Computer Arkitektur og Operativ Systemer

Denne forelæsning optages og gøres efterfølgende tilgængelig på Moodle
MEDDEL VENLIGST UNDERVISEREN, HVIS DU IKKE ØNSKER, AT OPTAGELSE FINDER STED

This lecture will be recorded and afterwards be made available on Moodle
PLEASE INFORM THE LECTURER IF YOU DO NOT WANT RECORDING TO TAKE PLACE

Computer Arkitektur og Operativ Systemer

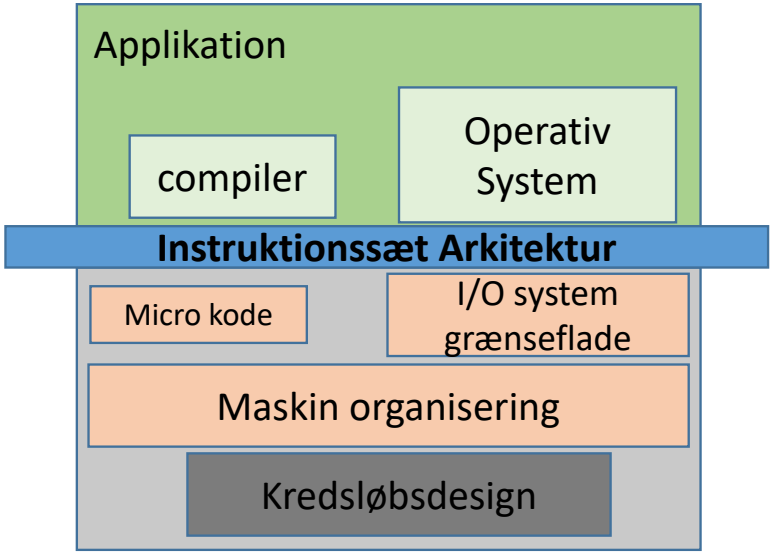
Assembler 1

Kursusgang 3
Brian Nielsen

Credits to
Randy Bryant & Dave O'Hallaron (CMU)

Hvad er en ISA ?

- ISA-Arkitektur: (instruction set architecture)
  De dele af processorens design, som man skal forstå for at kunne læse/skrive assembler og maskinkode.
  - Instruktioner? Ordlængde? Registre? Indkodning?, mv.

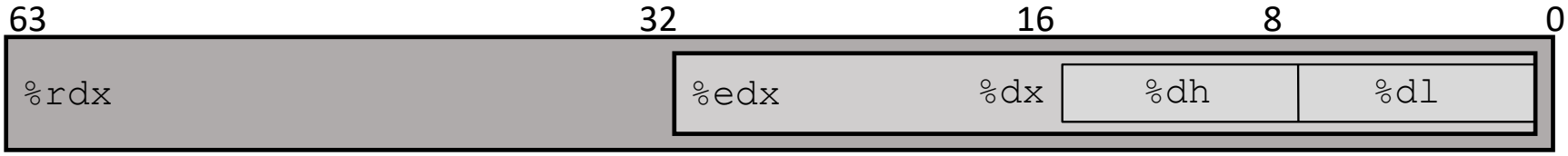


- Hvordan ser programmer ud på ISA-niveau?

> Those who say *"I understand the general principles, I don't want to bother with the details"* are eluding themselves*!*
> *(CSPP3, p201)*

Hvad er registre?

- Et lille stykke lager internt i en processer-kerne
  - Lagring af midlertidige variable og resultater
  - Rasende hurtig adgang, men få
  - I X86-64: 16 almene registre a 64 bits ord! Fx **`%rdx`**
  - ”Virtuelle registre”, fysisk delmængder af 64 bit registrene
- Modsat: primær ”hukommelsen”
  - maskinens RAM, Giga-bytes, externt til processoren, meget langsom ifht. processoren.



Hvis %rdx indeholder ordet `0xFFFFFFFFEEEEDDCC, så har`

- `%edx:0xEEEEDDCC`
- `%dx: 0xDDCC`
- `%dh: 0xDD`
- `%dl: 0xCC`

`movb` `$0x11,%dh, vil %rdx indeholde 0xFFFFFFFFEEEE11CC`

`movl` `$0x11,%edx,vil %rdx indeholde 0x0000000000000011` !64-bit finte (aside p220)

Hvordan laver vi variable, assignments, og udtryk i ASM?

- Variable findes kun som ”ord”, som er gemt i hukommelsen eller et register
  - Identificeres vha. den adresse ordet starter på,
  - eller vha. registernavn
- **`move`** instruktionerne bruges til at
  - Kopiere et ord fra hukommelse til register, eller tilbage
  - Kopiere et ord imellem 2 registre
  - Ord findes i forskellige størrelse 8,16,32,64 bits
- **`lea`** instruktionerne bruges til at beregne en adresse på et ord

Hvordan laver vi aritmetik?

- En række dedikerede instruktioner

```
add     Src,Dest    Dest = Dest + Src
…
sar     Src,Dest    Dest = Dest >> Src      //Aritmetisk højre skifte,…
```

- **`lea`** bruges også til at lave simple regne-stykker i én instruktion
- Shift instruktionerne anvendes som en form for billig multiplikation

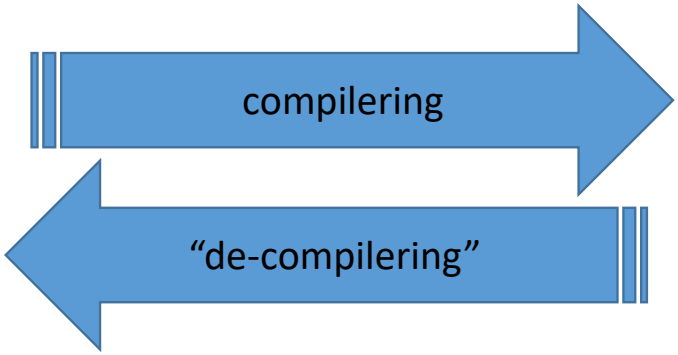
- Komplekse udtryk brydes op i mange instruktioner, bruger mange registre (evt. hukommelsen) til at gemme mellem resultater.

Fra C-til Assembler og tilbage

- Bogen anvender en vis metodik som kan bruges til lettere at forstå
  1. Hvordan compileren laver assembler
  2. Forstå assembler som C-kode.

Idag: data I register kan opfattes som temporære variable i C

```
long t, long *dest;
…
*dest = t;
long t2=t;
```




```
#dest in %rbx
#t in %rax
movq %rax, (%rbx)
movq %rax, %rsi
```

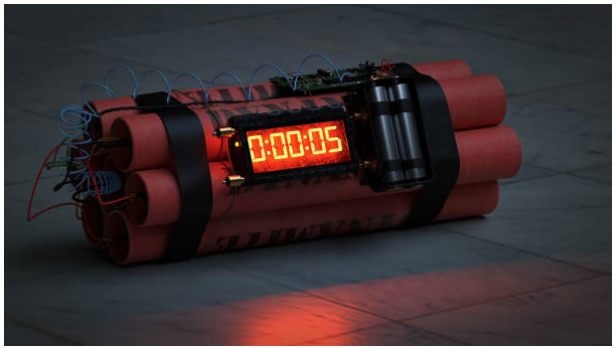
```
long absdiff
  (long x, long y)
{
  long result;
  if (x > y)
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```
long absdiff_j
  (long x, long y)
{
    long result;
    long ntest = x <= y;
    if (ntest) goto Else;
    result = x-y;
    goto Done;
 Else:
    result = y-x;
 Done:
    return result;
}
```

```
#%rdi=x, %rsi=y
#%rax=result
absdiff:
   cmpq     %rsi, %rdi   # x:y
   jle      .L4
   movq     %rdi, %rax
   subq     %rsi, %rax
   ret
.L4:        # x <= y
   movq     %rsi, %rax
   subq     %rdi, %rax
   ret
```

Øvelserne

- Adressering af og indlæsning fra hukommelsen
- Kan vi læse og forstå ASM?
- De-kompilering (fra Asm til C)
  - Tildelinger og sammensatte udtryk
- Bitshift, aritmetik med bit-shift
- Challenge 2: afmontering af en binær-bombe!!



Brug Hjælpelærer assistancen!