LV5 Fo12

Aktivera Kursens mål:

- Konstruera en dator mha grindar och programmera denna

Aktivera Förra veckans mål:

- Konstruera styrenheten…. genom att….
- implementera olika maskininstruktioner i styrenheten.
- Kunna använda instruktionslistan och skriva mycket enkla assemblerprogram
- Studera olika instruktionstyper och adresseringsmoder
- Använda utvecklingsmiljön för FLISP

Veckans mål:

- Konstruera styrenheten…. genom att….
- …. implementera olika maskininstruktioner i styrenheten.
- Villkorliga hopp
- Subrutiner och stack
- Skriva enkla program för FLISP
- Strukturerad assemblerprogrammering

Programexempel för FLEX

Addera de 16-bitars talen P och Q.
P hittas på minnesadress 20_{16} (och 21_{16}.)
Q hittas på minnesadress 22_{16} (och 23_{16}.)

Placera resultatet på adress 24_{16} (och 25_{16}.)

Programmets startadress är 40_{16}

LV5 Fo12

Dagens mål: Du ska kunna…..

- **Förstå villkorliga hopp i program**
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner

- ”Programmera i FLEX-miljön”

Villkorliga hopp:

…någon addition…
Om C=1
- så hoppa till felrutin
Annars
- fortsätt beräkningen

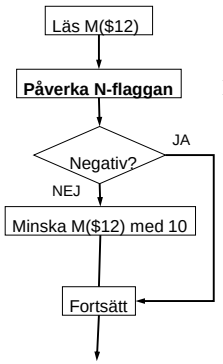
Om PIN-koden är korrekt given
- öppna telefonen
Annars
- stäng telefonen

1) Om vi inte bearbetat alla dataord
- bearbeta nästa dataord
- börja om från 1)
2) Annars fortsätt med annat arbete

Vid villkorliga hopp används ***relativa hoppinstruktioner***

Villkorliga hopp -Instruktioner

if SALDO ≥ 0
minska SALDO med 10:-
fortsätt



Negativt när N=1

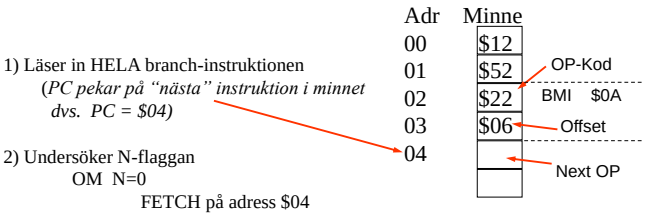
"Testa" SALDO

Lämpliga hoppinstruktioner
BMI eller BPL

Vad gör processorn vid BMI ?

1) Läser in HELA branch-instruktionen
(*PC pekar på "nästa" instruktion i minnet*
dvs. PC = $04)

2) Undersöker N-flaggan
OM N=0
FETCH på adress $04
OM N=1
PC + offset → PC
FETCH på adress $0A



PC $04
+Offset $06
=Ny PC $0A

LV5 Fo12

Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- **Implementera BEQ-instruktionen i styrenheten.**
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner

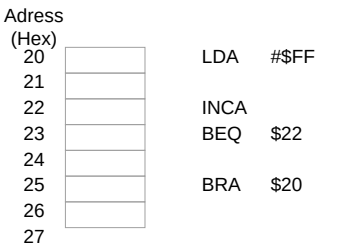
FOKUS på....

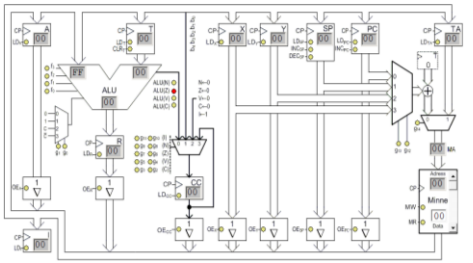
- "Programmera i FLEX-miljön"

Relativa villkorliga hopp

Om Z = 1
PC+Offset → PC, FETCH
Annars
FETCH

Testa först i
FLISP-simulatorn





Execute

BEQ $22

State	Summa-term	RTN	Styrsignaler = 1	Kommentar

Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- **Använda villkorliga hoppinstruktioner**
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner

Fokus på….

- ”Programmera i FLEX-miljön”

Villkorliga hopp

Ext 9

Instruktionsuppsättningen för FLEX-processorn har ett antal villkorliga hoppinstruktioner.

De kan indelas i följande tre grupper:

1. **Enkla hoppvillkor.**
2. **Hoppvillkor för tal utan inbyggt tecken.**
3. **Hoppvillkor för tal med inbyggt tecken.**
   (2-komplementrepresentation)

Villkorliga hopp - forts

Ext 9

1. Enkla hoppvillkor.

Vid de enkla villkorliga hoppen testas innehållet i
en av flaggvipporna N, Z, V eller C
och hoppet utförs om villkoret är uppfyllt,
dvs den aktuella flaggvippans värde, är 0 resp 1.

Villkorliga hopp - forts

Ext 9

2. Hoppvillkor för tal utan inbyggt tecken.

Förutsätt att flaggorna har påverkats av en subtraktion X - Y enligt:

```
1) LDAA      XVALUE    Läs X från minnet till  A
2) CMPA      #Y        Låt skillnaden X - Y påverka flaggorna
3) B(Villkor) Hoppadress Utför hoppet om villkoret är uppfyllt
```

X och Y är 8-bitars tal som tillhör intervallet [0, 255].

Villkorliga hopp - forts

Ext 9

2. Hoppvillkor för tal utan inbyggt tecken.
X och Y är 8-bitars tal som tillhör intervallet [0, 255]. ***Flaggor C och Z***

X > Y, X≥Y, X = Y, X≠ Y, X ≤Y och X < Y.

Relation	C	Z	Sant om
X > Y	0	0	C'• Z' = 1
X = Y	0	1	Z = 1
X < Y	1	0	C = 1

Villkorliga hopp - forts

Ext 9

3. Hoppvillkor för tal med inbyggt tecken. (2-komplementstal)

Förutsätt att flaggorna har påverkats av en subtraktion X - Y enligt:

```
1) LDAA      XVALUE    Läs X från minnet till  A
2) CMPA      #Y        Låt skillnaden X - Y påverka flaggorna
3) B(Villkor) Hoppadress Utför hoppet om villkoret är uppfyllt
```

X och Y är 8-bitars tal som tillhör intervallet [-128, 127].

Villkorliga hopp - forts

Ext 9

3. Hoppvillkor för tal med inbyggt tecken.
X och Y är 8-bitars tal som tillhör intervallet [-128,127].
Flaggor N, V och Z

X > Y, X≥Y, X = Y, X≠ Y, X ≤Y och X < Y.

Relation	N⊕V	Z	Sant om
X > Y	0	0	(N ⊕ V)' •Z' = 1
X = Y	0	1	Z = 1
X < Y	1	0	N ⊕ V = 1

Uppgift

Två variabler P och Q är lagrade i minnet.
Jämför talen och skriv det största till variabeln R.

P är placerad på adress 20_{16} i minnet.
Q är placerad på adress 21_{16} i minnet.
R är placerad på adress 22_{16} i minnet.

Programmets startadress 40_{16} i minnet

Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- **Förstå begreppen stack, stackpekare och stackinstruktioner**
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner

- ”Programmera i FLEX-miljön”

Fokus på….

STACK och STACKPEKARE

Ext 17

STACK: Ett minnesutrymme
Används för att lagra temporära data (registerinnehåll och återhhoppsadresser)

STACKPEKARE: Ett register (**Reg SP**)
som pekar på det senast ditlagda

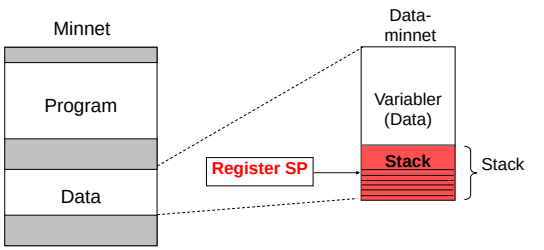
S - SP

INSTRUKTIONER:

PSH: Placera ett registerinnehåll ***PÅ*** stacken
PUL: Hämta ***FRÅN*** stacken ***TILL*** ett register

Stacken - forts

Ext 17



Stacken: ett minnesutrymme som vi temporärt utnyttjar

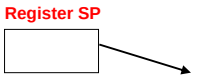
Stacken – forts – några instruktioner

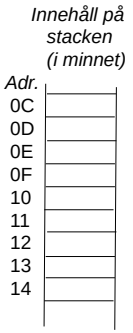
PSHA.

Innehållet i register A skrivs till stacken (till minnet).

RTN-beskrivningen:

1) SP-1 →SP
2) A→ M(SP)





PULA.

Hämta ett dataord från stacken till register A

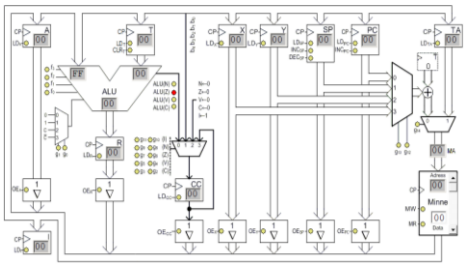
RTN-beskrivningen:

1) M(SP)→ A
2) SP+1→ SP

Uppgift

Definiera en stack som börjar på adress $7F_{16}$.

Placera sedan följande på stacken:
$3B_{16}$, 12_{16}, 66_{16} och $F8_{16}$.



Arb s 139

Execute

PSHA

State	Summa-term	RTN	Styrsignaler = 1	Kommentar

Dagens mål: Du ska kunna…..

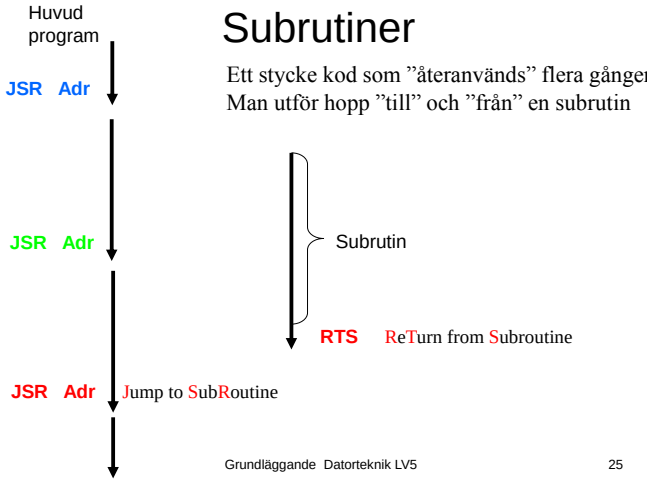
- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- **Förstå användningen av subrutiner**
- Skriva subrutiner

FOKUS på….

- ”Programmera i FLEX-miljön”

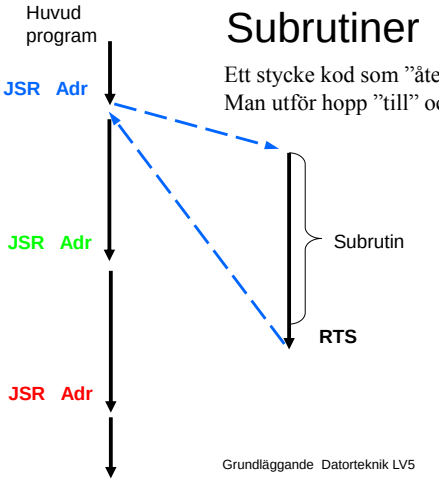
Subrutiner

Ett stycke kod som ”återanvänds” flera gånger.
Man utför hopp ”till” och ”från” en subrutin



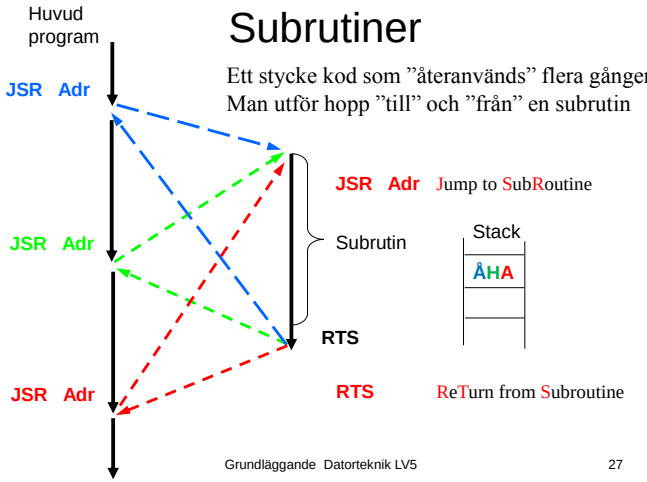
Subrutiner

Ett stycke kod som ”återanvänds” flera gånger.
Man utför hopp ”till” och ”från” en subrutin



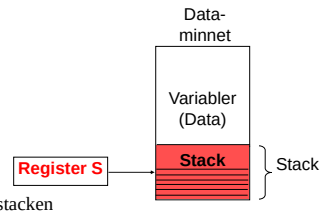
Subrutiner

Ett stycke kod som ”återanvänds” flera gånger.
Man utför hopp ”till” och ”från” en subrutin



Aktivera ….

Subrutin o stack



Stack: Ett minnesutrymme

Stackpekare: Register S. Pekar på översta elementet på stacken

PSH: Kopiera ett reg TILL stacken

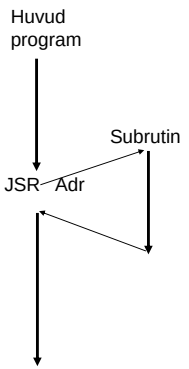
PUL: Hämta FRÅN stacken TILL ett reg

Subrutin: Ett stycke kod, avslutat med instruktionen RTS

JSR: Hopp TILL subrutin

RTS: Återhopp FRÅN subrutin

Subrutiner



JSR Adr	Jump to SubRoutine
RTN	S-1→ S PC → M(S) Adr → PC

RTS	ReTurn from Subroutine
RTN	M(S) → PC S+1→ S

Dagens mål: Du ska kunna…..

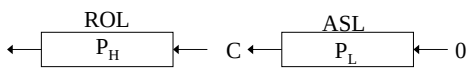
- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- **Skriva subrutiner**

- ”Programmera i FLEX-miljön”

Multiplicera den 16-bitars variabeln P med 2
”Gör en subrutin av det”

Adr	Minne	Assembler prog		
$80	$3B	**ASL**	**$21**	2*P_L
$81	$21			
$82	$3D	**ROL**	**$20**	2*P_H
$83	$20			
$84	$43	**RTS**		
$85				

Adr	Minne
20	P_H
21	P_L



Multiplicera den 16-bitars variabeln P med 4
”Gör en subrutin av det”

Adr	Minne	Assembler prog		
$80	$3B	ASL	$21	2*P_L
$81	$21			
$82	$3D	ROL	$20	2*P_H
$83	$20			
$84	$43	RTS		
$85	$34	**JSR**	**$80**	**Mul2**
$86	$80			
$87	$34	**JSR**	**$80**	**Mul2**
$88	$80			
$89	$43	**RTS**		
$8A				

Adr	Minne
20	P_H
21	P_L

Dagens mål: Du ska kunna…..

- **Förstå villkorliga hopp i program**
- **Implementera BEQ-instruktionen i styrenheten.**
- **Använda villkorliga hoppinstruktioner**
- **Förstå begreppen stack, stackpekare och stackinstruktioner**
- **Implementera PSH-instruktionen i styrenheten.**
- **Förstå användningen av subrutiner**
- **Skriva subrutiner**

- ”Programmera i FLEX-miljön”

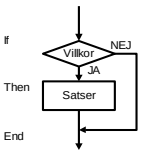
Veckans mål:

- Konstruera styrenheten…. genom att….
- …. implementera olika maskininstruktioner i styrenheten.
- Villkorliga hopp
- Subrutiner och stack
- ”Programmera i FLISP-miljön”
- Strukturerad assemblerprogrammering

Dagens mål: Du ska kunna….

- Skriva kontrollstrukturer
  - If then else; While, Do while
  - ”testa bitar”
- Använda IO-simulatorer
- Skriva strukturerade assemblerprogram för FLISP
  - Programmoduler
  - Top-down/Bottom up
  - Inre och yttre dokumentation

Kontrollstruktur if (...) {Satser}



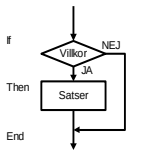

```
if (DipSwitch != 0)
   HexDisp = Dipswitch;
```

”Rättfram” kodning...

```
                ...
                TST   DipSwitch
                BNE   assign
                BRA   end

assign          LDA   DipSwitch
                STA   HexDisp
end             ...
                ...
```

BNE	“Hopp” om ICKE *zero*	Z=0
BEQ	“Hopp” om *zero*	Z=1

Kontrollstruktur if (...) {Satser}



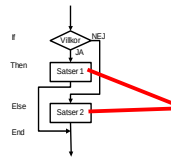

```
if (DipSwitch != 0)
   HexDisp = Dipswitch;
```

Bättre kodning...

```
                ...
                TST   DipSwitch
                BEQ   end

                LDA   DipSwitch
                STA   HexDisp
end             ...
                ...
```

BNE	“Hopp” om ICKE *zero*	Z=0
BEQ	“Hopp” om *zero*	Z=1

Kontrollstruktur **if (...) {Satser1} else {Satser2}**

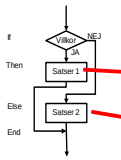



```
if (DipSwitch == 0)
      HexDisp = 1;
else
      HexDisp = 2;
```

```
            ...
            LDA   DipSwitch
            TSTA
            BEQ   then

            LDA   #2
            STA   HexDisp
            BRA   end      {Satser2}

then        LDA   #1
            STA   HexDisp  {Satser1}
end         ...
            ...
```

BEQ	"Hopp" om *zero*	Z=1

Kontrollstruktur **if (...) {Satser1} else {Satser2}**

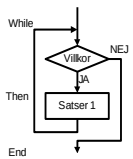



```
if (DipSwitch == 0)
      HexDisp = 1;
else
      HexDisp = 2;
```

```
            ...
            LDA   DipSwitch
            TSTA
            BNE   else

            LDA   #1
            STA   HexDisp
            BRA   end      {Satser1}

else        LDA   #2
            STA   HexDisp  {Satser2}
end         ...
            ...
```

BNE	"Hopp" om ICKE *zero*	Z=0

Kontrollstruktur **while (...) {Satser}**

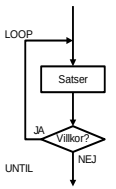



```
while (DipSwitch != 0)
      HexDisp = 1;
```

```
            ...
while
            LDA   DipSwitch
            TST
            BEQ   end_while

            LDA   #1
            STA   HexDisp
            BRA   while    {Satser}
end_while
            ...
            ...
```

BEQ	"Hopp" om *zero*	Z=1

Kontrollstruktur **do {Satser} while (...)**




```
do
{
      HexDisp = 0;
}while (DipSwitch >= 10);
```

```
            ...
do
            CLRA
            STA   HexDisp  {Satser}

            LDA   DipSwitch
            CMPA  #10
            BHS   do
            ...
```

Test av tal utan tecken		
BHS	Villkor: R≥M	C=0

Villkorlig programflödeskontroll

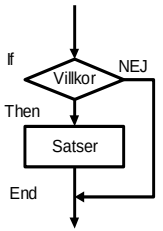
Mnemonic	Funktion	Villkor
Enkla flaggtest		
BCS	"Hopp" om *carry*	C=1
BCC	"Hopp" om ICKE *carry*	C=0
BEQ	"Hopp" om *zero*	Z=1
BNE	"Hopp" om ICKE *zero*	Z=0
BMI	"Hopp" om *negative*	N=1
BPL	"Hopp" om ICKE *negative*	N=0
BVS	"Hopp" om *overflow*	V=1
BVC	"Hopp" om ICKE *overflow*	V=0
Test av tal utan tecken		
BHI	Villkor: R>M	C + Z = 0
BHS	Villkor: R≥M	C=0
BLO	Villkor: R<M	C=1
BLS	Villkor: R≤M	C + Z = 1
Test av tal med tecken		
BGT	Villkor: R>M	Z + (N ⊕ V) = 0
BGE	Villkor: R≥M	N ⊕ V = 0
BLT	Villkor: R<M	N ⊕ V = 1
BLE	Villkor: R≤M	Z + (N ⊕ V) = 1

Dagens mål: Du ska kunna….

- Skriva kontrollstrukturer
  - If then else; While, Do while
  - **"testa bitar"**
- Använda IO-simulatorer
- Skriva strukturerade assemblerprogram för FLISP
  - Programmoduler
  - Top-down/Bottom up
  - Inre och yttre dokumentation

Att testa bitar på inporten – *EN BIT*

If villkor then
Satser
end



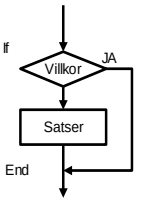
if b_3=1 then
Satser
end

```
      LDA  Inport
      ANDA #%00001000
      BEQ        end
      ---              Satser
end
      ---
```

```
      LDA        Inport
      BITA       #%00001000
      BEQ        end
      ---              Satser
end
      ---
```


Att testa bitar på inporten – *FLERA BITAR*

if (b_3=1) && (b_0=1) then goto end
Satser
end




```
      LDA  Inport
      ANDA #%00001001
      CMPA #%00001001
      BEQ        end
      ---              Satser 1
end
      ---
```

Dagens mål: Du ska kunna….

- Skriva kontrollstrukturer
  - If then else; While, Do while
  - ”testa bitar”
- **Använda IO-simulatorer**
- Skriva strukturerade assemblerprogram för FLISP
  - Programmoduler
  - Top-down/Bottom up
  - Inre och yttre dokumentation

Exempel IN- o UT-matning / Testa en bit

Skriv ett program som hela tiden läser inporten (DipSwitch),
om b_6 av inporten noll, skriv 7 till utporten (Hexdisplay)
annars skriv 22 till utporten

Programmets startadress: $10

Exempel IN- o UT-matning / Testa en bit

Skriv ett program som hela tiden läser inporten (Keyboard),
om tangent nummer 6 är aktiverad,
skriv $66 till utporten (HexDisplay)
annars skriv $FF till utporten

Programmets startadress: $30

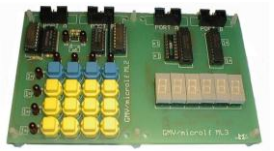
LV5 Fo13

Dagens mål: Du ska kunna….

- Skriva kontrollstrukturer
  - If then else; While, Do while
  - ”testa bitar”
- Använda IO-simulatorer
- **Skriva strukturerade assemblerprogram för FLISP**
  - Programmoduler
  - Top-down/Bottom up
  - Inre och yttre dokumentation

Programmeringsprojekt:

Bygg ett stoppur



Du ska kunna:

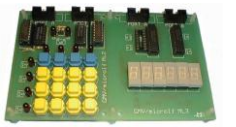
1. *Starta klockan*
2. *Stoppa klockan*
3. *Nollställa klockan*

Programmeringsprojekt

Bygg ett stoppur

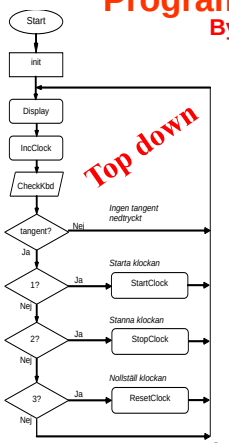
Vad behövs?

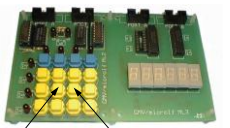
1. *Räknare som håller reda på tiden*
2. *Huvudprogram*
3. *Rutin som visar tiden*
4. *Hur lagras tiden, format?*
5. *Rutin som ändrar (ökar) tiden*
6. *Rutin som läser av tangentbord…*
7. *Koda tangentnumren (Start, stop, nollställ klockan)*
8. *Rutin som startar klockan*
9. *Rutin som stannar klockan*
10. *Rutin som nollställer klockan*
11. *mm*



Programmeringsprojekt

Bygg ett stoppur





Bottom up

Adress	Data	Kommentar
Clock	$02	Timmar
	$05	Tio minuter
	$01	Minuter
	$03	Tio sekunder
	$07	Sekunder
	$04	Tiondelar

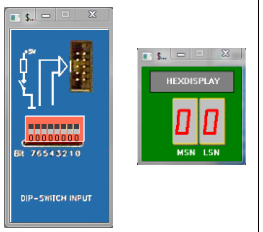
YTTRE Dokumentation

```
****************************************
*SUBRUTIN – DELAY
* Beskrivning: Skapar en fördröjning om
* ANTAL x 500 ms.
*
* Indata: Antal intervall,om 500 ms i A
* Anrop : LDA  #6     Fördröj 6*500ms = 3s
*         JSR  DELAY
*
* Utdata: Inga
* Register-påverkan:Ingen
* Anropad subrutin: Ingen.
****************************************
```

INNRE Dokumentation

```
DELAY PSHA
      PSHX          Spara reg på stacken

      TSTA          Är fördröjningsvärde noll ?
      BEQ   DelExit Hoppa om noll

ALOOP LDX   #Konst

* Snurra som tar 11 klockcykler
XLOOP LEAX  -1,X    4 cykler
      CMPX  #0      3 cykler
      BNE   XLOOP   4 cykler

      DECA
      BNE   ALOOP   Ytterligare fördröjning ?
      ---
```


Skriv en rutin som räknar antal ettställda bitar på inporten (Dipswitch) och skriver resultatet på utporten (Hexdisplay)

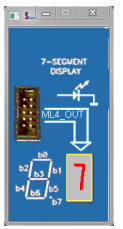


Programmets startadress: $20

Inport adress: $FB

Utport adress: $FC

Skriv ut siffrorna [0..9_{10}] på sifferindikatoren



Testa programmet med ”Run Slow” och ”Run Fast”

Skriv strömbrytarnas värde [0..F_{16}] på sifferindikatoren