Adresācija un komandu kopu klasifikācija

CPU veiktspēja

CPU laiks	=	Sekundes / Programu	=	Komandas / Programā	x	Taktis / Komandai	x	Sekundes / Taktij

- 500 MHz Pentium III procesoram lai izpildītu lietotni ar 200K komandām ir nepieciešamas 2 ms.
- 300 MHz UltraSparc procesoram tai pašai lietotnei (kura dotajā komandu kopā satur 230K komandas) ir nepieciešamas 1.8 ms.
- Kāds ir CPI katram procesoram dotajai programmai?
  - CPI = Taktis / Komandu skaitu = CPU laiks X Takts frekvence / Komandu skaits
  - $CPI_{Pentium} = 2*10^{-3}$ X $500*10^{6}$ / $2*10^{5}$ = 5.00
  - $CPI_{SPARC} = 1.8*10^{-3}$ X $300*10^{6}$ / $2.3*10^{5}$ = 2.35
- Kurš ir ātrāks un par cik?
  - UltraSparc ir 2/1.8 = 1.11reizes ātrāks, jeb par 11% ātrāks.

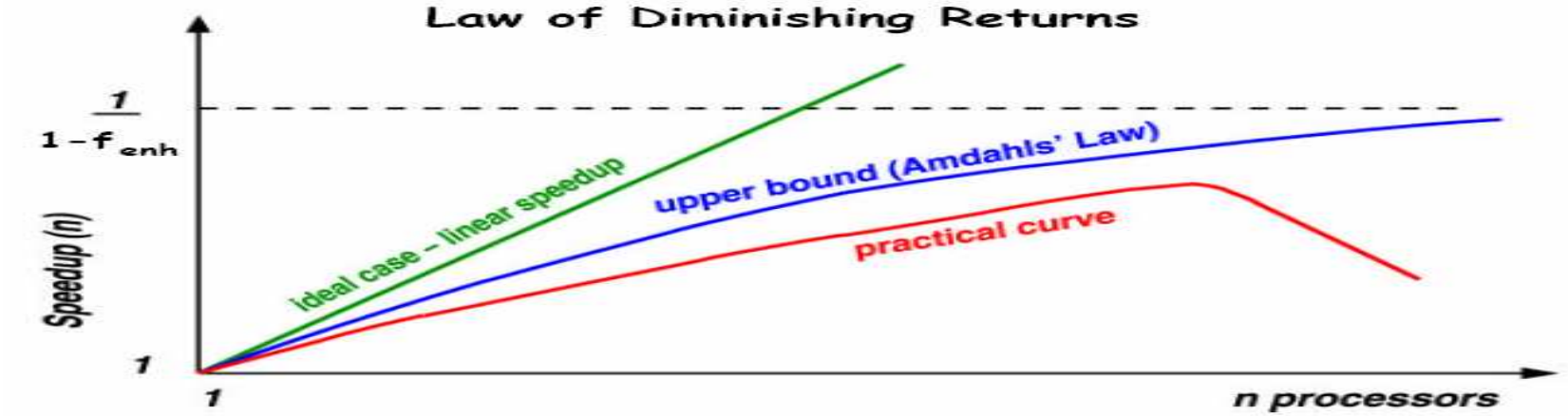
Kā uzlabo veiktspēju

- Ātrākas tehnoloģijas bet:
  - Izmaksas aug
  - Uzticamība krītas
  - 3.10^8 m/sek
- Lielākas matricas (SOC - System On a Chip)
  - Mazāk vadu bet zemāks iznākums IC dēļ kļūdām
- Paralēlā skaitļošana nCPU
  - Vai var sagaidīt S = n ?
- Konveijerapstrāde

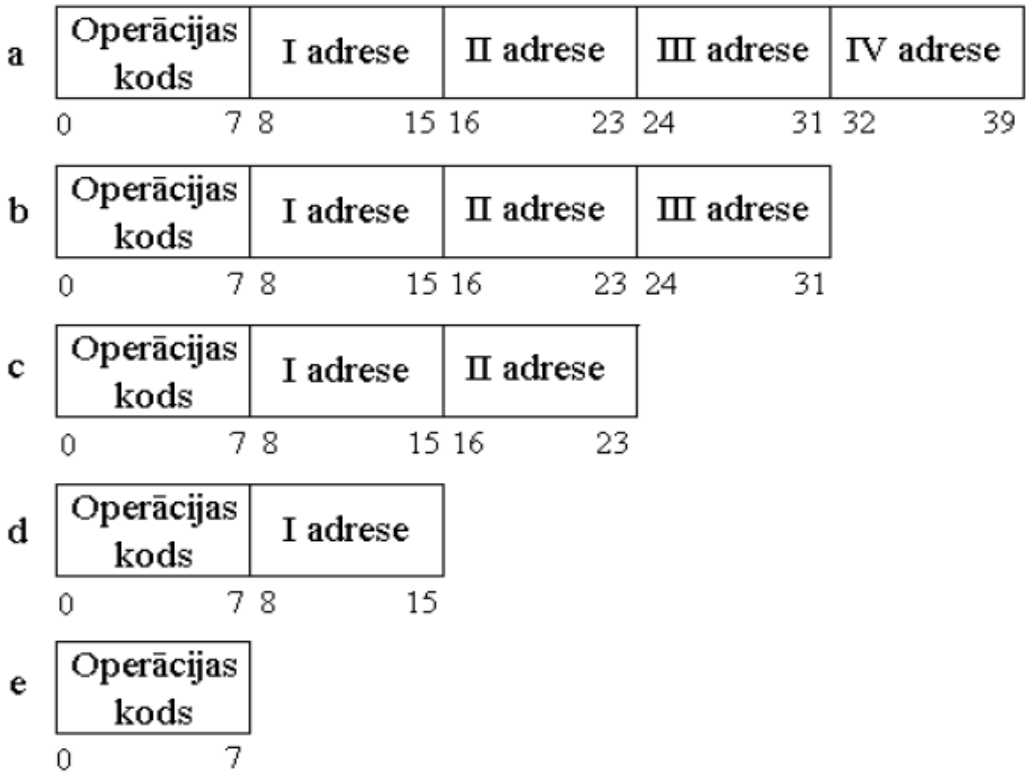
Amdala likums

- Amdala likums apgalvo ka ja ir kāda programmas daļa kuru var optimizēt tad kopējais ieguvums ir izsakāms kā:

  Kopējais uzlabojums=1/((1-P)+(P/S))
- Piemērs:
  - ir 10% programmas koda kuru var izmest (S=inf.) tad labākais rezultāts būs: 1/(1-0.1)=1.11111...reizes
  - Ir 90% programmas koda kuru var uzlabot par 20% un tad rezultāts būs: 1/ ((1-0.9)+(0.9/1.2))=1.1746...reizes



Komandu formāti un adresācijas veidi



4.1. zīm. Adresācija komandās.

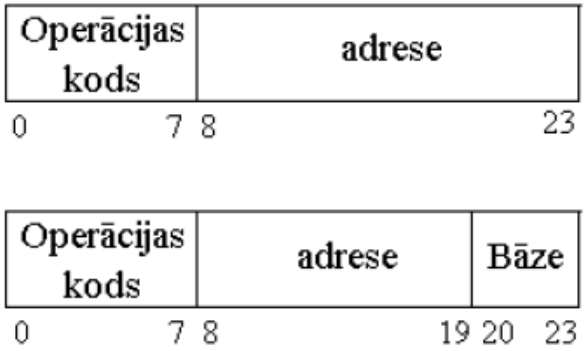
a -četradresu; b -trīsadresu; c -divadresu; d -vienadresu; e -bezadresu.

Komandas adresācijas iespējas

- Pieņemot komandas garumu 4 baiti no kuriem operācijas kodam atvēlot 8 bitus iegūstam šādu tiešās adresācijas apjomu:
- 1. trīsadrešu sistēmā 8 bitus, kas ļauj adresēt 2^8=256 atmiņas šūnas;
- 2. divadrešu sistēmā 12 bitus, kas ļauj adresēt 2^12=4096 atmiņas šūnas;
- 3. vienadreses sistēmā 24 bitus, kas ļauj adresēt 2^24=16 777 216 atmiņas šūnas.

Komandas adresācijas iespējas

- Bāzes adreses izmantošana un ar to saistītā atmiņas lappušu organizācija ļauj izmantot vienu reģistru par bāzes adreses vietu un komandā tieši norādīt tikai nobīdi attiecībā pret šo bāzi.
- Šim nolūkam vienadresu komandas adrešu daļu sadala adreses un bāzes laukā
- Rezultātā adrese veidojas, komandas adrešu daļu saskaitot ar bāzes reģistra saturu
- Bāzes reģistra saturs parasti norāda vecākos adreses bitus jeb lappusi, bet komandas adrešu daļa - jaunākos adreses bitus jeb adresi lappusē.
- Piemēram aplūkotā komanda nodrošina 2^16 = 65536 atmiņas šūnu adresāciju, bet bāzes adresācijā - 16 bāzes reģistru un 2^12=4096 atmiņas šūnu adresāciju.
- Tā kā katrs bāzes reģistrs ir tikpat garš kā komanda un satur 24 bitus, tad kopējais tieši adresējamais atmiņas apjoms sastāda 2^12*2^24 = 2^36 jeb daudz.....



4.2. zīm. Adresācijas iespējas komandā.

a -ar tiešo adresāciju; b -ar bāzes adresāciju.

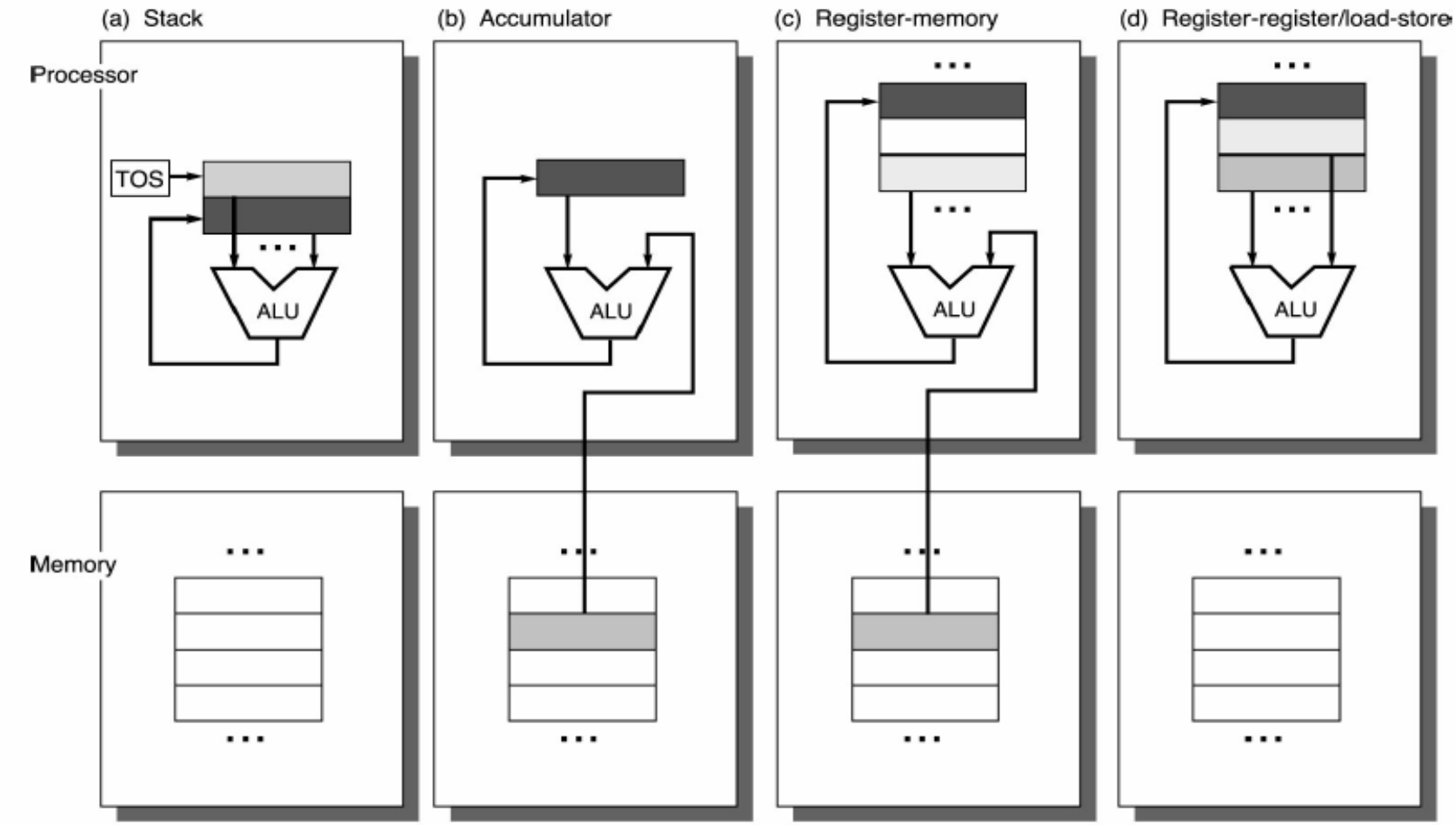
Komandu struktūra

- Mainīga
  - Komandu garums mainās atkarībā no operācijas koda un adrešu specifikātoriem (skaita / izmēra)
  - Piemēram VAX komandas var mainīties garuma ziņā no 1 līdz 53 baitiem bet x86 no 1 līdz 17 baitiem.
  - Kompakts kods bet grūti dekodēt un konveijerizēt
- Fiksēta
  - Vienāda garuma komandas
  - Piemēram MIPS, Power PC, Sparc
  - Ne tik kompakts kods bet to vieglāk dekodēt un konveijerizēt
- Jaukta
  - Var būt vairāki garuma formāti ko nosaka opkods
  - piemēram IBM 360/370
  - Kompromiss

Adresācijas veidi

Adresācijas veids	Piemērs	Darbība
1. Tiešā reģistru	Add R4, R3	R4 <- R4 + R3
2. Tūlītējā	Add R4, #3	R4 <- R4 + 3
3. Bāzes	Add R4, 100(R1)	R4 <- R4 + M[100 + R1]
4. Netiešā reģistru	Add R4, (R1)	R4 <- R4 + M[R1]
5. Indeksējamā	Add R4, (R1 + R2)	R4 <- R4 + M[R1 + R2]
6. Tiešā	Add R4, (1000)	R4 <- R4 + M[1000]
7. Netiešā atmiņas	Add R4, @(R3)	R4 <- R4 + M[M[R3]]
8. Autoinkrementā	Add R4, (R2)+	R4 <- R4 + M[R2] R2 <- R2 + d
9. Autodekrementā	Add R4, (R2)-	R4 <- R4 + M[R2] R2 <- R2 - d
10. Mērogotā	Add R4, 100(R2)[R3]	R4 <- R4 + M[100 + R2 + R3*d]

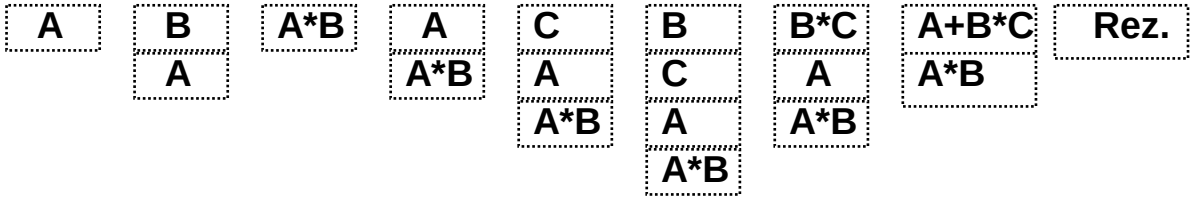
Komandu kopu klasifikācija



Steka komandu kopa

- Arhitektūra ar skaidri definētu "steku" kurš:
  - Darbojas datu avots un rezultāta vieta
  - Push un Pop komandām ir viena adrese pēc noklusējuma
- Komandu kopa:
  add, sub, mult, div, . . .
  push A, pop A
- Piemērs: A*B - (A+C*B)

```
push A
push B
mul
push A
push C
push B
mul
add
sub
```



Steka komandu kopa

- Pozitīvi:
  - Kompakts kods
  - Vienkārša aparatūras realizācija
  - Vienkārši kompilatori
- Negatīvi:
  - Steks ir “šaurā” vieta
  - Ļoti maz iespēju komandas izpildīt paralēli vai konveijerā
  - Dati ne vienmēr ir steka augšpusē (top, swap..)
  - Grūti veidot optimizējošus kompilatorus

Akumulatora komandu kopa

- Arhitektūra ar vienu netiešu reģistru kas:
  - Darbojas kā izeja un/vai mērķis
  - Otrs operands ir ***tiešs***
- Akumulators ir šaurā vieta?
  - x86 veselo skaitļu aritmētikas mezgls
- Pozitīvi:
  - Vienkāršāka aparatūras realizācija
  - Viegli realizēt un saprast
- Negatīvi:
  - Akumulators būs šaurā vieta*
  - Maz iespēju veidot paralēlu izpildi vai konveijerizāciju*
  - Liela atmiņas datu plūsma
- Komandu kopa:

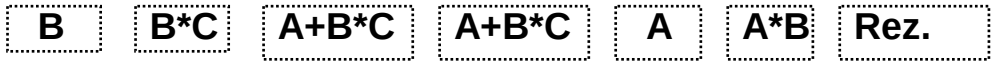
  add A, sub A, mult A, div A, . . .

  load A, store A

Akumulatora komandu kopa

- Piemērs: A*B - (A+C*B)

load B
mul C
add A
store D
load A
mul B
sub D



Atmiņa – Atmiņa komandu kopa

- Komandu kopa:

(3 operandi)	add A, B, C	sub A, B, C	mul A, B, C
(2 operandi)	add A, B	sub A, B	mul A, B

- Piemērs: A*B - (A+C*B)
  - 3 operandi

```
mul A, B, D
mul C, B, E
add E, A, E
sub D, E, E
```

2 operandi

```
mov A, D
mul B, D
mov C, E
mul B, E
add A, E
sub D, E
```

Atmiņa – Atmiņa komandu kopa

- Pozitīvi:
  - Maz komandas (it īpaši 3 operandu gadījumā)
  - Viegli veidot kompilatorus (it īpaši 3 operandu gadījumā)

- Negatīvi
  - Ļoti liela datu plūsma uz/no atmiņas (it īpaši 3 operandu gadījumā)
  - Mainīgs CPI
  - Divu operandu gadījumā vēl vajag papildus datu pārvietošanu

Reģistrs – Atmiņa komandu kopa

- Komandu kopa:

  add R1, A sub R1, A mul R1, B

  load R1, A store R1, A
- Piemērs: A*B - (A+C*B)

  load R1, A

  mul R1, B A*B

  store R1, D

  load R2, C

  mul R2, B C*B

  add R2, A A + CB

  sub R2, D AB - (A + C*B)

Reģistrs – Atmiņa ISA

- Pozitīvi
  - Daļu no datiem nevajag vispirms “ielādēt”
  - Vienkāršs komandu formāts un viegli tās iekodēt
  - Kompakts kods
- Negatīvi
  - Operandi nav vienlīdzīgi
  - Mainīgs CPI

Reģistrs – Reģistrs komandu kopa

- Visplašāk izplatītā komandu kopa
  - Ātra,neliela pēc apjoma uzglabāšanas vieta
  - Tieši operandi (reģistru IDs)
  - Visi RISC komandu kopas datori ir load/store arhitektūras

Mājas darbi

- http://dt.cs.rtu.lv/viewfile.php/18/file/99/454/4.clekcija.pdf
- Minēt kādu reģistrs-atmiņa komandu kopas arhitektūru
- Vai var saskaitīt, atņemt un veikt citas aritmētiski loģiskās darbības ar datora komandām?