8

A typical 8-bit microprocessor

The 8-bit microprocessor is described in some detail to provide a foundation upon which others can be built since the story of microprocessors is one of evolution rather than 'bolts out of the blue'.

An 8-bit microprocessor – the Z80180

This microprocessor is the modern version of an old favorite called the Z80 which was developed in 1978 which was itself just an improved version of the Intel 8080. There are many similarities since some of the engineers that designed the 8080 left Intel and went to work for Zilog to build the Z80 which proved to be one of the most popular and longest living microprocessors – it is still widely used even in mobile phones.

When Zilog developed the Z80180, they were naturally anxious to hold on to their previous customers by making the new version compatible with the earlier Z80. This allows customers an easy and inexpensive way of updating their system without the high design costs of starting afresh with a completely different device.

The Z80180 can run any programs that were written for the Z80 as well as adding some new features. Technically, this is referred to as providing full backwards compatibility.

The X in Z8X180

If we were to go to a Ford dealer to buy a truck, they will offer us a whole series of them, all of which are basically the same but with a choice of engine sizes, gearboxes, seating arrangements and many other options. In this way, they can please the maximum number of customers without incurring significant redesign costs.

Microprocessors are much the same. They start with a basic design and similar versions are indicated by a slight change in the type number. So far, Zilog have produced the Z80180, Z8S180, and the Z8L180. By comparing the type numbers, we can see that they only differ in the third number or letter and by replacing this letter by X we can refer to a generalized type as the Z8X180. This use of X is in general use throughout the industry. The group of microprocessors are referred to as a family.

Despite the push from the computer industry for faster and faster microprocessors, there are many purposes for which small, slow and simple microprocessors is ideal. Billions of 4- and 8-bit microprocessors and microcontrollers are whirring away embedded inside things like microwave ovens, printers and instruments. Most of us share our homes with about fifty of these devices without being aware of their existence. They massively outnumber the well publicized high-speed computer microprocessors.

Inside of the central processing unit (CPU)

Figure 8.1 shows the main parts of the Z80180 CPU. It consists of a number of different registers which store or make use of the data when

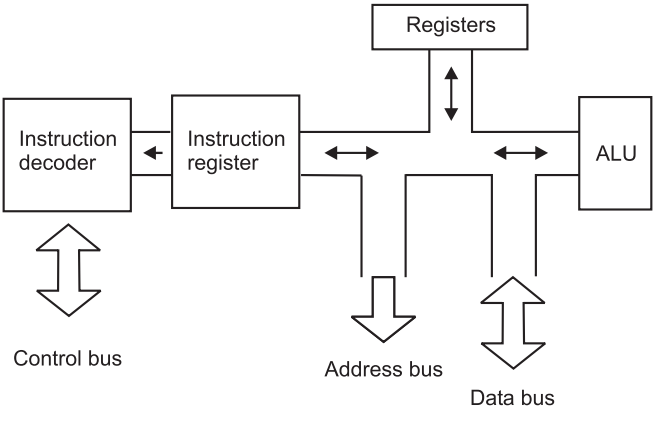


Figure 8.1
The central processing unit (CPU)

carrying out the instructions in the program. In order to maintain backward compatibility, most of the registers use the same system of labelling with letters to keep it familiar to anyone who has used the Z80.

Instruction register

The instruction register is just a small memory able to store 8 bits, or 1 byte of information. This information is an instruction for the microprocessor to carry out. The information is latched into the instruction register to release the internal data bus for other purposes. It doesn't do anything with the binary input – it just remembers what it is.

Instruction decoder

The instruction decoder is the part of the microprocessor that is able to actually carry out an instruction. Its first step is to identify which instruction that has been entered. It does this by comparing its binary code with an internally stored list. Once it has located the instruction it follows a built-in program called a microprogram. This microprogram is designed into the microprocessor by the manufacturer and details all the necessary steps to complete any instruction of which it is capable. This is the commercially sensitive and critical part of the microprocessor.

When the Z80180 is given a job to do, it must be given an instruction. An instruction is in the form of an 8-bit binary number. We will be coming back to this in a little while.

Arithmetic and logic unit (ALU)

This is a simple pocket calculator. It can add, subtract, multiply, divide, and do various tricks with logic gates AND, OR, XOR etc. Compared with the average scientific calculator, it is pretty poor.

CPU register banks

Technical information about a microprocessor will always include a diagram or listing of the internal registers. Registers are small memories, of between 8 and 128 bits each, depending on the microprocessor being considered. In the Z80180, they are all 8 or 16 bits in size.

Each microprocessor has its own collection of registers so it is essential to read the technical information carefully to see what they do.

There are 'general purpose registers' that are under the control of the user and can be used for a variety of temporary storage purposes.

There are also special purpose registers that are dedicated to particular functions, like the instruction register, for instance.

The Z80180 has sixteen general-purpose registers, and six for special purposes. Figure 8.2 shows the registers.

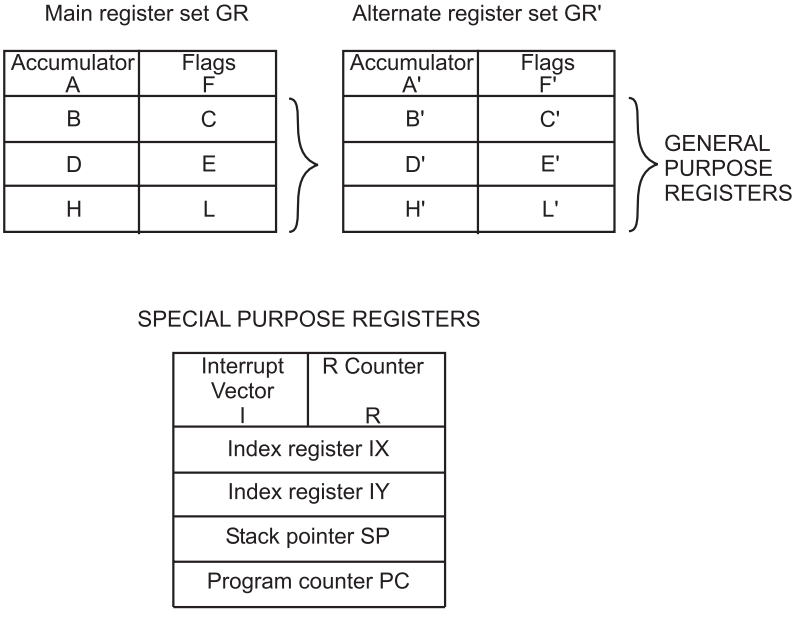


Figure 8.2 Z80X80 internal registers

We will concentrate on the upper two registers for the moment. The Main register set and the alternate register set are identical except for the small ‘ against the letters. This ‘ is called ‘prime’ so D’ would be read as ‘dee prime’. The prime is only there to distinguish between the two sets of registers.

Why have two sets? Having two sets allows the programmer to use the main set of registers while the prime registers can be loaded with other information ready for immediate use when required.

Imagine the microprocessor is busily controlling a printing machine when the fire alarm is activated. When the fire alarm signal reaches the microprocessor, it immediately abandons the printing program and starts to run the ‘Fire Alarm’ program alerting the fire fighters, evaluating the building etc. Some of the information for running this program could be stored in the spare set of registers. When the alarm program has been completed, the microprocessor calmly switches back to its main register set, and continues its printing work until it melts in the fire.

The accumulator

The programmer can use it at any time to store an 8-bit binary number. It can also store numbers to be used in arithmetic operations or for logic operations like AND, OR etc. The results of such calculations are put back into the accumulator after completion. Being only an 8-bit register, it can only hold one byte at a time so any previous data stored in this register will be overwritten as soon as anything else is stored.

The flag register

This is an 8-bit register but it is unusual in that each bit stored is quite independent of all the others. In all other registers, each bit is just part of a single binary value so each bit has a numerical value.

A status register or flag register is like a window for us to see into the workings of the microprocessor. It's rather like a simple communication system. For example, it allows us to say to the microprocessor, 'if you see a negative number during your calculation, just let me know'.

We do not have to take any notice of the flags, they are just indicators that we may take note of or ignore, just as we wish – rather like a thermometer on the wall.

Each flag is identified by a letter and these are shown in Figure 8.3. You will see that two of the bits, 3 and 5, are marked with an X to show that they are not used. These bits were left spare for later development (which has never happened).

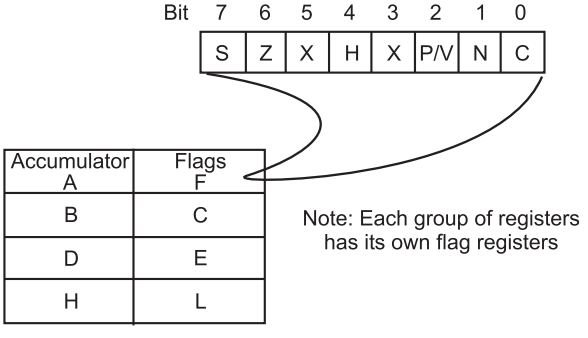


Figure 8.3
The Z80180 flag register

A note: In microprocessors, computers and electronics generally, something that is marked with an X is not used, and so we don't care what the voltage or binary value is.

Another note: a flag is said to be 'set' when its value is 1 and is 'cleared' to 0.

Sign flag (S)

The S flag is just a copy of the bit 7 of the accumulator. A negative number has a 1 in bit 7 and a positive number has a 0 in bit 7 so this flag indicates the sign of the number. You may remember that signed magnitude numbers use a 1 to indicate a negative number and 0 to indicate a positive number. Likewise a negative number expressed in two's complement form will have its left-hand bit as a 1. The number zero is treated as a positive number so we cannot use the S flag to spot a zero result. We do this by employing a special 'zero' flag as we will see in a moment.

Zero flag (Z)

The Z flag spends all its time watching for a result of zero.

You may remember that when we were looking at the uses of an XOR gate, we used this gate to check the code number in a cash dispenser. When the entered value and the card number were the same, it allowed cash to be withdrawn. What actually happened was that the two numbers were compared and if they were the same, then the result would be zero and the Z flag would spring into action. It 'sets' or changes to a binary 1 if it sees a zero result and stays at binary 0 at all other times.

Two things to be careful about. The Z flag only goes to a one state if all bits in the latest result are zero. Also, be careful about the way round: result 0 makes the Z flag = 1; result not zero, Z flag = 0.

All these results would give a Z flag value of zero:

0100 0000
1111 1111
0000 0001
1001 1010

Only a result of 0000 0000 would make the Z flag go to a binary 1.

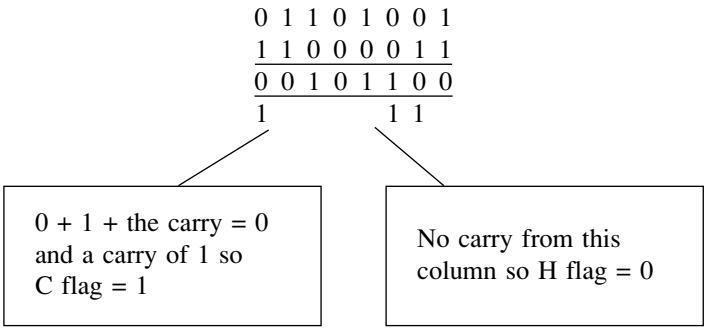
Add/subtract flag (N)

The N flag is just there to tell us whether the last arithmetic instruction was 'add' or 'subtract'. N = 0 for add, N = 1 for subtract. Not very exciting.

Carry flag (C) and the half-carry flag (H)

When addition is carried out, it sometimes results in a bit being carried over to the next column. The C flag copies the value of the carry from bit 7 and the H flag records the carry from bit 3 into bit 4 during additions. It also reflects the value of the 'borrow' in subtractions.

Here is an example of an addition showing the C and H flags.



Parity/overflow flag (P/V)

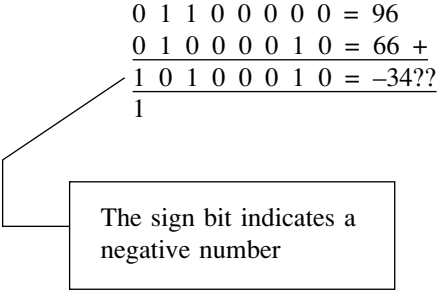
This is used for two separate purposes.

The 'V' in the title refers to oVerflow in arithmetic calculations. Some combinations of numbers can be misinterpreted and give an apparently incorrect answer, as we will see.

When the flag spots a possible error it is set, when no error is likely, it is cleared.

Here is an example where we could see a problem.

We are going to add two positive numbers +96 and +66 so even if we can't guess the result, we do at least know that it is a positive number.



In this example, we have added two numbers that start with zero and are therefore positive and the result starts with a one, which is a negative number so the P/V flag will be set.

An easy way to check for a possible error is to look at the carry into bit 7 and the carry out of bit 7.

The rule is:

Carry in and carry out or No carry in and no carry out	= No error so P/V flag = 0

In our example, there was a 1 carried into bit 7 from the previous column but there was no bit carried out from bit 7. This does not fit into the rule above so the P/V flag will be set to 1 indicating a possible error in the result.

Its second purpose is to operate in its parity mode.

In this mode, it checks a byte of data and counts the number of 1 states, if the total is an even number, the P/V flag is set and if odd, it is cleared.

So the byte 10110101 would mean P/V = 0 since the number 1 occurs 5 times, which is an odd number. And 10010110 would result in P/V = 1 because there are four occasions when the number 1 occurs and 4 is an even number.

This flag operates in the P or V mode depending on the instruction being carried out at the time.

We will look at parity in more detail in Chapter 17.

The general-purpose registers

The general-purpose registers are all 8-bit registers but if we like, we can use them two at a time as 16-bit registers. When pairing them up, our choice is restricted to the way suggested by Figure 8.2. We can combine BC, DE or HL but it's not a free choice, we cannot choose any two such as B and L. We can use a pair like BC as a 16-bit register and, at the same time, use D/E/H/L as separate 8-bit registers. As usual, the alternate register set behaves in exactly the same manner.

They have labelled the first four as B, C, D and E, so it may seem a little odd to use H and L for the last two.

The reason for this is that these registers are also used to keep track of memory addresses. The Z8018X has a 16-bit address bus and so requires a 16-bit register to be able to store a full address. The H and L stand for High and Low bytes of the address so if we wanted to store the address 2684H, we would put 84 in the L register and 26 in the H register, using binary of course. As well as being used to hold an address, the H and L registers can still be used for any other purpose that we want, just like B,C, D and E.

Special-purpose registers

Program counter (PC)

A program is a list of instructions for the microprocessor to carry out. Before use, they must be stored in some ROM or RAM. Let's assume that we used addresses starting from 6400H. To run the program we must tell the microprocessor to 'go' from 6400H.

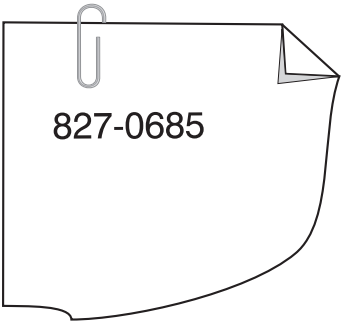
What we are actually doing is loading the address 6400H into the program counter and the microprocessor starts from there. Once it has completed the instruction in 6400H, it goes to the next address 6401H and then 6402H etc until it reaches the end of the program.

The purpose of the program counter is to keep track of the address that is going to be used next.

Stack pointer (SP)

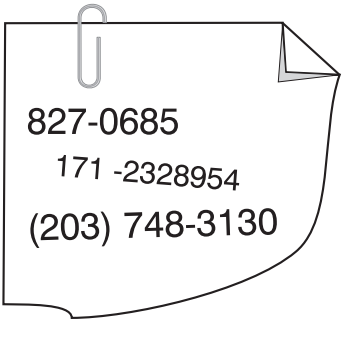
If someone gave us a telephone number to remember, we would be likely to reach for a scrap of paper and a pencil to write it down. Our note may look like Figure 8.4.

Figure 8.4
A number to remember



If we were then given another couple of numbers, we are likely to jot them down, in order, under the first one as in Figure 8.5.

Figure 8.5
A few more numbers



We could read them back in order by reading the bottom one first, then the next and finally the top one. The first one to be entered was the last one to come out.

The microprocessor could do something similar by storing information in a series of 16-bit memory locations called a 'stack'. The stack is loaded in order and then unloaded starting from the last number stored and working back to the top of the stack. The first number that was stored would be retrieved last and for this reason it would be called a 'first-in, last-out' method or FILO. For maximum confusion, this can also be called a LIFO system that, of course, means exactly the same. Have a think about it.

Inside the microprocessor, a series of RAM locations are reserved to be used as a stack and an address counter must be employed to keep track of what address of the stack is to be used next. This counter is called a stack pointer since it 'points at' the next address to be used.

The stack is generally used by the microprocessor to store data on a very temporary basis. We are allowed to use it but the microprocessor takes priority and doesn't know that we have been fiddling with it. For example, if the microprocessor had stored three bits of data in order as in Figure 8.6(a) and we put one piece of our data on the end as in Figure 8.6(b), there would now be data in four locations. However, the microprocessor would not be aware of this. When the microprocessor wished to retrieve its data, it would unload three bytes and accidentally get the wrong one as in Figure 8.6(c).

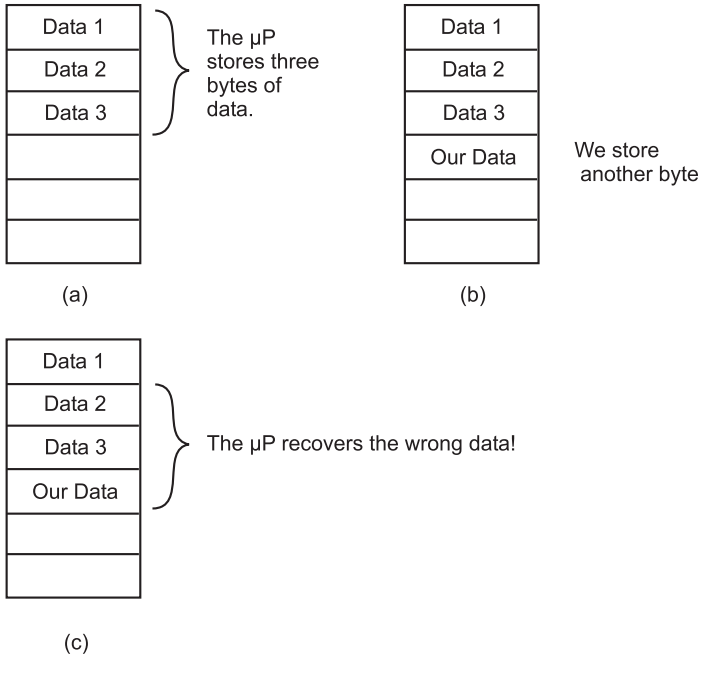


Figure 8.6
Using the stack

The moral of this story is leave the stack alone – or be very careful! The technical data will always give you sufficient information to be able to know whether or not the microprocessor will be using the stack but it still takes a lot of working through to be sure.

Index registers

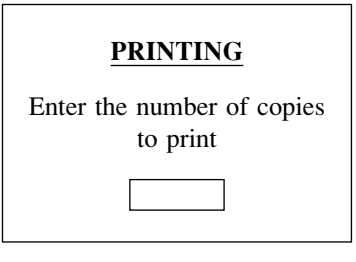
We have two index registers, one called X and the other called Y. They both perform the same function.

An index is used in situations in which we wish to perform a sequence of similar tasks one after the other. Perhaps to use data stored in a series of memory locations. The 16-bit content of the index register can be added to the contents of the program counter (PC) to produce a new address.

As an example, let's assume that we are running a program which after using address 2600H we would like it to use some data stored in address 3800H. We could do this by loading the 16-bit number 1200H into the X register and instruct the microprocessor to go to the address written as PC + IX (program counter and index X). In this case, the microprocessor goes to address 2600H + 1200H = 3800H to retrieve the data to be used. This jump from 2600H to 3800H is called an offset or we may say that the index X register has provided an offset of 1200H. In this example, index register Y could have been used equally well.

Index registers are often used to input data previously entered into a table or from another part of the memory. The user of a program may be asked to type in a number. This number would be entered in RAM but the program that is going to use this data is stored in ROM. A suitable offset in one of the index registers could shift the address sufficiently to bring it into the RAM area of the memory map.

These registers are also used in situations where the program seeks a response from the user. The screen may show:



We could implement this by storing the response in one of the index registers and, as the printer performs each copy, the number stored is decreased by one (decremented) until it reaches zero. Every time that the counter is decremented, we instruct the microprocessor to check

the state of the Z flag. When the count finally reaches zero, the Z flag will be set and the printer can be stopped.

R counter (R)

This is a simple counter in which the lower seven bits are used to count the number of instructions that have been carried out by the microprocessor while it has been running the present program. It is reset to zero whenever the micro is restarted.

Figure 8.7 shows the remaining parts of a microprocessor. All these parts are inside the single block marked 'microprocessor' (μP) in Figure 7.4. These include all the facilities for connecting the device to external circuits and a few other essentials.

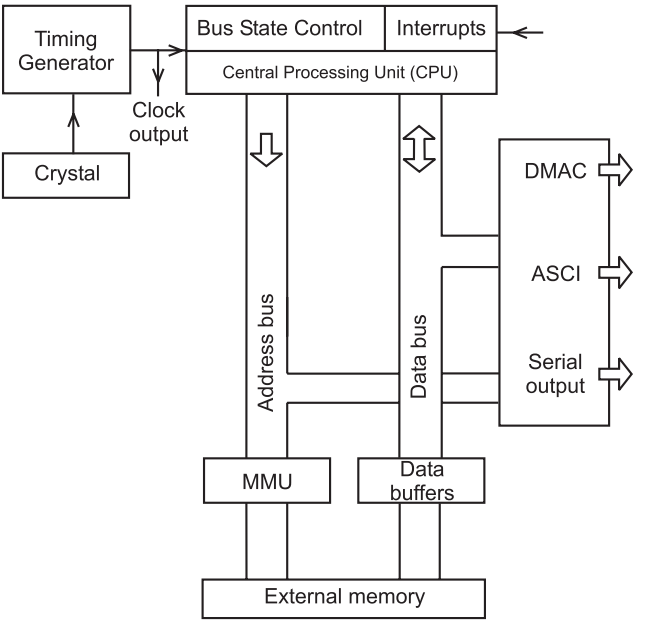


Figure 8.7
The Z80180 microprocessor

Data buffers

The data buffer is an 8-bit register to store the information being provided by the external data bus. Once the buffer stores the binary information then tri-state buffers can lock-in the information disconnecting the external data bus so it can be used for carrying other data. This process is called 'latching'. As a general rule, all inputs to a chip from a bus are latched in. This is because the buses take time to set up new voltages on their lines and need to get started on this job as soon as possible.

The address and data buses

As with all microprocessors, there are three buses, the address, data and control. As we saw in the previous chapter, the control bus is a loose collection of connections that send signals around the system to make connections and control the operation of each area as necessary. Here we will concentrate on the address and data buses in the Z80180.

The Z80180 has a (sort of)16-bit address bus and an 8-bit data bus.

In the smaller microprocessors, it was usual practice to make the address bus twice the width of the data bus but there is no reason why this has to be the case. The choice of address bus width will depend on the anticipated size of the memory that needs to be addressed in the final designs. Unusually, in this microprocessor there are two conflicting requirements: it has to remain compatible with the 16-bit address bus of the Z80 but we would like to be able to address 1 MB which requires 20 address lines.

These requirements are met by the block shown as MMU or memory management unit.

Memory management unit (MMU)

This circuit includes an 8-bit register that holds address information that can be controlled by the software being used. These 8 bits are combined with the top four address lines, A12–A15, of the address bus. The lower half of these 8 bits are added to the top 4 bits of the address bus and the upper 4 bits are used to expand the useable memory by adding the four extra address lines, A16–A19, to allow a total of twenty lines to be addressed – see note below. The process is shown in Figure 8.8.

The Z80180 family is manufactured in three different packages. The original 40 pins of the old Z80 has increased to 64 pins in the dual-in-line (d.i.l.) package similar to that shown at the top of Figure 1.6. They also make a 68 pin plastic leaded chip carrier (PLCC) version which is just like the square package in Figure 1.6. Finally, there is an 80 pin surface mount type.

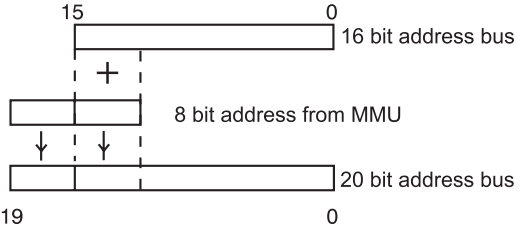


Figure 8.8
Expanding the address bus

Surface mount devices do not have pins that go through holes in the printed circuit board but have connectors that come out horizontally and then soldered onto the surface of the board. This method allows significant space savings.

Note

The smaller number of pins on the d.i.l. layout has limited the number of address pins to nineteen and therefore the maximum memory that can be accessed is now 2^{19} or 512k (really 524288 since a 'k' in this case is taken as 1024 as we met in Chapter 2).

Address buffers

These are tri-state buffers just like the data buffers except they are one-way devices. The microprocessor sends addresses out along the address bus but no address information can come in this way. If we need to send an address into the microprocessor then we are loading information and all information whatever it is goes into the data bus which, as we have seen, can send any data out to the external circuits or accept any data into the microprocessor.

Clock generator

As we saw in Chapter 7, a microprocessor needs a square wave signal to keep all the internal operations in step otherwise it will all end in chaos with data and instructions moving at the wrong times and getting jumbled up.

In this particular microprocessor the clock speed can be set to values between 6 MHz and 33 MHz. The clock signal originates from either a crystal or an external signal source – but never at the same time. Using an external signal can be useful where the microprocessor is used as part of a larger installation and this would allow the microprocessor timing to be governed by the surrounding circuitry.

There are three nice things about increasing the frequency of a crystal: they get smaller, lighter and cheaper. The clock circuitry includes a divide-by-two circuit to allow the use of a double-frequency crystal with the above-mentioned benefits.

Interrupts

You may remember our sad little story of our microprocessor-based system controlling the printer while all around the office was burning. It stopped its main program and went off to phone the fire services, alert the maintenance staff, activate sprinklers etc.

Fire is not the only hazard our little friend can safeguard us from. It could warn for other things like the paper running out in the printer or data corruption on the telephone cable.

Now, we would not want it to send for more paper to deal with a possible problem with the telephone cable so the microprocessor needs to be told what the problem is and what to do about it.

The different programs for each of these problems are stored in a group of addresses in a ROM chip. As an example, we could load the programs at these addresses:

0800–0855H = paper supplies
0870–08A8H = telephone data.

Both of these programs have 08 as the high byte of the address. This value would be stored in the interrupt vector register. The low bytes 00 and 70 are supplied by the sensing device. As soon as the telephone data is corrupted it is noticed by a sensor, and it sends a signal to interrupt the microprocessor together with the 70H.

It then combines the 08H from the I register with the 70H from the external device and puts it into the program counter. The microprocessor program then switches to address 0870H and calls the telephone maintenance engineer.

When the microprocessor is interrupted, it stores information internally about what it was doing at the time of the interrupt so that when the interrupt is dealt with it can return to its previous task.

Interrupt priorities

Not all problems are treated equally, by us or by a microprocessor. It is unlikely that we would ever meet a sentence like: 'I have forgotten to put any sugar in the coffee – I'll go and get some – and I have also noticed that your house is on fire'.

As with all other microprocessor, Z80180 interrupts are partly generated by external circuits and some result from internal sources. The external ones can usually be blocked or 'masked' so we can tell the microprocessor to ignore them and all interrupts are placed in order of priority so multiple interrupts are prioritized.

In the case of the Z80180, an unrecognized instruction code is given top priority to prevent any random operation due to corruption during programming or transmission. This is called a TRAP.

After this comes a single non-maskable interrupt or NMI. This is used for critical situations that must interrupt any other program that is running.

Then follows three levels of external but preventable (maskable) interrupts. If we don't want the program interrupted, all we have to do is to insert a 'don't interrupt' code into the software. Some documentation refers to this type of interrupts as INT0, INT1, INT2, where the figure show the priority whereas as others use IRQ to stand for Interrupt ReQuest.

The remaining seven levels refer to internal interrupts generated by interrupt control registers and internal circuitry may need to interrupt the program for a moment in order to perform some other task.

There will be a little more on interrupts in Chapter 17.

Power saving

The power consumed by a microprocessor is mainly the result of internal activity so the more we make it do, the more power it uses. Changing the clock speed from 6 MHz to 33 MHz nearly doubles the power requirement.

We can add the 'sleep' instruction to the program and this has the effect of stopping the CPU clock and the data and address buses are disconnected. To wake the CPU up, we apply a signal to one of the interrupt inputs or activate the reset circuit by holding the voltage on the reset pin low.

We can also switch off the input/output circuitry on the microprocessor putting it into a 'system stop' mode.

The Z8X180 family has three members: Z80180, Z8S180 and the Z8L180. The S version operates with a power supply of 5 volts and the L version supply is reduced to 3.3 volts.

The effect on the power consumption of switching between normal operation and 'system stop' mode is shown in Table 8.1.

Bus state controller

In a system, the microprocessor is connected to all the surrounding circuits by a series of connections called the system bus. Sometimes

Table 8.1

	Normal operation	*'System Stop' mode*
Z80180	100 mW	25 mW
Z8S180	90 mW	15 mW
Z8L180	60 mW	6 mW

these devices may wish to send data along the system bus and the bus state controller ensures that we don't have multiple devices trying to send data along the same connections at the same time.

Direct memory access (DMA) controller

As the name may suggest, this circuit provides direct access to the memory without using the central processing unit. The DMA can provide high-speed transfer of data from one part of the memory to another. It can also pass data directly to and from the memory and external devices. This provides significant improvements in operating speed of the whole system.

Dynamic memory refresh

Do you remember that the problem with dynamic memories was the constant refreshing that was needed to keep the RAM data intact? Well, an internal register is used to handle the refreshing process for us. It fits in the job between instructions and does not slow the microprocessor in any way and neither the programmer nor the user is aware that it is continuing in the background. Some microprocessors do not have this register and an external chip is added to the system to perform the refreshing. Either way, it has no obvious effect on the operation of the system.

Wait states

To make it easier to send information to relatively slow external devices the microprocessor is able to insert 'wait states' which can insert more time into its bus cycle timing. They can be programmed in by the software being used or internally by a wait state generator.

Serial inputs and outputs

To move data in or out in serial mode in which the stream of data is applied to a single input connection one bit after another we are given a choice of two options.

First is a clocked serial input/output (CSIO). This is a simple high-speed data connection to another microprocessor that is capable of sending and receiving data, though not at the same time. The transmission is synchronized to the microprocessor clock.

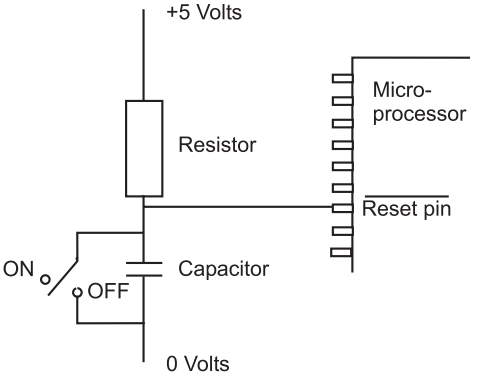
There are also two asynchronous serial communications interfaces (ASCI). These provide two other data connections that can be programmed to select the required speed of transmission and provide two-way transmission.

How the system works

In the beginning

After the power supplies are first switched on, there is a short delay built in to allow the voltages to settle and for the clock to start. This delay is produced by a circuit similar to that shown in Figure 8.9.

Figure 8.9
The ON/OFF switch



The circuit uses a capacitor. A capacitor is a device designed to store electricity rather like a bucket can be filled with water.

When the micro system is switched off, the capacitor has a short circuit across it and the current that is flowing through the resistor bypasses the capacitor hence there is no electricity stored and hence there is no voltage across it. The same effect is achieved by holding the reset pin at zero volts for a time equivalent to at least six clock cycles.

Then we switch on and the current flowing through the resistor now accumulates in the capacitor and the voltage starts to increase. The voltage grows in the way shown in Figure 8.10 and after a short period of time, less than a microsecond, the voltage reaches two volts, which is enough to switch the microprocessor on.

The clock starts ticking and the microprocessor follows a sequence of steps called a start-up microprogram that was built into the micro-processor by the manufacturer.

It performs some internal tests and similar housekeeping jobs and then puts an address on the address bus. This address is, reasonably enough, called the startup address and is again fixed by the manufacturer. In the Z80180 the startup address happens to be 0000H and so this part of the memory map must contain some ROM to hold the start-up program.

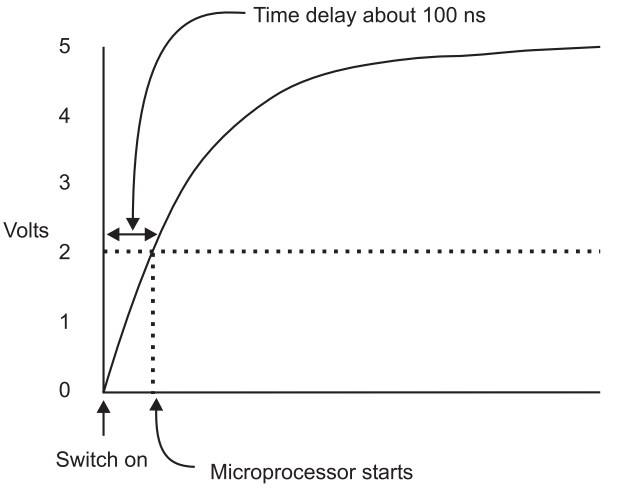


Figure 8.10
Generating a short delay when switching on

Back at Figure 6.22 we saw a typical memory map in which the start-up program was held in the high end of the memory. The Z80180 would require a quite different map and this indicates yet another incompatibility between microprocessors.

The first instruction

The startup address is read by the address decoder. The decoder then applies a chip select signal on the Control bus. This signal then selects a ROM chip. All other chips ignore the signal since they have not been selected. The address enters the ROM chip and the row and address decoders access a memory cell. The contents of that cell are put on the data bus to be read by the microprocessor.

The second read

The microprocessor sends out the next address onto the address bus. The address decoder will activate the ROM chip again. The ROM chip will put some more data on the data bus, which will again be read by the microprocessor.

What does the microprocessor do with all this data?

If the first read of the ROM resulted in the data C0H and the second read produced 86H, the microprocessor combines the two to produce a 16-bit address, C086H. Remember that all this will be in binary – the use of hex is just to help us see what is happening.

The microprocessor places this new address C086H on the address bus. The choice of C086H is not made by the microprocessor designer.

This number is chosen by the system designer – that is, the person that incorporates the microprocessor into a microsystem.

Then what?

The address, C086H in our example, accesses a ROM chip, which sends a binary number back to the microprocessor along the data bus. Inside the microprocessor, this number is interpreted as an instruction. Comparing the incoming number with an internal list of built-in control codes discovers exactly what the instruction is. This is all performed by the micro-program.

What happens next will depend on what the instruction was.

Why?

Why was it an instruction? Because the microprocessor said so! Microprocessors assume that the first binary number that arrives represents an instruction so if we wanted to give it the instruction ADD 25H then the binary code meaning ADD will go in first, followed by the data to be used, in this case 25H.

What if we made a mistake and put the 25 in first, followed by the binary code for ADD in our program?

The microprocessor would interpret the 25 as an instruction – which could mean anything at all, or nothing. If this mystery instruction needed a number, it would use the binary equivalent of our ADD input – which again could be anything. If the false instruction did not need a number, then ADD instruction would be correctly read as an ADD instruction but it would then take the next available binary input as the number to be added. Our microprocessor has now carried out an incorrect instruction using incorrect data and the program could do almost anything. The whole program has got out of step and may do something quite unexpected. Not much fun in our dynamite factory.

The address being used at any time is known to the microprocessor by referring to the current value in the program counter, which is increased by one every time an instruction or some data is used.

The fetch–execute cycle

This is the order of operations by a microprocessor and is the cause of all the confusion in the last paragraph.

The microprocessor applies no intelligence at all. It follows the pattern shown in Figure 8.11 regardless of whether it is following the program or it has been fed with a program with an error in it and is now carrying out a totally useless set of random instructions.

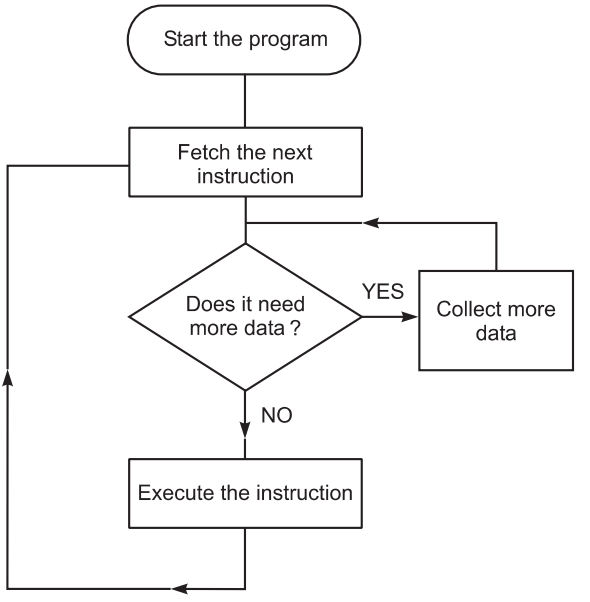


Figure 8.11
The fetch–execute cycle

It will follow our instructions so don't blame the microprocessor – it's up to us to feed it with something sensible. Always remember GIGO – garbage in, garbage out.

As we have seen, there is nothing inherently different between an instruction and data. They are both binary numbers and the interpretation is only a matter of what the microprocessor is expecting at that particular time.

Quiz time 8

In each case, choose the best option.

1 The stack is:

(a) an abbreviation for stack pointer.
(b) a series of RAM locations that can be used by the microprocessor to store data.
(c) a collection of programs.
(d) a chimney.

2 A CPU:

(a) is an essential part of any microprocessor.
(b) is an abbreviation for computer processing unit.
(c) can contain more than one MPU.
(d) is a function of a register.

3 Adding the binary numbers 1100 1100 to 0010 1001 within a Z80180 microprocessor would result in C and H flags being in the condition:

(a) C clear and H clear.
(b) C clear and H set.
(c) C set and H clear.
(d) C set and H set.

4 The interrupt vector register (I) in the Z80180 microprocessor is used to store the:

(a) high byte of the interrupt address.
(b) start up address for the microprocessor.
(c) low byte of the interrupt address.
(d) stack pointer address.

5 The fetch–execute cycle:

(a) works in hexadecimal.
(b) assumes the second, fourth, sixth etc. inputs are data.
(c) assumes the first input is an instruction.
(d) is a system used by the Instruction register to channel information to the correct part of the microprocessor.