Tentamen i kursen Datorsystemteknik (EDA330 för D och EDA370 för E)

Datorsystemteknik för D/E

19/5 2001

Tentamensdatum: Lördag 19/5 2001 kl. 14.15 i sal V

Examinator: Jonas Vasell (D) och Peter Folkesson (E)

Institution: Datorteknik

Förfrågningar: Peter Folkesson (ankn. 1676)

Lösningar: anslås måndag 21/5 på institutionens anslagstavla utanför laboratoriet

Resultat: anslås senast tisdag 5/6 på institutionens anslagstavla utanför laboratoriet

Rättningsgranskning: tid och plats anslås tillsammans med resultaten

Betygsgränser: 3: 24-35 poäng, 4: 36-47 poäng, 5: 48-60 poäng

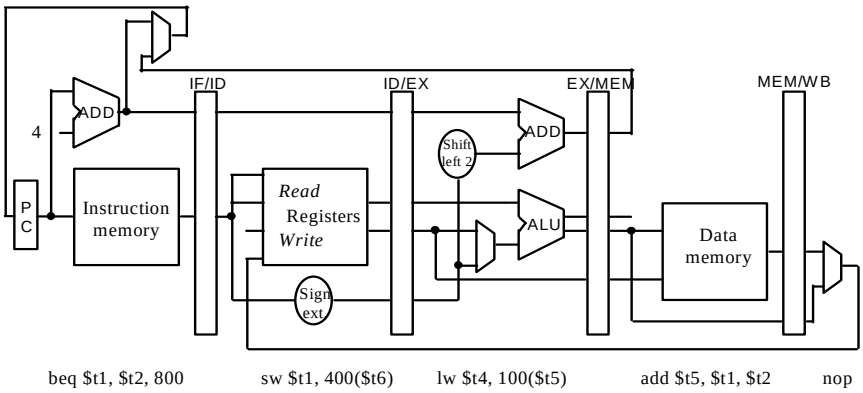
Tillåtna hjälpmedel: Typgodkänd kalkylator

Allmänt: För full poäng på en uppgift krävs både ett korrekt svar och en motivering. En bra motivering är minst lika viktig som ett korrekt svar. Redovisa noggrannt alla gjorda antaganden utöver de som anges i uppgiftstexten. Skriv tydligt och använd gärna figurer. Maximal poäng på varje deluppgift anges inom parentes efter uppgiftstexten.

Lycka till!

Uppgifter (1-5):

1. a. Vilken typ av pipelinekonflikt uppstår i nedanstående figur då den givna instruktionsföljden exekveras. Var uppstår konflikten? (1 p)



b. Nämn två sätt att lösa konflikten i uppgift a)? Rita figur. (2 p)

c. Beräkna den exakta CPU-tiden för nedanstående program. Programmet exekveras av en 500 MHz MIPS processor med 5 pipelinesteg och separat instruktions- och datacache. Det får förutsättas att cachen rymmer all programkod och data. Eventuella datakonflikter (*data hazards*) löses med *forwarding* (*bypassing*) och vid styrkonflikter (*control hazards*) används fördröjt hopp (*delayed branch*). En översikt av MIPS maskininstruktioner finner du i bilaga 1. (6 p)

```
        addi  $a1,  $zero, 10
        add   $t1,  $a1,  $zero
L1:     add   $a2,  $t1,  $zero
        add   $a2,  $a2,  $a2
        add   $a2,  $a2,  $a2
        addi  $t0,  $zero, 1
        sub   $a1,  $a1,  $t0
        jal   L3
        nop
L2:     sub   $a2,  $a2,  $t0
        jal   L3
        nop
        bne   $a2,  $zero, L2
        sw    $v0,  0($a3)
        bne   $a1,  $zero, L1
        nop
        j     L4
        nop
L3:     sw    $t1,  0($sp)
        lw    $t1,  0($a1)
        add   $v0,  $v0,  $t1
        lw    $t1,  0($a2)
        add   $v0,  $v0,  $t1
        add   $v0,  $v0,  $v0
        add   $v0,  $v0,  $v0
        lw    $t1,  0($sp)
        jr
        nop
L4:     sw    $v0,  4($a3)
```

d. Ge förslag på tre olika förbättringar (optimeringar) av koden i uppgift c) så att CPU-tiden minskar. (3 p)

2. Antag att vi har ett datorsystem med följande karakteristik: Processorn adresserar virtuellt minne (kombinerat data- och instruktionsminne) med 52-bitars virtuella adresser. Det finns maximalt 256 MB fysiskt primärminne, och ett fyrvägs associativt cacheminne med kapacitet att lagra 64 KB data. Cacheminnets åtkomsttid vid träff är 5 ns. För sidöversättningar finns en tvåvägs associativ TLB för 128 översättningar. TLBns åtkomsttid vid träff är 3 ns. Sidstorleken är 4 KB, och cacheminnets blockstorlek är 32 bytes. För såväl cacheminnet, TLBn som det virtuella minnet tillämpas write-back (copy-back) som skrivningsstrategi. Som utbytesalgoritm för cacheminnet och det virtuella minnet används LRU med 2-bitars tidsstämpel för varje block (sida). För TLBn tillämpas LRU med 1 bits tidsstämpel för varje sidöversättning. Virtuella sidor som tillhör operativsystemet är markerade med en flaggbit (protection bit) för att skyddas mot åtkomst från användarprocesser. Sekundärminnesadresser för sidor som inte finns i primärminnet lagras inte i sidtabellerna.

   a. Vad är det totala antalet bitar som cacheminnet måste kunna lagra? (4 p)

   b. Hur stort minnesutrymme räknat i bytes krävs för att lagra en sidtabell om ett helt antal bytes används för att lagra varje rad i sidtabellen. (3 p)

   c. Vad är det totala antalet bitar som TLB måste kunna lagra? (3 p)

   d. Vilken klockfrekvens kan processorn ha som bäst om man antar att en minnesåtkomst i normalfallet ska klaras av inom en processorklockcykel? (2 p)

3. Du har som processortillverkare ställts inför ett svårt val. I samband med en processkrympning från 0.25 µm till 0.18 µm har möjligheten öppnats för att lägga L2-cache på processorchipet. Tidigare versioner av processorn har haft 2x64 kB L1-cache och 4 MB extern L2-cache och nu vill man se vad man kan uppnå genom att lägga L2-cachen på processorchipet och behålla den ursprungliga L1-cachen.

   Den befintliga processorn kostar 500 kr att tillverka och till detta kommer 4 MB L2-cache i SRAM kretsar som kostar 2000 kr.

   I den nya tillverkningsprocessen kan man räkna ut att processorn kommer kosta enligt följande tabell:

L2-Cache	Pris
0 kB	250 kr
256 kB	350 kr
512 kB	500 kr
1024 kB	1000 kr
2048 kB	3000 kr

Större cache än 2 MB på processorchipet är ur tillverkningssynpunkt orimligt.

En fördel med att lägga L2-cache på processorchipet är att man får mycket snabbare accesstid till cachen. Med den gamla designen tog en access till L2 cachen 30 ns. Med den nya designen kommer en access endast att ta 10 ns. Eftersom man kommer använda samma typ av primärminne till den nya processorn så gäller det att en primärminnesaccess tar 125 ns för bägge processorerna. Man accessar inte en nivå i minneshierarkin förrän man vet att man har missat i tidigare nivå.

För ett representativt benchmark vet man vad träffsannolikheterna för olika L2-cache storlekar är. Man vet att 5 % av alla dataminnesaccesser missar i L1-datacachen och 1 % av alla instruktionshämtningar missar i L1-instruktionscachen samt att 25 % av alla instruktioner är av datatransfer typ.

L2-Cache	Träffsannolikhet
0 kB	0 %
256 kB	55 %
512 kB	75 %
1024 kB	80 %
2048 kB	85 %
4096 kB	90 %

Målfrekvensen för den nya processorn är 1GHz medan den gamla processorn endast kunde köras i 800MHz. Den gamla processorn tog 20 sekunder att köra benchmarket som är helt processor/minnes begränsat (dvs. ingen tid tas upp av I/O). Programexekveringen består av ca 10 miljarder instruktioner.

a. Vilken av de nya cachekonfigurationerna ger bäst prestanda/kr förhållande? (8p)

b. Hur lång tid kommer benchmarkprogrammet ta att köra på den nya processorn med den konfiguration du fann i uppgift a)? (2p)

c. Vilken är den billigaste av de nya processorerna om man vill att prestandan skall öka mer än linjärt med klockfrekvensen? (2p)

4. I ett visst datorsystem är en CPU med klockfrekvensen 400 MHz och ett gränssnitt mot en systembuss kopplade till primärminnet (DRAM) via en processor-minnebuss. Denna buss är synkron med frekvensen 200 MHz och multiplexad överföring av ett ord (32 bitar) data eller adress varje busscykel. En överföring kan räknas som mottagen först i slutet av busscykeln.

   En enhet (CPU eller systembussgränssnitt) som vill använda bussen för en transaktion skickar en begäran till en busstyrenhet som gör arbitrering och tidigast i påföljande busscykel skickar tillbaka ett tillstånd att använda bussen. Om bussen är upptagen skickas tillståndet så snart bussen kommer att vara fri i påföljande

busscykel. När enheten fått tillstånd att använda bussen kan den börja sin transaktion i busscykeln efter att tillståndet givits. Bussen hålls kvar av enheten till hela transaktionen är klar, men måste sedan släppas.

Varje transaktion på bussen innebär att data motsvarande ett block data som omfattar 16 byte läses från eller skrivs till primärminnet, vilket motsvarar ett block i CPUns inbyggda cache. Primärminnet behöver 75 ns för åtkomst av ett block efter att det fått en blockadress, och därefter kan orden i blocket överföras ett i taget varje busscykel. Förfarandet är oberoende av om det är en läsning eller skrivning, skillnaden är bara i vilken riktning data skickas.

a. Hur många processorcykler tar det som mest från det att en cachemiss detekteras i CPU tills det sökta blocket laddats in om man antar att write-back tillämpas och att processor-minnebussen inte är upptage av annan aktivitet (t.ex. DMA från systembussen)? (6 p)

b. Vad är den maximala effektiva bandbredd som kan uppnås med processor-minnebussen i föregående deluppgift? (2 p)

c. Räkna upp de fyra typer av bussarbitrering som gåtts igenom i kursen och förklara hur en av dem fungerar. (4p)

5. Nedan följer ett antal frågor med tre svarsalternativ (1, X, 2) vardera, varav endast ett är rätt. Ställ upp svaren som en tipsrad. Varje rätt svar ger ett pluspoäng och varje felaktigt svar ger ett minuspoäng. Inget svar ger noll poäng. Minsta poäng på hela uppgiften är noll. (12 p)

a. Vad avses med begreppet utbyte (*yield*) vid tillverkning av VLSI-kretsar? (1) Antalet kretsar (*dies*) som kan produceras per kiselbricka (*wafer*). (X) Andelen felfria kapslar (*packaged dies*) av det totala antalet kretsar producerade per kiselbricka. (2) Andelen felfria kretsar (*dies*) av det totala antalet kretsar producerade per kiselbricka.
b. Ett cacheminne bör implementeras med (1) DRAM-kretsar. (X) SRAM-kretsar. (2) Skivminne.
c. Om kravet är att programkoden i ett datorsystem måste vara extremt minnessnål, bör man välja en processor med (1) encykelimplementering. (X) flercykelimplementering. (2) pipelining.
d. Med hög rumslokalitet i minnessystem avses att (1) om en adress refereras så är det stor sannolikhet att den snart refereras igen. (X) om en adress refereras så är det stor sannolikhet att en närliggande adress snart refereras. (2) få adresser refereras.
e. Asynkron kommunikation används ofta för (1) processor-minnesbussar. (X) I/O-bussar. (2) bakplansbussar.
f. *Write-invalidate* och *write-update* är exempel på (1) utbytesstrategier för virtuella sidor.(X) tekniker för att lösa pipelinekonflikter. (2) metoder för att uppnå cache-koherens i parallelldatorsystem.
g. Booths algoritm är en metod för (1) flyttalsmultiplikation. (X) flyttalsdivision. (2) heltalsmultiplikation.

h. Att använda *stalling* som sätt att lösa pipelinekonflikter lämpar sig bäst för (1) datakonflikter. (X) styrkonflikter. (2) resurskonflikter.
i. Den typ av cache som kräver den mest avancerade adresseringslogiken är en (1) direktmappad cache. (X) set-associativ cache. (2) fullt associativ cache.
j. Snooping är (1) en teknik för att upprätthålla minneskoherens i multiprocessorsystem. (X) ett sätt att kommunicera med I/O-enheter. (2) en metod för snabb uppdatering av sidtabeller.
k. Det största talet som kan representeras med dubbel precision (*IEEE 754 double precision*) är c:a $2.0*10^{308}$. Vilket är det största tal som kan representeras med enkel precision (*IEEE 754 single precision*)? C:a (1) $2.0*10^{38}$. (X) $2.0*10^{154}$. (1) $1.0*10^{308}$.
l. Vilken typ av buss är PCI en standard för? (1) Processor-minnesbuss. (X) Bakplansbuss. (2) I/O-buss.

TREVLIG SOMMAR!

Bilaga 1: MIPS maskininstruktioner

Common MIPS instructions.

Notes: *op*, *funct*, *rd*, *rs*, *rt*, *imm*, *address*, *shamt* refer to fields in the instruction format. The program counter PC is assumed to point to the next instruction (usually 4 + the address of the current instruction). M is the byte-addressed main memory.

Assembly instruction	Instr. format	*op* *op*/*funct*	Meaning	Comments
`add $rd, $rs, $rt`	R	0/32	`$rd = $rs + $rt`	Add contents of two registers
`sub $rd, $rs, $rt`	R	0/34	`$rd = $rs - $rt`	Subtract contents of two registers
`addi $rt, $rs, imm`	I	8	`$rt = $rs + imm`	Add signed constant
`addu $rd, $rs, $rt`	R	0/33	`$rd = $rs + $rt`	Unsigned, no overflow
`subu $rd, $rs, $rt`	R	0/35	`$rd = $rs - $rt`	Unsigned, no overflow
`addiu $rt, $rs, imm`	I	9	`$rt = $rs + imm`	Unsigned, no overflow
`mfc0 $rt, $rd`	R	16	`$rt = $rd`	*rd* = coprocessor register (e.g. epc, cause, status)
`mult $rs, $rt`	R	0/24	`Hi, Lo = $rs * $rt`	64 bit signed product in Hi and Lo
`multu $rs, $rt`	R	0/25	`Hi, Lo = $rs * $rt`	64 bit unsigned product in Hi and Lo
`div $rs, $rt`	R	0/26	`Lo = $rs / $rt, Hi = $rs mod $rt`	
`divu $rs, $rt`	R	0/27	`Lo = $rs / $rt, Hi = $rs mod $rt` (unsigned)	
`mfhi $rd`	R	0/16	`$rd = Hi`	Get value of Hi
`mflo $rd`	R	0/18	`$rd = Lo`	Get value of Lo
`and $rd, $rs, $rt`	R	0/36	`$rd = $rs & $rt`	Logical AND
`or $rd, $rs, $rt`	R	0/37	`$rd = $rs | $rt`	Logical OR
`andi $rt, $rs, imm`	I	12	`$rt = $rs & imm`	Logical AND, unsigned constant
`ori $rt, $rs, imm`	I	13	`$rt = $rs | imm`	Logical OR, unsigned constant
`sll $rd, $rs, shamt`	R	0/0	`$rd = $rs << shamt`	Shift left logical (shift in zeros)
`srl $rd, $rs, shamt`	R	0/2	`$rd = $rs >> shamt`	Shift right logical (shift in zeros)
`lw $rt, imm($rs)`	I	35	`$rt = M[$rs + imm]`	Load word from memory
`sw $rt, imm($rs)`	I	43	`M[$rs + imm] = $rt`	Store word in memory
`lbu $rt, imm($rs)`	I	37	`$rt = M[$rs + imm]`	Load a single byte, set bits 8-31 of $*rt* to zero
`sb $rt, imm($rs)`	I	41	`M[$rs + imm] = $rt`	Store byte (bits 0-7 of $*rt*) in memory
`lui $rt, imm`	I	15	`$rt = imm * 216`	Load constant in bits 16-31 of register $*rt*
`beq $rs, $rt, imm`	I	4	`if($rs==$rt) PC = PC + imm` (PC always points to next instruction)	
`bne $rs, $rt, imm`	I	5	`if($rs!=$rt) PC = PC + imm` (PC always points to next instruction)	
`slt $rd, $rs, $rt`	R	0/42	`if($rs<$rt) $rd = 1; else $rd = 0`	
`slti $rt, $rs, imm`	I	10	`if($rs<imm) $rt = 1; else $rt = 0`	
`sltu $rd, $rs, $rt`	R	0/43	`if($rs<$rt) $rd = 1; else $rd = 0` (unsigned numbers)	
`sltiu $rt, $rs, imm`	I	11	`if($rs<imm) $rt = 1; else $rt = 0` (unsigned numbers)	
`j destination`	J	2	`PC = address*4`	Jump to *destination*, *address* = *destination*/4
`jal destination`	J	3	`$ra = PC; PC = address*4` (Jump and link, *address* = *destination*/4)	
`jr $rs`	R	0/8	`PC = $rs`	Jump to address stored in register $*rs*

MIPS registers

Name	Number	Usage
`$zero`	0	constant 0
`$at`	1	reserved for assembler
`$v0 - $v1`	2-3	expression evaluation and function results
`$a0 - $a3`	4-7	arguments
`$t0 - $t7`	8-15	temporary, saved by caller
`$s0 - $s7`	16-23	temporary, saved by called function
`$t8 - $t9`	24-25	temporary, saved by caller
`$k0 - $k1`	26-27	reserved for kernel (OS)
`$gp`	28	points to middle of a 64K block in the data segment
`$sp`	29	stack pointer (top of stack)
`$fp`	30	frame pointer (beginning of current frame)
`$ra`	31	return address
`Hi, Lo`	-	store partial result of mult and div operations
`PC`	-	contains the address of the next instruction to be fetched (this is not a real MIPS register, and is only used to define instructions)
`status`	-	register 12 in coprocessor 0, stores interrupt mask and enable bits
`cause`	-	register 13 in coprocessor 0, stores exception type and pending interrupt bits
`epc`	-	register 14 in coprocessor 0, stores address of instruction causing exception

MIPS Instruction formats

Format	Bits 31-26	Bits 25-21	Bits 20-16	Bits 15-11	Bits 10-6	Bits 5-0
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	imm		
J	op	address				

MIPS Assembler syntax

```
                            # This is a comment
        .data               # Store following data in the data
                            # segment
items:                      # This is a label connected to the
                            # next address in the current segment
        .word 1, 2          # Stores values 1 and 2 in next two
                            # words
hello:  .asciiz “Hello”     # Stores null-terminated string in
                            # memory
        .text               # Store following instructions in
                            # the text segment
main:   lw $t0, items($zero)  # Instruction that uses a label to
                            # address data
```