8086 Processor

1.1 Introduction

Intel marketed the first microprocessor, named the 4004**.** This device caused a revolution in the electronics industry because previous electronic systems had a fixed functionality. With this processor the functionality could be programmed by software. Amazingly, by today's standards, it could only handle four bits of data at a time (a nibble), contained 2000 transistors, had 46 instructions and allowed 4 KB of program code and 1 KB of data. From this humble start the PC has since evolved using Intel microprocessors (Intel is a contraction of *Int*egrated *El*ectronics).

The second generation of Intel microprocessors began in 1974. These could handle 8 bits (a byte) of data at a time and were named the 8008, 8080 and the 8085. They were much more powerful than the previous 4-bit devices and were used in many early microcomputers and in applications such as electronic instruments and printers. The 8008 has a 14-bit address bus and can thus address up to 16 KB of memory (the 8080 has a 16-bit address bus giving it a 64 KB limit).

The third generation of microprocessors began with the launch of the 16-bit processors. Intel released the 8086 microprocessor which was mainly an extension to the original 8080 processor and thus retained a degree of software compatibility. IBM's designers realized the power of the 8086 and used it in the original IBM PC and IBM XT (eXtended Technology). It has a 16-bit data bus and a 20-bit address bus, and thus has a maximum addressable capacity of 1 MB. The 8086 could handle either 8 or 16 bits of data at a time (although in a messy way).

A stripped-down, 8-bit external data bus, version called the 8088 is also available. This stripped-down processor allowed designers to produce less complex (and cheaper) computer systems. An improved architecture version, called the 80286, was launched in 1982, and was used in the IBM AT (Advanced Technology).

In 1985, Intel introduced its first 32-bit microprocessor, the 80386DX. This device was compatible with the previous 8088/8086/80286 (80×86) processors and gave excellent performance, handling 8, 16 or 32 bits at a time. It has a full 32-bit data and address buses and can thus address up to 4 GB of physical memory. A stripped-down 16-bit external data bus and 24-bit address bus version called the 80386SX was released in 1988. This stripped-down processor can thus only access up to 16 MB of physical memory.

In 1989, Intel introduced the 80486DX which is basically an improved 80386DX with a memory cache and math co-processor integrated onto the chip. It had an improved internal structure making it around 50% faster with a comparable 80386. The 80486SX was also introduced, which is merely an 80486DX with the link to the math co-processor broken. Clock doubler/trebler 80486 processors were also released. In these the processor runs at a higher speed than the system clock. Typically, systems with clock doubler processors are around 75% faster than the comparable non-doubled processors. Typical clock doubler processors are DX2-66 and DX2-50 which run from 33 MHz and 25 MHz

clocks, respectively. Intel have also produced a range of 80486 microprocessors which run at three or four times the system clock speed and are referred to as DX4 processors. These include the Intel DX4-100 (25 MHz clock) and Intel DX4-75 (25 MHz clock).

The Pentium (or P-5) is a 64-bit 'superscalar' processor. It can execute more than one instruction at a time and has a full 64-bit (8-byte) data bus and a 32-bit address bus. In terms of performance, it operates almost twice as fast as the equivalent 80486. It also has improved floating-point operations (roughly three times faster) and is fully compatible with previous 80×86 processors.

The Pentium II (or P-6) is an enhancement of the P-5 and has a bus that supports up to four processors on the same bus without extra supporting logic, with clock multiplying speeds of over 300 MHz. It also has major savings of electrical power and the minimization of electromagnetic interference (EMI). A great enhancement of the P-6 bus is that it detects and corrects all single-bit data bus errors and also detects multiple-bit errors on the data bus.

1.2 8088 microprocessor

The great revolution in processing power arrived with the 16-bit 8086 processor. This has a 20-bit address bus and a 16-bit address bus, while the 8088 has an 8-bit external data bus. Figure 1.1 shows the pin connections of the 8088 and also the main connections to the processor. Many of the 40 pins of the 8086 have dual functions. For example, the lines AD0–AD7 act either as the lower 8 bits of the address bus (A0–A7) or as the lower 8 bits of the data bus (D0–D7). The lines A16/S3–A19/S6 also have a dual function, S3–S6 are normally not used by the PC and thus they are used as the 4 upper bits of the address bus. The latching of the address is achieved when the ALE (address latch enable) goes from a high to a low.

The bus controller (8288) generates the required control signals from the 8088 status lines $\overline{S0} - \overline{S1}$. For example, if $\overline{S0}$ is high, $\overline{S1}$ is low and $\overline{S2}$ is low then the $\overline{MEMR}$ line goes low. The main control signals are:

- $\overline{IOR}$ (I/O read) which means that the processor is reading from the contents of the address which is on the I/O bus.
- $\overline{IOW}$ (I/O write) which means that the processor is writing the contents of the data bus to the address which is on the I/O bus.
- $\overline{MEMR}$ (memory read) which means that the processor is reading from the contents of the address which is on the address bus.
- $\overline{MEMW}$ (memory write) which means that the processor is writing the contents of the data bus to the address which is on the address bus.
- $\overline{INTA}$ (interrupt acknowledgement) which is used by the processor to acknowledge an interrupt ($\overline{S0}$, $\overline{S1}$ and $\overline{S2}$ all go low). When a peripheral wants the attention of the processor it sends an interrupt request to the 8259 which, if it is allowed, sets INTR high.

The processor either communicates directly with memory (with $\overline{MEMW}$ and $\overline{MEMR}$) or communicates with peripherals through isolated I/O ports (with $\overline{IOR}$ and $\overline{IOW}$).

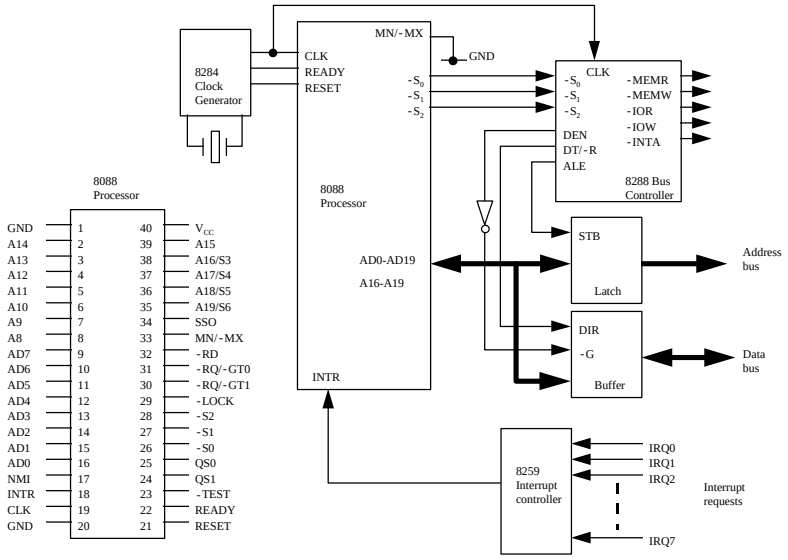

Figure 1.1 8088 connections.

1.2.1 Registers

Each of the PC-based Intel microprocessors is compatible with the original 8086 processor and is normally backwardly compatible. Thus, for example, a Pentium can run 8086, 80386 and 80486 code. Microprocessors use registers to perform their operations. These registers are basically special memory locations within the processor that have special names. The 8086/88 has 14 registers which are grouped into four categories, as illustrated in Figure 1.2.

General-purpose registers

There are four general-purpose registers that are AX, BX, CX and DX. Each can be used to manipulate a whole 16-bit word or with two separate 8-bit bytes. These bytes are called the lower and upper order bytes. Each of these registers can be used as two 8-bit registers, for example, AL represents an 8-bit register that is the lower half of AX and AH represents the upper half of AX.

The AX register is the most general purpose of the four registers and is normally used for all types of operations. Each of the other registers has one or more implied extra functions. These are:

- AX is the accumulator. It is used for all input/output operations and some arithmetic operations. For example, multiply, divide and translate instructions assume the use of AX.
- BX is the base register. It can be used as an address register.
- CX is the count register. It is used by instructions which require to count. Typically is it is used for controlling the number of times a loop is repeated and in bit-shift operations.
- DX is the data register. It is used for some input/output and also when multiplying and dividing.

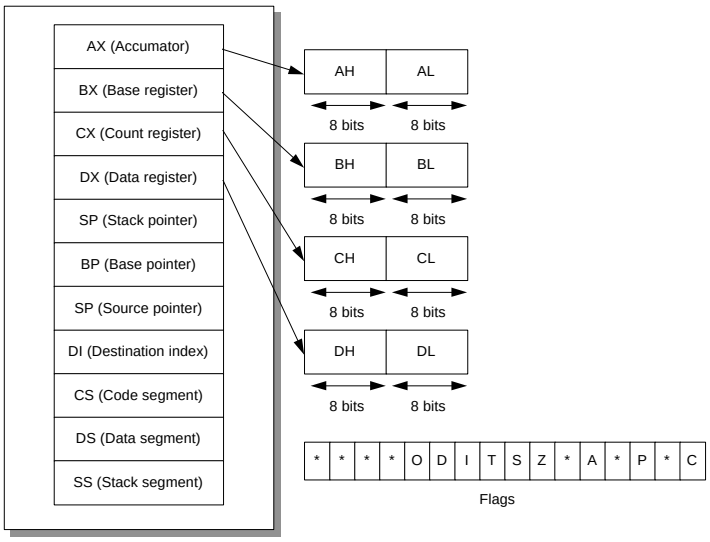

Figure 1.2 8086/88 registers.

Addressing registers

The addressing registers are used in memory addressing operations, such as holding the source address of the memory and the destination address. These address registers are named BP, SP, SI and DI, which are:

- SI is the source index and is used with extended addressing commands.
- DI is the destination index and is used in some addressing modes.
- BP is the base pointer.
- SP is the stack pointer.

Status registers

Status registers are used to test for various conditions in an operations, such as ‘is the result negative’, ‘is the result zero’, and so on. The two status registers have 16 bits and are called the instruction pointer (IP) and the flag register (F):

- IP is the instruction pointer and contains the address of the next instruction of the program.
- Flag register holds a collection of 16 different conditions. Table 1.1 outlines the most used flags.

Segments registers

There are four areas of memory called segments, each of which are 16 bits and can thus address up to 64 KB (from `0000h` to `FFFFh`). These segments are:

- Code segment (cs register). This defines the memory location where the program code (or instructions) is stored.
- Data segment (ds register). This defines where data from the program will be stored (ds stands for data segment register).

- Stack segment (ss register). This defined where the stack is stored.
- Extra segment (es).

All addresses are with reference to the segment registers.

The 8086 has a segmented memory, the segment registers are used to manipulate memory within these segments. Each segment provides 64 KB of memory, this area of memory is known as the current segment. Segmented memory will be discussed in more detail in Section 1.3.

Table 1.1 Processor flags.

Bit	*Flag position*	*Name*	*Description*
C	0	Set on carry	Contains the carry from the most significant bit (left-hand bit) following a shift, rotate or arithmetic operation.
A	4	Set on 1/2 carry	
S	7	Set on negative result	Contains the sign of an arithmetic operation (0 for positive, 1 for negative).
Z	6	Set on zero result	Contains results of last arithmetic or compare result (0 for nonzero, 1 for zero).
O	11	Set on overflow	Indicates that an overflow has occurred in the most significant bit from an arithmetic operation.
P	2	Set on even parity	
D	10	Direction	
I	9	Interrupt enable	Indicates whether the interrupt has been disabled.
T	8	Trap	

Memory addressing

There are several methods of accessing memory locations, these are:

- Implied addressing which uses an instruction in which it is known which registers are used.
- Immediate (or literal) addressing uses a simple constant number to define the address location.
- Register addressing which uses the address registers for the addressing (such as AX, BX, and so on).
- Memory addressing which is used to read or write to a specified memory location.

1.3 Memory segmentation

The 80386, 80486 and Pentium processors run in one of two modes, either virtual or real. In virtual mode they act as a pseudo-8086 16-bit processor, known as the protected mode. In real-mode they can use the full capabilities of their address and data bus. This

mode normally depends on the addressing capabilities of the operating system. All DOS-based programs use the virtual mode.

The 8086 has a 20-bit address bus so that when the PC is running 8086-compatible code it can only address up to 1 MB of physical memory. It also has a segmented memory architecture and can only directly address 64 KB of data at a time. A chunk of memory is known as a segment and hence the phrase 'segmented memory architecture'.

Memory addresses are normally defined by their hexadecimal address. A 4-bit address bus can address 16 locations from `0000b` to `1111b`. This can be represented in hexadecimal as `0h` to `Fh`. An 8-bit bus can address up to 256 locations from `00h` to `FFh`. Section 1.7 outlines the addressing capabilities for a given address bus size.

Two important addressing capabilities for the PC relate to a 16- and a 20-bit address bus. A 16-bit address bus addresses up to 64 KB of memory from `0000h` to `FFFFh` and a 20-bit address bus addresses up to 1 MB from `00000h` to `FFFFFh`. The 80386/80486/Pentium processors have a 32-bit address bus and can address from `00000000h` to `FFFFFFFFh`.

A segmented memory address location is identified with a segment and an offset address. The standard notation is `segment:offset`. A `segment` address is a 4-digit hexadecimal address which points to the start of a 64 KB chunk of data. The `offset` is also a 4-digit hexadecimal address which defines the address offset from the segment base pointer. This is illustrated in Figure 1.3.

The `segment:offset` address is defined as the logical address, the actual physical address is calculated by shifting the segment address 4 bits to the left and adding the offset. The example given next shows that the actual address of `2F84:0532` is `2FD72h`.

Segment (2F84):	0010	1111	1000	0100	0000
Offset (0532):		0000	0101	0011	0010
Actual address:	0010	1111	1101	0111	0010

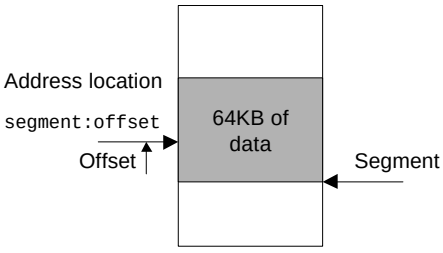

Figure 1.3 Memory addressing.

1.3.1 Accessing memory using C and Pascal

In C the address `1234:9876h` is specified as `0x12349876`. Turbo Pascal accesses a memory location using the predefined array `mem[]` (to access a byte), `memw[]` (a word) or `memw[]` (a long integer). The general format is `mem[segment:offset]`.

1.3.2 Near and far pointers

A near pointer is a 16-bit pointer which can only be used to address up to 64 KB of data whereas a far pointer is a 20-bit pointer which can address up to 1 MB of data. A far

pointer can be declared using the `far` data type modifier, as shown next.

```
char   far *ptr;     /* declare a far pointer           */
ptr=(char far *) 0x1234567;/*initialize far pointer     */
```

In the program shown in Figure 1.4 a near pointer `ptr1` and a far pointer `ptr2` have been declared. In the bottom part of the screen the actual addresses stored in these pointers is displayed. In this case `ptr1` is `DS:1234h` and `ptr2` is `0000:1234h`. Notice that the address notation of `ptr1` is limited to a 4-digit hexadecimal address, whereas `ptr2` has a `segment:offset` address. The address of `ptr1` is in the form `DS:XXXX` where `DS` (the data segment) is a fixed address in memory and `XXXX` is the offset.

There are several modes in which the compiler operates. In the small model the compiler declares all memory addresses as near pointers and in the large model they are declared as far pointers. Figure 1.5 shows how the large memory model is selected in Borland C (`Options` → `Compiler` → `Model` → `Large`). The large model allows a program to store up to 1 MB of data and code. Normally, for DOS-based program, the small model is the default and only allows a maximum of 64 KB for data and 64 KB for code.

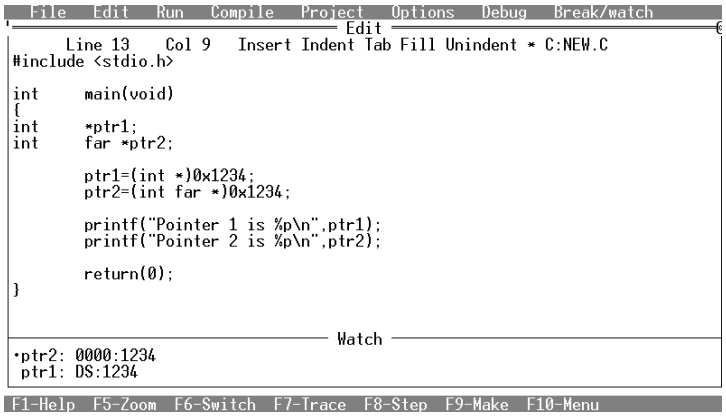

Figure 1.4 Near and far pointers.

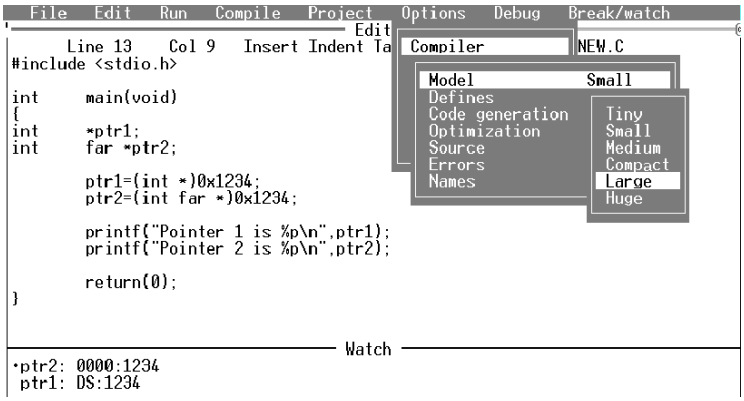

Figure 1.5 Compiling a program in the large model.

1.4 View inside the processor

To be able to view the processor the user must use a debugging program. Figure 1.6 shows an example of Turbo Debugger which is available with many of the Borland software development products and can be used to view the operation of a program. It can be seen that the machine code and equivalent assembly language macro appears in the top left-hand window. A sample code line is:

```
cs:01FA→55              push    bp
```

which specifies that the memory location is `01FA` in the code segment (`cs:01FA`). The machine code at this location is `55` (0101 0101) and the equivalent Assembly Language instruction is `push bp`. Note that the `cs` segment address in this case is `5757h`, thus the actual physical address will be with reference to the address `57570h`.

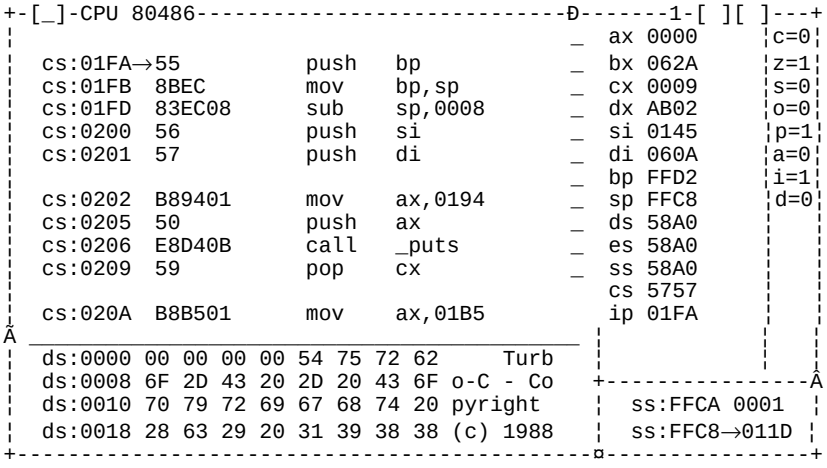

Figure 1.6 Example screen from Turbo Debugger.

The contents of the flag register is shown on the right-hand side. In this case the flags are:

C=0, Z=1, S=0, O=0, P=1, A=0, I=1 and D=0.

The registers are shown to the left of the flag register. In this case the contents are:

AX=0000h, BX=062Ah, CX=0009h, DX=AB02h, SI=0145h, DI=060Ah, BP=FFD2h, SP=FFC8h, DS=58A0h, ES=58A0h, SS=58A0h, CS=5757h, IP=01FAh.

The data (in the data segment) is shown at the bottom left-hand corner of the screen. The first line:

```
ds:0000 00 00 00 00 54 75 72 62     Turb
```

shows the first 8 bytes in memory (from `DS:0000` to `DS:0007`). The first byte in memory

is 00h (0000 0000) and the next is also 00h. After the 8 bytes are defined the 8 equivalent ASCII characters are shown. In this case, these are:

```
Turb
```

The ASCII equivalent character for 5A (1001 1010) is 'T' and for 75 (0111 0101) it is 'u'. Note that, in this case, the data segment register has 58A0h. Thus the location of the data will be referenced to the address 58A00h.

The bottom right-hand window shows the contents of the stack.

1.5 Machine code and assembly language

An important differentiation is between machine code and assembly language. The actual code that runs on the processor is machine code. These are made up to unique bit sequences which identify the command and other values which these commands operate on. For example, for the debugger screen from Figure 1.6, the assembly language line to move a value into the AX register is:

```
mov     ax,0194
```

the equivalent machine code is:

```
B8 94 01
```

where the code B8h (1011 1000b) identifies the instruction to move a 16-bit value into the AX register and the value to be loaded is 0194h (0000 0001 1001 0100b). Note that the reason that the 94h value is stored before the 01h value is that on the PC the least significant byte is stored in the first memory location and the most significant byte in the highest memory location. Figure 1.7 gives an example of storage within the code segment. In this case the two instructions are mov and push. In machine code these are B8h and 50h, respectively.

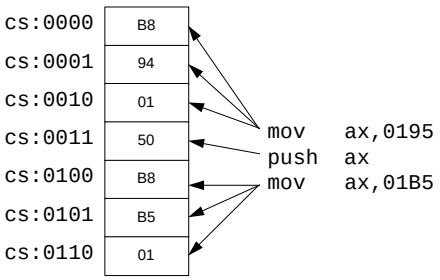

Figure 1.7 Example memory storage for code segment.

1.6 Exercises

1.6.1 How much memory can a 16-bit address bus address?

1.6.2 Outline how the 8086 differs from the 8088. Also, outline how the 80386DX differed from the 80386SX.

1.6.3 For the debug screen given in Figure 1.8 determine the following:

(i) Contents of AX, BX, CX, DX, SI, DI.
(ii) Contents of AH, AL, BH and BL.
(iii) The first assembly language command.
(iv) The physical memory address of the first line of code (Hint: the `cs:02C2` and the value in the `cs` register need to be used).
(v) The physical memory address of the data (Hint: the `ds:0000` and the value in the `ds` register need to be used).

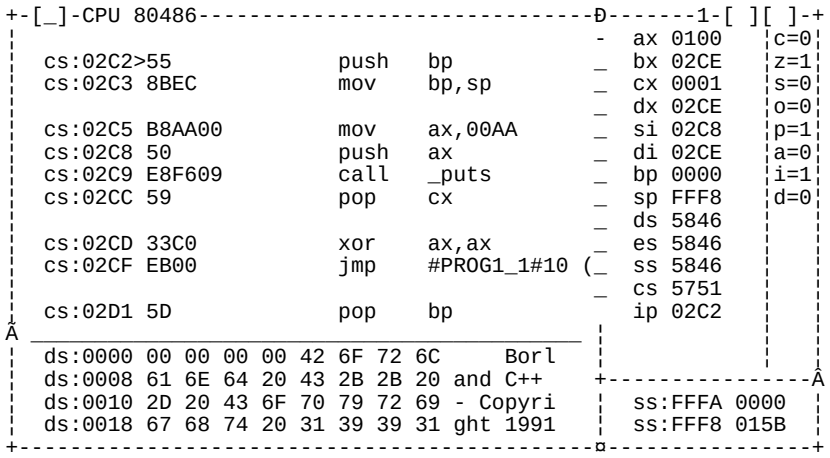

Figure 1.8 Example screen from Turbo Debugger.

1.7 Memory address reference

Address bus size	*Addressable memory (bytes)*	*Address bus size*	*Addressable memory (bytes)*
1	2	15	32K
2	4	16	64K
3	8	17	128K
4	16	18	256K
5	32	19	512K
6	64	20	1M†
7	128	21	2M
8	256	22	4M
9	512	23	8M
10	1K*	24	16M
11	2K	25	32M
12	4K	26	64M
13	8K	32	4G‡
14	16K	64	16GG

* 1K represents 1024 † 1M represents 1 048 576 (1024 K)
‡ 1G represents 1 073 741 824 (1024 M)