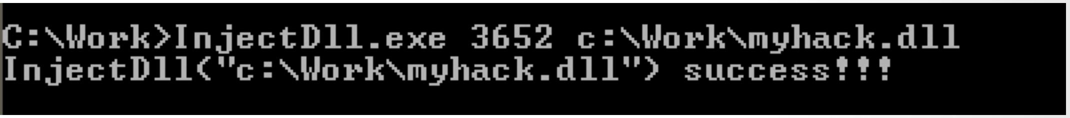
Class 3

CSC 471 Modern Malware Analysis
IA-32 Registers & Byte Ordering

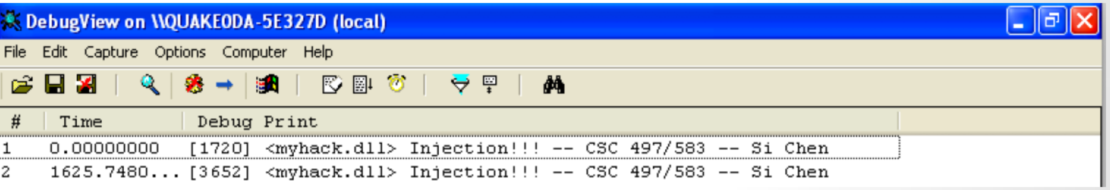
Si Chen (schen@wcupa.edu)

Review

Screenshots

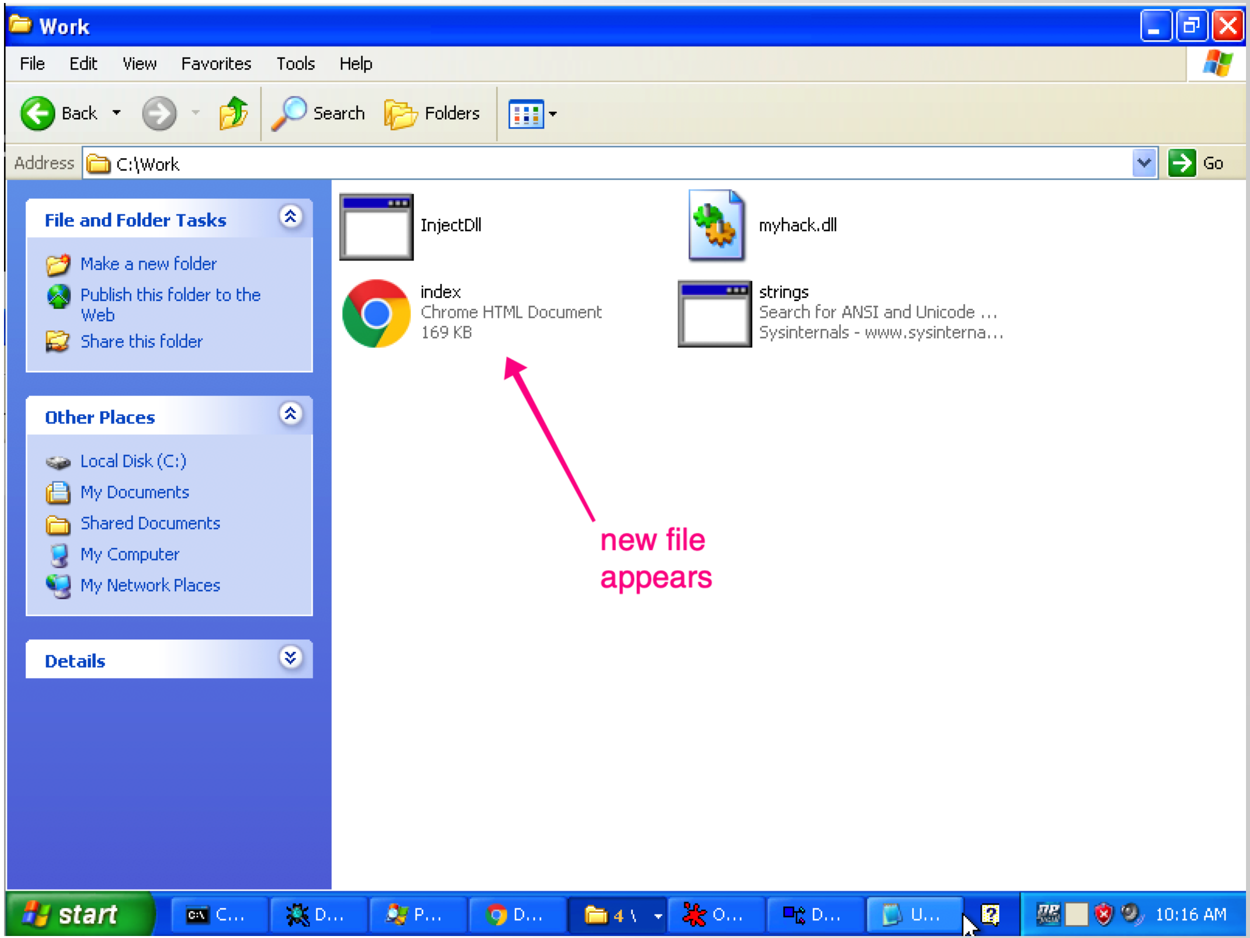


```
C:\Work>InjectDll.exe 3652 c:\Work\myhack.dll
InjectDll("c:\Work\myhack.dll") success!!!
```



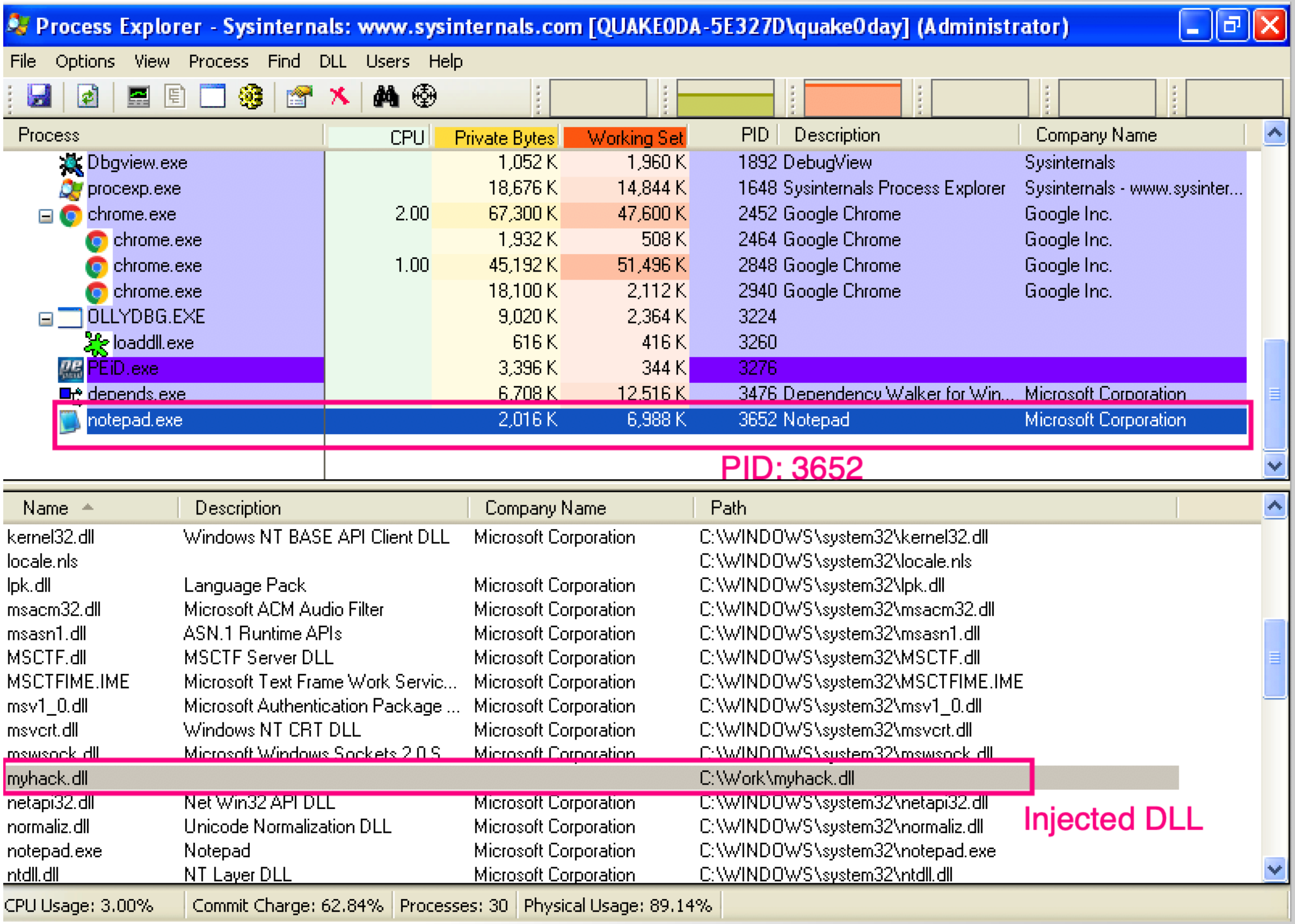
DebugView on \\QUAKEODA-5E327D (local)

```
#   Time          Debug Print
1   0.00000000    [1720] <myhack.dll> Injection!!! -- CSC 497/583 -- Si Chen
2   1625.7480...  [3652] <myhack.dll> Injection!!! -- CSC 497/583 -- Si Chen
```



new file appears

Screenshots


Process Explorer - Sysinternals: www.sysinternals.com [QUAKE0DA-5E327D\quake0day] (Administrator)
File
Options
View
Process
Find
DLL
Users
Help

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
Dbgview.exe		1,052 K	1,960 K	1892	DebugView	Sysinternals
procexp.exe		18,676 K	14,844 K	1648	Sysinternals Process Explorer	Sysinternals - www.sysinter...
chrome.exe	2.00	67,300 K	47,600 K	2452	Google Chrome	Google Inc.
chrome.exe		1,932 K	508 K	2464	Google Chrome	Google Inc.
chrome.exe	1.00	45,192 K	51,496 K	2848	Google Chrome	Google Inc.
chrome.exe		18,100 K	2,112 K	2940	Google Chrome	Google Inc.
OLLYDBG.EXE		9,020 K	2,364 K	3224		
loaddll.exe		616 K	416 K	3260		
PEiD.exe		3,396 K	344 K	3276		
depends.exe		6,708 K	12,516 K	3476	Dependency Walker for Win...	Microsoft Corporation
notepad.exe		2,016 K	6,988 K	3652	Notepad	Microsoft Corporation

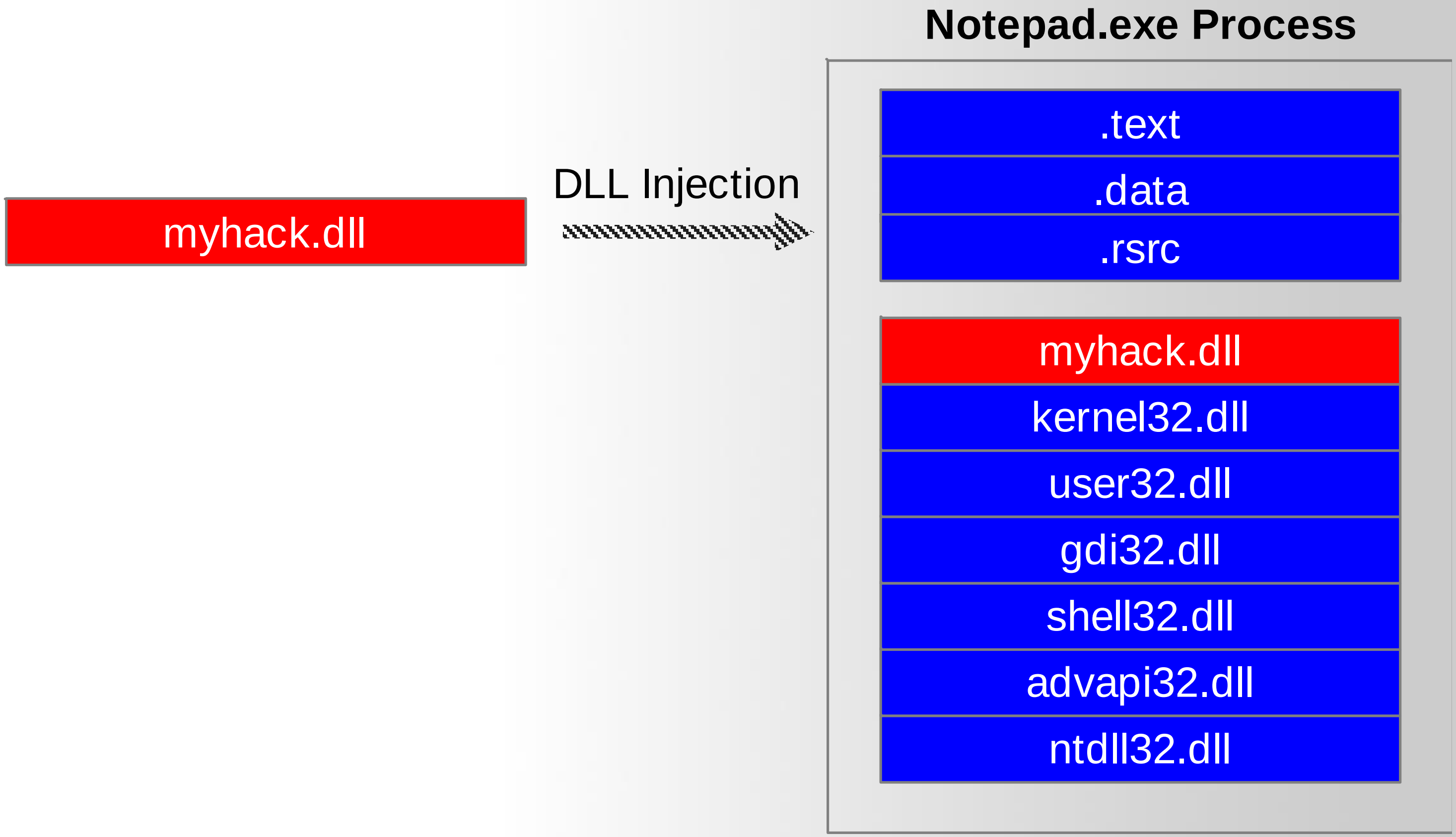
PID: 3652

Name	Description	Company Name	Path
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\WINDOWS\system32\kernel32.dll
locale.nls			C:\WINDOWS\system32\locale.nls
lpk.dll	Language Pack	Microsoft Corporation	C:\WINDOWS\system32\lpk.dll
msacm32.dll	Microsoft ACM Audio Filter	Microsoft Corporation	C:\WINDOWS\system32\msacm32.dll
msasn1.dll	ASN.1 Runtime APIs	Microsoft Corporation	C:\WINDOWS\system32\msasn1.dll
MSCTF.dll	MSCTF Server DLL	Microsoft Corporation	C:\WINDOWS\system32\MSCTF.dll
MSCTFIME.IME	Microsoft Text Frame Work Servic...	Microsoft Corporation	C:\WINDOWS\system32\MSCTFIME.IME
msv1_0.dll	Microsoft Authentication Package ...	Microsoft Corporation	C:\WINDOWS\system32\msv1_0.dll
msvcrt.dll	Windows NT CRT DLL	Microsoft Corporation	C:\WINDOWS\system32\msvcrt.dll
mswsock.dll	Microsoft Windows Sockets 2.0 S...	Microsoft Corporation	C:\WINDOWS\system32\mswsock.dll
myhack.dll			C:\Work\myhack.dll
netapi32.dll	Net Win32 API DLL	Microsoft Corporation	C:\WINDOWS\system32\netapi32.dll
normaliz.dll	Unicode Normalization DLL	Microsoft Corporation	C:\WINDOWS\system32\normaliz.dll
notepad.exe	Notepad	Microsoft Corporation	C:\WINDOWS\system32\notepad.exe
ntdll.dll	NT Layer DLL	Microsoft Corporation	C:\WINDOWS\system32\ntdll.dll

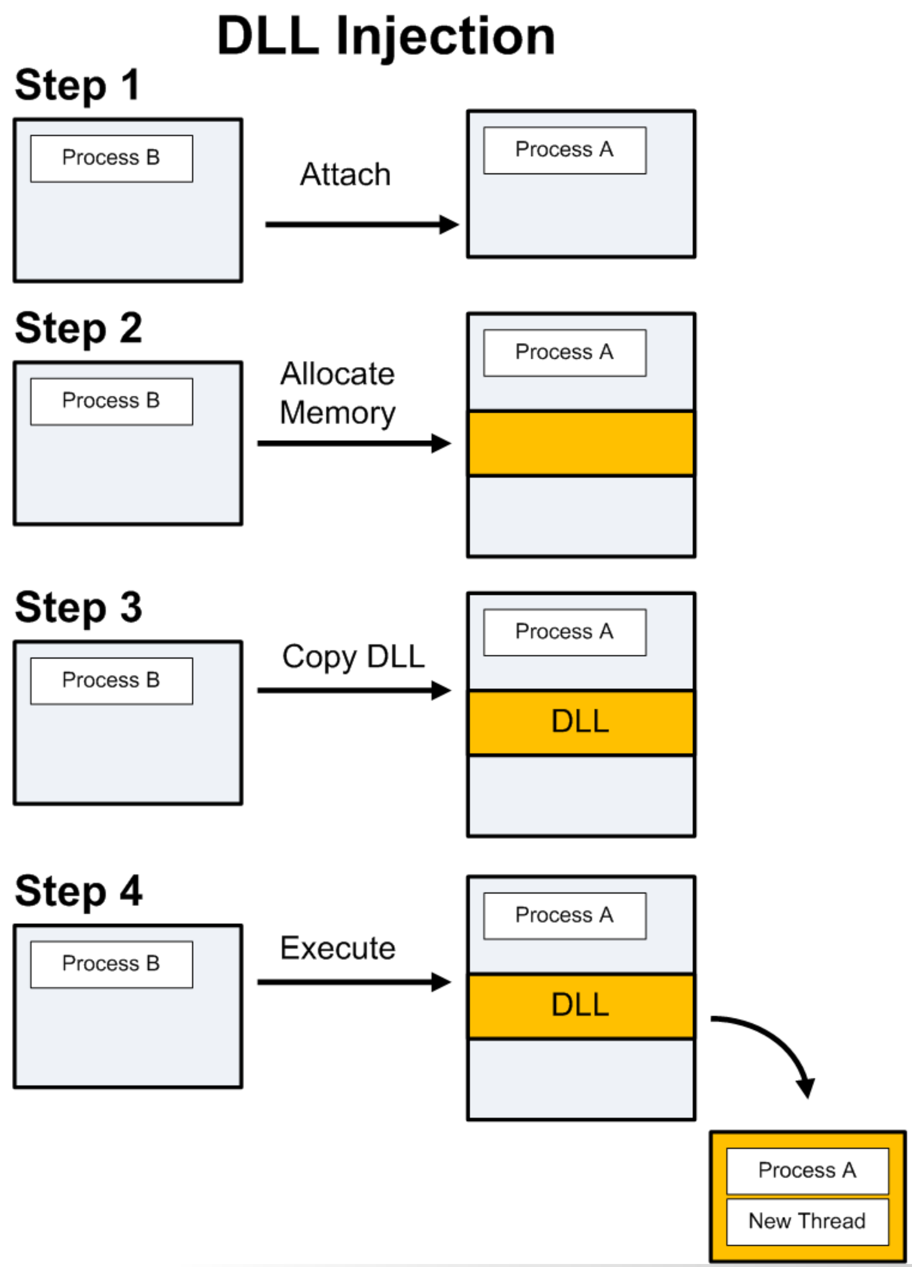
Injected DLL

CPU Usage: 3.00% | Commit Charge: 62.84% | Processes: 30 | Physical Usage: 89.14%

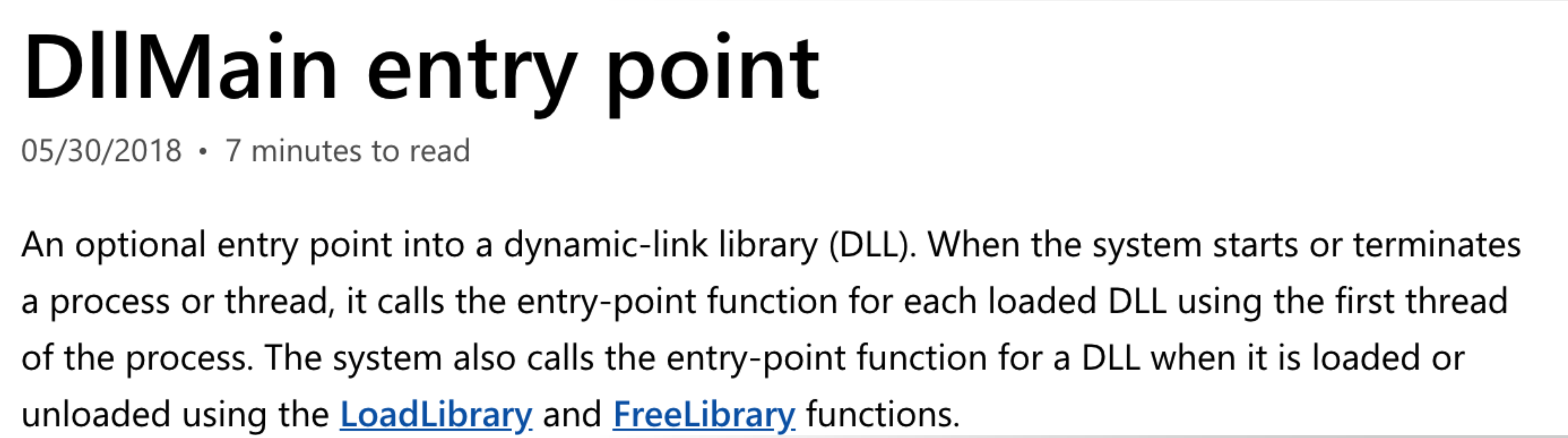
DLL Injection



DLL Injection



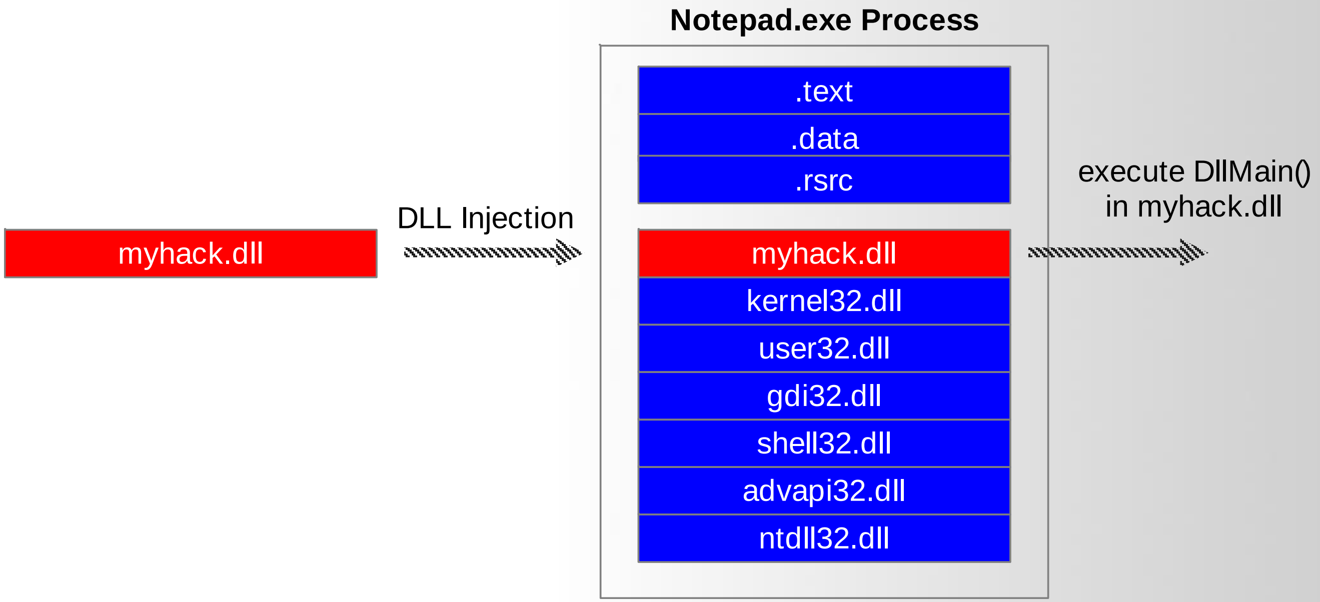
DllMain()



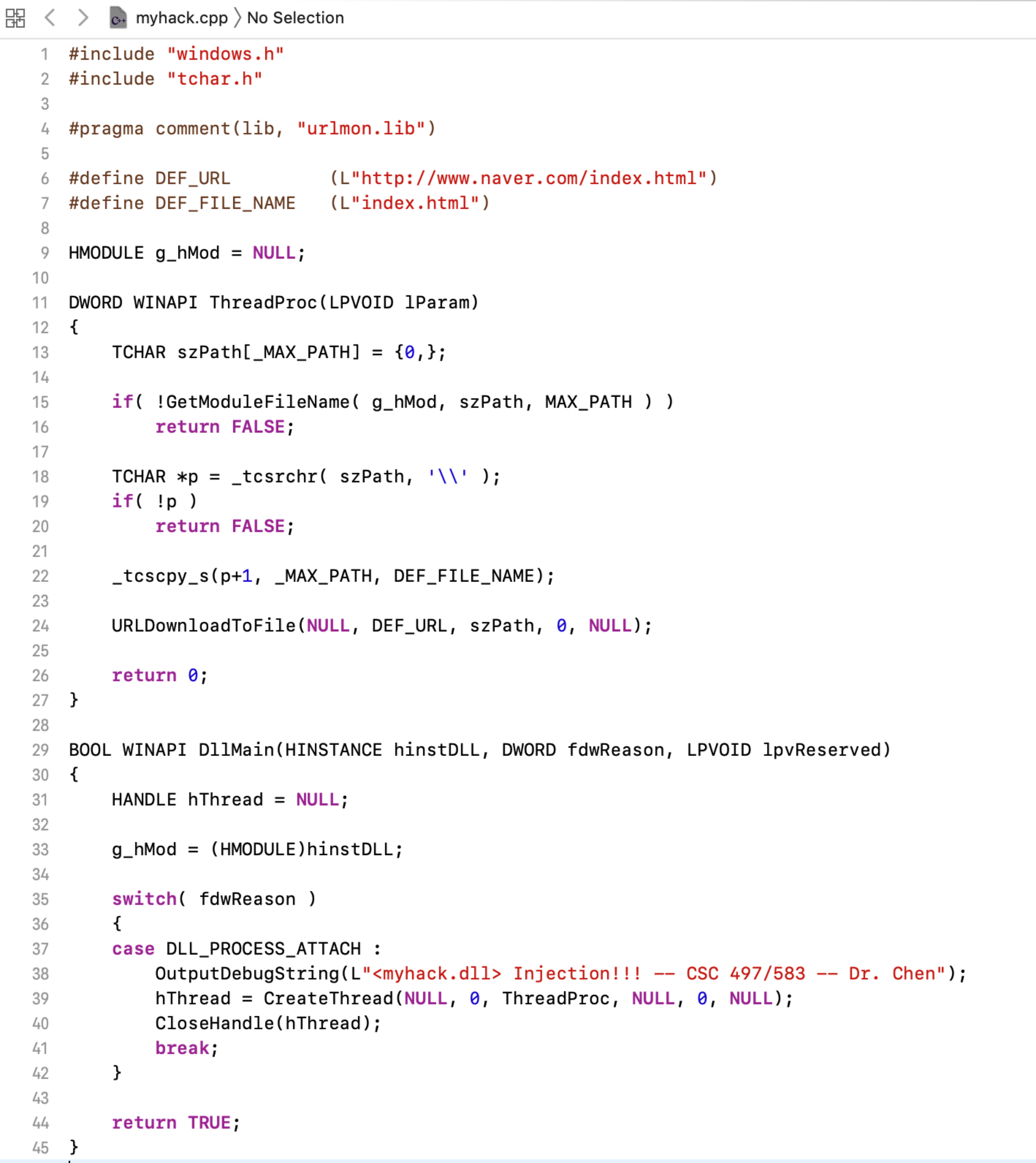
DllMain entry point

05/30/2018 • 7 minutes to read

An optional entry point into a dynamic-link library (DLL). When the system starts or terminates a process or thread, it calls the entry-point function for each loaded DLL using the first thread of the process. The system also calls the entry-point function for a DLL when it is loaded or unloaded using the LoadLibrary and FreeLibrary functions.



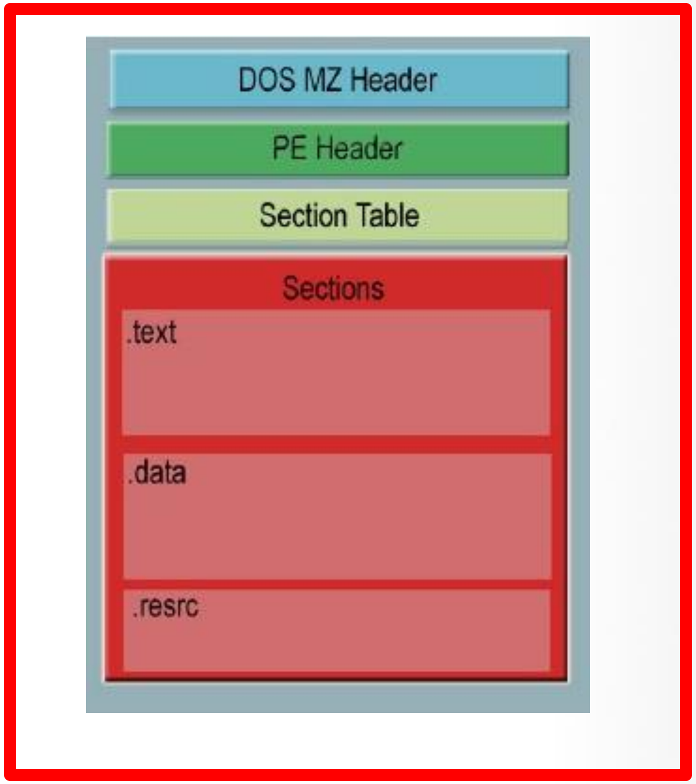
Source Code of myhack.dll



```
#include "windows.h"
#include "tchar.h"

#pragma comment(lib, "urlmon.lib")

#define DEF_URL         (L"http://www.naver.com/index.html")
#define DEF_FILE_NAME   (L"index.html")

HMODULE g_hMod = NULL;

DWORD WINAPI ThreadProc(LPVOID lParam)
{
    TCHAR szPath[_MAX_PATH] = {0,};

    if( !GetModuleFileName( g_hMod, szPath, MAX_PATH ) )
        return FALSE;

    TCHAR *p = _tcsrchr( szPath, '\\' );
    if( !p )
        return FALSE;

    _tcscpy_s(p+1, _MAX_PATH, DEF_FILE_NAME);

    URLDownloadToFile(NULL, DEF_URL, szPath, 0, NULL);

    return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    HANDLE hThread = NULL;

    g_hMod = (HMODULE)hinstDLL;

    switch( fdwReason )
    {
    case DLL_PROCESS_ATTACH :
        OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
        hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
        CloseHandle(hThread);
        break;
    }

    return TRUE;
}
```

Portable Executable (PE) file

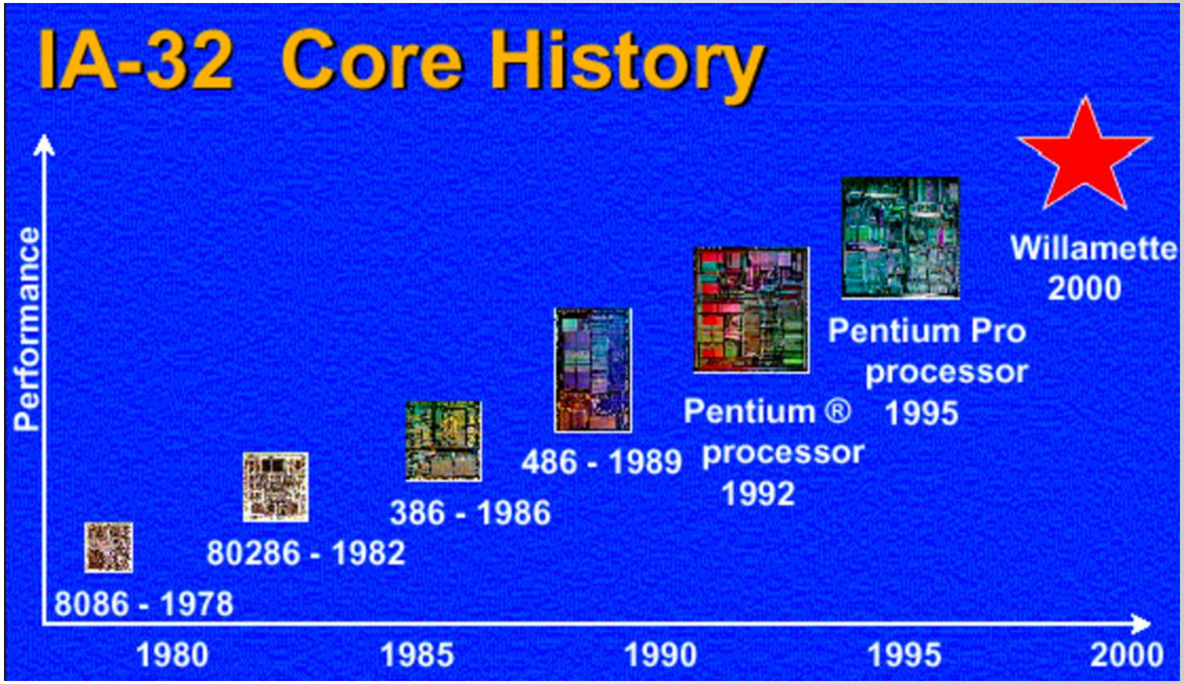
- A Portable Executable (**PE**) **file** is the standard **binary file** format for an **Executable (.exe) or DLL** under Windows NT, Windows 95, and Win32.



IA-32 Register

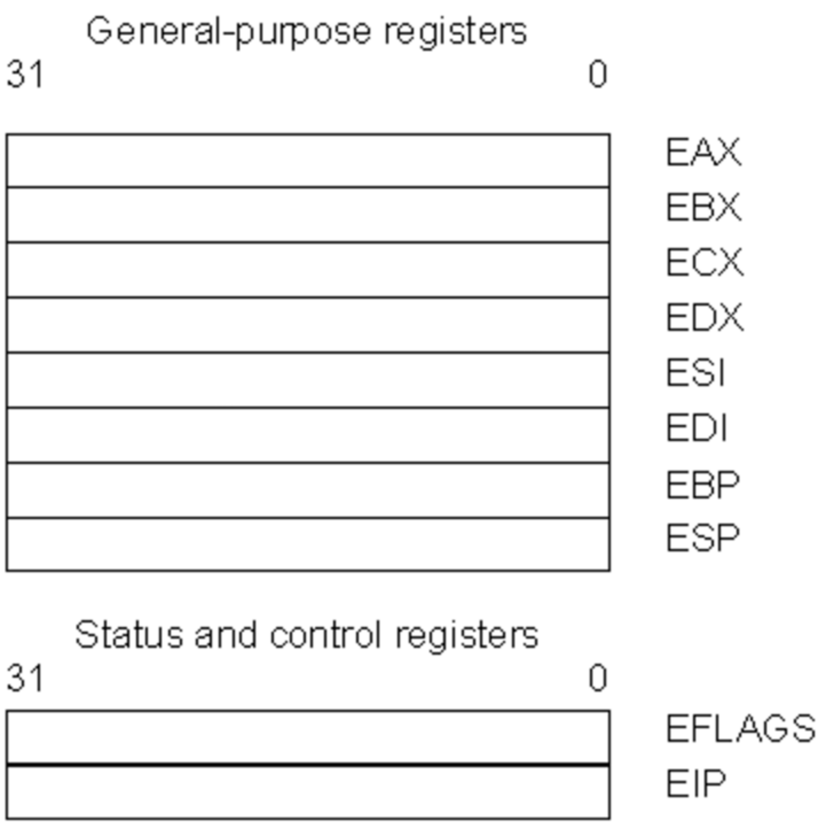
Intel IA-32 Processor

- Intel uses IA-32 to refer to Pentium processor family, in order to distinguish them from their 64-bit architectures.



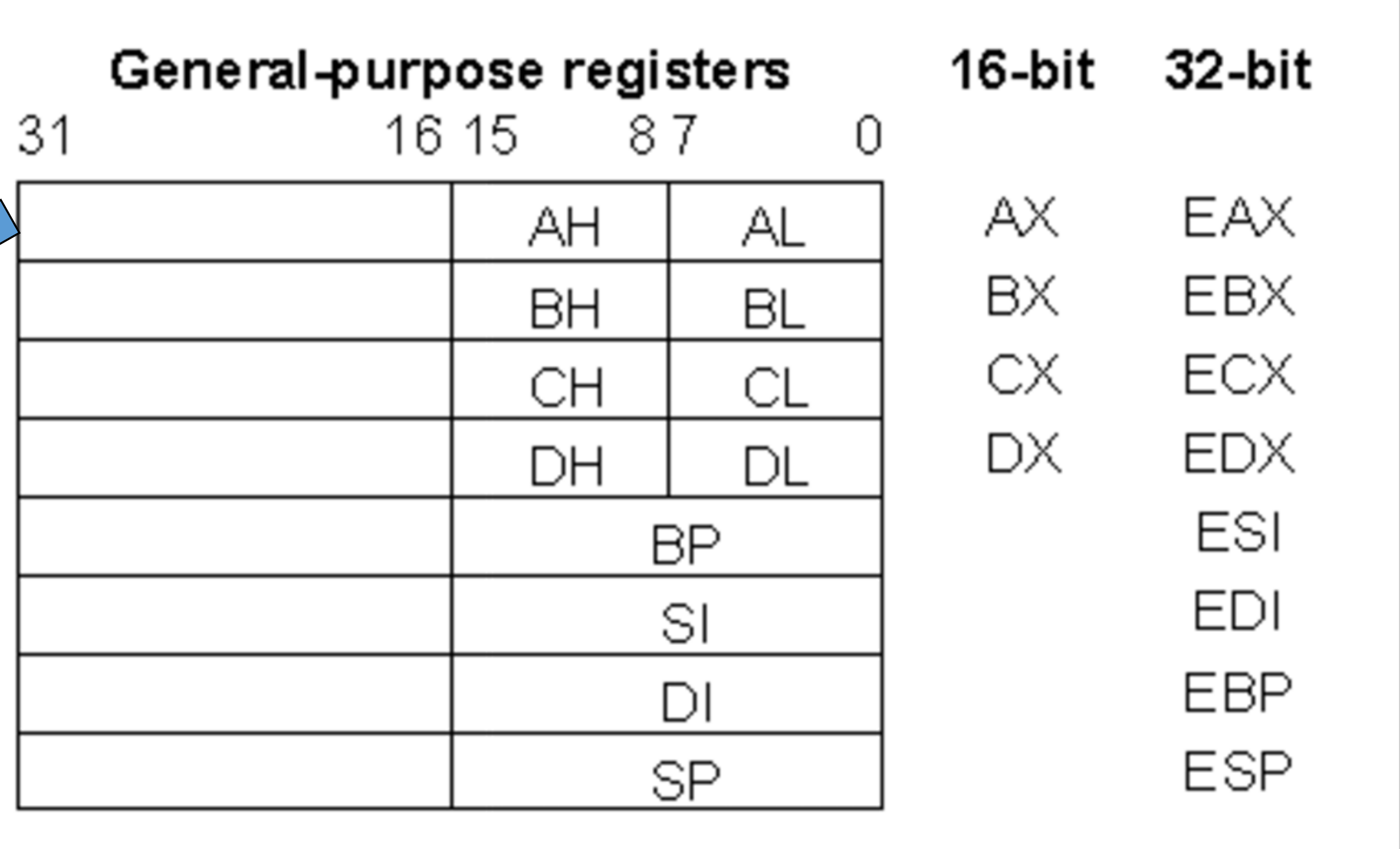
Register Set

- There are three types of registers:
  - general-purpose data registers,
  - segment registers,
  - status and control registers.



General-purpose Registers

- The **eight** 32-bit general-purpose data registers are used to hold operands for logical and arithmetic operations, operands for address calculations and memory pointers



General-purpose registers			16-bit	32-bit
31 16	15 8	7 0		
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			ESI
	SI			EDI
	DI			EBP
	SP			ESP

4 Bytes

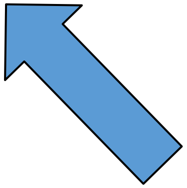
Other uses…

- – EAX—Accumulator for operands and results data.
- – EBX—Pointer to data in the DS segment.
- – ECX—Counter for string and loop operations.
- – EDX—I/O pointer.

1. We use these four registers when we perform arithmetic operations (ADD, SUB, XOR, OR) -- store constant or variable’s value.
2. Some assembly operations (MUL, DIV, LODS) directly operate these register and altered the value when finished.
3. ECX is used for loop count → decrease 1 after each loop
4. EAX is used for storing the return value of a function (Win32 API)

Other uses…

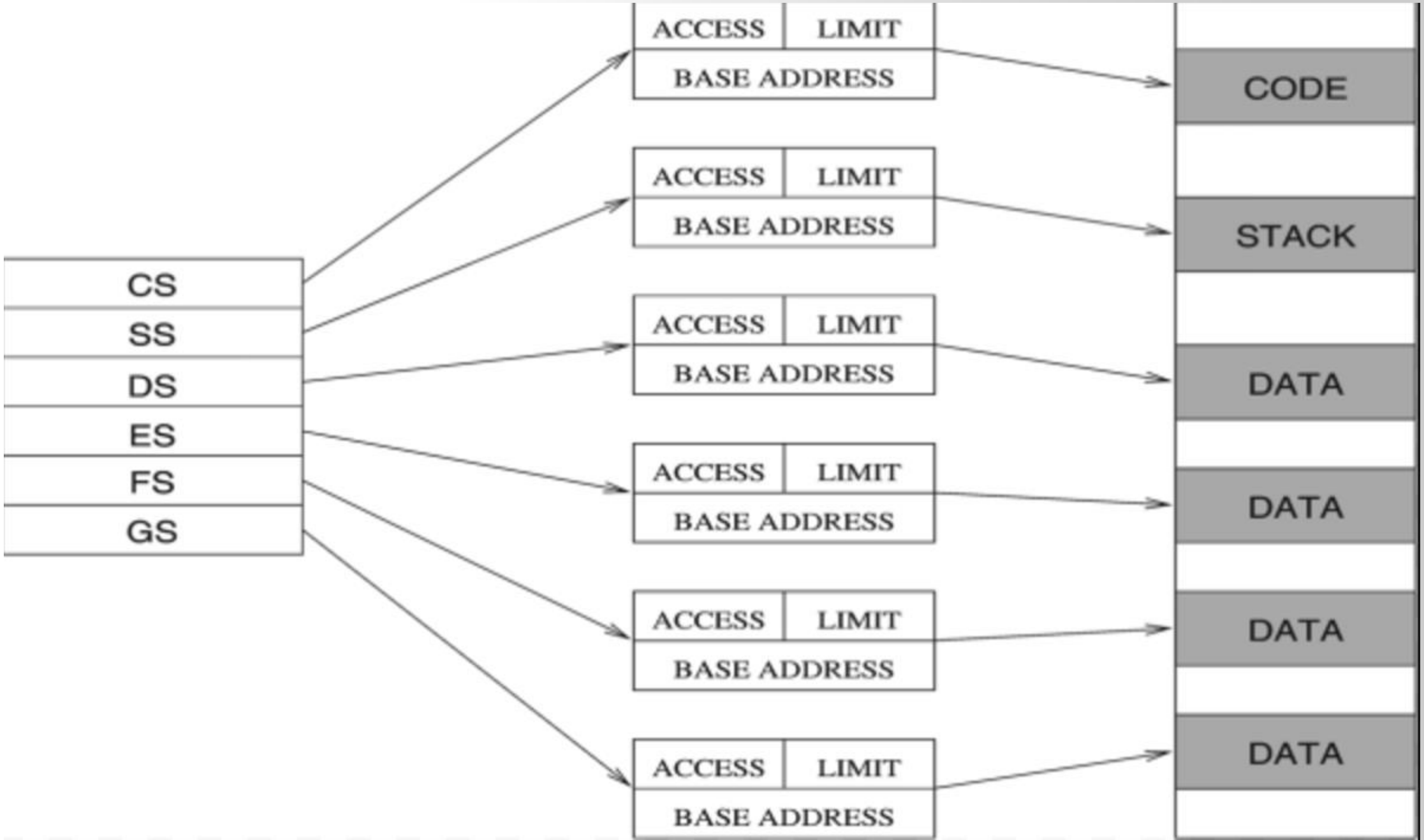
- ESI—Pointer to data in the segment pointed to by the DS register; source pointer for string operations.
- EDI—Pointer to data (or destination) in the segment pointed to by the ES register; destination pointer for string operations.
- **EBP—Pointer to data on the stack.**
- **ESP—Stack pointer.**



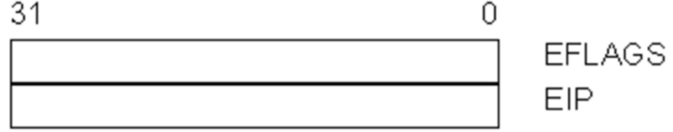
PUSH, POP, CALL, RET

Segment Registers

- There are six segment registers that hold 16-bit segment selectors. A segment selector is a special pointer that identifies a segment in memory.
  - CS: code segment register
  - SS: stack segment register
  - DS, ES, FS, GS: data segment registers

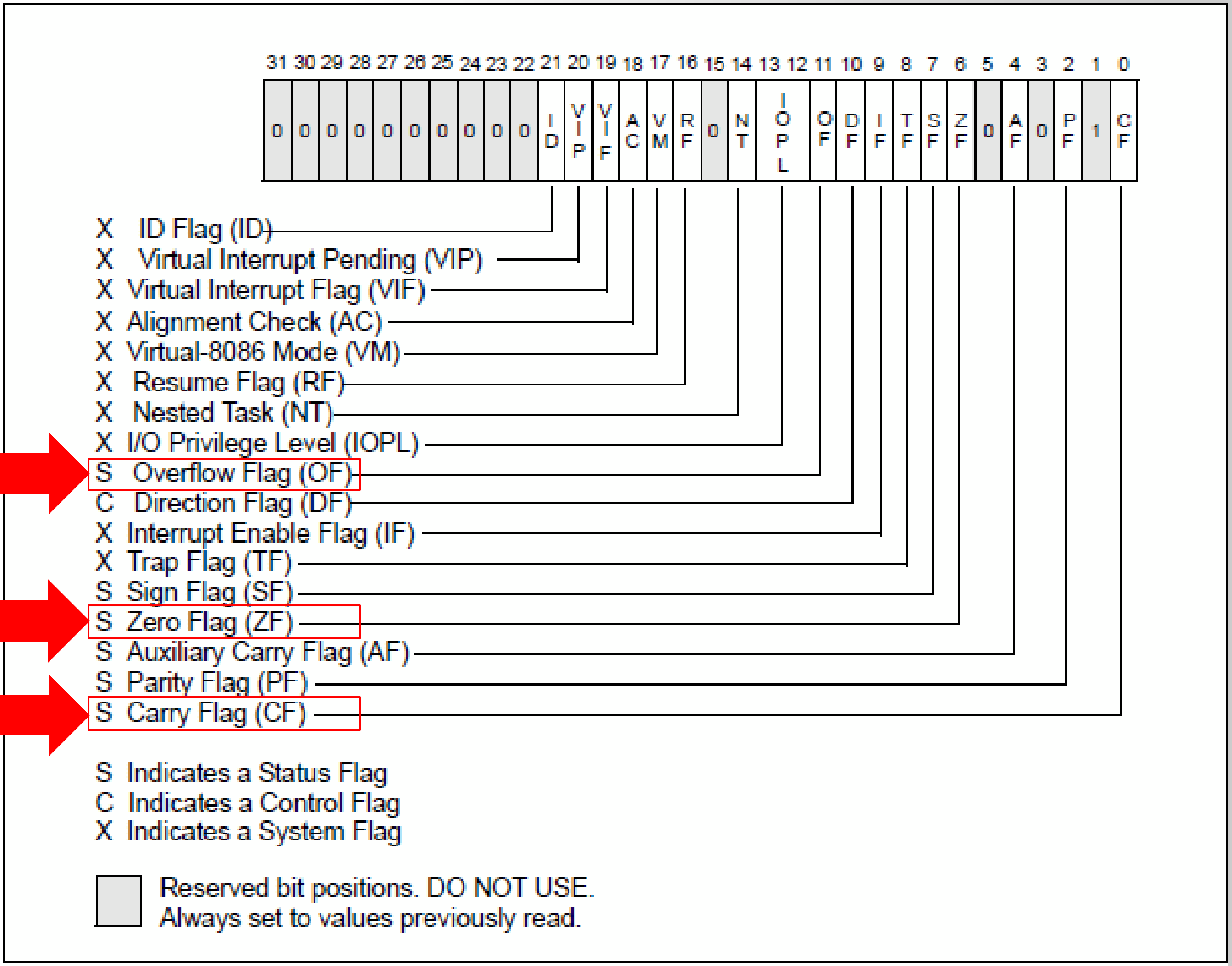


Status and Control Registers



The 32-bit EFLAGS register contains **a group of status flags**, **a control flag**, and **a group of system flags**.

JCC



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag

□ Reserved bit positions. DO NOT USE.
Always set to values previously read.

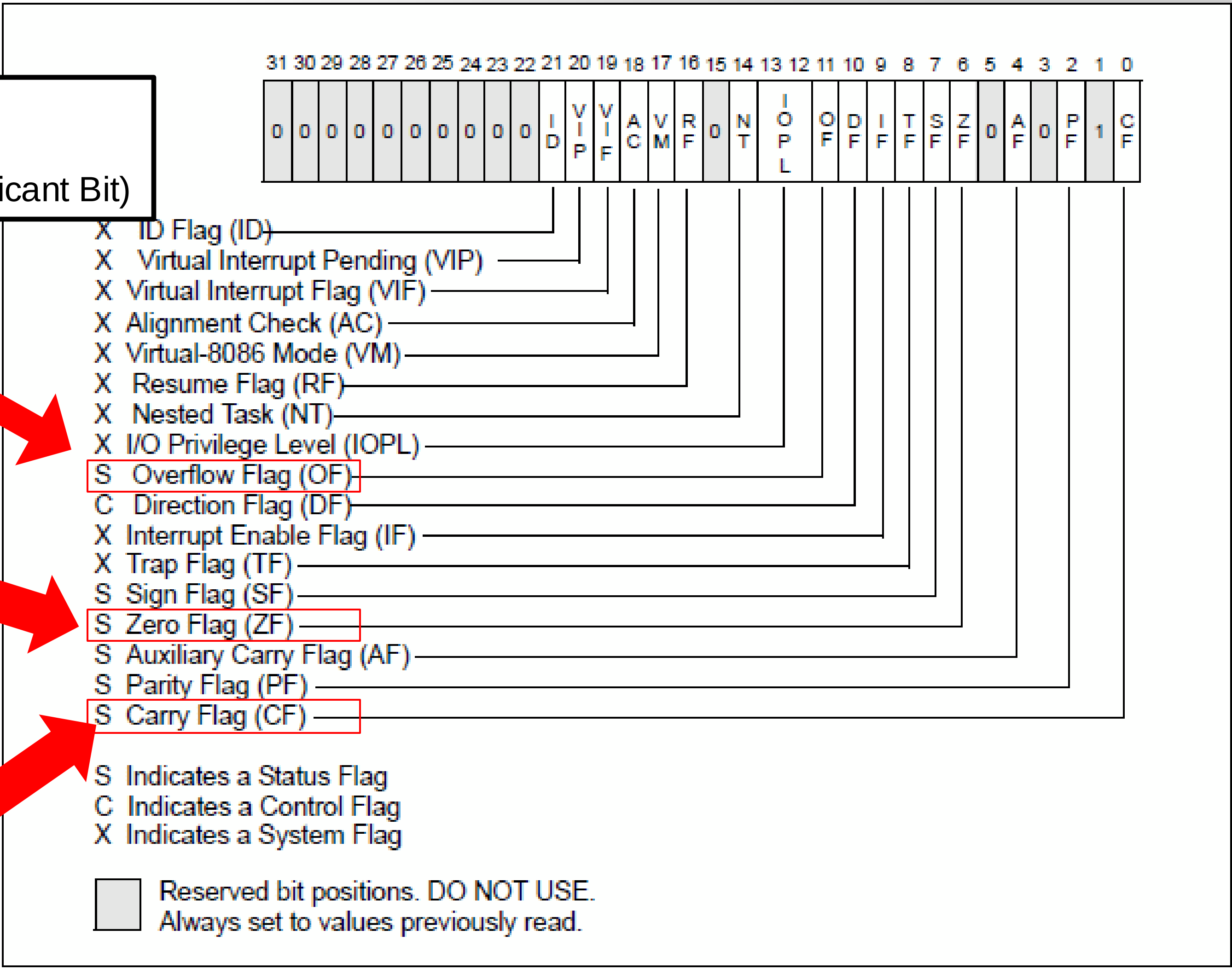
EFLAGS Register

Status and Control Registers

Change to ‘1’ if:
- Signed integer overflow
- Change in MSB (Most Significant Bit)

Change to ‘1’ if:
- Calculation result is 0

Change to ‘1’ if:
- unsigned integer overflow



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

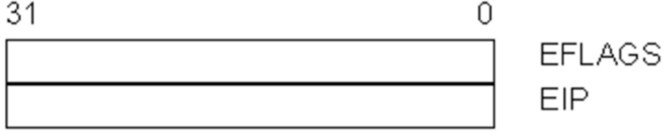
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag

▢ Reserved bit positions. DO NOT USE.
Always set to values previously read.

EFLAGS Register

Status and Control Registers



EIP Register (Instruction Pointer)

The EIP register (or instruction pointer) can also be called "**program counter**."

It contains the **offset** in the current code segment for the **next instruction to be executed**.

It is advanced from one instruction boundary to the next in straight-line code or it is moved ahead or backwards by a number of instructions when executing JMP, Jcc, CALL, RET, and IRET instructions.

Byte Order

Little endian

- IA-32 processors use "little endian" as their byte order. This means that the bytes of a word are numbered starting from the least significant byte and that the least significant bit starts of a word starts in the least significant byte.

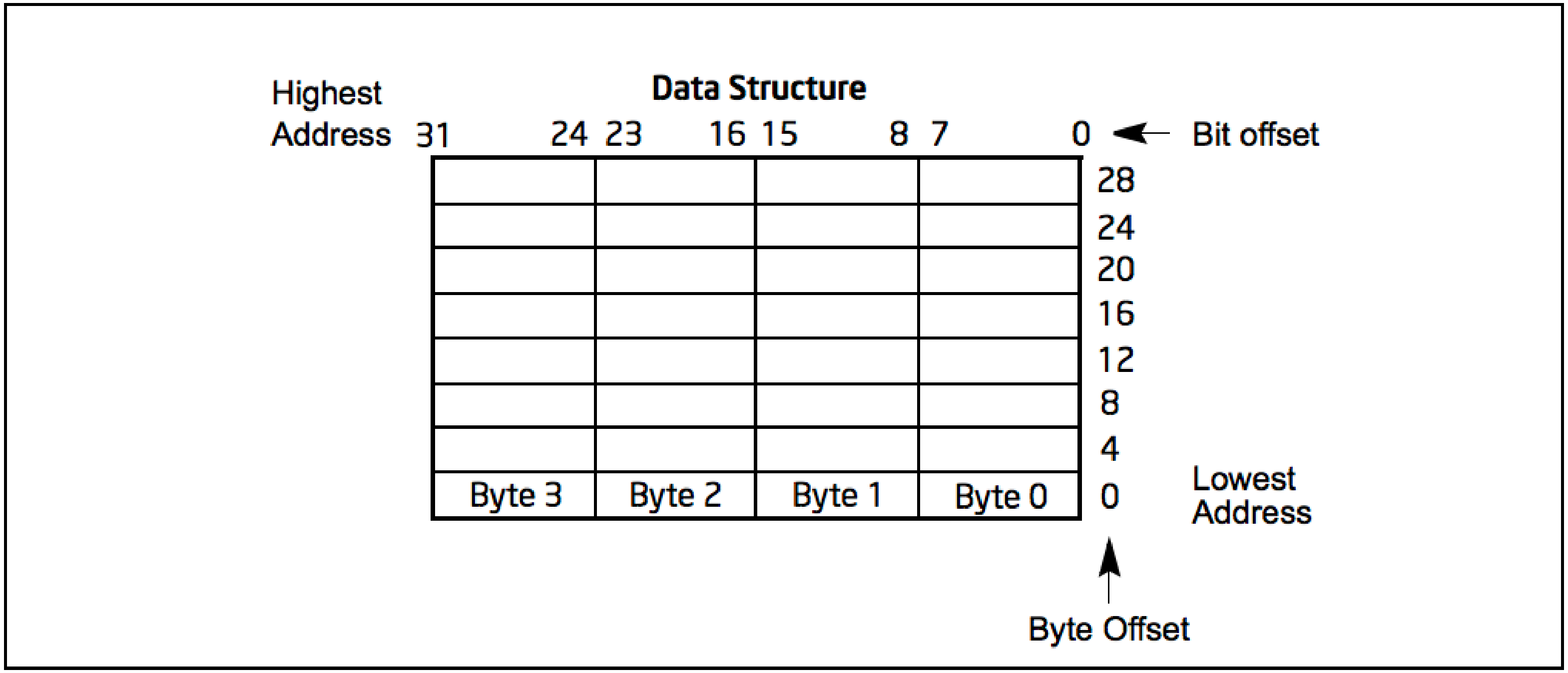
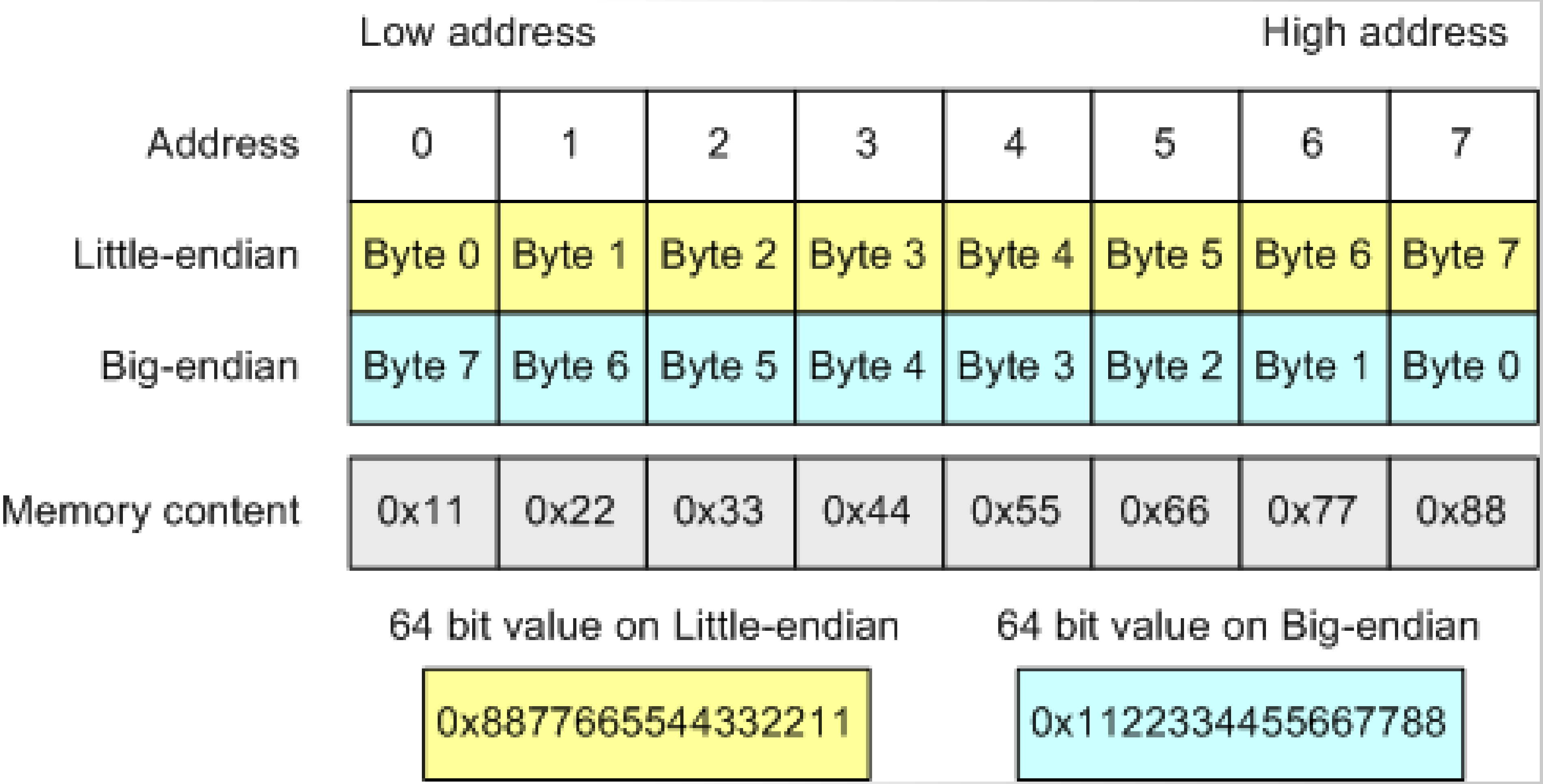


Figure 1-1. Bit and Byte Order

Byte Order

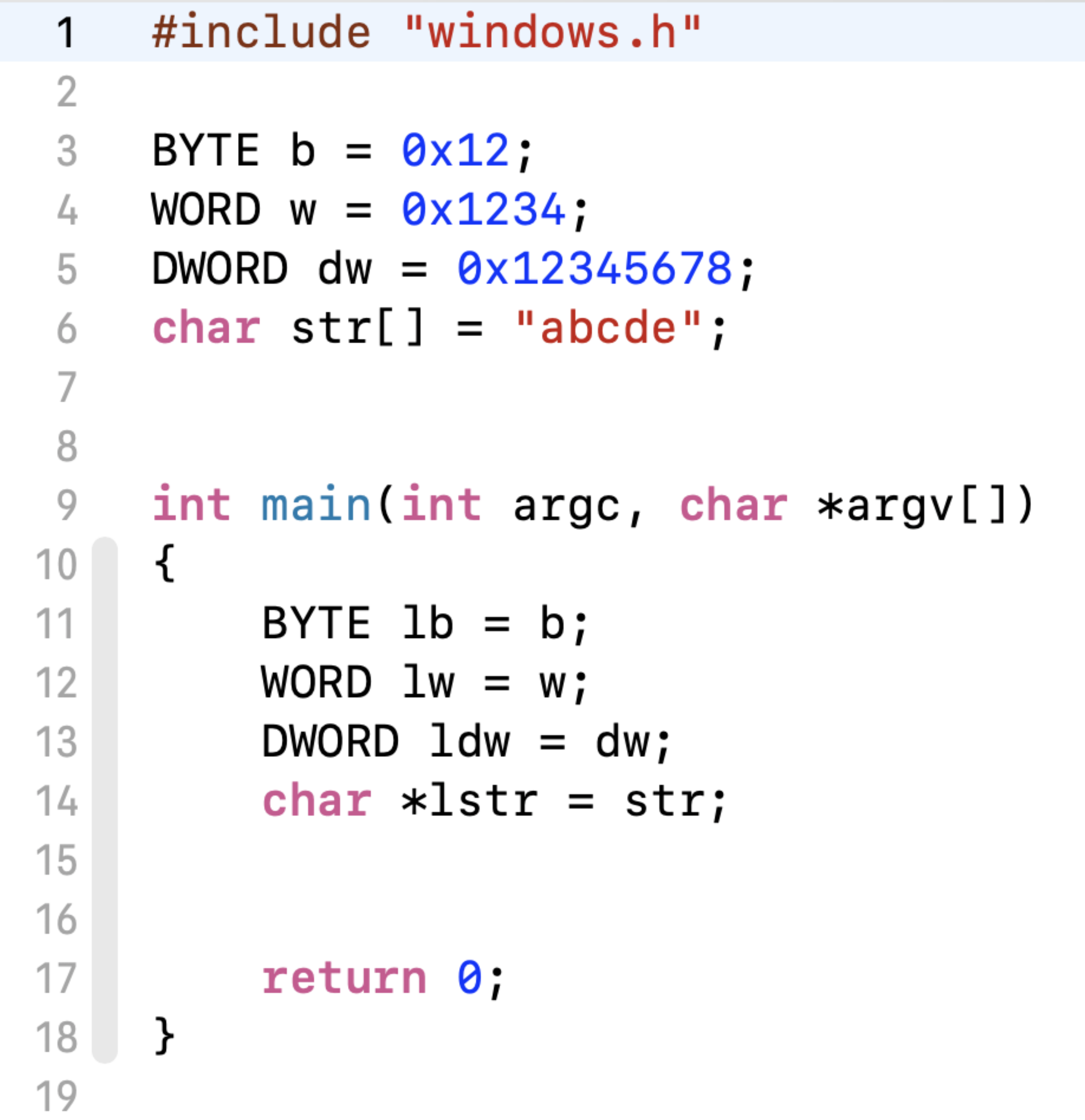


Low address   High address

Address	0	1	2	3	4	5	6	7
Little-endian	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Big-endian	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Memory content	0x11	0x22	0x33	0x44	0x55	0x66	0x77	0x88

64 bit value on Little-endian	64 bit value on Big-endian
0x8877665544332211	0x1122334455667788

LittleEndian.cpp



```cpp
#include "windows.h"

BYTE b = 0x12;
WORD w = 0x1234;
DWORD dw = 0x12345678;
char str[] = "abcde";


int main(int argc, char *argv[])
{
    BYTE lb = b;
    WORD lw = w;
    DWORD ldw = dw;
    char *lstr = str;


    return 0;
}

```

X86 ASM

MOV

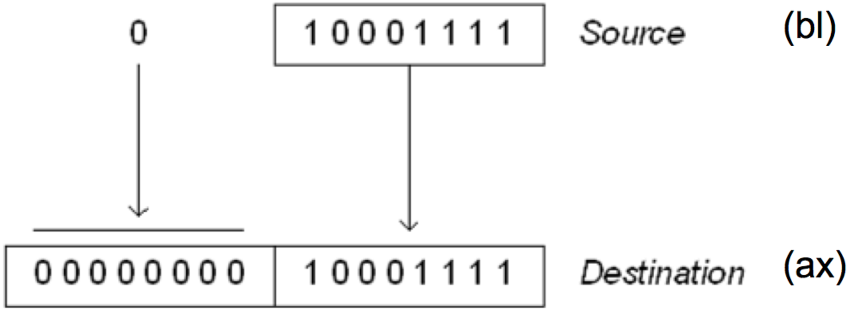
- Move **reg/mem** value to **reg/mem**
  - mov A, B is "Move B to A" (A=B)
  - Same data size

```
mov eax, 0x1337
mov bx, ax
mov [esp+4], bl
```

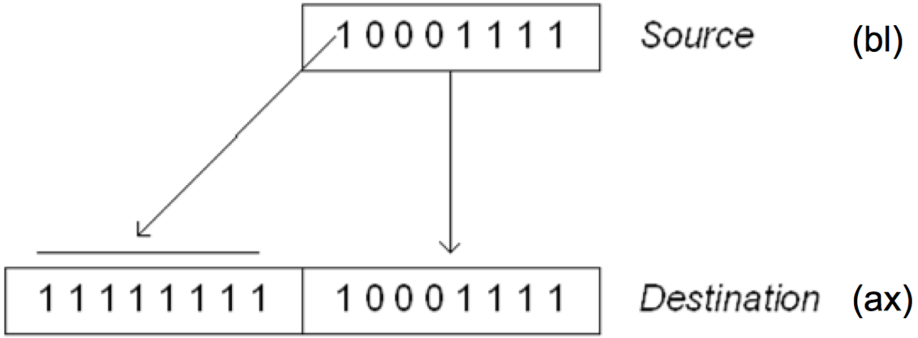
MOVZX / MOVSX

- From small register to large register
- Zero-extend (MOVZX) / sign-extend (MOVSX)
- Example: movzx ebx, al

When copy a smaller value into a larger destination, MOVZX instruction fills (extends) the upper half of the destination with zeros



MOVSX fills the upper half of the destination with a copy of the source operand's sign bit



More About Memory Access

- mov ebx, [esp + eax * 4] **Intel**
- mov (%esp, %eax, 4), %ebx **AT&T**
- mov BYTE [eax], 0x0f
  You must indicate the data size: BYTE/WORD/DWORD

ADD / SUB

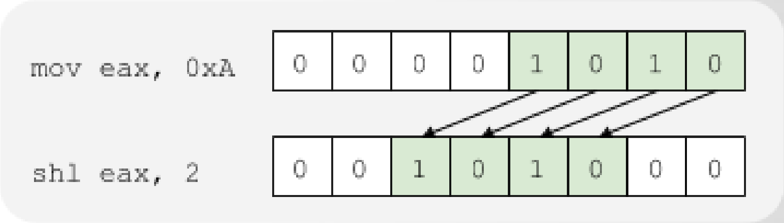
- ADD / SUB
- Normallly "reg += reg" or "reg += imm"
- Data size should be equal
  - ADD eax, ebx
  - sub eax, 123
  - sub eax, BL ; Illegal

INC / DEC

- **inc, dec** — Increment, Decrement
- The **inc** instruction increments the contents of its operand by one.
  The **dec** instruction decrements the contents of its operand by one.
- *Syntax*
  inc <reg>
  inc <mem>
  dec <reg>
  dec <mem>
- *Examples*
  DEC EAX — subtract one from the contents of EAX.
  INC DWORD PTR [var] — add one to the 32-bit integer stored at location *var*

SHL / SHR / SAR

- Shift logical left / right
- Shift arithmetic right
- Common usage: **SHL eax, 2** (when calculate memory address)

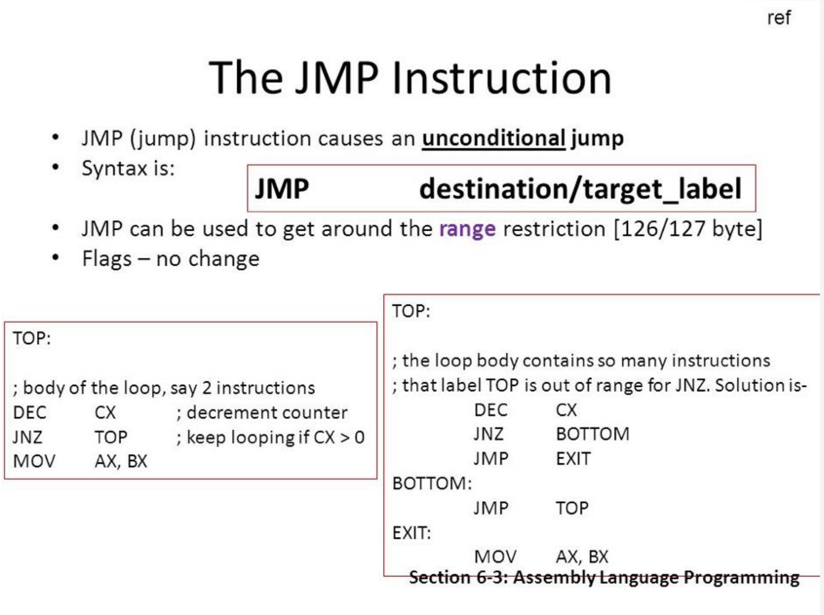


```
mov eax, 0xA
shl eax, 2
```

mov eax, 0xA	0	0	0	0	1	0	1	0
shl eax, 2	0	0	1	0	1	0	0	0

Jump

- Unconditional jump: jmp
- Conditional jump: je/jne
  and ja/jae/jb/jbe/jg/jge/jl/jle ...
- Sometime with ”cmp A, B” -- compare these two values and set eflags
- Conditional jump is decided by some of the eflags bits.



ref

The JMP Instruction

- JMP (jump) instruction causes an **unconditional jump**
- Syntax is:

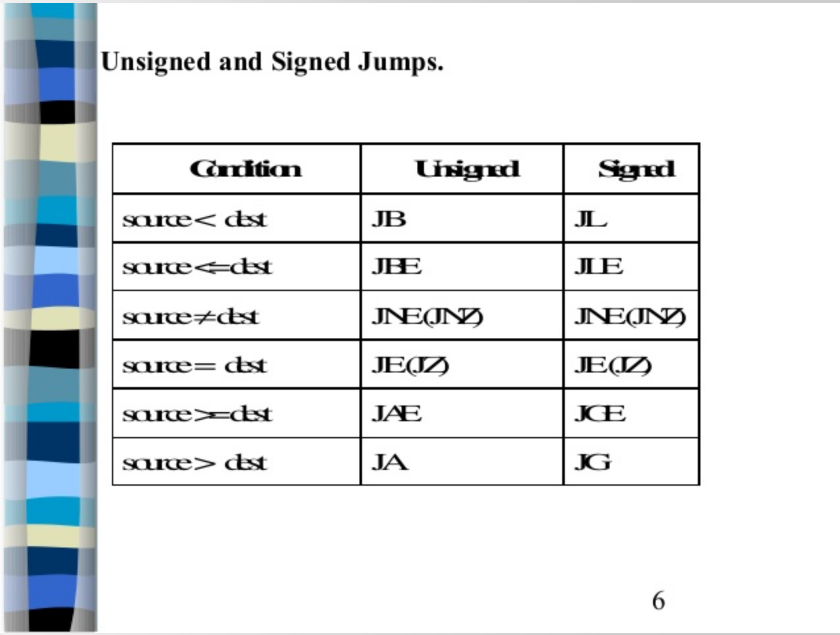
JMP destination/target_label

- JMP can be used to get around the **range** restriction [126/127 byte]
- Flags – no change

```
TOP:

; body of the loop, say 2 instructions
DEC     CX      ; decrement counter
JNZ     TOP     ; keep looping if CX > 0
MOV     AX, BX
```

```
TOP:

; the loop body contains so many instructions
; that label TOP is out of range for JNZ. Solution is-
        DEC     CX
        JNZ     BOTTOM
        JMP     EXIT
BOTTOM:
        JMP     TOP
EXIT:
        MOV     AX, BX
```

Section 6-3: Assembly Language Programming



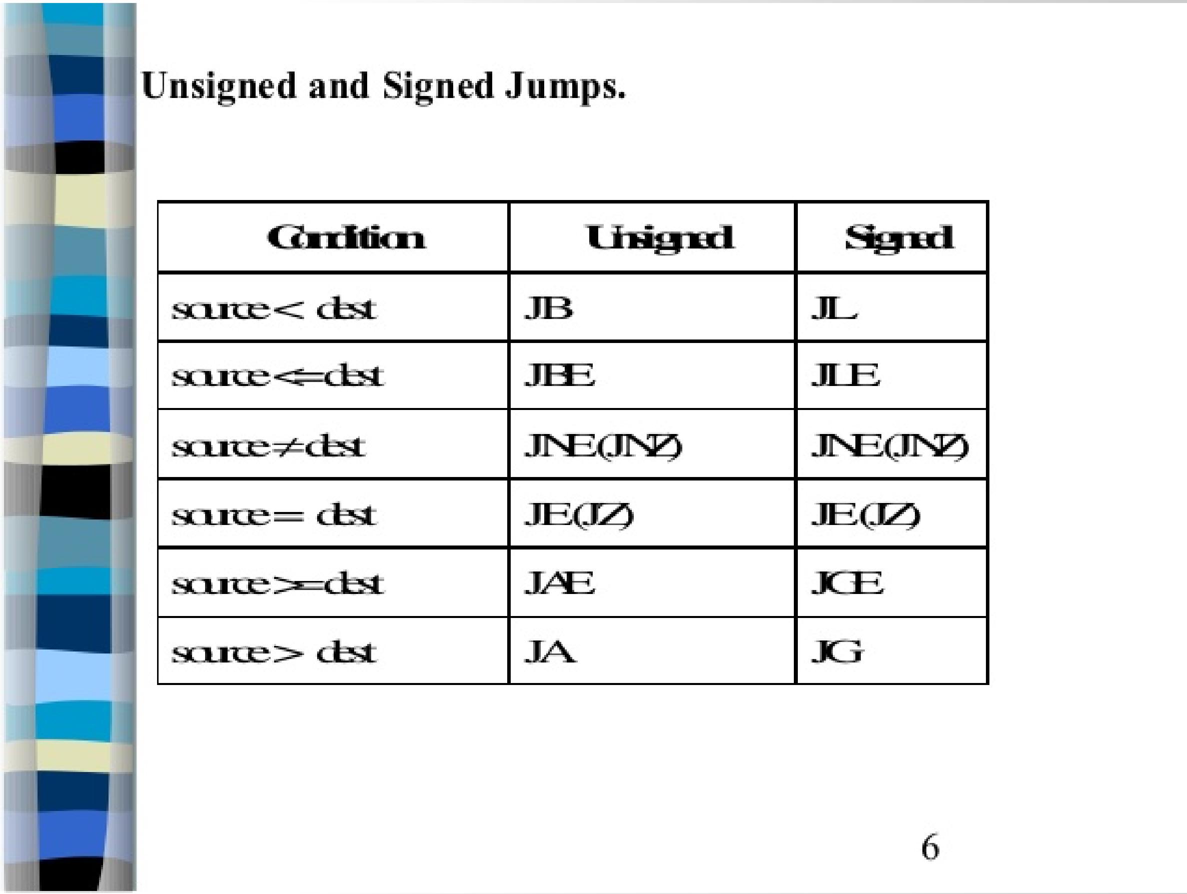
Unsigned and Signed Jumps.

Condition	Unsigned	Signed
source < dest	JB	JL
source <= dest	JBE	JLE
source ≠ dest	JNE(JNZ)	JNE(JNZ)
source = dest	JE(JZ)	JE(JZ)
source >= dest	JAE	JGE
source > dest	JA	JG

6

Jump

- ja/jae/jb/jbe are unsigned comparison
- jg/jge/jl/jle are signed comparison



Unsigned and Signed Jumps.

Condition	Unsigned	Signed
source< dest	JB	JL
source<=dest	JBE	JLE
source≠dest	JNE(JNZ)	JNE(JNZ)
source= dest	JE(JZ)	JE(JZ)
source>=dest	JAE	JGE
source> dest	JA	JG

6

CMP

- **cmp** — Compare
- Compare the values of the two specified operands, setting the condition codes in the machine status word appropriately. This instruction is equivalent to the sub instruction, except the result of the subtraction is discarded instead of replacing the first operand. *Syntax*
  cmp <reg>,<reg>
  cmp <reg>,<mem>
  cmp <mem>,<reg>
  cmp <reg>,<con>
- *Example*
  cmp DWORD PTR [var], 10
  jeq loop
- If the 4 bytes stored at location *var* are equal to the 4-byte integer constant 10, jump to the location labeled *loop*.

Q & A