Computer System

- A computer system has many distinct parts:
  - Microprocessor
  - Memory system
  - I/O system
  - Buses
- The way in which these parts are interconnected change for different computer systems.
80X86

- 80X86 designates a series of general purpose microprocessors produced by Intel.
- Supports 16 and 32 bits address and data buses.
- 32 bit processors are capable of addressing 4 gigabytes of physical memory and 64 terabytes of virtual memory.
## 8086 Architecture

<table>
<thead>
<tr>
<th>16</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AH</td>
<td>AL</td>
<td>Accumulator</td>
</tr>
<tr>
<td>BX</td>
<td></td>
<td>Base</td>
</tr>
<tr>
<td>BH</td>
<td>BL</td>
<td>Count</td>
</tr>
<tr>
<td>CX</td>
<td></td>
<td>Data</td>
</tr>
<tr>
<td>CH</td>
<td>CL</td>
<td></td>
</tr>
<tr>
<td>DX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DH</td>
<td>DL</td>
<td></td>
</tr>
<tr>
<td>DI</td>
<td></td>
<td>Destination Index</td>
</tr>
<tr>
<td>SI</td>
<td></td>
<td>Source Index</td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>BP</td>
<td></td>
<td>Base Pointer</td>
</tr>
<tr>
<td>IP</td>
<td></td>
<td>Instruction Pointer</td>
</tr>
<tr>
<td>FLAGS</td>
<td>Flags</td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS</td>
<td></td>
<td>Segment registers</td>
</tr>
<tr>
<td>ES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# 80386 Architecture

## Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>16-bit</th>
<th>32-bit</th>
<th>8-bit</th>
<th>8-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>AX</td>
<td>AH</td>
<td>AL</td>
<td>AX</td>
</tr>
<tr>
<td>EBX</td>
<td>BX</td>
<td>BH</td>
<td>BL</td>
<td>BX</td>
</tr>
<tr>
<td>ECX</td>
<td>CX</td>
<td>CH</td>
<td>CL</td>
<td>CX</td>
</tr>
<tr>
<td>EDX</td>
<td>DX</td>
<td>DH</td>
<td>DL</td>
<td>DX</td>
</tr>
<tr>
<td>EDI</td>
<td>DI</td>
<td></td>
<td></td>
<td>DI</td>
</tr>
<tr>
<td>ESI</td>
<td>SI</td>
<td></td>
<td></td>
<td>SI</td>
</tr>
<tr>
<td>ESP</td>
<td>SP</td>
<td></td>
<td></td>
<td>SP</td>
</tr>
<tr>
<td>EBP</td>
<td>BP</td>
<td></td>
<td></td>
<td>BP</td>
</tr>
<tr>
<td>EIP</td>
<td>IP</td>
<td></td>
<td></td>
<td>IP</td>
</tr>
<tr>
<td>EFLAGS</td>
<td>FLAGS</td>
<td></td>
<td></td>
<td>FLAGS</td>
</tr>
</tbody>
</table>

### Segment Registers
- CS
- DS
- ES
- SS
- FS
- GS

### Other Registers
- EIP: Instruction Pointer
- ESP: Stack Pointer
- EBP: Base Pointer
- EDI: Destination Index
- ESI: Source Index

### Flags
- Acc
- Carry
- Parity
- Zero
- Sign
- Overflow
- Directive
Registers

- General purpose registers.
  - There are eight 16/32 bits registers.
  - They are used to hold operands for logical and arithmetic operations and to hold addresses.
  - Access may be done in 8, 16 or 32 bits.
Registers

- There is no direct access to the upper 16 bits of the 32 bits registers.

  - Some instructions incorporate dedicated registers in their operations which allows for decreased code size, but it also restricts the use of the register set.
Registers

- AX/EAX – Accumulator is used as a dedicated register by some arithmetic operation, and adjustment instructions.
- BX/EBX – Base index.
- CX/ECX – Count register used by the LOOP, REP, shift and rotate type instructions.
Registers

- **DX/EDX** – Data register used to hold results of multiplication and part of the dividend before for a division instruction.
- **BP/EBP** – Base pointer.
- **DI/EDI** – Destination index, used by string instructions.
- **SI/ESI** – Source index used by string instructions.
Registers

- IP/EIP – Instruction pointer which holds the address of the next instruction to be executed.
- SP/ESP – Stack pointer addresses the stack.
Registers

- Segment registers.
  - There are six 16 bits registers (CS, DS, ES, FS, GS, and SS).
  - They are used to hold the segment selector.
  - Each segment register is associated with a particular kind of memory access.
Registers

- Other registers.
  - EFLAGS controls certain operations and indicates the status of the 80836 (carry, sign, etc).
  - Flags:
    - C – Carry.
    - P – Parity. Count of ones in a number expressed as even or odd. It finds little use in today’s microprocessors.
    - A – Auxiliary carry. Carry or borrow between bits 3 and 4. Used by instructions like DAA and DAS.
Registers

- **Z** – Zero. If **Z** = 1 the result of a computation is zero.
- **S** – Sign. If **S** = 1 sign is negative.
- **T** – Trap. If **T** = 1 debugging is enabled. In this state the microprocessor interrupts the flow of the program to allow the user to check registers, memory, etc.
- **I** – Interrupt. If **I** = 1, the INTR pin is enabled. **STI** and **CLI** controls the state of the **I** flag.
- **D** – The direction flag selects if the **DI** or **SI** registers are incremented or decremented when an string instruction is used.
- **O** – Overflow.
Memory Organization

- Sequence of bytes each with a unique physical address.
- Data types:
  - Byte.
  - Word.
  - Double word.
Memory Organization

8088 microprocessor (memory is only 1M bytes)
80286 microprocessor
80386SX microprocessor
80386SL microprocessor (memory is 32M bytes)
80386SLC microprocessor (memory is 32M bytes)
Memory Organization
Alignment

- It is best to align words with even numbered addresses, and double words to addresses divisible by four.
- The alignment allows for more efficient memory access, but it is less flexible.
Little Endian Notation

- The 80386 stores the least significant byte of a word or double word in the memory location with the lower address.

Assuming EAX = 11223344H

mov ds:[500H], EAX

<table>
<thead>
<tr>
<th>500H</th>
<th>44H</th>
</tr>
</thead>
<tbody>
<tr>
<td>33H</td>
<td></td>
</tr>
<tr>
<td>22H</td>
<td></td>
</tr>
<tr>
<td>11H</td>
<td></td>
</tr>
</tbody>
</table>
Effective, Segment and Physical Addresses

- **Effective address (EA).**
  - Selects an address within a paragraph (64kb memory segment). Also called offset.

- **Segment address (SA).**
  - Defines the beginning of a 64Kb memory segment. Also called segment selectors.

- **Physical address (PA).**
  - Location in memory.
  - \( PA = SA \times 16 + EA \)
## Default Segments

<table>
<thead>
<tr>
<th>Segment</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>EIP</td>
<td>Instruction address</td>
</tr>
<tr>
<td>SS</td>
<td>ESP and EBP</td>
<td>Stack address</td>
</tr>
<tr>
<td>DS</td>
<td>EBX, EDI, ESI, 8 bit and 16 bit numbers</td>
<td>Data address</td>
</tr>
<tr>
<td>ES</td>
<td>DI for string instructions</td>
<td>String destination address</td>
</tr>
<tr>
<td>FS</td>
<td>NONE</td>
<td>General address</td>
</tr>
<tr>
<td>GS</td>
<td>NONE</td>
<td>General address</td>
</tr>
</tbody>
</table>
Program Relocation

Even though the segment and offset addressing scheme seems complicated, it allows an important advantages to the system:

- Programs may be easily relocated in memory.
- Programs will work in any area of memory without modification.
- Programs will work in real and protected modes.
Real Mode Memory Addressing

- This mode allows the microprocessor to address only the first 1Mb of memory space even if the processor is capable of accessing more than that.
- The first Mb of memory is called *real* or *conventional memory*.
Protected Mode Memory Addressing

- This mode allows the microprocessor to address all memory space, in other words, the memory space above and within the first 1Mb of memory. This addressing mode requires a change to the segment plus an offset addressing scheme used in real mode memory addressing.
Data Format

- ASCII.
- BCD.

**Integer:**
- Byte, word, double word, quad word.
- Signed and unsigned.

**Real:**
- Single and double precision.
Character Codes

- **ASCII** – American Standard Code for Information Interchange.
  - Control (0x0 – 0x1F).
  - Alphabet and alphanumeric characters (0x20 – 0x7f).
  - Special characters (0x80 – 0xff).
## ASCII Table

<table>
<thead>
<tr>
<th></th>
<th>0H</th>
<th>1H</th>
<th>2H</th>
<th>3H</th>
<th>4H</th>
<th>5H</th>
<th>6H</th>
<th>7H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1H</td>
<td>0H NUL</td>
<td>1H DLE</td>
<td>2H SP</td>
<td>3H 0</td>
<td>4H @</td>
<td>5H P</td>
<td>6H `</td>
<td>7H p</td>
</tr>
<tr>
<td>2H</td>
<td>1H SOH</td>
<td>2H DC1</td>
<td>3H !</td>
<td>4H 1</td>
<td>5H A</td>
<td>6H Q</td>
<td>7H a</td>
<td>8H q</td>
</tr>
<tr>
<td>3H</td>
<td>2H STX</td>
<td>3H DC2</td>
<td>4H &quot;</td>
<td>5H 2</td>
<td>6H B</td>
<td>7H R</td>
<td>8H b</td>
<td>9H r</td>
</tr>
<tr>
<td>4H</td>
<td>3H ETX</td>
<td>4H DC3</td>
<td>5H #</td>
<td>6H 3</td>
<td>7H C</td>
<td>8H S</td>
<td>9H c</td>
<td>0AH s</td>
</tr>
<tr>
<td>5H</td>
<td>4H EOT</td>
<td>5H DC4</td>
<td>6H $</td>
<td>7H 4</td>
<td>8H D</td>
<td>9H T</td>
<td>0AH d</td>
<td>0BH t</td>
</tr>
<tr>
<td>6H</td>
<td>5H ENQ</td>
<td>6H NAK</td>
<td>7H %</td>
<td>8H 5</td>
<td>9H E</td>
<td>0AH U</td>
<td>0BH e</td>
<td>0BH u</td>
</tr>
<tr>
<td>7H</td>
<td>6H ACK</td>
<td>7H SYN</td>
<td>8H &amp;</td>
<td>9H 6</td>
<td>0AH F</td>
<td>1AH V</td>
<td>2AH f</td>
<td>2AH v</td>
</tr>
<tr>
<td>8H</td>
<td>7H BEL</td>
<td>8H ETB</td>
<td>9H '</td>
<td>0AH 7</td>
<td>1AH G</td>
<td>2AH W</td>
<td>3AH g</td>
<td>3AH w</td>
</tr>
<tr>
<td>9H</td>
<td>8H BS</td>
<td>9H CAN</td>
<td>0AH (</td>
<td>1AH 8</td>
<td>2AH H</td>
<td>3AH X</td>
<td>4AH h</td>
<td>4AH x</td>
</tr>
<tr>
<td>0AH</td>
<td>9H HT</td>
<td>0AH EM</td>
<td>1AH )</td>
<td>2AH 9</td>
<td>3AH I</td>
<td>4AH Y</td>
<td>5AH l</td>
<td>5AH y</td>
</tr>
<tr>
<td>0BH</td>
<td>LF</td>
<td>1AH SUB</td>
<td>2AH *</td>
<td>3AH :</td>
<td>4AH J</td>
<td>5AH Z</td>
<td>6AH j</td>
<td>6AH z</td>
</tr>
<tr>
<td>0CH</td>
<td>VT</td>
<td>2AH ESC</td>
<td>3AH +</td>
<td>4AH ;</td>
<td>5AH K</td>
<td>6AH [</td>
<td>7AH k</td>
<td>7AH {</td>
</tr>
<tr>
<td>0DH</td>
<td>FF</td>
<td>3AH FS</td>
<td>4AH &lt;</td>
<td>5AH L</td>
<td>6AH \</td>
<td>7AH l</td>
<td>8AH l</td>
<td>8AH ]</td>
</tr>
<tr>
<td>0EH</td>
<td>CR</td>
<td>4AH GS</td>
<td>5AH =</td>
<td>6AH M</td>
<td>7AH ]</td>
<td>8AH m</td>
<td>9AH }</td>
<td>9AH }</td>
</tr>
<tr>
<td>0FH</td>
<td>SO</td>
<td>5AH RS</td>
<td>6AH &gt;</td>
<td>7AH N</td>
<td>8AH ^</td>
<td>9AH n</td>
<td>0AH ~</td>
<td>0AH ~</td>
</tr>
<tr>
<td>0FH</td>
<td>SI</td>
<td>6AH US</td>
<td>7AH /</td>
<td>8AH ?</td>
<td>9AH O</td>
<td>0AH _</td>
<td>1AH o</td>
<td>1AH DEL</td>
</tr>
</tbody>
</table>
Character Codes

- EBCDIC.
  - IBM.
- Baudot.
  - TDDs – telecommunication devices for the deaf.
  - HAM radio applications.
Binary Coded Decimal

- Also known as BCD or 8421 weighted code.
- BCD encodes the decimal digits 0 – 9 with their unsigned binary representation, \(0000_2 – 1001_2\).
- Packed BCD – a byte represents two BCD numbers.
- Unpacked BCD - a byte represents only one BCD number.
BCD Addition

- Similar to unsigned binary addition, except that a correction is required should the result exceed $1001_2$.
- The correct result may be obtained by adding $0110_2$.
- Ex:
  - $6 + 8 = 14$
  - $3 + 3 = 6$
  - $9 + 9 = 18$
Integer

- **Unsigned:**
  - Range: 0 to $2^n - 1$.

- **Signed (two’s complement):**
  - Range: $2^{n-1}$ to $2^{n-1} - 1$. 
Floating Point Format

- **Single precision:**
  - 1 bit sign;
  - 8 bit biased exponent;
  - 23 bit normalized significand.

- **Double precision:**
  - 1 bit sign;
  - 11 bit biased exponent;
  - 52 bit normalized significand.
Floating Point Format

- **Normalizing:**
  - Adjusting the number so that its value is at least 1 but less than 2.

- **Biased exponent:**
  - Bias is 127 for single precision.
  - Bias is 1023 for double precision.
  - Biased exponent = Bias + exponent.
Assembly Language Program

- Series of statements which are either assembly language instructions or directives.
  - Instructions are statements like ADD AX,BX which are translated into machine code.
  - Directives or pseudo-instructions are statements used by the programmer to direct the assembler on how to proceed in the assembly process.

- Statement format:
  - [label:] mnemonic [operands][;comments]
Assembly Language Program

- **Label:**
  - Cannot exceed 31 characters.
  - Consists:
    - Alphabetic characters both upper and lower case.
    - Digits 0 through 9.
    - Special characters ( ? ), ( . ), ( @ ), ( _ ), and ( $ ).
  - The first character cannot be a digit.
  - The period can only be used as the first character, but its use is not recommended. Several reserved words begin with it in later versions of MASM.
Assembly Language Program

- **Label:**
  - Must end with a colon when it refers to an opcode generating instruction.
  - Do not need to end with a colon when it refers to a directive.

- **Mnemonic and operands:**
  - Instructions are translated into machine code.
  - Directives do not generate machine code. They are used by the assembler to organize the program and direct the assembly process.
Assembly Language Program

- Comments:
  - Begin with a “;”.
  - Ignored by the assembler.
  - Maybe be on a line by itself or at the end of a line:
    - ;My first comment
    - MOV AX,1234H ;Initializing....
  - Indispensable to the programmers because they make it easier for someone to read and understand the program.
Segment Definition

The CPU has several segment registers:

- CS (code segment).
- SS (stack segment).
- DS (data segment).
- ES (extra segment).
- FS, GS (supplemental segments available on 386s, 486s and Pentiums).

Every instruction and directive must correspond to a segment.
Segment Definition

- Normally a program consists of three segments: the stack, the data, and the code segments.
- Model definition.
  - .MODEL SMALL
    - Most widely used memory model.
    - The code must fit in 64k.
    - The data must fit in 64k.
  - .MODEL MEDIUM
    - The code can exceed 64k.
    - The data must fit in 64k.
  - .MODEL COMPACT
    - The code must fit in 64k.
    - The data can exceed 64k.
Segment Definition

- **MEDIUM** and **COMPACT** are opposites.

- **.MODEL LARGE**
  - Both code and data can exceed 64k.
  - No single set of data can exceed 64k.

- **.MODEL HUGE**
  - Both code and data can exceed 64k.
  - A single set of data can exceed 64k.

- **.MODEL TINY**
  - Used with COM files.
  - Both code and data must fit in a single 64k segment.
Segment Definition

- Segment definition formats:
  - Simplified segment definition.
  - Full segment definition.

- The Simplified segment definition uses the following directives to define the segments:
  - .STACK
  - .DATA
  - .CODE
  - These directives mark the beginning of the segments they represent.
Segment Definition

The full segment definition uses the following directives to define the segments:

- Label SEGMENT [options]
  ;Statements belonging to the segment.
  Label ENDS

- The label must follow naming conventions previously discussed.
Constants

- EQU is used to define constants or to assign names to expressions.

Form:
  - Name EQU expression.

Examples:
  - PI  EQU  3.1415
  - Radius EQU 25
  - Circumference EQU 2*PI*Radius
Variables

- **DB** - define byte.
- **DW** - define word.
- **DD** – define double word.
- **DQ** – Define quad word.

**Form:**
- Variable Directive oper, . . ,oper

**Examples:**
- Alpha db ‘ABCDE’
- Alpha2 db ‘A’,’B’,’C’,’D’,’E’
- Alpha3 db 41h,42h,43h,44h,45h
- Word1 dw 3344h
- Double_word1 dd 44332211h
Variables

- **REAL4** – Defines single precision real number.
- **REAL8** - Defines single precision real number.

Form:
- Variable Directive oper, . . ,oper

Ex:
- NUM1 REAL4 -12.65
- NUM2 DD 2.25
- NUM3 REAL8 4.4E3
- NUM4 DQ 144.55
Duplicate Directive

- This directive makes it easy to create arrays.

- Form:
  - Expression DUP(Operand,….,Operand)
    - Expression must evaluate to a positive value.
  - ARRAY DW 2 DUP(0,1,2 DUP(2,3),4)
Data Structures

- Used to specify how information is stored in a memory array. It is a template for data.

- Form:
  - Label STRUC
    - Array defined in this area;
  - Label ENDS
Data Structures

- Example:
  - INFO STRUC
    - NAMES DB 32 DUP(?)
    - STREET DB 32 DUP(?)
    - CITY DB 16 DUP(?)
    - STATE DB 2 DUP(?)
    - ZIP DB 5 DUP(?)
  - Label ENDS
;SIMPLIFIED SEGMENT DEFINITION ;FULL SEGMENT DEFINITION

.MODEL SMALL

.STACK 64

.DATA
N1 DW 1432H
N2 DW 4365H
SUM DW 0H

.CODE
BEGIN PROC FAR
MOV AX,@DATA
MOV DS,AX
ADD AX,N1
MOV AX,4CH
INT 21H
BEGIN ENDP

CDSEG SEGMENT
BEGIN PROC FAR
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
MOV AX,DTSEG
MOV DS,AX
ADD AX,N2
MOV AX,4CH
INT 21H
BEGIN ENDP

END BEGIN