



Pemrograman Dasar Bahasa Assembly x86 nasm di Linux

by /0xTAUFAN

taufanlinux@gmail.com

Blog: www.taufanlubis.wordpress.com

Contents

- ✓ Introduction
- ✓ Participant
- ✓ Tools (nasm,gdb,editor)
- ✓ Program exit/helloworld
- ✓ CPU x86
- ✓ Memory Lay out di linux
- ✓ Virtual Memory
- ✓ System Calls
- ✓ Constants
- ✓ Variables
- ✓ Arithmetic Instructions
- ✓ Logical Instructions
- ✓ Conditions
- ✓ Loops
- ✓ Numbers

Introduction

- Bahasa Assembly ada bahasa pemrograman tingkat rendah (low-level programming language) sesuai dengan spesifikasi arsitektur komputer atau peralatan tertentu yang dapat di program.
- Bahasa Assembly di konversi menjadi bahasa mesin dengan program yang biasa disebut 'assembler'. (Nasm, Masm, Gnu assembler dll).

Peserta

- Tutorial ini di rancang untuk mereka yang ingin mempelajari dasar bahasa assembly dari nol tapi cukup sebagai dasar untuk ke level yang lebih tinggi.

Prasyarat peserta

- Mengerti apa yang dimaksud dengan bahasa pemrograman komputer
- Lebih baik jika menguasai bahasa pemrograman tertentu.
- Menggunakan OS Linux
- Ter-install: **nasm** (netwide assembler), **gdb** (gnu debugger) dan **vim/nano**.

Aplikasi yang dibutuhkan

- Nasm (netwide assembler) → program assembler untuk arsitektur intel x86 dan merupakan assembler yang paling banyak digunakan di linux. Dibuat oleh Simon Tatham (creator putty)
 - **\$ *sudo apt-get install nasm***
- gdb (gnu debugger) → program untuk melihat apa yang terjadi didalam program lain saat dijalankan.
 - **\$ *sudo apt-get install gdb***
- Gcc (gnu compiler collection) → standard compiler bahasa c di linux.
 - **\$ *sudo apt-get install gcc***

Mengapa NASM?

- Netwide assembler, NASM, adalah assembler x80-86 dan x80-64 yang dirancang untuk portability dan modularity.
- Support berbagai .obj format.
- Bisa langsung menghasilkan format binary (khusus dos).
- Baris perintah dirancang simple dan mudah di pahami.
- Mendukung semua arsitektur x86 dan dukungan kuat di MACROS.

Bagaimana Assembler yang lain?

- **a86** → bagus, tetapi tidak bisa 32bit dan hanya di DOS.
- **gas** → free, bisa DOS dan Linux, tetapi tidak terlalu bagus karena dirancang sebagai backend gcc. Syntax-nya rumit.
- **as86** → khusus minix dan linux, tetapi kurang dokumentasi.
- **MASM** → bagus, tetapi mahal. Dan hanya di DOS.
- **TASM** → Lebih baik, tetapi mahal. Dan hanya di DOS.

Compile/Execute nasm

\$ nasm -hf --display object format.

bin	flat-form binary files (e.g. DOS .COM, .SYS)
ith	Intel hex
srec	Motorola S-records
aout	Linux a.out object files
aoutb	NetBSD/FreeBSD a.out object files
coff	COFF (i386) object files (e.g. DJGPP for DOS)
elf32	ELF32 (i386) object files (e.g. Linux)
elf64	ELF64 (x86_64) object files (e.g. Linux)
as86	Linux as86 (bin86 version 0.3) object files
obj	MS-DOS 16-bit/32-bit OMF object files
win32	Microsoft Win32 (i386) object files
win64	Microsoft Win64 (x86-64) object files
rdf	Relocatable Dynamic Object File Format v2.0
ieee	IEEE-695 (LADsoft variant) object file format
macho32	NeXTstep/OpenStep/Rhapsody/Darwin/MacOS X (i386) object files
macho64	NeXTstep/OpenStep/Rhapsody/Darwin/MacOS X (x86_64) object files
dbg	Trace of all info passed to output stage
elf	ELF (short name for ELF32)
macho	MACHO (short name for MACHO32)
win	WIN (short name for WIN32)

NASM Command Line

- **nasm -f <format> <filename> [-o <output>]**
 - **nasm -f elf myfile.asm**
 - **nasm -f bin myfile.asm -o myfile.com** → hanya di DOS
 - **nasm -f elf myfile.asm -l myfile.lst** → kode mesin
- **nasm -h** → help
- **nasm -hf** → tampilkan format
- **\$ file /usr/bin/nasm** → Untuk pemakai linux jika tidak yakin tipe file yang dipakai.
 - /usr/bin/nasm: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped

ELF

- ELF (**executable and linkable format object file**) adalah format yang digunakan di Linux juga di Unix System V, Solaris x86, UnixWare, SCO unix

Intel x86

- Materi yang dibahas hanya tentang pemrograman di **Intel 32 bit x86** CPU atau CPU 32bit yang kompatibel dengan intel.
- CPU yang ada termasuk 8086, 8088, 80286, 80386, 80486, Pentium, Pentium Pro, Pentium MMX, Pentium II, Pentium III, pentium 4, Pentium Xeon, Pentium II Xeon, Pentium Core, Celeron, dll.

Beda "nasm" dan "as"

- Gnu Assembler → gas → 'as' adalah program assembler yang digunakan di gnu project dan merupakan default back-end dari gcc.
- AS memakai format **AT&T**, Nasm format **INTEL**.

```
darklinux@darklinux: ~  
(gdb) list 1,10  
1      section .text  
2  
3      global _start  
4      _start:  
5  
6          mov eax,1  
7          mov ebx,2  
8          int 0x80  
(gdb)
```

```
darklinux@darklinux: ~  
(gdb) disassemble _start  
Dump of assembler code for function _start:  
0x08048060 <+0>:    mov     $0x1,%eax  
0x08048065 <+5>:    mov     $0x2,%ebx  
0x0804806a <+10>:   int     $0x80  
End of assembler dump.  
(gdb)
```

```
darklinux@darklinux: ~  
(gdb) disassemble _start  
Dump of assembler code for function _start:  
0x08048060 <+0>:    mov     eax,0x1  
0x08048065 <+5>:    mov     ebx,0x2  
0x0804806a <+10>:   int     0x80  
End of assembler dump.  
(gdb)
```

Program pertama assembly, EXIT

```
Terminal - darklinux@darklinux: ~
darklinux@darklinux:~$ cat exit.asm
section .text
global _start
_start:
    mov eax,1
    mov ebx,2
    int 0x80
darklinux@darklinux:~$ nasm -f elf32 exit.asm -o exit.o
darklinux@darklinux:~$ ld exit.o -o exit
darklinux@darklinux:~$ ls -l exit
-rwxrwxr-x 1 darklinux darklinux 497 2018-04-29 09:23 exit
darklinux@darklinux:~$ ./exit
darklinux@darklinux:~$ echo $?
2
darklinux@darklinux:~$
```


Program kedua, 'Hello world'

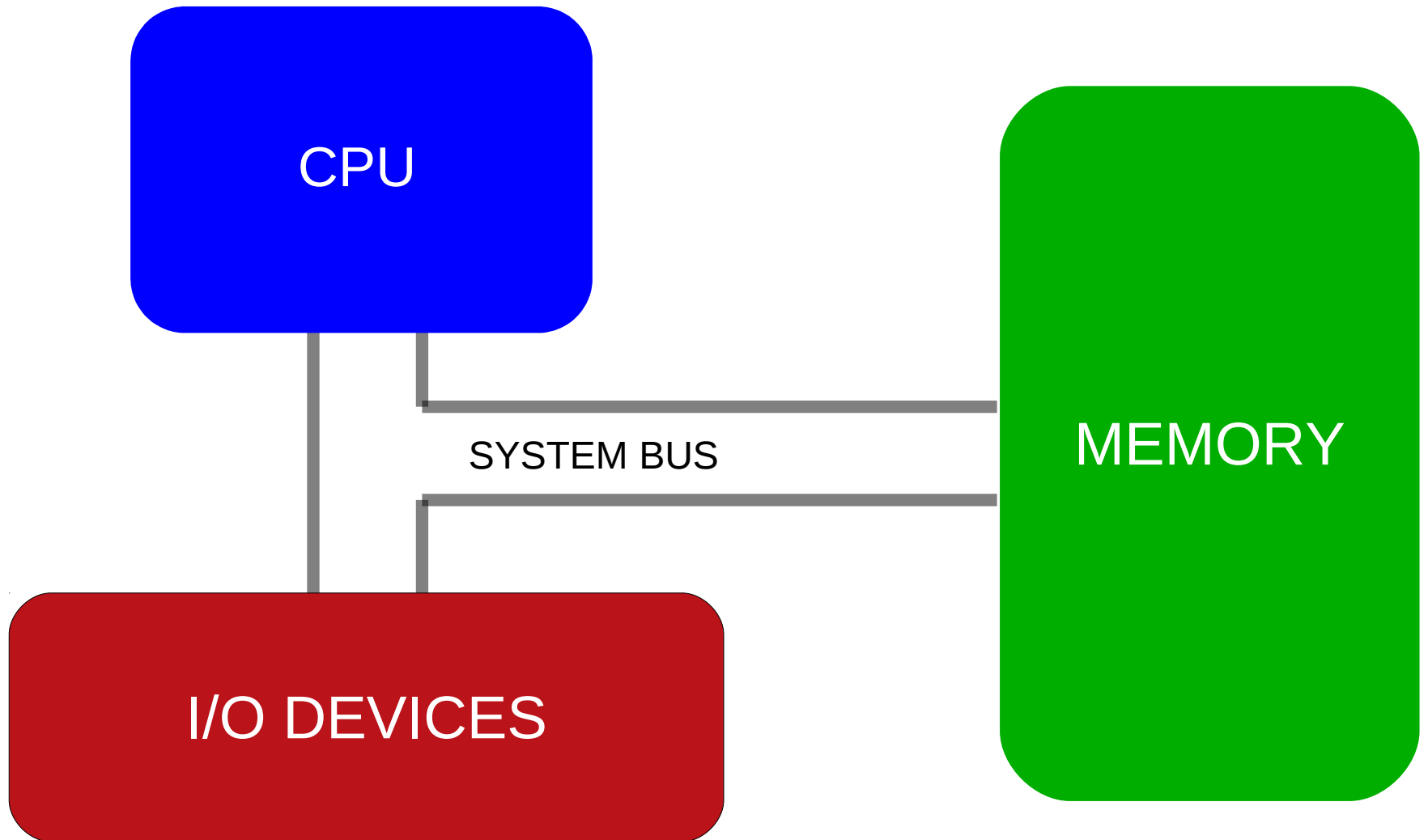
```
Terminal - hello.asm (~/.assembly) - VIM
1 section .text
2 global _start
3 _start:
4
5     mov eax,4
6     mov ebx,1
7     mov edx,lenmsg
8     mov ecx,msg
9     int 0x80
10
11     mov eax,1
12     int 0x80
13
14 section .data
15     msg db 'Welcome to Assembly programming',0xa
16     lenmsg equ $ - msg
~
16,1 All
```

```
Terminal - darklinux@darklinux: ~/.assembly
darklinux@darklinux:~/assembly$ ls -l hello*
-rw-rw-r-- 1 darklinux darklinux 228 2018-04-18 05:37 hello.asm
darklinux@darklinux:~/assembly$ nasm -f elf32 hello.asm -o hello.o
darklinux@darklinux:~/assembly$ ld hello.o -o hello
darklinux@darklinux:~/assembly$ ls -l hello*
-rwxrwxr-x 1 darklinux darklinux 685 2018-04-18 05:57 hello
-rw-rw-r-- 1 darklinux darklinux 228 2018-04-18 05:37 hello.asm
-rw-rw-r-- 1 darklinux darklinux 640 2018-04-18 05:57 hello.o
darklinux@darklinux:~/assembly$ ./hello
Welcome to Assembly programming
darklinux@darklinux:~/assembly$
```

Nasm command line

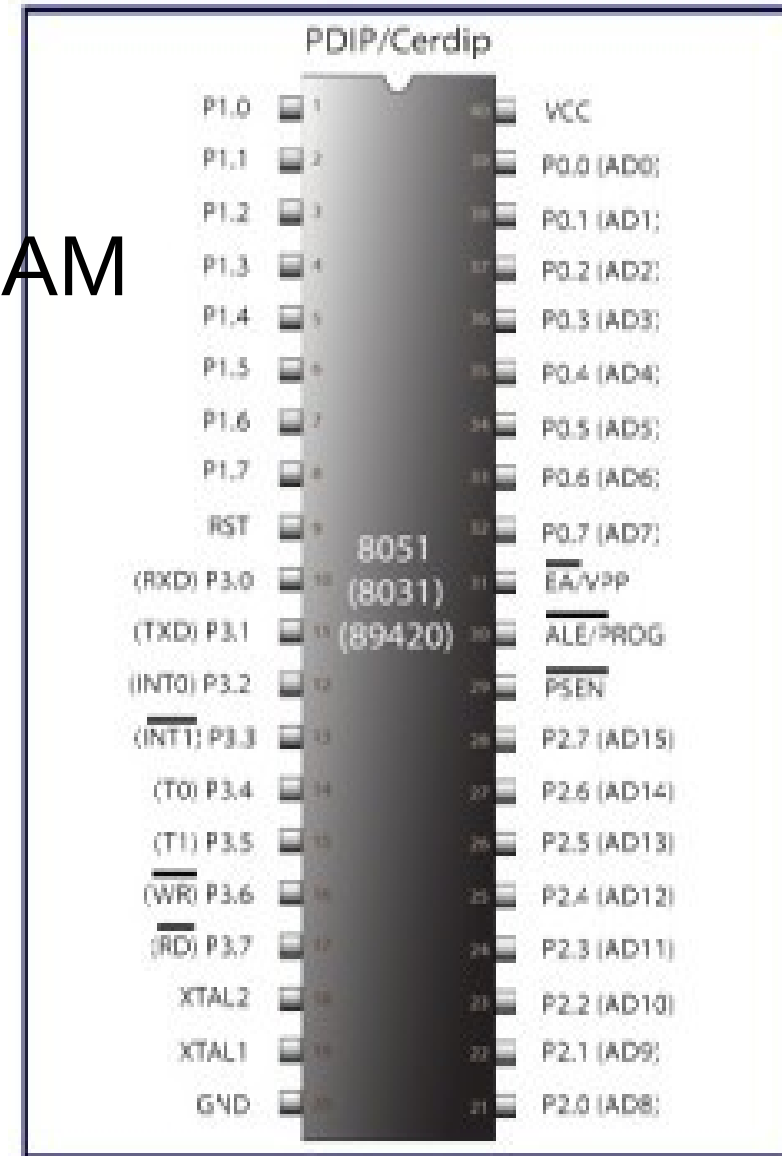
- **nasm** -f elf32 hello.asm -o hello.o → convert jadi object file.
- **ld** hello.o -o hello → link it
- **nasm** -f elf32 **-gstabs+** stack.asm -o stack.o → debugging information untuk gdb.

Pengenalan CPU x86

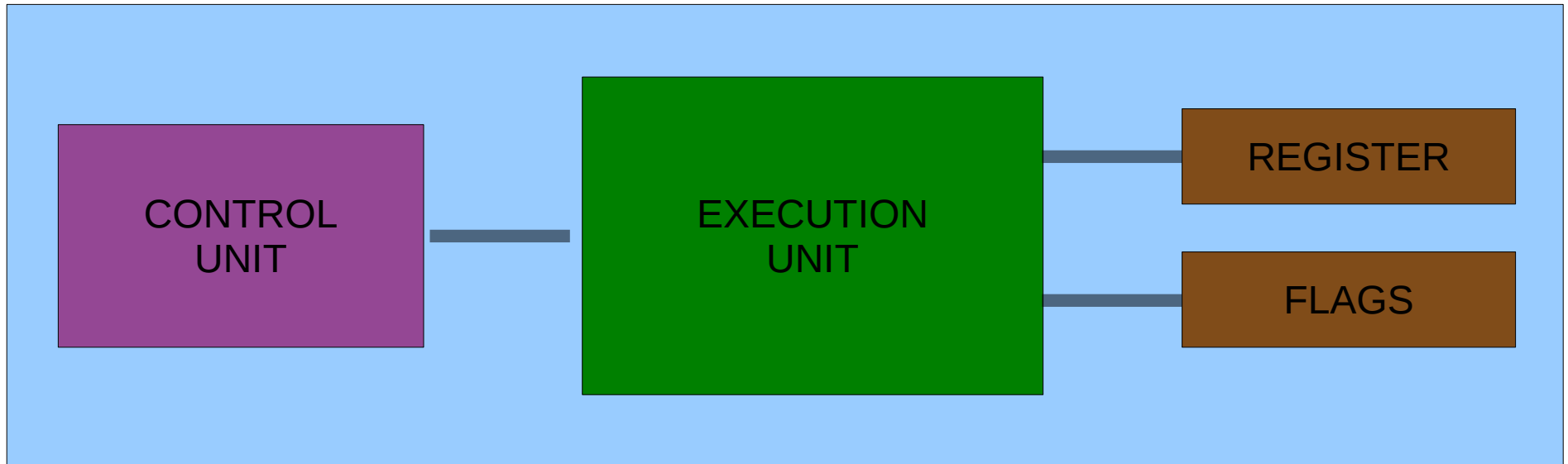


Intel 8051

- **8-bit** microcontroller
- 128/256*(8052) byte on-chip RAM
- Up to 64 kbyte on-chip ROM
- 4x8-bit bi-directional I/O port
- 1xUART
- 2/3 x 16-bit counter/timers
- 9 interrupts sources
- Two-level interrupt priority
- Low-power Idle and power-down modes



Pengenalan CPU x86



Control Unit → Retrive/Decode instructions, Retrieve/Store Data in Memory

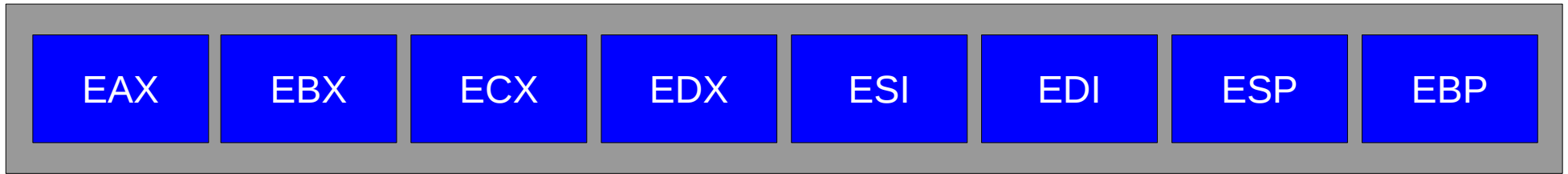
Execution Unit → Actual Execution of intructions happen here

Registers → Internal Memory Locations used as "variables"

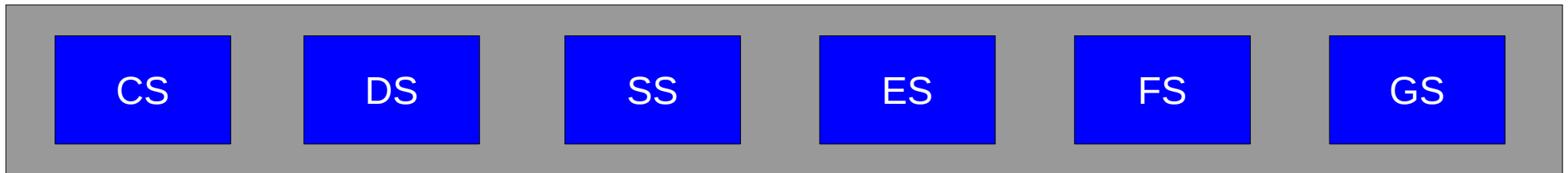
Flags → uses to indicate various 'event' when executing is happening

Pengenalan CPU x86

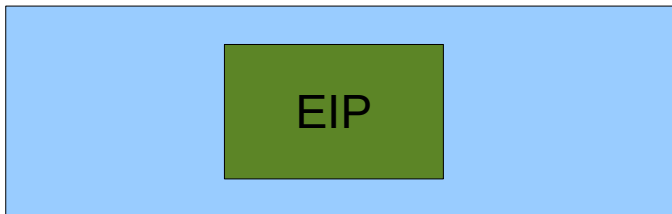
General Purpose Registers (4 arithmetic: EAX, EBX, ECX, EDX) dan 4 address registers ESI, EDI, ESP, EBP)



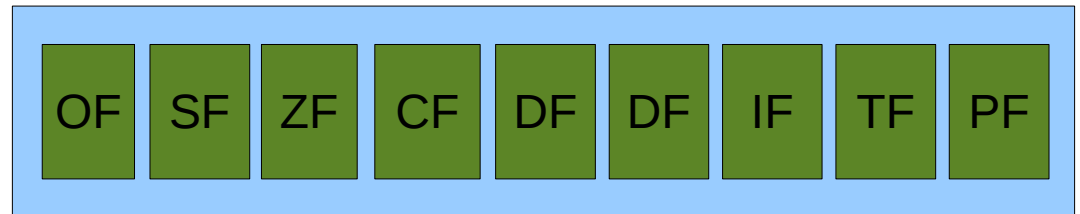
Segment Registers (16 bit, tidak digunakan lagi)



Instruction Pointer Register
32 bit (read only)



FLAG registers (32 bit) mewakili status cpu

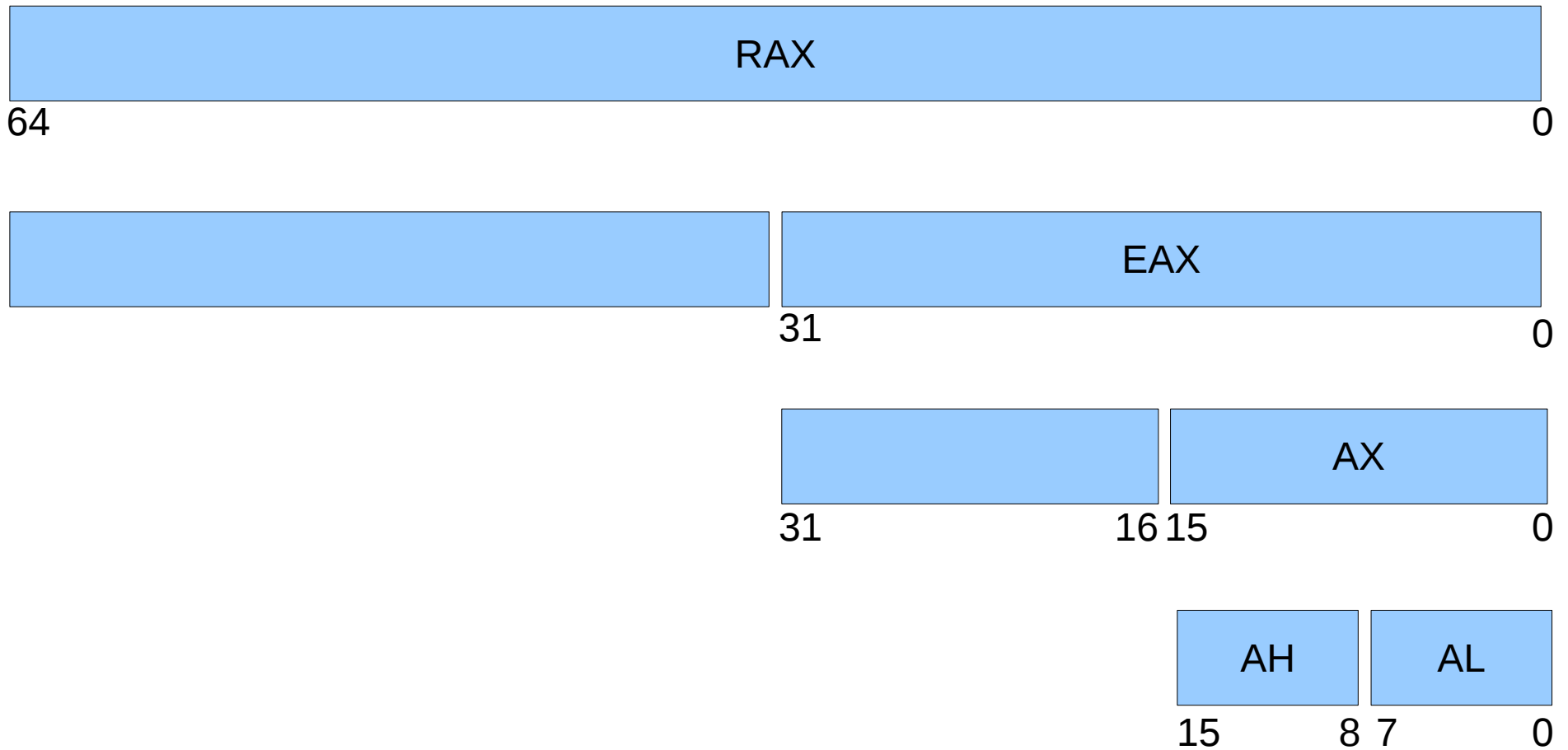


Pengenalan CPU x86

FLAG REGISTERS

- OF → overflow flag
- SF → sign flag
- ZF → zero flag
- CF → carry flag
- DF → direction flag
- IF → interrupt enable flag
- TF → trap flag
- AF → auxiliary carry flag
- PF → parity flag

Pengenalan CPU x86



Pengenalan CPU x86

- Materi yang dibahas difokuskan untuk x86 platform.
- General Purpose CPU registers
 - Register adalah memory kecil di CPU tetapi merupakan sarana tercepat untuk CPU mengakses data.
 - Untuk kumpulan instruksi x86, CPU menggunakan 8 register umum, EAX, EDX, ECX, ESI, EDI, EBP, ESP dan EBX. Masing-masing di rancang untuk keperluan tertentu.

Pengenalan CPU x86

- EAX (Accumulator register) → untuk perhitungan juga untuk menyimpan 'return value' dari fungsi yang di panggil. Penambahan, Pengurangan & Pembandingan menggunakan register ini.
- EDX (Data register) → dasarnya adalah extensi dari register EAX. Fungsinya menyimpan data untuk perhitungan yang lebih komplek seperti perkalian dan pembagian.
- ECX (Count register) → digunakan untuk looping. ECX menghitung kebawah bukan ke atas.

Pengenalan CPU x86

- ESI (Source Index) → menyimpan lokasi input data.
- EDI (Destination index) → menyimpan lokasi hasil operasi data.
 - *Note: ESI untuk membaca EDI untuk menulis.*
- ESP (stack pointer) → mengatur panggilan fungsi dan operasi stack. ESP menunjuk bagian paling atas dari stack/return address.
- BSP (base pointer) → Menunjuk bagian paling bawah dari stack.
- EBS → Satu-satunya register yang tidak di rancang untuk spesifikasi tertentu. Digunakan untuk penyimpanan extra.

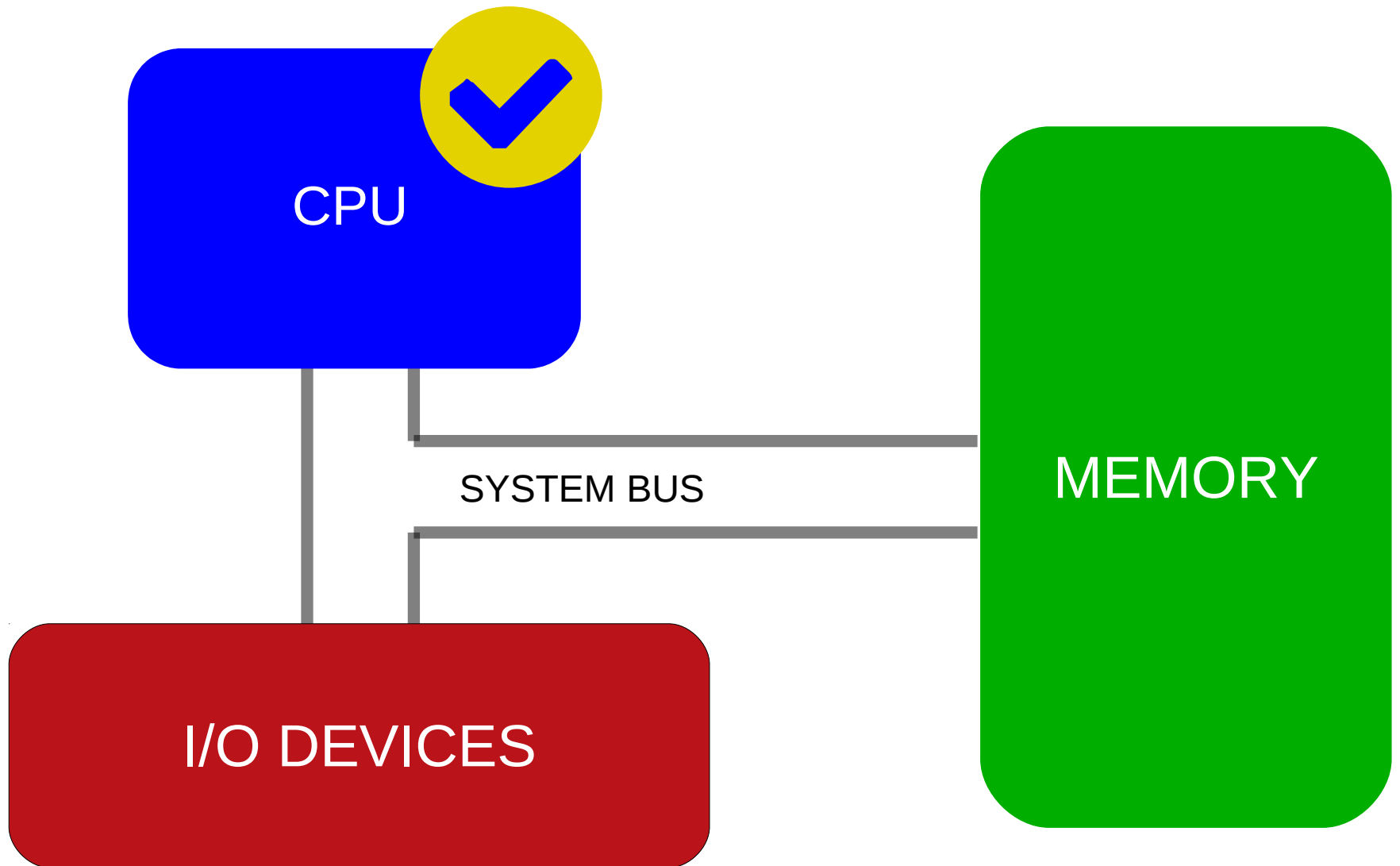
Pengenalan CPU x86

- EIP (Index pointer) → menunjukkan alamat dari baris perintah yang sedang dijalankan.

Pengenalan CPU x86

```
Terminal - darklinux@darklinux: ~/assembly
(gdb) info reg
eax          0x0      0
ecx          0x0      0
edx          0x0      0
ebx          0x0      0
esp          0xbffff3f0 0xbffff3f0
ebp          0x0      0x0
esi          0x0      0
edi          0x0      0
eip          0x8048060 0x8048060 <_start>
eflags       0x212    [ AF IF ]
cs           0x73     115
ss           0x7b     123
ds           0x7b     123
es           0x7b     123
fs           0x0      0
gs           0x0      0
(gdb) █
```

Pengenalan CPU x86



Memory Layout di Linux

- Pada saat program di load ke memory, setiap 'section' di load ke masing-masing wilayah memory.
- Semua kode dan data yang di 'declare', di bawa bersama-sama, bahkan jika source code-nya terpisah.
- Instruksi sebenarnya dari .text section di letakkan pada alamat **0x08048000**. Berikutnya **.data** section dan **.bss** section.
- Alamat terakhir di linux adalah **0xbFFFFFFF**.

Memory Layout di Linux

- Linux memulai stack disini dan turun kebawah menuju masing-masing ***section***.

Virtual memory organized

0x804800

.text segment

- Instructions, static data
- read only

.data segment

- global dan static variables
- heap

.bss

- .bss (block started by symbol)
- Uninitialized data

Heap

- Dynamic memory - malloc()

Unused memory

Stack segment

- Local variables
- control data

0xBFFFFFFF

Byte order / endiannes

BIG ENDIAN

- Start with *MSB* (most significant byte)
- spark, motorolla 68000
- our decimal system

LITTLE ENDIAN

- Start with *LSB* (lowest significant byte)
- i386

```
7      gets(buffer);  
(gdb)  
12345678  
8      puts(buffer);  
(gdb) x/8xw $esp  
0xbffff360:    0xbffff364    0x34333231    0x38373635    0xbbff8100  
0xbffff370:    0xbffff378    0x080484a6    0x00000000    0x0014a113  
(gdb) █
```

CONFIGUREABLE

- MIPS, ARM, IA64

Virtual memory organized

Setiap alamat memory adalah Kebohongan

- Setiap program di letakkan pada lokasi yang sama di memory. Tetapi mengapa mereka tidak bersinggungan atau tumpang tindih?
- Karena program hanya meng-akses '***Virtual memory***'.
- Physical Memory adalah RAM chip di komputer anda. Virtual Memory adalah cara program berpikir tentang memory.

Virtual memory organized

- Sebelum me-load program anda, Linux mencari lokasi kosong dari physical memory yang cukup besar untuk menampung program anda.
- Kemudian memberitahu processor untuk berpura-pura bahwa alamat memory ini adalah alamat sebenarnya dari **0x08048000** tempat program anda yang diletakkan.
- Setiap program mendapatkan masing-masing **sandbox** untuk dimainkan.

Virtual memory organized

- Setiap program percaya bahwa mereka masing-masing adalah stand alone and menikmati semua memory.
- Alamat yang program percaya untuk digunakan dinamakan ***virtual address*** sedangkan alamat sebenarnya di chip memory dinamakan ***physical address***.
- Proccess penunjukkan virtual address ke physical address dinamakan ***mapping***.

Virtual memory organized

```
wait.asm + (~) - VIM
1 section .text
2
3 global _start
4 _start:
5
6     ;display message
7     mov eax,4
8     mov ebx,1
9     mov ecx,msg
10    mov edx,lenmsg
11    int 0x80
12
13    ;wait keypress
14    mov eax,3
15    mov ebx,0
16    int 0x80
17
18    ;exit
19    mov eax,1
20    mov ebx,2
21    int 0x80
22
23 section .data
24     msg db 'Please, press any key to continue',0xa
25     lenmsg equ $-msg
~
25,1-4 All
```

Virtual Memory address

```
darklinux@darklinux: ~
darklinux@darklinux:~$ ./wait1
Press any key to continue...
█

darklinux@darklinux: ~
darklinux@darklinux:~$ ./wait2
Press any key to continue...
█

darklinux@darklinux: ~
darklinux@darklinux:~$ ps -aux | grep wait
Warning: bad ps syntax, perhaps a bogus '-?' See http://procps.sf.net/faq.html
1000      2013  0.0  0.0   148    4 pts/0    S+   08:18   0:00 ./wait1
1000      2071  0.0  0.0   148    4 pts/1    S+   08:18   0:00 ./wait2
1000      2202  0.0  0.0  4444   792 pts/3    S+   08:19   0:00 grep --color=auto wait
darklinux@darklinux:~$ cat /proc/2013/maps
0069c000-0069d000 r-xp 00000000 00:00 0          [vdso]
08048000-08049000 r-xp 00000000 08:07 332740      /home/darklinux/wait1
08049000-0804a000 rwxp 00000000 08:07 332740      /home/darklinux/wait1
bf7f0000-bf811000 rwxp 00000000 00:00 0          [stack]
darklinux@darklinux:~$ █

darklinux@darklinux: ~
darklinux@darklinux:~$ cat /proc/2071/maps
005f6000-005f7000 r-xp 00000000 00:00 0          [vdso]
08048000-08049000 r-xp 00000000 08:07 334441      /home/darklinux/wait2
08049000-0804a000 rwxp 00000000 08:07 334441      /home/darklinux/wait2
bf902000-bf923000 rwxp 00000000 00:00 0          [stack]
darklinux@darklinux:~$ █
```

Note:

Vdso (virtual Dynamic Shared Object)

Mekanisme di kernel untuk mengekport kernel-space routines untuk user-space aplikasi
Sehingga aplikasi dapat memanggil kernel-space routines (syscall) saat berjalan.

Virtual Memory address

```
root@darklinux: ~
root@darklinux:~# echo 0 > /proc/sys/kernel/randomize_va_space
root@darklinux:~# cat /proc/sys/kernel/randomize_va_space
0
root@darklinux:~#

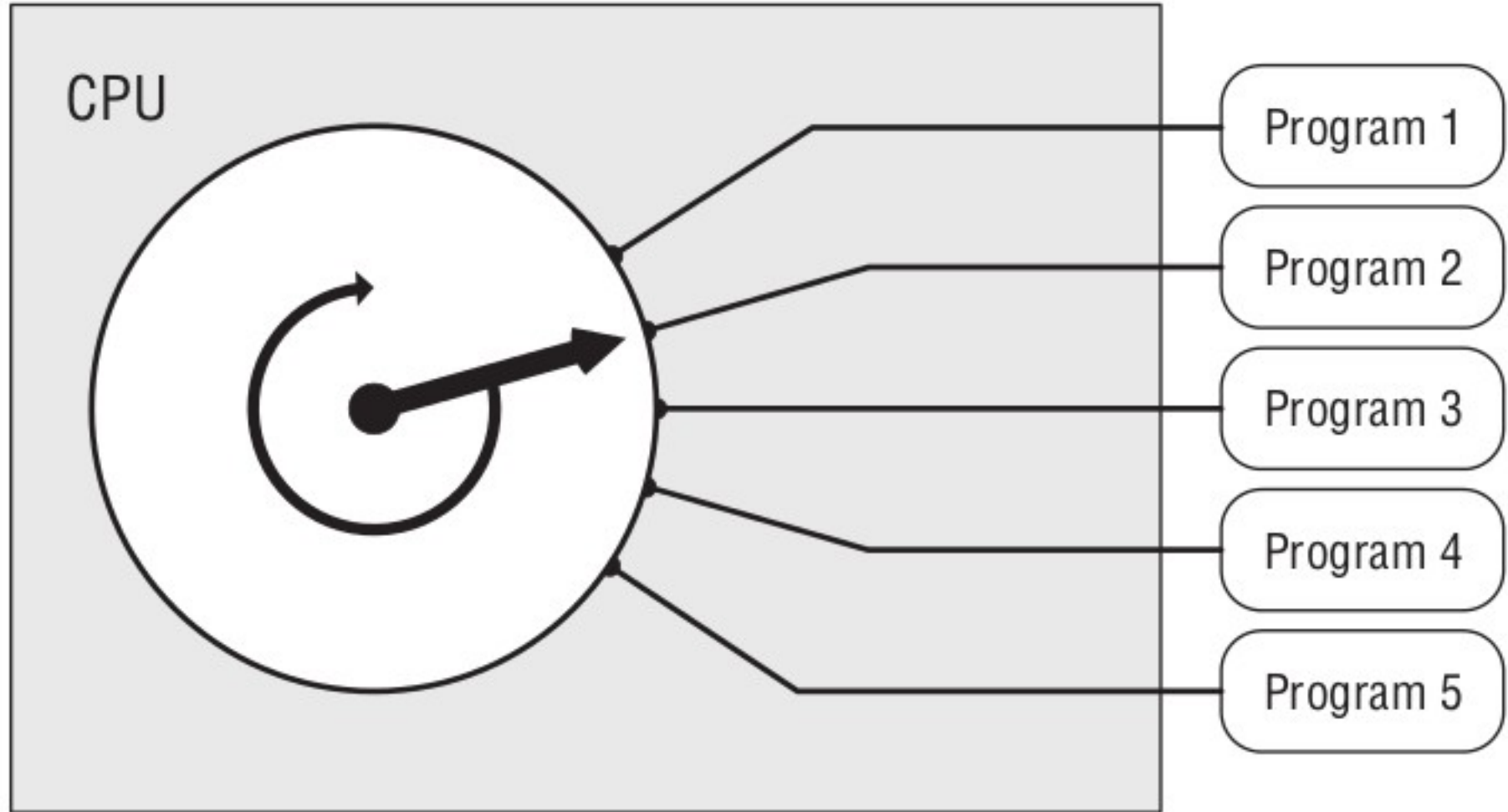
darklinux@darklinux: ~
darklinux@darklinux:~$ ./wait1
Press any key to continue...
█

darklinux@darklinux: ~
darklinux@darklinux:~$ ./wait2
Press any key to continue...
█

darklinux@darklinux: ~
darklinux@darklinux:~$ ps -aux | grep wait
Warning: bad ps syntax, perhaps a bogus '-'? See http://procps.sf.net/faq.html
1000      2418  0.0  0.0   148    4 pts/1    S+   08:25   0:00 ./wait2
1000      2419  0.0  0.0   148    4 pts/0    S+   08:25   0:00 ./wait1
1000      2421  0.0  0.0  4444   792 pts/3    S+   08:25   0:00 grep --color=auto wait
darklinux@darklinux:~$ cat /proc/2418/maps
00110000-00111000 r-xp 00000000 00:00 0          [vdso]
08048000-08049000 r-xp 00000000 08:07 334441      /home/darklinux/wait2
08049000-0804a000 rwxp 00000000 08:07 334441      /home/darklinux/wait2
bffd000-c0000000 rwxp 00000000 00:00 0          [stack]
darklinux@darklinux:~$ █

darklinux@darklinux: ~
darklinux@darklinux:~$ cat /proc/2419/maps
00110000-00111000 r-xp 00000000 00:00 0          [vdso]
08048000-08049000 r-xp 00000000 08:07 332740      /home/darklinux/wait1
08049000-0804a000 rwxp 00000000 08:07 332740      /home/darklinux/wait1
bffd000-c0000000 rwxp 00000000 00:00 0          [stack]
darklinux@darklinux:~$ █
```

Keajaiban Multitasking

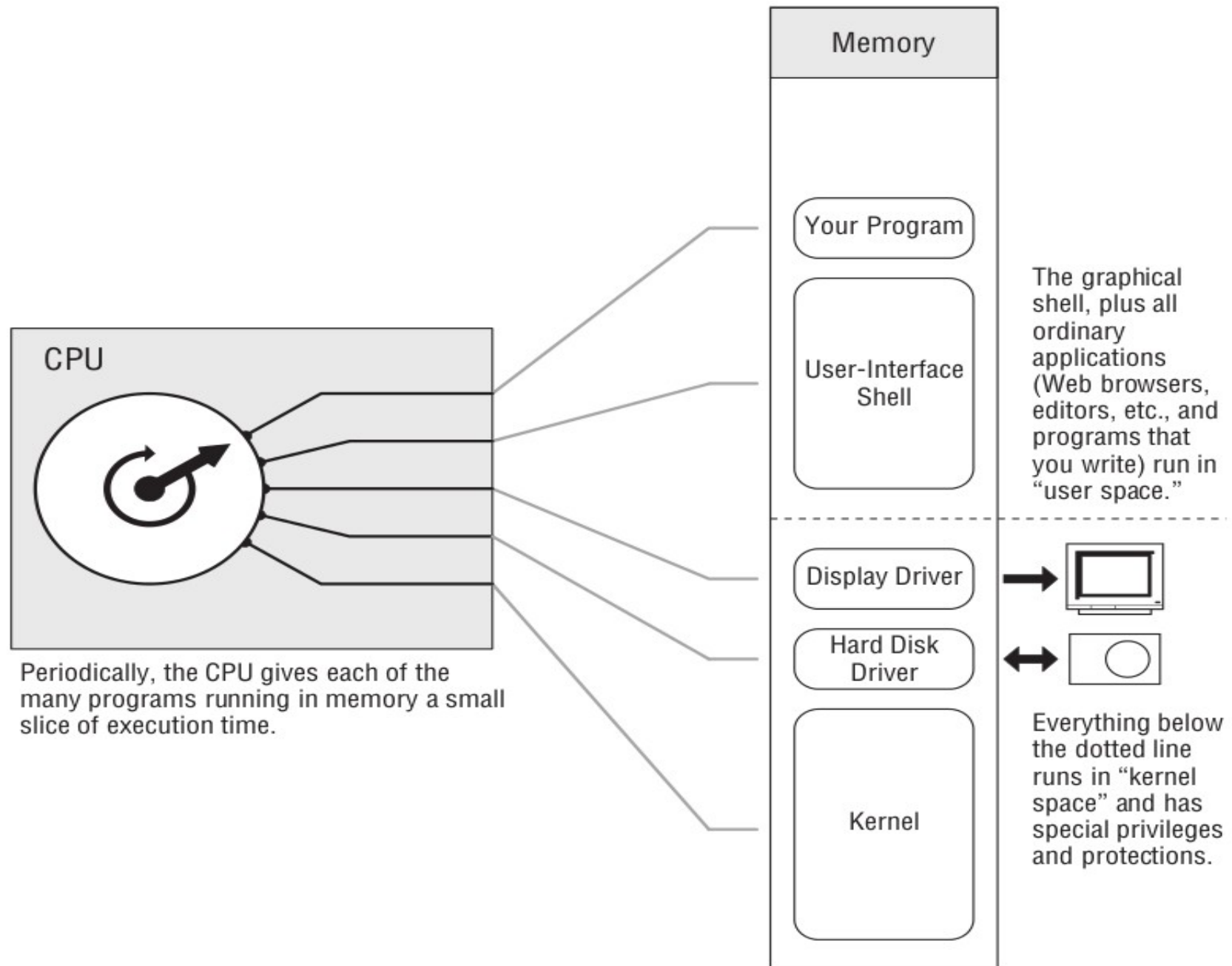


The idea of multitasking

Keajaiban Multitasking

- Tahun 1995, Microsoft me-release windows 95.
- Berjalan di 32-bit protected mode, tetapi tidak full karena masih mempunyai DOS dan aplikasi DOS yang berjalan di dalamnya.
- Win 95 membuat ilusi seolah-olah semua program di memory berjalan bersama.
- OS mendefinisikan priority dari masing-masing program, yang mana membutuhkan 'clock-cycle' yang lebih banyak dan yang tidak.

Keajaiban Multitasking



Keajaiban Multitasking

- Tahun 1991, Linus Torvalds me-release Linux.
- Tidak ada GUI seperti Win95 tetapi dapat menangani multitasking dan struktur internalnya lebih powerful.
- Core dari Linux adalah block code yang dinamakan **Kernel**.
- System memory di tandai sebagai **Kernel Space** dan **User space**.
- Komunikasi diantara keduanya di tangani oleh ***System Call***.

Keajaiban Multitasking

- Akses langsung ke hardware dibatasi di software yang berjalan di kernel space dan hanya bisa dilakukan via **kernel mode device drivers**.
- Microsoft me-release Windows NT, Unix-inspired OS, tahun 1993. Sama seperti Linux, device driver berjalan di kernel yang lainnya di user space.
- Basic design ini masih digunakan di windows 2000, xp, vista, 7 dan 10.

Linux System Call

- System calls adalah API (application program interface) yang menghubungkan user dan kernel.
- Cara menggunakan system call
 - Letakkan system call number di register EAX.
 - Masukkan arguments di register EBX, ECX, EDX, ESI, EDI atau EBP.
 - Panggil interrupt 80h (0x80).
 - Hasil biasanya di kembalikan ke register EAX

Linux System Call

- Bagaiman kita mengetahui register mana yang dibutuhkan untuk system call tertentu?
 - *Setiap arsitektur mendefinisikan ABI (application binary interface) yang pada dasarnya mengatakan 'argument pertama harus disini, argumen kedua harus disana, return value disini'.*
- Bagaimana kita mengetahui system call ini untuk mesin yang mana?
 - *<http://syscalls.kernelgrok.com/>*
- Apakah akan berbeda untuk setiap processor i586 dan i386?
 - *Nomor syscall sendiri berbeda antara platform. Contoh: syscall 'open' di x86_64 adalah 2 sedangkan x86 adalah 5.*

Linux System Call

linux/syscall_32.tbl at master · torvalds / linux

GitHub, Inc. (US) | https://github.com/torvalds/linux/blob/master/

Search

torvalds / linux

Watch 6,279

Code Pull requests 203 Projects 0 Insights

Branch: master linux / arch / x86 / entry / syscalls / syscall_32.tbl

Dominik Brodowski syscalls/core, syscalls/x86: Rename struct pt_regs-based sys_*() to _...

5 contributors

399 lines (398 sloc) 21.6 KB Raw

```
1 #
2 # 32-bit system call numbers and entry vectors
3 #
4 # The format is:
5 # <number> <abi> <name> <entry point> <compat entry point>
6 #
7 # The __ia32_sys and __ia32_compat_sys stubs are created on-the-fly for
8 # sys_*() system calls and compat_sys_*() compat system calls if
9 # IA32_EMULATION is defined, and expect struct pt_regs *regs as their only
10 # parameter.
11 #
12 # The abi is always "i386" for this file.
13 #
14 0      i386      restart_syscall      sys_restart_syscall      __ia32_sys_restart_syscall
15 1      i386      exit                  sys_exit                  __ia32_sys_exit
16 2      i386      fork                  sys_fork                  __ia32_sys_fork
17 3      i386      read                  sys_read                  __ia32_sys_read
18 4      i386      write                 sys_write                 __ia32_sys_write
19 5      i386      open                  sys_open                  __ia32_compat_sys_open
20 6      i386      close                 sys_close                 __ia32_sys_close
```

Linux System Call

linux/syscall_64.tbl at master · torvalds / linux

Code Pull requests 203 Projects 0 Insights

Branch: master linux / arch / x86 / entry / syscalls / syscall_64.tbl

Dominik Brodowski syscalls/core, syscalls/x86: Rename struct pt_regs-based sys_*() to _...

4 contributors

387 lines (385 sloc) 15.2 KB

```
1 #
2 # 64-bit system call numbers and entry vectors
3 #
4 # The format is:
5 # <number> <abi> <name> <entry point>
6 #
7 # The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls
8 #
9 # The abi is "common", "64" or "x32" for this file.
10 #
11 0      common  read      __x64_sys_read
12 1      common  write     __x64_sys_write
13 2      common  open      __x64_sys_open
14 3      common  close     __x64_sys_close
15 4      common  stat      __x64_sys_newstat
16 5      common  fstat     __x64_sys_newfstat
17 6      common  lstat     __x64_sys_newlstat
18 7      common  poll      __x64_sys_poll
```

Linux System Call

https://syscalls.kernelgrok.com



Search



Linux Syscall Reference

Show 10 entries

Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	-	-	-	fs/open.c:933
9	sys_link	0x09	const char __user *oldname	const char __user *newname	-	-	-	fs/namei.c:2520

Showing 1 to 10 of 338 entries

First Previous 1 2 3 4 5 Next Last

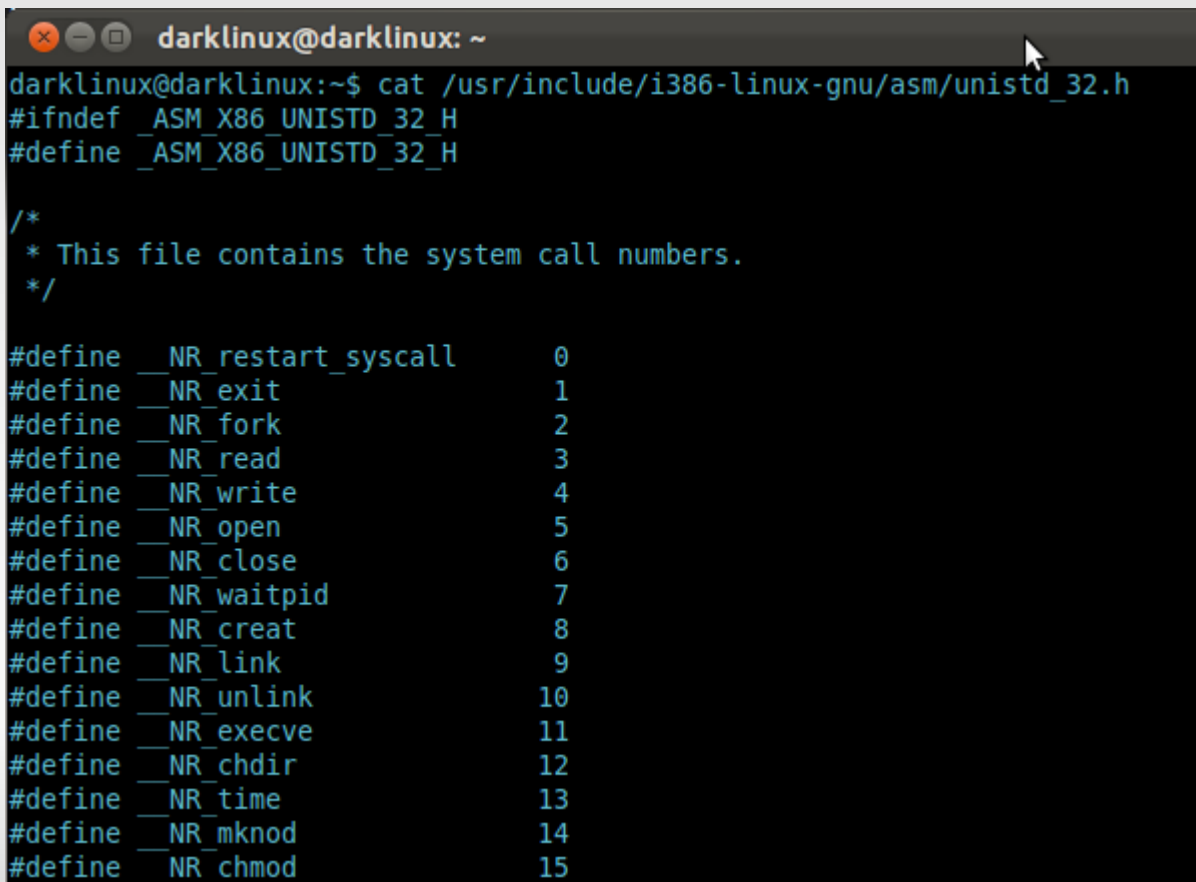
Generated from Linux kernel 2.6.35.4 using **Exuberant Ctags**, **Python**, and **DataTables**.
Project on **GitHub**. Hosted on **GitHub Pages**.

Linux System Call

- Semua system calls di daftarkan di:

```
$ ls -l /usr/include/i386-linux-gnu/asm/unistd_32.h
```

```
-rw-r--r-- 1 root root 10152 2011-10-08 05:10 /usr/include/i386-linux-gnu/asm/unistd_32.h
```

A terminal window titled 'darklinux@darklinux: ~' showing the output of the command 'cat /usr/include/i386-linux-gnu/asm/unistd_32.h'. The output is a C header file defining system call numbers for i386 Linux. It starts with a conditional compilation block for _ASM_X86_UNISTD_32_H, followed by a comment stating 'This file contains the system call numbers.' and a list of #define statements for various system calls, each with a corresponding number from 0 to 15.

```
darklinux@darklinux:~$ cat /usr/include/i386-linux-gnu/asm/unistd_32.h
#ifndef _ASM_X86_UNISTD_32_H
#define _ASM_X86_UNISTD_32_H

/*
 * This file contains the system call numbers.
 */

#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
#define __NR_mknod              14
#define __NR_chmod              15
```

Kode Mesin

- Instruksi assembler sebenarnya hanya angka.
- Kita sudah mengetahui jika komputer hanya mengenal bilangan 1 dan 0.
- Instruksi `mov eax, 1` → `B801000000` → `101110000000000001000000000000000000000000000000`

```

darklinux@darklinux: ~
darklinux@darklinux:~$ objdump -d coding.o

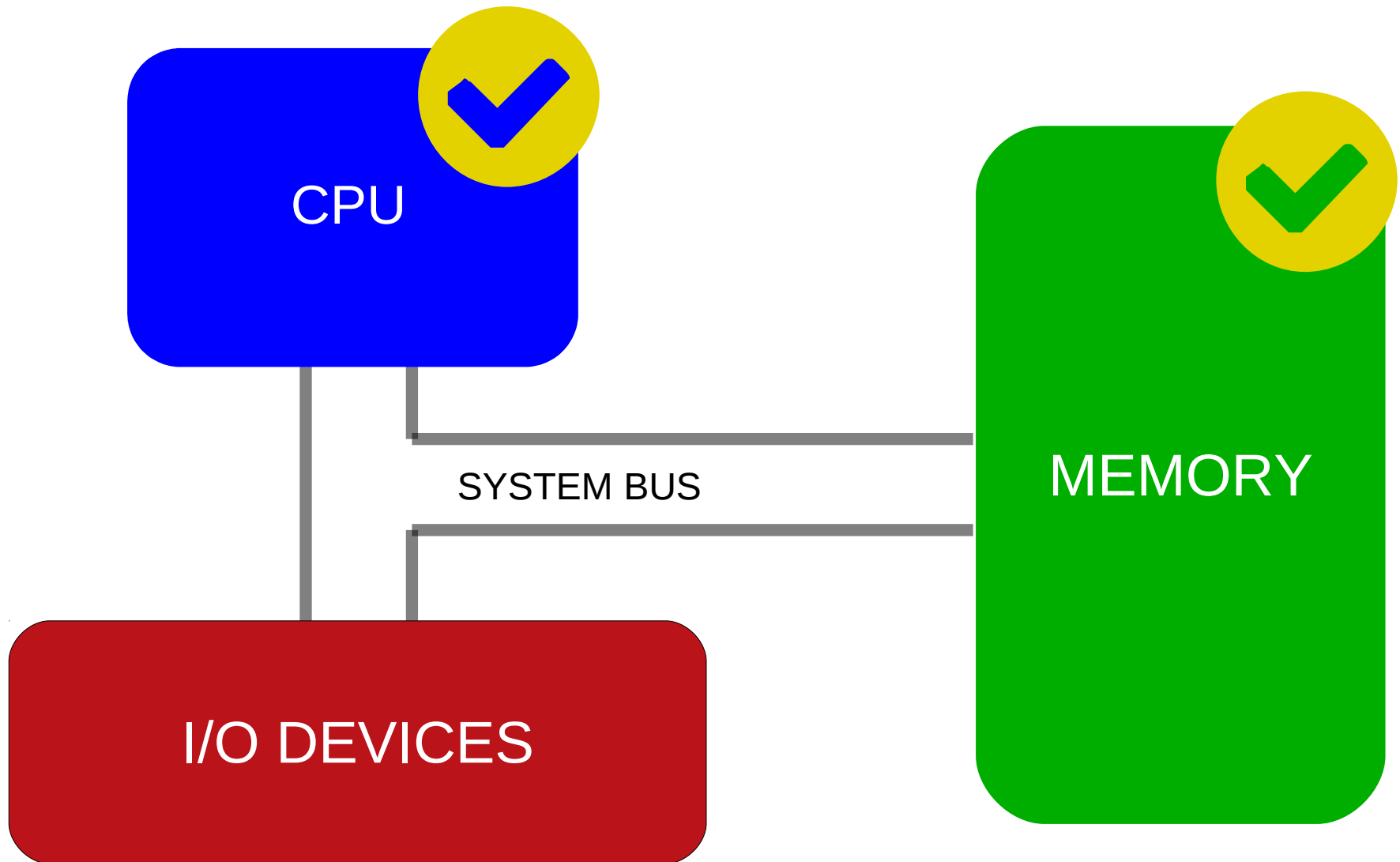
coding.o:          file format elf32-i386


Disassembly of section .text:

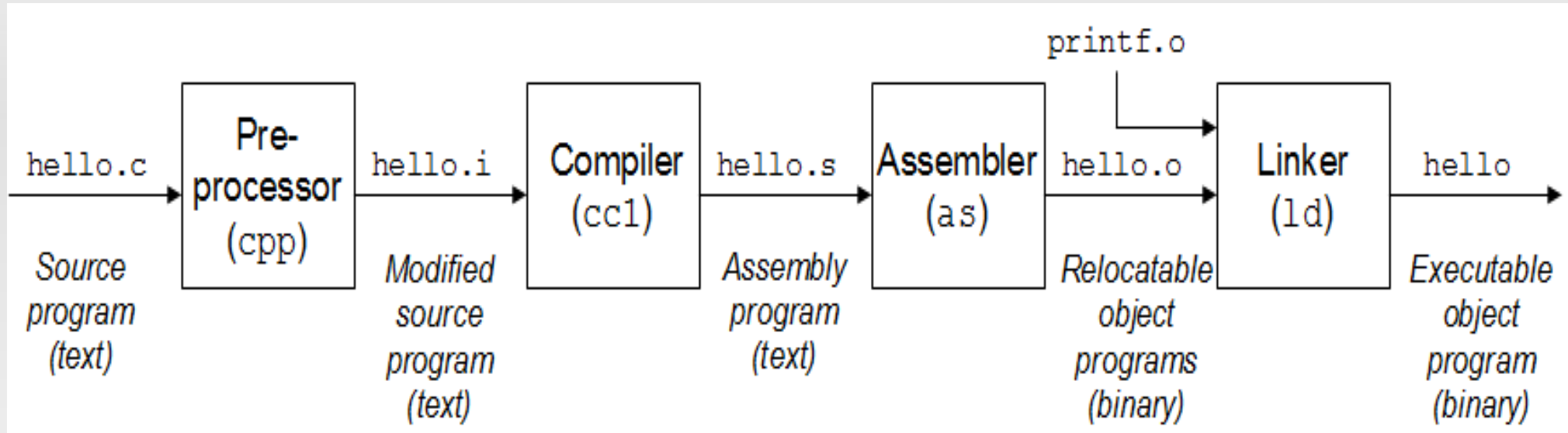
00000000 <_start>:
   0:  b8 01 00 00 00      mov     $0x1,%eax
   5:  bb 01 00 00 00      mov     $0x1,%ebx
   a:  b8 01 00 00 00      mov     $0x1,%eax
   f:  ba 01 00 00 00      mov     $0x1,%edx
  14:  b9 01 00 00 00      mov     $0x1,%ecx
  19:  29 d8              sub     %ebx,%eax
  1b:  29 ca              sub     %ecx,%edx
  1d:  01 d8              add     %ebx,%eax
  1f:  01 ca              add     %ecx,%edx
  21:  b8 01 00 00 00      mov     $0x1,%eax
  26:  cd 80              int     $0x80
darklinux@darklinux:~$ █

```

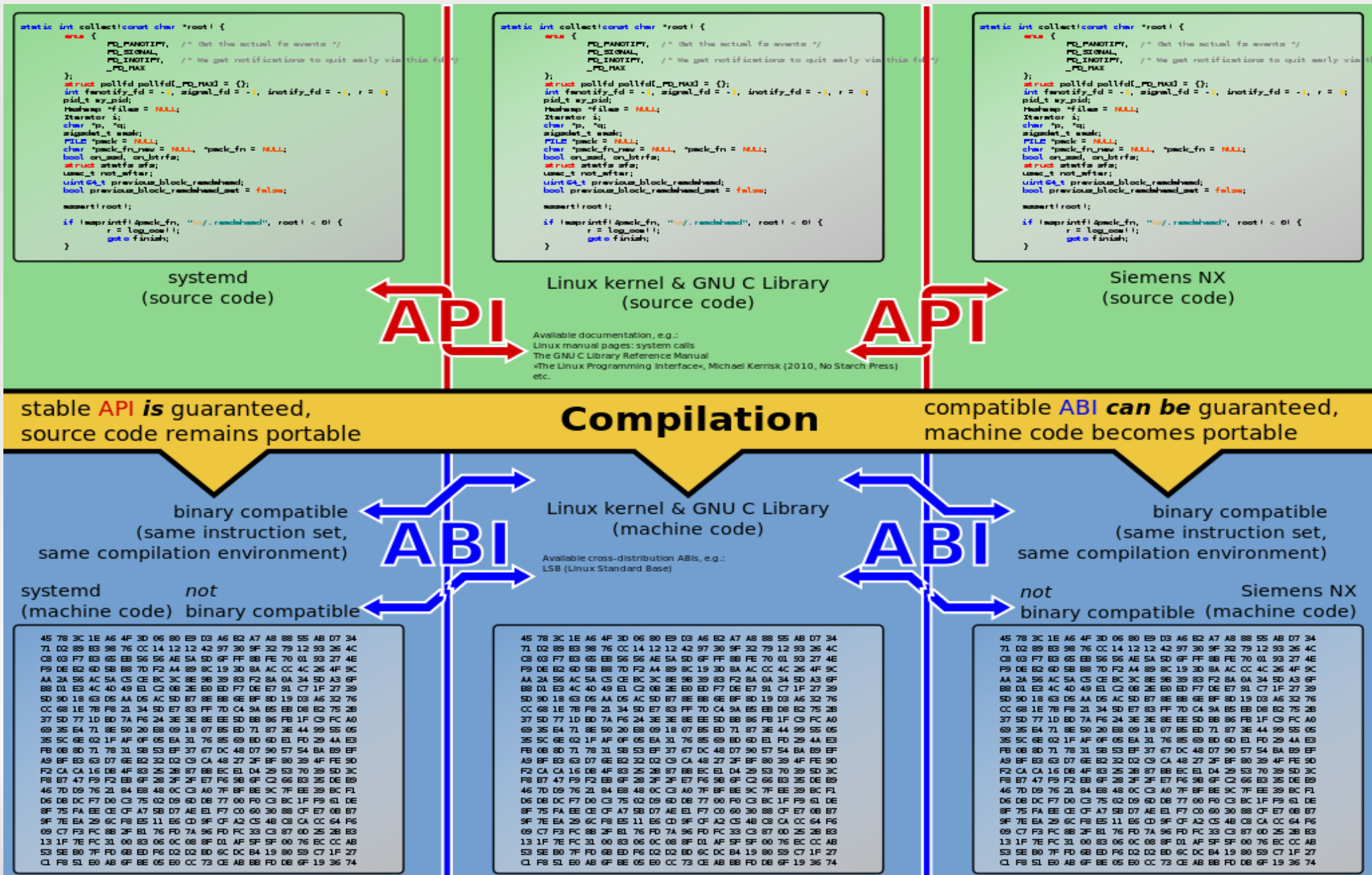
Pengenalan CPU x86



Compiler di bahasa c



ABI (Application Binary Interface)



Constant

```
Terminal - luas.asm + (~/.assembly) - VIM
1 section .text
2 global _start
3 _start:
4
5     mov ebx,luas
6     add ebx,'0'
7     mov [msg],ebx
8
9     mov eax,sys_write
10    mov ebx,stdout
11    mov edx,1
12    mov ecx,msg
13    int 0x80
14
15    mov eax,sys_write
16    mov ebx,stdout
17    mov edx,2
18    mov ecx,newline
19    int 0x80
20
21    mov eax,sys_exit
22    int 0x80
23
24 section .bss
25     msg resb 1
26
27 section .data
28     panjang equ 2
29     lebar equ 3
30     luas equ panjang * lebar
31     sys_write equ 4
32     sys_exit equ 1
33     stdout equ 1
34     newline db 0xa,0xd
35
```

35,0-1 All

Note:

\$ → address of the current location.
The byte of current string minus the bytes
of the last string = lenght of string
Lenstring equ \$-string

Constant

```
Terminal - luas.asm + (~/.assembly) - VIM
1 section .text
2 global _start
3 _start:
4
5     mov ebx,luas
6     add ebx,'0'
7     mov [msg],ebx
8
9     mov eax,sys_write
10    mov ebx,stdout
11    mov edx,1
12    mov ecx,msg
13    int 0x80
14
15    mov eax,sys_write
16    mov ebx,stdout
17    mov edx,2
18    mov ecx,newline
19    int 0x80
20
21    mov eax,sys_exit
22    int 0x80
23
24 section .bss
25     msg resb 1
26
27 section .data
28     panjang equ 2
29     lebar equ 3
30     luas equ panjang * lebar
31     sys_write equ 4
32     sys_exit equ 1
33     stdout equ 1
34     newline db 0xa,0xd
35
```

35,0-1 All

Note:

\$ → address of the current location.
The byte of current string minus the bytes
of the last string = lenght of string
Lenstring equ \$-string

Declaration

- `db 100` → alokasikan 1 byte memory dan masukkan nilai 100 (desimal) kedalamnya.
- `db 64h`
- `db 'd'`
- `db 'd',64h,100`
- `db 'ddd',C4h,17,0x80`
- `dw 100` → define word
- `dd 100` → define double word
- `dq 100` → define quad word
- `dt 1` → define 8 byte double real
- `do 1` → define 10 byte extended real
- `dy 1` → define 10 byte bcd

Declaration

- `resb 22` → reserve space 22 byte
- `resw 22` → reserve space 22 word
- `resd 22` → reserve space 22 double word
- `resq 22` → reserve space 22 quad word
- `rest 22` → reserve space 22 ten byte
- `reso 22` → reserve space
- `resy 22` →

Variables

- Setiap byte character disimpan dalam ASCII dalam bentuk Hexadesimal.
- Setiap nilai desimal secara otomatis dikonversi menjadi 16 bit binary dan disimpan sebagai bilangan hexadesimal.

Variables

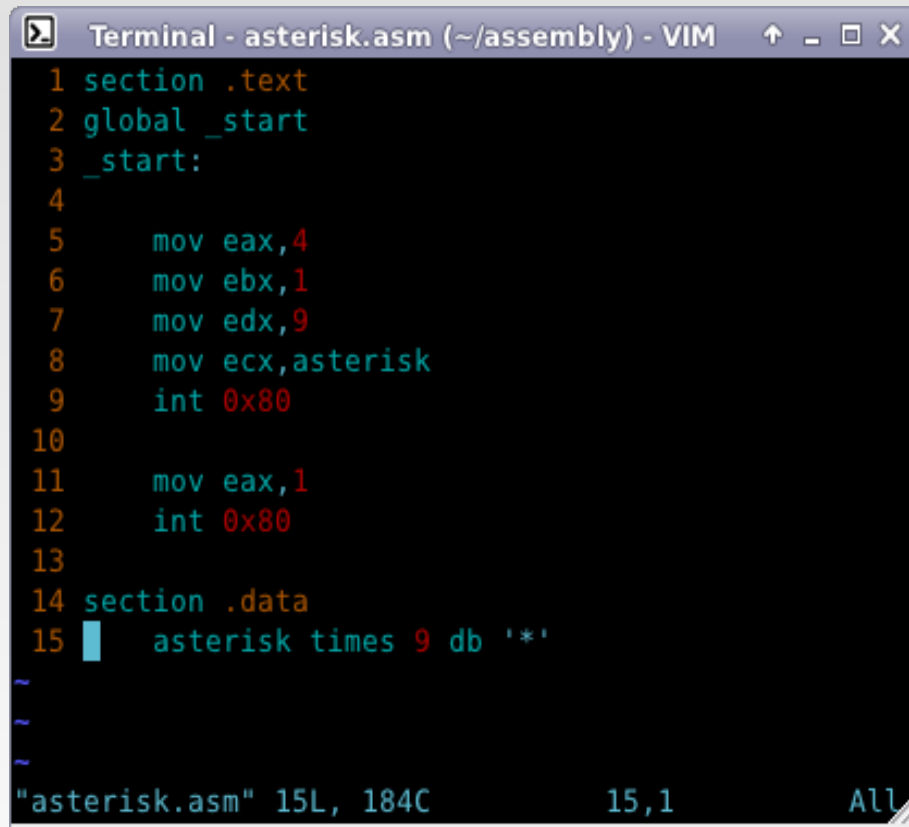
- Alokasi ruang penyimpan untuk "initialized data".
 - db (define byte) → 1 bytes
 - dw (define word) → 2 bytes
 - dd (define double word) → 4 bytes
 - dq (define quard word) → 8 bytes
 - dt (define ten bytes) → 10 bytes

Variables

- Alokasi ruang penyimpanan untuk "uninitialized data".
 - resb (reserve a byte)
 - resw (reserve a word)
 - resd (reserve a double word)
 - resq (reserve a quad word)
 - Rest (reserve a ten word)

Variables

- Multiple Initialization



```
Terminal - asterisk.asm (~/.assembly) - VIM
1 section .text
2 global _start
3 _start:
4
5     mov eax,4
6     mov ebx,1
7     mov edx,9
8     mov ecx,asterisk
9     int 0x80
10
11     mov eax,1
12     int 0x80
13
14 section .data
15     asterisk times 9 db '*'
~
~
~
"asterisk.asm" 15L, 184C      15,1      All
```

• Arithmetic Instructions

- inc (increase) → inc destination
- dec (decrease) → dec destination
- add/sub → add/sub destination,source
- mul/imul (multiply) → mul/imul multiplier
- div/idiv (divide) → div/idiv divisor

Logical Instruction

- AND → AND operand1, operand2
- OR → OR operand1,operand2
- XOR → XOR operand1,operand2
- TEST → TEST operand1,operand2
- NOT → NOT operand1
- The operand1 bisa di register or memory.
- The operand2 bisa di register/memory atau immediate value (constant).
- Operasi Memory ke memory tidak dimungkinkan.
- Instruksi ini membandingkan atau menyamakan bits dari operand dan set flags (CF, OF, PF,SF dan ZF).

Conditions

- Control flow program dilakukan via GOTO (jump, branches dan calls).
- Effect dari instruksi ini, mengganti langsung nilai di program counter.
- **Jump (jmp)** adalah GOTO tanpa kondisi.
 - `jmp 5 == mov eip,5`
- Hampir semua operasi di register-register, seperti Pertambahan, Pengurangan mempunyai efek merubah status **flags**.

Conditions

- Bagaimana cara test jika dua nilai adalah sama?
 - $8==8$?
 - dengan cara mengurangi 8 dengan 8. jika hasilnya nol maka nilainya sama.
 - **beq** (branch if equal to zero) sama dengan **je**(jump if equal), **jz**(jump if zero)

Conditions

- `cmp` → *cmp destination, source*
- unconditional jump → *jmp label*
- conditional jump (untuk operasi aritmetika) →
 - `je/jz` → jump if equal/jump if zero
 - `jne/jnz` → jump if not equal/jump if not zero
 - `jg/jnle` → jump if greater/jump if not less/equal
 - `jge/jnl` → jump if greater/equal or jump if not less
 - `jl/jnge` → jump if less or jump if not greater/equal
 - `jle/jng` → jump if less/equal or jump if not greater

Conditions

- Call adalah GOTO tanpa kondisi.
- Perbedaan dengan **jmp** adalah, pada saat di panggil, call akan menyimpan (**push**) alamat berikutnya di stack. Sehingga instruksi **RET** bisa mengambil (**pop**) alamat tersebut nanti setelah semua instruksi di bagian call di jalankan.

Loops

```
Terminal - smile.asm (~/.assembly) - VIM
1 section .text
2 global _start
3 _start:
4
5     repeat:
6     mov eax,4
7     mov edx,lenasmile
8     mov ecx,smile
9     mov ebx,1
10    int 0x80
11
12    loop repeat
13
14 section .data
15     smile db ":) "
16     lenasmile equ $-smile
~
~
"smile.asm" 16L, 205C      16,1      All
```

Numbers

- Data bilangan secara umum adalah dalam bentuk binary.
- Instruksi aritmetika bekerja dalam bentuk binary.
- Data yang ditampilkan ke layar atau yang dimasukkan dari keyboard dalam bentuk ASCII.
- Data harus dikonversi dalam bentuk binary saat perhitungan dan di kembalikan dalam bentuk ASCII saat akan di tampilkan.

Numbers

```
Terminal - math.asm (~/.assembly) - VIM
1 section .text
2 global _start
3
4 _start:
5     ;show message var1
6     mov eax,4
7     mov ebx,1
8     mov edx,lenmsgvar1
9     mov ecx,msgvar1
10    int 0x80
11    ;read var1
12    mov eax,3
13    mov ebx,0
14    mov edx,2
15    mov ecx,num1
16    int 0x80
17
18    ;show message var2
19    mov eax,4
20    mov ebx,1
21    mov edx,lenmsgvar2
22    mov ecx,msgvar2
23    int 0x80
24    ;read var2
25    mov eax,3
26    mov ebx,0
27    mov edx,2
28    mov ecx,num2
29    int 0x80
30
31    ;sum var1,var2
32    mov eax,[num1]
33    sub eax,'0'
34    mov ebx,[num2]
35    sub ebx,'0'
36    add eax,ebx
37    add eax,'0'
38    mov [result],eax
39
```

39,0-1 Top

```
Terminal - math.asm (~/.assembly) - VIM
40    ;display result
41    mov eax,4
42    mov ebx,1
43    mov edx,lenmsgresult
44    mov ecx,msgresult
45    int 0x80
46    mov eax,4
47    mov ebx,1
48    mov edx,1
49    mov ecx,result
50    int 0x80
51    mov eax,4
52    mov ebx,1
53    mov edx,1
54    mov ecx,newline
55    int 0x80
56
57    ;exit
58    mov eax,1
59    int 0x80
60
61 section .bss
62     num1 resb 1
63     num2 resb 1
64     result resb 1
65
66 section .data
67     msgvar1 db 'Enter first number: '
68     lenmsgvar1 equ $-msgvar1
69
70     msgvar2 db 'Enter second number: '
71     lenmsgvar2 equ $-msgvar2
72
73     msgresult db 'Total value: '
74     lenmsgresult equ $-msgresult
75
76     newline db 0xa,0xd
77
```

76,1 Bot

Referensi

- **Assembly Language Step-By-Step - Programming with Linux**, 3rd edition (Wiley, 2009, 0470497025)
- **Assembly Language for x86 Processors** (Kip R Irvine, 2011)
- **X86-64 Assembly Language Programming with Ubuntu** (Ed Jorgensen, 2018)
- **NASM Assembly Language Programming** (tutorialspoint, 2014)
- **Guide to Assembly Language Programming in Linux** (sivarama p. Dandamundi, 2005)

Referensi

- **Introduction to NASM programming** (Henri Casanova)
- **Introduction to Linux Intel Assembly Language using NASM** (Norman Matloff, 2002)
- **NASM — The Netwide Assembler** (The NASM Development Team, 2012)
- **Guide to Assembly Language A Concise Introduction** (James T. Streib, 2011)

