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

Datorsystemteknik för D/E

17/1 2002

Tentamensdatum: Torsdag 17/1 2002 kl. 8.45 i sal M

Examinatorer: Peter Folkesson och Jonas Vasell

Institution: Datorteknik

Förfrågningar: Peter Folkesson (ankn. 1676)

Lösningar: anslås fredag 18/1 på institutionens anslagstavla utanför laboratoriet

Resultat: anslås senast fredag 1/2 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. Studera programmet nedan som exekveras på en MIPS-processor utan pipelining (en instruktion hämtas inte förrän föregående instruktion är helt avslutad):

```
200: lw $3, 0($2)
204: lw $4, 100($2)
208: slti $6, $2, 16
212: add $5, $3, $4
216: sw $5, 100($2)
220: addi $2, $2, 4
224: bne $6, $0, -28
```

   Varje programrad ovan inleds med byte-adressen till det ord där instruktionen ligger lagrad. Alla konstanter, inklusive adresser, är decimala (bas 10). Processorn ingår i ett datorsystem med ett gemensamt cacheminne för instruktioner och data. Cacheminnet är tvåvägs partiellt associativt (two way set associative) med blockstorleken 2 ord och total datalagringskapacitet 16 ord. Utbytesalgoritmen är LRU. Vid skrivmissar hämtas det aktuella blocket alltid in till cacheminnet. Registret $2 innehåller värdet noll innan programkörningen.

   a. Lista den sekvens av minnesreferenser i form av byte-adresser som programmet ger upphov till. (3 p)

   b. Beskriv hur tag, index, och block-offset beräknas för varje minnesreferens i detta fall. (3 p)

   c. Ange för var och en av de första 10 minnesreferenserna om den ger träff eller miss i cacheminnet. Ange också i tabellform innehållet i varje cacheblock efter varje referens genom att ange tag för det minnesblock som ligger där. Utgå ifrån att cacheminnet är tomt från början. Vad blir hit rate för de första 10 minnesreferenserna? (4 p)

2. Fyra olika strategier för att hantera styrkonflikter (control hazards) i instruktions-pipelines är "always stall", "assume not taken", "assume taken", och "delayed branch".

   a. Förklara kortfattat hur var och en av dessa tekniker fungerar, och vilka relativa för- och nackdelar de har. (8 p)

   b. Beräkna hur många cykler följande MIPS-program tar att exekvera med den pipeline som visas i bilaga 2 för var och en av de tre första av strategierna ovan ("always stall", "assume not taken", "assume taken"):

```
      addi $5, $1, 12
L1:   lw  $4, 0($5)
      addi $5, $5, -4
      add  $3, $3, $4
      add  $3, $3, $3
      bne  $5, $1, L1
      sw   $3, 0($5)
```

Utgå från att eventuella datakonflikter hanteras med hjälp av forwarding, och att de därför inte ger upphov till några stalls. Räkna cykeln när första instruktionen hämtas som cykel ett, och svara med numret på den cykel då den sista instruktionen utförs i sista pipeline-steget. För att referera till instruktioner efter den sista instruktionen i programmet (`sw $3, 0($5)`), använd beteckningarna sw+1, sw+2, osv. (6 p)

3. För en processor med hårdvarustöd för flyttalsberäkningar, kallad FP, gäller följande:

Ett program *P* exekveras med följande instruktionsmix:

flyttalsmultiplikationer	20%
flyttalsadditioner	10%
flyttalsdivisioner	5%
övriga operationer	65%

På FP tar:

flyttalsmultiplikationer	5 klockcykler
flyttalsadditioner	4 klockcykler
flyttalsdivisioner	16 klockcykler
övriga operationer	2 klockcykler

Klockfrekvensen för FP är 500 MHz.

Processorn finns även tillgänglig i en variant utan flyttalsenhet, kallad IP. För IP gäller att flyttalsoperationerna måste utföras enligt följande:

flyttalsmultiplikationer	30 instruktioner
flyttalsadditioner	20 instruktioner
flyttalsdivisioner	50 instruktioner

Klockfrekvensen för IP är 1 GHz och CPI_{IP}=1.

a. Man avser att vidareutveckla IP. Efter hur många månader kan man räkna med att IP kan exekvera *P* lika snabbt som nuvarande version av FP om man antar att klockfrekvensen för IP utvecklas i enlighet med Moores lag,? (6 p)

b. Ett sätt att minska exekveringstiden är att förbättra kompilatorn så att den genererar effektivare kod. Antag att man kan erhålla följande instruktionsmix för *P*:

flyttalsmultiplikationer	10%
flyttalsadditioner	6%
flyttalsdivisioner	2%
övriga operationer	82%

Den nya kompilatorn medför dock att antalet exekverade operationer för *P* ökar med 1,3 gånger. Vilken av processorerna drar mest nytta av den nya kompilatorn och hur mycket snabbare går det att exekvera *P* med den nya kompilatorn? (4p)

4.

a. Ett datorsystem är byggt kring en bakplansbuss till vilken är kopplad processor, primärminne i form av DRAM, och ett eller flera DMA-gränssnitt till I/O-bussar. Bakplansbussen är synkron med multiplexad överföring av ett ord (32 bitar) data eller adress varje busscykel med en frekvens av 100 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 bakplansbussen. Varje DMA-gränssnitt kan hantera två överföringar samtidigt, men endast en DMA kan initieras åt gången och det tar 1 ms att initiera en ny DMA. Till I/O-bussarna kopplas skivminnen med 10 ms genomsnittlig sök- och rotationstid, och en överföringsbandbredd på 10 MB/s. Hur många I/O-bussar kan bakplansbussen maximalt belastas med om man antar all DMA gäller 64 KB stora block till eller från skivminnen och att två DMA-överföringar ständigt är igång samt att bakplansbussen är ledig för I/O 56% av tiden? (6 p)

b. Räkna upp de fyra olika typer av bussarbitrering som gåtts igenom i kursen och beskriv hur var och en fungerar. (6 p)

c. Ange två olika för- eller nackdelar med asynkrona bussar jämfört med synkrona bussar. (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**. Inget svar ger noll poäng. Minsta poäng på hela uppgiften är noll. (12 p)

a. En N-kub är en nätverkstopologi där (1) varje nod har N grannoder. (X) alla meddelanden måste gå igenom N steg. (2) noderna formar en kub med N processorer i varje hörn.
b. Snooping är (1) en metod för snabb uppdatering av sidtabeller. (X) en teknik för att upprätthålla minneskoherens i multiprocessorsystem. (2) ett sätt att kommunicera med I/O-enheter.

c. I en direktavbildad (direct mapped) cache är (1) varje block lagrad var som helst i cache-minnet. (X) varje block lagrad på exakt ett ställe i cacheminnet. (2) varje cacheblock associerat med ett primärminnesblock.
d. Den av Flynns datormodeller som mest liknar vektormaskiner (vector processing computers) är (1) SISD. (X) SIMD. (2) MISD.
e. Kapaciteten för de största DRAM-kretsarna ökar på tre år ungefär med en faktor (1) 2. (X) 3. (2) 4.
f. Det minsta positiva tal som kan representeras i IEEE 754 single precision är c:a (1) $2*10^{-38}$. (X) $2*10^{-127}$. (2) $2*10^{-308}$.
g. Write-update är en teknik för (1) uppdatering av dynamiska RAM. (X) skrivning i cacheminnen. (2) minneskoherens.
h. Vilka egenskaper i en minneshierarki påverkas genom att variera blockstorlek? (1) Utnyttjande av tidslokalitet och överföringstid. (X) Utnyttjande av både tids- och rumslokalitet. (2) Utnyttjande av rumslokalitet och överföringstid.
i. Dynamisk pipeline-schedulering i kombination med hopp-prediktering (branch prediction) brukar kallas (1) superskalär exekvering. (X) spekulativ exekvering. (2) superpipelining.
j. Hur många block innehåller ett 4-vägs partiellt associativt cache med datalagringskapaciteten 32 KB och blockstorlek 16 B? (1) 512. (X) 1024. (2) 2048.
k. Det vanligaste sättet att hantera resurskonflikter i en pipeline är att (1) stanna delar av pipelinen tills konflikten löses upp. (X) tidigarelägga tillgång till resultat. (2) förutsäga resultatet av exekveringen.
l. Om kravet är att programkoden i ett datorsystem måste vara extremt minnessnål, bör man välja en processor med (1) encykelimplementering. (X) flercykelimplementering. (2) pipelining.

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

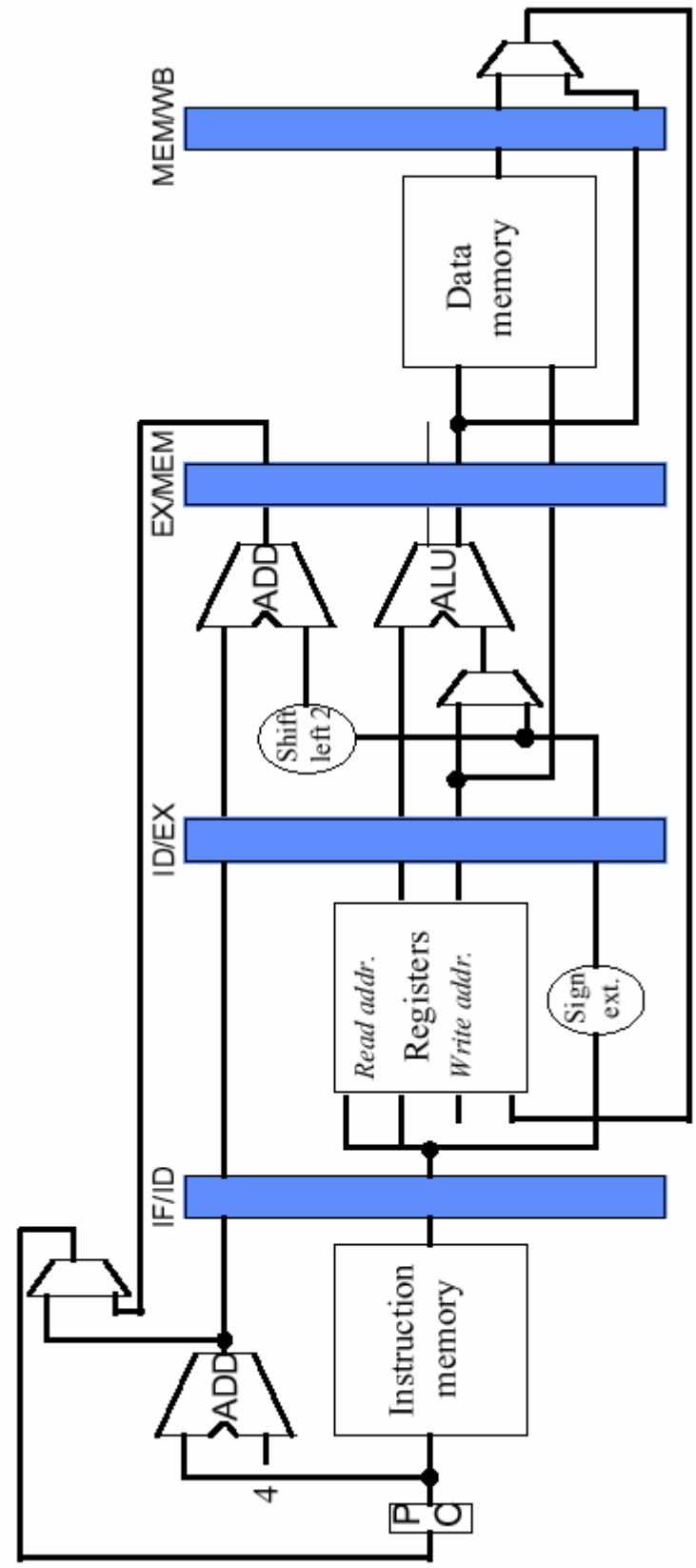
MIPS Instruction formats

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


MEM/WB
Data memory
EX/MEM
ADD
ALU
Shift left 2
ID/EX
Read addr.
Registers
Write addr.
Sign ext.
IF/ID
Instruction memory
ADD
4
P
C