# Memory Interfacing

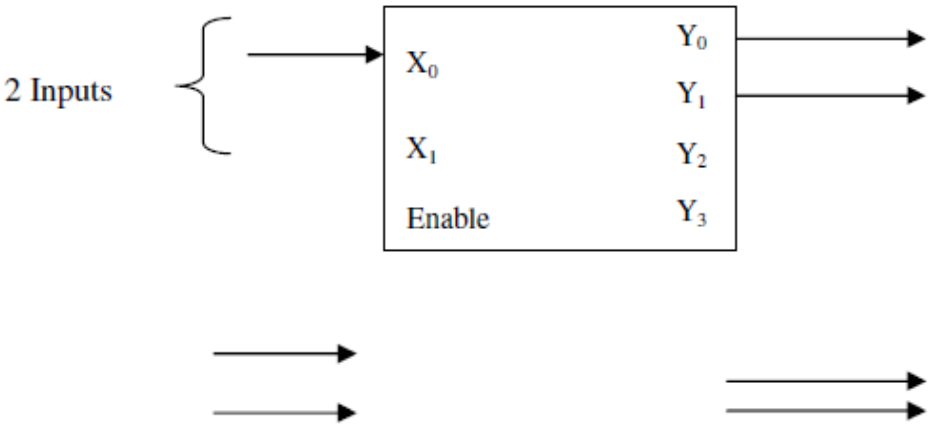
## Memory Address Decoding

Binary Decoders - Decoders have n-inputs and 2n outputs, each input combination results in a single output line having a 1, all other lines have a 0 on the output. Examples of use are decoding CPU instructions and memory addresses. Decoders typically have an *enable* when 1 enables decoding the input to 1 on a single output, when not enabled all outputs are 0. The switching function for an *enabled* two-input binary decoder
is:

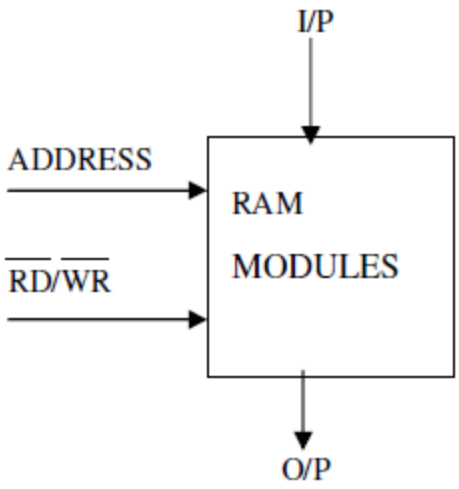
Switching function

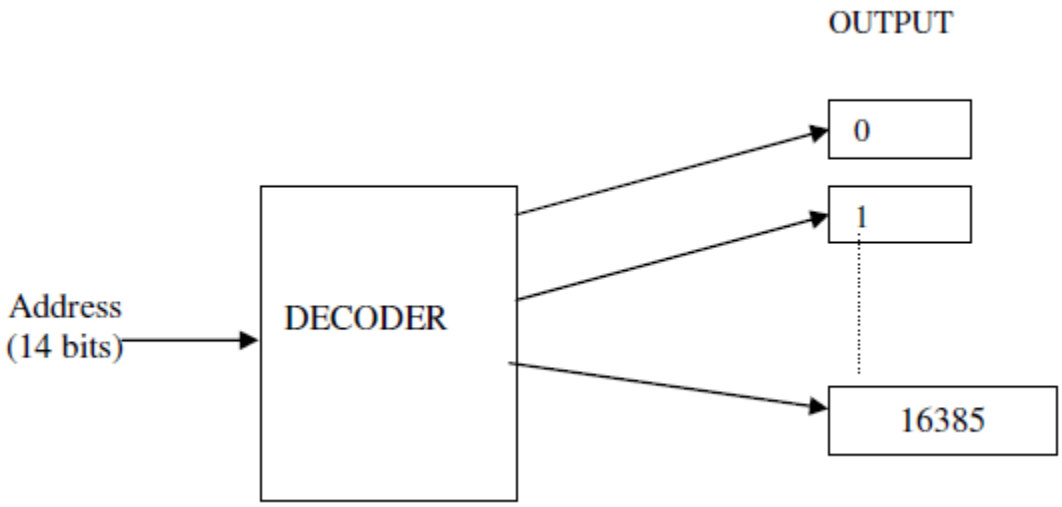
| Enabled | $x_1$ | $x_0$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | - | - | 0 | 0 | 0 | 0 |

The *2 to 4* decoder representation is:




Memory Address Decoding – Figure illustrates a 16K by 1 bit word memory (8 bit words are implemented by selecting 8 bits as a group, for example). Since 214 is approximately 16K, a single decoder would require 14 inputs and 214 output




The memory decoder is connected to the CPU by the *address bus.* Each memory cell is connected to an input and output *data bus,* a *read/write control,* and the decoder which enables the memory cell when the appropriate address appears. The decoder ensures that only a single memory cell is activated at a time for either input or output.

# Cache Memory

Caching is a technology based on the memory subsystem of your computer. The main purpose of a cache is to accelerate the computer while keeping the price of the computer low. Caching allows to do computer tasks more rapidly.

Cache technology is the use of a faster but smaller memory type to accelerate a slower but larger memory type.

*Cache* is small high speed memory usually Static RAM that contains the most recently accessed pieces of the time it takes to an instruction (or piece of data) into the processor is very long when compared to the time to execute the instruction.

When using a cache, we must check the cache to see if the item is in the cache. If it is, that is called a cache hit. If not, it is called a cache miss and the computer must wait for a round trip from the larger, slower memory area.

A cache has some maximum size that is much smaller that the larger storage area.

**Cache** memory helps by decreasing the time it takes to move information to and from the processor. **Cache** memory allows small portions of main memory to be accessed 3 to 4 times faster than DRAM It applies

"Locality of Reference." At any time the processor will be accessing memory in a small or localized region of memory. The **cache** loads this region allowing the processor to access the memory region faster. some of the common terms used when talking

## Cache Hits

When the **cache** contains the information requested, the transaction is said to be a **cache** hit.

## Cache Miss

When the **cache** does not contain the information requested, the transaction is said to be a **cache** miss.

## Cache Consistency

Since **cache** is a photo or copy of a small piece main memory, it is important that the **cache** always reflects what is in main memory.

Some common terms used to describe the process of maintaining **cache** consistency are:

## Snoop

When a **cache** is watching the address lines for transaction, this is called a snoop. This function allows the

**cache** to see if any transactions are accessing memory it contains within itself.

## Snarf

When a **cache** takes the information from the data lines, the **cache** is said to have snarfed the data. This

function allows the **cache** to be updated and maintain consistency.

Snoop and snarf are the mechanisms the **cache** uses to maintain consistency.

Two other terms are commonly used to describe the inconsistencies in the **cache** are:

## Dirty Data

When data is modified within **cache** but not modified in main memory, the data in the **cache** is called "dirtydata."

## Stale Data

When data is modified within main memory but not modified in **cache**, the data in the **cache** is called stale data.

## Cache Architecture

Caches have two characteristics , a read architecture and a write policy.

The read architecture may be either "Look Aside" or "Look Through."

The write policy may be either "Write-Back" or "Write-Through."

Both types of read architectures may have either type of write policy, depending on the design.

## Read Architecture

## Look Aside Cache

In "look aside" **cache** architecture the main memory is located opposite the system interface. Both the

main memory and the **cache** see a bus cycle at the same time. Hence the name "look aside."

Look Aside **Cache** Example

When the processor starts a read cycle, the **cache** checks to see if that address is a **cache** hit.

HIT: If the **cache** contains the memory location, then the **cache** will respond to the read cycle and terminate the bus cycle.

MISS: If the **cache** does not contain the memory location, then main memory will respond to the processor and terminate the bus cycle. The **cache** will snarf the data, so next time the processor requests this data it

will be a **cache** hit. Look aside caches are less complex, which makes them less expensive. This architecture also provides better response to a **cache** miss since both the DRAM and the **cache** see the bus

cycle at the same time. The draw back is the processor cannot access **cache** while another bus master is accessing main memory.

## Read Architecture: Look through

Main memory is located opposite the system interface. The discerning feature of this **cache** unit is that it sits between the processor and main memory. It is important to notice that **cache** sees the processors bus cycle before allowing it to pass on to the system bus. Look Through Read Cycle Example When the processor starts a memory access, the **cache** checks to see if that address is a **cache** hit.

HIT: The **cache** responds to the processor's request without starting an access to main memory.

MISS: The **cache** passes the bus cycle onto the system bus. Main memory then responds to the processors request. **Cache** snarfs the data so that next time the processor requests this data, it will be a **cache** hit. This architecture allows the processor to run out of **cache** while another bus master is accessing main memory, since the processor is isolated from the rest of the system. However, this **cache** architecture is more complex because it must be able to control accesses to the rest of the system. The increase in complexity increases the cost. Another down side is that memory accesses on **cache** misses are slower because main memory is not accessed until after the **cache** is checked. This is not an issue if the **cache** has a high hit rate and their are other bus masters.

### write policy

A write policy determines how the **cache** deals with a write cycle. The two common write policies are Write-Back and Write-Through. In Write-Back policy, the **cache** acts like a buffer. That is, when the processor starts a write cycle the **cache** receives the data and terminates the cycle. The **cache** then writes the data back to main memory when the system bus is available. This method provides the greatest performance by allowing the processor to continue its tasks while main memory is updated at a later time.

However, controlling writes to main memory increase the cache's complexity and cost. The second method is the Write-Through policy. As the name implies, the processor writes through the **cache** to main memory.

The **cache** may update its contents, however the write cycle does not end until the data is stored into main memory. This method is less complex and therefore less expensive to implement. The

performance with a Write-Through policy is lower since the processor must wait for main memory to accept the data.

### Cache Components

The **cache** sub-system can be divided into three functional blocks: SRAM, Tag RAM, and the **Cache Controller**. In actual designs, these blocks may be implemented by multiple chips or all may be combined into a single chip.

### SRAM

Static Random Access Memory (SRAM) is the memory block which holds the data. The size of the SRAM determines the size of the **cache**.

## Tag RAM

Tag RAM (TRAM) is a small piece of SRAM that stores the addresses of the data that is stored in the SRAM.

## Cache Controller

The **cache controller** is the brain behind the **cache**. Its responsibilities include: performing the snoops and snarfs, updating the SRAM and TRAM and implementing the write policy. The **cache controller** is also responsible for determining if memory request is cacheable and if a request is a **cache** hit or miss. The cache controller detects cache misses and controls sending and receiving the cells. This device also controls interface. The cache controller contains a status register, which can be read by the processor. The bits of the status register may be used to signal to the processor.

The cache controller accepts commands from the processor. Examples of the commands are

_ Reset the Tag Rams

_ Set interrupt mask

The cache controller will send a cell to request a cache line when a miss occurs. This cell may also flush an existing dirty line. It expects a cell to be returned containing the data, and the CPU is stalled until such a cell is received.

# SUMMARY

__ 8086 has a powerful set of registers containing general purpose and special purpose registers. All the registers of 8086 are 16-bit registers. The general purpose registers, can be used as either 8-bit registers or 16-bit registers.

__ The 8086's PSW contains 16 bits, but 7 of them are not used. Each bit in the PSW is called a flag. The 8086 flags are divided into the conditional flags, which reflect the result of the previous operation involving the ALU, and the control flags, which control the execution of special functions.

__ Addressing mode indicates a way of locating data or operands. Depending upon the data types used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes, or some instruction may not belong to any of the addressing modes

__ Caching is a technology based on the memory subsystem of your computer. The main purpose of a cache is to accelerate the computer while keeping the price of the computer low. Caching allows to do computer tasks more rapidly.

__ The **cache controller** is the brain behind the **cache**. Its responsibilities include: performing the snoops and snarfs, updating the SRAM and TRAM and implementing the write policy.