A microprocessor-based system

How simple can a microprocessor-based system actually be? It must obviously contain a microprocessor otherwise it is simply another electronic circuit. A microprocessor must be programmed. This means that it must be provided with a series of instructions to be followed. However we program the microprocessor, the result is a series of binary numbers that represent the simple step by step instructions to be followed. These instructions must be stored in some memory. But do the instructions have to be stored in RAM or ROM? It must be in ROM. Remember that RAM will hold random data when first switched on and if our microprocessor was controlling the operation of a dynamite factory, the last thing that we would want is for it to start following random instructions at the rate of a million a second!

What determines how fast the microprocessor carries out the instructions? For the moment we will say that regular pulses of voltage applied to the microprocessor determine its speed. This voltage pulse is called a clock pulse.

The clock

A clock circuit controls the operation of the microprocessor. This produces a series of voltage pulses like a ticking clock. The whole system runs sequentially, doing the required jobs one after the other. One step completed for each tick of the clock system.
The clock circuit can be mostly internal to the microprocessor or it can be entirely external. It is unlike a normal watch or clock in that the exact speed is not important. It may be running at a nominal rate of 200 MHz but if it actually ran at 199 MHz or 201 MHz there would be no great panic. With a modern watch, the equivalent error of seven minutes a day would mean a trip back to the supplier. In Figure 7.1 a typical clock pulse is shown.

Figure 7.1
A 200 MHz clock pulse

The shape of the clock pulse is stipulated in detail in the microprocessor specification. If it does not stay within the limits stated, serious problems can occur. Let’s assume that two circuits have to switch at the same time but they operate at slightly different voltages. In Figure 7.2 the two operating voltages occur at virtually the same moment due to the very fast rise time. In Figure 7.3 the clock pulse has a very slow rise and fall time and Circuit 1 will switch before Circuit 2. In this example, the difference is about 2 ns. Whether or not a 2 ns difference is significant will depend on the circuit being considered.

Figure 7.2
An almost vertical leading edge means zero rise time

1 nanosecond is 1 billionth of a second
A microprocessor-based system

**Figure 7.3**
The rise time is about 2 ns

![Rise Time Diagram](image)

All microprocessor technical data will stipulate the maximum rise time and fall time and between which voltages it is being measured. For example, the Intel 80386 has a supply voltage of 0–5 V and must have a rise time (and fall time) of less than 2 ns when measured between 0.8 V and 4.2 V.

If the rise time (written as $t_r$ or $t_{rise}$) is not shown, it is normally measured between 10 and 90% values of the supply voltage. Each microprocessor has a maximum frequency for the clock pulses and a minimum value.

**Why is there a minimum speed?**

Microprocessors have internal registers and storage areas called capacitors that are like dynamic RAM and need refreshing at intervals. As the clock speed is reduced, the interval between refreshing gets longer until the register or capacitor can no longer hang on to the information and the whole operation collapses. Typically, the minimum clock speed is about a quarter of the maximum speed. The Intel 80386 will run down to 800 kHz.

The first microprocessor, in 1971, was the Intel 4004. It ran at a clock speed of 0.108 MHz and handled 4 bits at a time. In 1998, the 64-bit Alpha 21164 runs at 600 MHz, king for a day but will, no doubt, soon take its place on the museum shelf. (It is 700 MHz now and the ink is barely dry!)

The most basic system is therefore going to include a microprocessor, clock and a ROM chip to provide the built-in sequence of instructions. The only other essential is to have wires or other conductors to connect the circuit. Figure 7.4 shows the most basic microprocessor-
based system with just the bare essentials. As usual, the power supplies are not shown.

All microprocessors are connected internally and to the rest of the system by collections of conductors called buses.

**Buses**

These conductors are in groups, since many will be going to the same place for much the same purpose. For example, an 8-bit microprocessor normally uses 8 connectors to carry the data between the microprocessor and the memory. It would make diagrams very complicated if each wire were to be shown separately so we group them together and refer to them as the data bus. A bus is therefore a collection of conductors providing a similar function.

In a microprocessor-based system we have three main buses: the data bus, the address bus and a control bus. The data bus is a two-way bus carrying data around the system. Information going into the microprocessor and results coming out. The address bus carries addresses and is a one-way bus from the microprocessor to the memory or other devices. The control bus is rather different from the other two. It is a somewhat looser collection of conductors. If we look at a microprocessor-based system we can easily see the data and address buses since they consist of many parallel connections. However, the control bus is just an association of all the other necessary connections such as those to the chip select and read/write pins.

Address information, data and control signals have to be carried around inside the microprocessor as well as in the external system. We will therefore meet internal as well as external buses.
Input/output circuits

The signal on the data bus has only a very low power level and to be of use it must be amplified.

The first response was to put a series of amplifiers; one on the end of each of the connections on the data bus but this was soon superseded by a more sophisticated chip with more facilities. For example, if we wanted to send a signal to a printer at 1 ms intervals it would not be sensible to tie up the main processor with counting out these time intervals. It would be much better to tell the output chip to do its own timing and thus release the microprocessor for more important jobs.

The output devices have become quite complex and now go under a variety of names:

- I/O controller – input/output controller
- PIA – programmable interface adapter
- VIA – versatile interface adapter
- PIO – programmable input/output
- PPI – programmable peripheral interface

. . . and many others.

Whatever name is given to the device, they are basically the same and are essential to any microprocessor-based system. An improved basic system is shown in Figure 7.5.

Figure 7.5
An input/output chip has been added
A practical microprocessor system

Apart from in the most minimal of circuits, some RAM is needed. Even if the microprocessor-based system is controlling an oven, we still need the facility to vary the instructions to change the temperature, the time cycle, the fan speed etc., so some RAM must be added. Some microprocessors have a small amount of RAM included internally, enough for this sort of system to work but still quite limited.

If we add some external RAM, the microprocessor is controlling the operation of three chips: ROM, RAM and I/O. To control the flow of information it needs to send chip select and read/write information along the control bus.

Read/write signals tell the RAM and the I/O chip whether they have to read, i.e. accept information from the data bus or to write information onto the data bus.

Chip select is the on/off switch for each of the chips and we have to be very careful to ensure that only one set of information is being connected to the data bus at one time.

If a ROM chip were to be putting a binary 0 onto one of the data bus connections and, at the same time, another ROM or a RAM chip was applying a binary 1, there would be a disagreement between the two chips. What would happen? It is sad but the likely outcome is a fight to the finish with one or other of the chips being condemned to the waste bin.

To prevent this from happening, an address decoder circuit samples the address bus and selects the appropriate chip. If the microprocessor wishes to send some data to the RAM chip, it applies a suitable address to the address bus that is applied to the ROM, RAM and the I/O controller but there are no problems at the moment since all these chips are switched off. The address decoder applies the inputs from the address bus to an array of logic gates that have been organized to comply with the memory map of the system. The output from the address decoder then switches the RAM chip on and the ROM and I/O chips off. The design of the address decoder can be modified to control any number of external chips in the system. An upgraded system is shown in Figure 7.6.

How it all works

To demonstrate its operation we can ask it to perform a simple task.

Instruction: Send the number 25H which is in the ROM and store it in the RAM at address 2500H. This is what happens – follow the action on Figure 7.6.

1 The microprocessor has to collect the instruction from an address in ROM. It does this by putting the address onto the address bus.
2 The address is applied to the ROM and the RAM as well as the address decoder. This will not cause any problems because all the chip selects will be switched off at the moment. When the logic gates within the address decoder responds to the input from the address bus the result will be that the ROM is switched on and the other two are kept off.

3 Switching on the ROM will mean that it takes in the address from the address bus. Inside the ROM chip, the row and column decoders activate one of the memory locations and the binary number stored at that location is placed on the data bus by switching on the tri-state buffers. As soon as the information is read, the chip select will switch the ROM chip off.

4 The information which is now on the data bus is read by the microprocessor. It is an instruction which can be interpreted as ‘go to address F600H and read the number that is stored in that address’.

5 In response to this instruction, the microprocessor puts the address F600H onto the address bus.
6 The address decoder applies this number to its logic gates and this results in the chip select of the ROM chip being switched on again. The ROM chip accepts the address F600H into its row and column decoders and then puts the number 25H onto the data bus.

7 This number is stored temporarily in the microprocessor.

8 The microprocessor then puts the number 2500H onto the address bus and the address decoder puts a signal on the chip select of the RAM chip to switch it on. It then sends a logic 1 on the read/write line. The RAM is switched on and it is told to read the data on the data bus. The read/write line goes to the I/O chip as well but again, this causes no problem because its chip select line is keeping it switched off.

The number 25 is now safely stored in the RAM chip and will remain there until it is over-written with new information or the power is switched off.

**Another look at the address decoder**

We have seen in Chapter 6 that the number of locations that can be addressed is $2^n$ where $n$ is the number of address lines. By feeding the numbers into our calculators we can see that the relationship between lines and locations is as shown in Table 7.1.

We can also use this table to identify the number of lines needed to access a known number of address locations. For example, if we wanted to access 700 locations, we can see that 9 lines could access 512 locations which is too few. Therefore, we would have to go to 10

<table>
<thead>
<tr>
<th>Number of address lines</th>
<th>Number of locations</th>
<th>Number of address lines</th>
<th>Number of locations</th>
<th>Number of address lines</th>
<th>Number of locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>13</td>
<td>8k</td>
<td>25</td>
<td>32M</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>14</td>
<td>16k</td>
<td>26</td>
<td>64M</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>15</td>
<td>32k</td>
<td>27</td>
<td>128M</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16</td>
<td>64k</td>
<td>28</td>
<td>256M</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>17</td>
<td>128k</td>
<td>29</td>
<td>512M</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>18</td>
<td>256k</td>
<td>30</td>
<td>1024M=1G</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>19</td>
<td>512k</td>
<td>31</td>
<td>2G</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>20</td>
<td>1024k=1M</td>
<td>32</td>
<td>4G</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>21</td>
<td>2M</td>
<td>33</td>
<td>8G</td>
</tr>
<tr>
<td>10</td>
<td>1024 = 1k</td>
<td>22</td>
<td>4M</td>
<td>34</td>
<td>16G</td>
</tr>
<tr>
<td>11</td>
<td>2k</td>
<td>23</td>
<td>8M</td>
<td>35</td>
<td>32G</td>
</tr>
<tr>
<td>12</td>
<td>4k</td>
<td>24</td>
<td>16M</td>
<td>36</td>
<td>64G</td>
</tr>
</tbody>
</table>
A microprocessor-based system

lines which would actually give access to 1024 locations. The ‘real’ answer of 9.45 is not sensible because we cannot have 0.45 of a connecting wire so if 9 is not enough, it will have to be 10.

For those who like to see the calculations, the mathematical result is given by:

\[
\text{number of address lines} = \frac{\log_{10} (\text{number of locations})}{\log_{10}(2)}
\]

**Designing a decoding circuit**

Let’s imagine that we have a microprocessor-based system using eight memory chips, ROM or RAM it doesn’t matter. Each of the chips holds 8 kbytes of memory. From Table 7.1 we can see that an 8 kbytes chip will require 13 address lines in order to access each of their internal locations. Assume too, that the microprocessor that we are using has a 16-bit address bus so we have the situation shown in Figure 7.7. The address lines are numbered from A0 (address line number 0) to A15. The 13 bits A0–A12 are heading off towards the ROM and RAM chips. The remaining three address lines, A13–A15, are used by the address decoder.

![Figure 7.7](image)

**The decoding chip**

The decoder circuit can be made from separate logic gates or can be bought ready-built in a single integrated circuit. For ease of construction, most designers opt for this choice for the result is smaller, dissipates less heat and is less expensive (and it works first time). There is very little to be said for the build-it-yourself approach.

The basic requirements are three input address lines and eight output lines each connected to one of the chip select pins on a memory chip.

To switch the chips on, the chip select must be taken to a logic 0 voltage. A logic 1 voltage level will switch the chip off. It is vital, of course, that only one chip can be switched on at the same time otherwise they will load competing data onto the data bus and are likely to be destroyed. The three addresses can result in \(2^3 = 8\) different inputs to the logic gates.
The internal design ensures that when the address pins are all at zero, the first output goes to a logic 0 and all the others remain high. The memory chip to which this first output is connected is switched on and all the others are off. When the next combination of inputs 0, 0, 1 is applied, the second memory chip is switched on and the others are off. The next combination switches on the next memory chip and so on until the three input wires have switched on each of the memory chips with a single combination of addresses (see Figure 7.8). With three inputs and eight outputs, it is referred to, reasonably enough, as a 3 to 8 decoder.

Table 7.2 looks a lot worse than it really is. It is really just a summary of the decoder chip outputs. If the microprocessor put the address C2F1H on the address bus, then in binary it would be: 1100 0010 1111 0001. It has been broken up into groups of four

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1 0 1 1</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 1 0 1</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 1 1 0</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 1 0</td>
<td>1 1 1 1 0 1 1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1 1 1 1</td>
<td>1 1 1 1 0 1 1</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>1 1 1 1</td>
<td>1 1 1 1 0 1 1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
<td>1 1 1 1 1 0 1</td>
</tr>
</tbody>
</table>
A microprocessor-based system

In Table 7.3, we can see that 13 out of the 16 address lines go to the memory chip and the other three are fed to the decoder chip. The three lines going to the decoder chip carry the data 1 1 0. We can see that the values C = 1, B = 1 and A = 0 occur near the bottom of the table. These values result in Chip 6 receiving a logic 0 value and thus being selected for use. All other chips are deselected by the logic 1.

### Full and partial decoding

#### Full decoding

In the above example, the 8 kbyte memory chips used 13 address lines and the decoder used three. This makes a total of 16 lines used out of a 16-bit address bus. There are no unused lines and this is referred to as ‘full decoding’.

#### Partial decoding

Now let’s make a small change. The memory chips used are 4 kbyte each rather than 8 kbyte. What effect would this have?

The first result would be that the number of address lines going to the chips would be reduced to 12. There are still only eight chips to be selected so a 3–8 decoder is enough. So what have we got now? Twelve address lines to the chips and three to the decoder and one left over and unused. If nothing is connected to this line, then it cannot matter what voltage it carries (see Table 7.4).

We will look at our previous address C2F1H. If it happened to go to a value of 1, the address would change from:

$$1100001011110001 \text{ (C2F1H)}$$

to

$$110000101011110001 \text{ (D2F1H)}.$$
We now have two numbers that can be placed on the address bus which will result in access to the same memory location since all the bits that are actually used are identical. If we instructed the microprocessor to store some information in the address C2F1H and then to recover the information from address D2F1H, we would get the same information again. The address D2F1H is referred to as a ghost address or an image address. It is important to appreciate that ghost or image addresses have no effect at all on the operation of the microprocessor system. They are merely alternative names for a single address. Incomplete or partial decoding always gives rise to image addresses, their number and their addresses are easily worked out. In technospeak, we say that partial decoding results in more than one software addresses pointing to the same hardware address.

**A worked example**

The eight memory chips were now changed to 1 kbyte chips. Some data is stored in the address 4000H. Find (a) how many image addresses will occur and (b) the image addresses.

(a) A 1 kbyte memory chip would have 10 address lines. This was taken from Table 7.1 and the decoder chip still needs three inputs. The 16-bit address we were given, 4000H would result in three unused lines as in Table 7.5. These three binary digits can take on the values 000, 001, 010, 011, 100, 101, 110 and 111 which results in eight different values. This could have been done the slick way by saying $2^3 = 8$. Putting these eight numbers into the address will result in eight different addresses.

One real address and seven ghosts or images. But which is the ghost and which is the real one? It doesn’t matter. We can assume
A microprocessor-based system

anything we like since they all point to the same physical memory. Having said this, most people seem to opt for the lowest address as the ‘real’ one.

(b) Now, to find the actual ghost addresses, all we have to do is to feed in all the binary options, 000, 001 etc. into the ‘real’ address to generate each of the ghost addresses. This is shown in Table 7.6.

The resulting addresses are 4000H, 4400H, 4800H, 4C00H, 5000H, 5400H, 5800H and 5C00H. Strictly speaking, we should, perhaps, only give the last seven addresses since the question asked for the image addresses and the first one is actually the real address. Notice how the addresses increase in a definite pattern. It always works out this way.

Image addresses and the memory map

You may remember the memory maps that we looked at in the last chapter. They showed the addresses taken by the various memory

---

**Table 7.5** There are now three unused address lines

| A15 | A14 | A13 | A12 | A11 | A10 | A9  | A8  | A7  | A6  | A5  | A4  | A3  | A2  | A1  | A0
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

To the decoder ↑ ↑ ↑ Three unused lines ←→ To the 1 kbyte memory chip

**Table 7.6** Generating the image addresses

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100000000 4000</td>
</tr>
<tr>
<td>0100000100 4400</td>
</tr>
<tr>
<td>0100001000 4800</td>
</tr>
<tr>
<td>0100010000 4C00</td>
</tr>
<tr>
<td>0100010100 5000</td>
</tr>
<tr>
<td>0100011000 5400</td>
</tr>
<tr>
<td>0100011100 5800</td>
</tr>
<tr>
<td>0100111110 5C00</td>
</tr>
</tbody>
</table>

97
devices and the unused addresses in between them. When we have partial decoding, it gives rise to image addresses. Now, if we put these image addresses on the memory map a curious thing occurs. Every image address always falls in one of the unused areas on the map so they will never cause problems by clashing with an existing memory chip.

Quiz time 7

In each case, choose the best option.

1. The Digital Alpha 21064 microprocessor has a 34-bit address bus which can access a memory of:

   (a) 64 Gbytes.
   (b) 64 Mbytes.
   (c) 16 Gbytes.
   (d) 160 Mbytes.

2. The data bus:

   (a) is bi-directional.
   (b) consists of eight conductors.
   (c) can be used to carry data to RAM and ROM memory chips.
   (d) is connected to the chip select pins.

3. A 16-bit address bus is carrying the address 4567H and is partially decoded with lines A15 and A13 being unused. Which one of these addresses would access a different hardware location:

   (a) E567H.
   (b) 4567H.
   (c) 6567H.
   (d) 8567H.

4. Which one of the following is NOT essential in a microprocessor-based system:

   (a) an address bus.
   (b) ROM.
   (c) a clock signal.
   (d) RAM.

5. Image addresses are:

   (a) also called ghost addresses.
   (b) due to several hardware addresses pointing to the same software address.
   (c) the same as partial addresses.
   (d) caused by full decoding.