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

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 011000000000000000000000
   b. 1 10000011 000100000000000000000000
   c. 1 1000000 000000000000000000000000
   d. 0 00000001 110100000000000000000000
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

<table>
<thead>
<tr>
<th>Intel Processor</th>
<th>Date Introduced</th>
<th>Micro-architecture</th>
<th>Clock Frequency at Introduction</th>
<th>Transistors per Die</th>
<th>Register Sizes¹</th>
<th>System Bus Bandwidth</th>
<th>Max. Extern. Addr. Space</th>
<th>On-die Caches²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium III processor³</td>
<td>1999</td>
<td>P6</td>
<td>700 MHz</td>
<td>28 M</td>
<td>GP: 32</td>
<td>Up to 1.06 GB/s</td>
<td>64 GB</td>
<td>32KB L1; 256KB L2</td>
</tr>
<tr>
<td>Pentium 4 processor</td>
<td>2000</td>
<td>Intel NetBurst micro-architecture</td>
<td>1.50 GHz</td>
<td>42 M</td>
<td>GP: 32</td>
<td>3.2 GB/s</td>
<td>64 GB</td>
<td>12K µop Execution Trace Cache; 8KB L1; 256KB L2</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8086</td>
<td>1978</td>
<td>8 MHz</td>
<td>29 K</td>
<td>16 GP</td>
<td>16</td>
<td>1 MB</td>
<td>None</td>
</tr>
<tr>
<td>Intel 286</td>
<td>1982</td>
<td>12.5 MHz</td>
<td>134 K</td>
<td>16 GP</td>
<td>16</td>
<td>16 MB</td>
<td>Note 3</td>
</tr>
<tr>
<td>Intel386 DX Processor</td>
<td>1985</td>
<td>20 MHz</td>
<td>275 K</td>
<td>32 GP</td>
<td>32</td>
<td>4 GB</td>
<td>Note 3</td>
</tr>
<tr>
<td>Intel486 DX Processor</td>
<td>1989</td>
<td>25 MHz</td>
<td>1.2 M</td>
<td>32 GP 80 FPU</td>
<td>32</td>
<td>4 GB</td>
<td>L1: 8KB</td>
</tr>
<tr>
<td>Pentium Processor</td>
<td>1993</td>
<td>60 MHz</td>
<td>3.1 M</td>
<td>32 GP 80 FPU</td>
<td>64</td>
<td>4 GB</td>
<td>L1: 16KB</td>
</tr>
<tr>
<td>Pentium Pro Processor</td>
<td>1995</td>
<td>200 MHz</td>
<td>5.5 M</td>
<td>32 GP 80 FPU</td>
<td>64</td>
<td>64 GB</td>
<td>L1: 16KB L2: 256KB or 512KB</td>
</tr>
<tr>
<td>Pentium II Processor</td>
<td>1997</td>
<td>266 MHz</td>
<td>7 M</td>
<td>32 GP 80 FPU 64 MMX</td>
<td>64</td>
<td>64 GB</td>
<td>L1: 32KB L2: 256KB or 512KB</td>
</tr>
<tr>
<td>Pentium III Processor</td>
<td>1999</td>
<td>500 MHz</td>
<td>8.2 M</td>
<td>32 GP 80 FPU 64 MMX 128 XMM</td>
<td>64</td>
<td>64 GB</td>
<td>L1: 32KB L2: 512KB</td>
</tr>
</tbody>
</table>

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.
BASIC EXECUTION ENVIRONMENT

Figure 3-1. IA-32 Basic Execution Environment
BASIC EXECUTION ENVIRONMENT

General-Purpose Registers

<table>
<thead>
<tr>
<th>31</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
<th>16-bit</th>
<th>32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AH</td>
<td>AL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BH</td>
<td>BL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CH</td>
<td>CL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DH</td>
<td>DL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
- **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
Register from Register

MOV EAX, ECX
Register from Register Indirect

MOV EAX, [ECX]
Addressing Modes

Register from Memory

MOV  EAX, [08A94068]
MOV  EAX, [x]
Register from Immediate

```
MOV EAX, 1734
```
### Register Indirect from Immediate

**MOV [EAX], DWORD 1734**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>08A94068</td>
</tr>
<tr>
<td>EBX</td>
<td></td>
</tr>
<tr>
<td>ECX</td>
<td></td>
</tr>
<tr>
<td>EDX</td>
<td></td>
</tr>
<tr>
<td>EBP</td>
<td></td>
</tr>
<tr>
<td>ESI</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td></td>
</tr>
<tr>
<td>ESP</td>
<td></td>
</tr>
</tbody>
</table>
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
  - examine contents of memory
  - set breakpoints
  - single-step through program

• READ THE GDB SUMMARY ONLINE!

UMBC, CMSC313, Richard Chang <chang@umbc.edu>
<table>
<thead>
<tr>
<th>Command</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td></td>
<td>start program</td>
</tr>
<tr>
<td>quit</td>
<td></td>
<td>quit out of gdb</td>
</tr>
<tr>
<td>cont</td>
<td></td>
<td>continue execution after a break</td>
</tr>
<tr>
<td>break [addr]</td>
<td>break *_start+5</td>
<td>sets a breakpoint</td>
</tr>
<tr>
<td>delete [n]</td>
<td>delete 4</td>
<td>removes nth breakpoint</td>
</tr>
<tr>
<td>delete</td>
<td></td>
<td>removes all breakpoints</td>
</tr>
<tr>
<td>info break</td>
<td></td>
<td>lists all breakpoints</td>
</tr>
<tr>
<td>stei</td>
<td></td>
<td>execute next instruction</td>
</tr>
<tr>
<td>stei [n]</td>
<td>stei 4</td>
<td>execute next n instructions</td>
</tr>
<tr>
<td>nexti</td>
<td></td>
<td>execute next instruction, stepping over function calls</td>
</tr>
<tr>
<td>nexti [n]</td>
<td>nexti 4</td>
<td>execute next n instructions, stepping over function calls</td>
</tr>
<tr>
<td>where</td>
<td></td>
<td>show where execution halted</td>
</tr>
<tr>
<td>disas [addr]</td>
<td>disas _start</td>
<td>disassemble instructions at given address</td>
</tr>
<tr>
<td>info registers</td>
<td></td>
<td>dump contents of all registers</td>
</tr>
<tr>
<td>print/d [expr]</td>
<td>print/d $ecx</td>
<td>print expression in decimal</td>
</tr>
<tr>
<td>print/x [expr]</td>
<td>print/x $ecx</td>
<td>print expression in hex</td>
</tr>
<tr>
<td>print/t [expr]</td>
<td>print/t $ecx</td>
<td>print expression in binary</td>
</tr>
<tr>
<td>x/NFU [addr]</td>
<td>x/12xw &amp;msg</td>
<td>Examine contents of memory in given format</td>
</tr>
<tr>
<td>display [expr]</td>
<td>display $eax</td>
<td>automatically print the expression each time the program is halted</td>
</tr>
<tr>
<td>display/i [expr]</td>
<td>display/i $eip</td>
<td>print machine instruction each time the program is halted</td>
</tr>
<tr>
<td>info display</td>
<td></td>
<td>show list of automatically displays</td>
</tr>
<tr>
<td>undisplay [n]</td>
<td>undisplay 1</td>
<td>remove an automatic display</td>
</tr>
</tbody>
</table>
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