3 Input Output (I/O) Organization and Bus Operations

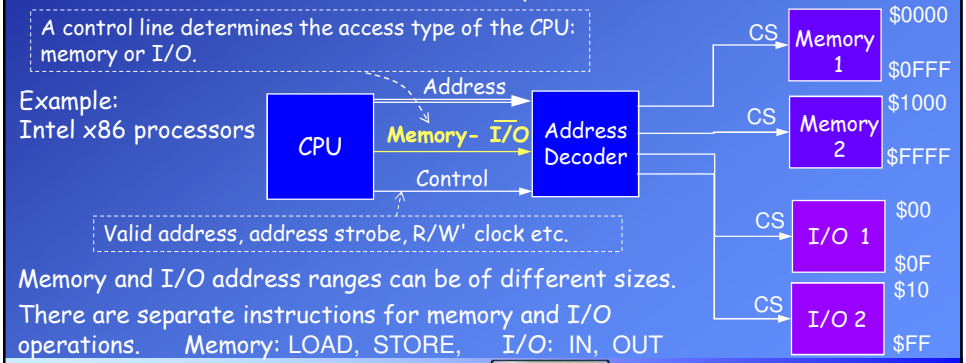
The I/O module manages the data transfer between internal storage (registers, memory) and external input/output devices (keyboard, mouse, hard disk, network interface). The computer system's I/O architecture is its interface to the outside world.

3.1 CPU – I/O Interface Connections

A) Isolated I/O Map: Two separate address spaces for memory and I/O

a) Two separate busses (address and data), one for memory and the other for I/O can be used.

b) A common bus can be used both for memory and I/O units.

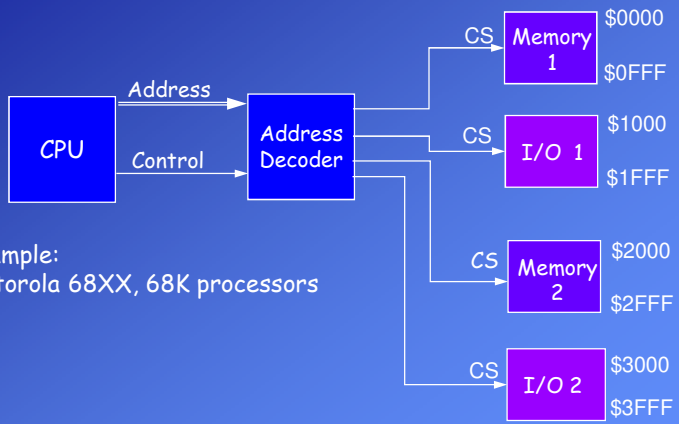


Memory and I/O address ranges can be of different sizes.

There are separate instructions for memory and I/O operations. Memory: LOAD, STORE, I/O: IN, OUT

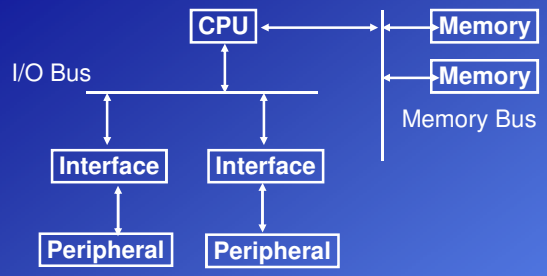
B) Memory-Mapped I/O: Same address space is shared by memory and I/O

- Common address and data buses for both memory and I/O units with common control lines
- Same instructions are used for both memory and I/O operations.
  MOVE, LOAD, STORE

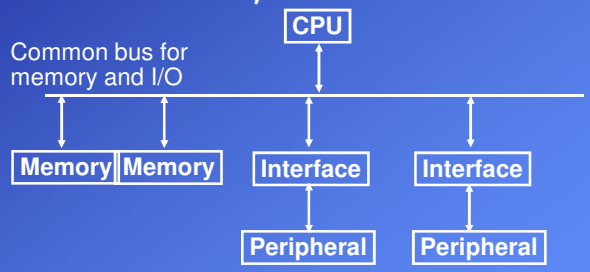


Example:
Motorola 68XX, 68K processors

Separate Buses for Memory and I/O units:



Common bus for Memory and I/O:

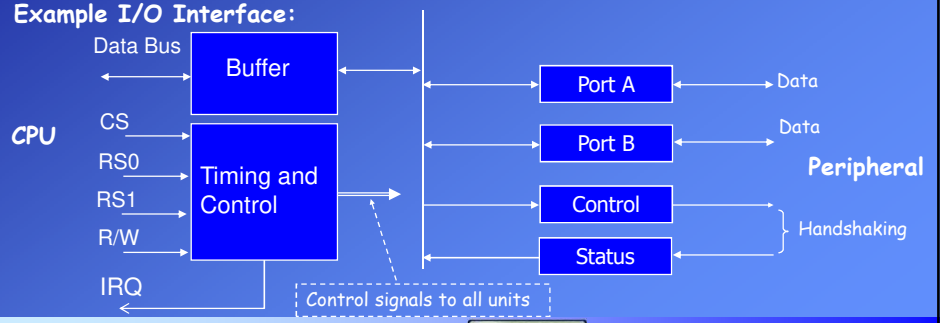


3.2 I/O Interface Module

Peripherals are connected to CPUs over an I/O interface unit. Functions:

- Checking the status of the peripheral, data buffering: Data transfer rate of a peripheral is often much slower than that of the CPU. So, it is necessary to check if the peripheral is ready for data transfer. The data coming from memory or CPU are sent to an I/O interface, buffered, and then sent to the peripheral device at its data rate.
- Data conversion: Coding, encoding, different formats
- Error detection
- Signal conversion (peripherals-CPU): Magnetic, electromechanic, electronic

Example I/O Interface:



3.3 Data Transfer Modes between I/O Interfaces and Memory

1. **Programmed I/O** (software polling): It is the responsibility of the processor
   a) to periodically check the status of the I/O interface (ready/busy, complete)
   b) to perform the data transfer between memory (registers) and I/O interface.

Read from the I/O Interface:

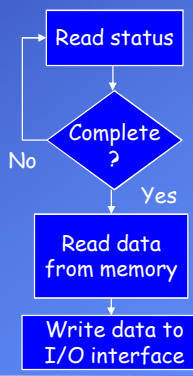
The CPU runs a program to check the status of the I/O interface.

If the I/O interface has received data from a peripheral it sets the "READY" flag.

The CPU reads data from the I/O interface and writes to the memory.

Write to the I/O Interface:

If the CPU has data to send, it checks the status of the I/O interface.



If the I/O interface has finished the previous operation it sets the "COMPLETE" flag.

The CPU reads data from the memory and writes it to the I/O interface.

1. **Programmed I/O** cont'd:

Disadvantage:

The main disadvantage of this technique is the **busy-waiting** of the CPU while checking the status of the I/O units.

The CPU performs both I/O operations:

a) Checking the status of the I/O units.

While checking the status, the CPU cannot run other programs (**busy-waiting**).

b) Data transfer is also performed by the CPU (The data goes over the CPU).

Advantage:

This technique is **simple**. Additional hardware units are not necessary.

- When the CPU does not have any tasks other than performing I/O operations or
- If the CPU cannot execute another program without performing the I/O operation,

then busy-waiting is not a problem.

For such systems, programmed I/O is a simple and suitable technique for I/O operations.

2. Interrupt-Driven I/O:

In the interrupt-driven technique, the CPU sets the I/O interface to send an interrupt request if it is ready.

Advantage:

The CPU does not need to check the status continuously. The "busy- waiting" problem does not exist.

The CPU can run other programs while the I/O interface is receiving data from or sending to a peripheral.

The I/O interface will then interrupt the processor to request service when it is ready to exchange data with the CPU.

The processor interrupts its current program, runs the interrupt service routine in which the data transfer is executed, and then resumes its former processing.

In this technique, the CPU does not check the status, but it is still the responsibility of the processor to perform the data transfer.

Disadvantage:

Interrupt processing has its own overhead (saving the return address, program status, and registers, as well as performing some other operations) (Section 4).

At the end of the service routine, return address and program status are read.

Interrupt-driven I/O is not suitable for applications where I/O operations are performed very frequently.

3.Direct Memory Access (DMA):

In the programmed and interrupt-driven techniques, the CPU is responsible for transferring data between memory and I/O interfaces.

The CPU must execute a number of instructions for each I/O transfer.

The direct memory access (DMA) technique involves an additional hardware module on the system bus, called the DMA controller (DMAC).

The DMAC is capable of acting as the CPU and of taking over control of the system bus from the processor.

When the CPU needs to read or write a block of data, it initializes the DMAC by sending the necessary information (address, size, transfer mode etc.).

Thus, it delegates responsibility for the I/O operation to the DMAC.

The CPU can continue with its other programs during the transfer of data.

The data does not go through the CPU.

The DMAC uses the system bus only when the processor does not need it, or it must force the processor to suspend the bus operations temporarily.

The DMA technique is suitable for applications where large volumes of data are transferred and I/O operations are performed very frequently.

An additional hardware module (DMAC) is necessary.

DMA is explained in Section 5.

Summary of Data Transfer Modes:

The table below shows which device is responsible for checking the status of the I/O interface and which device is responsible for transferring the data.

Task / Method	Check the status of the I/O interface	Data transfer between I/O interface and memory
Programmed I/O:	CPU (Program)	CPU (Program)
Interrupt driven I/O:	Interrupt Mechanism	CPU (ISR)
Direct Memory Access:	DMAC	DMAC

3.4 Asynchronous Data Transfer:

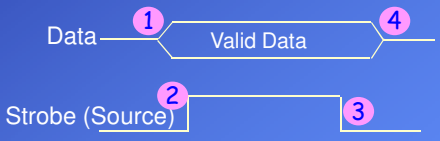
Problems:

a) Has the sender sent the data (Is the data on the data bus valid)?

b) Has the receiver received the data (Is the receiver busy)?

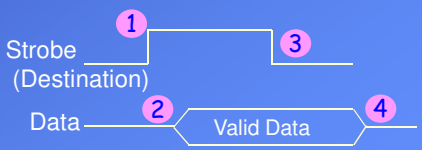
1. Strobe Control:

a) Source-initiated strobe :



The duration of valid data is predetermined according the speed of the destination.
The source does not know whether the destination has really received the data.

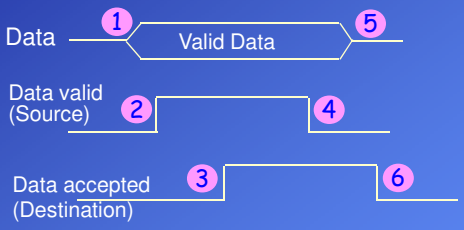
b) Destination-initiated strobe:



Time to transfer (sample) the data from the bus is predetermined according the speed of the source.
The destination does not know whether the source has really sent the data.

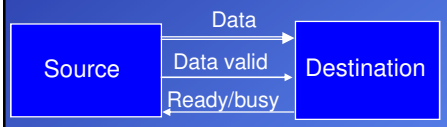
2. Handshaking:

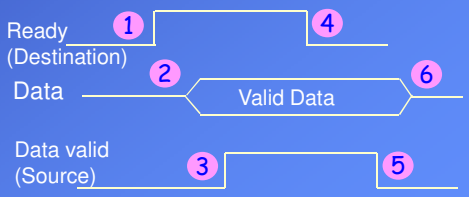
a) Source-initiated:



The source waits for the "Data accepted" signal.
To avoid "infinite waiting" when the destination does not respond with a "Data accepted" signal due to an error, a **time-out** mechanism must be used.

b) Destination-initiated:





The destination waits until the "Data valid" signal is received.
To avoid infinite waiting, a **time-out** mechanism is necessary.

3.5 Data Transfer between the CPU – Memory (or I/O Interface)

CPUs also use synchronous or asynchronous data transfer mechanisms to access the memory.

3.5.1 Synchronous bus operation with strobe

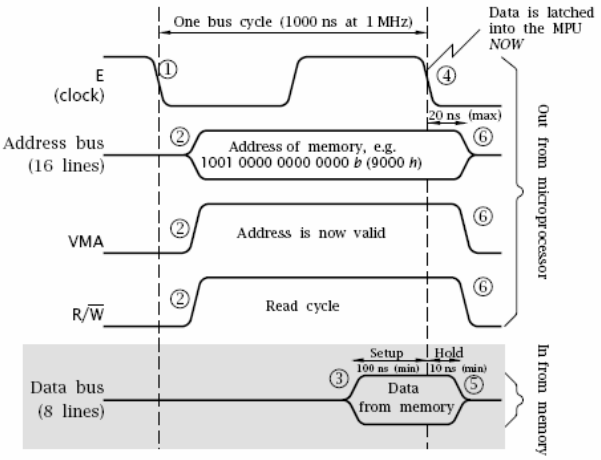
For example, MC 6802 uses a **strobe mechanism** in memory access.

This operation is also **synchronized** with the clock signal (E) of the processor.

The Valid Memory Address (VMA) signal indicates that the address on the bus is valid and initiates the bus cycle (strobe).

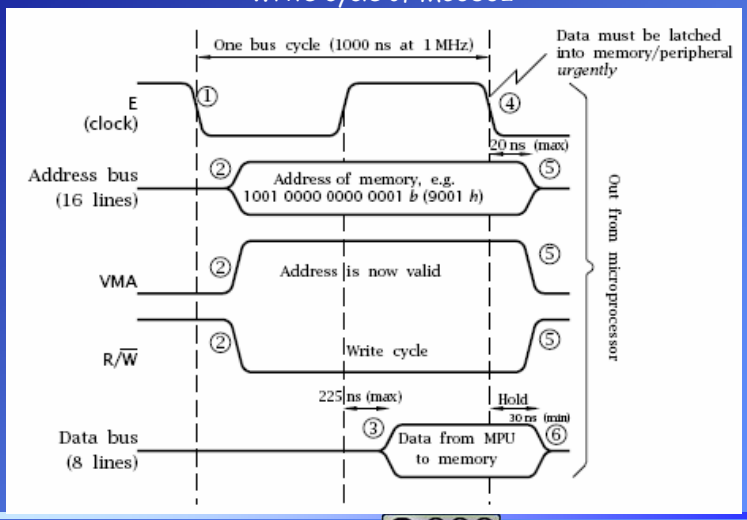
When the clock signal goes from 1 to 0, data is latched, and the bus cycle is terminated.

Read cycle of MC6802:



Similarly, the "write cycle" also starts with the activation of the VMA.
Chip select is sent to the memory via the address decoder.
When the clock signal goes from 1 to 0, the bus cycle is terminated.

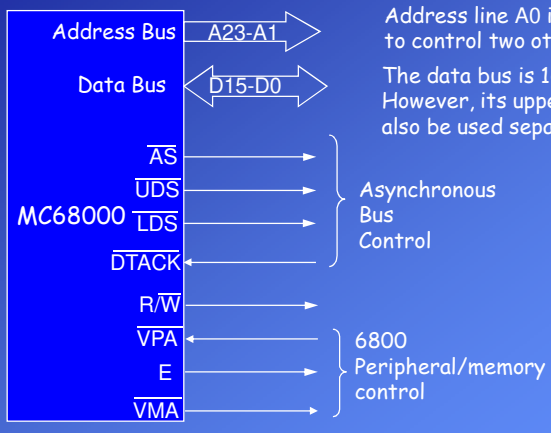
Write cycle of MC6802:



3.5.2 Asynchronous bus operations with handshaking

For example, the MC68000 access the memory (and I/O interfaces) using the **asynchronous handshaking** mechanism.

It can also use the strobe mechanism that is synchronized with the clock signal (E) like the processors of the 68xx family.



Address line A0 is used inside the processor to control two other signals: UDS' and LDS'.

The data bus is 16 bits wide.
However, its upper and lower 8-bit parts can also be used separately.

Control Signals of MC68000 used for memory access

- AS' (*Address Strobe*): It is asserted (active low) by the processor to indicate that a valid memory address exists on the address bus.

  It starts the bus cycle. First handshaking signal.

- UDS' (*Upper Data Strobe*) and LDS' (*Lower Data Strobe*): They determine the size of the data being accessed (word or byte).

  Word: Both are asserted (low).

  Byte (odd address): LDS' asserted, D0-D7 used

  Byte (even address): UDS' asserted, D8-D15 used

- DTACK' (*Data Transfer Acknowledge*): Handshaking input pin of 68000

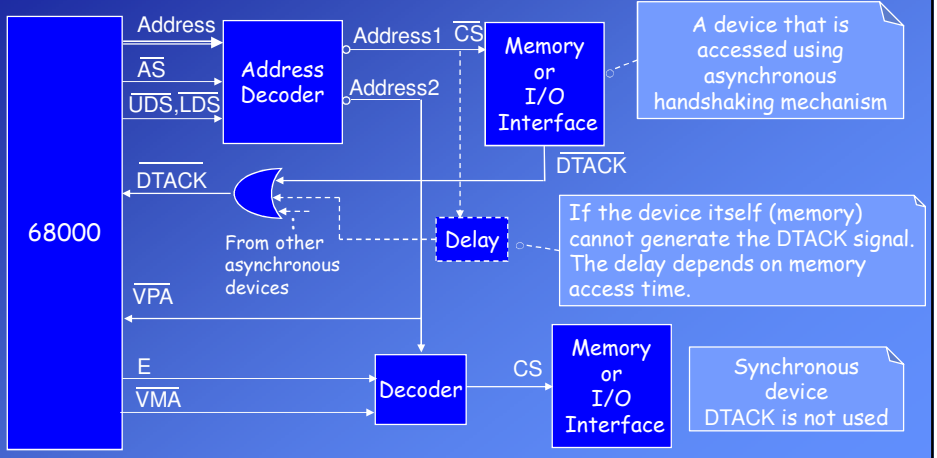
  Handshake signal generated by the device (memory/interface) being accessed indicates that the data bus contents are valid and that the 68000 may proceed with the data transfer.

- VPA' (*Valid Peripheral Address*): This input informs the 68k that it has addressed a 6800 peripheral and that the data transfer should be synchronized with the E clock.

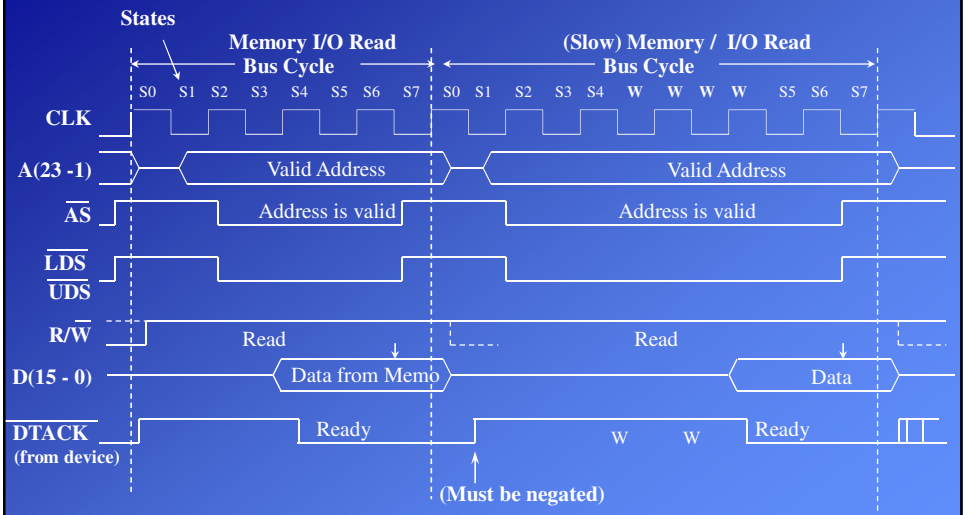
  If VPA' is asserted during a bus operation (AS' is active), the 68000 acts like a 68xx and uses VMA and E signals to access the peripheral.

MC68000 - Memory (I/O Interface) Connection

The MC68000 accesses memory (and I/O interfaces) using the asynchronous handshaking mechanism.
It can also act like a processor of the 68xx family and perform a synchronous bus operation that starts with a strobe (VMA).



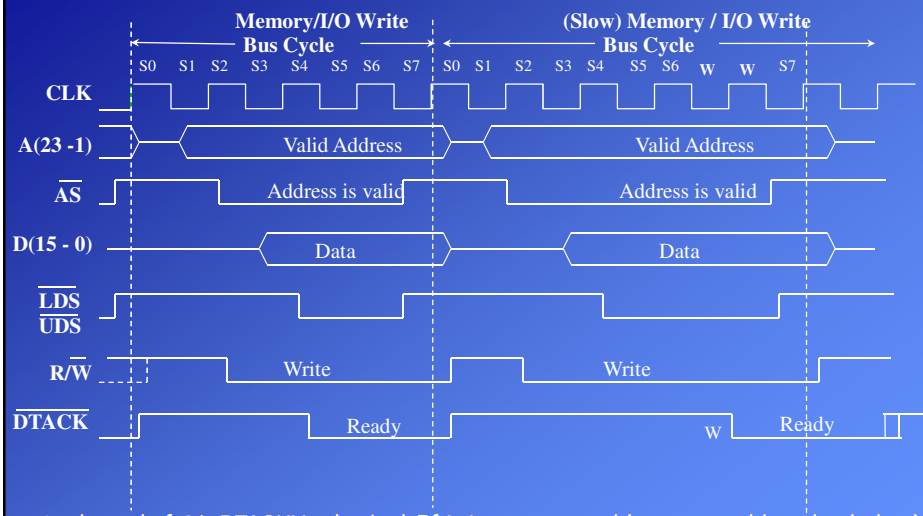
MC68000 Asynchronous Read-Cycle Timing



On the falling edge at the end of S6 (start of S7), data is latched from the bus.

If the DTACK is not asserted the processor inserts *wait states*.

MC68000 Asynchronous Write-Cycle Timing



At the end of S6, DTACK' is checked. If it is not asserted (memory could not latch data), wait states are inserted.

Avoiding Infinite Waiting

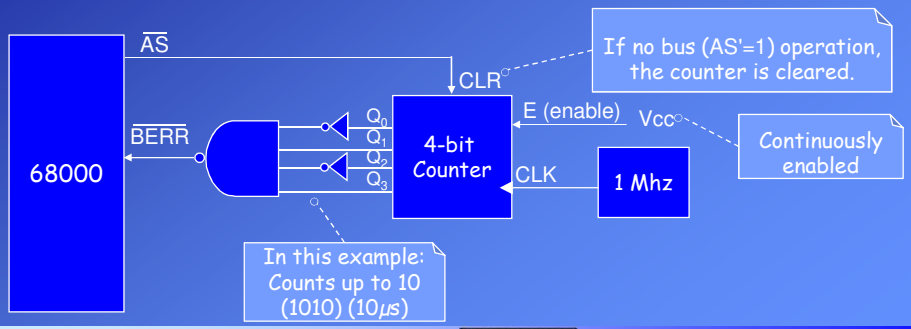
MC68000 has an exception input called **BERR'** (*Bus Error*) that can be asserted by an external logic if an error in the current bus cycle is detected.

If this input is asserted (active 0) the 68000 terminates the current bus cycle, saves the current status into the stack (accessed address, current instruction etc.), and jumps to an exception handler program.

BERR' will be explained in the chapter "Exceptions".

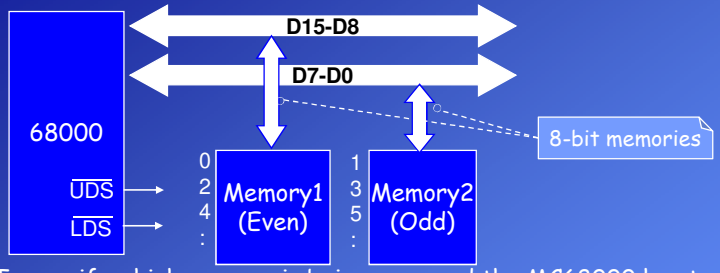
To avoid infinite waiting a counter can be connected to the BERR' as shown below:

If the bus cycle takes (AS' stays active) longer than expected, BERR' is asserted.



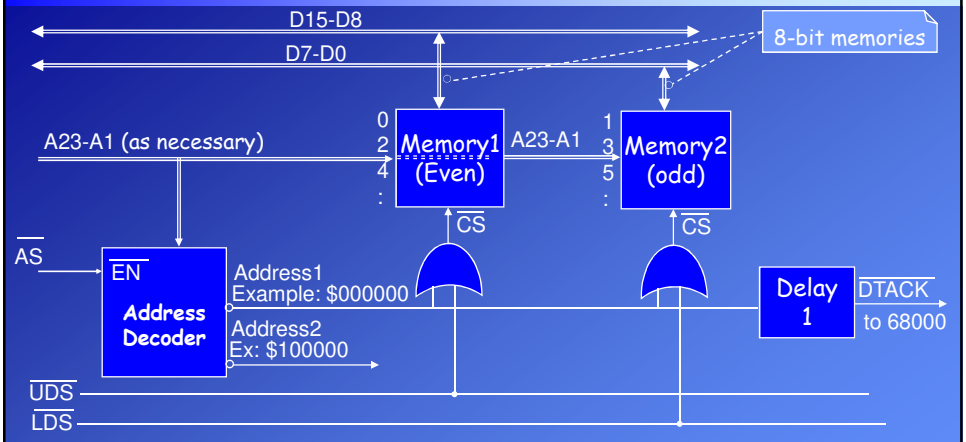
8/16-bit Bus operations in MC68000 (access to odd/even addresses)

In MC68000, the width of the data bus is 16 bits.
Instructions can operate on 8, 16 or 32-bit operands (bus operations).
Therefore, 8-bit memories are connected to the 16-bit data bus in parallel.



To specify which memory is being accessed the MC68000 has two outputs:
UDS' (*Upper Data Strobe*) and **LDS'** (*Lower Data Strobe*).
The value of internal address line A0 determines the values of UDS' and LDS'.

UDS	LDS	D15-D8	D7-D0	Description:	
H	H	---	---	No bus operation	
H	L	---	Data	Byte access to an odd address	A0=1
L	H	Data	---	Byte access to an even address	A0=0
L	L	Data	Data	Word access to an even address	A0=0



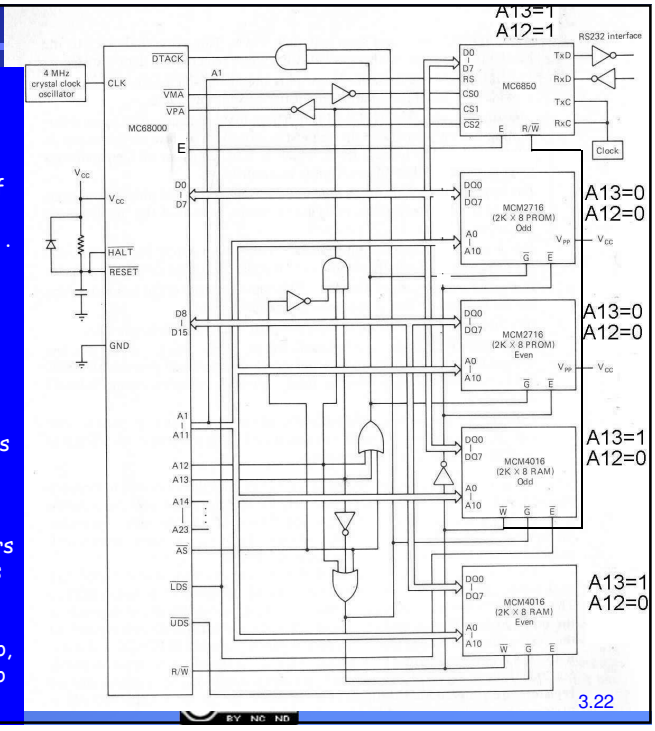
Example:

Address: (A_0 is in the processor)

	$A_{23}A_{22}$ ..... A_1 A_0	UDS	LDS
MOVE.B ($000000),D1	0 0 ..... 0 0	0	1
MOVE.B ($000001),D1	0 0 ..... 0 1	1	0
MOVE.W ($000000),D1	0 0 ..... 0 0	0	0
MOVE.W ($000001),D1	Exception: Address error		

Example:

In the example on the right the following units are connected to the MC68000 :
One serial communication interface (MC6850), a total of 4Kx8 ROM (2x 2716) and a total of 4Kx8 SRAM (2x4016) .
Only two address lines, A13 and A12, are used for address decoding.
The serial communication interface (MC6850) is connected as a synchronous unit (VMA and E are used).
Note: It communicates with its peer interface using the **asynchronous** mechanism.
Between the CPU and memories, data transfer occurs according to the asynchronous handshaking mechanism (AS and DTACK are used).
Since ROMs are not written to, UDS/LDS lines are not used to select these units.



Function Code Outputs in MC68000

MC68000 has 3 outputs that indicate the type of the operations:

Function Codes Outputs: FC2, FC1, FC0.

These outputs get valid values in each bus cycle (when AS' is asserted) and indicate the type of the operation.

User and supervisor modes are explained in section 4.5.1 *Privilege Modes*.

FC2	FC1	FC0	Description:
0	0	0	Undefined (Reserved)
0	0	1	User Mode, Data access (*User Data*)
0	1	0	User Mode, Program access (*User Program*)
0	1	1	Undefined (Reserved)
1	0	0	Undefined (Reserved)
1	0	1	Supervisor Mode, Data access (*Supervisor Data*)
1	1	0	Supervisor Mode, Program access (*Supervisor Program*)
1	1	1	Interrupt Acknowledge

These outputs can be used in address decoding.

- Access to specific devices and memory addresses can be restricted. These addresses can be accessed only in supervisor mode.
- Separate memory spaces can be created for programs and data.

Example:

Placing separate program and data memories to the same address space

In this example, by using the FC0 output of the MC68000, two separate memory modules are placed into the same address space.
The first module is selected when FC0 = 0 (program access).
The second one is selected if FC0 = 1 (data access).