Freie Universität Berlin

Einführung in NASM

T. Tegethoff, angepasst von L. König

Freie Universität Berlin

Rechnerarchitektur WiSe 2017

Freie Universität Berlin

Grundlagen

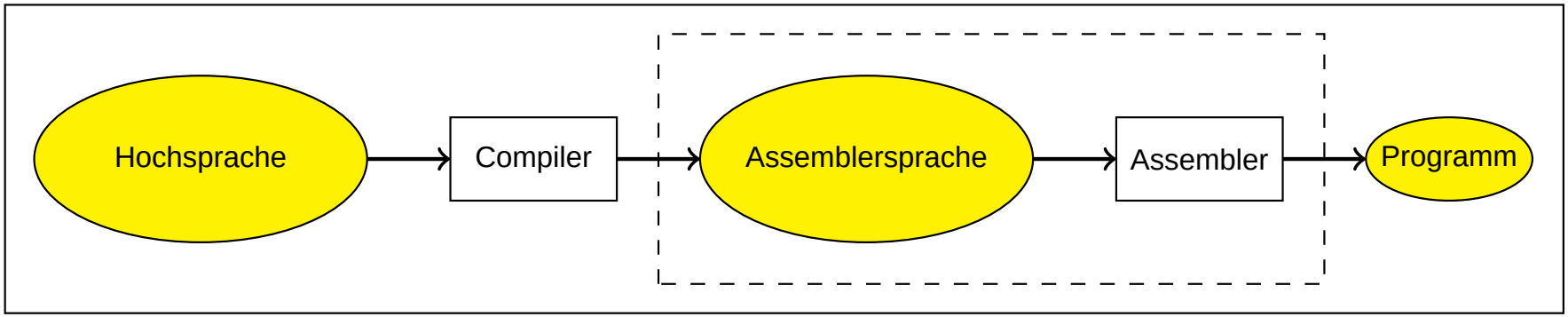
Register

Arithmetik

Compiliern und Ausführen

Anhang

Assembler




```
int main(void) {
  write(
    1,
    "Hallo Welt!\n",
    12
  );
  _exit(0);
}
```

```
main:
mov     edi, 0x2
sub     rsp, 0x8
mov     edx, 0xC
mov     esi, msg
call    write
xor     edi, edi
call    _exit
```

```
main:
bf 01 00 00 00
48 83 ec 08
ba 0c 00 00 00
be 14 06 40 00
e8 b8 ff ff ff
31 ff
e8 a1 ff ff ff
```

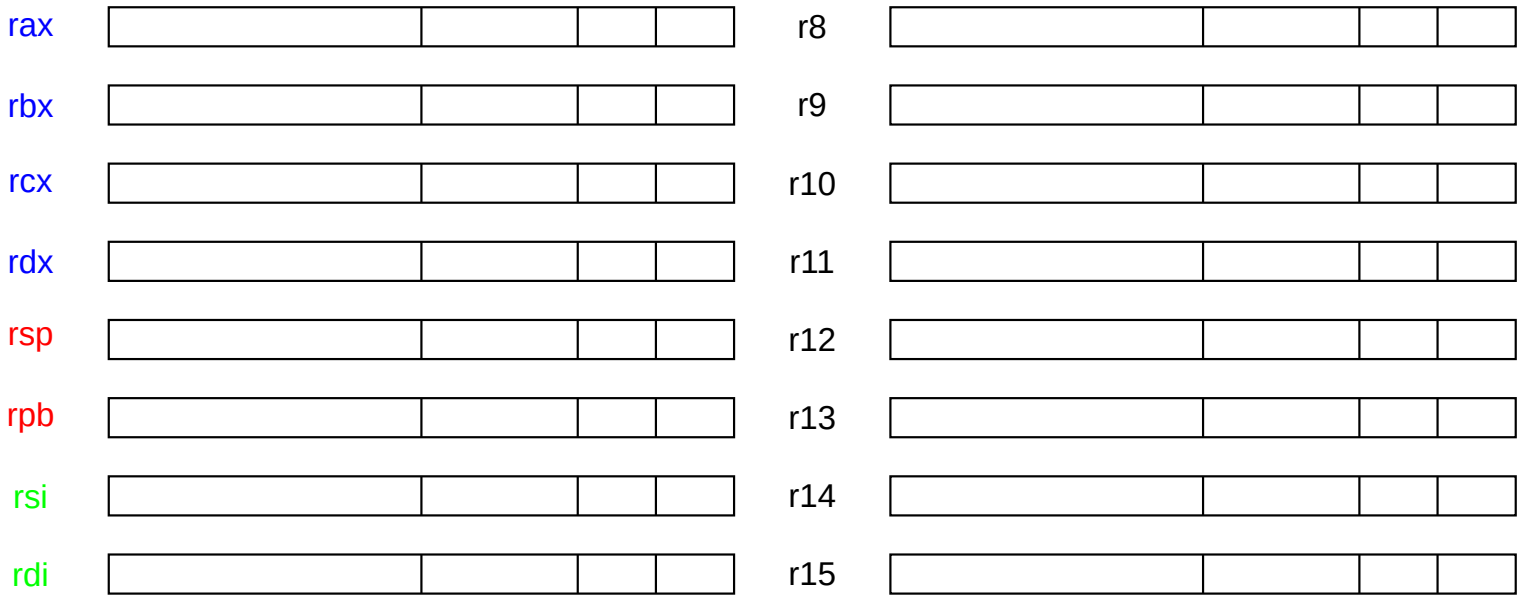
- Assemberbefehle sind direkte Repräsentationen von CPU-Befehlen in einer für Menschen lesbaren Form

NASM

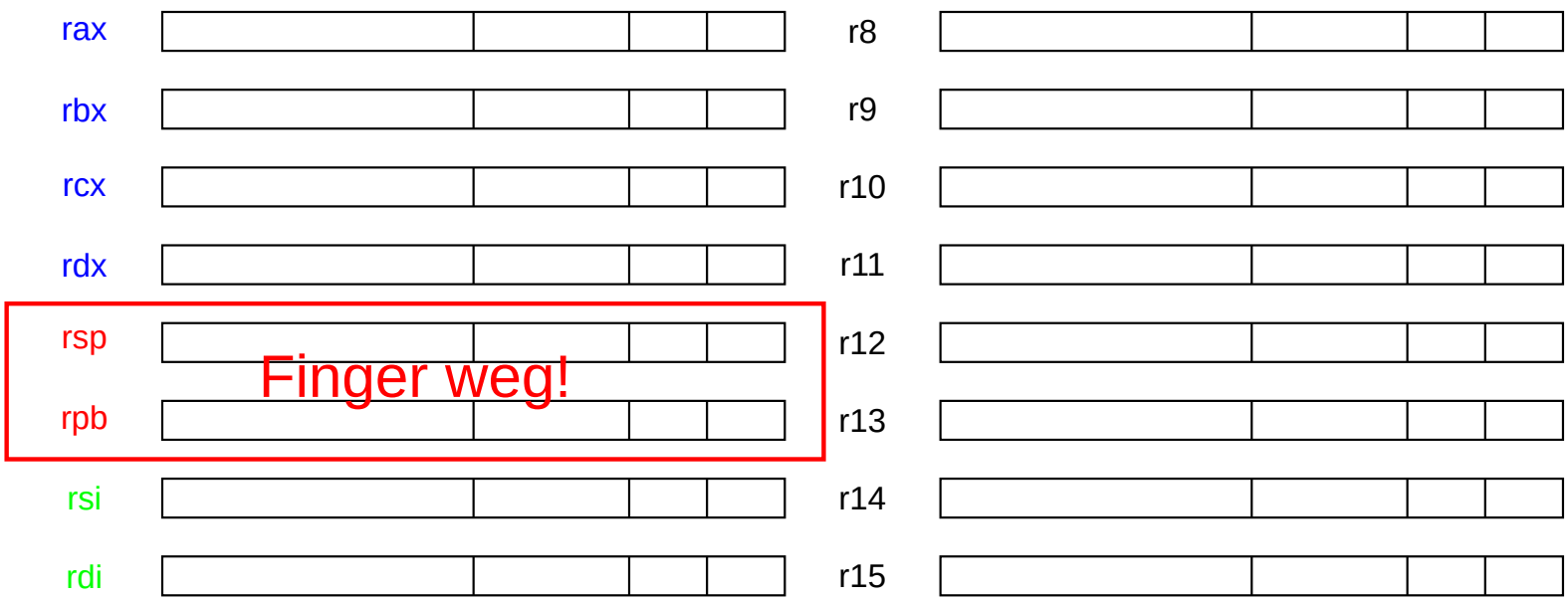
- NASM: Netwide Assembler
- NASM $\rightarrow$ x86-Assembler in Intelsyntax
- GAS $\rightarrow$ x86-Assembler in AT&T-Syntax
- NASM unterstützt 16bit, 32bit und 64bit
- nur 64bit kompatible Abgaben werden akzeptiert!

Freie Universität Berlin

General Purpose Register



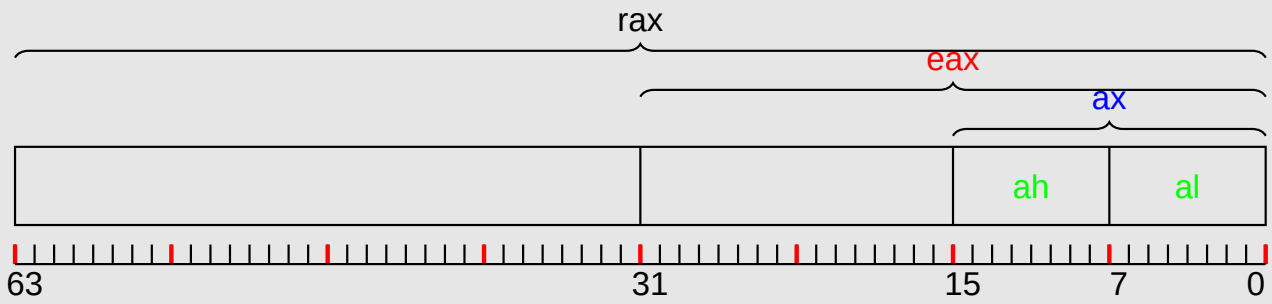
Stackregister



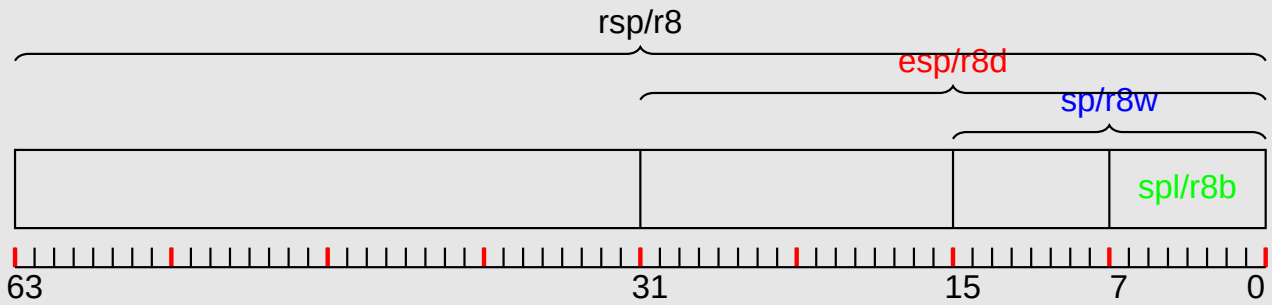
- rsp und rbp werden zum Verwalten des Callstacks benötigt!

Aufbau der Register

rax, rbx, rcx, rdx



rsp, rbp, rsi, rdi und r8 - r15



- wenn ax verändert wird, wird auch rax verändert und umgekehrt!

Spezialregister

rip

- Instruction Pointer
- enthält Adresse des Befehls der als nächtes geladen werden soll
- kann nicht direkt gelesen oder geschrieben werden!

rflags

- enthält das Prozessorstatuswort (PSW)
- PSW besteht z.B. aus JumpFlags, Ausführungsmodus etc.
- Lesen und Schreiben über spezielle Befehle möglich

Verschiebung, Addition und Subtraktion

Grundsätzlicher Aufbau

```
OPERATION [OPERAND1, OPERAND2, OPERAND3]
```

Register und Register

```
MOV rax, rbx ; rax = rbx
ADD rdi, rsi ; rdi = rdi + rsi
SUB r8, r9 ; r8 = r8 - r9
```

Register und Konstante

```
MOV rax, 8 ; rax = 8
ADD rdi, 2 ; rdi = rdi + 2
SUB r10, 3 ; r10 = r10 - 3
```

Freie Universität Berlin

Multiplikation und Division

MUL

- unsigned Multiplikation

```
MUL rbx ; rdx:rax = rax · rbx
```

IMUL

- signed multiplikation

```
IMUL rbx              ; rax = rax · rbx
IMUL rcx, rbx         ; rcx = rcx · rbx
IMUL rcx, rbx, rax    ; rcx = rbx · rax
```

DIV

- unsigned Division

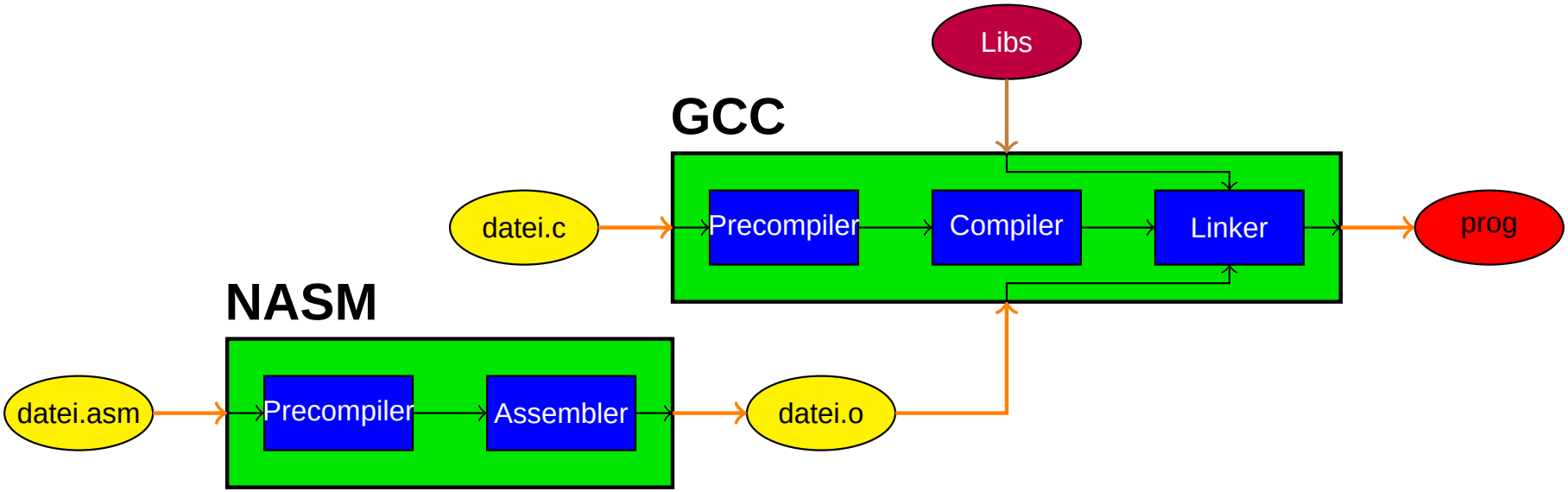
```
DIV rbx ; rax = rdx:rax / rbx , rdx = rest
```

IDIV

- signed Division

```
IDIV rbx ; rax = rdx:rax / rbx , rdx = rest
```

NASM und GCC



Befehle

- nasm -f elf64 <datei.asm> $\rightarrow$ <datei.o>
- gcc -std=c11 -Wall -Wextra -pedantic -O2 -o prog <datei.c> <datei.o> $\rightarrow$ prog
- *Ausführen mit:* ./prog <param1 param2 param3 ...>

Freie Universität Berlin

Bsp. Code

C-Datei

```c
#include <stdlib.h>

extern void doNothing();

int main(void) {
  doNothing();
  return EXIT_SUCCESS;
}
```

NASM-Datei

```
GLOBAL doNothing

SECTION .text

doNothing:
  ; Befehle hier einfügen
  RET
```

Calling Convention

1. Para	2. Para	3. Para	4. Para	5. Para	6. Para	7. Para $\rightarrow \infty$. Para	Return
rdi	rsi	rdx	rcx	r8	r9	Stack	rax

- zusätzlich werden noch r10 und r11 als Volantile betrachtet
- Non-Volantile Register müssen am Ende der Funktion im selben Zustand sein, wie am Anfang!
- d.h. ihr Inhalt muss vor Benutzung gesichert werden!

Weiterführende Links

NASM Webseite
NASM Dokumentation
64bit NASM
Intel Assembly Introduction
Intel Handbücher
x86 Assembly Wikibooks
Runtimebasic Assembler

Böse C-Funktionen

Schlechte Funktionen	Alternativen
`sprintf` → Buffer Overflow!	`snprintf`
`strcat` → Buffer Overflow!	`strncat`
`strcpy` → Buffer Overflow!	
`strncpy` → Bad Behavior!	`memcpy`
`puts` → Unwanted Char!	`printf, fputs`
`gets` → Buffer Overflow!	`fgets`
`scanf, fscanf` → Undefined Behavior!	
`atoi` → deprecated!	
`atol` → deprecated!	`strtol, strtoul`
`atoll` → deprecated!	`strtoll, strtoull`
`atof` → deprecated!	`strtof, strtod, strtold`