Memory Models, Instruction Operand Notation and Data Transfer Instructions

Outline of the Lecture
- Memory Models.
- Instruction Operand Notation.
- Data Transfer Instructions.

Memory Models

Memory Models that can be used in assembly language are the following:

1. **TINY MODEL (.MODEL TINY):**
   - The model uses maximum of 64K bytes for Code and Data.

2. **SMALL MODEL (.MODEL SMALL):**
   - The model uses maximum of 64K bytes for Code and 64K bytes for Data (Code<=64K and Data <=64K).
   - This model is the most widely used memory model and is sufficient for all the programs to be used in this course.

3. **MEDIUM MODEL, (.MODEL MEDIUM):**
   - The model uses maximum of 64K bytes for Data and Code can exceed 64K bytes (Code>64K and Data <=64K).

4. **COMPACT MODEL, (.MODEL COMPACT):**
   - The model uses maximum of 64K bytes for Code and Data can exceed 64K bytes (Code<=64K and Data >64K).

5. **LARGE MODEL, (.MODEL LARGE):**
   - Both Code and Data can exceed 64K bytes. However no single data set (i.e. array) can exceed 64K bytes (Code>64K and Data >64K).

6. **HUGE MODEL, (.MODEL HUGE):**
   - Both Code and Data can exceed 64K bytes. Additionally, a single data set (i.e. array) can exceed 64K bytes (Code>64K and Data >64K).

7. **FLAT MODEL, (.MODEL FLAT)**
   - Window NT Application

Attributes of Memory Models

<table>
<thead>
<tr>
<th>Memory Model</th>
<th>Default Code</th>
<th>Default Data</th>
<th>Operating System</th>
<th>Data and Code Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny</td>
<td>Near</td>
<td>Near</td>
<td>MS-DOS</td>
<td>Yes</td>
</tr>
<tr>
<td>Small</td>
<td>Near</td>
<td>Near</td>
<td>MS-DOS, Windows</td>
<td>No</td>
</tr>
<tr>
<td>Medium</td>
<td>Far</td>
<td>Near</td>
<td>MS-DOS, Windows</td>
<td>No</td>
</tr>
<tr>
<td>Compact</td>
<td>Near</td>
<td>Far</td>
<td>MS-DOS, Windows</td>
<td>No</td>
</tr>
<tr>
<td>Large</td>
<td>Far</td>
<td>Far</td>
<td>MS-DOS, Windows</td>
<td>No</td>
</tr>
<tr>
<td>Huge</td>
<td>Far</td>
<td>Far</td>
<td>MS-DOS, Windows</td>
<td>No</td>
</tr>
<tr>
<td>Flat</td>
<td>Near</td>
<td>Near</td>
<td>Windows NT</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Example

TITLE Add and Subtract (addsub.asm)
; This program adds and subtracts integers
.686
.MODEL flat, stdcall
.STACK
INCLUDE Irvine32.inc
.code
main PROC
    mov eax, 60000h ; EAX = 60000h
    add eax, 80000h ; EAX = EAX + 80000h
    sub eax, 20000h ; EAX = EAX - 20000h
    exit
main ENDP
END main

- The .MODEL directive tells the assembler to use standard conventions for names and procedure calls.
- The .686 is a processor directive used before the .MODEL FLAT directive to provide access to the 32-bit instructions and registers available in the Pentium Processor.
- The STDCALL directive tells the assembler to use standard conventions for names and procedure calls.

Instruction Operand Notation

<table>
<thead>
<tr>
<th>Operand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r8</td>
<td>8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL</td>
</tr>
<tr>
<td>r16</td>
<td>16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP</td>
</tr>
<tr>
<td>r32</td>
<td>32-bit general-purpose register: EAX, EBX, ECX, EDX, ESP, EBP</td>
</tr>
<tr>
<td>reg</td>
<td>Any general-purpose register</td>
</tr>
<tr>
<td>sreg</td>
<td>16-bit segment register: CS, DS, SS, ES, FS, GS</td>
</tr>
<tr>
<td>imm</td>
<td>8-, 16-, or 32-bit immediate value</td>
</tr>
<tr>
<td>imm8</td>
<td>8-bit immediate byte value</td>
</tr>
<tr>
<td>imm16</td>
<td>16-bit immediate word value</td>
</tr>
<tr>
<td>imm32</td>
<td>32-bit immediate doubleword value</td>
</tr>
<tr>
<td>r/m8</td>
<td>8-bit operand which can be an 8-bit general-purpose register or memory byte</td>
</tr>
<tr>
<td>r/m16</td>
<td>16-bit operand which can be a 16-bit general-purpose register or memory word</td>
</tr>
<tr>
<td>r/m32</td>
<td>32-bit operand which can be a 32-bit general register or memory doubleword</td>
</tr>
<tr>
<td>mem</td>
<td>8-, 16-, or 32-bit memory operand</td>
</tr>
</tbody>
</table>
Data Transfer Instructions

MOV Instruction
- Move source operand to destination, the syntax is
  `mov destination, source`
- Source and destination operands can vary
  `mov reg, reg`
  `mov mem, reg`
  `mov reg, mem`
  `mov mem, imm`
  `mov reg, imm`
  `mov r/ml16, sreg`
  `mov sreg, r/ml16`

Rules
- Both operands must be of same size
- No memory to memory moves
- No immediate to segment moves
- No segment to segment moves
- Destination cannot be CS

MOV Examples

```
.DATA
count BYTE 100
bVal BYTE 20
wVal WORD 2
dVal DWORD 5
.CODE
mov bl, count ; bl = count = 100
mov ax, wVal ; ax = wVal = 2
mov count,al ; count = al = 2
mov eax, dVal ; eax = dval = 5

; Assembler will not accept the following moves – why?
mov ds, 45 ; immediate move to DS not permitted
mov esi, wVal ; size mismatch
mov eip, dVal; EIP cannot be the destination
mov 25, bVal; immediate value cannot be
            ; destination
mov bVal,count; memory-to-memory move not
            ; permitted
```