Fortsetzung Assembler

Speicher, Erweiterte Befehle etc.

Thomas Tegethoff

Rechnerarchitektur
Institut der Informatik, FUB
26.-30.10.2015

Speichereinteilung:

- Callstack:
  - halb-dynamischer Speicherbereich
  - wächst von hohen zu niedrigen Adressen
  - zwei Register zum Ansteuern: RSP und RBP
  - wird benutzt um Sprünge in und aus Unterfunktionen zu steuern
  - spezielle Befehle: CALL, RET, PUSH, POP
- Heap:
  - dynamischer Speicherbereich
  - Speicherreservierungen zur Laufzeit durch Syscalls
  - wächst von niedrigen zu hohen Adressen
- Static:
  - statischer Speicherbereich
  - sections .data & .bss
  - Größe des gewünschten Speichers muss zur Compile-Zeit bekannt sein
- Code:
  - enthält den Bytecode des Programms
  - Steuerung über RIP

Schlüsselwörter für Speicher:

- BYTE - 1byte — 8bit
- WORD - 2byte — 16bit
- DWORD - 4byte — 32bit
- QWORD - 8byte — 64bit

.data

- DB - speichert einen Block in Bytes
- DW - speichert einen Block in Words
- DD - speichert einen Block in DoubleWords
- DQ - speichert einen Block in QuadWords
- DT, DO, DY & DZ - Floats

.bss

- RESB - reserviert Bytes
- RESW - reserviert Words
- RESD - reserviert DoubleWords
- RESQ - reserviert QuadWords
- RESY, RESZ - Floats

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
```

Assembler Anweisungen:

OPERATION OPCODE1, OPCODE2, OPCODE3

Speicher → Prozessor:

```
MOV rax, [foo]
ADD eax, [bar]

MOV rax, [rbx]
ADD  ax, [rcx]
```

Prozessor → Speicher:

```
MOV [foo], rax
ADD [foo], ebx

MOV [rbx], rbx
ADD [rbx], ecx

INC QWORD [foo]
DEC DWORD [rbx]
```

Sprunglabel:

- Pointer (Adressvariable) in RAM-Speicherbereich für Programm
- $\rightarrow$ Adresse für eine Anweisung
- Zulässig Zeichen: Klein und Großbuchstaben, Ziffern, Unterstrich, Punkt

```
 7 SECTION .text
 8 GLOBAL _start
 9
10 _start:
11     ADD rax, rbx
12     loop:
13         SUB rbx, rdx
14         SHR rdx
15     CMP rbx, rax
16     JNE loop
17
```

Sprünge:

- einzige Möglichkeit den Programmfluss zu steuern
- implizieter Schreibvorgang auf RIP
- Beziehen sich immer auf FLAGS
- spezieller Sprung: LOOP

Sichere Art:

```
CMP rax, 0
JNZ _foo

TEST rbx, rax
JL _bar
```

Auch möglich:

```
SUB rcx, r11
JNZ _foo

MUL rbx
JP _bar
```

LOOP:

```
_loop:
...
LOOP _loop

_loop:
...
DEC rcx
JNZ _loop
```

MUL:

- unsigned Multiplikation

MUL rbx

$rdx : rax = rax * rbx$

IMUL:

- signed multiplikation

IMUL rbx

$rax = rax * rbx$

IMUL rcx , rbx

$rax = rcx * rbx$

DIV:

- unsigned Division

DIV rbx

$rax = \frac{rdx:rax}{rbx}$
$rdx = rest$

IDIV:

- signed Division

IDIV rbx

$rax = \frac{rdx:rax}{rbx}$
$rdx = rest$