Contents

1 Summary

1.1 Lectures

- registers
- program counter
- condition codes
- status codes
- processing cycle
- pipelining
- forwarding

- cutting in line
- out-of-order execution

1.2 Labs

1.2.1 Lab 01

- `gcc <file> -o <ouput> -S`: produces an assembly files
- `gcc <file> -o <ouput> -g`: produces a file that is useful for gdb
- `gdb ./<exe> -tui`: start gdb with a graphical view of the code

1.2.2 Lab 02

- `lea <var>(%rip) %<reg>`: load effective address, `<var>(%rip)` 64 bit address of the next instruction, basically loads the memory address of the variable into the specified register
- `xor %eax, %eax`: this sets the return value of a function to 0, `%eax` holds the return values of functions, needs to be set before the function returns

1.2.3 Lab 03

- `mov <var>(%rip) %<reg>`: reads the specified variable from memory and puts it into the register

1.2.4 Lab 04

- `push %rbp`: push the frame pointer of the previous stack frame unto the stack
- `mov %rsp, %rbp`: move the current stack frame address to the frame pointer, %rsp always points to the stop of the stack
- `leave`: undoes the two previous steps
- `mov %rbp, %rsp`, `pop %rbp`: does the same thing as leave

1.2.5 Lab 05

- `sub $0x8, %rsp`: this reserves some space on the stack for local variables to call the functions
- `add $0x8, %rsp`: frees the space that was previously allocated
- `call scanf@plt`: procedural linkage table, contains the address of where scanf is relative to the program, makes function reuse easier

1.2.6 Lab 06

- `call <func>`, `ret`: call pushes the return address onto the stack, return pops it off again to return to where the function was entered

- `cltq`: convert long to quad, basically a cast from `int` to `long` in `C`

1.2.7 Lab 07

- `jmp`: jumps to the label specified, can be used with conditions
- `cmp`: compares two registers, tells if equal, smaller or larger, can be used to condition jump instructions
- different from `call`, this does not push or pop addresses, it just jumps to different parts of the code
- `test` vs `cmp`: `test` is a bitwise `and` while `cmp` is an arithmetic operation
  `test <reg>, <reg> == cmp <reg>, 0`

1.3 TODO

- `.global` labels
- section of assembly code (`.section`)
- order of registers for arguments to functions
- multi-register operations
- `int x 0, 0`
- position independent code
- got (global offset table)
- syscall vs call
- call functions
- jumps
- loops using labels

Chapter 1.1 to 1.10

Chapter 4.1 to 4.3

2 Lecture 27.01.2020

Class was cancelled

3 Lab 29.01.2020

- logging into the server or setting up the working environment
- we'll work with assembly files and then go ahead with stuff
- well do some linking and just optimizing a bit of asm
- `@plt` is some table that allows you to call functions from outside of your program
- topic is `position independent code`, look that up
- `plt` table has the locations of all the functions that you might want to call from you program
- `got` global offset table works with `plt` to make it happen

- `xor %eax, %eax` can also be used instead of `mov $0, %rax`, xor with itselg sets all the bits to zero in `%eax`
- `%eax` is half of `%rax`, meaning that we can set `%rax` to zero by calling `xor` on `%eax`
- well use `syscall` in lab 3 to do some stuff
- look at the `syscall` docs linked in lab 3
- number 03 will not be on the exam

4 Lecture 03.02.2020

- we are going to try to link the labs and the lectures together
- we're going to chapter 4 and x86-64 architecture
- learning an ISA
  - if you know how the processor works helps you understand how the whole computer works
  - understanding how CPUs work can help you write better code as well
  - helps one make decisions on hardware design
  - maybe some of us will work on actual CPU design
- registers are used as super fast short term storage
- program counter keeps track of the instructions that are being executed at the moment
- condition code
- status code indicates the overall state of the programs execution
- Y86 has immediate to memory, register to memory, memory to register, register to register moves
- logic gates are the basic components of a CPU and a PC in general, how they work is not to complicated at the basics, but it gets super complex if you have billions of them

`hello_world.c`

```c
#include <stdio.h>

int main() {
    puts("Hello, World!\n");
    return 0;
}
```

`hello_world.asm`

```
main:
    subq    $8,     %rsp
    movl    $.LC0, %edi
    call    puts
    movl    $0,     %eax
    addq    $8,     %rsp
    ret
```

`sum.c`

```
long sum(long *start, long count) {
    long sum = 0;
    while (count) {
        sum += *start;
        start++;
        count--;
    }
    return sum;
}
```

sum.asm

```
sum:
    movl    $0,     %eax
    jmp     .L2
.L2:
    addq    (%rdi), %rax
    addq    $8,     %rdi
    subq    $1,     %rsi
.L3:
    testq   %rsi,   %rsi
    jne     .L3
    rep; ret
```

5 Lecture 10.02.2020

- every processing cycle does
  - *fetch:*
    - many modern CPUs fetch hundrets of instructions in one go and then runs through all them, they are saved in L1 cache
    -
  - *decode:*
  - *execute:*
  - *memory:*
  - *write back:*
- *fetch* and *write back* can be combined into one thing
- each cycle the `PC` (program counter) is incremented
- pipelining can be somewhat compared to a car factory – you perform the first step of the first instruction moving on to the second step, then you perform the first step of the second instruction and so on
- this can be very efficient because there is little time being wasted
- *forwarding, pipelining, cutting in line, and the other techniques*

6 Lab 12.02.2020

- you go to labels for conditionals and simple functions: `<name>:`

- when you call the function by `call <name>` it jumps there and executes the code
- two types of jumps: conditional (depends on some condition), unconditional (it jumps in any case)
- why does the lab not work?

7 Lecture 17.02.2020

- we will review stuff before the midterm
- midterm will be on chapters 1, 2, 3, and a little bit of 4
- bits and bytes will be the topic for today
- `0x01234567` stored as
  - `01 | 23 | 45 | 67` in big endian
  - `67 | 45 | 23 | 01` in little endian
- three types of notations
  - unsigned notation – standard binary or hex encoding
  - signed notation (two's complement) – inverting digits and adding one to represent negative numbers
  - floating point – mathematical/scientific notation of numbers with rational number and exponent
- types might not be the same length of bits between different machines and operating systems – super important to keep that in mind when dealing with this kind of stuff
- we can do bit shifting `x >> 4` to the right or `x << 4` to the left, for example `0101 0111 >> 4 = 0000 0101`
- arithmetic shifting `10010101 >> 4 = 11111001` uses the digit in the left most place to fill the new places
- bit masking using `&` like `0x25AF3255 & 0x00FF0000 = 0x00AF0000`
- remember little and big endian – big endian means that the most significant bit comes first, little endian means that the least significant bit is first
- `&` logical and, `|` logical or, `^` logical xor, `~` logical not
- **read Chapter 2**
- overflow is a thing as well – two positive numbers added together could yield a negative result for example – we need to pay attention to that

8 Lab 19.02.2020

- doing a fibonacci loop
- writing a recursive function for gce
- values for multiplication should be in `%rax` and another register, result is in `%rdx` and `rax`
- division: `%rax` and another register, `%rax` has full result, `%rdx` has remainder
- *we can skip the next classes if we don't have any questions*
- look at chapter 3 of the book for memory allocation, recursive procedures etc.
- look at inline assembly

9 Lecture 11.03.2020

- create an array and put random variables inside of it
- these are tasks 13 and 14 of lab01
- now we're supposed to write a program that contains an array of random numbers

10 Lecture 30.03.2020

- professor is late, how else could it be
- he hates online teaching, but we'll have to make due
- the theoretical midterm will be online through e-course
- **Midterm will be next week**
- practice midterm is already up – we can try it out – midterm will be those questions with different numbers etc.
- now we can actually do some programming in the lectures too – he finally got the idea
- for the midterm we may have video review lectures
- maybe he will just make videos instead of actual lectures
- take the practice exam for testing and give feedback
- midterm will be announced next Sunday or Monday and then a couple of days of time
- we'll so some actual C to assembly compiling, real and practical examples of assembly code

11 Lab 08.04.2020

- the project will be the same as the final defense
- ask him to post the templates on e-course or github because I never use the server and never have used it
- template1.s is a pseudorandom number generator
- just do all the stuff that piazza tells us to do
- the next classes will probably be cancelled – next week and so on
- the project will be done by us ourselves and there is nothing really left for him to teach us
- all the sources and references are on the piazza page
- SIMD: only use AVX and the XMM- registers because that is all the server supports
- Bektur has a fork of Toksaitov's code that we can look at and then use
- maybe optimize the code to recognize which type of SIMD is supported to really speed it up
- we have the references, we just need to find the function we need
- just do all the tasks – put them into a big make file and run with it
- we can allocate space using C++ if we want
- for the next 4 weeks there will be no lab classes
- we have to convert ints to floats too at some point
- when we are done just contact him and set a date for the check

12 Lecture 13.04.2020

- today we will just talk about the preliminary midterm
- he doesn't really know what he is doing when it comes to C and its functions and data types
- he really has no clue what the fuck he is doing
- he wants us to double check our answers with an online calculator?
- he posted videos to the wrong page?

13 Lab 06.05.2020

1. How is the final exam going to work?
   We will not have to write any code, just to present our project and answer his questions about it – some people may be asked to write some code if he questions our knowledge, and even then just bits, like adding two floating points or multiplication using SIMD
2. The registration form will be put online on Friday, exam will be next week Wednesday, from 9am to the end of the day. We'll get about 20 to 30 minutes, 2 students at a time