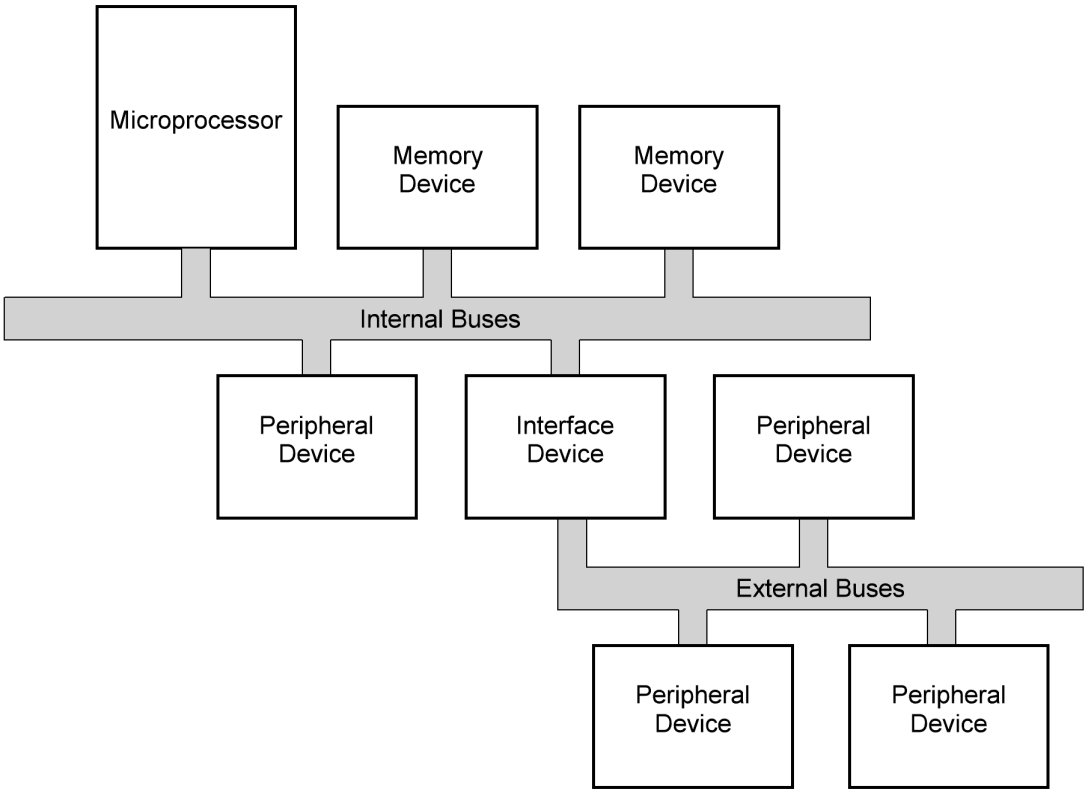
Chapter 3
Microprocessor Systems

Latest update: 04/03/2024

Table of Contents

I. Introduction

A microprocessor system is made up of at least one microprocessor, some memory devices and a variety of peripherals. The main function of the microprocessor is to control the whole system.



A bus is a collection of wires used to transfer information between components inside a computer. For instance, an 8-bit bus is made up of 8 wires and can transfer 8-bit values.

A microprocessor has three internal buses (an address bus, a data bus and a control bus) that allow it to communicate with a large range of memory and peripherals. However, interfaces are commonly used to extend the ability of the microprocessor to communicate with external equipment. Not only can these interfaces provide enough electrical energy to power a lot of memory and peripheral devices, but they also allow standardizing some external buses that can be used to connect equipment from different makers (e.g. USB, PCI bus, ISA bus).

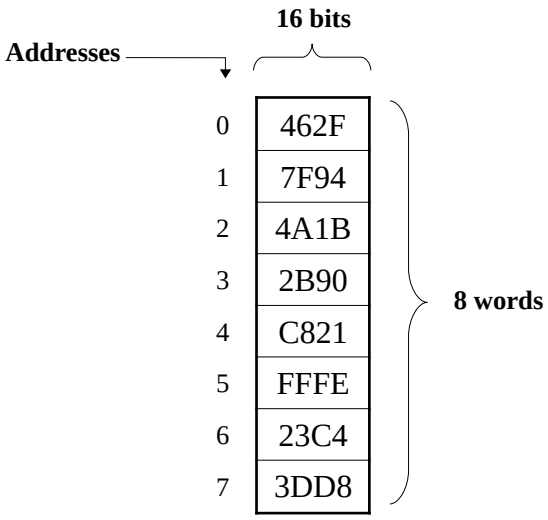
II. Memory Devices

1. Definitions

A memory device is made up of electronic components and allows storing digital data in a binary-word form. A binary word is a binary number expressed in a given number of bits (e.g. a byte is an 8-bit binary word).

A memory device can be seen as an array of cells. Each cell contains a binary word and can be located by a **unique address number**.

For instance, the following memory contains **8 words of 16 bits**:



For example:
- The address **0** contains **$462F_{16}$**.
- The address **4** contains **$C821_{16}$**.

A memory device can be defined by its **number of words** (or **addresses**) and by the **size (in bits) of each word**. These values are commonly called the **depth** and the **width** of the memory respectively.

We can also express the capacity of a memory device in bits or in bytes; that is to say, the number of bits or the number of bytes contained in the memory.
- **Capacity in bits** = Depth × Width
- **Capacity in bytes** = Depth × Width / 8

Therefore, the memory above, which contains 8 words of 16 bits, has:
- a capacity of 128 bits (8 × 16).
- a capacity of 16 bytes (8 × 16 / 8).

2. Main Types of Memory Devices

2.1. Random-Access Memory (RAM)

RAM devices have the following main characteristics:
- They are volatile: data is lost whenever the power is switched off.
- Data can be either read or written.

RAM devices come in a wide variety of classes. We will touch on the two main ones only.

2.1.1. Static Random-Access Memory (SRAM)

SRAM devices use flip-flops as storage locations.

They are very fast and low-powered but unfortunately still expensive.

The cache memory of a computer is usually made up of SRAM devices.

2.1.2. Dynamic Random-Access Memory (DRAM)

DRAM devices use the capacitive effect of transistors as storage locations.

The main drawback of this storage technology is that data can be stored for a short time only. Therefore, the electric charge has to be refreshed periodically, which requires a dedicated circuit.

However, the big advantage of dynamic memory is its high density, which is much higher than that of static memory. That is why dynamic memory is much cheaper than static memory.

The main memory of a computer is usually made up of DRAM devices.

2.2. Read-Only Memory (ROM)

ROM devices have the following main characteristics:
- They are non-volatile; data is not lost whenever the power is switched off.
- Data can be read but not modified.

The BIOS (Basic Input Output System) of a computer is usually stored in a ROM device.

ROM devices also come in a wide variety of classes. We will touch on the main ones only. As we said previously, a ROM device can be read from but not written to; they are programmed by the manufacturer. However, some subclasses of ROM can be written to by specific electronic devices.

2.2.1. Programmable Read-Only Memory (PROM)

PROM devices can be written to only once. Their content cannot be erased. Once they have been programmed, they can be used as ROM devices.

2.2.2. Erasable Programmable Read-Only Memory (EPROM)

EPROM devices can be programmed and erased a great number of times. Once they have been programmed, they can be used as ROM devices.

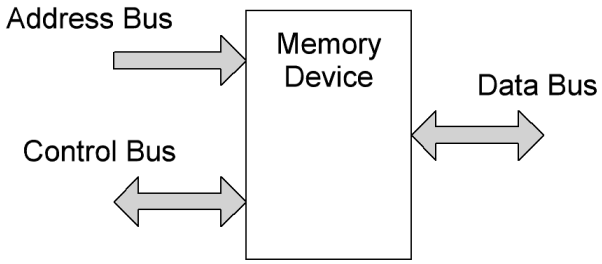
The two main subclasses of EPROM are :

- **UV-EPROM (UV Erasable Programmable Read-Only Memory):** This kind of memory can be erased by being exposed to ultraviolet rays for some time. They are no longer commonly used.

- **EEPROM (Electrically Erasable Programmable Read-Only Memory):** This kind of memory can be programmed and erased electrically.

3. The Buses

A memory device has three buses:



3.1. The Address Bus

The address bus is used to contain the address of the memory cell we want to access. It is an input.

If *a* is the number of wires of the address bus, we can deduce that: **depth = 2^a.**

3.2. The Data Bus

The data bus is used to transfer the data of the memory.

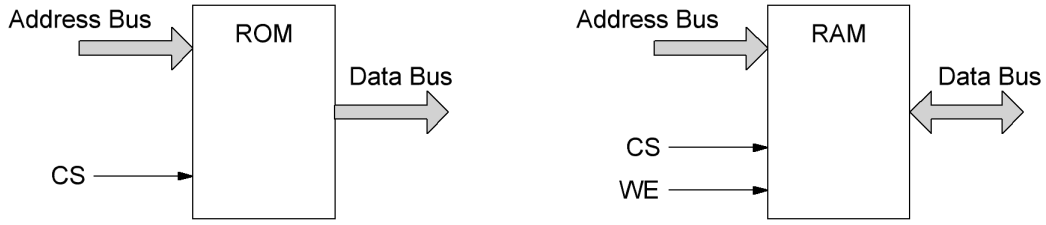
This bus is unidirectional (output) on ROM devices (data can only be read) and bidirectional (input and output) on RAM devices (data can be either read or written).

If *d* is the number of wires of the data bus, we can deduce that: **width = *d*.**

3.3. The Control Bus

The control bus is used to convey a variety of control signals that synchronize the data exchange between the memory and the other components. These signals can be either unidirectional or bidirectional and their number depends on the features of the memory.

In this chapter, we will assume that the control bus will be reduced to the minimum. Therefore, here is what RAM and ROM devices should look like:



3.3.1. Chip Select (CS)

A ‘chip select’ input can be found on every type of memory device. This input allows activating or deactivating the device. We will call it ‘*CS*’, but some manufacturers can use different names; for instance ‘*CE*’, which stands for ‘chip enable’.

When a memory device is deactivated, it ignores the voltage on its inputs and disconnects its outputs. It means that the outputs are neither 0 nor 1; they are in a third state called ‘high impedance’. We can consider that the device is completely disconnected.

3.3.2. Write Enable (WE)

A ‘write enable’ input can be found on every RAM device but does not exist on ROM devices. This input allows selecting the access mode of the device (read or write mode). We will call it ‘*WE*’, but some manufacturers can use different names; for instance ‘$\overline{R}/W$’, which stands for ‘$\overline{\text{Read}}$ / Write’.

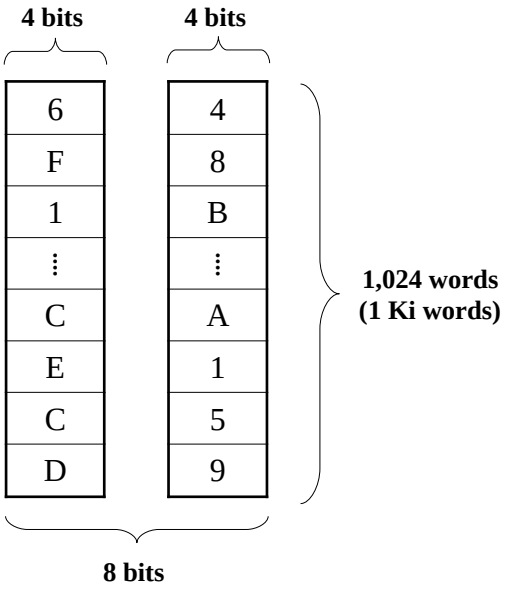
4. Connecting Memory Devices

Memory devices can be connected together in order to enlarge their capacity, their width and their depth. These kinds of connections can be found on SIMM and DIMM memory modules.

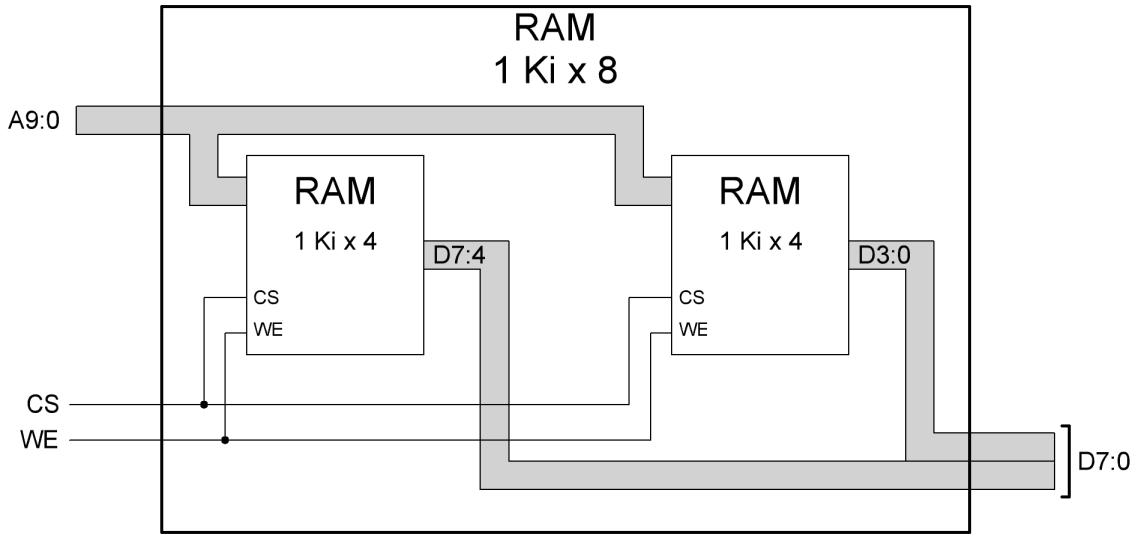
4.1. Connecting Memory Devices in Parallel

Connecting memory devices in parallel enlarges the width of the memory, that is, the size of the binary words. Therefore, this type of connection enlarges the data bus.

For instance, let us take two RAM devices with a 10-bit address bus and a 4-bit data bus each. Both of them have a depth of 1,024 (2^{10}) and a width of 4.



We can see these two 4-bit-wide memories as only one 8-bit-wide memory. Practically speaking, this can be achieved by connecting the devices as shown below:



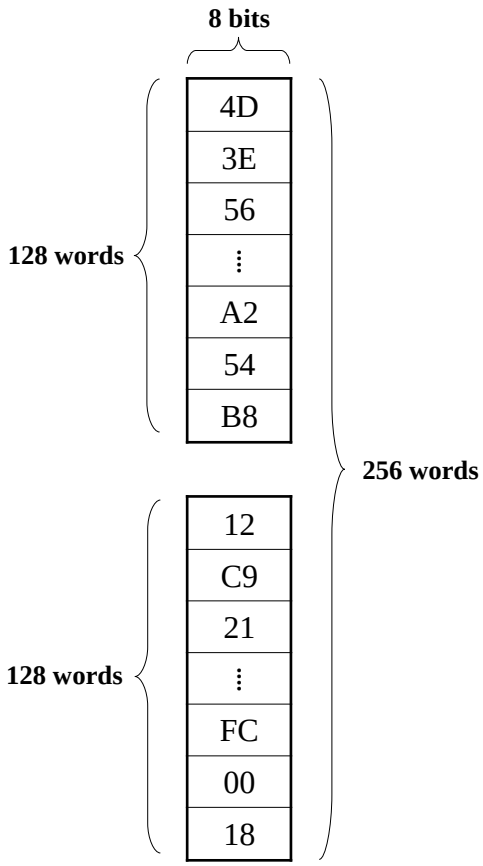
It is noteworthy that:

- The address buses of the internal memories are connected. They make up the address bus of the external memory.
- The control buses of the internal memories are connected. They make up the control bus of the external memory.
- The data buses of the internal memories are not connected. They are placed side by side and make up the data bus of the external memory.
- Both internal memories have to be activated at the same time in order to form a complete 8-bit value on the data bus of the external memory.

4.2. Connecting Memory Devices in Series

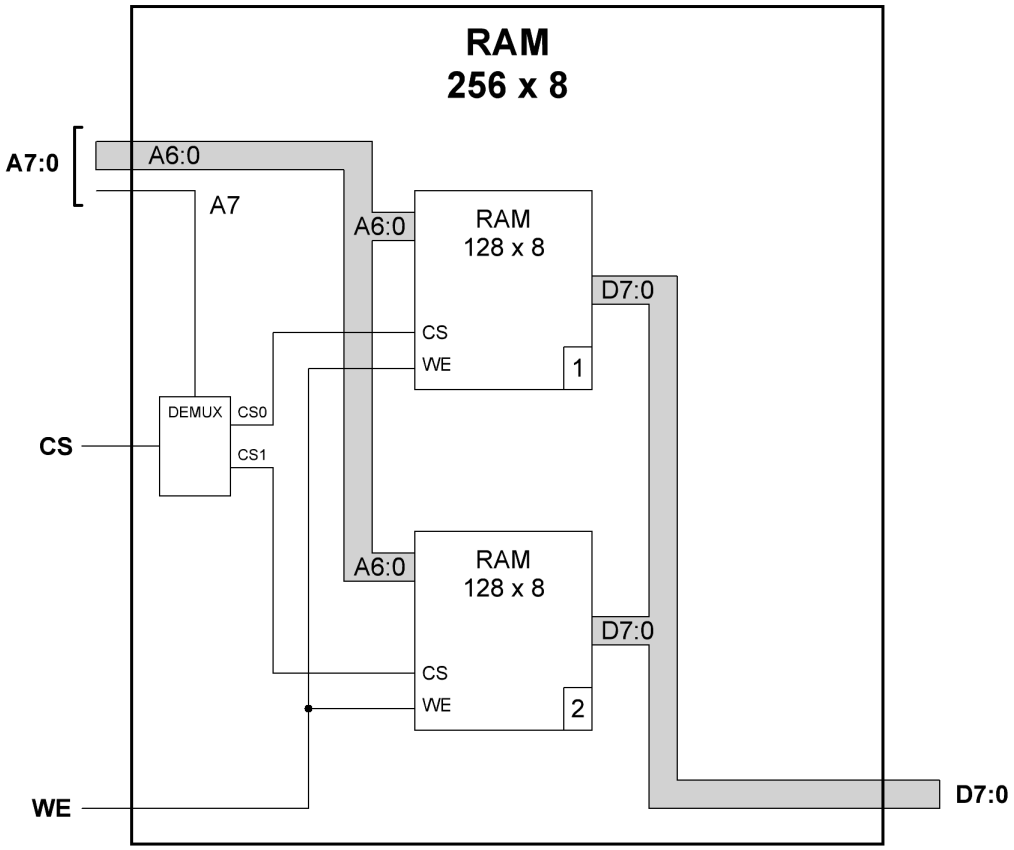
Connecting memory devices in series enlarges the depth of the memory, that is, the number of words. Therefore, this type of connection enlarges the address bus.

For instance, let us take two RAM devices with a 7-bit address bus and an 8-bit data bus each. Both of them have a depth of 128 (2^7) and a width of 8.



We can see these two memories (with a depth of 128 each) as only one memory (with a depth of 256).

Practically speaking, this can be achieved by connecting the devices as shown below:



It is noteworthy that:

- The data buses of the internal memories are connected. They make up the data bus of the external memory.
- Apart from the *CS* inputs, the control buses of the internal memories are connected. They make up the control bus of the external memory.
- The address buses of the internal memories are connected. They make up the least significant address bits of the external memory.
- The external memory needs an additional address bit in order to get the 256 addresses. This bit can be used with a multiplexer to select one or the other internal memory.
- The internal memories must not be activated at the same time in order to avoid access conflicts on the data bus. The *CS* of the external memory is sent to one of the internal memories while the other one is deactivated.

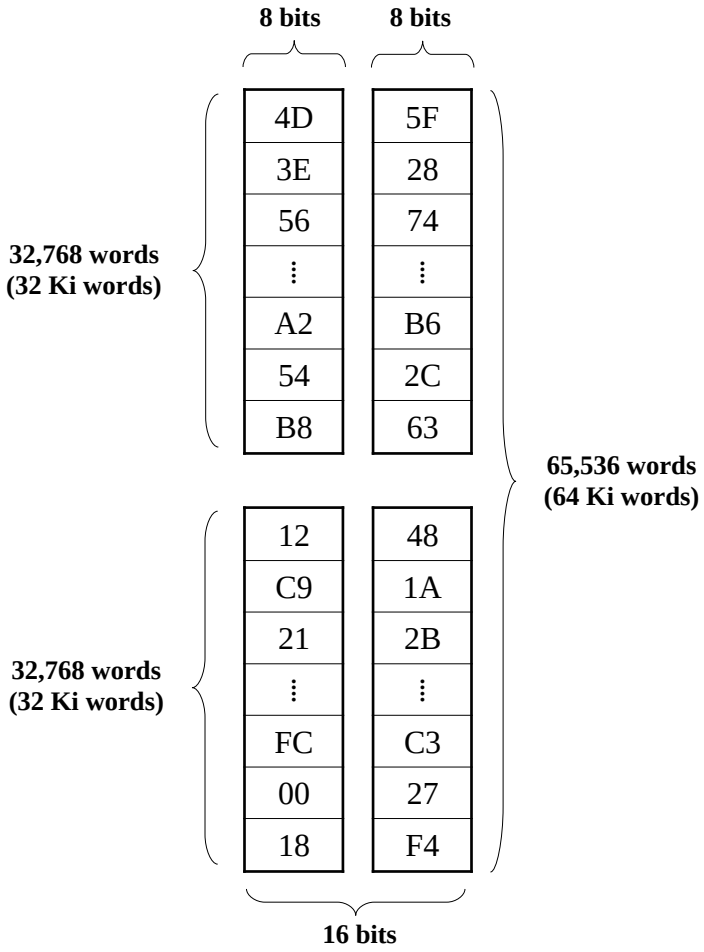
When the *CS* of the external memory is 1:

- If *A7* = 0 then *CS0* = 1 and *CS1* = 0 → the memory device number 1 is activated.
- If *A7* = 1 then *CS0* = 0 and *CS1* = 1 → the memory device number 2 is activated.

When the *CS* of the external memory is 0, every internal memory is deactivated.

4.3. Connecting Memory Devices in Parallel and Series

We can also connect memory devices in parallel and series at the same time. For instance, let us take four RAM devices with a 15-bit address bus and an 8-bit data bus each. All of them have a depth of 32,768 (2^{15}) and a width of 8.



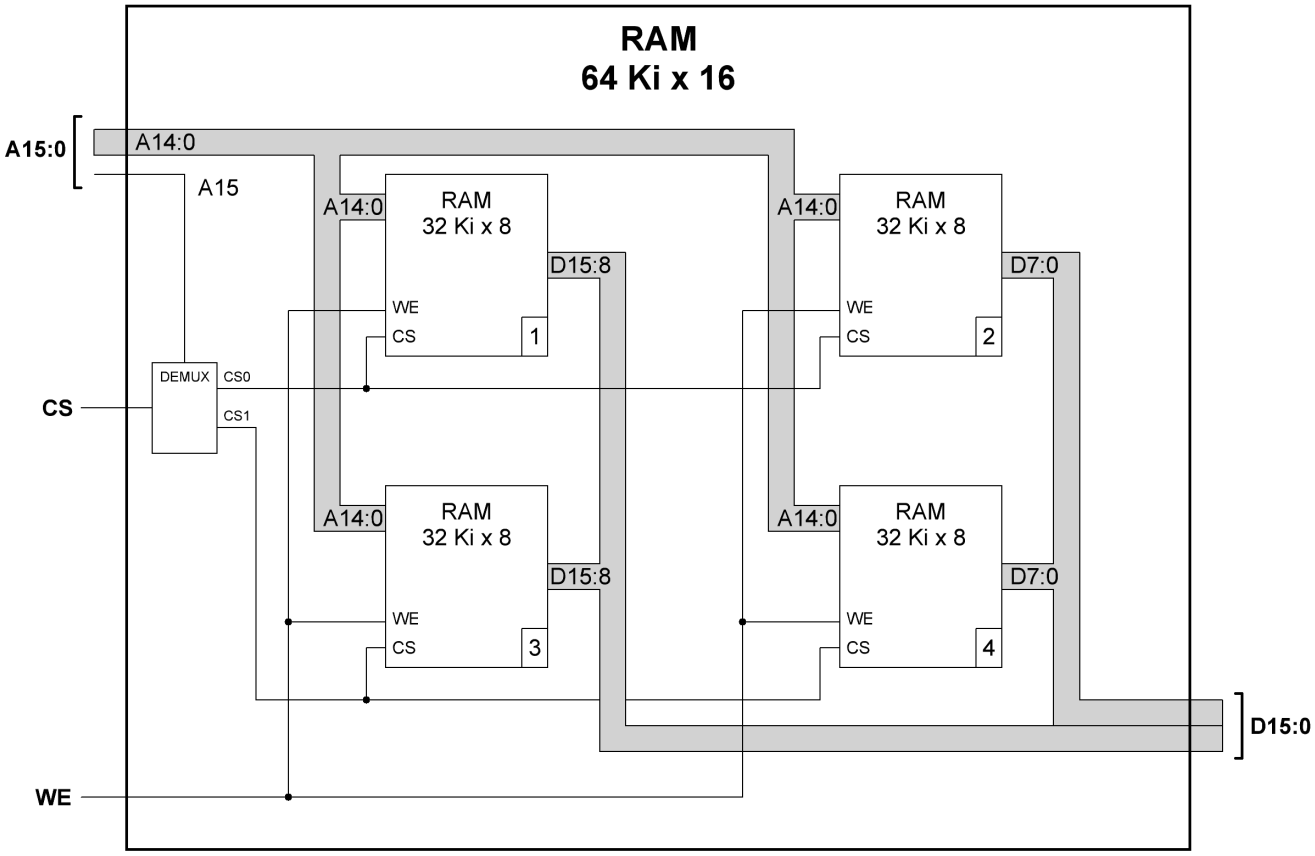
We can see these four memories as only one memory with a depth of 65,536 and a width of 16.

The two memories at the top of the diagram have to be activated at the same time in order to form a complete 16-bit value (for instance, the address 0 contains $4D5F_{16}$).

The same goes for the two memories at the bottom of the diagram (for instance, the address 65,535 contains $18F4_{16}$).

When the two top memories are active, the two bottom memories have to be inactive or vice versa, failing which, two different values will be sent to the data bus, which will lead to access conflicts.

Practically speaking, this can be achieved by connecting the devices as shown below:



We can easily identify the parallel and series connections:

- Memories 1 and 2 are in parallel. They have to be active at the same time (they have the same *CS*).
- Memories 3 and 4 are in parallel. They have to be active at the same time (they have the same *CS*).
- Memories 1 and 3 are in series. They must not be active at the same time.
- Memories 2 and 4 are in series. They must not be active at the same time.

When the *CS* of the external memory is 1:

- If *A15* = 0 then *CS0* = 1 and *CS1* = 0 → the memory devices number 1 and 2 are activated.
- If *A15* = 1 then *CS0* = 0 and *CS1* = 1 → the memory devices number 3 and 4 are activated.

When the *CS* of the external memory is 0, every internal memory is deactivated.

III. Address Decoding

1. Introduction

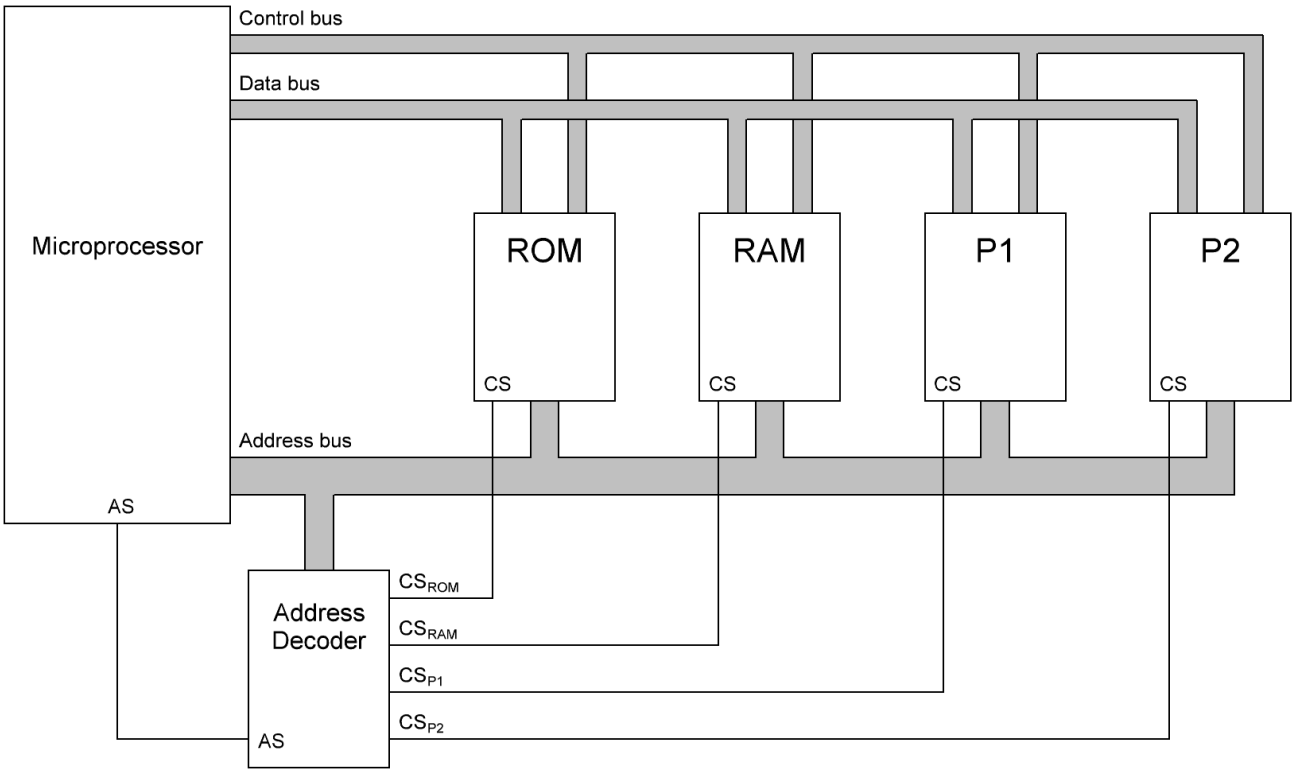
The purpose of the address decoding is to allow the microprocessor to communicate with more than one device.

In order to illustrate the main techniques of address decoding, we will go through a practical example.

Let us assume that a microprocessor needs to communicate with a ROM device, a RAM device and two peripheral devices (**P1** and **P2**).

Device	Address Bus (bits)	Data Bus (bits)	Capacity (bytes)
Microprocessor	20	8	1 Mi
ROM	16	8	64 Ki
RAM	15	8	32 Ki
P1	10	8	1 Ki
P2	4	8	16

The microprocessor communicates with the devices through the data bus, the address bus and the control bus. The address decoding requires an address decoder.



The outputs of the address decoder are the *CS* signals for all the devices. Its inputs are the address bus and the *AS* signal. The *AS* signal is generated by the microprocessor – it is part of its control bus – and *AS* stands for 'Address Strobe'. This signal is active (*AS* = 1) when the address on the address bus of the microprocessor is valid. Therefore, when it is inactive (*AS* = 0), none of the devices should be selected. This is why the address decoder has to take the *AS* signal into account.

Every device that needs to communicate with the microprocessor has to have a 'chip select' input because the data bus can be accessed by only one device at a time. Otherwise, access conflicts will occur.

The microprocessor uses its address bus to select a device.

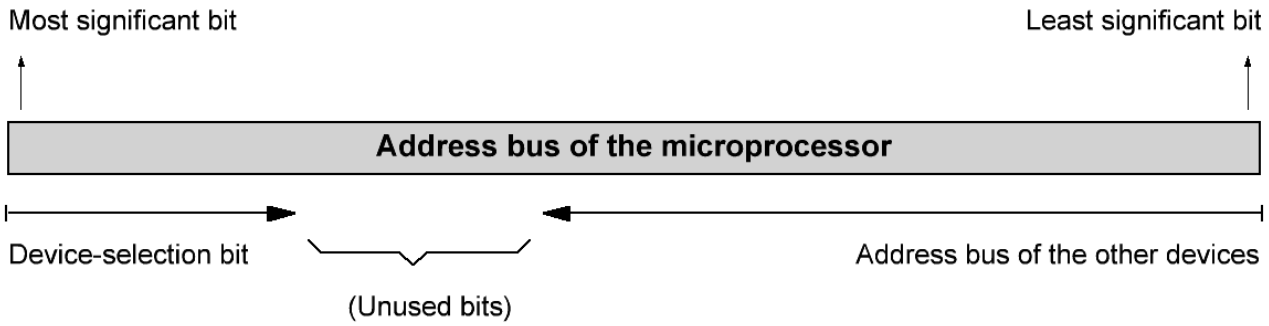
Address decoding is the method of generating CS signals from the address bus of the microprocessor; that is to say, a method of selecting a device from an address value.

The address decoder generates the *CS* signals for all the devices according to the value of the address. The value of the address is decoded so that the right device is selected.

There are several address-decoding techniques. So we will limit ourselves to two techniques: the **linear address decoding** and the **block address decoding**. That will be enough to grasp the key principles of address decoding.

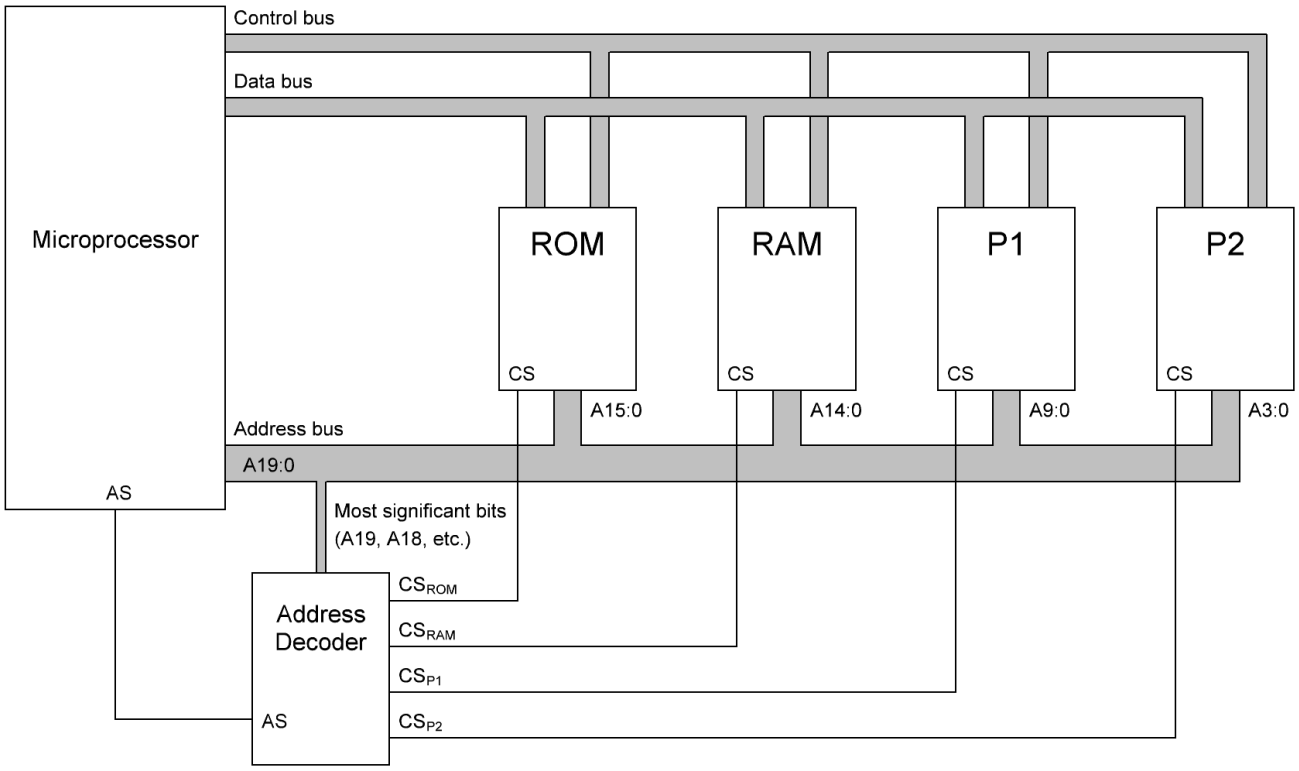
Let us start by examining the common points of these two techniques.

The address bus is split into two sections:
- The most significant bits are used to generate the *CS* signals for all the devices.
- The least significant bits are used to select the address of a device.



In some configurations (number of devices, size of the address bus, etc.), some bits of the address bus may be unused. For the time being, these bits will be set to zero.

We can now specify the numbers of lines for the different address buses:



- The least significant address bits of the microprocessor are connected to the address buses of the different devices.
- The most significant address bits of the microprocessor are the inputs of the address decoder. They are called ‘device-selection bits’ and their number depends on which address-decoding technique is used.

2. Linear Address Decoding

The key principle of the linear decoding is to pair one address bit with one device (starting from the most significant bit).

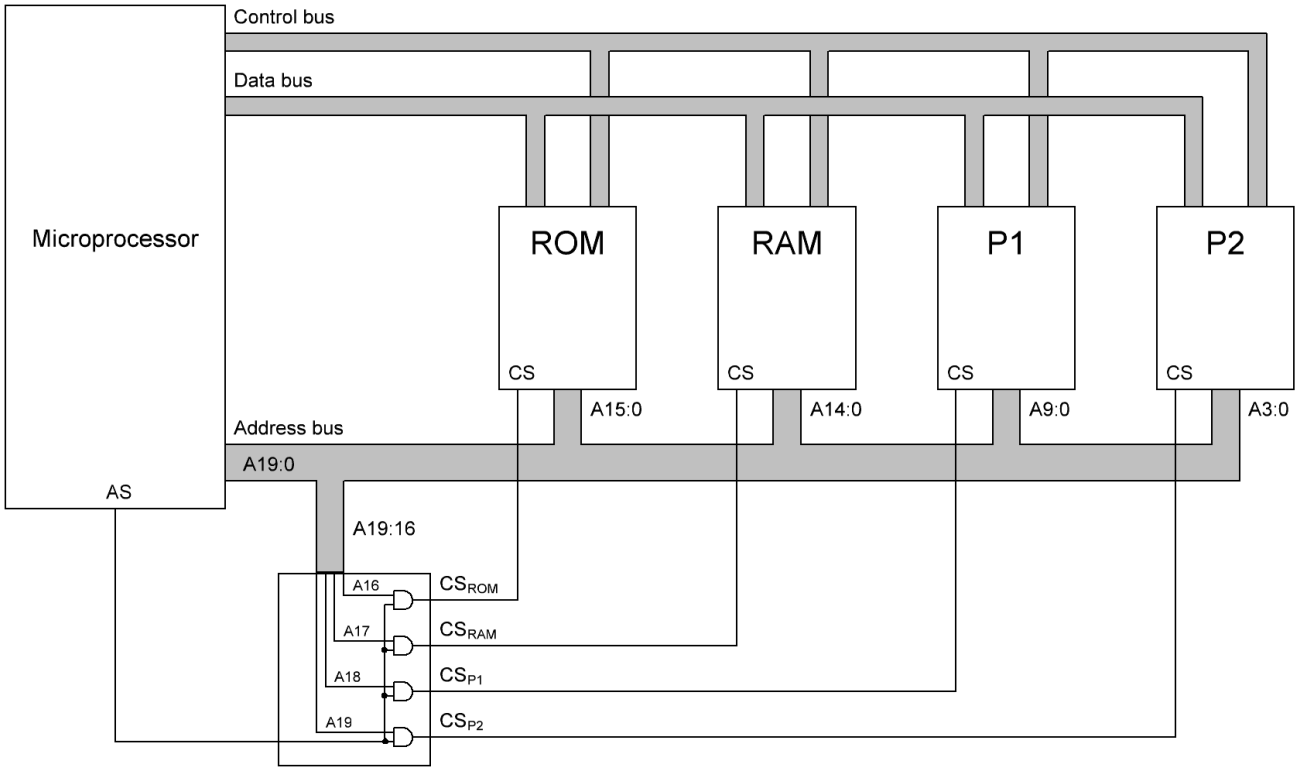
Therefore, in our example, four devices require four device-selection bits: *A19*, *A18*, *A17* and *A16*. Each of these bits has to be associated with a device. Out of context, the choice to associate a bit with one device or another is arbitrary. Designers will make a choice according to their needs.

Address Bit	Device	
A16	ROM	← When *A16* is 1, the ROM has to be activated.
A17	RAM	← When *A17* is 1, the RAM has to be activated.
A18	P1	← When *A18* is 1, P1 has to be activated.
A19	P2	← When *A19* is 1, P2 has to be activated.

2.1. The Address Decoder

A linear address decoder is very simple. The *CS* input of a device has to be set to 1 when its associated device-selection bit is 1. Also, as we have previously said, when the value on the address bus is invalid (*AS* = 0), none of the devices should be activated.

CS_{ROM} = AS.A16
CS_{RAM} = AS.A17
CS_{P1} = AS.A18
CS_{P2} = AS.A19



2.2. The Memory Map

A memory map is a representation of the memory space. It shows the location of the devices, that is to say, all the addresses that can be used by the microprocessor to access the different devices.

Therefore, we need to determine the lowest and highest addresses of each device in the memory space.

To achieve this, we have to determine the most significant address bits (the device-selection bits) and the least significant address bits (the address of the selected device) of the microprocessor for each device. Any unused address bits will be set to 0.

Take the ROM for instance:

- *A16* is set to 1 to select the ROM device.
- *A17*, *A18* and *A19* are set to 0 to deactivate all the other devices.
- To obtain its lowest address, its 16 address bits are set to 0.
- To obtain its highest address, its 16 address bits are set to 1.

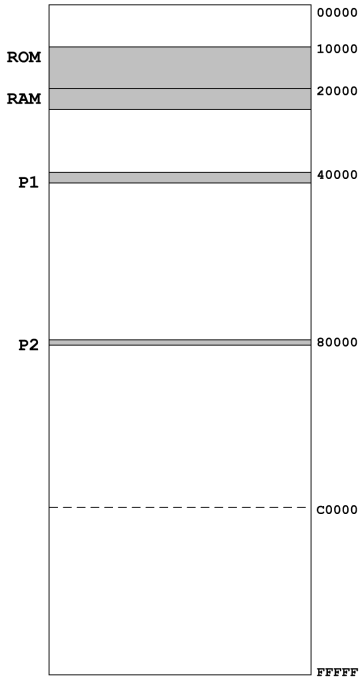
Here is what the memory map should look like:

ROM (lowest): $\mathbf{0001\ 0000\ 0000\ 0000\ 0000}_2 = 10000_{16}$
ROM (highest): $\mathbf{0001\ 1111\ 1111\ 1111\ 1111}_2 = 1FFFF_{16}$

RAM (lowest): $\mathbf{0010}\ 0000\ 0000\ 0000\ 0000_2 = 20000_{16}$
RAM (highest): $\mathbf{0010}\ 0111\ 1111\ 1111\ 1111_2 = 27FFF_{16}$

P1 (lowest): $\mathbf{0100}\ 0000\ 0000\ 0000\ 0000_2 = 40000_{16}$
P1 (highest): $\mathbf{0100}\ 0000\ 0011\ 1111\ 1111_2 = 403FF_{16}$

P2 (lowest): $\mathbf{1000}\ 0000\ 0000\ 0000\ 0000_2 = 80000_{16}$
P2 (highest): $\mathbf{1000}\ 0000\ 0000\ 0000\ 1111_2 = 8000F_{16}$



For instance, when the microprocessor sends the value $207F2_{16}$ to its address bus, the address $7F2_{16}$ of the RAM is selected.

Address bus of the microprocessor

	A19 P2	A18 P1	A17 RAM	A16 ROM	A15	A14:0
207F2 →	0	0	1	0	0	000 0111 1111 0010
	↓				↓	↓
	The RAM device is selected				Unused bit	Address bus of the RAM : 7F2

2.3. Conclusion

Here are some aspects of the linear address decoding:

- It can be built easily: the address decoder is a very simple combinational circuit.
- The number of devices that can be connected is extremely limited: using one bit per device is not the best option.
- There is no protection against access conflicts.

Notes:
The main drawback of the linear-decoding technique is that several devices can be activated at the same time, which could lead to access conflicts and may cause severe damage.

Therefore, the address decoder should be changed so that only one device can be selected at a time. When the address decoder activates a device, it should deactivate all the other devices. In other words, the *CS* input of a device should be activated only when its device-selection bit is 1 and those of the other devices are 0.

$CS_{ROM} = AS.\overline{A19}.\overline{A18}.\overline{A17}.A16$
$CS_{RAM} = AS.\overline{A19}.\overline{A18}.A17.\overline{A16}$
$CS_{P1} = AS.\overline{A19}.A18.\overline{A17}.\overline{A16}$
$CS_{P2} = AS.A19.\overline{A18}.\overline{A17}.\overline{A16}$

3. Block Address Decoding

The block decoding address splits the memory space into several equal-sized blocks. Each block consists of a combination of device-selection bits. The latter are always the most significant address bits.

For instance:
- 1 device-selection bit makes up 2 blocks.
- 2 device-selection bits make up 4 blocks.
- *n* device-selection bits make up 2^n blocks.

1 bit	A19
Block 0	0
Block 1	1

2 bits	A19:18
Block 0	00
Block 1	01
Block 2	10
Block 3	11

In our example, the largest device is the ROM with 16 address lines. Since the microprocessor has 20 address lines, four of them remain available for the selection. Therefore, there are four possibilities:
- 1 device-selection bit → 2 blocks of 512 KiB.
- 2 device-selection bits → 4 blocks of 256 KiB.
- 3 device-selection bits → 8 blocks of 128 KiB.
- 4 device-selection bits → 16 blocks of 64 KiB.

With the block address decoding, one block (a combination of device-selection bits) is paired with one device. So, the four devices require at least four blocks; that is to say, at least two device-selection bits. The number of blocks can be greater than or equal to the number of devices. Some blocks can be unused.

Now, each device has to be paired with a block. Out of context, the choice to associate a block with one device or another is arbitrary. Designers will make a choice according to their needs. Unused blocks can be reserved for future expansion.

We will choose to split the memory space into eight blocks as shown below:

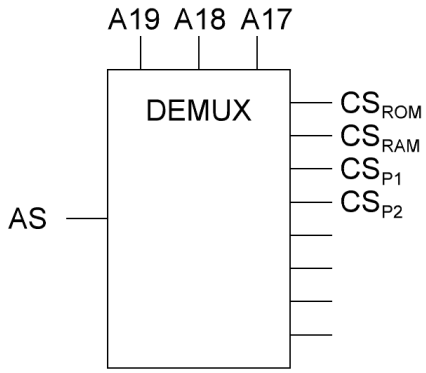
Block	A19	A18	A17	Device
0	0	0	0	ROM
1	0	0	1	RAM
2	0	1	0	P1
3	0	1	1	P2
4	1	0	0	*unused block*
5	1	0	1	*unused block*
6	1	1	0	*unused block*
7	1	1	1	*unused block*

3.1. The Address Decoder

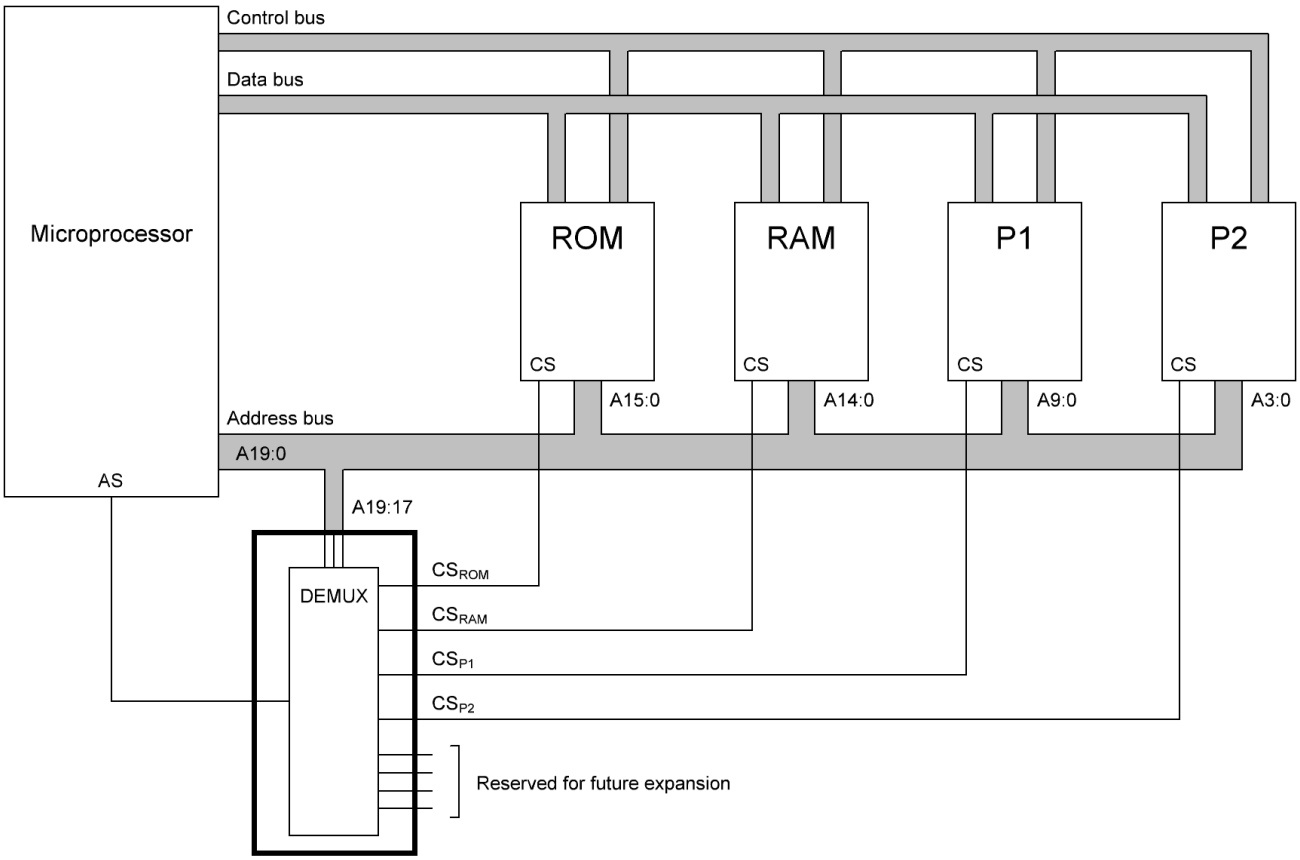
When the value of the device-selection bits matches the combination associated with a particular device, this device should be activated. Also, as we have previously said, when the value on the address bus is invalid (*AS* = 0), none of the devices should be activated.

$CS_{ROM} = AS.\overline{A19}.\overline{A18}.\overline{A17}$
$CS_{RAM} = AS.\overline{A19}.\overline{A18}.A17$
$CS_{P1} = AS.\overline{A19}.A18.\overline{A17}$
$CS_{P2} = AS.\overline{A19}.A18.A17$

It is noteworthy that this address decoder is actually a demultiplexer (3 selected lines, 8 outputs).



Here is what the circuit diagram should look like:



3.2. The Memory Map

We need to determine the lowest and highest addresses of each device in the memory space.

To achieve this, we have to determine the most significant address bits (the device-selection bits) and the least significant address bits (the address of the selected device) of the microprocessor for each device. Any unused address bits will be set to 0.

Take the ROM for instance:

- *A16* is set to 0 because it is unused.
- *A17*, *A18* and *A19* are set to 0 to select the ROM device.
- To obtain its lowest address, its 16 address bits are set to 0.
- To obtain its highest address, its 16 address bits are set to 1.

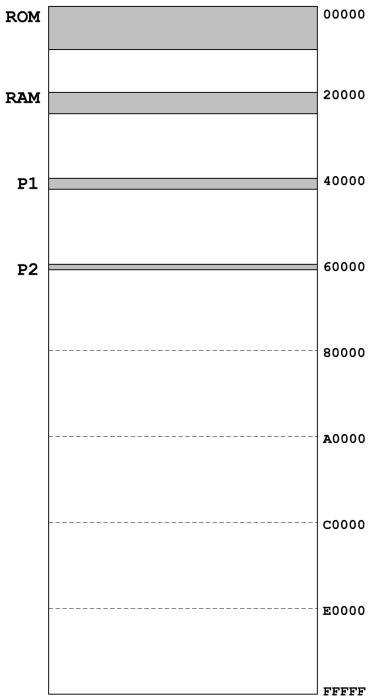
Here is what the memory map should look like:

ROM (lowest): **000**0 **0000 0000 0000 0000**$_2$ = 00000$_{16}$
ROM (highest): **000**0 **1111 1111 1111 1111**$_2$ = 0FFFF$_{16}$

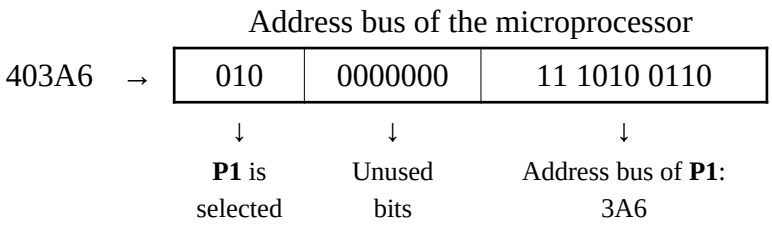
RAM (lowest): **001**0 **0000 0000 0000 0000**$_2$ = 20000$_{16}$
RAM (highest): **001**0 **0111 1111 1111 1111**$_2$ = 27FFF$_{16}$

P1 (lowest): **010**0 0000 0000 **0000 0000**$_2$ = 40000$_{16}$
P1 (highest): **010**0 0000 0011 **1111 1111**$_2$ = 403FF$_{16}$

P2 (lowest): **011**0 0000 0000 0000 **0000**$_2$ = 60000$_{16}$
P2 (highest): **011**0 0000 0000 0000 **1111**$_2$ = 6000F$_{16}$



For instance, when the microprocessor sends the value 403A6$_{16}$ to its address bus, the address 3A6$_{16}$ of **P1** is selected.



3.3. Conclusion

Here are some aspects of the block address decoding:

- It can be built easily: the address decoder is a demultiplexer.
- In comparison with the linear address decoding, more devices can be connected.
- The greater the number of blocks, the smaller the size of a device (and vice versa).
- Access conflicts cannot occur.

4. Memory Mirroring and Redundant Images

When some address bits of the microprocessor are unused, the memory of a device can be found at different places in the memory space.

For instance, the ROM used in the previously block-decoding configuration has the following lowest and highest addresses:
ROM (lowest): $\mathbf{0000\ 0000\ 0000\ 0000\ 0000}_2 = 00000_{16}$
ROM (highest): $\mathbf{0000\ 1111\ 1111\ 1111\ 1111}_2 = 0FFFF_{16}$

We can easily notice that *A16* is unused.

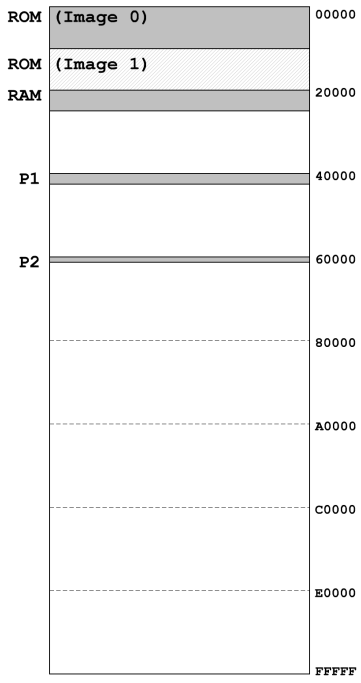
So far, we have set the unused bits to 0, but we can also set them to 1. Whatever the value of these bits, the selected device and the selected address are the same.

For instance, if we set *A16* to 1:
ROM (lowest): $\mathbf{0001\ 0000\ 0000\ 0000\ 0000}_2 = 10000_{16}$
ROM (highest): $\mathbf{0001\ 1111\ 1111\ 1111\ 1111}_2 = 1FFFF_{16}$

Therefore, the ROM can also be found between the addresses 10000_{16} and $1FFFF_{16}$. It is a redundant image of the ROM.

To select the address $ABCD_{16}$ of the ROM, the microprocessor can send either $0ABCD_{16}$ or $1ABCD_{16}$ to the address bus.

Here is what the memory map should look like if we represent the two redundant images of the ROM:



We can calculate the number of images for each device.

The number of images is the number of combinations that can be made with the unused address bits of the microprocessor.

Unused bits = 20 address bits – 3 device-selection bits – Address bits of a device

ROM: 20 – 3 – 16 = 1 unused bit → $\mathbf{2^1}$ **= 2 images.**
RAM: 20 – 3 – 15 = 2 unused bits → $\mathbf{2^2}$ **= 4 images.**
P1: 20 – 3 – 10 = 7 unused bits → $\mathbf{2^7}$ **= 128 images.**
P2: 20 – 3 – 4 = 13 unused bits → $\mathbf{2^{13}}$ **= 8,192 images.**

Here is what the memory map should look like: