The Ultimate C64 Overview

Michael Steil, http://www.pagetable.com/
25th Chaos Communication Congress 2008

Retrocomputing is cool as never before. People play C64 games in emulators and listen to SID music, but few people know much about the C64 architecture and its limitations, and what programming was like back then. This paper attempts to give a comprehensive overview of the Commodore 64, including its internals and quirks, making the point that classic computer systems aren't all that hard to understand - and that programmers today should be more aware of the art that programming once used to be.

Commodore History

Commodore Business Machines was founded in 1962 by Jack Tramiel. The company specialized on electronic calculators, and in 1976, Commodore bought the chip manufacturer MOS Technology and decided to have Chuck Peddle from MOS evolve their KIM-1 computer kit (a design that demos their new MOS 6502 8 bit CPU) into the Commodore PET series: computers with built-in monitors for the home, school and small business market that ended up competing with devices from Atari and Apple.

In 1981, Commodore introduced the VIC-20, a 5 KB stripped down monitorless computer-in-the-keyboard design based on the PET for the home computer market. This was followed by the (incompatible) higher-end Commodore 64 in 1982 that included more PET features, came with 64 KB of RAM (an immense amount compared to the rest of the market) and was very aggressively priced at US$595 beating the competition by a factor of two. This was made possible by designing and building most of the system in-house.

Although some features of the C64 were taken from the PET models of the time, it had to be connected to a TV set, which only made 40 columns of text possible (as opposed to 80 columns on the PET). Also, the BASIC 4.0 codebase was stripped down to the old 2.0 feature set to make it fit into 8 KB.

In the beginning, the C64 did well in the competition. The superior but compatible C128 from 1985 did well, too, but was never more popular than the C64, which continued to be sold. The direct successor to the low-end VIC-20, the 1984 Plus/4 and its siblings, the C16 and the C116, failed, mostly because they were incompatible with the C64, which at that time already had a remarkable software library.

A few years after the introduction, the C64 was still offered as a low-end alternative to the Commodore Amiga, and while it became less popular in the USA, it gained more and more popularity in Europe. In the early 90s, when manufacturing costs of a C64 were as low as $25, it gained a second life in Eastern Europe. Production did not end until the liquidation of Commodore in 1994. According to the 1993 Annual Report, 17 million C64 had been produced in by then, as well as 4.5 million C128.

Look and Feel

A C64 only needs to be connected to power and a TV set (or monitor) to be fully functional. When turned on, it shows a blue-on-blue theme with a startup message and drops into a BASIC interpreter derived from Microsoft BASIC. In order to load and save BASIC programs or use third party software, the C64 requires mass storage - either a “datasette” cassette tape drive or a disk drive like the 5.25" Commodore 1541.

Unless the user really wanted to interact with the BASIC interpreter, he would typically only use the BASIC instructions LOAD, LIST and RUN in order to access mass storage. LOAD"$",8 followed by LIST shows the directory of the disk in the drive, and LOAD"filename",8 followed by RUN would load and start a program. If a tape drive is connected, LOAD and RUN will launch the first program on tape - pressing SHIFT and the RUN/STOP key has the same effect.

By default, typing characters without SHIFT will result in upper case characters being shown on the screen. This can be changed by pressing the Commodore key and SHIFT at the same time, which switches to the upper/lower character set. This behavior is due to the fact that the first PET only had uppercase characters, and that BASIC required keywords unshifted - but all tutorials taught them as uppercase. This is also the reason why in Commodore's version of ASCII, called PETSCII, the codes for uppercase and lowercase characters are reversed.

The text based user interface is not a line editor, but a screen editor, i.e. the cursor can be moved freely on the screen, and pressing RETURN on a line with existing text will present the text to the application as if it were just typed in.

While a physical screen line is only 40 characters, the screen editor logic can extend it to a logical 80 characters - whenever a character is entered on the 40th column, the rest of the screen is moved down by one line, ”opening” a line that extends the previous line to 80 characters.

The C64 screen editor supports selecting one out of 16 color for the foreground text by pressing the key combinations Ctrl+1 to Ctrl+8 and Commodore+1 to Commodore+8. Reverse text mode can be turned on and off with Ctrl+9 and Ctrl+0.

Ports and Connections

The C64 has a whole range of connection possibilities. On the side, it has two 9 pin Atari-style joystick connectors that can also be used for a mouse, light pens or paddles. On the back, there is the expansion port, which exports the complete processor bus, allowing not only game cartridges but also cartridges with I/O chips that map themselves into the CPU's address space - or even cartridges that completely replace the CPU. For a conventional TV connection, there is an

RCA connector that outputs an RF signal. For monitors, there is an extra DIN connector that carries separate chroma, luma and audio signals (S-Video).

For connecting Commodore compatible printers and disk drives, there is a DIN connector for the IEC bus. There is also a dedicated connector for datasette drives. The ”User Port” consists of several GPIO pins that can be used for custom hardware projects, or as a RS-232 port (with TTL levels), for which support exists in the ROM.

Board

On the C64 motherboard, there is a dedicated IC each for the main tasks. There is the MOS 6510 CPU, eight 64 KBit RAM chips (later consolidated into 2), three ROM chips with KERNAL (I/O library), BASIC and the character set (KERNAL and BASIC were later consolidated), two 6526 CIA I/O controllers (one for keyboard and joystick, one for the IEC bus and the user port), the 6581 SID sound chip, and the 6567/6569 VIC video chip, as well as the RAM chip that holds the 512 bytes of Color RAM.

All non-RAM chips are custom chips designed manufactured by MOS, Commodore's inhouse chip company.

Address Space

The 8 bit C64 design has a 16 bit address bus, allowing the CPU to address 64 KB of memory. Since the C64 has 64 KB of RAM, filling the complete address space, ROM and I/O chips are mapped into regions of the address space that are shared with RAM: The CPU can switch these regions between RAM and a second or third mapping. These regions are as follows:

- $0000-$9FFF: RAM
- $A000-$BFFF: RAM or BASIC ROM
- $C000-$CFFF: RAM
- $D000-$DFFF: RAM or memory mapped I/O chips or character ROM
- $E000-$FFFF: RAM or KERNAL ROM

In contrast to CPUs like the Z80 and the 8086, and like most modern CPUs, I/O devices are memory mapped on the C64's 6510 CPU. The mapping is as follows:

- $D000-$D3FF: VIC video controller
- $D400-$D7FF: SID sound controller
- $D800-$DBFF: Color RAM
- $DC00-$DCFF: CIA 1 I/O controller
- $DD00-$DDFF: CIA 2 I/O controller
- $DE00-$DFFF: for extensions on the expansion port

6502 CPU

The CPU inside the C64 is a 0.985 MHz (on PAL) MOS 6510, which is a close derivative of the well-known 8 bit little-endian MOS 6502. The 6502 was introduced in 1975 by MOS Technology, a company formed earlier the same year by former Motorola engineers, headed by Chuck Peddle. The philosophy of the 6502 was to have a reduced instruction set and a small register file, making it simpler and faster than CPUs like the Z80 at the same clock speed, as well as cheaper to manufacture.

Unlike other CPUs, the 6502 does not have a set of general purpose registers. Instead, it has a single accumulator A (for arithmetic and logic), two index registers X and Y (for incrementing, decrementing and indexing memory) and a stack pointer. All these registers are 8 bits. The processor status, consisting of the negative (N), overflow (V), break (B), decimal (D), interrupt (I), zero (Z) and carry (C) flags is exposed as register P. The program counter (PC) is 16 bits wide. The fact that the stack pointer is 8 bit means that the stack is confined to the area between $0100 and $01FF in the address space, i.e. the upper half of the effective stack pointer is hard-coded to $01. There is another special area in the address space: The first 256 bytes, at $0000 to $00FF are referred to as the zero page (ZP). Many instructions support special encodings for zero page addresses, which saves one byte in the instruction encoding as well as at least one cycle of execution time. This can be seen as an extension of the register file to another 256 (though external) registers.

All opcodes are one byte, and have 0, 1 or 2 byte operands. The 8x8 opcode matrix is somewhat logical (e.g. branch instructions are encoded as $10, $30, $50, ...), but there is no easy rule to construct the opcode table. Nevertheless, the opcode table is a minimal encoding for optimal decoding in the 6502's internal PLA ROM.

Instruction Set

The instruction set is very streamlined, and avoids redundancies. There are load instructions (LDA/LDX/LDY to load A, X and Y respecively), store instructions (STA/STX/STY), read-modify-write instructions (logic: ASL/LSR/ROL/ROR, count: INC/DEC), arithmetic (ADC/SBC; note that these always include the carry: CLC/ADC is a regular addition, and SEC/SBC is a regular subtraction, because of the one's complement logic), compare (CMP/CPX/CPY; these are subtractions without storing the result), logic (AND/ORA/EOR, and BIT, which is AND without storing the result), as well as branch instructions, flag manipulation, register transfer and stack manipulation.

Addressing Modes

Each instruction supports one or more addressing modes. Common instructions like LDA (load accumulator) support more addressing modes than less common ones (BIT).

- The immediate addressing mode is indicated with a # sign: LDA #$17 loads the immediate value of $17 into the accumulator.
- Absolute addressing specifies a 16 bit address as an operand: LDA $0314 loads from the memory address $0314.
- Zero page addressing is an optimized version of absolute addressing: LDA $02 will read from ad-

dress $0002 in memory, but the instruction can be encoded more tightly, and execution is faster.

- Absolute-X-indexed addressing reads from a specified address, to which the contents of the X register is added. LDA $0200,X reads from the address $020A, in case X is $0A. This allows reading from tables.
- Absolute-Y-indexed is the same thing, but with the Y register.
- Zero-Page-X-indexed is an optimized version of Absolute-X-indexed. LDA $F0,X reads from the Xth location in a table stored starting at $00F0 in memory. Note that zero page addresses will wrap around, so $F0 + $10 = $00.
- Zero-Page-Y-indexed is the same thing, but with the Y register.
- Zero-Page-X-indexed-indirect adds X to a specified zero page address, reads a 16 bit pointer from the resulting address and finally accesses memory at that address. So LDA ($80,X) will read from an address specified by the array of pointers at $0080 and the index X into the array. This addressing mode is rarely used.
- Zero-Page-indirect-Y-indexed treats two consecutive bytes in zero page as an address and adds Y to the address. LDA ($14),Y will read from $E020, if the address stored at $14/$15 is $E000 and Y is $20. This addressing mode is the most convenient way to work with pointers, as no register can hold 16 bits.

Register Transfer and Stack

There are several 1 byte instructions without operands that move data between registers. TAX, TXA, TAY and TYA move between A, X and Y. TSX and TXS copies between X and the stack pointer.

The stack pointer always points to the next address that is written to. This means that an empty stack has a stack pointer of $FF, and pushing a value first writes the value and then decrements the stack pointer. The 6502 can move the accumulator from and to the stack (PHA/PLA), as well as the processor status P (PHP/PLP).

Control Transfer

Next to the absolute JMP instruction, there is an indirect version that jumps over a vector (e.g. JMP ($FFFC)). JSR (jump to subroutine) only has an absolute version, and stores the address of the next instruction minus one on the stack. RTS (return from subroutine) takes the address from the stack, adds one, and moves it into the program counter. The ”minus one” logic was chosen because it could save one cycle in the implementation of JSR.

A hardware interrupt, unless disabled by a set interrupt (I) flag, pushes the address of the next instruction minus one (just like JSR), pushes the processor status afterwards, disables interrupts, and jumps over the vector at $FFFE/$FFFF. RTI return from interrupt) is the same as the combination of PLP and RTS. BRK causes a software interrupt and behaves the same as a hardware interrupt, except that it sets the B flag on the stack to 1 (a hardware interrupt sets it to 0).

NMIs behave the same as IRQs, but they cannot be masked, and they use the $FFFA/$FFFB vector. The reset vector is at $FFFC/$FFFD.

Flags and Branches

All load and logic instructions set N and Z accordingly, shift instructions also modify C, and arithmetic instructions touch N V, Z and C. The D (decimal), I (interrupt disable) and C flags can be set and cleared programmatically (CLD/SED, CLI/SEI, CLC/SEC), while the V flag can only be cleared (CLV). Conditional branches are possible based on the value of the negative (BPL/BMI), overflow (BVC/BVS), zero (BNE/BEQ) and carry (BCC/BCS) flags. Branches encode an 8 bit relative offset and can therefore reach code in the area of +127 and -128. Since a compare is the same as a subtraction, BCC is a branch on (unsigned) below, and BCS is a branch on above-or-equal.

NOP

NOP (no operation) does nothing. Its encoding is $EA.

Decimal Mode

If the D flag is set, all ADC and SBC operations will be BCD-adjusted afterwards, i.e. $09+$02 won't be $0B, but $11, since 9+2=11. The BCD correction circuit has been patented in US patent 3,991,307.

Cycle Counting

It is quite straightforward to find out how many cycles an instruction takes. As a rule of thumb, an instruction takes as many cycles as the number of memory fetches it has to perform, but at least two.

Therefore, single-byte opcodes (one byte fetches; NOP/TAX/INX etc.) as well as instructions with immediate operands take two cycles. Zero page instructions take three memory accesses (opcode, address, data), so they are three cycles. Absolute instructions take four accesses (opcode, address low, address high, data), so they are four cycles.

Read-modify-write instructions (INC/DEC/shift/rotate) are an exception: They require 4 memory accesses for the zero page case and 5 otherwise, but they take 5 and 6 cycles, respectively.

Branches take 3 cycles if they are taken and two if they are not taken. And extra cycle has to be added if the branch crosses a page boundary. JMP is 3, push is 3, pull is 4, JSR and RTS are 6 each.

All other timings can be looked up in the 6502's reference, but they are very easy to memorize.

Common Tricks

- The BIT instruction exists in a two-byte (immediate operand) and three-byte (absolute operand) variant. Since BIT only changes the flags, it effectively skips one or two bytes in the instruction stream. This can be used to replace a two-byte branch or a three-byte JMP with a one-byte BIT if only one or two bytes have to be skipped.
- The architecture allows safe self-modifying code, so a common optimization for copy loops is to use LDA $nn00,X and STA $mm00,X, looping X from $00 to $FF and then incrementing the bytes that encode nn and mm for the next page. Compared to

a LDA (zp1),Y: STA (zp2),Y sequence, this gets the inner loop down from 16 cycles (5 LDA, 6 STA, 2 INY, 3 BNE) to 14 (4 LDA, 5 STA, 2 INY, 3 BNE).

- A PHA/PLA combination is 7 cycles, while an STA/LDA combination in the zero page is 6 cycles, so unless there is no free zero page space, PHA/PLA should be avoided to quickly store a value. Using an absolute store to write the value into the operand of a future immediate load (i.e. self modification) is the same speed at the zero page solution, but does not waste zero page space.
- An elegant way to store a flag is to have it in bit #7 of a zero page address. While a load/store combination has to be used to set the flag, it can be cleared with a simple LSR (5 cycles) and tested with BIT (3 cycles), without affecting register contents.
- Since the 6502 is so register starved, only 3 bytes can be passed to a subroutine in registers. Also, the stack is small, and accessing it is slow, so stack frames as seen on modern architectures are very uncommon. Many applications and libraries (e.g. GEOS) use a dedicated area in the zero page as virtual registers.
- The 6502 has no instructions for multiplication, division or floating point arithmetic. Most 6502-based computers have a BASIC interpreter in ROM though, and they typically include a math and floating point library.

Bugs and Quirks

The original 6502 implementation has a series of bugs and other anomalies that have never been fixed in MOS chips (not counting the 65CE02, which was only used in Amiga peripherals).

- The indirect version of JMP loads the program counter from the wrong address, if the vector's address lies on a page boundary: JMP ($23FF) will read the address from $23FF and $2300 instead of $23FF and $2400.
- When in decimal mode, the negative flag reflects the original binary result, not the effective decimal result.
- If a software interrupt (BRK) and a hardware interrupt occur at the same time, the BRK is dropped.
- Read instructions (LDA/AND etc.) with the absolute-indexed addressing mode first read from the absolute address without the index register added, and then read again from the correct address. LDX #$07 LDA $D019,X will first read from $D019, discard the result, and then read from $D020. On the C64, this read form $D019 would ACK all pending VIC interrupts, while it is only supposed to read the border color ($D020).
- Read-modify-write instructions with absolute addresses first read the value, but one cycle before they store the result, they store the original value again. On a C64, this can be seen when incrementing the screen border color at a defined area of the screen, as every write to the register will cause a tiny gray dot on the screen. When this is used with certain I/O ports, this can have other side effects. The latter two quirks have been used heavily for obfuscating copy protection software.
- Instruction decoding in the 6502 is done by a PLA that compares the current cycle number within the instruction and the current opcode against a ROM of 130 mask lines, of which any number can fire independently. The outputs of these lines are then fed into components like the ALU, bus control, register control and program counter logic. The instruction set only consists of 151 defined opcodes, and since handling the remaining 105 opcodes as NOPs or traps would have required extra lines in the PLA, they will match against some lines that were meant for instructions with similar opcodes. Some of these ”illegal opcodes” lead to useful results and are used in some software (SAX = store A & X), but most of the instructions make little sense (SHX = store X & the upper 8 bits of the program counter), and some even lock up the CPU, disabling IRQs and NMIs (CRA/KIL).

The MOS 6510

Except for the pin layout, the MOS 6510 that is used in the C64 differs from the generic MOS 6502 in two ways: It can make the bus tri-state when not used, so the VIC can use it, and it has a 6 bit I/O port built in, which can be controlled using zero page locations 0 and 1. In register 0, each bit from 0-5 set it to output if 1, and to input if 0. Bits 0-5 in register 1 are the actual I/O pins. On the C64, bits 0-2 are outputs and control bank switching, they turn the ROMs and the I/O area on and off. Bits 3-5 go to the tape connector and control the motor and the data sent to the head, and detect whether a key on the tape deck is pressed.

BASIC

Microsoft had a strong position in the market for (mostly ROM) BASIC interpreters in 8080-based home computers when the MOS 6502 was released in 1975, so they rewrote their interpreter in 6502 assembly. Microsoft BASIC was pure 6502 code with a minimal character I/O interface to the machine's ”monitor”, i.e. I/O library.

Commodore decided to license the interpreter for the 1977 PET and extended it slightly to interface with their disk and tape libraries. Commodore BASIC was very buggy, so they went back to Microsoft for an update, which, with the Commodore changes re-applied, shipped in newer PETs as BASIC V2. For version 4, Commodore added several extensions, both language constructs as well as support for graphics and sound.

Being a low-end machine, Commodore took the bug-fixed BASIC V4 codebase and removed all features after V2, making it independent of the machine's graphics and sound features again, and fitting it back into 8 KB, and shipped this version on the VIC-20. The Commodore 64 got the exact same version, except that it runs at a different memory address ($A000-$BFFF).

Microsoft BASIC is a line-based editor, that is, lines can be shown with the LIST command, and they can be modified by re-typing them. This integrates nicely with the KERNAL screen editor: The cursor can be moved up to LISTed lines, the lines can be modified, and when RETURN is pressed, the whole line is fed into BASIC again.

A nice feature of this and later versions of Commodore BASIC is the fact that all important parts, like the tokenizer, the detokenizer and the interpreter loop jump over a jump table in RAM before they do their work, allowing the user to extend BASIC arbitrarily. The most well-known BASIC extension is Simons' BASIC, a cartridge that maps 8 KB of extra ROM at $8000-$9FFF.

KERNAL

The C64 has an 8 KB I/O library at $E000-$FFFF which is utilized by BASIC, but is intended to be used by other applications as well. All Commodore 8 bit systems have a standardized library call interface in the form of jump tables at the very top of memory that call into machine-specific functions for I/O.

KERNAL is started form the RESET vector, initializes the machine, sets up an interrupt service routine that handles the keyboard, animates the cursor and does the real time clock. The C64 has a hardware clock in each of the CIA chips, but KERNAL has not been updated to use this feature since the VIC-20.

KERNAL provides an abstract character I/O interface to a number of devices. All devices support open, read, write and close. The open call takes three parameters: The logical file number (there is a maximum of 16 channels), the device number and the secondary address. There are 16 device numbers statically assigned to the devices. The 8 bit secondary address can signal something to the device, like speed or an operation mode. Some devices (tape and IEC) support an optional filename.

Device 0 is the keyboard. While KERNAL exports raw key presses, the keyboard can also be accessed through character I/O, which will go through the screen editor and replay all characters on the screen that are in the line of the cursor, regardless of whether the user typed them or they had been there before.

Device 1 is the tape drive. KERNAL reads and writes blocks of data at a time and buffers them for character I/O.

Device 2 is RS-232. KERNAL contains a very sophisticated (but rarely used) software RS-232 implementation that supports up to 2400 baud.

Device 3 is the screen. KERNAL interprets special codes, manages the cursor position and handles scrolling.

Devices 4 to 15 will be directed to the IEC bus. By convention, devices 4 to 7 are printers and plotters, and devices 8 to 15 are floppy or hard disks.

KERNAL allows interacting with the IEC bus manually by sending TALK and LISTEN requests to the bus.

GEOS

While KERNAL is a minimal character-based operating system in ROM, there is also a disk-based operating system with a graphical user interface for the C64. GEOS was released by Berkeley Softworks in 1986 and Commodore bundled it with the C64 for some time. The GUI, which can be controlled by a joystick or a mouse, runs in 320x200 graphics mode and resembles early versions of MacOS. GEOS is a 16 KB library that includes an optimized disk interface (faster, support for timstamps, icons and multi-fork ”VLIR” files), library code for drawing to the screen, high level UI primitives for menus, buttons and dialog boxes (with callbacks) and a simple memory swapping facility. Furthermore GEOS allows input and printer driver plugins, as well as proportional fonts in different sizes. Internally, GEOS has a jump table to its library routines that consists of about 150 entries.

GEOS came with applications like GeoPaint and GeoWrite; Berkeley Softworks themselves offered solutions like GeoPublish and GeoCalc, and more software was available from third parties.

GEOS' only requirement is a 1541 disk drive, but a 3.5" 1581 drive, a RAM extension or one of the later hard drives helped speed it up a lot.

6526 CIA

The C64 has two identical 6526 CIAs (Complex Interface Adapter) that are mostly used for I/O. One CIA features 16 general purpose I/O pins (8 bit port A and 8 bit port B) that can be used either as an input or an output, two programmable timers and a real-time-clock.

The timers have 16 bit counters and count down by one either on each clock cycle, or on an external event, or on a timer A underflow (in the case of timer B). This allows concatenating the timers to one 32 bit timer. On an underflow, the CIA can be programmed to cause an IRQ or to send data through a serial shift register. The CIA also supports receiving data through a shift register.

The real-time-clock has a resolution of 1/10 of seconds and supports generating interrupts at a certain time.

CIA 1 is hooked up to the keyboard and the joystick ports. Since the keyboard consists of 64 keys (plus SHIFT LOCK, which is parallel to the left SHIFT key, and RESTORE, which is directly connected to the CPU's NMI line), these can be laid out in a 8x8 matrix of lines, key presses connecting the intersections. One side of the matrix is connected to port A (output), and the perpendicular side is connected to port B (input). The keyboard driver can now write the values of $01, $02, $04 etc. in port A and test the input of port B to see which keys are pressed. The two joysticks are connected in parallel to port A and port B, so they can cause spurious keyboard events.

CIA 2 is hooked up to the IEC bus, and I/O lines control the VIC bank. The rest is exposed on the user port, and can be used for RS-232.

KERNAL uses CIA 1 for the 50 Hz system timer, but, apart from the ports, doesn't use any of the extra features of either CIA.

6581 SID

The SID (Sound Interface Device) is a whole topic of its own.

6567/6569 VIC

The video chip inside the C64 is called the MOS 6567/6569 VIC-II (Video Interface Controller) - the video chip in the VIC-20 had been the original VIC, and was the reason for the marketing name of the VIC-20.

The VIC supports a 40x25 text mode, a 320x200 bitmap mode, 16 colors and 8 sprites - all of these features have lots of sub-modes and options. The amount of memory the VIC can address is 16 KB, and while by default, it accesses the first 16 KB of the C64 RAM, it can be configured to use any of the four banks.

The 16 colors of the VIC are divided into two sets of eight. The first eight are the more important colors, as some modes only support the first eight. The colors are, in the original order: black, white, red, cyan, purple, green, blue, yellow, orange, brown, pink, dark gray, gray, light green, light blue and light gray.

Character Mode

The C64 has two built-in character sets that the VIC can access. They can be shown on the screen by writing the numbers 0 to 255 into screen RAM (at $0400 by default). The default font has uppercase characters and lots of line-drawing symbols in the lower 128 characters, and the second half consists of the same characters, but inverse. The alternative font has upper- and lowercase characters and omits some of the symbols.

The foreground color of the characters can be changed by writing the color numbers into the Color RAM, which is located at $D800-$DBFF. There is one byte per character, but only the lower 4 bits are actually preserved by Color RAM.

Each character is 8x8 pixels, and stored as eight bytes in the character ROM, one line being one byte. A 1-bit will take the color from the Color RAM ($D800-$DBFF), and a 0 bit will take it from the global background color register ($D021). The pixel matrix is determined by looking up the character index in the screen RAM (at $0400-$07FF by default) and consequently looking up the pattern the current character set (the VIC sees the default font at $1000, although for the 6502 it is invisible there).

In Extended Color Mode (ECM), it is possible to chose between one of four background colors (registers $D021 to $D024) with the upper two bits of the character index, but then only 6 bits will be used to look up the character pattern, decreasing the number of possible characters to 64. The built-in (uppercase) character set is well-suited for this: While it is similar to the ASCII encoding, it has the uppercase characters mapped to codes $01 to $1A, so the most important characters are within the first $40.

Multi-Color Character Mode allows up to four colors per character and is intended for tile-based games, like platformers. If bit 3 of the value in Color RAM is 0, then the character gets displayed just like in non-multicolor mode, but colors are restricted to the first eight. If bit 3 is 1, then pairs of horizontally adjacent bits are combined in their meaning: 00 represents the screen background ($D021), 01 is the second background register ($D022), 10 is the third background register ($D023), and 11 is the color specified in bits 0-2 of the Color RAM. Pixels in these characters are twice as wide, so the resolution of a character is 4x8.

Bitmap Mode

In hi-res graphics mode, the VIC supports a resolution of 320x200, which uses the same pixel frequency as 40x25 character mode (40*8=320, 25*8=200). The bitmap can reside at $0000 or $2000, and the VIC reads one byte for 8 pixels. But hi-res mode does not only support monochrome graphics: The foreground and background colors of each 8x8 tile are taken from the high and low nibble of screen RAM, which would otherwise be unused. Color RAM is not used in this mode.

The encoding of the bitmap is identical to the encoding of a character set, making it a non-linear framebuffer: The first eight bytes of the bitmap represent the pixels in the tile at character position (0,0), the second eight bytes represent the tile at character position (1,0), which is pixel position (8,0), and so on. This layout makes pixel addressing in software slower.

In Multi-Color Bitmap Mode, the horizontal resolution is halved to 160x200, and pixels are twice the width. Every set of two bits in the encodes one of four colors per tile: 00 takes it from the global background register ($D021), 01 and 10 take it from the upper and lower nibble of screen RAM, respectively, and 11 takes it from Color RAM.

Scrolling

The VIC supports hardware X and Y scrolling by 0 to 7 pixels. Since the 40th column is half visible and another column left of the first column is half-visible when the horizontal shift register is set to e.g. 4, a 41th column would be needed. Instead, it is possible to switch the screen to 38 column mode, i.e. the whole screen is a little narrower, and more border is shown on the left and on the right. The screen can also be switched from 25 to 24 lines the same way.

Sprites

The VIC has eight hardware sprites (also called MOBs, movable objects). Each sprite is 24x21 pixels, which is encoded in 63 bytes. Set bits will be drawn in the sprite's individual foreground color, and cleared bits will be transparent. The index to the sprite's bitmap data in memory is an 8 bit value that is read from the last 8 bytes of screen RAM - since the screen is only 1000 characters, the last 24 characters of the $0400=1024 bytes area would otherwise be unused.

There is also a multicolor mode for sprites, which makes pixels twice as wide and decreases the horizontal resolution to 12 pixels. In this mode, 00 is still transparent, and 10 encodes the sprite's individual color. The codes 01 and 11 take the color out of the sprite multicolor registers ($D025 and $D026), which are shared among all sprites.

Sprites can be positioned at arbitrary pixel positions on the screen, and overlap. In this case, sprites with lower numbers have priority over sprites with higher numbers. Each sprite can either be shown in front or behind background pixels. Sprites can be X- and Y-expanded by a factor of two, and collision of two sprites or of a sprite and background pixels is de-

tected by hardware: Whenever two non-transparent sprite pixels are drawn at the same position on the screen, they have collided. Whenever a non-background pixel is drawn by the character generator at the same position where a non-background pixel of a sprite is draw, the sprite has collided with the background. An exception from this rule is the 01 code in the character data, which also counts as background. This way, a background picture can be constructed that does not cause collisions in certain areas. In practice, most newer games do not use the hardware functionality, but instead test for overlapping sprite bounding boxes in software.

Memory Layout

The VIC can address 16 KB at a time. All VIC data structures can be stored anywhere in these 16 KB, but they have to be aligned to their size.

- The screen RAM is $0400 bytes in size and can be at $0000, $0400, ...
- The character set is $0800 bytes, and can be at $0000, $0800, ...
- The bitmap is $2000 bytes and can be at $0000 or $2000.
- Sprites are $40 bytes, and can be at $0000, $0040, ...

Two GPIO pins of the second 6526 CIA are connected to bits 6 and 7 of RAM when the VIC accesses it. By changing the lower 2 bits of $DD02, the VIC can be switched between banks $0000-$3FFF (11), $4000-$7FFF (10), $8000-$BFFF (01) and $C000-$FFFF (00).

If the VIC is set to banks $0000 or $8000, then the two built-in character sets shadow RAM in the area of $1000-$1FFF. This means that the built-in character set can be used on those banks without occupying RAM, but it also means that the area from $1000-$1FFF cannot be used for bitmap, screen RAM or sprite data either.

For timing reasons, color information is not taken from main RAM, but from a dedicated Color RAM. These $0400 half-bytes are accessible to the C64 at $D800-$DBFF and can not be bank switched.

Timing

For advanced VIC programming, it is necessary to not just set up a certain mode and have the VIC display it, but to reprogram the VIC while it is drawing the picture. For this, it is necessary, to understand its timing.

While the pixels within the screen area are 320 by 200, the VIC actively draws pixels in the border color outside of this area, which (on PAL) is 403x284 pixels. Analog TV standards specify an H blank area at the end of every line, and V blank area at the end of every screen. So counting this timing as pixels, this gives an absolute resolution of 504x312 pixels. The interesting and very useful connection about the pixel clock and the system clocks is that an 8 pixel character is drawn every system clock cycle, i.e. about 1 million times a second. The 504 horizontal pixels therefore mean that a line is drawn on the screen every 63 cycles. With this information, it is possible to do cycle-exact timing of assembly code to switch a VIC register at an 8 pixel granularity.

Further timing details (badlines, sprite timing), as well as the application of this information to do tricks like FLD, FLI and AGSP would go beyond the scope of this article, but are talked about in the presentation at the 25th Chaos Communication Congress.

Memory Configuration

In a running system with BASIC and KERNAL, the BASIC and KERNAL ROMs are turned on and visible at $A000-$BFFF and $E000-$FFFF respectively, and at $D000-$DFFF, the I/O area is visible. Using the 3 lowest bits in the processor port at address 1, this configuration can be altered. The ROMs can be turned off, revealing RAM instead, and the I/O area can be configured to show either RAM or the character set ROM. Note that writing to ROM will always direct the data to the underlying RAM.

In practice, many programs run in the default configuration and use both KERNAL library routines, as well as functions in BASIC, to keep their own code as small as possible. More recent programs and almost all games turn off all of ROM, to get direct control of the interrupt vector without having to go though the KERNAL handler first. The I/O area is typically configured to show the I/O registers and Color RAM, and only rarely switched to a different configuration to temporarily access the RAM underneath. A few applications read the character ROM at program start and modify the copy.

The C64 supports another ROM bank at $8000-$9FFF, which can only be serviced by an external cartridge connected to the expansion port. If KERNAL detects the magic string ”CBM80” at $8004 on startup, it will jump to the code of the cartridge right away.

Tape Interface

The tape interface consists of a single line each for data input and output, motor control and key sense. The raw data is read from and written to the data lines, and all encoding and decoding of the data stream is done in software. 3 of the required lines are connected to the processor port at zero page location 1, and one (data input) is connected to CIA 2.

IEC Bus

The IEC bus is a serial version of the IEEE-488 bus used on the PET. Devices on the IEC bus are daisy-chained, and are all connected to the same three lines: ATN (attention), clock and data. IEC has a single bus master, which is the computer. It is the only device to ever raise ATN, while every device can output to clock and data, depending on the state of the bus.

If the computer raises ATN, every device on the bus listens for the 4 bit device number and compares it with its own. The protocol on who sends and who receives is determined by the computer sending TALK/UNTALK and LISTEN/UNLISTEN requests, quick is

an ATN sequence, followed by one of the four commands.

While KERNAL exports the interface at this level, it also allows high-level open, close, read and write operations on the IEC bus, as well as load and save operations. The BASIC LOAD and SAVE commands are directly hooked up to this interface.

The IEC bus was designed for the serial shift register in the VIA (Versatile Interface Adapter) of the VIC-20 and its disk drive, but it turned out that the VIA had a bug that made the shift register unusable, so the IEC protocol had to be implemented in software. While the C64 has CIAs, in which the bug has been fixed, the 1541 still used VIAs. It wasn't until the C128 (in its native mode only) that the computer could talk to the floppy drive (Commodore 1570/1571/1581) in its intended speed.

1541 Disk Drive

The Commodore 1541 Disk Drive is the most common disk drive used with the C64. It uses 5.25" SS/DD (single side, double density) disks, but disks can be flipped, and the other side can be used as well, if the disks are double sided. The 1541 does not use the index hole, and uses software markers (SYNC) instead to be able to tell the start of a sector. Due to reliability problems of early drives, the 1541 only uses 35 out of the 40 possible tracks on a 5.25" disk. The tracks have a variable number of 256 byte sectors, ranging from 21 on the outside to 17 on the inside. The data is written in 4 different speeds. This makes an overall 683 sectors, or 174,848 bytes.

The file system is stored on track 18. Track 18, sector 0 contains the disk name and the BAM (block availability map), which stores one bit per sector (1 = free). Track 18, sector 1 is the first sector containing directory entries: There are eight 32 byte entries per sector, with a maximum filename length of 16 characters. The first two bytes of a directory sector point to the next directory entry sector.

The files on disk are also stored as a linked list. The first two bytes of every sector are either the track and sector number of the next block, or the first byte is 0 and the second byte is the number of valid bytes in this sector.

The 1541 is a stripped down version of the PET drive series, which had a parallel connection, and contained two 6502 CPUs: One for doing the filesystem and communicating with the computer, and one for reading data from disk and writing data to it, as well as encoding and decoding the data. The 1541 only has a single 6502 CPU running at 1 MHz, which (using timer IRQs) regularly switches itself between the two modes. The two virtual CPUs still communicate with each other using a messaging interface in the zero page. The 1541 has 2 KB of RAM at $0000-$07FF.

The 1541 has two VIA I/O controllers at $1800 (for the IEC bus) and at $1C00 (for the drive). The firmware is located at $C000-$FFFF.

Since loading an application or a game takes minutes on an unmodified C64, several ”floppy speeders” appeared (either as software on disk or built into applications, as ROM extension cartridges, or as internal replacement ROMs), that consisted of implementations of more optimized protocols for the IEC bus for both the C64 and the 1541. The 1541 code was uploaded using the old bus protocol. Such a new protocol would for example not do a handshake on every bit using the clock line, but shift a complete byte through in 4 steps, two bits at a time, using the clock and data line at the same time. This would of course only work if both CPUs were not interrupted. VIC timing on the C64 side could already affect this, so many floppy speeders turned off the screen while loading.

Other Peripherals

The 1541 is the necessary companion to a C64. It can be replaced by a 1570 (1541 with fast bus routines for the C128) or a 1571 (double-sided 1570), since they include a 1541 compatibility mode. The 3.5" Commodore 1581, which supports 880 KB per disk, can hardly be a replacement for a 1541, because most applications contain their own floppy speeders that make lots of assumptions on the exact on-disk format. For GEOS, it can be very useful though.

Creative Micro Devices sold and continues to sell a line of hard drives that have an IEC connection but contain a 3.5" SCSI drive inside. Although they have a 6502 CPU built in and allow code execution on their CPU, they are not compatible enough to replace a 1541 either.

Several memory extension cartridges exist for the expansion port of the C64: The Commodore REU (contains a DMA chip that transfers data between itself and main RAM), as well as the third party GeoRAM (maps a block to the $DE00-$DFFF area) and RAMLink (battery backed, designed as RAM disk).

Freezer cartridges like the Action Replay and the Final Cartridge were not only popular because they could dump all of memory to disk and thus copy certain copy-protected games, but also because they featured floppy speeders that disabled the original routines directly at startup time, without any effort from the user.

In the mid 90s, CPU speeders for the expansion port became popular. The 8 MHz Flash 8 is rare today, but many enthusiasts have a SuperCPU, which replaces the onboard CPU with a 20 MHz 65C816, which has a native 16 bit mode that can address up to 16 MB or RAM. There are a few applications and games that require a SuperCPU. The speedup of GEOS with a SuperCPU is significant.

In the 2000s, the enthusiast scene created all kinds of peripherals, like ethernet interfaces, IDE interfaces and SD card readers.

And there are not only peripherals: In 2004, the Commodore 64 DTV, a reimplemented C64 appeared in the form of a Joystick. The device is fairly compatible and can be extended to connect to a keyboard and a 1541.