CHALMERS
UNIVERSITY OF TECHNOLOGY

Digital- och datorteknik

Föreläsning #18

Biträdande professor Jan Jonsson

Institutionen för data- och informationsteknik
Chalmers tekniska högskola

CHALMERS
UNIVERSITY OF TECHNOLOGY

Assemblerprogrammering

Assemblatorer vs kompilatorer

En assemblator är ett program som översätter program från assemblerspråk till maskinspråk.

Assemblerspråket gör det möjligt att skriva program där vi har fullständig kontroll över den maskinkod som hamnar i minnet.

För en given processor har assemblerspråk och assemblator vanligen konstruerats tillsammans. Oftast är de konstruerade av processortillverkaren själv.

En kompilator är ett program som översätter program från ett högnivåspråk (t ex C, C++, Ada) till maskinspråk.

Ofta fungerar assemblerspråket som ett mellansteg vid översättningen från högnivåspråk till maskinspråk.

CHALMERS
UNIVERSITY OF TECHNOLOGY

Assemblerprogrammering

Varför programmera i assemblerspråk?

Situationer när det finns anledning att skriva program i processorns assemblerspråk:

- När det inte finns någon kompilator, som genererar kod, för den aktuella processorn.
- När det finns hårdvarunära uppgifter som inte går att lösa i ett högnivåspråk.
- När någon uppgift är så tidskritisk att en kompilator inte klarar av att göra koden så att den klarar tidskraven.

Övning i att programmera i assemblerspråk ger även förståelse för hur program i högnivåspråk verkligen realiseras som maskinprogram, d v s i maskinspråket.

CHALMERS
UNIVERSITY OF TECHNOLOGY

Assemblerprogrammering

Tvåpass-assemblatorn

Assemblatorn går igenom assemblerprogrammet från början till slut två gånger (i två pass).

Arbetssättet beror huvudsakligen på att assemblatorn först måste bestämma värdet hos samtliga symboler (pass 1) innan den genererar maskinkoden (pass 2).

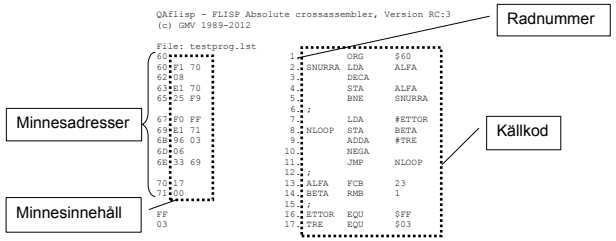
Resultatet av pass 2 blir två filer (i ASCII-format):

- En listfil med en kombination av programtexten och maskinkoden.
- En objektfil med det assemblerade programmets maskinkod och motsvarade minnesadresser.

CHALMERS

Assemblerprogrammering

Assemblator – listfil (.lst)



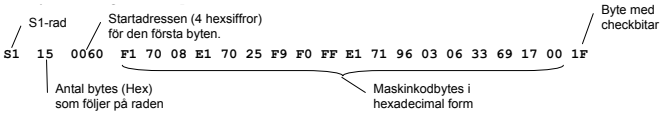
Listfilen används i första hand för dokumentation och är mycket användbar när man skall felsöka program.

CHALMERS

Assemblerprogrammering

Assemblator – objektfil (.s19)

```
S1150060F17008E17025F9F0FFE171960306336917001F
S90300609C
```



Objektfilen används för laddning av programkoden till minnet, antingen i en simulator eller i den verkliga datorn.

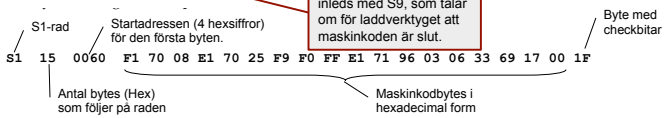
CHALMERS

Assemblerprogrammering

Assemblator – objektfil (.s19)

```
S1150060F17008E17025F9F0FFE171960306336917001F
S90300609C
```

Slutraden i en objektfil inleds med S9, som talar om för laddverktyget att maskinkoden är slut.



Objektfilen används för laddning av programkoden till minnet, antingen i en simulator eller i den verkliga datorn.

CHALMERS

Variabler

Högnivåspråk vs assemblerspråk

En variabeldeklaration används i högnivåspråk för att reservera minnesutrymme. Deklarationen anger ett unikt symbolnamn och för korrekt behandling i uttryck ges den också en datatyp.

`char` är en datatyp som, i programspråket ”C”, implementeras med 8 bitar. Med tilläggen unsigned respektive signed anges vilken talrepresentation som gäller för datatypen.

Talområdet för en `unsigned char` är [0 .. 255], medan talområdet för en `signed char` blir [-128 .. 127].

Exempel: variabeldeklarationen

```
unsigned char Counter;
```

anger att en byte ska reserveras för symbolen `Counter`, och att byten skall behandlas som ett 8-bitars tal utan tecken.

CHALMERS

Variabler

Högnivåspråk vs assemblerspråk

I assemblerspråk kan inga datatyper uttryckligen anges, utan det är programmeraren som måste göra lämpliga antaganden, t ex vad gäller den talrepresentation som används.

En global variabel kan refereras av alla delar av programmet oavsett om det gäller huvudprogrammet eller någon subrutin. För deklaration av en global variabel ska därför något av direktiven `RMB`, `FCB` eller `FCS`, användas tillsammans med variabelns (unika) symbolnamn.

Exempel: variabeldeklarationen

```
Counter RMB 1
```

anger att en byte ska reserveras för symbolen `Counter`. Observera att direktivet RMB inte innehåller någon information om datatyp.

CHALMERS

Variabler

Högnivåspråk vs assemblerspråk

I assemblerspråk kan inga datatyper uttryckligen anges, utan det är programmeraren som måste göra lämpliga antaganden, t ex vad gäller den talrepresentation som används.

En lokal variabel kan bara refereras av den subrutin där den deklareras. Någon speciell deklaration av en lokal variabel finns inte, utan de skapas genom att utrymme reserveras på stacken eller genom att man reserverar ett av processorns register.

Exempel: assemblerinstruktionen

```
LEASP -1,SP
```

anger att en byte ska reserveras för en lokal variabel. Observera att det här inte finns någon information om symbolnamn eller datatyp för den lokala variabeln, utan endast dess placering på stacken blir känd.

CHALMERS

Variabler

Högnivåspråk vs assemblerspråk

I assemblerspråk kan inga datatyper uttryckligen anges, utan det är programmeraren som måste göra lämpliga antaganden, t ex vad gäller den talrepresentation som används.

Vid subrutinanrop används ofta speciella lokala variabler som kallas parametrar (indata till subrutinen) respektive returvärde (utdata från subrutinen).

Någon speciell deklaration av parametrar och returvärden finns inte, utan de skapas genom att utrymme reserveras på stacken (med `LEASP`) och/eller genom att man reserverar ett eller flera av processorns register för detta ändamål.

CHALMERS

Assemblerprogrammering

Demonstrationsexempel 1 – sökning i tabell

En tabell med fem olika 8-bitars tal utan tecken lagras med början på adress 18_{16} i minnet. Skriv en subrutin FndMax som letar upp det största talet i tabellen. Av de interna registren får subrutinen endast påverka A- och Y-registren.

Värdet på det största talet skall returneras i A-registret.

Adressen för det största talet skall returneras i Y-registret.

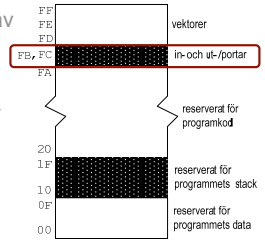
CHALMERS

In- och utportar

Primärminnets användning

Den rekommenderade användningen av primärminnet i FLIS-processorn är:

- Adress 20_{16} – FA_{16}: Programkod
- Adress 00_{16} – $0F_{16}$: Globala variabler (data med relativt lång livslängd)
- Adress 10_{16} – $1F_{16}$: Programstack (data med relativt kort livslängd)
- Adress FB_{16} – FC_{16}: In- och ut-portar (reserverat för kommunikation med sensor och/eller ställdon)
- Adress FD_{16} – FF_{16}: Vektorer (reserverat för lagring av speciella hoppadresser)



CHALMERS

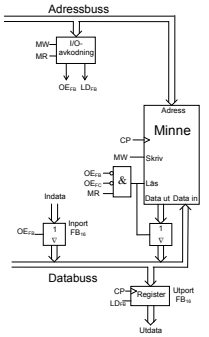
In- och utportar

Minnesavbildade I/O-portar

För att kommunicera med omvärlden behöver datorn tillgång till in- och utportar (I/O-portar), d v s dataregister som ansluter till yttre enheter som tangentbord, display, givare, etc.

I sammanhang där bara enstaka IO-portar behövs kan man använda en överlagringsteknik där en IO-port aktiveras i samma adressrum som någon minneskrets.

FLIS-datorn har två I/O-portar, på minnesadresserna FB_{16} resp. FC_{16}.



CHALMERS

In- och utportar

Minnesavbildade I/O-portar

En komplikation med I/O-portar jämfört med variabler i minnet är att en utport inte nödvändigtvis behöver vara läsbar. Man kan alltså vid skrivning till en port inte kan förvänta sig att det skrivna värdet är det som man får vid läsning av samma port.

I de fall man bara vill ändra enstaka bitar på en icke läsbar utport kan man införa en så kallad "skuggvariabel" vars syfte är att innehålla exakt samma värde som utporten hade haft om den varit läsbar. Bitmanipulation kan då först göras på skuggvariabeln, därefter kopieras skuggvariabelns värde till den icke läsbara utporten.

CHALMERS

In- och utportar

Minnesavbildade I/O-portar

```
        ORG 0
Shadow  RMB 1
        ORG $20
        …
; ettställ bit 0 på utport FB
        LDA Shadow
        ORA #1
        STA Shadow
        STA $FB

; nollställ bit 1 på utport FB
        LDA Shadow
        ANDA #~2
        STA Shadow
        STA $FB
```

CHALMERS

In- och utportar

Minnesavbildade I/O-portar

```
        ORG 0
Shadow  RMB 1
        ORG $20
        …
; ettställ bit 0 på utport FB
        LDA Shadow
        ORA #1
        STA Shadow
        STA $FB

; nollställ bit 1 på utport FB
        LDA Shadow
        ANDA #~2
        STA Shadow
        STA $FB
```

```
LDA $FB
ORA #1
STA $FB
Fungerar inte!!!
```

Bitvis komplement av 2 = 11111101_2

CHALMERS

Assemblerprogrammering

Demonstrationsexempel 2 – mätning av tid

Följande kod genererar en puls i bit 0 på utport FB_{16}.

```
START   CLR $FB
        LDA #25
SCOUNT  DECA
        BNE SCOUNT
        LDA #%00000001
        STA $FB
        LDA #100
PCOUNT  DECA
        BNE PCOUNT
        CLR $FB
```

Antag att FLIS-processorn har klockfrekvensen 1 MHz och att sekvensen startas vid tidpunkten t = 0.

a) När i tiden genereras pulsen? b) Hur lång är pulsen?

CHALMERS

Assemblerprogrammering

Demonstrationsexempel 3 – kodomvandling

Skriv en instruktionssekvens som omvandlar ett 4-bitars binärt tal till Graykod. Det binära talet skall kontinuerligt läsas från bit b_0-b_3 på inport FB_{16}. Bit b_4-b_7:s värden är ej kända och kan variera mellan läsningarna.

Graykoden skall matas ut på bit b_0-b_3 på utport FB_{16} med bit b_4-b_7 nollställda.

Graykoden för siffrorna 0_{16}-F_{16} finns lagrade i bit b_0-b_3 i adress 30_{16} - $3F_{16}$, med bit b_4-b_7 nollställda.