Architektura komputerów 2

laboratorium

Informacje organizacyjne

Semestr letni 2017/2018 r.
Aleksandra.Postawka@pwr.edu.pl

Zakład Architektury Komputerów
http://zak.ict.pwr.wroc.pl/

Organizacja laboratorium

- serwer **lak.iiar.pwr.wroc.pl** (w serwerowni)
- Ubuntu 14.04 LTS w wersji 64-bitowej
  - konta lokalne: login **student** / **lstudent** hasło **pwr-stud-013** (lub **stud013**)
  - konta zdalne (*indywidualne*):

    login **sXXXXXX** hasło **indywidualne** (XXXXXX – nr indeksu)
- usługa SSH działa na porcie **22**, alternatywnie na **2222**

Informacje i materiały

- konsultacje: *PN 13:00 – 14:30, WT 14:00 – 15:30*
- C-3, 220
- materiały:
- https://1drv.ms/f/s!Ai2JqCAKqcOFgahwif_9hjRkExbRlA
- http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium%20AK2/
  -> Linux-asm-lab-2015.pdf
  - podstawy

- -> wzorzec sprawozdania'15.pdf
  - format oddawanych sprawozdań

Zasady zaliczania

- obecność obowiązkowa
  - dopuszczalna jedna **usprawiedliwiona** nieobecność
  - konieczność odrobienia na innych zajęciach
  - 7 terminów zajęć + 1 termin odróbkowy

- punkty za lab 0 – 5, ocena końcowa jest **średnią**
  - **obowiązkowe przygotowanie do zajęć**
  - przykładowe programy na OneDrive
  - ocena wystawiana na zajęciach na podstawie programów napisanych w trakcie laboratorium i **rozmowy** (5pkt)
  - kończenie zadań w domu (w terminie do następnych zajęć):
    - 75% bazowej liczby punktów + szczegółowa odpowiedź
    - oddanie zadań w jeszcze późniejszym terminie to 50% pkt + odpowiedź

Zasady zaliczania c.d.

- **sprawozdania** +/- 0.5 oceny
  - sprawozdania – według wzoru, błędy ortograficzne obniżają ocenę
  - w sprawozdaniu ważne fragmenty kodu, **wraz z opisem** (nie chodzi tu o komentarze!)
  - powinny zawierać nową teorię, która była niezbędna do wykonania zadań z laboratorium (nie dublujemy informacji z poprzednich sprawozdań)
  - wysyłane na pocztę e-mail z tematem np. „**sprawozdanie [czw7_TN]**” w formacie PDF
  - termin – 2 tygodnie od ostatnich zajęć

- Algorytm doliczania sprawozdań do oceny końcowej:
  - bardzo dobre sprawozdanie +0.1
  - sprawozdanie akceptowalne +0.0
  - lakoniczne sprawozdanie -0.1
  - brak sprawozdania -0.2
  - pod koniec semestru punkty są sumowane i dodawane do średniej ocen (przy czym max +0.5 i max -0.5)
  - np. średnia ocen to 3.15, korekta dla sprawozdań +0.3 -> 3.45 -> 3.5 do indeksu

Zasady zaliczania c.d.

- **plagiat** powoduje automatyczne niezaliczenie kursu
  - przywłaszczenie fragmentów innych sprawozdań
  - oddanie kodu, którego fragmentu „autor” nie rozumie, nie potrafi wyjaśnić
  - „Pisałem/łam z kolegą/koleżanką”
- **niezapowiedziane kartkówki** +/- 1 pkt
- termin wystawiania ocen: **ostatnie zajęcia, najpóźniej 12.06**

Zasady BHP


PAMIĘTAJ
ŻE URODZIŁEŚ SIĘ
BEZ CZĘŚCI
ZAPASOWYCH
bebzol.com
Zasady BHP
nie ma to jak dobra motywacja

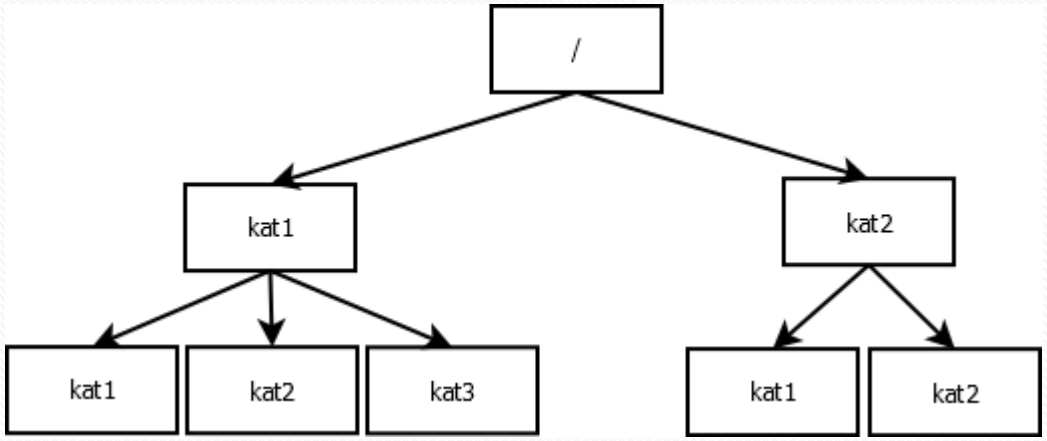
Lab 0 (wprowadzenie)

- poruszanie się po systemie – komendy: **man**, **cd**, **ls**, **rm**, **cat**…
- podstawy składni programów dla gnu asembler
- kompilacja, konsolidacja: **as**, **ld**, **gcc**
- funkcje standardowe – syscall
- zdalne logowanie do systemów **unix**opodobnych (SSH, putty, winscp, …)
- pierwszy program „Hello, world!”
- modyfikacje programu
- debugger

Wprowadzenie do zajęć

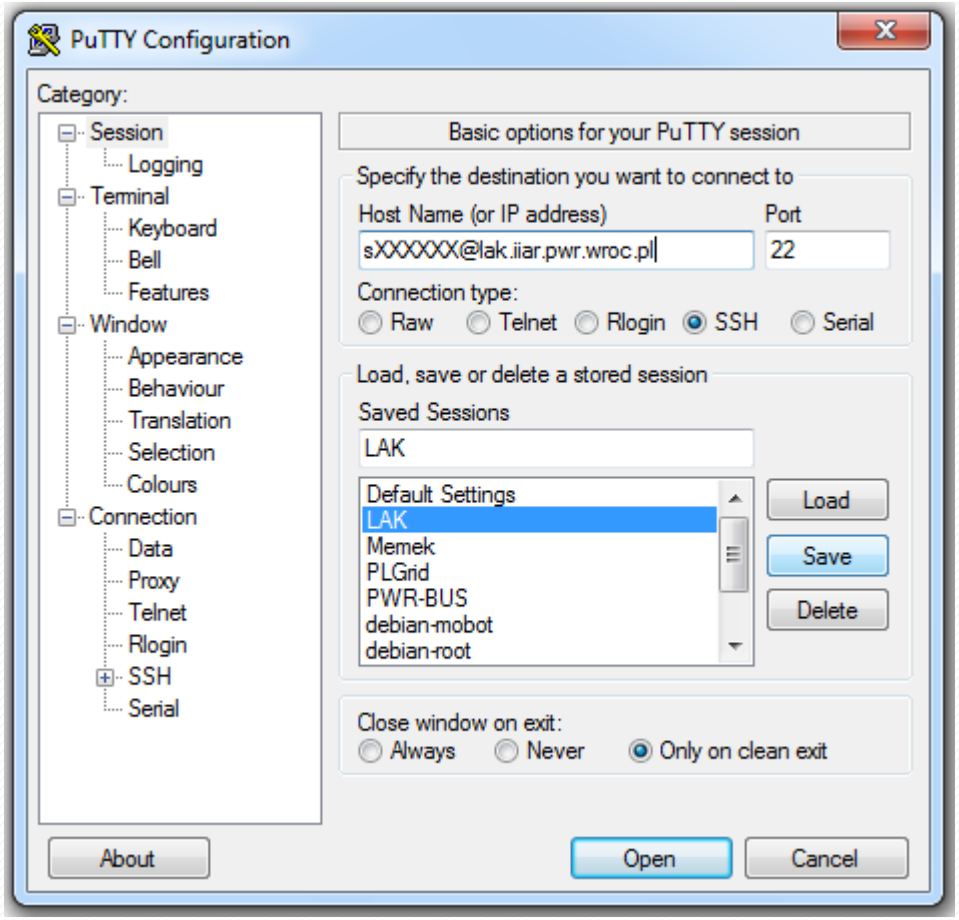
- `ssh sXXXXXX@lak.iiar.pwr.wroc.pl`
- tworzenie katalogu: `mkdir kat1`
- poruszanie się po systemie: `cd kat1`, `cd ../kat2`, `cd ~`
- ścieżki względne, bezwzględne
- tworzenie pliku: `touch plik1`
- edytory: vim, mcedit, nano, … , gedit
- komendy `pwd`, `whoami`, `cat plik`, `ls [dir]`, `mv a b`
- manual: `man ls` [wyjście: q]
- usuwanie pliku: `rm plik`, `rm -R katalog`
- kopiowanie pliku: `cp kat1/plik1 kat2/`, `cp -R …`

Ścieżki względne i bezwzględne

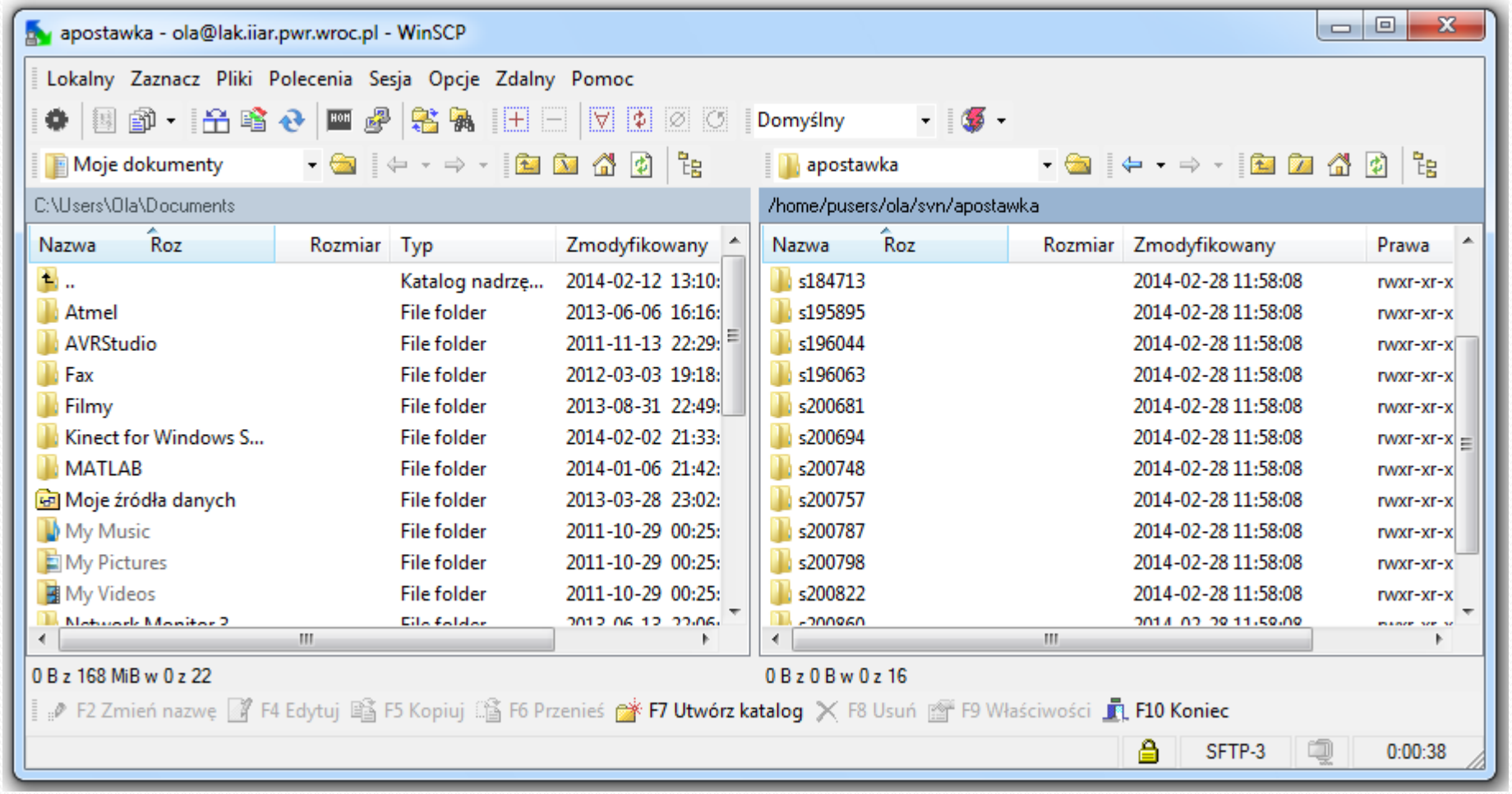


- `cp  /kat1/kat1/plik  /kat2/kat1`
- `cp  kat2/pliczek  kat3`

Szybkie logowanie z Windowsa



Szybkie logowanie z Windowsa c.d.



Pliki Makefile

- zacznijmy od „Hello, world!” w C

Pliki Makefile

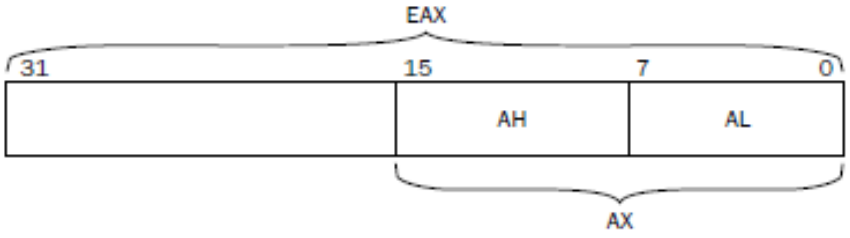
- zacznijmy od „Hello, world!” w C
- i utwórzmy plik o nazwie Makefile

```
plik:   [/tab]plik.o
        [/tab]gcc -o plik plik.o

plik.o:[/tab]plik.c
        [/tab]gcc -c plik.c
```

- polecenie `make`
- uruchomienie programu po kompilacji: `./plik`

Rejestry



General-Purpose Registers

31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

Table 3-2. Addressable General Purpose Registers

Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8L - R15L
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

[2]

Mnemonik + sufiks (mov, add, sub, …)

- `b` dla liczb 8-bitowych
- `w` dla liczb 16-bitowych
- `l` dla liczb 32-bitowych
- `q` dla liczb 64-bitowych

- `movb $7, %al`
- `movw %bx, %ax`
- `movl %ebx, %eax`
- `movq $123, %rcx`

Sekcje programu

```
.section.data

       <  initialized data here>

.section .bss

       < uninitialized data here>

.section .text
.globl _start
_start:

    <instruction code goes here>
```

[1]

DATA

```
tekst:
.ascii "Architektura komputerow <3\n"
```

Directive	Data Type
.ascii	Text string
.asciz	Null-terminated text string
.byte	Byte value
.double	Double-precision floating-point number
.float	Single-precision floating-point number
.int	32-bit integer number
.long	32-bit integer number (same as .int)
.octa	16-byte integer number
.quad	8-byte integer number
.short	16-bit integer number
.single	Single-precision floating-point number (same as .float)

[1]

BSS

```
.comm buffer_name, 512
```

Directive	Description
.comm	Declares a common memory area for data that is not initialized
.lcomm	Declares a local common memory area for data that is not initialized

[1]

System calls – wywołania 32b

- inicjowane przez `int $0x80`
- numery funkcji w `/usr/include/asm/unistd_32.h`
- wynik w `eax`

rejestr	opis
EAX	numer funkcji systemowej
EBX	pierwszy parametr
ECX	drugi parametr
EDX	trzeci parametr
ESI	czwarty parametr
EDI	piąty parametr

```
#define __NR_restart_syscall      0
#define __NR_exit                 1
#define __NR_fork                 2
#define __NR_read                 3
#define __NR_write                4
#define __NR_open                 5
#define __NR_close                6
#define __NR_waitpid              7
#define __NR_creat                8
#define __NR_link                 9
#define __NR_unlink              10
#define __NR_execve              11
#define __NR_chdir               12
#define __NR_time                13
```

System calls – przykład 32b

```
ssize_t write(int fd, const void *buf, size_t count);
```

EAX **EBX** **ECX** **EDX**

```
.data
SYSEXIT = 1
SYSREAD = 3
SYSWRITE = 4
STDOUT = 1
STDIN = 0
EXIT_SUCCESS = 0

buf: .ascii "Hello, world!\n"
buf_len =  .-buf

.text
.globl _start

_start:

movl $SYSWRITE, %eax
movl $STDOUT, %ebx
movl $buf, %ecx
movl $buf_len, %edx
int $0x80

movl $SYSEXIT, %eax
movl $EXIT_SUCCESS, %ebx
int $0x80
```

- 0 (STDIN):
- 1 (STDOUT):
- 2 (STDERR):

System calls – wywołania 64b

- inicjowane przez `syscall`
- numery funkcji w `/usr/include/asm/unistd_64.h`
- wynik w `rax`

rejestr	opis
RAX	numer funkcji systemowej
RDI	pierwszy parametr
RSI	drugi parametr
RDX	trzeci parametr
R10	czwarty parametr
R8	piąty parametr

```
#define __NR_read                                0
__SYSCALL(__NR_read, sys_read)
#define __NR_write                               1
__SYSCALL(__NR_write, sys_write)
#define __NR_open                                2
__SYSCALL(__NR_open, sys_open)
#define __NR_close                               3
__SYSCALL(__NR_close, sys_close)
#define __NR_stat                                4
__SYSCALL(__NR_stat, sys_newstat)
#define __NR_fstat                               5
__SYSCALL(__NR_fstat, sys_newfstat)
#define __NR_lstat                               6
__SYSCALL(__NR_lstat, sys_newlstat)
#define __NR_poll                                7
__SYSCALL(__NR_poll, sys_poll)
```

System calls – przykład 64b

```
ssize_t write(int fd, const void *buf, size_t count);
```

RAX **RDI** **RSI** **RDX**

```
.data
SYSREAD = 0
SYSWRITE = 1
SYSEXIT = 60
STDOUT = 1
STDIN = 0
EXIT_SUCCESS = 0


buf: .ascii "Hello, world!\n"
buf_len =  .-buf

.text
.globl _start

_start:

movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $buf, %rsi
movq $buf_len, %rdx
syscall

movq $SYSEXIT, %rax
movq $EXIT_SUCCESS, %rdi
syscall
```

- 0 (STDIN):
- 1 (STDOUT):
- 2 (STDERR):

zad0.s

Pliki Makefile c.d.

```
zad0:[\tab]    zad0.o
     [\tab]    ld -o zad0 zad0.o

zad0.o:[\tab] zad0.s
       [\tab] as -o zad0.o zad0.s
```

Program wczytaj - wypisz

- utwórz bufor wejściowy
- wczytaj dane ze standardowego wejścia (klawiatura) do bufora
- wypisz dane z bufora na standardowe wyjście (ekran)

Program wczytaj - wypisz

```
.data
STDIN = 0
STDOUT = 1
SYSWRITE = 1
SYSREAD = 0
SYSEXIT = 60
EXIT_SUCCESS = 0
BUFLEN = 512

.bss
.comm textin, 512
.comm textout, 512

.text
.globl _start
_start:
movq $SYSREAD, %rax
movq $STDIN, %rdi
movq $textin, %rsi
movq $BUFLEN, %rdx
syscall


movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $textin, %rsi
movq $BUFLEN, %rdx
syscall


movq $SYSEXIT, %rax
movq $EXIT_SUCCESS, %rdi
syscall
```

Modyfikacja poprzedniego programu

- dodaj bufor, w którym będą zapisywane dane wyjściowe
- utwórz prostą pętlę, w której:
  - odczytasz kolejne litery z bufora wejściowego
  - zamienisz wielkie litery na małe, a małe na wielkie
  - zapiszesz zmienione litery do bufora wyjściowego
- wypisz dane z bufora wyjściowego

```
offset(%base, %index, multiplier)
base + index * multiplier + offset
```

```
.data
STDIN = 0
STDOUT = 1
SYSWRITE = 1
SYSREAD = 0
SYSEXIT = 60
EXIT_SUCCESS = 0
BUFLEN = 512

.bss
.comm textin, 512
.comm textout, 512

.text
.globl _start
_start:
movq $SYSREAD, %rax
movq $STDIN, %rdi
movq $textin, %rsi
movq $BUFLEN, %rdx
syscall

dec %rax          #'\n'
movl $0, %rdi     #licznik
```

offset(%base, %index, multiplier)
base + index * multiplier + offset

```
zamien_wielkosc_liter:
movb textin(, %rdi, 1), %bh
movb $0x20, %bl
xor %bh, %bl
movb %bl, textout(,%rdi, 1)
inc %rdi
cmp %rax, %rdi
jl zamien_wielkosc_liter

movb $'\n', textout(, %rdi, 1)

movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $textout, %rsi
movq $BUFLEN, %rdx
syscall

movq $SYSEXIT, %rax
movq $EXIT_SUCCESS, %rdi
syscall
```

gdb – debugger (1)

- kompilacja: `gcc -g -o plik plik.s`
- zamiana `.globl` na `.global`
- zamiana `_start` na `main`

```
gdb plik
break main
run
next
step
print $rax
quit
```

```
gdb plik
b main
r
n
s
p $rax
q
```

gdb – debugger (2)

- kompilacja `as -gstabs -o plik.o plik.s`
  `ld -o plik plik.o`

- uwaga! NIE DZIAŁA

```
.globl _start
_start:
mov ...
```

```
gdb plik
b *_start
r
```

- konieczne dodanie instrukcji `NOP`

```
.globl _start
_start:
nop
mov ...
```

```
gdb plik
b *_start+1
r
```

gdb – wyświetlanie danych

- `info registers`

- `print`
  - `print/d` – wydrukuj dziesiętnie
  - `print/x` – wydrukuj szesnastkowo
  - `print/t` – wydrukuj binarnie

```
print/x $rax
p/x $rax
```

- `x/nsf`
  - gdzie n[umber] to liczba pól
  - gdzie s[ize]: b (byte), h (16b), w (32b)
  - gdzie f[ormat]: c (character), d (decimal), x (hexadecimal)

```
x/50bc &tab
```

Co na następne zajęcia?

- instrukcje arytmetyczne (add, adc, sub, sbb, mul, div)
- instrukcje logiczne (not, or, and, xor, shl, shr, …)
- instrukcja porównania (cmp) i skoków warunkowych (jl, jg, jb, ja, je, jne, jge, jle, jz, jnz, …)
- zaznajomić się z dokumentacją
- opanować gdb

Źródła obrazków

[1] R. Blum „Professional Assembly Language”

[2] Intel 64 and IA-32 Architectures Software Developer’s Manual