The Intel IA-32 Architecture

Babak Kia
Adjunct Professor
Boston University
College of Engineering

Historical Perspective

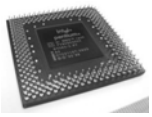
- Intel's 8086 and 8088 16-bit processors were the forefathers of the IA-32 architecture
- Developed in 1978, the 8086 sported a 16-bit external data bus and a 1 MB addressing capability (20 address lines)
- Both the 8086 and 8088 introduced a 16 bit *segment register* which pointed to a memory segment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

Historical Perspective

- The year 1985 brought the 386 processor, which was Intel's first 32-bit processor
- The 386 provided support for
  - 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

Historical Perspective

- 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



Historical Perspective

Intel Processor	Date Introduced	Max. Clock Frequency at Introduction	Transistors per Die	Register Sizes[1]	Ext. Data Bus Size[2]	Max. Extern. Addr. Space	Caches
8086	1978	8 MHz	29 K	16 GP	16	1 MB	None
Intel 286	1982	12.5 MHz	134 K	16 GP	16	16 MB	Note 3
Intel386 DX Processor	1985	20 MHz	275 K	32 GP	32	4 GB	Note 3
Intel486 DX Processor	1989	25 MHz	1.2 M	32 GP 80 FPU	32	4 GB	L1: 8KB
Pentium Processor	1993	60 MHz	3.1 M	32 GP 80 FPU	64	4 GB	L1:16KB
Pentium Pro Processor	1995	200 MHz	5.5 M	32 GP 80 FPU	64	64 GB	L1: 16KB L2: 256KB or 512KB
Pentium II Processor	1997	266 MHz	7 M	32 GP 80 FPU 64 MMX	64	64 GB	L1: 32KB L2: 256KB or 512KB
Pentium III Processor	1999	500 MHz	8.2 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32KB L2: 512KB
Pentium III and Pentium III Xeon Processors	1999	730 MHz	28 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32KB L2: 256KB

Historical Perspective

Intel Processor	Date Introduced	Micro-Architecture	Clock Frequency at Introduction	Transistors Per Die	Register Sizes[1]	System Bus Bandwidth	Max. Extern. Addr. Space	On-Die Cache[3]
Pentium 4 Processor	2000	Intel NetBurst Micro-architecture	1.50 GHz	42 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K μop Execution Trace Cache; 8KB L1; 256 KB L2
Intel Xeon Processor	2001	Intel NetBurst Micro-architecture	1.70 GHz	42 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K μop Trace Cache; 8 KB L1; 256 KB L2
Intel Xeon Processor	2002	Intel NetBurst Micro-architecture; Hyper-Threading Technology	2.20 GHz	55 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K μop Trace Cache; 8 KB L1; 512 KB L2
Intel Xeon Processor MP	2002	Intel NetBurst Micro-architecture; Hyper-Threading Technology	1.60 GHz	108 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K μop Trace Cache; 8 KB L1; 256 KB L2; 1 MB L3
Intel Pentium 4 Processor supporting Hyper-Threading Technology	2002	Intel NetBurst Micro-architecture; Hyper-Threading Technology	3.06 GHz	55M	GP: 32 FPU: 80 MMX: 64 XMM: 128	4.2 GB/s	64 GB	12K μop Execution Trace Cache; 8KB L1; 512 KB L2
Intel Pentium M Processor	2003	Intel Pentium M Processor	1.60 GHz	77M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	L1: 64KB L2: 1MB

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

7

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

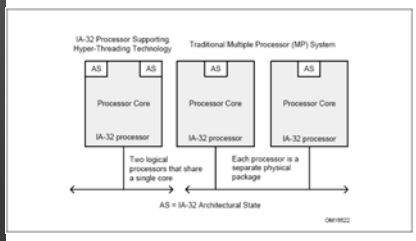
8

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 processor, each with its own copies of data, control, and segment registers, as well as debug and interrupt control

9

Hyper-Threading Technology



10

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

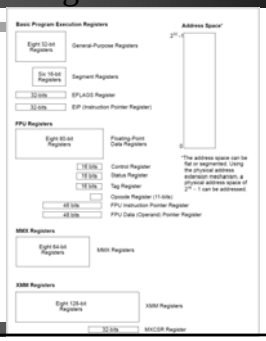
11

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

12

The IA-32 Register Model



13

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.**

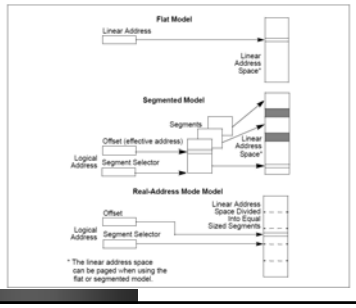
14

The Basic Memory Model

- ***Segmented 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**

15

The Memory Management Model



16

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**

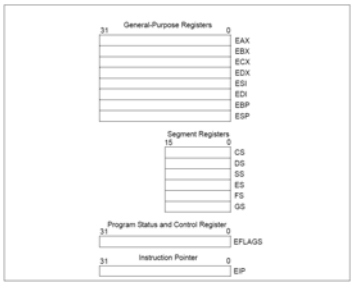
17

The Register Model

- **The IA-32 provides a number of general and special-purpose registers**
  - **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**

18

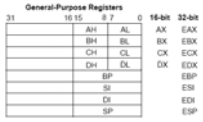
The Register Model



19

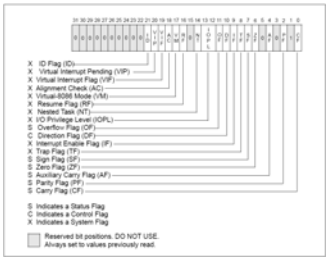
The Register Model

- 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



20

EFLAGS



21

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

22

Segmentation over Flat Memory

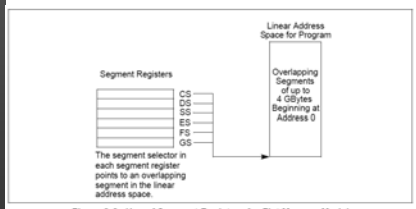


Figure 3-6. Use of Segment Registers for Flat Memory Model

23

Segmentation over Segmented Memory

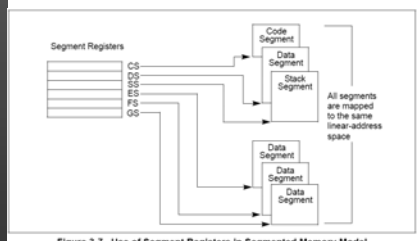
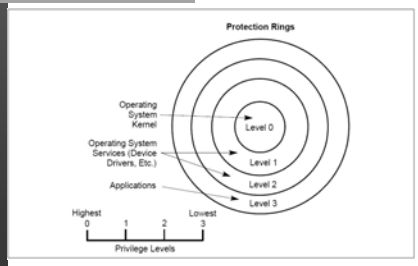


Figure 3-7. Use of Segment Registers in Segmented Memory Model

24

Protection Rings



25

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**

26

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**

27

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**

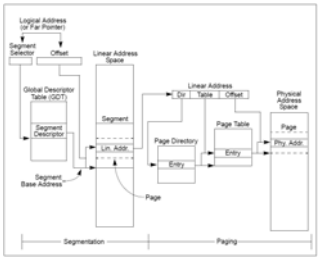
28

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 eachother**
- **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**

29

Segmentation and Paging

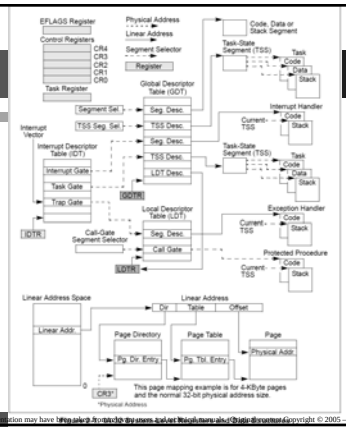


30

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**

31



32