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

Datorsystemteknik för D/E

25/5 2002

Tentamensdatum: Lördag 25/5 2002 kl. 14.15 i sal V

Examinatorer: Peter Folkesson och Jonas Vasell

Institution: Datorteknik

Förfrågningar: Peter Folkesson (ankn. 1676)

Lösningar: anslås måndag 27/5 på institutionens anslagstavla utanför laboratoriet

Resultat: anslås senast tisdag 11/6 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 (1-5):

1.

a. Ange instruktionssekvensen för det minsta antal MIPS-instruktioner som krävs för att ladda flyttalskonstanten -769×2^{-41} i register 5. Konstanten ska representeras i flyttalsformatet IEEE 754 med enkel precision (32 bitar). Exponenten i detta flyttalsformat är 8 bitar med bias 127. Konstanten får ej lagras i dataminnet. Använd endast de MIPS-instruktioner som anges i bilaga 1. (4 p)

b. Om IEEE-representationen av flyttalskonstanten i uppg. a tolkas som ett heltal i tvåkomplementrepresentation, vilket heltal representerar ordet? Ge svaret på formen heltal $\times 2^{\text{exponent}}$. (2 p)

c. Om IEEE-representationen av flyttalskonstanten i uppg. a tolkas som en MIPS-instruktion, vilken instruktion representerar ordet? (2 p)

d. Visa hur multiplikationen 9×14 utförs med Booths algoritm. (4 p)

2. Ett datorsystem har ett primärminne med maximalt 1M 16-bitars ord. Till primärminnet är kopplad en processor med ett fyrvägs associativt (*4-way set-associative*) cacheminne och *write-through* som uppdateringsstrategi. Cacheminnet rymmer 4K ord och 64 ord per block. Som utbytesalgoritm används LRU med 2-bitars tidsstämpel för varje block. Klockfrekvensen för processorn är 400 MHz. Processor-primärminnes bussen är synkron med multiplexad överföring av adress eller data varje busscykel med en frekvens av 100 MHz. Primärminnet producerar 2 16-bitars ord åt gången, och stöder åtkomst av 2 - 8 ord. Efter att en blockadress givits till minnet tar det 50 ns tills de första två orden är tillgängliga, och sedan tar det ytterligare 1 busscykel för varje ytterligare par av ord att läsas ut.

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

b. Antag att cachen 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 cacheminnet jämfört med ett system utan cacheminne men med möjlighet till blockvis access av 2 ord åt gången? (4 p)

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

3. Många inbyggda datorsystem är tidskritiska vilket gör det mycket viktigt att kunna analysera deras realtidsegenskaper. En sådan analys är att ta reda på den längsta möjliga exekveringstiden för ett givet avsnitt kod (t.ex. en subrutin), så kallad WCET (Worst Case Execution Time). Ofta är det dock omöjligt att exakt förutsäga WCET eftersom exekveringstiden kan bero på faktorer man inte känner till, och man strävar då att hitta en minsta säkra gräns för WCET, d.v.s. en uppskattning av exekveringstiden som är så liten som möjligt men ändå inte underskrider WCET.

   Din uppgift är att ta fram en sådan uppskattning av WCET för MIPS-koden nedan räknat i klockcykler från det att den första instruktionen hämtas i cykel 1 tills instruktionen efter den sista hämtas (jr) i cykel N+1, dvs du ska beräkna en uppskattning av N som är garanterat större än det verkliga värdet på N. Utgå från en MIPS-pipeline med stegen Instruction Fetch, Instruction Decode, Execute, Memory Access, och Write Back med funktionalitet som i kursboken. Data forwarding tillämpas i alla fall det är möjligt. Assume-not-taken används som hoppstrategi och alla hopp utförs i Instruction Decode-steget. Du får också anta att hela kodsekvensen slutförs utan avbrott.

```
      addi    $4, $0, 64
      addi    $6, $0, 0
L1:   lw      $8, 12($4)
      beq     $8, $0, L2
      sll     $8, $8, 2
      sw      $8, 12($4)
      j       L3
L2:   addi    $6, $6, 1
L3:   addi    $4, $4, -4
      bne     $4, $0, L1
      jr      $31
```

   Alla instruktions- och datareferenser går genom I-cache respektive D-cache. Varje cacheminne är 16 KB stort, tvåvägs associativt (*two-way set-associative*), och har 16 byte stora block. Cacheminnena är kopplade till primärminnet via en synkron buss med 32 bitars databredd, multiplexad data- och adressöverföring, och halva processorns klockfrekvens. Ett cacheminne kan reservera bussen för en hel blocköverföring. Att reservera bussen tar 1-2 busscykler beroende på kollisioner med annan busstrafik. Varje blockläsning inleds med att adressen skickas till primärminnet. Åtkomst i primärminnet av första ordet i ett block tar 2 busscykler från det att adressen skickats tills ordet kan läsas ut, och åtkomst av påföljande ord tar en busscykel vardera. Innehållet i cacheminnena och primärminnet vid början av exekveringen av den aktuella MIPS-koden är okänt. (12 p)

4. Ett datorsystem tänk att arbeta som video-server är byggt kring en systembuss (bakplansbuss) till vilken är kopplad processor, primärminne i form av DRAM, och 20 st DMA-gränssnitt kopplade till I/O-bussar till vilka det är möjligt att ansluta maximalt ett skivminne vardera. Systembussen är synkron med multiplexad överföring av ett ord (64 bitar) data eller adress varje busscykel med en frekvens av 200 MHz. Varje överföring till eller från minnet inleds med en cykel då adressen läggs ut på bussen. I de påföljande cyklerna sker sedan överföring av data ett ord i taget. För varje adress kan antingen ett eller fyra ord överföras. Vid DMA överförs alltid fyra ord åt gången på systembussen. Systembussen är ledig för I/O 40% av tiden. För skivminnena gäller att de består av 4 dubbelsidiga skivor med ett läs/skriv-huvud per yta. Antalet cylindrar är 30000 och det går 500 sektorer per spår (*track*). Varje sektor rymmer 512 bytes och skivorna roterar med 7200 varv per minut. Söktiden (*seek time*) är 10 ms.

   a. Vad blir systemets totala skivminnes-lagringskapacitet om man vill vara säker på att alla skivminnen man ansluter till I/O-bussarna ska kunna accessas med den maximala överföringsbandbredden (*peak bandwidth*) samtidigt? (4 p)

   b. Hur stor andel av den totala accesstiden ägnas i medeltal åt att söka upp rätt block om 8 KB stora block accessas på slumpmässiga ställen på skivminnet? (3 p)

   c. Vad blir svaret i uppg. b om man antar att 99% av alla accesser sker till block som följer omedelbart efter det block som accessades gången innan? (3 p)

   d. Ange en typisk skillnad och en typisk likhet mellan en systembuss (bakplansbuss) och en I/O-buss. (2 p)

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**. Ej angivet svar ger noll poäng. Minsta poäng på hela uppgiften är noll. (12 p)

   a. Antalet instruktioner i ett program beror främst på (1) hur minnet är organiserat. (X) vilken ISA som används. (2) vilken hårdvaruteknologi som används.
   b. En modern ISA karakteriseras av tillgång till få (1) minnesadresseringsmetoder. (X) instruktionsformat. (2) generella register.
   c. RISC (1) är anpassat för kompilatorer och VLSI teknik. (X) är anpassat till högnivåspråk och diskret implementering. (2) gynnar ”svåra fallet” och liten minnesåtgång.
   d. I en pipelinad processor (1) tar en instruktion en klockcykel att genomföra. (X) är genomströmningshastigheten låg. (2) krävs likformig exekveringscykel.
   e. Datakonflikter hanteras oftast genom att (1) stanna delar av pipelinen tills konflikten löses upp. (X) gissa resultatet innan det beräknas. (2) göra resultatet tillgängligt innan det lagras.

f. Styrkonflikter hanteras oftast genom att (1) stanna delar av pipelinen tills konflikten löses upp. (X) gissa resultatet innan det beräknas. (2) göra resultatet tillgängligt innan det lagras.
g. Resurskonflikter hanteras oftast genom att (1) stanna delar av pipelinen tills konflikten löses upp. (X) gissa resultatet innan det beräknas. (2) göra resultatet tillgängligt innan det lagras.
h. Valet av utbytesalgoritm spelar störst roll för en (1) write-back cache. (X) fullt associativ cache. (2) direktavbildad cache.
i. Handskakningsprotokoll krävs i allmänhet för (1) synkrona bussar. (X) asynkrona bussar. (2) både asynkrona och synkrona bussar.
j. Om man vill använda sig av bussar med flera bus masters bör man främst använda (1) multiplexing. (X) arbitrering. (2) handskakning.
k. En TLB implementeras företrädesvis i form av ett cacheminne som är (1) fullt associativ. (X) direktavbildad. (2) fyrvägs associativt (*4-way set-associative*).
l. SPEC är (1) en samling syntetiska benchmarkprogram. (X) en standardiserad uppsättning riktiga program. (2) ett antal benchmarkprogram från ett speciellt tillämpningsområde.

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
```