

Chapter 1

Hardware

To study the IBM PC, it is easiest to first break it down to small parts. Six sub-systems form a pipeline: Inputs, CPU, RAM, Disk, Video, and Audio.

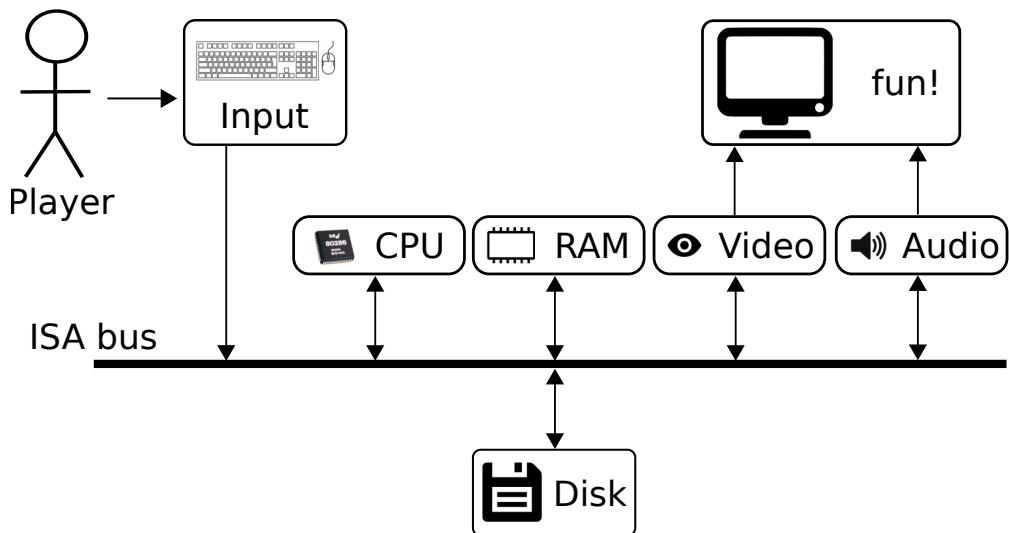


Figure 1.1: Hardware pipeline.

A lot of friction was present since manufacturers had not embraced the gaming industry yet. Parts quality varied from bad, terrible, to downright impossible to deal with.

Stage	Quality
RAM	Bearable
Video	Impossible
Audio	Very Poor
Inputs	Ok
CPU	Very Poor
Disk	Ok

Figure 1.2: Component quality for a game engine.

1.1 CPU: Central Processing Unit

In 1989 around 15% of the households owned a computer¹. The performance of these machines was so overwhelmingly determined by the CPU that a PC was referred to not by its brand or GPU², but by the main chip inside. If a PC had an Intel 8088 or equivalent, it was called a "XT". If it had an Intel 80286, it was a "286" or "AT".

1.1.1 Overview

Intel released the 8086 in 1979, which was the first microchip of the successful x86 family line. One year later, in 1979, it released the 8088 which was a variant of the 8086. The main difference between the two is that there are only eight data lines for the external data bus in the 8088 instead of the 8086's 16 lines. However, because it retained the full 16-bit internal registers and the 20-bit address bus, the 8088 ran 16-bit software and was capable of addressing a full 1MB of RAM. IBM chose the 8088 over the 8086 for its original PC/XT, because Intel offered a better price for the former and could supply more units.

In 1982 Intel released the 80286 microchip. A typical 8088 chip was running at 4.77Mhz, where the 80286 was running at 8Mhz and later versions at 12-16Mhz. The 80286 was employed for the IBM PC/AT, introduced in 1984, and then widely used in most PC/AT compatible computers until the early 1990s. Commander Keen could run on a 8088, but an Intel 286 was recommended.

¹<https://www.statista.com/statistics/184685/percentage-of-households-with-computer-in-the-united-states-since-1984/>

²There was no GPU yet. The term was coined by Nvidia in 1999, who marketed the GeForce 256 as "the world's first GPU", or Graphics Processing Unit.

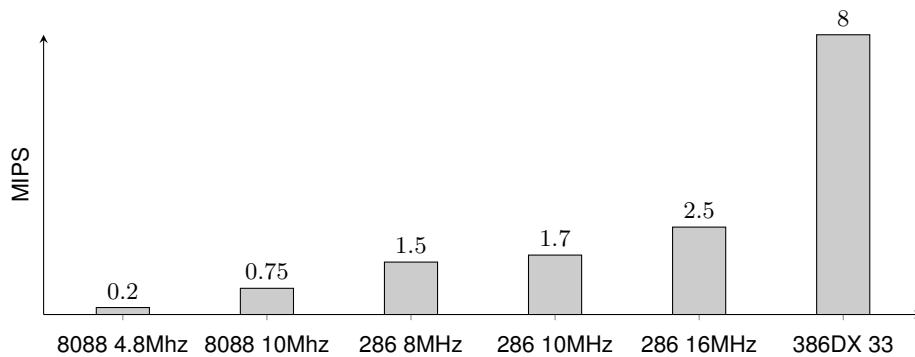


Figure 1.3: Comparison³ of CPUs with MIPS

Trivia : A modern processor such as the Intel Core i7 3.33 GHz operates at close to 180,000 MIPS.

1.1.2 The Intel 80286

The Intel 80286 chip, first introduced in 1982, is the CPU behind the original IBM PC AT (Advanced Technology). Other computer makers manufactured what came to be known as IBM clones, with many of these manufacturers calling their systems AT-compatible or AT-class computers.



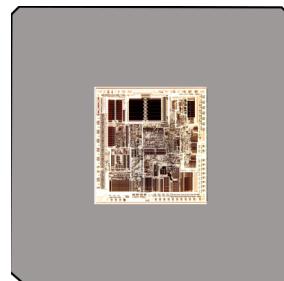
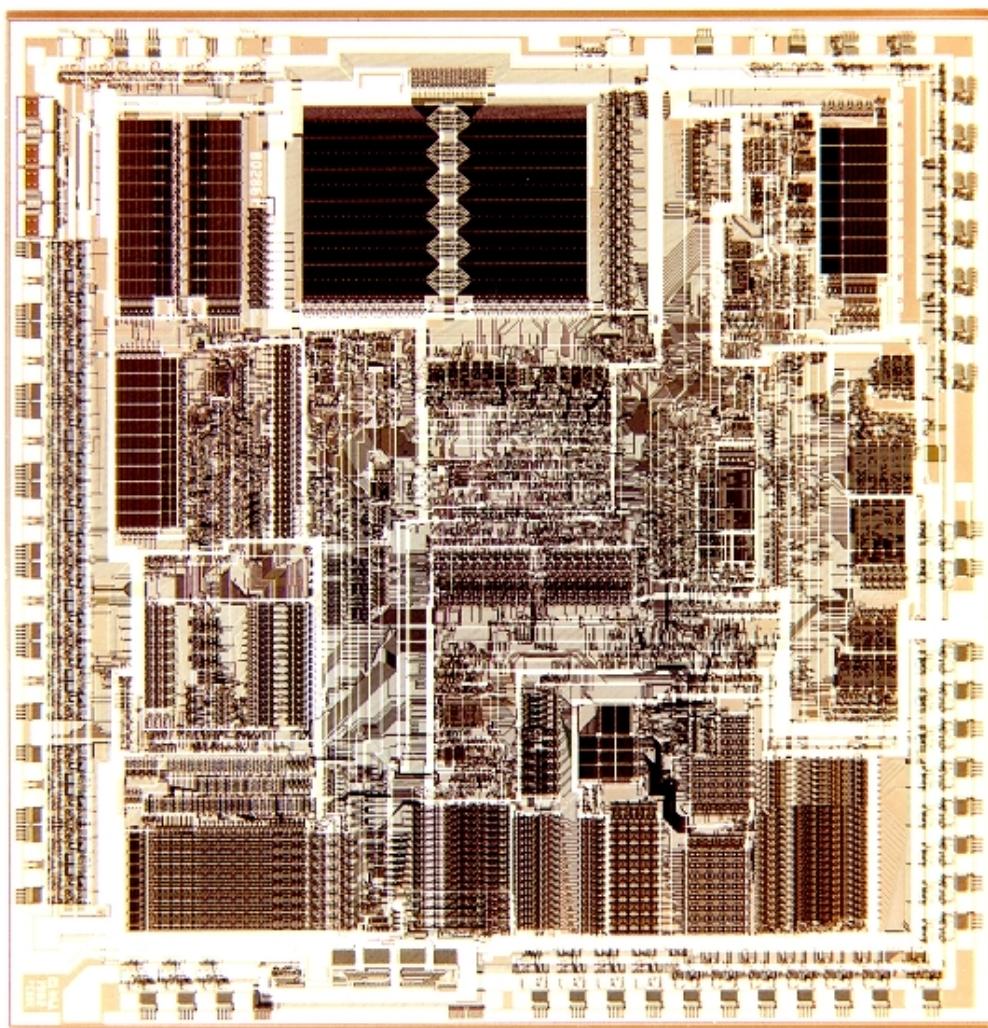
When IBM developed the AT, it selected the 286 as the basis for the new system because the chip provided compatibility with the 8088 used in the PC and the XT. Therefore, software written for those chips should run on the 286. The 286 chip is many times faster than the 8088 used in the XT, and at the time it offered a major performance boost to PCs used in businesses. The processing speed, or throughput, of the original AT (which ran at 6MHz) is five times greater than that of the PC running at 4.77MHz. 286 systems are faster than their predecessors for several reasons. The main reason is that 286 processors are much more efficient in executing instructions. An average instruction takes 12 clock cycles on the 8086 or 8088, but takes an average of only 4.5 cycles on the 286 processor. Additionally, the 286 chip can handle up to 16 bits of data at a time through an external data bus twice the size of the 8088.

The 286 chip has two modes of operation: real mode and protected mode. The two modes are distinct enough to make the 286 resemble two chips in one. In real mode, a 286 acts

³Roy Longbottom's PC Benchmark Collection: <http://www.roylongbottom.org.uk/mips.htm>.

essentially the same as an 8086 chip and is fully compatible with the 8086 and 8088. In the protected mode of operation, the 286 was truly something new. In this mode, a program designed to take advantage of the chip's capabilities has access to 1GB of memory (including virtual memory). The 286 chip, however, can address only 16MB of hardware memory. A significant failing of the 286 chip is that it cannot switch from protected mode to real mode without a hardware reset (a warm reboot) of the system (It can, however, switch from real mode to protected mode without a reset).

While the 8088 used a $3.0\mu\text{m}$ process, the 20286 used a $1.5\mu\text{m}$ process. The smaller process and increased surface (from 33mm^2 to 49mm^2) allowed Intel to pack 134,000 transistors on a 286 chip versus 29,000 on a 8088 chip.



Despite the apparent complexity, the 80286 can be summarized by functional units and a three-stage instruction pipeline.

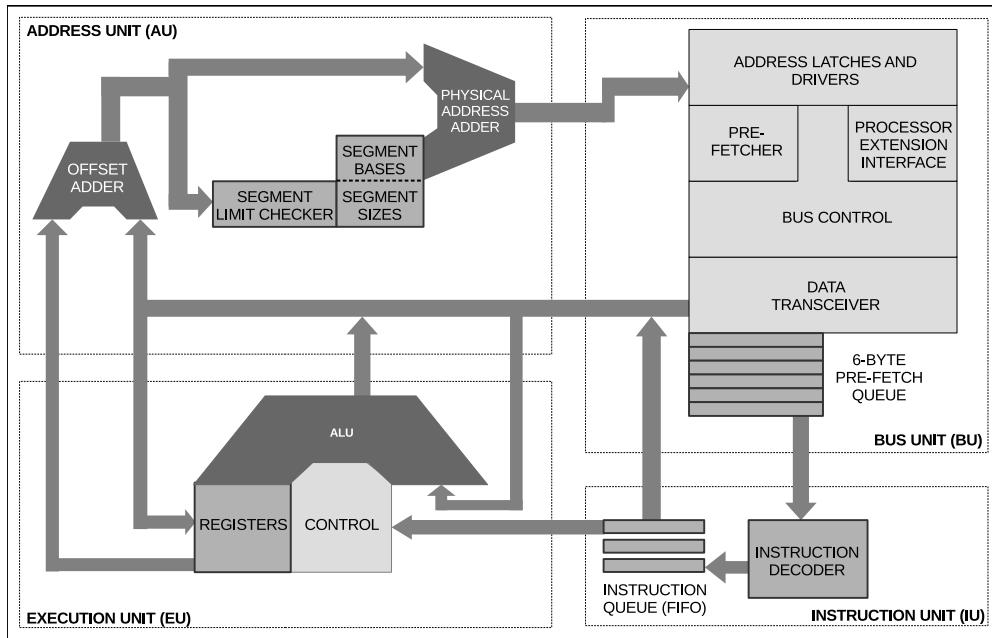
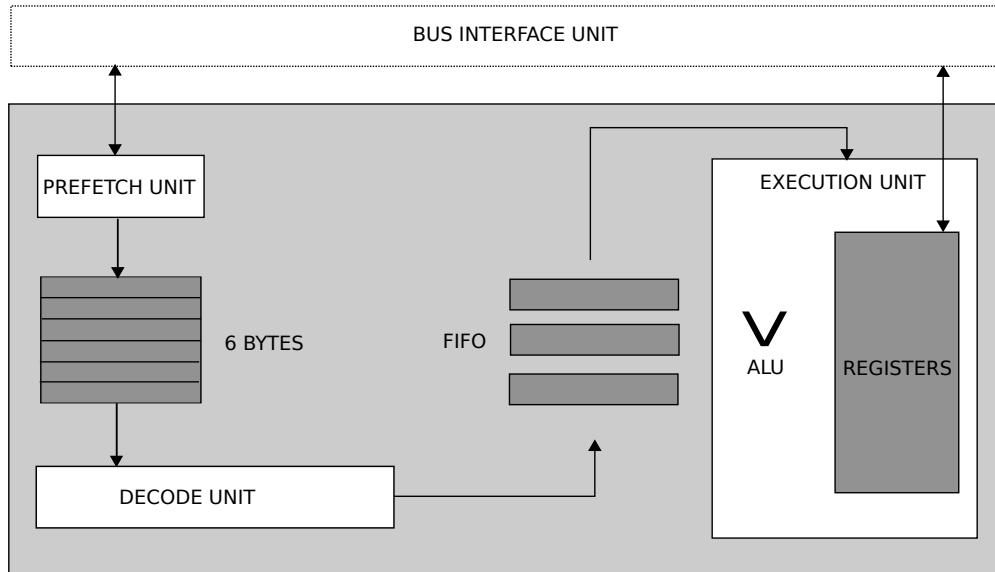


Figure 1.4: Internal block diagram of the 80286 processor

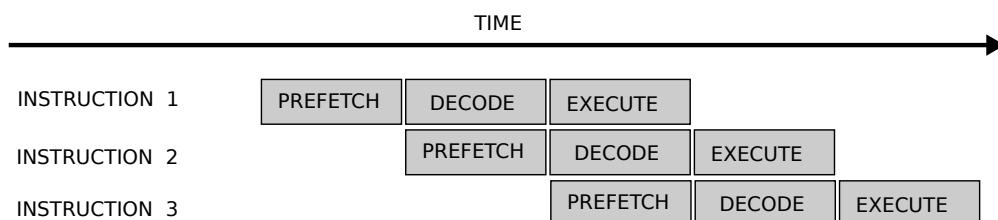
The four functional units can be described by

- **address unit (AU)** is used to determine the physical addresses of instructions and operands which are stored in memory. The address lines derived by AU can be used to address different peripheral devices such as memory and I/O devices.
- **bus unit (BU)** interfaces the 80286 with memory and I/O devices. The bus unit is used to fetch instruction bytes from the memory and stores them in the prefetch queue.
- **instruction unit (UI)** receives instructions from the prefetch queue and an instruction decoder decodes them one by one. The decoded instructions are latched onto a decoded instruction queue.
- **execution unit (EU)** is responsible for executing the instructions received from the decoded instruction queue. The execution unit consists of the register bank, arith-

metic and logic unit (ALU) and control block. The ALU is the core of the EU and perform all the arithmetic and logical operations.



The three units in the execution group form a three stage pipeline: Prefetch, Decode, and Execute. The Prefetch Unit wakes up when the Execution unit is performing but not using the bus and fetches instructions in a 6-byte queue. The prefetcher is linear and cannot predict the result of a branch. As a result, a jump (JMP) instruction triggers a flush of the entire pipeline. Instructions go down the pipeline and are decoded by the Decode Unit: the result of the decode operation is stored in a three-element FIFO where it is picked up by the Execution Unit.



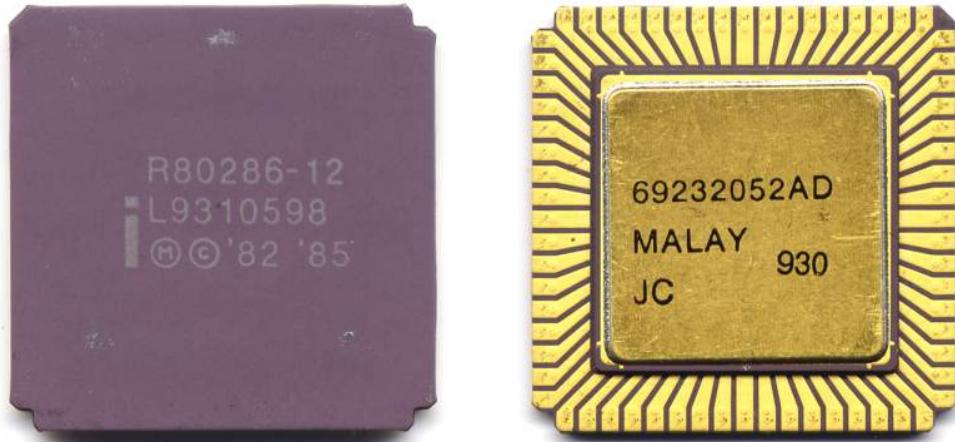


Figure 1.5: The Intel 286, 10mm by 10mm packing 134,000 transistors

From a programming perspective, a 286 CPU can be summarized by the following elements:

- Arithmetic Logic Unit performing add, sub, mul et cetera.
- 14 registers:
 - 16-bit General Purpose Registers: AX, BX, CX, DX
 - 16-bit Index Registers: SI, DI, BP, SP
 - 16-bit Segment Registers: CS, DS, ES, SS
 - 16-bit Status and Control Register
 - 16-bit Program Counter: IP
- A 24-bit address bus for up to 16MB of flat addressable RAM
- Memory Management Unit

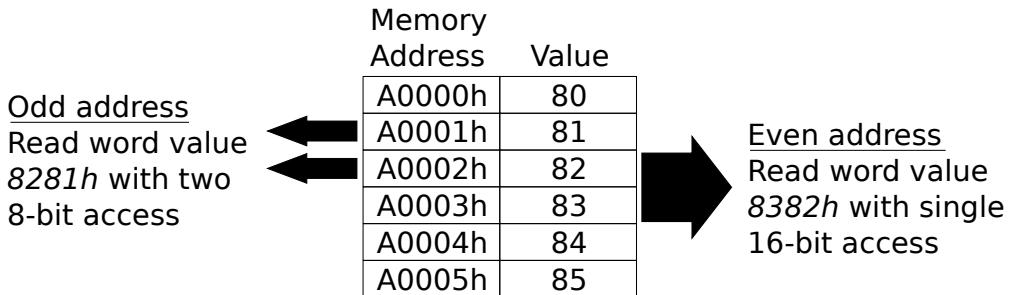
Despite its pipeline design, the 286 cannot do an operation in less than two cycles. Even a simple ADD reg, reg or INC reg takes two clocks. This is due to the absence of a SRAM on-chip cache and a slow decoding unit. Also have a look at multiplications which cost 24 cycles. So as a game developer you really want to avoid many multiplications during game runtime.

Instruction type	Clocks
ADD reg8, reg8	2
INC reg8	2
IMUL reg16, reg16	24
IDIV reg16, reg16	28
MOV [reg16], reg16	5
OUT [reg16], reg16	3
IN [reg16], reg16	5

Figure 1.6: 286 instruction costs⁴

1.1.3 16-bit Data alignment

Compared to the Intel 8088, the 286 CPU contained a 16-bit external data bus where the 8088 only had a 8-bit bus. Thanks to its 16-bit bus, the 286 can access and write word-sized memory variables just as fast as byte-sized variables. There's a catch, however: That's only true for word-sized variables that start at even memory addresses. When the 286 is asked to perform a word-sized access starting at an odd memory address, it actually performs two separate accesses, each of which fetches 1 byte, just as the 8088 does for all word-sized accesses. In other words, the effective capacity of the 286's external data bus is halved when a word-sized access to an odd address is performed⁵.



The way to deal with the data alignment cycle-eater is straightforward: Don't perform word-sized accesses to odd addresses on the 286 if you can help it. This is not an issue for small memory operations, but it will harm performance when copying large memory blocks.

⁴Intel 80286 programmer's reference manual - 1987.

⁵See Michael Abrash's Graphics Programming Black Book Special Edition, chapter 11.

1.2 RAM

The first CPUs in the Intel x86 family were designed in 1976. It was introduced at a time when the largest register in a CPU was only 16-bits long which meant it could address only 65,536 bytes (64 KiB⁶) of memory, directly. As example both the Apple II and the Commodore 64 both shipped with 64KiB which was enough to write and run amazing things.

But everyone was hungry for a way to run much larger programs. Rather than create a CPU with larger register sizes, the designers at Intel decided to keep the 16-bit registers for their new 8086 and 8088 CPU and added a different way to access more memory: They expanded the instruction set, so programs could tell the CPU to group two 16-bit registers together whenever they needed to refer to an absolute memory location beyond 64 KiB.

1.2.1 Memory addressing

If the designers had allowed the CPU to combine two registers into a high and low pair of 32-bits, it could have referenced up to 4 GiB of memory in a linear fashion. Keep in mind, however, this was at a time when many never dreamed we'd need a PC with more than 640 KiB of memory for user applications and data⁷. So, instead of dealing with whatever problems a linear addressing scheme of 32-bits would have produced, they created the Segment:Offset scheme which allows a CPU to effectively address about 1 MiB⁸ of memory. The Segment:Offset schema combines two 16-bit registers, one designating a segment and the other an offset within that segment.

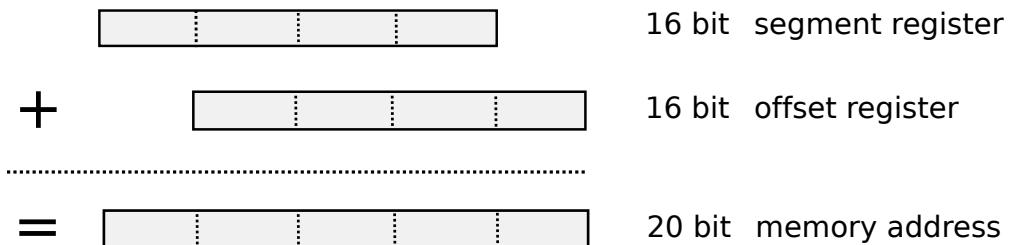


Figure 1.7: How registers are combined to address memory.

⁶This book uses IEC notation where KiB is 2^{10} and KB is 10^3 .

⁷We've often heard that Bill Gates said something to the effect: "640K of memory should be enough for anyone.". Though many of us no longer believe he ever said those exact words, he did, however, during a video interview with David Allison in 1993 for the National Museum of American History, Smithsonian Institution, say: "I laid out memory so the bottom 640KiB was general purpose RAM and the upper 384KiB I reserved for video and ROM, and things like that."

⁸This book uses IEC notation where MiB is 2^{20} and MB is 10^6 .

To support this architecture, the CPU keeps track of four different segments. Each of these segments has its own purpose and segment register:

- CS segment register, where the machine instructions resides.
- DS segment register, where the data resides.
- SS segment register, where the stack resides.
- ES segment register, which is used as extra data segment.

In C-language a memory address can be accessed directly using pointers. A pointer is a variable that stores the memory address of another variable as its value. There are two kinds of pointers: `near` and `far`.

A `near` pointer refers to a function or data object that is within the default segment. It is 16 bits long and contains an offset into the current DS data segment if it's a data pointer, or into the current CS code segment if it's a function pointer. A `far` pointer could refer to a function or data that is in a different segment than the current, default segment. It is 32 bits long and contains a segment and offset, which identifies the location where the code or data is stored.

Accessing code or data with a `near` reference is much quicker than accessing it with a `far` pointer. When you use a `far` reference, your program must first find the segment and then find the code or data within that segment. When you use a `near` reference, your program only needs to find the code or data via the offset. For a faster program, one should make as many `near` references as possible. The downside of using `near` pointers only is that it cannot handle more than 64KiB of memory for the program or data.

Note that a `far` pointer only increments the offset, not the segment. If you iterate on a data array larger than 64KiB there will be no overflow handling, so you will be able to handle only 64KiB of memory.

```
char far *p = (char far *) 0xA000FFFF;
p++;
printf("%04X:%04X\n", FP_SEG(p), FP_OFF(p));
```

Will output:

```
A000 : 0000
```

You could use yet another type of pointer named `huge` to make pointer arithmetic work beyond 64KiB.

```
char huge *p = (char huge *)0xA000FFFF;
p++;
printf("%04X:%04X\n", FP_SEG(p), FP_OFF(p));
```

Will output the address:

```
B000:0000
```

The huge pointer is based on the absolute (or linear) 20-bit memory location and Segment:Offset normalized address. The absolute memory address can be calculated by

```
Absolute memory address = (Segment * 0x10h) + Offset
```

For example the absolute address of A000:002F is A0000h + 002Fh = A002Fh. By confining the offset to just the hexadecimal values 0h through Fh, we have a unique way to reference all Segment:Offset memory pair locations. To convert an arbitrary Segment:Offset pair into a normalized address is a two-step process that's quite easy for an assembly programmer:

1. Convert the Segment:Offset pair into a single absolute address.
2. Then simply insert the colon (:) between the last two hex digits and add three zeros before the last hex digit.

For example, the normalized address for A000:002F is A002:000F. A huge pointer is normalized when pointer arithmetic is performed on it.

```
# include <stdio.h>
# include <dos.h>
int main (void){
    char huge *p = MK_FP (0xA000, 0xFFFF);
    p--;
    p++;
    printf("%04X:%04X\n", FP_SEG(p), FP_OFF(p));
}
```

Will output:

```
AFFF:000F
```

Trivia : Since the normalized form will always have three leading zero bytes in its offset, programmers often write it with just the digit that counts: AFFF:F

A huge reference is much slower than the far reference as it comes with additional overhead to update the segment and address normalization after every arithmetic manipulation. So, most programmers avoided the huge pointer, unless really needed.

1.2.2 80286 Real and Protected mode

By 1986, hardware had gotten cheaper and Intel made a departure from the old architecture with its 286. This new CPU could be put in what is called "protected mode" featuring a 24-bit-wide address bus for up to 16 MiB of flat RAM protectable with a MMU⁹. To make sure old programs could still run, the 286 processor could be put in "real mode" which replicates how the Intel 8086 and 8088 operated: 16-bit registers, 20-bit address bus giving 1MiB addressable RAM with segmented addressing.

For compatibility reasons all PCs have to start in real mode. You may assume that programmers of the late 80s promptly switched the CPU to protected mode to unleash the full potential of the machines and ditch the 20-year-old real mode. It would have worked out if the operating system had been able to run in protected mode. However, in the name of backward compatibility, Microsoft's DOS could only handle real mode which effectively locked developers into 16-bit programming¹⁰.

With protected mode unavailable, 1990 developers programmed like it was 1976: with a 20-bit-wide address bus offering only 1MiB of addressable RAM. Regardless how much memory was installed on the machine, only 1MiB could be addressed. The memory layout for the real mode is as follows:

- From 00000h to 003FFh : the Interrupt Vector Table.
- From 00400h to 004FFh : BIOS data.
- From 00500h to 005FFh : command.com+io.sys.
- From 00600h to 9FFFFh : Usable by a program (about 620KiB in the best case).
- From A0000h to FFFFFh : UMA (Upper Memory Area): Reserved to BIOS ROM, video card and sound card mapped I/O.

Out of the original 1024KiB, only 640KiB (called Conventional Memory) was accessible to a program. 384KiB was reserved for the UMA and every single driver installed (.SYS and .COM) took away from the remaining 640KiB.

⁹Memory Management Unit

¹⁰16-bit programming and memory managers were covered in Game Engine Black Book: Wolfenstein 3D.

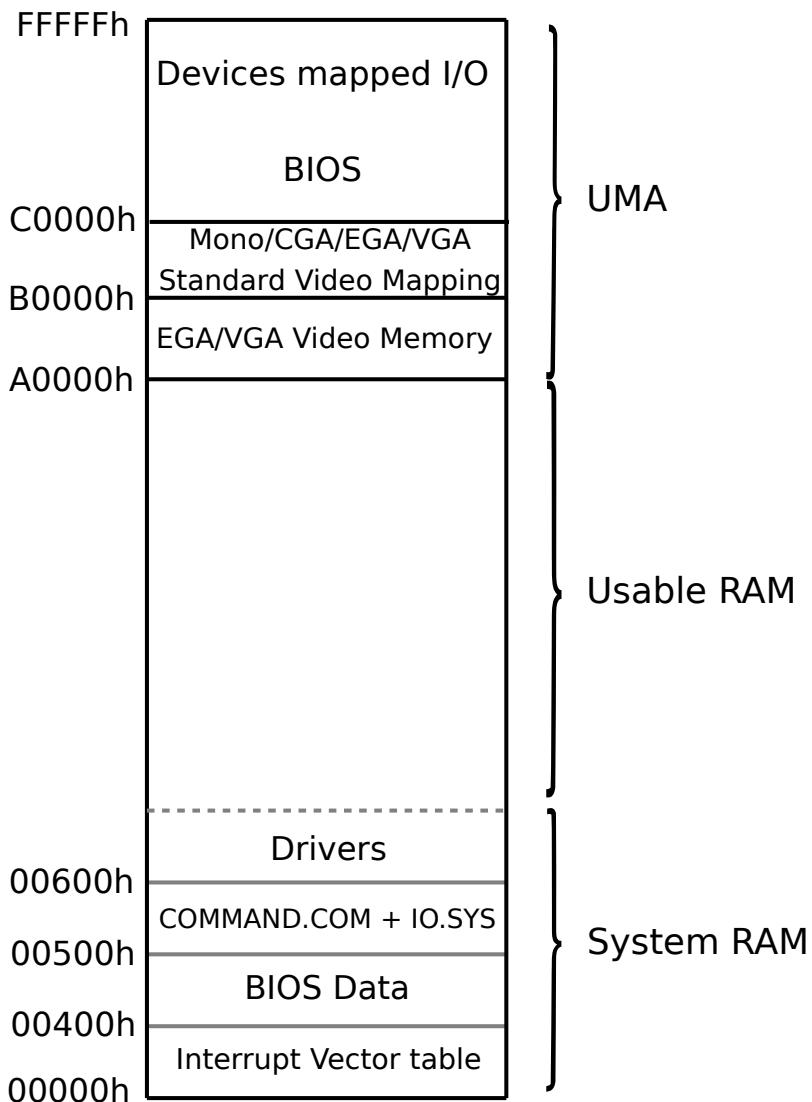


Figure 1.8: First 1MiB of RAM layout.

1.2.3 Real mode: Memory models

When a program is compiled and executed, the operating system allocates a chunk of memory to the program. This memory is divided into different segments as shown below

- The code section stores the program executable. When you compile a C program, the compiler converts your code to assembly instructions that the CPU will execute.
- The data section stores initialized and uninitialized global and static variables.
- The heap section is memory that you can dynamically reserve from calling `malloc()` function. The heap typically grows upwards, which means it grows toward larger memory addresses.
- The stack section is memory for local variables and data inside your functions. The stack grows downwards (i.e. grow toward lower memory addresses).

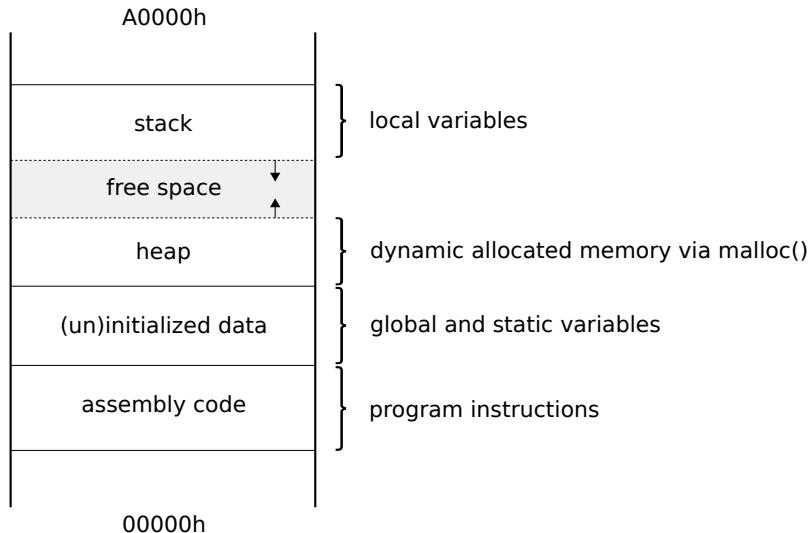


Figure 1.9: Memory layout in x86 real mode.

The x86 real mode provided different memory segment layouts which are called *memory models*. Each memory model controls how the segments are organized, as well as defining the default pointer type for functions and data to *near* or *far*. A *near* pointer is automatically associated with either CS segment register for functions or DS segment register for data. In total six different memory models¹¹ are defined, making trade-offs between minimum system requirements, maximizing code efficiency, and gaining access to every available memory location.

¹¹See Borland C++ 3.1 Programmer's guide, section DOS Memory management.

Model	Default pointer type		Size		Definition
	Code	Data	Code	Data	
Tiny	near	near	<64KiB		CS=DS=SS
Small	near	near	<64KiB	<64KiB	DS=SS
Medium	far	near	>64KiB	<64KiB	DS=SS, multiple code segments
Compact	near	far	<64KiB	>64KiB	single code segment, multiple data segments
Large	far	far	>64KiB	>64KiB	multiple code and data segments
Huge	far	far	>64KiB	>64KiB	multiple code and data segments, global data >64KiB

In the tiny memory model, all three segment registers, CS, DS, and SS, start at the same memory location. The first part of the memory is used for all code instructions, followed by data section. The heap is starting directly after the data section. Finally, the stack is starting at the opposite side of the segment and is growing downwards. Both the heap and stack can grow or shrink dynamically during program execution. If both the heap and stack continue to grow, they may eventually collide, resulting in a system or application crash.

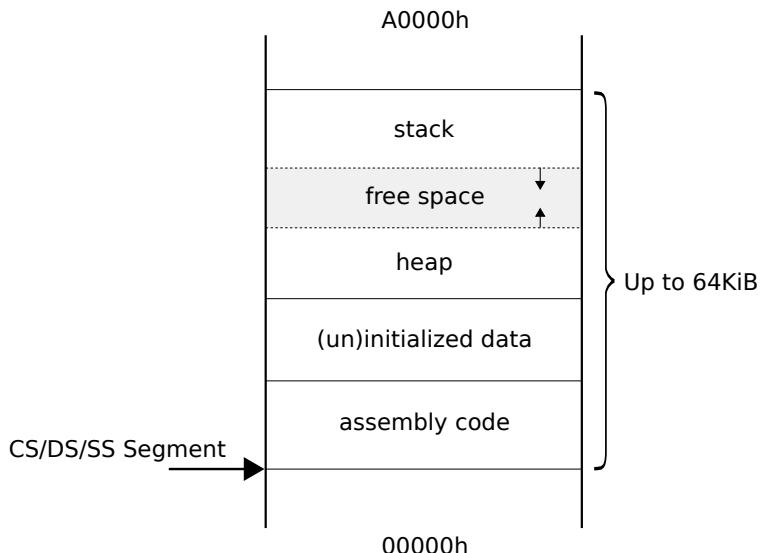


Figure 1.10: Tiny memory model.

All function and data pointers are accessed via near offset references, using the CS and DS segment registers respectively. Since only near references are used, program execution is very efficient. However, there is a downside: everything resides in one segment of a maximum of 64KiB, similar to programming on a 16-bit computer.

Trivia : The tiny memory model was required by programs that ended with the .COM extension, and it existed for backward compatibility with CP/M. CP/M ran on the 8080 processor which supported a maximum of 64KB of memory.

The small memory model separates instructions from data, each having their own 64KiB segment. The code segment has a maximum size of 64KiB for instructions, while the global data, stack, and heap share a separate 64KiB segment. Code execution remains efficient as all functions and data pointers use near references.

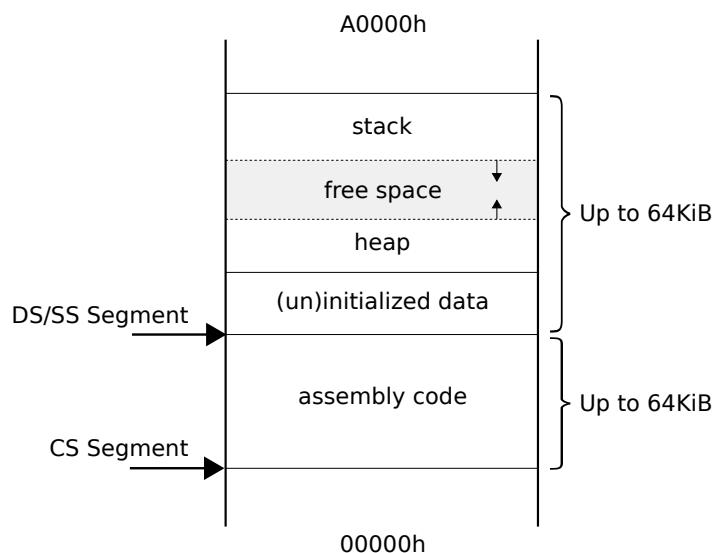


Figure 1.11: Small memory model, code and data each have 64KiB.

The medium model is useful for programs that have a lot of code but not a lot of data. Many computer games fell into this category, because they had a lot of game logic, but not a lot of game state data. Commander Keen was no exception. In the medium memory model each C-source file has its own code segment, allowing up to 64 KiB for each file. A table with all code segment references is maintained, and the CS register points to only one segment at a time. There is no default CS segment, and functions are reached using a far reference pointer.

Trivia : When you compile a source file, the resulting code for that file cannot be greater

than 64KiB, since it must all fit inside one code segment. If the source file is too big to fit into one 64KiB code segment, the programmer must break it up into different source files and compile each file separately.

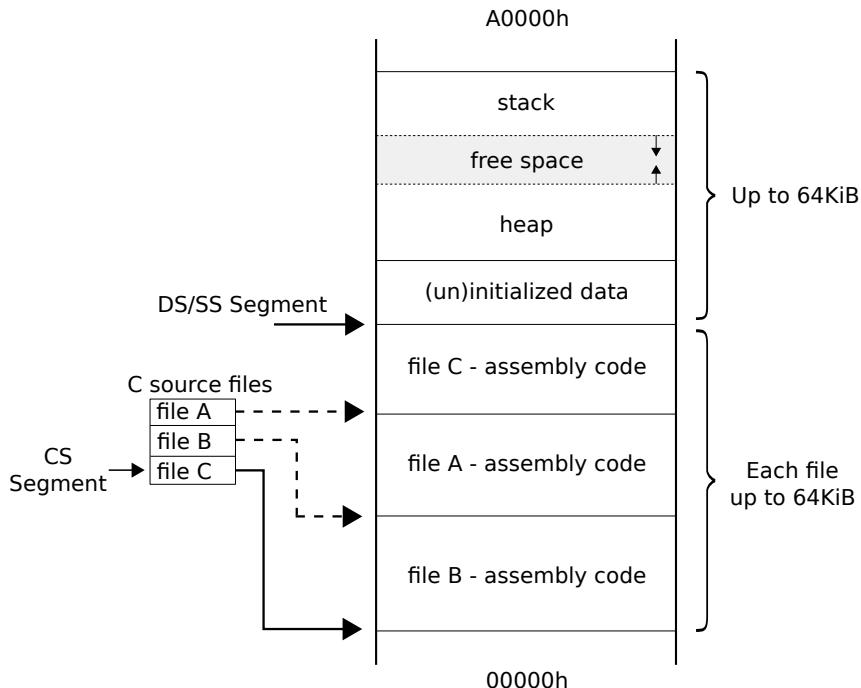


Figure 1.12: Medium memory model, code can be larger than 64KiB.

The total data segment size remains 64KiB. For most games, this is enough for global variables and the stack. However, there is an issue with the size of the heap, which is used to load game data such as graphic assets and levels. One option to increase the data segment size is to use the large memory model, but this comes at the cost of slower data access as it is using far reference pointers for all data access.

There is, luckily, a way to access additional memory using the medium model. A large amount of unallocated memory is available between the start of the heap and the high address **A0000h**. This part of the memory is called the *far heap* and can be allocated using the `farmalloc()` function with far pointer reference. This kind of programming, where you combine a standard memory model with your own *near* and *far* pointer allocation, is called *mixed model programming*. You keep the benefits from the medium memory model, using near pointer references for global data and the stack, and have sufficient memory for dynamic allocation.

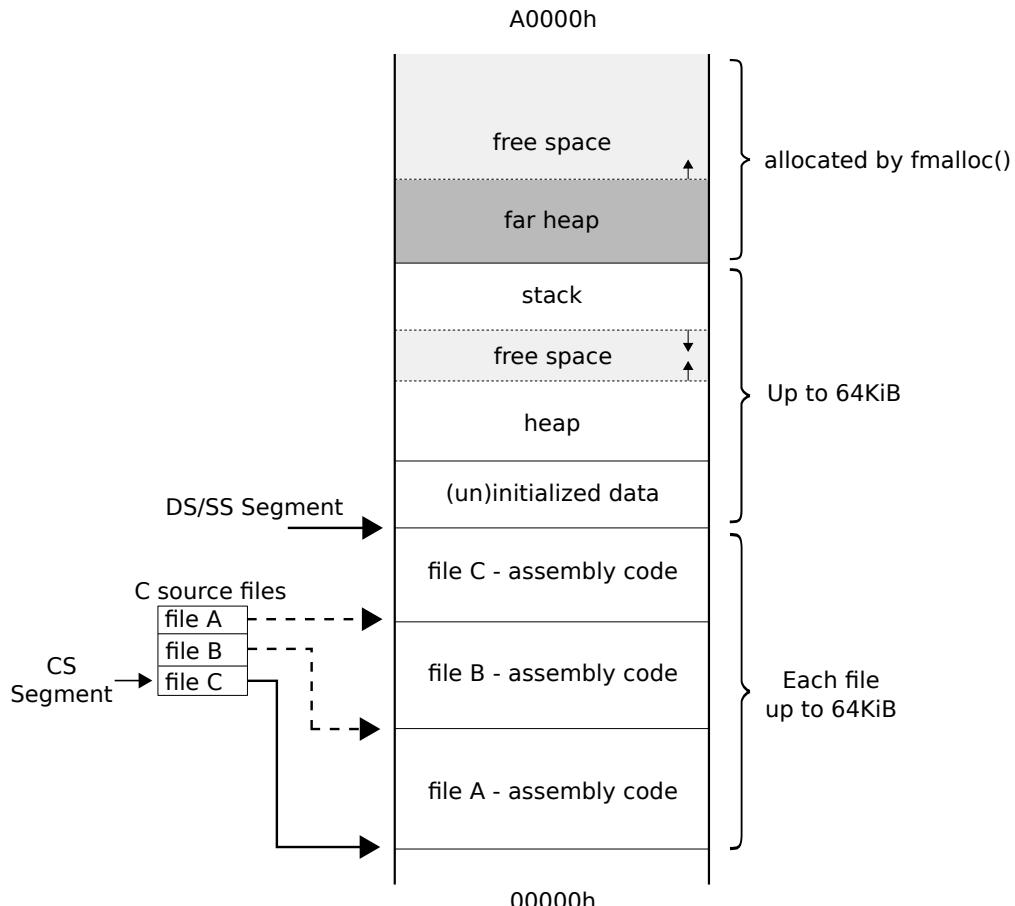


Figure 1.13: Far heap memory.

The opposite of the medium model is the compact model. Here the total function code cannot exceed 64KiB, but there is more space for global data, stack and heap. The large model can deal with function code and data larger than 64KiB, but is slow in execution as only far pointers are used for both code and data. Borland C++ limits the size of all global data to 64KiB. The huge memory model sets aside that limit, allowing global data to occupy more than 64KiB¹².

¹²Also here the same limitation as before: If the source file is too big to fit into one 64KiB data segment, the programmer must break it up into different source files and compile each file separately.

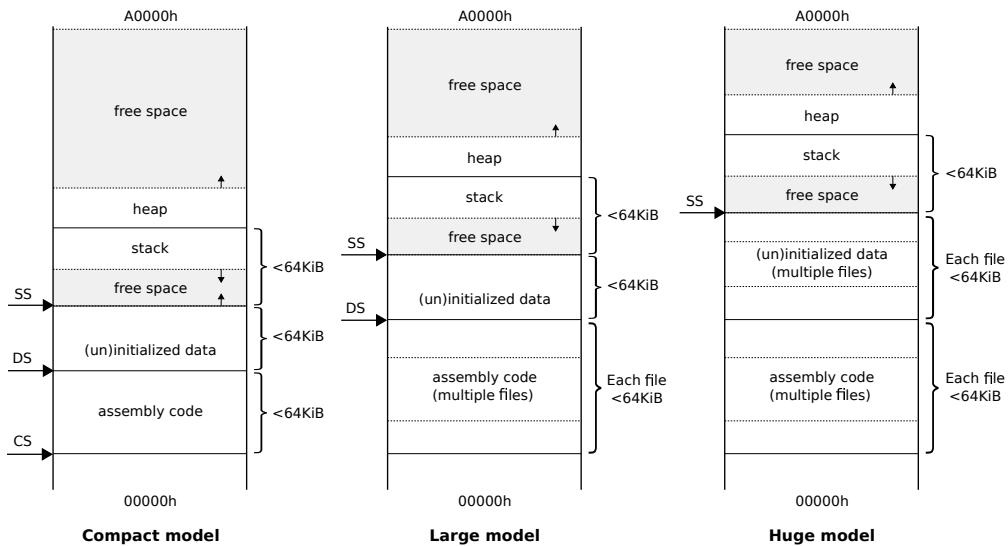


Figure 1.14: Compact, Large and Huge memory model.

Trivia : Although the name implies differently, the huge memory model in Borland C++ is still limited to segments up to 64KiB using far pointers, and not using huge pointers. The Watcom compiler, which gained popularity in the 90's, was able to break the 64KiB segment barrier using the huge pointer reference in the huge memory model¹³.

¹³See <https://open-watcom.github.io/open-watcom-v2-wikidocs/clr.html>

1.3 Video

PCs were connected to CRT monitors: big, heavy, small diagonal, cathode ray-based, curved-surface screens. Most had a 14" diagonal with a 4:3 aspect ratio.

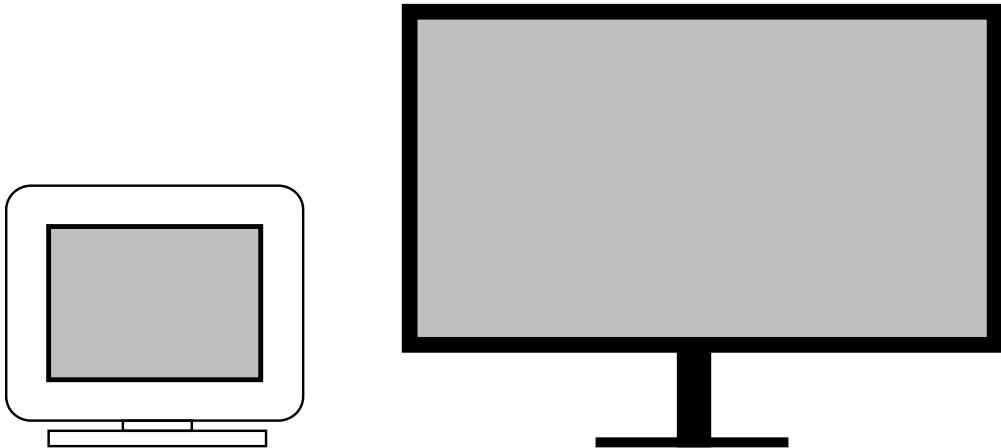


Figure 1.15: CRT (left) vs LCD (right)

To give you an idea of the size and resolution, figure 1.15 shows a comparison between a 14" CRT from 1990 (capable of a resolution of 640x200) and a 30" Apple Cinema Display from 2014 (capable of a resolution of 2560x1600).

Trivia : Despite their difference of capabilities, both monitors are the same weight: 27.5 pounds (12 kg).

1.3.1 CRT Monitor

All standard PC monitors use a raster-scan display to create the image. In a raster-scan display, the position of the electron beam is continually sweeping across the surface of the tube. The tube's surface is coated with phosphors that glow when struck by electrons (and for a short time thereafter), and, of course, the beam may be turned on in order to light a phosphor or off to leave it black.

The electron beam scans the phosphor-coated screen from left to right and top to bottom. The period during which the beams return to the left is known as the horizontal retrace. During most of the retrace, the guns must be turned off to prevent writing in the active display area (the area which contains the actual character and/or graphics data); this is

known as horizontal blanking.

The area immediately surrounding the display area, in which the beam may be turned on during the retrace interval, is called the overscan (or border). The active display area is the portion of the screen that contains characters and/or graphics. These components of the scan are shown in simplified form in Figure 1.16.

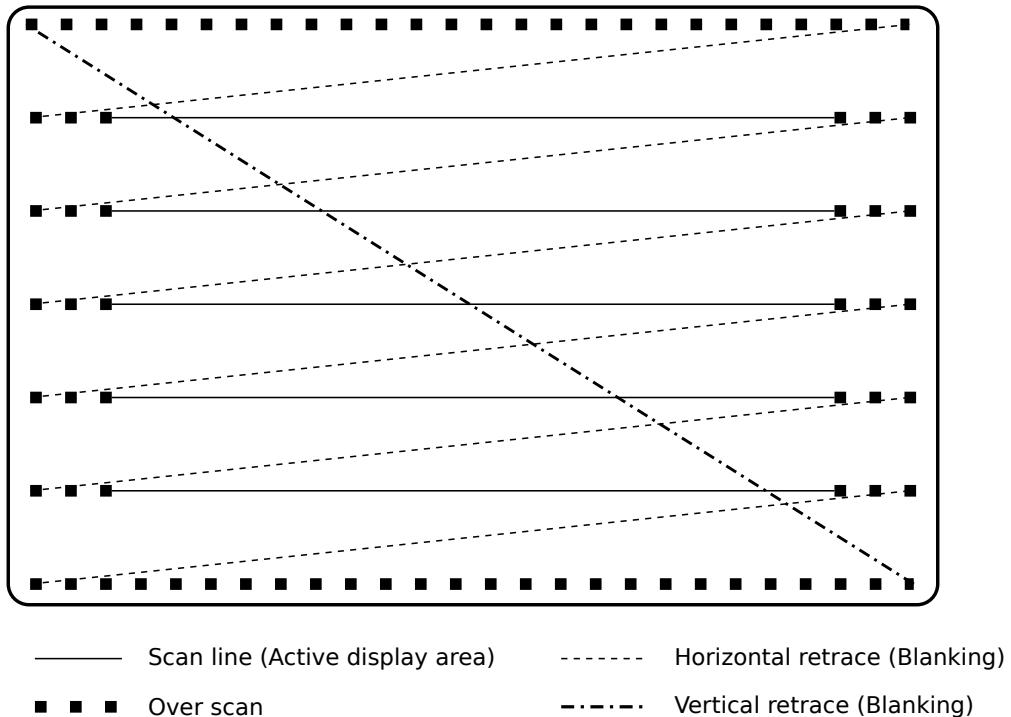


Figure 1.16: Simplified CRT monitor scan.

After a horizontal scan has been completed, the beam is moved to the next line during the horizontal retrace. This sequence continues until the last line, at which point the vertical retrace begins. The vertical retrace is similar to the horizontal retrace; the electron beam may be enabled through a small overscan area and then turned off (vertical blanking) as the beam returns to the top left corner of the screen.

If the vertical refresh is too slow, the display will flicker. Most people can detect flicker when the refresh rate drops below 60 Hz, and thus most displays use vertical refresh frequencies of about 60Hz (EGA) to 70Hz (VGA).

1.3.2 History of Video Adapters

The Monochrome Display Adapter (MDA) was released in 1981 with the IBM PC 5150. It offered two colors, allowing 80 columns by 25 lines of text. While not great, it was standard on every PC. Many other systems followed over the years, each of them preserving backward compatibility.

Name	Year Released	Memory	Max Resolution
MDA (Monochrome Display Adapter)	1981	4KiB	80x25 ¹⁴
Hercules	1982	64KiB	720x348
CGA (Color Graphics Adapter)	1981	16KiB	640x200
EGA (Enhanced Graphics Adapter)	1985	64KiB	640x350
VGA (Video Graphics Array)	1987	256KiB	640x480

Figure 1.17: Video interface history.

Each iteration added new features and by 1990 the predominant graphic system was EGA, although the VGA system was rapidly becoming the new standard. All video cards installed on PCs had to follow the standard set by IBM. The universality of that system was a double-edged sword. While developers had to program for only one graphic system, there was no escaping its shortcomings.

1.3.3 Introduction of EGA Video Card

IBM introduced the Enhanced Graphics Adapter (EGA) in 1985 as the successor of CGA. The standard card was shipped with only 64KiB video memory, but it had the option to expand the memory using the onboard graphics memory expansion card. Figure 1.19 shows the original IBM EGA card, a clunky beast full of discrete components. The memory consists out of TMS4416 RAM, a common memory chip for (home) computers around that period. Each chip contains 16KiB of 4-bits memory, so one needs two chips to end up having 16KiB of 8-bit memory and eight chips for 64KiB of 8-bit memory.

Trivia : Texas Instruments introduced the first 16KiB by 4-bits as TMS4416 in 1980¹⁵. Still it took until 1983-84 until they became widely available and lower priced than four TMS4116 chips (16KiB by 1-bit). However, at that time 64 KiB RAM was the way to go for new designs. Computers with only 16 KiB as base memory - and that's where TMS4416 would have been a cost saver - were already on the way out.

¹⁴Text mode only.

¹⁵<https://pdf1.alldatasheet.com/datasheet-pdf/view/103706/TI/TMS4416.html>

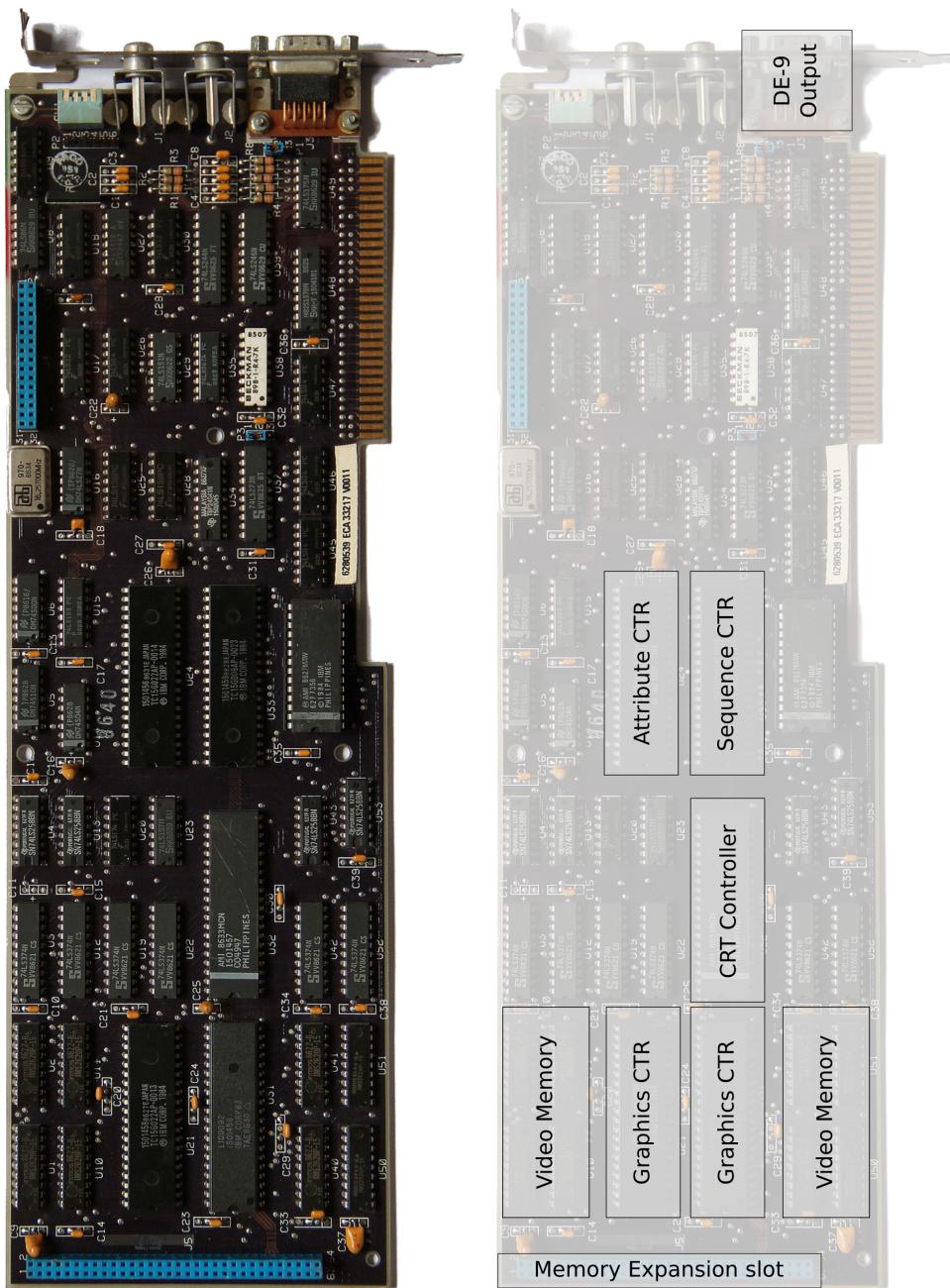


Figure 1.18: Original IBM EGA card

To add additional video memory to the IBM EGA card a Graphics Memory Expansion Card could be purchased. By default only the bottom row of memory was populated with chips, expanding the total EGA video memory to 128KiB. The expansion card provided DIP (dual in-line package) sockets for further memory expansion. Populating the DIP sockets with a Graphics Memory Module Kit adds two additional rows of 64KiB, bringing the EGA memory to its maximum of 256KiB.

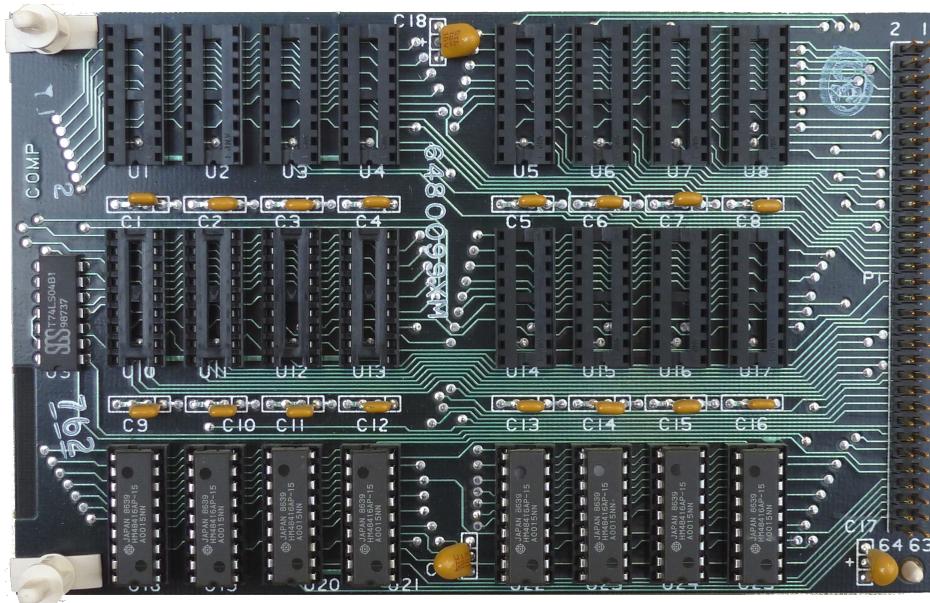


Figure 1.19: EGA Graphics Memory Expansion Card, bottom row populated with chips.

The EGA clones that started coming along in 1986-87 were based on integrated chipsets, and the vast majority of them came with the maximum of 256KiB on board. When Commander Keen came out, the headcount of EGA cards with less than 256KiB would've been practically negligible¹⁶.

Below an ATI EGA Wonder 800 (8-bit ISA). The eight chips on the left of the card form the VRAM where the framebuffers are stored.

¹⁶PC Tech Journal Oct 1986 (page 82-83) and PC Tech Journal Nov 1986 (page 148-149)



1.3.4 EGA Architecture

EGA can be summarized as three major systems made of input, storage, and output:

- The Graphic Controller and Sequence Controller controlling how EGA RAM is accessed (the CPU-VRAM interface)
- The framebuffer (the VRAM) made of four memory banks with 64KiB (rather than one bank of 256KiB).
- The CRT Controller and the Attribute Controller taking care of converting the palette-indexed framebuffer to RGB and then to digital TTL¹⁷ signal for display

Trivia : In the 1980's integrated video DACs¹⁸ were expensive and difficult to embed into custom chips. Most home computers with RGB output used TTL for digital output. With the introduction of VGA the DAC became the standard.

The most surprising part of the architecture is obviously the framebuffer. Why have four small fragmented banks instead of one big linear one?

The main reason was RAM latency and the need for minimum bandwidth. A CRT running at 60Hz and displaying 640x350 in 16 colors needs a pixel every $\frac{1}{640*350*60} = 74$ nanosecond. At this resolution, one pixel is encoded with 4 bits. Each nibble is translated to a RGB color via the TTL. So that means it requires one byte every 148 nano-seconds.

Unfortunately, RAM access latency was 200ns - not nearly fast enough¹⁹ to refresh the screen at 60hz, so the TTL would starve. If latency could not be reduced, the throughput could still be improved by reading from four banks at a time. Reading in parallel gave an amortized RAM latency of $200/4 = 50$ ns, which was fast enough.

Keep in mind that this architecture reduced the penalty of read operations, but plotting a pixel in the framebuffer with a write operation was still slow. Writing to the VRAM as little as possible was crucial to maintaining a decent framerate.

¹⁷Transistor Transistor Logic

¹⁸Digital to Analog Converter

¹⁹Computer Graphics: Principles and Practice 2nd Edition, page 168.

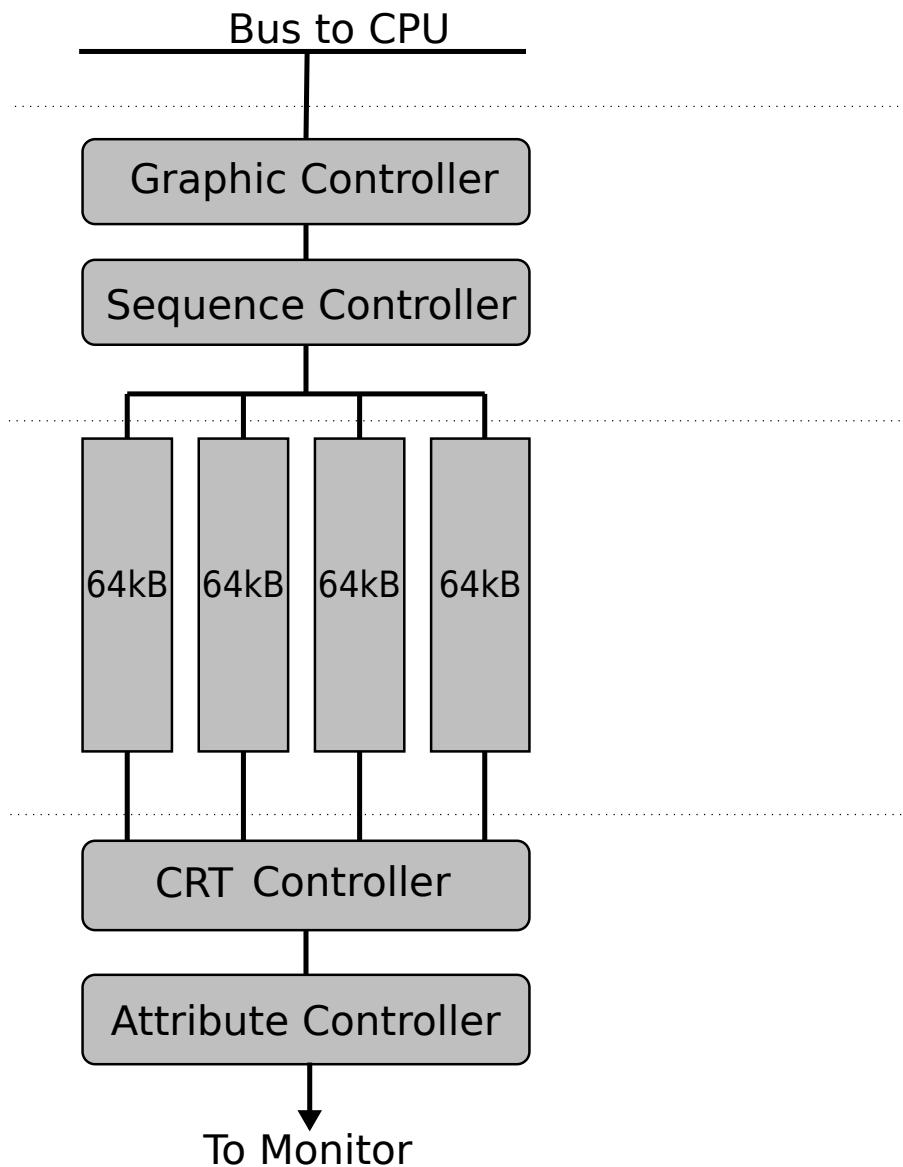


Figure 1.20: EGA Architecture.

1.3.5 EGA Planar Madness

Four memory banks grant enough throughput to reach high resolutions at 60Hz. This type of architecture is called "planar". Each plane is like a black-and-white image that stores

information about a single color. For EGA there are 4 planes, where combining one bit from each plane results in a color index. This color index is then via the attribute controller translated into a color to the screen. This layout is better explained with a drawing.

To write the color of the first pixel, a developer has to write the first bit of the first byte in plane 0, the second in plane 1, the third in plane 2 and the fourth in plane 3. The CRT Controller then reads 4 bytes at a time (one from each plane) and converts them via the Attribute Controller into 8 pixels on screen.

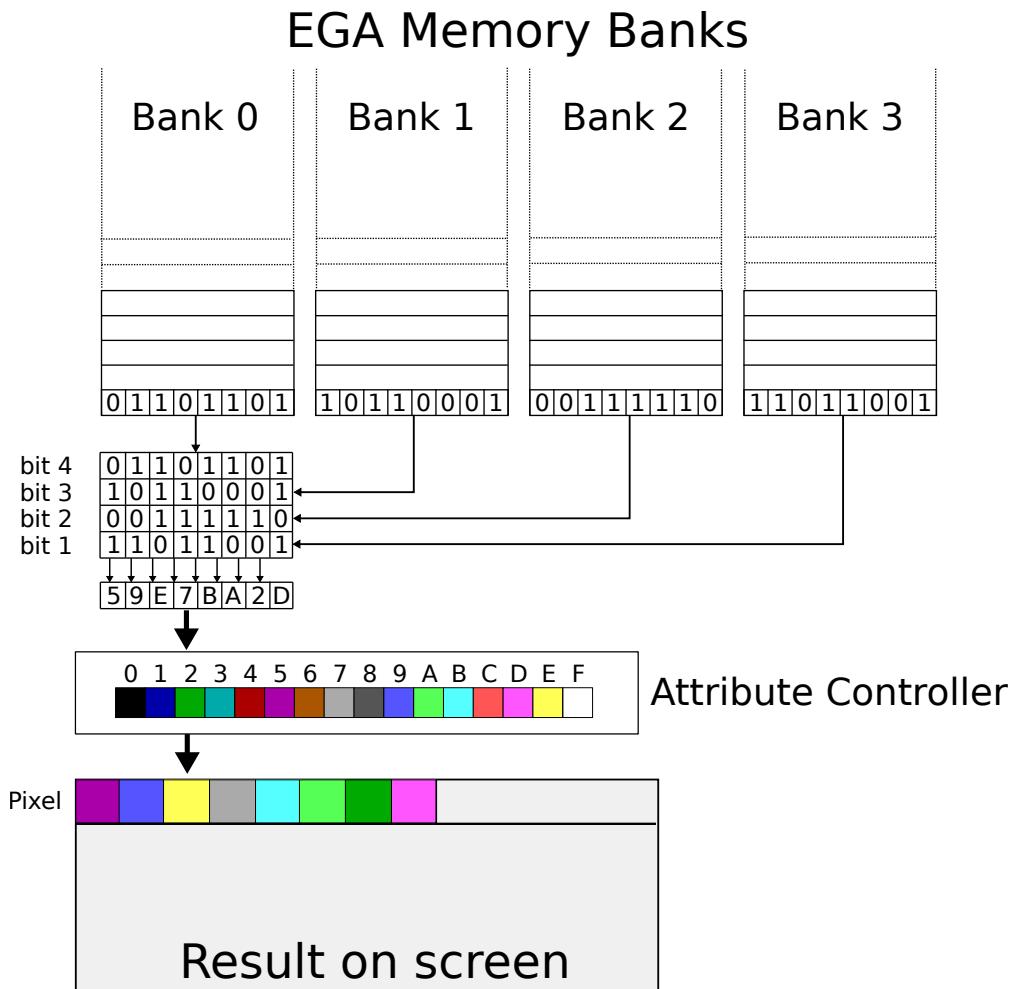


Figure 1.21: EGA mode 0Dh, how bank layout appears on screen.

1.3.6 EGA Modes

In order to configure this mess of planes and the controllers, 50 poorly documented internal registers must be set. Needless to say few programmers dove into the internals of the EGA.

Figure 1.20, which described the architecture, was actually deceptively simplified. Figure 1.22 shows how IBM's reference documentation explained the EGA. The maze of wire showcases well the actual complexity of the system.

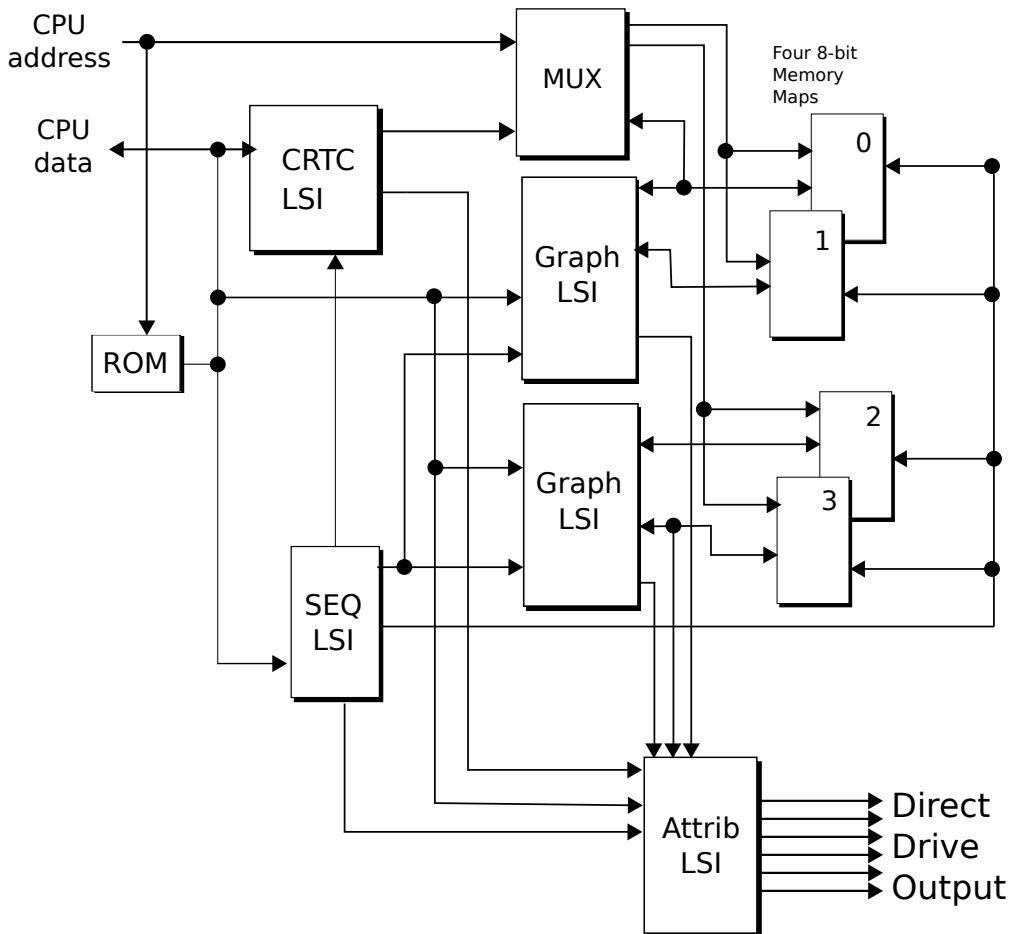


Figure 1.22: IBM's EGA Documentation.

To compensate for the complexity, IBM provided a routine to initialize all the registers via one BIOS call. One mode can be selected out of 12 available with an associated resolution, number of colors, and memory layout. The BIOS can be called to configure the EGA as follows:

Mode	Type	Format	Colors	RAM Mapping	Hz
00h	text	40x25	16 (monochrome)	B8000h	60
01h	text	40x25	16	B8000h	60
02h	text	80x25	16 (monochrome)	B8000h	60
03h	text	80x25	16	B8000h	60
04h	CGA Graphics	320x200	4	B8000h	60
05h	CGA Graphics	320x200	4 (monochrome)	B8000h	60
06h	CGA Graphics	640x200	2	B8000h	60
07h	MDA text	9x14	3 (monochrome)	B0000h	60
0Dh	EGA graphic	320x200	16	A0000h	60
0Eh	EGA graphic	640x200	16	A0000h	60
0Fh	EGA graphic	640x350	3	A0000h	60
10h	EGA graphic	640x350	16	A0000h	60

Figure 1.23: EGA Modes available.

Trivia : The Modes 08h-0Ah are reserved for PCjr (or Tandy Graphics Adapter) graphics modes, which offered 160x200 with 16 colors, 320x200 with 16 colors and 640x200 with 4 colors. Modes 0Bh and 0Ch are reserved for internal EGA BIOS.

To setup the EGA in Mode 0Dh using the BIOS is incredibly easy. It can be done with only two instructions:

```
_AX = 0xd ; AH=0 (Change video mode), AL=0Dh (Mode)
geninterrupt (0x10) ; Generate Video BIOS interrupt
```

The geninterrupt (0x10) instruction is a software interrupt caught by the BIOS routine in charge of graphic setup. It looks up the ax register, which can be set in the Borland Compiler by _AX, to setup all EGA registers with the corresponding mode.

1.3.7 EGA Programming: Memory Mapping

To write to the VRAM, the RAM's 1MiB address space maps 64KiB starting as indicated in figure 1.23. In mode 0Dh for example, the VRAM is mapped from 0xA0000 to 0xFFFF. One of the first questions to come to mind is "How can I access 256KiB of RAM with only

64KiB of address space?" The answer is "bank switching" as summarized in figure 1.24. Write and Read operations are routed based on a mask register indicating which bank should be read or written to.

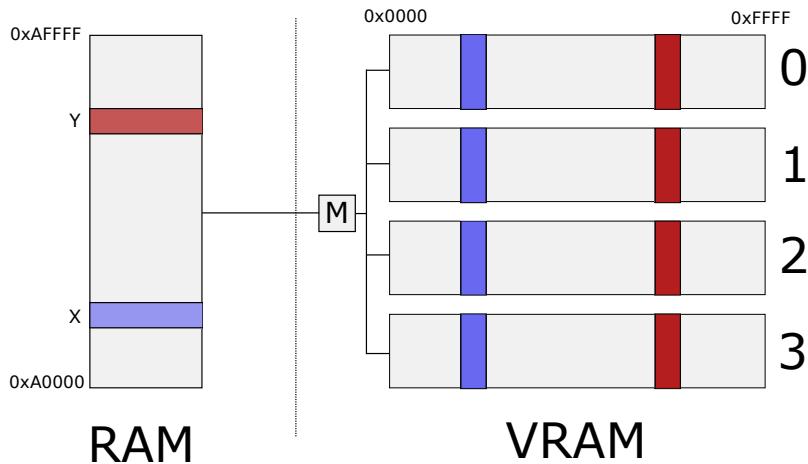


Figure 1.24: Mapping PC RAM to EGA VRAM banks.

1.3.8 EGA Color Palette

The EGA CRTC does not expect RGB values to generate pixels. Instead it is based on an index-based color palette system. Each pixel is a 4-bit index number, assigned to a color from the Attribute Controller. The default color palette are all 16 CGA colors, but it allows substitution of each of these colors with any one from a total of 64 colors.

When calculating the intended value in the 64-color EGA palette, the 6-bit number of the intended entry is of the form "rgbRGB" where a lowercase letter is the least significant bit of the channel intensity ($\frac{1}{3}$ color intensity) and an uppercase letter is the most significant bit of intensity ($\frac{2}{3}$ color intensity). The more intensity, the brighter the color is. For example, 02h will produce green, 10h will produce dim green and 12h will produce bright green. The color magenta is created by setting both "R" and "B", which is color code 05h. Each of the 16 color indexes could be reassigned to one color from the "rgbRGB" palette.

	00h	01h	02h	03h	04h	05h	06h	07h	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh
00h	Black	Dark Green	Cyan	Red	Purple	Yellow-Green	Grey	Dark Blue	Green	Blue	Magenta	Dark Magenta	Light Blue	Light Magenta	Light Green	Light Purple
10h	Dark Green	Blue	Light Green	Cyan	Brown	Yellow-Green	Light Green	Light Blue	Blue	Green	Light Cyan	Light Brown	Light Blue	Light Magenta	Light Green	Light Cyan
20h	Dark Red	Purple	Light Green	Red	Yellow	Yellow-Green	Light Pink	Dark Purple	Blue	Green	Light Blue	Red	Light Magenta	Light Orange	Light Pink	
30h	Dark Olive	Light Green	Orange	Yellow	Yellow	Yellow	Light Yellow	Dark Grey	Green	Light Green	Light Cyan	Red	Light Magenta	Light Orange	Light Yellow	

Figure 1.25: EGA "rgbRGB" color palette (64 values from 00h to 3Fh)

However, standard EGA monitors did not support use of the extended color palette in 200-line modes. Both CGA and EGA cards use a female nine-pin D-subminiature (DE-9) connector for output. The EGA monitor could only distinguish EGA and CGA cards based on the Vertical Sync signal, which is either 200- or 350-line mode. If the Vertical Sync is 350-line mode, the monitor switched to Mode 2 operations which supported the extended rgbRGB-color information²⁰. But in the 200-line mode, the monitor cannot distinguish between being connected to a CGA or an EGA card.

The CGA color output is based on the form "RGBI", where the 'I' stands for Intensity and adds brightness to the RGB color. Compared to CGA, EGA redefines some pins of the DE-9 connector to carry the extended rgbRGB-color information. If the monitor were connected to a CGA card, these pins would not carry valid color information, and the screen might be garbled if the monitor were to interpret them as such.

Pin	Mode 2: EGA mode (rgbRGB)	Mode 1: CGA mode (RGBI)
1	Ground	Shield Ground
2	Secondary Red (Intensity)	Signal Ground
3	Primary Red	Red
4	Primary Green	Green
5	Primary Blue	Blue
6	Secondary Green (Intensity)	Intensity
7	Secondary Blue (Intensity)	Reserved
8	Horizontal Sync	Horizontal Sync
9	Vertical Sync	Vertical Sync

Figure 1.26: EGA and CGA DE-9 connector pin signals.

²⁰IBM Enhanced Color Display documentation.

Suppose one assigns the color brown (rgbRGB is 010100b) to one of the color indexes, the resulting color on the CGA pin assignment is light red; The secondary green pin ("r" in rgbRGB) is mapped to the Intensity pin in CGA mode, which results to the color red with intensity and not the expected brown color.

For this reason, EGA monitors will use the CGA pin assignment (mode 1) in 200-line modes so the monitor can also be used with a CGA card and vice versa. Therefore, the EGA card is fully backwards compatible with a standard CGA monitor. Thereby it is able to show all 16 CGA (RGBI-)colors simultaneously, instead of only 4 colors when using a CGA card.

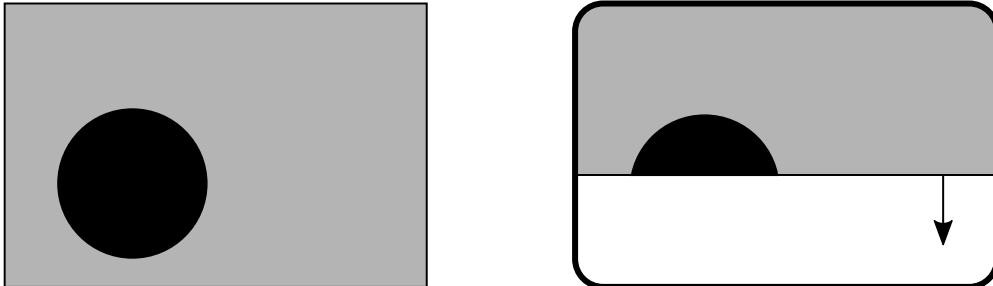
Index Number	Color	rgbRGB	RGBI
00h	Black	000000b	0000b
01h	Blue	000001b	0010b
02h	Green	000010b	0100b
03h	Cyan	000011b	0110b
04h	Red	000100b	1000b
05h	Magenta	000101b	1010b
06h	Brown	010100b	1100b
07h	Light grey	000111b	1110b
08h	Dark grey	111000b	0001b
09h	Bright blue	111001b	0011b
0Ah	Bright green	111010b	0101b
0Bh	Bright cyan	111011b	0111b
0Ch	Bright red	111100b	1001b
0Dh	Bright magenta	111101b	1011b
0Eh	Yellow	111110b	1101b
0Fh	White	111111b	1111b

Figure 1.27: Default EGA 16-color palette

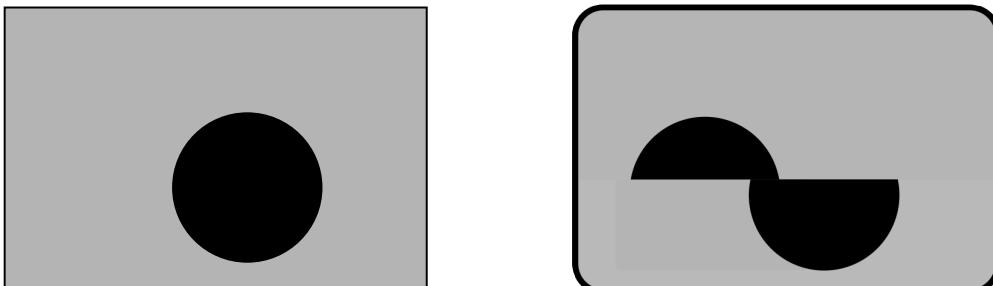
1.3.9 The Importance of Double-Buffering

Double buffering has been mentioned often while describing the hardware, but so far we have not reviewed why it is paramount to achieving smooth animation. With only one buffer

the software has to work at exactly the frequency of the CRT (60Hz). Otherwise a phenomenon known as "tearing" appears. Let's take the example of an animation rendering a circle moving from the left to the right:



In this example the CPU has finished writing the framebuffer (on the left) and the CRT's (on the right) electron beam has started to scan it onto the screen. At this point in time the electron beam has scanned half the framebuffer, so the circle has been partially drawn on the screen.



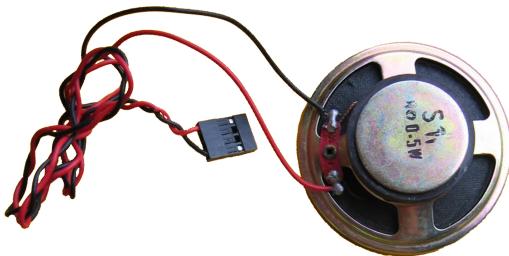
If the CPU is faster than the frequency of the CRT (60Hz), it can write the framebuffer again, before the scan is completed. This is what happened here. The next frame was drawn with the circle moved to the right. The electron beam did not know that and kept on scanning the framebuffer. The result on screen is now a composite of two frames. It looks like two frames were torn and taped back together. Hence the name "tearing".

With two buffers (a.k.a double buffering) the CPU can start writing in the second framebuffer without messing with the framebuffer being scanned to the screen²¹. No more tearing! A full screen of 320x200 pixels with 16 colors requires $320 \times 200 \times 4\text{-bits} = 32\text{KB}$ of VRAM (spread over 4 planes, each 8KB), so there is plenty of video memory left to keep multiple screens in an EGA card with 256KiB memory.

²¹Now the CPU speed is capped by the CRT refresh rate. Triple buffering can solve this at the price of frame latency.

1.4 Audio

For the first 5-6 years of the IBM PC and its compatibles, their audio output came from nothing more than a simple loudspeaker with a tone generator. For business, this was acceptable - even preferable, since a PC in an office environment really shouldn't be a distraction to others! The loudspeaker, commonly known as a "PC Speaker", was capable of generating a square wave via 2 levels of output.



1.4.1 History of Sound Cards

The introduction of real game music and sounds on the PC started with Sierra back in 1988. They prepared to change all this by creating games that contained serious, high quality musical compositions drawing on add-on hardware. *King's Quest IV* was the first commercially released game for IBM PC compatibles to support sound cards. In addition to the familiar PC speaker and Tandy 1000, it could utilize

- Roland MT-32
- IBM Music Feature Card
- AdLib

Sierra struck a deal with two companies, Roland and AdLib, where Sierra would also become a reseller for these soundcards.

The Roland MT-32 was the higher end of these music devices. In today's terminology, it would be labeled a "Wavetable Synthesizer". A wavetable synthesizer usually implies that real instrument sounds are recorded into the hardware of the device. This device can then manipulate them to play them back at the various notes you need. The MT-32 had the ability to manipulate parts of its built in sounds using something called "Linear Arithmetic (LA)" synthesis. It was a very good device that can rival even today's sound cards. To connect the MT-32 to a PC required, what Roland called an MPU-401²², in one of the PC's

²²Midi Processing Unit-401

expansion slots. Sierra sold The MT-32 with a necessary MPU-401 interface for \$550. The high price prevented it from dominating the end-user market of players.



Figure 1.28: Roland MT-32 synthesizer box.

The IBM Music Feature Card was launched in March 1987 as a collaboration between IBM and Yamaha. Essentially the Music Feature Card was a synthesizer installed on an 8-bit expansion card²³. The Music Feature Card had 8 FM voices, controllable via 4 frequency operators. It came with over 300 high-quality synthesized instruments on-board, and it was actually possible to have two Music Feature Cards in a single PC to get 16 voices! With a tag price of \$495 it was just like the Roland MT-32 an expensive card, and its audience was primarily business users.

²³Roland released the LACP-I in 1989, which basically was similar to the Music Feature Card: a MT-32-compatible Roland synthesizer with a MPU-401 unit, integrated onto one single full-length 8-bit ISA card.



Figure 1.29: IBM Music Feature Card.

Trivia : The IBM Music Feature Card was the very first general purpose "sound card" for the PC, beating AdLib by several months.

1.4.2 AdLib

AdLib was the other company, beside Roland, that struck a deal with Sierra as a reseller. The company was founded in 1987 by Martin Prevel, a former professor of music from Quebec. The AdLib soundcard used a technology called FM synthesis. The technology is based on the idea of generating superimposing waveforms to create a sound. This technology was much less expensive than Roland's Wavetable technology.

The AdLib card was built around the Yamaha YM3812, also known as the OPL2 chip, and could produce either 9 sound channels or 6 sound channels plus 5 hit instruments simultaneously using Frequency Modulation (FM). Ideally, if you have enough generators and can fine tune the waveforms well enough, you can create a realistic sound. However, to reach this ideal, you need lots of skilled people, lots of money for equipment, and lots of time to develop. The reality is, this is a cheap PC soundcard offering a better alternative than the PC Speaker. Thus, FM synthesis sounded very artificial. Still, this was a great improvement over the PC Speaker. With a price tag of \$219.99, it was much cheaper than the Roland MT-32 and IBM Music Feature Card, and soon ruled at the top of the early PC sound card market.

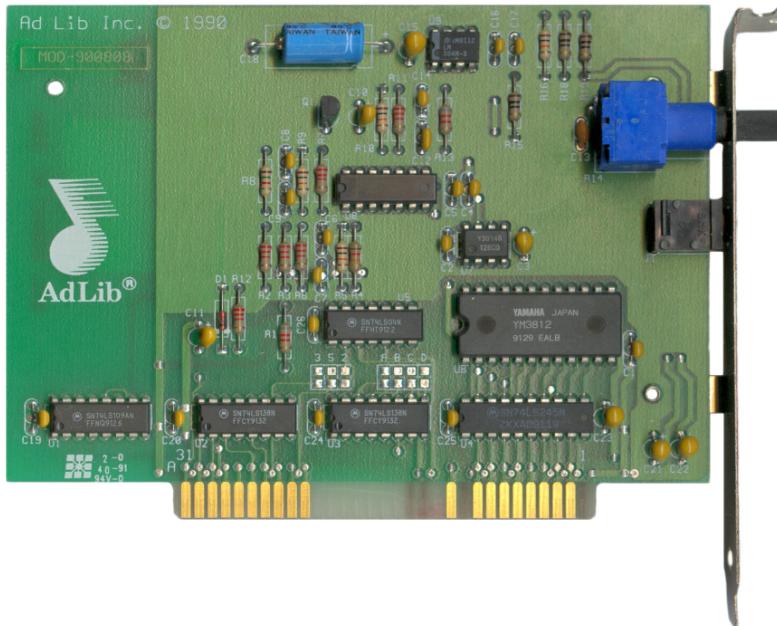


Figure 1.30: An AdLib sound card. Notice the big YM3812 chip and the 8-bit ISA connector.

The AdLib card dominated the PC market for almost three years. In 1989, Creative Labs released the competing SoundBlaster, which quickly dominated over AdLib. To compete with SoundBlaster, AdLib planned a new 12-bit stereo sound card called AdLib Gold. Sadly, due to AdLib's dependence on Yamaha who suffered long delays introducing their latest multimedia chipset, their new product, AdLib Gold PC-1000, was never to see the light of day under AdLib's management. Unable to remain solvent, AdLib closed its doors on 1st May 1992.

1.4.3 SoundBlaster

Creative Labs, the company behind the SoundBlaster, didn't enter the sound card industry until 1987 with the introduction of their Creative Music System (C/MS). From inception in 1981, they were a computer repair shop in Singapore. Creative Music System, or "Game Blaster" as it was renamed a year later, was an FM synthesizer card similar to AdLib. Based on the Philips SAA1099 chip which was essentially a square-wave generator, it sounded much like twelve simultaneous PC speakers would have, except for each channel having amplitude control. The card did not sell well.

The original Sound Blaster v1.0 and v1.5 were released in 1989 as the successor to their Game Blaster. Not only was it equipped with the OPL2 chip, providing 100% compatibility with AdLib music playback, but it was also technologically superior with a DSP²⁴ allowing PCM playback (digitized sounds) at 8 bits per sample and up to 22.05kHz sampling rate. The card also came with a DA-15 port allowing joystick connection. Most importantly, the SoundBlaster was \$90 cheaper than the AdLib.

Trivia: Creative Labs boasts in their own advert AdLib compatibility and even uses an image of AdLib Inc's product²⁵! On the other hand they sued every company that tried to market a 'Sound Blaster compatible' product for using the name of their product.



Figure 1.31: A SoundBlaster 1.5, model CT1320B

Figure 1.31 is the SoundBlaster model CT1320B. Notice the OPL2 chip (labeled FM1312) and the CT1321 DSP on the middle top. In the middle of the card are the two CMS-301 chips, to ensure backwards compatibility with the Creative Music System.

Trivia : The CT1320B model (Sound Blaster 1.5) was a cost-cutting measure. Having recognised that C/MS was unpopular, they replaced the two C/MS chips with sockets. You could still purchase the C/MS chips for \$29.95 if you wished and install them into these sockets.

²⁴An Intel MCS-51 "Digital Sound Processor", not "Digital Signal Processor".

²⁵Advert p20 in Compute!, April 1990.

1.4.4 Disney Sound Source

The parallel port is nothing more than a way to pass out 8 bits simultaneously. Each PC had this port, and the conversion of this kind of data to an analogue signal is pretty simple and can either be achieved by using a DAC²⁶ or simply a set of resistors. Hence the only thing missing was the interface, which a company named Covox delivered with their "Speech Thing" for only \$70.

The difficulty about this sound device was to get exact timing. All bytes are converted the moment they reach the parallel port. In many ways it works pretty similar to the PC-speaker, only its connected to the parallel port. The CPU was in the center of all this, and it required lots of CPU resources to timely move information from RAM to the parallel port. This left limited CPU time for the game or program, in a time period where the CPU already was the bottleneck.



Figure 1.32: The Disney Sound Source speaker box (DAC not shown).

The Disney Sound Source came out around 1990 and fixed multiple issues: It contained a small buffer, so even an unsteady data stream resulted in a good audio signal. It also allowed to pass through the printer port, so there's no need to switch cables to print. And at \$14 it was dirt cheap! It came with its own amplifier, which ran with a 9 Volt battery.

²⁶Digital to Analogue Converter.

It would have made programmers and customers happy if not for one serious limitation: The buffer was fixed at a 7,000Hz, which is about the quality you get from an old fashioned phone. This was still enough to produce pleasant music, but fell short for sound effects when compared to AdLib and Sound Blaster.

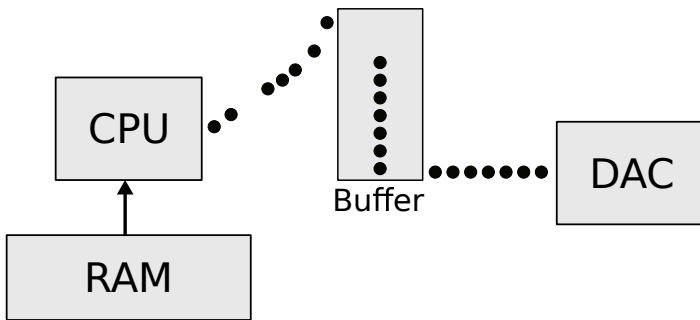


Figure 1.33: Disney Sound Source with buffer, running at fixed 7,000Hz.

1.5 Floppy Disk Drive

In the time before the internet, a floppy disk was the main medium to share and distribute software and data. The original XT systems were equipped with 5 $\frac{1}{4}$ -inch floppy disk with a capacity of 360Kb. In 1984, IBM introduced with its PC AT the 1.2 MB dual-sided 5 $\frac{1}{4}$ -inch floppy disk, but it never became very popular. IBM started using the 720 KB double density 3 $\frac{1}{2}$ -inch floppy disk in 1986 and the 1.44 MB high-density version in 1987. The advantages of the 3 $\frac{1}{2}$ -inch disk were its higher capacity, its smaller physical size, and its rigid case which provided better protection from dirt and other environmental risks. By the mid-1990s, 5 $\frac{1}{4}$ -inch drives had virtually disappeared, as the 3 $\frac{1}{2}$ -inch disk became the predominant floppy disk.

Trivia : An USB stick of 128GB contains more than 91K high-density 3 $\frac{1}{2}$ -inch (1.44MB) floppy disks.

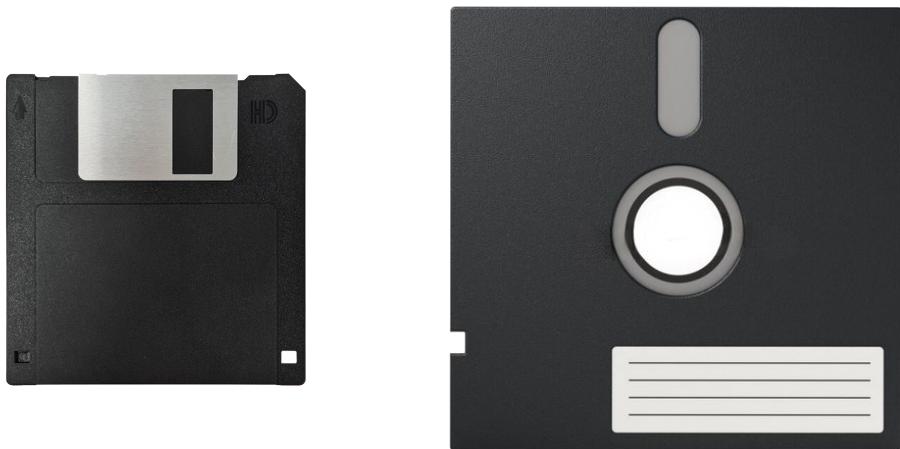


Figure 1.34: 3 1/2-inch and 5 1/4-inch floppy disk.

A floppy disk is essentially a very flexible piece (hence the term floppy disk) of plastic coated on both sides in a magnetic material. This 'disk' of plastic is contained within a protective envelope or hard plastic case, which is then inserted into the drive and automatically locked onto a spindle. It is then rotated at a constant speed, 360 rpm for standard PC floppy drives. A head assembly consisting of two magnetic read/write heads, one in contact with the upper surface and one in contact with the lower surface of the disk, may be moved in discrete steps across the disk and read the data from the disk.

The floppy disk is controlled via the Floppy Disk Controller (FDC), a typical read operation from the floppy disk contains the following steps:

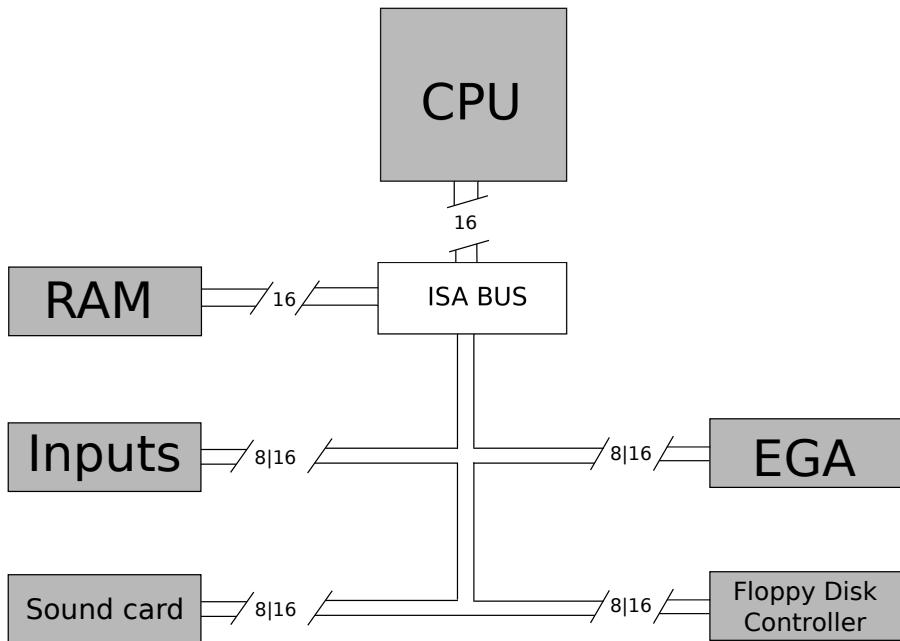
- Turn the disk motor on. When you turn a floppy drive motor on, it takes quite a few milliseconds to "spin up", to reach the (stabilized) speed needed for data transfer.
- Perform seek operation, which moves the head to the correct location for reading the data.
- Read the data from the floppy disk and store the data via the FDC to RAM memory.
- Turn the disk motor off.

The controller waited a few seconds before turning off the motor. The reason to leave the motor on for a few seconds is that the controller may not know if there is a queue of sector reads or writes that are going to be executed next. If there are going to be more drive accesses immediately, they won't need to wait for the motor to spin up again.

1.6 Bus

Although developers had no control over them, it is still worth mentioning how these components were connected to each other.

The ISA²⁷ bus connects the CPU to all devices, including RAM. It was almost 10 years old in 1990 but still used universally in PCs. The data path to the RAM is 16 bits wide for 286 machines. It runs at the same frequency as the CPU.



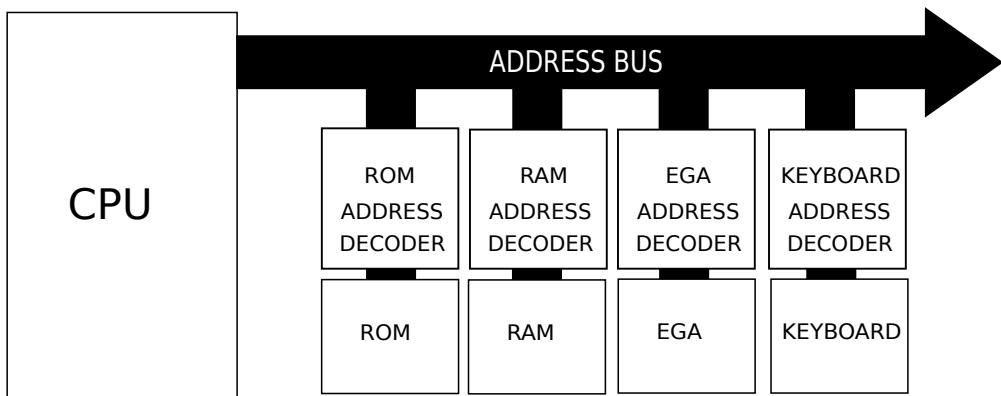
²⁷Industry Standard Architecture.

The rest of the bus connecting to everything that is not the RAM can be either:

- 8 bits wide at 4.77 MHz for 19.1 Mbit/s
- 16 bits wide at 8.33MHz for 66.7 Mbit/s²⁸.

It is also backward compatible and an 8-bit ISA card can be plugged into a 16-bit ISA bus.

Trivia : On ISA all devices are connected to the bus at all times and listen on the bus address lane. Each device features an "address decoder" to detect if it should reply to a bus request. This is how the EGA RAM is "mapped" in RAM. The EGA card "address decoder" filters out everything that is not within A0000h and AFFFFh. Accordingly, the RAM disregards any request that is within the range [A0000h - AFFFFh].



1.7 Summary

To say a PC was difficult to program for games would be an understatement. It was a nightmare. The CPU was good at doing the wrong thing, the best graphic interface didn't allow double buffering, and the memory model only allowed 1 MiB with an address composed of two separate 16-bit registers. Last, but not least, the default sound system could only produce square waves.

Yet despite all these unfavorable conditions, teams of developers gathered to tame the beast and unleash its power to gamers. One of these called themselves *Ideas From the Deep*²⁹.

²⁸https://en.wikipedia.org/wiki/List_of_device_bit_rates.

²⁹They originally called themselves Ideas From the Deep but then decided to shorten it to simply id, which stands for "in demand", and is pronounced as in "did" or "kid." The name also refers to id, the part of the brain that behaves by the pleasure principle in Freudian psychology.

