EECE416 :Microcomputer Fundamentals and Design

X86 Assembly Programming

Part 1

Dr. Charles Kim

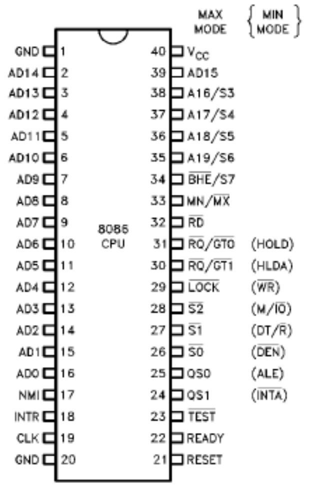
Department of Electrical and Computer Engineering

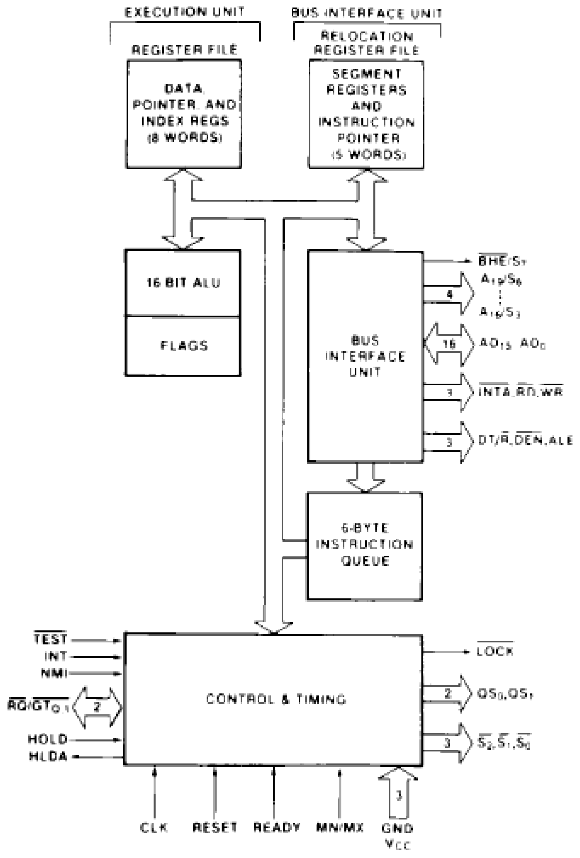
Howard University

www.MWFTR.com

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 : 64KB
    - 4 Segmentation registers hold 4*64KB =256KB





x86 Architecture

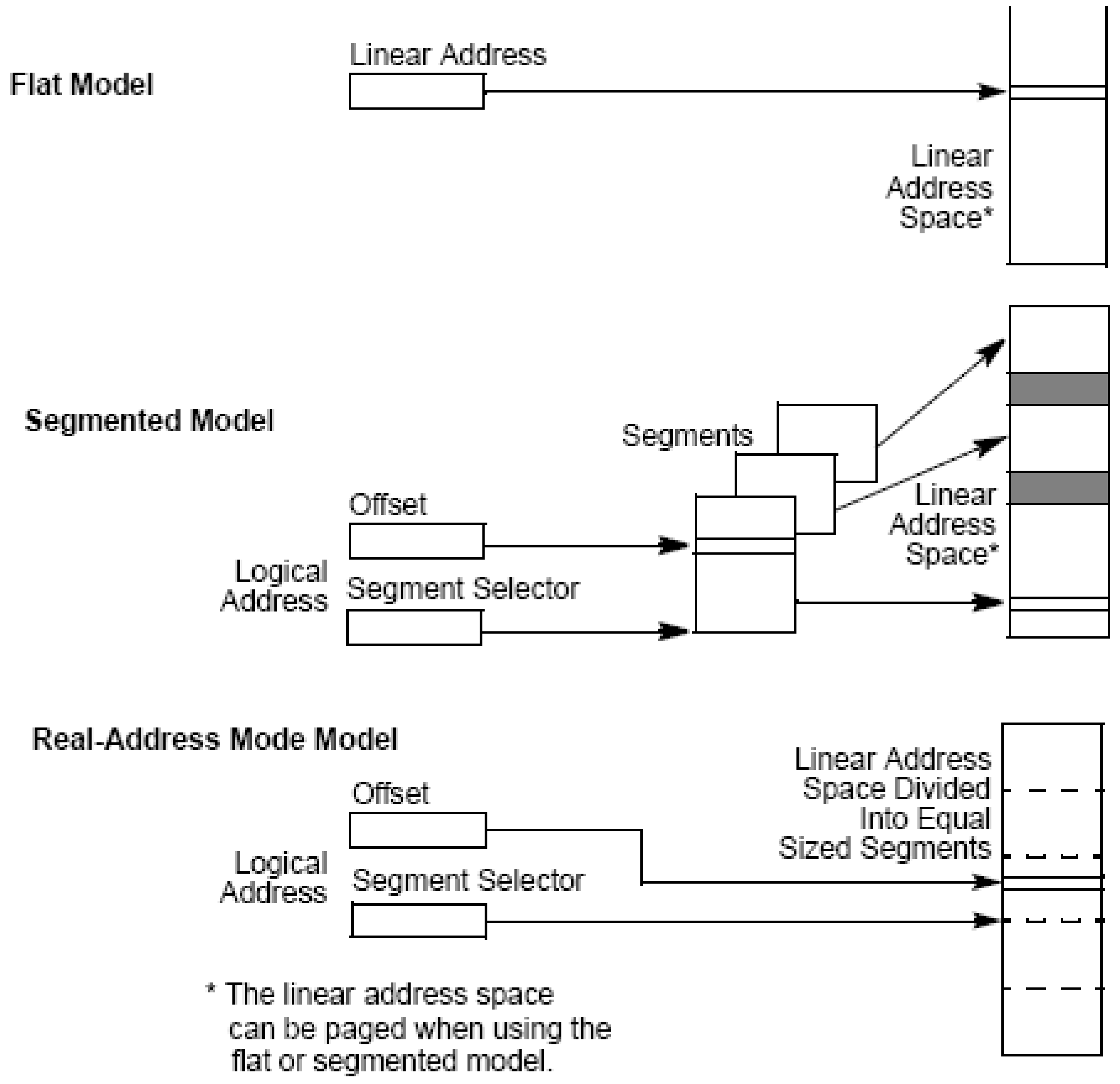
- 80286
  - “Protected Mode” first introduced
    - Segment register contents as selector or pointer
    - 24-bit base address → 16MB memory size
- 80386
  - 32-bit registers for operands and addressing(→4GB space)
  - Lower half of 32 bits is equivalent to 16 bits of earlier generations [Backward (upward) 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
- Real-address Mode (Intel 8086 model):
  - **fixed size** segments
  - Unlimited Direct Software access to all Memory, I/O, and peripherals
  - No hard-ware level memory protection
- 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
  - **different sizes**
  - Why segmentation:
    - Increase reliability of programs and systems – avoid overwriting, Memory Protection
- 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
```

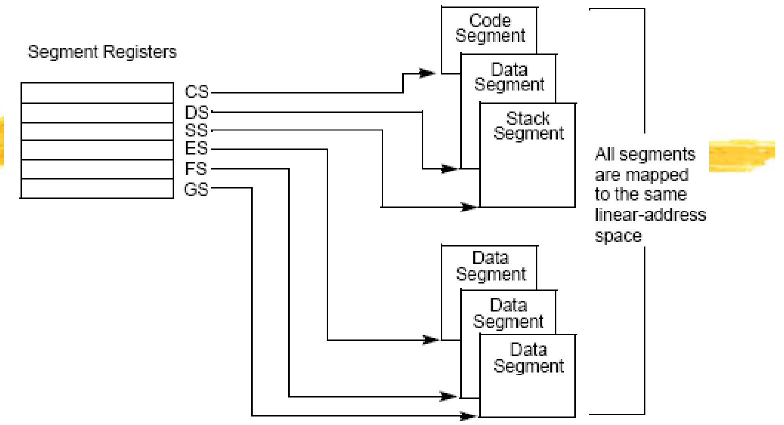
Memory Management Model



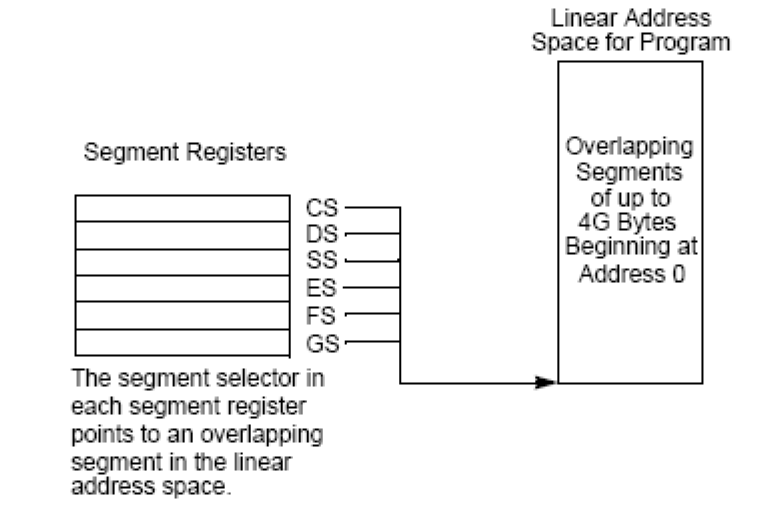
* The linear address space can be paged when using the flat or segmented model.

386 Segment registers

- 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 directive
- **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.
- **Flat (un-segmented) Memory Model Case**:
  - Overlapped and starts at 0: Code Seg and Data Seg and Stack Seg



Use of Segment Registers in Segmented Memory Model



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

Modes of Operation

- Operating mode determines which instructions and architectural features are accessible
- Protected mode (from i286)
  - 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 (for backward compatibility)
  - No memory level protection

Overview of Basic Execution

- Set of resources for **Executing instructions** and for **Storing code, data, and state information**
- Resources:
  - 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

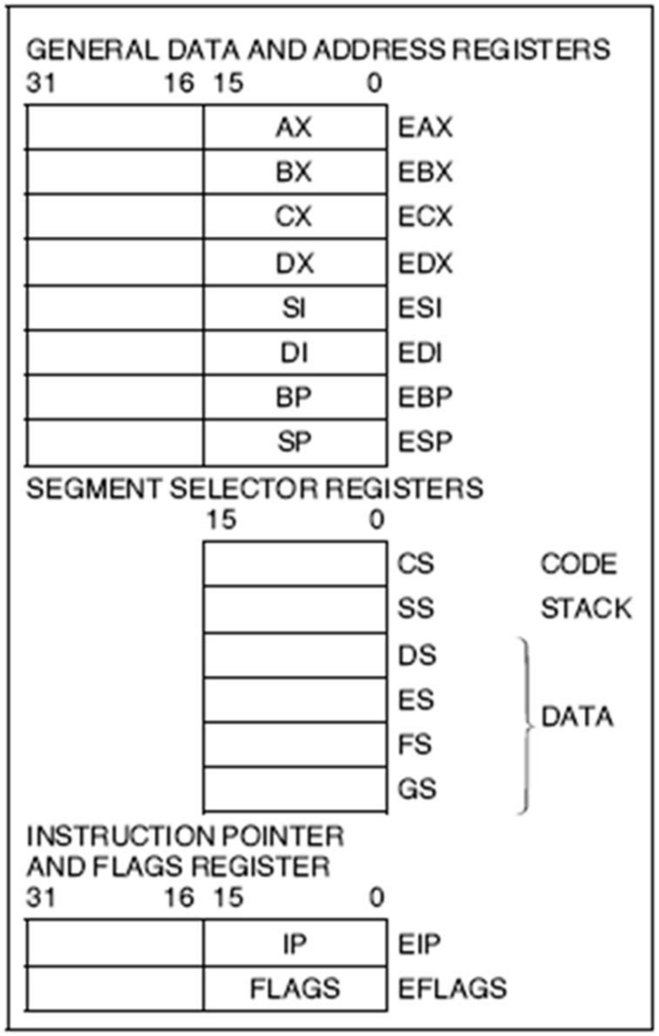
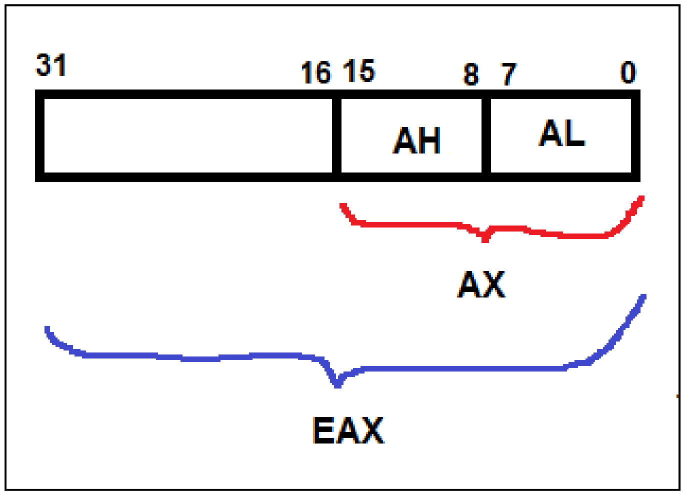


Figure 2-1. Intel386™ DX Base Architecture Registers



General-Purpose Data Registers

- Primaries
  - EAX (accumulator for operands and results data)
  - EBX (Pointer to data in Segment)
  - ECX (Counter)
  - EDX (for I/O pointer)
- Secondaries
  - 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.

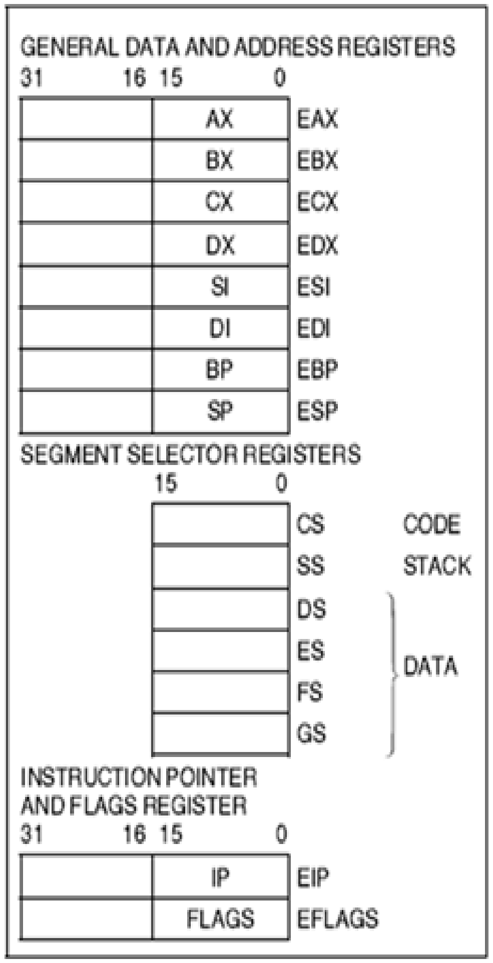


Figure 2-1. Intel386™ DX Base Architecture Registers

EFLAG Register

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

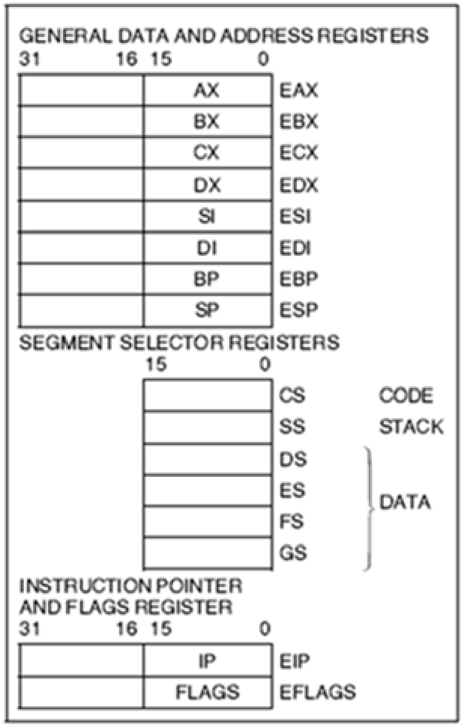
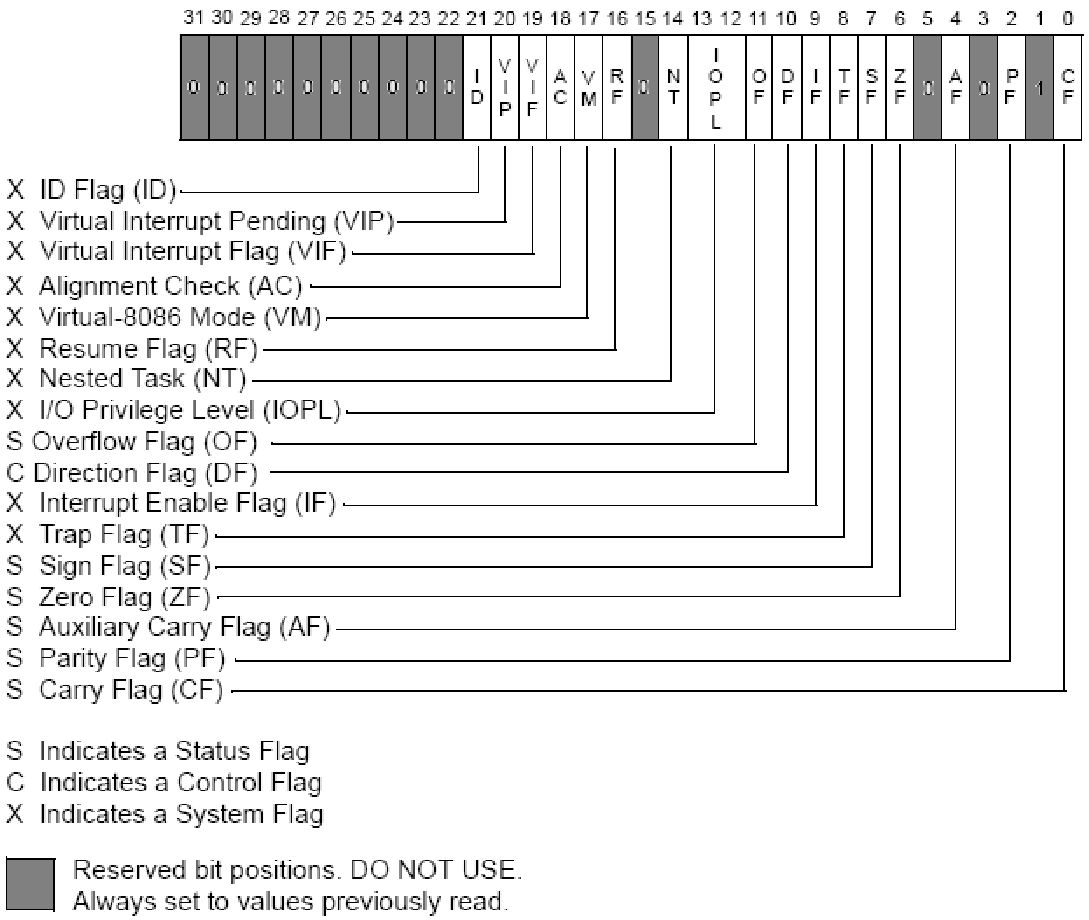
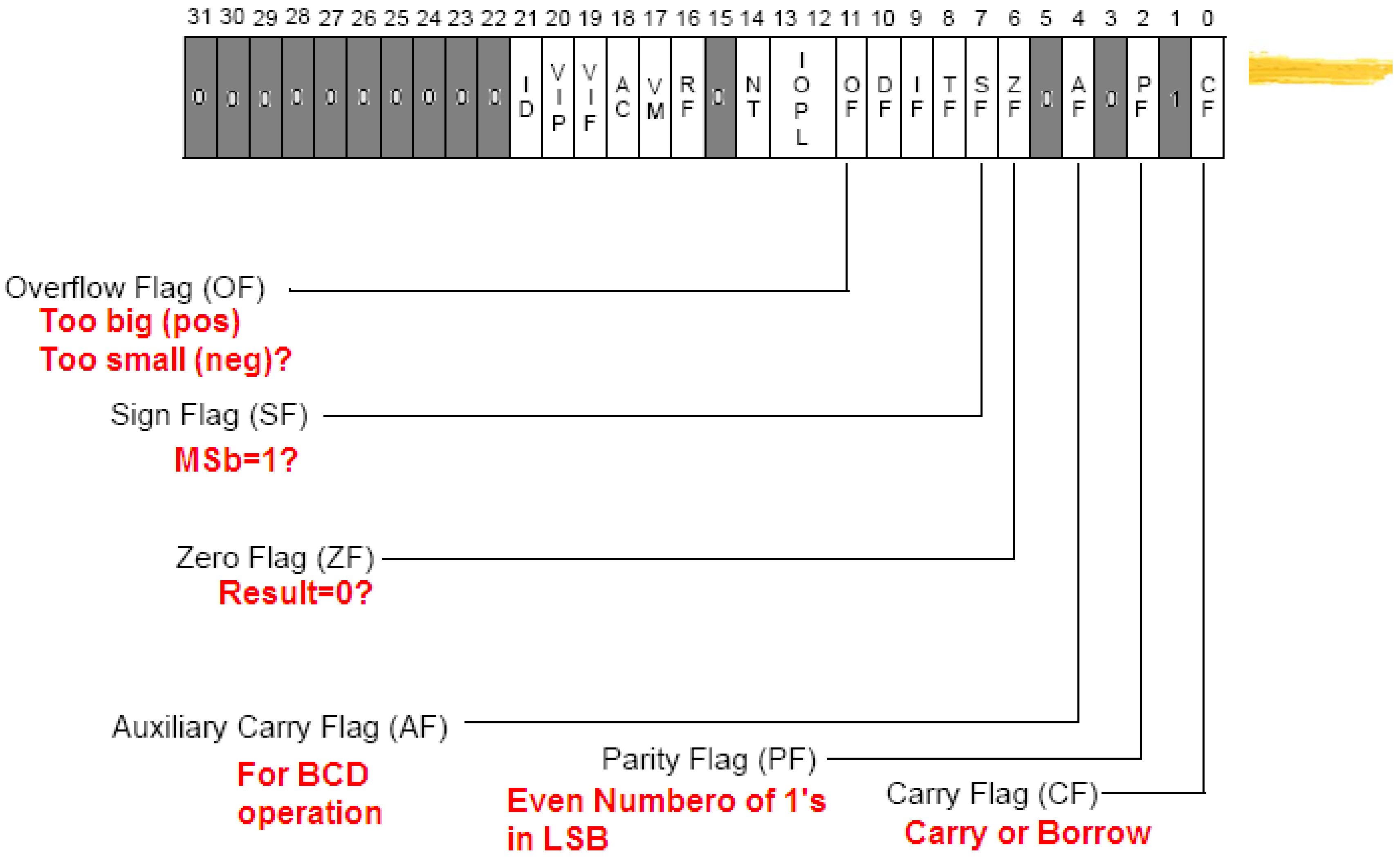


Figure 2-1. Intel386™ DX Base Architecture Registers



31 30 29 28 27 26 25 24 23 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
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.

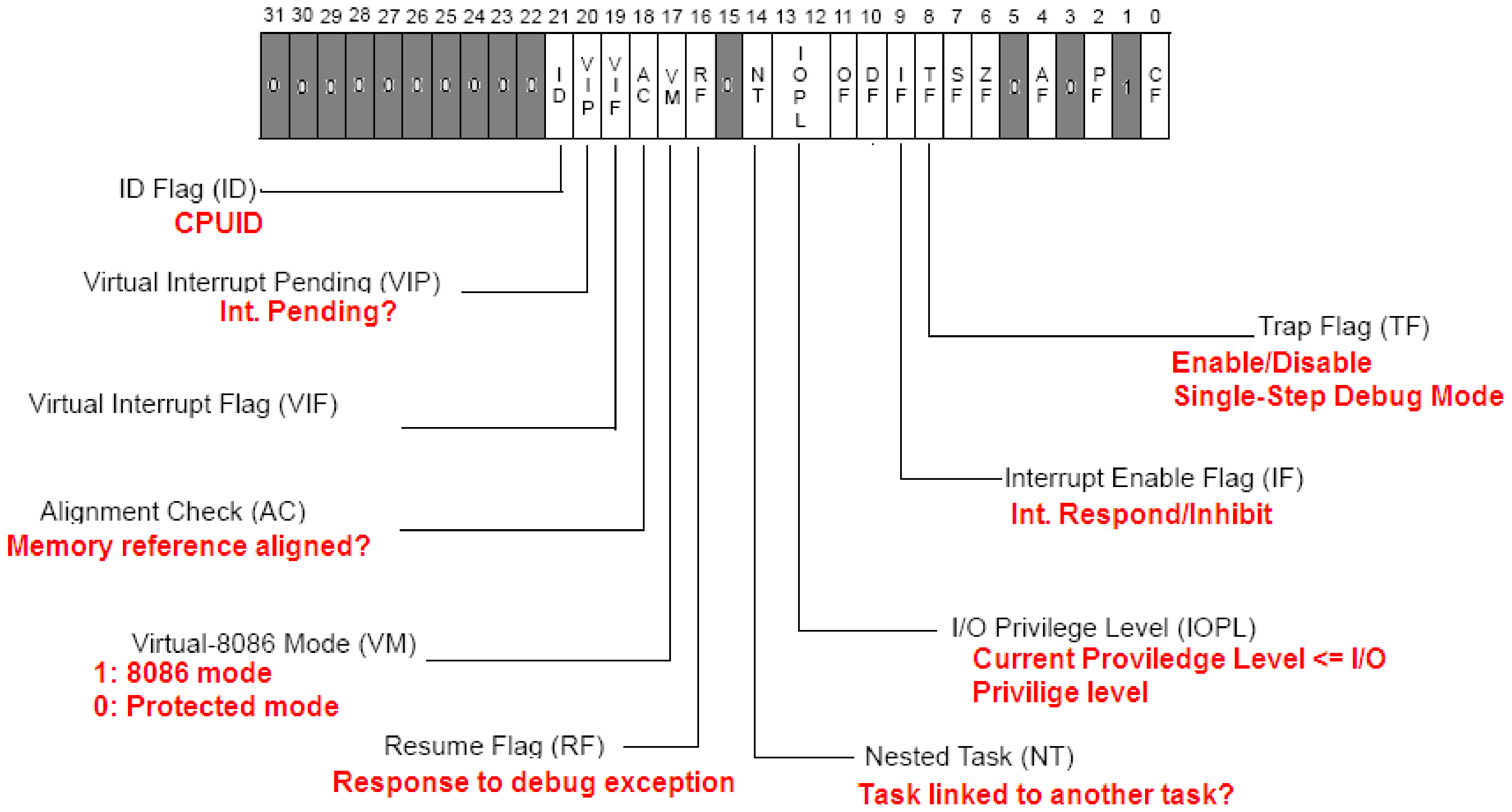
Status Flags



Control Flag (DF)

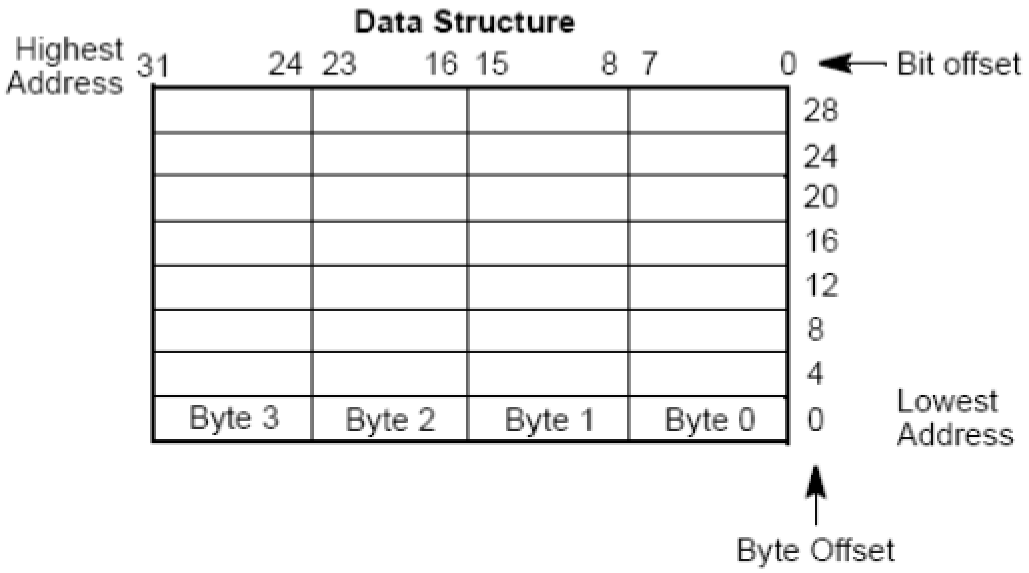
- DF (Direction Flag)
  - The direction flag controls the string instructions (MOVS, CMPS, SCAS, LODS, and STOS).
  - DF=1 → string instructions to auto-decrement (that is, to process strings from high addresses to low addresses).
  - DF=0 → string instructions to auto-increment (process strings from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

System Flags


31 30 29 28 27 26 25 24 23 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
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
Nested Task (NT)
Task linked to another task?
I/O Privilege Level (IOPL)
Current Proviledge Level <= I/O Privilige level
Interrupt Enable Flag (IF)
Int. Respond/Inhibit
Trap Flag (TF)
Enable/Disable Single-Step Debug Mode

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



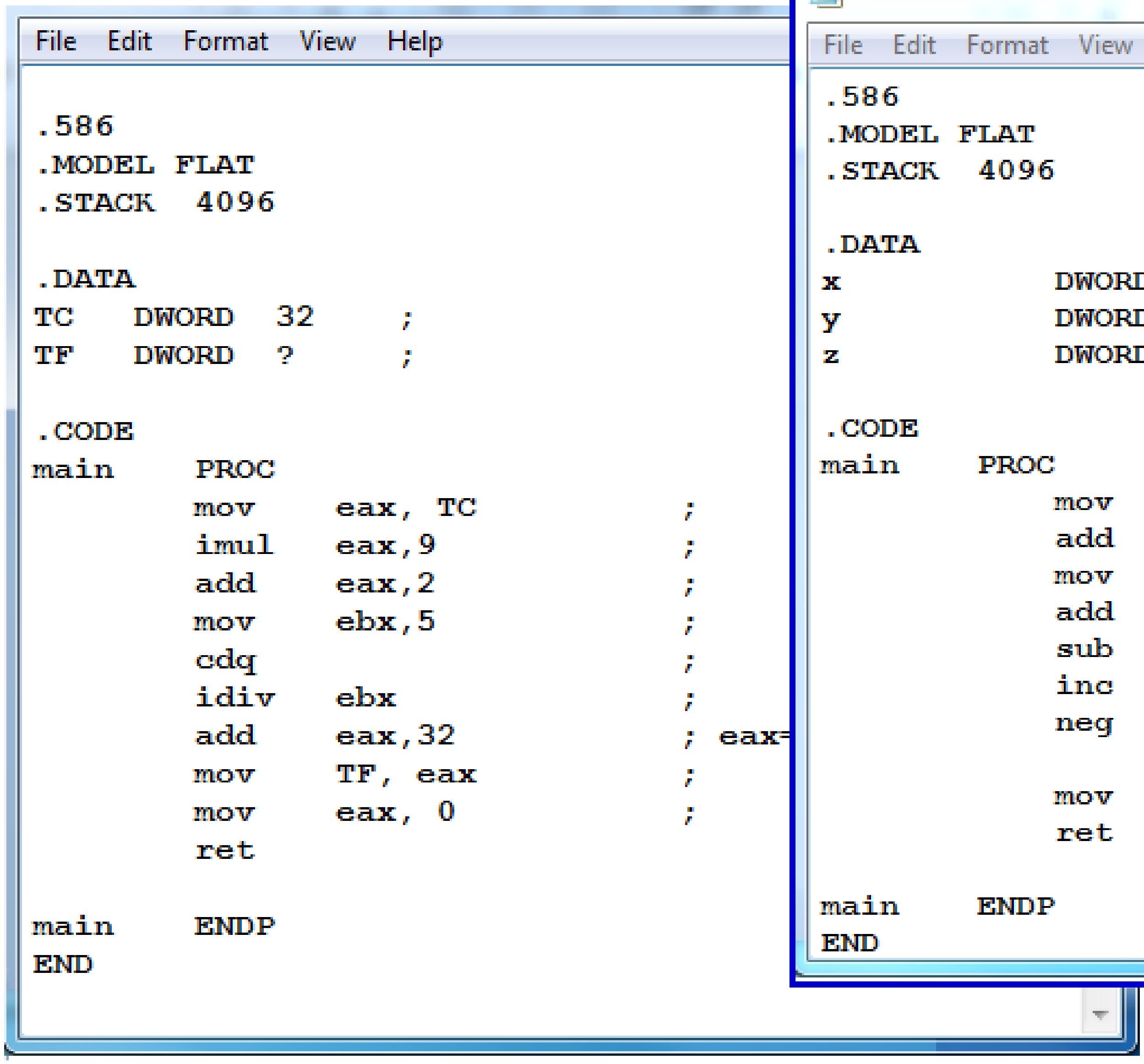
Conventions

- 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.
    - 0 to 3 operands
    - 2 types of form: literals (i.e., numer) or identifiers for data items.
  - When two operands are present in an arithmetic or logical instruction
    - the right operand is the **source** and
    - the left operand, the **destination.**
  - Example:

```
LOADREG:    MOV   EAX, SUBTOTAL
; label          mnemonic     dst          src
```

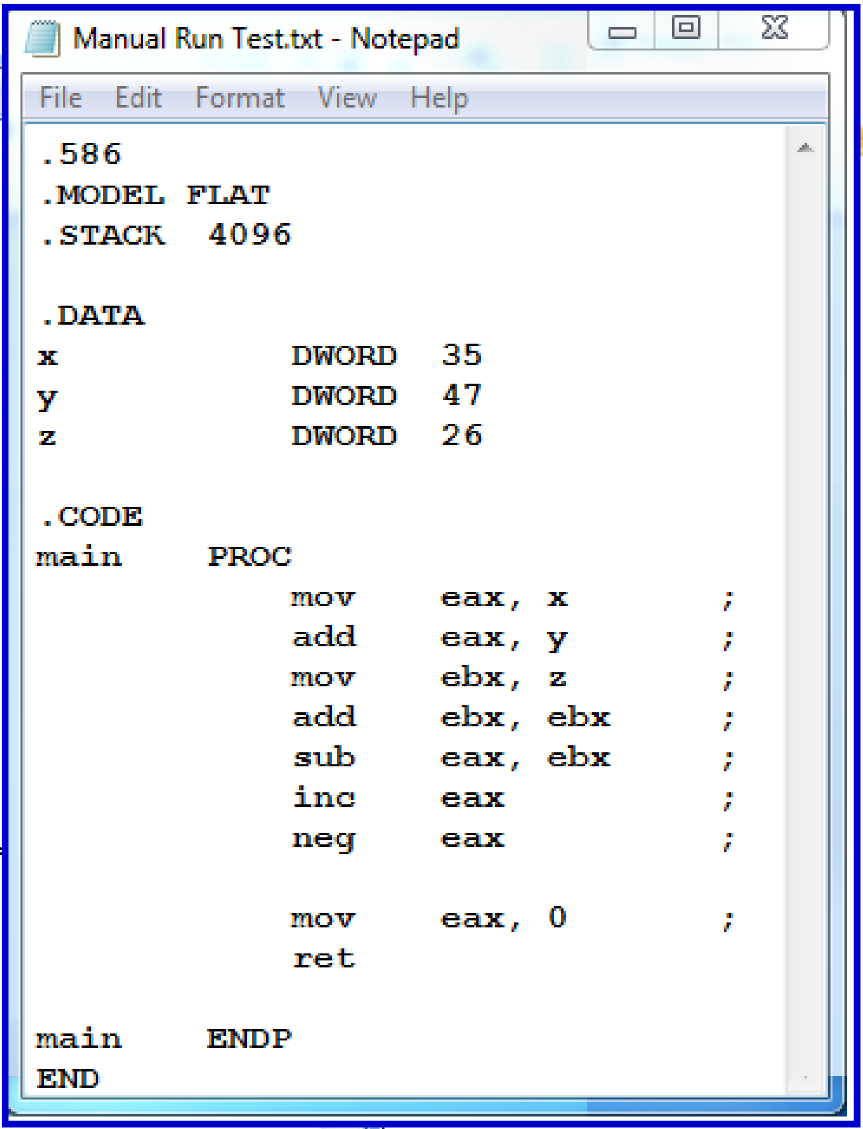
- Number Representation Convention (i386)
  - Default : decimal
  - Binary: a number with 1's and 0's followed by letter B
  - Hexadecimal: a number with 1 – 9-A-F followed by H

How does a simple code look?



File Edit Format View Help

```
.586
.MODEL FLAT
.STACK  4096

.DATA
TC    DWORD   32      ;
TF    DWORD   ?       ;

.CODE
main      PROC
          mov     eax, TC             ;
          imul    eax,9               ;
          add     eax,2               ;
          mov     ebx,5               ;
          cdq                         ;
          idiv    ebx                 ;
          add     eax,32              ; eax=
          mov     TF, eax             ;
          mov     eax, 0              ;
          ret

main      ENDP
END
```



Manual Run Test.txt - Notepad

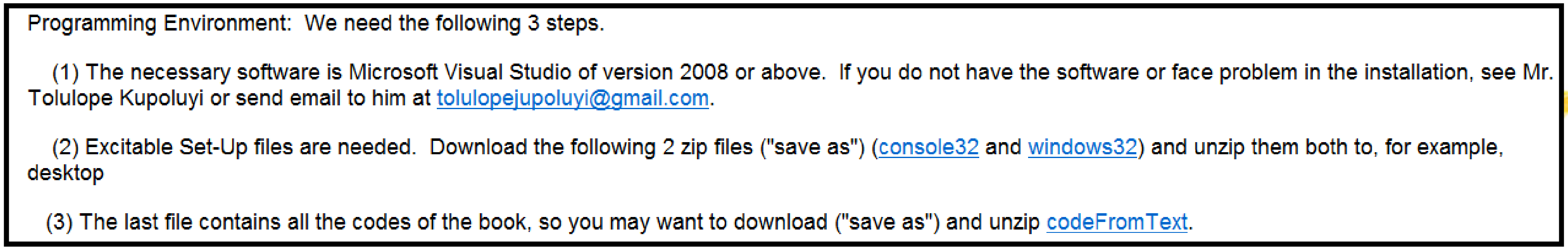
File Edit Format View Help

```
.586
.MODEL FLAT
.STACK  4096

.DATA
x            DWORD   35
y            DWORD   47
z            DWORD   26

.CODE
main      PROC
             mov     eax, x        ;
             add     eax, y        ;
             mov     ebx, z        ;
             add     ebx, ebx      ;
             sub     eax, ebx      ;
             inc     eax           ;
             neg     eax           ;

             mov     eax, 0        ;
             ret

main      ENDP
END
```

Programming Environment

- Next Class:
  - Visual Studio
  - Console 32 (and Console64)
  - Windows 32 (and Windows 64)
  - Bring your PC/Laptop/etc
  - TA will guide you in setting up the system
  - First coding practice
- After today’s class
  - Check the web for Installation Guide
  - Download Zipped filed
  - (Encourage to) install Visual Studio

Web Page and Instruction



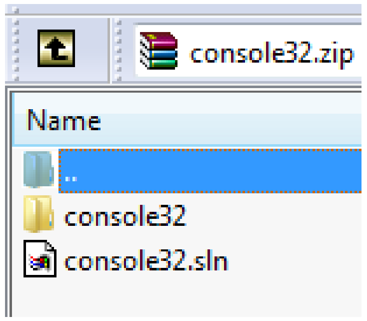
Programming Environment: We need the following 3 steps.

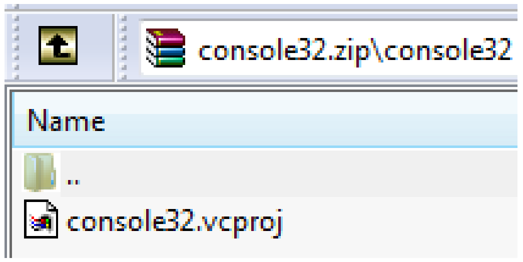
(1) The necessary software is Microsoft Visual Studio of version 2008 or above. If you do not have the software or face problem in the installation, see Mr. Tolulope Kupoluyi or send email to him at tolulopejupoluyi@gmail.com.

(2) Excitable Set-Up files are needed. Download the following 2 zip files ("save as") (console32 and windows32) and unzip them both to, for example, desktop

(3) The last file contains all the codes of the book, so you may want to download ("save as") and unzip codeFromText.

- After unzip, make sure you have these thing right in your computer
  - There must be console32 folder which contains
    - Consol232.sln
    - a sub-folder named 'console32'
      - The sub-folder console32 should contain
        - Console32.vcproj
  - Separately, there must be windows32 folder with
    - Windows32.sln
    - A sub-folder named windows32
      - which contains a sub-sub-folder ("Debug") and several files including window32.vcproj
  - All the codes of the textbook


console32.zip
Name
..
console32
console32.sln


console32.zip\console32
Name
..
console32.vcproj