TSEA28 Datorteknik Y (och U)

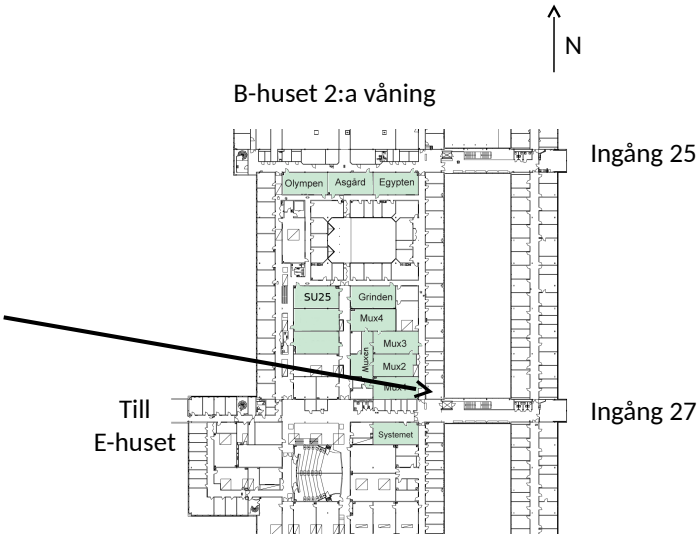
Föreläsning 1

Kent Palmkvist, ISY

LINKÖPINGS UNIVERSITET

Vem är jag

- Föreläsare och kursansvarig
  - Kent Palmkvist
  - Kent.Palmkvist@liu.se
  - Kontor 3B:502 (andra våningen



Tillgänglig information

- Alla slides finns på kursens hemsida
  - https://www.isy.liu.se/edu/kurs/TSEA28
- Kursmaterial (förutom bok) även på kurshemsidan
  - Anvisningar för kursen (deadlines etc.)
  - Labbmaterial
  - Gamla tentor
  - Material från tidigare versioner av kursen
- Lisam används också
  - Labanmälan
  - Eventuellt videos från tidigare år

Kursens mål

- Förstå
  - Hur en dator är uppbyggd och fungerar på registernivå
  - Binär aritmetik
- Kunna
  - Skriva små enkla assemblerprogram
  - Hantera in/utmatning
  - Implementera instruktioner m h a mikroprogrammering
- Känna till
  - Principer för hur cache fungerar
  - Olika sätt att öka exekveringshastigheten
  - Hur processorn påverkar operativsystemet

Varför detta är intressant

- En programmerings-intresserad U:are
  - Förstå begränsningar hos olika datorer
- En matematikintresserad Y:are
  - Snabba upp beräkningsprogram och simuleringar
- En fysikintresserad Y:are
  - Använda sensorer för insamling av mätdata
- En elektronikintresserad student
  - Nästan all elektronik innehåller en dator

Det är roligt!

Senare kurser

- För Y
  - Elektronik kandidatprojekt
  - Datorteknik och realtidssystem
  - Inbyggda DSP processorer
  - Datorarkitektur
  - Datorteknik – ett datorsystem på ett chip
- För U
  - Processprogrammering och operativsystem
  - Kompilatorkonstruktion?

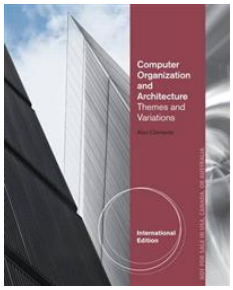
Administrativ information

Kursens innehåll

- Går VT1 + VT2
- 6 hp (3 hp tentamen, 3 hp labb)
- 16 föreläsning (8 under VT1)
- 4 lektioner (2 under VT1)
- 5 laborationer (3 under VT1)
- 1 tentamen (4h)
- Självstudier > 100h !!
  - Laborationsförberedelser ~ 12h per lab
  - Varierar mellan studenter, vissa kan behöva mer tid!

Kurslitteratur

- Computer Organization and Architecture – Themes and Variations; Alan Clements
  - Referenslitteratur
- Laborationsanvisningar (finns på kurshemsidan)
- Notering: Tidigare års (Roos mfl) kurslitteratur går att använda, men saknar många detaljer
  - Grundläggande Datorteknik (Roos)
  - Kompendium i Datorteknik (Wiklund)



Föreläsningar och lektioner

- Föreläsningar bygger delvis på kursboken, men även annat material ingår
  - Slideskopior finns på kurshemsidan
  - Går att klara sig utan kursbok
- Lektioner introducerar och beskriver labbuppgifterna
  - Se dom som frågestunder
  - Se till att vara förberedd innan lektion!
  - Läs igenom labb-beskrivning och dokumentation innan lektion!

Laborationer

- Fem laborationer uppdelat på 2 x 2h (totalt 4h per labb)
  - Lab 1: Kodlås (introduktion till asembler)
  - Lab 2: Avbrott
  - Lab 3: Digitalur
  - Lab 4: Mikroprogrammering
  - Lab 5: Bussar och cache
- Laborationsgrupper
  - Max 2 personer per grupp
  - Max 15 grupper per labbtillfälle
- Labbanmälan via lisam
  - Måndag 23/1 kl 12.30 öppnar anmälan
- Viktigt!
  - Labbanmälan stänger dagen innan första labbtillfället
  - Bara anmälda studenter får delta på laborationen

Laborationsförberedelser

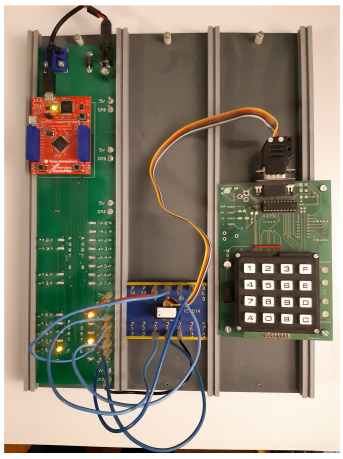
- Förbered innan labbtillfälle och lektion
  - Läs labbkompendie och annat material
  - Lös uppgifter och/eller skriv kod innan labbtillfället
- Använd gärna möjligheten att köra utrustningen hemifrån
  - Möjligt köra distansversionen för att se om kod fungerar
  - Tillgängligt redan nu, se anvisningar på kurshemsidan
- Redovisning av lab görs på plats
  - Närvaro krävs vid redovisningen (går inte sitta på distans)
  - Inkluderar uppkoppling och test
- Labb tillgängligt utanför schemalagd tid, start efter 1:a labb-tillfället)

Förändringar från tidigare år

- Nya slides (4:3 -> 16:9)

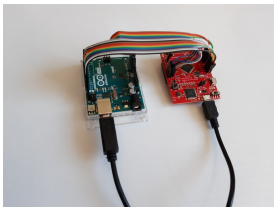
Laborationshårdvara Lab 1-3: Darma

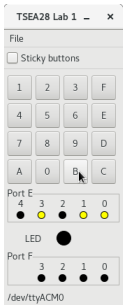
- System med mikrokontroller baserad på Cortex-M4 processor
  - Tillverkare: Texas Instruments (det lilla röda kortet)
  - Processordesign: ARM
  - Gröna kortet till vänster: Darma, lokalt LiU
  - Knappar, displayer etc. kopplas in via kopplingspunkter i det gröna kortet
- Mjukvaran för programmering gratis nedladdningsbart från ti.com
  - Fungerar på windows, mac och linux
  - För att testa kod behövs ett inkopplat kort



Labutrustning för arbete på distans

- Emulerar anslutna tryckknappar etc mha programvara
  - Fast koppling
- Styr och mät av värden på anslutningar via en Arduino Uno
  - GUI för att se och styra (skrivet i python)
- Design tillgänglig på gitlab.liu.se för dom som vill bygga eget eller bara är nyfikna
  - Röda kortet är ett EK-TM4C123GXL (200-300 kr)

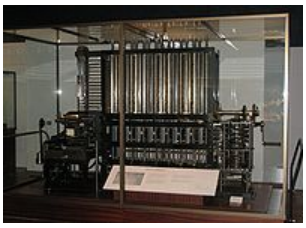




Lite datorhistoria

De tidiga experimenten

- Analytic engine (1837, Charles Babbage)
  - Ritningar men ingen fungerande maskin
  - Ca 3 minuter för multiplikation av två 20-siffriga tal
- Z3 (1941, Konrad Zuse)
  - Elektromekanisk (relä)
  - 3 sekunder för multiplikation



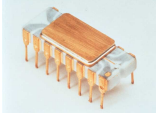
https://commons.wikimedia.org/wiki/File:Babbage_Difference_Engine.jpg

https://commons.wikimedia.org/wiki/File:Z3_Deutsches_Museum.JPG

Transistorbaserad elektronik

- Eniac, 1946
  - Radiorör, 150 kW effektförbrukning
  - 5000 additioner/subtraktioner per sekund
- Intel 4004, 1971
  - 2300 transistorer
  - Första enchips mikroprocessorn
  - 740 kHz klockfrekvens, 0.095 MIPS (Miljoner Instruktioner Per Sekund)

https://commons.wikimedia.org/wiki/File:ENIAC_Penn1.jpg#/media/File:ENIAC_Penn1.jpg



https://commons.wikimedia.org/wiki/File:Legendary_Chip_Designer_Betting_on_Human_Mind.jpg#/media/File:Legendary_Chip_Designer_Betting_on_Human_Mind.jpg

Datorer börjar dyka upp i hemmiljö

- Cray-I, 1975
  - Superdator, vektormaskin, 80 MHz, 160 MIPS
- Apple II, 1977, resp. VIC20, 1981 (tidiga hemdatorer)
  - 8-bitars 6502 processor
  - 1 MHz klockfrekvens, 3500 transistorer
  - 48 Kbyte RAM minne (max)

https://commons.wikimedia.org/wiki/File:Commodore-VIC-20-FL.jpg#/media/File:Commodore-VIC-20-FL.jpg

Två viktiga designer som påverkar än

- IBM PC, 1981
  - 16-bitars 8088 processor, 29000 transistorer
  - 4.77 MHz klockfrekvens, max 640 KB RAM
  - Ursprunget till PC-datorer (windows-maskiner)
- ARM, 1985
  - 32-bitars RISC-processor, 25000 transistorer
  - Acorn Archimedes
  - ARM Ltd startat av Apple och Acorn 1990

https://commons.wikimedia.org/wiki/File:IMB_PC-IMG_7271.jpg#/media/File:IMB_PC-IMG_7271.jpg

http://www.oldcomputers.net/ibm5150.html

https://commons.wikimedia.org/wiki/File:AcornArchimedes-Wiki.jpg#/media/File:AcornArchimedes-Wiki.jpg

Början av 2000-talet, större effektutveckling

- AMD64, 2003
  - 64-bitars 8086 kompatibel
  - 100 miljoner transistorer
  - 1 GHz klockfrekvens
- Intel Haswell (i3, i5, i7), 2013
  - Inkluderar grafikprocessor, 1.4 miljarder transistorer, 3 GHz
- AMD Epyq, 2017
  - 32 kärnor (processorer), 19 miljarder transistorer, 180W

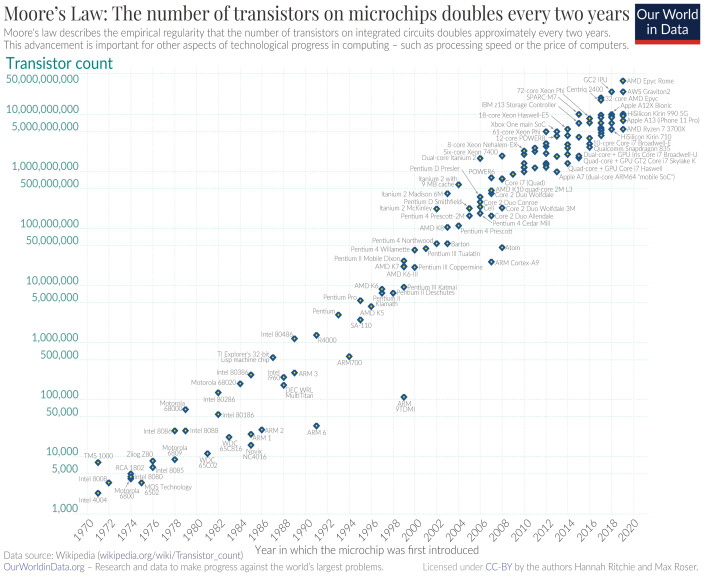
Effektutveckling blir en viktig begränsning för persondatorer runt år 2000

Bärbara/mobiler

- Mobila enheter har också en ökande komplexitet
  - Inkluderar oftast grafikgenerering, 5G och WLAN
- Numera även stöd för avancerade kameror och AI-applikationer
- Apple A5 (Iphone 4S), 2011
  - 2 kärnor, 800 Mhz
- Apple A11 Bionic (Iphone 8), 2017
  - 6 kärnor (2 stycken på 2.4GHz), 4.3 miljarder transistorer
- Apple A12 Bionic (Iphone XR), 2018
  - 6 kärnor (2 stycken på 2.5GHz), 7 miljarder transistorer

Lite datorhistoria, trender

- Miniatyrisering ger allt snabbare datorer
- Exponentiell prestandaökning
  - Moores lag
  - Bildens y-axel visar exponentiell skala
- Fysiska begränsingar finns
  - Tillverkningsprocessernas namn speglar inte direkt de fysiska måtten (7 nm, 5 nm etc.)



Prestandamått. Aktuella trender

- Beräkningshastighet
  - Instruktioner / s (MIPS)
- Effektförbrukning (joule/instruktion)
  - Värmeutveckling ger krav på kylning
- Storlek
  - Fler transistorer och längre ledare ger större kapacitancer och större effektförbrukning
- Pris
- Hårdvaran allt mer specialiserad
  - Kryptering, videokodning och avkodning, Signalbehandling
- Många parallellkopplade datorer
  - Ibland delar datorerna vissa delar, (inkl delar av processorn)
- Mjukvaran måste anpassas till denna hårdvara
  - Kräver god förståelse för datorteknik

Summering

- Hastigheten (instruktioner per sekund) ökar exponentiellt
- Priset går ned trots ökad prestanda
- Moores lag
  - Antal transistorer på ett chip dubbleras vartannat år
- Generellt: Prestanda ökar exponentiellt
  - Olika exponenter => differensen ökar också exponentiellt
  - Exempel: Under 1980-talet tog det lika lång tid för processorn att utföra en instruktion som en läsning/skrivning i minnet. Antal instruktioner/sekund har ökat fortare än antal läsningar/sekund i minnet

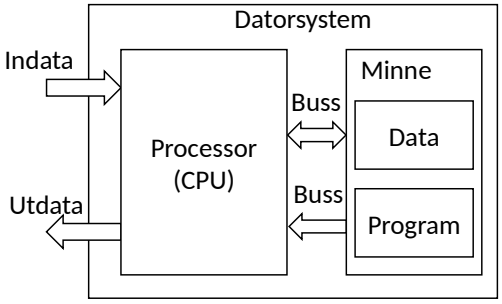
En dators inre uppbyggnad

En programmerares vy av en dator

- Högnivåspråk
  - Interpreterande (tolka kod under körning, t ex java, python)
  - Kompilerande (översätt innan körning till maskinkod, t ex C)
  - Generella, går att använda på många olika datorer
- Assemblerspråk
  - Datorns egna interna språk, unikt för varje datorfamilj
  - Alla högnivåspråk måste översättas till eller tolkas av program beskrivna i assembler
- Mikroprogrammering (inget en vanlig användare ser)
  - Styrning av interna funktioner i datorns inre

Översiktsbeskrivning av en dator

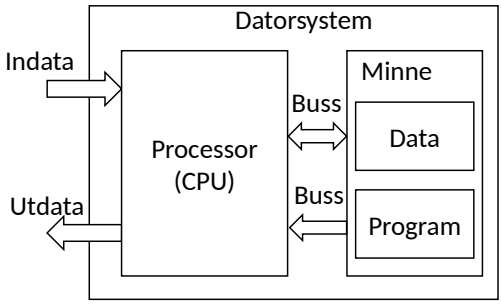
- Indata
  - Switchar, sensorer, kommunikationsmoduler
- Utdata
  - Lysdioder, reläer, video, etc.
- In och utdata i digital form
  - Binära tal
- Två huvuddelar
  - Minne lagrar data och program
  - Processor utför operationer



- Minnet innehåller två typer av information
  - Instruktioner (vad som ska göras)
  - Datavärden (värden att räkna på)

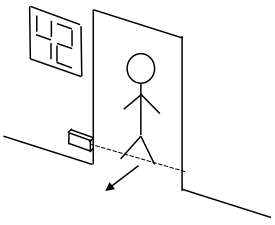
Principiell funktion

- Processorn läser instruktioner från minnet
  - Lagrade i binär form
- Utför instruktionerna på värden från indata och minne
  - Kan lagra mellanresultat inuti processorn
- Slutligt resultat hamnar i minne och som utsignal



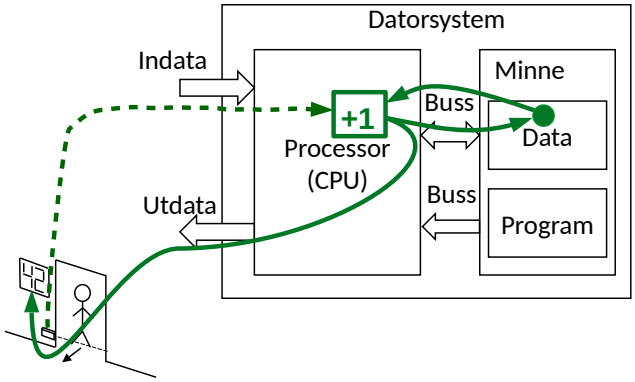
Litet exempel: Räkna studenter

- Jag vill räkna antal studenter som besöker mig på kontoret under en dag
  - Sätt sensor i dörren och en sifferdisplay
  - En dator får räkna
  - Ni som läst digitalteknik: ni vet hur man bygger detta med två räknare, 7-segmentsavkodare, enpulsare etc.
    - Detta byggsätt saknar flexibilitet när nya funktioner ska läggas till



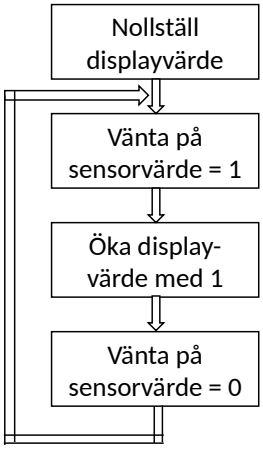
Litet exempel, lösning med dator

- Datorprogrammet ökar ett variabel-värde i minnet när någon passerar öppningen och uppdaterar display
- Displayvärde lagrat i minnet
- Öka värdet i minnet när sensor anger att någon passerar
- Skicka nya värdet till display



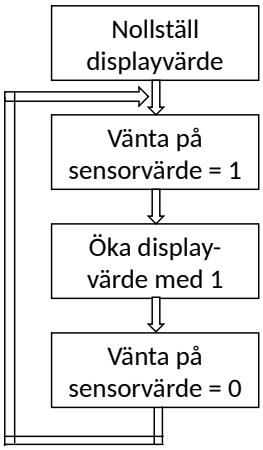
Första steget: definiera en algoritm

- Varje gång sensorvärdet ändras från 0 till 1 ska displayvärdet ökas med 1
- Dela upp i mindre steg
  - Initiering när strömmen slås på
  - Hitta sensorvärdesändring 0 till 1
    - Läs sensorvärde
    - Om sensorvärde inte 1 börja om med läsning
  - Öka displayvärde
  - Hitta sensorvärdesändring 1 till 0
    - Läs sensorvärde
    - Om sensorvärde inte 0 börja om med läsning
  - Börja om att leta efter sensor=1



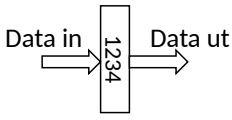
Funktioner/instruktioner som behövs

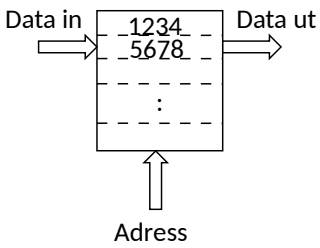
- Processorn kommer utföra instruktion efter instruktion (i en sekvens)
  - Jämför med att läsa ett recept
- Varje instruktion måste vara väldigt enkel (sak finnas hårdvara som utför detta senare)
- Tänk igenom och räkna upp olika aktiviteter
  - Sätta värde i minnet
  - Läsa värde från minnet
  - Skicka värde till display
  - Läsa av sensor (ljusmängd, t ex 0 eller 1 som värde)
  - Jämför värden
  - Börja om i sekvens från tidigare steg
  - Börja om i sekvens om speciellt värde fåtts



Typer av byggblock i processorn

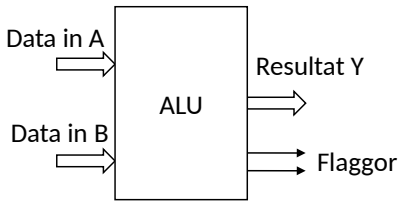
- Register
  - Lagrar ett värde tillfälligt
  - En eller flera bitars värde
- Minne
  - Större vektor av registervärden
  - Minnescell väljs via adress
  - Ibland samma anslutning både för data in och data ut
  - Styrsignaler som väljer om data sparas eller läses





Typer av byggblock i processorn, forts.

- Aritmetikenhet för enklare beräkningar
  - Addition, subtraktion, and, or, etc.
  - Resultat analyseras (kontrollera om beräkning gav svar = 0, om minnessiffra från addition etc.)
    - Flaggor visar egenskaper om beräkningen/resultatet
- Styrsignaler väljer vilken funktion som ska beräknas.



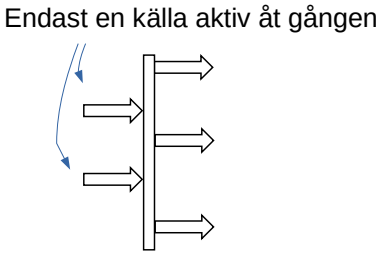
Y = f(A,B)

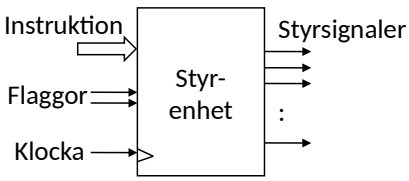
där
f kan vara AND, OR, XOR, NOT, ADD,SUB,SHIFT,ROTATE,...

Flaggor anger egenskaper hos resultat, t ex =0, minnessiffra, etc.

Typer av byggblock i processorn, forts.

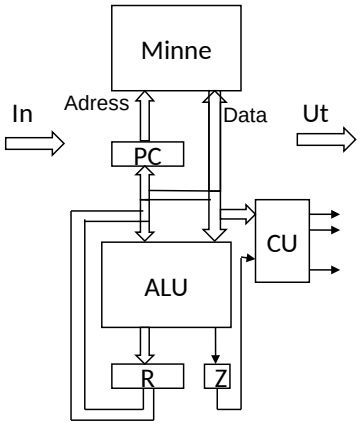
- Bussar
  - Skickar vidare värden från en block till många andra
  - Bara ett värde åt gången på bussen
- Styrenhet
  - Hämta nästa instruktion
  - Avkoda vilken instruktion som ska utföras
    - Hämta nästa instruktion
    - Avkoda instruktion
    - Styr vilka värden som skickas vart
    - Påverkas ibland av flaggornas värde
  - Klocka styr hur ofta styrsignaler ändras





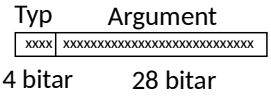
Exekvering av program kräver minst

- Programräknare (PC)
  - Håller reda på position för aktuell instruktion i minnet
- Aritmetikenhet (ALU)
  - Beräknar addition etc.
  - Z: flagga = 1 om resultat = 0
- Tillfälliga register (R)
- Styrenhet (CU)



Programmet lagras i minnet (exempel)

- Kallas en maskininstruktion
  - Binärt datamönster
  - Tolkas av styrenheten i processorn
- Lagras i en minnescell
  - I detta exempel: 32 bitar i en minnescell
- Två delar
  - Vilken typ av operation
    - Exempel: valt 4 bitar => 16 olika typer
  - Argument till operationen
    - Resten av bitarna: 32-4 = 28 bitar argument



Typkod	Förklaring
0000 (0)	Ladda in argument i R
0001 (1)	Addera argument till R
0010 (2)	Jämför argument med R, Z=1 om R=argument,Z=0 annars
0011 (3)	Sätt PC till argument om Z=0
0100 (4)	Sätt PC till argument om Z=1
0101 (5)	Sätt PC till argument
0110 (6)	om argument = 2000 skicka R till display
0110 (6)	om argument = 2100 hämta aktuellt displayvärde till R
0110 (6)	om argument = 2200 hämta sensorvärde till R, R=1 om någon står i dörren, R=0 annars
0111 (7)	läs värde till R från minne på adress i argumentet
1000 (8)	skriv värde I R till minne på adress i argumentet

Assemblerinstruktioner

- Svårt komma ihåg och läsa binärmönster
  - Använd så kallade mnemonics istället
  - Oftast förkortningar

Typkod	Assemblerinstruktion	Föklaring
0000 (0)	mov R,#värde	Ladda in argument i R (MOVe)
0001 (1)	add R,#värde	Addera argument till R
0010 (2)	cmp R,#värde	Jämför R med argument, Z=1 om R=argument, Z=0 annars (CoMPare)
0011 (3)	bne adress	Sätt PC till argument om Z=0 (Branch Not Equal)
0100 (4)	beq adress	Sätt PC till argument om Z=1 (Branch EQual)
0101 (5)	b adress	Sätt PC till argument (Branch)
0110 (6)	bl 2000	om argument = 2000 skicka R till display (Branch and Link)
0110 (6)	bl 2100	om argument = 2100 hämta aktuellt displayvärde till R
0110 (6)	bl 2200	om argument = 2200 hämta sensorvärde till R, R=1 om någon står i dörren, R=0 annars

Assemblerinstruktioner, forts

- Olika processorfamiljer har olika mnemonics
  - Olika namn för samma funktion
  - Samma namn för olika funktioner
- Finns även pseudo-instruktioner i assembler
  - Styr assemblerns generering av maskininstruktioner
  - Kan ibland översätta en instruktion till en/flera andra

Z80	ARM	68000	x86_64	Betydelse
ld	ldr	move	mov	Läs data från minne
ld	str	move	mov	Skriv data till minne
jr	b	bra	jmp	Hoppa i programmet
cp	cmp	cmp	cmp	Jämför
add	add	add	add	Addera

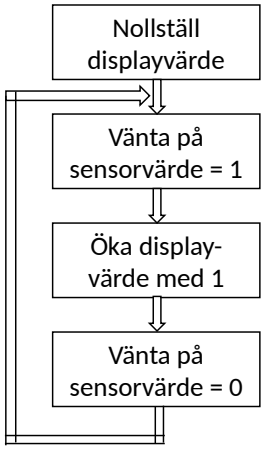
Pseudoinstruktion: Clr r0
Ska nollställa R0

Assembler översätter kanske till
xor r0,r0 ; r0 = r0 xor r0

Implementering i exempeldatorn

- Beskriv varje blocks funktion med hjälp av befintliga instruktioner

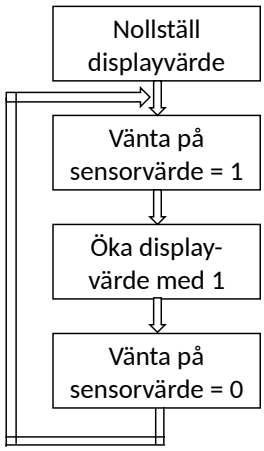
Adress	Maskinkod	Assembler-instruktion	Förklaring
0		mov R,#0	; Nollställ displayvärde
1		bl 2000	; sätt display till 00
2		bl 2200	; Hämta sensorvärde
3		cmp R,#1	; Står någon i dörren?
4		bne 2	; nej, gör om
5		bl 2100	; hämta displayvärdet
6		add R,#1	; öka displayvärdet
7		bl 2000	; visa det nya värdet
8		bl 2200	; Hämta sensorvärde
9		cmp R,#0	; Är dörren tom?
10		bne 8	; nej, kontrollera igen
11		b 2	; börja om



Implementering i exempeldatorn, forts.

- Översätt sedan till binär form
  - Dessa binära värden är vad som lagras i minnet

Adress	Maskinkod Typ	Arg.	Assembler-instruktion	Förklaring
0	0	0	mov R,#0	; Nollställ displayvärde
1	6	2000	bl 2000	; sätt display till 00
2	6	2200	bl 2200	; Hämta sensorvärde
3	2	1	cmp R,#1	; Står någon i dörren?
4	3	2	bne 2	; nej, gör om
5	6	2100	bl 2100	; hämta displayvärdet
6	1	1	add R,#1	; öka displayvärdet
7	6	2000	bl 2000	; visa det nya värdet
8	6	2200	bl 2200	; Hämta sensorvärde
9	2	0	cmp R,#0	; Är dörren tom?
10	3	8	bne 8	; nej, kontrollera igen
11	5	2	b 2	; börja om



Assembler (hjälpprogram)

- Översätter assemblerinstruktioner (text) till maskinkod (binärdata)
  - Skriv en textfil med assemblerinstruktioner
- Håller även reda på adresser
  - Ange bara symboliska namn på platser i programmet (kallas label)

```
wait0:  bl 2200      ; Hämta sensorvärde
        cmp R,#0     ; Är dörren tom?
        bne wait0    ; nej, kontrollera igen
```

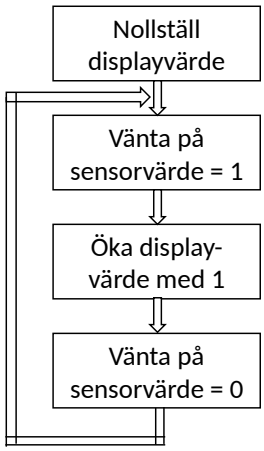
label

Adress	Maskinkod Typ	Arg.
8	6	2200
9	2	0
10	3	8

Assemblerversion av programmet

- Textbeskrivning, inklusive kommentarer
  - Label måste starta längst till vänster
  - Instruktioner måste ha ett mellanslag innan

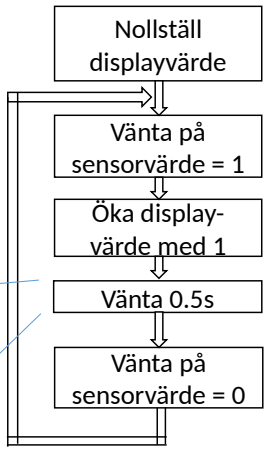
```
start:  mov R,#0     ; Nollställ R
        bl 2000      ; sätt display till 00
wait1:  bl 2200      ; Hämta sensorvärde
        cmp R,#1     ; Står någon i dörren?
        bne wait1    ; nej, gör om
        bl 2100      ; hämta displayvärdet
        add R,#1     ; öka displayvärdet
        bl 2000      ; visa det nya värdet
wait0:  bl 2200      ; Hämta sensorvärde
        comp R,#0    ; Är dörren tom?
        bne wait0    ; nej, kontrollera igen
        b wait1      ; börja om
```



Ytterligare funktion: fördröjning

- Varje besökare kan ge flera pulser
  - Två ben, väska, kläder
- Lösning: vänta en stund efter display räknat upp
  - En loop i programmet som bara tar tid att utföra

```
Label    Assembler-instruktion   Förklaring
start:   mov R,#0                ; Nollställ R
         bl 2000                 ; sätt display till 00
wait1:   bl 2200                 ; Hämta sensorvärde
         cmp R,#1                ; Står någon i dörren?
         bne wait1               ; nej, gör om
         bl 2100                 ; hämta displayvärdet
         add R,#1                ; öka displayvärdet
         bl 2000                 ; visa det nya värdet
         mov R,#0                ; starta timer på 0
delay:   add R,#1                ; öka timer med 1
         cmp R,#10000            ; lämpligt antal klockcykler?
         bne delay               ; inte tillräckligt många varv
wait0:   bl 2200                 ; Hämta sensorvärde
         cmp R,#0                ; Är dörren tom?
         bne wait0               ; nej, kontrollera igen
         b wait1                 ; börja om
```



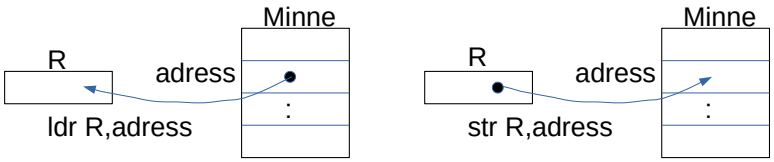
Billigare exempelsystem: Förenkla display

- Enklare display => billigare system
  - Ta bort möjligheten att läsa av värdet
  - Kan bara skriva värde => värde behöver lagras i datorn också
  - R-registrets värde förstörs när sensor läses, kan inte användas
- Två alternativ:Fler register eller lagring i minnet
  - Fler register => fler operationstyper
    - En uppsättning operationer för varje register
  - Lagra i minnet => Två nya instruktioner för att läsa och skriva i minnet
    - Välj adress som inte används till annat (i detta exempel: 100)

Instruktioner för att läsa och skriva i minnet

- Lägg till instruktioner för att spara och hämta ett värde i minnet

Kod	Assemblerinstr.	Förklaring
7	ldr R,adress	Läs värdet på angiven adress i minnet och placera det i R
8	str R,adress	Skriv värdet i R på angiven adress i minnet



Skillnad mellan instruktionstyp 0 och 7

- Olika funktion!

0	mov R,#värde	t ex mov R,#1	Placera värdet 1 i R
7	ldr R,adress	t ex ldr R,1	Läs minnesadress 1 och placera dess värde i R.

  - Motsvarande maskinkod (binärt format)

    mov R,#1 => 00000000000000000000000000000001

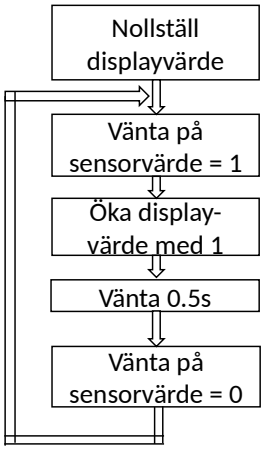
    ldr R,1 => 01110000000000000000000000000001

  - Dvs, samma argument i maskinkod, men olika innebörd pga olika operationstyp
    - Skillnaden indikeras i assemblerkoden i detta fall med tecknet # framför argumentet (samt att här är mnemonic olika)

Exempel med displayvärdet i minnet

- Operation
  bl 2100
- ersatt med
  ldr R,100
  och
  str R,100

```
start:  mov R,#0        ; Nollställ R
        str R,100       ; spara displayvärde adress 100
        bl 2000         ; sätt display till 00
wait1:  bl 2200         ; Hämta sensorvärde
        cmp R,#1        ; Står någon i dörren?
        bne wait1       ; nej, gör om
        ldr R,100       ; hämta displayvärdet
        add R,#1        ; öka displayvärdet
        bl 2000         ; visa det nya värdet
        str R,100       ; spara nya värdet i minnet
        mov R,#0        ; starta timer på 0
delay:  add R,#1        ; öka timer med 1
        cmp R,#10000    ; lämpligt antal klockcykler?
        bne delay       ; inte tillräckligt många varv
wait0:  bl 2200         ; Hämta sensorvärde
        cmp R,#0        ; Är dörren tom?
        bne wait0       ; nej, kontrollera igen
        b wait1         ; börja om
```



LINKÖPINGS UNIVERSITET

LINKÖPINGS UNIVERSITET