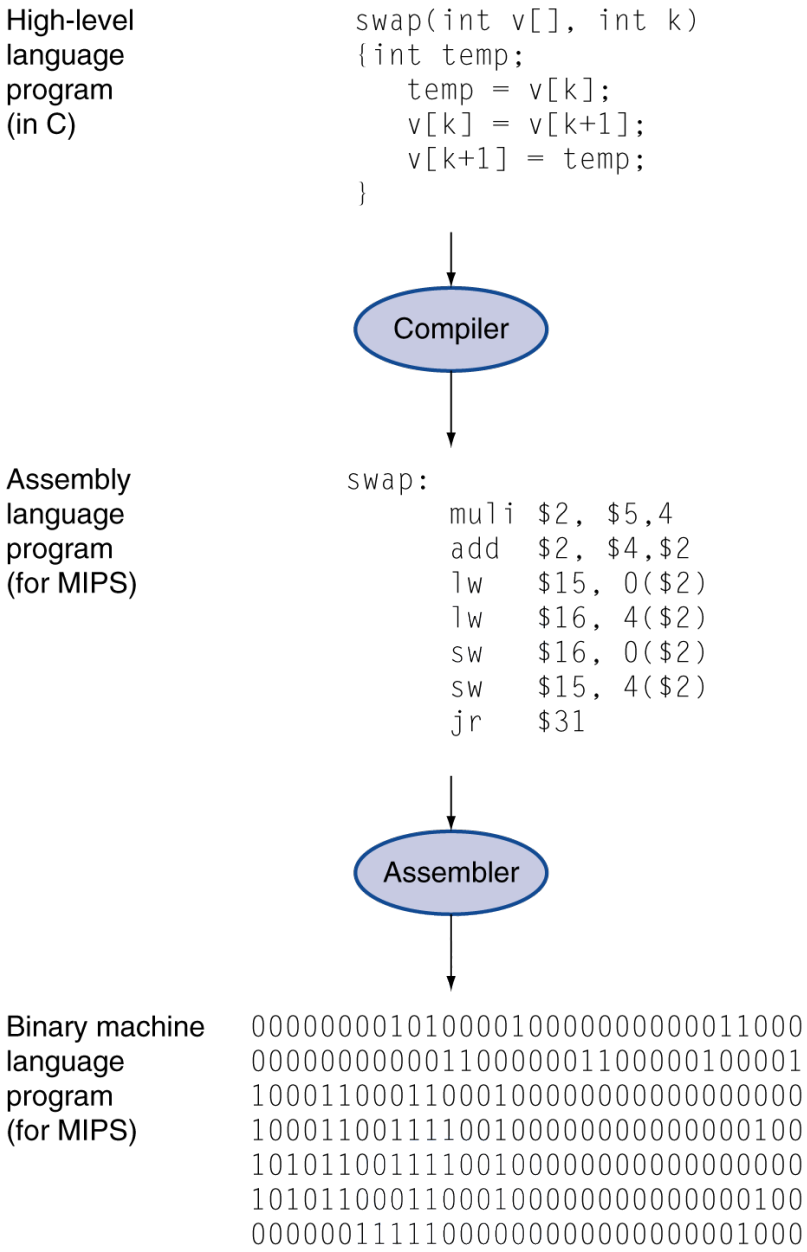
LUNDS UNIVERSITET

Datorteknik

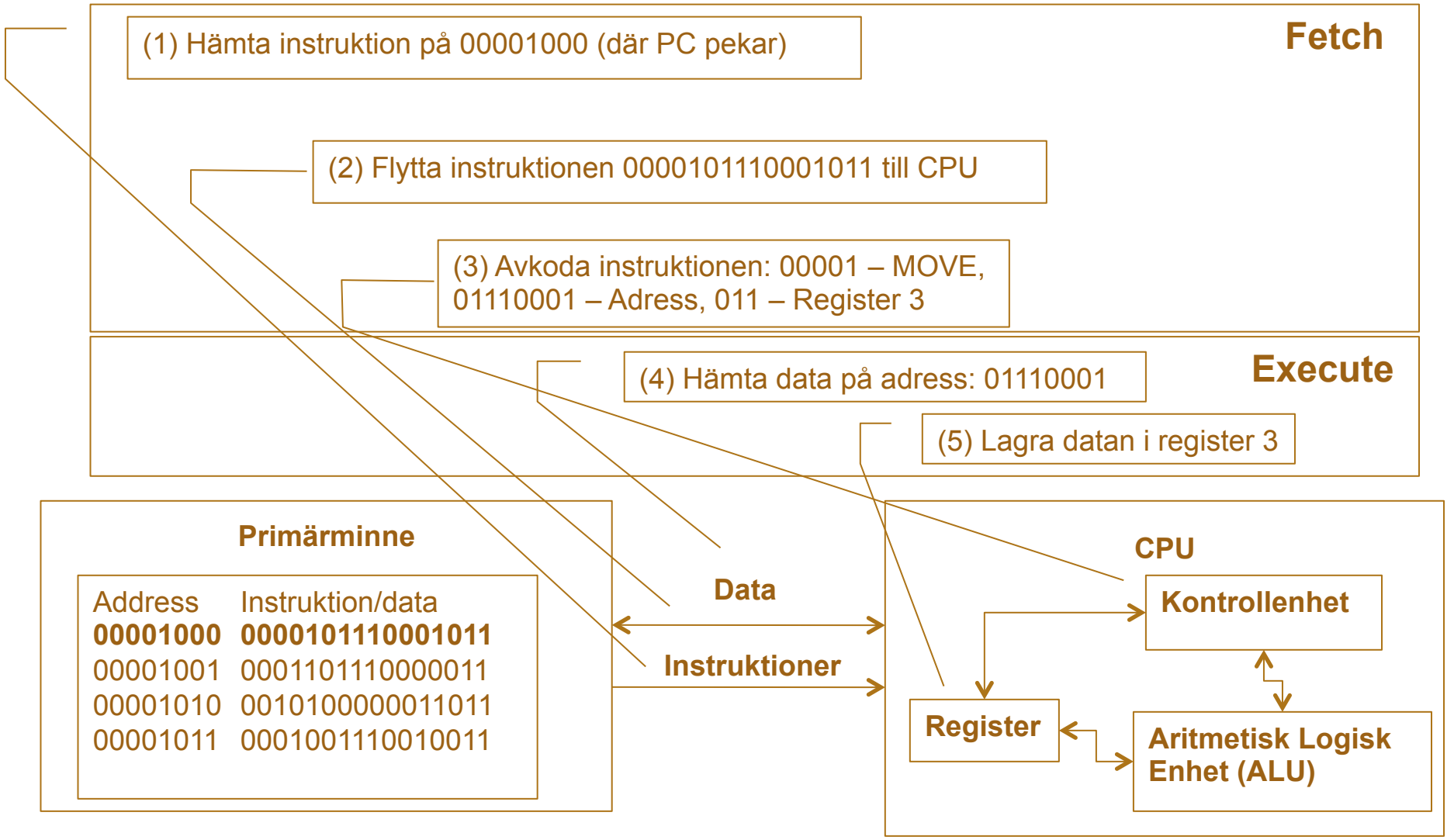
ERIK LARSSON

Program

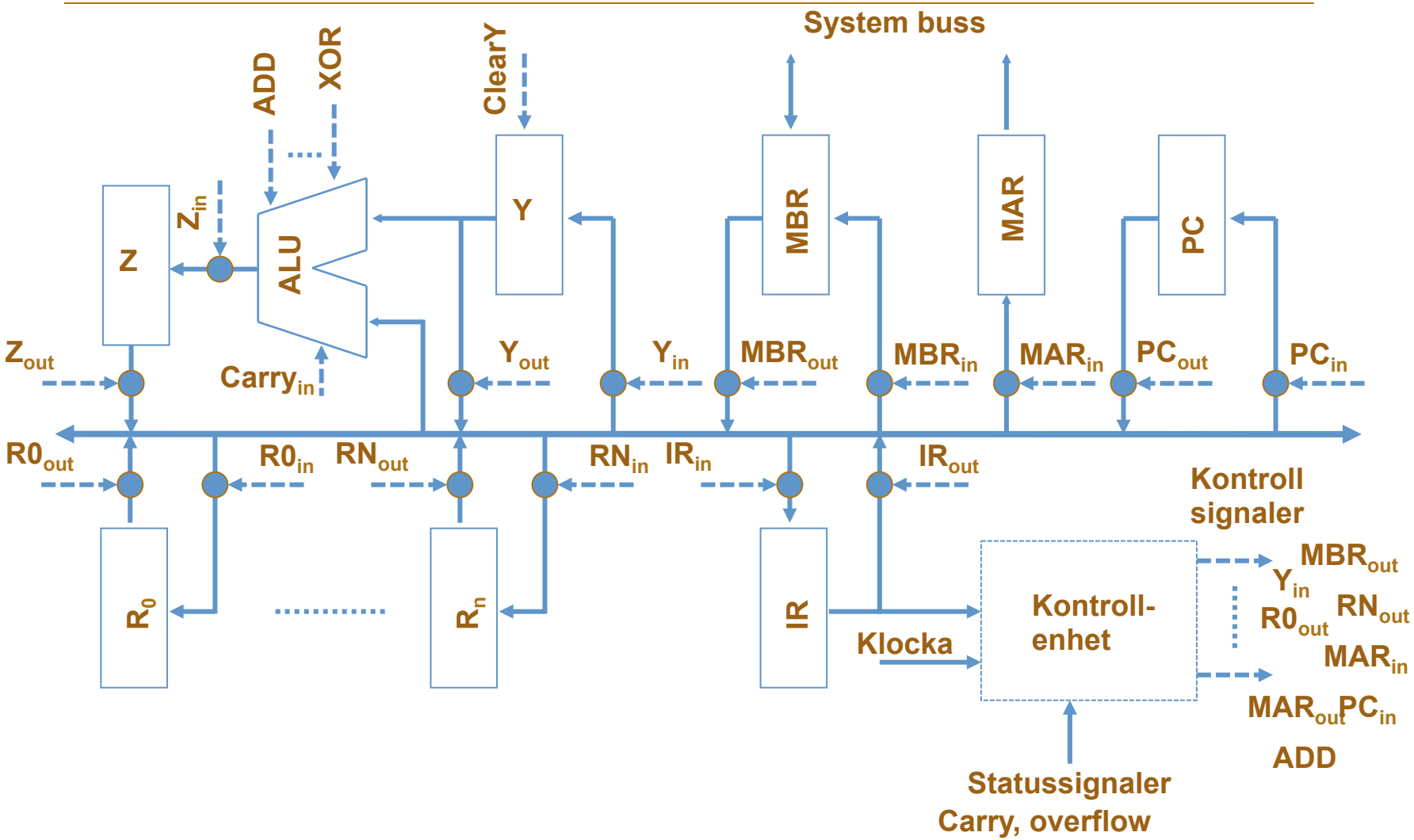
- Abstraktionsnivå:
  - Högnivåspråk
    - » t ex C, C++
  - Assemblyspråk
    - » t ex ADD R1, R2
  - Maskinspråk
    - » t ex 001101….101



Exekvering av en instruktion



Kontrollenhet



Exekveringstid

- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg = Klockcykler per instruktion (CPI)
  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

LUNDS UNIVERSITET

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

LUNDS UNIVERSITET

Exekveringstid

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

LUNDS UNIVERSITET

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

LUNDS UNIVERSITET

Maskininstruktioner

- Typer av instruktioner:
  - Aritmetiska och logiska (ALU)
  - Dataöverföring
  - Hopp
  - In- och utmatning

LUNDS UNIVERSITET

Maskininstruktioner

- Definitioner:
  - Vad ska göras (operationskod)?
  - Vem är inblandad (source operander)?
  - Vart ska resultatet (destination operand)?
  - Hur fortsätta efter instruktionen?

LUNDS UNIVERSITET

Maskininstruktioner

- Att bestämma:
  - Typ av operander och operationer
  - Antal adresser och adresserings format
  - Registeraccess
  - Instruktionsformat
    - » Fixed eller flexibelt

LUNDS UNIVERSITET

Aritmetiska operationer

- Addition (+) och subtraktion (-)
  - Två källor (sources) och en destination (destination)

```
add a, b, c // a = b + c
```

- Alla aritmetiska funktioner följer detta mönster
- Design regel: Enkelhet föredrar regelbundenhet
  - Regelbundenhet gör implementation enklare
  - Enkelhet ökar möjligheten till högre prestanda till lägre kostnad

LUNDS UNIVERSITET

Aritmetiska operationer

- C kod:

```
f = ( g + h ) - ( i + j );
```

- Kompileras till assemblykod (MIPS):

```
add t0, g, h        # temp t0 = g + h
add t1, i, j        # temp t1 = i + j
sub f, t0, t1       # f = t0 - t1
```

LUNDS UNIVERSITET

Adressering

- Immediate adressering

```
ADD R4,#3       //R4<-R4+3
```

Operanden finns direkt i instruktionen

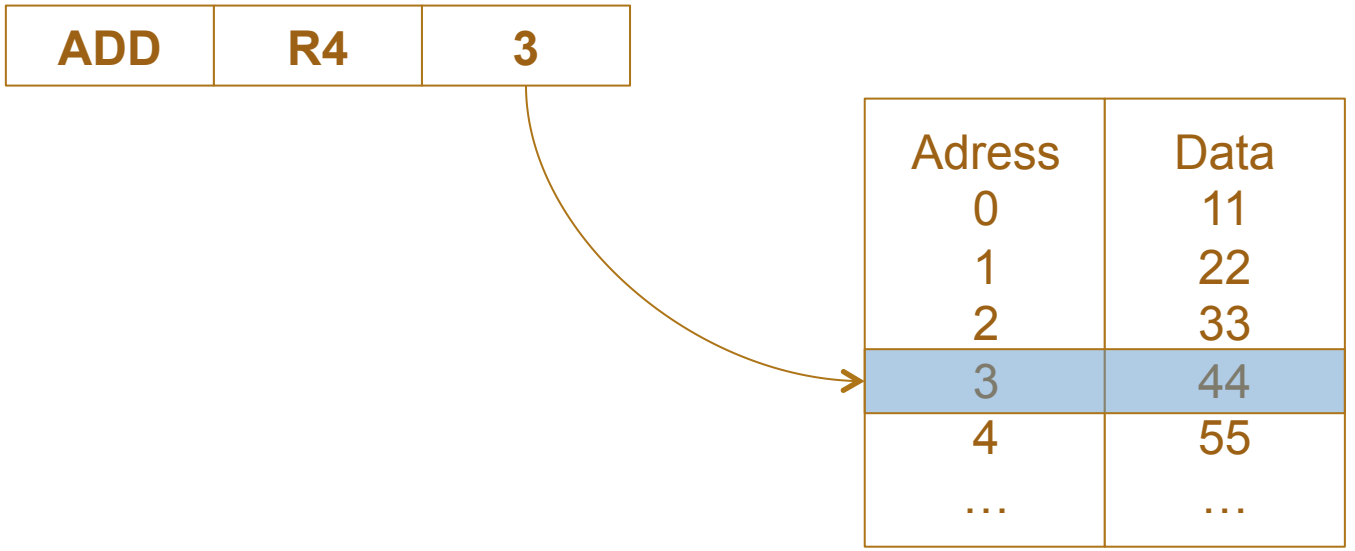
ADD	R4	3

Adressering

- Direkt adressering

```
ADD R4, 3      //R4<-R4+[3]
```

Adressen till operanden ligger i instruktionen



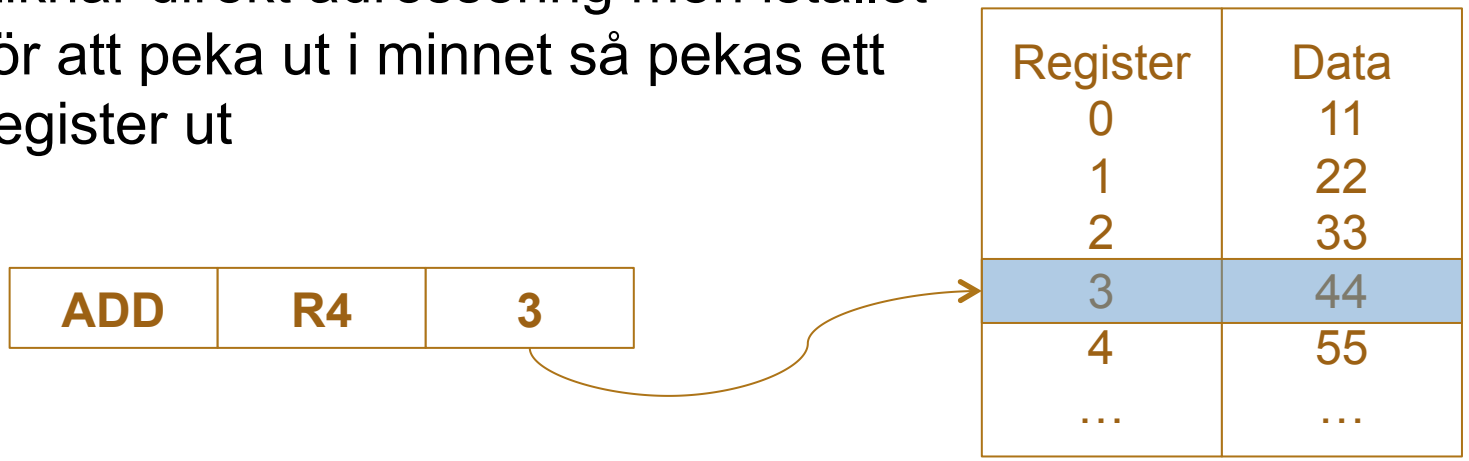
LUNDS UNIVERSITET

Adressering

- Register adressering

```
ADD R4,R3      //R4<-R4+R3
```

Liknar direkt adressering men istället för att peka ut i minnet så pekas ett register ut

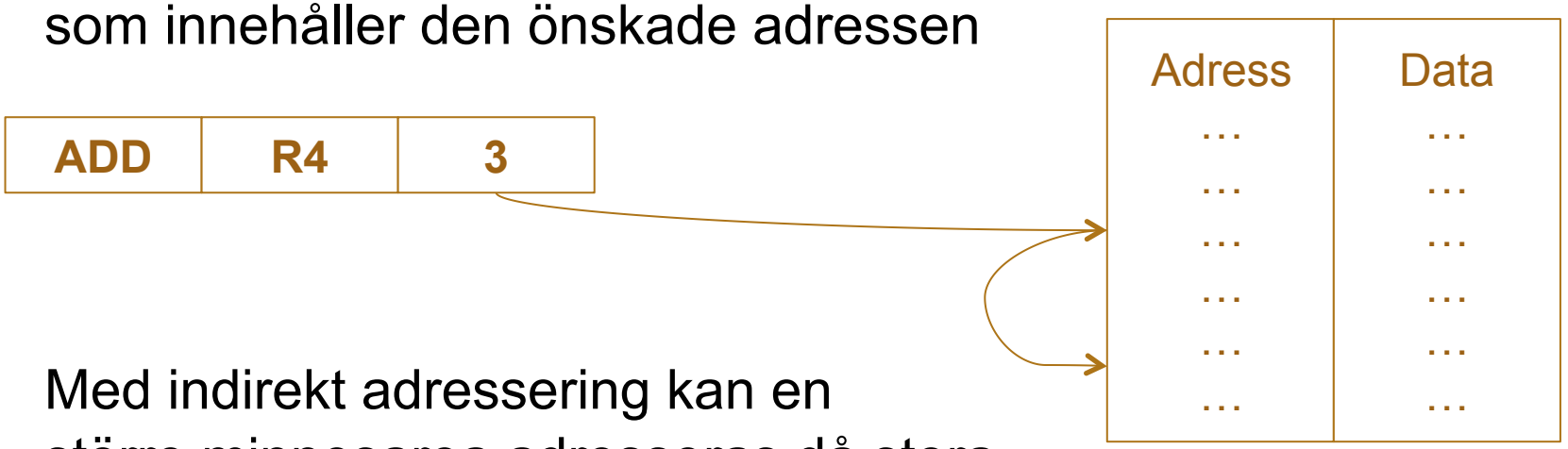


Adressering

- Memory indirect

  `ADD R4,(3)  R4<-R4+[[3]]`

  Instruktionen innehåller adressen till en minnesplats som innehåller den önskade adressen



  Med indirekt adressering kan en större minnesarea adresseras då stora adresser kan läggas i minnet och instruktionen pekar ut dem.

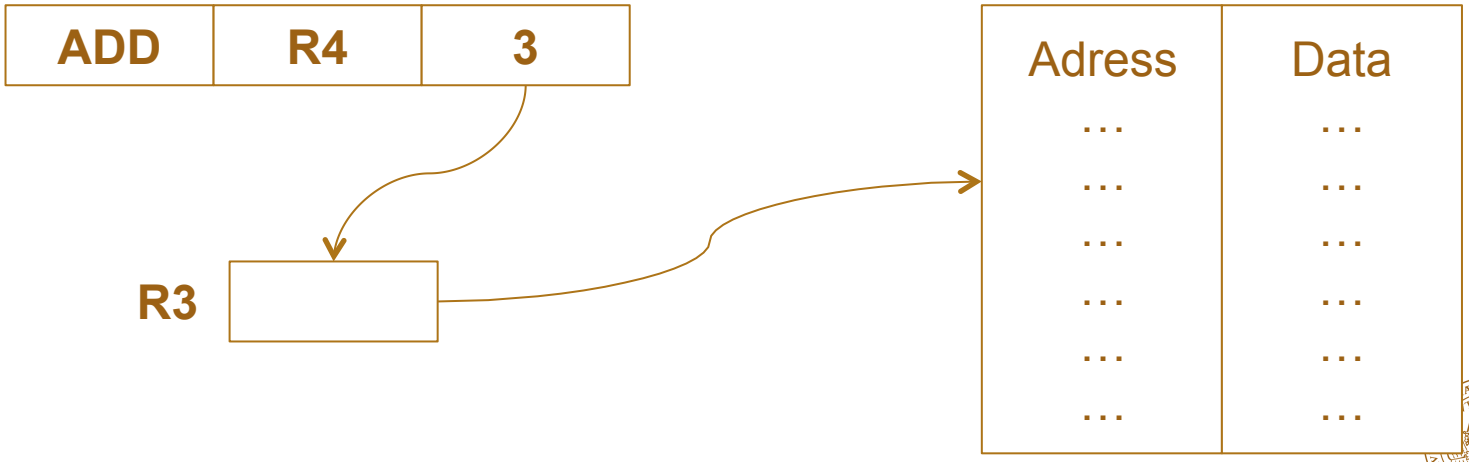
LUNDS UNIVERSITET

Adressering

- Register indirect

```
ADD R4,(R3)    R4<-R4+[R3]
```

Liknar indirekt adressering men instruktion pekar ut register istället för minnesplats



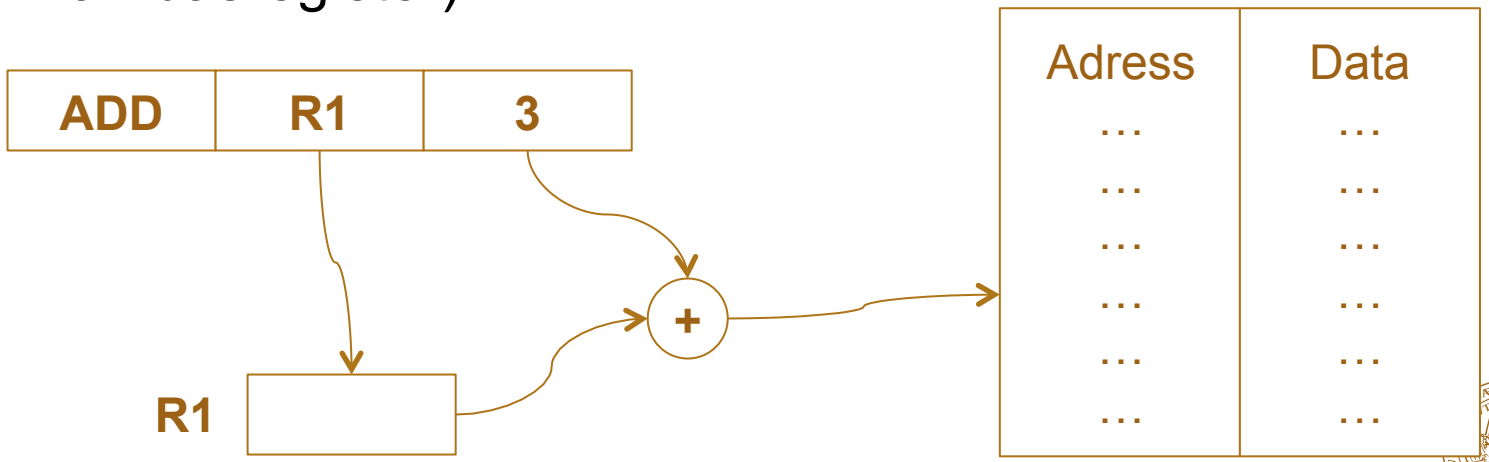
LUNDS UNIVERSITET

Adressering

- Displacement

  `ADD R4,R1.X  R4←R4+[R1+X]`

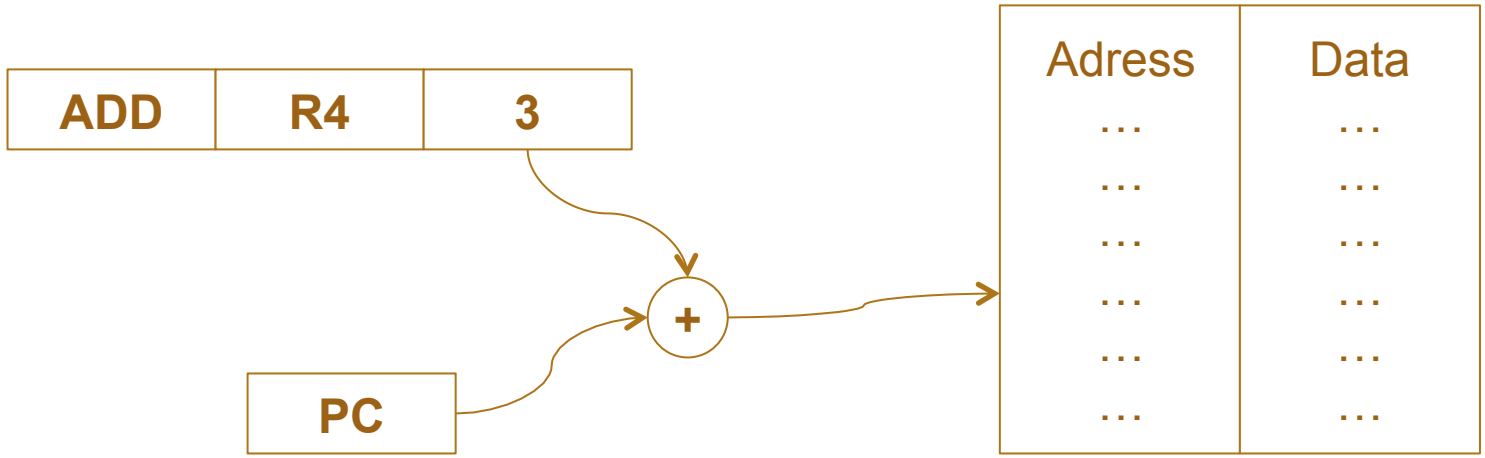
  Ett basregister (vanligen ett general purpose register) ger basen och ett par bitar ger displacement (en offset från basregister)



LUNDS UNIVERSITET

Adressering

- Relativ adressering
  - Använd t ex PC för att skapa adress. Enbart liten bit finns i instruktionen



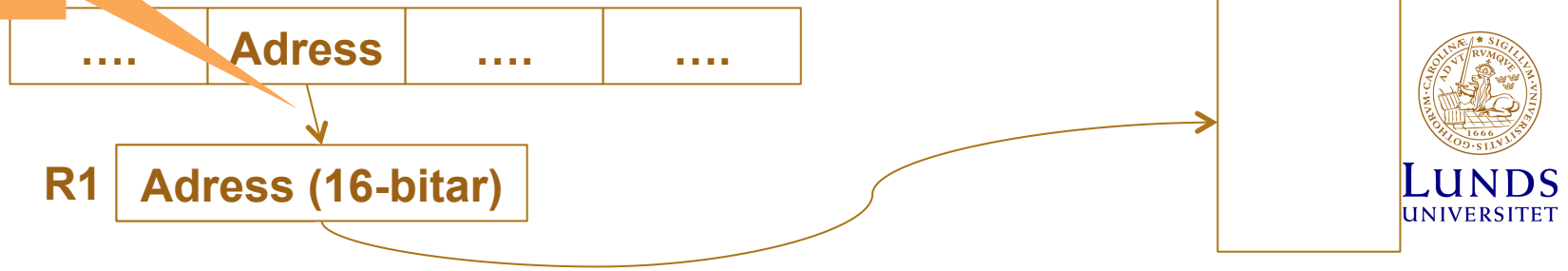
Varför olika sätt att adressera?

- Öka flexibiliteten vid programmering, vilken förenklar programmering
- Antal bitar för att specificera en adress är begränsat.
  - Exempel: Om 8-bitar används:

kan 2^8 adresser pekas ut.

Extra tid (jmf ovan)

- Men, om 8-bitar används att peka ut ett register som innehåller adress kan 2^{16} adresser pekas ut



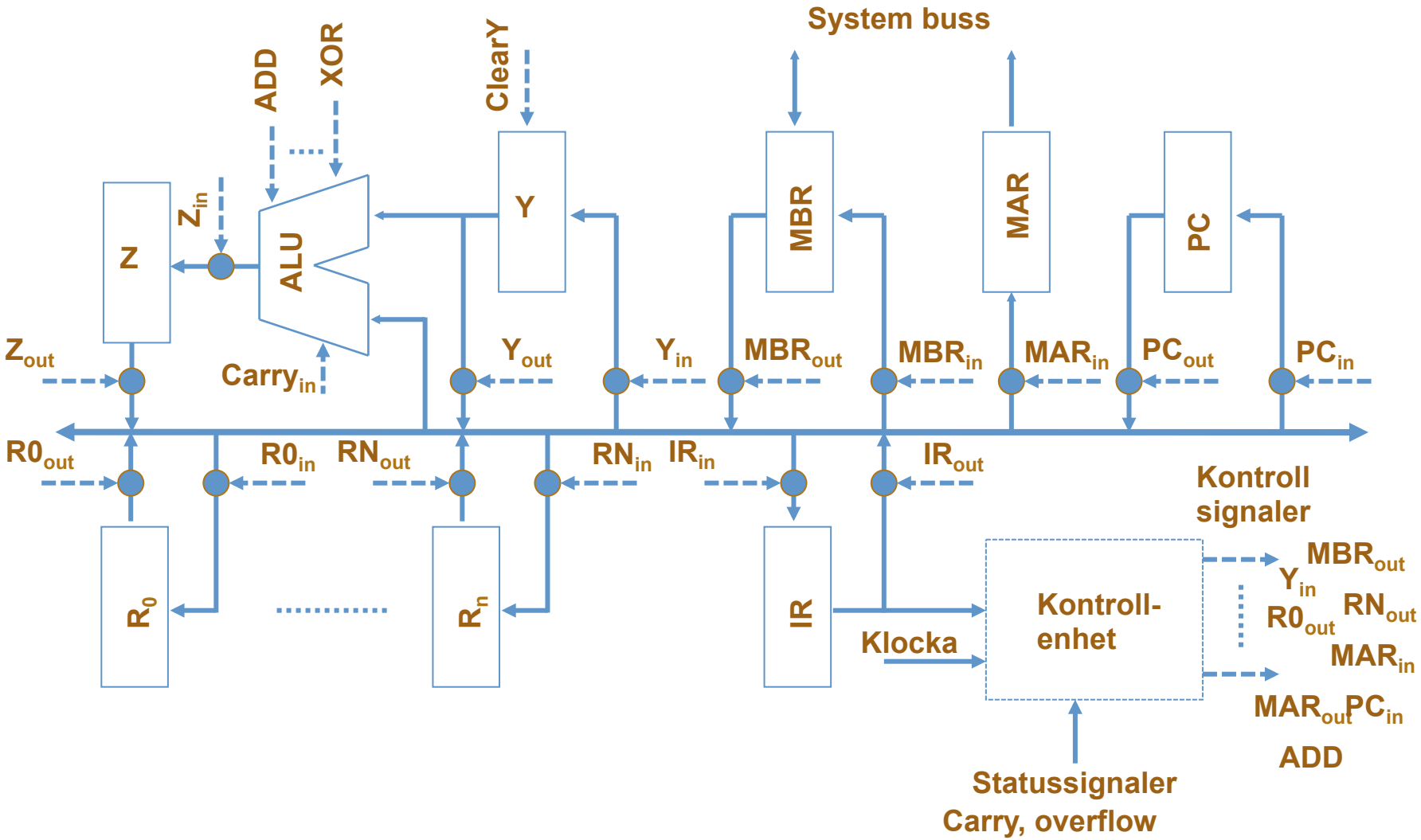
LUNDS UNIVERSITET

Adressering: avvägningar

- Få och enkla adresseringsmöjligheter
  - Stora adressfält i instruktion för att kunna täcka stor adressrymd
  - Begränsar programmering
  - Snabbare adressberäkning
  - Mindre komplex processor
- Många och avancerade adresseringsmöjligheter
  - Tillåter mindre addressfält utan att påverka adressrymd
  - Flexibel programmering
  - Kan ge långsam adressberäkning
  - Ökar komplexitet av processorn

LUNDS UNIVERSITET

Kontrollenhet – komplexitet



Hoppinstruktioner

Exempel 1

```
a=a+b;
c=a-b;
d=a*f;
```

Instruktionerna utförs i ordning

Instruktionen här görs flera gånger

Exempel 2

```
a=0;
while (a<10)
      a=a+1;
if (b==5)
   c=10;
else
   c=0;
d=a*f;
```

Beroende på villkor görs olika saker (olika instruktioner exekveras)

LUNDS UNIVERSITET

Hoppinstruktion

- Nästa instruktion är normalt nästa i programmet.

  ```
  PC=PC+1;
  ```

  Programräknaren (program counter) räknas upp och nästa instruktion hämtas

- För att göra hopp till annan del av kod, ladda programräknaren med nytt värde

  ```
  PC=ditt hopp ska sek
  ```

- Två typer av hopp
  - Ovillkorliga hopp
  - Villkorliga hopp

LUNDS UNIVERSITET

Ovillkorliga hopp

- Exempel:

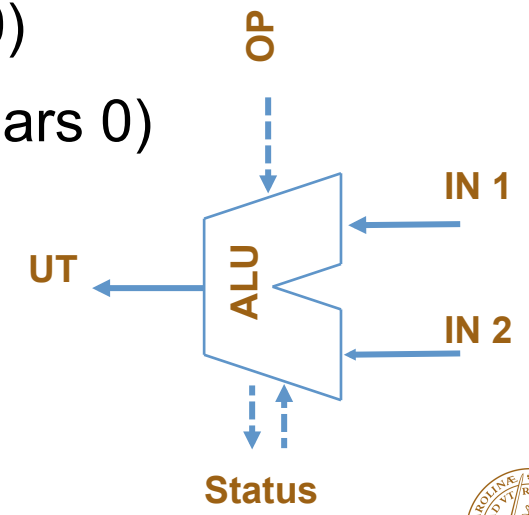
Programräknaren får nytt värde och hämtar nästa instruktion på nytt ställe

Adress	Instruktion	Kommentar
.....	.....	
01011	Instruktion 29	//Instruktion 29,PC=PC+1
01100	BR 10101	// PC = 10101, PC=PC+1
01101	Instruktion 31	//Instruktion 31, PC=PC+1
.....	.....	
10101	Instruktion 89	//Instruktion 89, PC=PC+1
.....	.....	

LUNDS UNIVERSITET

Villkorliga hopp

- Villkor bestäms av flaggor i statusregister
- De vanligaste flaggorna är:
  - N: 1 om resultatet är negativt (annars 0)
  - Z: 1 om resultatet är noll (annars 0)
  - V: 1 om aritmetiskt “overflow” (annars 0)
  - C: 1 om “carry” (annars 0)



LUNDS UNIVERSITET

Villkorliga hopp

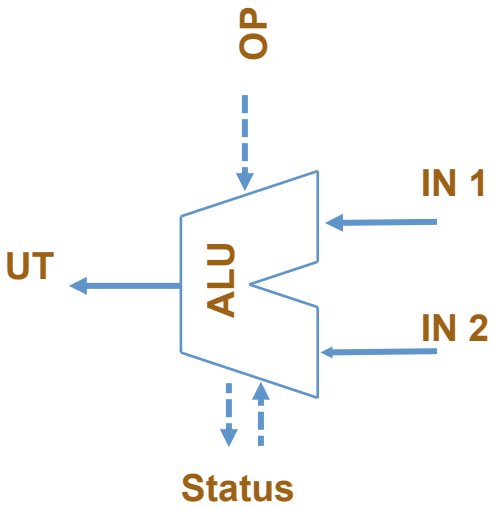
- Exempel:

```
SUB R1, #1        // R1=R1-1
BEZ TARGET        // IF Z=1 THEN PC=TARGET
```

Kan påverka statusflagga

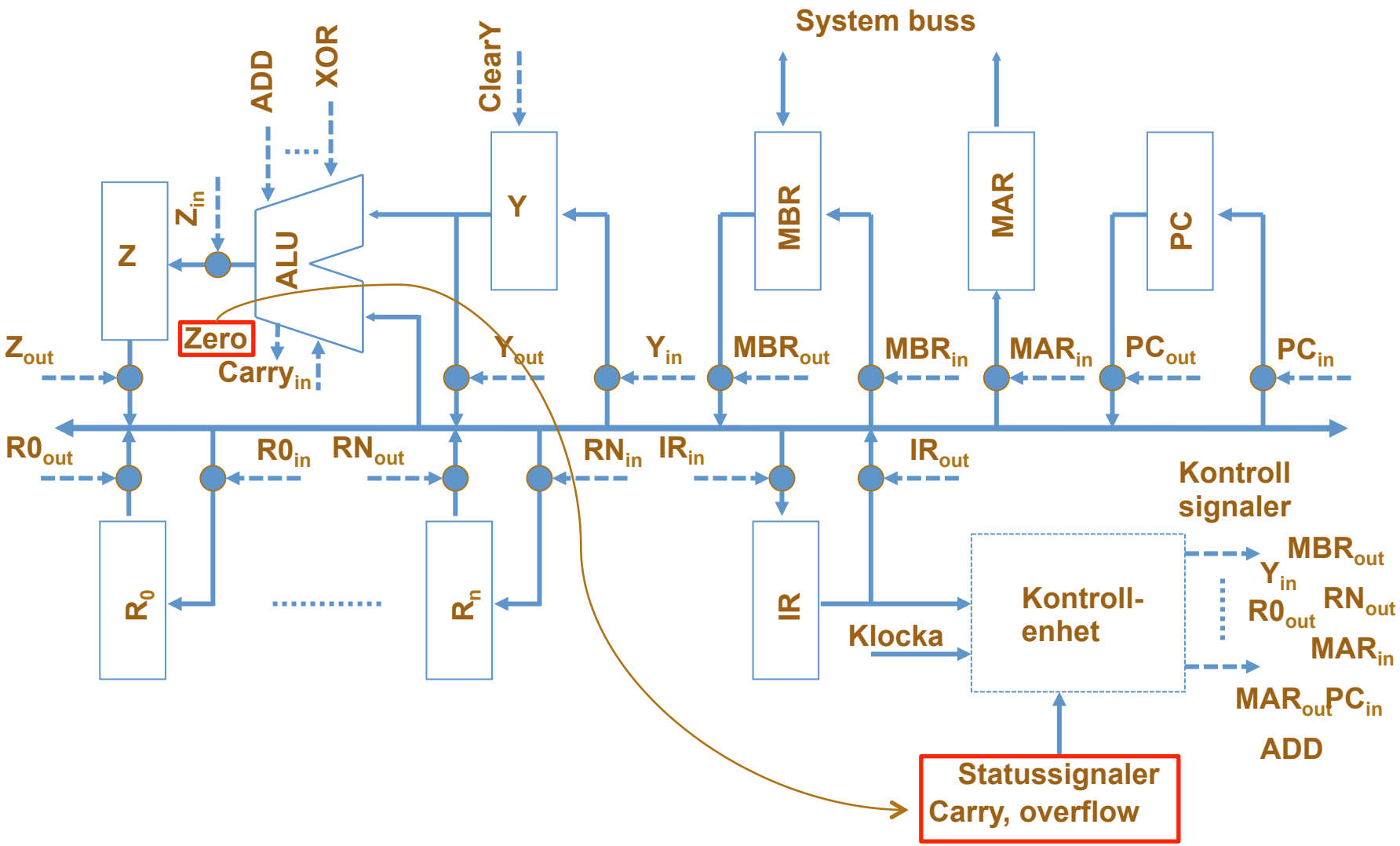
Kollar status-flagga

Statusregister



LUNDS UNIVERSITET

Villkorliga hopp



Subrutiner och funktioner

Program:

```
void main(void){
int a, b, c;
  a=5;
  b=6;
  c=my_add(a,b);
}


int my_add(int x, int y)
{
  return x+y;
}
```

Exekvering:

a=5 //instruktion 1
b=6 //instruktion 2

funktionsanrop
+parameteröverföring

c=my_add(a,b); instruktion 11

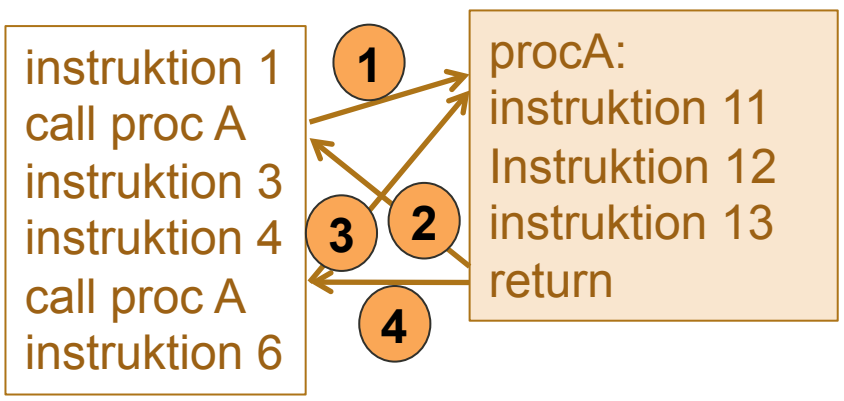
återhopp+parameteröverföring

LUNDS UNIVERSITET

Subrutiner och funktioner

- Problem
  - Hopp till subrutin
  - Återhopp från subrutin
  - Parameteröverföring
  - Nästlade subrutinanrop

	Exekvering:	
	instruktion 1	
1	instruktion 11	//Proc A
	Instruktion 12	//Proc A
	Instruktion 13	//Proc A
2	instruktion 3	
	instruktion 4	
3	instruktion 11	//Proc A
	Instruktion 12	//Proc A
	instruktion 13	//Proc A
4	instruktion 6	

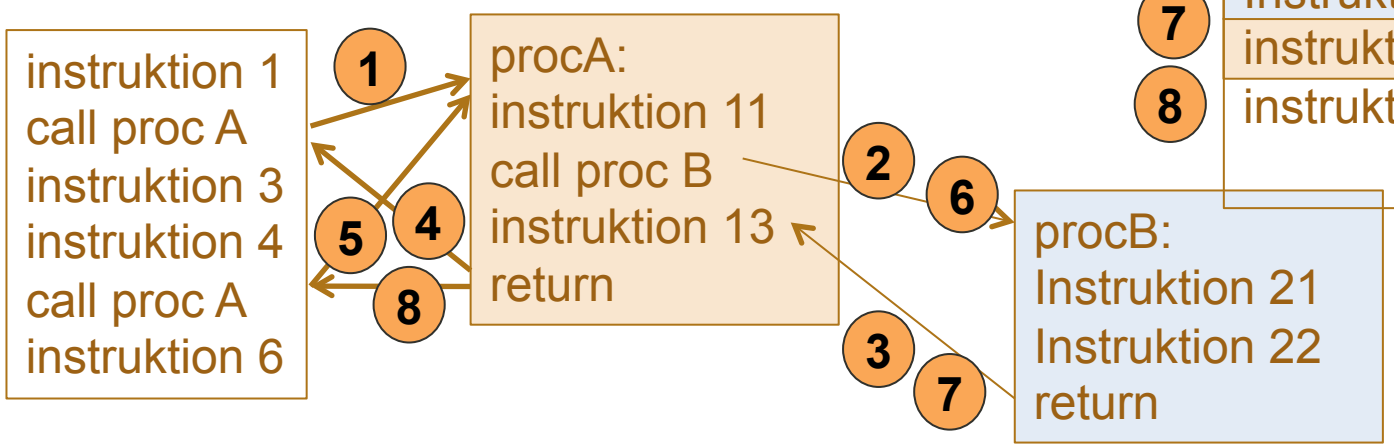


LUNDS UNIVERSITET

Subrutiner och funktioner

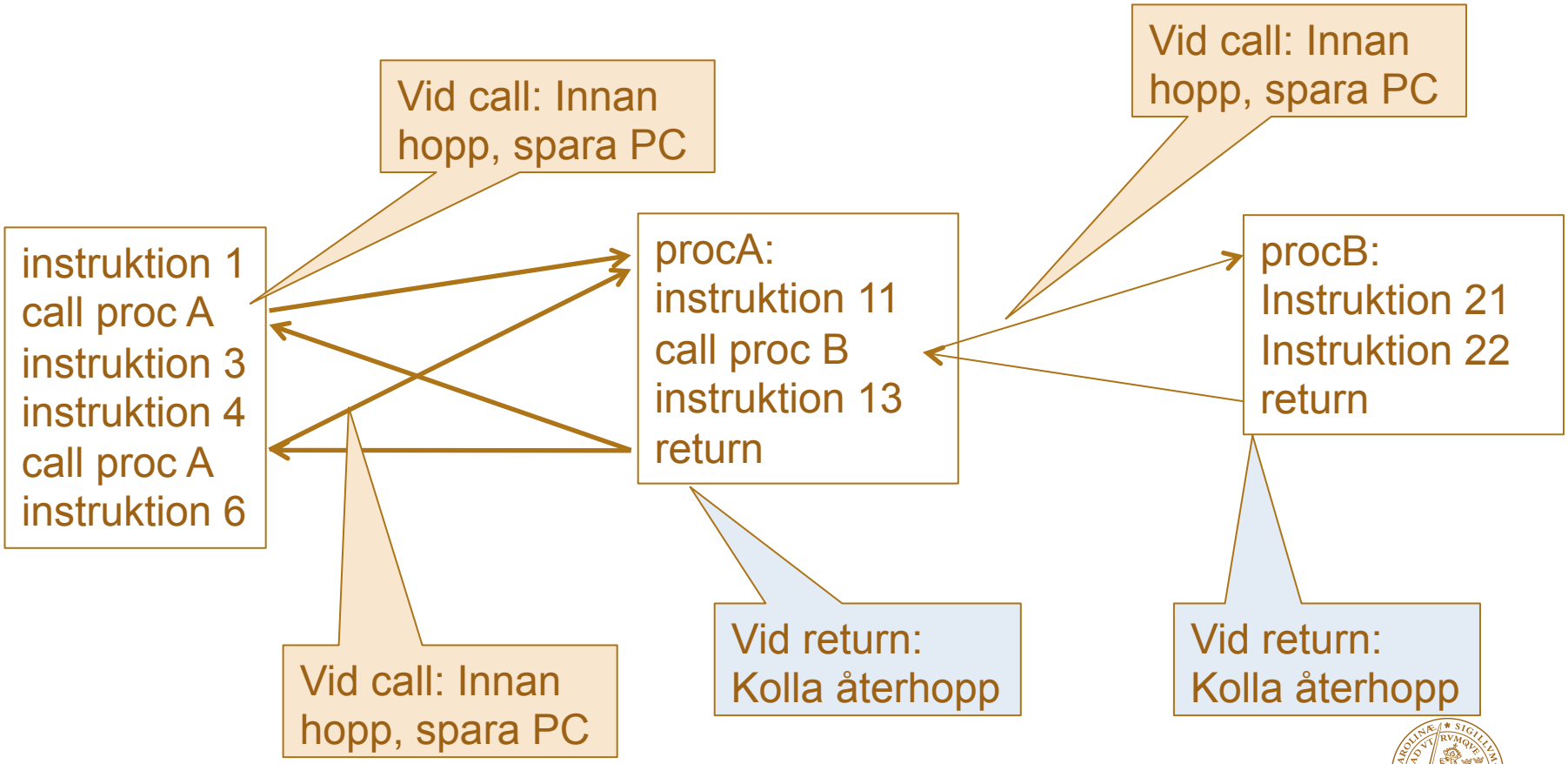
- Problem
  - Hopp till subrutin
  - Återhopp från subrutin
  - Parameteröverföring
  - Nästlade subrutinanrop

	Exekvering:	
	instruktion 1	
1	instruktion 11	//Proc A
2	Instruktion 21	//Proc B
	Instruktion 22	
3	instruktion 13	//Proc A
4	instruktion 3	
	instruktion 4	
5	instruktion 11	//Proc A
6	Instruktion 21	//Proc B
	Instruktion 22	
7	instruktion 13	//Proc A
8	instruktion 6	



LUNDS UNIVERSITET

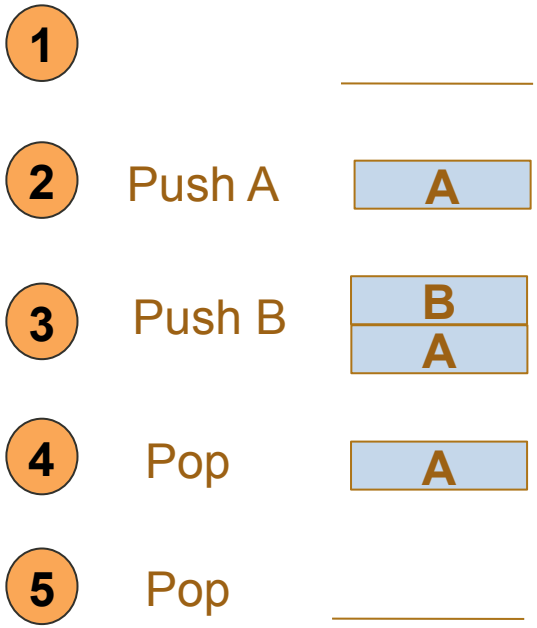
Subrutiner och funktioner

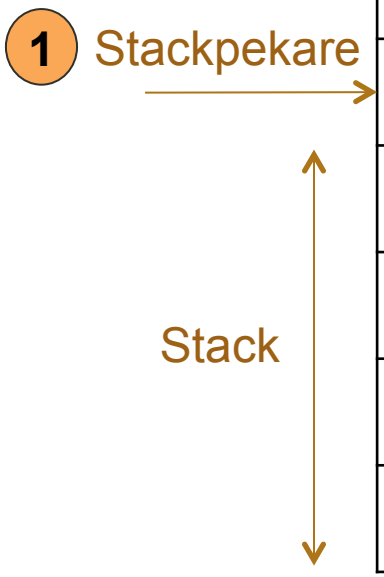


LUNDS UNIVERSITET

Stack

- Två operationer: push och pop





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

Stack

- Två operationer: push och pop

1

2 Push A

3 Push B

4 Pop

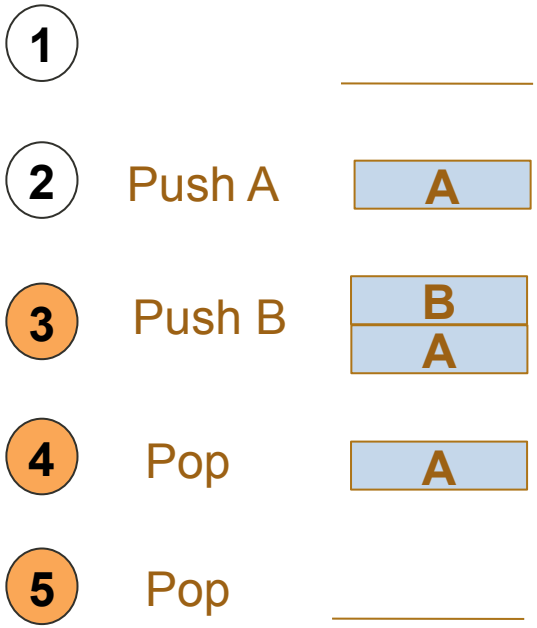
5 Pop

2 Stackpekare

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

Stack

- Två operationer: push och pop

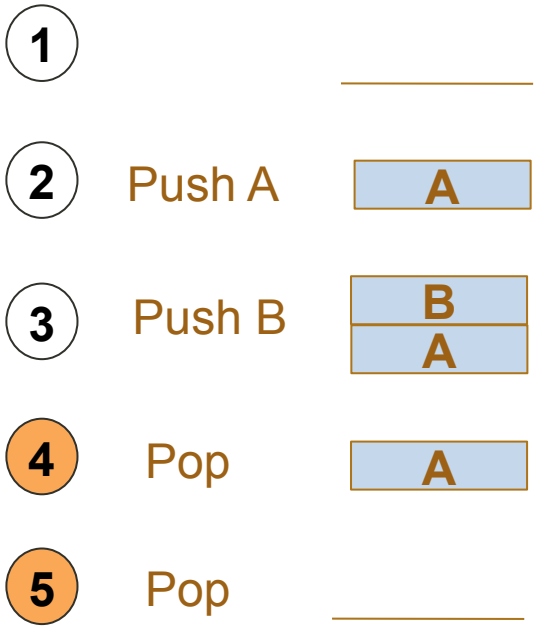


Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	A
5	5	B
6	6	1111 0000
7	7	1010 1010

3 Stackpekare

Stack

- Två operationer: push och pop

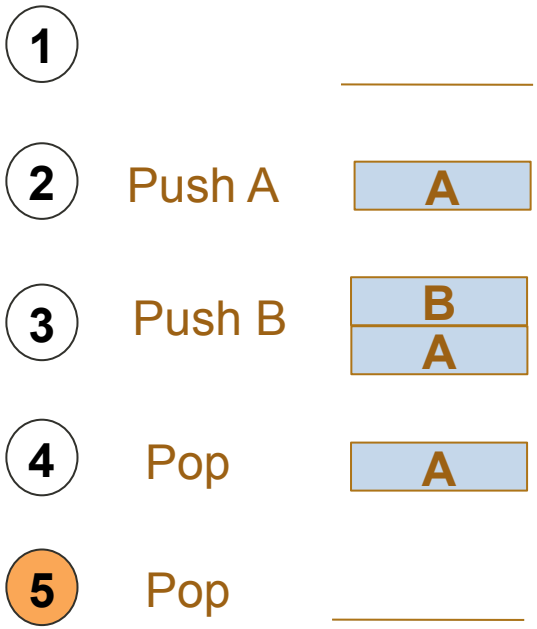


4 Stackpekare

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

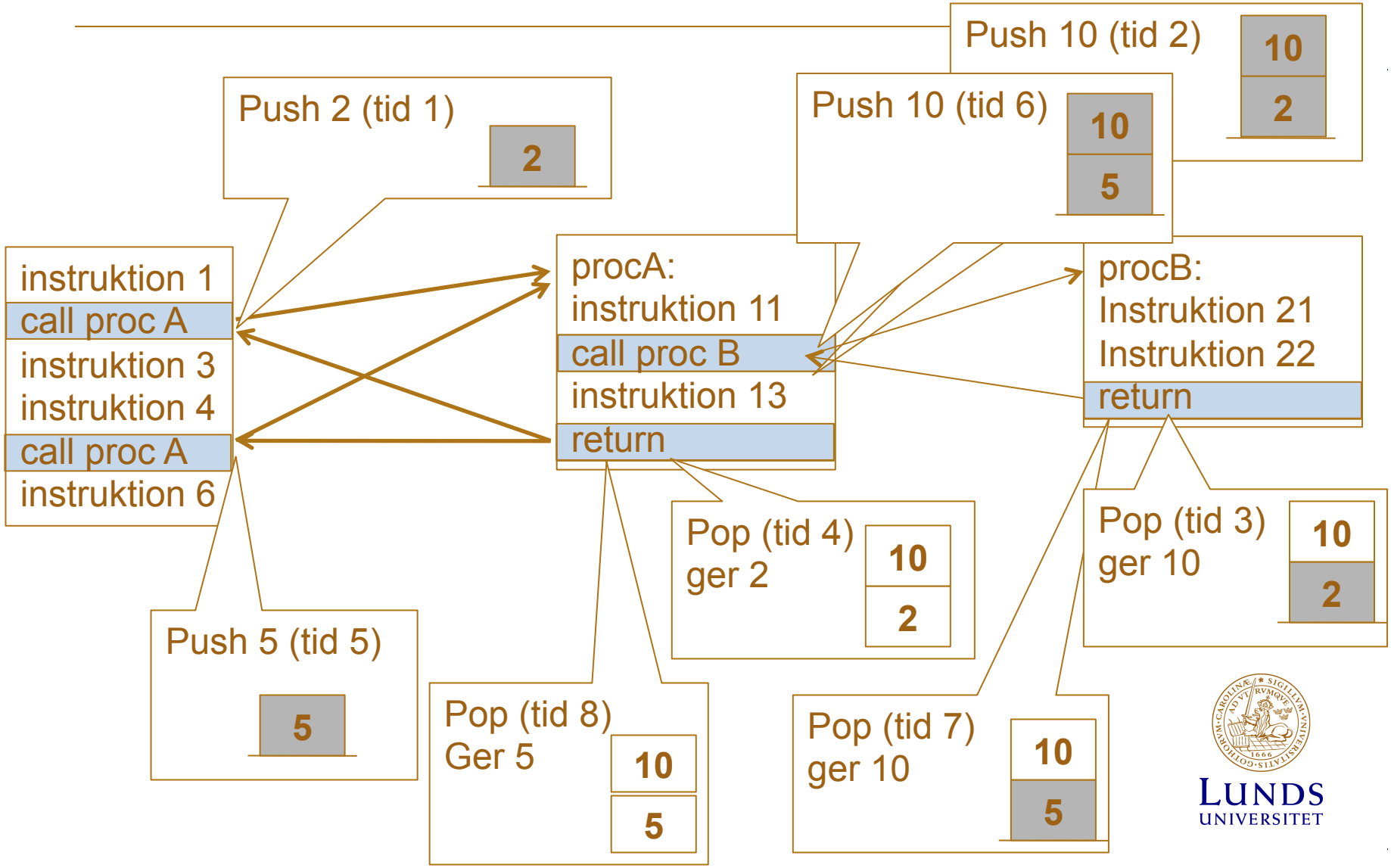
Stack

- Två operationer: push och pop



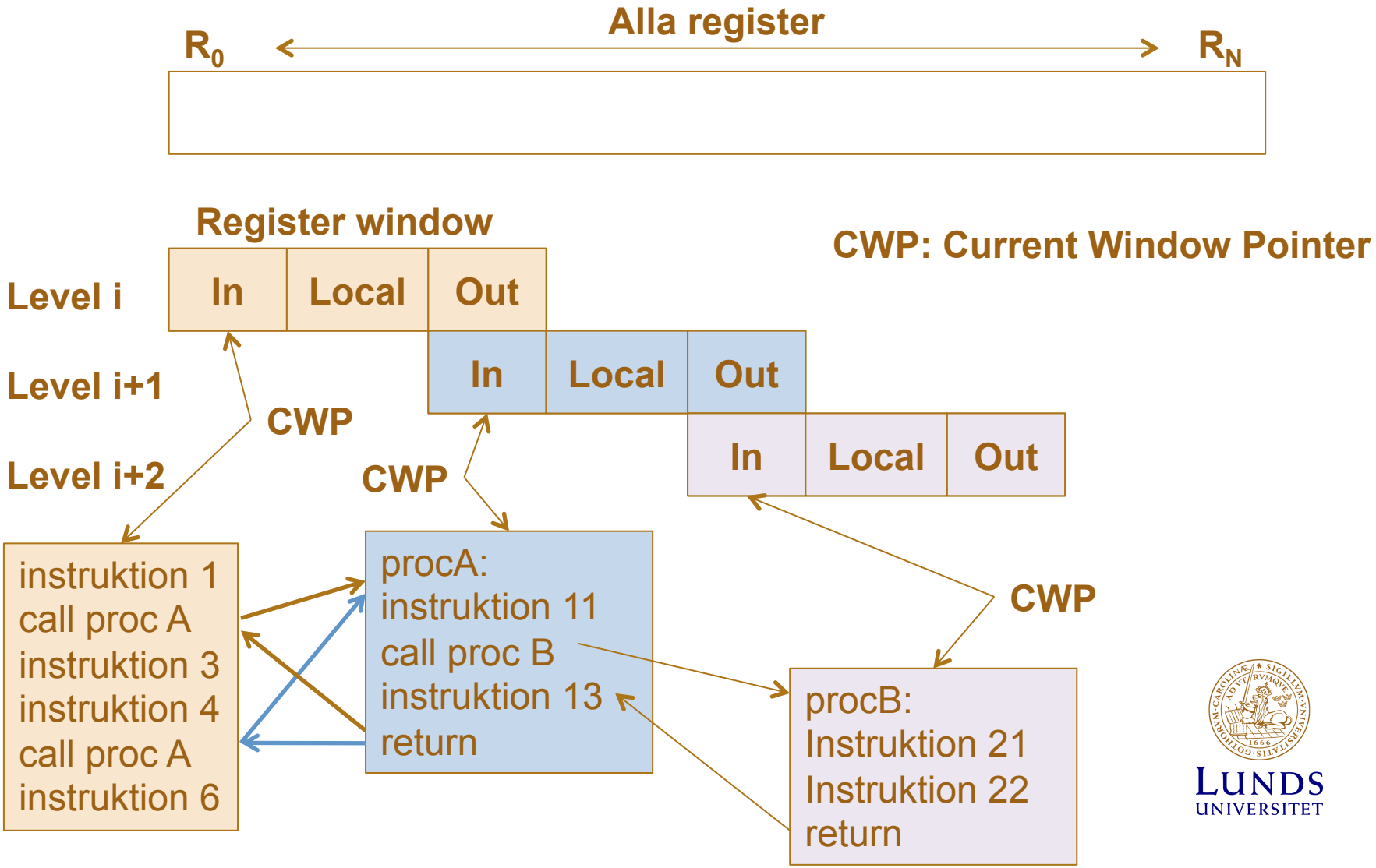
Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	A
5	5	B
6	6	1111 0000
7	7	1010 1010

Stack för subrutiner och funktioner



LUNDS UNIVERSITET

Register för subrutiner och funktioner



LUNDS UNIVERSITET

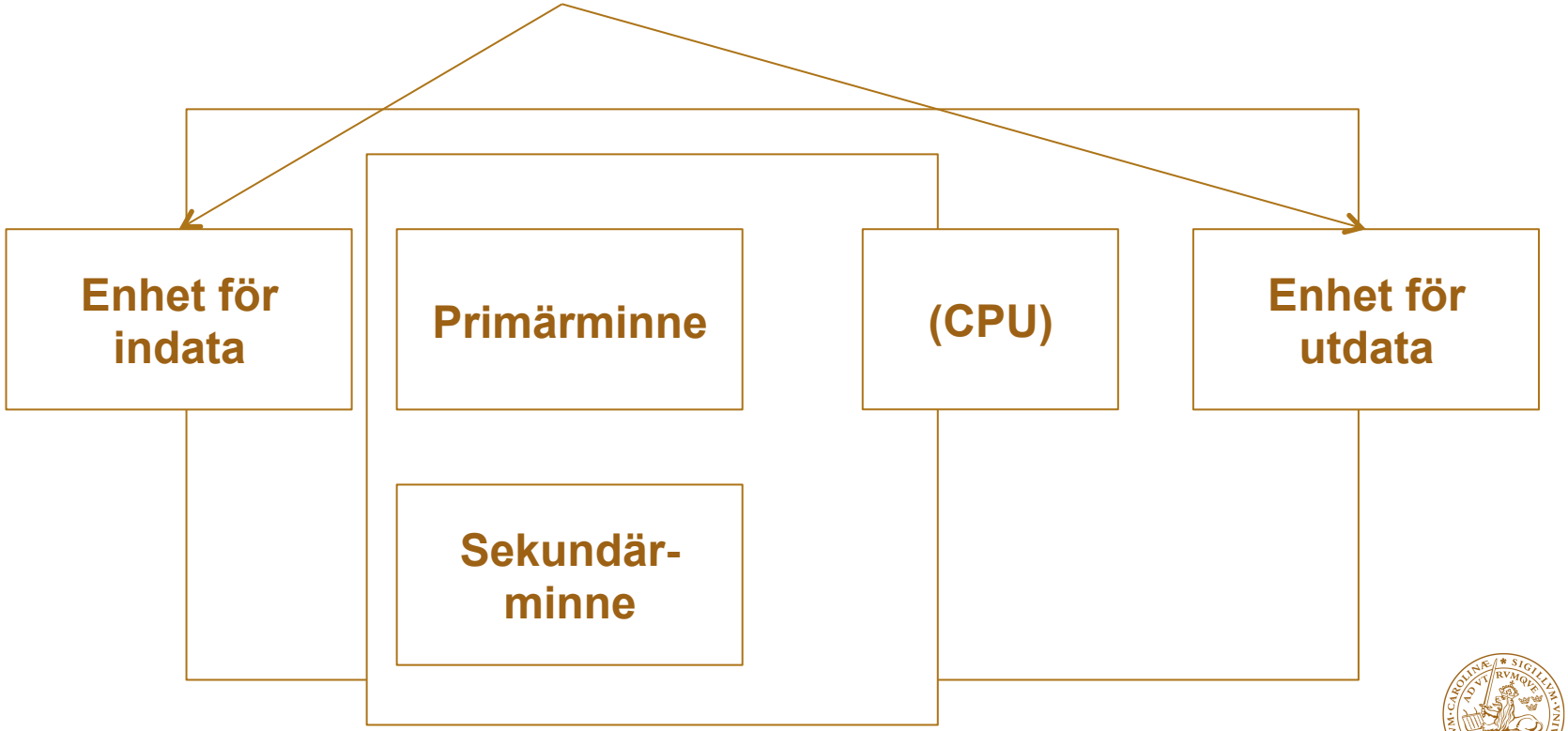
Återhopp och parameteröverföring

- Register
  - Använd ett antal register för att spara återhoppsadress och för parameteröverföring
  - Fördel: snabbt. Nackdel: register behövs
- Stack
  - Reservera en del av minnet och skapa en kö som fungerar enligt Last-In First-Out
  - Fördel: stort. Nackdel: långsamt (jmf register)

LUNDS UNIVERSITET

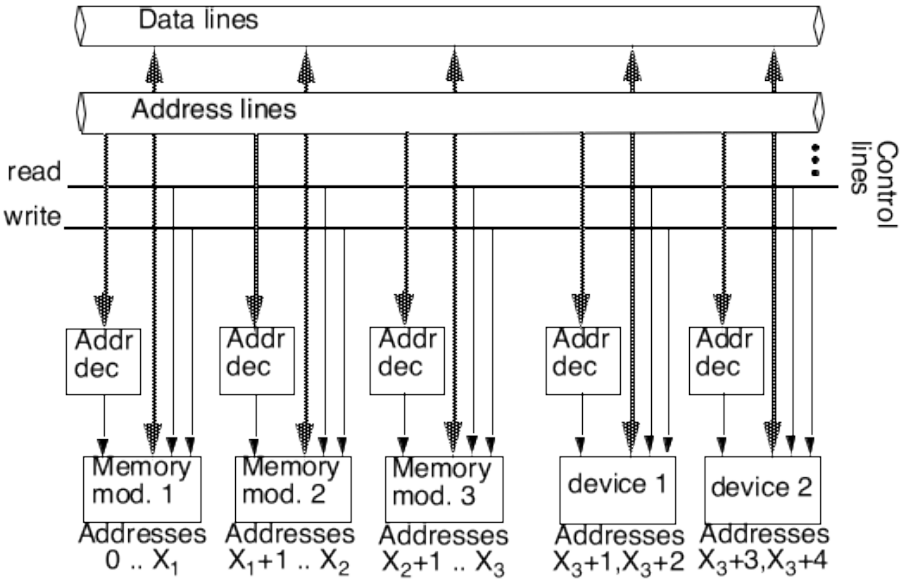
In- och utmatning

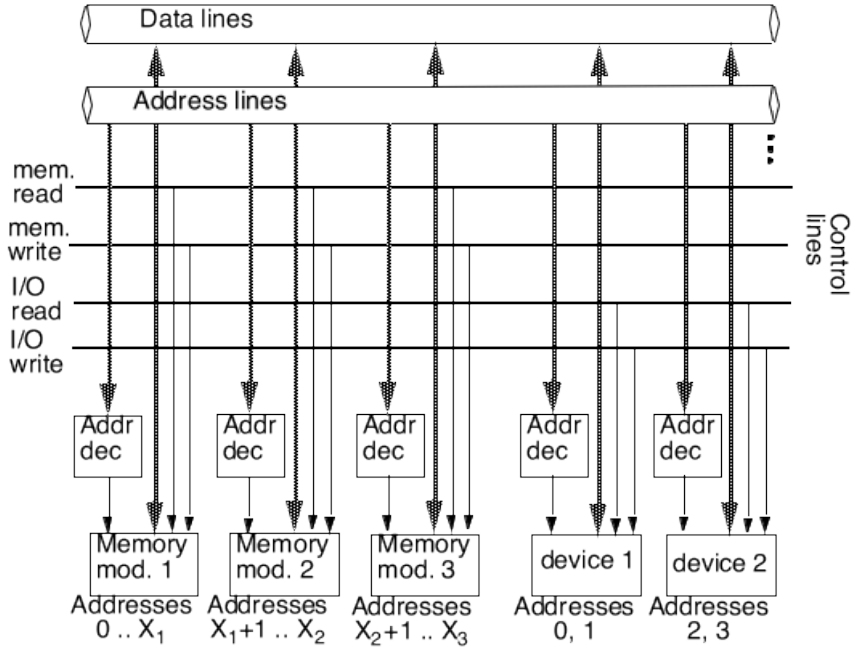
- Läsning och skrivning:



LUNDS UNIVERSITET

Minnesmappad och isolerad I/O



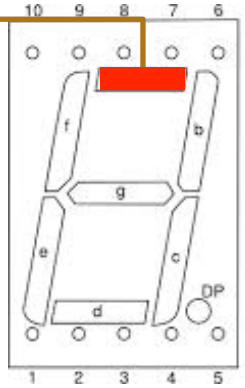


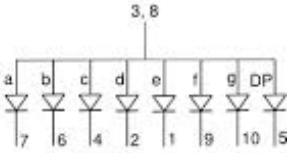
Exempel på minnesmappad I/O

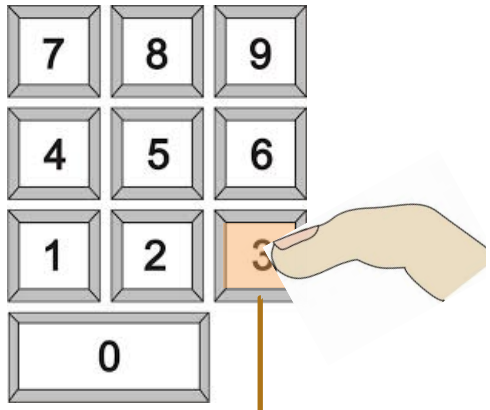
LOAD R1, 7 //Ladda R1 med värde på adress 7

STORE R1, 6 //Lagra värdet i R1 på adressplats 6

Adress	Instruktion/ Data
0	1111 0000
1	1010 0101
2	1100 0011
3	0011 0011
4	1111 1111
5	0000 1111
6	0010 0000
7	0010 0000







LUNDS UNIVERSITET

Maskininstruktioner

- Aritmetiska och logiska (ALU)
- Dataöverföring (adressering)
- Hopp och subrutiner
- Inmatning/utmating (Input/Output (I/O))

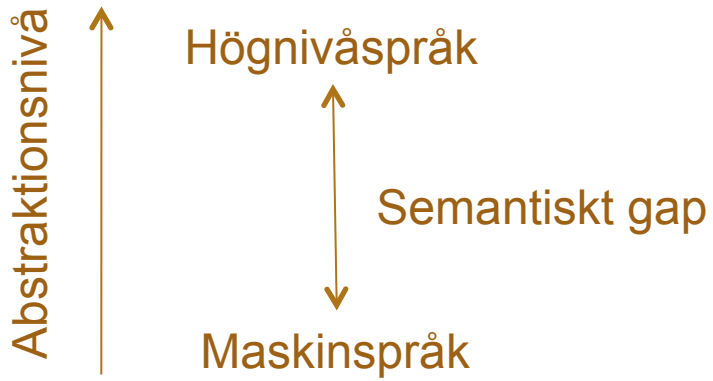
LUNDS UNIVERSITET

Instruktionsformat

- Hur definiera fälten i instruktioner?

Opcode	Operand1	Operand2

- Hur ska högnivåspråk kompileras och exekveras effektivt?



LUNDS UNIVERSITET

Antal adresser i instruktion

- För att beräkna: X = (A+B)*C
- 4-adress instruktioner:
  - Op A1, A2, A3, A4 //A1 <- [A2] op [A3], nästa instruktion = A4

```
I1: ADD X, A, B, I2
I2: MUL X, X, C, I3
I3:
```

- 3-adress instruktioner:
  - Op A1, A2, A3 //A1 <- [A2] op [A3]

```
I1: ADD X, A, B
I2: MUL X, X, C
I3:
```

LUNDS UNIVERSITET

Antal adresser i instruktion

- För att beräkna: X = (A+B)*C
- 2-adress instruktioner:
  - Op A1, A2 //A1 <- [A1] op [A2]

```
MOVE A, X
ADD X, B
MUL X, C
```

- 1-adress instruktioner:
  - Op A1 //Acc <- [Acc] op [A1]

```
LOAD A
ADD B
MUL C
STORE X
```

LUNDS UNIVERSITET

Antal adresser i instruktion

- Vanligast med 2 och 3 adresser i instruktioner
- 4 adress instruktioner är opraktiskt.
  - Nästa instruktion antas vara nästa instruktion. PC räknas upp “av sig själv”. Måste ha hopp-instruktioner
- 1 adress instruktion är ganska begränsande
- Exempel, antag 32 register (5 bitar behövs):
  - 3-adress format: 15 bitar
  - 2-adress format: 10 bitar
  - 1-adress format: 5 bitar

- Få adresser, få instruktioner förenklar processorn men gör den mer primitiv

LUNDS UNIVERSITET

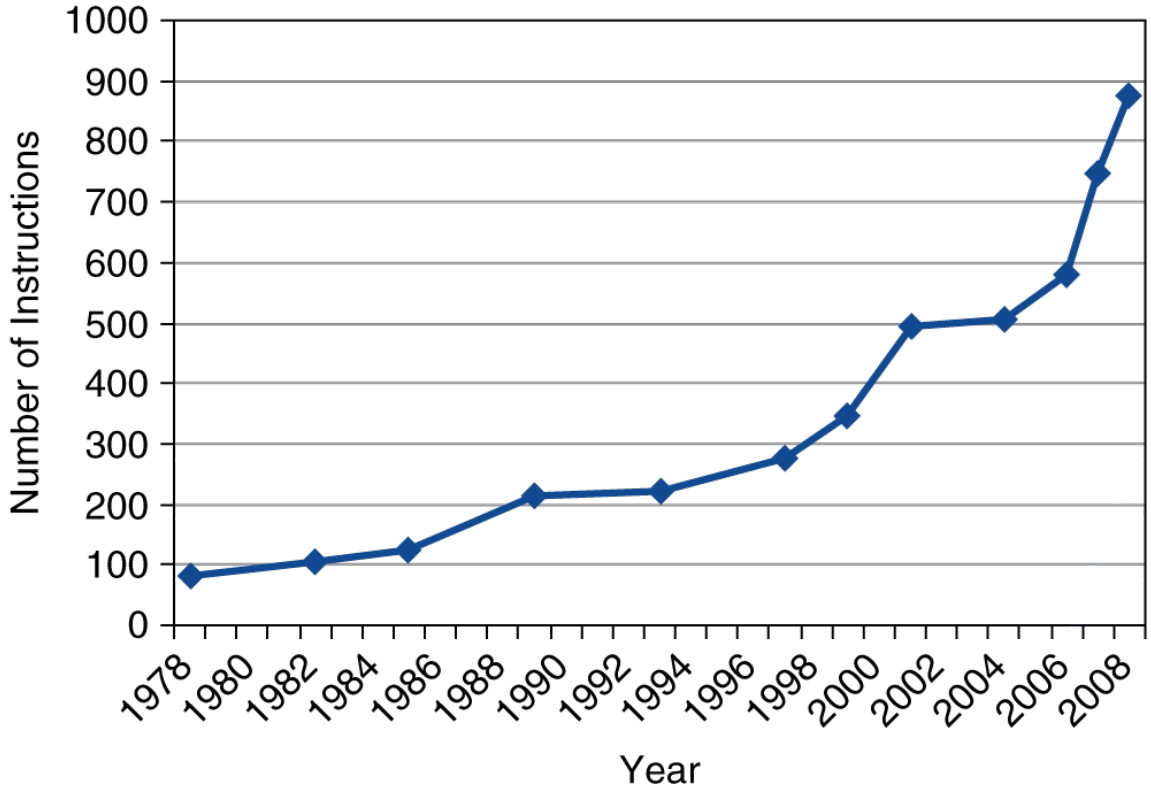
Intel x86 ISA

- Utveckling med backward compatibility
  - 8080 (1974): 8-bit microprocessor
    - » Accumulator, plus 3 index-register pairs
  - 8086 (1978): 16-bit extension to 8080
    - » Complex instruction set (CISC)
  - 8087 (1980): floating-point coprocessor
    - » Adds FP instructions and register stack
  - 80286 (1982): 24-bit addresses, MMU
    - » Segmented memory mapping and protection
  - 80386 (1985): 32-bit extension (now IA-32)
    - » Additional addressing modes and operations
    - » Paged memory mapping as well as segments

LUNDS UNIVERSITET

Intel x86 ISA

- Bakåtkompabilitet (Backward compatibility)



LUNDS UNIVERSITET

Semantiskt glapp

Två alternativ:

- CISC (Complex Instruction Set Computers):
  - utveckla komplex arkitektur med många instruktioner och många adresseringsmöjligheter; ta med instruktioner som liknas högnivåinstruktioner.
- RISC (Reduced Instruction Set Computers):
  - förenkla instruktionsuppsättningen och anpassa den till de verkliga kraven ett applikationsprogram har

LUNDS UNIVERSITET

Utvärdering av applikationsprogram

- Många studier har gjorts för att ta fram karaktärsdrag av maskinkod som genererats från högnivåspråk
- Intressanta aspekter:
  - Hur ofta används varje instruktion?
  - Hur används operander? Och hur ofta?
  - Hur mycket används hopp, loopar, anrop till procedurer/ funktioner/subrutiner?

LUNDS UNIVERSITET

Utvärdering av applikationsprogram

Typ	Instruktioner i högnivåspråk (%)		Instruktioner i maskinspråk (%)		Minnesreferenser (%)	
	Pascal	C	Pascal	C	Pascal	C
Tilldelning	45	38	13	13	14	15
Loop	5	3	42	32	33	26
Call	15	12	31	33	44	45
Villkor (if)	29	43	11	21	7	13
Andra	6	1	3	1	2	1

Många tilldelningar i högnivåspråk men dessa ger få instruktioner i maskinspråk och relativt få minnesreferenser

Relativt få loopar men looparna resulterar i många maskininstruktioner och många minnesreferenser

Utvärdering av applikationsprogram

Typ	Instruktioner i högnivåspråk (%)		Instruktioner i maskinspråk (%)		Minnesreferenser (%)	
	Pascal	C	Pascal	C	Pascal	C
Tilldelning	45	38	13	13	14	15
Loop	5	3	42	32	33	26
Call	15	12	31	33	44	45
Villkor (if)	29	43	11	21	7	13
Andra	6	1	3	1	2	1

Få CALL (anrop till funktioner/ procedurer/subrutiner) men de ger många instruktioner i maskinspråk och relativt många minnesreferenser

Relativt många villkor med de ger relativt få maskininstruktioner och relativt få minnesreferenser

LUNDS UNIVERSITET

Utvärdering av applikationsprogram

- Fördelning av maskininstruktioner (frekvens av användande):
  - Flytta data (move): ~33%
  - Villkorliga hopp: ~20%
  - ALU operationer: ~16%
- Adressering
  - Komplexa adressering: ~18% (memory indirect, indexed+indirect, displacement+indexed, stack)
  - Enkel adressering: vanligast (adress kan beräknas i en klockcykel, register, register indirekt, displacement)

LUNDS UNIVERSITET

Utvärdering av applikationsprogram

- Typer av operander
  - 74-80% av operander är skalärer (heltal, flyttal, tecken, etc), vilka kan lagras i register (icke-skalär, t ex array)
  - Resterande (20-26%) är arrayer; 90% är globala variabler
  - 80% av skalärer är lokala variabler
- Slutsats: Lagring i register

LUNDS UNIVERSITET

Utvärdering av applikationsprogram

- Trots att endast 10-15% av högnivå instruktionerna var funktion/procedure/ subrutin anrop, så svarar dessa CALL och RETURN instruktioner för:
  - 31-43% av maskininstruktionerna
  - 44-45% av minnesreferenserna
- För procedurer:
  - Endast 1.25% har fler än 6 parametrar
  - Endast 6.7% har fler än 6 lokala variabler
  - Antal nästlingar är ofta få (korta) och mycket sällan längre än 6

```
int x, y; float z;
void proc3(int a, int b, int c) {
    ---------------
}
void proc2(int k) {
    int j, q;
    ---------------
    proc3(j, 5, q);
    }
void proc1() {
    int i;
    ---------------
    proc2(i);
    }
main () {
    proc1();
    }
```

LUNDS UNIVERSITET

Karaktärsdrag hos RISC processorer

- Enkla och få instruktioner
- Enkla och få adresseringsmöjligheter
- Fixt (fast) instruktionsformat
- Stort antal register
- Load-and-store architecture

LUNDS UNIVERSITET

Karaktärsdrag hos RISC processorer

- Ett litet exempel:
  - Antag ett program med 80% enkla instruktioner och 20% komplexa instruktioner som exekveras på en CISC och en RISC
  - CISC: enkla instruktioner tar 4 klockcykler medan komplexa instruktioner tar 8 klockcykler. Antag klockcykel tid på 100 ns (10^{-7} s)
  - RISC: enkla instruktioner 1 klockcykel medan komplexa instruktioner implementeras som en sekvens av enkla. Antag att det tar i genomsnitt 14 instruktioner för en komplex instruktion. Antag klockcykel tid på 75 ns ($0.75*10^{-7}$).
- Hur lång tid tar det att exekvera 1 000 000 instruktioner?
  - CISC: $(10^6*0.80*4 + 10^6*0.20*8)*10^{-7}=0.48s$
  - RISC: $(10^6*0.80*1 + 10^6*0.20*14)*0.75*10^{-7}=0.27s$

LUNDS UNIVERSITET

Karaktärsdrag hos RISC processorer

- Ett litet exempel (fortsättning):
  - Komplexa instruktioner tar längre tid på en RISC, men det är relativt fler enkla instruktioner
  - På grund av sin (relativa) enkelhet, kan RISC operera med kortare cykeltid (högre klockfrekvens). För CISC bromsas enkla instruktioner av mer komplex data path och mer komplex kontrollenhet (det som inte är kontrollenhet kallas för data path)

LUNDS UNIVERSITET

Karaktärsdrag hos RISC processorer

- Instruktioner använder få adresseringsmöjligheter:
  - Typiskt: register, direkt, register indirekt, displacement
- Instruktioner har fixt (fast) längd och uniformt format
  - Detta förenklar och snabbar upp laddning och avkodning. Processorn behöver inte vänta på att hela instruktionen är laddad innan avkodning kan påbörjas
  - Uniformt format, dvs opcode och adressfält är placerat på samma position för olika instruktioner, gör att avkodning förenklas.

LUNDS UNIVERSITET

Karaktärsdrag hos RISC processorer

- Stort antal register
  - Register kan användas för att lagra variabler och för "mellanlagring" resultat. Det minskar behov av att använda upprepade load och store med primärminnet.
  - Alla lokala variabler från CALL/RETURN strukturer (anrop till funktioner, procedurer, subrutiner) kan lagras i register.
- Vad händer vid anrop till funktioner och procedurer?
  - Register innehåll måste sparas, parametrar ska föras över, återhopps adresser ska sparas. Relativt mycket minnes access, vilket tar mycket tid.
  - Med många register, kan nya register allokeras vid anrop

LUNDS UNIVERSITET

RISC eller CISC?

- Ett entydigt svar är svårt att ge
- Flera prestanda jämförelser (benchmarking) visar att RISC exekverar snabbare än CISC
- Men, det är svårt att identifiera vilket karaktärsdrag som är avgörande. Det är svårt att jämföra; t ex olika halvledartekniker, kompilatorer
- Ett argument för CISC: Den enklare RISC konstruktionen gör att program behöver mer minne (fler instruktioner) jämfört med samma program för en CISC processor
- Processorer av idag använder sig ofta av lite RISC och lite CISC

LUNDS UNIVERSITET

Några exempel

- CISC
  - VAX 11/780

    Nr. of instructions: 303
    Instruction size: 2 – 57 bytes
    Instruction format: not fixed
    Addressing modes: 22
    Number of general purpose registers: 16

  - Pentium

    Nr. of instructions: 235
    Instruction size: 1 – 11 bytes
    Instruction format: not fixed
    Addressing modes: 11
    Number of general purpose registers: 8 (32-bitar mode), 16 (64-bitar mode)

Några exempel

- RISC
  - Sun SPARC

    Nr. of instructions: 52
    Instruction size: 4 bytes
    Instruction format: fixed
    Addressing modes: 2
    Number of general purpose registers: up to 520

  - PowerPC

    Nr. of instructions: 206
    Instruction size: 4 bytes
    Instruction format: not fixed (but small differences)
    Addressing modes: 2
    Number of general purpose registers: 32

LUNDS UNIVERSITET

Några exempel

- ARM (Advanced RISC Machine och Acorn RISC Machine)

  Antal instruktioner (standard set): 122
  Instruktions storlek: 4 (standard), 2 (Thumb) byte
  Instruktions format: fixerat (olika mellan standard och Thumb)
  Adresserings modes: 3
  Antal general purpose register: 27 (16 kan användas samtidigt)
  Dagens ARM processorer kan exekvera både standard set (32 bitars instruktioner och 16 bitars Thumb instruktions set. Thumb består av en delmängd av 32-bitars setet, kodat som 16-bitars instruktioner).

LUNDS UNIVERSITET

LUNDS
UNIVERSITET