Know Your Personal Computer

4. The CPU Base-Architecture

S K Ghoshal

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.

This article surveys the CPU chips of the IBM Personal Computer using the Intel 80X86 family as the base.

Introduction

Intel 80X86 CPUs are used in the IBM PC. 80X86 means all CPUs in the family - (or should we call it a dynasty?) - the 8086 (and that includes its shorter external data-bus width cousin 8088), 80186 (and its cousin 80188), 80286, 80386 (and its cousin 80386 SX), 80486 (including the cousin 80486 SX) and the Pentium. Each does what its predecessor did. It does them faster and adds some more capabilities. This is called *upward compatibility*. The cousins are cheaper although slower and it is easier to build motherboards with them. We will first discuss the base architecture that runs across the dynasty. 80X86 is a complex instruction set CPU (CISC CPU).

Each 80X86 instruction typically has two operands, *source* and *destination*, even though the destination operand initially contains one of the inputs to the operation. The format of an instruction is 'Operation-name *destination*, *source*'. The instruction operates on the operands and then overwrites the *destination* operand with the result. As an example, the instruction 'XOR AX, BX' performs an exclusive-OR of the contents of the CPU registers whose names are AX and BX and overwrites AX with the result. The 80X86 allows all types of instructions except those in which both operands are in memory, as shown in the margin.

Destination	Source
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

There are 14 registers in the 8086: Eight general-purpose

that is unaffected by the baryonic density in the universe. What is that route?

In the QSSC the primary particle created has the Planck mass

$$m_P = \sqrt{ch/G}$$

which is about 10^{-5} gram. This particle is short lived and decays into baryons of high energy. The particle theory of baryons tells us that these are distributed in octets of which only two, the neutron and the proton survive. These form helium nuclei while the remaining six decay to protons, i.e., hydrogen nuclei. Thus the mass abundance of helium is nearly 25%. More accurate calculation reduces this to between 22% to 23% while producing other light nuclei like deuterium, lithium etc. and some metals.

To Sum Up

Thus the QSSC provides, prima facie, viable scenarios for the present observed features of the universe. Yet it differs radically from the hot big bang scenario. Can we distinguish between the two theories by suitable tests?

Notice that the QSSC has contracting as well as expanding phases of the universe. So there is a possibility of detecting faint *blue-shifted* galaxies. Then, the QSSC predicts the dark matter to be largely baryonic, being made of stellar remnants. It also predicts the existence of stars of low mass (half the Sun's mass) from the previous cycle that are just about branching off as red giants. Such stars would be 40-50 billion years old.

None of these possibilities are allowed by the standard hot big bang cosmology. And so here are ways of distinguishing between the two cosmologies. Only then will we be able to assert whether the universe had a definite origin or whether it has been going on for ever.

Suggested Reading

J V Narlikar. The Primeval Universe. Oxford University Press. 1989.

J V Narlikar. Introduction to Cosmology. Cambridge University Press. 1993.

John Barrow. The Origin of the Universe. Weidenfield & Nicolson, London. 1994.

Address for correspondence
Jayant V Narlikar,
IUCAA, P B No.4,
Ganeshkind,
Pune-411 007,
India.

CISC and RISC

CISC and RISC are terms that classify CPUs into two categories subjectively. The terms RISC (reduced instruction set computer) and CISC (complex instruction set computer) are misnomers because they refer to the CPU. CISC CPUs allow many ways to get at an operand that resides in memory. RISC CPUs allow only move-operations between a memory location and a CPU register. CISC instructions can have a variable length. RISC instructions always have a fixed length. CISC instructions can take a variable amount of time to complete execution. All RISC instructions take the same amount of time. Some CISC instructions can be very powerful. All RISC instructions have the same strength. Instructions occur in a stream and the stream is compiled from a high-level language source code. All CPUs execute or at least try to execute more than one instruction from the stream simultaneously. This is known as *pipelining*. RISC CPUs allow better coordination among instructions in the *pipeline* and this results in better sharing of different resources within the CPU. That is because the compiler can predict which instruction will need which resource, at what stage of its execution and for how long. So RISC CPUs yield better performance. RISC is the modern trend of CPU design. There are many vendors who brand their CPUs as RISCs, whereas actually they are not. Also, the intelligent reader may note that just a CPU being an RISC does not guarantee better performance on a given application. It depends on how the high-level language code is compiled to take advantage of the CPU architecture. Thus CISCs can be exploited by executing their powerful instructions and that may outperform an RISC at times. There is no reason for a vendor to feel defensive about her CPU being a CISC.

16-bit registers, (name given by Intel) four 16-bit segment registers (name given by Intel) and the two program control registers (name given by me).

The eight 16-bit registers, (*viz.* AX, BX, CX, DX, SP, BP, SI and DI) hardly justify the name *general purpose registers* (GPRs). Each register has a specific use. It is recommended that one keep certain types of operands in certain registers. The experienced assembly language programmer knows by instinct which register to use to keep what operand but it is unfair to expect a compiler to be endowed with such an 'instinct'. And what is worse, some instructions must use some specific registers as their destination and you must leave the register free before you try to execute the instruction.

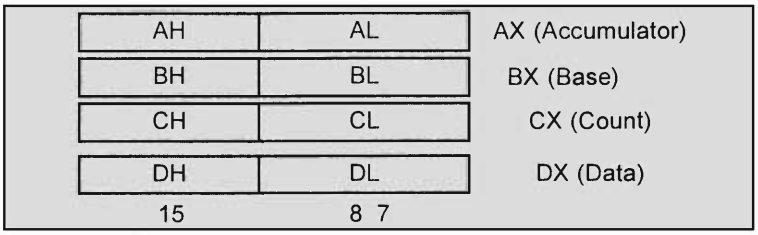


Figure 1 General - purpose registers of the 8088 CPU

Of the eight GPRs, four are better at handling data. They are AX, BX, CX and DX. Each of them can be seen either as two 8-bit registers (e.g. AH and AL) joined end to end or as one 16-bit register (e.g. AX). The 8-bit registers were there in the 8-bit 8085 (Figure 1).

Even among the data registers, there are 'specialists'. AX is good at collecting the results of arithmetic and logic operations. It is very strongly tied to the ALU (arithmetic and logic unit) and can serve as the source and destination of instructions that use the ALU. Thus when you operate on a program variable over and over, AX can be used to hold the variable during the long chain of instructions being executed and accumulate the end result. So AX is called the *accumulator*. BX is a specialist of

How the Intel 8088 Registers Got their Names

The predecessor of the Intel 8088 microprocessor (the first CPU of the IBM PC) was the highly successful 8085 microprocessor from Intel which had a data word length of 8 bits and which could address 216 bytes of memory (i.e. it had a 64KB long virtual address space). The 8088 had 16-bit wide internal data paths and registers. However to reduce the pin count of the chip, the external data path width of the 8088 chip was kept at eight. By then, the 8085 had featured in a sufficient number of single-board computers to bolster the confidence of a board-level computer designer to be able to use a CPU that (like the 8085) also had a data-width of 8 bits, in the single-board computer. Internal to the die, the 8088 had 16-bit registers that were extensions of the corresponding 8-bit registers of the 8085 (See *Figure 1*). All the registers with an 'L' as the last alphabet of their name are 8-bit registers which were there in the 8085. They were extended with another 8-bit register with an 'H' as the last alphabet of their name, to make a 16-bit register with an 'X' as the last alphabet of their name. The two 8-bit registers can be address- ed either separately or together as a 16-bit register.

Code, Data and Stack

To support a program in execution (some call a prog- ram in execution as a process), one needs to logically subdivide memory into three different regions:

Code region from where the CPU fetches, decodes and executes instructions,

Data region where variables used in a program are stored,

Stack region where the CPU temporarily stores addresses of instruction so that a program can call subprograms or invoke operating system calls. The stack is organized in such a way that the last item pushed onto the stack is the first item to be taken off (*popped*) and hence the name, as in a stack of cafeteria trays mounted on a dispenser. Stack is also used to pass values (called *parameters*) to subprograms.

In a flat memory model code, data and stack regions are all squeezed into one common virtual address space and can overrun each other if the program misbehaves, causing havoc that can result in a system crash. On a CPU that supports segmentation, a misbehaving program cannot cause as much damage. A region is recognized and protected by the CPU as part of its architectural responsibilities.

Using BP, one can also quickly pick out local variables allocated to a subprogram, and other temporary objects created on stack during the execution of the subprogram. This is very useful in supporting the execution of subprograms that call themselves.

Address for correspondence
S K Ghoshal
Supercomputer Education and Research Centre,
IISc Bangalore 560 012, India

Data can be moved between any of these registers over paths that are 16-bit wide and run internal to the CPU. The 8088 has only a 8-bit wide data path that comes out of the CPU. That makes motherboards made out of the 8088 a whole lot cheaper, rugged and reliable. The CPU-memory subsystem of an 8088-based motherboard works a bit slower though, compared to an equivalent 8086-based motherboard.

Earlier 8-bit board level products were common. Thus the 8088 having an eight bit external data path width certainly helped in making 8088-based motherboards acceptable among people who wanted their board-level products to interface with the IBM PC. These people pioneered to make the IBM PC the most diversely interfaced computer of the world.

There are two 16-bit program control registers IP and FLAGS. These registers are not meant for storing data. Their contents

Suggested Reading

Peter Norton and John Socha, Peter Norton's Assembly Language Programming Book for the IBM PC, Prentice Hall of India. 1989.

Patterson and Hennessy, Computer Architecture - A Quantitative Approach, Morgan Kaufmann. 1990.

memory address arithmetic. Scaling of offsets and indexing into a block of memory is better done using BX to hold the base address. Some programmers call it the base register. CX is good at keeping count of operations. Loop instructions are designed by the CPU architect to use CX as the count register. CL keeps count of things that can be done sensibly only up to a maximum of 255 times. For example, CL counts the number of shifts and rotate operations done on other registers holding bit-patterns. DX points at addresses of Input/Output ports, peripheral controller devices, their control registers and the like. I do not know why it is called the data register. May be because eventually all data get in or get out through Input/Output devices.

SP	(Stack Pointer)
BP	(Base Pointer)
SI	(Source Index)
DI	(Destination Index)

Figure 2 Pointer and Index registers of the 8088

There are four GPRs belonging to the pointer and index group. They are 16-bit long and none of them can be used as two 8-bit registers joined end to end. (See *Figure 2*). These are good at pointing at objects in memory and they increment and decrement fast. SI (source index) and DI (destination index) are specialists in holding the addresses of the source and the destination operands respectively. They come in handy for string operations when they increment and decrement automatically after each byte is moved while CX keeps the count. Thus large blocks of memory can be scanned, copied and initialized very fast just by writing one string instruction. This is one example of the power of a CISC architecture.

The stack pointer, SP points to the top of the stack. When any object is pushed, it decrements by an amount that equals the size of the object in bytes. When an object is popped, SP increments. The careful reader will note that SP is a highly special-purpose register, even though it is branded as a GPR. You cannot write any useful program that uses SP to do anything other than point at the last object pushed into the stack. BP too is a specialist in pointing at objects in the stack. It follows SP whenever any subprogram is called and serves as a base pointer from which the parameters passed to the subprogram can be accessed easily.

What is an Immediate Operand?

When the value of an operand is encoded within the instruction it is called an immediate operand as its value is known as soon as the instruction is decoded. For example, the instruction MOV AX, 0FFFFH is encoded in three consecutive bytes (each byte needs two hexadecimal digits to store) as B8FFFF. The byte B8 is the op-code of the move instruction. FFFF is the immediate operand of the same instruction.

The SX Idea

Even today the trend, where the width of the external data-path is half that of the internal data-path and operand size of the CPU, is continued by some designers. At Intel, such a CPU is called an SX CPU, whereas the regular one, whose external data-path is as wide as the internal data-path, is called a DX CPU. It is an unwise name given by Intel, as there is a register called DX within the CPU. For example, the 80386 is a 32-bit CPU. It has internal data-paths and registers that are 32 bits wide. The 80386 DX has 32 data pins to exchange information with its main memory. The 80386 SX has 16 data pins. Motherboards with SX CPUs (referred hereafter as SX motherboards) are more easily designed than motherboards with DX CPUs (referred hereafter as DX motherboards). This is because there are fewer wires that run between the CPU and the memory. They cost much less than DX motherboards and perform only slightly worse in the majority of applications. SX motherboards are more rugged, smaller in size and consume less electrical energy than DX motherboards.

affect the way the CPU executes instructions. The programmer should not try to load values into these registers, though there are ways of doing it. Depending on the result of executing an instruction, the contents of these two registers are set by the CPU automatically. Thus these two registers are a very important

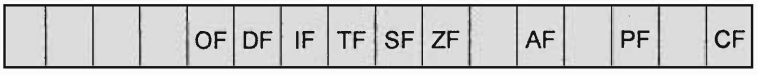
Keep the Pin-count Low

Another challenge those days was to package the silicon chip (also called a "die") inside a 40-pin Dual In-line Package (DIP). The total number of wires coming out of the chip could not exceed 40. So having 20 address pins to take the address (out of the die and the DIP) produced by the 8088 CPU could not be managed. Thus, the address lines and data lines had to share the same pins of the 8088 chip, by using them at different points of time during the memory access process (it is called a protocol in terminology of information exchange between two chips on a single board). This technique of sending two types of information over one bus is called multiplexing. With the advent of packaging technology, the pin-count has increased considerably. A new package is called Pin Grid Array (PGA) and can have up to 500 pins as of 1995. So multiplexing of address and data buses have been discontinued on the 386 and 486 and all other modern CPU chips. Now the challenge is to connect all these pins to other chips on the same printed circuit board (PCB) which serves as the motherboard without shorting them out. Thus SX motherboards remain easier to manufacture than DX motherboards albeit for a different reason.

part of the 'context' of a process, which must be saved when the process is interrupted, in order that the process can be resumed after a return from the interrupt. IP is the instruction pointer. It points to the instruction that is going to be fetched next at any point during the execution of a program. FLAGS register is the collection of the control and status flags, as shown in Figure 3. The blank bit-positions are for future designs of the same CPU dynasty.

In the next article, we will study the instruction set and assembly language programming of the base architecture.

Figure 3 Flags of the 8088



Flags and their Role

Flags can be classified into 2 groups: control and status flags. The state of the control flags dictates how the CPU should execute instructions. There are 3 control flags, DF, IF and TF.

- Setting DF=1 makes string instructions automatically decrement SI and DI after every byte is moved. Resetting DF = 0 makes string operations auto-increment.
- Setting IF=1 lets the CPU acknowledge and service interrupts. Resetting IF=0 makes the CPU ignore them.
- Setting TF =1 puts the CPU into single-step mode. It stops after every instruction and allows the program to be examined closely.

Control flags can be set/reset by special instructions. For example, executing the instruction STI sets the interrupt flag IF = 1.

Status Flags of 8088

There are 6 status flags:

- AF is the auxiliary carry flag. It is set if there is a carry from the lower 4-bits into the higher 4-bits. It is used for decimal arithmetic.
- CF, the carry flag is set whenever there is carry from, or a borrow into the result after any arithmetic operation.
- OF, the overflow flag is set, if an arithmetic overflow occurs.
- SF, the sign flag is set whenever the result is negative, if interpreted in a 2's complement notation.
- PF, the parity flag is set whenever the result has an even number of bits set to 1.
- ZF is the zero flag. It is set to zero, whenever the result of an operation is 0.

The CPU sets or resets these bits after every result is produced. In addition, CF can be set/reset by executing special instructions. STC sets it to 1. CLC resets it to zero.