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

Datorsystemteknik för D/E

25/8 2001

Tentamensdatum: Lördag 25/8 2001 kl. 8.45 i sal V

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

Institution: Datorteknik

Förfrågningar: Peter Folkesson (ankn. 1676)

Lösningar: anslås måndag 27/8 på institutionens anslagstavla utanför laboratoriet

Resultat: anslås senast tisdag 11/9 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. I en minnesposition i en dator baserad på MIPS-processorn hittas vid ett tillfälle följande ord:

$$10101100000100000000000000000000_2$$

Om ordet tolkas som ett flyttal på IEEE 754-format (8-bitars exponent-representation och bias 127), vilket flyttal representerar ordet? Ge svaret på formen heltal $\times 2^{\text{exponent}}$. (2 p)

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

c. Om ordet i uppg. a tolkas som en MIPS-instruktion, vilken instruktion representerar ordet? En sammanställning över MIPS maskininstruktioner finns i bilaga 1. (2 p)

d. Nämn två skäl till att man väljer att stödja flyttalsaritmetik i en CPU med hjälp av separata flyttalsregister istället för att använda sig av de vanliga registren? (2 p)

e. Visa hur multiplikationen 11*12 utförs med Booths algoritm. (4 p)

2. En styrdator i ett industriellt processövervakningssystem har visat sig få alltför dåliga prestanda i samband med en tidskritisk funktion där en stor mängd data regelbundet ska skickas ut över ett nätverkskort via datorns systembuss. Systembussen är 32 bitar bred med multiplexad överföring av adresser och data, har centraliserad parallell arbitrering, och är synkron med 67 MHz klockfrekvens. Varje överföring av 32 bitar över bussen tar en klockcykel. Till systembussen är förutom nätverkskortet även systemets primärminne och processor kopplade.

Processorn är av den typ som visas i bilaga 2 med 200 MHz klockfrekvens, och innehåller separata instruktions- och datacache som använder systembussen vid cachemissar. Vid en miss i endera cacheminnet stoppas processorn tills hela det saknade blocket hämtats in från primärminnet. En träff i cacheminnet avslutas på en cykel. Instruktionscacheminnet i processorn har konfigurerbar blockstorlek som kan ställas in till ett antal ord som är en jämn tvåpotens mellan 2 och 16 ord.

Primärminnet är av DRAM-typ och producerar block om upp till 4 par av 32-bitars ord för varje given blockadress. Antalet ordpar kan ställas in för varje åtkomst. Efter att en blockadress givits till minnet tar det 75 ns tills de första två orden är till gängliga, och sedan tar det ytterligare 25 ns för varje ytterligare därpå följande par av ord att läsas ut. Före varje blockläsning reserveras bussen för hela läsningen, och släpps fri för andra överföringar först när hela blockläsningen är klar.

Din uppgift är att försöka hitta en lämplig blockstorlek för instruktionscacheminnet som håller nere belastningen på systembussen så mycket som möjligt, men samtidigt inte innebär en oacceptabel försämring av processorns prestanda. Missannolikheten för instruktionscacheminnet har vid mätningar visat sig vara 4% vid 2 ord/ block, 2% vid 4 ord/block, 1% vid 8 ord/block, och 0,8% vid 16 ord/block. CPI utan inverkan av instruktionsmissar har uppmätts till 1,6. Du kan också anta att förutom för instruktionscachemissar så används bussen 40% av tiden för överföringar av block om 4 ord mellan primärminnet och datacacheminnet eller nätverkskortet, och att vid arbitrering av bussen ges högst prioritet åt instruktionscacheminnet. Arbitreringen sker parallellt med blocköverföringarna så att direkt en överföring är klar så kan nästa påbörjas.

a. Vilken blockstorlek för instruktionscacheminnet ger minst antal cykler då processorn måste stoppas på grund av instruktionshämtningsmissar? (4 p)

b. Vilken blockstorlek för instruktionscacheminnet ger minst andel av tiden som bussen är reserverad för läsningar av instruktioner? (4 p)

c. Hur mycket vinner man vid blockstorleken 4 ord i processorprestanda respektive bussbelastning om bussens klockfrekvens skulle ökas till 100 MHz? (4 p)

3.

a. Ditt företag hyr sin maskinpark på 50 datorer från en annan firma. Processorerna i dessa datorer klockas med 500 MHz. I hyresavtalet ingår utbyte av maskinparken vartannat år. Nu har man fått ett erbjudande om att byta ut maskinparken mot datorer med antingen 1 GHz processorer eller 600 MHz processorer. Datorerna med 1 GHz processorer är identiska med de nuvarande datorerna (förutom klockfrekvensen), medan 600 MHz datorerna har en nyutvecklad CPU med en utökad instruktionsuppsättning som drastiskt effektiviserar vissa beräkningar, t ex multimediatillämpningar.

   Programvaran ni använder utför multimediaberäkningar i 40% av alla instruktionerna som exekveras. Dessa beräkningar har en genomsnittlig CPI på 3,0 för 500 MHz och 1 GHz processorerna. CPI för övriga delar av programvaran är 1,0.

   För 600 MHz processorn kan en specialoptimerad version av programvaran användas där andelen multimediaberäkningar som exekveras sjunkit till 10% av det totala antalet instruktioner som exekveras. Dessa beräkningar har en genomsnittlig CPI på 0,5. Genomsnittlig CPI för övriga delar av programvaran är oförändrat 1,0. *De övriga delarna av den specialoptimerade programvaran är i själva verket identiska med den icke-optimerade versionen.*

   Vilket av utbytesalternativen ger bästa prestanda och hur många gånger snabbare går det jämfört med det gamla alternativet? (8 p)

b. För att mäta prestanda hos datorsystem kan man använda SPEC. Vad är SPEC? Vad är för- och nackdelarna med att använda SPEC? (4 p)

4. Betrakta följande MIPS-program:

```
        sw   $1, 0($2)
        lw   $3, 0($1)
        slti $4, $3, 100
        beq  $4, $0, L1
        :
L1:     add  $3, $3, $2
        :
```

   a. Antag att programmet exekveras med den pipeline som visas i bilaga 2 och att alla konflikter hanteras i hårdvaran genom pipeline stalling. Hur många klockcykler tar det från att den första instruktionen exekveras tills den sista instruktionen lämnat WB-steget om hoppet tas? (3 p)

   b. Hur mycket kan antalet klockcykler minska jämfört med föregående deluppgift om data forwarding (bypassing) införs för att hantera datakonflikter där så är möjligt? (3 p)

   c. Nämn två olika metoder för att lösa hoppkonflikten ovan med hjälp av hårdvara. Beskriv hur varje metod fungerar, samt redogör för metodernas för- och nackdelar. (6 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**. Inget svar ger noll poäng. Minsta poäng på hela uppgiften är noll. (12 p)

   a. Om kravet är att programkoden i ett datorsystem måste vara extremt minnessnål, bör man välja en processor med (1) encykelimplementering. (X) flercykelimplementering. (2) pipelining.
   b. Asynkron kommunikation används ofta för (1) processor-minnesbussar. (X) I/O-bussar. (2) bakplansbussar.
   c. *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.
   d. Vilken typ av buss är SCSI en standard för? (1) Processor-minnesbuss. (X) Bakplansbuss. (2) I/O-buss.
   e. För cache-minnen betyder direktavbildat (*direct mapped*) att (1) varje cache-block är associerat med ett primärminnesblock. (X) varje block kan lagras var som helst i cache-minnet. (2) varje block kan lagras på exakt ett ställe i cache-minnet.
   f. 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.
   g. MIPS-arkitekturen tillåter adressering av (1) 2^{32} ord. (X) 2^{30} ord. (2) 2^{30} byte.

h. Mikroprogrammering är en teknik för att (1) styra I/O kretsar. (X) specificera styrsignalbeteendet i en CPU. (2) generera en så minimal assemblerkod som möjligt.
i. Flynns klassificering gäller (1) nätverkstopologier. (X) processortyper. (2) typer av multiprocessorsystem.
j. Daisy-chain är benämning på (1) en metod för bussarbitrering. (X) en metod för seriekoppling av minneskretsar. (2) en teknik för utrullning av programsnurror under kompilering.
k. Dynamisk pipelineschemaläggning (1) är en konstruktionsmetod för pipelines. (X) innebär att maskininstruktioner kan exekveras i en annan ordning än vad som anges i maskinprogram. (2) innebär att en processor har multipla funktionsenheter för instruktionsexekvering.
l. 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.

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 * 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
```

Bilaga 2: MIPS pipeline