

---

# Basic Computer Organization

Chapter 2  
S. Dandamudi

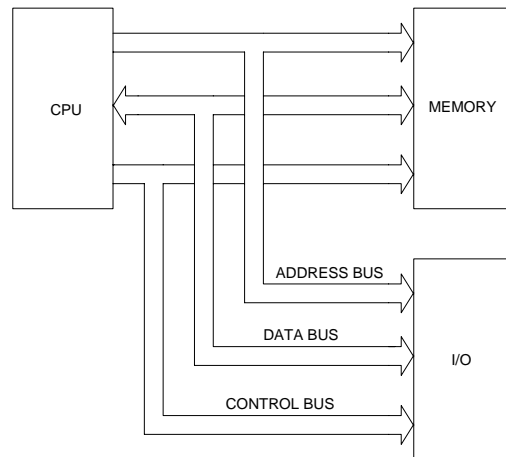
---

---

## Outline

- Basic components of a computer system
  - » Processor
  - » Memory
  - » I/O
  - » System bus
- The processor
  - \* Pentium processor details
    - » Pentium Registers
- The memory
  - \* Basic memory operations
  - \* Types of memory
  - \* Storing multibyte data
- Pentium memory architecture
  - \* Real mode
  - \* Protected mode
- Input/output
  - \* Basic I/O operations
  - \* Memory-mapped I/O
  - \* Isolated I/O
- Performance: Effect of data alignment
  - » Use bubble sort example

## Basic Components of a Computer System



1998

© S. Dandamudi

Introduction: Page 3

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## The Processor

- Processor can be thought of executing the fetch-decode-execute cycle forever
  - » Fetch an instruction from the memory
  - » Decode the instruction
    - Find out what the operation is
  - » Execute the instruction
    - Perform the specified operation

|← execution cycle →|



→ time

1998

© S. Dandamudi

Introduction: Page 4

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## The Pentium Processor

---

- Pentium registers
  - \* Pentium has ten 32-bit registers and six 16-bit registers
    - » Internal to the processor
    - » Provide faster access
  - \* These registers are grouped into
    - » General registers
      - Data registers
      - Pointer registers
      - Index registers
    - » Control registers
    - » Segment registers

1998

© S. Dandamudi

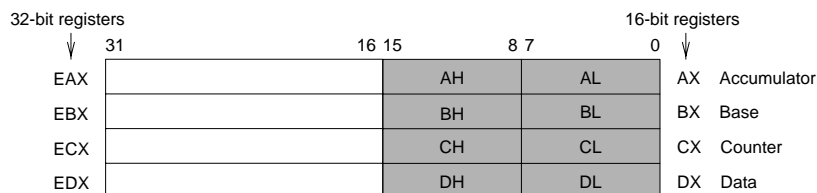
Introduction: Page 5

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## The Pentium Processor (cont'd)

---

- Data registers
  - \* Four 32-bit registers
  - \* Can be used as
    - Four 32-bit register (EAX, EBX, ECX, EDX)
    - Four 16-bit registers (AX, BX, CX, DX)
    - Eight 8-bit registers (AH, AL, BH, BL, CH, CL, DH, DL)
    - A valid combination of these



1998

© S. Dandamudi

Introduction: Page 6

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.



## The Pentium Processor (cont'd)

- Segment registers

- \* Necessary to support segmented memory organization of Pentium

|    |    |               |
|----|----|---------------|
| 15 |    | 0             |
|    | CS | Code Segment  |
|    | DS | Data Segment  |
|    | SS | Stack Segment |
|    | ES | Extra Segment |
|    | FS | Extra Segment |
|    | GS | Extra Segment |

1998

© S. Dandamudi

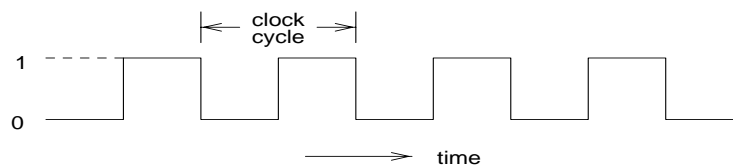
Introduction: Page 9

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## The Pentium Processor (cont'd)

- System clock

- » System clock provides timing signal to synchronize the operations of the whole system
- \* Clock period = length of time taken by one clock cycle  
clock period =  $1/\text{clock rate}$
- \* Example: A 200 MHz clock yields a clock period of  
 $1/(200 * 10^6) = 5 \text{ ns}$



1998

© S. Dandamudi

Introduction: Page 10

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Intel 80X86 Family Processors

---

- 8086 in 1979
  - \* 20-bit address bus (1MB)
  - \* 16-bit data bus
  - \* No floating-point coprocessor
- 80286 in 1982
  - \* 24-bit address bus (16MB)
  - \* 16-bit data bus
  - \* No floating-point coprocessor
  - \* Memory protection capabilities
- 80386 in 1985
  - \* First 32-bit processor
  - \* 32-bit address bus (4GB)
  - \* 32-bit data bus
  - \* No floating-point coprocessor
- 80486 in 1989
  - \* Improved version of 386
  - \* 32-bit address bus (4GB)
  - \* 32-bit data bus
  - \* Includes floating-point coprocessor and internal cache

1998

© S. Dandamudi

Introduction: Page 11

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Intel 80X86 Family Processors (cont'd)

---

- Pentium in 1993
  - \* Enhanced version of 486
  - \* 32-bit address bus (4GB)
  - \* 64-bit data bus
  - \* All internal registers are 32-bit wide
    - » Still a 32-bit processor
    - » All instructions operate on at most 32-bit operands
  - \* Several versions
    - » Some versions are stripped down for under \$1000 PC market
    - » Some enhanced with MMX technology

1998

© S. Dandamudi

Introduction: Page 12

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Memory

- Memory can be viewed as an ordered sequence of bytes
- Each byte of memory has an address
  - \* Memory address is essentially the sequence number of the byte
  - \* Such memories are called *byte addressable*
  - \* Number of address lines determine the memory address space of a processor

| Address<br>(in decimal) |             | Address<br>(in hex) |
|-------------------------|-------------|---------------------|
| $2^{32}-1$              |             | FFFFFFFF            |
|                         |             | FFFFFFFE            |
|                         |             | FFFFFFFD            |
|                         | •<br>•<br>• |                     |
| 2                       |             | 00000002            |
| 1                       |             | 00000001            |
| 0                       |             | 00000000            |

1998

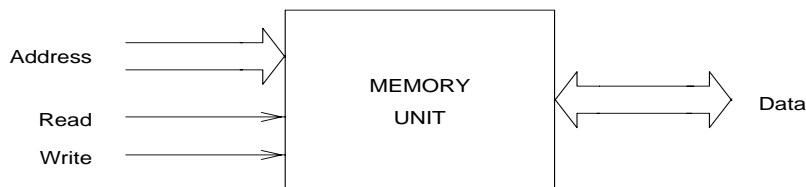
© S. Dandamudi

Introduction: Page 13

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Memory (cont'd)

- Two basic memory operations
  - » Read operation (read from memory)
  - » Write operation (write into memory)
- Access time
  - » Time needed to retrieve data at addressed location
- Cycle time
  - » Minimum time between successive operations



1998

© S. Dandamudi

Introduction: Page 14

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Memory (cont'd)

---

- Steps in a typical read cycle
  - » Place the address of the location to be read on the address bus
  - » Activate the memory read control signal on the control bus
  - » Wait for the memory to retrieve the data from the addressed memory location
  - » Read the data from the data bus
  - » Drop the memory read control signal to terminate the read cycle
- \* A simple Pentium memory read cycle takes 3 clocks
  - Steps 1&2 and 4&5 are done in one clock cycle each
- \* For slower memories, *wait cycles* will have to be inserted

## Memory (cont'd)

---

- Steps in a typical write cycle
  - » Place the address of the location to be written on the address bus
  - » Place the data to be written on the data bus
  - » Activate the memory write control signal on the control bus
  - » Wait for the memory to store the data at the addressed location
  - » Drop the memory write control signal to terminate the write cycle
- \* A simple Pentium memory write cycle takes 3 clocks
  - Steps 1&3 and 4&5 are done in one clock cycle each
- \* For slower memories, *wait cycles* will have to be inserted



## Memory (cont'd)

---

- Some properties of memory
  - \* Random access
    - » Accessing any memory location takes the same amount of time
  - \* Volatility
    - » Volatile memory
      - Needs power to retain the contents
    - » Non-volatile memory
      - Retains contents even in the absence of power
- Basic types of memory
  - \* Read-only memory (ROM)
  - \* Read/write memory (RAM)

1998

© S. Dandamudi

Introduction: Page 17

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Memory (cont'd)

---

- Read-only memory (ROM)
    - » Cannot be written into this type of memory
    - » Non-volatile memory
    - » Most are factory programmed (i.e., written)
  - \* Programmable ROMs (PROMs)
    - » Can be written once by user
      - A fuse is associated with each bit cell
      - Special equipment is needed to write (to blow the fuse)
    - » PROMS are useful
      - During prototype development
      - If the required quantity is small
- Does not justify the cost of factory programmed ROM

1998

© S. Dandamudi

Introduction: Page 18

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Memory (cont'd)

---

- \* Erasable PROMs (EPROMs)
  - » Can be written several times
  - » Offers further flexibility during system prototyping
  - » Can be erased by exposing to ultraviolet light
    - Cannot erase contents of selected locations
    - All contents are lost
- \* Electrically erasable PROMs (EEPROMs)
  - » Contents are electrically erased
  - » No need to erase all contents
    - Typically a subset of the locations are erased as a group
    - Most EEPROMs do not provide the capability to individually erase contents of a single location

## Memory (cont'd)

---

- Read/write memory
  - » Commonly referred to as random access memory (RAM)
  - » Volatile memories
- \* Two basic types
  - » Static RAM (SRAM)
    - Retains data with no further maintenance
    - Typically used for CPU registers and cache memory
  - » Dynamic RAM (DRAM)
    - A tiny capacitor is used to store a bit
    - Due to leakage of charge, DRAMs must be *refreshed* to retain contents
    - Read operation is destructive in DRAMs

## Memory (cont'd)

### \* SRAM versus DRAM

- » SRAMs are expensive but faster
  - Typical access time in 10-20 ns range
  - Cost in \$200-400 per MB
- » DRAMs are cheaper but slower
  - Typical access time in 50-100 ns range
  - Cost in \$30-100 per MB
  - Typically used as main memory
    - Due to the cost advantage
    - Cache memory is used to handle the slow access time of DRAM
  - A wide variety of implementations are available

1998

© S. Dandamudi

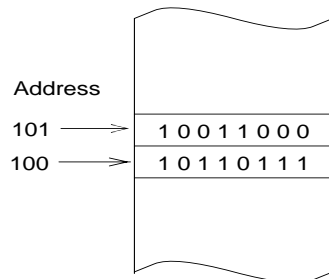
Introduction: Page 21

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

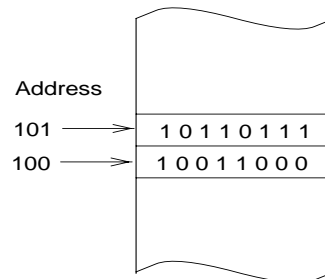
## Storing Multibyte Data

1 0 0 1 1 0 0 0 | 1 0 1 1 0 1 1 1

(a) 16-bit data



(b) Little endian byte ordering



(c) Big endian byte ordering

1998

© S. Dandamudi

Introduction: Page 22

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Storing Multibyte Data (cont'd)

---

- Little endian
  - » used by Intel 80x86 processors
- Big endian
  - » used by Motorola 680x0 processors
- PowerPC supports both byte orderings
  - » Big endian is the default
- Not a problem when working with same type of machines
  - » Need to convert the format if working with a different machine
  - » Pentium provides two instructions for conversion
    - **xchg** for 16-bit data
    - **bswap** for 32-bit data

1998

© S. Dandamudi

Introduction: Page 23

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Pentium Memory Architecture

---

- Two modes
  - \* Real mode
    - » Uses 16-bit addresses
    - » Supports segmented memory architecture
    - » Provides backward compatibility
      - To run 8086 programs
  - \* Protected mode
    - » Native mode of Pentium
    - » Uses 32-bit addresses
    - » Supports segmentation and paging
      - Paging is useful to implement virtual memory
- Our focus is on the segmented memory architecture

1998

© S. Dandamudi

Introduction: Page 24

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Real Mode Memory Architecture

- Real mode memory architecture
  - \* Pentium operates like a faster 8086 processor
  - \* The 8086 has
    - » 20 address lines (can address 1 MB)
    - » All registers are 16-bit wide
  - \* Memory is organized as segments of (up to) 64 KB
    - Due to 16-bit registers ( $2^{16} = 64 \text{ K}$ )
  - \* Two components are required to specify a location
    - » Segment base address (segment start address)
    - » Offset within a segment
    - » Both are 16-bit numbers

1998

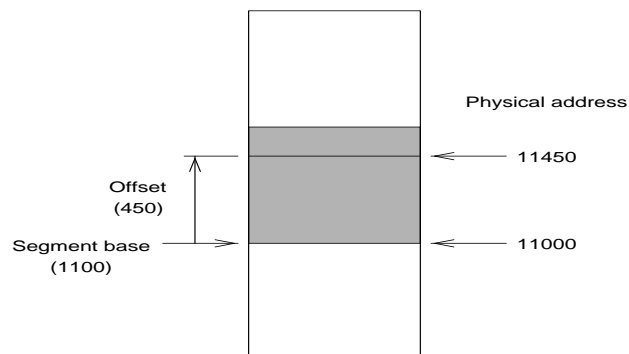
© S. Dandamudi

Introduction: Page 25

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Real Mode Memory Architecture (cont'd)

- \* 16-bit segment base address restricts segments to start at addresses that are multiple of 16
  - Segments can only start at addresses 0, 16, 32, 48, ...



1998

© S. Dandamudi

Introduction: Page 26

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Real Mode Memory Architecture (cont'd)

- Logical address consists of two 16-bit components  
**segment:offset**
- Physical address is a 20-bit number
- Translation from logical to physical address:
  - » Append four least significant zeros to the segment base address
    - Just add a zero when using hex numbers
  - » Add the offset value
- Example: Translate logical address 1100:450
$$\begin{array}{r} 11000 \text{ (add zero to segment address)} \\ + \quad 450 \text{ (offset value)} \\ \hline 11450 \text{ (20-bit physical address)} \end{array}$$

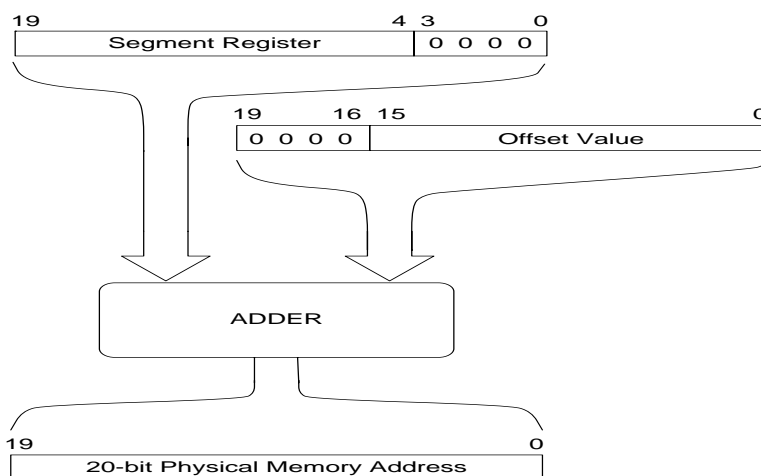
1998

© S. Dandamudi

Introduction: Page 27

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Real Mode Memory Architecture (cont'd)



1998

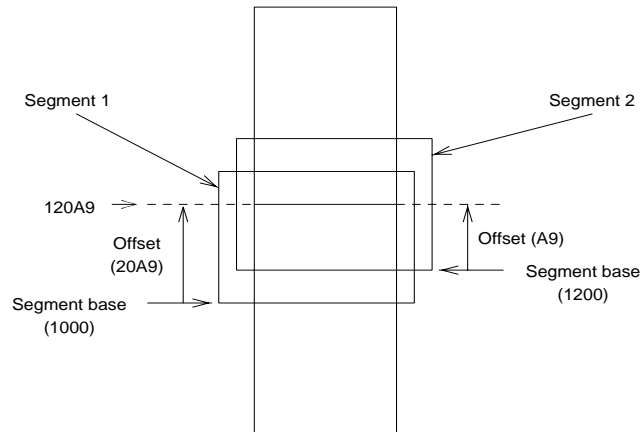
© S. Dandamudi

Introduction: Page 28

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Real Mode Memory Architecture (cont'd)

Multiple logical addresses can map to the same physical address



1998

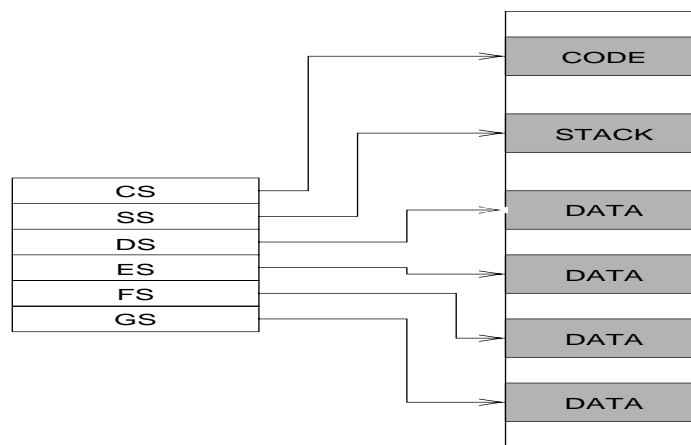
© S. Dandamudi

Introduction: Page 29

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Real Mode Memory Architecture (cont'd)

Six segments of the memory system in real mode



1998

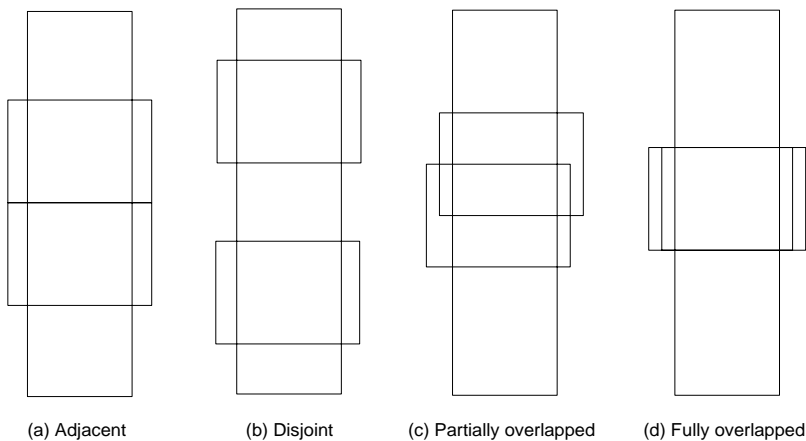
© S. Dandamudi

Introduction: Page 30

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Real Mode Memory Architecture (cont'd)

Various ways of placing segments in the memory



1998

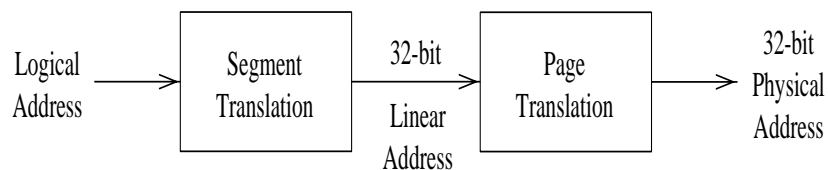
© S. Dandamudi

Introduction: Page 31

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Protected Mode Architecture

- In protected mode, Pentium supports
  - \* More sophisticated segmentation
    - » Segmentation can be made invisible (flat model)
  - \* Paging for virtual memory
    - » Paging can be turned off



1998

© S. Dandamudi

Introduction: Page 32

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.



## Protected Mode Architecture (cont'd)

- Uses 16-bit segment registers
  - » Real or protected mode, segments registers are 16-bit wide
- Segment register provides an index into a segment descriptor table
  - » Does not provide segment start address as in the real mode
- Segment register provides:
  - » 13-bit index value into a segment descriptor table (INDEX)
    - Can select one of  $2^{13} = 8192$  descriptors
  - » 1-bit table indicator (TI)
    - Local (1) or global (0) descriptor table
  - » 2-bit requestor privilege level (RPL)
    - Privilege level provides protected access
    - Smaller the RPL ==> higher the privilege

1998

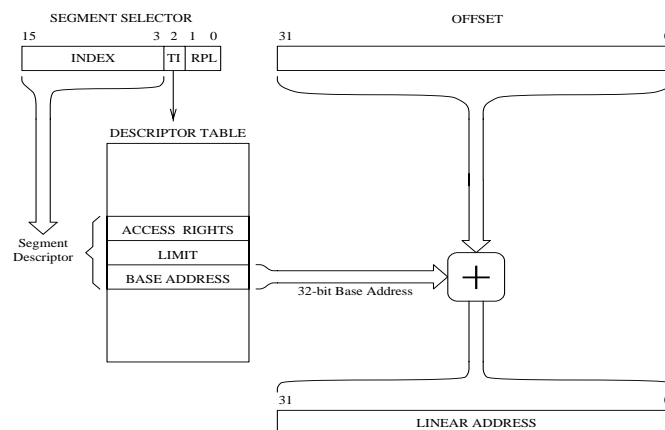
© S. Dandamudi

Introduction: Page 33

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Protected Mode Architecture (cont'd)

### Protected mode segment translation



1998

© S. Dandamudi

Introduction: Page 34

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Protected Mode Architecture (cont'd)

- Segment registers have two parts:
  - » Visible part
    - This is the 16-bit part we talked about
  - » Invisible part
    - Stores the segment descriptor associated with the index
      - Keeps the base address, size, access rights etc.

| Visible Part     | Invisible Part                                  |    |
|------------------|---|----|
| Segment Selector | Segment Base Address, Size, Access Rights, etc. | CS |
| Segment Selector | Segment Base Address, Size, Access Rights, etc. | SS |
| Segment Selector | Segment Base Address, Size, Access Rights, etc. | DS |
| Segment Selector | Segment Base Address, Size, Access Rights, etc. | ES |
| Segment Selector | Segment Base Address, Size, Access Rights, etc. | FS |
| Segment Selector | Segment Base Address, Size, Access Rights, etc. | GS |

1998

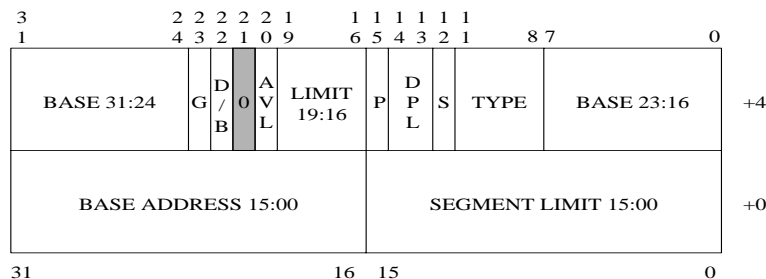
© S. Dandamudi

Introduction: Page 35

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Protected Mode Architecture (cont'd)

- Segment descriptor provides attributes of a segment
  - » 32-bit base address
  - » 20-bit segment size
  - » Control and status information



1998

© S. Dandamudi

Introduction: Page 36

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Protected Mode Architecture (cont'd)

---

- Segment descriptor tables
  - \* One global descriptor table (GDT)
    - Available to all tasks within the system
    - Typically used for operating system code and data
  - \* Local descriptor table (LDT)
    - Can be several LDTs
      - Each contains descriptors for code, data, stack, ...
    - A segment cannot be accessed by a program unless there is a descriptor for the segment either in the current LDT or GDT
  - \* One Interrupt descriptor table (IDT)
    - Used in interrupt processing

1998

© S. Dandamudi

Introduction: Page 37

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Protected Mode Architecture (cont'd)

---

- Segmentation models
  - \* Flat model
    - » Segmentation can be made invisible
      - Use a single segment of 4 GB size
    - » Useful for programming environments such as UNIX
  - \* Multisegment model
    - » Uses capabilities of segmentation to the full extent
    - » There can be up to six active segments
    - » A program can have more than six segments
    - » A segment that is not active can be made active
      - By loading its selector into a segment register
      - Processor automatically gets the segment descriptor into the "invisible" part of the segment register

1998

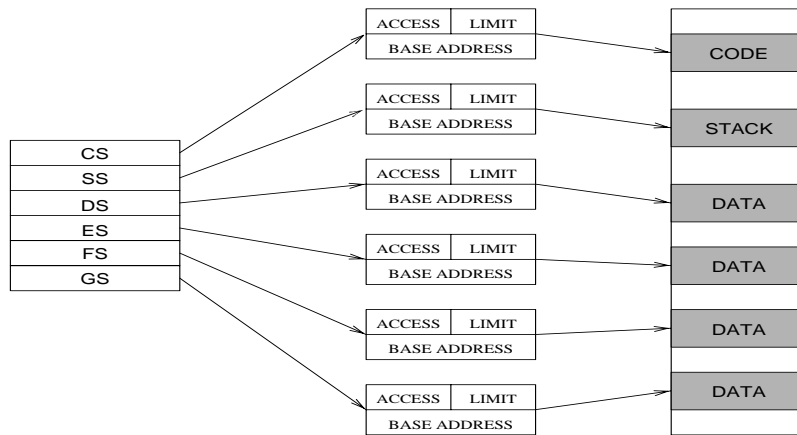
© S. Dandamudi

Introduction: Page 38

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Protected Mode Architecture (cont'd)

Segments in a multisegment model



1998

© S. Dandamudi

Introduction: Page 39

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Mixed Mode Operation

- Real mode
  - » Default: 16-bit model
- Protected mode
  - » Default: 32-bit model
- Default size is indicated by D/B bit of the segment descriptor
- We can mix 16- and 32-bit data and addresses in either real or protected mode
  - » Pentium provides two size override prefixes
    - One for data and one for addresses
    - But, there is a performance penalty

1998

© S. Dandamudi

Introduction: Page 40

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Default Segment Register Association

- Instruction fetch
  - » CS segment register
  - » IP or EIP provides offset
- Stack operations
  - » SS segment register
  - » SP or ESP provides the offset for operations like **pop** and **push**
  - » BP or EBP provide the offset for other operations
- Data
  - » DS segment register
  - » Offset: depends on the addressing mode

1998

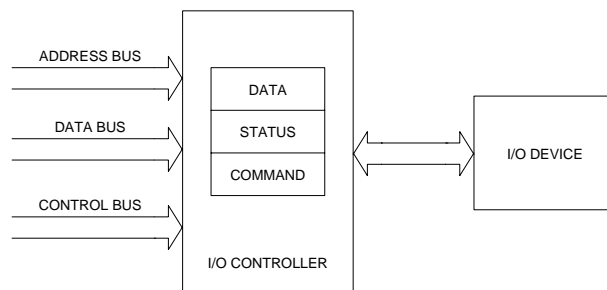
© S. Dandamudi

Introduction: Page 41

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Input/Output

- I/O controller provides the necessary interface to I/O devices
  - » Takes care of low-level, device-dependent details
  - » Provides necessary electrical signal interface



1998

© S. Dandamudi

Introduction: Page 42

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Input/Output (cont'd)

---

- Processor and I/O interface points for exchanging data are called *I/O ports*
- Two ways of mapping I/O ports
  - \* Memory-mapped I/O
    - » I/O ports are mapped to the memory address space
      - Reading/writing I/O is similar to reading/writing memory
        - Can use memory read/write instructions
    - » Motorola 68000 uses memory-mapped I/O
  - \* Isolated I/O
    - » Separate I/O address space
    - » Requires special I/O instructions (like **in** and **out** in Pentium)
    - » Intel 80x86 processors support isolated I/O

1998

© S. Dandamudi

Introduction: Page 43

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Input/Output (cont'd)

---

- Pentium I/O address space
  - \* Provides 64 KB I/O address space
  - \* Can be used for 8-, 16-, and 32-bit I/O ports
  - \* Combination cannot exceed the total I/O address space
    - » can have 64 K 8-bit ports
    - » can have 32 K 16-bit ports
    - » can have 16 K 32-bit ports
    - » A combination of these for a total of 64 KB
  - \* I/O instructions do not go through segmentation or paging
    - » I/O address refers to the physical I/O address

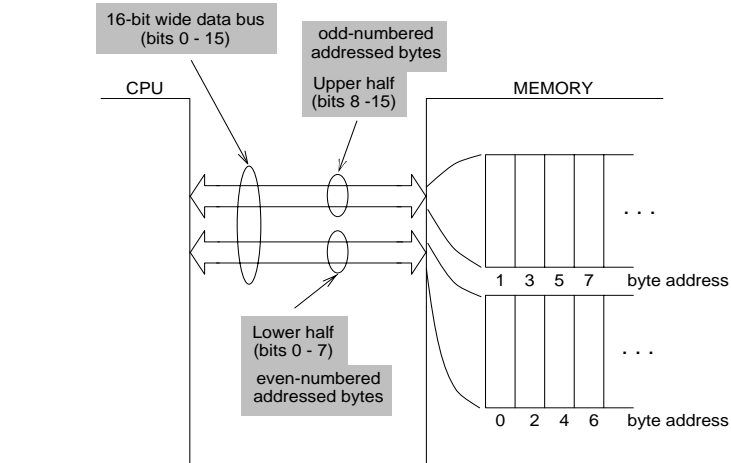
1998

© S. Dandamudi

Introduction: Page 44

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Performance: Effect of data Alignment



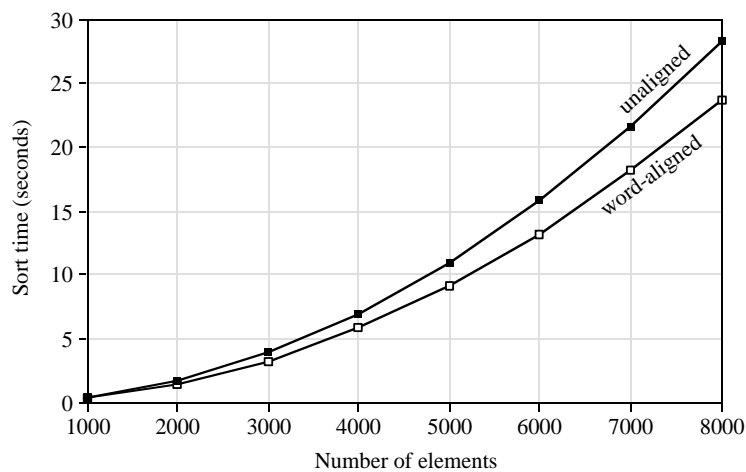
1998

© S. Dandamudi

Introduction: Page 45

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Performance: Effect of data Alignment (cont'd)



1998

© S. Dandamudi

Introduction: Page 46

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Performance: Effect of data Alignment (cont'd)

---

- Data alignment
  - \* Soft alignment
    - » Data is *not* required to be aligned
      - Data alignment is optional
        - Aligned data gives better performance
    - » Used in Intel 80X86 processors
  - \* Hard alignment
    - » Data *must* be aligned
    - » Used in Motorola 680X0 and Intel i860 processors

---

1998

© S. Dandamudi

Introduction: Page 47

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.

## Performance: Effect of data Alignment (cont'd)

---

- Data alignment requirements for byte addressable memories
  - \* 1-byte data
    - » Always aligned
  - \* 2-byte data
    - » Aligned if the data is stored at an even address (i.e., at an address that is a multiple of 2)
  - \* 4-byte data
    - » Aligned if the data is stored at an address that is a multiple of 4
  - \* 8-byte data
    - » Aligned if the data is stored at an address that is a multiple of 8

---

1998

© S. Dandamudi

Introduction: Page 48

To be used with S. Dandamudi, "Introduction to Assembly Language Programming," Springer-Verlag, 1998.