Einführung in Assembler

Grundlagen, Register etc.

Thomas Tegethoff

Rechnerarchitektur
Institut der Informatik, FU Berlin
12.-16.10.2015

Grundlagen:

- direkte Übersetzung von Maschinencode in eine für Menschen lesbare Form
- Maschinencode → Zahlen im Speicher
- NASM → x86 Assembler mit Intelsyntax
- GAS → x86 Assembler mit AT&T Syntax

```c
#include <unistd.h>

int main() {
    write(1, "Hello, World!\n", 14);
    _exit(0);
}
```

Abbildung : HalloWelt.c

```
0xba 0x0e 0x00 0x00 0x00 | mov edx, 0xe
0xbe 0xf4 0x05 0x40 0x00 | mov esi, 0x4005f4
0xbf 0x01 0x00 0x00 0x00 | mov edi, 0x1
0xe8 0xc8 0xfe 0xff 0xff | call dword 0x420
0xbf 0x00 0x00 0x00 0x00 | mov edi, 0x0
0xe8 0xae 0xfe 0xff 0xff | call dword 0x410
```

Abbildung : HexDump

Eigenständiges NASM Programm:

```
1  SECTION .DATA
2      msg db  'Hallo Welt!', 0xA, 0
3      len equ $-msg
4
5  SECTION .BSS
6
7  SECTION .TEXT
8  GLOBAL _start
9
10 _start:
11     MOV rax, 1
12     MOV rdi, 1
13     MOV rsi, msg
14     MOV rdx, len
15     SYSCALL
16
17     MOV rax, 60
18     MOV rdi, 0
19     SYSCALL
20
```

Compiliern, Linken und Ausführen:

```
nasm -f elf64 <datei.asm>
ld -s -o <prog_name> <datei.o>
./prog_name
```

Einbinden in C-Datei:

foo.c:

```c
#include <stdio.h>

int foo(int, int, int);

int main()
{
    int i = foo(42,1337,2047);
    printf("i hat den Wert: %d",i);
    return 0;
}
```

foo.asm:

```
section .text
global foo
foo:
    add rdi, rsi
    sub rdi, rdx
    mov rax, rdi
    ret
```

Compiliern und Ausführen:

```
nasm -f elf64 <datei.asm>
gcc -std=c99 <datei.c> <datei.o> -o <prog_name>
./prog_name
```

Verwenden von C-Funktion:

Anzahl	1. Para	2. Para	3. Para	4. Para	5. Para	6. Para	7. Para
1 Para:	rax rdi r11						
2 Para:	rax rdi	rdx rsi r11					
3 Para:	rax rdi	rsi rcx	rdx r11				
4 Para:	rax rdi	rsi	rdx	rcx r11			
5 Para:	rax rdi	rsi	rdx	rcx	r8 r11		
6 Para:	rax rdi	rsi	rdx	rcx	r8	r9 r11	
7 Para:	rax rdi	rsi	rdx	rcx	r8	r9 r11	Stack

- Rückgabewert: Inhalt von ’rax’ von RET aufgerufen wird
- Calling Convention: rdi, rsi, rdx, rcx, r8, r9

General-Purpose Register:

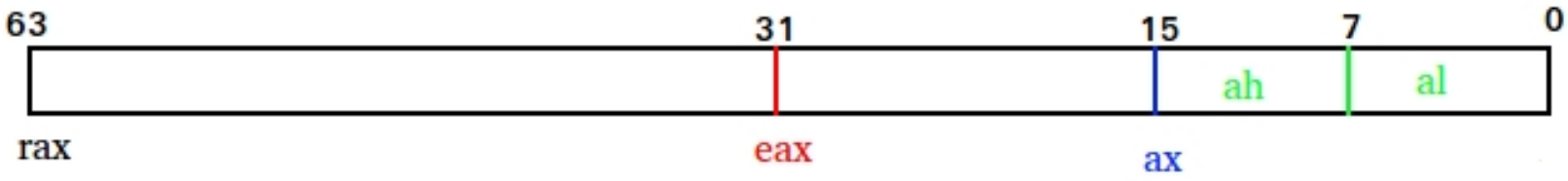
RAX
RBX
RCX
RDX
RSI
RDI
R8
.
.
.
R15

spezielle Register:

RSP - Stack Point
RBP - Stack Basepointer

RIP - Instruction Pointer
FLAGS

Aufbau eines Register:



Assembler Befehle:

OPERATION OPCODE1, OPCODE2, OPCODE3

Register und Register:

```
MOV rax, rbx
ADD rax, rbx
```

Register und Konstante:

```
MOV rax, 8
ADD rax, 2
```

Register und Label:

```
MOV rax, foo
ADD rbx, foo
```

Syscalls:

- spezieller Interupt
- 64bit Befehl: INT 0x21
- Macro: SYSCALL

- Syscall-Nr.: rax
- 1. Parameter: rdi
- 2. Parameter: rsi
- 3. Parameter: rdx
- 4. Parameter: rcx
- 5. Parameter: r8
- 6. Parameter: r9

http://blog.rchapman.org/post/36801038863/linux-system-call-table-for-x86-64

Speicherlabel:

- Pointer (Adressvariable) in RAM-Speicherbereich für Daten
- $\rightarrow$ erste Adresse eines Speicherbereichs
- Zulässig Zeichen: Klein und Großbuchstaben, Ziffern, Unterstrich, Punkt

```
1 SECTION .data
2     msg db 'Hallo Welt', 0xA
3
4 SECTION .bss
5     buf resb 100
6
```

Kommentare:

- mit einem ’;’ (Semikolon)
- oder einem ’#’ (Hashtag/Gatter)
- oder einem ’!’ (Ausrufezeichen)
- oder, oder, oder...

Links:

http://www.nasm.us/
http://www.nasm.us/doc/nasmdo11.html
http://www.nasm.us/doc/nasmdoc3.html
https://software.intel.com/en-us/articles/introduction-to-x64-assembly
http://runtimebasic.net/Assembler:Assembler
https://page.mi.fu-berlin.de/jonascleve/