EECE416 :Microcomputer Fundamentals and Design ("Microcomputer & Microprocessor")

X86 Assembly Programming

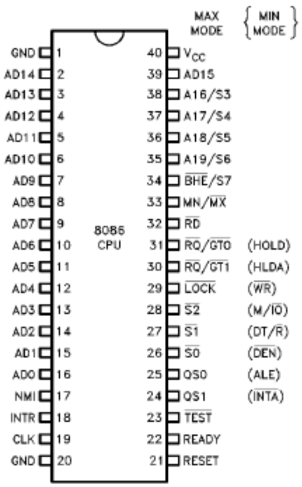
Dr. Charles Kim

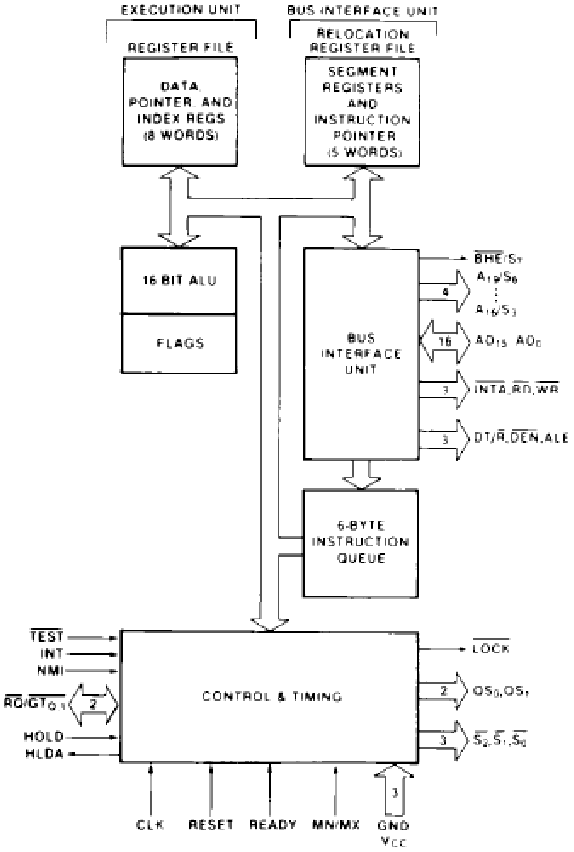
Department of Electrical and Computer Engineering

Howard University

x86 Architecture

- First x86 Family member: 8086 (→ 8088). 1978
  - Cf. 4004 → 8080 → 8085
- 8086
  - 16-bit registers, external data bus
  - 20-bit addressing (→ 1MB address space)
  - Segmentation (by 16-bit) : 64KB
    - 4 Segmentation registers hold 4*64KB =256KB
    - Upto 256KB can be addressed without switching between Segments





x86 Architecture

- 80286
  - Protected Mode
    - Segment register contents as selector or pointer → discriptor table
  - 24-bit base address → 16MB memory size
  - Application protection
- 386
  - 32-bit registers for operands and addressing(→4GB space)
  - Lower half of 32 bits is equivalent to 16 bits of earlier generations [Backward (upward) compatibility with 16-bit registers]
  - Some new instructions was added (like bit manipulation)
  - Max 4GB segmentation of physical space
  - New Parallel Processing Stages introduced: Bus Interface Unit, Code Prefetch Unit, Instruction Decode Unit, Execution Unit, Segment Unit (logical address → Linear address), Paging Unit (Linear address → physical address)

Memory Organization and Memory Models

- Physical Memory
  - The memory -- the processor addresses on its bus
  - Organized as a sequence of 8-bit bytes
  - Each byte is assigned a unique address, a physical address
  - Range: 36 address lines → 64 GB
- Flat memory model (a single continuous address space) → linear address space
  - Code, data, stack are all contained in this address space
  - Byte accessible

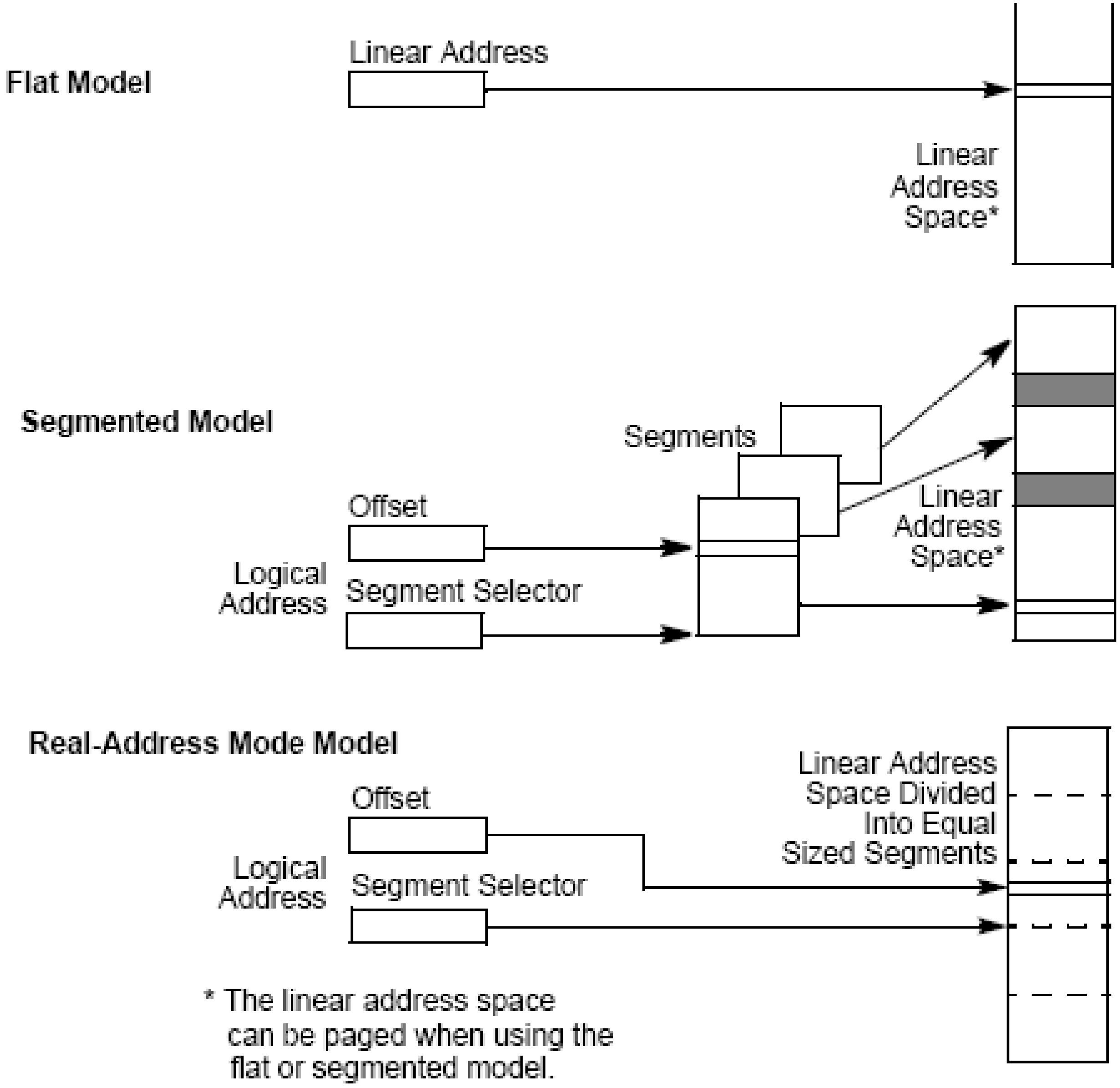
.386
.MODEL FLAT

.STACK 4096

.DATA

- Segmented memory model (memory grouped into independent address spaces, segments)
  - Code, data, stacks are contained in separate segments
  - Logical address (segment selector and an offset) to address
  - Up to 16K segments of different sizes (max 64 GB)
  - Why segmentation:
    - Increase reliability of programs and systems – avoid overwriting
- Real-Address Mode (Intel 8086 model)

Memory Management Model



* The linear address space can be paged when using the flat or segmented model.

Modes of Operation

- Operating mode determines which instructions and architectural features are accessible - 3 Operating modes
- Protected mode
  - Native State of Processor
  - All instructions and architectural features are available – highest performance and capability
  - Recommended mode
- Real-address mode
  - Programming environment of Intel 8086
  - Processor is in this mode following power-up or reset
- System management mode (SMM)
  - Power management and system security
  - Enters SMM by SMM interrupt (SMI) or APIC (Advanced Programmable Interrupt Controller)

Overview of Basic Execution

- Set of resources for Executing instructions and for Storing code, data, and state information
- Resources:
  - Address space: 36 address lines
  - 8 General data registers
  - 6 Segment registers
  - Status and control registers
- Holding the following items (for all):
  - Operands for logical and arithmetic operations
  - Operands for address calculations
  - Memory pointers
- EAX (accumulator for operands and results data)
- EBX (Pointer to data in Segment)
- ECX (Counter)
- EDX (for I/O pointer)

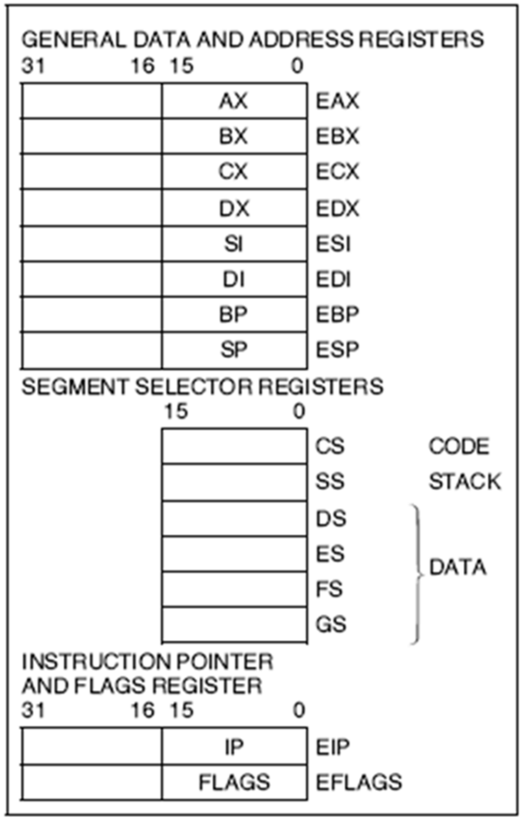

GENERAL DATA AND ADDRESS REGISTERS
31
16
15
0
AX
EAX
BX
EBX
CX
ECX
DX
EDX
SI
ESI
DI
EDI
BP
EBP
SP
ESP
SEGMENT SELECTOR REGISTERS
15
0
CS
CODE
SS
STACK
DS
ES
DATA
FS
GS
INSTRUCTION POINTER AND FLAGS REGISTER
31
16
15
0
IP
EIP
FLAGS
EFLAGS

Figure 2-1. Intel386™ DX Base Architecture Registers

General-Purpose Data Registers

- EBP (base pointer to data on the stack in DS segment)
- ESI (Source pointer)
- EDI (data pointer) for string instructions
- ESP (Stack pointer)holds the stack pointer (restricted use)
- ESP points to the top item on the stack and the EBP points to the "previous" top of the stack before the function was called.

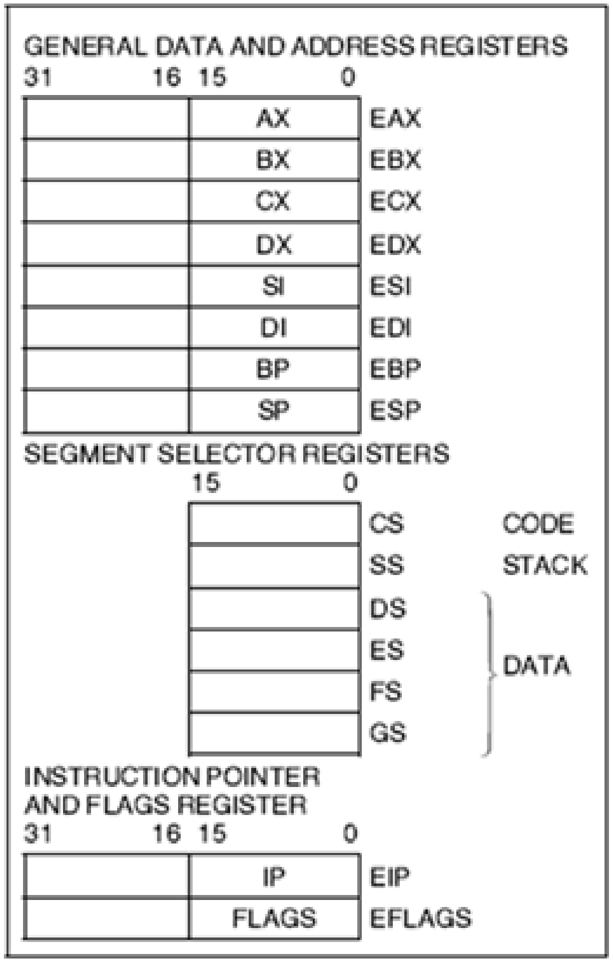
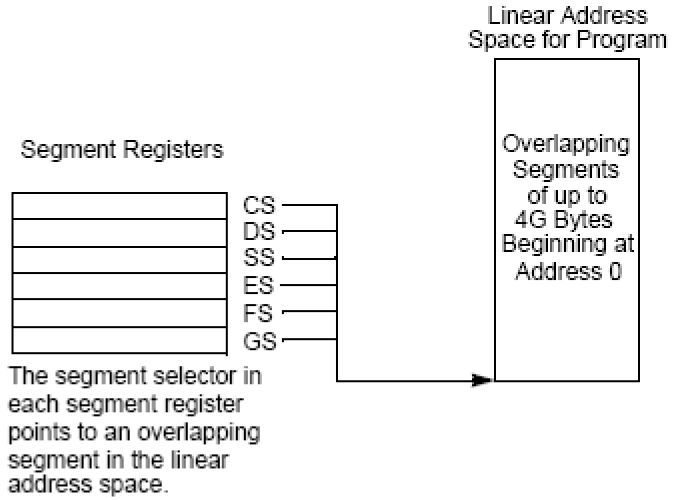

GENERAL DATA AND ADDRESS REGISTERS
31
16
15
0
AX
EAX
BX
EBX
CX
ECX
DX
EDX
SI
ESI
DI
EDI
BP
EBP
SP
ESP
SEGMENT SELECTOR REGISTERS
15
0
CS
CODE
SS
STACK
DS
ES
FS
GS
DATA
INSTRUCTION POINTER AND FLAGS REGISTER
31
16
15
0
IP
EIP
FLAGS
EFLAGS

Figure 2-1. Intel386™ DX Base Architecture Registers

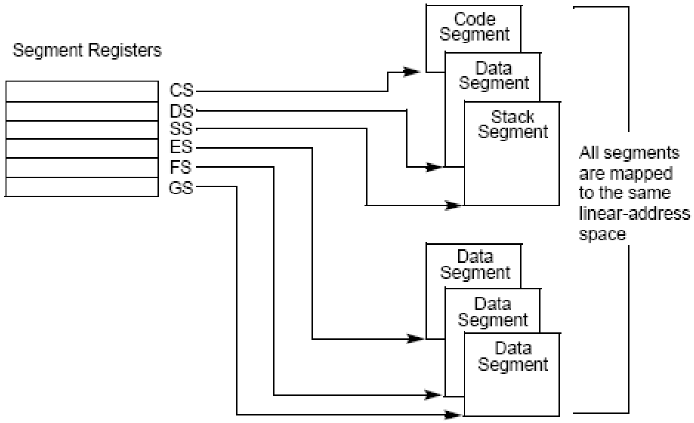
Segment registers - Revisit

- Hold 16-bit segment selectors
- Segment selector: a special pointer that identifies a segment in memory
- Associated with 3 types of storage:
  - Code (instructions are stored): CS + EIP (offset)
  - Data : DS, ES, FS, and GS
  - Stack (Procedure Stack is stored): SS
- Segment selector ← by Assembler directiv
- **Flat (un-segmented) Memory Model Case**:
  - Overlapped and starts at 0: Code Seg and Data Seg and Stack Seg
- **Segmented Memory Model Case:**
  - Loaded with different segments, pointing different segments
  - Program can access 6 different segments
  - To access a segment not pointed by the Segment registers? Load a segment selector to a segment register first.


Linear Address Space for Program
Segment Registers
CS
DS
SS
ES
FS
GS
Overlapping Segments of up to 4G Bytes Beginning at Address 0

The segment selector in each segment register points to an overlapping segment in the linear address space.

Use of Segment Registers for Flat Memory Model



Use of Segment Registers in Segmented Memory Model

EFLAG Register

- 32-bit register
  - Initial state: 00000002H
  - Contains a group of **status flags**, a **control flag**, and a group of **system flags**

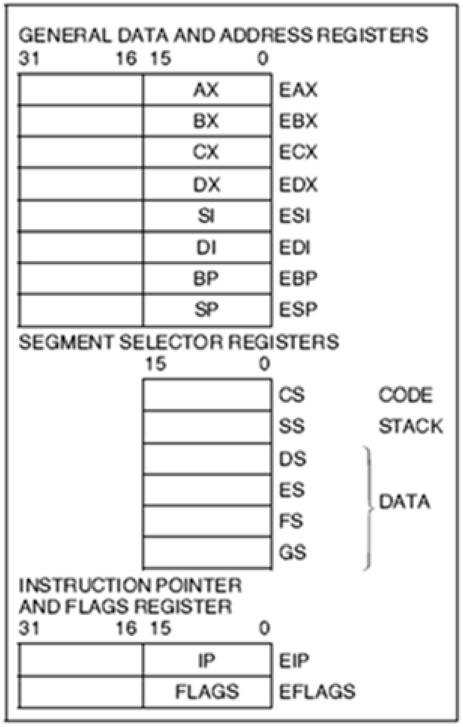
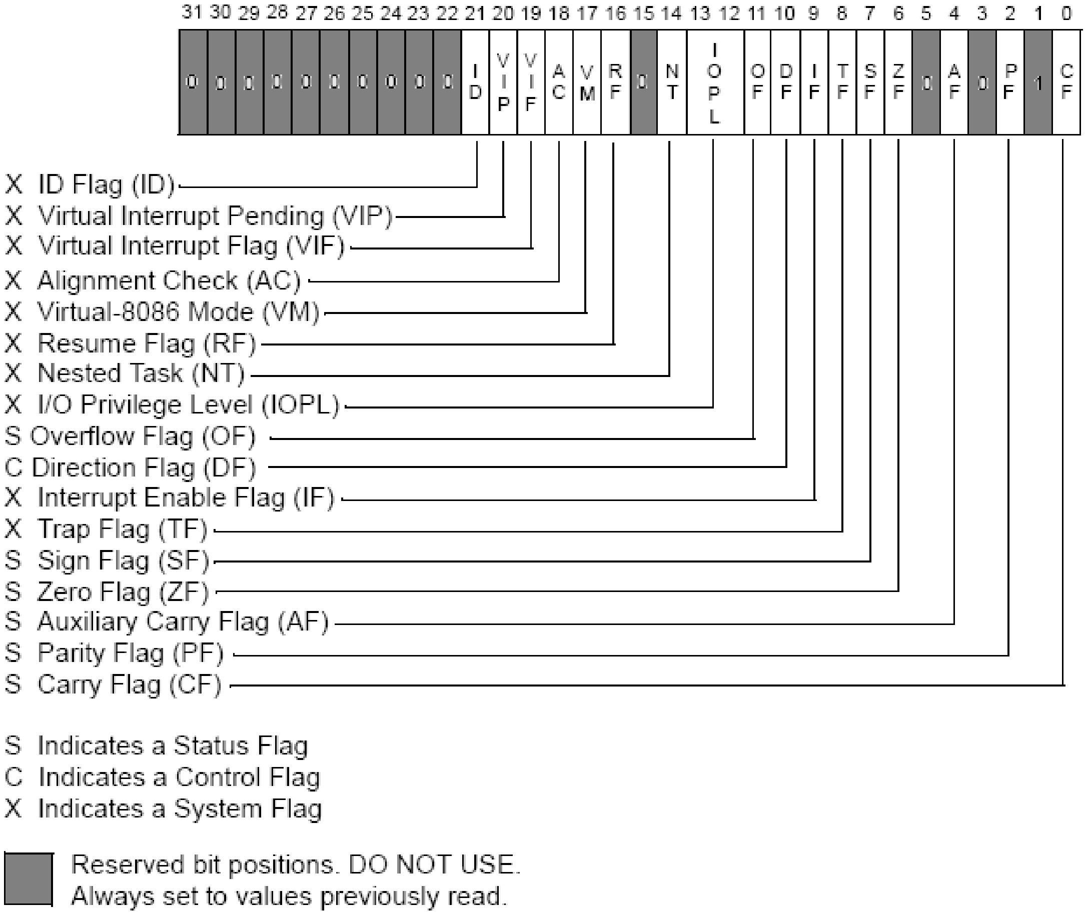


Figure 2-1. Intel386™ DX Base Architecture Registers

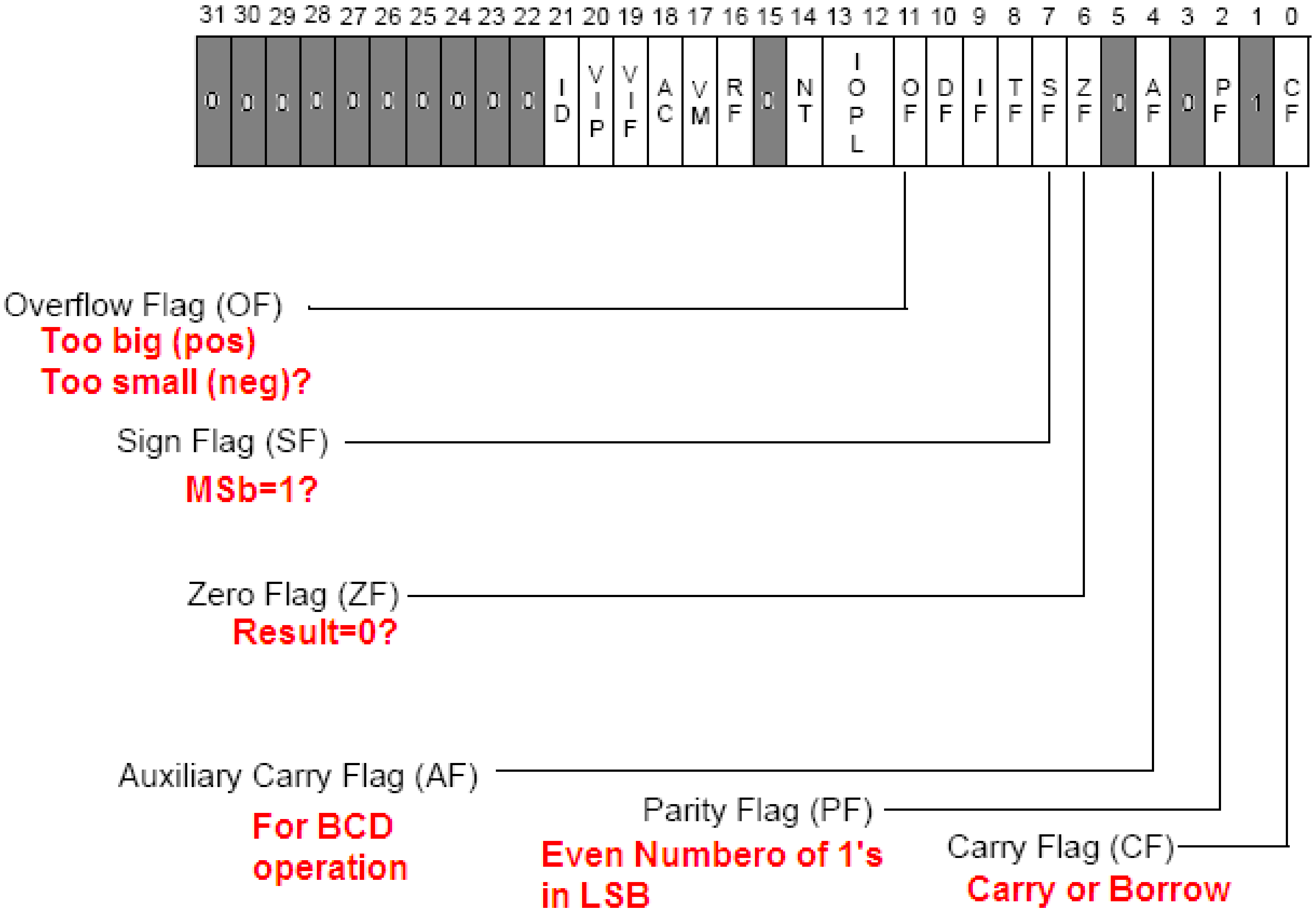


X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag

▨ Reserved bit positions. DO NOT USE.
Always set to values previously read.

Status Flags



Overflow Flag (OF)

Too big (pos)
Too small (neg)?

Sign Flag (SF)

MSb=1?

Zero Flag (ZF)

Result=0?

Auxiliary Carry Flag (AF)

For BCD operation

Parity Flag (PF)

Even Numbero of 1's in LSB

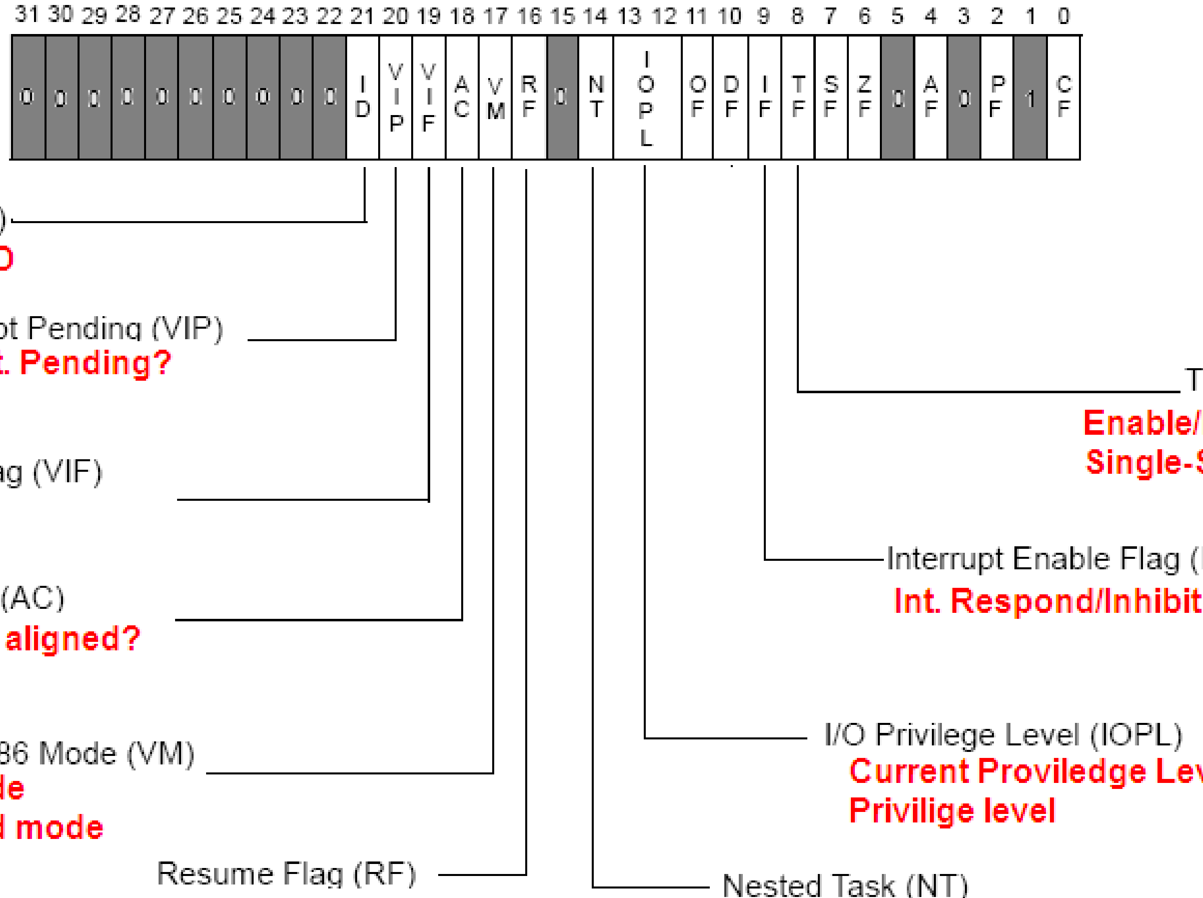
Carry Flag (CF)

Carry or Borrow

Control Flag (DF)

- DF (Direction Flag)
  - The direction flag controls the string instructions (MOVS, CMPS, SCAS, LODS, and STOS).
  - DF=1 → string instructions to auto-decrement (that is, to process strings from high addresses to low addresses).
  - DF=0 → string instructions to auto-increment (process strings from low addresses to high addresses).
  - STD → Set DF flag
  - CLD → Clear DF flag

System Flags



ID Flag (ID)
CPUID

Virtual Interrupt Pending (VIP)
Int. Pending?

Virtual Interrupt Flag (VIF)

Alignment Check (AC)
Memory reference aligned?

Virtual-8086 Mode (VM)
1: 8086 mode
0: Protected mode

Resume Flag (RF)
Response to debug exception

Trap Flag (TF)
Enable/Disable
Single-Step Debug Mode

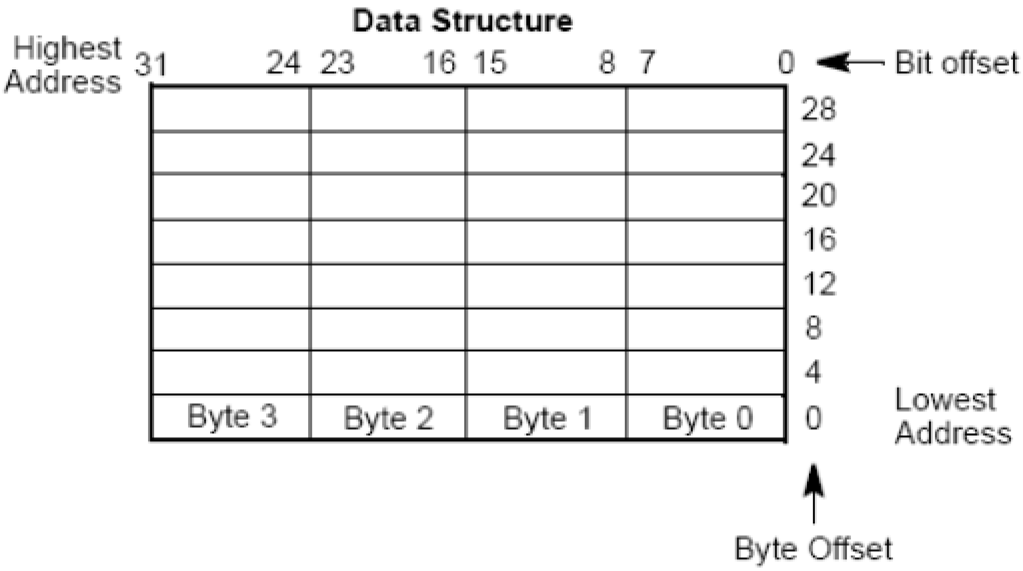
Interrupt Enable Flag (IF)
Int. Respond/Inhibit

I/O Privilege Level (IOPL)
Current Proviledge Level <= I/O
Privilige level

Nested Task (NT)
Task linked to another task?

Notational Conventions

- Bit and Byte Oder
  - Smaller address at the bottom of figure
  - Address increases toward top
  - Bit positions numbered from right to left
- Little-Endian Machine
  - the bytes of a word are numbered starting from the least significant byte



Conventions

⌘ Instruction Format

- Label: mnemonic argument1, argument2, argument3
- **Label:** Identifier (followed by a colon)
- **Mnemonic:** a reserved name for a class of instruction opcodes which have the same function
- **Operands (arguments**): The operands argument1, argument2, and argument3 are optional. There may be from zero to three operands, depending on the opcode. When present, they take the form of either literals or identifiers for data items. Operand identifiers are either reserved names of registers or are assumed to be assigned to data items declared in another part of the program.
- When two operands are present in an arithmetic or logical instruction, the right operand is the **source** and the left operand is the **destination.**
- Example: LOADREG: MOV EAX, SUBTOTAL
- label mnemonic dst src

Conventions

- Binary and Hexadecimal Numbers
  - Base 2 (binary) numbers are represented by a string of 1s and 0s, sometimes followed by the character B (for example, 1010B).
  - The “B” designation is only used in situations where confusion as to the type of number might arise.
  - Base 16 (hexadecimal) numbers are represented by a string of hexadecimal digits followed by the character H (for example, F82EH).
  - A hexadecimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Modes of Operation

- Operating mode determines which instructions and architectural features are accessible - 3 Operating modes
- Protected mode
  - Native State of Processor
  - All instructions and architectural features are available – highest performance and capability
  - Recommended mode
- Real-address mode
  - Programming environment of Intel 8086
  - Processor is in this mode following power-up or reset
- System management mode (SMM)
  - Power management and system security
  - Enters SMM by SMM interrupt (SMI) or APIC (Advanced Programmable Interrupt Controller)

.386
.MODEL FLAT
.STACK 4096
.DATA

386 Instruction Set

- 9 Operation Categories
  - Data Transfer
  - Arithmetic
  - Shift/Rotate
  - String Manipulation
  - Bit Manipulation
  - Control Transfer
  - High Level Language Support
  - Operating System Support
  - Processor Control
- Number of operands: 0, 1, 2, or 3

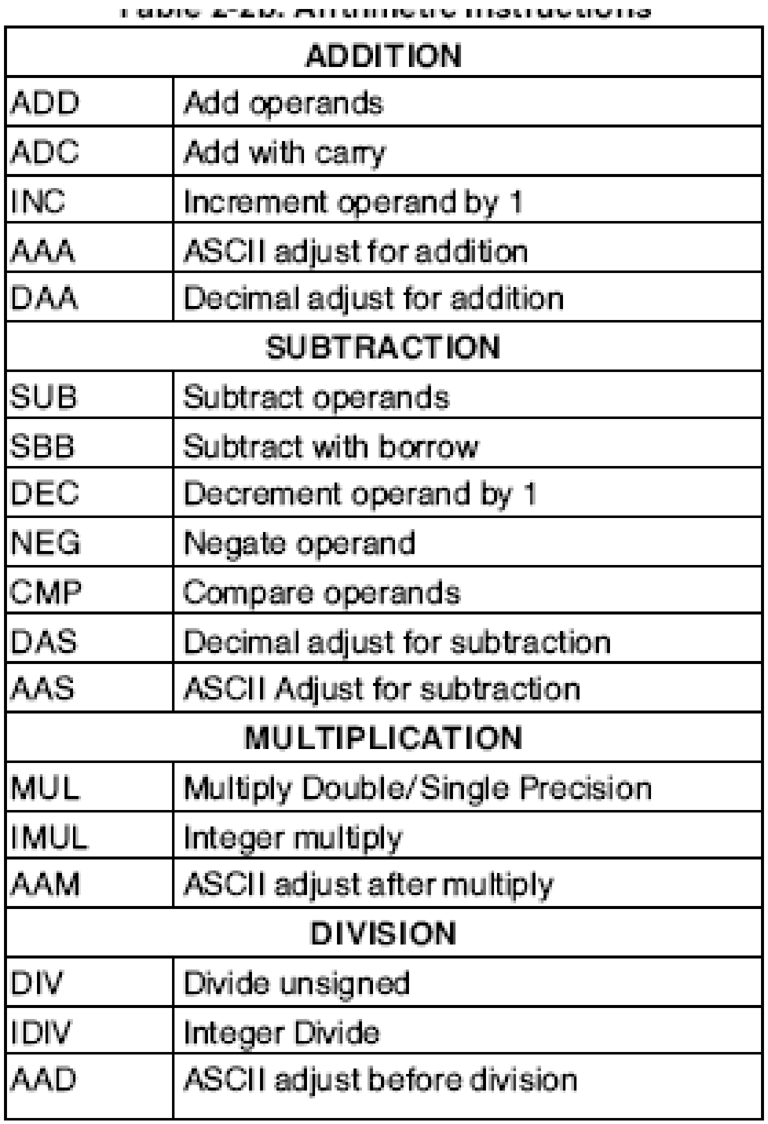


Table 2-2b. Arithmetic Instructions

ADDITION	
ADD	Add operands
ADC	Add with carry
INC	Increment operand by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract operands
SBB	Subtract with borrow
DEC	Decrement operand by 1
NEG	Negate operand
CMP	Compare operands
DAS	Decimal adjust for subtraction
AAS	ASCII Adjust for subtraction
MULTIPLICATION	
MUL	Multiply Double/Single Precision
IMUL	Integer multiply
AAM	ASCII adjust after multiply
DIVISION	
DIV	Divide unsigned
IDIV	Integer Divide
AAD	ASCII adjust before division