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

Datorsystemteknik för D/E

24/8 2002

Tentamensdatum: Lördag 24/8 2002 kl. 8.45 i sal V

Examinatorer: Peter Folkesson och Jonas Vasell

Institution: Datorteknik

Förfrågningar: Peter Folkesson (ankn. 1676)

Lösningar: anslås måndag 26/8 på institutionens anslagstavla utanför laboratoriet och kursens hemsida på Internet

Resultat: anslås senast tisdag 10/9 på institutionens anslagstavla utanför laboratoriet och kursens hemsida på Internet

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

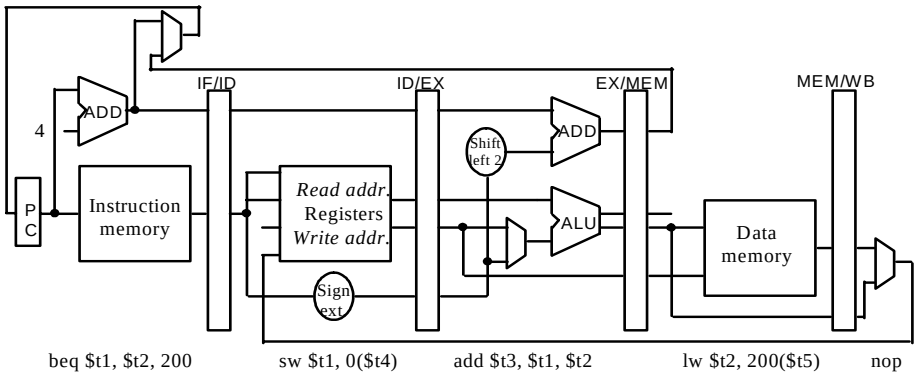
Allmänt: För full poäng på en uppgift krävs både ett korrekt svar och en motivering. En bra motivering är minst lika viktig som ett korrekt svar. Redovisa noggrannt alla gjorda antaganden utöver de som anges i uppgiftstexten. Skriv tydligt och använd gärna figurer. Maximal poäng på varje deluppgift anges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

Lycka till!

Uppgifter (1-4):

1.

a. Vilken typ av pipelinekonflikt uppstår i nedanstående figur då den givna instruktionsföljden exekveras? (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 1 GHz 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     $ra
        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. Följande deluppgifter gäller hantering av avbrott (exceptions) i allmänhet och för MIPS-processorn i synnerhet.

   a. Ange fyra olika typer av avbrott eller exceptions, och ange under vilka omständigheter de genereras. (4 p)

   b. Beskriv vad som händer i MIPS vid ett avbrott. (4 p)

   c. Beskriv vad en avbrottshanteringsrutin för MIPS måste göra allra först innan de avbrottsspecifika åtgärderna kan inledas. Ange speciellt hur situationer hanteras då flera avbrott uppstår samtidigt eller i tät följd. (4 p)

3. Ett visst datorsystem använder sig av ordstorleken 16 bitar och är byggt kring en systembuss till vilken är kopplad en processor med en separat instruktions- och datacache, primärminne i form av DRAM omfattande maximalt 1M ord, och ett eller flera DMA-gränssnitt till I/O-bussar. Datacachen är fyrvägs associativt (*4-way set-associative*) och rymmer 4K ord och 64 ord per block. Som uppdateringsstrategi tillämpas *write-through* och utbytesalgoritmen är LRU med 2-bitars tidsstämpel för varje block. Systembussen är synkron med multiplexad överföring av adress eller data varje busscykel med en frekvens av 100 MHz. Primärminnet stöder läsning och skrivning av 1-8 ord åt gången. Efter att en blockadress givits till minnet tar det ytterligare 1 busscykel för varje ytterligare ord att läsas ut eller skrivas. Vid DMA överförs alltid åtta ord åt gången på systembussen. Varje DMA-gränssnitt kan hantera två överföringar samtidigt, men endast en DMA kan initieras åt gången och det tar 200 μs att initiera en ny DMA. I/O-bussarna är synkrona och överför ett ord i taget med en databandbredd på 50 MB/s. Till I/O-bussarna kopplas skivminnen med 7 ms genomsnittlig sök- och rotationstid, och en överföringsbandbredd på 10 MB/s. Processorn har klockfrekvensen 400 MHz och CPI=1,0. 5% av instruktionerna skriver ett ord till minnet, och 25% av instruktionerna läser ett ord från minnet. Det får antas att instruktionscachen har en träffsannolikhet på 100%. Datacachen har en träffsannolikhet på 99.9% för läsningar, och för skrivningar tillämpas write-through av varje enskilt ord. Båda cacheminnena använder block om 64 ord.

   a. Hur många bitar måste datacachen kunna lagra totalt? (4 p)

   b. Antag att datacachen är outnyttjad från början och att processorn läser 4352 ord i ordning från adresserna 0, 1, 2, ..., 4351. Därefter upprepas denna access nio gånger. Vilken prestandaförbättring kan man förvänta sig av att använda datacacheminnet jämfört med ett system utan datacache om inte I/O bussarna används? (4 p)

   c. Vad blir svaren på ovanstående frågor (a-b) om en direktavbildad (*direct mapped*) datacache används i stället? (4 p)

   d. Vilken är den maximala databandbredden som systembussen kan klara? (3 p)

   e. Vad är databandbredden på systembussen för överföringar till och från cacheminnena? (3 p)

f. Vad är bandbreddskravet på systembussen för DMA från en I/O-buss om all DMA gäller 128 KB stora block till eller från skivminnen och två DMA-overföringar ständigt är igång? (3 p)

g. Hur många I/O-bussar skulle systembussen maximalt kunna belastas med under samma förutsättningar som i föregående deluppgift om hänsyn tas till trafiken till och från processorn? (3 p)

4. 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. En superskalär processor (1) kan starta exekvering av flera instruktioner samtidigt. (X) har en extra lång pipeline. (2) har en speciellt kraftfull ALU.
   b. Mikroprogrammering är en teknik för att (1) styra I/O kretsar. (X) specificera styrsignalbeteendet i en CPU. (2) optimera assemblerkod.
   c. *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.
   d. Minneskoherens innebär (1) att alla tillgängliga kopior av en del av minnet alltid är lika. (X) att det bara får finnas en kopia av varje del av minnet. (2) att minnet är skrivskyddat.
   e. Ett 4-vägs partiellt associativt cache med datalagringskapaciteten 32 KB och blockstorlek 16 B innehåller (1) 512 block. (X) 1024 block. (2) 2048 block.
   f. Om man vill använda sig av bussar med flera bus masters bör man främst tänka på att använda (1) multiplexing. (X) arbitrering. (2) handskakning.
   g. För cacheminnen betyder direktavbildat (*direct mapped*) att (1) varje cache-block är associerat med ett primärminnesblock. (X) varje block kan lagras var som helst i cacheminnet. (2) varje block kan lagras på exakt ett ställe i cacheminnet.
   h. SCSI är en vanlig standard för (1) processor-minnesbussar. (X) bakplansbussar. (2) I/O-bussar.
   i. Med *Big Endian* avses vilken ordning (1) de olika instruktionerna i en superskalär pipeline måste exekvera. (X) delarna av ord med större ordlängd än minnet måste lagras. (2) delarna av ord med större ordlängd än ALUn måste bearbetas vid aritmetiska operationer.
   j. MIPS-arkitekturen tillåter adressering av (1) 2^{32} ord. (X) 2^{30} ord. (2) 2^{30} byte.
   k. Flynns klassificering gäller (1) nätverkstopologier. (X) processortyper. (2) typer av multiprocessorsystem.
   l. Daisy-chain är benämning på (1) en metod för bussarbitrering. (X) en metod för seriekoppling av minneskretsar. (2) en teknik för utrullning av program-snurror under kompilering.

SLUT

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 * 2^16`	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
```