4 Input Output (I/O) Organization

This organization manages the data transfer between inside world (registers, memory) and outside world (keyboard, mouse, hard disk, network interface).

4.1 CPU – I/O Interface Connections

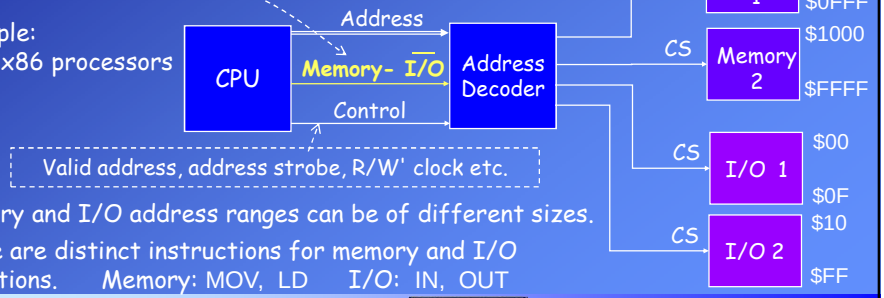
Isolated I/O Map:

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

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

A control line express the access type of the CPU; memory or I/O.

Example:
Intel x86 processors

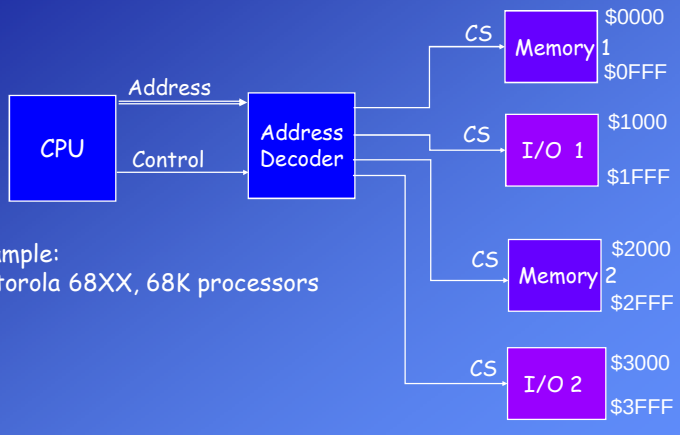


Memory and I/O address ranges can be of different sizes.

There are distinct instructions for memory and I/O operations. Memory: MOV, LD I/O: IN, OUT

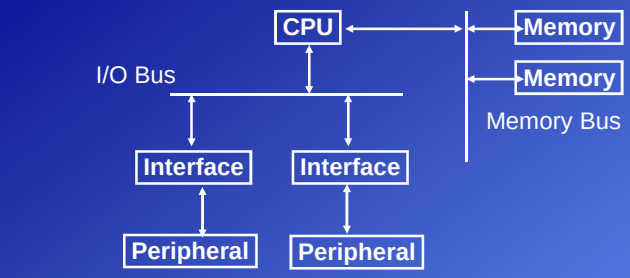
Memory Mapped I/O:

- Common address and data busses both for memory and I/O units with common control lines
- Same instructions are used both for memory and I/O operations.
  MOV, LD, ST

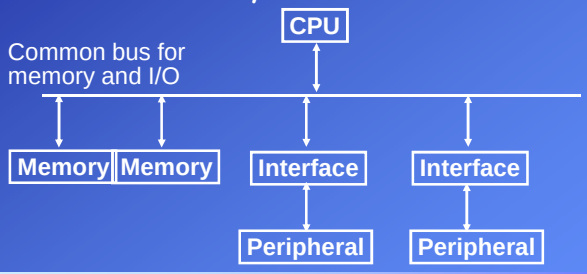


Example:
Motorola 68XX, 68K processors

Separate Busses for Memory and I/O units:



Common bus for Memory and I/O:



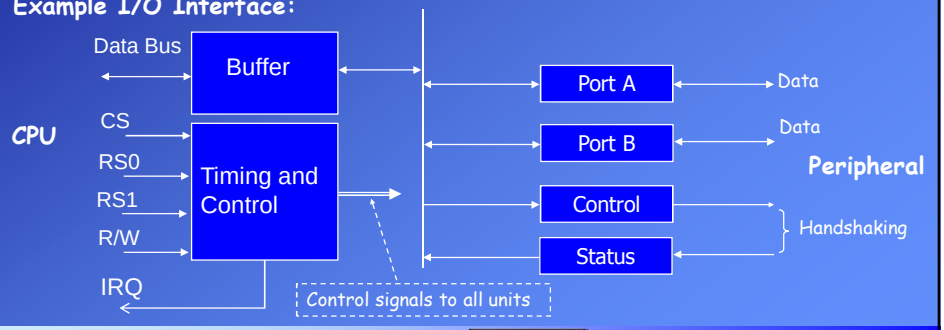
4.2 I/O Interface Module

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

Functions of an interface:

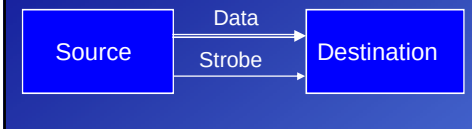
- The data transfer rates of peripherals and the CPU are different. Checking the status of the peripheral, data buffering
- Data conversion: Coding, encoding, different formats
- Error detection
- Signal conversion: Magnetic, electromechanic, electronic

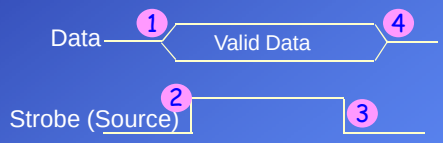
Example I/O Interface:



4.3 Asynchronous Data Transfer:

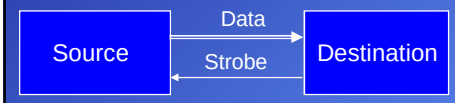
1. Strobe Control:
a) Source-initiated strobe :

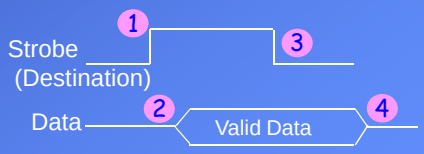




The duration of valid data is predetermined according the speed of the destination.
The source does not know whether the destination has really received the data.

b) Destination-initiated strobe:

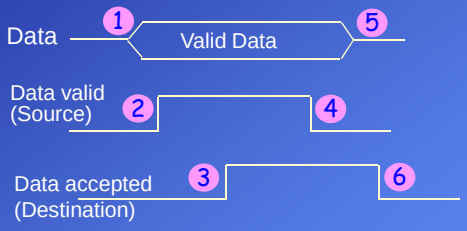




When the data is transferred (sampled) from the bus is predetermined according the speed of the source.
The destination does not know whether the source has really sent the data.

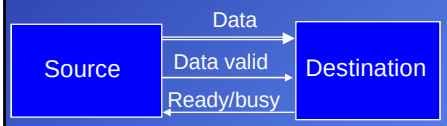
2. Handshaking:

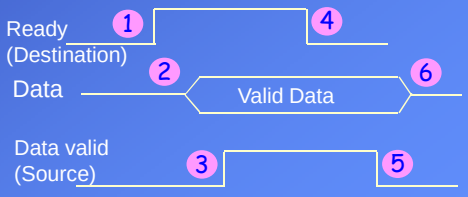
a) Source-initiated:



The source waits for the "Data accepted" signal.
To avoid "infinite waiting" when the "Data accepted" signal does not respond 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.

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

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

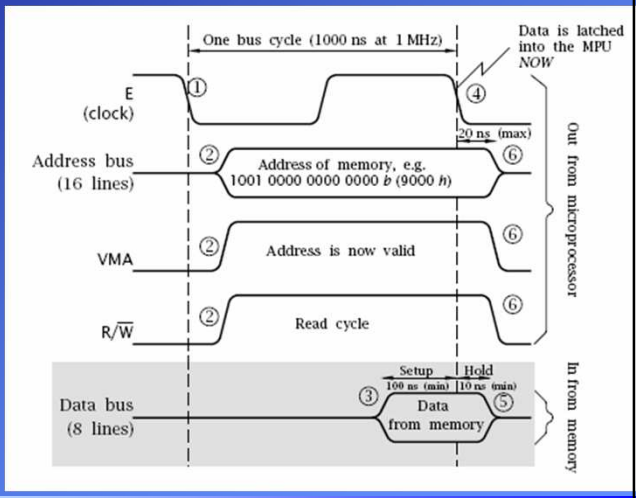
4.4.1 Reminder: **Memory access in MC6802** Read cycle of 6802:

MC 6802 uses 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.

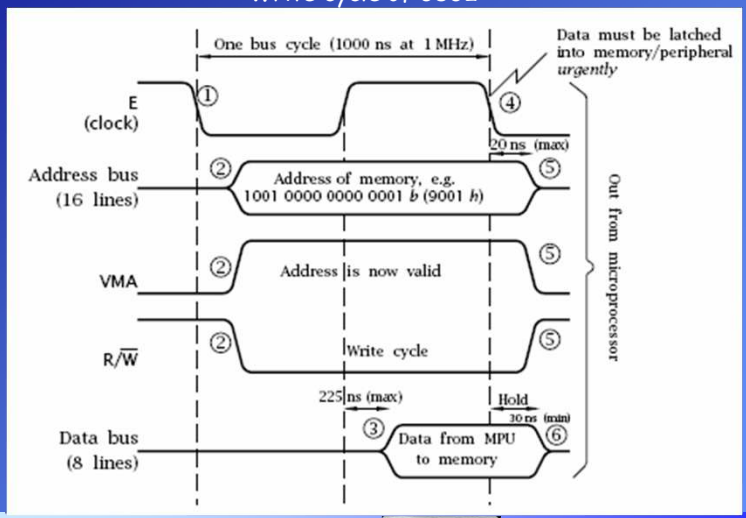


Similarly write cycle also starts with activation of the VMA.

Chip select is send to the memory via the address decoder.

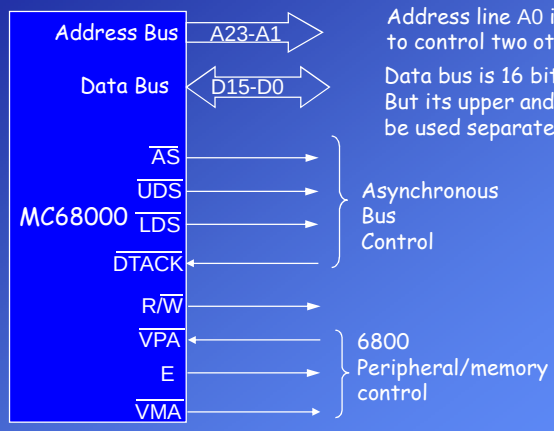
When the clock signal goes from 1 to 0 the bus cycle is terminated.

Write cycle of 6802:



4.4.2 Bus Operations in MC68000

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.

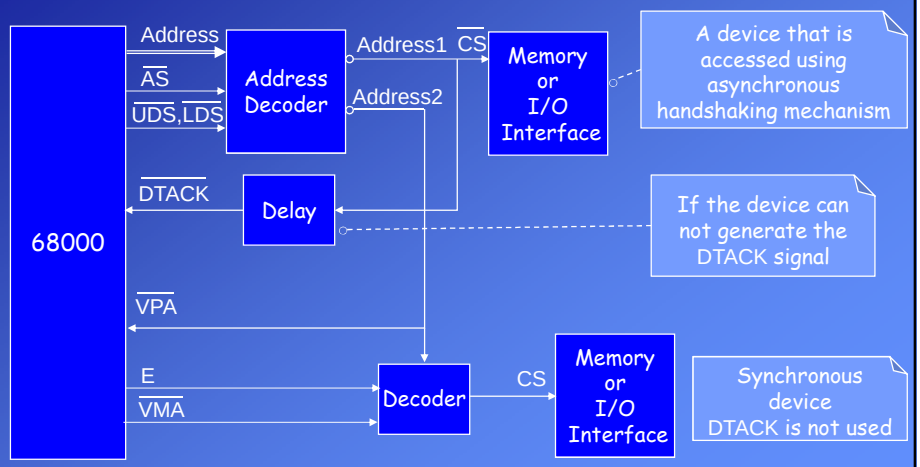


Control Signals of MC68000 used for memory access

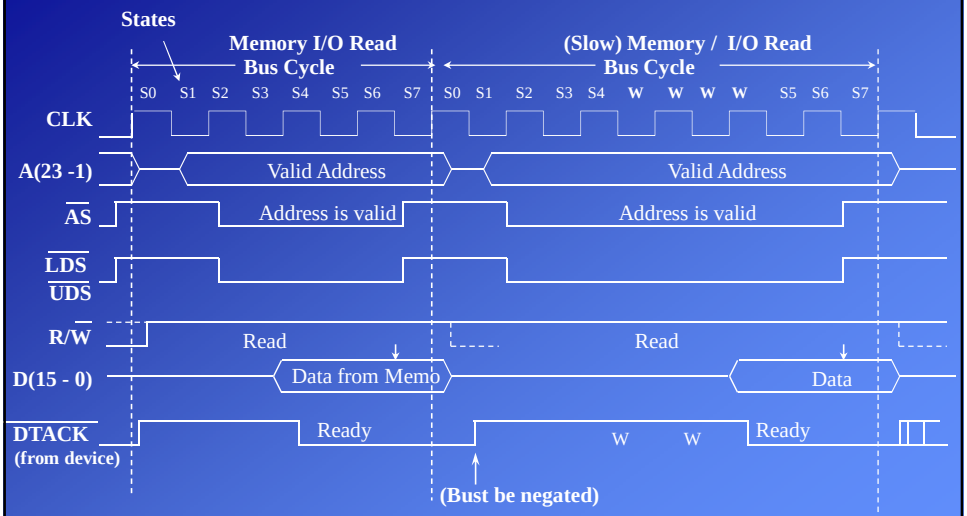
- AS' (*Address Strobe*): It is asserted (active low) by the processor to indicate that a valid memory address exists on the address bus.
  It starts the bus cycle.
- UDS' (*Upper Data Strobe*) and LDS' (*Lower Data Strobe*): They determine the size of the data being accessed (word or byte).
  Word: Both are asserted
  Byte (odd address): LDS' asserted, D0-D7 used
  Byte (even address): UDS' asserted, D8-D15 used
- DTACK' (*Data Transfer Acknowledge*): 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 it is asserted during a bus operation (AS' is active) 68000 acts like 68xx and uses VMA and E signals to access the peripheral

MC68000 - Memory (I/O Interface) Connection

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



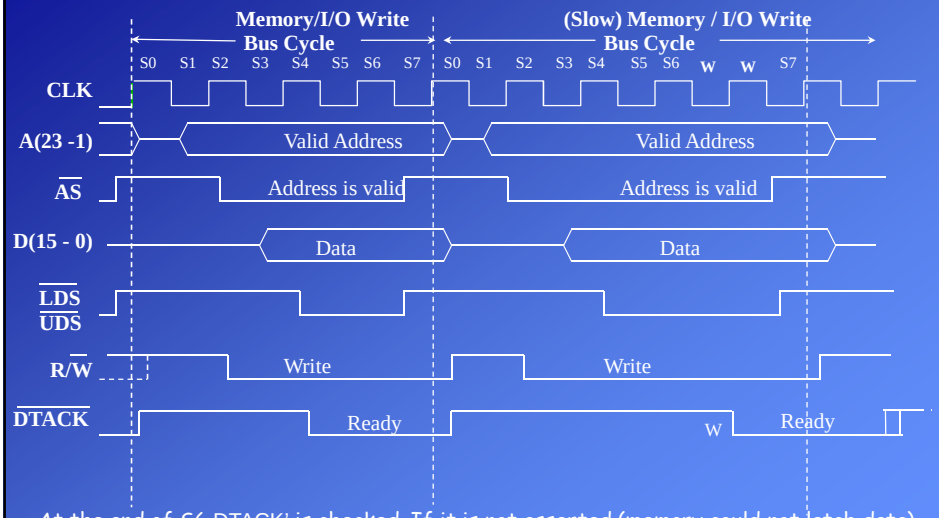
MC68000 Asynchronous Read-Cycle Timing



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

Avoiding Infinite Waiting

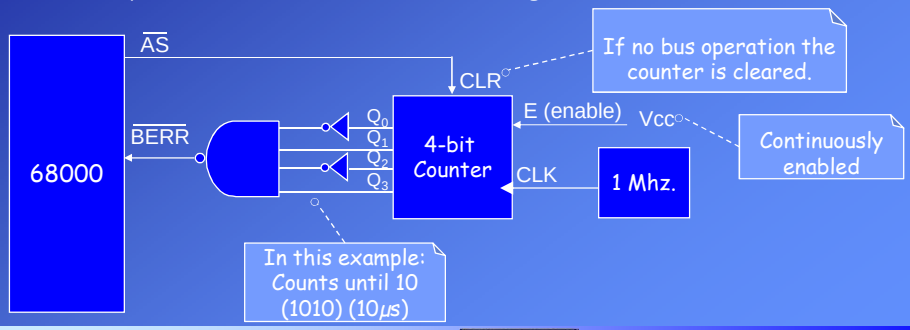
MC68000 has an exception input called **BERR'** (*Bus Error*) that can be asserted by an external logic if an error in the current bus cycle is detected.

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 a exception handler program.

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

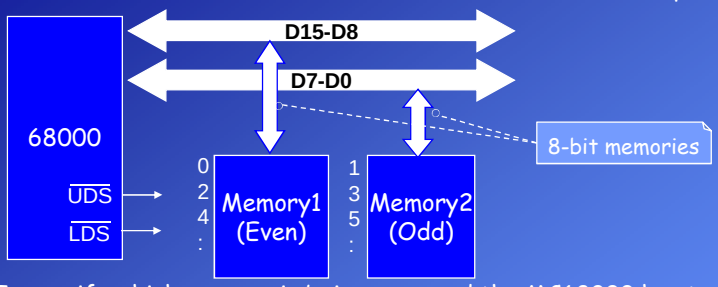
To avoid infinite waiting a counter can be connected to the BERR' as shown below:

If the bus cycle continues (AS' is active) too long BERR' is asserted.



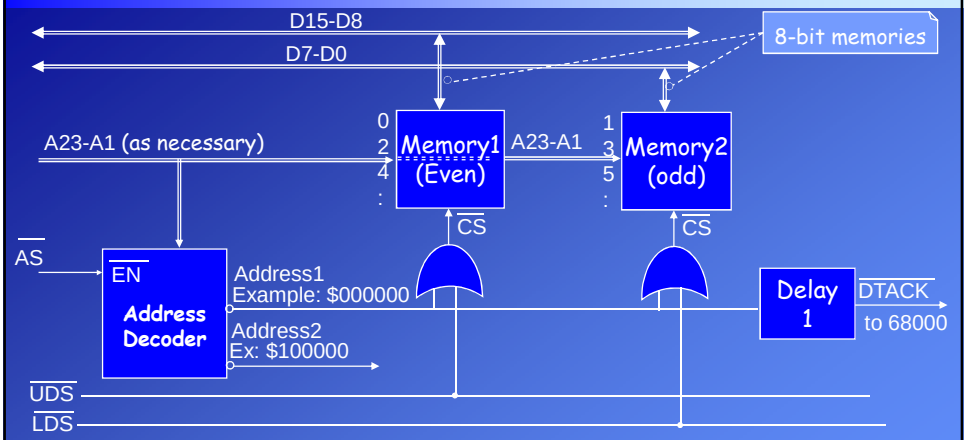
8/16-bit Bus operations in MC68000 (access to odd/even addresses)

Data bus of MC68000 is bus 16 bits.
Instructions can operate on 8, 16 or 32-bit operands (bus operations).
Therefore, 8-bit memories are connected to 16-bit data bus parallelly.



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



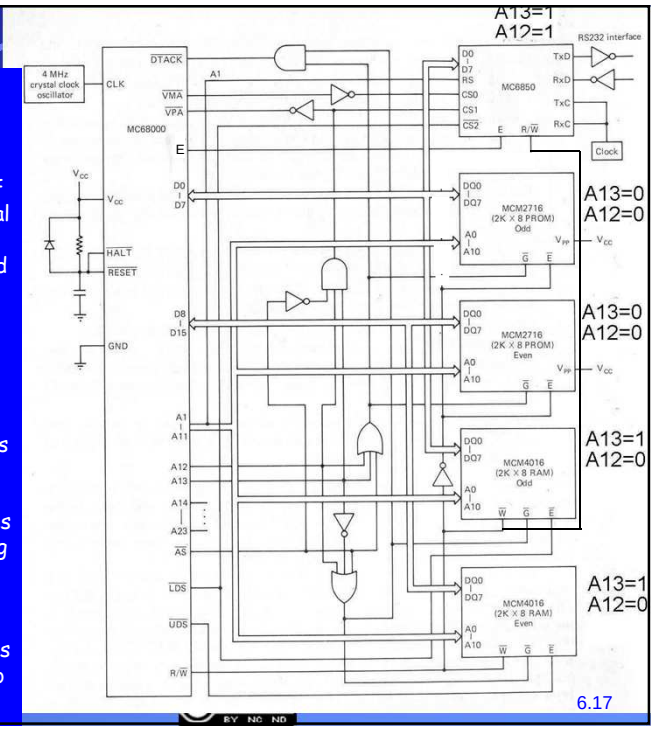
Example:

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 following units are connected to the MC68000 :
One serial communication interface (MC6850), a total of 4Kx8 ROM (2x 2716) and total of 4Kx8 SRAM (2x 4016) .
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 it is not written to ROMs 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

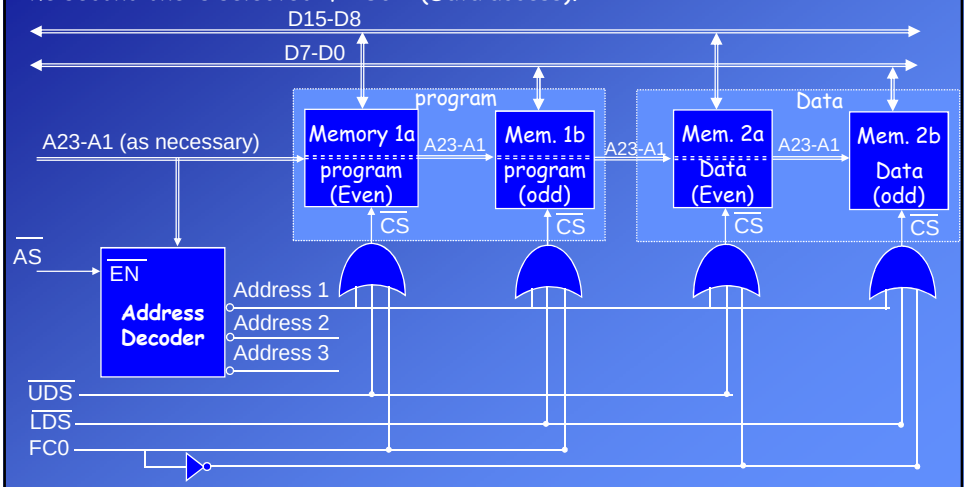
FC2	FC1	FC0	Description:
0	0	0	Undefined (Reserved)
0	0	1	User Mode, Data access (*User Data*)
0	1	0	User Mode, Program access (*User Program*)
0	1	1	Undefined (Reserved)
1	0	0	Undefined (Reserved)
1	0	1	Supervisor Mode, Data access (*Supervisor Data*)
1	1	0	Supervisor Mode, Program access (*Supervisor 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 same address space

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

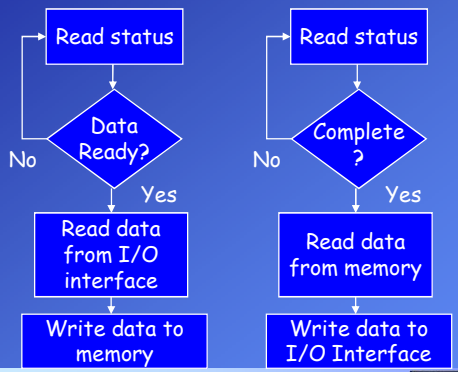


4.5 Data Transfer Modes between I/O Interfaces and Memory

1. **Programmed I/O** (software polling): It is the responsibility of the processor
   a) periodically to 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 a 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:

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

During checking the status the CPU can not run other programs.

When the CPU does not have any other task than performing I/O operations or if the CPU can not execute another program without performing the I/O operation, then busy-waiting is not a problem.

For such systems programmed I/O is simple an suitable technique for I/O operations.

2. Interrupt-Driven I/O:

The problem with the programmed I/O was busy-waiting for the I/O interface to be ready.

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

The CPU does not need to check the status continuously.

It 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 then interrupts its current program, executes the data transfer, as before, 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 between memory (registers) and I/O interface.

Remember that interrupt processing has its own overhead (saving return address, program status, registers and performing some other operations).

When large volumes of data are to be transferred and I/O operations are performed very frequently then this technique can degrade system performance.

3.Direct Memory Access (DMA):

In the programmed and interrupt-driven techniques the CPU is responsible to transfer data between memory and I/O interface.

The CPU is tied up in performing an I/O transfer;

A number of instructions must be executed for each I/O transfer and any data transfer must traverse a path through the CPU.

The direct memory access (DMA) technique involves an additional 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.).

So, it has delegated the I/O operation to the DMAC. It can continue with its other programs.

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

After the DMAC has finished its job (a block of data) it can signal the CPU by sending an interrupt request.