UMBC

CMPE-310

Lab02- Assembly Basics

Outline

- Sample NASM Source Code modified for gcc
- Assembly language basics
- Learn how to debug code
- Debugging Exercises

Hello World modified for gcc

```
;
; Assemble using NASM
;

section .data                   ; section declaration
msg     db  'Hello, world!',0xA ; our string
len     equ $ - msg             ; length of our string

section    .text                ; section declaration
global     main                 ; must be declared for compiler (gcc)

main:                           ; tell compiler entry point

   mov     eax,4                ; system call number (sys_write)
   mov     ebx,1                ; file descriptor (stdout)
   mov     ecx,msg              ; message to write
   mov     edx,len              ; message length
   int     0x80                 ; call kernel

                                ; final exit
   mov     eax,1                ; system call number (sys_exit)
   xor     ebx,ebx              ; sys_exit return status
   int     0x80                 ; call kernel
```

Hello World

Produce hello.o object file:

nasm –f elf hello.asm –l hello.lst

Produce hello ELF executable (gcc):

gcc –m32 hello.o -o hello

Run the program:

./hello

Declaring Initialized Data

Instruction	Operand	Comment
db	0x55	; just the byte 0x55
db	0x55,0x56,0x57	; three bytes in succession
db	‘a',0x55	; character constants are OK
db	‘hello',13,10,'$'	; so are string constants
dw	0x1234	; 0x34 0x12
dw	'a'	; 0x61 0x00 (it's just a number)
dw	'ab'	; 0x61 0x62 (character constant)
dw	'abc'	; 0x61 0x62 0x63 0x00 (string)
dd	0x12345678	; 0x78 0x56 0x34 0x12
dd	1.234567E+20	; floating-point constant
dq	0x123456789abcdef0	; eight byte constant
dq	1.234567E+20	; double-precision float
dt	1.234567E+20	; extended-precision float

Data and Constants

DB, DW, DD, DQ and DT are used for **initialized data**.

```
db 0x55                 ; The byte 0x55
```

RESB, RESW, RESD, RESQ and REST are used for **uninitialized data.**

```
buffer: resb 256        ; Reserve 256 bytes
```

Constants:

Suffixes **H, Q and B** are used for **hex, octal and binary** respectively. **0x also works for hex**.

```
mov eax,0xa2h       ; hex
mov eax,0xa2        ; hex
mov eax,777q        ; octal
mov eax,10010011b   ; binary
mov eax,'abcd'      ; ASCII chars 0x64636261
%define FOO 100     ; Defines numeric and string constants at the top of a file
```

Memory Addressing

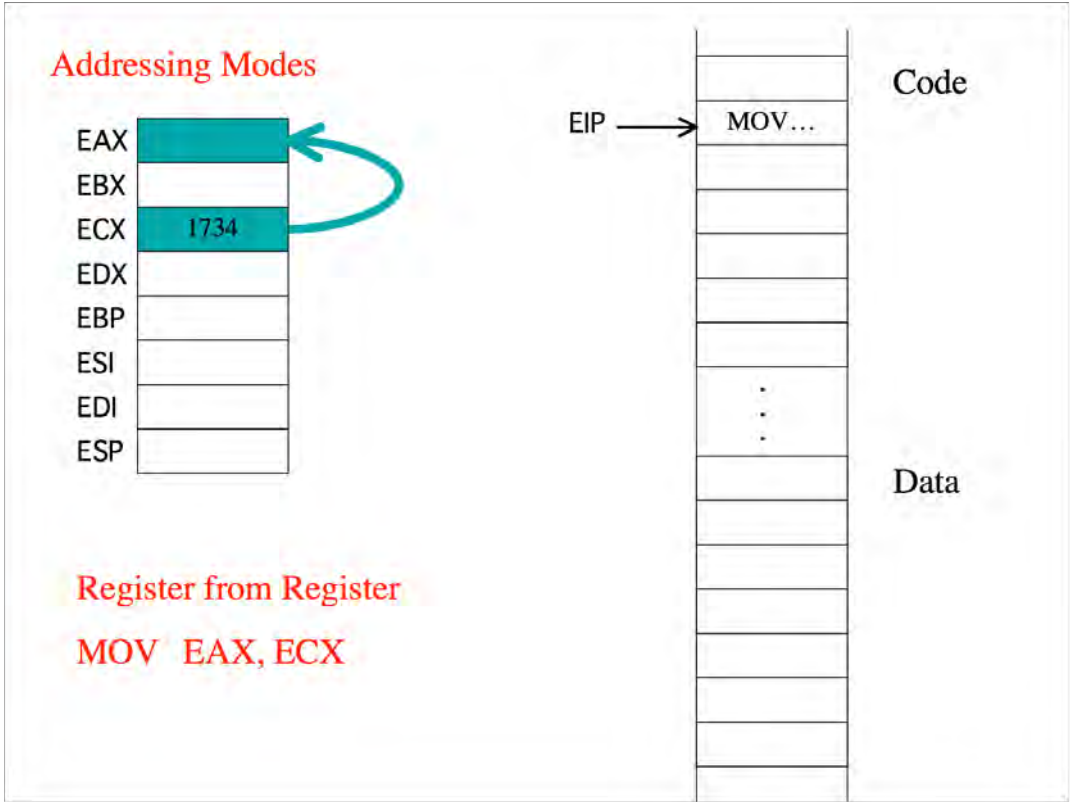
We want to store the value 1734h

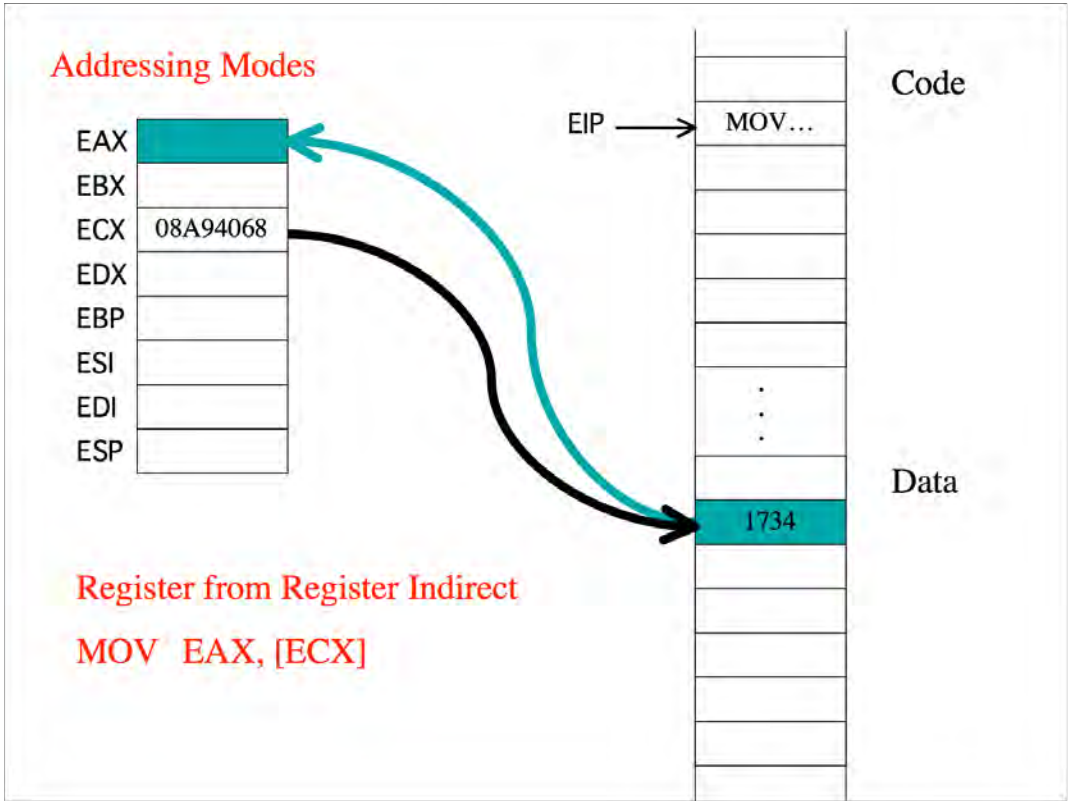
The value 1734h may be located in a register or in memory (or cache)

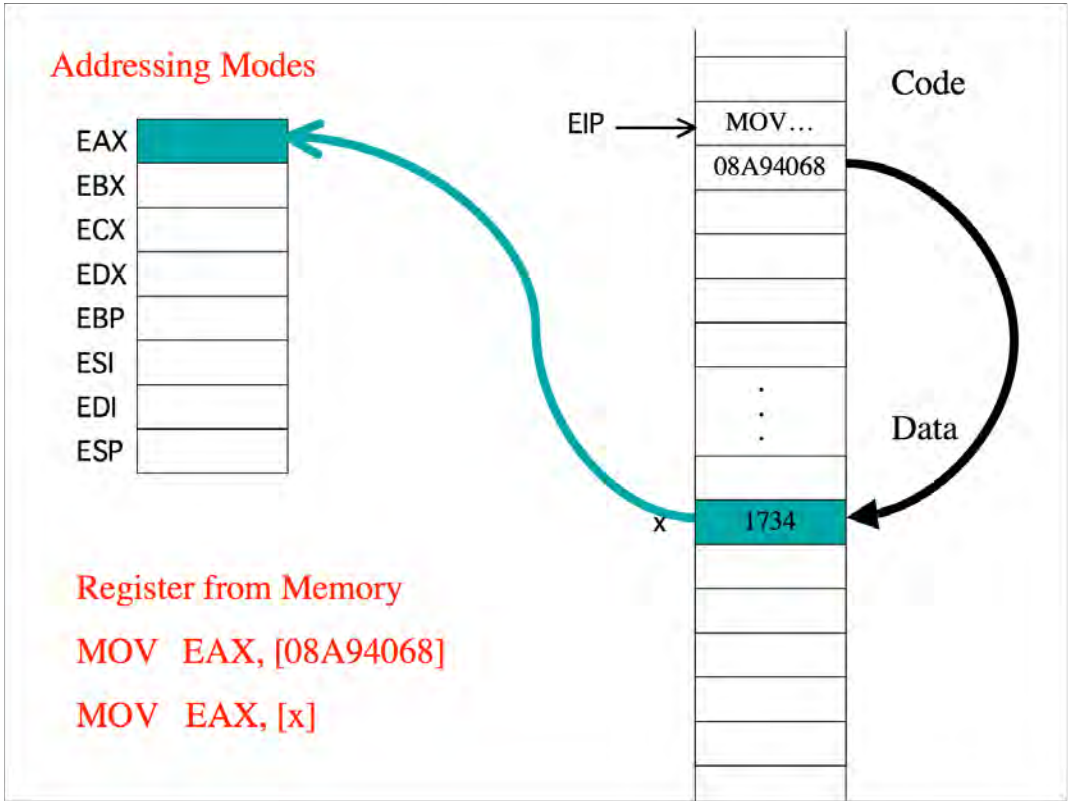
The location in memory might be specified by the code, by a register, ...

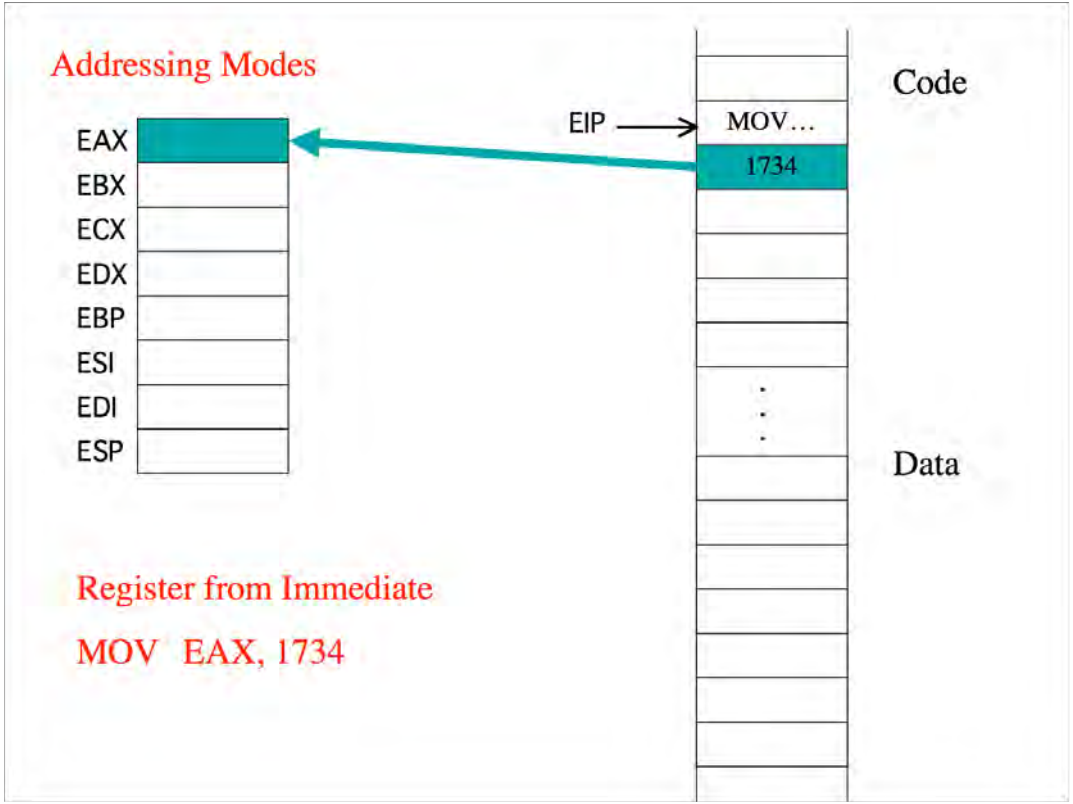
Assembly language syntax for **mov**

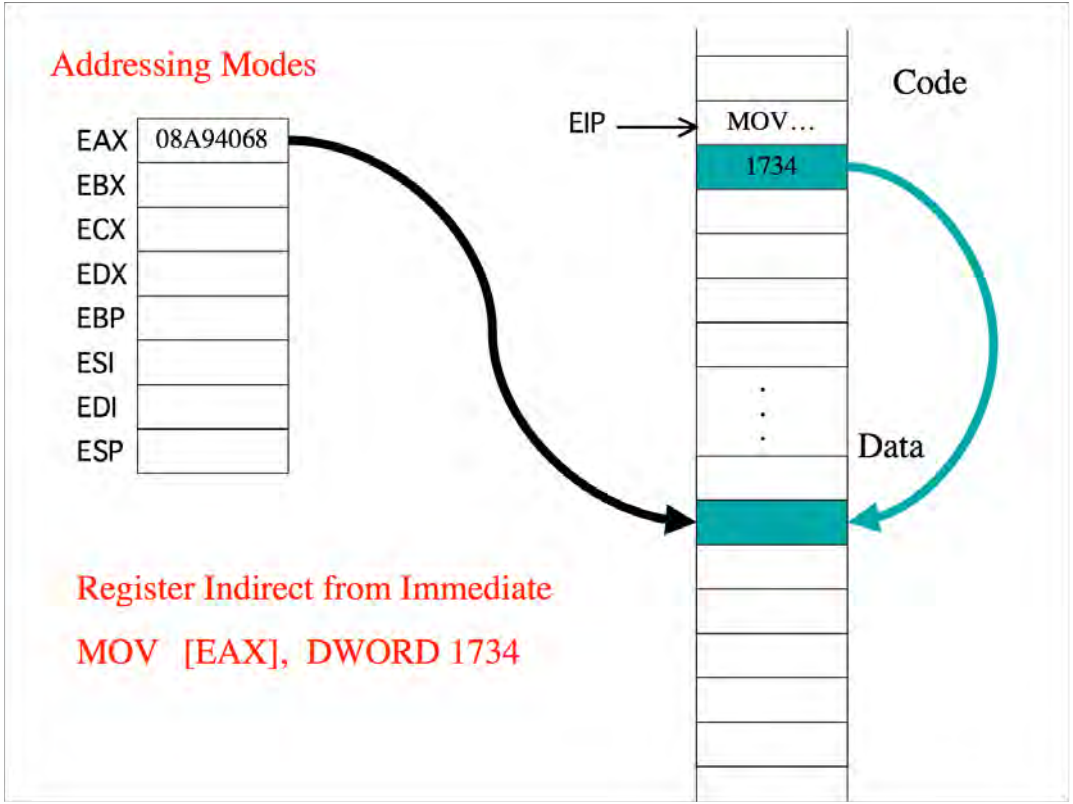
mov destination , source


Addressing Modes
EAX
EBX
ECX
1734
EDX
EBP
ESI
EDI
ESP
Register from Register
MOV EAX, ECX
EIP
MOV…
Code
Data

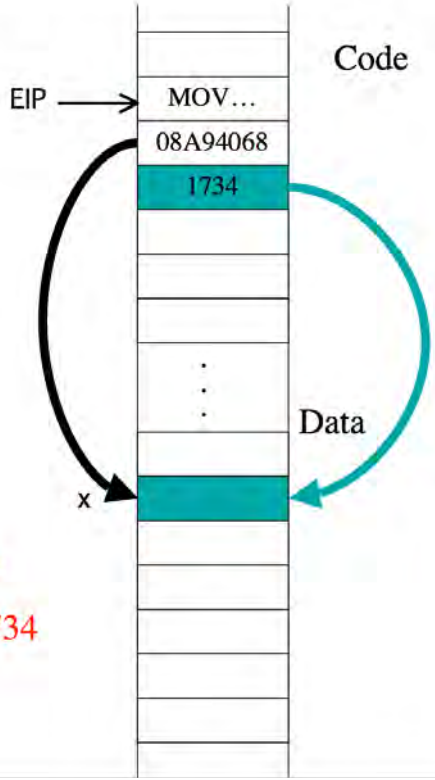

Addressing Modes
EAX
EBX
ECX
EDX
EBP
ESI
EDI
ESP
08A94068
EIP
MOV…
Code
Data
1734
Register from Register Indirect
MOV EAX, [ECX]


Addressing Modes
EAX
EBX
ECX
EDX
EBP
ESI
EDI
ESP
EIP
MOV…
08A94068
Code
Data
x
1734
Register from Memory
MOV EAX, [08A94068]
MOV EAX, [x]


Addressing Modes
EAX
EBX
ECX
EDX
EBP
ESI
EDI
ESP
EIP
MOV…
1734
Code
Data
Register from Immediate
MOV EAX, 1734


Addressing Modes
EAX
08A94068
EBX
ECX
EDX
EBP
ESI
EDI
ESP
EIP
MOV…
1734
Code
Data
Register Indirect from Immediate
MOV [EAX], DWORD 1734

Addressing Modes



Register Indirect from Immediate

MOV [08A94068], DWORD 1734

MOV [x], DWORD 1734

NASM Syntax

In order to **refer to the address of a variable**, leave them out, e.g.,

mov eax, bar –moves address specified by bar (memory) into eax

In order to **refer to the contents of a memory location**, use square brackets.

mov eax, [bar] –moves content of bar into eax

NASM does not understand variable types:

```
data dw 0
mov [data], 2          ; ERROR
mov word [data], 2     ; OK
```

NASM is case sensitive

System Calls

```
mov     eax,4       ; system call number (sys_write)
mov     ebx,1       ; file descriptor (stdout)
mov     ecx,msg     ; message to write
mov     edx,len     ; message length
int     0x80        ; call kernel
```

```
                    ; final exit to
mov eax, 1          ; standard output (screen/console)
xor ebx, ebx        ; first syscall argument: exit code
int 0x80            ; call kernel to take over
```

System calls for 32-bit linux OS – https://syscalls32.paolostivanin.com

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2475
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:935
2	sys_fork	0x02	-	-	-	-	-	kernel/fork.c:2116
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:566
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:581
5	sys_open	0x05	const char __user *filename	int flags	umode_t mode	-	-	fs/fhandle.c:257
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:1153
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1692
8	sys_creat	0x08	const char __user *pathname	umode_t mode	-	-	-	fs/open.c:1115

Command: xor

xor ebx, ebx – clears **ebx** by **xor***ing* **ebx** with itself and storing it back into **ebx**

Debugging Assembly

Cannot just add print statements everywhere (like in higher level languages)

Use gdb to:

- Examine the contents of registers
- Examine contents of memory set breakpoints
- Single-step through program

GDB Commands

Command	Example	Description
run		start program
quit		quit out of gdb
cont		continue execution after a break
break [addr]	break *_start	sets a breakpoint
delete [n]	delete 4	removes the nth breakpoint
delete		removes all breakpoints
info break		lists all breakpoints
list _start		lists a few lines of code around _start
list 7		list 10 lines of code starting around line 7
list 7, 20		list lines 7 thru 20 of the code

GDB Commands

Command	Example	Description
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 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

GDB Commands

Command	Example	Description
x/NFU [addr]	x/12xw &msg	Examine contents of memory in given format
display [expr]	display $eax	Automatically print the expression every time the program is halted
info display		show list of automatic displays
undisplay [n]	undisplay 1	remove an automatic display

Setup hello world for disassembly

Produce hello.o object file:

nasm –g –f elf –F dwarf hello.asm –l hello.lst

Produce hello ELF executable (gcc):

gcc –m32 hello.o -o hello

Debug the program:

gdb –tui hello

Exercise problems

Find the respective source codes under **Lab Material** from course website

1. Assemble and compile ex1.asm for debugging
   Insert a breakpoint at line number 23, and run
   Use info registers to list the contents of all the registers
   Insert a breakpoint at line number 26, and continue
   Print the contents of register EAX as a hexadecimal
2. Assemble and compile ex2.asm for debugging
   Insert a breakpoint at line number 16
   Insert a breakpoint at line number 19, and run
   Add register ECX to the display list as a decimal
   Continue to the next breakpoint once and watch for changes in ECX
   Remove the breakpoint at line number 16, and continue
   Step/Continue through the code and determine the final value in ECX before the program exits

Exercise problems

Find the respective source codes under **Lab Material** from course website

3. Assemble and compile ex3.asm for debugging

   Insert a breakpoint at line number 31, and run
   Examine all the contents of **matrix0** as decimals using a single **x** command
   Examine the value in **rowlen** to verify if it contains the expected value

4. Assemble and compile ex4.asm for debugging

   Insert a breakpoint at line number 31, and run
   Examine all the contents of **text0** as a string using a single **x** command
   Examine the value in **rlen** and **rowlen** to verify if they contain expected values

5. Assemble and compile hello.asm for debugging

   Using the 32-bit system call table, try to figure out why specific values are transferred into EAX, EBX, ECX, and EDX registers