Assemblerprogrammering – del 1

Dagens föreläsning behandlar:

Kompendiet kapitel 9
Arbetsboken kapitel 15

Ur innehållet:

- Assemblerspråket
- Programmerarens bild
- Assemblering/disassemblering

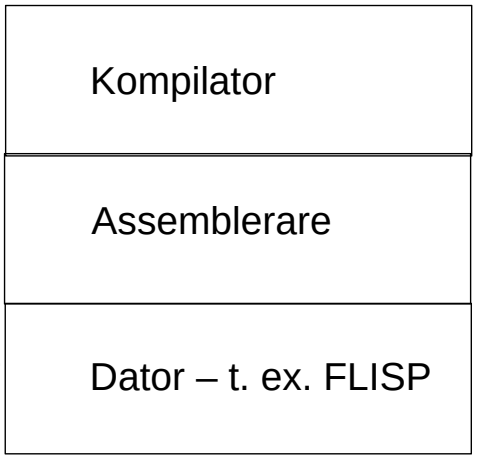
Överblick och motivation

Funktion:

Översätter program skrivet i ett **högnivå-språk** till ett **assembler-program**

Översätter program skrivet i ett assembler-språk till ett maskin-program för given dator

Utför vad som specificeras i programmet.



Gränssnitt:

Högnivåspråk – t. ex. Java, C

Assemblerspråk – 1-1 avbildning till maskinspråk för given dator

Maskinspråk – "ettor" och "nollor"

Assemblerprogrammering

Processen att översätta ett assemblerprogram till ett program i maskinspråk kallas att *assemblera* (från engelskans 'assembly', ’sätta samman’, dvs. sätta ihop ett program utav en sekvens instruktioner).

Den omvända processen, att "dissassemblera" (engelskans 'disassembly', 'plocka i sär') är således att utgå från ett komplett maskinprogram, dvs. ettor och nollor, och dela upp detta i sekvenser av assemblerinstruktioner.

Assemblerspråket står alltså i ”1-1”-förhållande till maskinspråket och är därför unikt för en speciell centralenhet (processor).

Assemblerprogrammets struktur

Källtexten är radorienterad…

Symbolfält	Mnemonic eller direktiv	Operand	Kommentarsfält
	ORG	$20	; Program börjar
start:	LDA	$FB	; Läs data
	STA	$FC	; Skriv data
	JMP	start	; Repetera...
Då symbolfältet används måste symbolnamnet börja i radens första position. Kolon (:) kan, men behöver inte anges.	I detta fält ska instruktioner eller assemblerdirektiv anges	Om instruktionen/direktivet har någon operand, ska denna anges här.	Det sista fältet kan användas för kommentarstext. Det är lämpligt att markera idetta med ett inledande semikolon, men det är inte nödvändigt.

..utöver detta är ”formatteringen” godtycklig…

```
 org $20 ; Program börjar
start: lda $fb ; Läs data
 sta $fc ; Skriv data
 jmp start ; Repetera...
```

Symboler

- Varje symbolnamn måste väljas *unikt* dvs, får bara definieras *en* gång i programmet. Symbolnamnet får vara högst 128 tecken långt.
- Symbolens *första* tecken måste vara en bokstav (a-z eller A-Z) *eller* en "understrykning", därefter kan symbolnamnet dessutom innehålla något av tecknen . "punkt" eller $ "dollartecken". Observera att de svenska tecknen å,ä och ö *inte* får användas i symbolnamn.
- Små respektive stora bokstäver betraktas som *olika* i symbolnamn.

Symbolfält	Mnemonic eller direktiv	Operand	Kommentarsfält
	ORG	$20	; Program börjar
start:	LDA	$FB	; Läs data
	STA	$FC	; Skriv data
	JMP	start	; Repetera...
Då symbolfältet används måste symbolnamnet börja i radens första position. Kolon (:) kan, men behöver inte anges.	I detta fält ska instruktioner eller assemblerdirektiv anges	Om instruktionen/direktivet har någon operand, ska denna anges här.	Det sista fältet kan användas för kommentarstext. Det är lämpligt att markera detta med ett inledande semikolon, men det är inte nödvändigt.

Symbolreferenser

- En symbol kan refereras hur många gånger som helst.
- Ofta, dock inte alltid, används symboler för att ange ett *läge* (*label*) dvs. en adress i assemblerprogrammet.
- Om symbolens absoluta adress är godtycklig sägs den vara *relokerbar*.

Då vi använder en symbol som operand eller i ett uttryck kallas detta *referens*:

```
            ORG     $20
start:      LDA     $FB     ; Symbolen 'start' definieras på denna rad
            STA     $FC
            JMP     start   ; Referens av symbolen 'start'
```

Följande visar hur symbolen kan refereras innan den definierats, *framåtreferens*:

```
            JMP     next    ; Referens av symbolen 'next'
            STA     $FC
next:       ...             ; Symbolen 'next' definieras på denna rad
```


Talprefix

Konstanter som anges med binär eller hexadecimal talbas föregås av ett *prefix*.
Inget prefix används för att ange konstanter i det decimala talsystemet.

Decimal konstant	Decimala konstanter anges utan prefix, siffrorna [0..9] används.
Binär konstant	Binära konstanter anges med ett procenttecken som prefix (%), endast siffrorna 0 och 1 kan användas i talet.
Hexadecimal konstant	Hexadecimala konstanter anges med ett dollartecken som prefix ($), siffrorna [0..9] och tecknen [a..f, A..F] kan användas i talet.
Teckenkonstant	Ett enskilt ASCII-tecken som omges av enkla citationstecken. Skrivsättet 'a' anger exempelvis ASCII-värdet 61_{16}, för gemena A.

Exempel:

```
11 = %1011 = $B
```


Absoluta symboler

Absoluta symboler kan definieras med assemblerdirektivet EQU (*equate*).
Innebörden är att symbolen ges ett värde som sedan inte ändras.

Symbolfält	Mnemonic eller direktiv	Operand	Kommentarsfält
	ORG	$20	; Program börjar
InPort	EQU	$FB	
UtPort	EQU	$FC	
start:	LDA	InPort	; Läs data
	STA	UtPort	; Skriv data
	JMP	start	; Repetera...

Uttryck

Uttryck kan sättas samman av konstanter (absoluta symboler) och operatorer.

Aritmetik

Operator	Betydelse
A+B	Aritmetisk summa av A och B.
A-B	Aritmetisk skillnad mellan A och B.
A*B	Aritmetisk produkt av A och B.
A/B	Resulterande heltalsdel efter division, "heltalsdivision", (*integer division*).
A%B	Resulterande bråktalsdel efter division, "restdivision", (*modulus division*).

Bitvis logik

Operator	Betydelse
~A	Bitvis komplement av A.
A&B	Bitvis logiskt "OCH" av A och B.
A|B	Bitvis logiskt "ELLER av A och B".
A^B	Bitvis "EXKLUSIVT ELLER" av A och B.

Skift

Operator	Betydelse
A<<B	Operand A skiftas B steg till vänster.
A>>B	Operand A skiftas B steg till höger.

Relation

Operator	Betydelse
A==B	Uttryckets värde är 1 om A och B är lika, 0 annars.
A!=B	Uttryckets värde är 1 om A och B är olika, 0 annars.
A<=B	Uttryckets värde är 1 om A är mindre än, eller lika med B, 0 annars.
A>=B	Uttryckets värde är 1 om A är större än, eller lika med B, 0 annars.
A<B	Uttryckets värde är 1 om A är mindre än B, 0 annars.
A>B	Uttryckets värde är 1 om A är större än B, 0 annars.

Logik

Operator	Betydelse
!A	Uttryckets värde är 0 om A är 0, i övriga fall är uttryckets värde 1.
A&&B	Uttryckets värde är 0 om A någon av B är 0, i övriga fall är uttryckets värde 1.
A||B	Uttryckets värde är 0 om A och B båda är 0, i övriga fall är uttryckets värde 1.

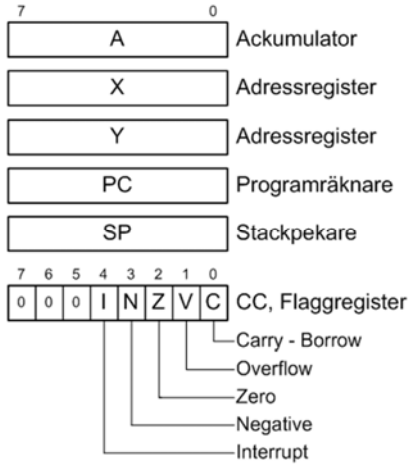
Assemblerdirektiv

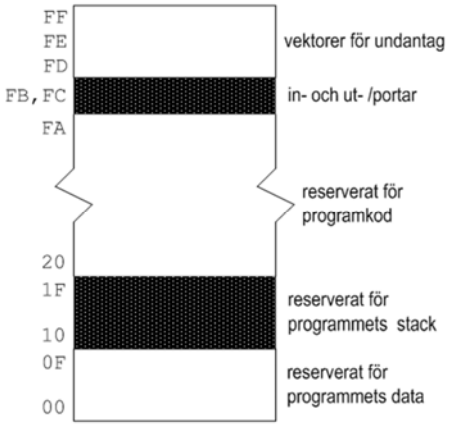
	ORG	<Val>	”ORIGIN”: Anger startadress för påföljande kod/data. Om en symbol används för att ange startadressen måste symbolen vara definierad, dvs inga framåtreferenser är tillåtna här.
Sym	EQU	<Val>	”EQUATE”: Symbolen ’Sym’ representerar värdet <Val>.
[Sym]	FCB	<Val>,<Val>...	”FORM CONSTANT BYTE”: Skapar en sträng med initierade data i minnet I detta fall kan ett symbolnamn anges, men det är inte nödvändigt.
[Sym]	FCS	”<ASCII tecken>”	”FORM CONSTANT STRING": Skapar en sträng med ASCII-tecken i minnet. Symbolnamn kan, men behöver inte, anges.
[Sym]	RMB	<Val>	”RESERVE MEMORY BYTES”: Reservera <Val> bytes i minnet. Minnesinnehållet på dessa adresser är odefinierat. Symbolnamn kan, men behöver inte, anges.

Statisk allokering av minne

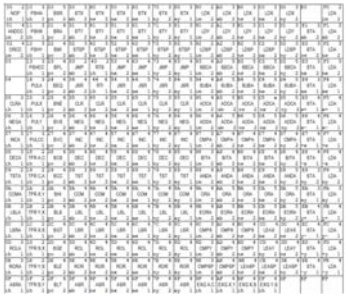
Adress	Innehåll	Assemblerkod (disassemblering)	
		ORG	$20
20	38	FCB	$38
21	98	FCB	%10011000
22		RMB	1
23	41	FCB	´A´
24	61	FCS	”abcd”
25	62		
26	63		
27	64		
28			

Programmeringsmodell för FLISP





Instruktionsuppsättningen

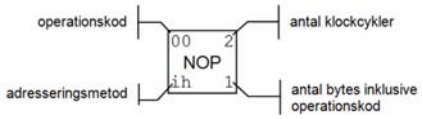


En betraktelse av processorns instruktionslista visar att merparten av instruktionerna tillhör någon av följande kategorier:

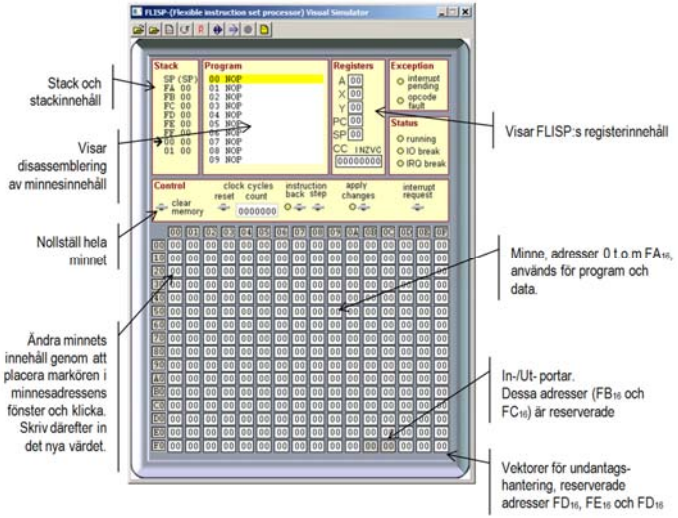
- **datakopiering**
- **databearbetning**, (aritmetik- och logik- operationer)
- **jämförelser** och **tester**
- **programflödesändring**

De två första används för att forma sekvenser, dvs göra tilldelningar och beräkna uttryck.

De två sista används huvudsakligen för att uttrycka selektioner och iterationer dvs. skapa villkorliga och ovillkorliga programflöden.



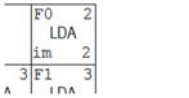
FLISP



Assemblering – assemblerkod till maskinkod

```
    LDA    #7
L1:
    SUBA   #1
    INC    5
    TSTA
    BNE    L1
L2:
    BRA    L2
```

1



Instruktion	Adressering				
LD					
Variant	metod	OP	#	~	
LDA #Data	Immediate	F0	2	2	Dat
LDA Adr	Absolute	F1	2	3	M/

Omedelbar (Immediate) [im]
Operanden är kodad tillsammans med operationen.
RTN: M(PC+1)
Assemblersyntax: #<DATA>

2

Adress	Maskin-kod	Assemblerkod		RTN
?	F0	LDA	#7	7→A
	07			

3

Exempel

Uppgift 15.5

- Handassemblera följande instruktionssekvens, dvs. översätt mnemonics med operander till maskinkod.

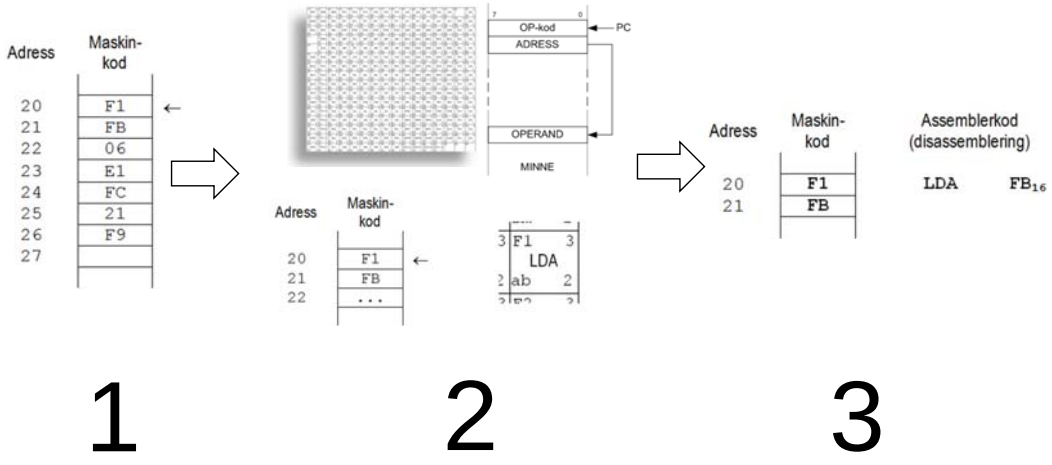
Adress	Maskin-kod		Assemblerkod		RTN
20			LDX	#start	"start"→X
21					
22		L1:	INC	0,X	M(X)+1→M(X)
23					
24			LEAX	1,X	X+1→X
25					
26			CMPX	#stop	"stop"→X
27					
28			BNE	L1	if(Z=0) 2216→PC
29					
2A		L2:	BRA	L2	2A16→PC
2B					
2C					
2D		start:			
2E					
2F					
30		stop:			

Feltryck i arbetsboken här, RTN ska vara:

X – "stop" →CC

Vi löser på tavlan

Disassemblering – maskinkod till assemblerkod



Exempel

Uppgift 15.7

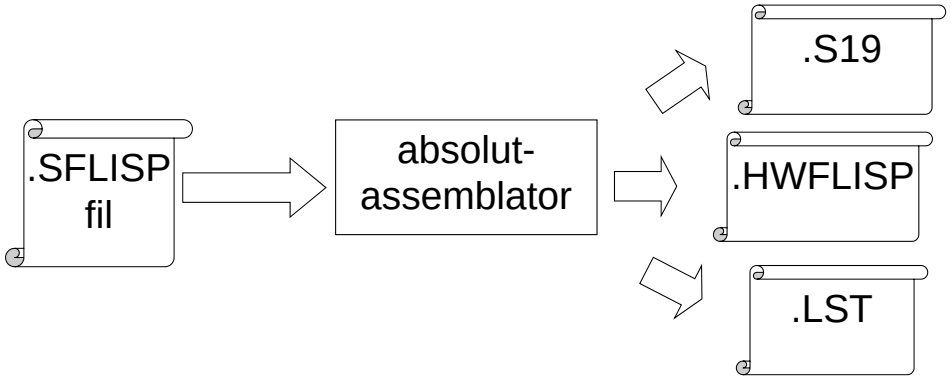
- Disassemblera följande sekvens av maskinkod.

Adress	Maskin-kod	Assemblerkod (disassemblering)
20	90	
21	A4	
22	91	
23	B8	
24	F5	
25	EB	
26	25	
27	FC	

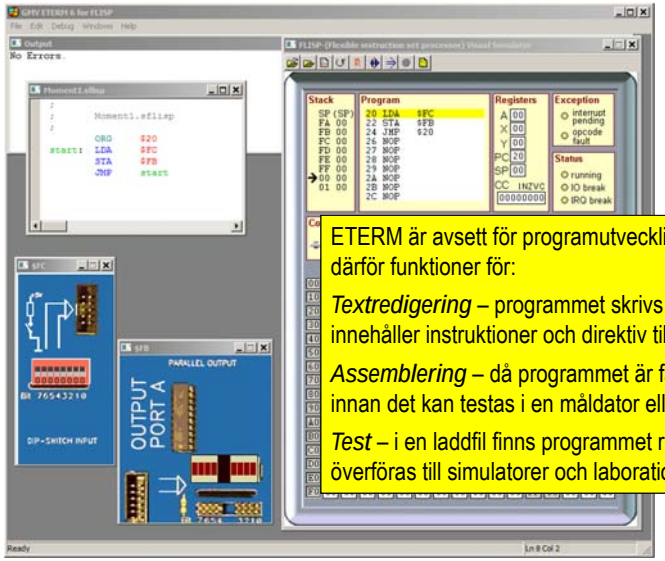
Vi löser på tavlan

Assemblering

- Källtext assembleras, symboliska adresser bestäms, mnemonics översätts till maskinkod.
- Resultatet är en ”bild” av program/minne färdig att överföras till FLISP simulator (.S19), eller laborationssystem (.HWFLISP) samt en ”listfil” (.LST)



ETERM – programutveckling för FLISP



ETERM är avsett för programutveckling i assemblerspråk och innehåller därför funktioner för:

Textredigering – programmet skrivs i form av källtext, dvs. en textfil som innehåller instruktioner och direktiv till assemblatorn.

Assemblering – då programmet är färdigt översätts det till maskinkod innan det kan testas i en måldator eller simulator.

Test – i en laddfil finns programmet representerat på en form som kan överföras till simulatorer och laborationsutrustning där det nu kan testas.