EECE416 :Microcomputer Fundamentals and Design

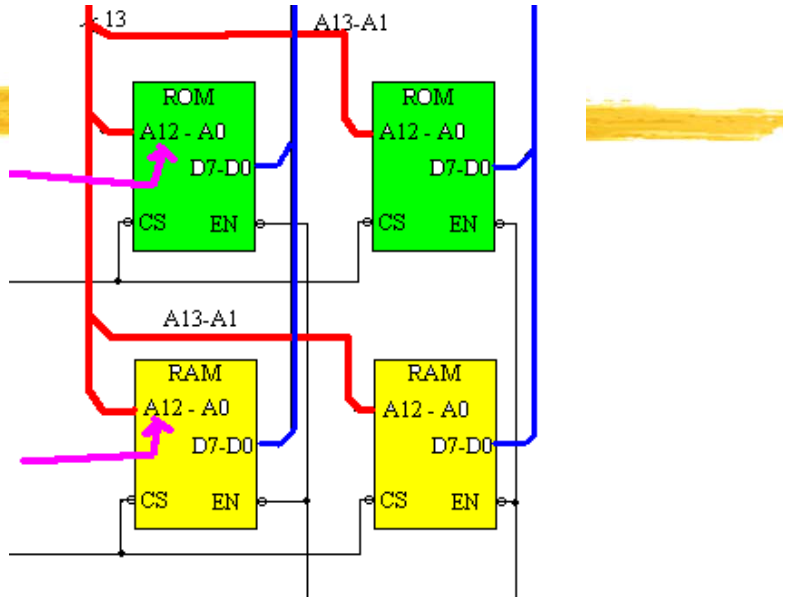
X86 Assembly Programming

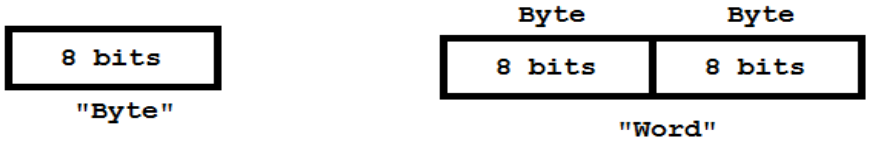
Part 1 – MASM

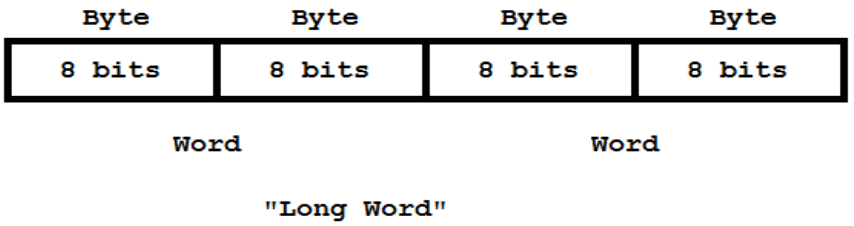
www.MWFTR.com

Multiple Address Access Issues

- Reminder: a single address locates only a byte of memory
- 8-bit processor
  - Accesses one address with a byte data
- 16-bit processor
  - Accesses two address spaces (Even (or Low Byte) and Odd(or High Byte)) at a single execution with 2-byte (or "Word") data
  - Where do we store each of the 2 bytes to each of the 2 address spaces?







Big-Endian vs. Little-Endian

- **Big-Endian**: Words are stored with the lower 8- bits ("Lower Byte") in the higher of the two storage locations ("Addresses"): **Motorola**
  - "**Big** guy ("Upper Byte") **ends at lower address**"
- **Little- Endian**: Lower-order byte stored in the lowest address: **Intel 80x86 family**
  - **Little** guys ("Lower Byte") **ends at lower address**"

Big-Endian vs. Little-Endian

Word Data | Byte1 (H) | Byte0 (L)

Store data

```
0x3210 = 12816
0x1032 =  4146
```

```
          dst   src
MOV        0,  $3210
MOV        2 ,  $76543210
```

000006h	
000005	76
000004	54
000003	32
000002	10
000001	32
000000	10

Little-Endian

Intel

```
          src   dst
MOVE  $3210,  0
MOVE  $76543210, 2
```

000006h	
000005	10
000004	32
000003	54
000002	76
000001	10
000000	32

Big-Endian

Motorola

“Endianness”

- Endian or Endian-Architecture
  - how multi-byte data is represented by a computer system and is dictated by the CPU architecture of the system
  - Not all computer systems are designed with the same endian architecture
  - Issues with software and interface

Computer System Endianness

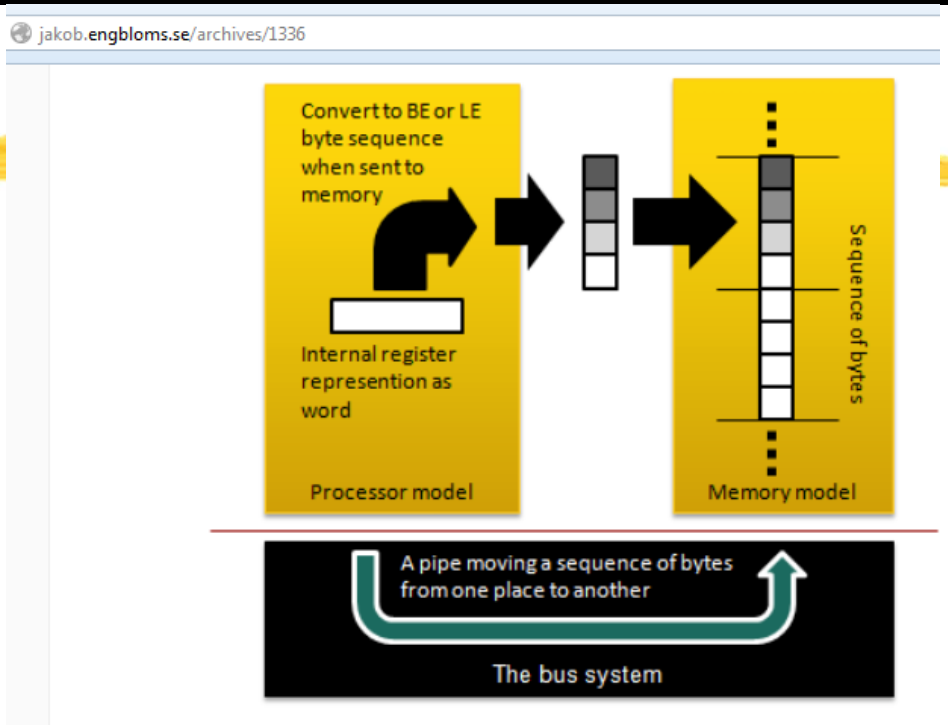
Platform	Endian Architecture
ARM*	Bi-Endian
DEC Alpha*	Little-Endian
HP PA-RISC 8000*	Bi-Endian
IBM PowerPC*	Bi-Endian
Intel® 80x86	Little-Endian
Intel® IXP network processors	Bi-Endian
Intel® Itanium® processor family	Bi-Endian
Java Virtual Machine*	Big-Endian
MIPS*	Bi-Endian
Motorola 68k*	Big-Endian
Sun SPARC*	Big-Endian

Common file formats

Little-Endian Format		Big-Endian Format		Variable or Bi-Endian Format	
BMP	(Windows* & OS/2)	PSD	(Adobe Photoshop*)	DXF	(AutoCAD*)
GIF		IMG	(GEM Raster*)	PS	(Postscript*, 8 bit interpreted text, no Endian issue)
FLI	(Autodesk Animator*)	JPEG, JPG		POV	(Persistence of Visionraytracer*)
PCX	(PC Paintbrush*)	MacPaint		RIFF	(WAV & AVI*)
QTM	(MAC Quicktime*)	SGI	(Silicon Graphics*)	TIFF	
RTF	(Rich Text Format)	Sun Raster		XWD	(X Window Dump*)
		WPG	(WordPerfect*)		
Bus Protocols		**Network Protocols**		**Bus Protocols**	
Infiniband		TCP/IP		GMII	(8 bit wide bus, no Endian issue)
PCI Express		UDP			
PCI-32/PCI-64					
USB					

Bi-Endian & Endian-Neutral Approaches

- Conversion
  - Byte Swap
  - Network I/O Macro



- "Endian Neutral":

HOMEWORK

- Technical Report on “Endian-Neutral Approaches”
  - What are they?
  - How do they work?
  - Who are involved and leading the works
  - Submission Due: Hardcopy + Softcopy by 5:30pm Thursday Oct 13

- How to write well for busy technical people ?

Compare this

www.cbsnews.com/8301-202_162-57600384/syria-strike-seems-inevitable-as-u.n-warns-against-unilateral-military-action-hunt-

Updated at 6:48 a.m. Eastern

DAMASCUS, SYRIA | U.N. chemical weapons experts investigating an alleged poison gas attack near Damascus left their hotel again Wednesday hoping to carry out their second field trip, which was delayed Tuesday for security reasons.

The team of about 20 inspectors left their hotel in the Syrian capital in a convoy of cars to visit the eastern Ghouta suburbs, where the Obama administration says President Bashar Assad's forces unleashed a chemical weapons attack on Aug. 21 that killed hundreds of people.

Local opposition activists told CBS News that the convoy had reached the town of Mleiha, in the sprawling Ghouta area, and videos posted online by the activists showed the U.N. inspectors interviewing patients at clinics in Mleiha and the nearby town of Zamalka.

Play VIDEO

Intercepted communications, tissue samples prove Syrian regime responsible for gas attack

On Tuesday, Vice President Joe Biden made it clear that regardless of what the U.N. inspectors find, the White House is now convinced the attack was carried out by Assad's forces.

The American government's assessment is based on the circumstantial evidence from videos posted on the internet, and, as CBS News correspondent David Martin reported Tuesday, intelligence -- much of it still classified -- ranging from intercepted Syrian communications to tests of tissue samples taken from victims.

Another key piece of circumstantial evidence which has been cited by both officials and analysts for days is the simple fact that the regime is the only entity in Syria known to have chemical weapons and the means to disperse them.

With this

By Oliver Holmes and Erika Solomon
BEIRUT | Wed Aug 28, 2013 7:59am EDT

(Reuters) - The United Nations Security Council was set for a showdown over Syria on Wednesday after Britain sought authorization for Western military action that seems certain to be vetoed by Russia and probably China.

U.N. chemical weapons experts investigating an apparent gas attack that killed hundreds of civilians in rebel-held suburbs of Damascus made a second trip across the front line to take samples. Secretary-General Ban Ki-moon pleaded for them to be given the time they need to complete their mission.

But the United States and European and Middle East allies have already pinned the blame on Assad and, even without full U.N. authorization, U.S.-led air or missile strikes on Syria look all but certain, though the timing is far from clear.

That has set Western leaders on a collision course with Moscow, Assad's main arms supplier, as well as with China, which also has a veto in the Security Council and disapproves of what it sees as a push for Iraq-style "regime change" - despite U.S. denials that President Barack Obama aims to overthrow Assad.

Uncertainty over how the escalation of the conflict at the heart of the oil-exporting Middle East will affect trade, and the world economy sent oil prices, and gold, to their highest levels in months while stocks fell. Fears over the economy of Syria's hostile neighbor Turkey pushed its lira to a record low.

Analysis & Opinion

Western powers could strike Syria within days

West mustnâ€™t rush into Syrian conflict

Related Topics

World »
Russia »
United Nations »
Syria »

Related Video

U.N. resumes Syria chemical attack probe
4:20am EDT

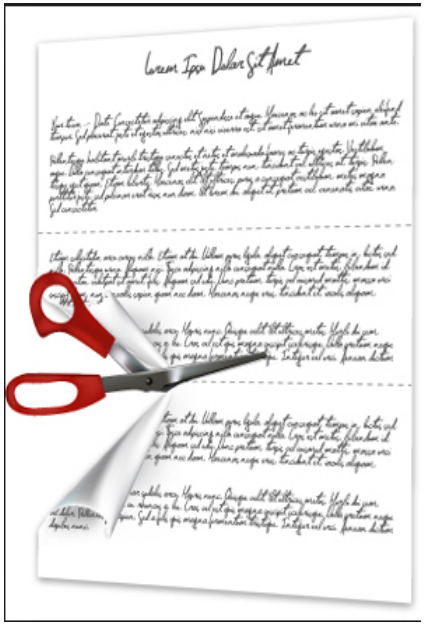
Rebels gain ground in Northern Syria

Israel will respond with force to any attack from Syria

Biden: No doubt Syrian regime used chemical weapons

Writing well for the Class

- People are more likely to read subjects/writings/emails that create curiosity or provide utility.
- When they are busy
  - Curiosity fades in importance
  - They read only the ones with practical importance ["utility"]
- So, write as if you are a staff writer (targeting for busy people) for a newspaper, and remember that you have an editor whose job is to cut your article to fit into a limited space, maybe just 1 inch in a column.
  - Important things [Conclusions and summary] in the first paragraph
    - Summary of the event/thing first so that it delivers message even though only that summary survives the "cutting"
  - Then expand your story after the First Paragraph
  - Use your own words

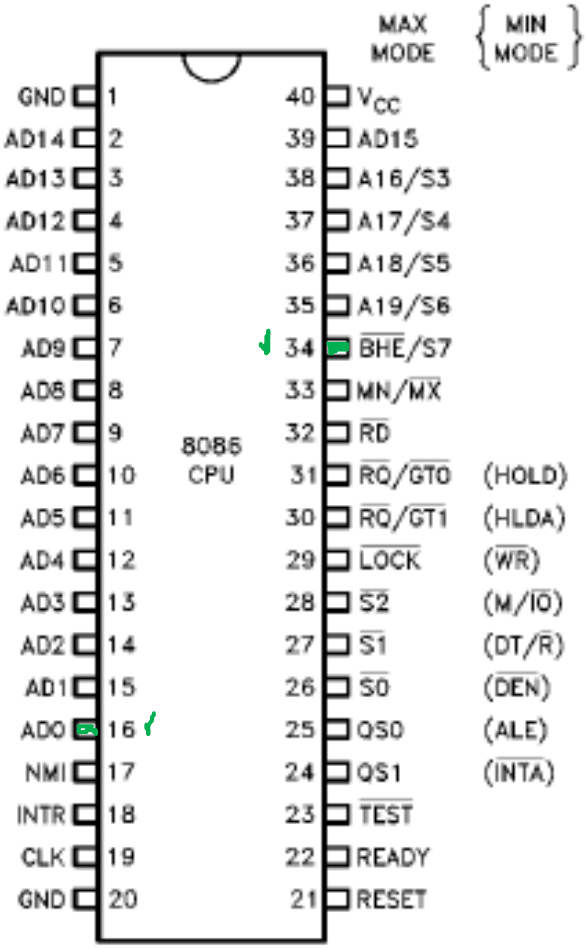


HOMEWORK - Recap

- Technical Report on "Endian-Neutral Approaches"
  - Subject
    - What are they?
    - How do they work?
    - Who are involved and leading the works
  - Format:
    - 2 - 3 pages
    - **First paragraph** must summarize the entire report (**the importance of the first paragraph – your own words** --- this determines the grade): ***you may want to write this first paragraph after completing the report.***
  - No figure, no photo, text only.
  - Submission Due: Hardcopy + Softcopy by 5:30pm Thursday Oct 13

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
    - Why? Internal 16-bit register cannot hold 20-bit address

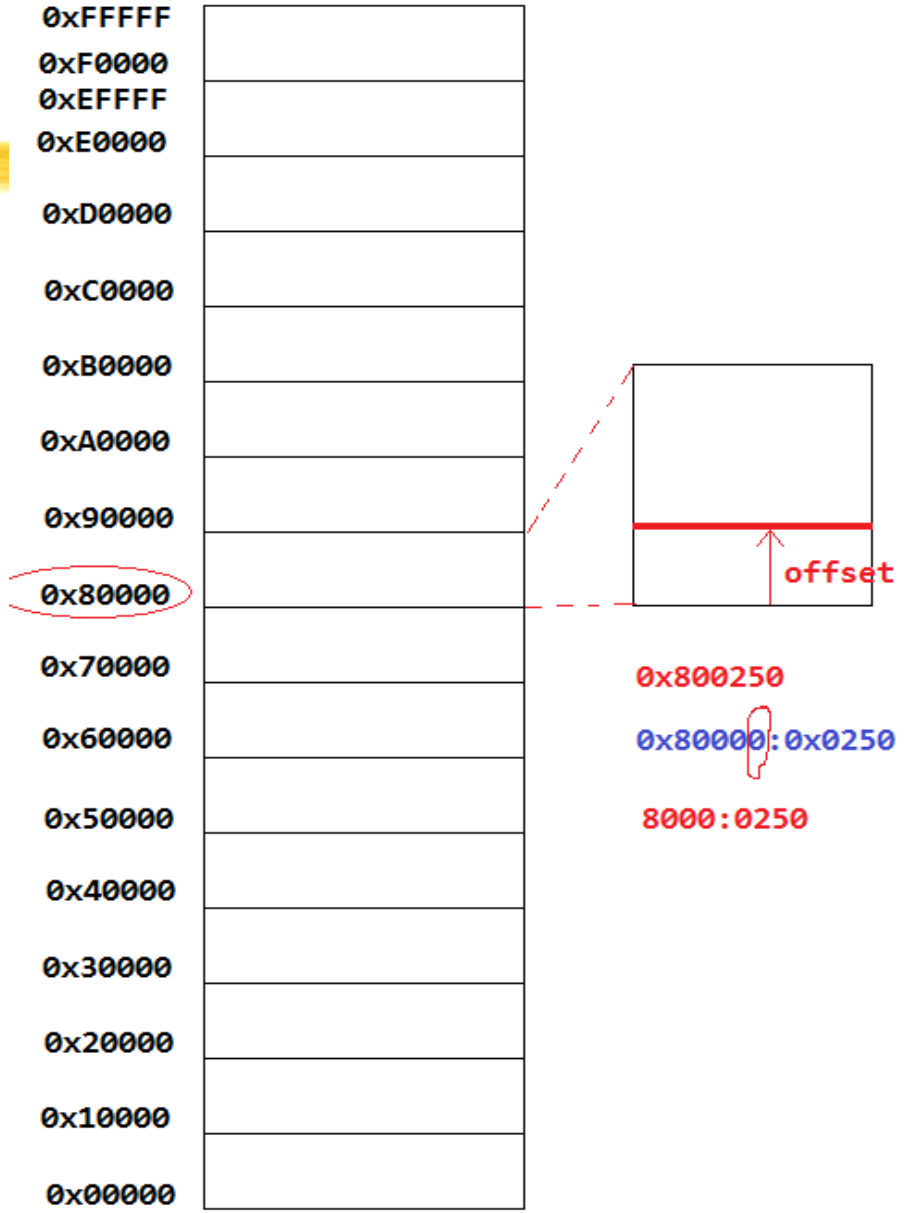


X86 Modes of Operation

- Real Address Mode (16-bit registers)
  - 1 MB Memory can be accessed 0x00000 – 0xFFFFF
  - Processor runs one program at a time
  - Application programs are allowed to access any memory location including that is linked directly to system hardware
  - MS-DOS and Windows95 and 98
  - A few extra features available for this mode by which direct access to system memory and hardware devices is possible
  - Programs running in real-address mode can cause the OS crash

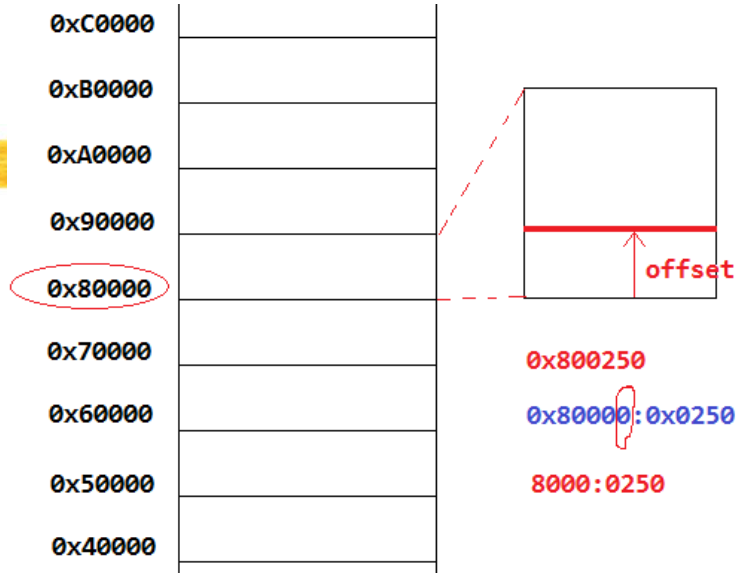
x86 Memory Management – Real Addr Mode

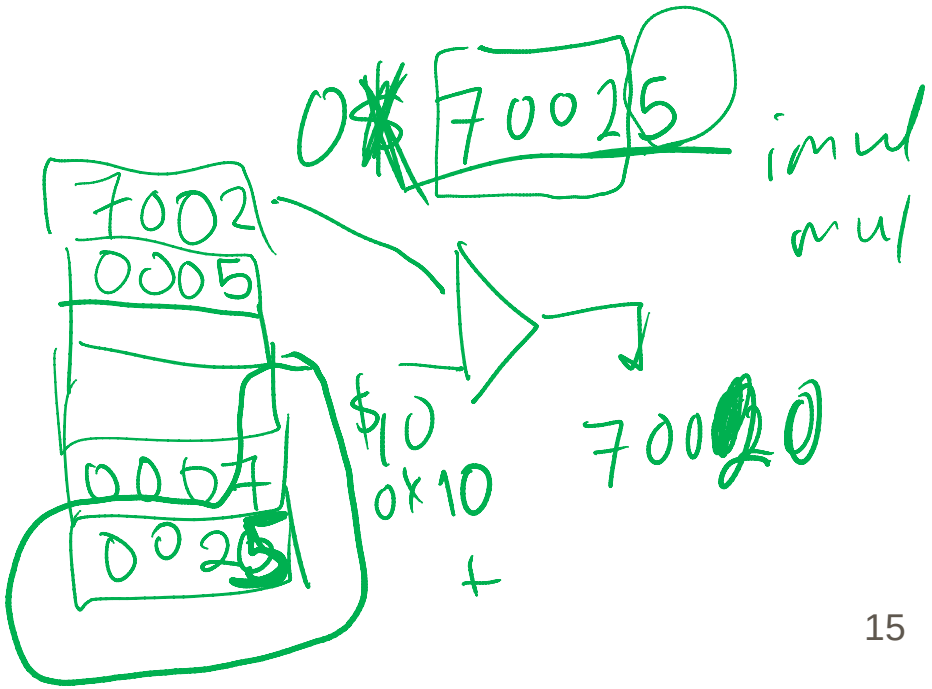
- Memory Segmentization
  - (for 8086 with 20 Addr lines) 1 MB Memory can be accessed 0x00000 – 0xFFFFF
  - But 16-bit register cannot hold the 20-bit address
  - Solution: Segmentize the 1MB memory space into multiple (16 exactly) areas (segments)
  - Accessing by **Segment** (expressed by first 4 hex digits only) **+ Offset** (max. offset = 0xFFFF)



x86 Memory Management – Real Addr Mode

- 20-bit address calculation
  - 16-bit segment value
  - 16-bit offset value
  - **Memory address = (16-bit Seg value) *0x10 + (16-bit Offset)**
  - 16-bit segment value is placed in one of the segment registers:
    - CS: Code seg register
    - DS: data seg register
    - SS: Stack seg register
    - ES, FS, GS, etc



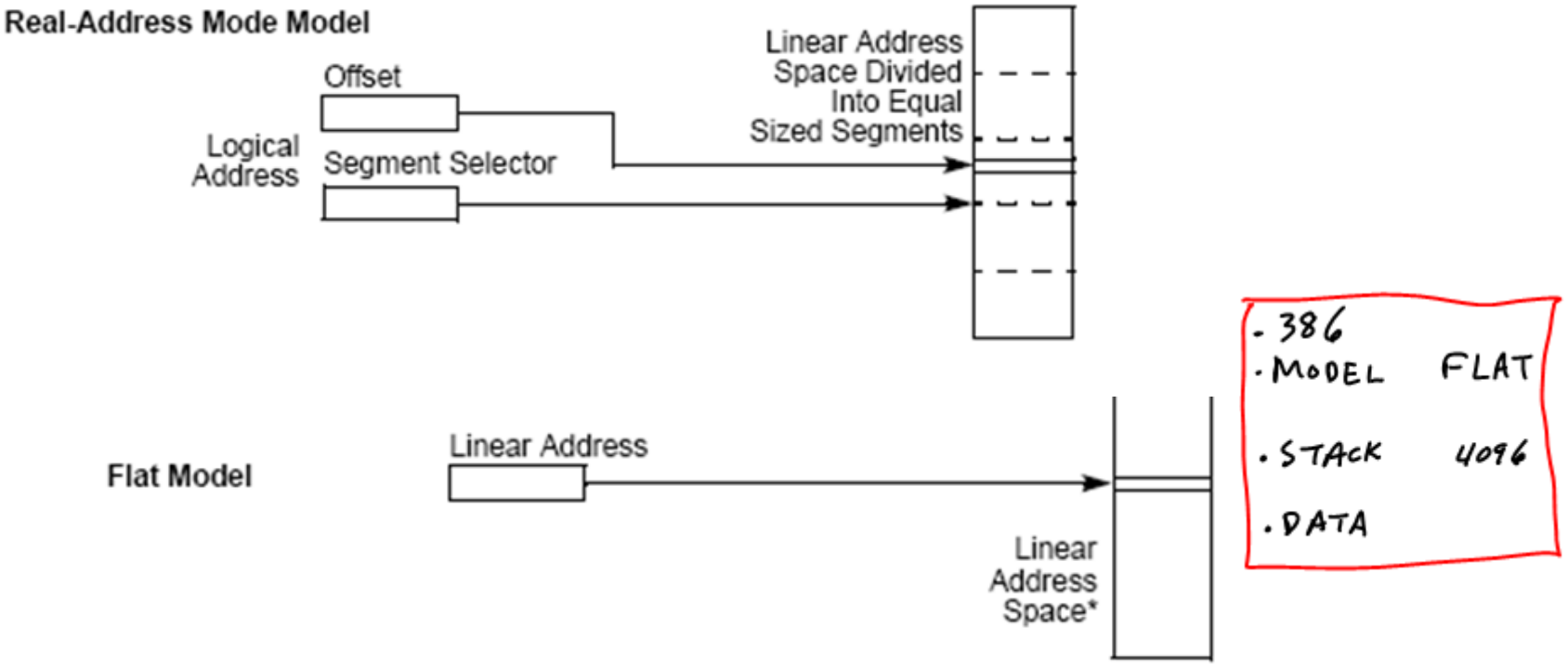


x86 Mode – Protected Mode

- Protected Mode [from i286] → with 32-bit registers
  - Native State of the Processor in which all instructions and features are available
  - Programs are given separate memory areas and the processor prevents programs from referencing memory outside their assigned segments.
  - Access to a total of 4GB Memory (for i386): 16MB for i286
  - Multiple programs run at the same time – own reserved memory area
  - Windows and Linux

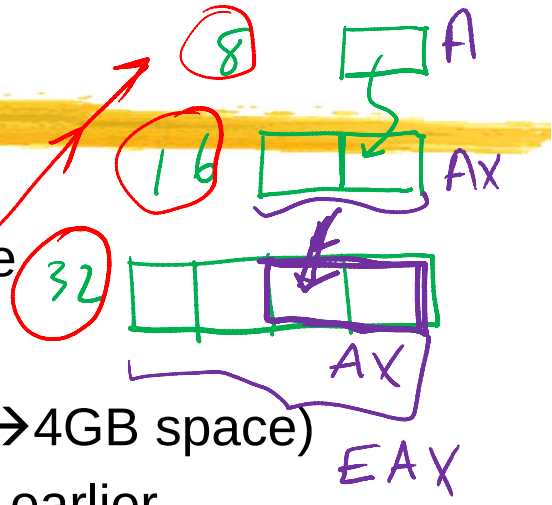
x86 Memory Management –Protected Mode

- In Microsoft Assembler (MASM)
  - "**Flat (Segment) Mode**" is appropriate for protected mode programming
  - Just one(1) 32-bit address register is enough (for up to 4 GB Memory space)
  - Actual address location calculation is done in the background



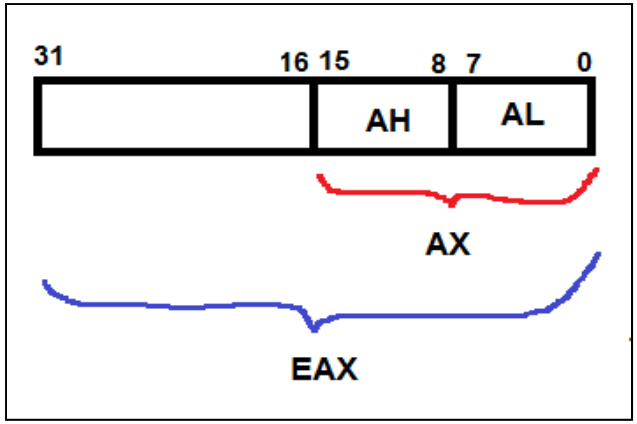
x86 Architecture

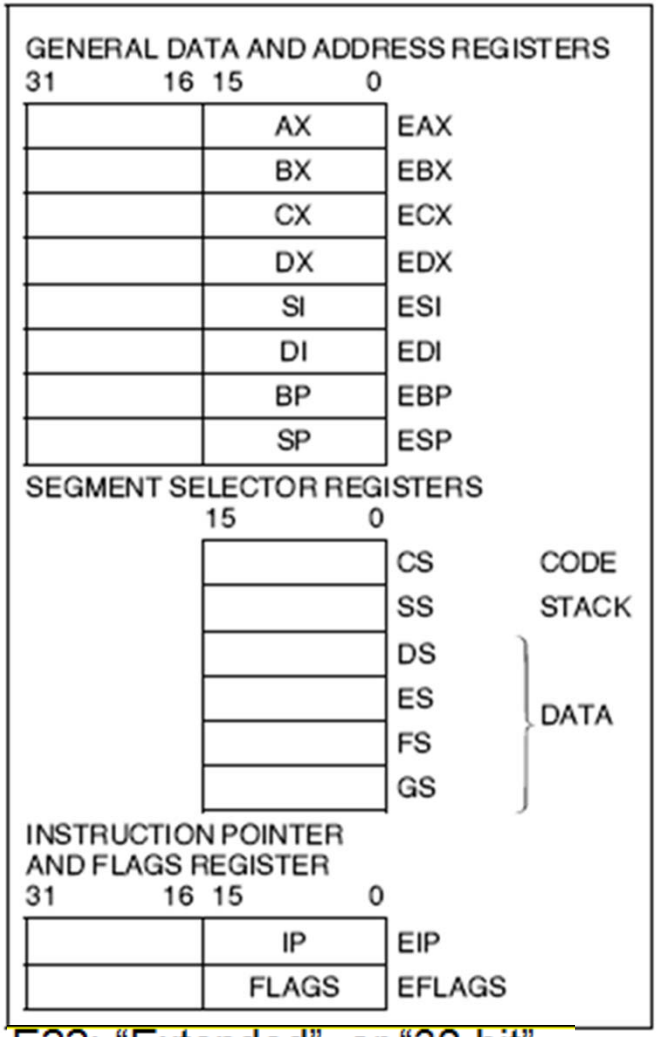
- 80286
  - "Protected Mode" first introduced
  - Segment register contents as selector or pointer
    - 24-bit base address → 16MB memory size
- 80386
  - 32-bit registers for operands and addressing(→4GB space)
  - Lower half of 32 bits is equivalent to 16 bits of earlier generations [Backward (upward) 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, Paging Unit



Overview of Basic Execution – 386 or higher

- 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

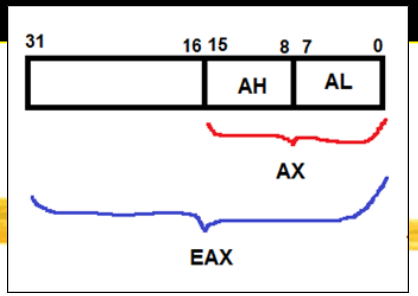




E??: "Extended" or "32-bit"
??X: "16-bit"

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.



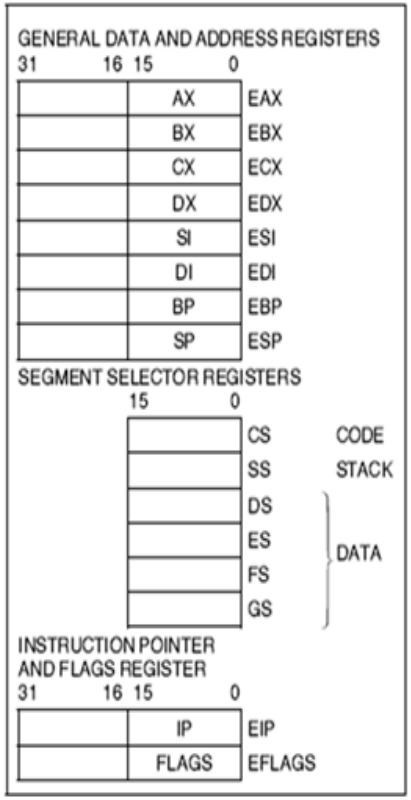


Figure 2-1. Intel386™ DX Base Architecture Registers

EFLAG Register

- 32-bit register
  - Initial state: 0x00000002
  - Contains a group of **status flags (S)**, a **control flag (C)**, and a group of **system flags (X)**

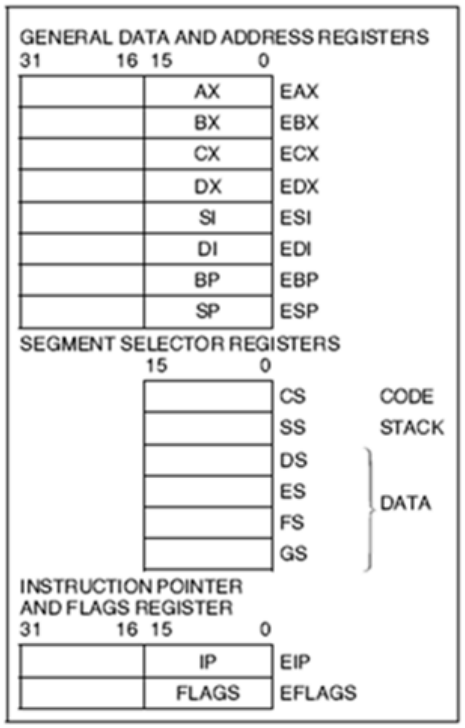
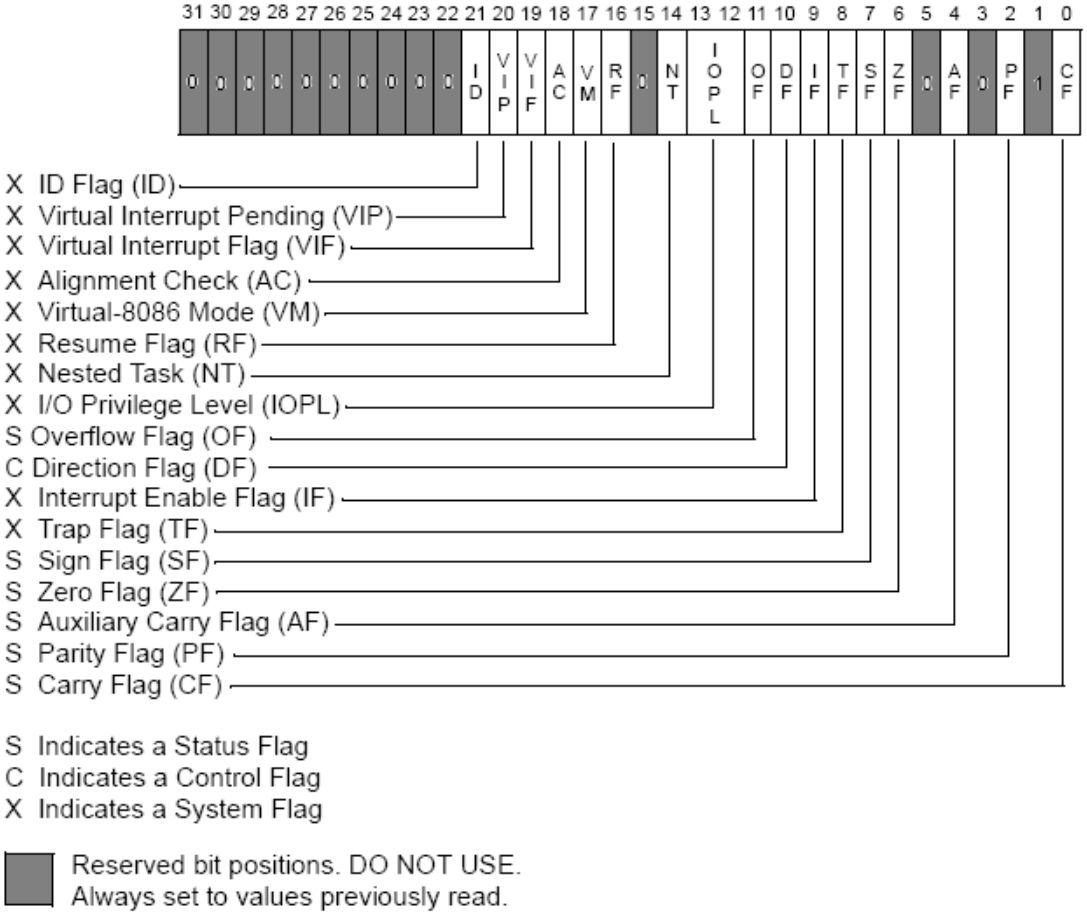
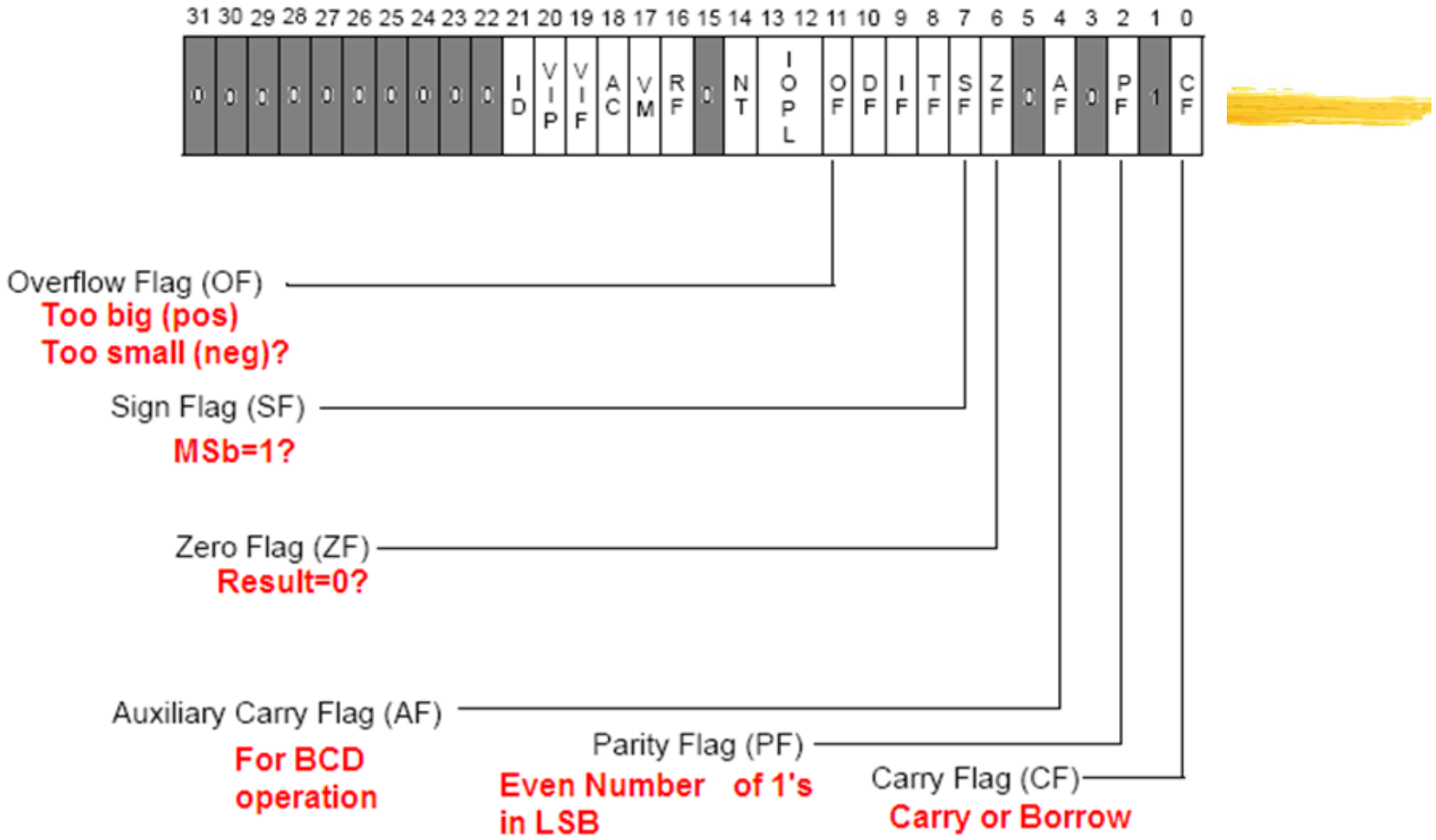


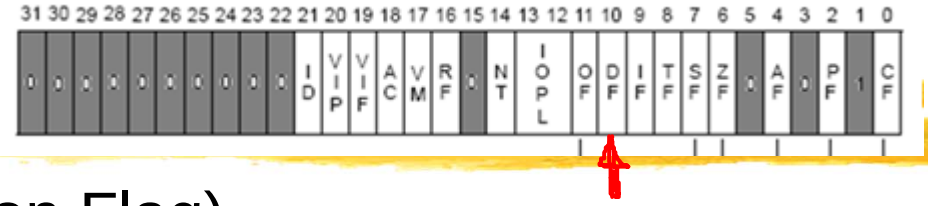
Figure 2-1. Intel386™ DX Base Architecture Registers



Status Flags

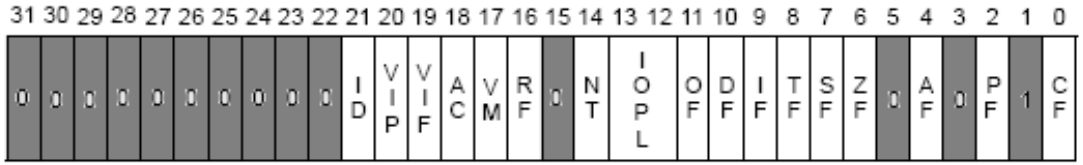


Control Flag (DF)



- 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



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

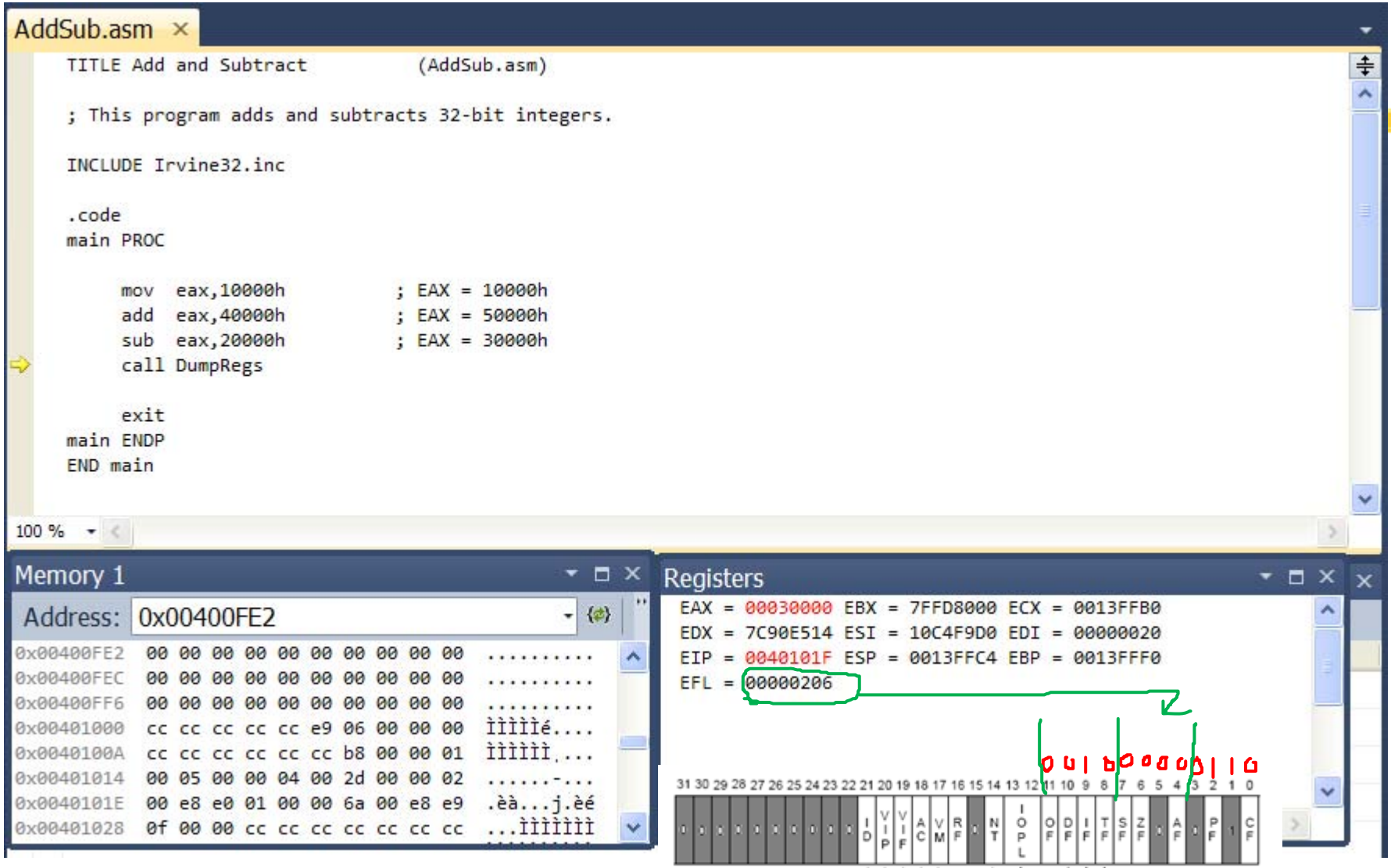
Nested Task (NT)
Task linked to another task?

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

MASM Screen Capture



Web Page for Instruction and Link Library

- **Individual Assignment:**
  - **Install a Visual Studio version in to your computer**
  - **Read the Instruction very carefully**
  - **And install the Link Library**
  - **Bring your computer to the Tuesday (Oct 11 2016) class**

EECE416 Microcomputers

Lecture 4: x86 Assembly Programming

Getting Started with MASM and Visual Studio:

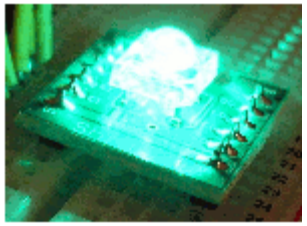
- Visual Studio 2010: Instruction (Read this first) + Link Library (.msi file. Download and save this file in your computer)
- Visual Studio 2012: Instruction (Read this first) + Link Library (.msi file. Download and save this file in your computer)
- Visual Studio 2013: Instruction (Read this first) + Link Library (.msi file. Download and save this file in your computer)

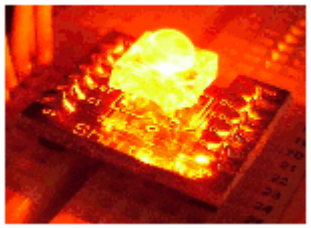
Microcomputer Project

- Week 1 (due Oct 11)
  - ISSUES – Driver Program and/or OS incompatibility
    - Mini Arduino
    - MSP430
  - SOLUTION1
    - Try to solve by Wednesday
    - If not solved, receive new type of controllers ASAP
  - SOLUTION2
    - Suggestion???

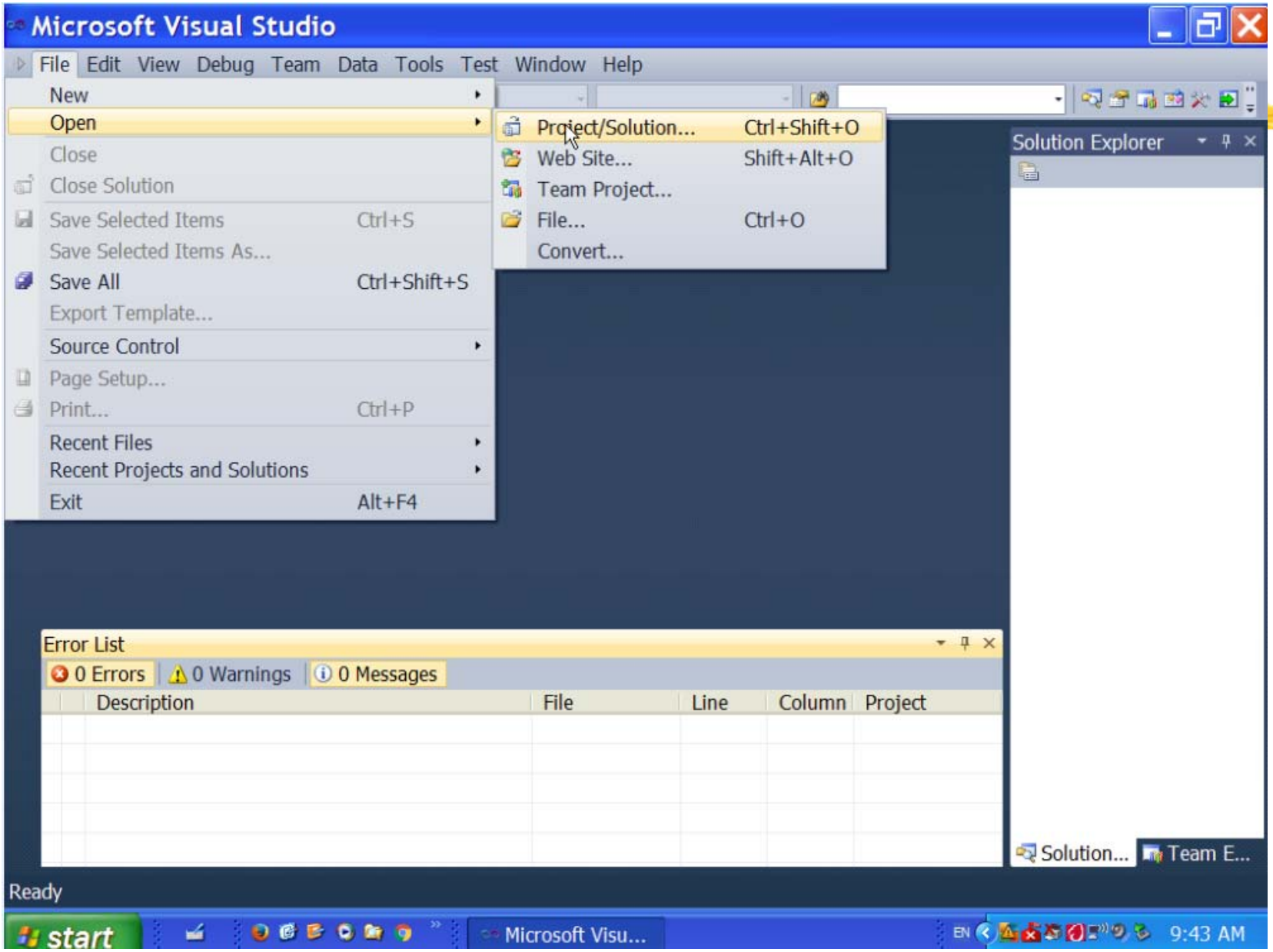
Microcomputer Project

- Week 2 (due Oct 18)
  - Connection of an RGB LED
  - Written Report – must include
    - Brief description of the project
    - Code
    - Connection Diagram
    - Screen captures or photo-shots of working system

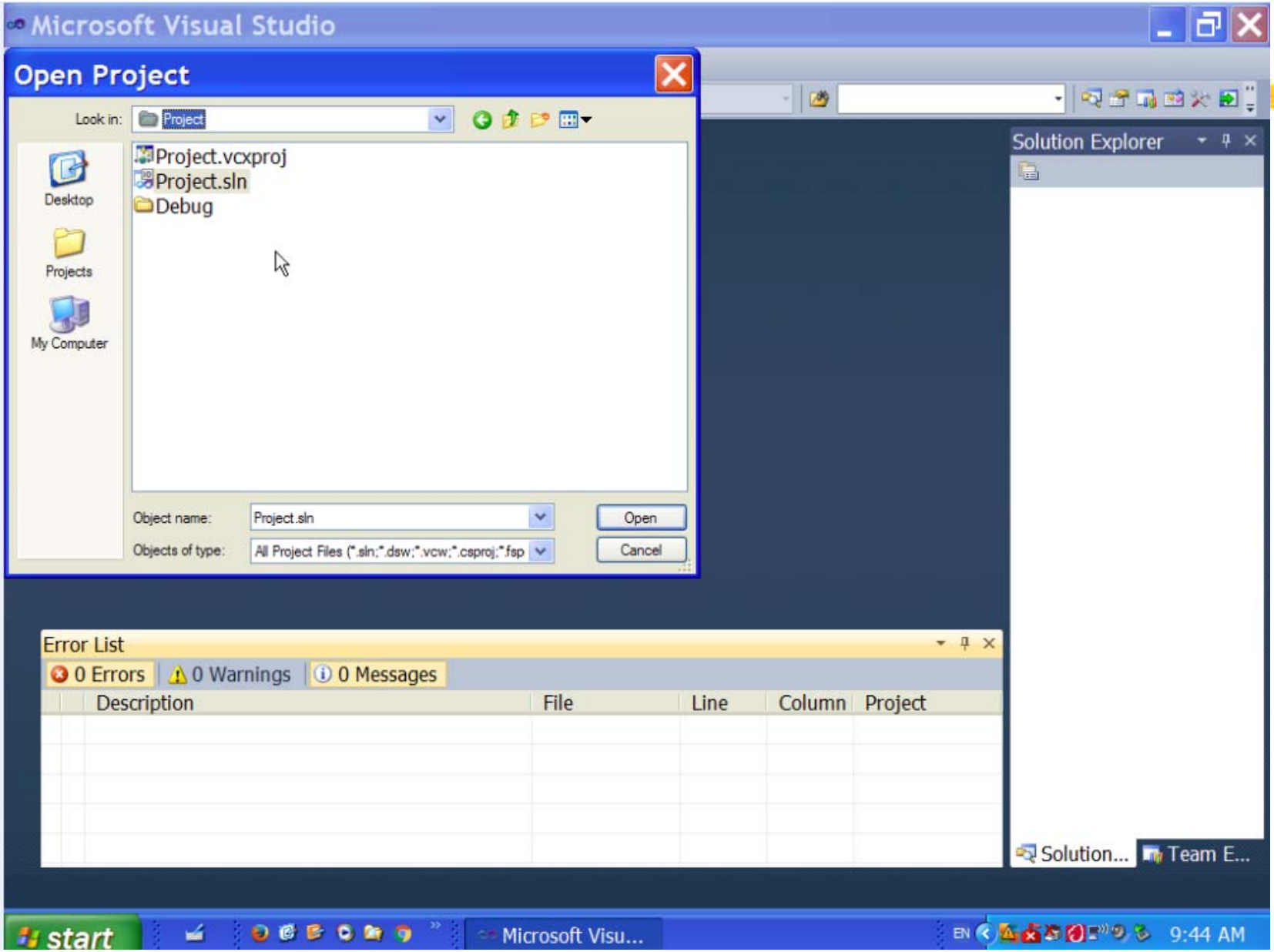




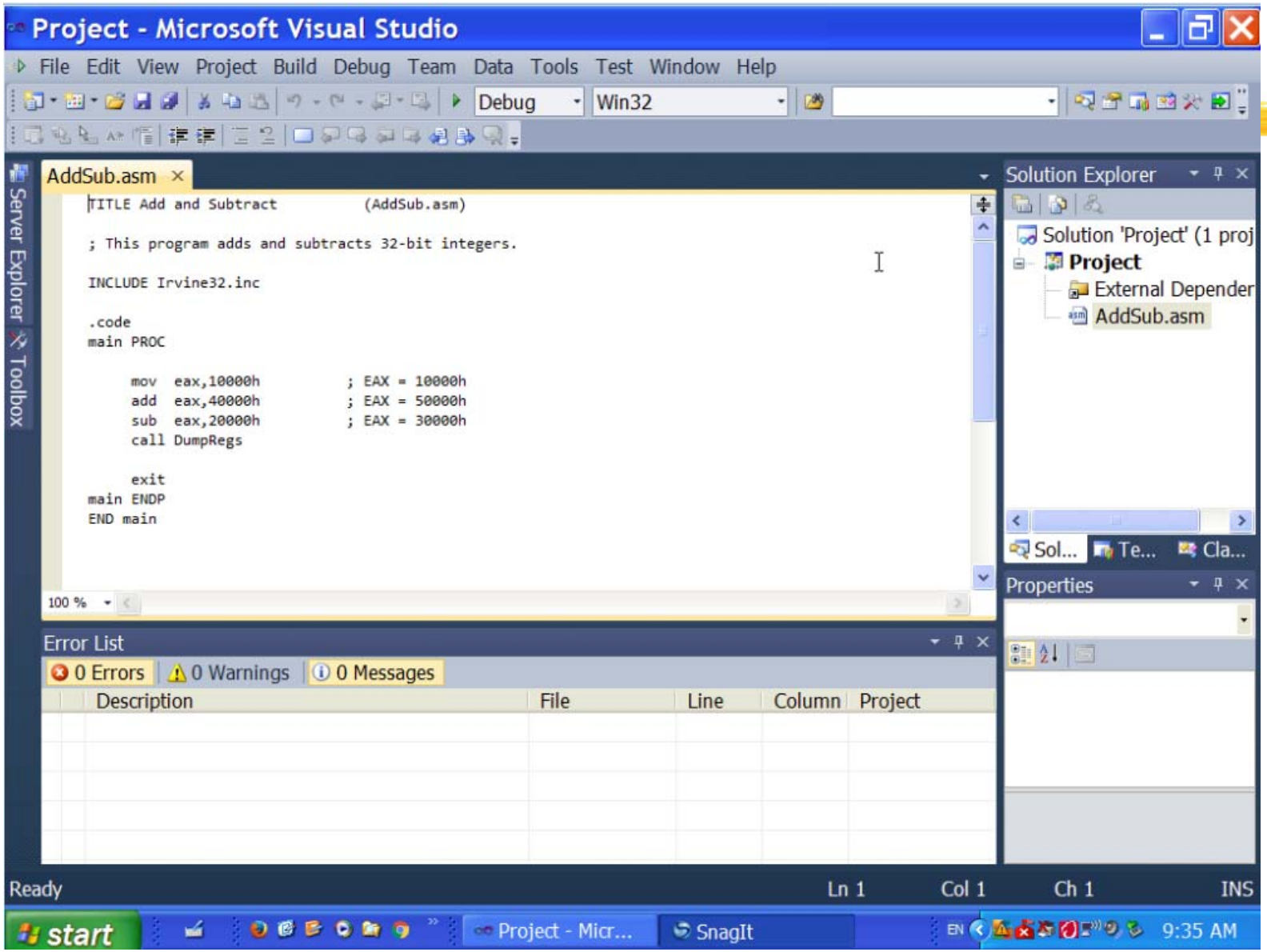
1 Sample Code and Run in Visual Studio – Open Project Solution



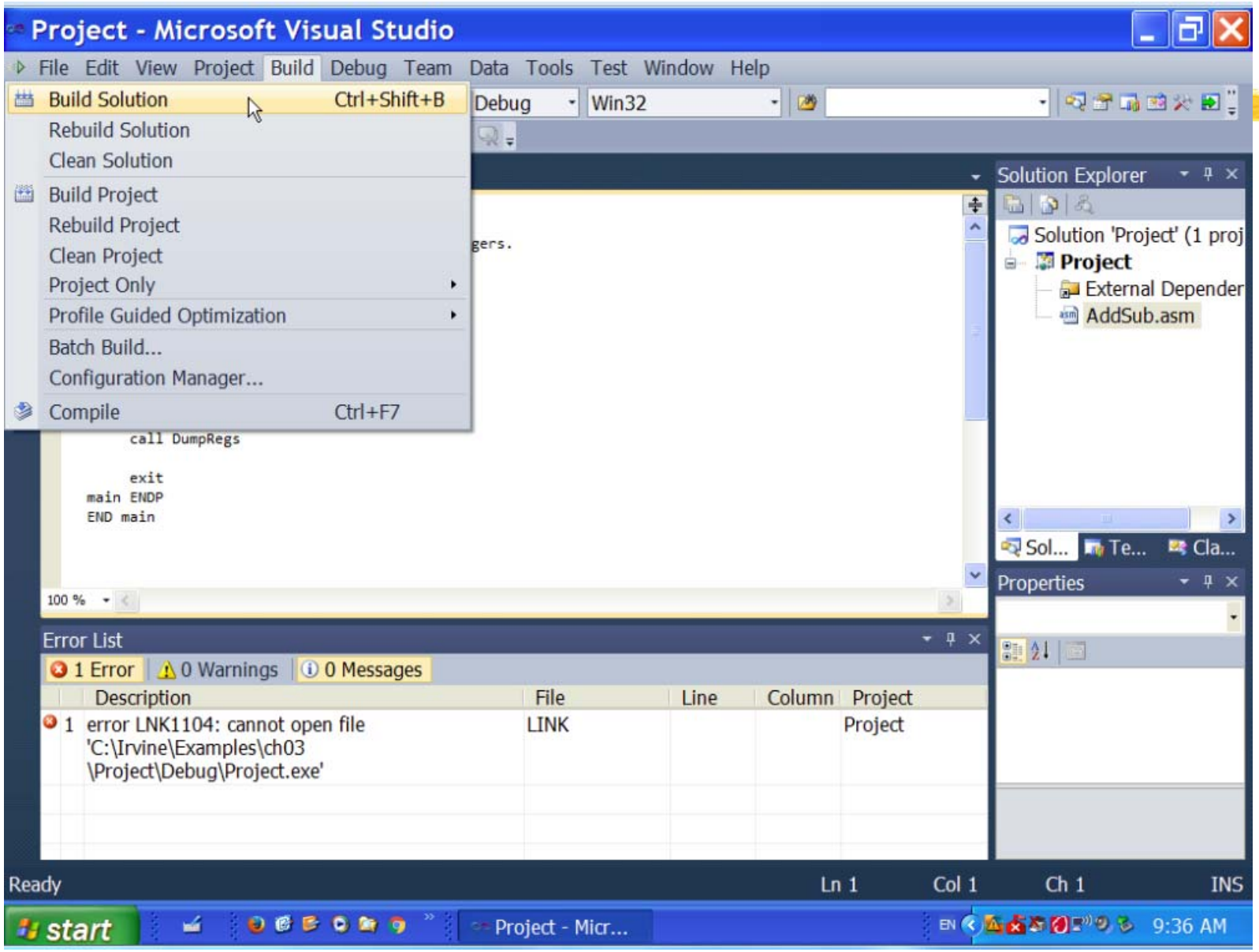
2 Sample Code and Run in Visual Studio – Select Project.sln



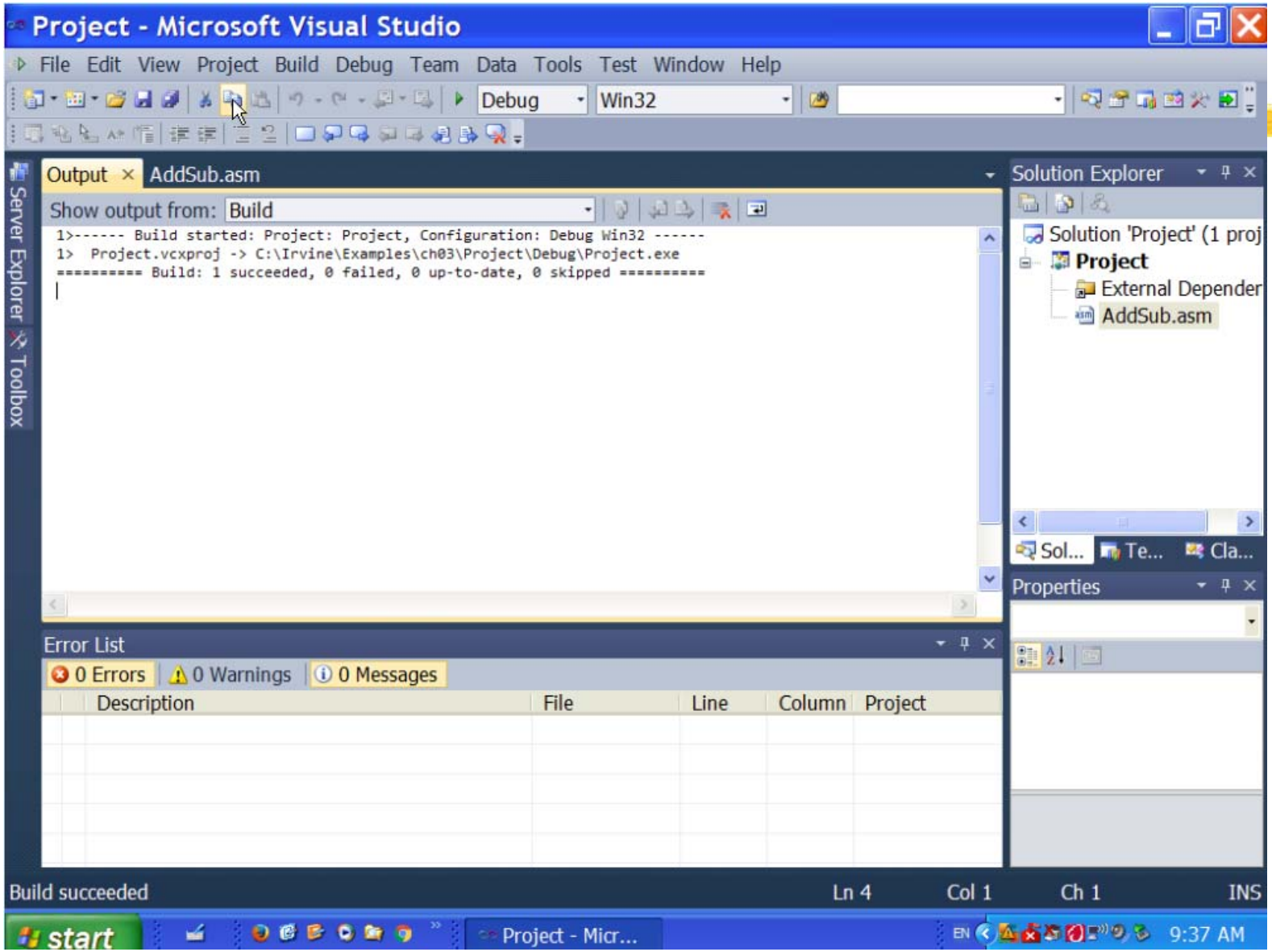
3.Sample Code and Run in Visual Studio – Addsub.asm



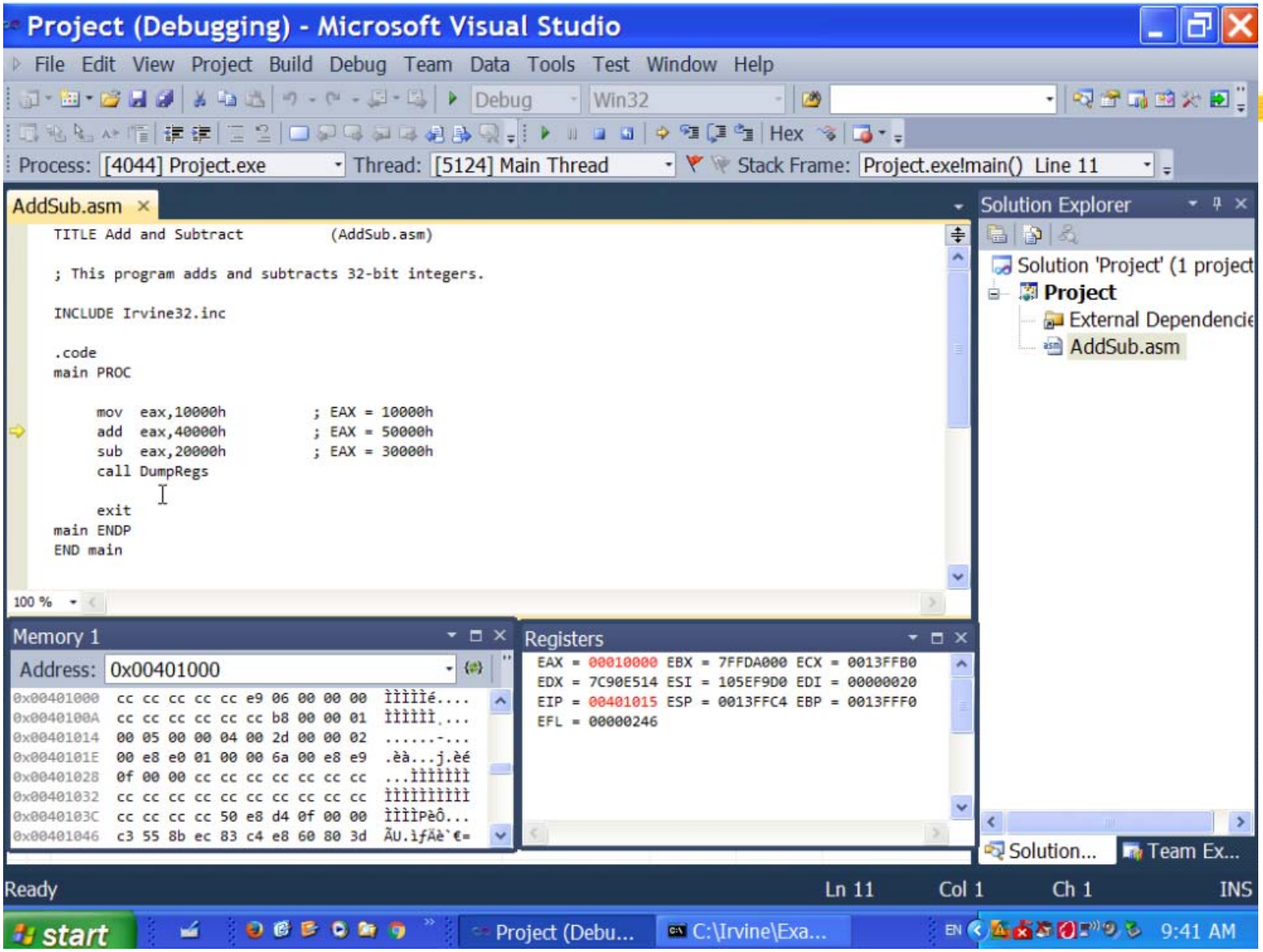
4 Sample Code and Run in Visual Studio – Build Solution/Compile



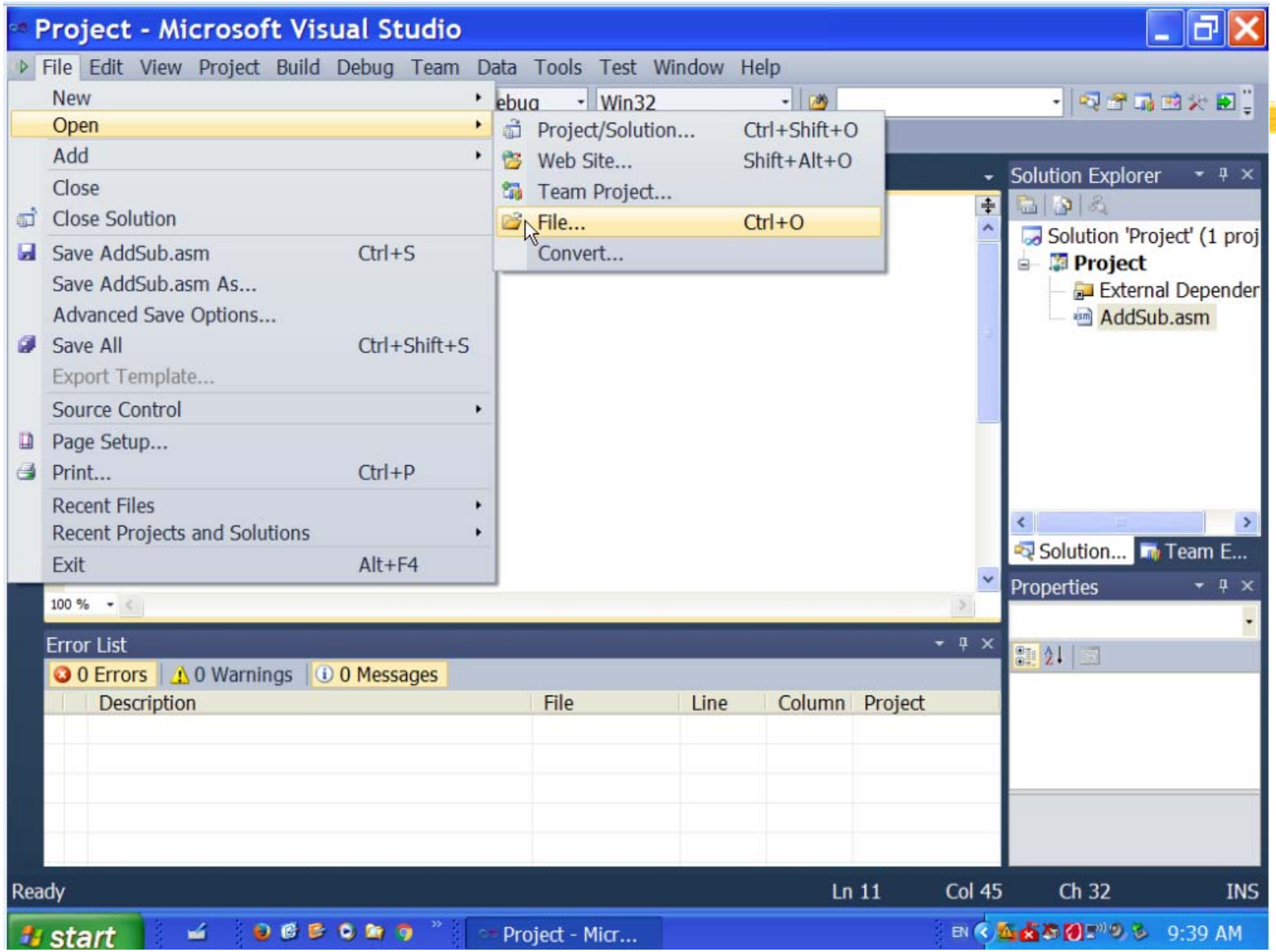
5. Sample Code and Run in Visual Studio – Build Success



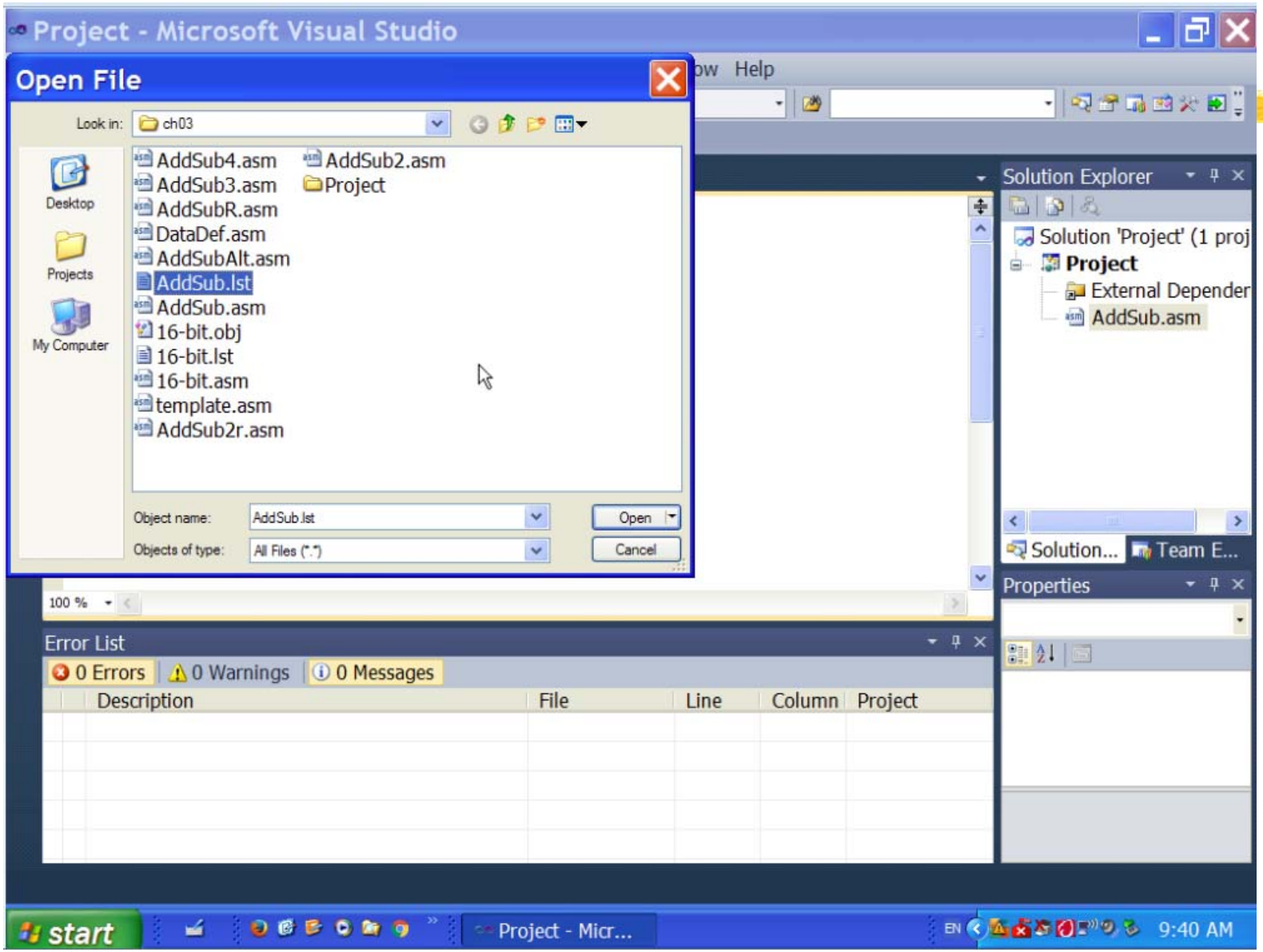
6 Sample Code and Run in Visual Studio --- Run/Debug (F10 key)



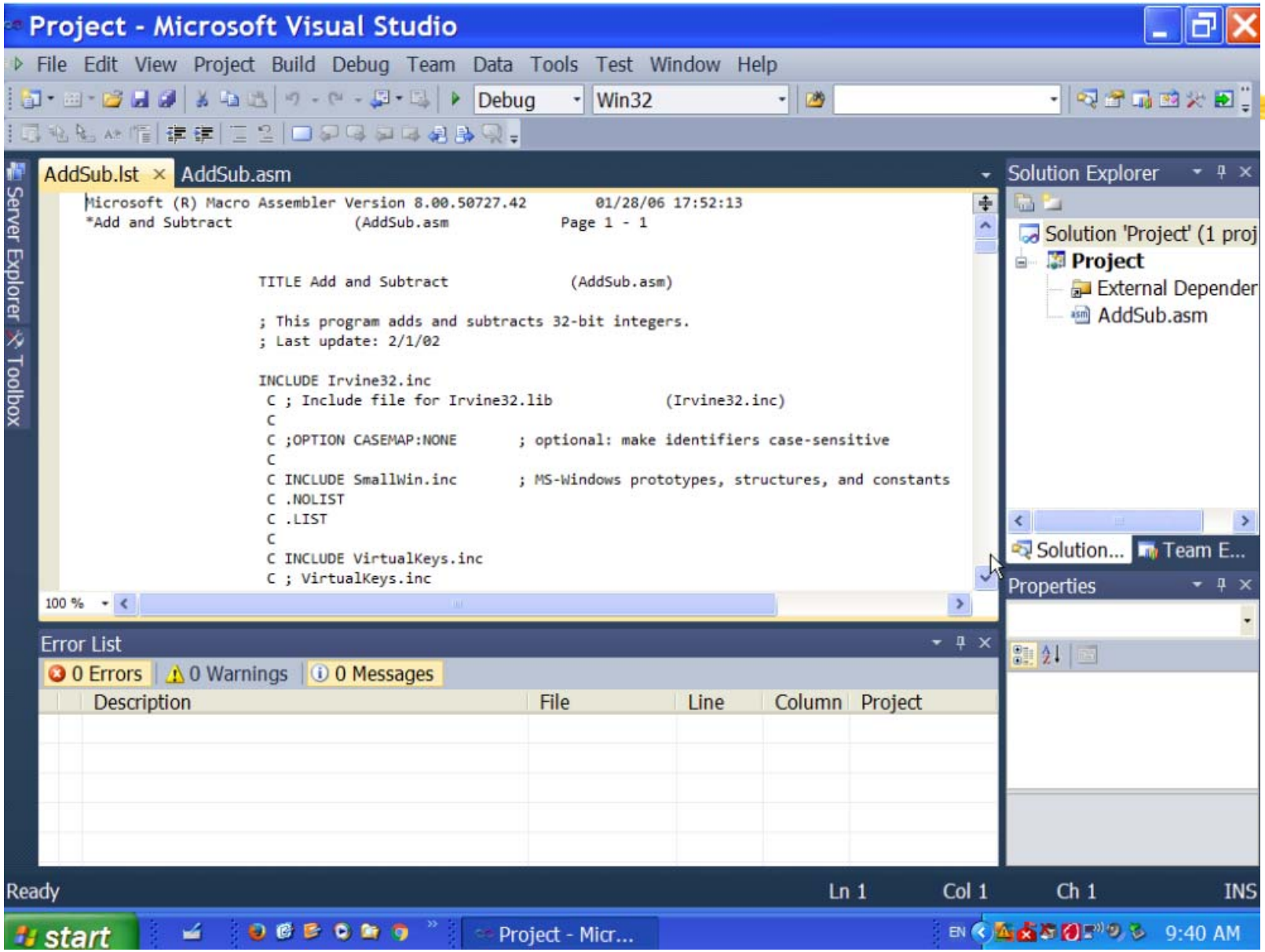
7. Sample Code and Run in Visual Studio --- LIST file open



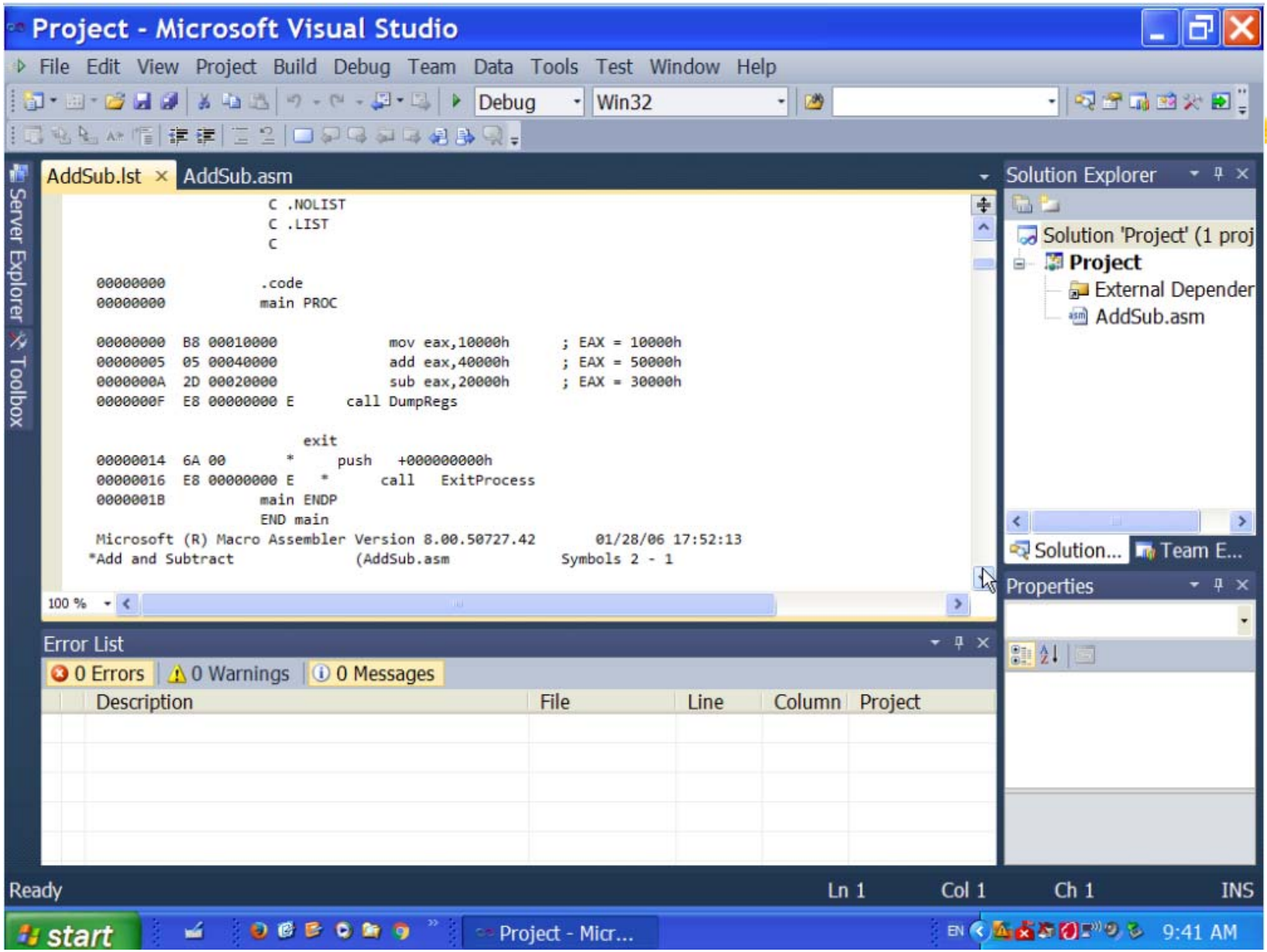
8 Sample Code and Run in Visual Studio --- .LST file



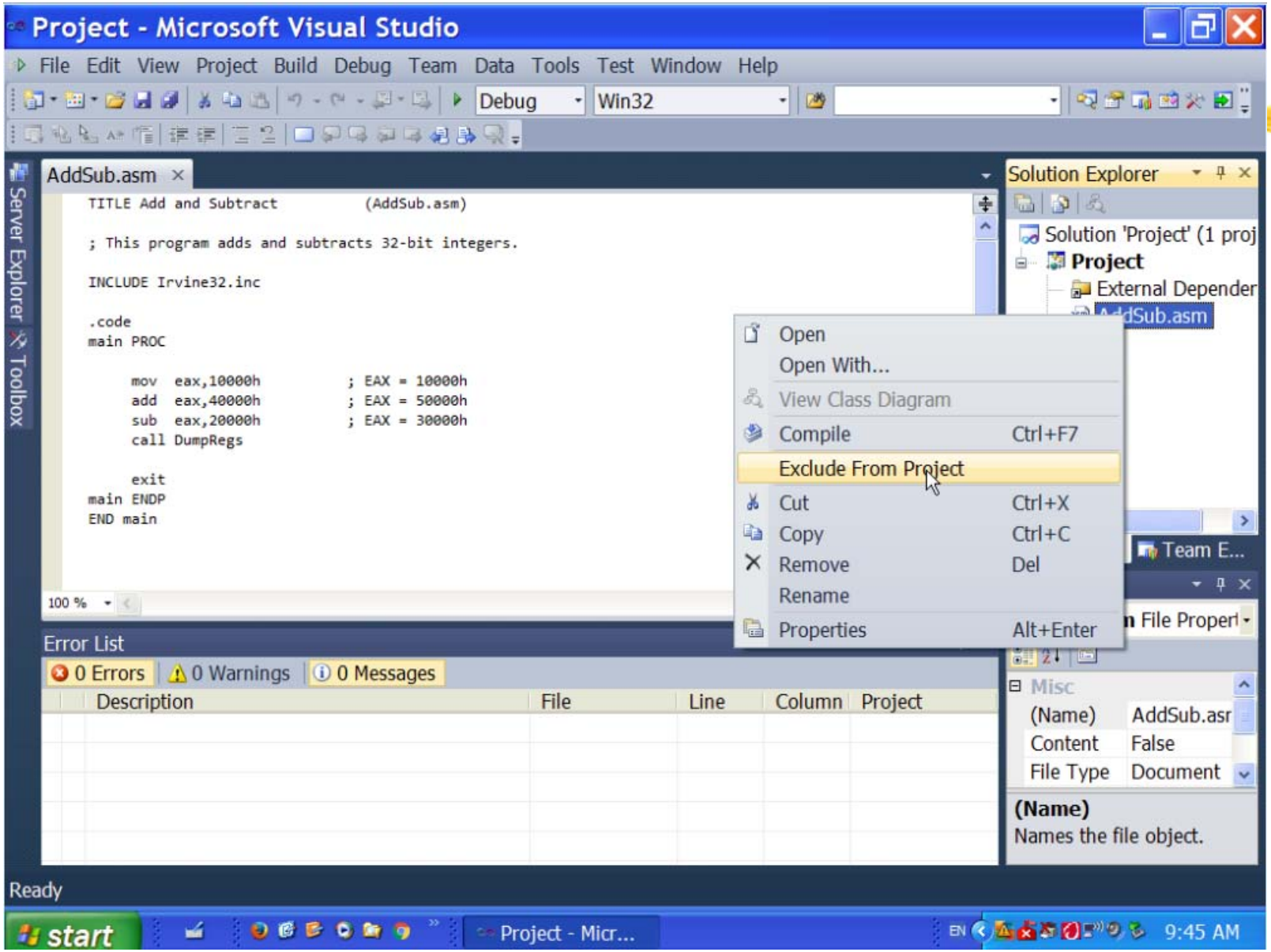
9 Sample Code and Run in Visual Studio -- .lst



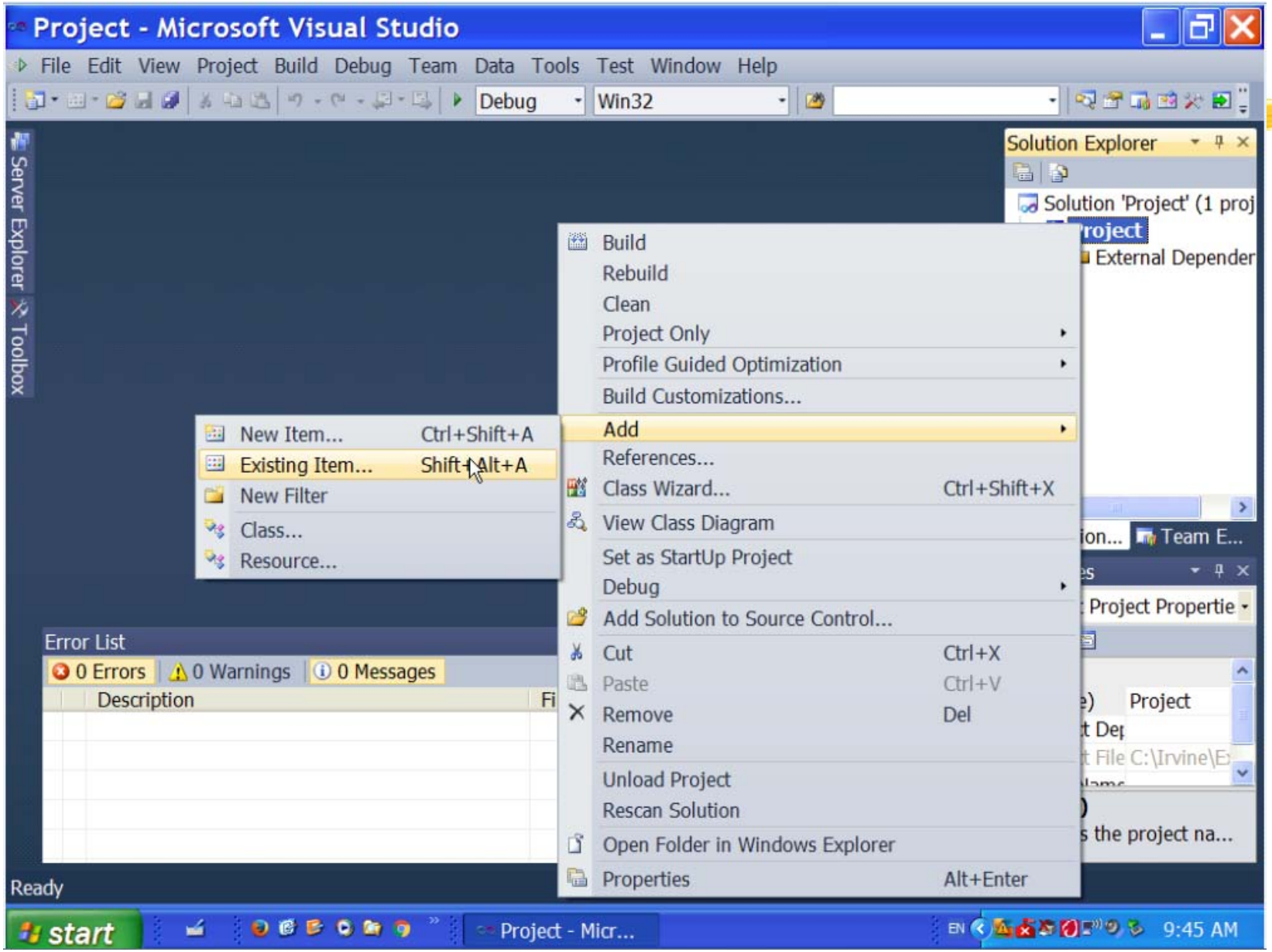
10 Sample Code and Run in Visual Studio --- .lst



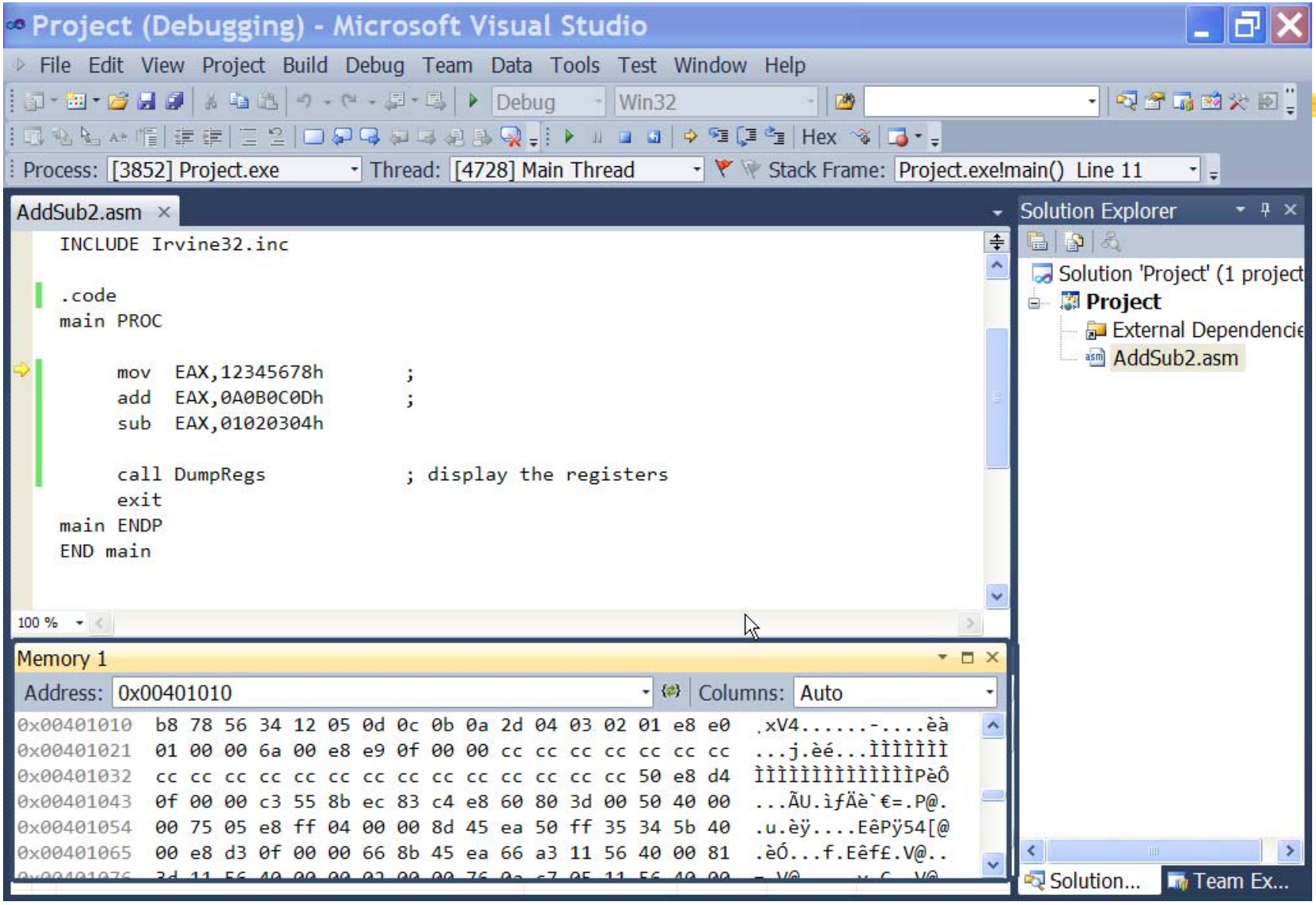
11 Sample Code and Run in Visual Studio – When a new code is tested



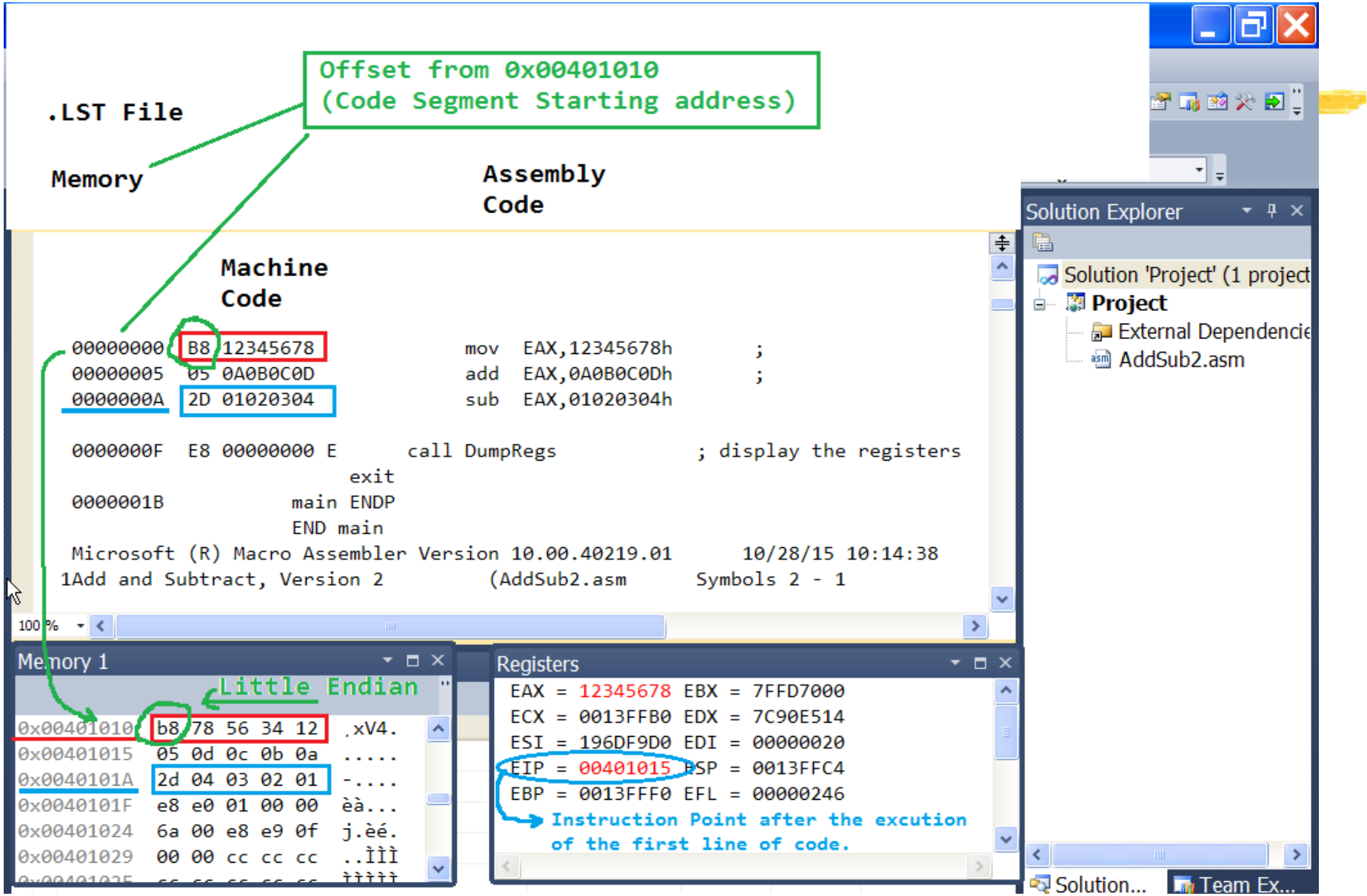
12 Sample Code and Run in Visual Studio --- Read in the new code



13 Sample Code and Run in Visual Studio --- New code

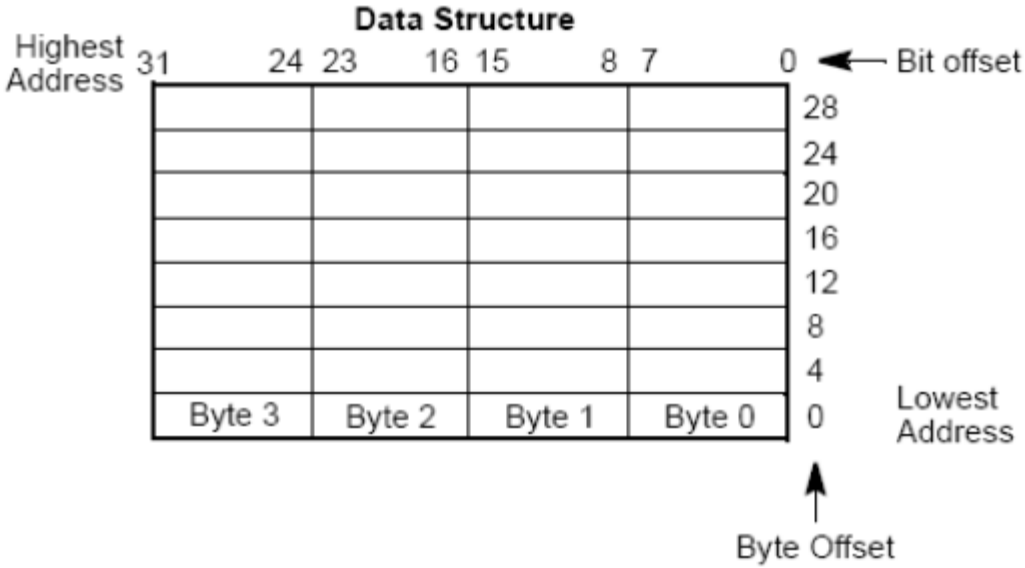


14.LST File and The Memory Contents of Code



Notational Conventions

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

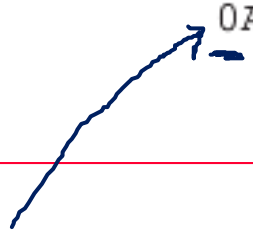


Assembly Language Fundamentals

- Integer Constants
  - [{+/-}] digits [radix]
  - Radix
    - H hexadecimal
    - q/o Octal
    - D Decimal {Default}
    - b Binary
    - Example:

```
main PROC
     mov   eax,5                ; move 5 to the EAX register
     add   eax,6                ; add 6 to the EAX register
     call  WriteInt             ; display value in EAX
     exit                       ; quit
main ENDP
```

26	Decimal	42o	Octal
26d	Decimal	1Ah	Hexadecimal
11010011b	Binary	0A3h	Hexadecimal
42q	Octal		



- *Note: Hex constant beginning with a letter must have a **leading 0**

Assembly Language Fundamentals

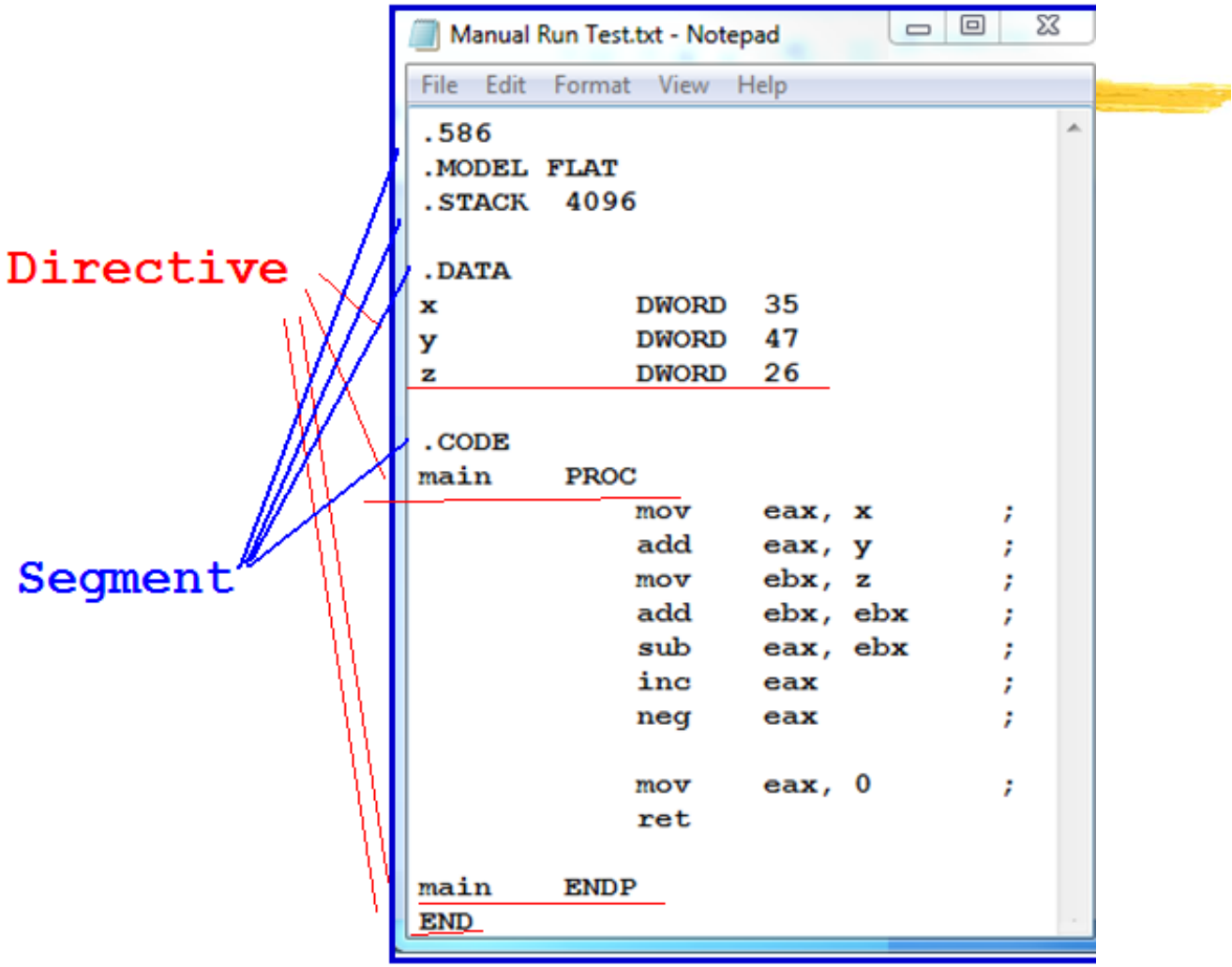
- Character Constants
  - A single character enclosed in single or double quotes
    - "A"
    - "d"
  - MASM stores the value in memory as the character's binary ASCII code
- String Constants
  - A sequence of characters (including spaces) enclosed in single or double quotes
    - 'ABC'
    - "Good night, Gracie"
    - 'Say "Good night," Gracie'

Assembly Language Fundamentals

- Directives
  - A command embedded in the source code that is recognized and acted upon by the assembler
  - Directives can define **variables, macros, and procedures**
  - Directives can assign names to **memory segments**
  - Directives do **not** execute at runtime
  - Directives are **case insensitive** in MASM
- Defining Segments (or Program Sections)
  - **.data** ; the area of program containing variables
  - **.code** ; the area of a program containing executable instructions
  - **.stack** ; the area of a program holding the runtime stack (with its size)

Assembly Language Fundamentals

- Directives
  - **Variables**
  - **memory segments**
- Defining Segments
  - **.data**
  - **.code**
  - **.stack**



Assembly Language Fundamentals

- **Instruction**:
  - "A statement that becomes executable when a program is assembled" – Instructions are translated by the assembler into machine language bytes, which are loaded and executed by the CPU at runtime
- Instruction Format
  - **Label, Instructional mnemonic, argument1, argument2, argument3**
  - **Label:** Identifier (followed by a colon)
  - **Mnemonic:** a reserved name for a class of instruction op-codes which have the same function
  - **Operands (arguments)**:
    - 0 to 3 operands
    - 2 types of form: literals (i.e., number) or identifiers for data items.

Instruction Format

- When two operands are present in an arithmetic or logical instruction
  - the right operand is the **source (src)** and
  - the left operand, the **destination (dst)**

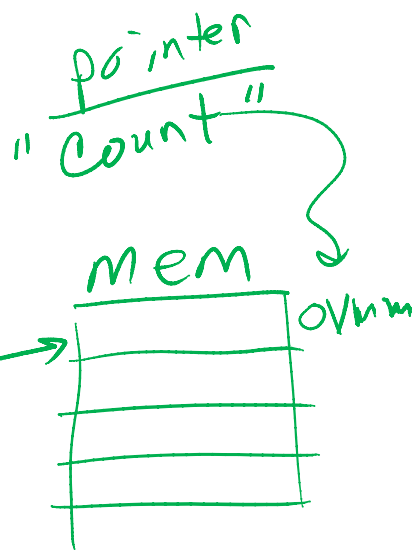
- **Example:**

```
Count     DWORD     100              ; data label
LOADREG:  MOV       EAX, SUBTOTAL
;label   mnemonic dst, src
```

Assembly Language Fundamentals

- Instruction Mnemonic: a short word that identifies an instruction
- Example

```
mov     Move (assign) one value to another
add     Add two values
sub     Subtract one value from another
mul     Multiply two values
jmp     Jump to a new location
call    Call a procedure
```



- Example

```
inc  eax                ; add 1 to EAX

mov count,ebx           ; move EBX to count

imul eax,ebx,5
```

EBX is multiplied by 5, and the product is stored in the EAX register.

Assembly Language Fundamentals

- Comments
- Single Line Comments (;)
- ```
  ; This line is a comment
  ```
- Block Comments: Begin with the COMMENT directive and a user-specified symbol

```
COMMENT  !
     This line is a comment.
     This line is also a comment.
!


COMMENT &
    This line is a comment.
    This line is also a comment.
&
```
```

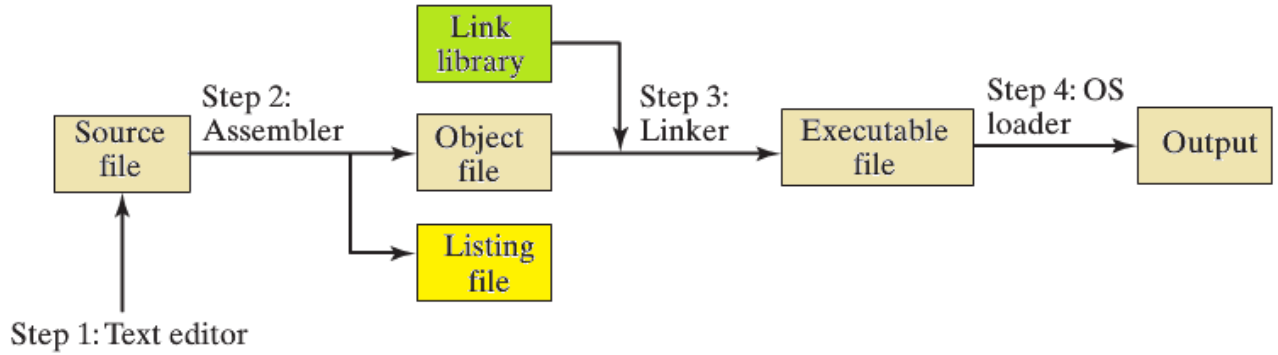
Assembly Language Fundamentals

- NOP (No Operation) Instruction
  - Takes up 1 byte of program storage
  - Does not do any work
  - Use when to align code to even-addressed boundaries (which, in x86, loads data more quickly.)

```
00000000  66 8B C3      mov ax,bx
00000003  90            nop                 ; align next instruction
00000004  8B D1         mov edx,ecx
```

Assembling and Running (Debugging) Programs

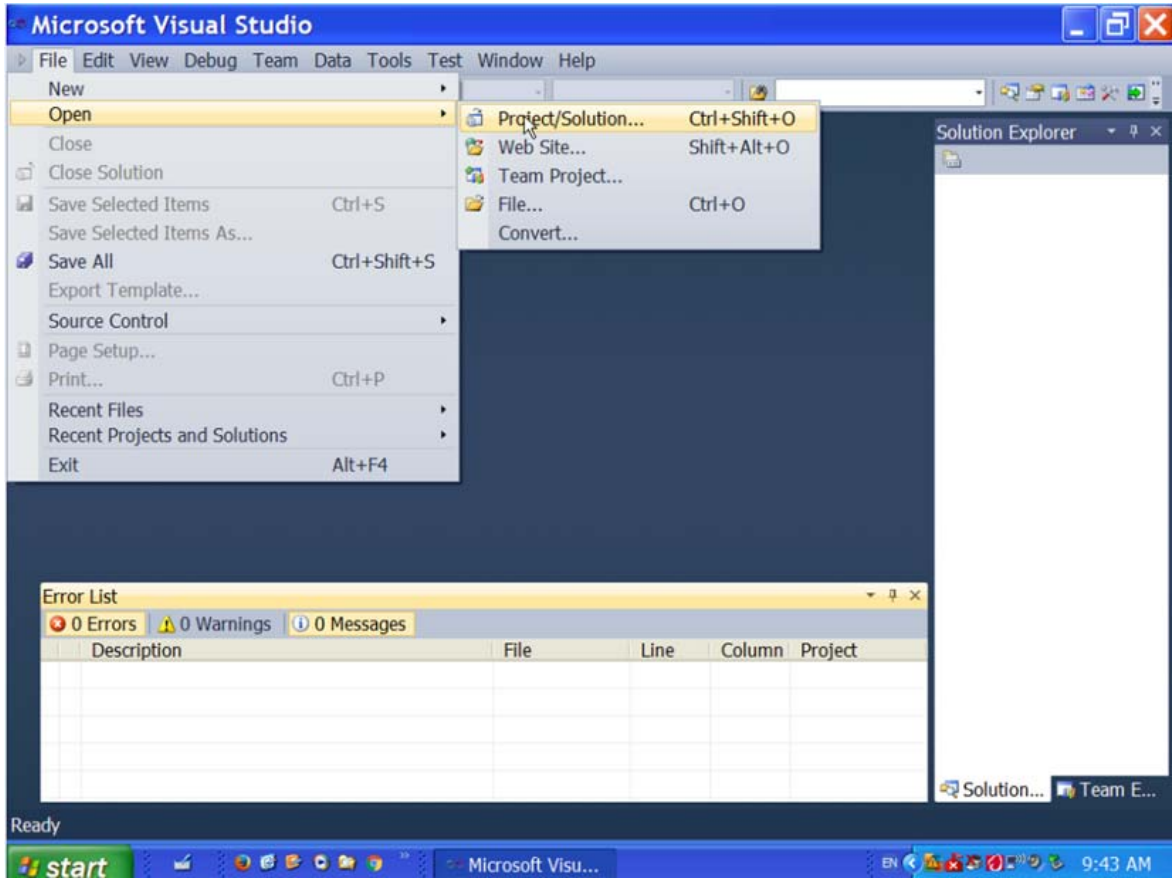
Assemble-Link-Execute Cycle



- 1: Create an ASCII text file named *source file* using a text editor
  - *__NOTE: Starting from an existing project (and source file) and revising and saving as a new .asm file is most highly recommended__
- 2: Assemble the source file. The assembler reads the source file and produces an *object file*, a machine language translation of the program. Optionally, it produced a *listing file*. {VS: BUILD SOLUTION}
- 3: Execute the program (by Debug)

Assembling and Running Programs in Visual Studio

- Step 1: Open an existing project and Revise and Save as a new file name using
- Step 2: Assemble the source file (by ***Build***).
- Step 3: Debug (F10 key for line execution)



Assembly Language Fundamentals - Summary

- Adding and Subtracting Integers
- TITLE directive
- Line comment
- INCLUDE directive

- .code directive
- PROC directive – start

- CALL a procedure
- Exit --- halt to program (Not a MASM keyword, but of Irvine32.inc)
- ENDP directive – end
- END directive – last line to be assembled
- Output (DumpRegs)

```
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
```

```
EAX=00030000  EBX=7FFDF000  ECX=00000101  EDX=FFFFFFFF
ESI=00000000  EDI=00000000  EBP=0012FFF0  ESP=0012FFC4
EIP=00401024  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1
```