Contents

Chapter 1.1 to 1.10

Chapter 4.1 to 4.3

1 Lecture 27.01.2020

Class was cancelled

2 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

3 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`

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

`sum.c`

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

`sum.asm`

```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
```

4 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*

5 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?