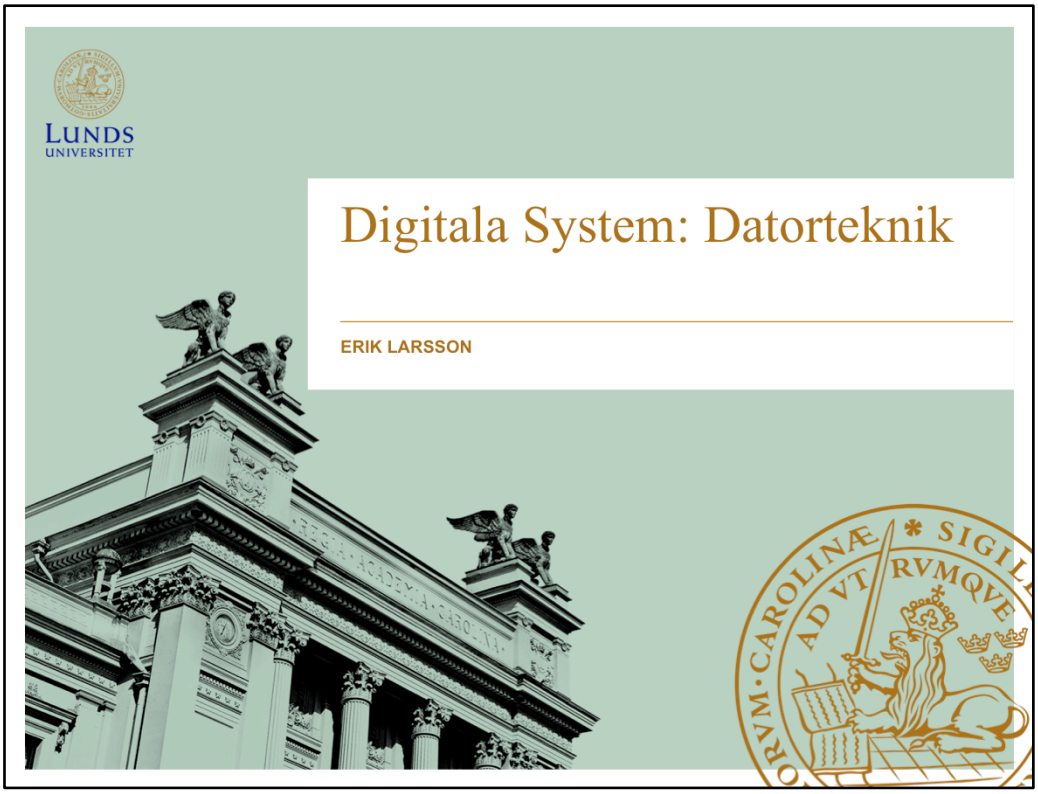

LUNDS
UNIVERSITET
Digitala System: Datorteknik
ERIK LARSSON

Digitala system: Översikt

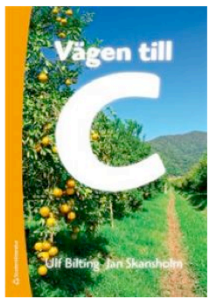
- Antal högskolepoäng: 4,5. Prestationsbedömning: Skriftlig tentamen. **Tid: 120 timmar**
- Antal högskolepoäng: 3. Prestationsbedömning: Godkända laborationsuppgifter. Delmomentet omfattar: Laborationer del 1. **Tid: 80 timmar**
- Antal högskolepoäng: 7,5. Prestationsbedömning: Godkända laborationsuppgifter. Delmomentet omfattar: Laborationer del 2. **Tid: 200 timmar**

Digitala System: Datorteknik

- Datormodellen: Datorns delar och funktion. CPU:n på registernivå
- Assemblyprogrammering: Data- och instruktionsformat. Adresseringsmetoder. Instruktionsrepertoar. ”Timing” och exekveringstid. Stack och subrutiner
- Programutveckling i C: Editering. Kompilering. Länkning. Testning med hjälp av högnivådebugger.

Litteratur

- Föreläsningsanteckningar
- Vägen till C, Jan Skansholm, Ulf Bilting
  Förlag: Studentlitteratur
  ISBN10: 9144076061
  ISBN13: 9789144076065



LUNDS UNIVERSITET

Föreläsningar

- FÖ1: Processorn
- FÖ2: C och assembly
- FÖ3: Minnen
- FÖ4: Cacheminne
- FÖ5: Pipeline

Blooms taxonomi - lärande

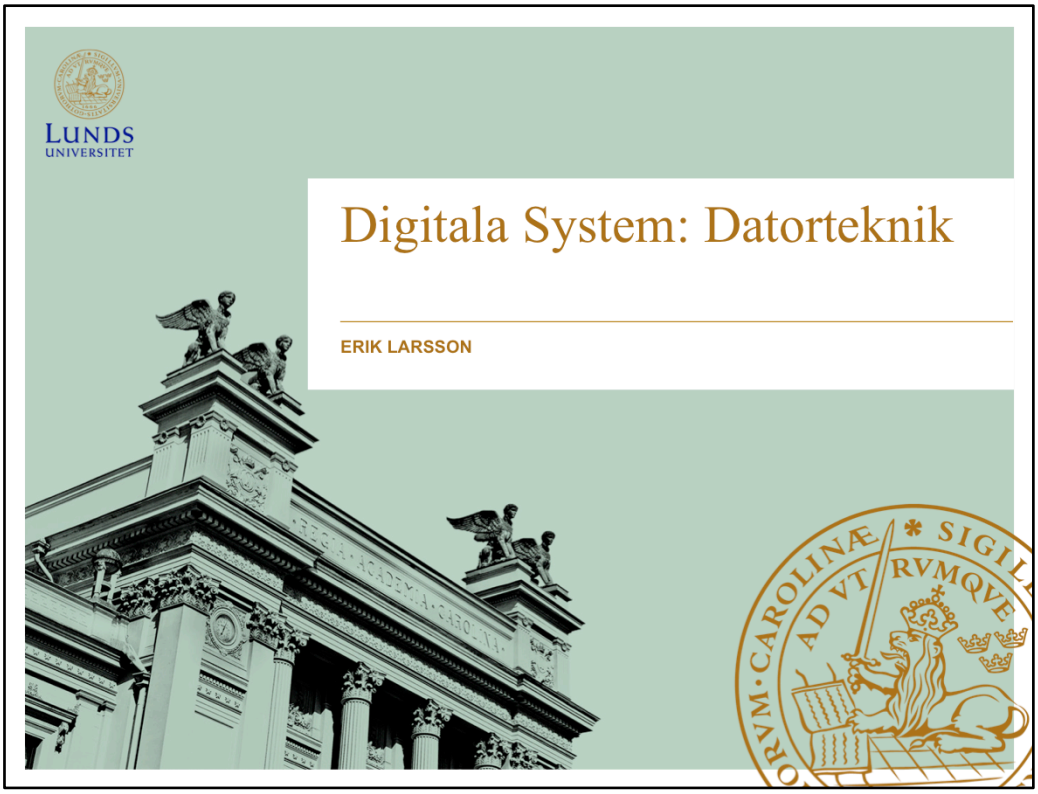
- Blooms taxonomi, lanserades av Benjamin Bloom, beskriver olika steg i lärandet:

1. Kunskap - Att kunna ta till sig fakta - memorera.
2. Förståelse - Att kunna förstå fakta som memorerats.
3. Tillämpning - Att kunna utföra något med de fakta och med hjälp av den kunskap som man fått.
4. Analys - Att kunna se samband mellan olika fakta.
5. Syntes - Att kunna dra egna slutsatser.
6. Utvärdering - Att kunna producera, planera och generera något nytt.

Benjamin Bloom konstruerade sin taxonomi (sitt klassificeringssystem) för att kunna kategorisera olika nivåer av abstraktion i lärandet. Som nybörjare sker vårt lärande helt skilt från arbetsuppgiften. Vi börjar med att lära oss begrepp, benämningar och annan fakta. Vi lär oss att återge dessa fakta, vi minns. Ju mer vi tränger in i ämnet, desto mer förstår vi av ämnet. Vi ser innebörd och samband. Nu kan vi börja tillämpa vår kunskap. Vi kan börja utföra uppgifterna. Med hjälp av instruktioner utför vi enklare uppgifter. I takt med vi klarar att utföra fler och fler uppgifter ökar vår förmåga att analysera problem. Varför fungerar inte maskinen ?

Så här långt kommer vi i de flesta kunskapsområden. Ett steg längre börjar vi kunna skapa ny kunskap. Vi konstruerar en ny krets. Slutligen uppnår vi expertis, vi har förmågan att kritiskt värdera lösningar och metoder och identifiera de bästa.

De kognitiva färdigheterna kan alltså presenteras hierarkiskt i form av en taxonomi för kognitiva mål, Blooms taxonomi.
Kunskap: Den studerande visar att han/hon kan återge faktakunskaper som han memorerat. Hjälpfrågor: Vad? Vem? Hur skulle du definiera?
Förståelse: Den studerande visar att han/hon förstår relationer och samband i det kunskapsstoff han/hon lärt sig. Hjälpfrågor: Vad är det viktigaste i artikeln? På vilket sätt skiljer sig / liknar teorierna varandra?
Tillämpning: Den studerande kan tillämpa sina kunskaper på nya, för honom relativt okända tillämpningsområden. Hjälpfrågor: Hur kan du tillämpa...?
Analys: Den studerande kan analysera fram beståndsdelar i en helhet och kan hitta det viktiga. Hjälpfrågor: Vilka bevis hittar du som stöder…?
Syntes: Den studerande kan producera och skapa något eget och relativt unikt (förena begrepp på ett nytt sätt). Hjälpfrågor: Vad händer om…? Hur kan vi förbättra / lösa problemet?
Värdering: Den studerande förmår utvärdera idéer, kunskap, metoder och lösningar. Hjälpfrågor: Vad är din åsikt om..?
Taget från: http://www.lund.se/Grundskolor/Grundskola-Gunnesbo/LaraLara---Portfolio/

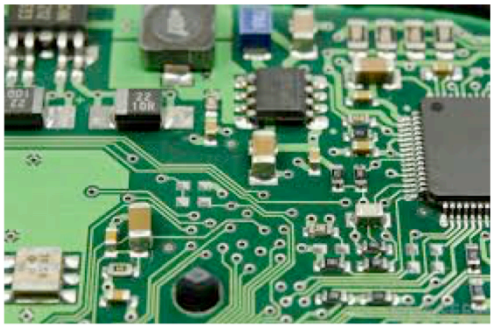

LUNDS
UNIVERSITET
Digitala System: Datorteknik
ERIK LARSSON

När man nämner datorer så tänker man först på stationära och bärbara datorer. Men datorer finns i många fler produkter. Faktum är att det är allt svårare att hitta en produkt som inte innehåller en dator. Utöver stationära och bärbara datorer är mobiltelefoner och surfplattor datorer. Kameror, tvättmaskiner, bilar, tåg, flyg och lastbilar innehåller datorer. Sedan finns det många datorer som inte syns. Till exempel när ett mobiltelefonsamtal ska kopplas fram så använder man ett telekommunikationssystem, som är en mängd datorer som ser till så att samtalet kan göras.

Man kan dela upp datorer så här:

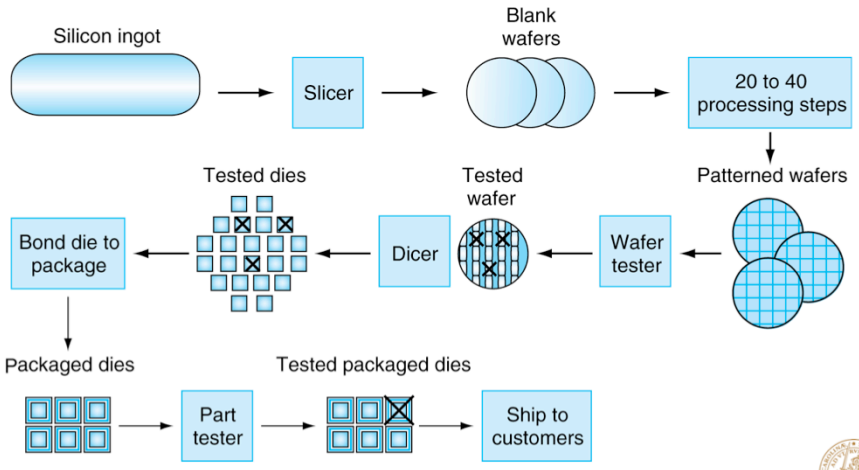
- Desktop datorer
    - Generella, olika program beroende på användare
    - Viktigt med kostnad/prestanda avvägning
- Serverdatorer
    - Nätverksbaserade
    - Hög prestanda, hög pålitlighet
    - Små servers till mycket stora system
- Inbyggda system
    - Datorn en del av systemet
    - Ofta hårda krav på effektförbrukning, prestanda/kostnad

Elektronik



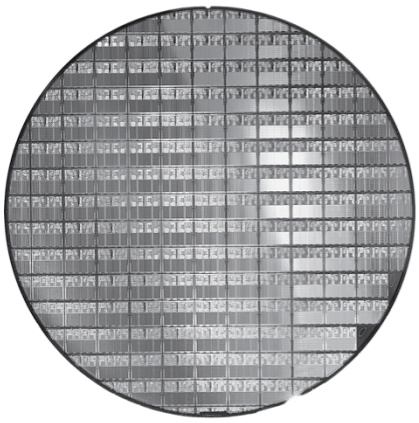
LUNDS UNIVERSITET

IC tillverkning

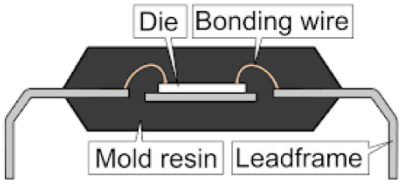


LUNDS UNIVERSITET

IC, die och wafer

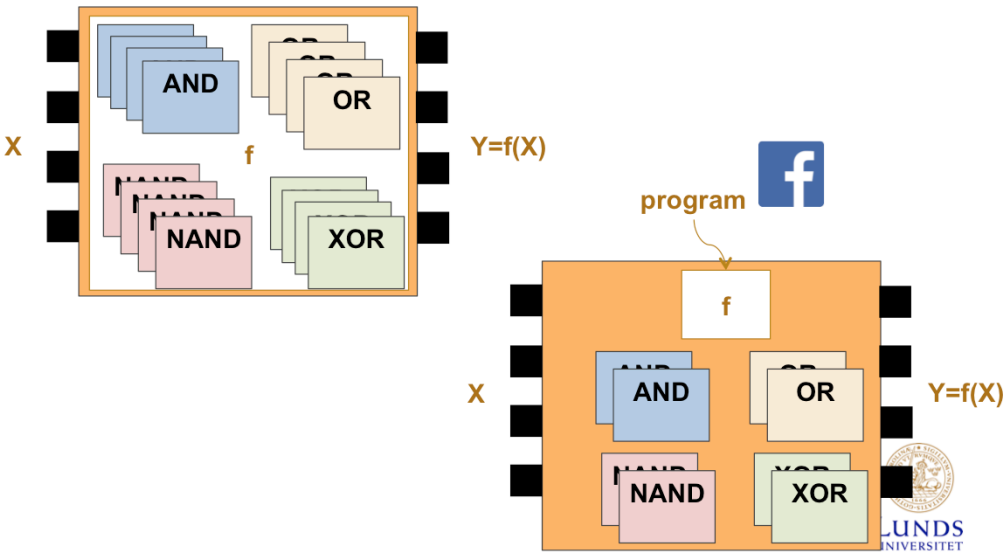


DIP



LUNDS UNIVERSITET

IC



Datorer

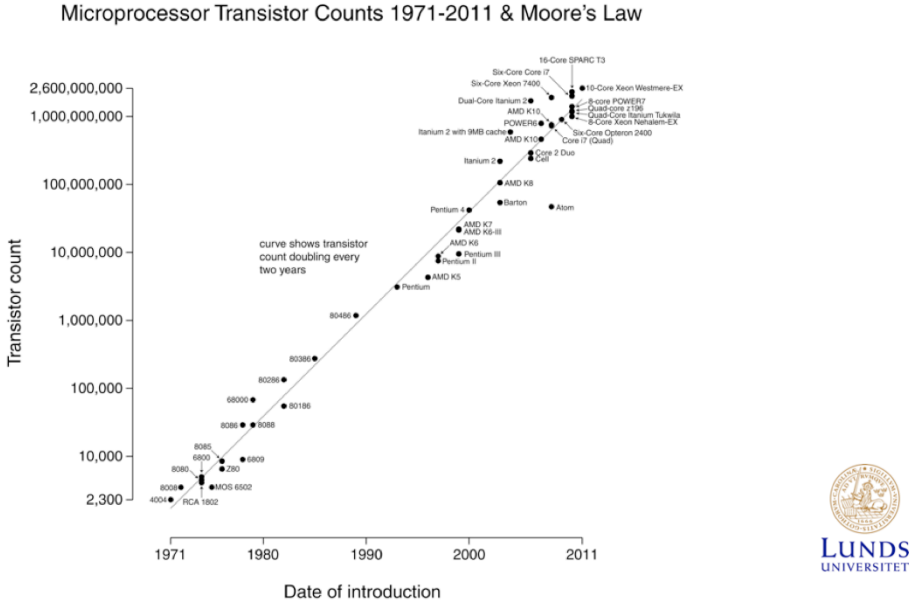
- "I think there is a world market for maybe five computers."
  -- Thomas Watson, chairman of IBM, 1943
- "Computers in the future may weigh no more than 1.5 tons."
  -- Popular Mechanics, forecasting the relentless march of science, 1949
- "There is no reason anyone would want a computer in their home."
  -- Ken Olson, president, chairman and founder of Digital Equipment Corp., 1977

Intel

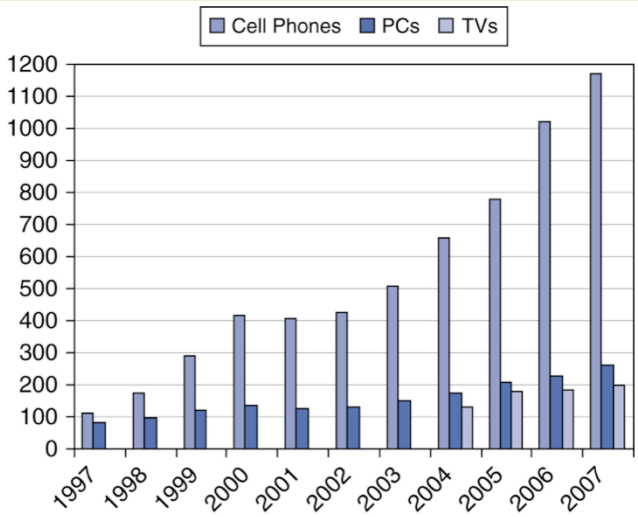
- Gordon E. Moore, co-founder of Intel (Integrated Electronics) (tillsammans med Robert Noyce)
- Första tanken på namn var: Moore & Noyce Electronics
- Moores lag (1965): Antalet transistorer på ett chip växer exponentiellt; det dubblas var 24:e månad.

LUNDS UNIVERSITET

Trender


Microprocessor Transistor Counts 1971-2011 & Moore's Law
Transistor count
2,600,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
curve shows transistor count doubling every two years
16-Core SPARC T3
Six-Core Core i7
Six-Core Xeon 7400
10-Core Xeon Westmere-EX
Dual-Core Itanium 2
8-core POWER7
Quad-core z196
AMD K10
Quad-Core Itanium Tukwila
POWER6
6-Core Xeon Nehalem-EX
Itanium 2 with 9MB cache
Six-Core Opteron 2400
AMD K10
Core i7 (Quad)
Core 2 Duo
Itanium 2
Cell
AMD K8
Barton
Atom
Pentium 4
AMD K7
AMD K6-III
AMD K6
Pentium III
Pentium II
AMD K5
Pentium
80486
80386
80286
68000
80186
8086
8088
8085
6800
6809
8080
Z80
8008
MOS 6502
4004
RCA 1802
1971
1980
1990
2000
2011
Date of introduction

Marknad för processorer

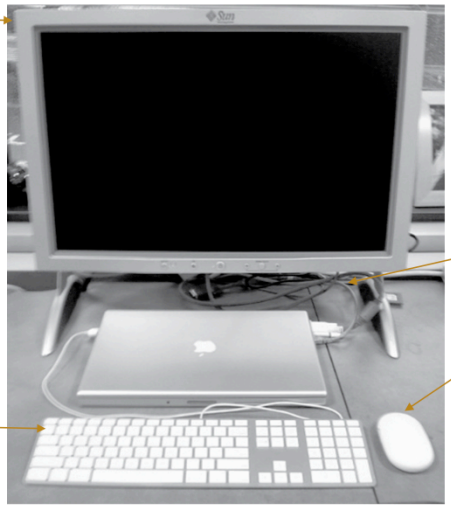


LUNDS UNIVERSITET

Datorer

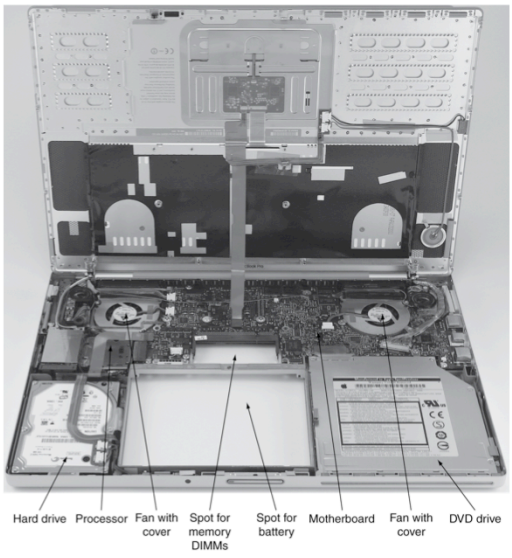
- Desktop datorer
  - Generella, olika program beroende på användare
  - Viktigt med kostnad/prestanda avvägning
- Server computers
  - Nätverksbaserade
  - Hög prestanda, hög pålitlighet
  - Små servers till mycket stora system
- Inbyggda system (Embedded computers)
  - Datorn en del av systemet
  - Ofta hårda krav på effektförbrukning, prestanda/ kostnad

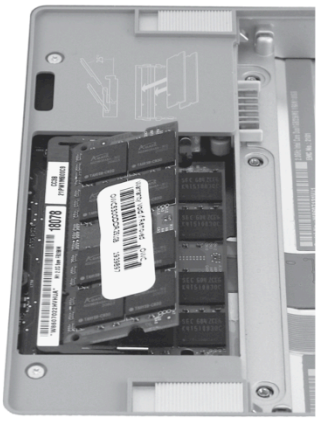
Datorns delar



LUNDS UNIVERSITET

Datorns insida

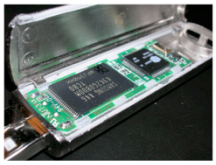




LUNDS UNIVERSITET

Lagring av data och instruktioner

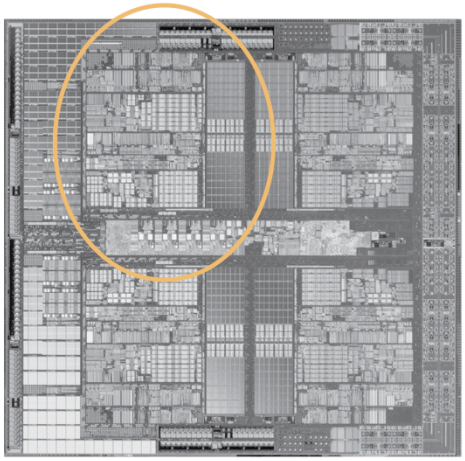
- Volatile main memory
  - Förlorar information (instruktioner och data) när strömmen stängs av
- Non-volatile secondary memory
  - Magnetic disk (hårddisk)
  - Flash memory
  - Optical disk (CDROM, DVD)

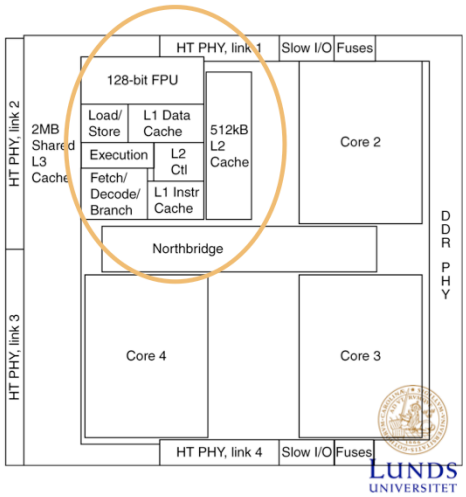


LUNDS UNIVERSITET

Processorn

- AMD Barcelona: 4 processor cores





LUNDS UNIVERSITET

Dator

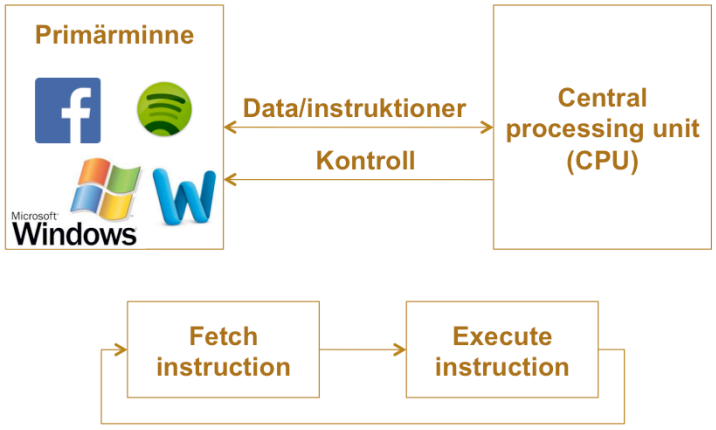
- John von Neumann arkitektur
  - Gemensamt minne för instruktioner och data (vad som är vad bestäms av kontext)
  - Aritmetisk enhet för logiska och aritmetiska operationer
  - Kontrollenhet
- Instruktioner exekveras sekventiellt

- John von Neumann (1903-1957)

Dator

- En dator är en maskin som behandlar data automatiskt under kotroll av en lista av instruktioner, så kallat program.
- Minnet är adresserbart (oberoende av vad som lagras)
- Instruktioner exekveras i ordning, så vida ordningen explicit ändras
- Programmet bestämmer vilket problem som ska lösas

Dator



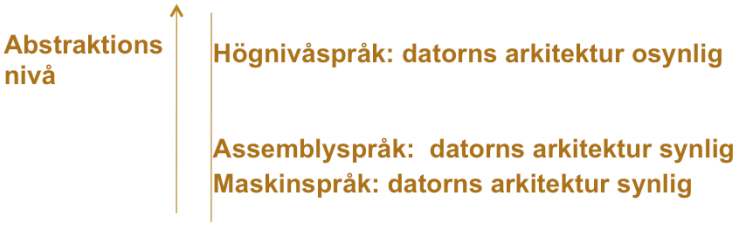
Program som ska exekveras ligger i primärminnet. Processorn hämtar instruktion för instruktion. Varje instruktion, som är ett antal 1:or och 0:or, tolkas och instruktionen exekveras.

Användarprogram, OS och hårdvara

- Applikationsprogram, t ex Word, Facebook, är ofta skrivna i ett högnivåspråk, t ex C, C++.
- System programvara
  - Kompilatorer: Översätter program i högnivåspråk till maskinspråk
  - Operativsystem:
    - » Hantera I/O, minne och schemaläggning av jobb (tasks)
- Hårdvara
  - Processor, minne, I/O-kontrollenhet

Program

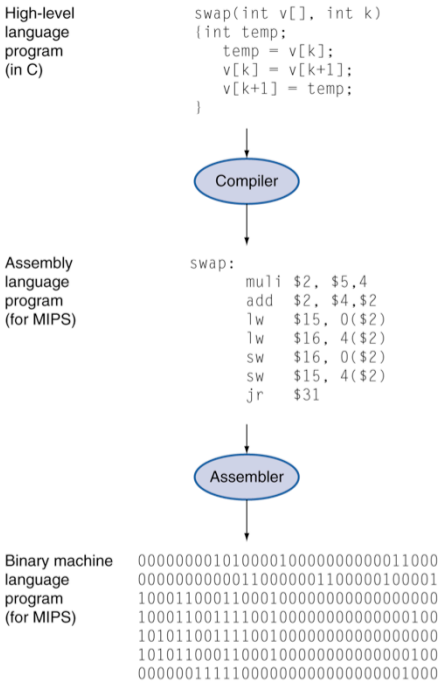
- För att bestämma vad som görs – (f(X)) – behövs program, programmeringsspråk.
- Programmeringsspråk:
  - Högnivåspråk (C, C++)
  - Assemblyspråk (t ex ADD R1, R2)
  - Maskinspråk (t ex 001101….101)



LUNDS UNIVERSITET

Program

- Abstraktionsnivå:
  - Högnivåspråk
  - Assemblyspråk
  - Maskinspråk



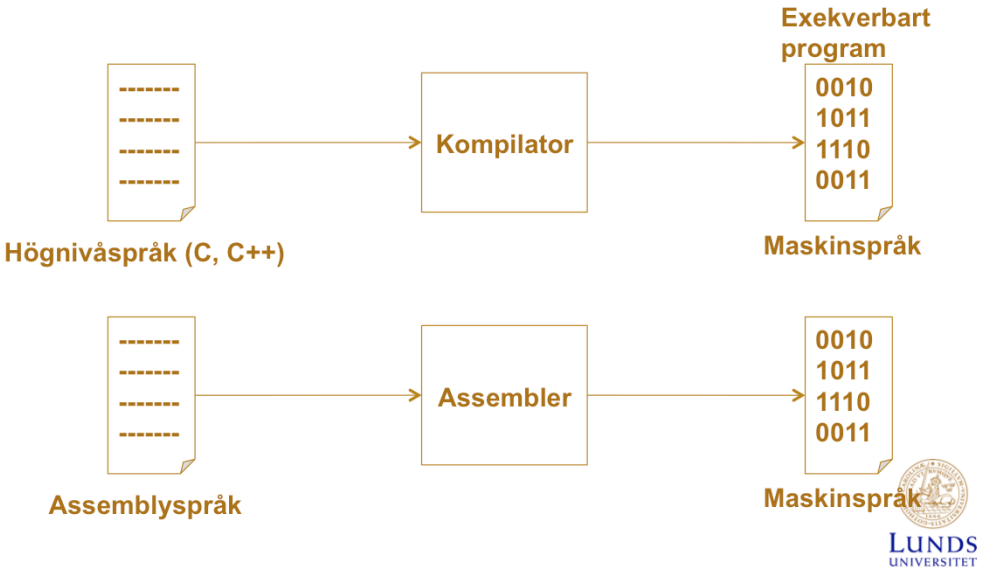
Program kan beskrivas på olika abstraktionsnivåer.

- Högnivåprogram: läsbart (för människor), hög abstraktionsnivå, enkelt att porta (flytta till en annan arkitektur), hårdvara osynlig
- Assembly- och maskinprogram: svåra att porta till annan arkitektur, hårdvara synlig

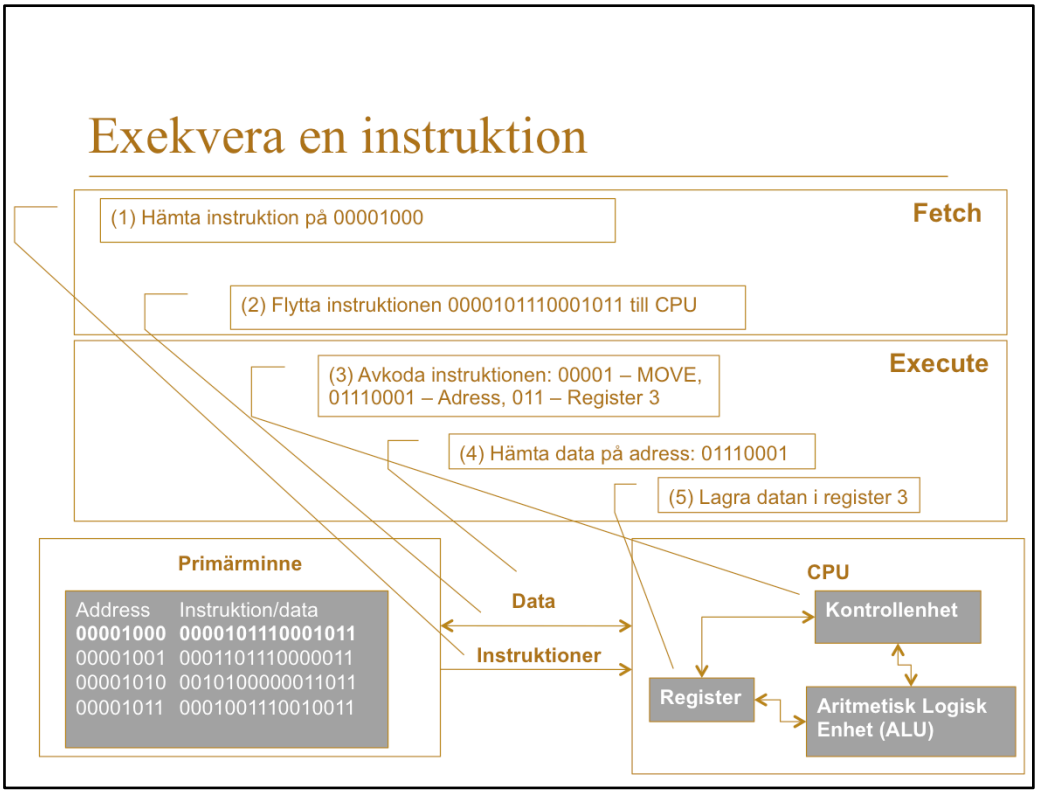
Olika processorer har olika instruktionsuppsättningar. Men med många gemensamma egenskaper. Tidiga datorer hade väldigt enkla instruktionsuppsättningar. Detta förenklade implementationen. När man bestämmer instruktionsuppsättningen tas hänsyn till hur komplext det blir och hur hårdvaran blir.
I kursboken används instruktionsuppsättning för MIPS

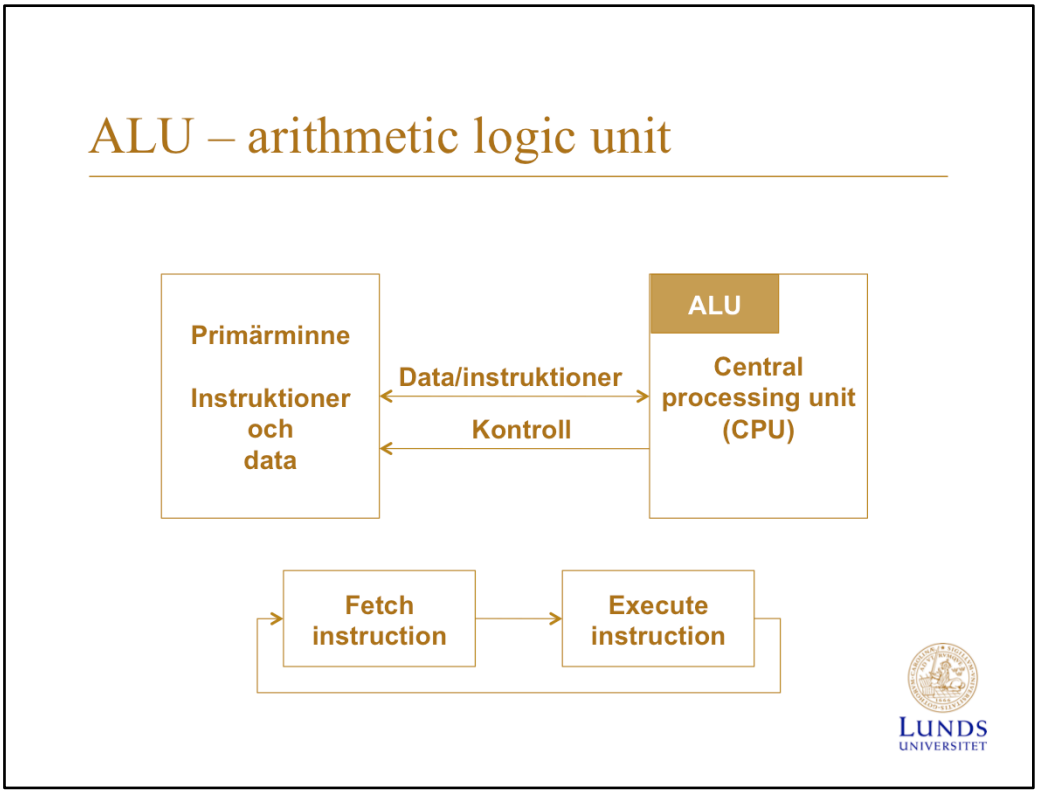
Exempel på arkitekturer:
Intel x86, IBM system/370, SUN Sparc
Exempel på organisation:
x86: Intel och AMD kan implementera en arkitektur på olika sätt (men, exekverbara program kan flyttas)

Program


Exekverbart program
Kompilator
0010
1011
1110
0011
Högnivåspråk (C, C++)
Maskinspråk
Assembler
Assemblyspråk
Maskinspråk

Exekvera en instruktion





En processor innehåller normalt ett antal register. I registren kan olika saker lagras, t ex data och instruktioner. Fördelen med register är att de är snabbare än primärminne (RAM). För att öka prestandan (tid för exekvering av ett program) kan en kompilator optimera maskinkoden så att mycket register används. Om en beräkning ska göras så kan kompilatorn välja att mellanlagra värden i register istället för att mellanlagra värden i minnet.

Register kan vara av olika typ. En del register är synliga för användare, t ex:

- Helt generella,
- Specifika för data och instruktioner
- Specifika för heltal och flyttal
- Speciella för t ex multiplikation/division för att kunna hantera resultat eller för adressberäkningar: basregister och stackpekare

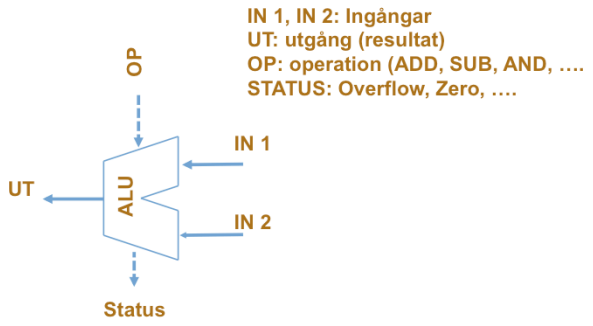
Register kan också vara av typen: Kontroll och statusregister. Dessa register är Inte direkt kontrollerbara av programmeraren, t ex:

- Programräknare (program counter (PC)): adress till den instruktion som ska hämtas.
- Instruktions register (IR): senaste instruktionen som hämtats
- Memory address register (MAR): adress som ska läsas/skrivas
- Memory buffer register (MBR): data som ska skrivas eller data som lästs från minnet
- Program status word (PSW): Olika flaggor (är avbrott på eller av, supervisor mode)

Olika processorer har olika konfiguration av register.

- Z8000: 16 generella register
- Intel 80X86, Pentium: 4 data register, 4 index&adress register, 4 bas (segment) register
- PowerPC: 2 grupper av generella register. Varje grupp innehåller 32 register och en grupp är för heltal och en är flyttal
- MIPS: 32 stycken 32-bitars register där R0=0 (viss register används till olika saker, t ex återhoppsadress lagras i R31. 2 extra register för multiplikation/division

ALU – arithmetic logic unit



LUNDS UNIVERSITET

Aritmetiska och logiska funktioner utförs av ALU. ALU:n får – via kontrollenheten (instruktionen) reda på vilken operation (OP) som ska utföras. Exempel på operationer som utförs av ALU:n är

Add, sub, mul, div, and, or, xor

En ALU tar som indata:

- IN1, IN2 och eventuella status signaler (t ex carry in)

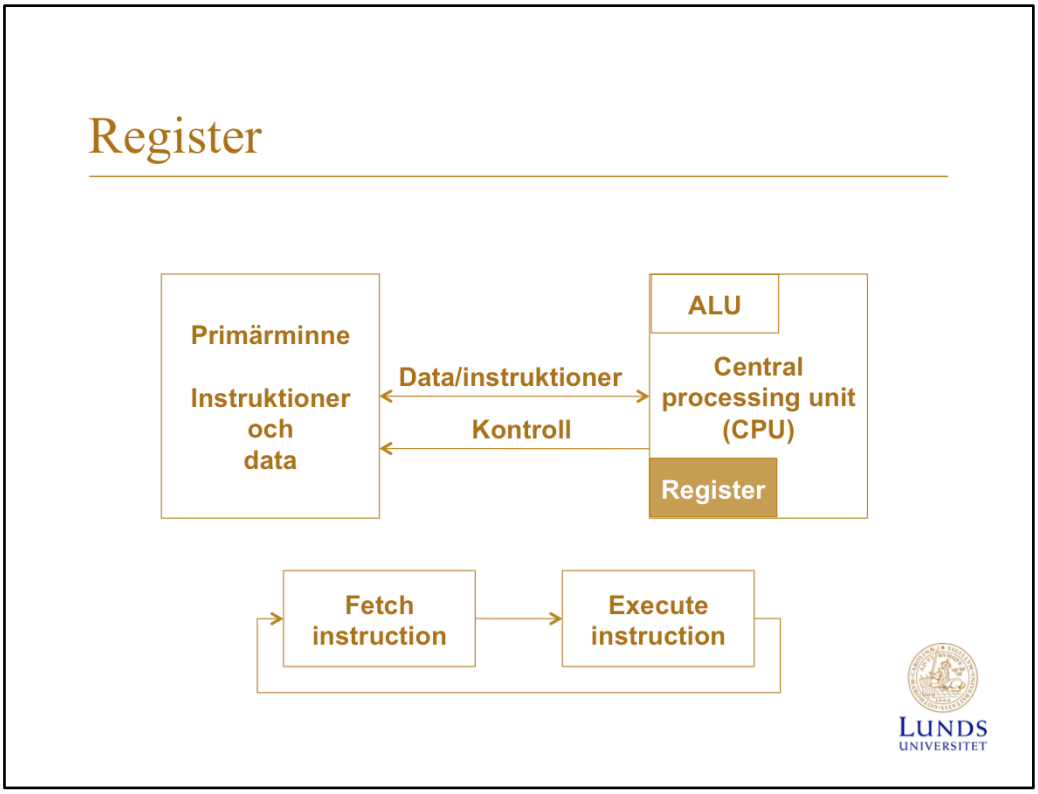
Och en ALU genererar:

- UT och eventuellt status signaler (t ex carry out)

- Bandbredd/hastighet i ALU:n bestäms av hur stora tal som kan hanteras på in och utgångar. Om 8-bitar används kan t ex heltal mellan 0-255 adderas. Fler bitar kräver större bandbredd mellan processor och register.

Beroende på vilken typ av processor, så ser ALU:n olika ut. Exempel på olika typer av processorer är:

- Central processing unit (CPU) – fokuserar på att lösa alla uppgifter
- Digital signaling processor (DSP) – fokuserar på att lösa signalbehandlings problem
- Graphics Processing Unit (GPU) – fokuserar på grafik
- Flyttals Processor Unit (FPU) – fokuserar på flyttal



En processor innehåller normalt ett antal register. I registren kan olika saker lagras, t ex data och instruktioner. Fördelen med register är att de är snabbare än primärminne (RAM). För att öka prestandan (tid för exekvering av ett program) kan en kompilator optimera maskinkoden så att mycket register används. Om en beräkning ska göras så kan kompilatorn välja att mellanlagra värden i register istället för att mellanlagra värden i minnet.

Register kan vara av olika typ. En del register är synliga för användare, t ex:

- Helt generella,
- Specifika för data och instruktioner
- Specifika för heltal och flyttal
- Speciella för t ex multiplikation/division för att kunna hantera resultat eller för adressberäkningar: basregister och stackpekare

Register kan också vara av typen: Kontroll och statusregister. Dessa register är Inte direkt kontrollerbara av programmeraren, t ex:

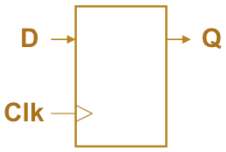
- Programräknare (program counter (PC)): adress till den instruktion som ska hämtas.
- Instruktions register (IR): senaste instruktionen som hämtats
- Memory address register (MAR): adress som ska läsas/skrivas
- Memory buffer register (MBR): data som ska skrivas eller data som lästs från minnet
- Program status word (PSW): Olika flaggor (är avbrott på eller av, supervisor mode)

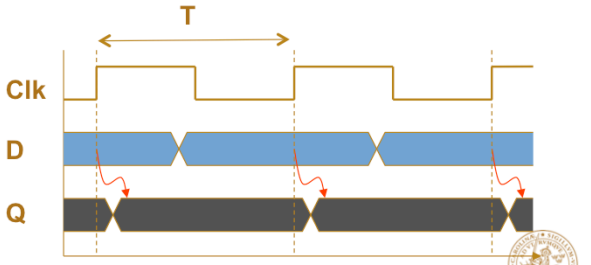
Olika processorer har olika konfiguration av register.

- Z8000: 16 generella register
- Intel 80X86, Pentium: 4 data register, 4 index&adress register, 4 bas (segment) register
- PowerPC: 2 grupper av generella register. Varje grupp innehåller 32 register och en grupp är för heltal och en är flyttal
- MIPS: 32 stycken 32-bitars register där R0=0 (viss register används till olika saker, t ex återhoppsadress lagras i R31. 2 extra register för multiplikation/division

Sekventiell logik

- Register: lagrar data
- Använder klocka för att bestämma när data ska uppdateras
- Flank-triggat (Edge-triggered): uppdaterar när Clk går från 0 till 1





LUNDS UNIVERSITET

Register

- Register synliga för användare, som kan vara:
  - Helt generella, eller
  - Specifika för data och instruktioner och/eller
  - Specifika för heltal och flyttal
  - Speciella för t ex multiplikation/division för att kunna hantera resultat eller för adressberäkningar: basregister och stackpekare

Register

- Kontroll och statusregister
  - Inte direkt kontrollerbara av programmeraren, t ex:
    - » Programräknare (program counter (PC)): adress till den instruktion som ska hämtas.
    - » Instruktions register (IR): senaste instruktionen som hämtats
    - » Memory address register (MAR): adress som ska läsas/ skrivas
    - » Memory buffer register (MBR): data som ska skrivas eller data som lästs från minnet
    - » Program status word (PSW): Olika flaggor (är avbrott på eller av, supervisor mode)

Primärminne

LOAD – läs data i minne

data läses från en given minnesplats (adress) och hamnar i processorn

STORE – skriv data i minne

data skrivs in i minnet på en given adress

Adress	Instruktion/Data
0	1111 0000
1	1010 0101
2	1100 0011
3	0011 0011
4	1111 1111
5	0000 1111
6	1111 0000
7	1010 1010

UNIVERSITET

Ett minne används för att lagra 1:or och 0:or. I minnet på bilden kan 8*8 = 64 bitar lagras. På varje plats, adress 0-7, kan 8-bitar med information lagras. På varje plats kan alltså 1 byte lagras. Till exempel, på adress 3 finns byte nummer 3 och dessa data är 0011 0011. Det som lagras i minnet är 1:or och 0:or. Om dessa 1:or och 0:or är en instruktion eller data beror på vad datorn vill att det ska vara. Om datorn vill ha en instruktion från plats 3, så kommer 00110011 tolkas som en instruktion. Om datorn vill ha data från plats 3, så kommer 00110011 tolkas som data. För att adressera ett minne med plats för 8 bytes, behövs 3 bitar (2^3=8).

Variabler och minne

- Exempel, C deklaration:

  x unsigned char; // värde mellan 0 och 255

1 byte – 8 bitar ger att dessa tal kan representeras:

Binärt	Decimalt
0000 0000	0
0000 0001	1
0000 0010	2
0000 0011	3
…….	
1111 1101	253
1111 1110	254
1111 1111	255

Variabler och minne

- Exempel, C deklaration

  x signed char; // värde mellan -128 och 127

1 byte – 8 bitar: Hur representera “-” talen?

Binärt	Decimalt
0000 0000	0
0000 0001	1
0000 0010	2
0000 0011	3
…….	
1111 1101	?
1111 1110	?
1111 1111	?

Variabler och minne

- Alternativ 1: MSB är teckenbit //Most Significant Bit
- Får 2 nollor (-0 och 0)

Binärt	Decimalt
0000 0000	0
0000 0001	1
0000 0010	2
…	….
1000 0000	-0
….	….
1111 1110	-126
1111 1111	-127

Variabler och minne

- Alternativ 2: Två-kompliment

Binärt	Decimalt
0000 0000	0
0000 0001	1
0000 0010	2
…	….
1000 0000	-127
….	….
1111 1110	-2
1111 1111	-1

- Exempel: -1 (1111 1111) + 1 = 0

Minne

- Exempel:

  C deklaration x char;
  // char behöver 1 byte

LOAD r1, 3

// ladda register r1 med data som finns på plats 3

STORE r1, 3
/ spara det som finns i register r1 på plats 3

Adress	Instruktion/Data
0	1111 0000
1	1010 0101
2	1100 0011
3	0011 0011
4	1111 1111
5	0000 1111
6	1111 0000
7	1010 1010

UNIVERSITET

Ett minne används för att lagra 1:or och 0:or. I minnet på bilden kan 8*8 = 64 bitar lagras. På varje plats, adress 0-7, kan 8-bitar med information lagras. På varje plats kan alltså 1 byte lagras. Till exempel, på adress 3 finns byte nummer 3 och dessa data är 0011 0011. Det som lagras i minnet är 1:or och 0:or. Om dessa 1:or och 0:or är en instruktion eller data beror på vad datorn vill att det ska vara. Om datorn vill ha en instruktion från plats 3, så kommer 00110011 tolkas som en instruktion. Om datorn vill ha data från plats 3, så kommer 00110011 tolkas som data. För att adressera ett minne med plats för 8 bytes, behövs 3 bitar (2^3=8).

Minne

- Exempel:

  C deklaration x int;
  // int behöver 2 bytes

LOAD r1, 3

// ladda register r1 med data som finns på plats 3 och ?

STORE r1, 3
/ spara det som finns i register r1 på plats 3 och ?

Adress	Instruktion/Data
0	1111 0000
1	1010 0101
2	1100 0011
3	0011 0011
4	1111 1111
5	0000 1111
6	1111 0000
7	1010 1010

UNIVERSITET

Ett minne används för att lagra 1:or och 0:or. I minnet på bilden kan 8*8 = 64 bitar lagras. På varje plats, adress 0-7, kan 8-bitar med information lagras. På varje plats kan alltså 1 byte lagras. Till exempel, på adress 3 finns byte nummer 3 och dessa data är 0011 0011. Det som lagras i minnet är 1:or och 0:or. Om dessa 1:or och 0:or är en instruktion eller data beror på vad datorn vill att det ska vara. Om datorn vill ha en instruktion från plats 3, så kommer 00110011 tolkas som en instruktion. Om datorn vill ha data från plats 3, så kommer 00110011 tolkas som data. För att adressera ett minne med plats för 8 bytes, behövs 3 bitar (2^3=8).

Byte eller wordadresserat minne

Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	1111 1111
5	5	0000 1111
6	6	1111 0000
7	7	1010 1010

Adress	Byte	Data
0	0	1111 0000
	1	1010 0101
	2	1100 0011
	3	0011 0011
1	4	1111 1111
	5	0000 1111
	6	1111 0000
	7	1010 1010

Byteadresserat minne:

Varje adress (0-7) pekar ut en byte som innehåller data. Till exempel, på adress 3 finns byte nummer 3 och dessa data är 0011 0011. 3-bitar behövs för att peka ut byte i minne med 8 bytes (2^3=8)

Wordadresserat minne:

Varje adress (0-1) pekar ut ett ord (word) some består av 4 bytes. Till exempel, på adress 1 finns byte 4, 5, 6, och 7. 1-bit behövs för att peka ut ord (words) i minne med 8 bytes (2^3=8)

Byte eller word adresserat minne?

- Byteadresserat
  - Måste hålla koll på vad som ska anses vara ord
  - Exempel: Läs adress 4, som är en del i ett ord.
- Wordadresserat:
  - Läs word (4 byte) per gång.
  - Fragmentering om en byte ska lagras (tar 4 bytes).

Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	1111 1111
5	5	0000 1111
6	6	1111 0000
7	7	1010 1010

UNIVERSITET

Hur sätts byte ihop till word?

- Byte 4, 5, 6, och 7 bildar ett ord, men hur ser ordet ut?
- Två alternativ:
  - 1111 1111 0000 1111 1111 0000 1010 1010 (ordning 4, 5, 6, 7)
  - 1010 1010 1111 0000 0000 1111 1111 1111 (ordning 7, 6, 5, 4)

Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	1111 1111
5	5	0000 1111
6	6	1111 0000
7	7	1010 1010

UNIVERSITET

Hur sätts byte ihop till word?

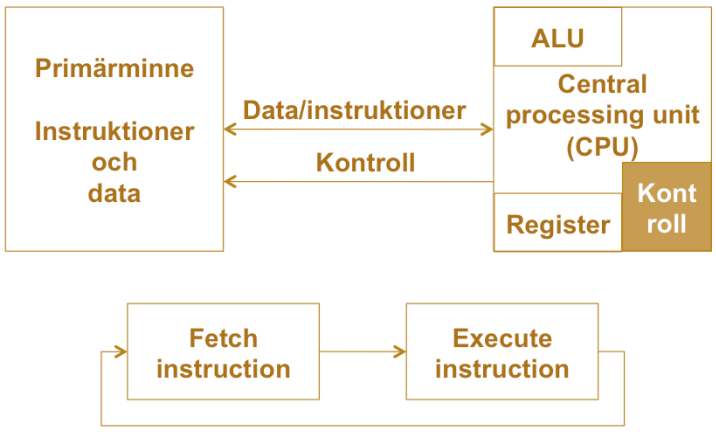
MSB LSB

Givet: **1111 1111** **0000 1111** **1111 0000** **1010 1010**

Big-endian	
Adress	Data
….	….
n	1111 1111
n+1	0000 1111
n+2	1111 0000
n+3	1010 1010
….	….

Little-endian	
Adress	Data
….	….
n	1010 1010
n+1	1111 0000
n+2	0000 1111
n+3	1111 1111
….	….

Kontrollenhet

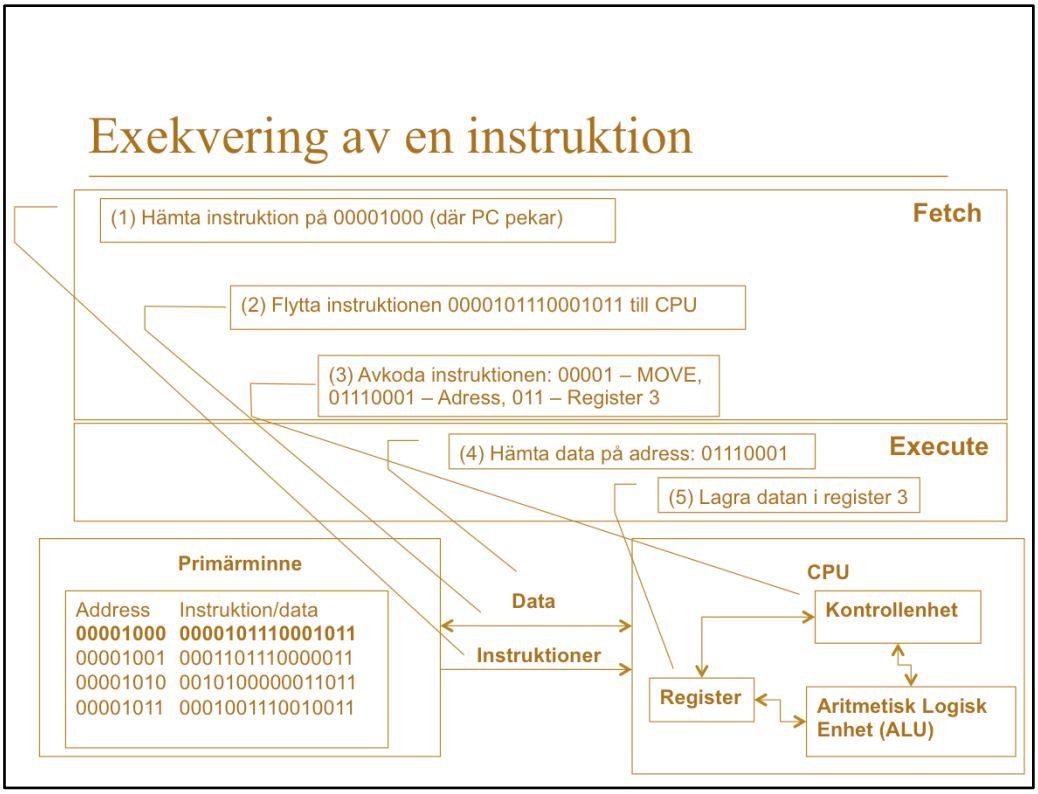


LUNDS UNIVERSITET

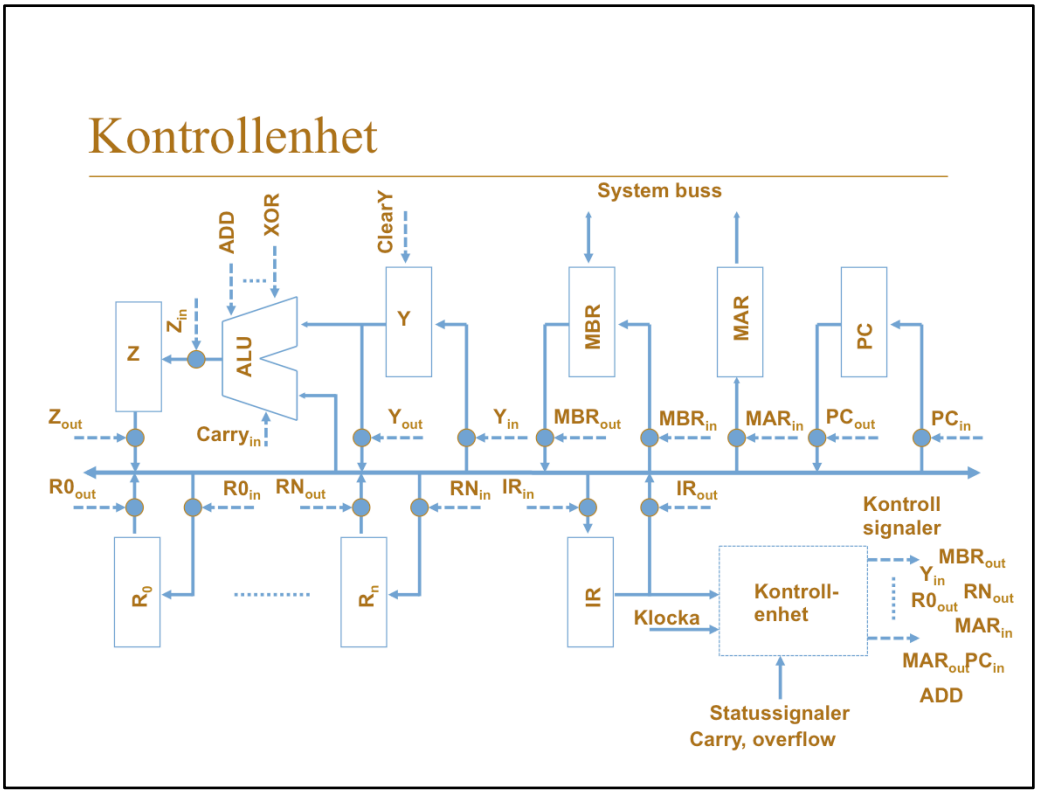
Kontrollenheten koordinerar processorn genom att varje klockcykel ge rätt kontrollsignaler. Kontrollsignalerna aktiverar mikrooperationer. En kontrollenhet kan vara hårdkodad (hardwired) eller mikroprogrammerad

- En hardwired består av kombinatorisk logik
- En mikroprogrammerad implementeras som en CPU i en CPU där varje instruktion kan ses som ett program

Hardwired är snabbare än mikroprogrammerad men svårare att implementera. Kontrollenheter blir mer och mer mikroprogrammerade.



Ett exempel på de steg som måste utföras för att exekvera en instruktion. Instruktionen finns i minnet. Programräknaren (PC) håller koll på exekveringen och talar om på vilken plats nästa instruktion ska hämtas. Processorn hämtar därför det som finns på den adress som PC innehåller. Processorn läser in den data som finns på adressplatsen och placerar bitarna i instruktionsregisteret (IR). IR är ett register som innehåller den instruktion som just nu exekveras. Bitarna i IR kollas av för att ta reda på vad det är för instruktion (avkodning). I det här fallet är den en flytt av data från en viss plats i minnet till ett register. Processorn läser därför i minnet igen. De bitar som läses in är data och lagras i ett register. I detta fall register 3.



En något mer detaljerad bild av en processor. De tre delarna i processorn är:

- Nere till vänster finns ett antal register som används för att lagra data.
- Överst till vänster sitter ALU:n. ALU:n har två ingångar och en utgång. ALU:n har också ett anta kontrollingångar som används för att bestämma vad processorn ska göra (addition, subtraktion, etc).
- Nere till höger finns kontrollenheten. Kontrollenheten tar ett antal insignaler och genererar ett antal utsignaler. Insignaler är statussignaler från ALU:n, delar av IR (där instruktionen som ska exekveras lagras) och en klocksignal (som bestämmer hastigheten). Utsignaler är kontrollsignaler för att styra processorn. Det är signaler som bestämmer vad som ska finns på den interna bussen. Om innehållet i programräknaren ska placeras i memory address register (MAR) så ska kontrollenheten se så att PCout och MARin är öppna. Då kommer det som finns i PC att flyttas till MAR.

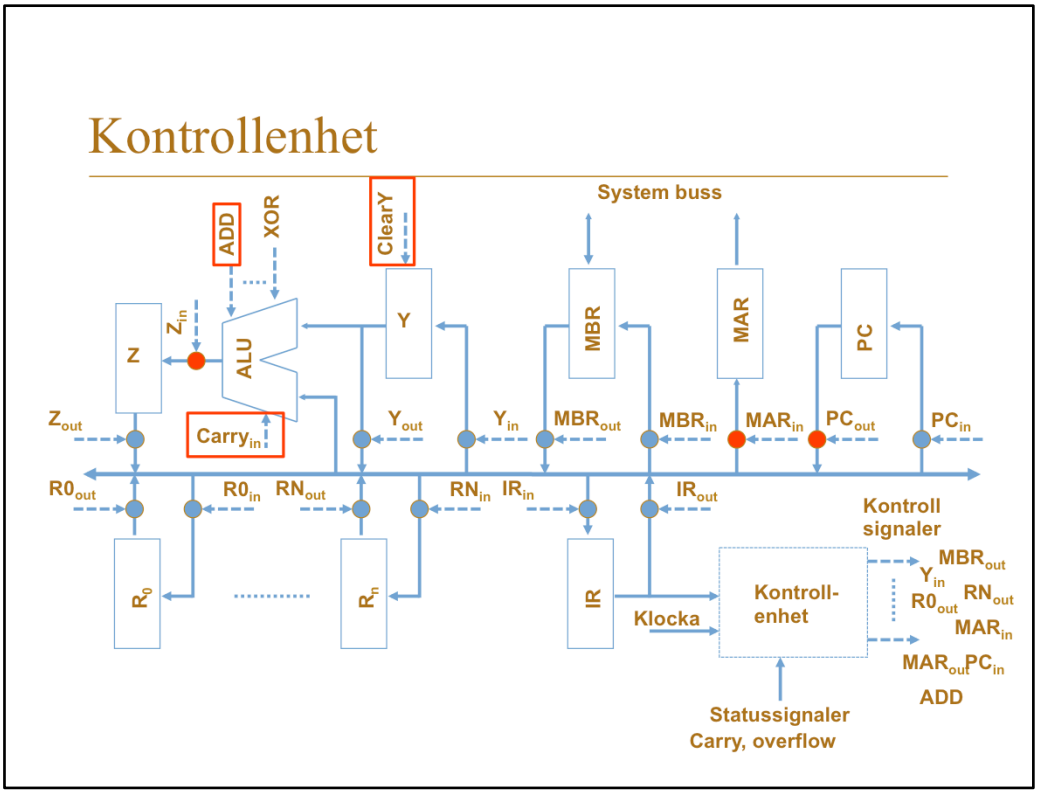
Eftersom den här processorn bara har en intern buss som kopplar samman allt så kan bara en sak göras på den internabuss. Det blir naturligtvis en konflikt om kontrollenheten öppnar PCout och MBRout samtidigt (det är då flera som ”driver” ledningarna i bussen).

Kontrollenhet

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

Antag att processorn ska exekvera instruktionen ADD R1, R3. När instruktionen är exekverad så att processorn tagit innehållet i R1 och R3 och med hjälp av ALU:n är värdena adderade och resultatet är skrivet i R3.
I det första kontrollsteget tas innehållet i PC och placeras i MAR. Det är för att processorn vill hämta den instruktion som finns på den plats som pekas ut av PC (det är så långt exekveringen har nått). Kontrollsignalen READ (läs i minnet) aktiveras så att minnet hämtar fram rätt data. När innehållet av PC är på den internabussen räknas PC upp ett steg för att förbereda hämtning av nästa instruktion. Så, ena ingången av ALU:n har värdet PC. Om Y nollställs, carry-in sätts, och ALU:n ställs på addition så har PC+1 beräknats och resultatet placeras i Z.

Notera. I exemplet som används för att illustrera principen om hur PC räknas upp kan man faktiskt läsa ut en hel del om processorn. För det första, om PC alltid räknas upp med 1 steg så är alla instruktioner av samma längd, dvs fixed length. För det andra, om minnet är adresserat på byte nivå, dvs den minsta enhet man kan adressera är 1 byte (8-bitar), så kan det inte finnas mer än 2^8 (256) instruktioner.

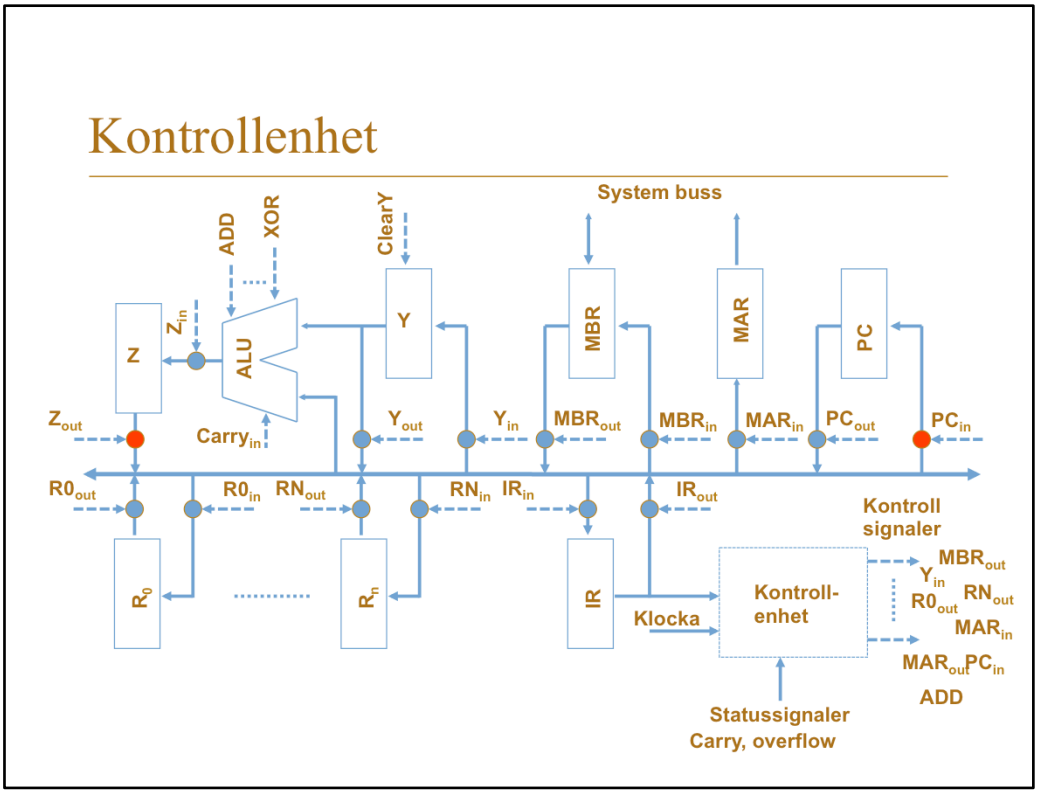


Här ser vi vilka kontrollsignaler som är aktiva i det första kontrollsteget.

Kontrollenhet

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. ~~Z_{out}, PC_{in}~~
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

I det andra kontrollsteget laddas PC med det nya värdet (det som beräknades i kontrollsteg 1)

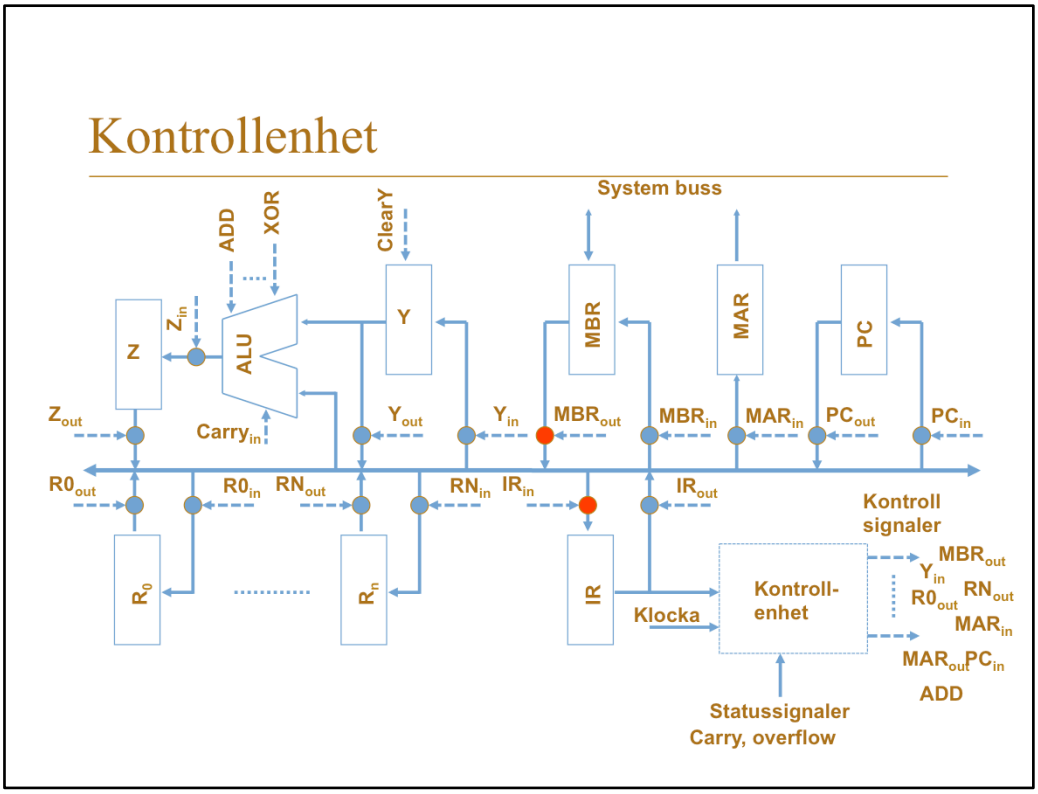


Här ser vi vilka kontrollsignaler som är aktiva i det andra kontrollsteget.

Kontrollenhet

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

I tredje kontrollsteget är minnet klar, dvs minnet har levererat den data som finns på den plats som PC pekade ut. MBRout aktiveras för att det inläsa värdet ska flyttas via den internabussen till instruktionsregistret (IR). IRin aktiveras så att det som finns på den internabussen läses in.

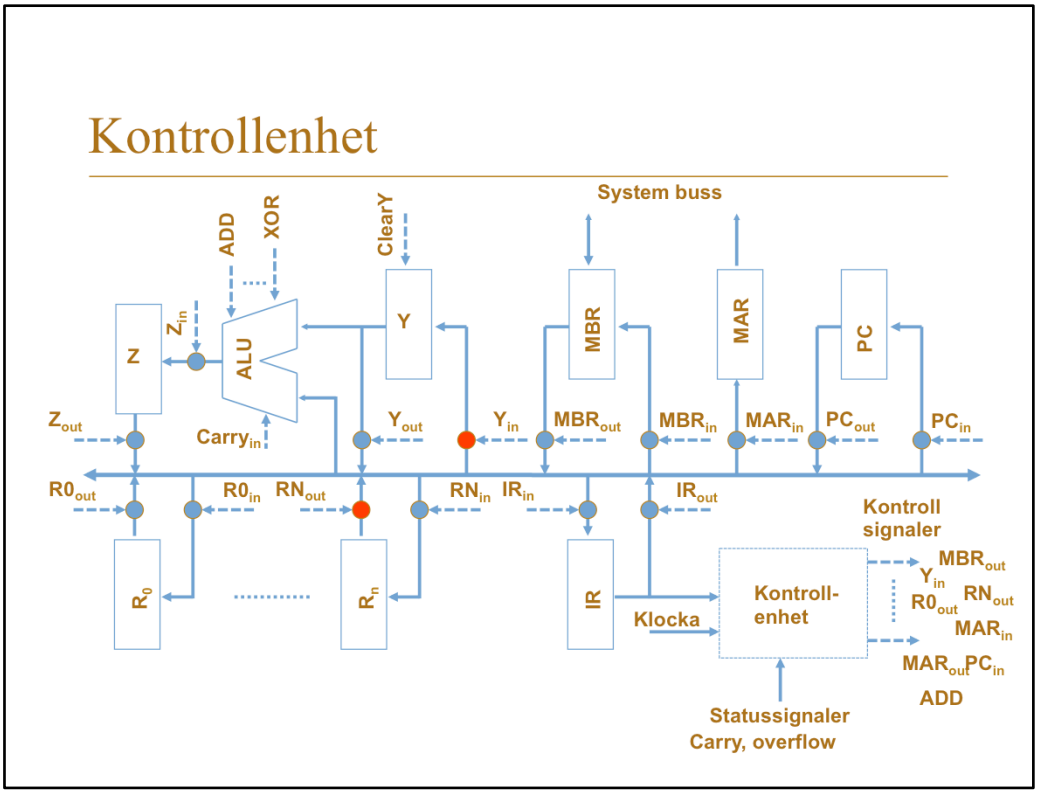


Här ser vi vilka kontrollsignaler som är aktiva i det tredje kontrollsteget.

Kontrollenhet

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, ~~Y_{in}~~
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

Nu är FETCH delen avklarad, dvs den del som är samma oavsett vilken instruktion det är. I EXECUTE delen, beror kontrollenheten på vilken instruktion som just nu finns i instruktionsregistret. I detta exempel är det en addition. Den kan inte göras i ett steg utan måste brytas upp i 3 steg. I det första (av de tre stegen) tas det värde som finns i R1 och flyttas till det temporära registret Y. Detta görs genom att kontrollsignalerna R1out och Yin aktiveras.

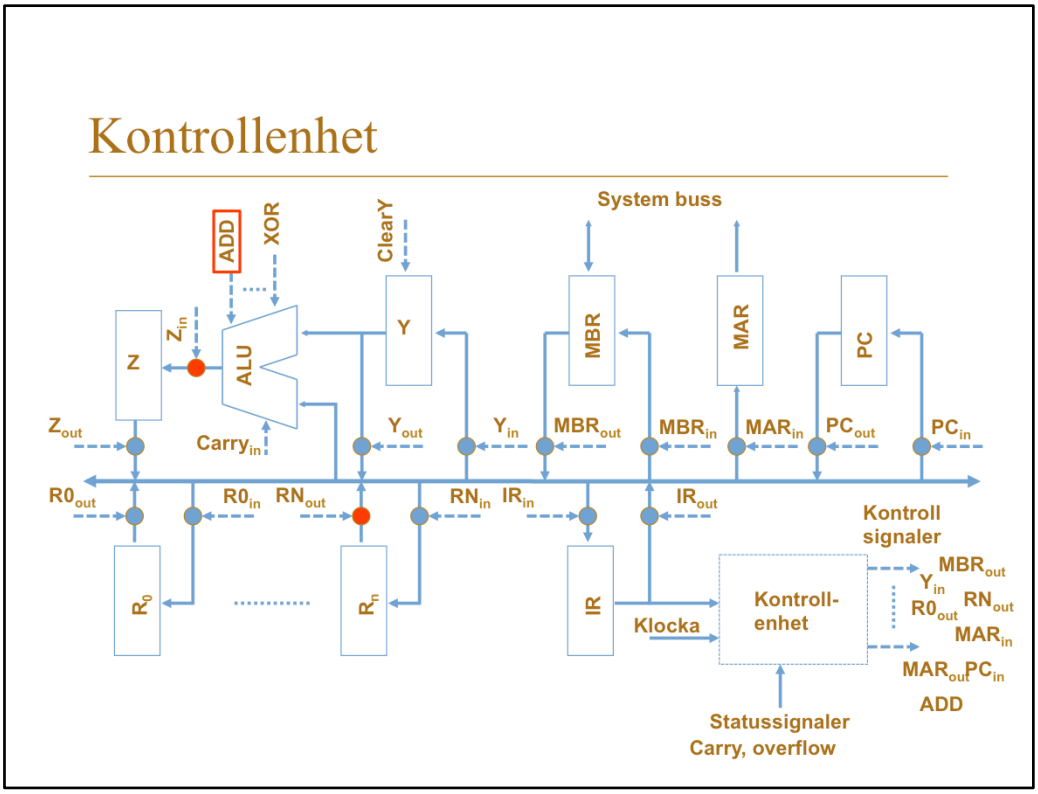


Här ser vi vilka kontrollsignaler som är aktiva i det fjärde kontrollsteget.

Kontrollenhet

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

I det femte kontrollsteget görs additionen. Värdet från R1 är redan kopierat till Y. Nu placeras värdet i R3 på andra ingången till ALU:n och ALU:n gör en addition genom kontrollsignalen ADD. Resultatet lagras i det temporäraregistret Z.



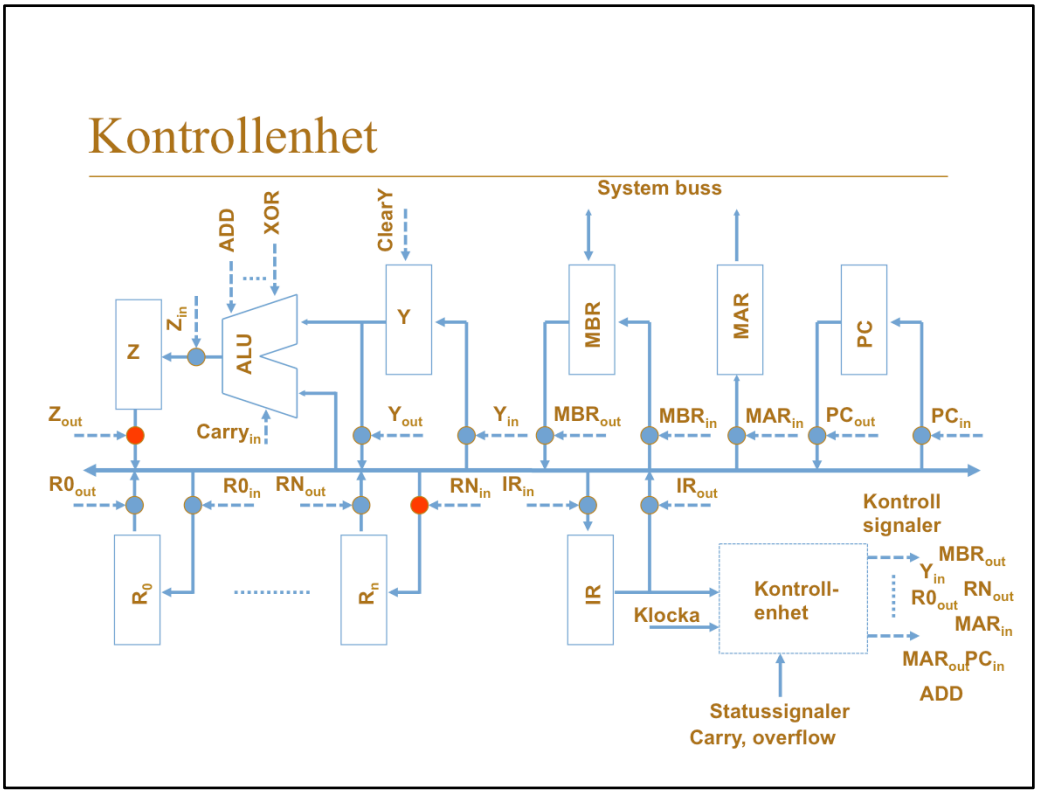
Här ser vi vilka kontrollsignaler som är aktiva i det femte kontrollsteget.

Kontrollenhet

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

I det sista kontrollsteget flyttas resultatet från Z till R1. Instruktionen är nu exekverad. Allt det som gjorts, kontrollsteg 1-6, tillhör instruktionscykeln.

Notera. Den här instruktionen tog 6 kontrollsteg att exekvera. Om varje kontrollsteg tar 1 klockcykel så tar denna instruktion 6 klockcykler. Prestandan kan ökas genom att öka klockfrekvensen men det gäller att vara vaksam så att den logik som måste exekvera i varje steg hinner med. Ett alternativ till att öka klockfrekvensen är att försöka minska antalet kontrollsteg. Det kan göras genom att försöka optimera det som görs i varje kontrollsteg. Det kan också göras genom att försöka bygga bort flaskhalsar. I den här arkitekturer fanns en internbuss. Den begränsade möjligheterna att snabbt flytta data, t ex additionen delades upp i tre steg eftersom det bara fanns en internbuss (vilket gör att bara en sak i taget kan göras).



Här ser vi vilka kontrollsignaler som är aktiva i det sjätte kontrollsteget.

Exekveringstid

- Antal klockcykler för att exekvera en maskininstruktion
  - Clocks per instruction (CPI)
- Tid för en klockcykel
  - Tid för en klockperiod (T)
  - Frekvens (f) är: f=1/T
- Antal maskininstruktioner
  - Instruction count (IC)
- Exekveringstid i klockcykler = CPI x T x IC

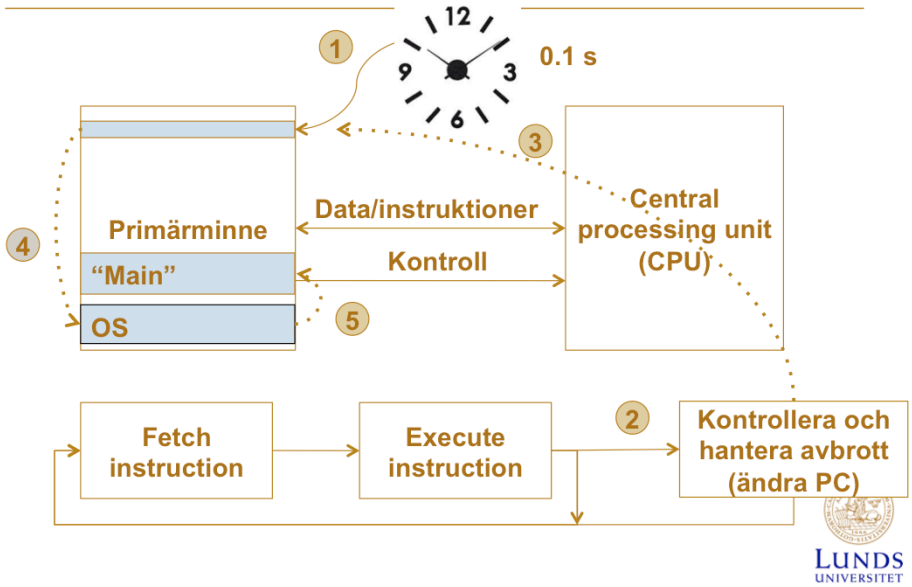
Exekveringstid

- Prestanda kan ökas genom:
  - Öka klockfrekvensen (f) (minska T)
  - Minska antal instruktioner (IC)
  - Minska antal klockcykler per instruktion (CPI)

Prestanda

- Algoritm
  - Bestämmer vilka och hur många *operationer* som ska utföras
- Programmeringsspråk, kompilator, arkitektur
  - Bestämmer hur många *maskininstruktioner* som ska utföras per *operation*
- Processor och minnessystem
  - Bestämmer hur snabbt instruktioner exekveras
- I/O (Input/Output) och operativsystem
  - Bestämmer hur snabbt I/O operationer ska exekveras

Avbrott



LUNDS UNIVERSITET

LUNDS
UNIVERSITET