Intel’s 8086 and 8088 16-bit processors were the forefathers of the IA-32 architecture. Developed in 1978, the 8086 sported a 16-bit external data bus and a 1 MB addressing capability (20 address lines). Both the 8086 and 8088 introduced a 16-bit segment register which pointed to a memory segment of 64 KB. In 1982 Intel introduced the 286 processor, and with it, the protected mode operation to support virtual memory management.

The year 1985 brought the 386 processor, which was Intel’s first 32-bit processor. The 386 provided support for:
- A 32-bit address space (4 GB physical memory)
- A segmented and a flat memory model
- Paging mode as further support for virtual memory management

In 1989 Intel released the 486 which was the first time Intel introduced level 1 cache along with power saving and other system management options into a processor.

With the advent of the Pentium processor in 1993 Intel added a second execution pipeline, doubled the cache, used a MESI protocol to support a more efficient write-back cache along with greater branch prediction support and on-chip branch table. Intel also introduced the APIC, and support for multiple processors, in particular support for a glueless two processor system. A subsequent stepping also introduced MMX.
The Pentium Pro (P6)

- The Pentium Pro processor, also referred to as P6 introduced a three-way superscalar pipelined architecture.
- Three-way superscalar refers to the fact that the P6 is capable of (on average) decoding, dispatching, and completing three instructions per clock cycle.
- In order to perform this feat, the P6 uses a decoupled, 12-stage superpipeline which supports out-of-order instruction execution.

P6 Microarchitecture

- At the heart of the P6 microarchitecture are three data-processing concepts:
  - Deep Branch Prediction – allowing the processor to decode instructions beyond branches.
  - Dynamic Dataflow Analysis – monitoring dataflow to take advantage of out-of-order execution opportunities.
  - Speculative Execution – enabling the processor to execute instructions which are beyond a conditional branch which has not yet been resolved.

Hyper-Threading Technology

- Hyper-Threading (HT) is another innovation which improves the performance of multi-threaded, or multi-tasking operations within the IA-32 architecture.
- HT enables a single physical processor to execute two or more distinct code streams (threads) concurrently.
- The processor is divided into two or more logical processors, each with its own copies of data, control, and segment registers, as well as debug and interrupt control.

Extended Memory 64 Technology

- Yet another innovation is the EM64T, which increases the linear address space of the processor to 64 bits, and supports a physical address space of up to 40 bits.
- In order for the IA-32 to take advantage of this feature, it must operate in the IA-32e mode.
- This is really beyond the realm of Embedded Systems and so we'll skip it.

Basic Modes of Operation

- The IA-32 is basically capable of operating in one of three modes:
  - Protected mode – native state of the processor.
  - Real-address mode – the programming model of the 8086, used for backward compatibility.
  - System Management Mode – used for power management, system security, etc.
- The operating mode defines which architectural features are available.
The IA-32 Register Model

- General Purpose Registers – a set of 8 registers for storing operands and pointers. These are: EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP
- Segment Registers – provide 6 segment registers
- EFLAGS Register – Status and Control register
- EIP Register – The 32-bit Instruction Pointer, pointing to the next instruction to be executed

The Basic Memory Model

- The memory model of the IA-32 is separated into three different models:
  - **Flat Memory Model** – memory appears to a program as a single, contiguous address space from 2^32 bytes. Code, data, and stack are all contained in this address space, also called the linear address space
  - **Segmented Memory Model** – memory appears to a program as a group of independent memory segments, where code, data, and stack are contained in separate memory segments.

The Basic Memory Model (cont.)

- To address memory in this model, the processor must use segment registers and an offset to derive a linear address. Programs running in this mode can access 16,383 different segments, each addressable to up to 2^32 bytes.
- The primary reason for having segmented memory is to increase system reliability, for example, preventing stack corruption
- **Real-address Memory Model** – is the original i8086 memory model and is present to provide backward compatibility support

The Memory Management Model

- When paging is enabled on the IA-32, linear address space is sectioned into pages which are then mapped into the virtual address space, and consequently, mapped into the physical address space as needed

Paging and Virtual Memory

- With the first two memory models (flat and segmented), linear address can be mapped into the processor’s physical address either directly, or through a paging mechanism
- When paging is enabled on the IA-32, linear address space is sectioned into pages which are then mapped into the virtual address space, and consequently, mapped into the physical address space as needed
The Register Model

- EAX – Accumulator
- EBX – Pointer to data in the DS segment
- ECX – Counter for string/loop operations
- EDX – I/O Pointer
- ESI – Source pointer for string operations
- EDI – Destination pointer
- ESP – Stack Pointer
- EBP – Pointer to data on stack

Segment Registers

- The segment registers hold the segment selectors which are special pointers that identify individual segments in memory
- Segment registers are used depending on the memory management model in use
- In a flat memory model, segment registers point to overlapping segments, each of which begins at address 0
- When using the segmented memory model, each segment is loaded with a different memory address

Segmentation over Flat Memory

Segmentation over Segmented Memory

Figure 3.4 – Use of Segment Registers in Flat Memory Model

Figure 3.5 – Use of Segment Registers in Segmented Memory Model
Real versus Protected Mode

- Real Mode:
  - From an applications point of view, protected mode and real mode are not that different
  - In Real Mode, memory segmentation is handled internally by use of segment registers
  - The contents of these segments form part of the physical address
- Protected Mode:
  - In Protected Mode, memory segmentation is defined by a set of tables called the Descriptor Tables, and the segment register is simply a pointer to these tables
  - Therefore in protected mode, segment registers don’t form part of the address

Protected Mode

- Protected mode offers many features that enhance multi-tasking and promote system stability
- These features offer memory protection, paging, and hardware support for virtual memory management
- Most x86 Operating Systems, including Linux and Windows run in protected mode

Real Mode

- Real mode disables protection features available on Protected mode to allow backward compatibility with old software running in DOS mode
- All x86 CPUs start up in real mode until they are switched into protected mode by an Operating System at its boot time

Memory Management Overview

- The memory management of IA-32 is divided into two parts: segmentation and paging
- Segmentation is a mechanism for isolating individual code, data, and stack segments so that multiple tasks can run on the same processor without interfering with each other
- Paging implements a mechanism where individual pieces of a program are mapped into the physical memory as may be necessary
- Segmentation is always used, paging is optional

Segmentation and Paging
Global/Local Descriptor Tables

- The Global Descriptor Table contains segmentation information which any application can access.
- The Local Descriptor Table contains segmentation information specific to a single task or program.
- Both tables contain entries called segment descriptors which provide the base address of segments, along with access rights, type, and usage information.