Today’s Topics

• Sections 3.1 to 3.3: Learning Assembly by Examples.
  – Program Formats
  – Some Simple Instructions
  – Assemble and Execute
• Section 3.4: Data Definition.
• Section 3.5: Constants.
Example: Adding 3 Integers

TITLE Add and Sub
; This is a comment
INCLUDE Irvine32.inc

.code
Main proc
    MOV EAX, 10000h
    ADD EAX, 40000h
    SUB EAX, 20000h
    CALL DumpRegs
    EXIT
Main endp
End Main

Format of Program

• Identifiers are case insensitive.
• Comments follow ‘;’ character.
• Sections: Data, Code, and Stack.
• Hexadecimal Numbers have ‘h’ suffix.
• Directives vs. Instructions.
  – Use Directives to tell assembler what to do.
  – Use Instructions to tell CPU what to do.
• Procedure defined by:
  – [Name] PROC
  – [Name] ENDP
• Instruction Format:
  – LABEL (optional), Mnemonic, Operands

Directives

• Commands that are recognized and acted upon by the assembler
  – Not part of the Intel instruction set
  – Used to declare code, data areas, select memory model, declare procedures, etc.
• Different assemblers have different directives
  – NASM != MASM, for example
Instructions

- Assembled into machine code by assembler
- Executed at runtime by the CPU
- Member of the Intel IA-32 instruction set
- Parts
  - Label
  - Mnemonic
  - Operand
  - Comment

I/O

- Not as easy as you think, if you program it yourself.
- We will use the library provided by the author of the textbook.
- Two steps:
  - Include the library (Irvine32.inc) in your code.
  - Call the subroutines.
Assemble and Run!

• The required software comes with the book:
  – MASM: Microsoft Macro Assembler
  – CodeView: A Debugger
  – Irvine32.inc: I/O procedures

Assemble-Link Execute Cycle

• The following diagram describes the steps from creating a source program through executing the compiled program.
• If the source code is modified, Steps 2 through 4 must be repeated.
make32.bat

- Called a batch file
- Run it to assemble and link programs
- Contains a command that executes ML.EXE (the Microsoft Assembler)
- Contains a command that executes LINK32.EXE (the 32-bit Microsoft Linker)
- Command-Line syntax:
  ```
  make32  progName
  ```
  
  (progName includes the .asm extension)

  (use make16.bat to assemble and link Real-mode programs)

Example: Adding and Subtracting Integers

```assembly
TITLE Add and Subtract           (AddSub.asm)
; This program adds and subtracts 32-bit integers.
INCLUDE Irvine32.inc
.code
main PROC
    mov eax,10000h        ; EAX = 10000h
    add eax,40000h        ; EAX = 50000h
    sub eax,20000h        ; EAX = 30000h
    call DumpRegs         ; display registers
exit
main ENDP
END main
```
Example Output

Program output, showing registers and flags:

```
EAX=00030000  EBX=7FFDF000  ECX=00000101  EDX=FFFFFFFF
ESI=00000000  EDI=00000000  EBP=0012FFC4  ESP=0012FFF0
EIP=00401024  EFL=00000206  CF=0  SF=0  ZF=0  OF=0
```

Alternative Version of AddSub

```
TITLE Add and Subtract              (AddSubAlt.asm)
; This program adds and subtracts 32-bit integers.
.386
.MODEL flat,stdcall
.STACK 4096

ExitProcess PROTO, dwExitCode:DWORD
DumpRegs PROTO

.code
main PROC
    mov eax,10000h        ; EAX = 10000h
    add eax,40000h        ; EAX = 50000h
    sub eax,20000h        ; EAX = 30000h
    call DumpRegs
    INVOKE ExitProcess,0
main ENDP
END main
```
Program Template

<table>
<thead>
<tr>
<th>TITLE Program Template           (Template.asm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>; Program Description:</td>
</tr>
<tr>
<td>; Author:</td>
</tr>
<tr>
<td>; Creation Date:</td>
</tr>
<tr>
<td>; Revisions:</td>
</tr>
<tr>
<td>; Date: Modified by:</td>
</tr>
<tr>
<td>INCLUDE Irvine32.inc</td>
</tr>
<tr>
<td>.data</td>
</tr>
<tr>
<td>; (insert variables here)</td>
</tr>
<tr>
<td>.code</td>
</tr>
<tr>
<td>main PROC</td>
</tr>
<tr>
<td>; (insert executable instructions here)</td>
</tr>
<tr>
<td>exit</td>
</tr>
<tr>
<td>main ENDP</td>
</tr>
<tr>
<td>; (insert additional procedures here)</td>
</tr>
<tr>
<td>END main</td>
</tr>
</tbody>
</table>

Listing File

• Use it to see how your program is compiled
• Contains
  – source code
  – addresses
  – object code (machine language)
  – segment names
  – symbols (variables, procedures, and constants)
• Example: addSub.lst
Map File

- Information about each program segment:
  - starting address
  - ending address
  - size
  - segment type
- Example: addSub.map

Data Declaration

- [Label], Type, Initialization (or just ?)
  - Example: Var1 BYTE 7
- Use ? if no initialization necessary.
  - Example: Var1 BYTE ?
- Other data types: (See Table 3-2 in p.80)
  - WORD (or DW),
  - DWORD (or DD), …etc.
Signed vs. Unsigned

• Signed vs. Unsigned:
  – SBYTE vs. BYTE
  – SWORD vs. WORD
  – …etc.
• Example:
  – Var1 BYTE 255
  – Var1 SBYTE -1

Characters and Strings

• How about characters?
• A few examples:
  – Var1 BYTE ‘A’
  – Var1 BYTE 41h
  – S1 BYTE ‘Hello, World!’
  – Line BYTE ‘-------------------’
How About A List or Array?

• Just list them! For example:
  – List1 BYTE 10, 32, 41h
• You may also mix them:
  – List1 BYTE 10, 32, ?, ‘A’
• So, Is List2 an Array?
  – Yes and No! (e.g., List2 actually means ‘g’.)

DUP

• Good for allocating space for a string or array.
• Examples:
  – Var1 BYTE 20 DUP (0)
  – Var2 BYTE 20 DUP (?)
  – Var3 BYTE 4 DUP (“STACK”)
EQU is for Constants

- EQU for constants.
- Also OK to use = for integers.
- Examples:
  - COUNT = 500
  - COUNT EQU 500

Expression and Text in EQU

- OK to use expressions in EQU:
  - Matrix1 EQU 10 * 10
  - Matrix1 EQU <10 * 10>
- No expression evaluation if within < >.
- EQU accepts texts too:
  - Msg EQU <“Press any key…”, 0>
• Why EQU? Why not storing the constants as variables (in memory)?
• Exercise: Does it save memory? Isn’t the constant stored somewhere in memory anyway?  (*Hint: See the slides last week.*)