Microprocessor 8086/8088

Advantages of segmented memory:

Segmented memory can be confusing at first. Simply, the program op-codes will be fetched from the code segment, while program data variables will be stored in the data and extra segments. Stack operations use register BP or SP and the stack segment.

An immediate advantage of having separate data and code segments is that one program work on several different sets of data. This is done by reloading register DS to point to the new data. Perhaps the greatest advantage of segmented memory is that programs that reference logical addresses only can be loaded and run anywhere in memory.

Microprocessor 8086/8088

This is because the logical addresses always range from 0000 to FFFFH, independent of the code segment base. Consider a multitasking environment in which the 8086/8088 is doing several different jobs at once. An inactive program can be temporarily saved on a magnetic disk and a new program, such programs are said to be relocatable, meaning that they will run at any location in memory. The requirements of writing relocatable programs are that no references be made to physical addresses, and no changes to the segment registers are allowed.

Microprocessor 8086/8088

Fetch and Execute:

Organization of the 8088/8086 CPU into a separate BIU and EU allows the fetch and execute cycles to overlap. To see this, consider what happens when the 8088/8086 is first started:

1.the BIU outputs the contents of the instruction pointer (IP) onto the address bus, cause the selected byte or word to be read into the BIU.

2.Register IP is incremented by one to prepare for the next instruction fetch.

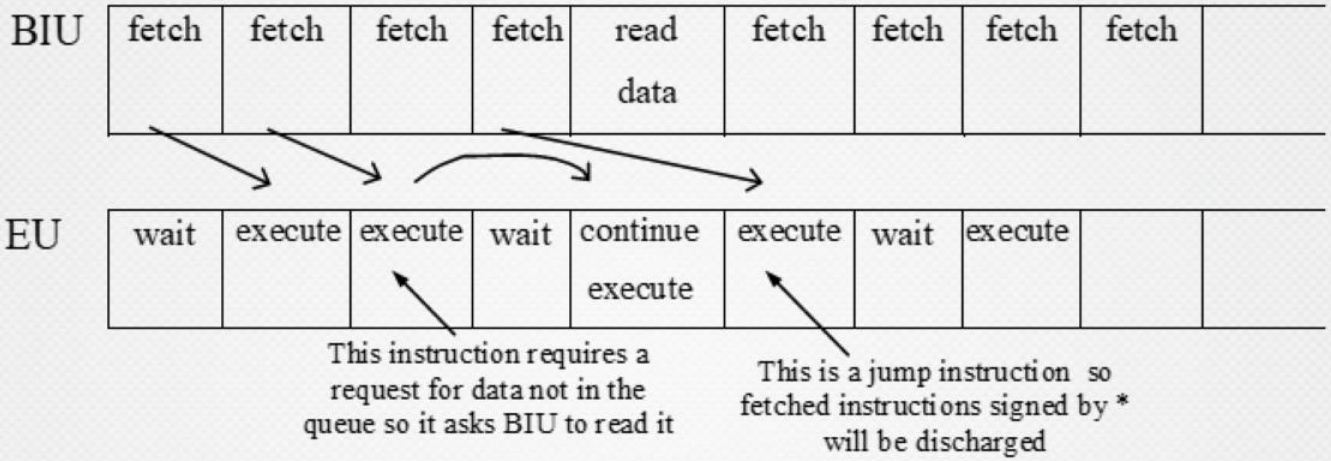
3.Once inside the BIU, the instruction is passed to the "instruction queue". This is a First_In_First_Out (FIFO) storage (cache memory) linked to a pipeline.

Microprocessor 8086/8088

1. Assuming that the queue is initially empty, the EU immediately draws this instruction from the queue and begins execution.
2. While the EU is executing this instruction, the BIU proceeds to fetch a new instruction, the BIU may fill the queue with several new instructions whenever the queue has room for one (with the 8088) or two (with 8086) additional bytes.

   The advantage of this pipelined architecture is that the EU can execute instructions almost continually instead of having wait for the BIU to fetch a new instruction as in the following figure:

Microprocessor 8086/8088



Microprocessor 8086/8088

There are three conditions that will cause the BIU and EU to enter a "wait" mode. The first occurs when an instruction requires access to a memory location not in the queue. The BIU must suspend fetching instructions and output the address of this memory location. After waiting for the memory access, the EU can resume executing instruction codes from the queue (and the BIU can resume filling the queue).

The second condition occurs when the instruction to be executed is a "Jump" instruction. In this case control is transferred to a new (non-sequential address). The queue, however, assumes that instruction will always be executed in sequence and this will be holding the "wrong" instruction codes.

Microprocessor 8086/8088

The EU must wait while the instruction at the jump address is fetched. Note that any bytes presently in the queue must be discarded (they are over written).

One another condition can cause the BIU to suspend fetching instructions. This occurs during execution of instructions that are slow to execute. For example, the instruction (AAM ASCII Adjust for Multiplication) requires 83 clock cycles to complete. At four clock cycles per instruction fetch, the queue will be completely filled during the execution of this single instruction. The BIU will thus have to wait for the EU to pull one or two bytes from the queue before resume the fetch cycle.

Microprocessor 8086/8088

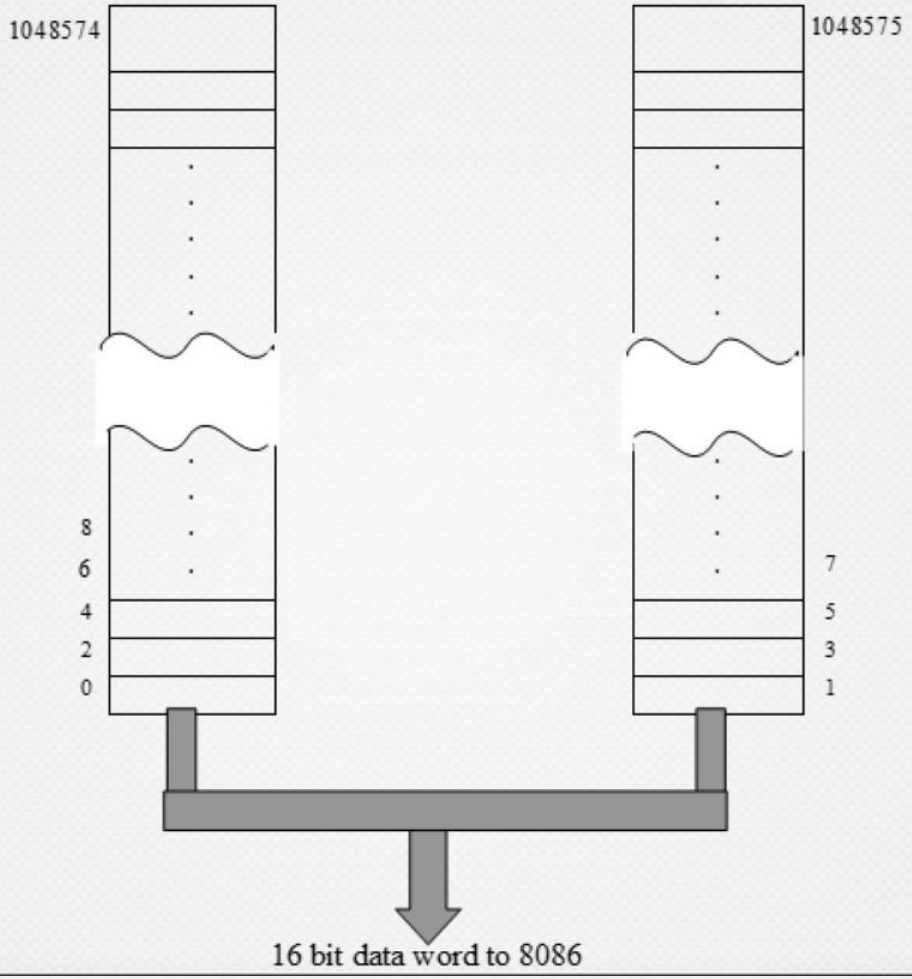
Reading / writing data to 8086:

Even though is considered a 16 bit MP (it has a 16 bit data bus width) its memory is still through of in bytes. At first this might seem a disadvantage. Actually there are a couple of good reasons first it allows the processor to work on bytes as will as words. This is specially important with I/O devices such as printers, terminal,…, etc (ASCII). Second many of the 8086s (and 8088s) operands are single bytes, other instructions may require any where from two to seven bytes. By being able to access individual bytes, these odd-lengthed instructions can be handled.

Microprocessor 8086/8088

The 8086 read 16 bits from memory by simultaneously addressed byte. For the reason the 8086 organized its memory into even-addressed bank and odd-addressed bank. 8086 provides control bus signals that can be decoded by the memory to determine if a byte or a word is to be accessed. If the CPU must access an odd word (e.g. the word stored in bytes 5 and 6), the CPU must perform two memory read cycles: one to fetch the low order byte and a second to fetch the high order byte. This slow down the processor but is transparent to the programmer.

Microprocessor 8086/8088



Microprocessor 8086/8088

That was for 8086. The 8088 with its 8-bit data bus interfaces to the 1MByte of memory as a single bank. Always when it is necessary to access a word (odd or even) two memory (read or write) cycles are performed, fortunately, for the programmer except for the slightly slower performance of the 8088, there is no difference between the two processors.