IA-32 Architecture

COE 205

Computer Organization and Assembly Language

Dr. Aiman El-Maleh

College of Computer Sciences and Engineering

King Fahd University of Petroleum and Minerals

[Adapted from slides of Dr. Kip Irvine: Assembly Language for Intel-Based Computers]

Outline

- ❖ Intel Microprocessors
- ❖ IA-32 Registers
- ❖ Instruction Execution Cycle
- ❖ IA-32 Memory Management

Intel Microprocessors

- Intel introduced the 8086 microprocessor in 1979
- 8086, 8087, 8088, and 80186 processors
  - 16-bit processors with 16-bit registers
  - 16-bit data bus and 20-bit address bus
    - Physical address space = 2^{20} bytes = 1 MB
  - 8087 Floating-Point co-processor
  - Uses segmentation and real-address mode to address memory
    - Each segment can address 2^{16} bytes = 64 KB
  - 8088 is a less expensive version of 8086
    - Uses an 8-bit data bus
  - 80186 is a faster version of 8086

Intel 80286 and 80386 Processors

- 80286 was introduced in 1982
  - 24-bit address bus ⇒ 2^{24} bytes = 16 MB address space
  - Introduced protected mode
    - Segmentation in protected mode is different from the real mode
- 80386 was introduced in 1985
  - First 32-bit processor with 32-bit general-purpose registers
  - First processor to define the IA-32 architecture
  - 32-bit data bus and 32-bit address bus
  - 2^{32} bytes ⇒ 4 GB address space
  - Introduced paging, virtual memory, and the flat memory model
    - Segmentation can be turned off

Intel 80486 and Pentium Processors

- 80486 was introduced 1989
  - Improved version of Intel 80386
  - On-chip Floating-Point unit (DX versions)
  - On-chip unified Instruction/Data Cache (8 KB)
  - Uses Pipelining: can execute up to 1 instruction per clock cycle
- Pentium (80586) was introduced in 1993
  - Wider 64-bit data bus, but address bus is still 32 bits
  - Two execution pipelines: U-pipe and V-pipe
    - Superscalar performance: can execute 2 instructions per clock cycle
  - Separate 8 KB instruction and 8 KB data caches
  - MMX instructions (later models) for multimedia applications

Intel P6 Processor Family

- P6 Processor Family: Pentium Pro, Pentium II and III
- Pentium Pro was introduced in 1995
  - Three-way superscalar: can execute 3 instructions per clock cycle
  - 36-bit address bus ⇒ up to 64 GB of physical address space
  - Introduced dynamic execution
    - Out-of-order and speculative execution
  - Integrates a 256 KB second level L2 cache on-chip
- Pentium II was introduced in 1997
  - Added MMX instructions (already introduced on Pentium MMX)
- Pentium III was introduced in 1999
  - Added SSE instructions and eight new 128-bit XMM registers

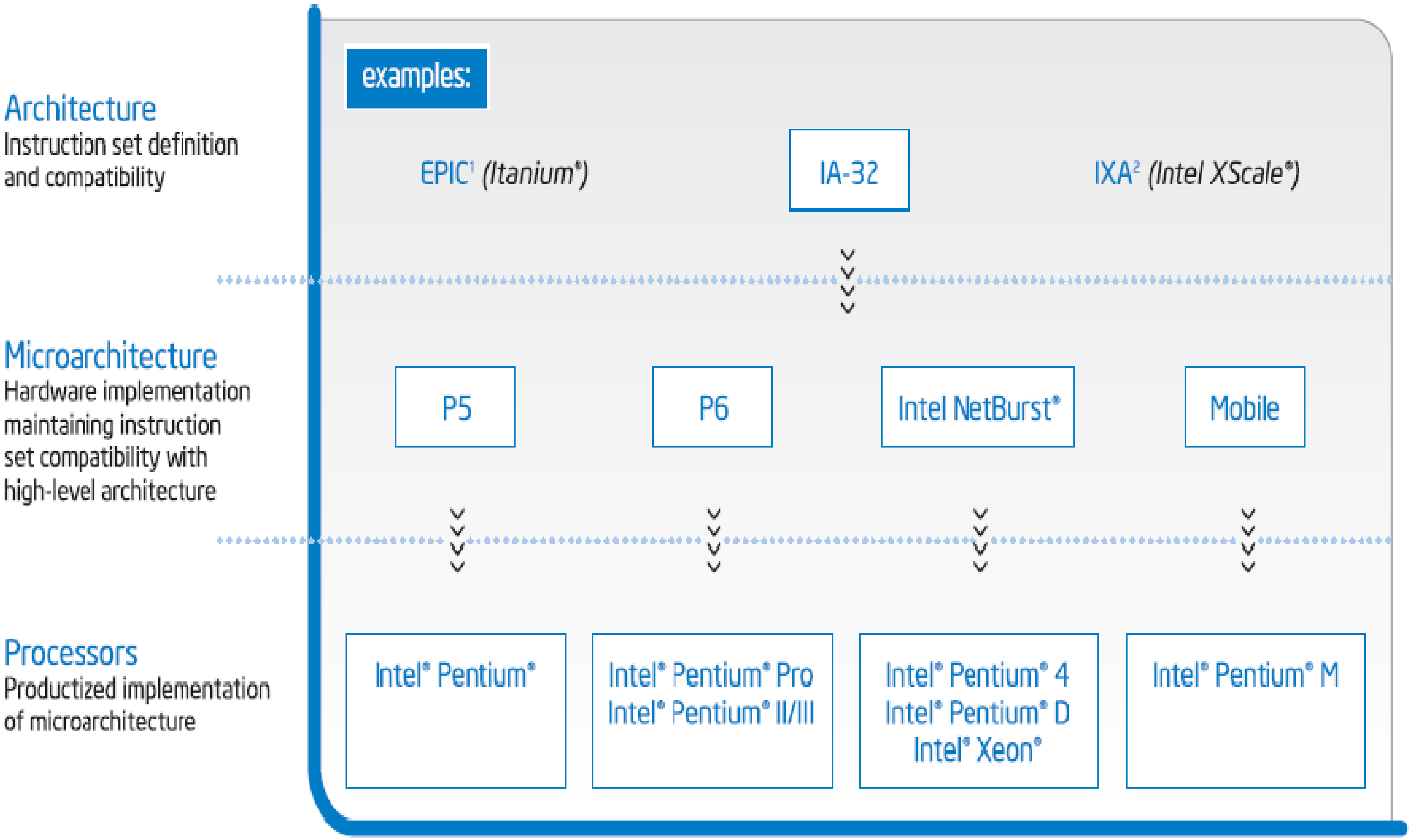
Pentium 4 and Xeon Family

- Pentium 4 is a seventh-generation x86 architecture
  - Introduced in 2000
  - New micro-architecture design called Intel Netburst
  - Very deep instruction pipeline, scaling to very high frequencies
  - Introduced the SSE2 instruction set (extension to SSE)
    - Tuned for multimedia and operating on the 128-bit XMM registers
- In 2002, Intel introduced Hyper-Threading technology
  - Allowed 2 programs to run simultaneously, sharing resources
- Xeon is Intel's name for its server-class microprocessors
  - Xeon chips generally have more cache
  - Support larger multiprocessor configurations

Pentium-M and EM64T

- Pentium M (Mobile) was introduced in 2003
  - Designed for low-power laptop computers
  - Modified version of Pentium III, optimized for power efficiency
  - Large second-level cache (2 MB on later models)
  - Runs at lower clock than Pentium 4, but with better performance
- Extended Memory 64-bit Technology (EM64T)
  - Introduced in 2004
  - 64-bit superset of the IA-32 processor architecture
  - 64-bit general-purpose registers and integer support
  - Number of general-purpose registers increased from 8 to 16
  - 64-bit pointers and flat virtual address space
  - Large physical address space: up to 2^{40} = 1 Terabytes

Intel MicroArchitecture History



Architecture
Instruction set definition and compatibility

Microarchitecture
Hardware implementation maintaining instruction set compatibility with high-level architecture

Processors
Productized implementation of microarchitecture

examples:

EPIC[1] *(Itanium®)*

IA-32

IXA[2] *(Intel XScale®)*

P5

P6

Intel NetBurst®

Mobile

Intel® Pentium®

Intel® Pentium® Pro
Intel® Pentium® II/III

Intel® Pentium® 4
Intel® Pentium® D
Intel® Xeon®

Intel® Pentium® M

Intel Core MicroArchitecture

- 64-bit cores
- Wide dynamic execution (execute four instructions simultaneously)
- Intelligent power capability (power gating)
- Advanced smart cache (shares L2 cache between cores)
- Smart memory access (memory disambiguation)
- Advanced digital media boost

See the demo at http://www.intel.com/technology/architecture/coremicro/demo/demo.htm?iid=tech_core+demo

CISC and RISC

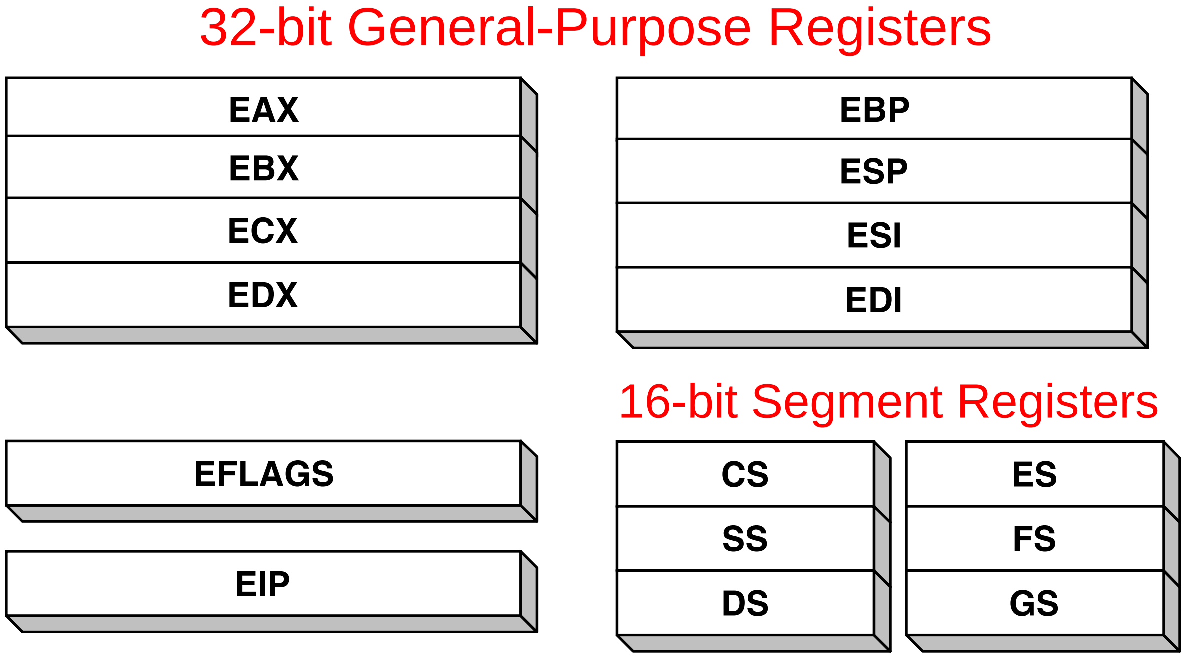
- CISC – Complex Instruction Set Computer
  - Large and complex instruction set
  - Variable width instructions
  - Requires microcode interpreter
    - Each instruction is decoded into a sequence of micro-operations
  - Example: Intel x86 family
- RISC – Reduced Instruction Set Computer
  - Small and simple instruction set
  - All instructions have the same width
  - Simpler instruction formats and addressing modes
  - Decoded and executed directly by hardware
  - Examples: ARM, MIPS, PowerPC, SPARC, etc.

Next ...

- ❖ Intel Microprocessors
- ❖ IA-32 Registers
- ❖ Instruction Execution Cycle
- ❖ IA-32 Memory Management

Basic Program Execution Registers

- Registers are high speed memory inside the CPU
  - Eight 32-bit general-purpose registers
  - Six 16-bit segment registers
  - Processor Status Flags (EFLAGS) and Instruction Pointer (EIP)

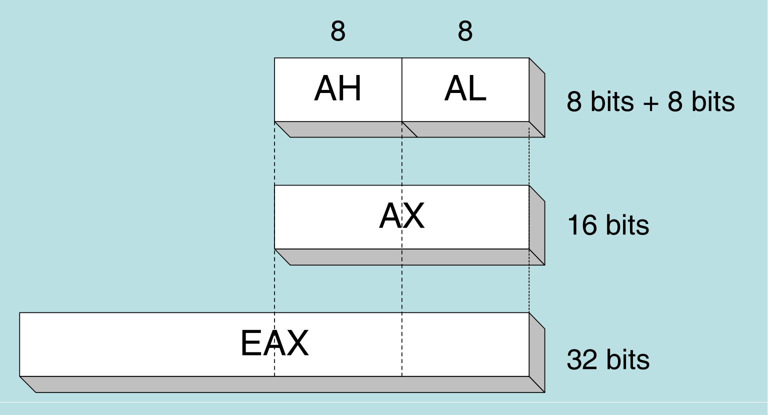


General-Purpose Registers

- Used primarily for arithmetic and data movement
  - `mov eax, 10` move constant 10 into register eax
- Specialized uses of Registers
  - EAX – Accumulator register
    - Automatically used by multiplication and division instructions
  - ECX – Counter register
    - Automatically used by LOOP instructions
  - ESP – Stack Pointer register
    - Used by PUSH and POP instructions, points to top of stack
  - ESI and EDI – Source Index and Destination Index register
    - Used by string instructions
  - EBP – Base Pointer register
    - Used to reference parameters and local variables on the stack

Accessing Parts of Registers

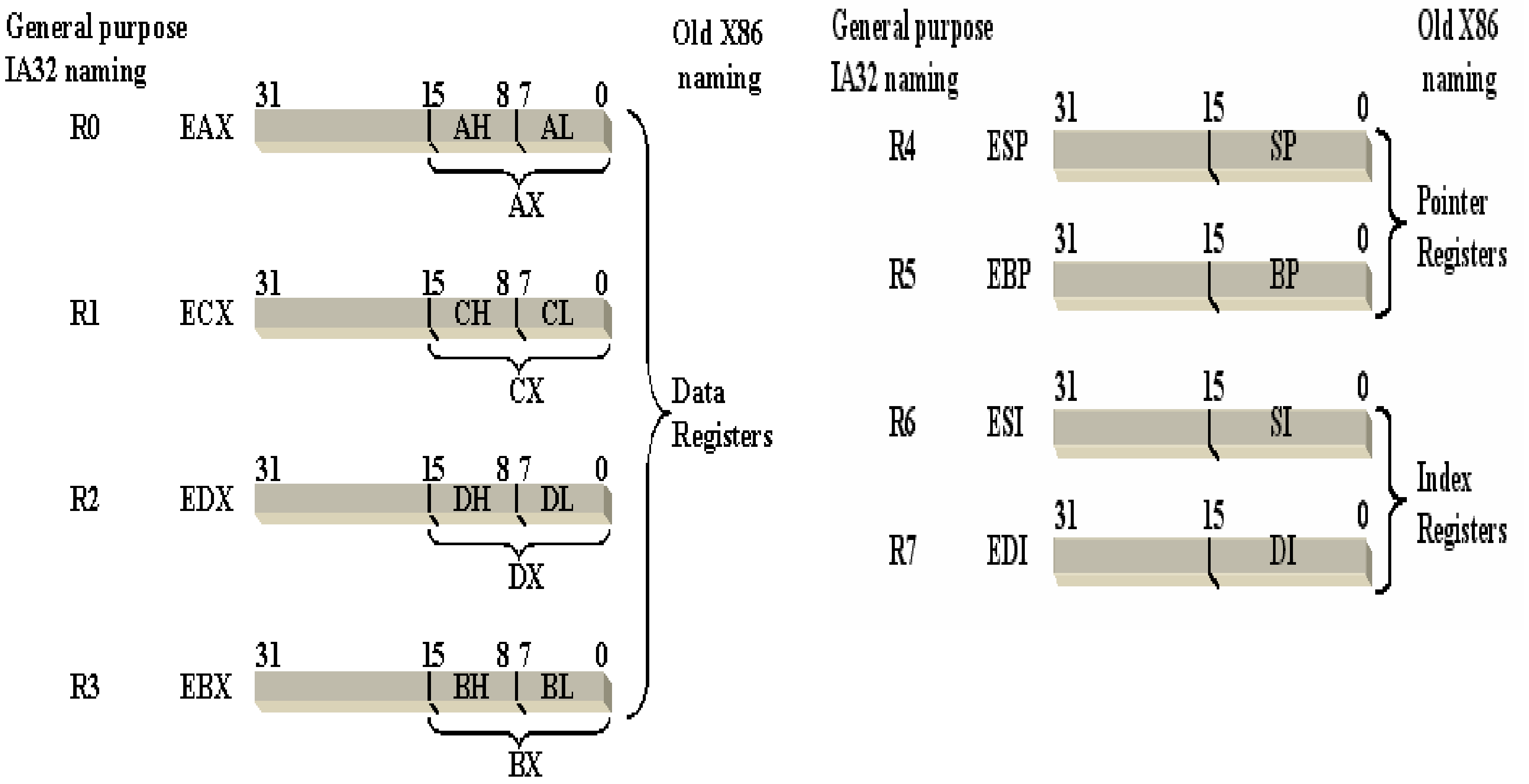
- EAX, EBX, ECX, and EDX are 32-bit Extended registers
  - Programmers can access their 16-bit and 8-bit parts
  - Lower 16-bit of EAX is named AX
  - AX is further divided into
    - AL = lower 8 bits
    - AH = upper 8 bits
- ESI, EDI, EBP, ESP have only 16-bit names for lower half



32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

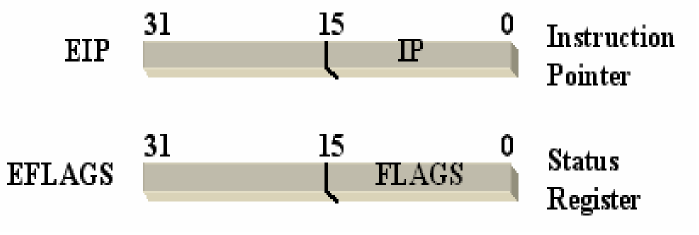
32-bit	16-bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

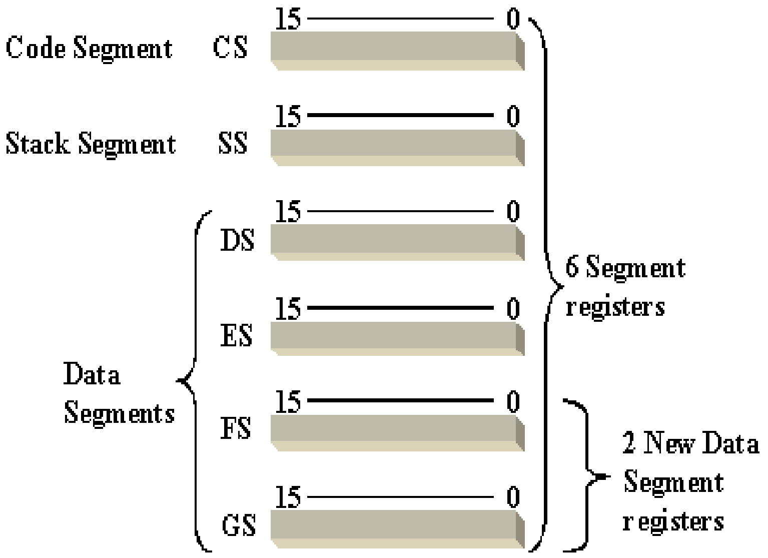
Accessing Parts of Registers



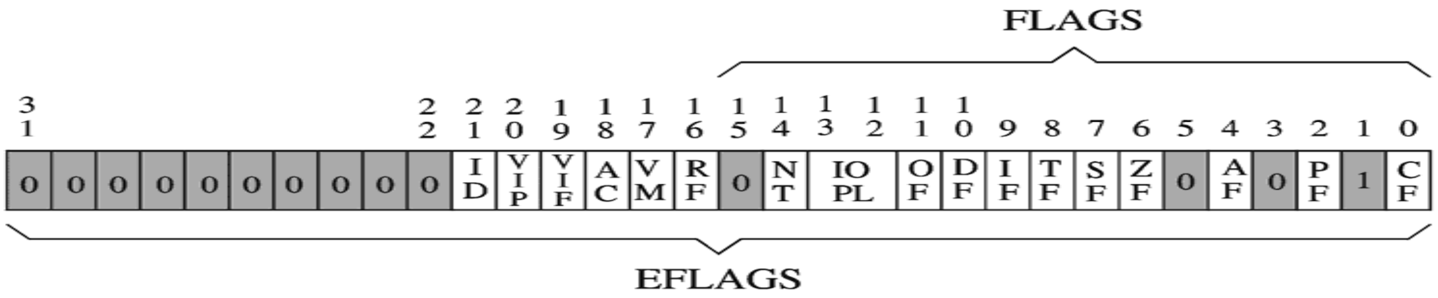
Special-Purpose & Segment Registers

- EIP = Extended Instruction Pointer
  - Contains address of next instruction to be executed
- EFLAGS = Extended Flags Regist
  - Contains status and control flags
  - Each flag is a single binary bit
- Six 16-bit Segment Registers
  - Support segmented memory
  - Six segments accessible at a time
  - Segments contain distinct contents
    - Code
    - Data
    - Stack





EFLAGS Register


FLAGS
31 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
EFLAGS

Status flags

CF = Carry flag
PF = Parity flag
AF = Auxiliary carry flag
ZF = Zero flag
SF = Sign flag
OF = Overflow flag

Control flags

DF = Direction flag

System flags

TF = Trap flag
IF = Interrupt flag
IOPL = I/O privilege level
NT = Nested task
RF = Resume flag
VM = Virtual 8086 mode
AC = Alignment check
VIF = Virtual interrupt flag
VIP = Virtual interrupt pending
ID = ID flag

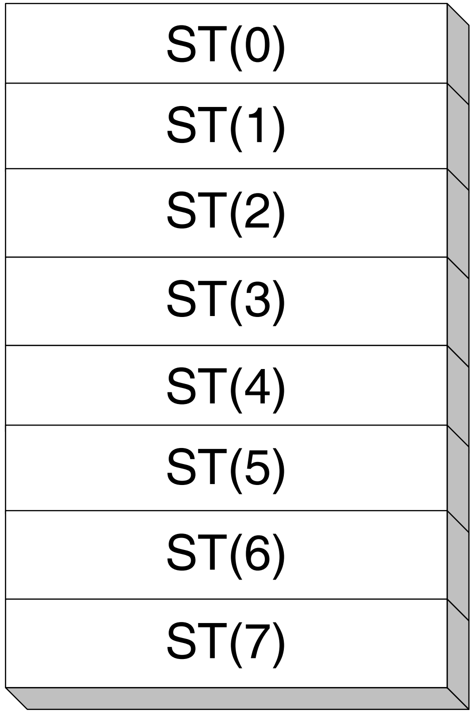
- Status Flags
  - Status of arithmetic and logical operations
- Control and System flags
  - Control the CPU operation
- Programs can set and clear individual bits in the EFLAGS register

Status Flags

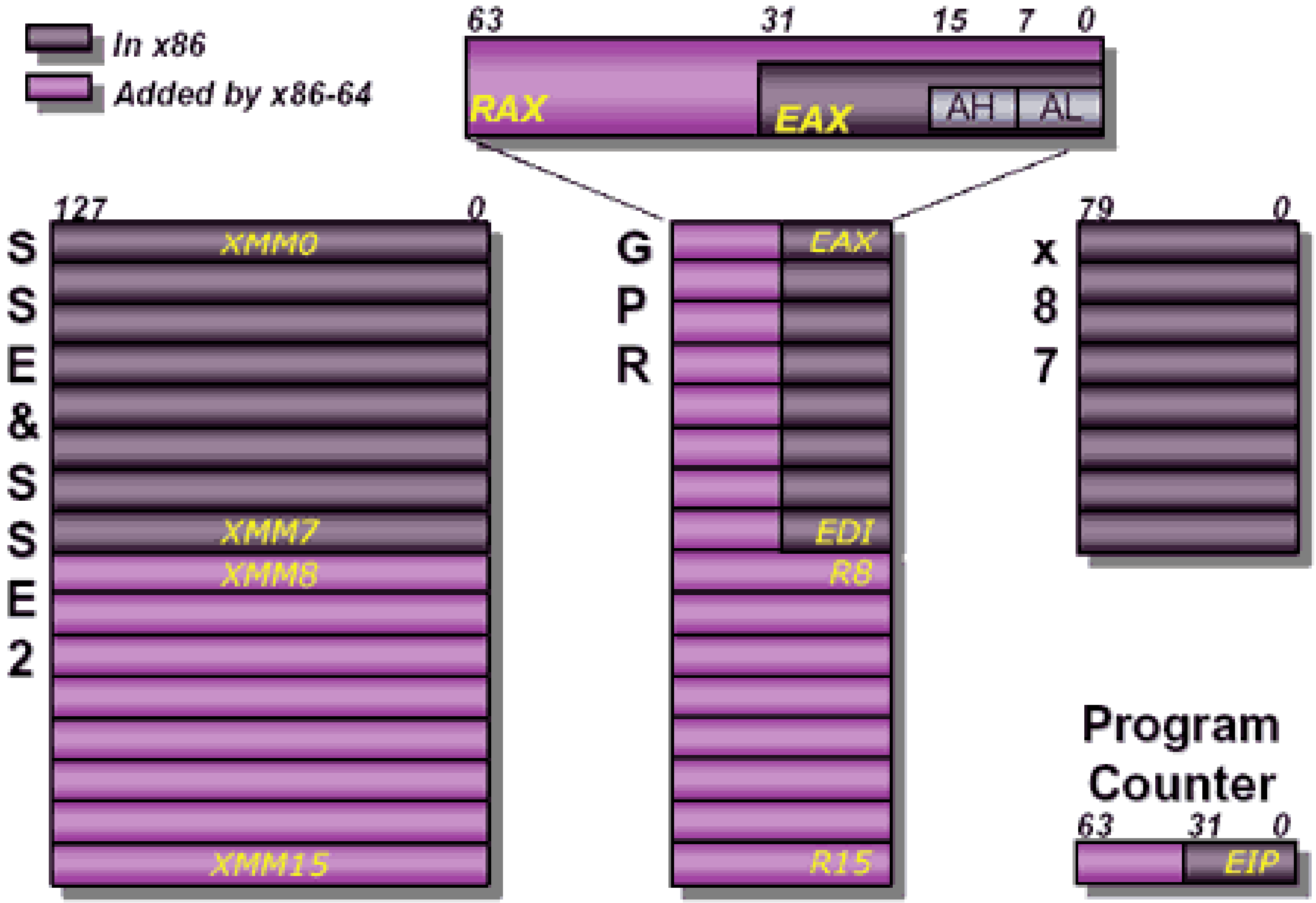
- Carry Flag
  - Set when unsigned arithmetic result is out of range
- Overflow Flag
  - Set when signed arithmetic result is out of range
- Sign Flag
  - Copy of sign bit, set when result is negative
- Zero Flag
  - Set when result is zero
- Auxiliary Carry Flag
  - Set when there is a carry from bit 3 to bit 4
- Parity Flag
  - Set when parity is even
  - Least-significant byte in result contains even number of 1s

Floating-Point, MMX, XMM Registers

- Floating-point unit performs high speed FP operations
- Eight 80-bit floating-point data registers
  - ST(0), ST(1), . . . , ST(7)
  - Arranged as a stack
  - Used for floating-point arithmetic
- Eight 64-bit MMX registers
  - Used with MMX instructions
- Eight 128-bit XMM registers
  - Used with SSE instructions



Registers in Intel Core Microarchitecture



Next ...

- Intel Microprocessors
- IA-32 Registers
- Instruction Execution Cycle
- IA-32 Memory Management

Fetch-Execute Cycle

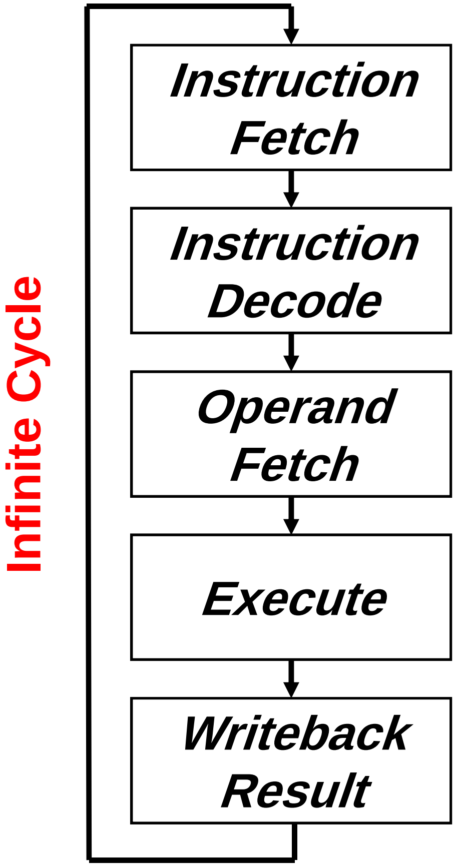
- Each machine language instruction is first fetched from the memory and stored in an **Instruction Register (IR).**
- The address of the instruction to be fetched is stored in a register called **Program Counter** or simply **PC**. In some computers this register is called the **Instruction Pointer** or **IP**.
- After the instruction is fetched, the **PC** (or **IP**) is incremented to point to the address of the next instruction.

Flash Movie

- The fetched instruction is decoded (to determine what needs to be done) and executed by the CPU.

Flash Movie

Instruction Execute Cycle



Obtain instruction from program storage

Determine required actions and instruction size

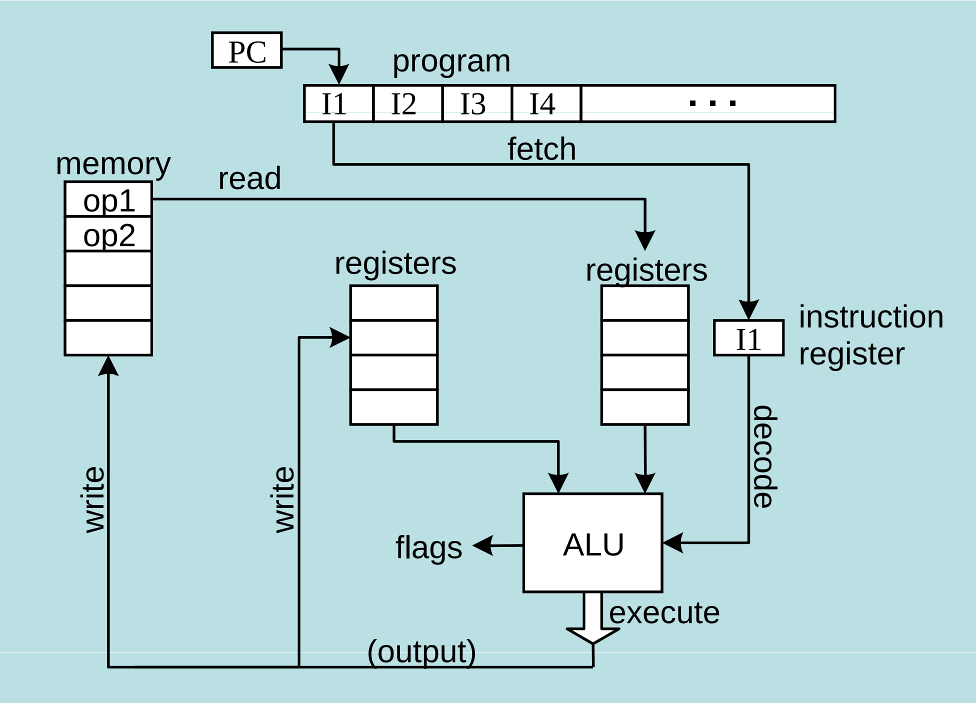
Locate and obtain operand data

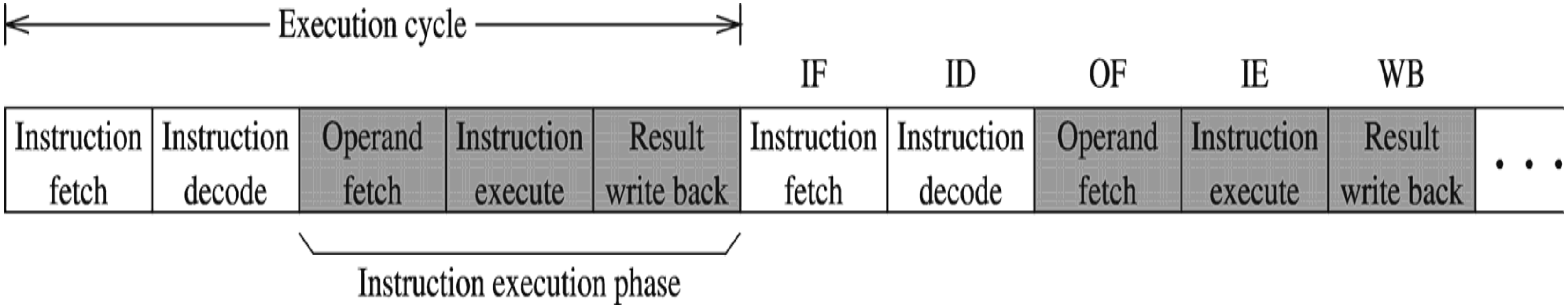
Compute result value and status

Deposit results in storage for later use

Instruction Execution Cycle – cont'd

- ❖ Instruction Fetch
- ❖ Instruction Decode
- ❖ Operand Fetch
- ❖ Execute
- ❖ Result Writeback





Pipelined Execution

- ❖ Instruction execution can be divided into stages
- ❖ Pipelining makes it possible to start an instruction before completing the execution of previous one

Stages

Cycles	S1	S2	S3	S4	S5	S6
1	**I-1**					
2		**I-1**				
3			**I-1**			
4				**I-1**		
5					**I-1**	
6						**I-1**
7	**I-2**					
8		**I-2**				
9			**I-2**			
10				**I-2**		
11					**I-2**	
12						**I-2**

Non-pipelined execution

Wasted clock cycles

For *k* stages and *n* instructions, the number of required cycles is: *k* + *n* – 1

Stages

Cycles	S1	S2	S3	S4	S5	S6
1	I-1					
2	I-2	I-1				
3		I-2	I-1			
4			I-2	I-1		
5				I-2	I-1	
6					I-2	I-1
7						I-2

Pipelined Execution

Wasted Cycles (pipelined)

- When one of the stages requires two or more clock cycles to complete, clock cycles are again wasted
  - Assume that stage S4 is the execute stage
  - Assume also that S4 requires 2 clock cycles to complete
  - As more instructions enter the pipeline, wasted cycles occur
  - For *k* stages, where one stage requires 2 cycles, *n* instructions require $k + 2n - 1$ cycles

Stages

exe

Cycles	S1	S2	S3	S4	S5	S6
1	I-1					
2	I-2	I-1				
3	I-3	I-2	I-1			
4		I-3	I-2	I-1		
5			I-3	I-1	▨	
6				I-2	I-1	▨
7				I-2	▨	I-1
8				I-3	I-2	▨
9				I-3	▨	I-2
10					I-3	▨
11						I-3

Superscalar Architecture

- ❖ A superscalar processor has multiple execution pipelines
- ❖ The Pentium processor has two execution pipelines
  - ✧ Called U and V pipes
- ❖ In the following, stage S4 has 2 pipelines
  - ✧ Each pipeline still requires 2 cycles
  - ✧ Second pipeline eliminates wasted cycles
  - ✧ For *k* stages and *n* instructions, number of cycles = *k* + *n*

Stages

				S4			
Cycles	S1	S2	S3	u	v	S5	S6
1	I-1						
2	I-2	I-1					
3	I-3	I-2	I-1				
4	I-4	I-3	I-2	I-1			
5		I-4	I-3	I-1	I-2		
6			I-4	I-3	I-2	I-1	
7				I-3	I-4	I-2	I-1
8					I-4	I-3	I-2
9						I-4	I-3
10							I-4

Next ...

- Intel Microprocessors
- IA-32 Registers
- Instruction Execution Cycle
- IA-32 Memory Management

Modes of Operation

- Real-Address mode (original mode provided by 8086)
  - Only 1 MB of memory can be addressed, from 0 to FFFFF (hex)
  - Programs can access any part of main memory
  - MS-DOS runs in real-address mode
- Protected mode
  - Each program can address a maximum of 4 GB of memory
  - The operating system assigns memory to each running program
  - Programs are prevented from accessing each other’s memory
  - Native mode used by Windows NT, 2000, XP, and Linux
- Virtual 8086 mode
  - Processor runs in protected mode, and creates a virtual 8086 machine with 1 MB of address space for each running program

Memory Segmentation

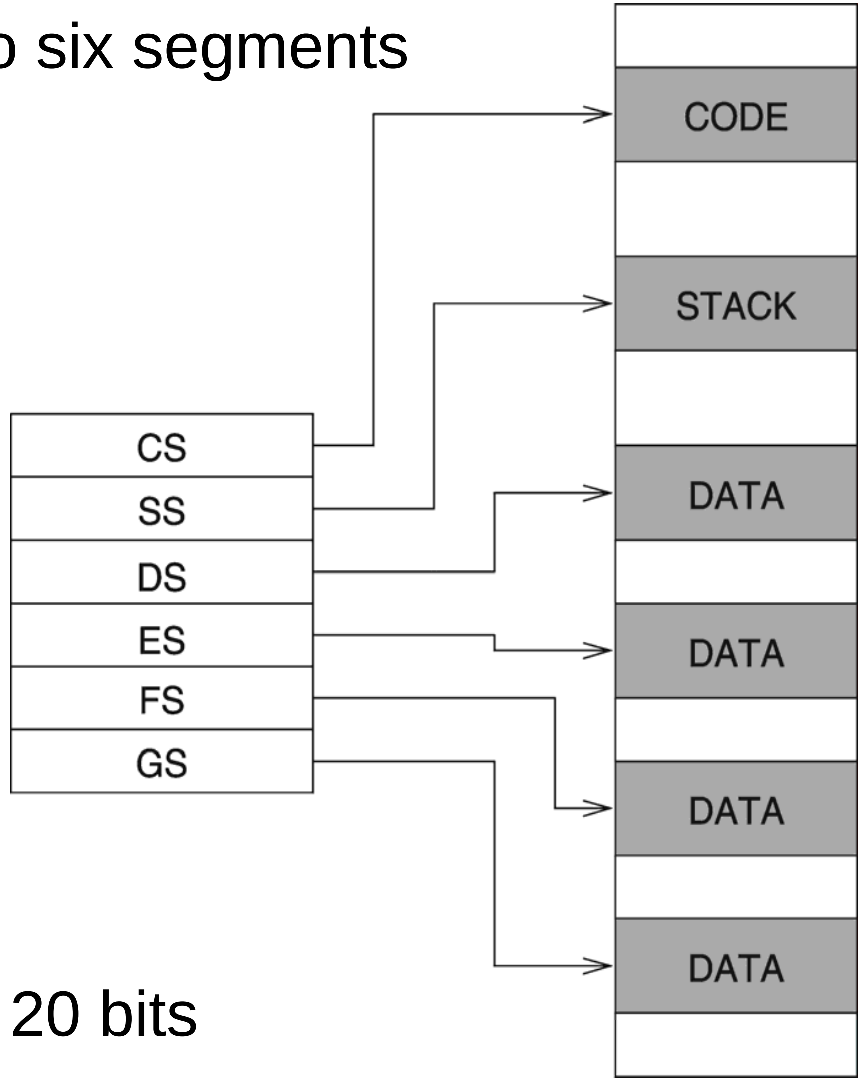
- Memory segmentation is necessary since the 20-bits memory addresses cannot fit in the 16-bits CPU registers
- Since x86 registers are 16-bits wide, a memory segment is made of 2^{16} consecutive words (i.e. 64K words)
- Each segment has a number identifier that is also a 16-bit number (i.e. we have segments numbered from 0 to 64K)
- A memory location within a memory segment is referenced by specifying its offset from the start of the segment. Hence the first word in a segment has an offset of 0 while the last one has an offset of FFFFh
- To reference a memory location its logical address has to be specified. The logical address is written as:
  - Segment number:offset
- For example, A43F:3487h means offset 3487h within segment A43Fh.

Program Segments

- Machine language programs usually have 3 different parts stored in different memory segments:
  - Instructions: This is the code part and is stored in the code segment
  - Data: This is the data part which is manipulated by the code and is stored in the data segment
  - Stack: The stack is a special memory buffer organized as Last-In-First-Out (LIFO) structure used by the CPU to implement procedure calls and as a temporary holding area for addresses and data. This data structure is stored in the stack segment
- The segment numbers for the code segment, the data segment, and the stack segment are stored in the segment registers CS, DS, and SS, respectively.
- Program segments do not need to occupy the whole 64K locations in a segment

Real Address Mode

- A program can access up to six segments at any time
  - Code segment
  - Stack segment
  - Data segment
  - Extra segments (up to 3)
- Each segment is 64 KB
- Logical address
  - Segment = 16 bits
  - Offset = 16 bits
- Linear (physical) address = 20 bits



Logical to Linear Address Translation

Linear address = Segment × 10 (hex) + Offset

Example:

segment = A1F0 (hex)

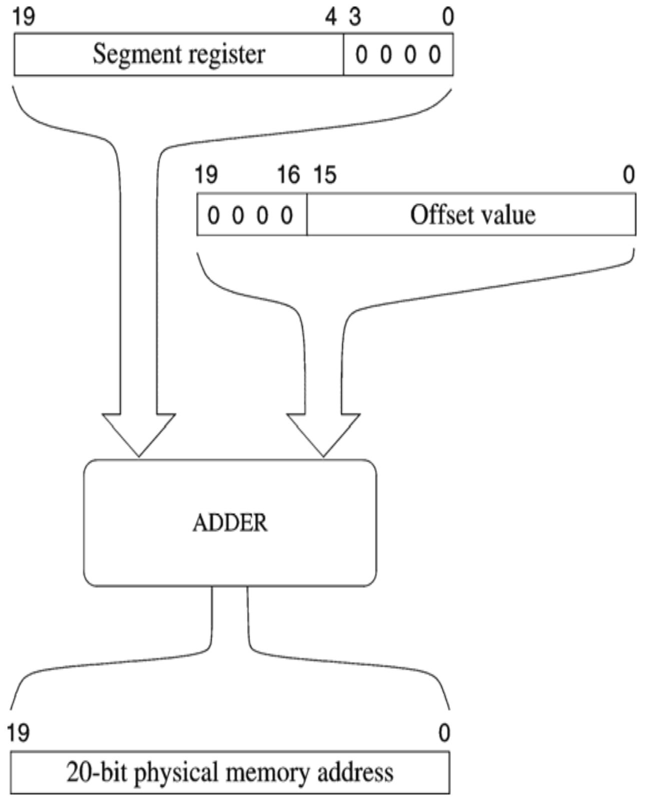
offset = 04C0 (hex)

logical address = A1F0:04C0 (hex)

what is the linear address?

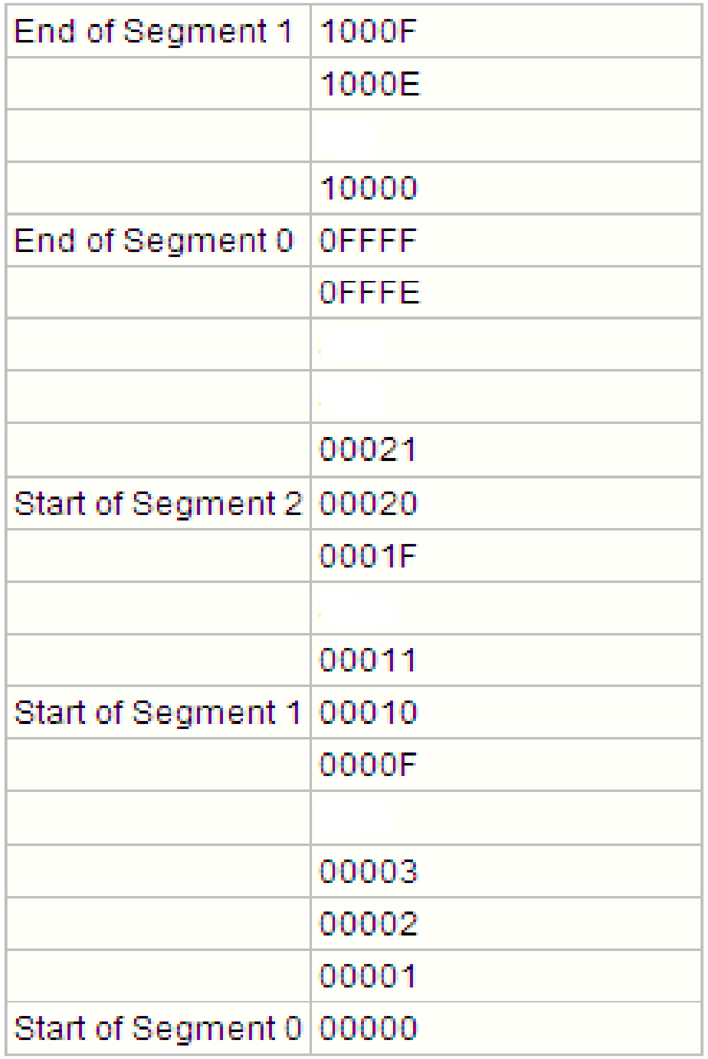
Solution:

```
  A1F00  (add 0 to segment in hex)
+ 04C0   (offset in hex)
  A23C0  (20-bit linear address in hex)
```



Segment Overlap

- ❖ There is a lot of overlapping between segments in the main memory.
- ❖ A new segment starts every 10**h** locations (i.e. every 16 locations).
- ❖ Starting address of a segment always has a 0**h** LSD.
- ❖ Due to segments overlapping logical addresses are not unique .



End of Segment 1	1000F
	1000E
	10000
End of Segment 0	0FFFF
	0FFFE
	00021
Start of Segment 2	00020
	0001F
	00011
Start of Segment 1	00010
	0000F
	00003
	00002
	00001
Start of Segment 0	00000

Your turn . . .

What linear address corresponds to logical address 028F:0030?

Solution: 028F0 + 0030 = 02920 (hex)

Always use hexadecimal notation for addresses

What logical address corresponds to the linear address 28F30h?

Many different segment:offset (logical) addresses can produce the same linear address 28F30h. Examples:

28F3:0000, 28F2:0010, 28F0:0030, 28B0:0430, . . .

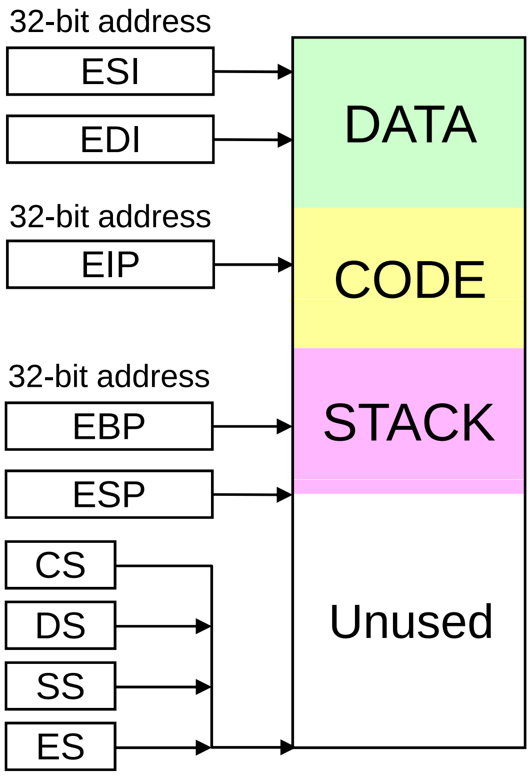
Flat Memory Model

- ❖ Modern operating systems turn segmentation off
- ❖ Each program uses one 32-bit linear address space
  - ✧ Up to 2^{32} = 4 GB of memory can be addressed
  - ✧ Segment registers are defined by the operating system
  - ✧ All segments are mapped to the same linear address space
- ❖ In assembly language, we use .MODEL flat directive
  - ✧ To indicate the Flat memory model
- ❖ A linear address is also called a virtual address
  - ✧ Operating system maps virtual address onto physical addresses
  - ✧ Using a technique called paging

Programmer View of Flat Memory

- Same base address for all segments
  - All segments are mapped to the same linear address space
- EIP Register
  - Points at next instruction
- ESI and EDI Registers
  - Contain data addresses
  - Used also to index arrays
- ESP and EBP Registers
  - ESP points at top of stack
  - EBP is used to address parameters and variables on the stack

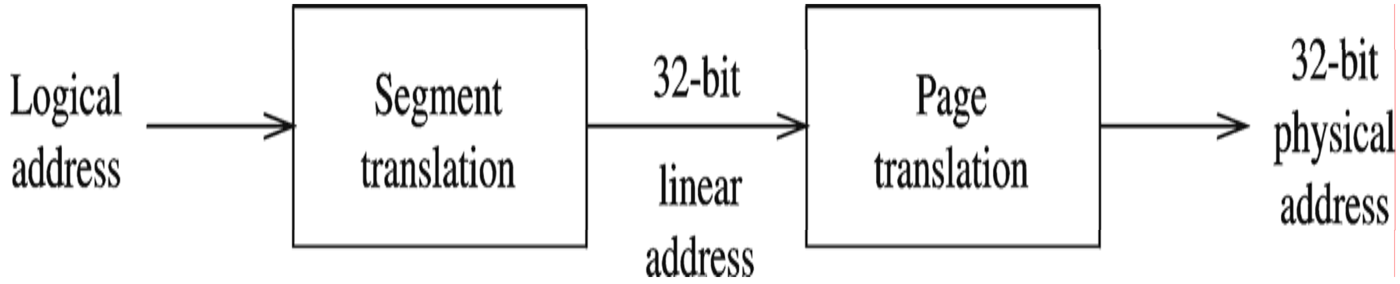
Linear address space of a program (up to 4 GB)



base address = 0
for all segments

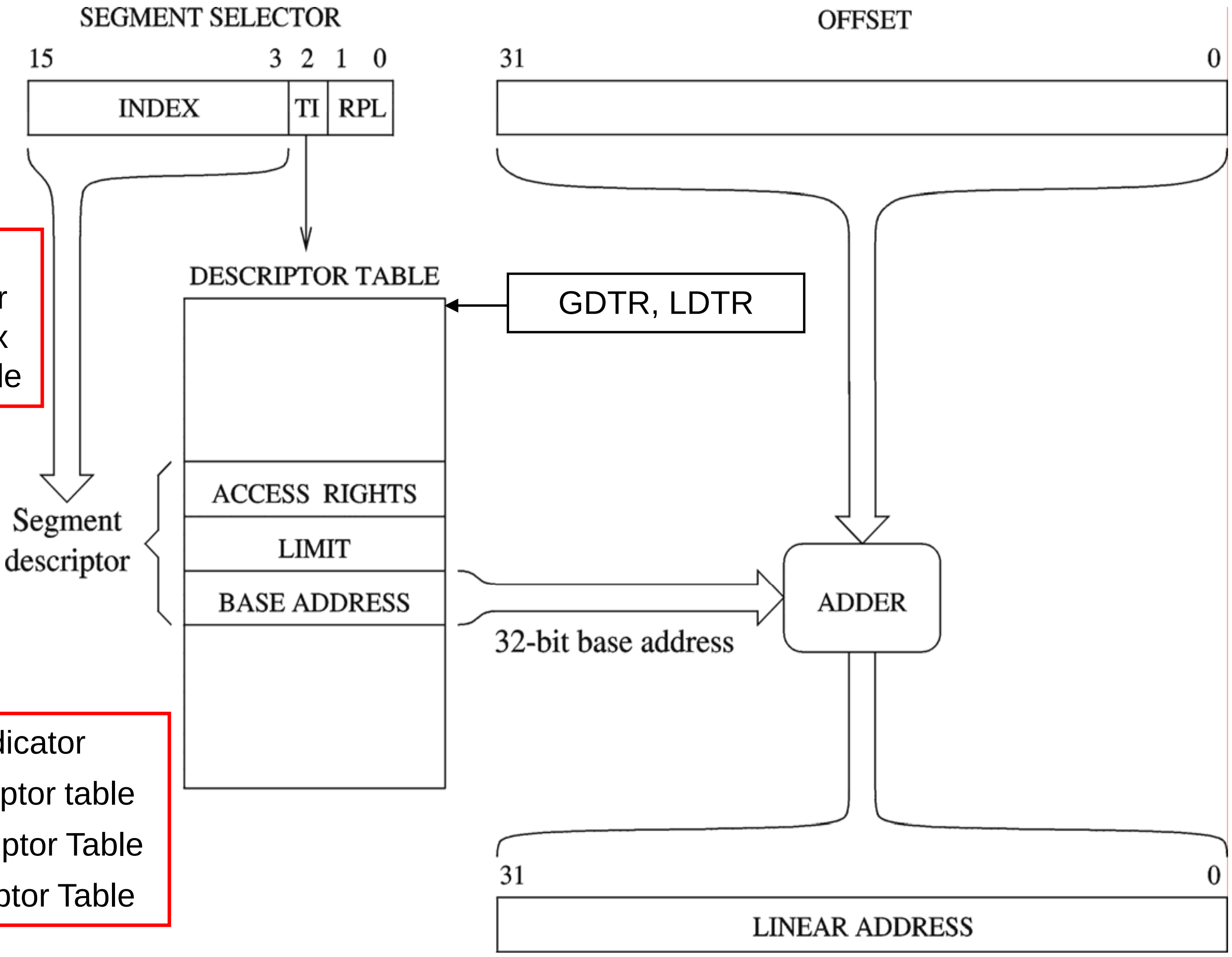
Protected Mode Architecture

- Logical address consists of
  - 16-bit segment selector (CS, SS, DS, ES, FS, GS)
  - 32-bit offset (EIP, ESP, EBP, ESI ,EDI, EAX, EBX, ECX, EDX)
- Segment unit translates logical address to linear address
  - Using a segment descriptor table
  - Linear address is 32 bits (called also a virtual address)
- Paging unit translates linear address to physical address
  - Using a page directory and a page table



Logical address → Segment translation → 32-bit linear address → Page translation → 32-bit physical address

Logical to Linear Address Translation



Upper 13 bits of segment selector are used to index the descriptor table

TI = Table Indicator

Select the descriptor table

0 = Global Descriptor Table

1 = Local Descriptor Table

Segment Descriptor Tables

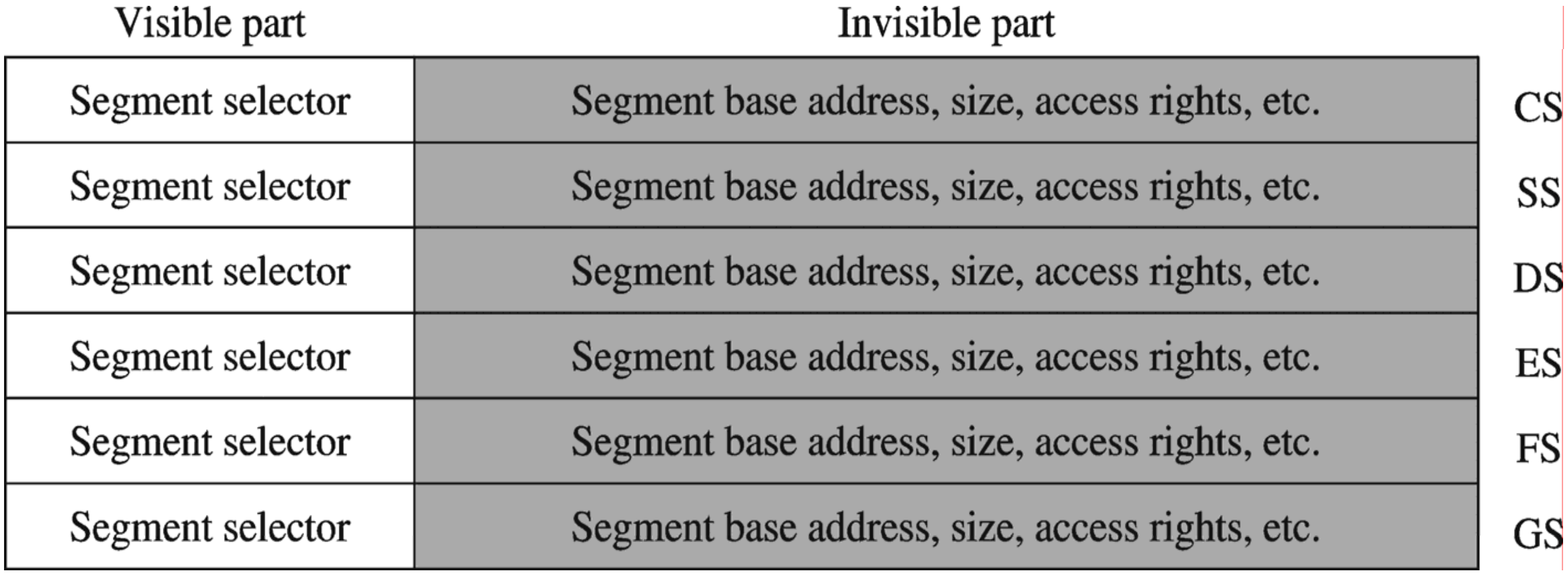
- Global descriptor table (GDT)
  - Only one GDT table is provided by the operating system
  - GDT table contains segment descriptors for all programs
  - Also used by the operating system itself
  - Table is initialized during boot up
  - GDT table address is stored in the GDTR register
  - Modern operating systems (Windows-XP) use one GDT table
- Local descriptor table (LDT)
  - Another choice is to have a unique LDT table for each program
  - LDT table contains segment descriptors for only one program
  - LDT table address is stored in the LDTR register

Segment Descriptor Details

- Base Address
  - 32-bit number that defines the starting location of the segment
  - 32-bit Base Address + 32-bit Offset = 32-bit Linear Address
- Segment Limit
  - 20-bit number that specifies the size of the segment
  - The size is specified either in bytes or multiple of 4 KB pages
  - Using 4 KB pages, segment size can range from 4 KB to 4 GB
- Access Rights
  - Whether the segment contains code or data
  - Whether the data can be read-only or read & written
  - Privilege level of the segment to protect its access

Segment Visible and Invisible Parts

- ❖ Visible part = 16-bit Segment Register
  - ✧ CS, SS, DS, ES, FS, and GS are visible to the programmer
- ❖ Invisible Part = Segment Descriptor (64 bits)
  - ✧ Automatically loaded from the descriptor table



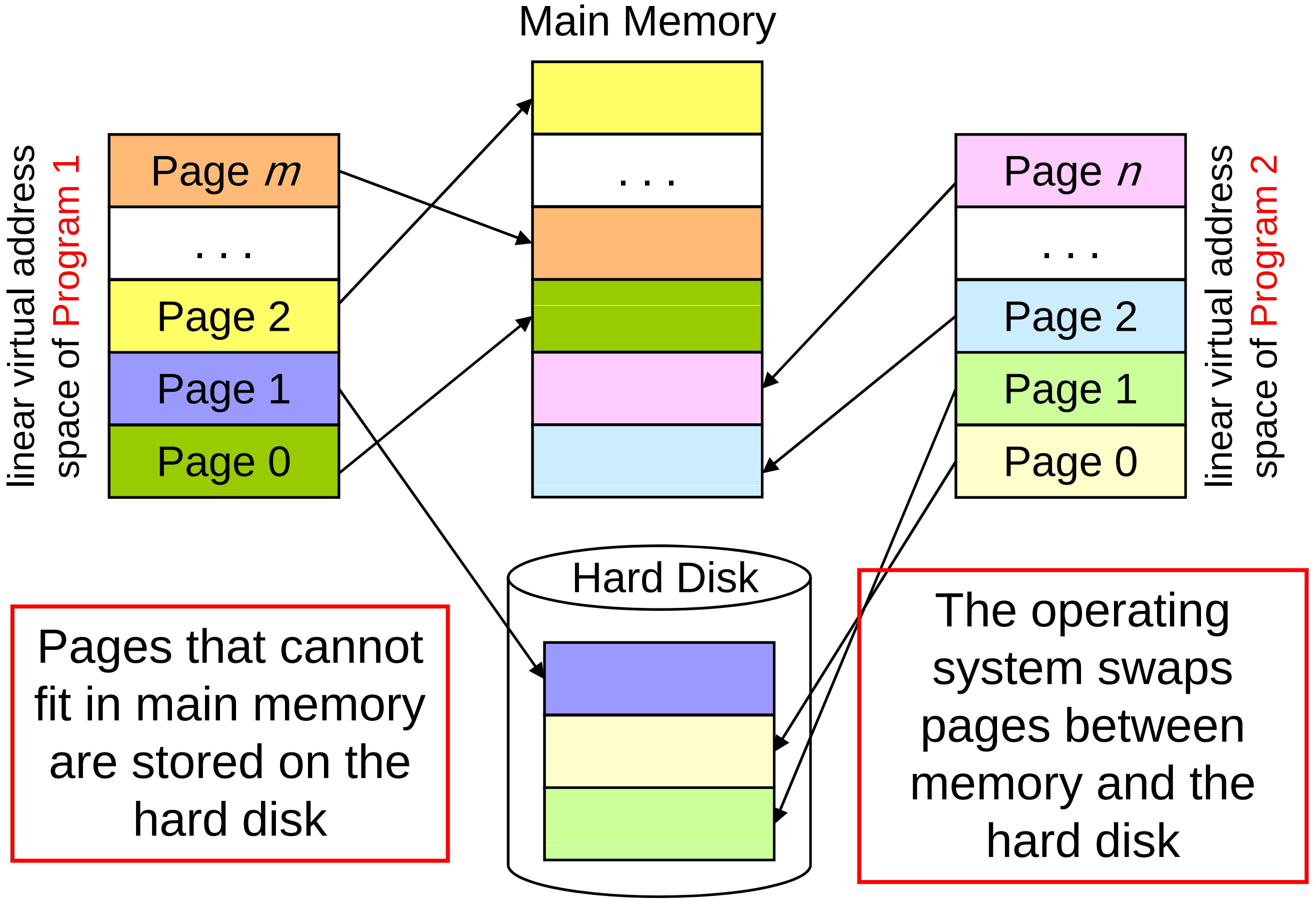
Visible part	Invisible part	
Segment selector	Segment base address, size, access rights, etc.	CS
Segment selector	Segment base address, size, access rights, etc.	SS
Segment selector	Segment base address, size, access rights, etc.	DS
Segment selector	Segment base address, size, access rights, etc.	ES
Segment selector	Segment base address, size, access rights, etc.	FS
Segment selector	Segment base address, size, access rights, etc.	GS

Paging

- Paging divides the linear address space into …
  - Fixed-sized blocks called pages, Intel IA-32 uses 4 KB pages
- Operating system allocates main memory for pages
  - Pages can be spread all over main memory
  - Pages in main memory can belong to different programs
  - If main memory is full then pages are stored on the hard disk
- OS has a Virtual Memory Manager (VMM)
  - Uses page tables to map the pages of each running program
  - Manages the loading and unloading of pages
- As a program is running, CPU does address translation
- Page fault: issued by CPU when page is not in memory

Paging – cont'd

The operating system uses page tables to map the pages in the linear virtual address space onto main memory



Each running program has its own page table

Pages that cannot fit in main memory are stored on the hard disk

The operating system swaps pages between memory and the hard disk

As a program is running, the processor translates the linear virtual addresses onto real memory (called also physical) addresses