Due: Tue 09/16/03, Section 0101 (Chang) & Section 0301 (Macneil)
Wed 09/17/03, Section 0201 (Patel & Bourner)

Instructions: For the following questions, *show all of your work*. It is not sufficient to provide the answers.

Exercise 1. Convert the following decimal numbers to hexadecimal representations of 16-bit two's complement numbers.

a. 798

b. 30142

c. -23456

d. -1024

Exercise 2. Convert the following 16-bit two's complement numbers in hexadecimal representation to decimal.

a. $FFF0_{16}$

b. $07FF_{16}$

c. $00A8_{16}$

d. 8000_{16}

Exercise 3. Write the following decimal numbers in IEEE-754 single precision format. Give your answers in binary.

a. 2.54

b. 2.71828

c. -74.6875

d. 64000

Exercise 4. Write the decimal equivalents for these IEEE-754 single precision floating point numbers given in binary.

a. `0 10000011 01100000000000000000000`

b. `1 10000011 00010000000000000000000`

c. `1 10000000 00000000000000000000000`

d. `0 00000001 11010000000000000000000`

CMSC 313 Lecture 04

- Moore's "Law"
- Evolution of the Pentium Chip
- IA-32 Basic Execution Environment
- IA-32 General Purpose Registers
- "Hello World" in Linux Assembly Language
- Addressing Modes

Moore’s “Law”

- In the mid-1960’s, Intel Chairman of the Board Gordon Moore observed that “the number of transistors that would be incorporated on a silicon die would double every 18 months for the next several years.”
- His prediction has continued to hold true.
- Perhaps a self-fulfilling prophecy?

Table 2-1. Key Features of contemporary IA-32 processors

Intel Processor	Date Intro-duced	Micro-architecture	Clock Frequency at Intro-duction	Transis-tors per Die	Register Sizes[1]	System Bus Bandwidth	Max. Extern. Addr. Space	On-die Caches[2]
Pentium III processor[3]	1999	P6	700 MHz	28 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	Up to 1.06 GB/s	64 GB	32KB L1; 256KB L2
Pentium 4 processor	2000	Intel NetBurst micro-architecture	1.50 GHz	42 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K µop Execution Trace Cache; 8KB L1; 256KB L2

NOTES:

1. The register size and external data bus size are given in bits.
2. First level cache is denoted using the abbreviation L1, 2nd level cache is denoted as L2.
3. Intel Pentium III and Pentium III Xeon processors, with advanced transfer cache and built on 0.18 micron process technology, were introduced in October 1999.

Table 2-2. Key Features of previous generations of IA-32 Processor

Intel Processor	Date Intro-duced	Max. Clock Frequency at Intro-duction	Transis-tors per Die	Register Sizes[1]	Ext. Data Bus Size[2]	Max. Extern. Addr. Space	Caches
8086	1978	8 MHz	29 K	16 GP	16	1 MB	None
Intel 286	1982	12.5 MHz	134 K	16 GP	16	16 MB	Note 3
Intel386 DX Processor	1985	20 MHz	275 K	32 GP	32	4 GB	Note 3
Intel486 DX Processor	1989	25 MHz	1.2 M	32 GP 80 FPU	32	4 GB	L1: 8KB
Pentium Processor	1993	60 MHz	3.1 M	32 GP 80 FPU	64	4 GB	L1:16KB
Pentium Pro Processor	1995	200 MHz	5.5 M	32 GP 80 FPU	64	64 GB	L1: 16KB L2: 256KB or 512KB
Pentium II Processor	1997	266 MHz	7 M	32 GP 80 FPU 64 MMX	64	64 GB	L1: 32KB L2: 256KB or 512KB
Pentium III Processor	1999	500 MHz	8.2 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32KB L2: 512KB

NOTES:

1. The register size and external data bus size are given in bits. Note also that each 32-bit general-purpose (GP) registers can be addressed as an 8- or a 16-bit data registers in all of the processors
2. Internal data paths that are 2 to 4 times wider than the external data bus for each processor.

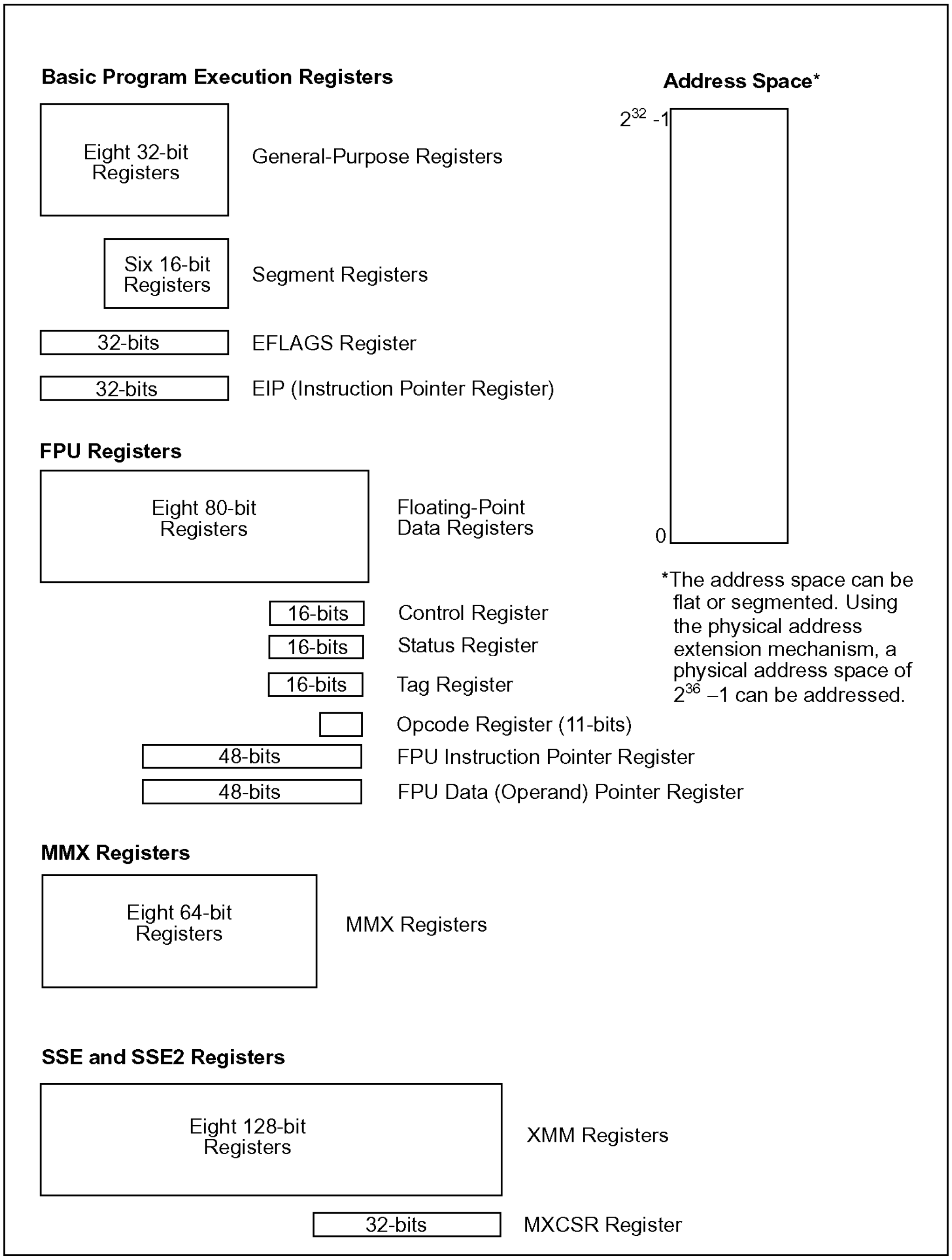


Figure 3-1. IA-32 Basic Execution Environment

General-Purpose Registers

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

Figure 3-4. Alternate General-Purpose Register Names

The special uses of general-purpose registers by instructions are described in Chapter 5, *Instruction Set Summary*, in this volume and Chapter 3, *Instruction Set Reference*, in the *Intel Architecture Software Developer's Manual, Volume 2*. The following is a summary of these special 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.
- ESI—Pointer to data in the segment pointed to by the DS register; source pointer for string operations.9
- EDI—Pointer to data (or destination) in the segment pointed to by the ES register; destination pointer for string operations.
- ESP—Stack pointer (in the SS segment).
- EBP—Pointer to data on the stack (in the SS segment).

“Hello World” in Linux Assembly

- Use your favorite UNIX editor (vi, emacs, pico, ...)
- Assemble using NASM on gl.umbc.edu

```
nasm -f elf hello.asm
```

- NASM documentation is on-line.
- Need to “load” the object file

```
ld hello.o
```

- Execute

```
a.out
```

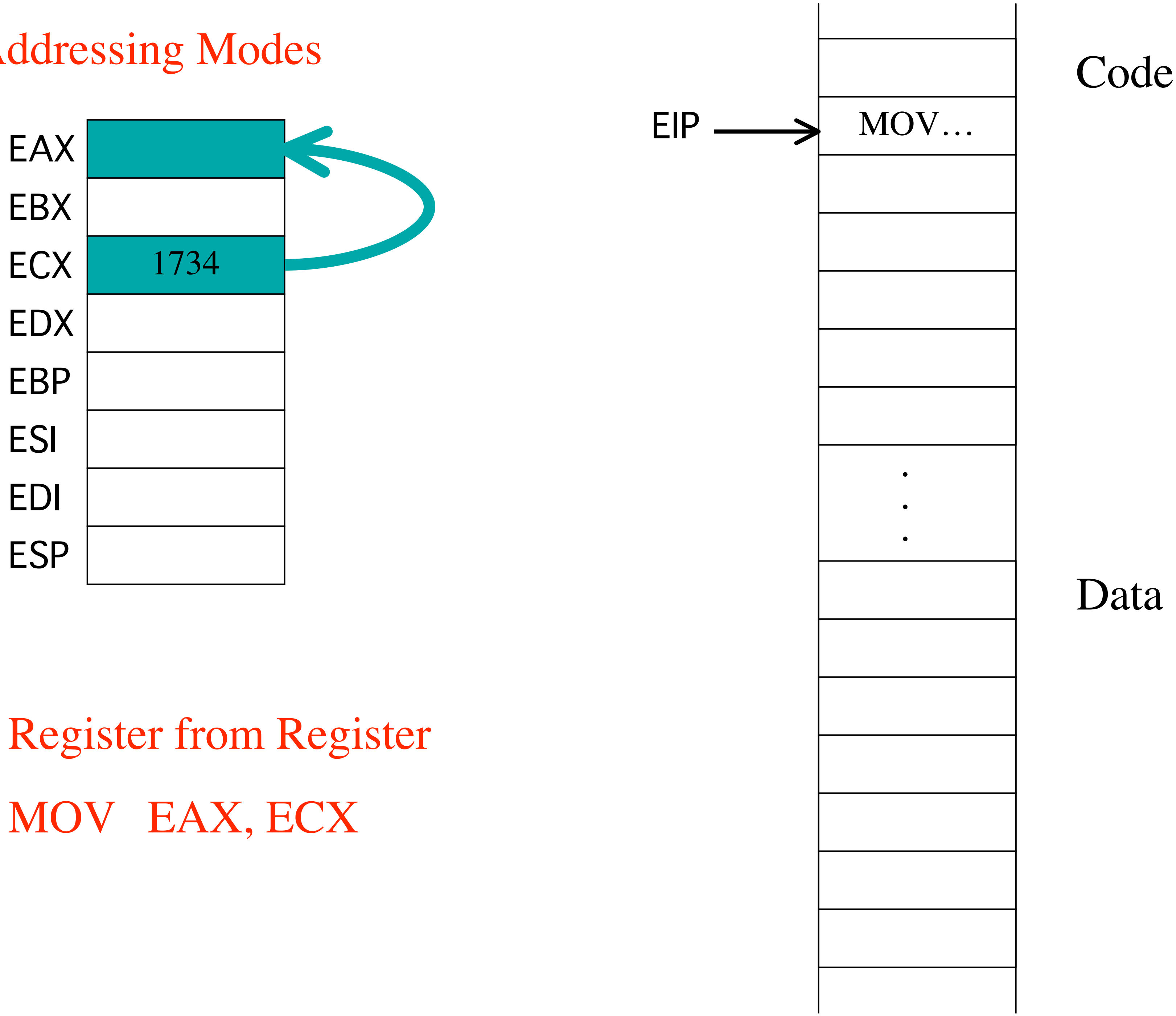
- CMSC 121 Introduction to UNIX

80x86 Addressing Modes

- We want to store the value 1734h.
- The value 1734h may be located in a register or in memory.
- The location in memory might be specified by the code, by a register, …
- Assembly language syntax for MOV

```
MOV        DEST,  SOURCE
```

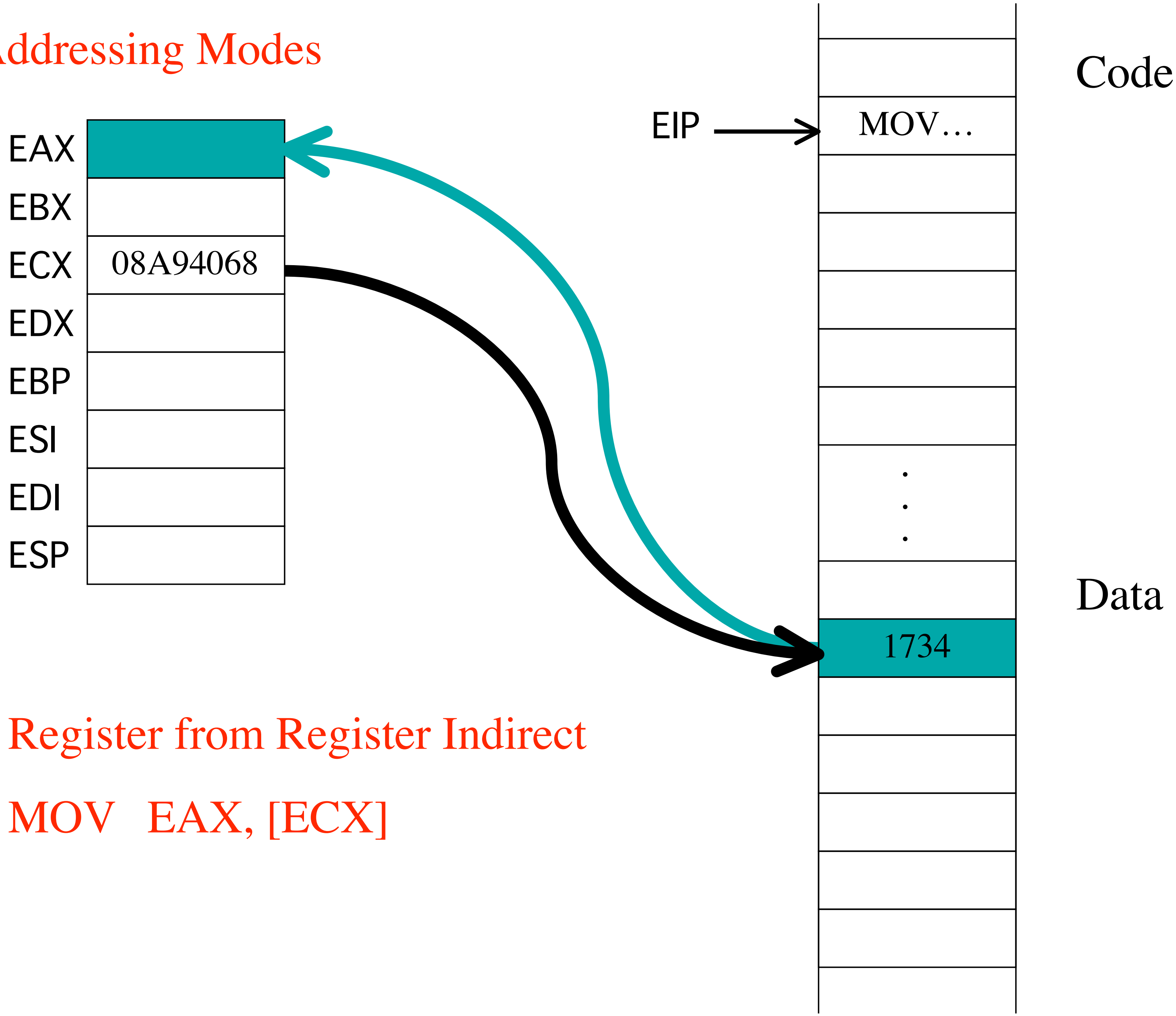
Addressing Modes



Register from Register

MOV EAX, ECX

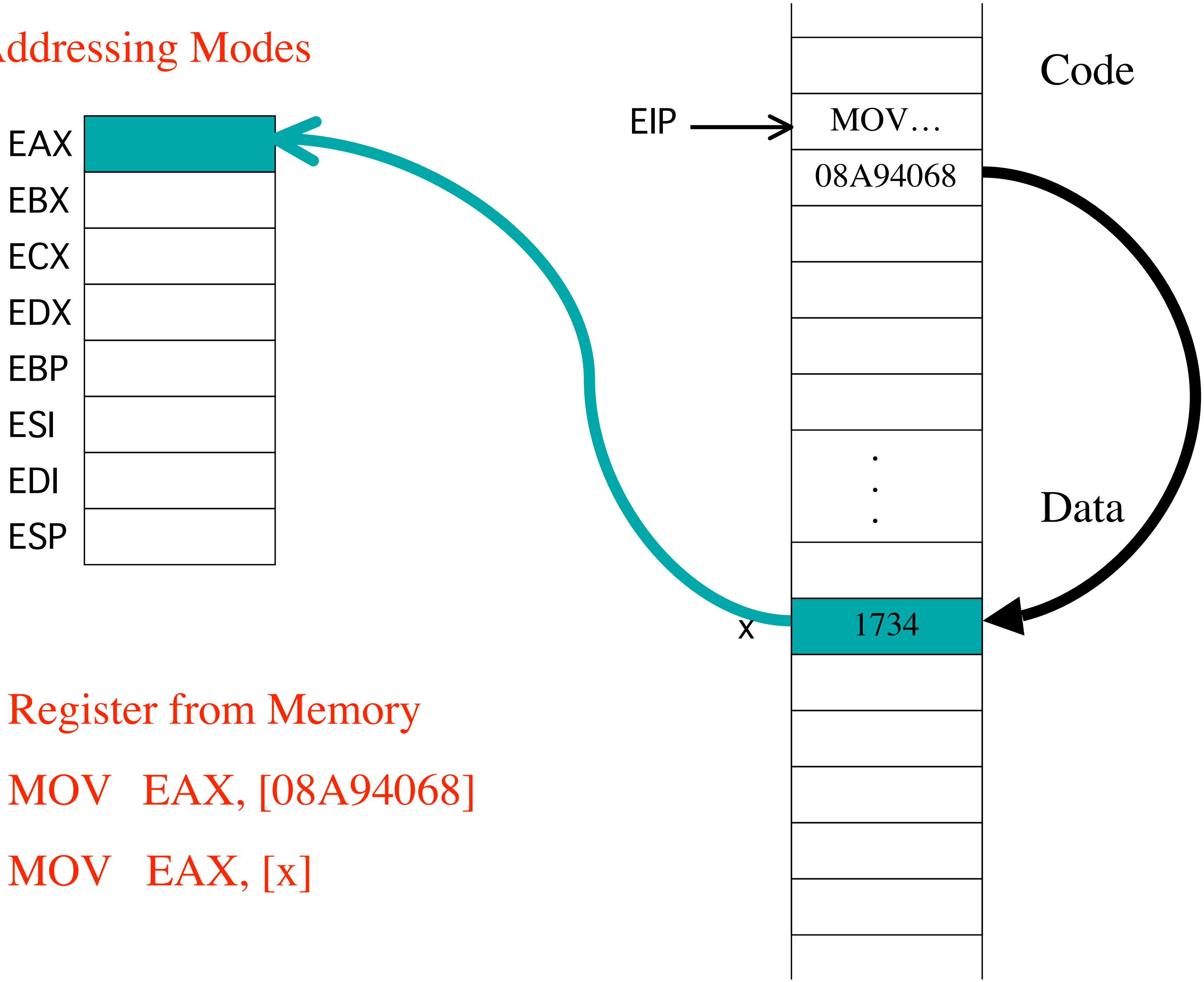
Addressing Modes



Register from Register Indirect

MOV   EAX, [ECX]

Addressing Modes

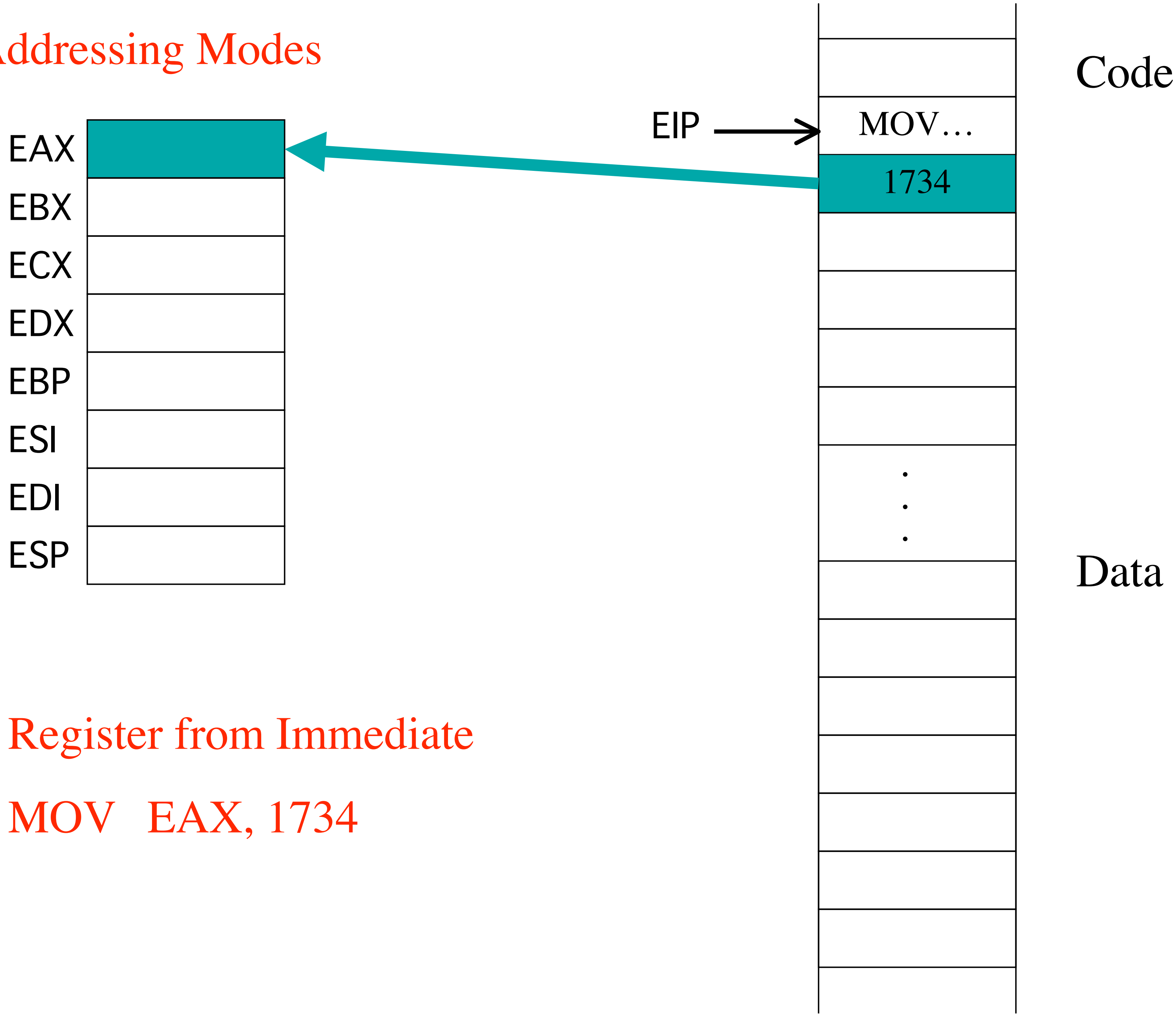


Register from Memory

MOV EAX, [08A94068]

MOV EAX, [x]

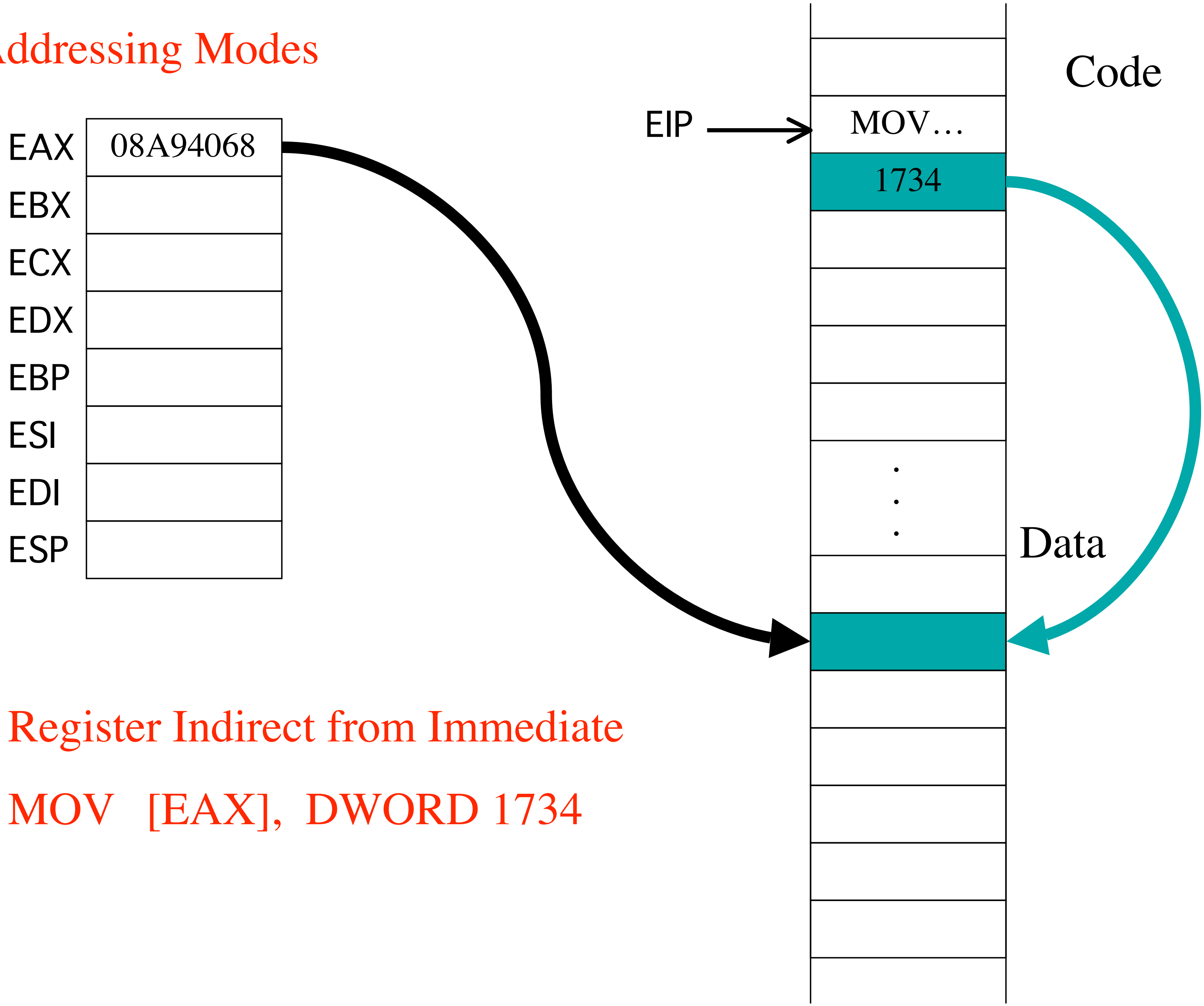
Addressing Modes



Register from Immediate

MOV   EAX, 1734

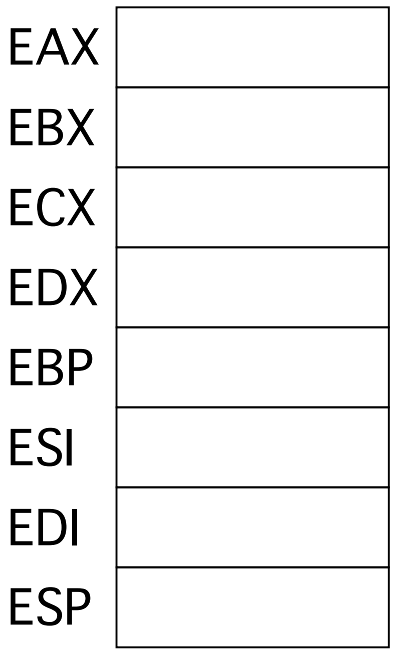
Addressing Modes

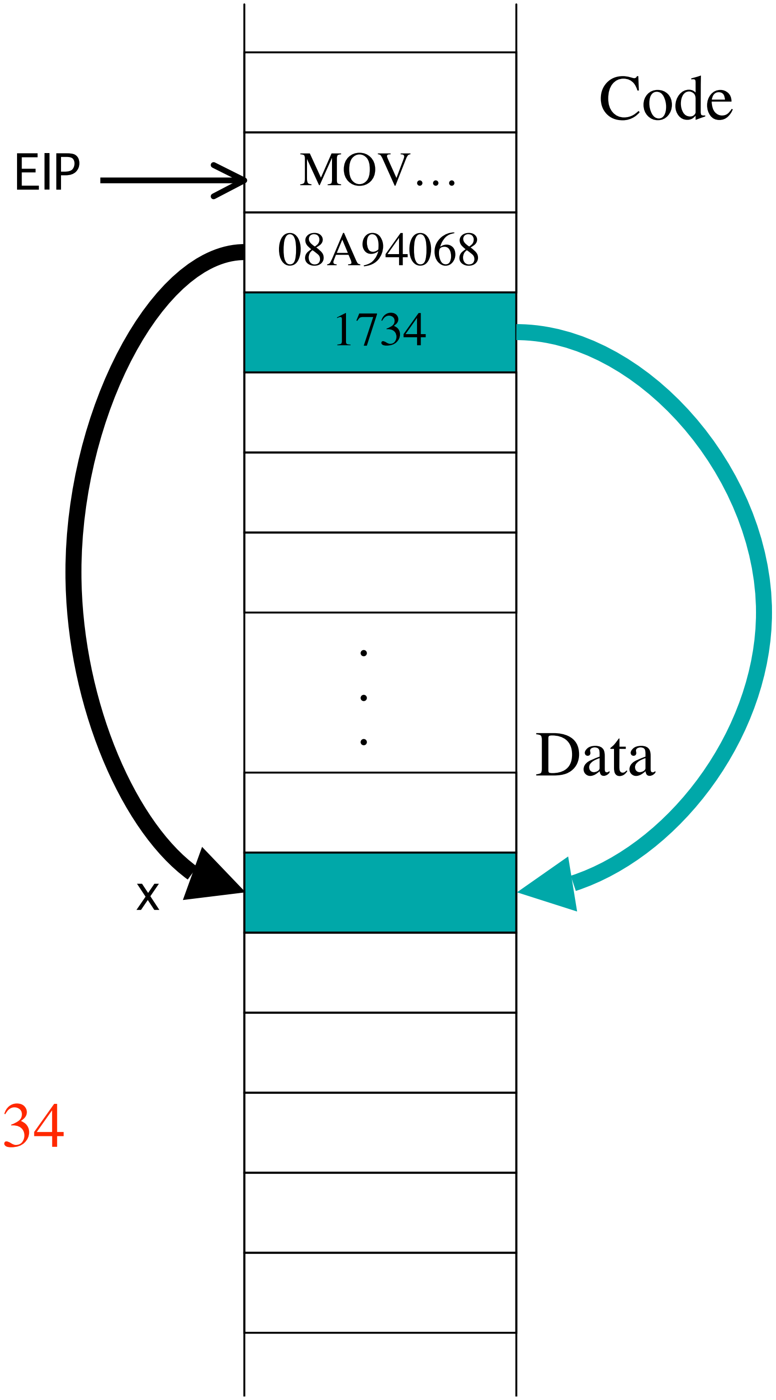


Register Indirect from Immediate

MOV [EAX], DWORD 1734

Addressing Modes





Register Indirect from Immediate

MOV [08A94068], DWORD 1734

MOV [x], DWORD 1734

Notes on Addressing Modes

- More complicated addressing modes later:

```
MOV        EAX, [ESI+4*ECX+12]
```

- Figures not drawn to scale. Constants 1734h and 08A94068h take 4 bytes (little endian).
- Some addressing modes are not supported by some operations.
- Labels represent addresses not contents of memory.

toupper.asm

- Prompt for user input.
- Use Linux system call to get user input.
- Scan each character of user input and convert all lower case characters to upper case.
- How to:
  - work with 8-bit data
  - specify ASCII constant
  - compare values
  - loop control

Debugging Assembly Language Programs

- Cannot just put print statements everywhere.
- Use gdb to:
  - examine contents of registers
  - exmaine contents of memory
  - set breakpoints
  - single-step through program
- READ THE GDB SUMMARY ONLINE!

gdb ommand Summary

Command	**Example**	**Description**
run		start program
quit		quit out of gdb
cont		continue execution after a break
break [addr]	break *_start+5	sets a breakpoint
delete [n]	delete 4	removes nth breakpoint
delete		removes all breakpoints
info break		lists all breakpoints
stepi		execute next instruction
stepi [n]	stepi 4	execute next n instructions
nexti		execute next instruction, stepping over function calls
nexti [n]	nexti 4	execute next n instructions, stepping over function calls
where		show where execution halted
disas [addr]	disas _start	disassemble instructions at given address
info registers		dump contents of all registers
print/d [expr]	print/d $ecx	print expression in decimal
print/x [expr]	print/x $ecx	print expression in hex
print/t [expr]	print/t $ecx	print expression in binary
x/NFU [addr]	x/12xw &msg	Examine contents of memory in given format
display [expr]	display $eax	automatically print the expression each time the program is halted
	display/i $eip	print machine instruction each time the program is halted
info display		show list of automatically displays
undisplay [n]	undisplay 1	remove an automatic display

Project 1: Change in Character

Due: Tue 09/16/03, Section 0101 (Chang) & Section 0301 (Macneil)
Wed 09/17/03, Section 0201 (Patel & Bourner)

Objective

This project is a finger-warming exercise to make sure that everyone can compile an assembly language program, run it through the debugger and submit the requisite files using the systems in place for the programming projects.

Assignment

For this project, you must do the following:

1. Write an assembly language program that prompts the user for an input string and a replacement character. The program then replaces all occurrences of the digits 0-9 with the replacement character. A sample run of the program should look like:

   ```
   Input String: Today's date is August 23, 2003.
   Replacement character: X
   Output: Today's date is August XX, XXXX.
   ```

   If the user enters several characters instead of a single replacement character, you can ignore the extra ones and just use the first character entered as the replacement. A good starting point for your project is the program `toupper.asm` (shown in class) which converts lower case characters in the user's input string to upper case. The source code is available on the GL file system at: `/afs/umbc.edu/users/c/h/chang/pub/cs313/`

2. Using the UNIX `script` command, record some sample runs of your program and a debugging session using `gdb`. In this session, you should fully exercise the debugger. You must set several breakpoints, single step through some instructions, use the automatic display function and examine the contents of memory before and after processing. The script command is initiated by typing `script` at the UNIX prompt. This puts you in a new UNIX shell which records every character typed or printed to the screen. You exit from this shell by typing `exit` at the UNIX prompt. A file named `typescript` is placed in the current directory. You must exit from the `script` command *before* submitting your project. Also, remember not to record yourself editing your programs — this makes the `typescript` file very large.

Turning in your program

Use the UNIX `submit` command on the GL system to turn in your project. You should submit two files: 1) the modified assembly language program and 2) the typescript file of your debugging session. The class name for submit is `cs313_0101`, `cs313_0201` or `cs313_0301` depending on which section you attend. The name of the assignment name is `proj1`. The UNIX command to do this should look something like:

```
submit cs313_0101 proj1 change.asm typescript
```

Notes

Additional help on running NASM, `gdb` and making system calls in Linux are available on the assembly language programming web page for this course:

<http://www.csee.umbc.edu/~chang/cs313.f03/assembly.shtml>

Recall that the project policy states that programming assignments must be the result of individual effort. *You are not allowed to work together.* Also, your projects will be graded on five criteria: correctness, design, style, documentation and efficiency. So, it is not sufficient to turn in programs that assemble and run. Assembly language programming can be a messy affair — neatness counts.

Next Time

- Overview of i386 instruction set.
- Arithmetic instructions, logical instructions.
- EFLAGS register