Technische Universität München
Lehrstuhl für Rechnertechnik und Rechnerorganisation

Carsten Trinitis

Course IN2075: Microprocessors

Winter Course 16/17

Exercise 1

Overview

In this exercise, we will investigate the C-routine ‘`toupper`’, which changes all lower-case characters of a string into upper-case characters. The goal is to optimise the routine at source code level, first by using different compiler optimisations, and finally by hand-crafted assembler code.
For this, you need to group up in teams of three to four. The results of your teamwork will be presented in the lecture by the group leader.

Which team implements the fastest ‘`toupper`’-routine?

Preparation

For now, open a terminal on a machine running your favourite Linux distribution (or log into it remotely)!
Hint: If you are into Windows but need to use you local PC or laptop you need to install Linux in a Virtual Machine, e.g. VirtualBox).
If you are into Windows and want to log into a remote machine , you can e.g. use the programs ‘putty‘ and ‘winscp’ to log in and copy files to and from the Linux box.

Transfer the file **`ex1.tar.xz`** (available on the web site) to the Linux machine and log in. Then type:

```
tar xJvf ex1.tar.xz
```

Then type:

```
cd uProc1717
make -s PARAMS="-d -l 8000"
```

for a first test (As you can see, ‘`toupper`’ is not working yet).
Hint: You can get information on the system you are working on by typing cat /proc/cpuinfo. Further information will be extracted in subsequent exercises.

Exercise 1.1

Implement the function `toupper_simple` using a simple loop which checks for each character of text whether it is in the range ‘a-z’ and subtracts the value 0x20 from it if necessary.

Test your implementation.

Furthermore, implement the function `gettime`. It should deliver the current system time in a double. You should use the system call `gettimeofday` for implementing this function.

You can now measure the time the function takes to perform the '`toupper`' operation, try:

```
make PARAMS="-d -l 200000 400000 10000"
```

Hint: If your machine is too fast adjust the parameters accordingly (see below for what they mean)!

Exercise 1.2

Implement the function `toupper_optimised` using assembler code or intrinsics and try to make it as fast as possible.
Compare the runtime results to those of the different optimization stages of `toupper_simple`.

Create some slides which document your progress, the measured results and possible reasons for the differences in runtime.
You can vary the length of the character arrays to be processed by adapting the command line argument:

```
make PARAMS="-d -l <size_min> <size_max> <size_step>"
```

Use `gnuplot` to visualise the results. Hint: If `gnuplot` is not installed you need to install it on your PC.

Start `gnuplot.`

For instance, type

```
gnuplot
```

and then

```
plot 'results1' using 2:8 title 'simple' with linespoints, '' using
2:10 title 'optimised' with linespoints
```

within `gnuplot` to create a graph of your measurements.

To create a postscript file of your graph type:

```
set terminal postscript
set output "graph.eps"
plot 'results1' using 2:8 title 'simple' with linespoints, '' using
2:10 title 'optimised' with linespoints
```

If you want to manually change the colours etc. of the graph, try

```
set terminal xfig
set output "graph.fig"
plot 'results1' using 2:8 title 'simple' with linespoints, '' using
2:10 title 'optimised' with linespoints
```

exit `gnuplot`, and type:

```
xfig graph.fig
```

Hint: You might need to install `xfig` as well.

Exercise 1.3

Repeat the steps referring to the C implementations using different compilers:

Intel Compiler:

- The Intel compiler (Intel Parallel Studio XE Cluster Edition) can be downloaded for students for free from:
  https://software.intel.com/en-us/qualify-for-free-software/student

Clang / LLVM:

- Download it from:
  http://llvm.org

Check and compare the relevant assembly code generated by the different compilers!