Tentamen i kursen EDA330

Datorsystemteknik D

1/6 1999

Tentamensdatum: Tisdag1/6 1999 kl. 8.45 i sal VV

Examinator: Jonas Vasell

Institution: Datorteknik

Förfrågningar: Jonas Vasell (ankn. 1689)

Lösningar: Anslås tisdag 1/6 på institutionens anslagstavla utanför laboratoriet

Resultat: Anslås senast måndag 14/6 på institutionens anslagstavla utanför laboratoriet

Rättningsgranskning: Måndag 14/6 klockan 13.00-14.00 hos Jonas Vasell

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. Vad är det totala antalet bitar ett cacheminne med följande egenskaper måste kunna lagra? Total datalagringskapacitet är 32 KB (1 KB = 1024 B), associativiteten är 4, blockstorleken är 8 ord. Cacheminnet är fysiskt adresserat med 30 bitars byte-adresser. Skrivningsstrategi är write back, och utbytesalgoritmen bygger på en implementering av LRU där varje block i ett set tilldelas ett unikt ordningsnummer mellan 0 och n-1 där n är antalet block i ett set. (4 p)

b. Hur stort minnesutrymme räknat i bytes krävs för att lagra sidtabeller i följande datorsystem med virtuellt minne? Sidstorleken är 8 KB, och det finns maximalt 32 processer med unika virtuella adressrum. Det virtuella adressrummet för varje process begränsas till byte-adresserna $0..2^{27}-1$. Fysiskt kan upp till 128 MB (1 MB = 1024 KB) adresseras. En approximation av LRU med 8 bitars tidstämplar tillämpas. Skrivningsstrategin är write back. För varje sida finns också två skyddsbitar som anger åtkomsträttigheter. Informationen om varje sidöversättning i sidtabellen måste lagras i ett helt antal bytes. (4 p)

c. Förklara begreppet sidfel (eng. page fault) i samband med virtuellt minne, och beskriv vilka åtgärder som måste vidtas vid ett sidfel. (4 p)

2. I bilaga 1 visas en implementering av MIPS i form av en pipeline. Styrsignalerna är inritade i bilden med streckade linjer (av utrymmesskäl så är flera styrsignaler kombinerade i en linje fram till dess de används). För multiplexrar gäller alltid att översta ingången motsvarar ett styrsignalvärde 0, ingången nedanför stysignalvärde 1, osv. Namn på styrsignalerna är angivna invid respektive styrsignal i närheten av där de används. Vissa andra signaler är också namngivna; delar av instruktionsformatet (se bilaga 2), och DestReg. En signals värde i ett visst pipelinesteg (IF, ID, EX, MEM, WB) kan refereras till genom en kombination av stegets namn och signalens namn. T.ex. står EX.MemRead för värdet på MemRead-signalen för den instruktion som för tillfället befinner sig i EX-steget.

a. Två multiplexrar för data forwarding är inkluderade, och styrs av signalerna fwdA respektive fwdB. Ange logiska ekvationer för hur fwdA och fwdB ska ställas in för att på ett korrekt sätt undvika så många datakonflikter som möjligt. I de logiska ekvationerna kan booleska operatorer och värden på namngivna signaler från olika pipelinesteg ingå. Det är också tillåtet att använda jämförelseoperatorer (t.ex. =, ≠). Hänsyn ska tas till alla de situationer som kan uppstå i MIPS med den instruktionsuppsättning som definieras i bilaga 2. Det går bra att sätta upp separata ekvationer för fallen fwdA=0, fwdA=1 osv. (6 p)

b. Ange på motsvarande sätt som i föregående deluppgift en logisk ekvation som beskriver fall då en datakonflikt uppstår som inte kan lösas med data forwarding. (4 p)

c. Beskriv vilka åtgärder som måste vidtas då datakonflikter uppstår som inte kan lösas med data forwarding eller genom att ändra i programmet. Ange i första hand åtgärder som får så liten prestandainverkan som möjligt. (2 p)

3.

a. Sätt upp en ekvation som anger medelåtkomsttiden för ett datacacheminne räknat i processorklockcykler.

Antag att datacacheminnet är kopplat till primärminnet via en synkron buss med halva processorns klockfrekvens, adress/data-multiplexing (adress måste skickas först innan minnet kan börja arbeta), och ett ords bredd. Tillgång till bussen måste begäras för varje överföring av upp till ett cacheblock till eller från primärminnet. Bussen hålls alltid kvar till minnesoperationen är slutförd.

Åtkomsttiden vid träff är en processorcykel. Vid en miss påbörjas primärinnesåtkomsten först i cykeln efter att missen upptäckts. Med en åtkomst i cacheminnet eller primärminnet avses här läsning eller skrivning av ett ord. Cacheminnet använder write back som skrivningsstrategi. Det finns ingen skrivbuffer mellan cache och primärminne.

Du kan använda följande parametrar i ekvationen (det är inte givet att alla behövs):

— r, andel läsningar av alla datacacheåtkomster,
— h_1, träffsannolikhet,
— w, sannolikhet att minst en skrivning sker till ett block medan det ligger i cache,
— B, blockstorlek räknat i ord,
— A, associativitet,
— t_1, primärminnets åtkomsttid för första ordet i ett cacheblock räknat i busscykler,
— t_2, primärminnets åtkomsttid för påföljande ord i ett cacheblock räknat i busscykler,
— t_b, genomsnittlig väntetid räknat i busscykler för att få tillgång till bussen.

(4 p)

b. Gör motsvarande som i förra deluppgiften men för ett cache som använder write through som skrivningsstrategi. Antag att med denna strategi så hämtas inte block in till cache vid miss (s.k. *no write-allocate*). Träffsannolikheten blir i detta fall h_2. Varför är det troligt att $h_1 \neq h_2$? (3 p)

c. Vilka respektive för- och nackdelar har de två skrivningsstragierna i de föregående deluppgifterna? (2 p)

d. Vad är en skrivbuffer? Hur förändras ekvationerna ovan om en skrivbuffer införs? På vilket sätt påverkar en skrivbuffer hur läsningar utförs? (3 p)

4. När du kommer hem från en semesterresa till Kreta visar det sig att bilderna på Greta som du tagit med din nya digitalkamera har blivit felexponerade. Som en händig datatekniker laddar du då ner bilderna i din ganska gamla och slöa arbetsstation som har en 100 MHz MIPS R2000-processor (med en enkel rak pipeline liknande den i bilaga 1) och kör ett bildbehandlingsprogram. Det innehåller ett kommando som för det mesta korrigerar felexponeringar. Beräkningen som krävs tar dock i genomsnitt 12 s per bild trots att inga andra program är igång samtidigt. Du kan också konstatera att beräkningen inte verkar kräva någon speciell I/O-aktivitet eftersom du inte märker av någon aktivitet på hårddisken, nätverket, eller skärmen medan beräkningen utförs.

   För att se om du kan få det att gå fortare så öppnar du din dator och upptäcker att du genom ett enkelt komponentbyte kan öka processorns frekvens till 150 MHz. Ändringen påverkar bara processorns klockfrekvens, och inga andra komponenter. Processorn verkar vara ett bra exemplar med sådana marginaler att den tål denna kraftiga överklockning (fast du får nog sätta in en kraftigare kylning). När du gjort det så kör du samma program igen, men upptäcker att beräkningstiden per bild obegripligt nog bara sjunkit till 10 s.

   Eftersom du läst en kurs i digital bildbehandling bestämmer du dig då för att undersöka om prestandaproblemen har med programmet att göra, och tar dig en titt på maskinkoden för den aktuella beräkningen. Beräkningen består av ett relativt enkelt förberedande avsnitt där samtliga bildpunkter gås igenom, följt av en mer komplicerad beräkning. I den förberedande delen utförs för dina bilder cirka 120 miljoner instruktioner, men för resten av programmet kan du inte beräkna antalet instruktioner. Du hittar dock ett sätt att halvera antalet instruktioner i den förberedande delen, vilket gör att exekveringstiden med den uppsnabbade processorn sjunker till 9 s.

   a. Antag att du kan visa att din programförbättring endast kan ha försumbar inverkan på CPI-talet för hela beräkningen. Beräkna CPI-talet för beräkningen före och efter ändringen av klockfrekvens. (5 p)

   b. Vad beror förändringen av CPI-talet på? (2 p)

   c. Det visar sig att 20% av de exekverade instruktionerna utför en datareferens. Antag att träffsannolikheten i cache för instruktionsreferenser är 99% och för datareferenser 90%, samt att i genomsnitt överförs (inklusive replacement) 1,11 cacheblock mellan primärminnet och cache vid cachemiss. Uppskatta den genomsnittliga tiden för att flytta ett cacheblock mellan primärminne och cache räknat i nanosekunder. (5 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.

   a. Hur snabbt ökar CPU-prestanda? (1) Cirka 25%/år. (X) Cirka 50%/år. (2) Cirka 75%/år.

   b. SCSI är en standard för (1) processor-minnesbussar. (X) bakplansbussar. (2) I/O-bussar.

   c. Write-update är en teknik för (1) minnekoherens. (X) skrivning i cacheminnen. (2) uppdatering av dynamiska RAM.

   d. En processors klockfrekvens beror av (1) hårdvaruteknologi och processororganisation. (X) instruktionsuppsättning och processororganisation. (2) hårdvaruteknologi och program.

   e. De minsta detaljer som kan implementeras med VLSI beräknas för 2010 vara c:a (1) 0,3 μm. (X) 0,07 μm. (2) 0,003 μm.

   f. Kapaciteten för de största DRAM-kretsarna ökar på tre år med ungefär en faktor (1) 2. (X) 3. (2) 4.

   g. Hög tidslokalitet i minnessystem innebär att (1) klocksignalerna är väl distribuerade. (X) om en adress refereras så är det stor sannolikhet att en närliggande adress snart refereras. (2) om en adress refereras så är det stor sannolikhet att den snart refereras igen.

   h. För cache-minnen betyder fullt associativ 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.

   i. Skillnaden i snabbhet mellan minnen och processorer tenderar (1) att minska. (X) att vara oförändrad. (2) att öka.

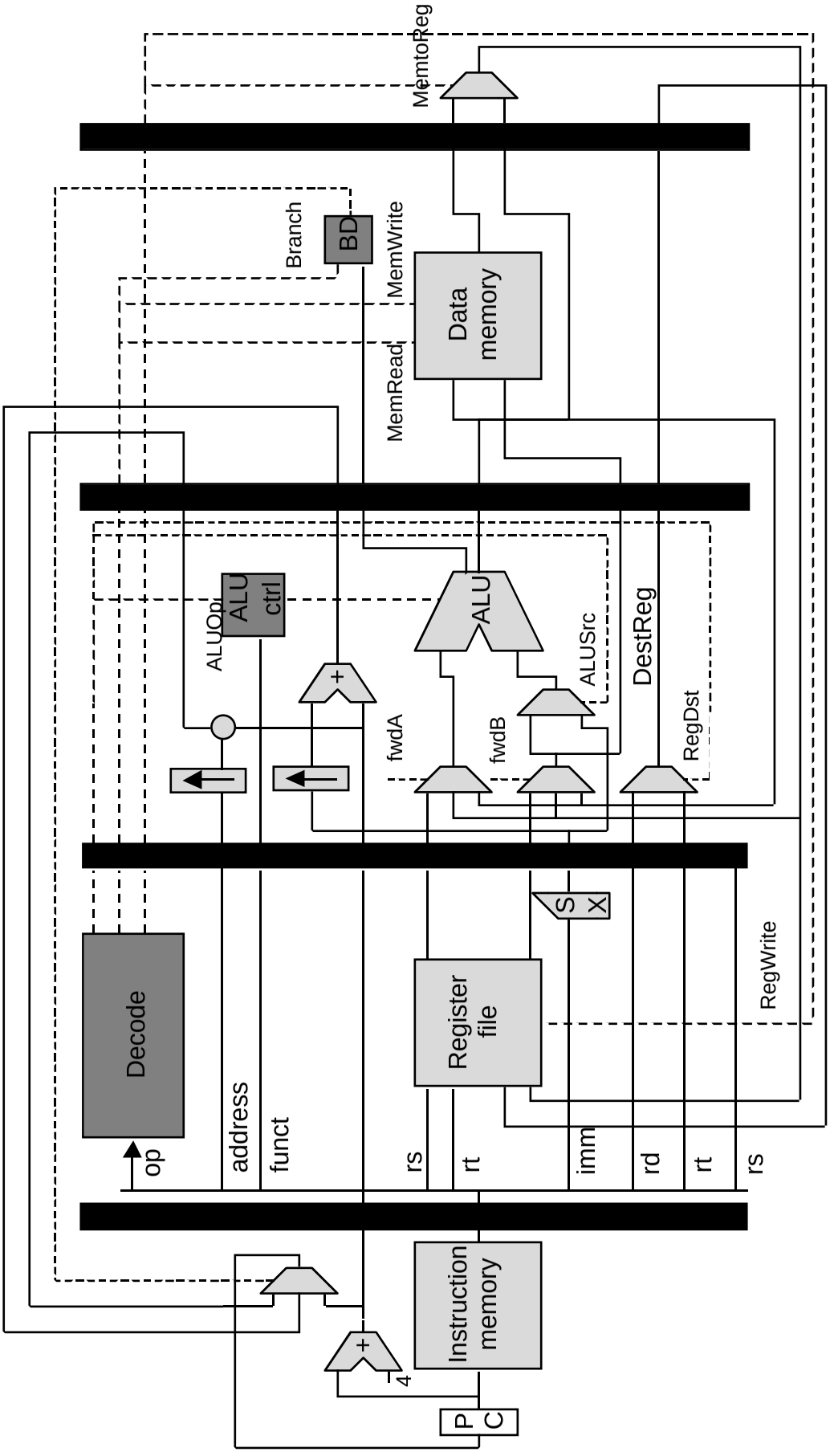
   j. Arbitrering är ett begrepp som i första hand används i samband med (1) virtuellt minne. (X) bussar. (2) registerfiler.

   k. En atomisk swap (1) är en processorinstruktion som utför en minnesläsning följt av en minnesskrivning utan att avbrytas av andra instruktioner. (X) är en processorinstruktion som byter innehåll mellan två register utan att kunna avbrytas. (2) är en processorinstruktion som utför en enda minnesläsning.

   l. Flynns klassificering gäller (1) nätverkstopologier. (X) processortyper. (2) typer av multiprocessorsystem.

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
```