# Assembly Programming

## - part one

Guo Yu

2013.12

School of Software Engineering
USTC-Suzhou

- QQ:  26271544
- QQ Group: 278210340

## Outline

Intel x86 platform

Data representation

Instruction set

Instruction Encoding

Assembler & Disassembler

## Intel x86 Architecture

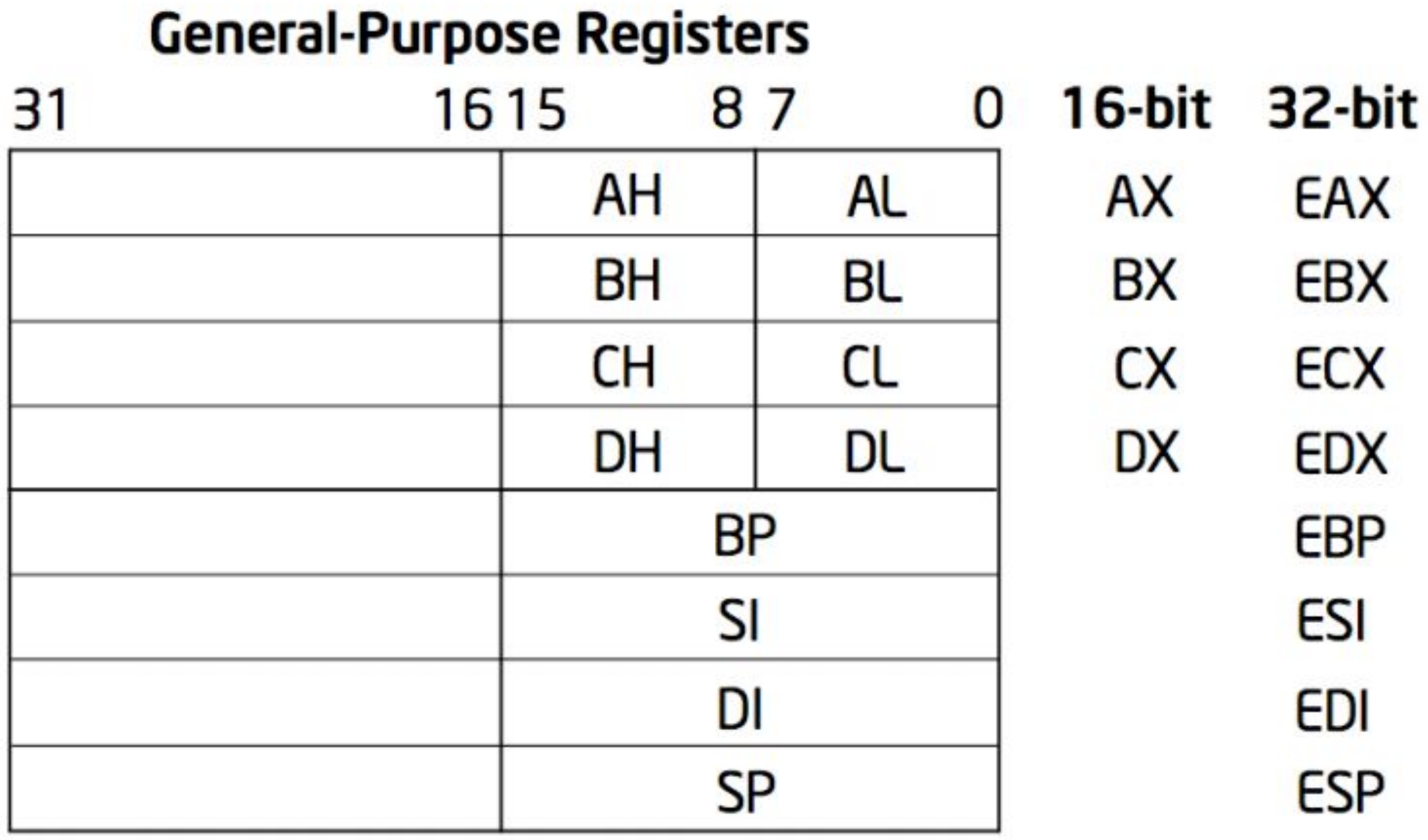
## Intel x86 Architecture

- 1978, 8086
  - First x86 microprocessor, 16-bit
- 1985, 80386
  - 32-bit
- 1989, 80486
- 1993, Pentium
  - MMX
- 2000, Pentium 4
  - Deeply pipelined, high frequency
- 2006, Intel Core 2
  - Low power, multi-core

## Intel x86 Architecture

- x86
  - i386, x86_32
  - AMD64, x86_64, EM64T
- OS
  - MS-DOS, Windows, Linux, BSD, Solaris, Mac OS X

# General Purpose Registers



# General Purpose Registers

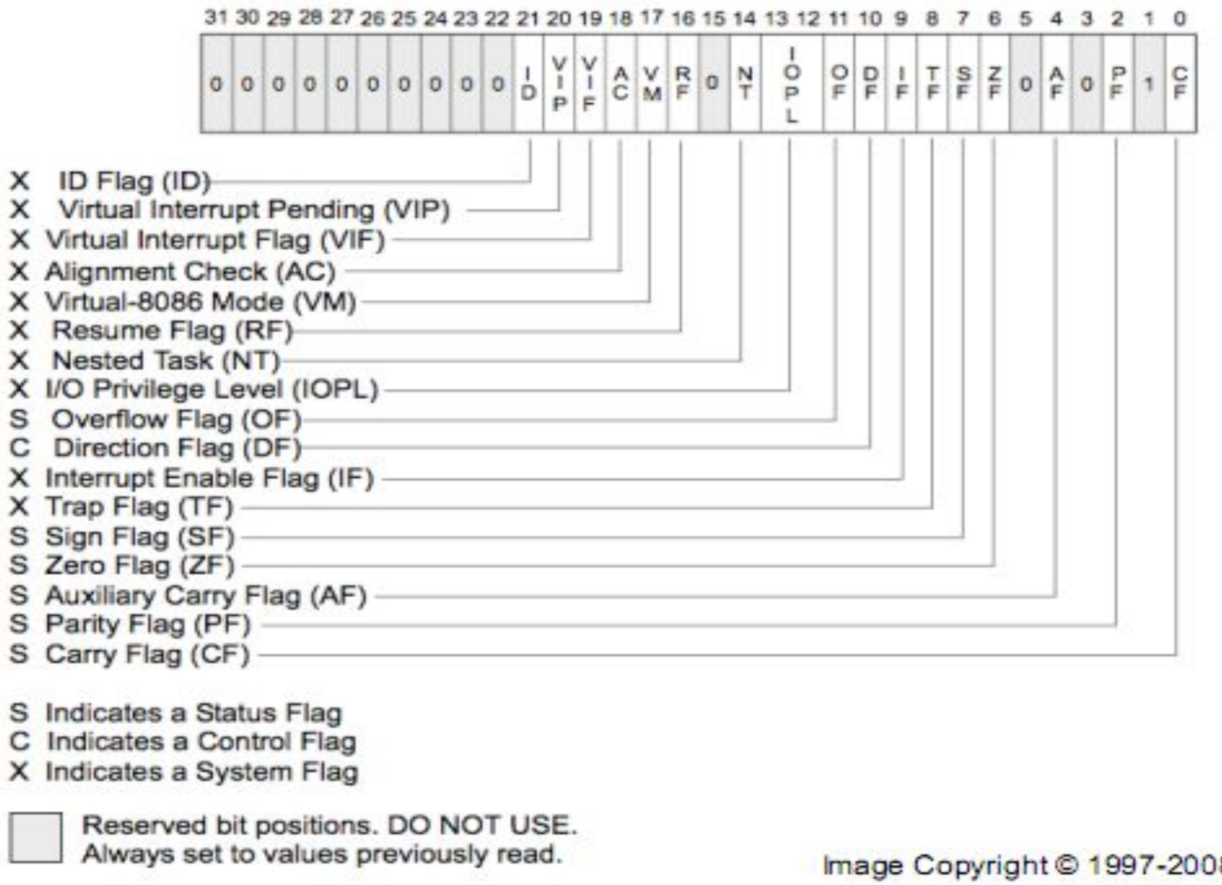
- EAX, accumulator
- EBX, base
- ECX, counter
- EDX, data/general
- ESI, source index for string operation
- EDI, destination index for string operation
- ESP, stack pointer for top address of the stack
- EBP, stack base pointer current stack frame

- EIP, instruction pointer ← *Not General*

## x86 mode

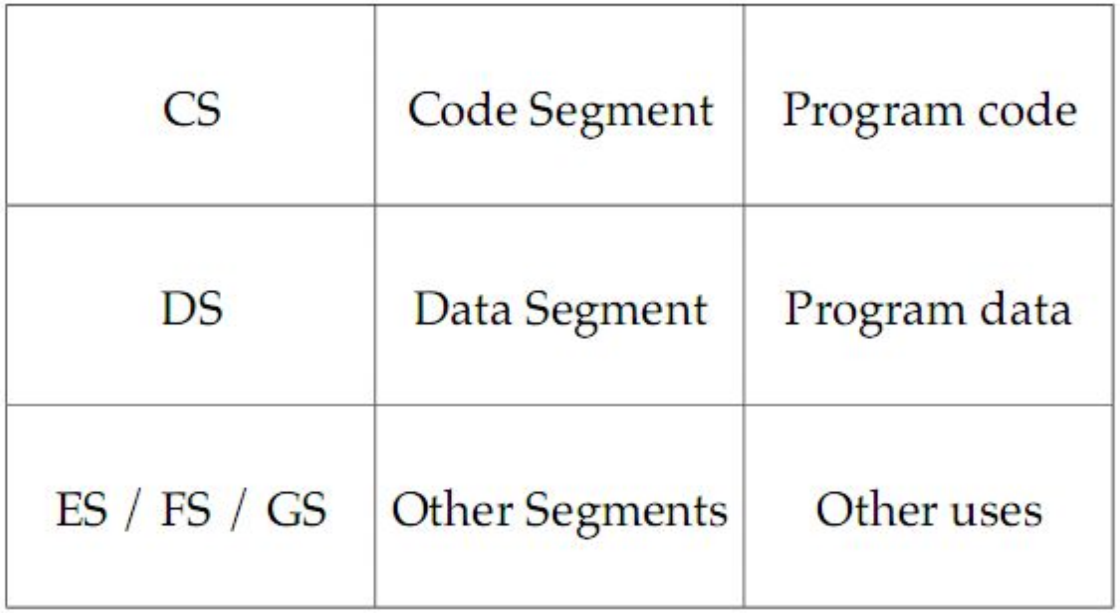
- Real mode
  - After machine power-on
  - 20-bit memory address, 1M
  - MS-DOS, OS bootloader
- Protected mode
  - Read, write, execution
  - Segmentation protection
  - Privilege-level protection
  - Paging and virtual memory
  - 32-bit memory address, 4G
  - Linux, Windows, FreeBSD …

## Flag Register


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.

## Segment Registers

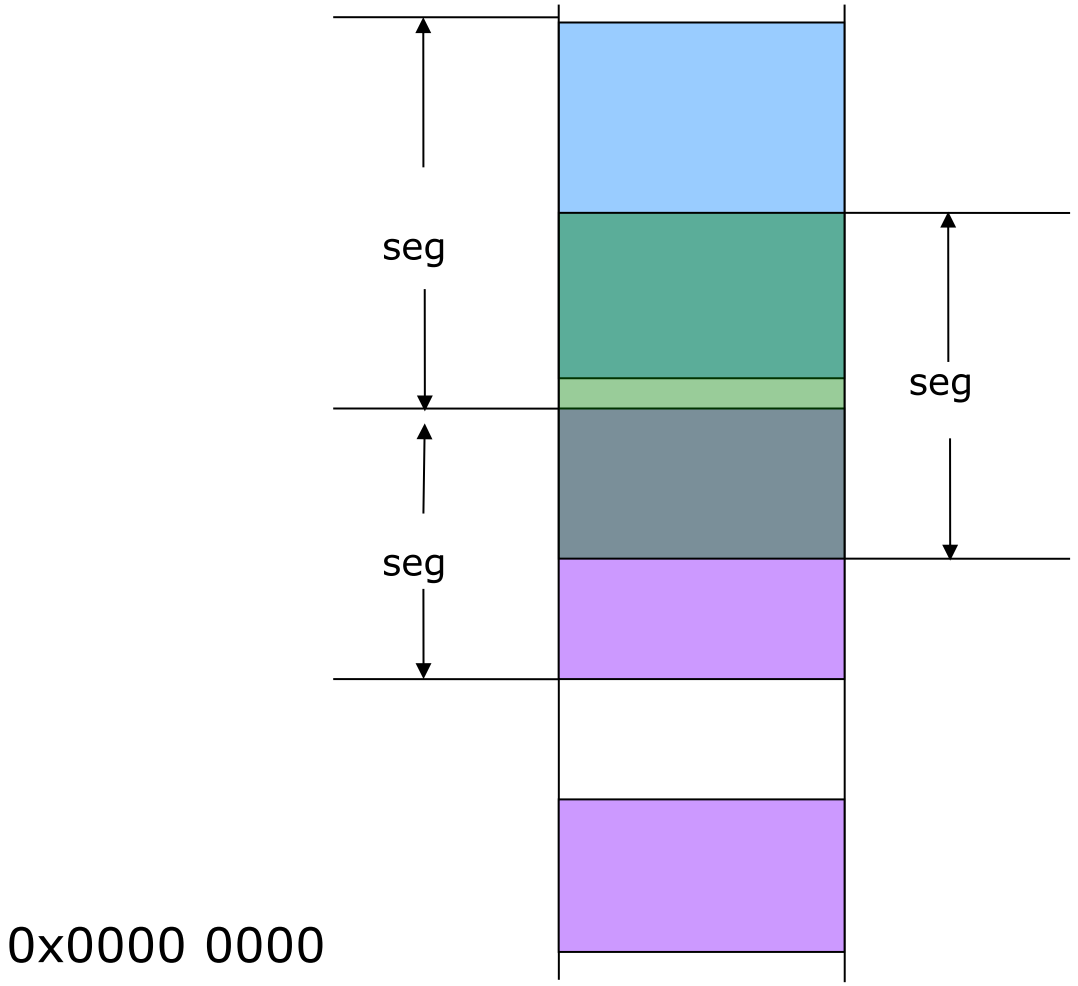
- CS, DS, SS, ES, FS, GS
  - 16-bit



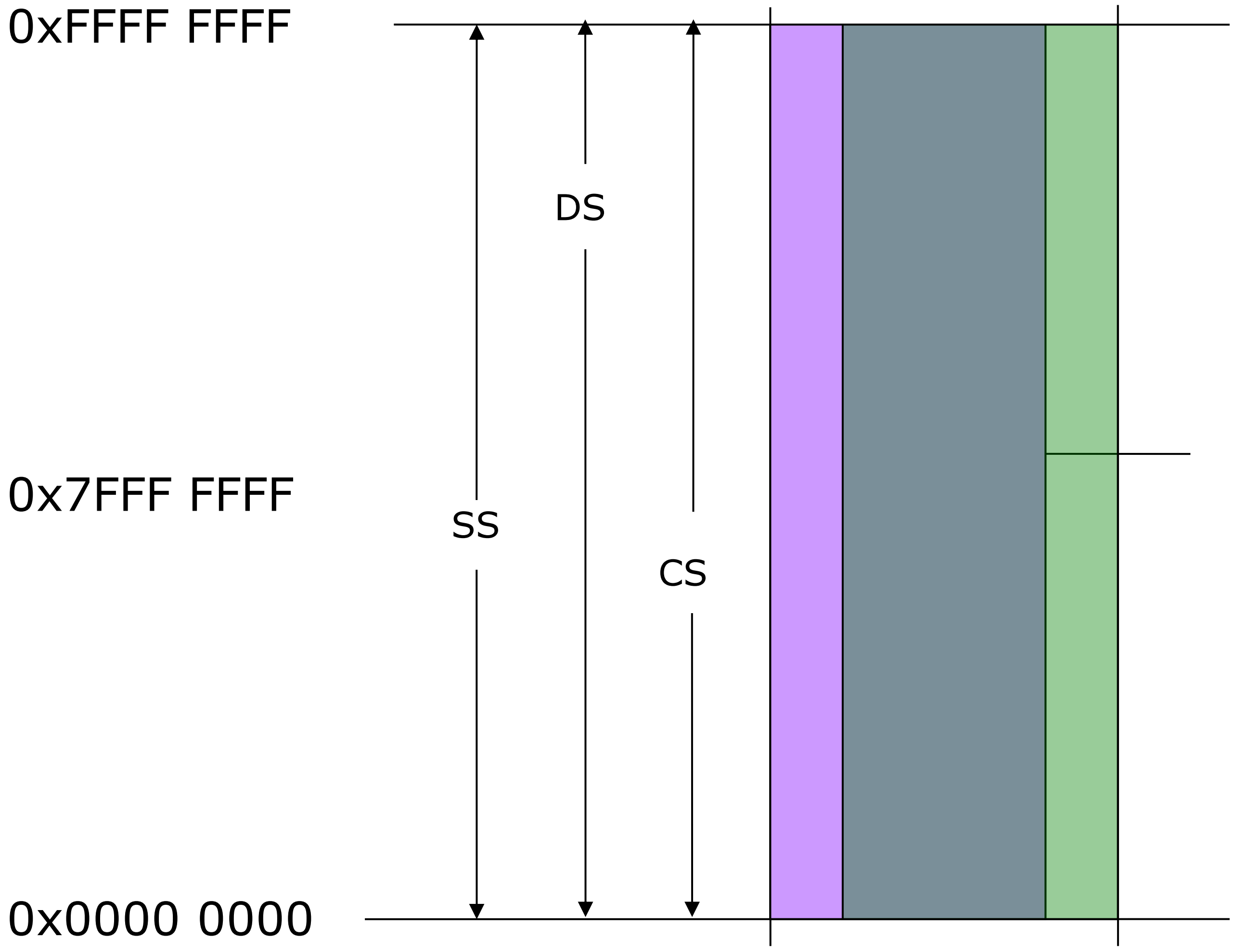
| | | |
|---|---|---|
| CS | Code Segment | Program code |
| DS | Data Segment | Program data |
| ES / FS / GS | Other Segments | Other uses |

# Segmentation

- Overlapped
- Different privilege level
- Length limit
- Protection mode



# In Windows NT



## Other Registers

- Control Registers
  - CR0~CR3
- Debug Registers
  - DR0~DR3, DR6, DR7
- Test Registers
  - TR4~TR7
- Descriptor Registers
  - GDTR, LDTR, IDTR
- Task Register
  - TR

## Outline

Intel x86 platform

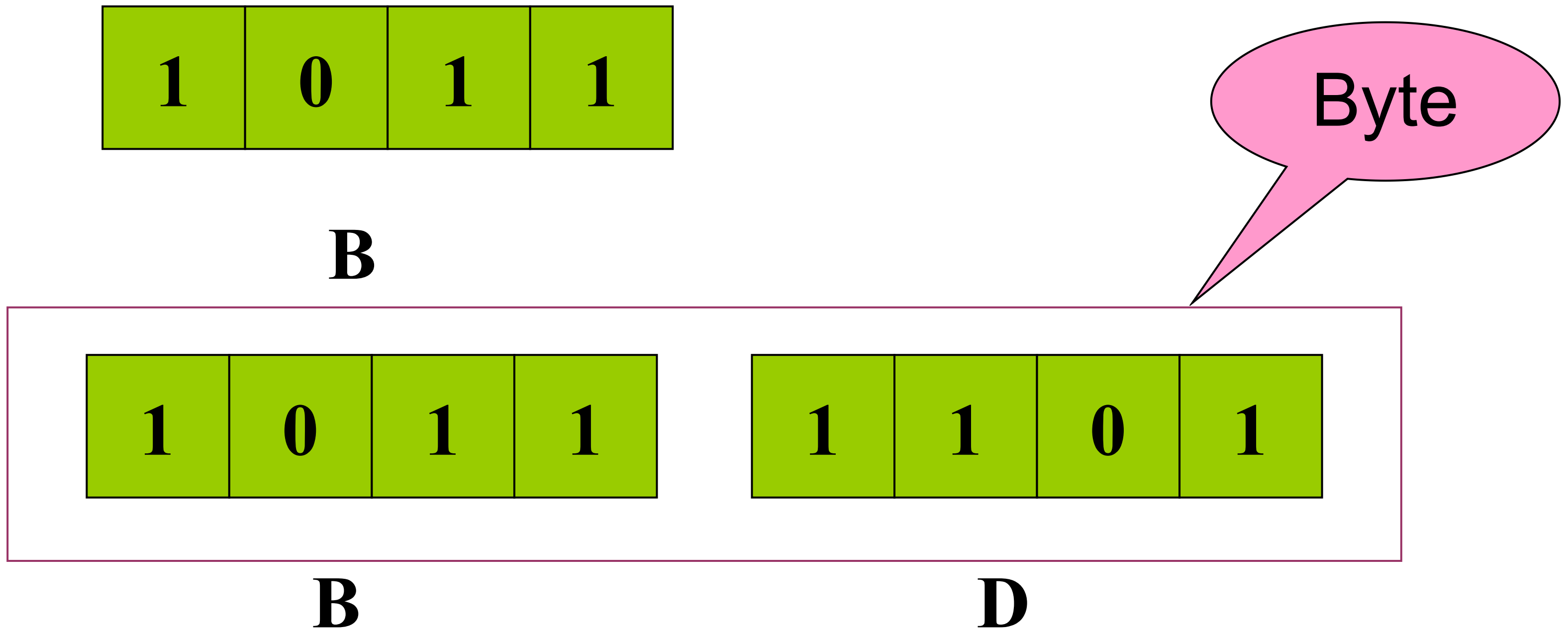
Data representation

Instruction set

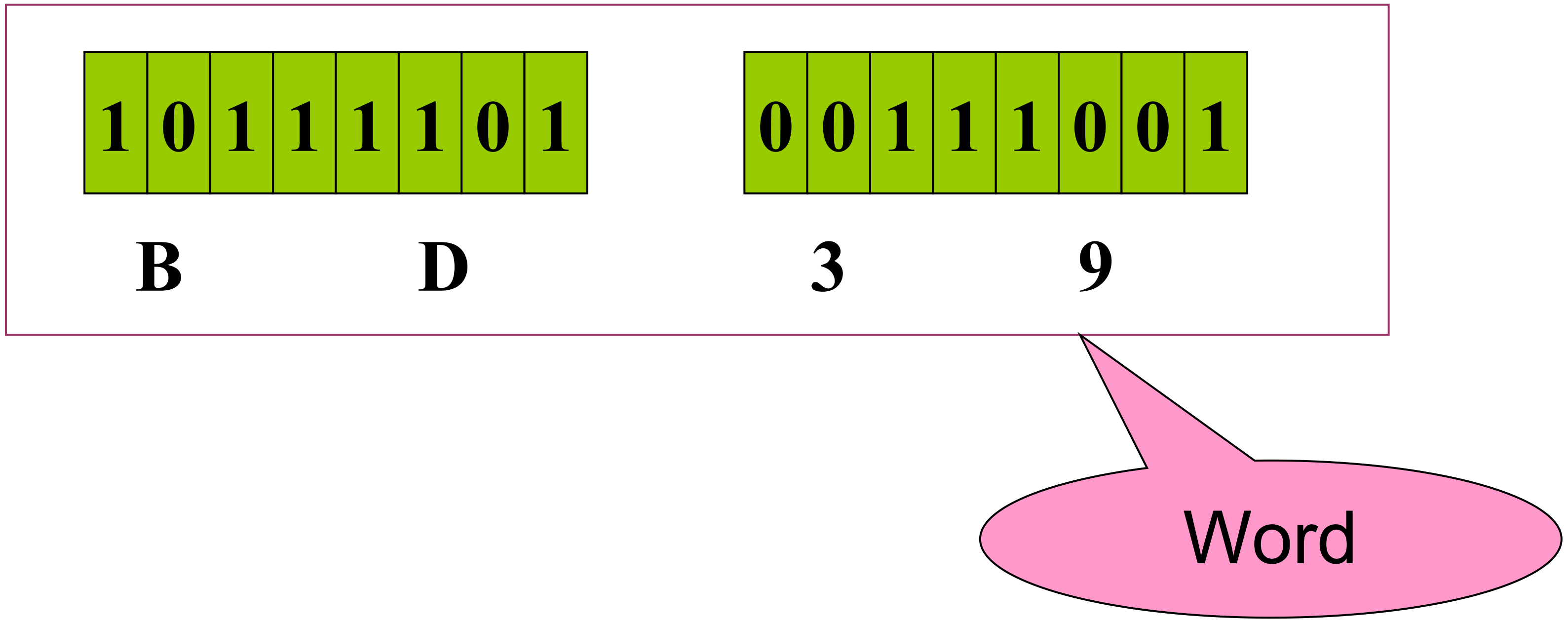
Instruction Encoding

Assembler & Disassembler

# Binary Number

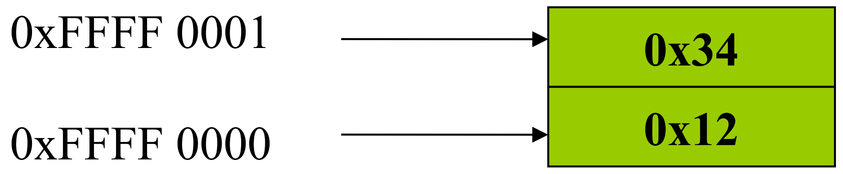


# Binary Number



## Byte Order

- Little Endian
  - 0x3412
- Big Endian
  - 0x1234

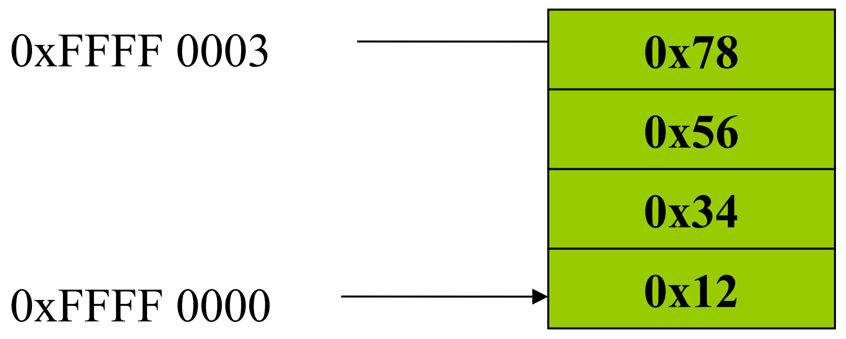


## Byte Order Matters

- Data exchange between computers
- Networking protocols
- File formats in the disk storage

## Little Endian DWord

- 0x 7856 3412



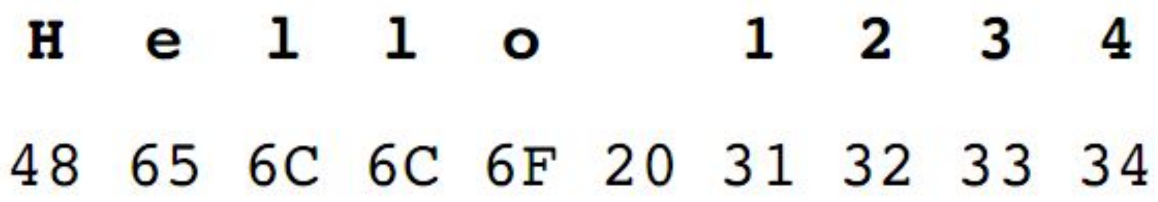
## System Endianness

| Little Endian | Big Endian | Switchable Endianness |
|---|---|---|
| Intel x86 | PowerPC (exc. G5) | ARM |
| Intel 8051 | Sparc (exc. v9) | Alpha |
| Most uControllers | System/370 | Intel IA64 |

# ASCII Code

| | | |
|---|---|---|
| 0x00 - 0x1F | Control Characters | Backspace, Line feed |
| 0x20 - 0x3F | Digits and Punctuation | 0-9 <> = .,: *-()! |
| 0x40 - 0x5F | Upper-case Letters and Special | ABCD... @[]\^_ |
| 0x60 - 0x7E | Lower-case Letters and Special | abcd... `{}\|~ |

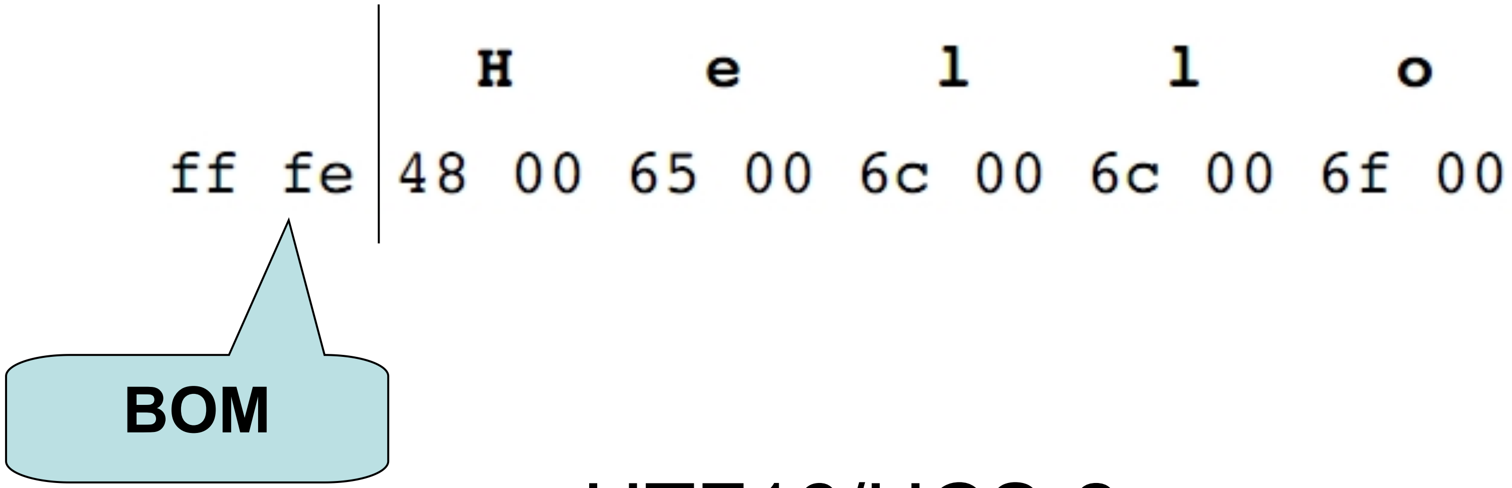
## ASCII Example



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| H | e | l | l | o | | 1 | 2 | 3 | 4 |
| 48 | 65 | 6C | 6C | 6F | 20 | 31 | 32 | 33 | 34 |

- http://en.wikipedia.org/wiki/ASCII

# Unicode String

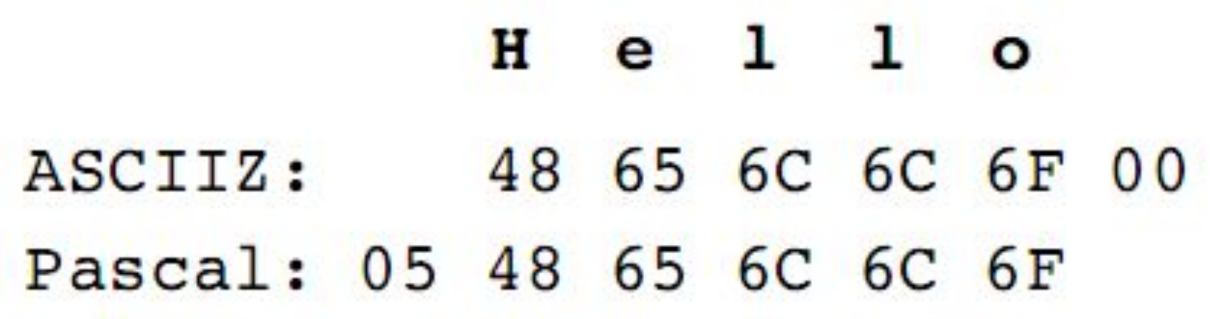


```
        H     e     l     l     o
ff fe | 48 00 65 00 6c 00 6c 00 6f 00
```

UTF16/UCS-2

# String Storage

- ASCIIZ
  - Zero-terminated ASCII
- Pascal
  - Size Byte + ASCII String
- Delphi
  - Size DWord + ASCII or Unicode String



```
             H  e  l  l  o
ASCIIZ:      48 65 6C 6C 6F 00
Pascal:   05 48 65 6C 6C 6F
```

## Outline

Intel x86 platform

Data representation

Instruction set

Instruction Encoding

Assembler & Disassembler

## Intel x86 Instruction Set

- CISC
  - Complex instruction set computer
  - Moto68k
- RISC
  - Reduced instruction set computer
  - ARM, MIPS, Sun SPARC, IBM PowerPC, DEC Alpha

## Assumption

- Operating in protected mode
- With 32-bit addressing in effect
- Intel Syntax (Windows)
  - AT&T Syntax (GNU Tools, Linux)

## Basic Arithmetic

- add opand1, opand2
  - opand1 = opand1 + opand2
- sub opand1, opand2
  - opand1 = opand1 – opand2
- cmp opand1, opand2
  - cmp reg mem, cmp reg reg, cmp mem reg, cmp reg imm
  - opand1 – opand2
- inc
  - inc reg
  - reg = reg + 1
- dec
  - dec reg
  - reg = reg - 1

# Logical Instructions

- and operand1, operand2
  - and reg, reg
  - and reg, mem
  - and reg, imm
  - operand1 = operand1 “and” operand2
- or, xor, not
  - xor eax, eax
- test operand1, operand2
  - operand1 “and” operand2

## Control Transfer

- jmp/je/jne  imm/reg
  - jmp rel
    - eip = eip + rel
  - jmp reg
    - eip = reg
  - jmp [reg]
    - eip = [reg]
- call imm/reg/mem
  - esp = esp – 4;  [esp] = eip; eip +=opand
- ret
  - eip = [esp]; esp = esp + 4
- int imm
  - Soft Interrupt

## Data Movement

- mov dst, src
  - mov reg, mem
  - mov mem, reg
  - mov reg, imm
  - mov reg, reg
  - mov mem, imm
- push reg/mem/imm
- pop reg/mem
- lea reg, mem

# Other Instructions

- See Intel manual

Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 3A:
System Programming Guide,
Part 1

Volume 3B:
System Programming Guide, Part
2

## Now,

- Let’s write a piece of code
  - printing “hello world”

- Wait, something is missing
  - Where does the code put temporary variables?
  - How does the code output strings?

# C Inline Assembly

- Microsoft C/C++ Compiler

```
__asm{ mov ebx, eax}
```

2 underline symbols

- GNU/Gcc Compiler

```
__asm__ ("movl %eax, %ebx\n\t" "… ");
```

## Hello world

```
#include <stdio.h>
int main()
{
   char *hw="hello world";
   __asm
   {
     push     hw
     call     printf
     add      esp, 4
   }
   return 0;
}
```

## X + Y = Z

```
#include <stdio.h>
int main()
{
  int x, y, z;
  x = 1;
  y = 2;
  __asm
  {
      mov     eax, dword ptr [x]
      add     eax, dword ptr [y]
      mov     dword ptr [z], eax
  }
  printf("%d + %d = %d", x, y, z);
  return 0;
}
```

## X - Y = X

```
#include <stdio.h>
int main()
{
  int x y z;
  x = 9;
  y = 2;
  __asm
  {
    mov      eax, dword ptr [x]
    mov      ebx, dword ptr [y]
    sub      eax, ebx
    mov      dword ptr [z], eax
  }
  printf("%d-%d=%d",x,y,z);
  return 0;
}
```

## Jumping

```
#include <stdio.h>
int main()
{
  int x = 9;
  __asm
  {
    jmp label
  }
  x = 6;
  __asm
  {
label:
    nop
  }
  printf("%d",x);
  return 0;
}
```

## Saving eax

```
#include <stdio.h>
int main()
{
  int x;
  __asm
  {
    mov eax, 9
    mov dword ptr [x], eax
  }
  printf("%d\n",x);
  __asm
  {
    mov dword ptr [x], eax
  }
  printf("%d",x);
  return 0;
}
```

# Saving eax

```
#include <stdio.h>
int main()
{
  int x;
  __asm
  {
    mov eax, 9
    mov dword ptr [x], eax
  }
  printf("%d\n",x);
  __asm
  {
    mov dword ptr [x], eax
  }
  printf("%d",x);
  return 0;
}
```

eax is changed

Quiz: Why?

# Saving eax

```
#include <stdio.h>
int main()
{
  int x;
  __asm
  {
    mov eax, 9
    mov dword ptr [x], eax
    push eax
  }
  printf("%d\n",x);
  __asm
  {
    pop eax
    mov dword ptr [x], eax
  }
  printf("%d",x);
  return 0;
}
```

Save eax

Restore eax

## Printing Address

```
#include <stdio.h>
int main()
{
  int x = 1;
  unsigned int y = 0;
  __asm
  {
    lea eax, dword ptr [x]
    mov dword ptr [y], eax
  }
  printf("the address of x: %p\n", y);

  return 0;
}
```

## Outline

Intel x86 platform

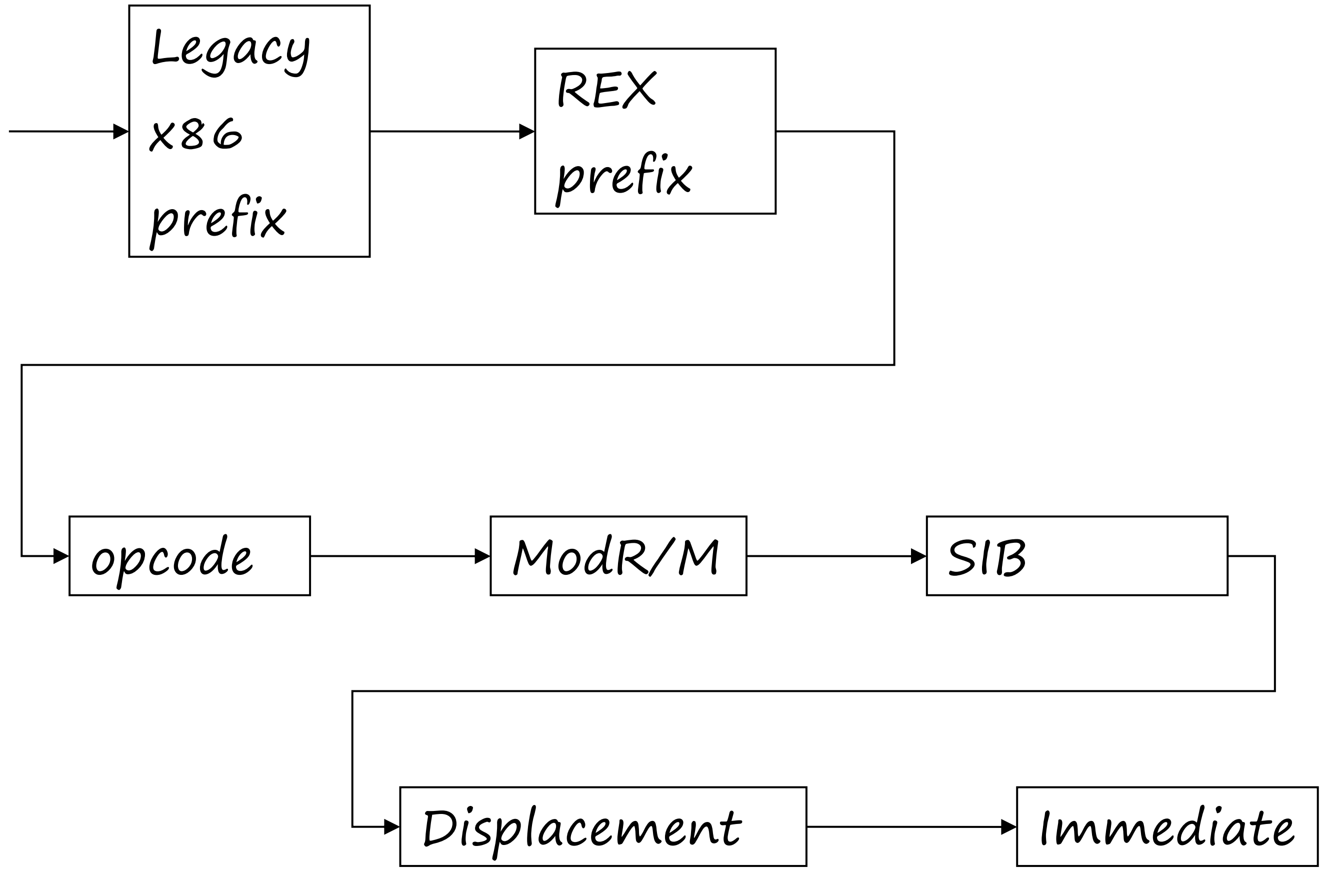
Data representation

Instruction set

Instruction Encoding

Assembler & Disassembler

# Instruction Encoding



# Prefix

- instruction prefix
  - lock prefix
  - rep prefix
  - segmentation override prefix
- Oprand and Address size prefix
  - 32bit default
- 64-bit prefix

# ModR/M Table (1)

| r32(/r) | | | EAX | ECX | EDX | EBX | ESP | EBP | ESI | EDI |
|---|---|---|---|---|---|---|---|---|---|---|
| /digit (Opcode) | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| REG = | | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +---Address--+ | +Mod | R/M-+ | +---------ModR/M Values in Hexadecimal----------+ | | | | | | | |
| [EAX] | | 000 | 00 | 08 | 10 | 18 | 20 | 28 | 30 | 38 |
| [ECX] | | 001 | 01 | 09 | 11 | 19 | 21 | 29 | 31 | 39 |
| [EDX] | | 010 | 02 | 0A | 12 | 1A | 22 | 2A | 32 | 3A |
| [EBX] | | 011 | 03 | 0B | 13 | 1B | 23 | 2B | 33 | 3B |
| [--] [--] | 00 | 100 | 04 | 0C | 14 | 1C | 24 | 2C | 34 | 3C |
| disp32 | | 101 | 05 | 0D | 15 | 1D | 25 | 2D | 35 | 3D |
| [ESI] | | 110 | 06 | 0E | 16 | 1E | 26 | 2E | 36 | 3E |
| [EDI] | | 111 | 07 | 0F | 17 | 1F | 27 | 2F | 37 | 3F |
| disp8[EAX] | | 000 | 40 | 48 | 50 | 58 | 60 | 68 | 70 | 78 |
| disp8[ECX] | | 001 | 41 | 49 | 51 | 59 | 61 | 69 | 71 | 79 |
| disp8[EDX] | | 010 | 42 | 4A | 52 | 5A | 62 | 6A | 72 | 7A |
| disp8[EBX]; | | 011 | 43 | 4B | 53 | 5B | 63 | 6B | 73 | 7B |
| disp8[--] [--] | 01 | 100 | 44 | 4C | 54 | 5C | 64 | 6C | 74 | 7C |
| disp8[EBP] | | 101 | 45 | 4D | 55 | 5D | 65 | 6D | 75 | 7D |
| disp8[ESI] | | 110 | 46 | 4E | 56 | 5E | 66 | 6E | 76 | 7E |
| disp8[EDI] | | 111 | 47 | 4F | 57 | 5F | 67 | 6F | 77 | 7F |

# ModR/M Table (2)

| r32(/r) | | | EAX | ECX | EDX | EBX | ESP | EBP | ESI | EDI |
|---|---|---|---|---|---|---|---|---|---|---|
| /digit (Opcode) | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| REG = | | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +---Address--+ | +Mod | R/M-+ | +---------ModR/M Values in Hexadecimal-----------+ | | | | | | | |
| disp32[EAX] | | 000 | 80 | 88 | 90 | 98 | A0 | A8 | B0 | B8 |
| disp32[ECX] | | 001 | 81 | 89 | 91 | 99 | A1 | A9 | B1 | B9 |
| disp32[EDX] | | 010 | 82 | 8A | 92 | 9A | A2 | AA | B2 | BA |
| disp32[EBX] | | 011 | 83 | 8B | 93 | 9B | A3 | AB | B3 | BB |
| disp32[--] [--] | 10 | 100 | 84 | 8C | 94 | 9C | A4 | AC | B4 | BC |
| disp32[EBP] | | 101 | 85 | 8D | 95 | 9D | A5 | AD | B5 | BD |
| disp32[ESI] | | 110 | 86 | 8E | 96 | 9E | A6 | AE | B6 | BE |
| disp32[EDI] | | 111 | 87 | 8F | 97 | 9F | A7 | AF | B7 | BF |
| EAX/AX/AL | | 000 | C0 | C8 | D0 | D8 | E0 | E8 | F0 | F8 |
| ECX/CX/CL | | 001 | C1 | C9 | D1 | D9 | E1 | E9 | F1 | F9 |
| EDX/DX/DL | | 010 | C2 | CA | D2 | DA | E2 | EA | F2 | FA |
| EBX/BX/BL | | 011 | C3 | CB | D3 | DB | E3 | EB | F3 | FB |
| ESP/SP/AH | 11 | 100 | C4 | CC | D4 | DC | E4 | EC | F4 | FC |
| EBP/BP/CH | | 101 | C5 | CD | D5 | DD | E5 | ED | F5 | FD |
| ESI/SI/DH | | 110 | C6 | CE | D6 | DE | E6 | EE | F6 | FE |
| EDI/DI/BH | | 111 | C7 | CF | D7 | DF | E7 | EF | F7 | FF |

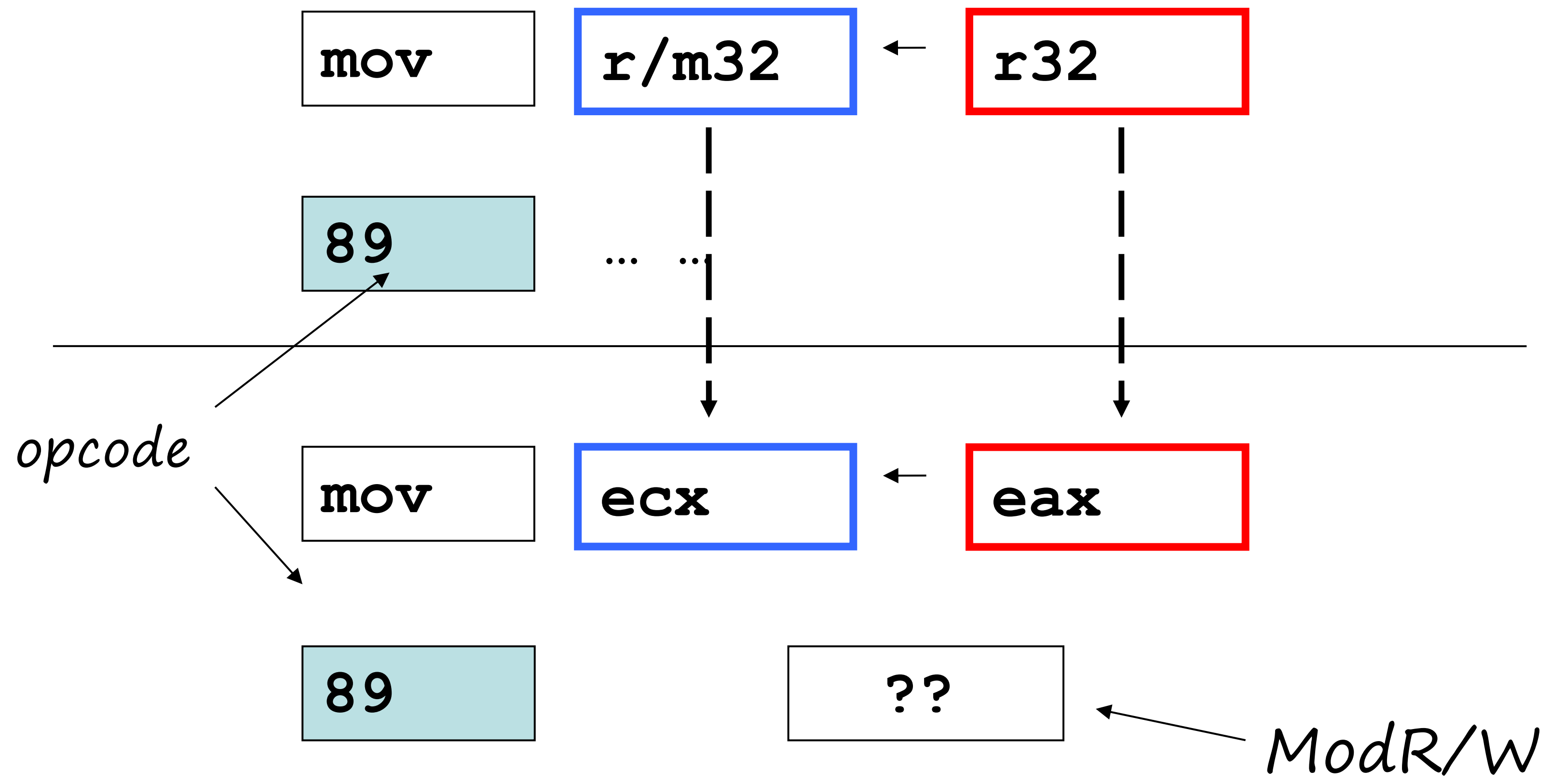
# SIB (1)

| r32 | | | EAX | ECX | EDX | EBX | ESP | [*] | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Base = | | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +Scaled Index+ | +SS | Index+ | +--------ModR/M Values in Hexadecimal--------+ | | | | | | | |
| [EAX] | | 000 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| [ECX] | | 001 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| [EDX] | | 010 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| [EBX] | | 011 | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |
| none | 00 | 100 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| [EBP] | | 101 | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |
| [ESI] | | 110 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| [EDI] | | 111 | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |
| | | | | | | | | | | |
| [EAX*2] | | 000 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| [ECX*2] | | 001 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
| [ECX*2] | | 010 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 |
| [EBX*2] | | 011 | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
| none | 01 | 100 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
| [EBP*2] | | 101 | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
| [ESI*2] | | 110 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| [EDI*2] | | 111 | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F |

# SIB (2)

| r32 | | | EAX | ECX | EDX | EBX | ESP | [*] | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Base = | | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +Scaled Index+ | +SS | Index+ | +--------ModR/M Values in Hexadecimal--------+ | | | | | | | |
| [EAX*4] | | 000 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 |
| [ECX*4] | | 001 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F |
| [EDX*4] | | 010 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 |
| [EBX*4] | | 011 | 98 | 89 | 9A | 9B | 9C | 9D | 9E | 9F |
| none | 10 | 100 | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 |
| [EBP*4] | | 101 | A8 | A9 | AA | AB | AC | AD | AE | AF |
| [ESI*4] | | 110 | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 |
| [EDI*4] | | 111 | B8 | B9 | BA | BB | BC | BD | BE | BF |
| | | | | | | | | | | |
| [EAX*8] | | 000 | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
| [ECX*8] | | 001 | C8 | C9 | CA | CB | CC | CD | CE | CF |
| [EDX*8] | | 010 | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 |
| [EBX*8] | | 011 | D8 | D9 | DA | DB | DC | DD | DE | DF |
| none | 11 | 100 | E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 |
| [EBP*8] | | 101 | E8 | E9 | EA | EB | EC | ED | EE | EF |
| [ESI*8] | | 110 | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 |
| [EDI*8] | | 111 | F8 | F9 | FA | FB | FC | FD | FE | FF |

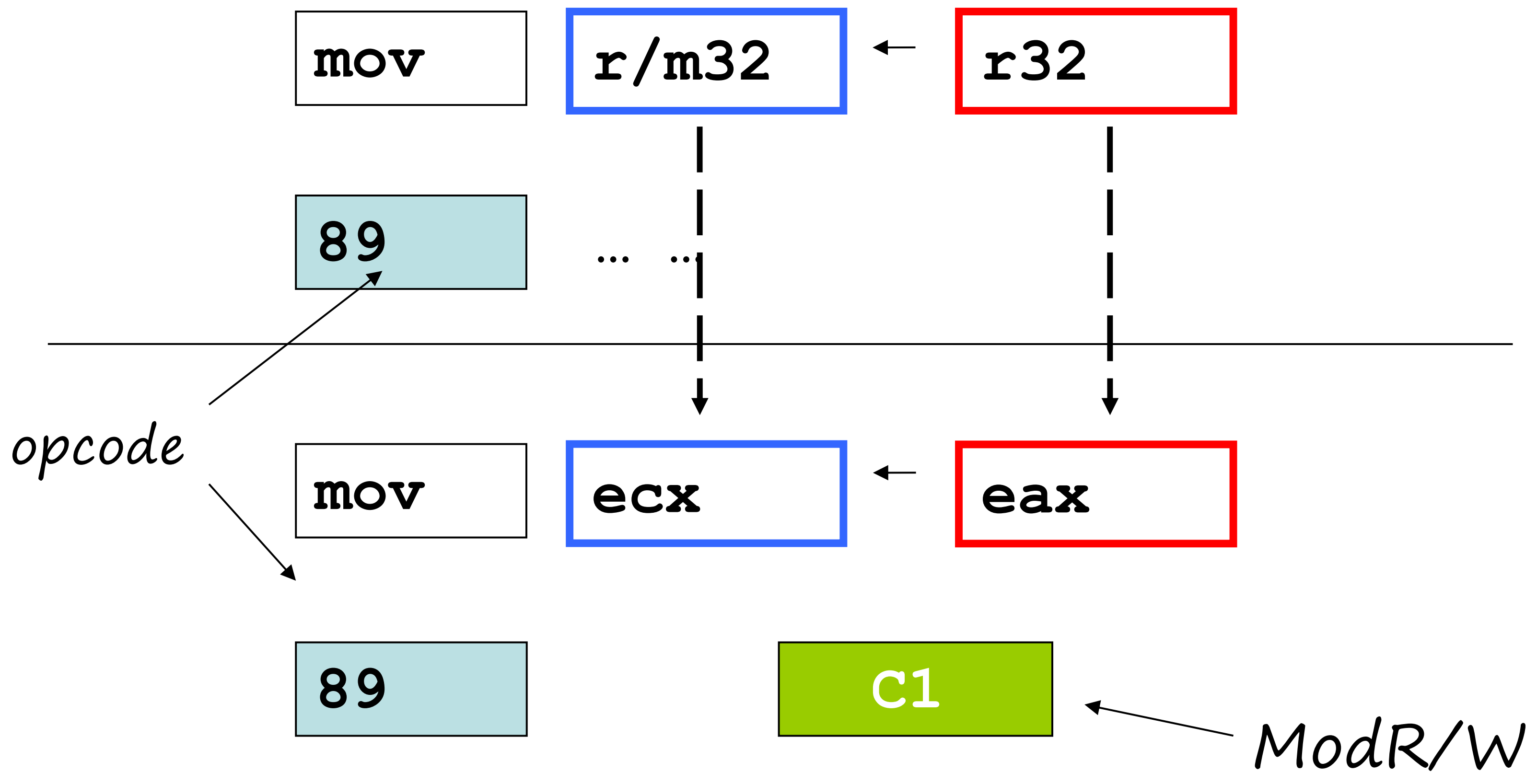
# Move Data



# ModR/M Table (2)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| r32(/r) | | | EAX | ECX | EDX | EBX | ESP | EBP | ESI | EDI |
| /digit (Opcode) | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| REG = | | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +---Address--+ | +Mod | R/M-+ | ----------ModR/M Values in Hexadecimal-----------+ | | | | | | | |
| disp32[EAX] | | 000 | 80 | 88 | 90 | 98 | A0 | A8 | B0 | B8 |
| disp32[ECX] | | 001 | 81 | 89 | 91 | 99 | A1 | A9 | B1 | B9 |
| disp32[EDX] | | 010 | 82 | 8A | 92 | 9A | A2 | AA | B2 | BA |
| disp32[EBX] | | 011 | 83 | 8B | 93 | 9B | A3 | AB | B3 | BB |
| disp32[--] [--] | 10 | 100 | 84 | 8C | 94 | 9C | A4 | AC | B4 | BC |
| disp32[EBP] | | 101 | 85 | 8D | 95 | 9D | A5 | AD | B5 | BD |
| disp32[ESI] | | 110 | 86 | 8E | 96 | 9E | A6 | AE | B6 | BE |
| disp32[EDI] | | 111 | 87 | 8F | 97 | 9F | A7 | AF | B7 | BF |
| | | | | | | | | | | |
| EAX/AX/AL | | 000 | C0 | C8 | D0 | D8 | E0 | E8 | F0 | F8 |
| ECX/CX/CL | | 001 | C1 | C9 | D1 | D9 | E1 | E9 | F1 | F9 |
| EDX/DX/DL | | 010 | C2 | CA | D2 | DA | E2 | EA | F2 | FA |
| EBX/BX/BL | | 011 | C3 | CB | D3 | DB | E3 | EB | F3 | FB |
| ESP/SP/AH | 11 | 100 | C4 | CC | D4 | DC | E4 | EC | F4 | FC |
| EBP/BP/CH | | 101 | C5 | CD | D5 | DD | E5 | ED | F5 | FD |
| ESI/SI/DH | | 110 | C6 | CE | D6 | DE | E6 | EE | F6 | FE |
| EDI/DI/BH | | 111 | C7 | CF | D7 | DF | E7 | EF | F7 | FF |

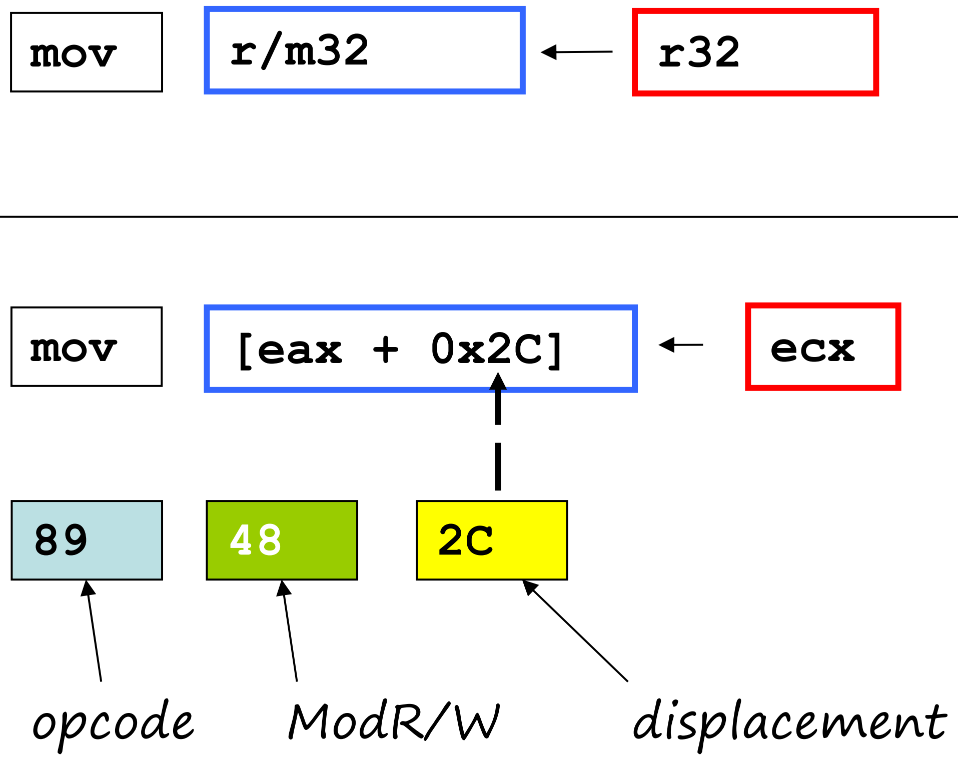
# Move Data



## Move Data

```
#include <stdio.h>
int main()
{
  int x=9, y=0;
  __asm
  {
    mov      eax, dword ptr [x]
    _emit    0x89
    _emit    0xC1
    mov      dword ptr [y], ecx
  }
  return 0;
}
```

# Move Data



mov
r/m32
r32
mov
[eax + 0x2C]
ecx
89
48
2C
opcode
ModR/W
displacement

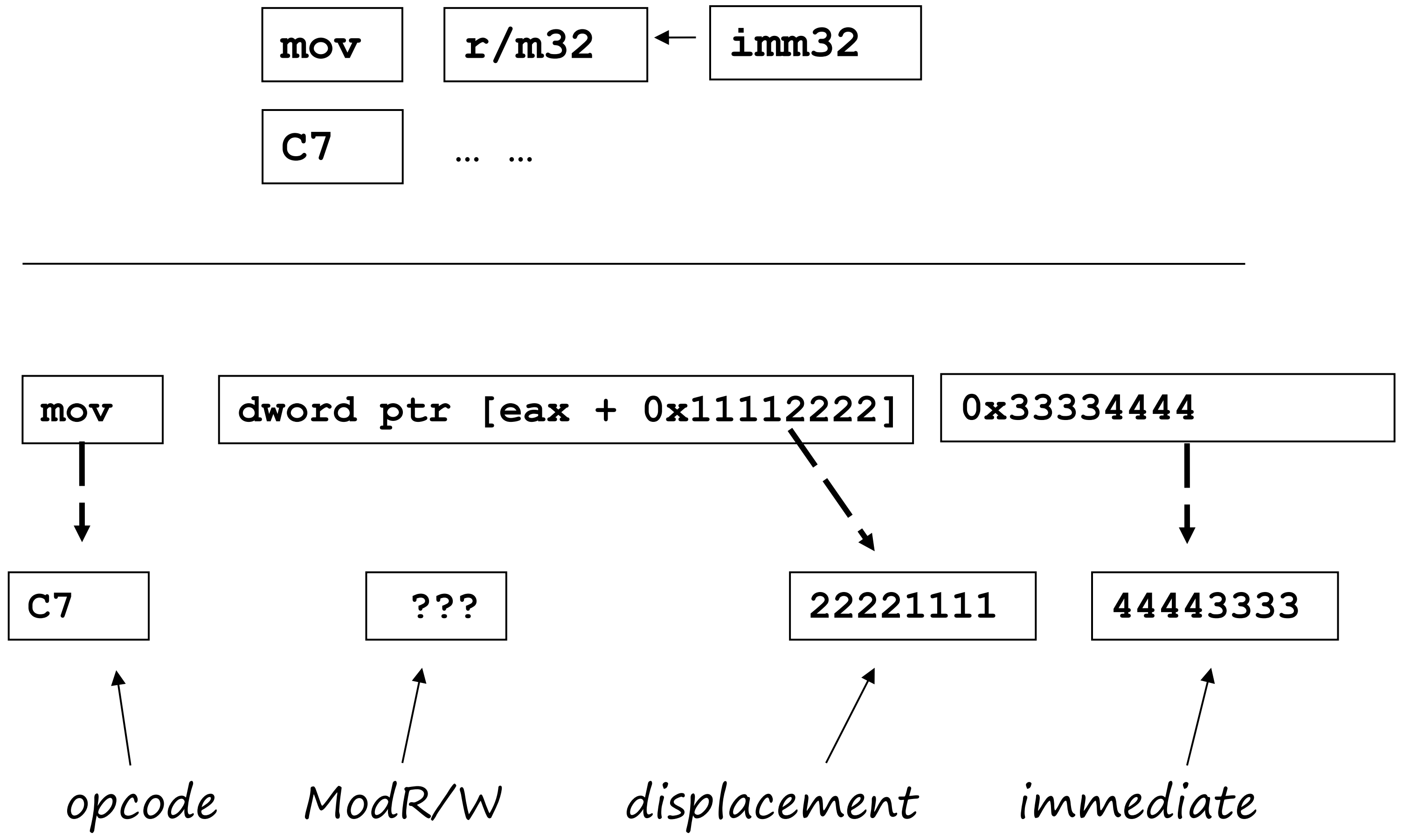
# ModR/M Table (1)

| r32(/r) | | | EAX | ECX | EDX | EBX | ESP | EBP | ESI | EDI |
|---|---|---|---|---|---|---|---|---|---|---|
| /digit (Opcode) | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| REG = | | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +---Address--+ | +Mod | R/M-+ | +---------ModR/M Values in Hexadecimal----------+ | | | | | | | |
| [EAX] | | 000 | 00 | 08 | 10 | 18 | 20 | 28 | 30 | 38 |
| [ECX] | | 001 | 01 | 09 | 11 | 19 | 21 | 29 | 31 | 39 |
| [EDX] | | 010 | 02 | 0A | 12 | 1A | 22 | 2A | 32 | 3A |
| [EBX] | | 011 | 03 | 0B | 13 | 1B | 23 | 2B | 33 | 3B |
| [--] [--] | 00 | 100 | 04 | 0C | 14 | 1C | 24 | 2C | 34 | 3C |
| disp32 | | 101 | 05 | 0D | 15 | 1D | 25 | 2D | 35 | 3D |
| [ESI] | | 110 | 06 | 0E | 16 | 1E | 26 | 2E | 36 | 3E |
| [EDI] | | 111 | 07 | 0F | 17 | 1F | 27 | 2F | 37 | 3F |
| disp8[EAX] | | 000 | 40 | 48 | 50 | 58 | 60 | 68 | 70 | 78 |
| disp8[ECX] | | 001 | 41 | 49 | 51 | 59 | 61 | 69 | 71 | 79 |
| disp8[EDX] | | 010 | 42 | 4A | 52 | 5A | 62 | 6A | 72 | 7A |
| disp8[EBX]; | | 011 | 43 | 4B | 53 | 5B | 63 | 6B | 73 | 7B |
| disp8[--] [--] | 01 | 100 | 44 | 4C | 54 | 5C | 64 | 6C | 74 | 7C |
| disp8[EBP] | | 101 | 45 | 4D | 55 | 5D | 65 | 6D | 75 | 7D |
| disp8[ESI] | | 110 | 46 | 4E | 56 | 5E | 66 | 6E | 76 | 7E |
| disp8[EDI] | | 111 | 47 | 4F | 57 | 5F | 67 | 6F | 77 | 7F |

# Move Data – more opcodes

|  |  |
|---|---|
| mov r32 ← r/m32<br>8B … … |  |
| mov r32 ← imm32<br>B8 + *rd* … … |  |
| mov r/m32 ← imm32<br>C7 … … |  |
| mov eax ← moffs32<br>A1 … … | mov eax ← [0x11223344]<br>A1 44332211 |

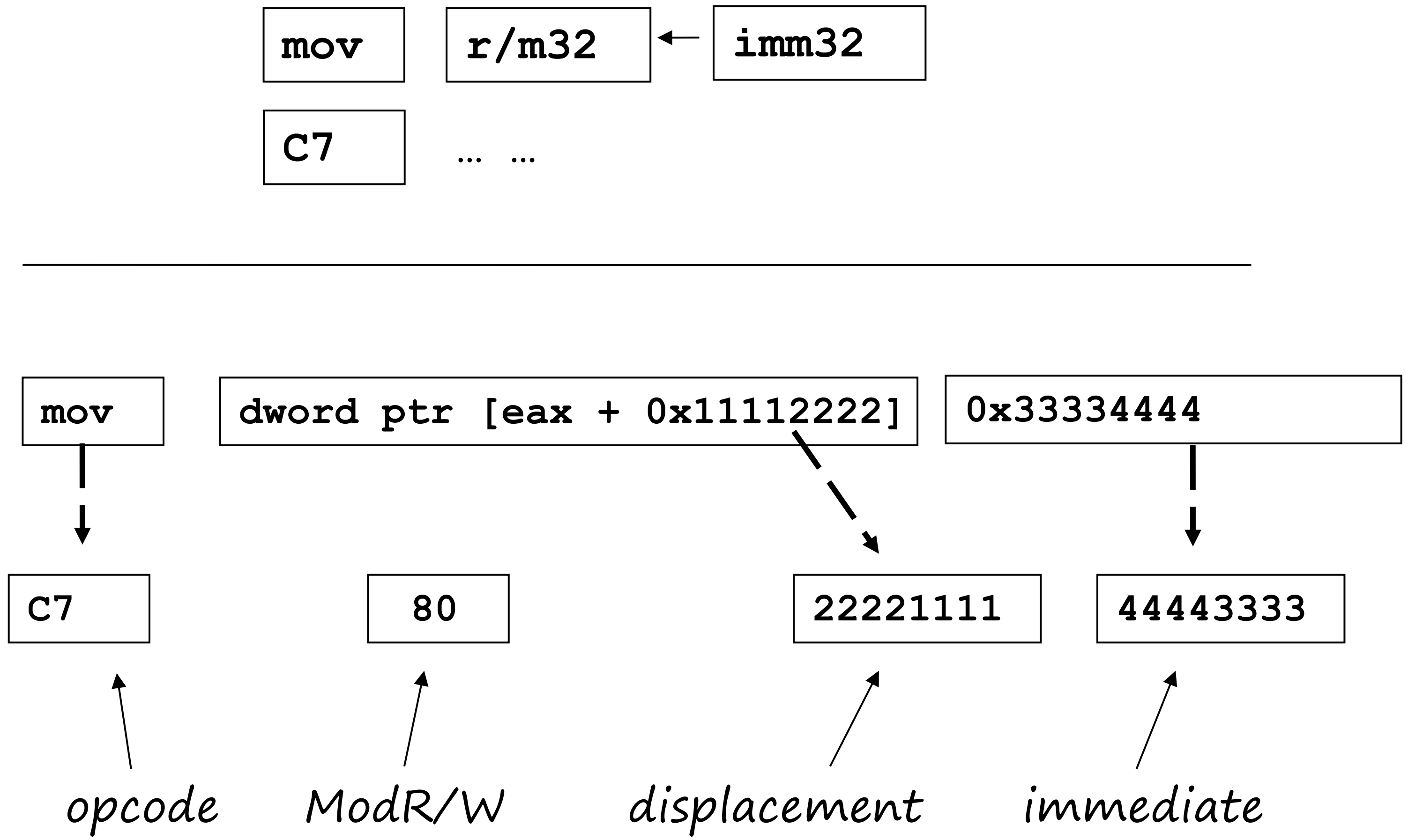
# Move Data


mov
r/m32
imm32
C7
… …
mov
dword ptr [eax + 0x11112222]
0x33334444
C7
???
22221111
44443333
opcode
ModR/W
displacement
immediate

# ModR/M Table (2)

| r32(/r) |  |  | EAX | ECX | EDX | EBX | ESP | EBP | ESI | EDI |
|---|---|---|---|---|---|---|---|---|---|---|
| /digit (Opcode) |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| REG = |  |  | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +---Address--+ | +Mod | R/M-+ | +----------ModR/M Values in Hexadecimal-----------+ |  |  |  |  |  |  |  |
| disp32[EAX] |  | 000 | 80 | 88 | 90 | 98 | A0 | A8 | B0 | B8 |
| disp32[ECX] |  | 001 | 81 | 89 | 91 | 99 | A1 | A9 | B1 | B9 |
| disp32[EDX] |  | 010 | 82 | 8A | 92 | 9A | A2 | AA | B2 | BA |
| disp32[EBX] |  | 011 | 83 | 8B | 93 | 9B | A3 | AB | B3 | BB |
| disp32[--] [--] | 10 | 100 | 84 | 8C | 94 | 9C | A4 | AC | B4 | BC |
| disp32[EBP] |  | 101 | 85 | 8D | 95 | 9D | A5 | AD | B5 | BD |
| disp32[ESI] |  | 110 | 86 | 8E | 96 | 9E | A6 | AE | B6 | BE |
| disp32[EDI] |  | 111 | 87 | 8F | 97 | 9F | A7 | AF | B7 | BF |
|  |  |  |  |  |  |  |  |  |  |  |
| EAX/AX/AL |  | 000 | C0 | C8 | D0 | D8 | E0 | E8 | F0 | F8 |
| ECX/CX/CL |  | 001 | C1 | C9 | D1 | D9 | E1 | E9 | F1 | F9 |
| EDX/DX/DL |  | 010 | C2 | CA | D2 | DA | E2 | EA | F2 | FA |
| EBX/BX/BL |  | 011 | C3 | CB | D3 | DB | E3 | EB | F3 | FB |
| ESP/SP/AH | 11 | 100 | C4 | CC | D4 | DC | E4 | EC | F4 | FC |
| EBP/BP/CH |  | 101 | C5 | CD | D5 | DD | E5 | ED | F5 | FD |
| ESI/SI/DH |  | 110 | C6 | CE | D6 | DE | E6 | EE | F6 | FE |
| EDI/DI/BH |  | 111 | C7 | CF | D7 | DF | E7 | EF | F7 | FF |

# Move Data



mov
r/m32
imm32
C7
… …
mov
dword ptr [eax + 0x11112222]
0x33334444
C7
80
22221111
44443333
opcode
ModR/W
displacement
immediate

# Push & Pop

```
#include <stdio.h>
int main()
{
  int x;
  __asm
  {
    mov eax, 9
    mov dword ptr [x], eax
    push eax
  }
  printf("%d\n",x);
  __asm
  {
    pop eax
    mov dword ptr [x], eax
  }
  printf("%d",x);
  return 0;
}
```

Save eax

Restore eax

# Push & Pop

|  |  |  |  |
|---|---|---|---|
| `push` | `r32` | `pop` | `r32` |
| `50+r` |  | `58+r` | … … |
| `push` | `eax` | `pop` | `eax` |
| `50` |  | `58` |  |

## Push & Pop

```
#include <stdio.h>
int main()
{
  int x;
  __asm
  {
    mov eax, 9
    mov dword ptr [x], eax
    _emit 0x50
  }
  printf("%d\n",x);
  __asm
  {
    _emit 0x58
    mov dword ptr [x], eax
  }
  printf("%d",x);
  return 0;
}
```

push eax

pop eax

# Push & Pop

| | |
|---|---|
| push | imm32 |
| 68 | |

| | |
|---|---|
| pop | r32 |
| 58+r | … … |

| | |
|---|---|
| push | 0x11223344 |
| 68 | 44332211 |

| | |
|---|---|
| pop | eax |
| 58 | |

# Push & Pop

```
#include <stdio.h>
int main()
{
  int x=0;
  __asm
  {
    _emit 0x68
    _emit 0x44
    _emit 0x33
    _emit 0x22
    _emit 0x11
  }
  printf("0x%08x\n",x);
  __asm
  {
    _emit 0x58
    mov dword ptr [x], eax
  }
  printf("0x%08x",x);
  return 0;
}
```

push

imm32

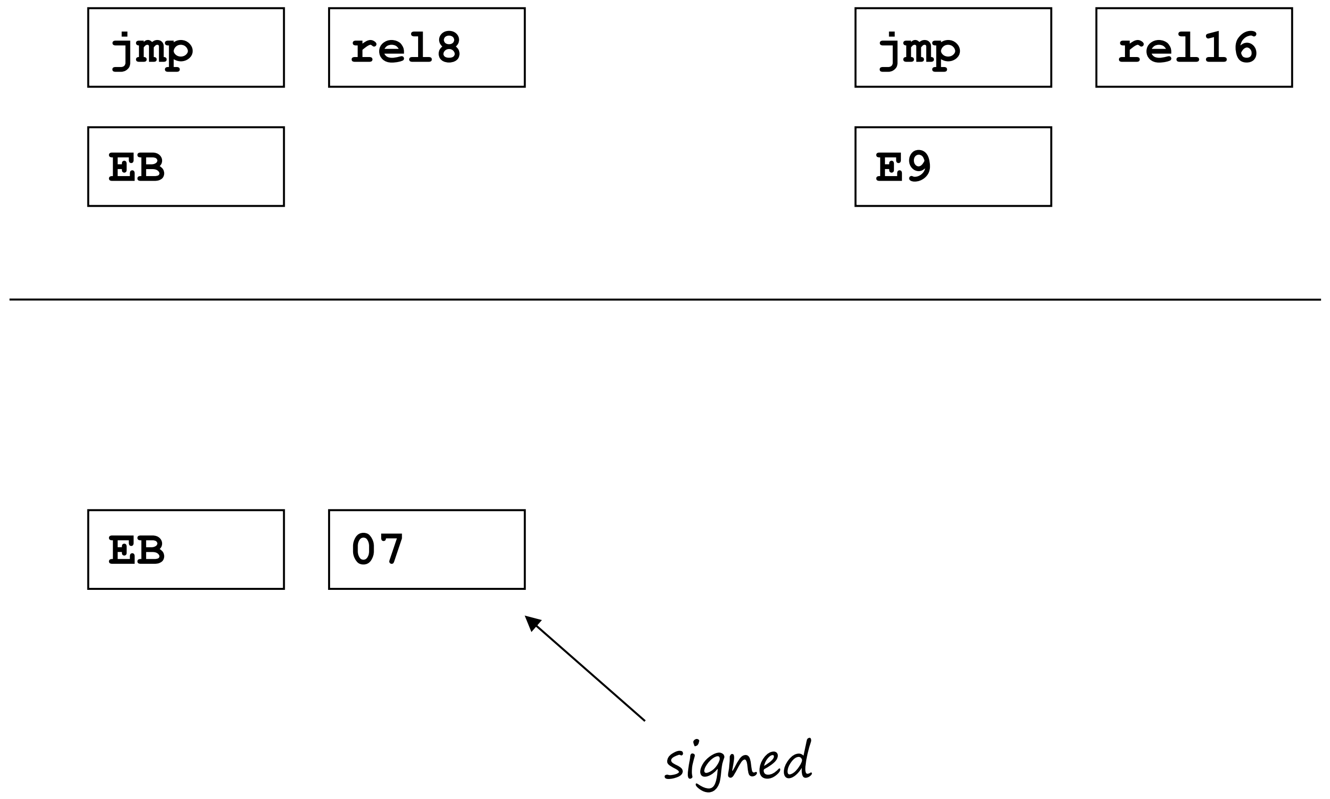
pop eax

## Direct Jump

```
#include <stdio.h>
int main()
{
  int x = 9;
  __asm
  {
    jmp label
    mov dword ptr [x], 6
label:
    nop
  }
  printf("%d",x);
  return 0;
}
```

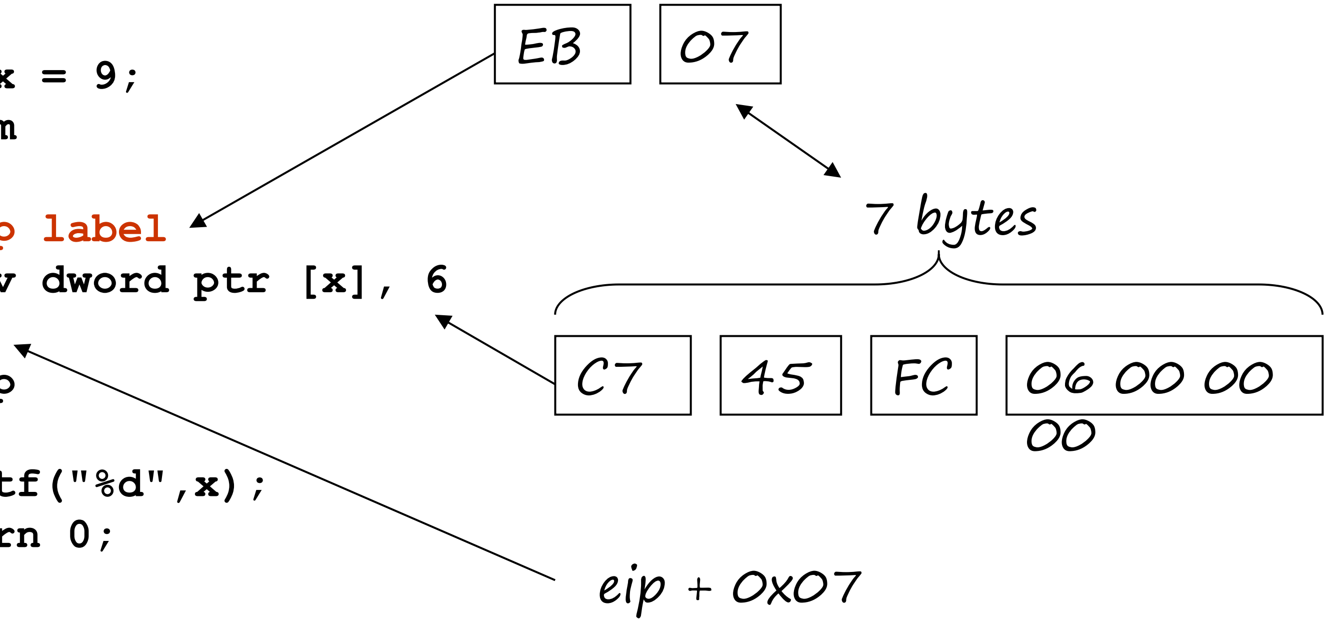
$\boxed{C7}$ $\boxed{45}$ $\boxed{FC}$ $\boxed{06\ 00\ 00\ 00}$

# Direct Jump



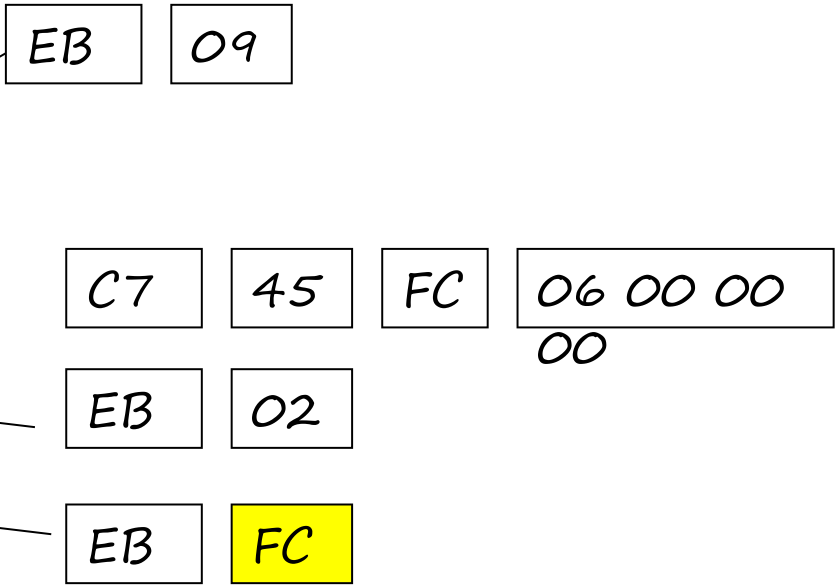
# Direct Jump

```
#include <stdio.h>
int main()
{
  int x = 9;
  __asm
  {
    jmp label
    mov dword ptr [x], 6
label:
    nop
  }
  printf("%d",x);
  return 0;
}
```



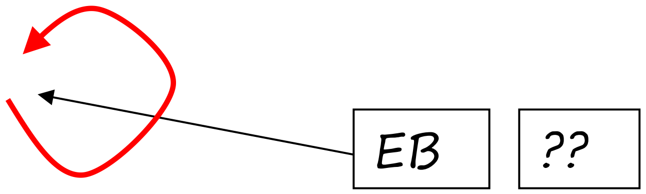
# Direct Jump

```
#include <stdio.h>
int main()
{
  int x = 9;
  __asm
  {
    jmp lab2
    mov dword ptr [x], 6
lab1:
    jmp lab3
lab2:
    jmp lab1
lab3:
    nop
  }
  printf("%d",x);
  return 0;
}
```

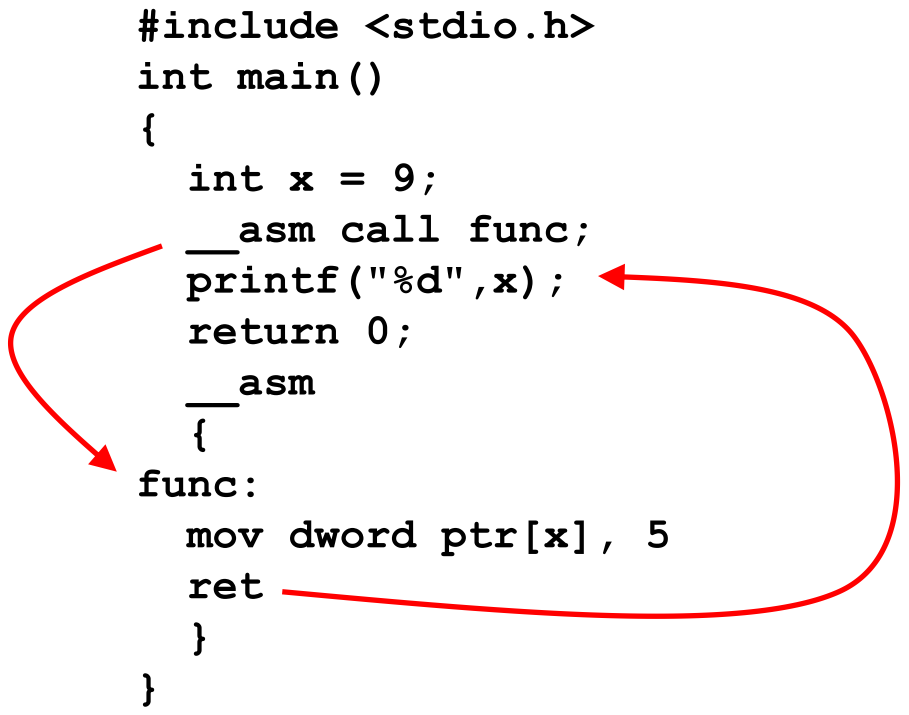


## Quiz

```
#include <stdio.h>
int main()
{
__asm
  {
lab1:
    jmp lab1
}
  return 0;
}
```



# Direct Call & Ret



```
#include <stdio.h>
int main()
{
  int x = 9;
  __asm call func;
  printf("%d",x);
  return 0;
  __asm
  {
func:
  mov dword ptr[x], 5
  ret
  }
}
```

# Direct Call & Ret

| | | |
|---|---|---|
| `call` | `rel32` | `ret` |
| `E8` | … … | `C3` |

# Direct Call & Ret

```
_main:
  00000000: 55                 push      ebp
  00000001: 8B EC              mov       ebp,esp
  00000003: 51                 push      ecx
  00000004: C7 45 FC 09 00 00  mov       dword ptr [ebp-4],9
            00
  0000000B: E8 15 00 00 00     call      00000025
  00000010: 8B 45 FC           mov       eax,dword ptr [ebp-4]
  00000013: 50                 push      eax
  00000014: 68 00 00 00 00     push      offset $SG2470
  00000019: E8 00 00 00 00     call      _printf
  0000001E: 83 C4 08           add       esp,8
  00000021: 33 C0              xor       eax,eax
  00000023: EB 08              jmp       0000002D
  00000025: C7 45 FC 05 00 00  mov       dword ptr [ebp-4],5
            00
  0000002C: C3                 ret
  0000002D: 8B E5              mov       esp,ebp
  0000002F: 5D                 pop       ebp
  00000030: C3                 ret
```

# Indirect Call

```
#include <stdio.h>
void func()
{
  x=5;
}

int main()
{
  int x = 9;
  __asm{
    mov eax, func
    call eax
  }
  printf("%d",x);
  return 0;
}
```

??

# Indirect Call & Jump

| | | |
|---|---|---|
| `call` | `r/m32` | |
| `FF` | `/2` | … … |

---

| | | |
|---|---|---|
| `jmp` | `r/m32` | |
| `FF` | `/4` | … … |

# ModR/M Table (1)

| r32(/r) | | | EAX | ECX | EDX | EBX | ESP | EBP | ESI | EDI |
|---|---|---|---|---|---|---|---|---|---|---|
| /digit (Opcode) | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| REG = | | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +---Address--+ | +Mod | R/M-+ | +---------ModR/M Values in Hexadecimal----------+ | | | | | | | |
| [EAX] | | 000 | 00 | 08 | 10 | 18 | 20 | 28 | 30 | 38 |
| [ECX] | | 001 | 01 | 09 | 11 | 19 | 21 | 29 | 31 | 39 |
| [EDX] | | 010 | 02 | 0A | 12 | 1A | 22 | 2A | 32 | 3A |
| [EBX] | | 011 | 03 | 0B | 13 | 1B | 23 | 2B | 33 | 3B |
| [--] [--] | 00 | 100 | 04 | 0C | 14 | 1C | 24 | 2C | 34 | 3C |
| disp32 | | 101 | 05 | 0D | 15 | 1D | 25 | 2D | 35 | 3D |
| [ESI] | | 110 | 06 | 0E | 16 | 1E | 26 | 2E | 36 | 3E |
| [EDI] | | 111 | 07 | 0F | 17 | 1F | 27 | 2F | 37 | 3F |
| disp8[EAX] | | 000 | 40 | 48 | 50 | 58 | 60 | 68 | 70 | 78 |
| disp8[ECX] | | 001 | 41 | 49 | 51 | 59 | 61 | 69 | 71 | 79 |
| disp8[EDX] | | 010 | 42 | 4A | 52 | 5A | 62 | 6A | 72 | 7A |
| disp8[EBX]; | | 011 | 43 | 4B | 53 | 5B | 63 | 6B | 73 | 7B |
| disp8[--] [--] | 01 | 100 | 44 | 4C | 54 | 5C | 64 | 6C | 74 | 7C |
| disp8[EBP] | | 101 | 45 | 4D | 55 | 5D | 65 | 6D | 75 | 7D |
| disp8[ESI] | | 110 | 46 | 4E | 56 | 5E | 66 | 6E | 76 | 7E |
| disp8[EDI] | | 111 | 47 | 4F | 57 | 5F | 67 | 6F | 77 | 7F |

# ModR/M Table (2)

| r32(/r) | | | EAX | ECX | EDX | EBX | ESP | EBP | ESI | EDI |
|---|---|---|---|---|---|---|---|---|---|---|
| /digit (Opcode) | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| REG = | | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| +---Address--+ | +Mod | R/M-+ | +---------ModR/M Values in Hexadecimal-----------+ | | | | | | | |
| disp32[EAX] | | 000 | 80 | 88 | 90 | 98 | A0 | A8 | B0 | B8 |
| disp32[ECX] | | 001 | 81 | 89 | 91 | 99 | A1 | A9 | B1 | B9 |
| disp32[EDX] | | 010 | 82 | 8A | 92 | 9A | A2 | AA | B2 | BA |
| disp32[EBX] | | 011 | 83 | 8B | 93 | 9B | A3 | AB | B3 | BB |
| disp32[--] [--] | 10 | 100 | 84 | 8C | 94 | 9C | A4 | AC | B4 | BC |
| disp32[EBP] | | 101 | 85 | 8D | 95 | 9D | A5 | AD | B5 | BD |
| disp32[ESI] | | 110 | 86 | 8E | 96 | 9E | A6 | AE | B6 | BE |
| disp32[EDI] | | 111 | 87 | 8F | 97 | 9F | A7 | AF | B7 | BF |
| EAX/AX/AL | | 000 | C0 | C8 | D0 | D8 | E0 | E8 | F0 | F8 |
| ECX/CX/CL | | 001 | C1 | C9 | D1 | D9 | E1 | E9 | F1 | F9 |
| EDX/DX/DL | | 010 | C2 | CA | D2 | DA | E2 | EA | F2 | FA |
| EBX/BX/BL | | 011 | C3 | CB | D3 | DB | E3 | EB | F3 | FB |
| ESP/SP/AH | 11 | 100 | C4 | CC | D4 | DC | E4 | EC | F4 | FC |
| EBP/BP/CH | | 101 | C5 | CD | D5 | DD | E5 | ED | F5 | FD |
| ESI/SI/DH | | 110 | C6 | CE | D6 | DE | E6 | EE | F6 | FE |
| EDI/DI/BH | | 111 | C7 | CF | D7 | DF | E7 | EF | F7 | FF |

## Indirect Call

```
#include <stdio.h>
int i = 0;
void func()
{
  x=5;
}

int main()
{
__asm{
    mov eax, func
    _emit 0xFF
    _emit 0xD0
  }
  printf("%d",x);
  return 0;
}
```

## Outline

Intel x86 platform

Data representation

Instruction set

Instruction Encoding

Assembler & Disassembler

## Assembler

Assembly Code ==> Binary Code

- translating assembly instruction mnemonics into opcodes,
- resolving symbolic names for memory locations

# Assembly Syntax

```
mov eax, 1
mov ebx, 0ffh
mov ebx, eax
```

Intel

```
movl $1,%eax
movl $0xff,%ebx
movl %eax, %ebx
```

AT&T

## Assembly Syntax

```
mov eax, dword ptr [ebx+3]
add eax, dword ptr [ebx+ecx*2h]
lea eax, dword ptr [ebx+ecx]
```

Intel

---

```
movl 3(%ebx),%eax
addl (%ebx,%ecx,0x2),%eax
leal (%ebx,%ecx),%eax
```

AT&T

## List of Assemblers

- GAS
- NASM
- TASM
- MASM
- Yasm
- HLA

## Disassembler

Binary Code ==> Assembly Code

## List of Disassembler

- dumpbin
- gdb
- IDA
- ILDASM
- Ollydbg

Any Question ?