODORE 64

MANUFACTURER:          COMMODORE BUSINESS SYSTEMS
                 1200 WILSON DRIVE
```

WEST CHESTER, PA 19380

SIZE: 2.75"X15.9"X8.0"

WEIGHT: 4.1 LBS.

POWER REQUIRED: LESS THAN 20 WATTS 8.5 WATTS AT 5.V DC

MPU: COMMODORE 6510 MPU

DATA WORD SIZE: 8-BITS

CPU CLOCK SPEED: 1.023 MHz

MEMORY SIZE: 64K

MASS STORAGE CAPABILITY:
UP TO 4 VIC-1541 DISK DRIVES
DATA CASSETTE RECORDER

KEYBOARD SIZE: 65 KEYS
157 CHARACTER CODES

TEXT DISPLAY: 40 UPPERCASE CHARACTERS (2-CHAR SETS)
24 LINES

GRAPHICS CAPABILITY: LOW RES - 160 X 200 PIXELS
HIGH RES - 320 X 200 PIXELS
USER DEFINED SPRITE GRAPHICS

COLOR CAPABILITY: 16 COLORS

INPUT/OUTPUT: CASSETTE I/O
2-CONTROL PORTS FOR GAME PADDLES
CARTRIDGE EXPANSION SLOT
24-PIN USER I/O PORT
6-PIN SERIAL I/O CONNECTION
RF MODULATOR OUTPUT FOR TV DISPLAY
NTSC COMPOSITE COLOR OUTPUT FOR MONITOR

BIBLIOGRAPHY:
----

1. "Beyond Games: Systems Software for Your 6502 Personal Computer" by Ken Skier 1981. This book was intended for the C= PET 2001 Computer.

2. "Machine Language for Beginners" by Richard Mansfield, 1983. This book was intended for the Atari, VIC, Apple, Commodore 64, and PET/CBM computers.

3. "Assembly Language Programming with the Commodore 64" by Marvin L. De Jong, 1984.

4. "Commodore 64 Troubleshooting & Repair Guide" by Robert C. Brenner, 1985.

5. "The Commodore 64 Programmer's Reference Guide" by CBM, 19xx.

6. "The Commodore 64 User's Guide" by CBM, 19xx.

7. "CHACKING MAG" (C) 1992 by Craig Taylor

8. "The PC Assembler Tutor" (C) 1989 by Chuck Nelson.