

## Den siste dagen

- ▶ En oppklaring fra forrige uke
- ▶ Pensumoversikt
- ▶ Eksamen
- ▶ Selvmodifiserende kode
- ▶ Overflyt
- ▶ Veien videre...

## Problem 1: Hvordan sjekket 4 byte

Hvordan sjekke raskt om minst én av fire byte er 0.

### Løsning

$$(((x - 0x01010101) \& \sim x) \& 0x80808080) == 0$$

1. Ved  $-1$  vil 0 skifte fortegn til 1, andre verdier ikke.
2. Ved  $\&\sim x$  vil de bit som skiftet  $0 \rightarrow 1$ , bli 1, de øvrige 0.
3. Ved siste  $\&$  nuller vi alle bit unntatt fortegnsbite-ene.

Når man snubler i en forklaring...

$$\boxed{(((x - 0x01010101) \& \sim x) \& 0x80808080) == 0}$$

$$00000000 - 00000001 \rightarrow 11111111 \& 11111111 \rightarrow \\ 11111111 \& 10000000 \rightarrow 10000000 \neq 0$$

$$0xxxxxxx - 00000001 \rightarrow 0yyyyyyy \& 1\overline{xxxxxxx} \rightarrow \\ 0zzzzzzz \& 10000000 \rightarrow 00000000 = 0$$

$$10000000 - 00000001 \rightarrow 01111111 \& 01111111 \rightarrow \\ 01111111 \& 10000000 \rightarrow 00000000 = 0$$

$$1xxxxxxx - 00000001 \rightarrow 1yyyyyyy \& 0\overline{xxxxxxx} \rightarrow \\ 0zzzzzzz \& 10000000 \rightarrow 00000000 = 0$$

# Oppsummering

- ▶ **Pensum**

Grovt sett er alt fra kompendiet, forelesningene og øvingsoppgavene pensum.

- ▶ Hovedtanker fra kurset

- ▶ **Litt om eksamen**

Hvorfor har dere tatt dette emnet?

## Hvorfor har dere lært dette?

Ikke mange programmerer assemblerkode i dag, men

- ▶ noen gjør det
- ▶ noen må jobbe med kompilatorer som lager slik kode
- ▶ kjennskap til slik kode er en forutsetning for å forstå hvordan datamaskinen er bygget opp og fungerer
- ▶ kjennskap til maskinkode gjør oss til bedre programmere i høynivåspråk.

Vi har lært Intel x86-kode, men *prinsippene* er de samme for alle prosessorer.

Kjernekunnskapen!

## Hovedtanke

Glem aldri!

**Alt er *bit*!**

Når kode også er bit

## Selvmodifiserende kode

Når programkode lagres som bit-mønstre, kan man da la programmet endre på seg selv?

```
1          .globl teller
2          .data
3 0000 55          teller: pushl %ebp
4 0001 89E5        movl %esp,%ebp
5
6 0003 B8010000    movl $1,%eax
6          00
7 0008 83050400    addl $1,teller+4
7          000001
8
9 000f 5D          popl %ebp
10 0010 C3          ret
```

Denne funksjonen returnerer 1 første gang den kalles. Samtidig endres instruksjonen slik at den vil gi 2 neste gang den utføres.



- ▶ Koden er plassert i .data for å kunne endres.
- ▶ På noen maskiner vil det kunne bli rot med data- og instruksjons-cache.

## Konklusjon

Det er morsomt at det går an, men slik kode kan neppe kalles hverken lettlest eller trygg.



Hva når vi ikke har bit nok?

## Overflyt

Dette programmet

```
#include <stdio.h>

int main (void)
{
    signed char v = 100;
    v = v + v;
    printf("v = %d\n", v);
    return 0;
}
```

gir galt svar:

$v = -56$

$200 = 11001000_2 = -56$

Feilen skyldes overflyt. Hva kan man gjøre med slikt?

- ▶ Strutseteknikken
- ▶ Sjekke data før operasjonen
- ▶ Sjekke etter operasjonen



Hva når vi ikke har bit nok?

## Heltall *uten* fortegns-bit

For addisjon og subtraksjon vil **C**-flagget bli satt ved overflyt.

Ved multiplikasjon blir det aldri problemer siden svaret kommer med dobbelt så mange bit.

Ved divisjon kan det bli et avbrudd!

```
ovfl:    .globl    ovfl
         movl    mill,%eax
         imull   mill
         idivl   ti
         ret
```

```
         .data
mill:    .long    1000000
ti:      .long    10
```

gir

Floating point exception



Hva når vi ikke har bit nok?

## Heltall *med* fortegns-bit

Ved addisjon og subtraksjon kan man bruke **O**-flagget som settes ved overflyt. Nærmere bestemt settes det når

1. begge operandene har likt fortegns-bit *og*
2. resultatet har motsatt fortegns-bit.

Multiplikasjon og divisjon er som for tall uten fortegns-bit.

## Konklusjon

Ved behov kan man sjekke på overflyt i assemblerprogrammering.

## Hvis dere likte kurset . . .

INF2100 Programmeringslaboratorium

INF3110 Programmeringsspråk

INF3151 Operativsystemer (20 stp)

INF5110 Kompilorteknikk

INF1400 Digital teknologi

INF1411 Elektroniske systemer

INF3400 Digital mikroelektronikk

INF3410 Analog mikroelektronikk

INF3430 Digital systemkonstruksjon

## Forventet kunnskap til eksamen

- ▶ oppbygningen av datamaskiner (registre, flagg, minne, stakk, ...)
- ▶ bit-lagring av instruksjoner, heltall (av ulike størrelser,  $\pm$  fortegns-bit), flyt-tall, C-tekster, ...
- ▶ notasjon for assemblerkode (registre, konstanter, adresser, ...)
- ▶ instruksjoner for dataflytting og -testing, regning, masking, hopp ( $\pm$  testing), skifting/rotasjoner, ...
- ▶ implementasjon av datastrukturer (vektorer, lister, C-tekster, struct-er, union-er ...)
- ▶ funksjoner og kall på disse
- ▶ overflyt, selvmodifiserende kode



# Eksamen

Eksamen er **10. juni 2010** klokken 14.30-17.30.

## Hjelpemidler

- ▶ *Alle* trykte og skrevne hjelpemidler er tillatt.
- ▶ Batteridrevet lommekalkulator er lov — og kan være nyttig.

## Hva slags oppgaver kan forventes fra meg?

- ▶ Programmering i C og assembler, for eksempel oversettelse den ene eller andre veien.
- ▶ Finne ut hva et program gjør.

## Hva er viktig?

- ▶ Overbevise sensor om at man har skjønt *ideen* bak stoffet.
- ▶ Gode *korte* kommentarer.
- ▶ Enkel kode.

## Hva er ikke så viktig?

- ▶ Syntaksdetaljer
- ▶ Rask kode



## Sjekk godkjenningslister

- ▶ Besøk <https://www.sifi.uio.no> og sjekk godkjentlisten for kurset for å se din status.
- ▶ Ta kontakt med gruppelærer (eller meg) hvis du mener det er gjort feil.
- ▶ I siste instans: Møt opp på eksamensdagen og krev å få avlegge eksamen.

## Problemer under eksamen?

Eksamensdagen vil faglærerne gå runde i eksamenslokalet omtrent klokken 15.30-16.00. Da er det anledning til å stille spørsmål om *alt*.

## Bruk tiden riktig

Det er mye bedre å svare litt på alle spørsmålene enn perfekt på noen og intet på andre.

**Forslag:** Bruk så mye av tiden som oppgaven teller (dvs bruk 25% av tiden på en oppgave som teller 25%).

*Lykke til!*