Know Your Personal Computer

6. Memory Organization

S K Ghoshal

This article briefly describes the memory organization of the base architecture by illustrating it with a map and explains how to expand or extend the memory.

Introduction

Merely adding one more bit to the address bus of a CPU doubles the size of the virtual address space. It is harder to provide the extra memory in the hardware. CPUs are developed with a long-term view. So when their design is new, they can address a much larger size of memory than what existsphysically on any first-generation motherboard featuring that CPU. And once the CPU has proven itself and its architecture becomes old and trusted, the opposite happens, viz. the physical memory becomes larger than the size of the virtual address space of the CPU. This happens in the case of any CPU in any computer architecture that has stood the test of time and adapted itself to the market forces. This has happened for instance with the IBM PC architecture. By the time the memory and system integration technology caught up, the CPU got obsolete. When a new CPU came in, the same process occurred. Thus on the IBM PC architecture, the size of the virtual address space can be either larger or smaller than the size of the physical address space. The Finite State Machines (FSMs) described in Part 2 of this series take part in this mapping process by transforming the address generated by the CPU into another address that actually reaches the memory (See *Figure 1*). The Bus-controller chips can alter the speed, width, size and protocol of the memory chips that are accessed through it by implementing FSMs, as explained in Part 2. Thus the way the address translation takes place is programmable in a modern motherboard. By executing a program

Siddhartha Kumar Ghoshal works with whatever goes on inside parallel computers. That includes hardware, system software, algorithms and applications. From his early childhood he has designed and built electronic gadgets. One of the most recent ones is a sixteen processor parallel computer with IBM PC motherboards.

The previous articles of this series were:
1. Introduction to computers, January 1996.
2.The personal computer hardware, February 1996.
3.The personal computer system software, April 1996.
4. The CPU base architecture, July 1996.
5. The CPU base instruction set and assembly language programming, November 1996.

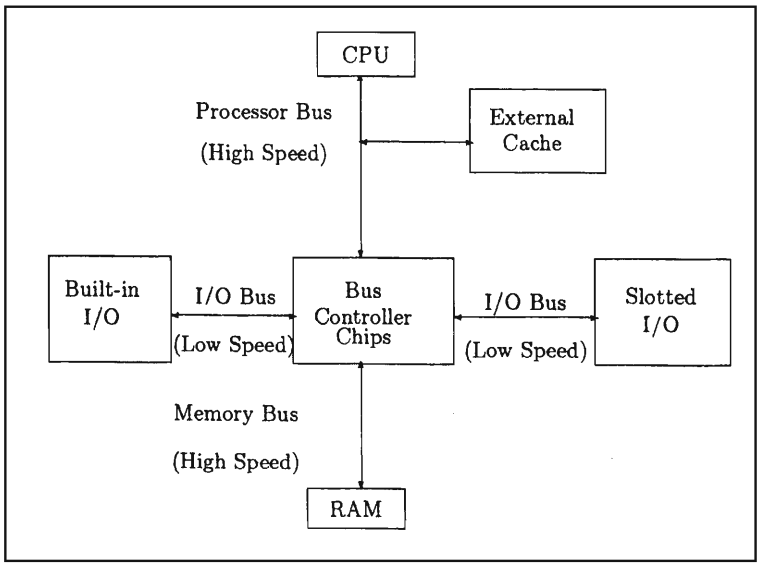

Figure 1 The CPU interacts with FSMs to address memory.

on a given motherboard, one can change its board-level architecture. The board level architecture of the IBM PC can emulate the architecture of any of its commercially successful predecessors. To come to this level of sophistication, it must incorporate almost all the brilliant ideas of computer architecture and system integration technology that have been devised thus far. Thus a survey of this evolution should be done with care and in detail as it has considerable academic significance. I shall try to do that in this article.

The Memory Map of the Primitive IBM PC

A diagram that describes the addresses of different types of memory chips and other devices that occupy different locations in the memory address space is called the *memory map*. A memory map can have holes, that is regions within the memory address space where nothing is mapped. *Figure 2* is the memory map of the original design of the IBM PC. The 8088 microprocessor can address 1MB of memory. In 1981, when the PC was designed motherboards barely had 64KB of memory. and most programs on PCs were tiny by today's standard and people were happy with what they could do with their PCs.

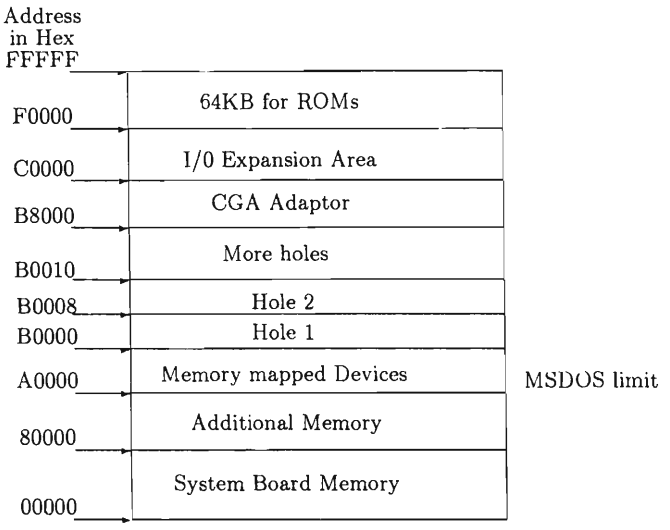

Figure 2 Memory map of the IBM PC with CGA adaptor.

Small is Beautiful

A Pascal compiler and text editor put together in 1983 could run using less than 32KB main memory, came in a 5.25 inch 360KB floppy as a file with size about 35KB. It executed directly from that floppy. A modern (1995) C++ compiler from the same company takes 28 floppies (each floppy of size 3.5 inch and 1.2MB capacity) to install. A minimum of 640KB main memory is required to run it and a hard disk is strongly recommended for any meaningful use of the compiler. Of course the modern compiler offers you a bewildering choice of libraries, options and tools that you get used to. Every feature in that range can be invoked by a single keystroke. And before you can blink, you have finished using it. In the past, there were long delays while the Pascal compiler accessed the disk. Sometimes one even had to manually change the disks. Nowadays nobody will tolerate such a system. One does not mind wasting memory as it is available in plenty. One wants power and speed. This philosophy guides the use of memory on IBM PCs and drives designers to provide larger and faster memories. Also in 1983, the executable programs produced by the compiler in translating programs had sizes around a few kilobytes. Today one habitually writes programs that compile into executable codes that occupy hundreds of kilobytes. So the end-user and application programmers certainly have a role in making programs fat and flabby. The IBM PC memory subsystems architect is merely designing large and fast memories that you may want on your PC and are willing to pay for.

Soon they wanted to port programs from other types of computers (e.g. minicomputers, mainframes and supercomputers) onto the IBM PC. To their surprise, they could port it successfully on the PC platform and use it effectively. And to IBM's surprise, the PC architecture ran out of memory in less than two years after the PC was introduced.

The initial rate of growth of the PC architectural capabilities and range of uses exceeded even the most optimistic projections by IBM and other microcomputer professionals many times over. MSDOS, when it was first designed could use only 640KB of main memory as that was all the PC architecture could provide. Some memory mapped devices began at the hexadecimal address A0000 and that meant the PC architecture could only have 640KB of RAMs into which the operating system and user programs could be kept (*Figure 2*). Wherever there is no device, there is a hole. Back in 1981 people thought that it would take decades to fill up 640KB. Thus MSDOS and the PC design were standardized and frozen. When in two years more memory was needed, *expanded memory* was devised to provide that. There are many techniques for expanding memory. Each technique has a hardware and a software component. Some of them are non-standard/ patented/ secret/ trade-marked/ copyright-protected. Let me explain the basic

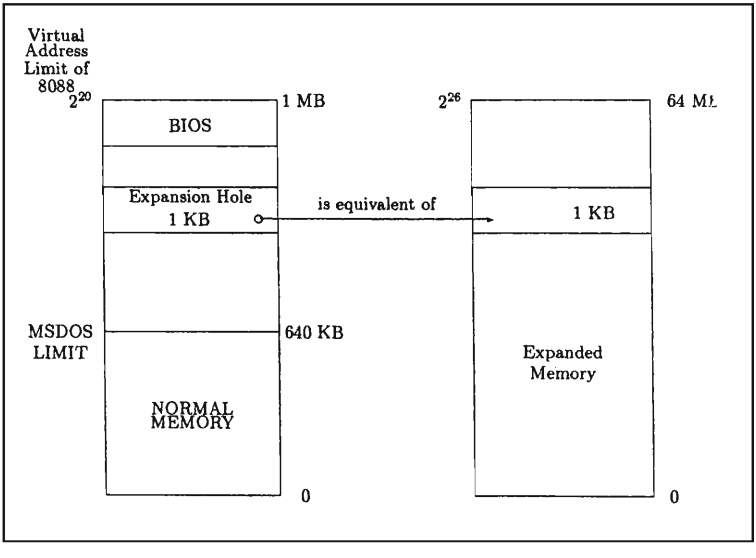

Figure 3 Memory map of Expanded Memory.

Do a Better Design

When the first versions of expanded memory were designed, one had to physically wire up AND gates and NOT gates. So the designs were error-prone. The design procedure was long, manpower-intensive and tedious. The resulting decoders were not flexible. They could not be configured as per the size and location of holes. The PC had to adapt to the expanded memory rather than the other way round. Now one just has to write down Equation 4 and a Computer-Aided Design Tool will automatically come up with the decoder design. Some tools will even accept Equation 1. One can thereafter implement the design on any suitable programmable logic device. Thus with these modern techniques, one can formulate designs better than any of the standards of expanding memory developed earlier.

where the hole will map onto the expanded memory. These 16 bits can easily come from two 8-bit I/O ports whose outputs are latched. I have called them *Base Ports* in *Figure 4*, as they set the base address of the image of the hole inside the expanded memory. When the hole has to be re-mapped, the CPU has to write a new value into the base port using the OUT instruction. Thus, from the guidelines and examples given in Part 5 of this series, one can easily write C–callable programs that map the expanded memory and act as an expanded memory driver. Using these programs in turn, one can write application programs in C that will effectively use the expanded memory that we have just designed.

The lower-most 10 bits are unchanged as they are fed to the expanded memory module, as seen from *Figure 4*. So one can freely write application programs which map data objects into the hole and manipulate them using instructions coded in a high-level language like C. Physically, memory objects inside the huge expanded memory will be modified and the C compiler need not be even aware of this. The great advantages of expanded memory are:

- Even an 8088 on an old generation motherboard running an old version of MSDOS can host an application program that effectively uses the expanded memory.

We Have to Live With Expanded Memory

The job of an expanded memory driver should ideally be to provide a mechanism, *and not a policy,* to access more physical memory than what the size of the CPU's virtual address space permits. However the programmer had to accept the policy when she badly needed the mechanism. And the policy dictated how programs are developed and sold on the PC platform. Even when a better mechanism called *extended memory* arrived, the old policy remained. Thus we must keep supporting expanded memory on the PC platform. All modern motherboards can emulate expanded memory, even though they only have extended memory physically. All the software that need expanded memory work correctly on these motherboards. Programming the FSMs appropriately makes extended memory behave like expanded memory.

- The compiler need not change. It can keep generating codes which at runtime refer to virtual addresses below 1MB. We can address much larger sizes of memory, as we just did in our design.

- Everything that worked on the PC before we added the expanded memory hardware goes on as before. Until one deliberately and explicitly addresses the hole, our expanded memory hardware will not interfere with the work. Nor does one have to reconfigure the PC in any way to accommodate the design.

The disadvantage of expanded memory is that whatever has to be manipulated by executing instructions on the CPU must fit within the size of the hole. That severely restricts the power of high-level languages to modify complex data objects residing in memory. One way to work around this difficulty is to convert programs, (that manipulate large data objects) brought in from regular mainframes or supercomputers with regular memory architectures, into those that have fragmented data structures. Each fragment fits the hole. One fragment is accessed at a time. After that the hole is re-mapped to cover another fragment. The composite huge data structure remains in the physical memory but cannot be manipulated or even seen by any CPU instruction. In fact some expanded memory drivers allow only a mechanism of access that resembles the handle of an open file. Using these drivers one can only access the object that the handle is currently pointing to, copy it into a regular memory area (below 640KB), manipulate it by CPU instructions and write it back using the handle. Many people and organizations had specialized in writing programs of this type although they were tedious, manpower-intensive and error-prone, simply because there was no alternative.

When Intel 80X86 microprocessors which could address beyond 1MB came up, motherboards that had more than 1MB memory on board were quickly designed. This memory could be addressed in a straightforward way. The CPU has N address lines where $N > 20$. These N bits are decoded and used to

A20 Gating

MSDOS has a *bug*. It uses an incorrect algorithm to compute the addresses of many memory objects that it must manipulate in order to do its own work. The algorithm actually yields a 21 bit address. So long as MSDOS worked only on 8088s nobody even knew of this bug and MSDOS was standardized during that period. The 8088 had only 20 address bits. The 21st bit got chopped internally. That was a wrong interpretation, but it did not matter then. What came out of the 8088 was the lower-most significant 20 bits which accessed objects. Thus when 80X86 microprocessors could address beyond 1MB, motherboards featuring such microprocessors with extended memory correctly interpreted the wrong algorithm to produce 21 bit addresses. These addressed locations in the extended memory, and not where MSDOS expected them to address and where MSDOS objects are kept. So MSDOS did not work. To make it work, the motherboard designers had to put additional on-board hardware to prevent the 21st address bit from reaching the physical memory. This mechanism is called A20 gating. You must keep it off to run MSDOS. And remember to turn it on before you try to access extended memory. A20 gating does not interfere with the working of expanded memory.

access a 2^N byte contiguous stretch of memory. This scheme is called the *extended memory*. An Intel 80386 CPU has 32 address bits. So it can have up to 4GB (2^{30} = 1G) of extended memory which is quite large by today's standards. And if you think I am thinking like the IBM PC designers as they thought in 1981, rest assured. Microprocessors with 64 address bits are already commercially available. So we need not expand the 4GB virtual address space. When the need arises, we will just extend it into a true 64-bit architecture. That process has already started in the workstation platforms. All vendors are changing over to true 64-bit architectures.

A true N–bit architecture conforms to the von Neumann model. With an N–-bit pointer you modify any object that lies in a virtual address space which is 2^N bytes wide. C programmers, compiler writers and operating system implementers love this model. The architecture does not interfere with observing and manipulating an object in memory. Thus an 80386-based motherboard with extended memory is a true 32-bit architecture, whereas an 80386-based motherboard with expanded memory is not. In the future we may discuss Unix and other software systems that work on true 32-bit PC platforms. If you have any comments please feel free to write to me.

Suggested Reading

- ***The MSDOS Encyclopedia*. Microsoft Press, 1988.**
- **W I Fletcher. *An Engineering Approach to Digital Design*. Prentice Hall of India, 1996.**

Address for Correspondence
S K Ghoshal
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore 560 012, India
email:
ghoshal@serc.iisc.ernet.in
Fax: (080) 334 1683