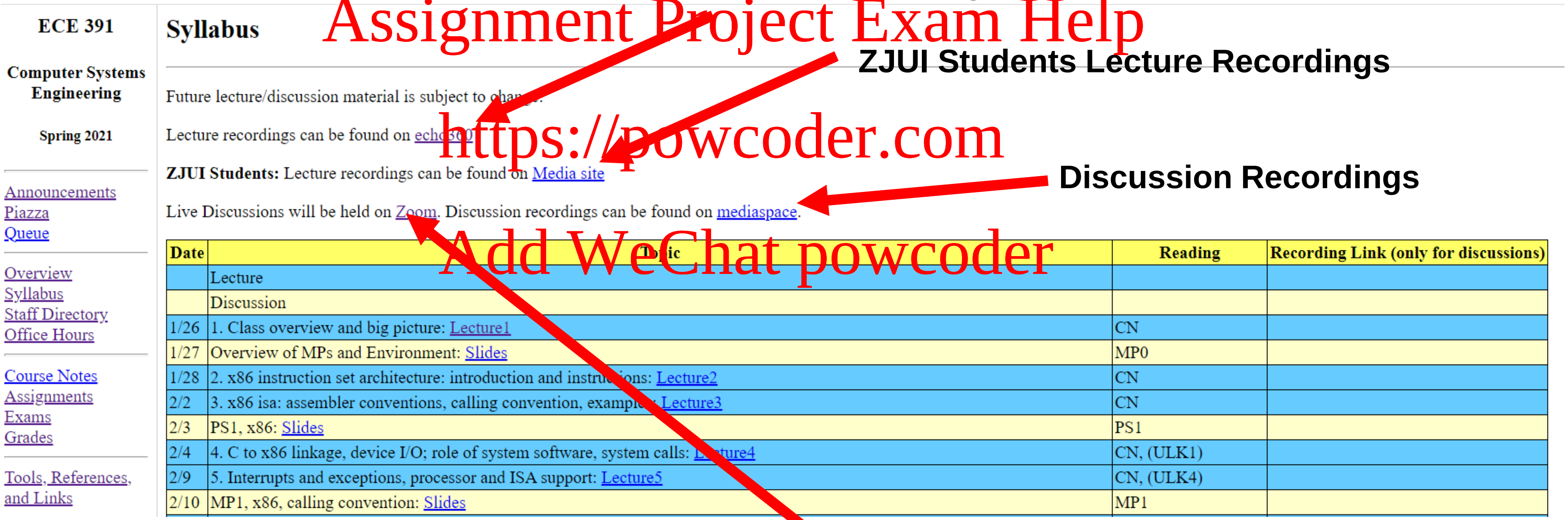
Lecture Topics

- x86 instructions
- Operate instructions
- Data movement instructions
- Conditional codes
- Control flow instructions
- Assembler conventions
- Code example

Aministrivia

- MP0
  - **In TAs office hours by 2/2**
  - you can hand in anytime during office hours



ECE 391

Computer Systems Engineering

Spring 2021

Announcements
Piazza
Queue

Overview
Syllabus
Staff Directory
Office Hours

Course Notes
Assignments
Exams
Grades

Tools, References, and Links

Syllabus

Future lecture/discussion material is subject to change.

Lecture recordings can be found on echo360.

ZJUI Students: Lecture recordings can be found on Media site

Live Discussions will be held on Zoom. Discussion recordings can be found on mediaspace.

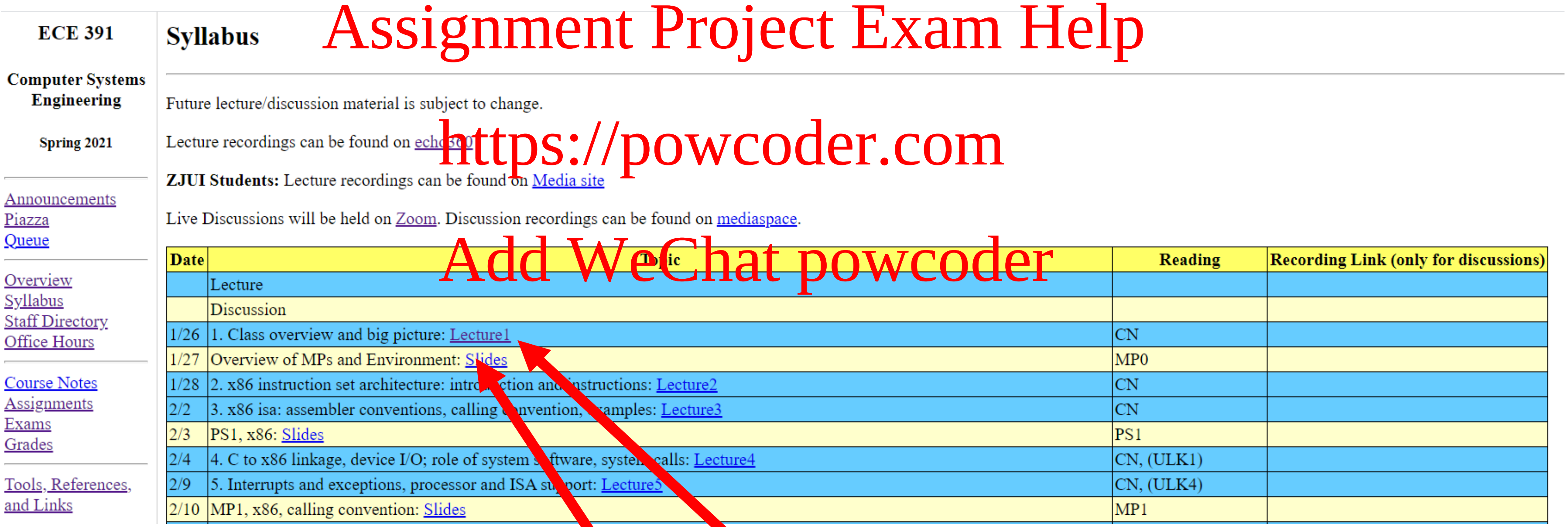
Date	Topic	Reading	Recording Link (only for discussions)
	Lecture		
	Discussion		
1/26	1. Class overview and big picture: Lecture1	CN	
1/27	Overview of MPs and Environment: Slides	MP0	
1/28	2. x86 instruction set architecture: introduction and instructions: Lecture2	CN	
2/2	3. x86 isa: assembler conventions, calling convention, examples: Lecture3	CN	
2/3	PS1, x86: Slides	PS1	
2/4	4. C to x86 linkage, device I/O; role of system software, system calls: Lecture4	CN, (ULK1)	
2/9	5. Interrupts and exceptions, processor and ISA support: Lecture5	CN, (ULK4)	
2/10	MP1, x86, calling convention: Slides	MP1	

Lecture Recordings

ZJUI Students Lecture Recordings

Discussion Recordings

Live Discussion (Zoom link)



ECE 391

Computer Systems Engineering

Spring 2021

Announcements
Piazza
Queue

Overview
Syllabus
Staff Directory
Office Hours

Course Notes
Assignments
Exams
Grades

Tools, References, and Links

Syllabus

Future lecture/discussion material is subject to change.

Lecture recordings can be found on echo360

ZJUI Students: Lecture recordings can be found on Media site

Live Discussions will be held on Zoom. Discussion recordings can be found on mediaspace.

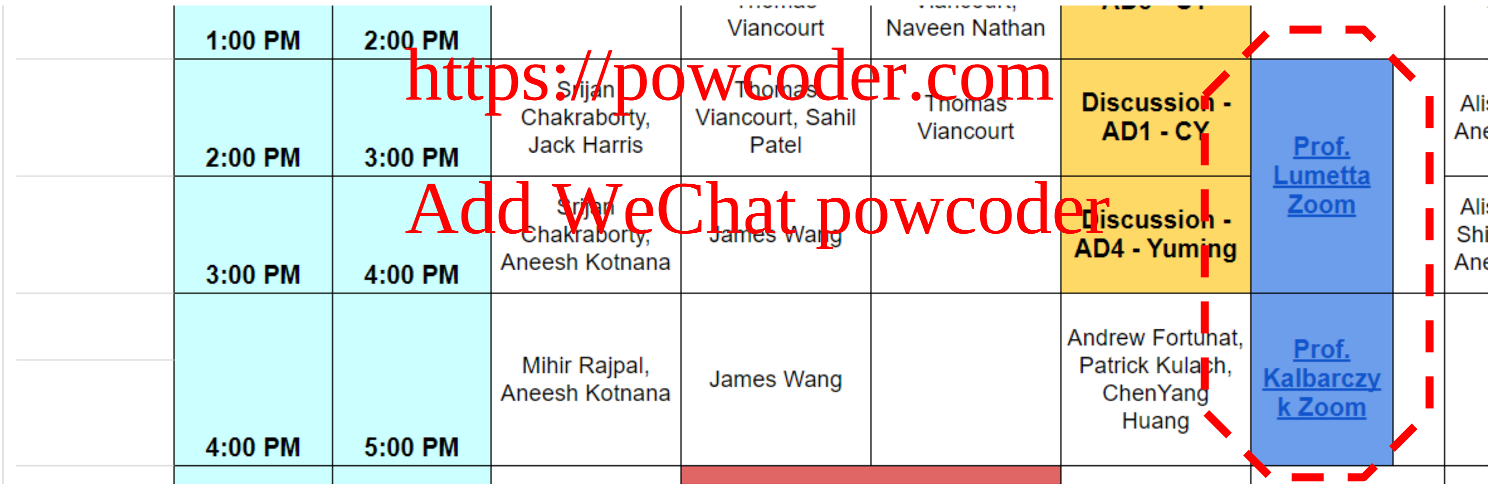
Date	Topic	Reading	Recording Link (only for discussions)
	Lecture		
	Discussion		
1/26	1. Class overview and big picture: Lecture1	CN	
1/27	Overview of MPs and Environment: Slides	MP0	
1/28	2. x86 instruction set architecture: introduction and instructions: Lecture2	CN	
2/2	3. x86 isa: assembler conventions, calling convention, examples: Lecture3	CN	
2/3	PS1, x86: Slides	PS1	
2/4	4. C to x86 linkage, device I/O; role of system software, system calls: Lecture4	CN, (ULK1)	
2/9	5. Interrupts and exceptions, processor and ISA support: Lecture5	CN, (ULK4)	
2/10	MP1, x86, calling convention: Slides	MP1	

Lecture Slides

Discussion Slides

Office Hours

Go to “Office Hours” tab on the class web sit:

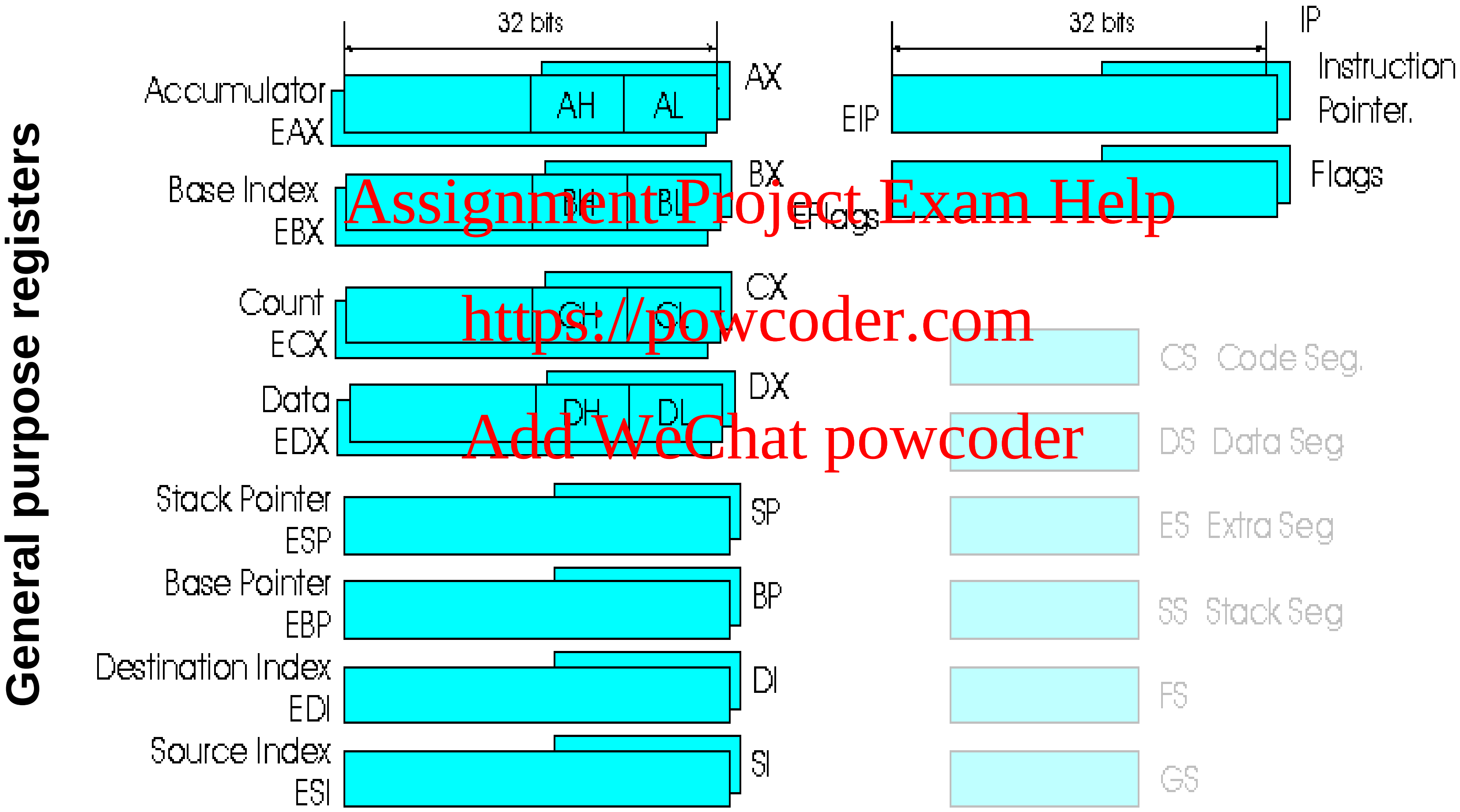


1:00 PM	2:00 PM		Viancourt	Naveen Nathan			
2:00 PM	3:00 PM	Srijan Chakraborty, Jack Harris	Thomas Viancourt, Sahil Patel	Thomas Viancourt	Discussion - AD1 - CY	Prof. Lumetta Zoom	Ali Ane
3:00 PM	4:00 PM	Srijan Chakraborty, Aneesh Kotnana	James Wang		Discussion - AD4 - Yuming		Ali Shi Ane
4:00 PM	5:00 PM	Mihir Rajpal, Aneesh Kotnana	James Wang		Andrew Fortunat, Patrick Kulach, ChenYang Huang	Prof. Kalbarczyk Zoom	

Introduction and Basics

- What is x86? (Intel-32-bit architecture)
  - variable-length instruction encoding (1-16 bytes)
  - small register set: 8 mostly general purpose
  - 32-bit, byte-addressable address space
  - complex addressing modes
  - many data types supported by hardware

Registers


General purpose registers
32 bits
Accumulator
EAX
AH
AL
AX
Base Index
EBX
BH
BL
BX
Count
ECX
CH
CL
CX
Data
EDX
DH
DL
DX
Stack Pointer
ESP
SP
Base Pointer
EBP
BP
Destination Index
EDI
DI
Source Index
ESI
SI
32 bits
IP
Instruction
Pointer.
EIP
EFlags
Flags
CS Code Seg.
DS Data Seg
ES Extra Seg
SS Stack Seg
FS
GS

Registers

extended, i.e., 32-bit

EAX accumulator

EBX base (of array)

ECX count (for loops)

EDX data (2nd operand)

ESI source index (string copy)

EDI destination index

EBP base pointer (base of stack frame)

ESP stack pointer

EIP instruction pointer

EFLAGS flags/condition codes

- Use % as a prefix for registers in assembly
- Other registers: floating-point, MMX, etc. (not discussed in this class)

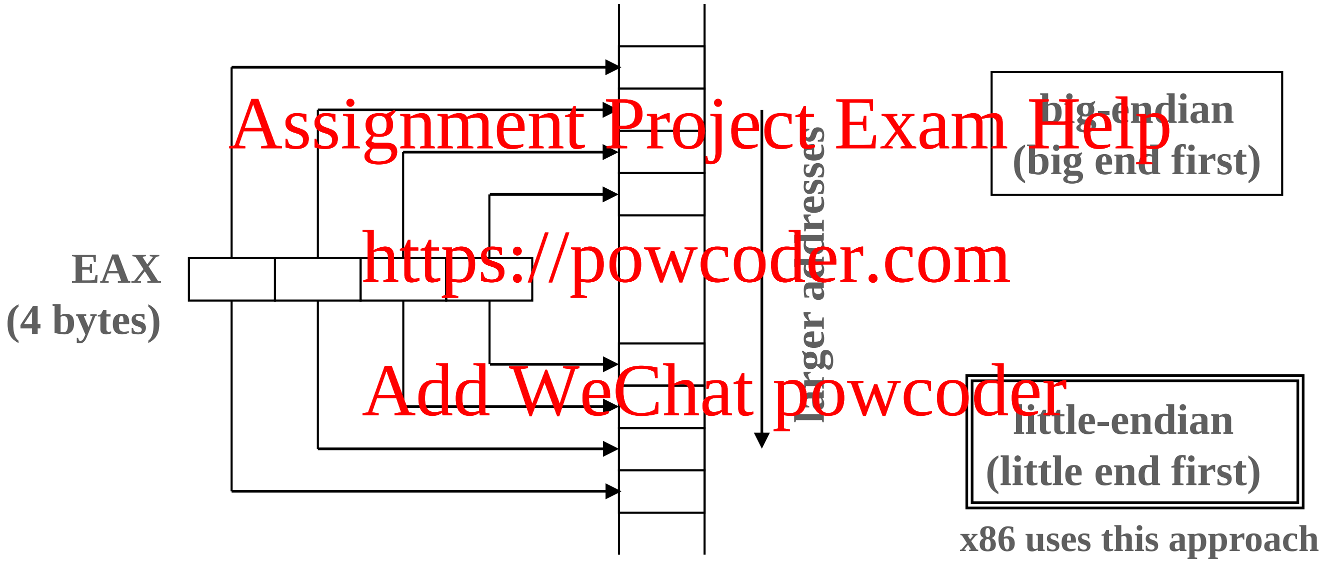
Data Types

- 8-, 16-, 32-bit unsigned and 2’s complement
- IEEE single- and double-precision floating point
- Intel “extended” f.p. (80-bit)
- ASCII strings
- Binary-coded decimal

Memory

- Microprocessor addresses a maximum of 2^n different memory locations, where n is a number of bits on the address bus .
- Memory
  - x86 supports byte addressable memory
  - byte (8 bits) is a basic memory unit
  - e.g., when you specify address 24 in memory, you get the entire eight bits
  - when the microprocessors address a 16-bit word of memory, two consecutive bytes are accessed

How are bytes stored to memory?



0x12345678 ⇨ **0x78, 0x56, 0x34, 0x12**

in consecutive memory locations

x86 Instructions – Basics

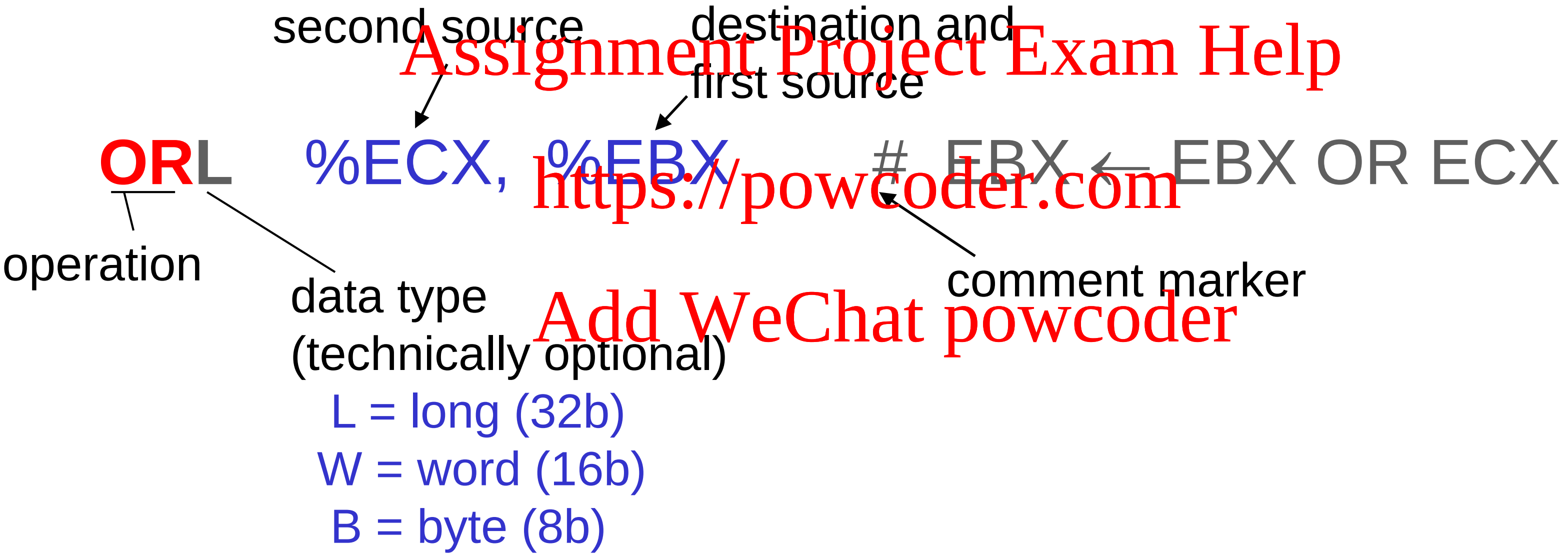
- Operations, data movement, condition codes, control flow, stack ops, data size conversion

Operations

arithmetic	logical	shift
ADD	AND	SHL
SUB	OR	SAR
NEG	NOT	SHR
INC	XOR	ROL
DEC		ROR

- typically 2-operand instructions (destination and one source are the same)

Operations – Example



second source

destination and first source

ORL %ECX, %EBX # EBX ← EBX OR ECX

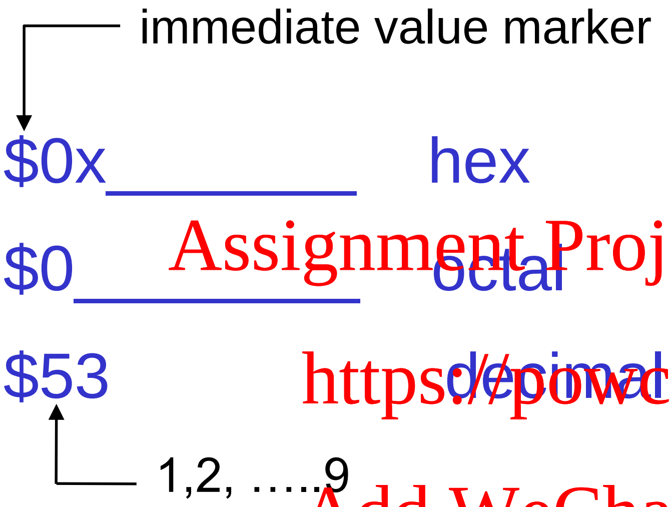
operation

data type (technically optional)

L = long (32b)
W = word (16b)
B = byte (8b)

comment marker

Immediate Values


immediate value marker

$0x________ hex

$0__________ octal

$53 decimal

1,2, …..9

what does the following instruction do?

ANDL 0, %EAX

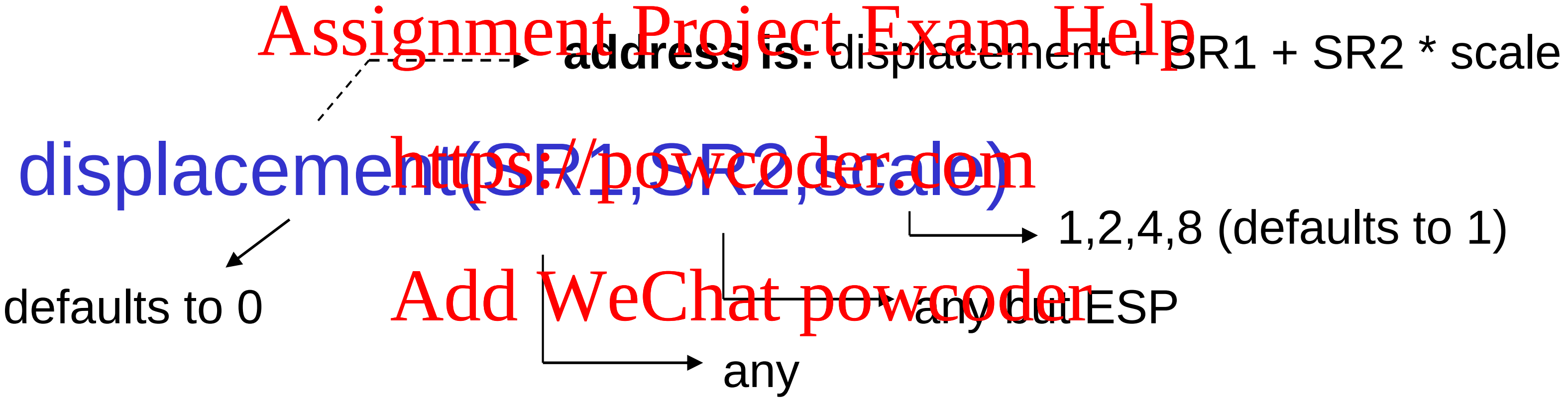
answer is NOT
EAX ← 0

instead:
EAX ← EAX AND M[0]
(usually crashes)

- how big can they get?
  - usually up to 32 bits
  - larger constants → longer instructions
  - length of operand must be encoded, too

Data Movement: Memory Addressing

Memory operand has this general form



address is: displacement + SR1 + SR2 * scale

displacement(SR1,SR2,scale)

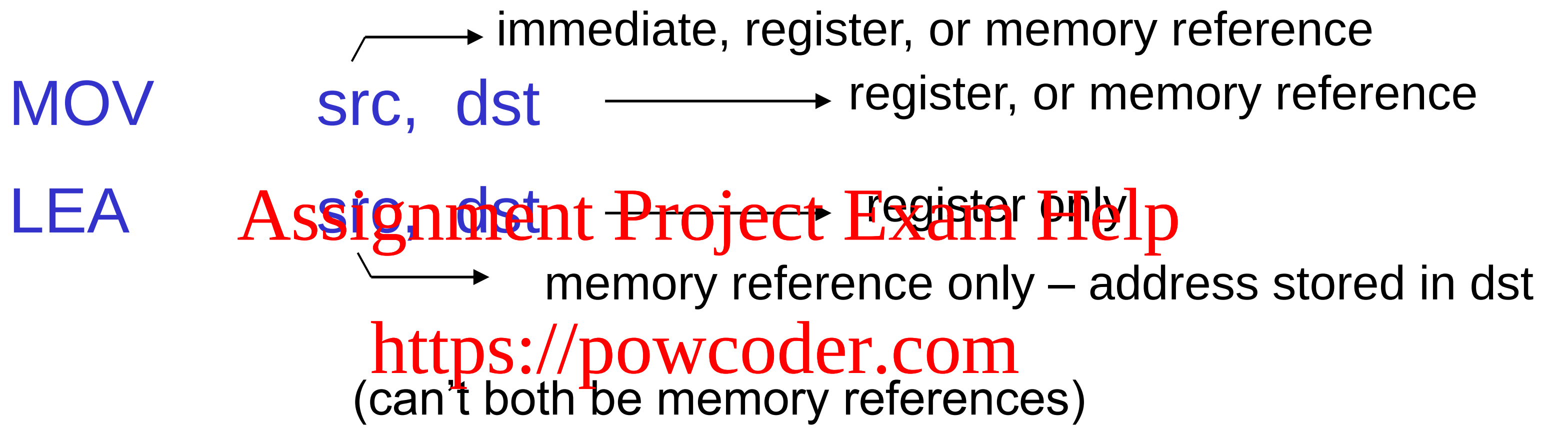
1,2,4,8 (defaults to 1)

defaults to 0

any but ESP

any

Instructions



MOV src, dst

immediate, register, or memory reference

register, or memory reference

LEA src, dst

register only

memory reference only – address stored in dst

(can't both be memory references)

- Examples:

MOVW %DX, 0x10(%EBP) # M[EBP + 0x10] ← DX

MOVB (%EBX,%ESI,4), %CL # CL ← M[EBX + ESI * 4]

Instructions: Examples to Solve

EAX ← M[0x10000 + ECX]

[answer] **MOVL** 0x10000(%ECX), %EAX

M[LABEL] ← DI

[answer] **MOVW** %DI, LABEL

ESI ← LABEL + 4 (two ways!)

[answer] **MOVL** $LABEL + 4, %ESI

LEAL LABEL + 4, %ESI

ESI ← LABEL + EAX + 4

[answer] **LEAL** LABEL + 4(%EAX), %ESI

expression calculated by assembler;
instruction holds one displacement value

Instructions: Examples to Solve

EAX ← M[0x10000 + ECX]

[answer] **MOVL 0x10000(%ECX) , %EAX**

M[LABEL] ← DI

[answer] **MOVW %DI , LABEL**

ESI ← LABEL + 4 (two ways!)

[answer] **MOVL $LABEL + 4 , %ESI**

LEAL LABEL + 4 , %ESI

ESI ← LABEL + EAX + 4

[answer] **LEAL LABEL + 4(%EAX), %ESI**

Condition Codes (in EFLAGS)

- Among others (not mentioned in this class)…

SF: sign flag: result is negative when viewed as
2's complement data type

ZF: zero flag: result is exactly zero

CF: carry flag: unsigned carry or borrow occurred
(or other, instruction-dependent meaning, e.g., on shifts)

OF: overflow flag: 2's complement overflow
(and other instruction-dependent meanings)

PF: parity flag: even parity in result (even # of 1 bits)

What Instructions Set Flags (condition codes)?

- Not all instructions set flags
- Some instructions set some flags!
- Use **CMP** or **TEST** to set flags:

CMPL %EAX, %EBX # flags ← (EBX – EAX)

TESTL %EAX, %EBX # flags ← (EBX *AND* EAX)

- Note that EBX does not change in either case
- What combinations of flags are needed for unsigned/signed relationships comparator?

Control Flow Instructions (1)

- Consider two three-bit values A and B; How to decide if A<B?

	#1	#2	#3	#4	#5	#6
A	010	010	010	110	110	110
B	-000	-110	-111	-000	-011	-111
C	010	100	011	110	011	111
CF	0	1	1	0	0	1
OF	0	1	0	0	1	0
SF	0	1	0	1	0	1
unsigned <	No	Yes	Yes	No	No	Yes
signed <	No	No	No	Yes	Yes	Yes

Control Flow Instructions (2)

- Note that CF suffices for unsigned <
- What about signed < ?

	CF/OF			
SF	00	01	11	10
0	0	1	x	0
1	1	x	0	1

- Answer: OF *XOR* SF

Branch Mnemonics

- Unsigned comparisons: “above” and “below”
- Signed comparisons: “less” and “greater”
- Both: equal/zero

unsigned	**`jne`**	**`jb`**	**`jbe`**	**`je`**	**`jae`**	**`ja`**
relationship	≠	<	≤	=	≥	>
signed	**`jne`**	**`jl`**	**`jle`**	**`je`**	**`jge`**	**`jg`**

- in general, can add “n” after “j” to negate sense
- forms shown are those used when disassembling
  - do not expect binary to retain your version
  - e.g., “**`jnae`**” becomes “**`jb`**”