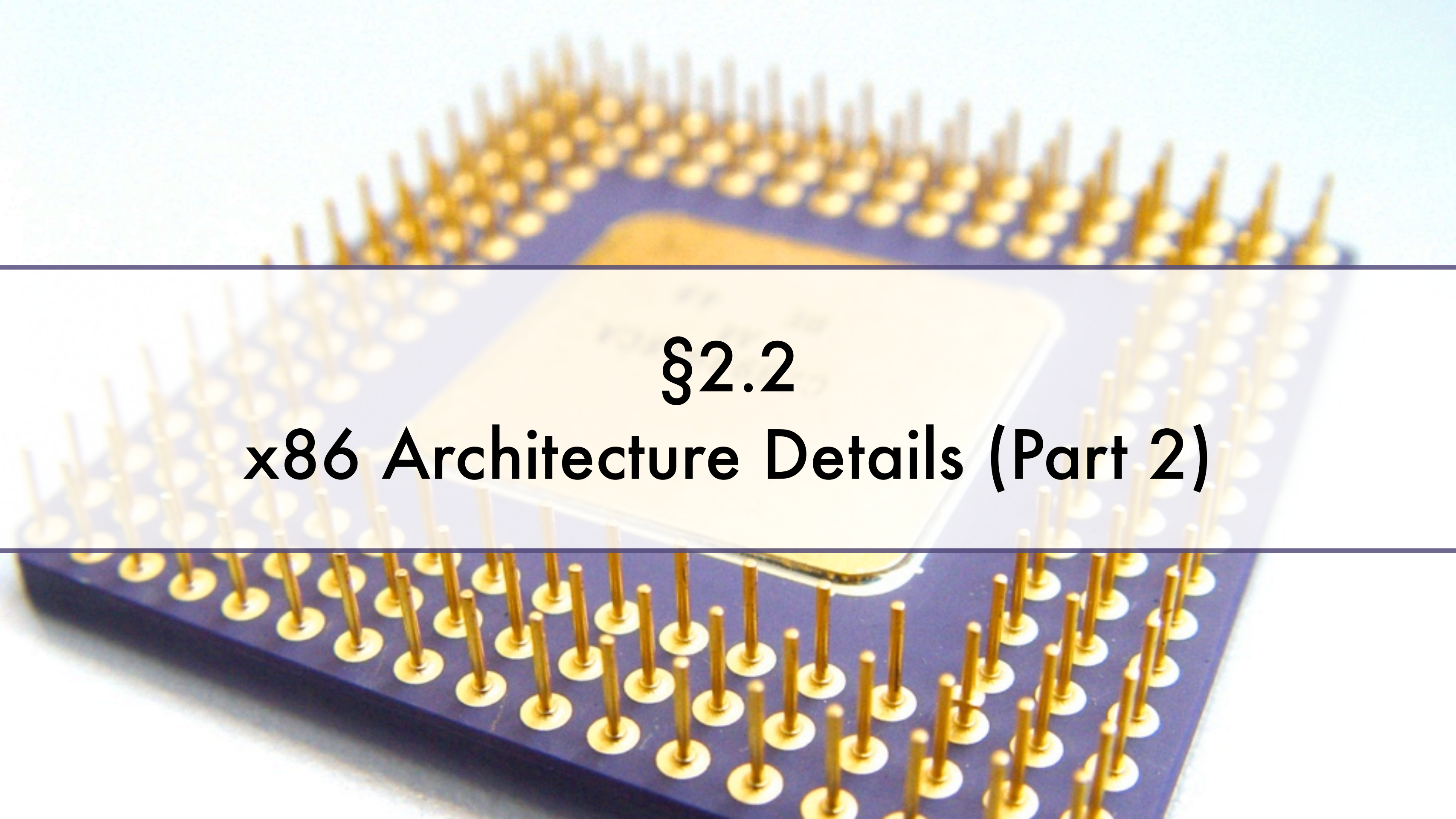




§2.2 x86 Architecture Details (Part 2)

—

§3.1 Basics of Assembly Language



§2.2

x86 Architecture Details (Part 2)

x86 Registers (Review)



32-bit General-Purpose Registers

EAX
EBX
ECX
EDX

EBP
ESP
ESI
EDI

EFLAGS
EIP

16-bit Segment Registers

CS	ES
SS	FS
DS	GS

x86 Registers (Review)



- ▶ **32-bit general-purpose registers:**

- ▶ EAX (Extended) Accumulator

- ▶ EBX Base

- ▶ ECX Count

- ▶ EDX Data

- ▶ ESI Source Index

- ▶ EDI Destination Index

- ▶ ESP Stack Pointer

- ▶ EBP Base Pointer

Do not use ESP or EBP for arithmetic/data transfer; they have special uses (Chapter 8)

Names are based on historical or special uses.

x86 Registers (Review)



- ▶ General-purpose registers (except EBP, ESP) can be used more or less arbitrarily, but some instructions use them for special purposes, e.g.,
 - ▶ The **mul** (multiplication) instruction requires one operand to be in EAX (Chapter 7)
 - ▶ EBP is used to access function parameters and local variables in procedures (Chapter 8)
 - ▶ ESI and EDI are used by high-speed memory transfer instructions, e.g., **movsb** (Chapter 9)

x86 Registers



- ▶ **16-bit segment registers:**
 - ▶ CS Code Segment
 - ▶ SS Stack Segment
 - ▶ DS Data Segment
 - ▶ ES Extra Segment
 - ▶ FS F Segment (Extra Segment)
 - ▶ GS G Segment (Extra Segment)
- ▶ Used when accessing memory
- ▶ Important in real-mode programming and when writing an OS; we won't need them in this course

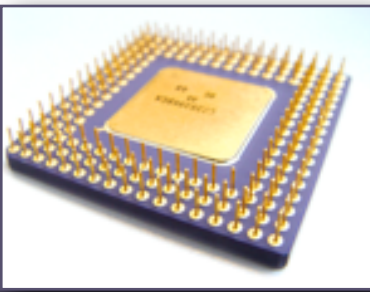
x86 Registers



▶ **EFLAGS – Extended Flags**

- ▶ Each bit has a different purpose
- ▶ Some bits are *control flags*
 - ▶ Setting/clearing these bits changes the CPU's operation
 - ▶ E.g., enter protected mode
 - ▶ E.g., break after each instruction (for debugging)
- ▶ Other bits are *status flags*
 - ▶ E.g., Carry flag: is set to 1 when unsigned arithmetic operation produces a result too large to fit in 32 bits
 - ▶ E.g., Zero flag: becomes set to 1 when an arithmetic or logical operation results in a 0 value

x86 Registers

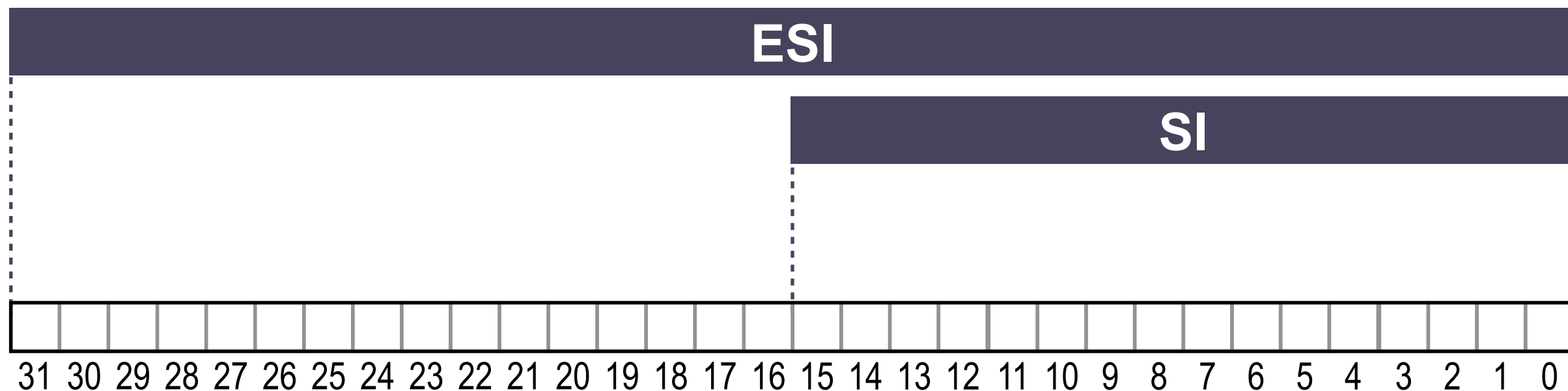


- ▶ **EIP – Extended Instruction Pointer**
 - ▶ Contains the memory address of the next instruction to be executed
 - ▶ Recall that the instruction pointer is incremented as part of the fetch-decode-execute cycle
 - ▶ Other instructions (e.g., **jmp**) change the instruction pointer

Accessing Parts of Registers



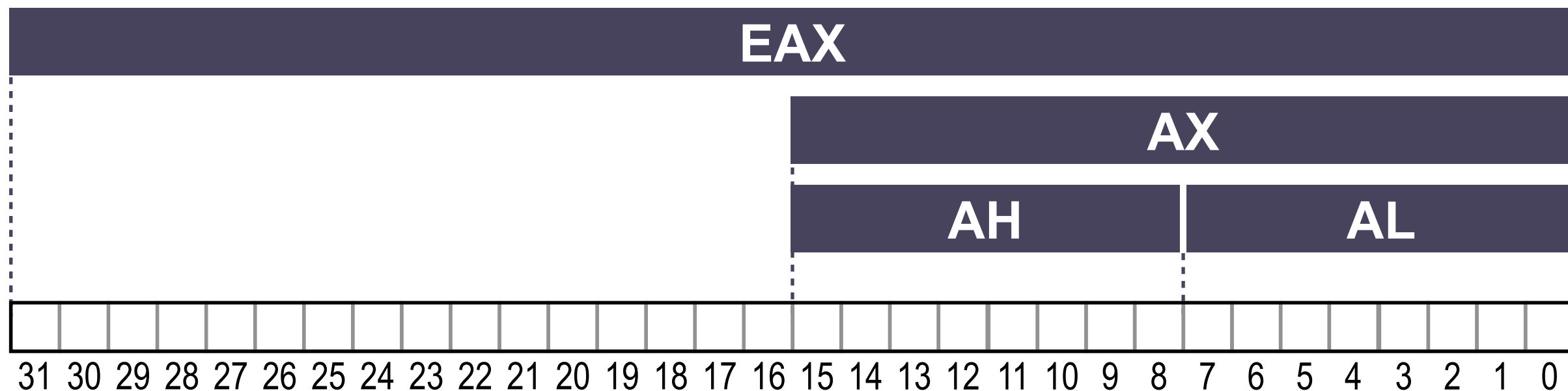
- ▶ Parts of some registers can be accessed by other names
 - ▶ ESI: Low 16 bits are called SI
 - ▶ EDI: Low 16 bits are called DI
 - ▶ EBP: Low 16 bits are called BP
 - ▶ ESP: Low 16 bits are called SP



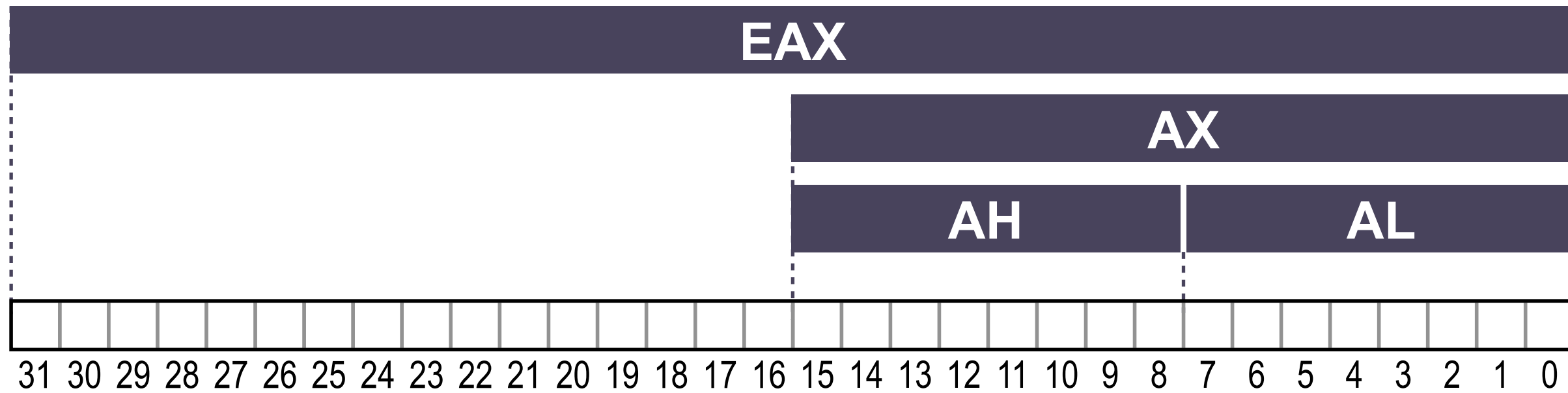
Accessing Parts of Registers



- ▶ Parts of some registers can be accessed by other names
 - ▶ EAX: bits 0–15 are AX, bits 8–15 are AH, bits 0–7 are AL
 - ▶ EBX, BX, BH, BL
 - ▶ ECX, CX, CH, CL
 - ▶ EDX, DX, DH, DL



Accessing Parts of Registers

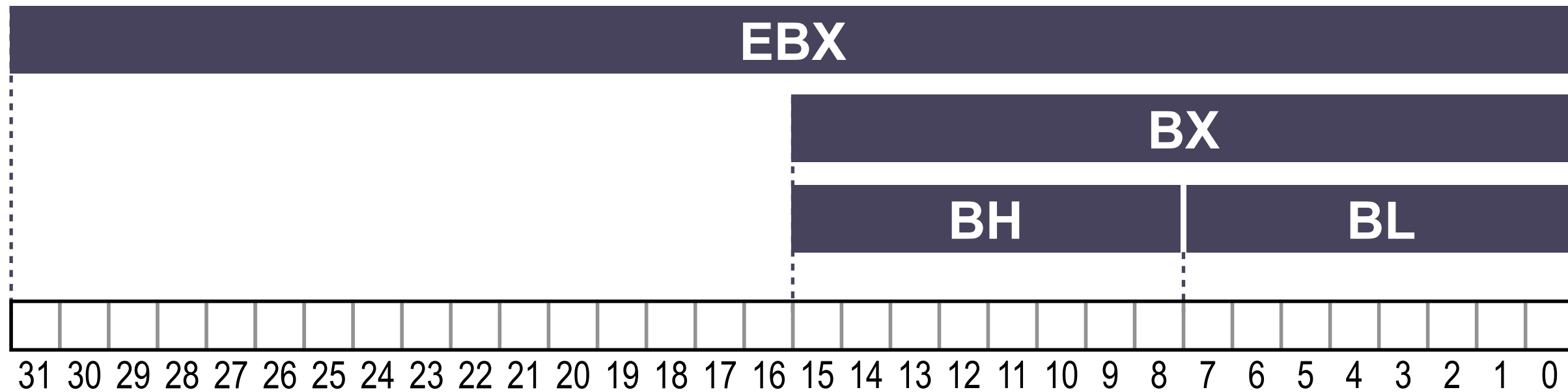


```
mov  eax,  AAAAAAAAAh
mov   ax,   BBBBh
mov   ah,   CCh
mov   al,   DDh
```

EAX Contains

```
AAAAAAAAAh
AAAABBBBh
AAAACCBBh
AAAACDDh
```


Accessing Parts of Registers



	<u>EBX</u>	<u>BX</u>	<u>BH</u>	<u>BL</u>
<code>mov ebx, 12345678h</code>	12345678h	5678h	56h	78h
<code>mov bx, 0ABCDh</code>	1234ABCDh	ABCDh	ABh	CDh
<code>mov bh, 0h</code>	123400CDh	00CDh	00h	CDh
<code>mov bl, 0FEh</code>	123400FEh	00FEh	00h	FEh

x86 Registers

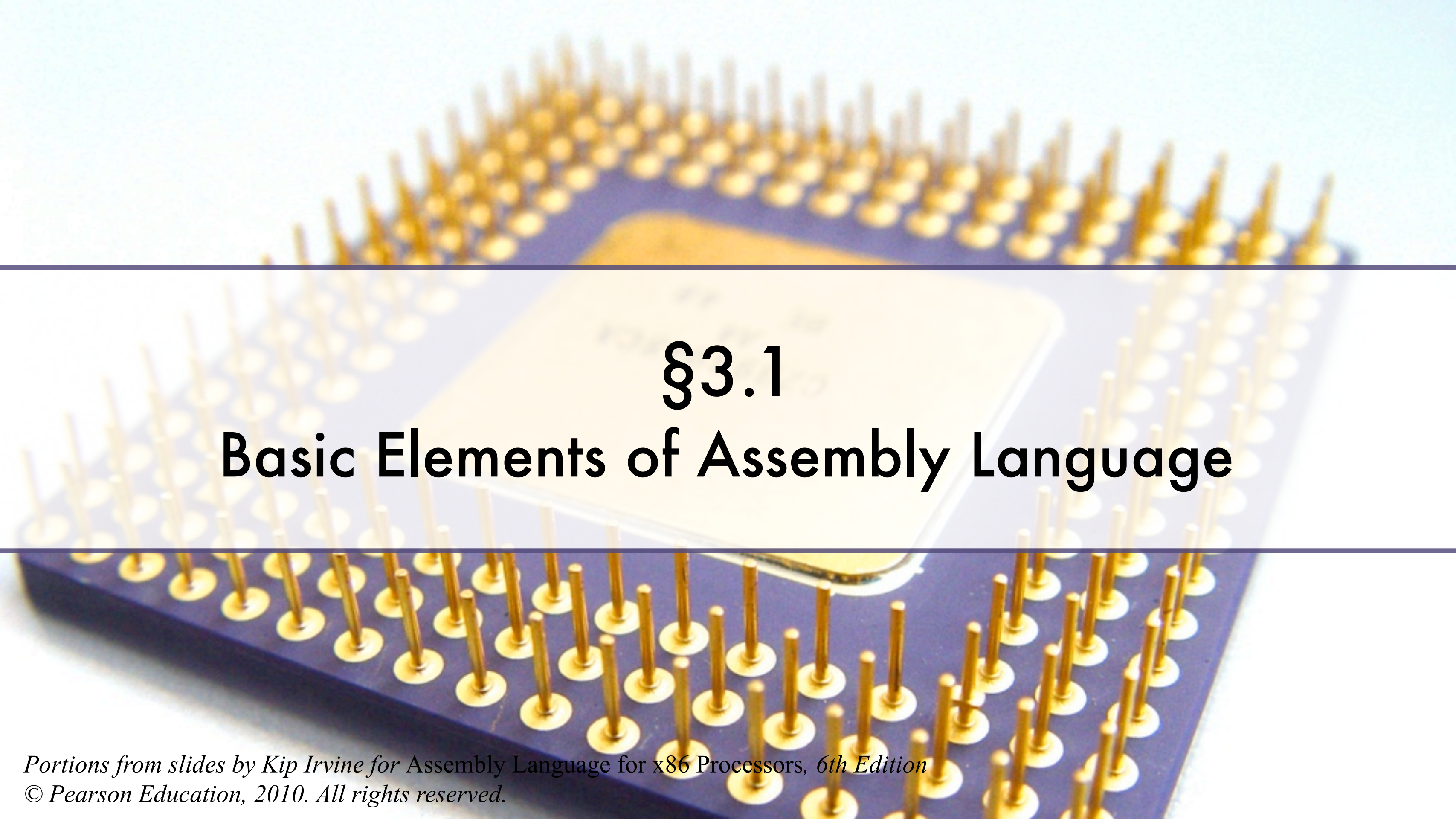


- ▶ Registers discussed so far are the *basic program execution registers*
 - ▶ EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI, EFLAGS, EIP, CS, SS, DS, ES, FS, GS
- ▶ Processor contains many more registers
 - ▶ Several for the floating-point unit (Chapter 12)
 - ▶ Others we won't use in this course

x86 Registers



- ▶ Activity 4 (front side)



§3.1

Basic Elements of Assembly Language

Important References



- ▶ Textbook Appendix A (p. 598): MASM Reference
- ▶ Textbook Appendix B (p. 620): The x86 Instruction Set
- ▶ Microsoft Macro Assembler Reference, VS2010
[http://msdn.microsoft.com/en-us/library/afzk3475\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/afzk3475(v=vs.100).aspx)
- ▶ Intel 64 and IA-32 Architectures
Software Developer Manuals

Note: Volume 2 is the instruction set reference

<http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html>

Basic Elements of Assembly Language



To be discussed:

- ▶ Instructions
- ▶ Mnemonics and Operands
- ▶ Reserved words and identifiers
- ▶ Labels
- ▶ Integer constants and integer expressions
- ▶ Character and string constants
- ▶ Directives
- ▶ Comments

Instructions (1) – JMP



- ▶ Instructions you saw in Lab 1: **mov add sub call**
- ▶ A new instruction: **jmp** (*unconditional jump*)
 - ▶ Like a “goto” statement – go to the instruction with a given label
 - ▶ Prefix any instruction with *label:* – then you can jmp to *label*

Example 1

```
mov eax, 2
jmp write
mov eax, 1
write: call WriteDec
```

*Skips over mov eax, 1
and displays 2*

Example 2

```
start: mov eax, 0
      jmp start
```

*Infinite loop: keep
setting EAX to 0*

Example 3

```
top: call ReadDec
     call WriteDec
     jmp top
```

*Infinite loop: read unsigned
integer, then display it*

Instructions (2)



- ▶ Assembled into machine code by assembler
- ▶ Executed at runtime by the CPU
- ▶ We use the Intel IA-32 instruction set

- ▶ An instruction contains:
 - ▶ Label (optional) → `something:`
 - ▶ Mnemonic (required) → `mov`
 - ▶ Operand(s) (depends on the instruction) → `b1, 0FEh`
 - ▶ Comment (optional) → `; Hello`

Mnemonics & Operands



Instruction Mnemonics

- ▶ memory aid
- ▶ examples: MOV, ADD, SUB, CALL, JMP

Instructions can take different types of operands

- ▶ constant `mov eax, 4000`
- ▶ constant expression `mov eax, (8*1000)/2`
- ▶ register `mov eax, 4000`
- ▶ memory (data label) *(you'll see examples of this later)*

Constants and constant expressions are often called *immediate values*

Instruction Format Examples



- ▶ No operands

- ▶ `stc` ; set Carry flag

- ▶ One operand

- ▶ `inc eax` ; register

- ▶ `inc myByte` ; memory

- ▶ Two operands

- ▶ `add ebx, ecx` ; register, register

- ▶ `sub myByte, 25` ; memory, constant

- ▶ `add eax, 36 * 25` ; register, constant-expression

Reserved Words & Identifiers



- ▶ Reserved words cannot be used as identifiers
 - ▶ Instruction mnemonics, directives, type attributes, operators, predefined symbols
 - ▶ See MASM reference in Appendix A
- ▶ Identifiers
 - ▶ 1–247 characters, including digits
 - ▶ Case *insensitive*
 - ▶ first character must be a letter, `_`, `@`, `?`, or `$`

Labels



- ▶ Act as place markers
 - ▶ marks the address (offset) of code and data
- ▶ Follow identifier rules
- ▶ Data label
 - ▶ must be unique
 - ▶ example: **msg** (not followed by colon)
- ▶ Code label
 - ▶ target of instructions like **jmp**
 - ▶ example: **done:** (followed by colon)

Example

```
.....  
INCLUDE Irvine32.inc  
  
.data  
msg BYTE "Hello", 0  
  
.code  
main PROC  
    mov edx, offset msg  
    call WriteString  
    jmp done  
done: exit  
main ENDP  
END main
```


Comments



- ▶ Comments are good!
 - ▶ Summarize the contents of a file or procedure
 - ▶ Explain tricky coding techniques
 - ▶ Don't just re-state what each instruction does
- ▶ Single-line comments begin with semicolon (;)
- ▶ Multi-line comments begin with COMMENT directive and a programmer-chosen character; end with the same programmer-chosen character

Integer Constants



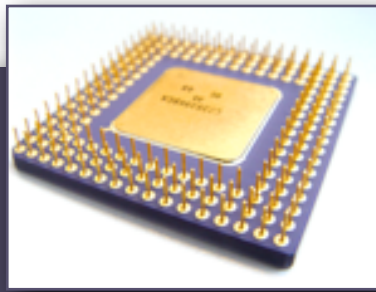
- ▶ Optional leading + or – sign
- ▶ Binary, decimal, hexadecimal, or octal digits
- ▶ Common radix characters:
 - ▶ h – hexadecimal
 - ▶ d – decimal
 - ▶ b – binary
 - ▶ r – encoded real

Examples: 30d, 6Ah, 42, 1101b

`mov bl, 0FEh`

Hexadecimal beginning with letter: 0A5h – **must prefix with 0!**

Integer Expressions (1)



- ▶ Anywhere you can use an integer constant, you can use an integer *expression*
- ▶ Operators and precedence levels:

Operator	Name	Precedence Level
()	parentheses	1
$+$, $-$	unary plus, minus	2
$*$, $/$	multiply, divide	3
MOD	modulus	3
$+$, $-$	add, subtract	4

▶ Examples:

Expression	Value
$16 / 5$	3
$-(3 + 4) * (6 - 1)$	-35
$-3 + 4 * 6 - 1$	20
$25 \text{ mod } 3$	1

Integer Expressions (2)



- ▶ Anywhere you can use an integer constant, you can use an integer *expression*
- ▶ Only *constant* expressions are permitted – can't contain register names, etc.
- ▶ Q. Which of the following are permitted?

In .data: **something BYTE 93-1** ✓

In .data: **five BYTE 5**
 six BYTE five+1 ✗ (five is a label – not constant-valued)

In .code: **mov eax, 3*100+3** ✓

In .code: **mov eax, 5**
 mov ebx, eax+1 ✗ (eax is a register – not constant-valued)

Character & String Constants



- ▶ Enclose character in single or double quotes
 - ▶ 'A', "x"
 - ▶ ASCII character = 1 byte
- ▶ Enclose strings in single or double quotes
 - ▶ "ABC"
 - ▶ 'xyz'
 - ▶ Each character occupies a single byte
- ▶ Embedded quotes:
 - ▶ 'Say "Goodnight," Gracie'

Directives



- ▶ Commands that are recognized and acted upon by the assembler
 - ▶ Not part of the Intel instruction set
 - ▶ Used to declare code, data areas, select memory model, declare procedures, etc.
 - ▶ Case *insensitive*
- ▶ Different assemblers have different directives
 - ▶ NASM not the same as MASM, for example

Homework



- ▶ **Homework 1** due in Canvas by Friday at 2:00 p.m. – come by office hours for help
- ▶ For next class (Friday, September 5):
 - ▶ Read **Section 3.1** (6/e pp. 58–66, 7/3 pp. 54–63) - mostly covered today
 - ▶ Read **Section 3.2** (6/e pp. 66–70, 7/e pp. 63–70) - *not covered in any lectures*
 - ▶ Goal: understand the sample program so you can read Section 3.4 later on
 - ▶ Sample program changed slightly from 6th edition to 7th edition; read either one