Vad handlar kursen om?

- Pipelinade mikroprocessorer
- Minneshierarki: Cache och virtuellt minne
- Multiple-Issues processorer (Superskalär & VLIW)
- I/O - (Skivminne, bussar, SSD)
- Datoraritmetik (float)
- Flertrådade processorer
- Flerkärniga processorer
- Optimering av kod och minnessystem för en viss applikation (en Gauss-elimination)

Åtta stycken drivande ideér

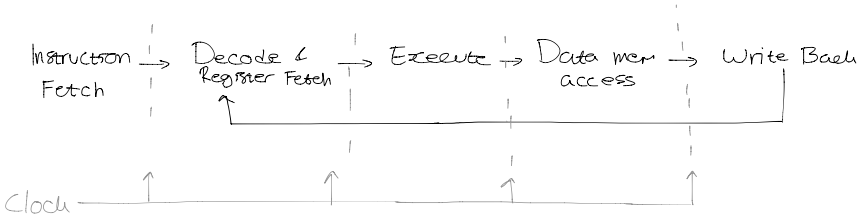
- Konstruktionerna baserade på Moores law
- Abstraktion för att hantera komplexitet
- Fokusera på common case
- Hög prestanda via parallellism
- Hög prestanda via pipelining
- Hög prestanda via prediktion
- Hierarki av minne
- Tillförlitlighet via redundans

Minnesteknologier

- Cacheminne görs av SRAM.
  - 0.5 ns - 2.5 ns
- Dynamisk RAM
  - 50ns - 70ns
- Magnetiska skivor
  - 5ms - 20ms

Idealt minne: Accesstid som SRAM
Kapacitet och kostnad/GB som hårddisk.

Prestanda via Pipelining



Effektutv = capacitiv last × Matningsspänning^2 × Frekvens

Multicores

- Fler än en CPU per chip
- Långsammare klocka → mindre värme
- Problemet flyttas till mjukvaran

Datorprestanda

Beror av: Den underliggande algoritmen, bestämmer antalet operationer
Programmeringsspråket
Processorn och minne
I/O-System (inkl. OS)

PC
- Stort SW-utbud
- Kostnad / prestanda

Server
- Nätverksbaserade
- Jobs/sekund

Superdator
- Högprestanda
- Top500.org
- Green500.org

CPU Clocking
- Klockperiod: längden på en klockcykel
- Klockfrekvens: cykler per sekund

Bästa mått för prestanda är: Ekkveringstid av typiska program! (Kolla slides.)

$$CPU_{Tid} = \frac{\text{Instruktioner}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruktion}} \times \frac{\text{Sekunder}}{\text{Klockcykler}} = IC \times CPI \times T_c$$

Prestandan beror av:
- Algoritm (IC, ev CPI)
- Prog språk (IC, CPI)
- Kompilatorn
- ISA

SPEC Benchmarks

Kostnad/Prestanda ökar
Pipelining & Cache = ökad prestanda
Exekveringstid.....

RISC
Reduced Instruction Set Computer
- Enklare instruktioner och addresseringssätt
- Gör processorerna mindre komplexa → snabbare
- De flesta moderna processorer är av RISC
  - x86 är dock CISC externt

CISC
Complex Instruction Set Computer
- Komplexa instruktioner och addressering

Mips
- Typexempel på RISC.
- Används i inbyggda system.
- Aritmetiska operationer använder två källregister och ett destinationsregister. Detta ger regularitet.
- Har 32×32 bit register file (RF)
  - 32 bit data kallas word

Massor av exempel. Finns i kap 1 av kompendium.

Processor

CPU prestanda: $T_{exe} = IC * CPI * T_c$

IC avgörs av ISA, kompilatorn, algoritmen och programspråket.

CPI avgörs av HW.

Instruktionsexekvering

PC → Instruktionsminnet, hämta instruktion

Registernummer → Registerfil, läs reg

Beroende på type:

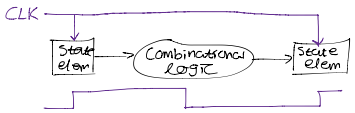
- Använd ALU för att beräkna: Aritmetiska resultat
  Minnesadresser för load/store instr
  Hoppadresser och hoppvillkor
- Läs/skriv dataminnet för load/store instr.
- PC = PC + 4 eller hoppadress

Logikkonstruktorer

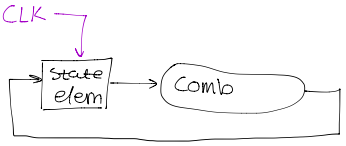
Info kodas binärt

Kombinatoriska element
  Grindar, osv

Tillståndselement
  Lagrar data



Dataväg



Tillståndsmaskin

R-Formcat Instruktioner

Läs två reg operander

Utför aritmetisk/logisk operation

Skriv resultatvärdet till registerfilen

I-format

lw $destreg, offs($basreg)

sw $källreg, offs($basreg)

Behöver teckenutvidga

Branch

beq/bne r_1, r_2, label

Teckenutvidga

Skifta vänster 2-bitpositioner

Addera detta till PC

Ickе pipelined

En instruktion per cykel.

Slide Chapter 4 - The processor - 18

Styrenhet

R-type: 0, rs, rt, rd, shamt, funct

L/S 35/43, rs, rt, offset

Jumps

Använder ord-addresser

Uppdatera PC med concat:
  Högsta 4 bitar av PC
  26 bitar från jump-instr

Prestanda

Den längsta signalfördröjningen avgör clockperioden

MIPS Pipeline

Fem pipesteg:

1: IF Instruction Fetch
2: ID Instruction Decode & Reg read
3: EX: Execute Instruction eller beräkna adress till DM
4: MEM: Access DM
5: WB: Write Back resultat till reg

Om alla pipesteg är balanserade: Pipeline Speedup = $\frac{T_{c,icke}}{T_{c,Pipe}} \approx$ Antal Pipesteg

Om pipen är obalanserad så är speedup mindre

Speedup pga genomströmning
Latenstiden för en enskild instruktion minskar dock inte

Konflikter (Hazards) i datavägen

Situationer som förhindrar start av instruktion i nästa klockcykel.

- Strukturell: Ett block är upptaget S 34
- Datakonflikt: Föregående instruktion ej klar S 35
- Styrkonflikt: Hoppvillkor och adress evalueras en bit in i pipen. S. 39

Pipeline Summering

- ? Pipelining ökar prestanda via ökad instruktions- genomströmning
- ? Flera instruktioner exekverar parallellt (via överlappning av deloperationer)
- ? Pipelinekonflikter (eng. hazards) måste hanteras
- ? Strukturkonflikt, datakonflikt (RAW), styrkonflikt
- ? Tydligt beroende mellan ISA och komplexitet hos pipeline implementation

Summering

- ? Pipelining förbättrar instruktions- genomströmningen via parallellism (överlappning)
- ? Fler instruktioner slutförs per sekund
- ? Kortare signalvägar möjliggör en högre CPU klockhastighet
- ? Hazarder: strukturell, data, styr konflikt
- ? Nästa föreläsning: Detaljerad styrning av pipelinen, samt Multiple-Issue och Spekulativa processorer

Styrning av Pipen

Hur löser vi RAW hazards? 4-8 §4.7
Vi kan använda forwarding. Finns några olika varianter i slides.

Double data hazard: Hazard på både ID/EX + EX/MEM och IE/EX + MEM/WB

add $s1, $s1, $s2
add $s1, $s1, $s3
add $s1, $s1, $s4

Då krävs reviderad forwarding 4-14

Hur löser vi load-use? 4-17
Liknande data hazard med fin aritmetik.
Om vi behöver stalla: sätts en NOP in.
: PC ändras inte.

Hazarder vid hopp? 4-24

Hoppstrategier - Fördröjda hopp
Kör alltid instruktionen efter hoppinstruktionen (delay slot)
Fyll den platsen med en nyttig instruktion, annars NOP

Mer avancerade metoder 4-33
Djupare pipes => prestanda förluste vid hopp större
Använd dynamisk hopp-prediktion
Branch Target Buffer

Instruction Level Parallelism 4-39
Exekvera flera instruktioner parallellt via överlappning

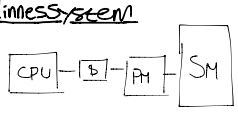
Öka ILP => Djupare pipelines => Mindre arbete p steg
Fler hazarder

Multiple Issue CPU

Mips med statisk Dual Issue 4-42

Loop unrolling 4-45

Minnessystem



* Minnesväggen
* Minnes Hierarki
  - Cacheminne
    - Direktmappad alt associcetive
  - Flernivåscache
  - Hur ökar vi bandbredden mellan $ och PM.
  - Sekundärminne
    - Skriviminne
    - Flash

Minnesteknologier

- SRAM
  - 0.5ns - 2.5ns, $2000 - $5000 per GB
- DRAM
  - 50ns - 70ns, $20 - $75 per GB
- Magnetiska Skivor
  - 5-20 ms, $0.2 - $2 per GB

Idealt minne: Accesstid som SRAM
Kapacitet och kostnad som för hårddisk

Lokalitetsprincipen (slide 7)

Instruktioner och data accessas ganska sekvensiellt.

Spatial
Om en address accessas är det sannolikt att en närliggande adress kommer att accessas snart.

Temporal
Om en address accessas är det sannolikt att samma adress kommer accessas snart igen.

Cacheminne

Ett litet, snabbt, minne nära processorn och större långsammare minne längre ifrån.

h_x = Hit rate
m_x = Miss rate
T_x = Hit time
$(T_y - T_x)$ - Miss penalty

$$AMAT = h_1T_1 + m_1T_2 = h_1T_1 + (1-h_1)T_2$$

Fyra viktiga:
1. Var skall blocket placeras?
2. Hur hittar vi?
3. Slide 28
4.

Cache adressering

Tag | Index | Offset

~Slide 30

Massor av exempel, se slides.

forts.

Hur ökar vi bandbredden till PM? slide 60-62

Sekundärminne

Flash

100-1000 ggr snabbare än skivminne
Mindre och mer robust
Kostar mellan DRAM och skivminne

NOR-flash
- Random read/write access
- Används som instruktionsminne i inbyggda system.

NAND-flash
- Tätare bits/area men access sker blockvis.
- Billigare per GB.

Flash nöts ut med tiden!

Skivminne

Slide 67....

Inlämningsuppgift

Implementera gauss-elimination i MARS-assembler. (Flyttal)
Optimera
Väl cache parametrar
OSV OSV

```
for(i=0; i<N; i++)
    A[i] = A[i+M]
```

Två block med samma index ger miss vid varje access

Prestandaberäkningar - Tenta 2010-05-25

a) 1000 mätningar · 10 ms = 10 s

$T = IC \cdot CPI \cdot T_c$

$IC = 2 \cdot 10^9$, $T_c = \frac{1}{10^8} = 10^{-8}$, $CPI = ?$

$CPI = 1.3 + CPI_{\Delta D\$} = 1.3 + 0.10 \cdot 10 \cdot 0.2 = 1.5$

(0.10: Missrate, 10: Miss penalty, 0.2: Andel data rel)

$T = 2 \cdot 10^9 \cdot 1.5 \cdot 10^{-8} = 30$ s

$T_{tot} = 30 + 10 = 40$ ms

b) I$: MP = 10 cykler, MR = 0.01

$CPI_{\Delta I\$} = 10 \cdot 0.01 = 0.1$

Beräkningstiden minskar med faktorn $\frac{1.5 - 0.1}{1.5} = 0.933$

$\frac{T_{tot}}{T_{tot}} = \frac{10 + 30 \cdot 0.933}{10 + 30} = 0.95$

c) Halvera waiting time på sensorn eller halvera miss penalty?

Halvera väntetiden => 5s förbättring
Halvera inverkan av D$ missar => $\frac{0.2}{2} = 0.1$
Hela beräkningstiden minskar med $\frac{0.1}{1.5} \cdot 30 = 2$s

Det är alltså bättre att halvera väntetiden på sensorn.

Virtuellt Minne

Man har virtuella och fysiska adresser. Den virtuella är oftast större vilket ger illussionen av ett större PM.

VM använder PM som cache för SM.
Varje program har ett eget virtuellt adressrum.

Physical address: Address in PM

PAGE FAULT

Sidan måste hämtas från disk.
- Tar miljoner clock cycles
- Hanteras av OS kod

Det finns sidtabeller i PM som innehåller placeringsinformation.

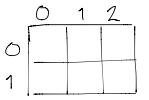
5.1)

```
for (i=0; i<8; i++)
   for (j=0; j<8000; j++)
      A[i][j] = B[j][0] + A[j][i];
```

5.1.1 How many 32 bit ints in 16 byte cache block?
4

5.1.2 Which variables exhibit temporal locality?
j, i

5.1.3 Spatial locality
C codes gets stored rows first.



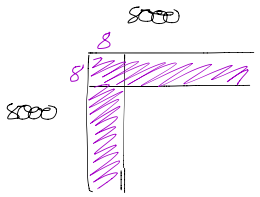
3×2 So we have spatial locality when accessing several items on the same row.

```
for i=1:8
  for j=1:8000
    A(i,j) = B(i,0) + A(j,i);
  end
end
```

5.1.4 How many 16-byte cache blocks are needed to store all 32-bit elems referenced?
$(8\times8000 + 8000 + 8000\times8 - 8\times8)\cdot\frac{1}{4} = 33984$

5.1.5 Temporal
i, j

5.1.6 Spatial
A(j,i), B(j,0)

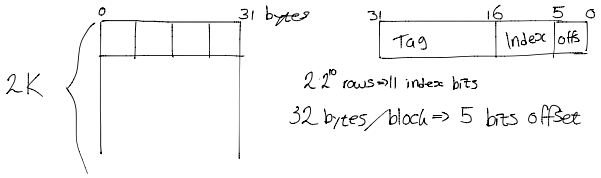


5.5)
512 KiB working set with address stream: 0 2 4 6 8 10 12 ... (512 KiB)

5.5.1 Assume 64 KiB direct mapped cache with 32-byte block. What is the MR? How is it sensitive to the set size or cache size?

$\frac{64}{32}$ = 2K blocks

1 bytes per elem
32 bytes per block



2^{11} rows → 11 index bits
32 bytes/block ⇒ 5 bits offset

First access ⇒ Miss, read 16 elems to cache (1, 3, 5, ... garbage)
17th access ⇒ —— 11 ——

Miss every 16th access. Only cold misses due to streams not having locality.

5.5.2 MR for 16-byte, 64 and 128-byte block size.

16 byte: 1/8
64 byte: 1/32
128 byte: 1/64

What kind of locality is exploited?
Spatial. We read sequentially.

5.5.3 Assume a two-entry stream buffer.
First read would give a miss, then no misses.
Cache block size (B) can affect both miss rate and miss latency. Assuming CPI=1 and an avg of 1.35 references per instruction find the optimal block size given the following MR for block sizes:

B:	8	16	32
	4%	3%	2%

5.5.4 What is the optimal block size for miss latency of $20 \times B$ cycles?

B=8: 0.04(8·20)
B=16: 0.03(16·20) 8 wins!
⋮

5.5.5 $24+B$ cycles. 24 units of time to fetch the first elem and the consecutive elems takes 1 unit.

B=8: 0.04(24+8)
B=8: 0.03(24+16)
⋮

5.5.6 For const miss latency?
Only the missrate matters!

5.11

5.11 As described in Section 5.7, virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. This exercise shows how this table must be updated as addresses are accessed. The following data constitutes a stream of virtual addresses as seen on a system. Assume 4 KiB pages, a 4-entry fully associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number.

4669, 2227, 13916, 34587, 48870, 12608, 49225

TLB

Valid	Tag	Physical Page Number
1	11	12
1	7	4
1	3	6
0	4	9

Page table

Valid	Physical Page or in Disk
1	5
0	Disk
0	Disk
1	6
1	9
1	11
0	Disk
1	4
0	Disk
0	Disk
1	3
1	12

5.11.1 Show the final stage of the system given the above info. Also show if it's a hit in TLB Page table or page fault.

4 KiB pages $\Rightarrow 4 \cdot 2^{10} \Rightarrow$ 12 bits to encode

4669: 0001 0010 0011 1101 (12 bits)
Tag — Byte offset in page

Tag 1 is not in TLB $\Rightarrow$ TLB miss
(Tag = Virtual Page)

5.11.2 Use 16 KiB instead of pages. Advantages / Disadvantages?

+ Fewer misses since you load larger chunks.
- Larger miss penalty!

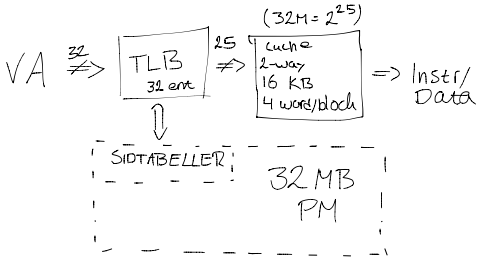
Example: TLB och Cacheinteraktion

Virtuellt minne exempel (tentamensfråga 4 2010-08-16

- Assume a computer systems with the following characteristics: The CPU uses virtual memory (unified data- and instructions) using 32-bit virtual addresses. There is a maximum of 32 MB of physical primary memory, and a two-way set associative 16 KB cache. The hit-time in this cache is 1,5 ns. A 32-entry fully associative TLB is used for virtual-to-physical address translations . The TLB hit-time is 1 ns. The page size is 16 KB, and the cache uses a block size of 4 words.
- The cache, the TLB, and the virtual memory uses a "write-back" ("copy-back") strategy. The replacement algorithm used by the cache is "Random", and for the virtual memory an approximative "LRU" using 2-bit time "time stamps" per page. Virtual memory pages that belongs to the OS are marked with a "protection bit" to protect it from user process accesses. In maximum, 16 processes are active at any given time. The Page Tables do not store secondary storage addresses when the pages are not in primary memory.

2010-08-16, uppg.4 , forts

- a. How many bits must the cache memory be able to store ? (4 p)
- b. How many bytes is needed to store the Page Tables ? (4 p)
- c. How many bits must the TLB be able to store ? (4 p)
- d. What is the maximum clock frequency the processor can have (assuming the critical timing path is constrained by TLB and cache) (6 p)?



a) Cache entry: | V | D | TAG | Data | (1 1, 4·32 = 128 bit)

Offset: 2^4 ⇒ 16 byte

Index: $\frac{16\,KB}{16\,B}$ = 1K blocks, 2-way ⇒ $\frac{1K}{2} = 512 = 2^9$ ⇒ 9 bitar

Tag: 25 − 9 − 4 = 12 bitar

1 + 1 + 12 + 128 = 142 bitar/block ⇒ 142 · 1024 = 145408 bitar

b) PTE (Page Table Entry): | V | D | P | LRU | Physical Page | (1 1 1 2)

Page offset: 16 KB ⇒ 2^{14} bits

Physical Page: 25 − 14 = 11

PTE = 1 + 1 + 1 + 2 + 11 = 16 bitar

There's one PTE for each virtual page.

How many pages? 2^{32-14}, 32: VA, 14: Page offset = 2^{18} = 256 K

256K · 2B · 16 = 8MB (16 ← #processes)

c) TLB-entry: | V | D | P | LRU | TAG | Physical Page | (1 1 1 2)

Tag in TLB = Virtuell page number = 18

Physical Page = 11

Totalt: 1 + 1 + 1 + 2 + 18 + 11 = 34 ⇒ 32 entries in TLB · 34 = 1088 bitar i TLB

d) Physically addressed cache: VA → TLB (1 ns) → FA → $ (1.5 ns) → Instr/Data $f_{max} = \frac{1}{1+1.5} = 400$ MHz

Pipelined: VA → TLB → [] → $ → $f_{max} = \frac{1}{1.5} = 667$ MHz

Virtually addressed: VA → $ → Instr/Data; VA → TLB → FA. TLB is not in CP ⇒ TLB is only used if cache miss. $f_{max} = \frac{1}{1.5} = 667$ MHz

Mipsaritmetik

Multiplikation: Två 32-bitars register för produkten
- HI (32 MSB)
- LO (32 LSB)

Division: Använder HI/LO registren för resultat
- HI: rest
- LO: kvot

Flyttal

Representation av icke-heltal.

Binärt: $\pm 1.xxxxx_2 \cdot 2^{yyy}$ ← Normaliserat

Det gäcka värdena förs...

Standardiserad enl IEEE

S	Exp	Fraction

$x = (-1)^S \cdot (1 + \text{Fraction}) \cdot 2^{(\text{Exponent} - \text{Bias})}$

Exponent: Excess representation: aktuell exponent + Bias

Sök på flyttal på dochdetdå.com!

Exempel °F → °C

C-kod:

```
float f2c(float f){
    return ((5.0/9.0)*(f-32));
}
```

MIPS:

```
f2c: lwc1   $f16, const5($gp)
     lwc1   $f18, const9($gp)
     div.s  $f16, $f16, $f18
     lwc1   $f18, const32($gp)
     sub.s  $f18, $f12, $f18
     mul.s  $f    $f    $f
     jr     ra
```

Exempel från tenta

a) Nyttjar lokaliteten bättre.
b) Får bara plats med 4 block => Fler konflikter.
c) Vad är den genomsnittliga missfrekvensen?

4B: $0.38(4+4/4) = 1.9$
16B: $0.22(4+16/4) = 1.76$
64B: $0.19(4+64/4) = 3.8$
256B: $0.27(4+\frac{256}{4}) = 18.36$

Exempel Hopphazarder

```
    ADDI  $5, $0, 2
L1: LW    $4, 100($5)
    ADDI  $5, $5, -1
    ADD   $3, $3, $4
    BNE   $5, $0, L1
    SW    $3, 100, $0
```

Always stall: 2 cykler extra pga hazards: Om hoppvillkor beräknas och används i ID. IO+2+4 = 16 (4: Töm pipe) Om Ex: IO+2·3+4=20
Assume not taken: Gissar fel en gång. => 1 cykel extra om hoppvillkoret beräknas och används i ID-steget
IO+1+4=15
Om vi beräknar i EX och använder i MEM => 3 cykler extra.
IO+3+4=17

L1 körs 299r.
Alla databazarder kan lösas med forwarding.
Hur många extra cykler pga hopphazarder?
Räkna inte med förutsagda hopp!

Multithreading (MT)

Flera PC, fler registerfiler, snabb växling mellan trådar.
Fin resp grovkornig MT.

Parallellism exempel

Summera 10 tal sekventiellt och utför också en 10×10 matrisaddition. Vi kan använda 10 eller 100 CPUs

1 CPU: $T = (10+100) \cdot t_{add} = 110 \cdot t_{add}$

10 CPU: $T = 10 \cdot t_{add} + \frac{100}{10} \cdot \ldots\ldots$

Tenta Q1 2010-08-16

RAW

```
    slt   v0, a0, a1
    beq   v0, zero, L1
    add   v0, a0, zero
    add   v1, a1, zero
    beq   zero, zero, L2
L1: add   v0, a1, zero
    add   v1, a0, zero
L2
```

a) Sorts the values in the registers depending on the values.

b) In data: a_x

Out: v_x

c) How many stalls if $a_0 = 5$, $a_1 = 7$.

1. beq vill ha v0 i ID men kan inte få det förrän slt skrivit tillbaka i WB. Det krävs 2 stallcykler för detta.
2. 3 stall cykler

Summa: 5

d) För att göra till en subrutine: Lägg till Label och avsluta med jr $ra

anrop: addi a0, zero, 5
addi a1, zero, 1
jal Label

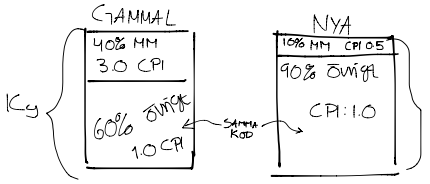
Q2 2009-05-19

50 computers @ 500 MHz

Offer: 1 GHz samma ISA
600 MHz bättre ISA

1 GHz = dubbelt så snabba som 500 MHz ty samma ISA.

$T_{exe} = IC \cdot CPI \cdot T_c$



$T_G = T_{mmG} + T_{oG} = 0.4 \cdot IC_G \cdot 3.0 \cdot \frac{1}{500 \cdot 10^6} + 0.6 \cdot IC_G \cdot 1 \cdot \frac{1}{500 \cdot 10^6}$

$T_N = T_{mmN} + T_{oN} = 0.1 \cdot IC_N \cdot 0.5 \cdot \frac{1}{600 \cdot 10^6} + 0.9 \cdot IC_N \cdot 1.0 \cdot \frac{1}{600 \cdot 10^6}$

$0.9 \cdot IC_N = 0.6 \cdot IC_G \Rightarrow IC_N = \frac{0.6}{0.9} IC_G$

Uppsnabbning $\Rightarrow \frac{T_G}{T_N} = \frac{(0.4 \cdot 3.0 + 0.6) \cdot IC_G \cdot \frac{1}{500 \cdot 10^6}}{(0.1 \cdot 0.5 + 0.9) \cdot \frac{0.6}{0.9} IC_G \cdot \frac{1}{600 \cdot 10^6}} = 3.41$

1 GHz ger 2 ggr ECO

600 MHz ger 3.41 ggr ECO

Q3 2011-01-12

a) Avg read/write time?

Sector size: 512 byte

7200 RPM

Avg seek time: 8 ms

Transfer rate: 20 MiB/s

Controller OH: 2 ms

Disk access = Seek + rotational latency + transfer + controller OH =

$8 + 0.5 \cdot \frac{60}{7200} \cdot 1000 + \frac{512}{20 \cdot 2^{20}} \cdot 1000 + 2 = 14.19$ ms

b) 3 step process: Read 4KiB $= 8 + 0.5 \cdot \frac{60}{7200} \cdot 1000 + \frac{4 \cdot 1024}{20 \cdot 2^{20}} \cdot 1000 + 2 = 14.37$ ms

Process $= \frac{20 \cdot 10^6}{400 \cdot 10^6} = 50$ ms

Write 4KiB $=$ Read $= 14.37$ ms

Total time: $2 \cdot 14.37 + 50 = 78.74$ ms $\Rightarrow \frac{1}{78.74} \cdot 1000 = 12.76$ blocks/s

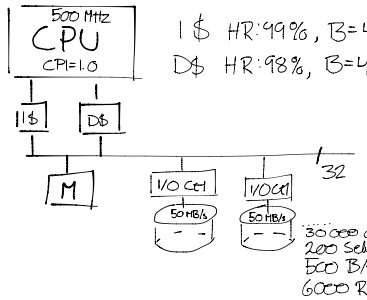
c) What is the bottle neck?

Time to process 64 Ki block $\frac{20 \cdot 10^6}{3 \cdot 10^9} = 0.67$ ms.

Transfer a 64 KiB block on bus: $\frac{64 \cdot 2^{10}}{640 \cdot 2^{20}} = 0.097$

Time for I/O $9 + \frac{64 \cdot 2^{10}}{64 \cdot 2^{20}} \cdot 1000 + \frac{10^6}{3 \cdot 10^9} \cdot 1000 = 10.33$ ms

Q4 2009-05-26



I$ HR: 99%, B=4 ord
D$ HR: 98%, B=4 ord

25% reads
5% write

30 000 cylinder
200 Sektorer/spår
500 B/sektor
6000 RPM

7ms seek + rotational latency
4:2 ytor

Bussen används för överföring av 4 ord åt gången. (Address + Block = 1+4=5 cykler)

a) $\text{Busstid}_{I\$Miss} = 0.01 \cdot 5 \cdot 5 = 0.25$

Missrate, MP, CPU Egen synlighet på buss, # CPU cykler / # Busscykler

$\text{Busstid}_{D\$Miss,läs} = 0.02 \cdot 5 \cdot 5 \cdot 0.25 = 0.125$

$\text{Busstid}_{D\$Miss,skriv} = 0.05 \cdot 2 \cdot 5 = 0.5$

Data + Cykler

Systembussen är upptagen med trafik mellan cache och minne: $0.25 + 0.5 + 0.125 = 87.5\%$ av tiden.

b) Max bandbredd per disk R/W-huvud $= \frac{6000}{60} \cdot 500 \cdot 200 = 10$ MB/s

→ 8 ytor: $8 \cdot 10$ MB/s $= 80$ MB/s

c) Max bandbredd på systembussen $= \frac{4 \text{ ord}}{5 \text{ cykler}} = \frac{4 \cdot 4 \text{ B}}{\frac{5}{100 \cdot 10^6}} = 320$ MB/s

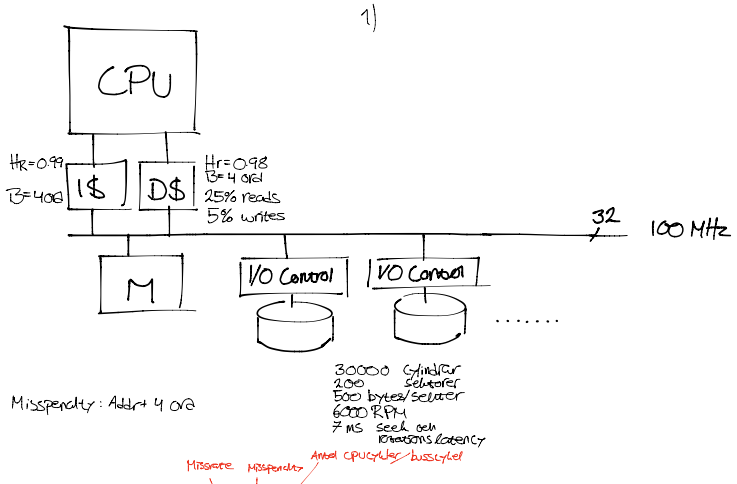
d) Använd bandbredd per I/O-buss?

DMA (128 K block) $= 2 \text{ ms} + 7 \text{ ms} + \frac{128 \cdot 2^{10}}{10 \cdot 2^{20}} \cdot 1000$

DMA setup, SEEK + ROT

Tenta Q4 2009-05-26

1)



Missrate · Misspenalty · Antal CPUcykl/ax · busscykel

1) $\text{Busstid}_{I\$Miss} = 0.01 \cdot 5 \cdot 5 = 0.25$

2) Läsmiss

$\text{Busstid}_{D\$RM} = 0.02 \cdot 5 \cdot 5 \cdot 0.25 = 0.125$ (0.25 = 25% reads)

Skrivmiss (5% writes)

$\text{Busstid}_{D\$SM} = 0.05 \cdot 2 \cdot 5 = 0.5$ (2 = Addr+Data)

$\text{Busstid}_{Miss} = 0.25 + 0.125 + 0.5 = 0.875$ => Bussen är upptagen med missar 87.5% av tiden. 12.5% går alltså att använda till I/O-operationer.

3) Max bandbredd per disk R/W-head: $\frac{6000}{60} \cdot 500 \cdot 200 = 10$ MB/s. 8 ytor => 80 MB/s.

4) Bandbredd för bussen: $\frac{4}{5} \frac{\text{ord}}{\text{cykel}} = \frac{16\text{B}}{\frac{5}{100 \cdot 10^6}} = 320$ MB/s

5) Använd bandbredd per I/O-buss?

DMA (128 KiB) = 2 ms (DMA Setup) + 7 ms (seek/rotation) + $\frac{128\text{ KiB}}{10\text{ MB/s}} \cdot 1000 = 22.1$ ms (transfertid)

2 DMA överföringar kan ske samtidigt => $\frac{2 \cdot 128\text{ KiB}}{22.1\text{ ms}} = 11.86$ MB/s (effektiv databandbredd per I/O-buss)

Bandbredd för disk > Bandbredd för I/O-Buss, alltså är det I/O-bussen som flaskar.

6) $0.125 \cdot 320 = 40$ MB/s ledigt för I/O => $3 \cdot 11.86 < 40$ => Vi kan ha 3 I/O-kontroller.

Tid ledig

Q2a. 2011-01-12

add	R4 = R1 + R0	Lila löses med ALU FWD
sub	R9 = R3 - R4	Röd ger ingen hazard.
add	R4 = R5 + R6	Grön måste stallas en cykel och sedan använda ME -> EX FWD.
lw	R2 = M[R3 + 100]	
lw	R2 = M[R2 + 0]	
sw	M[R4 + 100] = R9	Uppgiften är dock att göra ett diagram som beskriver dessa.
and	R3 = R3 & R1	
beq	R9 == R1, Target	
and	R9 = R9 & R1	

Q4. 2012-01-11

Physical

Cache | TAG | SetIndex | Byteoff |

Blocksize: 128 B => Byte offset = 7 = $\log_2(128)$ => C = 7
Cachesize: 64 KiB => $\frac{64 \cdot 2^{10}}{128 \cdot 2} = 256$ => Index = 8 => B = 8 2-way assoc
Physical address = 32 bits => Tag = 32 - 8 - 7 = 17 => A = 17
128B block => 128·8 = 1024 bitar => D = 1024
Tag/Status: 17 + (1+1) = 19 => E = 19 Valid+Dirty

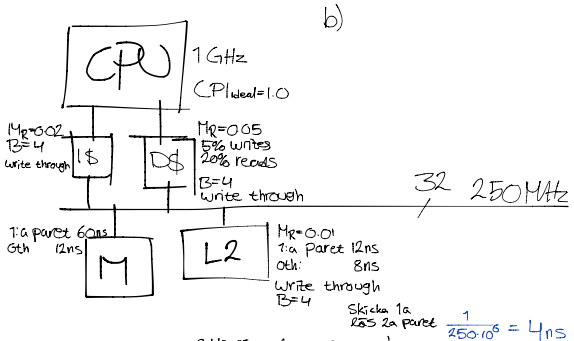
Virtual

TLB | Tag | sets/Index | Page offset |

Virtual address: 64 bitar
8 KiB sider: 13 bitar ($2^3 2^{10}$) sido offset => H = 13
512 entries: $\frac{512}{2} = 256$ => 8 bitar index => G = 8
Tag: 64 - 8 - 13 = 43 bitar => F = 43
PA = 32 bitar, Page = 8KiB => 32 - 13 = 19 bitar physical pageno => I = 19

Q4 2009-08-17

a) Beräkna CPI
b)



Skicka 1a läs 2a paret $\frac{1}{250 \cdot 10^6} = 4$ ns

a)
address, läs 1a paret, Skicka 2a paret
L2s missp enalty: (4) + 12 + 8 + 8 = 32 ns

PMs mp: 4 + 60 + 12 + 8 = 84 ns
Det tar 8ns att skicka men 12 att hämta.

ΔCPI

Inst.hämt L2: $0.02 \cdot 32 = 0.64$ (MR_{L1})
Inst.hämt PM: $0.01 \cdot 0.02 \cdot 84 = 0.0168$ (MR_{L2}, MR_{L1})
Data.läs L2: $.005 \cdot 0.20 \cdot 32 = 0.32$ (20% reads)
Data.läs PM: $0.005 \cdot 0.01 \cdot 0.2 \cdot 84 = 0.0084$

CPI = 1.0 + 0.64 + 0.0168 + 0.32 + 0.0084 = 1.9852