X 86 machine instructions assembly

Cs342 March 8, 2021

Computer Science Department, CCNY

Instructor: Professor Izidor Gertner

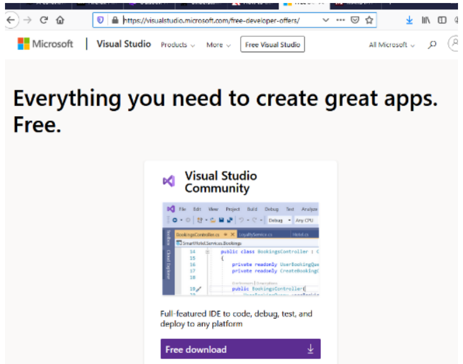
Instruction Set Architectures

1. In the course we study X86 instruction set architecture

The windows in this presentation were obtained using

MS 32 bit compiler and debugger for intel i7 processor

https://visualstudio.microsoft.com/free-developer-offers/



Everything you need to create great apps. Free.

Visual Studio Community

Full-featured IDE to code, debug, test, and deploy to any platform

In following lectures we will study:

2. MIPS instruction set architecture (described in the textbook Chapter2) using MARS simulator

- https://courses.missouristate.edu/KenVollmar/MARS/download.htm

https://courses.missouristate.edu/KenVollmar/MARS/do
Missouri State
UNIVERSITY
Search
a b c d e f g h i j k l m n o p q r s t u v w x y z
Home
Features
Download
License
Papers
Help & Info
Contact Us
Download MARS

V4.5, Aug. 2014 (jar archive including Java source code)

Note: Is your MARS text unreadably small? Download and use a new release Java 9, which contains a fix to automatically scale and size AWT and Swing components for High Dots Per Inch (HiDPI) displays on Windows and Linux. Technical details.

Previous MARS version: MARS v4.4, Aug. 2013

3. Intel I7 x64 Assembly, and instruction set architecture

Using 64 bit LINUX, 64 bit compiler gcc, debugger gdb
for intel i7 processor.

Note:IN ITEMS 1. and 3. IS THE SAME TARGET PROCESSOR I7.

We are asking the following questions:

1. What happens during
   a. Compilation time?
   b. Linking?
   c. Loading program to memory?
2. What happens when the program runs
   a. Run-time, execution-time?

Implementation of Important Concepts

1. **Stack** is a data structure that its logic is LIFO (last in first out) implemented operations: push, pop.
   1. In order to use the stack we need a special hardware: EBP, ESP registers (X86 ISA); FP, SP (MIPS,ARM)
   2. EBP – Special register used to store the address of the base of the stack (base/frame pointer). rbp in 64 bit cpu.
   3. ESP – Special register used to store the address of the top of the stack (stack pointer). rsp in 64 bit cpu.
2. **Scope**
3. **Static variables**
4. **Local, Automatic variable** – allocated in dynamic memory when entering scope and deallocated when we leave the scope.

Addressing

Memory address is a reference to a specific memory location displayed and manipulated as fixed length **UNSIGNED INTEGERS.**

Relative addressing mode - a very efficient way accessing memory.
At *execution time* the processor does the following:

- Accesses processor register Base Pointer (EBP), or Frame pointer (FP) (address of the bottom address of the stack)
- Gets the offset from the machine instruction,
- Calculates the effective address of specific memory location by adding the offset to EBP.

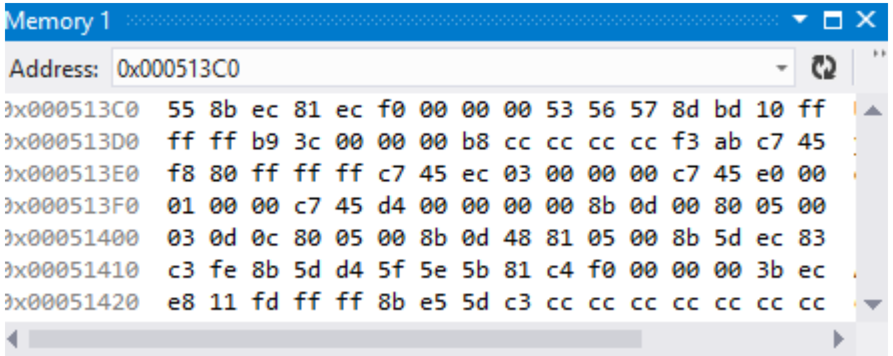
Absolute addressing mode -specifies explicitly the address.

Effective address. It is very efficient because addresses are 32 bit, while offset can be for example 8 bit.

Little and Big Endian Mystery **https://www.geeksforgeeks.org/little-and-big-endian-mystery**
Address of 32 bit integer is given by
Big Endian - the address of Most Significant BYTE or
Little Endian –the address of Least Significant BYTE

Machine Code (of Executable)

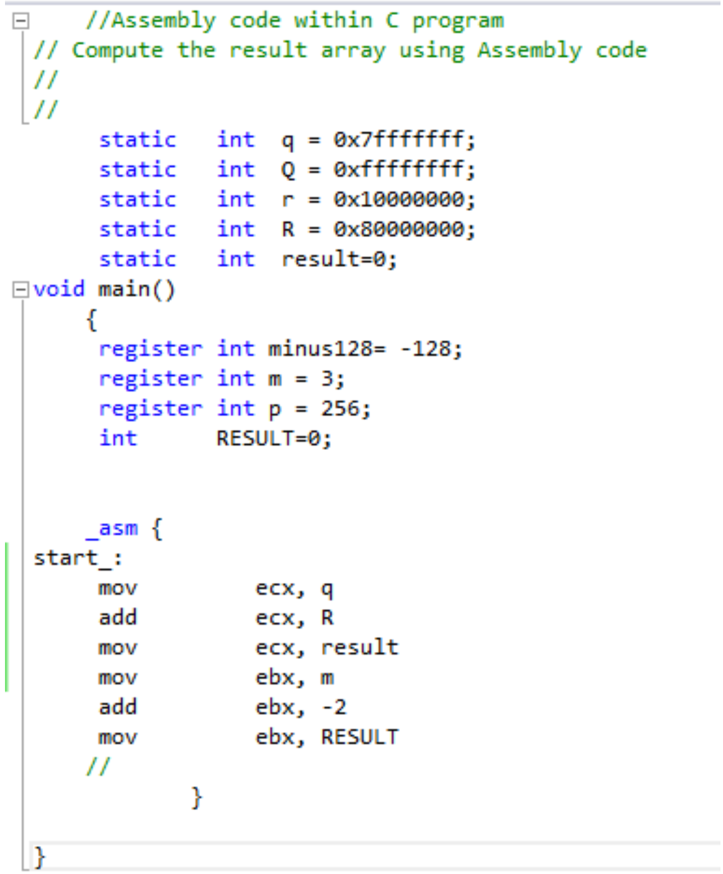
A series of Bits



Address: 0x000513C0

```
0x000513C0  55 8b ec 81 ec f0 00 00 00 53 56 57 8d bd 10 ff
0x000513D0  ff ff b9 3c 00 00 00 b8 cc cc cc cc f3 ab c7 45
0x000513E0  f8 80 ff ff ff c7 45 ec 03 00 00 00 c7 45 e0 00
0x000513F0  01 00 00 c7 45 d4 00 00 00 00 8b 0d 00 80 05 00
0x00051400  03 0d 0c 80 05 00 8b 0d 48 81 05 00 8b 5d ec 83
0x00051410  c3 fe 8b 5d d4 5f 5e 5b 81 c4 f0 00 00 00 3b ec
0x00051420  e8 11 fd ff ff 8b e5 5d c3 cc cc cc cc cc cc cc
```

Source code

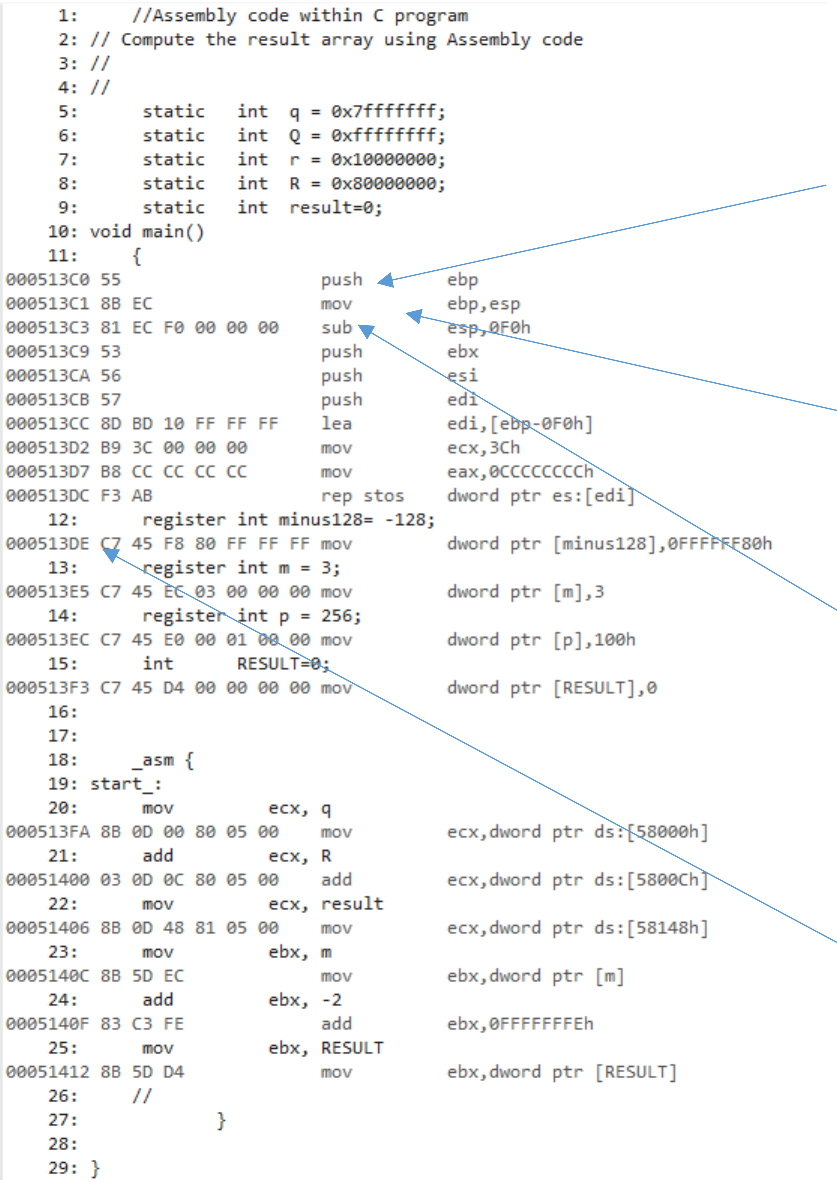


```
    //Assembly code within C program
// Compute the result array using Assembly code
//
//
     static   int  q = 0x7fffffff;
     static   int  Q = 0xffffffff;
     static   int  r = 0x10000000;
     static   int  R = 0x80000000;
     static   int  result=0;
void main()
    {
     register int minus128= -128;
     register int m = 3;
     register int p = 256;
     int      RESULT=0;


    _asm {
start_:
     mov         ecx, q
     add         ecx, R
     mov         ecx, result
     mov         ebx, m
     add         ebx, -2
     mov         ebx, RESULT
    //
            }

}
```

Disassembly Window



```
     1:      //Assembly code within C program
     2: // Compute the result array using Assembly code
     3: //
     4: //
     5:       static   int  q = 0x7fffffff;
     6:       static   int  Q = 0xffffffff;
     7:       static   int  r = 0x10000000;
     8:       static   int  R = 0x80000000;
     9:       static   int  result=0;
    10: void main()
    11:      {
000513C0 55                   push        ebp
000513C1 8B EC                mov         ebp,esp
000513C3 81 EC F0 00 00 00    sub         esp,0F0h
000513C9 53                   push        ebx
000513CA 56                   push        esi
000513CB 57                   push        edi
000513CC 8D BD 10 FF FF FF    lea         edi,[ebp-0F0h]
000513D2 B9 3C 00 00 00       mov         ecx,3Ch
000513D7 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
000513DC F3 AB                rep stos    dword ptr es:[edi]
    12:       register int minus128= -128;
000513DE C7 45 F8 80 FF FF FF mov         dword ptr [minus128],0FFFFFF80h
    13:       register int m = 3;
000513E5 C7 45 EC 03 00 00 00 mov         dword ptr [m],3
    14:       register int p = 256;
000513EC C7 45 E0 00 01 00 00 mov         dword ptr [p],100h
    15:       int      RESULT=0;
000513F3 C7 45 D4 00 00 00 00 mov         dword ptr [RESULT],0
    16:
    17:
    18:      _asm {
    19: start_:
    20:       mov         ecx, q
000513FA 8B 0D 00 80 05 00    mov         ecx,dword ptr ds:[58000h]
    21:       add         ecx, R
00051400 03 0D 0C 80 05 00    add         ecx,dword ptr ds:[5800Ch]
    22:       mov         ecx, result
00051406 8B 0D 48 81 05 00    mov         ecx,dword ptr ds:[58148h]
    23:       mov         ebx, m
0005140C 8B 5D EC             mov         ebx,dword ptr [m]
    24:       add         ebx, -2
0005140F 83 C3 FE             add         ebx,0FFFFFFFEh
    25:       mov         ebx, RESULT
00051412 8B 5D D4             mov         ebx,dword ptr [RESULT]
    26:      //
    27:             }
    28:
    29: }
```

Store (Push) the base pointer of the calling function on top of the stack.

Create new frame on top of the stack. .i.e.create a new base pointer :
EBP ← ESP

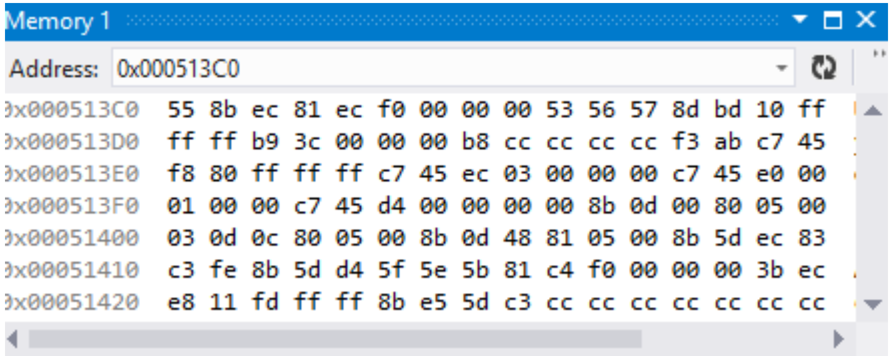
Allocate space on stack by 0F0H.
How many bytes in decimal?

Intel i7 processor machine instruction
During run-time
Allocates 32 bit memory on stack,
and initializes to -128.

Self check questions:

1. What are the addresses of the instructions that allocate memory to local variables?
2. What are the offsets to all local variables on stack?
3. Where are the offsets stored, and What are the addresses to the offsets?
4. What is the address of the first instruction in this program?

Code in Memory



Address: 0x000513C0

```
0x000513C0  55 8b ec 81 ec f0 00 00 00 53 56 57 8d bd 10 ff
0x000513D0  ff ff b9 3c 00 00 00 b8 cc cc cc cc f3 ab c7 45
0x000513E0  f8 80 ff ff ff c7 45 ec 03 00 00 00 c7 45 e0 00
0x000513F0  01 00 00 c7 45 d4 00 00 00 00 8b 0d 00 80 05 00
0x00051400  03 0d 0c 80 05 00 8b 0d 48 81 05 00 8b 5d ec 83
0x00051410  c3 fe 8b 5d d4 5f 5e 5b 81 c4 f0 00 00 00 3b ec
0x00051420  e8 11 fd ff ff 8b e5 5d c3 cc cc cc cc cc cc cc
```

1. What is the address in memory of the first instruction in this program?

Answer: 0x000513C0 Why?

2. What is the length in bytes of the first instruction?

Answer: 0x000513C1-0x000513C0 = 1 Why?

3. What is the length in bytes of the second instruction?

Answer:

4. Can you find instruction in memory 1 window, its address, that allocates memory to local variable p?

Answer:

C3 is op code(operation code) for the last machine instruction.

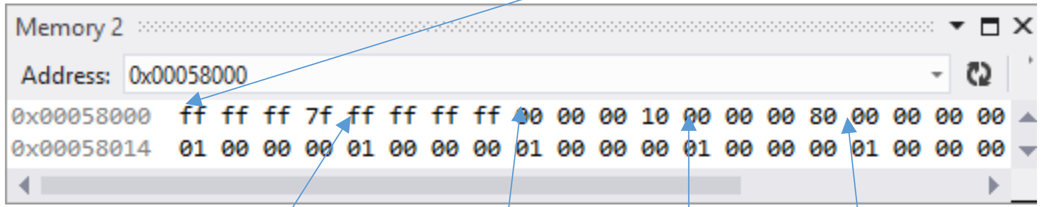
1. What is the address of the last instruction?

Answer:

2. What is the length in bytes of the machine code shown Memory1 window?

Answer:

Static variables in Data segment Memory



Address: 0x00058000

```
0x00058000  ff ff ff 7f ff ff ff ff 00 00 00 10 00 00 00 80 00 00 00 00
0x00058014  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
```

1. First static variable **q** in our program is stored a the address 0x00058000

The value at this address is the most positive integer that can be stored in 32 bit word 0x7FFFFFFF

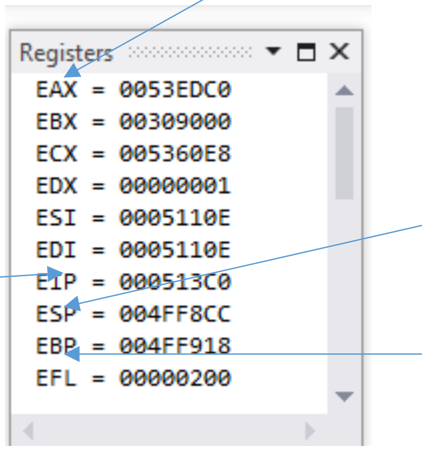
What is the static variable name, address, and value stored at these locations?

var1

var2 var3 var4

Registers

Original ACCUMULATOR register EAX, is frequently used for arithmetic operations.
Other registers
EBX, ECX, EDX, ESI, EDI
Can also be freely used for arithmetic operations.



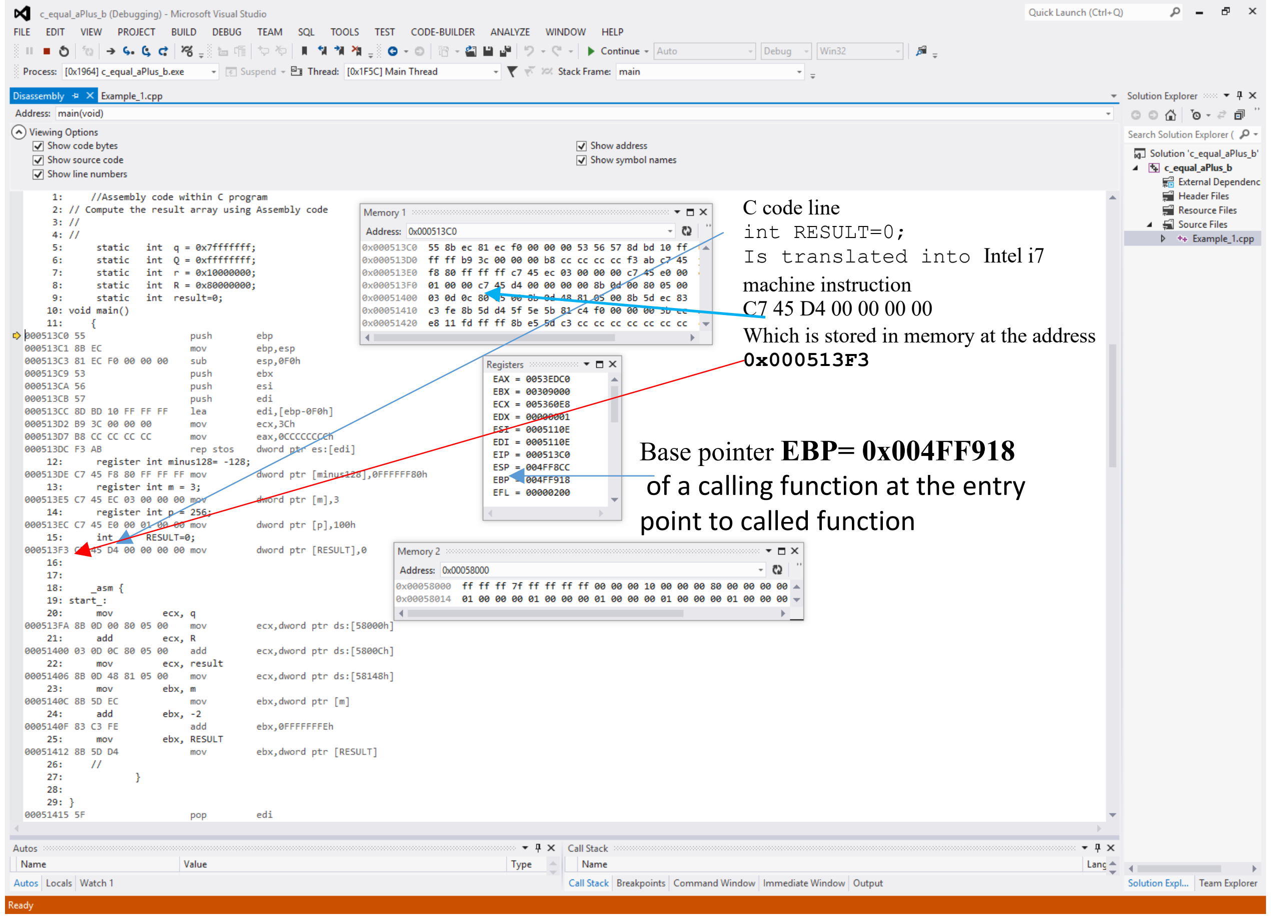
```
EAX = 0053EDC0
EBX = 00309000
ECX = 005360E8
EDX = 00000001
ESI = 0005110E
EDI = 0005110E
EIP = 000513C0
ESP = 004FF8CC
EBP = 004FF918
EFL = 00000200
```

Instruction Pointer register
EIP = 0x00513C0
Stores the address of instruction that will be executed next.

Stack pointer of a calling function at the entry point to called function

Base pointer of a calling function at the entry point to called function

Code and data, machine instructions


c_equal_aPlus_b (Debugging) - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST CODE-BUILDER ANALYZE WINDOW HELP
Continue
Auto
Debug
Win32
Process: [0x1964] c_equal_aPlus_b.exe
Suspend
Thread: [0x1F5C] Main Thread
Stack Frame: main
Disassembly
Example_1.cpp
Address: main(void)
Solution Explorer
Search Solution Explorer (
Solution 'c_equal_aPlus_b'
c_equal_aPlus_b
External Dependenc
Header Files
Resource Files
Source Files
Example_1.cpp
Autos
Name
Value
Type
Autos Locals Watch 1
Call Stack
Name
Lang
Call Stack Breakpoints Command Window Immediate Window Output
Solution Expl... Team Explorer
Ready

Viewing Options

☑ Show code bytes
☑ Show source code
☑ Show line numbers
☑ Show address
☑ Show symbol names

```
     1:     //Assembly code within C program
     2: // Compute the result array using Assembly code
     3: //
     4: //
     5:      static   int  q = 0x7fffffff;
     6:      static   int  Q = 0xffffffff;
     7:      static   int  r = 0x10000000;
     8:      static   int  R = 0x80000000;
     9:      static   int  result=0;
    10: void main()
    11:     {
000513C0 55                   push        ebp
000513C1 8B EC                mov         ebp,esp
000513C3 81 EC F0 00 00 00    sub         esp,0F0h
000513C9 53                   push        ebx
000513CA 56                   push        esi
000513CB 57                   push        edi
000513CC 8D BD 10 FF FF FF    lea         edi,[ebp-0F0h]
000513D2 B9 3C 00 00 00       mov         ecx,3Ch
000513D7 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
000513DC F3 AB                rep stos    dword ptr es:[edi]
    12:      register int minus128= -128;
000513DE C7 45 F8 80 FF FF FF mov         dword ptr [minus128],0FFFFFF80h
    13:      register int m = 3;
000513E5 C7 45 EC 03 00 00 00 mov         dword ptr [m],3
    14:      register int p = 256;
000513EC C7 45 E0 00 01 00 00 mov         dword ptr [p],100h
    15:      int      RESULT=0;
000513F3 C7 45 D4 00 00 00 00 mov         dword ptr [RESULT],0
    16:
    17:
    18:     _asm {
    19: start_:
    20:      mov         ecx, q
000513FA 8B 0D 00 80 05 00    mov         ecx,dword ptr ds:[58000h]
    21:      add         ecx, R
00051400 03 0D 0C 80 05 00    add         ecx,dword ptr ds:[5800Ch]
    22:      mov         ecx, result
00051406 8B 0D 48 81 05 00    mov         ecx,dword ptr ds:[58148h]
    23:      mov         ebx, m
0005140C 8B 5D EC             mov         ebx,dword ptr [m]
    24:      add         ebx, -2
0005140F 83 C3 FE             add         ebx,0FFFFFFFEh
    25:      mov         ebx, RESULT
00051412 8B 5D D4             mov         ebx,dword ptr [RESULT]
    26:     //
    27:             }
    28:
    29: }
00051415 5F                   pop         edi
```

Memory 1

Address: 0x000513C0

```
0x000513C0  55 8b ec 81 ec f0 00 00 00 53 56 57 8d bd 10 ff
0x000513D0  ff ff b9 3c 00 00 00 b8 cc cc cc cc f3 ab c7 45
0x000513E0  f8 80 ff ff ff c7 45 ec 03 00 00 00 c7 45 e0 00
0x000513F0  01 00 00 c7 45 d4 00 00 00 00 8b 0d 00 80 05 00
0x00051400  03 0d 0c 80 05 00 8b 0d 48 81 05 00 8b 5d ec 83
0x00051410  c3 fe 8b 5d d4 5f 5e 5b 81 c4 f0 00 00 00 3b cc
0x00051420  e8 11 fd ff ff 8b e5 5d c3 cc cc cc cc cc cc cc
```

Registers

```
EAX = 0053EDC0
EBX = 00309000
ECX = 005360E8
EDX = 00000001
ESI = 0005110E
EDI = 0005110E
EIP = 000513C0
ESP = 004FF8CC
EBP = 004FF918
EFL = 00000200
```

Memory 2

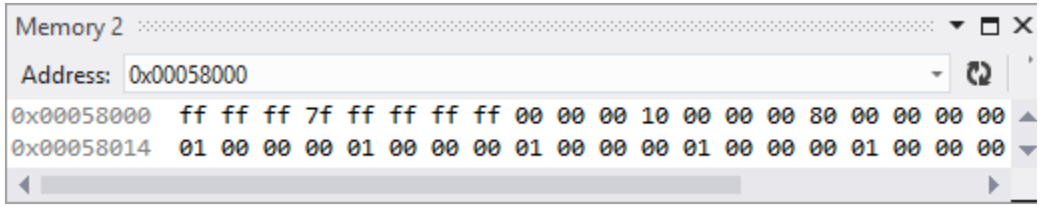
Address: 0x00058000

```
0x00058000  ff ff ff 7f ff ff ff ff 00 00 00 10 00 00 00 80 00 00 00 00
0x00058014  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
```

C code line
`int RESULT=0;`
`Is translated into` Intel i7
machine instruction
C7 45 D4 00 00 00 00
Which is stored in memory at the address
`0x000513F3`

Base pointer **EBP= 0x004FF918**
of a calling function at the entry
point to called function

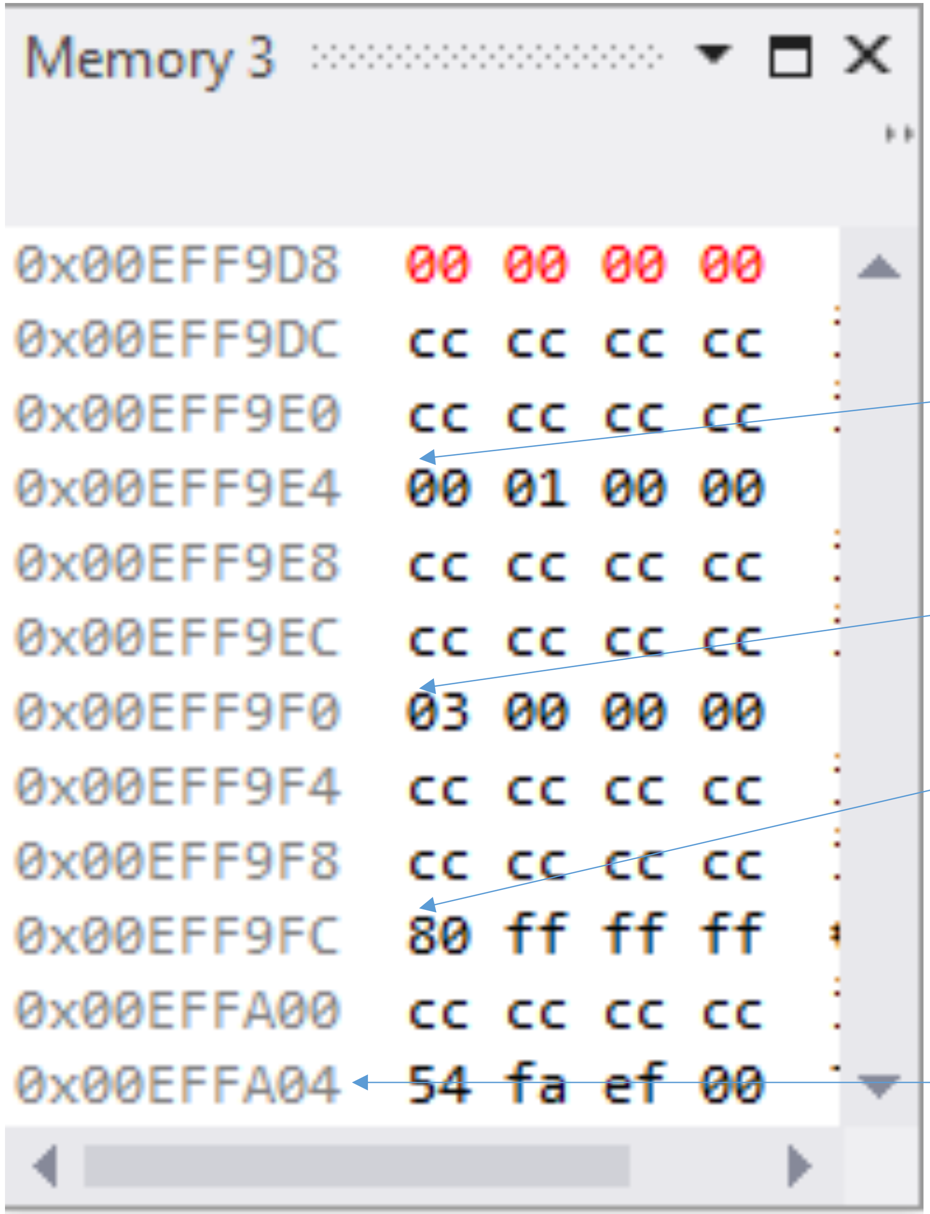
Static Data in Memory



Address: 0x00058000

```
0x00058000  ff ff ff 7f ff ff ff ff 00 00 00 10 00 00 00 80 00 00 00 00
0x00058014  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
```

Local variables on Stack



```
0x00EFF9D8  00 00 00 00
0x00EFF9DC  cc cc cc cc
0x00EFF9E0  cc cc cc cc
0x00EFF9E4  00 01 00 00
0x00EFF9E8  cc cc cc cc
0x00EFF9EC  cc cc cc cc
0x00EFF9F0  03 00 00 00
0x00EFF9F4  cc cc cc cc
0x00EFF9F8  cc cc cc cc
0x00EFF9FC  80 ff ff ff
0x00EFFA00  cc cc cc cc
0x00EFFA04  54 fa ef 00
```

Local variable RESULT on stack
Offset from Base pntr 0xffffffD4
Value 0x00000000

Self-Check: What is the signed decimal value of the offset?

Local variable p on stack
Offset from Base pntr 0xffffffe0
Value 0x00000100

Self-Check: What is the signed decimal value of the offset?

Local variable m on stack
Offset from Base pntr 0xffffffec
Value 0x00000003

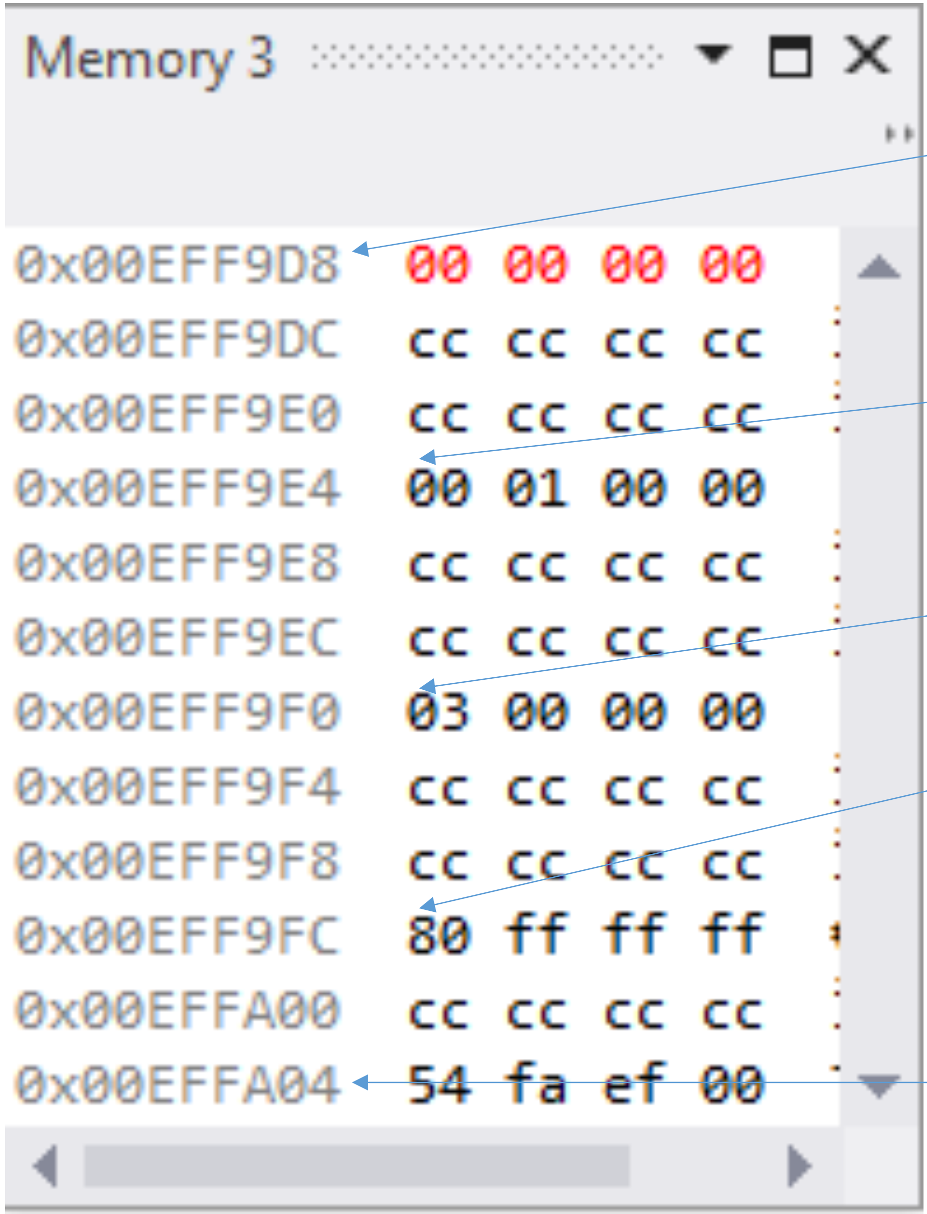
Self-Check: What is the signed decimal value of the offset?

Local variable minus128 on stack
Offset from Base pntr 0xfffffff8
Value 0xffffff80

Self-Check: What is the signed decimal value of the offset?

The Base Pointer of
current called function
is EBP=0x00EFFA04

ANSWERS Offsets to Local variables on Stack



```
0x00EFF9D8  00 00 00 00
0x00EFF9DC  cc cc cc cc
0x00EFF9E0  cc cc cc cc
0x00EFF9E4  00 01 00 00
0x00EFF9E8  cc cc cc cc
0x00EFF9EC  cc cc cc cc
0x00EFF9F0  03 00 00 00
0x00EFF9F4  cc cc cc cc
0x00EFF9F8  cc cc cc cc
0x00EFF9FC  80 ff ff ff
0x00EFFA00  cc cc cc cc
0x00EFFA04  54 fa ef 00
```

Local variable RESULT on stack

Offset from Base pntr 0xffffffD4

Value 0x00000000

Self-Check: What is the signed decimal value of the offset? **-44**

0xFA04-0xF9D8=0x02C->44, offset =**-44**

Local variable p on stack

Offset from Base pntr 0xffffffe0

Value 0x00000100

Self-Check: What is the signed decimal value of the offset? **-32**

0xFA04-0xF9E4=0s0020->32,

Local variable m on stack

Offset from Base pntr 0xffffffec

Value 0x00000003

Self-Check: What is the signed decimal value of the offset?**-20**

0xFA04-0xF9F0=0x0014->20,

Local variable minus128 on stack

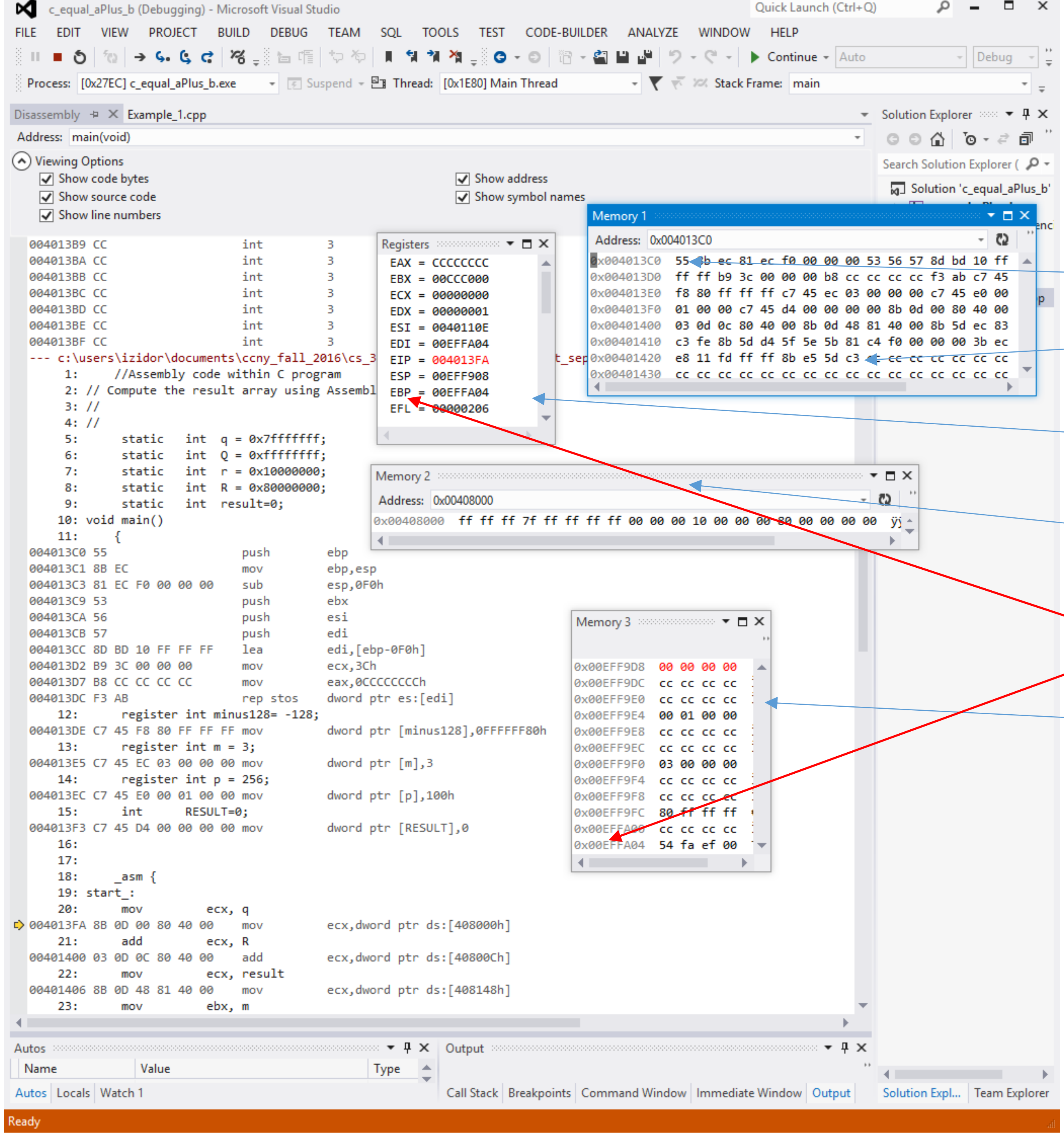
Offset from Base pntr 0xfffffff8

Value 0xffffff80

Self-Check: What is the signed decimal value of the offset? **-8**

0xFA04-0xF9FC=0x0008->8,

The Base Pointer of current called function ***is EBP=0x00EFFA04***


c_equal_aPlus_b (Debugging) - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST CODE-BUILDER ANALYZE WINDOW HELP
Continue
Auto
Debug
Suspend
Stack Frame:
Disassembly
Example_1.cpp
Viewing Options
Solution Explorer
Search Solution Explorer (
Solution 'c_equal_aPlus_b'
Autos
Name
Value
Type
Output
Autos Locals Watch 1
Call Stack Breakpoints Command Window Immediate Window Output
Solution Expl... Team Explorer
Ready

Process: [0x27EC] c_equal_aPlus_b.exe
Thread: [0x1E80] Main Thread
Stack Frame: main

Address: main(void)

☑ Show code bytes
☑ Show source code
☑ Show line numbers
☑ Show address
☑ Show symbol names

```
004013B9 CC                   int         3
004013BA CC                   int         3
004013BB CC                   int         3
004013BC CC                   int         3
004013BD CC                   int         3
004013BE CC                   int         3
004013BF CC                   int         3
--- c:\users\izidor\documents\ccny_fall_2016\cs_3
     1:     //Assembly code within C program
     2: // Compute the result array using Assembl
     3: //
     4: //
     5:      static   int  q = 0x7fffffff;
     6:      static   int  Q = 0xffffffff;
     7:      static   int  r = 0x10000000;
     8:      static   int  R = 0x80000000;
     9:      static   int  result=0;
    10: void main()
    11:     {
004013C0 55                   push        ebp
004013C1 8B EC                mov         ebp,esp
004013C3 81 EC F0 00 00 00    sub         esp,0F0h
004013C9 53                   push        ebx
004013CA 56                   push        esi
004013CB 57                   push        edi
004013CC 8D BD 10 FF FF FF    lea         edi,[ebp-0F0h]
004013D2 B9 3C 00 00 00       mov         ecx,3Ch
004013D7 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
004013DC F3 AB                rep stos    dword ptr es:[edi]
    12:      register int minus128= -128;
004013DE C7 45 F8 80 FF FF FF mov         dword ptr [minus128],0FFFFFF80h
    13:      register int m = 3;
004013E5 C7 45 EC 03 00 00 00 mov         dword ptr [m],3
    14:      register int p = 256;
004013EC C7 45 E0 00 01 00 00 mov         dword ptr [p],100h
    15:      int      RESULT=0;
004013F3 C7 45 D4 00 00 00 00 mov         dword ptr [RESULT],0
    16:
    17:
    18:     _asm {
    19: start_:
    20:      mov          ecx, q
004013FA 8B 0D 00 80 40 00    mov         ecx,dword ptr ds:[408000h]
    21:      add          ecx, R
00401400 03 0D 0C 80 40 00    add         ecx,dword ptr ds:[40800Ch]
    22:      mov          ecx, result
00401406 8B 0D 48 81 40 00    mov         ecx,dword ptr ds:[408148h]
    23:      mov          ebx, m
```

Registers

```
EAX = CCCCCCCC
EBX = 00CCC000
ECX = 00000000
EDX = 00000001
ESI = 0040110E
EDI = 00EFFA04
EIP = 004013FA
ESP = 00EFF908
EBP = 00EFFA04
EFL = 00000206
```

Memory 1

Address: 0x004013C0

```
0x004013C0  55 8b ec 81 ec f0 00 00 00 53 56 57 8d bd 10 ff
0x004013D0  ff ff b9 3c 00 00 00 b8 cc cc cc cc f3 ab c7 45
0x004013E0  f8 80 ff ff ff c7 45 ec 03 00 00 00 c7 45 e0 00
0x004013F0  01 00 00 c7 45 d4 00 00 00 00 8b 0d 00 80 40 00
0x00401400  03 0d 0c 80 40 00 8b 0d 48 81 40 00 8b 5d ec 83
0x00401410  c3 fe 8b 5d d4 5f 5e 5b 81 c4 f0 00 00 00 3b ec
0x00401420  e8 11 fd ff ff 8b e5 5d c3 cc cc cc cc cc cc cc
0x00401430  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
```

Memory 2

Address: 0x00408000

```
0x00408000  ff ff ff 7f ff ff ff ff 00 00 00 10 00 00 00 80 00 00 00 00  ÿÿ
```

Memory 3

```
0x00EFF9D8  00 00 00 00
0x00EFF9DC  cc cc cc cc
0x00EFF9E0  cc cc cc cc
0x00EFF9E4  00 01 00 00
0x00EFF9E8  cc cc cc cc
0x00EFF9EC  cc cc cc cc
0x00EFF9F0  03 00 00 00
0x00EFF9F4  cc cc cc cc
0x00EFF9F8  cc cc cc cc
0x00EFF9FC  80 ff ff ff
0x00EFFA00  cc cc cc cc
0x00EFFA04  54 fa ef 00
```

Executable

First machine instruction 0x55

Last machine instruction 0xC3

X86 Registers

Static Variables in data segment

The Base Pointer of current called function
is EBP=0x00EFFA04

Local variables on Stack