Register

- Variablen im Prozessor
- Keine explizite Datentypen
- 16 verschiedene General-Purpose-Register
- Zusätzlich u.a. Instruction Pointer (rip)
- 64 Bit groß

64-Bit Register

rax
rcx
rdx
rbx
rsp
rbp
rsi
rdi

r8
r9
r10
r11
r12
r13
r14
r15

Grundlegende Instruktionen

Mnemonic	Beispiel	Pseudocode
MOV dst, src	`mov rax, rdx`	`rax = rdx`
ADD dst, src	`add rax, rdx`	`rax = rax + rdx`
SUB dst, src	`sub rax, rdx`	`rax = rax - rdx`
IMUL dst, src	`imul rax, rdx`	`rax = rax * rdx`

Quiz

Wie kann man `rax` = `rcx` + `rdx` ausrechnen, wenn nur der Wert von `rax` `verändert werden soll`?

```
add rax, rcx
add rax, rdx
```

```
add rcx, rdx
mov rax, rcx
```

```
mov rax, rcx
add rax, rdx
```

Immediates

Mnemonic	Beispiel	Pseudocode
MOV dst, imm	`mov rax, 2`	`rax = 2`
ADD dst, imm	`add rax, 2`	`rax = rax + 2`
SUB dst, imm	`sub rax, 2`	`rax = rax - 2`
IMUL dst, src, imm	`imul rax, rdx, 2`	`rax = rdx * 2`

Immediates

- Mögliche Schreibweisen:
  - `add rax, 122`
  - `add rax, 0x7a`
  - `add rax, 0b1111010`
  - `add rax, 'z'`
- Immediates sind 32 Bit groß, auf 64 Bit sign-extended
  $\rightarrow$ `0x8000 0000` nicht erlaubt, `0xffff ffff 8000 0000` schon
- Wertebereich: $[-2^{31}, 2^{31} - 1]$
- Ausnahme: `MOV` verwendet 64 Bit Immediates

Quiz

Wie kann man `rax = rax + 0xffff ffff` ausrechnen?

☐
```
add rax, 0xffff ffff
```

☐
```
mov rdx, -1
add rax, rdx
```

☐
```
mov rdx, 0xffff ffff
add rax, rdx
```

Flags

- Zero (`ZF`) – gesetzt wenn das Ergebnis 0 ist
  z.B. bei $1 - 1 = 0$ gesetzt
- Sign (`SF`) – gesetzt wenn das Ergebnis negativ ist
  z.B. bei $0 - 1 = -1$ gesetzt
- Carry (`CF`) – gesetzt bei unsigned overflow
  z.B. bei $2^{63} + 2^{63} = 0$ gesetzt
- Overflow (`OF`) – gesetzt bei signed overflow
  z.B. bei $(2^{63} - 1) + 1 = -2^{63}$ gesetzt

Condition Codes

- e – equal
- ne – not equal
- l – signed less
- le – signed less or equal
- g – signed greater
- ge – signed greater or equal
- b – below (unsigned less)
- be – below or equal
- a – above (unsigned greater)
- ae – above or equal

CMP

- Instruktion um Werte zu vergleichen
- Funktioniert wie `SUB`, verwirft aber das Ergebnis
- `cmp rax, rdx`
  Berechnet `rax−rdx` und setzt Flags
- `cmp rax, 5`
  Berechnet `rax−5` und setzt Flags

Label

- Markieren eine Adresse
- Meistens für Sprünge verwendet
- Globales Label
  - Markiert Anfang von Funktionen, globale Variablen, Konstanten
  - z.B. `quicksort:`
- Lokales Label
  - Label innerhalb einer Funktion
  - z.B. `.Lloop:`

Sprungbefehle: JMP/Jcc

- Unbedingter Sprung
  - `JMP label`
  - Springt zu einem Label

- Bedingter Sprung
  - `Jcc label`
  - *cc* wird durch Condition Code ersetzt
  - Springt nur, wenn Condition erfüllt ist

- Beispiel JMP:

```
1   jmp .Lskip
2   mov rax, 0
3 .Lskip:
4   add rax, 1
```

- Beispiel *Jcc*:

```
1   cmp rax, rdx
2   je .Lskip
3   mov rax, 0
4 .Lskip:
5   add rax, 1
```

IF-Bedingungen

- Pseudocode:

```
1 if (rax >= 100)
2     rax -= 1;
3 else
4     rax += 1;
```

- Assembly:

```
1   cmp rax, 100
2   jge .Lsub
3   add rax, 1
4   jmp .Lend
5 .Lsub:
6   sub rax, 1
7 .Lend:
8   ...
```

Quiz

Wie kann man vorzeichenbehaftet `rax` = `max(0, rax)` berechnen?

```
  cmp rax, 0
  ja .Lskip
  mov rax, 0
.Lskip:
```

```
  cmp rax, 0
  jl .Lskip
  mov rax, 0
.Lskip:
```

```
  cmp rax, 0
  jg .Lskip
  mov rax, 0
.Lskip:
```

Quiz

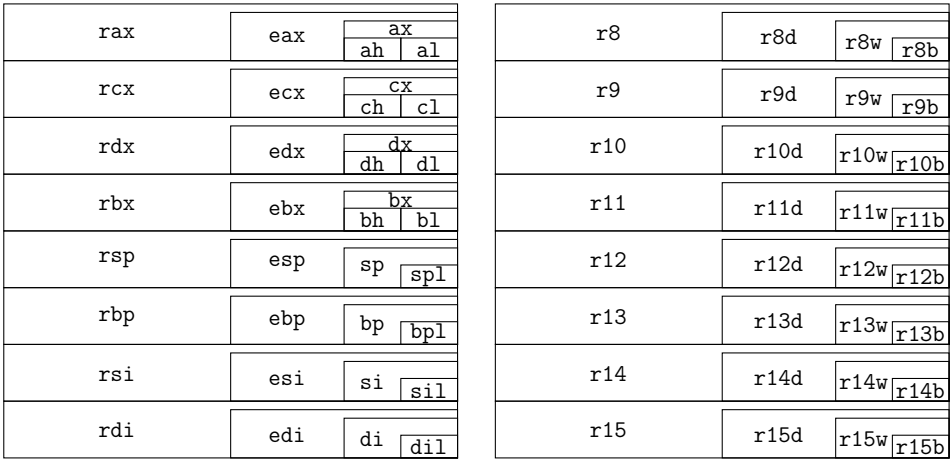
Nach Ausführen welcher Schleife ist der Wert in `rax` = 100?

```
  mov rax, 0
.Lloop:
  add rax, 10
  cmp rax, 100
  jne .Lloop
```

```
  mov rax, 0
  mov rdx, 10
.Lloop:
  add rax, 10
  sub rdx, 1
  cmp rdx, 0
  jge .Lloop
```

```
  mov rax, 0
  mov rdx, 0
.Lloop:
  cmp rdx, 10
  jge .Lend
  add rax, 10
  add rdx, 1
  jmp .Lloop
.Lend:
```

Teilregister



Quiz

Was passiert wenn man in das Register `ax` schreibt?

- ☐ Die unteren 32 Bit von `rax` werden beschrieben, die oberen 32 Bit bleiben unverändert
- ☐ Die unteren 32 Bit von `rax` werden beschrieben, die oberen 32 Bit werden auf 0 gesetzt
- ☐ Die unteren 16 Bit von `rax` werden beschrieben, die oberen 48 Bit bleiben unverändert
- ☐ Die unteren 16 Bit von `rax` werden beschrieben, die oberen 48 Bit werden auf 0 gesetzt

Quiz

Was steht nach `mov eax, 0x8000 0000` in `rax`?

- ☐ `0xffff ffff 8000 0000`
- ☐ `0x0000 0000 8000 0000`
- ☐ `0x8000 0000 0000 0000`
- ☐ Was in den oberen 32 Bit steht ist unbekannt

Quiz

Was steht nach `mov al, 0x80` in `rax`?

- ☐ `0xffff ffff ffff ff80`
- ☐ `0x0000 0000 0000 0080`
- ☐ `0x0000 0080 0000 0000`
- ☐ Was in den oberen 56 Bit steht ist unbekannt

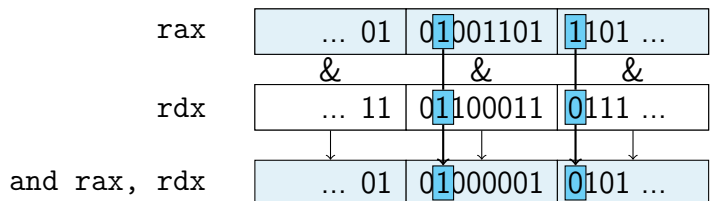
Quiz

Was passiert bei `mov eax, eax`?

- ☐ Der Wert in rax wird auf 0 gesetzt
- ☐ Die oberen 32 Bit von rax werden auf 0 gesetzt
- ☐ Der Wert in rax bleibt unverändert
- ☐ Der Wert in eax bleibt unverändert
- ☐ Der Wert von eax wird auf rax vorzeichenlos erweitert

Bitweise Operationen

- Beispiel: Bitweises AND



Bitweise Operationen

Mnemonic	Beispiel	Pseudocode
AND dst, src	`and rax, rdx`	`rax = rax` $\wedge$ `rdx`
OR dst, src	`or  rax, rdx`	`rax = rax` $\vee$ `rdx`
XOR dst, src	`xor rax, rdx`	`rax = rax` $\oplus$ `rdx`
NOT dst	`not rax`	`rax =` $\neg$`rax`

Bitweise Operationen

Mnemonic	Beispiel	Pseudocode
AND dst, imm	`and rax, 2`	`rax = rax` $\wedge$ `2`
OR dst, imm	`or  rax, 2`	`rax = rax` $\vee$ `2`
XOR dst, imm	`xor rax, 2`	`rax = rax` $\oplus$ `2`

Quiz

Was steht nach `xor rax, rax` in `rax`?

- ☐ Das Ergebnis von `rax` $\oplus$ `rax`
- ☐ 0x0000 0000 0000 0000
- ☐ 0xffff ffff ffff ffff

Quiz

Was passiert bei `xor eax, eax`?

- ☐ `eax` enthält das Ergebnis von `eax` $\oplus$ `eax`
- ☐ `rax` wird auf 0 gesetzt
- ☐ Die obere Hälfte von `rax` bleibt unverändert

Multiplikation

Mnemonic	Beispiel	Pseudocode
IMUL dst, src	`imul rax, rcx`	`rax = rax · rcx`
IMUL dst, src, imm	`imul rax, rcx, 2`	`rax = rcx · 2`
IMUL src	`imul rcx`	`rdx:rax = rax · rcx` (signed)
MUL src	`mul rcx`	`rdx:rax = rax · rcx` (unsigned)

Multiplikation

Mnemonic	Beispiel	Pseudocode
IMUL src	`imul ecx`	`edx:eax = eax · ecx`
IMUL src	`imul cx`	`dx:ax = ax · cx`
IMUL src	`imul cl`	`ax = al · cl`

Quiz

Wie kann man die vorzeichenlosen Werte in `eax` und `edx` multiplizieren, sodass das volle 64-Bit Ergebnis in `rax` steht (angenommen die oberen 32 Bit von `rax` und `rdx` sind 0)?

```
mul  edx
add  rax, rdx
```

```
mul  edx
mov rcx, 0x100000000
imul  rdx, rcx
add  rax, rdx
```

```
imul  rax, rdx
```

Division

Mnemonic	Beispiel	Pseudocode
IDIV src	`idiv rcx`	`rax = rdx:rax / rcx` (signed) `rdx = rdx:rax % rcx`
DIV src	`div rcx`	`rax = rdx:rax / rcx` (unsigned) `rdx = rdx:rax % rcx`

- Divide Exception falls:
  - Dividend 0
  - Ergebnis lässt sich nicht im Zielregister darstellen
- Divide sehr teuer

Quiz

Wie kann man vorzeichenlos `rax = rcx / 7` berechnen?

```
div rax, rcx, 7
```

```
mov rax, rcx
mov rdi, 7
div rdi
```

```
mov rax, rcx
mov rdi, 7
mov rdx, 0
div rdi
```

Quiz

Wie kann man vorzeichenbehaftet `rdx` = `rdx:rax` % 5 berechnen?

```
idiv 5
```

```
mov rcx, 5
div rcx
```

```
mov rcx, 5
idiv rcx
```

Quiz

Wie kann man vorzeichenlos `rax` = `rdx:rax` % 8 berechnen?

```
and rax, 7
```

```
mov rcx, 8
div rcx
mov rax, rdx
```

```
mov rcx, 8
idiv rcx
mov rax, rdx
```

Wichtige Instruktionen im Überblick

Mnemonic	Beispiel	Pseudocode
MOV dst, src	`mov rax, rdx`	`rax = rdx`
MOV dst, imm	`mov rax, 2`	`rax = 2`
ADD dst, src	`add rax, rdx`	`rax = rax + rdx`
ADD dst, imm	`add rax, 2`	`rax = rax + 2`
SUB dst, src	`sub rax, rdx`	`rax = rax - rdx`
SUB dst, imm	`sub rax, 2`	`rax = rax - 2`
MUL src	`mul rcx`	`rdx:rax = rax` $\cdot$ `rcx` (unsigned)
IMUL src	`imul rcx`	`rdx:rax = rax` $\cdot$ `rcx` (signed)
IMUL dst, src	`imul rax, rdx`	`rax = rax * rdx`
IMUL dst, src, imm	`imul rax, rdx, 2`	`rax = rdx * 2`
CMP src, src	`cmp rax, rdx`	Vergleicht `rax` mit `rdx`
JMP label	`jmp .Lskip`	Springt zum Label `.Lskip`
Jcc label	`je .Lskip`	Bedingter Sprung zu `.Lskip`
AND dst, src	`and rax, rdx`	`rax = rax` $\wedge$ `rdx`
OR dst, src	`or rax, rdx`	`rax = rax` $\vee$ `rdx`
XOR dst, src	`xor rax, rdx`	`rax = rax` $\oplus$ `rdx`
NOT dst	`not rax`	`rax` = $\neg$`rax`