Tentamen i kursen EDA330

Datorsystemteknik D

27/5 2000

Tentamensdatum: Lördag 27/5 2000 kl. 8.45 i sal VV

Examinator: Jonas Vasell

Institution: Datorteknik

Förfrågningar: Thomas Lundqvist (ankn. 1165)

Lösningar: Anslås måndag 29/5 på institutionens anslagstavla utanför laboratoriet

Resultat: Anslås senast fredag 9/6 på institutionens anslagstavla utanför laboratoriet

Rättningsgranskning: Fredag 9/6 klockan 13.00-13.30 hos Thomas Lundqvist, rum 6317, institutionen för datorteknik..

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

Tillåtna hjälpmedel: inga

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.

Välgångsönskning: Lycka till!

Uppgifter (1-5):

1.

   a. Hur bred är en tag i ett 32 KB stort cacheminne med associativitet 2 och 32 B stora block där uppslagningar sker med 32 bitars byte-adresser? (2 p)

   b. Vilken information lagras i en PTE (Page Table Entry) i samband med virtuellt minne? (2 p)

   c. Under vilka villkor kan uppslagning i ett fysiskt adresserat cacheminne ske parallellt med översättning av virtuella adresser? Vilka krav måste ställas för att dessa villkor ska uppfyllas med det cacheminne som beskrivs i deluppgift a? (2 p)

   d. Operativsystemet i en dator vill hämta in en virtuell minnessida från ett skivminne till en viss adress i primärminnet med hjälp av DMA. Vilka signaler och data måste skickas mellan processor, primärminne, I/O- och DMA-styrenhet, och skivminne? (2 p)

2. I bilaga 1 finns en bild som visar en pipelineimplementering av MIPS. En av enheterna kallas Decode och är ansvarig för att generera styrsignaler för varje enskild instruktion som i varje steg ser till att rätt funktion utförs. För enkelhets skull är vissa signaler bitvis kombinerade i en ritad linje, men vilka signaler som ingår och vart de är kopplade framgår ändå av bilden. De styrsignaler som genereras av Decode är ALUOp, ALUSrc, RegDest, Branch, MemRead, MemWrite, MemtoReg, och RegWrite. I de fall en styrsignal är kopplad till en multiplexer, så gäller att styrsignalvärdet noll motsvarar översta multiplexeringången, styrsignalvärdet ett multiplexeringången nedanför, osv. Observera att en styrsignal alltså kan omfatta flera bitar. Ett speciellt fall är styrsignalen Branch som kan anta värdena (binärt) 00 (hoppa ej), 01 (hoppa ovillkorligt), 10 (hoppa om lika), 11 (hoppa om ej lika). ALUOp kan anta värdena (binärt) 00 (add), 01 (subtract), och 10 (function code). ALU producerar en flaggbit som anger om resultatet är noll eller ej.

   a. Ange för alla styrsignaler utom ALUOp och Branch vilka värden de kan ha och vilken funktion de har. (6 p)

   b. Det finns två speciella styrlogikblock utöver Decode, som kallas ALU ctrl och BD. Beskriv vilken funktion respektive block har. (4 p)

   c. Rita upp ett grindnät som implementerar BD. (2 p)

   d. Ange i tabellform de styrsignalsvärden som Decode ska generera för följande MIPS-instruktioner (se bilaga 2): `add`, `beq`, `lw`, och `sw`. (4 p)

3. Du har fått i uppdrag att förbättra en implementering av MIPS som ser ut som den i bilaga 1. Implementeringen som kallas I0 har följande egenskaper:
   - Klockfrekvens 200 MHz.
   - “Assume not taken” som hoppgissningsstrategi.
   - Icke-associativt instruktionscache. Åtkomsttid vid träff i instruktionscache är 4,5 ns.
   - Icke-associativt datacache. Write-back används som skrivstrategi. Åtkomsttid vid träff i datacache är 4,8 ns.

- Åtkomsttid till primärminne vid cachemiss är 55 ns. Denna tid gäller för både läsning av ett block eller skrivning av ett block. Åtkomst till primärminnet startas först i cykeln efter att en cachemiss upptäckts.

Med hjälp av diverse analysverktyg har du fått fram följande data vid körning av ett testprogram som kallas T1:

- Träffsannolikhet i instruktionscache är 99%.
- Träffsannolikhet i datacache är 90%.
- 60% av alla block som hämtas in till datacache uppdateras någon gång innan de måste kastas ut ur cacheminnet.
- Var sjätte instruktion är någon form av hoppinstruktion, villkorlig eller ovillkorlig.
- Var fjärde instruktion är en load- eller store-instruktion som alltså läser eller skriver ett ord i dataminnet.
- T1 tar 20,5 ms att köra räknat i CPU-tid för just det programmet.

Med hjälp av en simulator testar du nu två olika konstruktionsförbättringar. Den första förbättringen (I1) innebär att datacachet byts ut mot ett som är två-vägs associativt, medan alla andra egenskaper hos implementeringen bibehålls. Enligt simulatorn blir då körningstiden för T1 18,3 ms och träffsannolikheten i datacachet 95%. Den andra förbättringen (I2) är att byta hoppgissningsstrategi till “assume taken” genom att lägga till en enhet i första pipelinesteget som upptäcker hoppinstruktioner och beräknar deras destinationsadress om hoppet tas. Alla andra egenskaper hos implementeringen är samma som i I0. Enligt simulatorn blir i detta fall körningstiden för T1 19,5 ms.

Svara med utgångspunkt i ovanstående information på följande frågor:

a. Hur stor andel av alla hopp (villkorliga och ovillkorliga) i T1 tas? Vad skulle CPI-talet för T1 vara om man hade en implementering med idealt instruktionscache (dvs 100% träffsannolikhet), idealt datacache (dvs 100% träffsannolikhet), och ideal hoppgissningsstrategi (dvs 100% korrekt gissning)? (6 p)

b. Vid en implementering av det bättre datacachet enligt I1 visar det sig att åtkomsttiden vid träff ökar med 0,7 ns jämfört med den ursprungliga implementeringen. Vid implementering av assume taken-strategin enligt I2 visar det sig att man direkt efter instruktionscachet måste lägga in ett logikblock som undersöker om det är ett hopp och därigenom bestämmer nästa PC-värde, vilket tar 1 ns extra. Vilken förbättring blir bäst om man tar hänsyn till dessa implementeringsdata: Förbättrat datacache (I1), assume taken-strategin (I2), eller båda tillsammans (I3)? Vilken blir körningstiden för T1 i bästa fallet? (6 p)

4. 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 1 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 tillgängliga, 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)

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 en poäng. (12 p)

   a. Hur många block innehåller ett 4-vägs partiellt associativt cache med datalagringskapacitet 64 KB och blockstorlek 16 B? (1) 4096. (X) 1024. (2) 4.

   b. Hur mycket ungefär ökar processorprestanda på tre år? (1) 10%. (X) 50%. (2) 300%.

   c. Hur mycket ungefär ökar lagringskapaciteten för DRAM på tre år? (1) 10%. (X) 50%. (2) 300%.

   d. Hur kan det binära ordet $10101100010000000000000000000000_2$ tolkas? (1) Som flyttalet 3×2^{-40}. (X) Som MIPS-instruktionen `sw $0, 0($2)`. (2) Som heltalet 335×2^{22}.

   e. Vad gör Booth's algoritm? (1) Utbyte av sidor i virtuellt minne. (X) Flyttalsmultiplikation. (2) Heltalsmultiplikation.

   f. Vad utmärker en superskalär processor? (1) Den kan starta exekvering av mer än en instruktion samtidigt. (X) Den kan utföra aritmetik på mycket stora operander. (2) Den har en mycket lång pipeline.

   g. Vad är ett exakt avbrott? (1) Ett avbrott vars ursprung är helt känt. (X) Ett avbrott från vilket återhopp aldrig sker. (2) Ett avbrott som hanteras omedelbart.

   h. I vilket sammanhang används handskakningsprotokoll? (1) Vid överföring på synkrona bussar. (X) Vid överföring på asynkrona bussar. (2) Vid överföring mellan cache och processor.

   i. Vad är daisy-chain en benämning på? (1) En metod för hoplänkning av programmoduler. (X) En metod för seriekoppling av minneskretsar. (2) En metod för bussarbitrering.

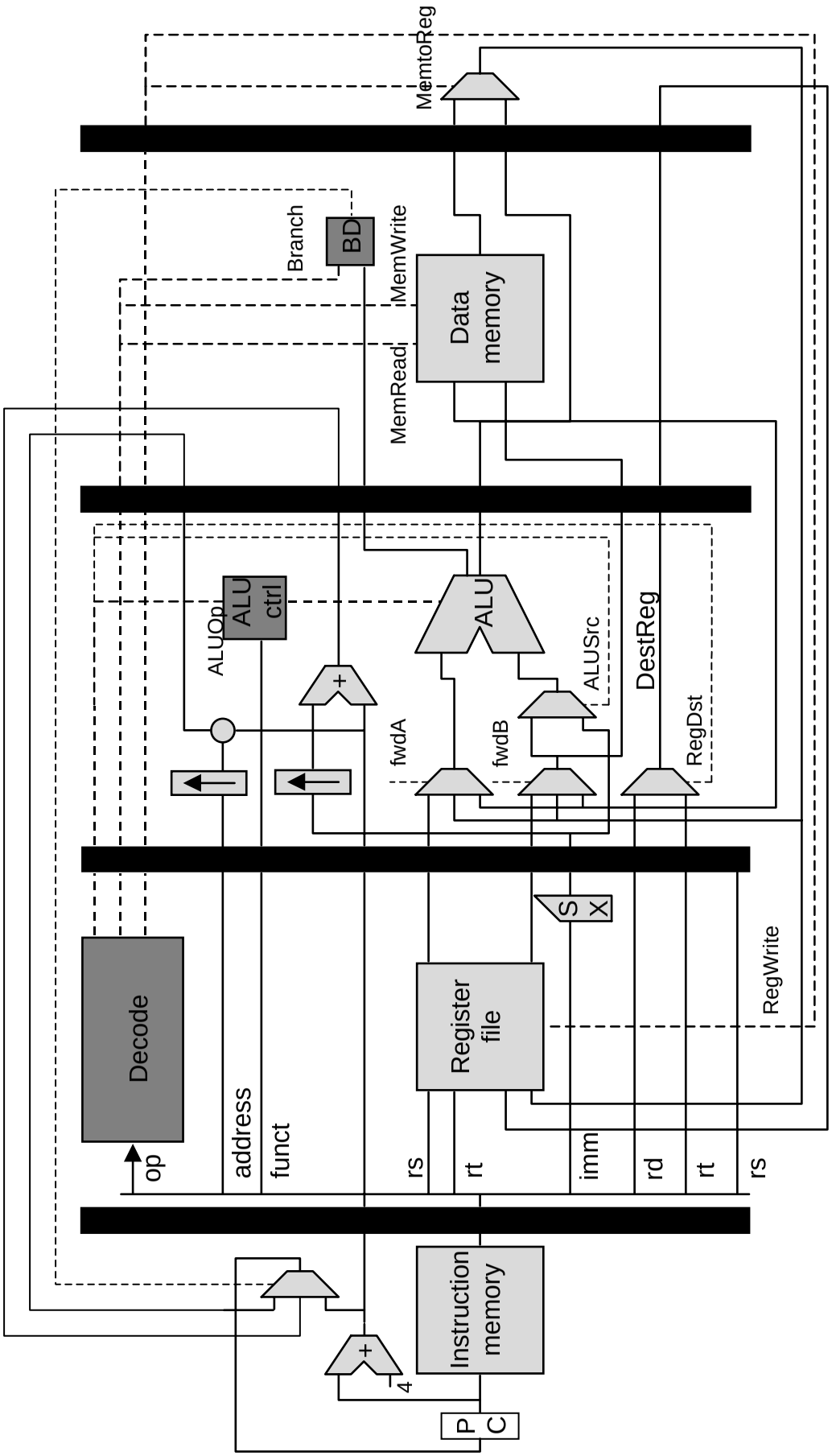
   j. Vad står MIMD för? (1) Multiple Instruction streams, Multiple Data streams. (X) Multiplexed Input Memory Device. (2) Multiple Input Memory Device.

   k. Vilken typ av buss är SCSI en standard för? (1) Processor-minnebuss. (X) Bakplansbuss. (2) I/O-buss.

   l. Vilket block byts ut enligt utbytesalgoritmen LRU? (1) Det senast använda. (X) Det som inte använts på längst tid. (2) Det som är minst troligt att det kommer att användas snart.

SLUT

Bilaga 1: MIPS pipeline



Bilaga 2: MIPS maskininstruktioner

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
```