Written Exam / Tentamen

Computer Organization and Components / Datorteknik och komponenter (IS1500), 9 hp
Computer Hardware Engineering / Datorteknik, grundkurs (IS1200), 7.5 hp

KTH Royal Institute of Technology

2018-06-05

08.00-13.00

Teacher on duty / Ansvarig lärare: David Broman, dbro@kth.se, +46 73 765 20 44
Examiner / Examinator: David Broman

Note that the exam questions are available both in English and in Swedish. See the rest of the document.

Instructions in English

- Allowed aids: One sheet of A4 paper with handwritten notes. You may write on both sides of the paper.
- Explicitly forbidden aids: Textbooks, electronic equipment, calculators, mobile phones, machine-written pages, photocopied pages, pages of different size than A4.
- Please write and draw carefully. Unreadable text may lead to zero points.
- You may write your answers in either Swedish or English.

The exam consists of two parts:

- **Part I: Fundamentals:** The maximal number of points for Part I is 48 points (for IS1500) and 40 points (for IS1200). There are 8 points for each of the six course modules. All questions in Part I expect only short answers. At most a few sentences are needed.
- **Part II: Advanced:** The maximal number of points for Part II is 50 points. In the answers, it is required that the student discuss, analyze, or construct. Furthermore, answers to these questions require clear motivations.

Grades

To get a pass grade (A, B, C, D, or E), it is required to pass Part I of the exam. For IS1500 students, it is required to get at least 2 points on each module (excluding bonus points), and in total at least 36 points on Part I (including bonus points). For IS1200 students, it is required to get at least 2 points on each module (excluding bonus points), and to get at least 30 points in total on questions 1, 2, 4, 5, and 6 on Part I (including bonus points).

Grading scale (For both IS1200 and IS1500):

- A: 41-50 points on Part II and the completion of an advanced project.
- B: 31-40 points on Part II and the completion of an advanced project.
- C: 21-30 points on Part II
- D: 11-20 points on Part II
- E: 0-10 points on Part II
- FX: At least 36 points (for IS1500) or at least 30 points (for IS1200) on Part I, and at most one module with less than 2 points.
- F: otherwise

Results

- The result will be announced at latest 2018-06-26.
- If a student received grade FX, it is possible to request a complementary examination. The examiner decides if it is oral or in written form. Such complementary examination must be requested by the student. Please send an email to `dbro@kth.se` at latest 2018-07-17.

Instruktioner på Svenska

- Tillåtna hjälpmedel: En A4-sida med handskrivna anteckningar. Det är tillåtet att skriva på båda sidorna av pappret.
- Förbjudna hjälpmedel: Läroböcker, elektroniska hjälpmedel, miniräknare, mobiltelefoner, maskinskrivna sidor, kopierade papper, sidor av andra storlekar än A4.
- Skriv och rita noggrant. Oläsbar text kan resultera i noll poäng.
- Du kan skriva dina svar på antingen engelska eller svenska.

Tentamen består av två delar:

- **Del I: Fundamentala delen:** Maximalt antal poäng för del I är 48 poäng (för IS1500) och 40 poäng (för IS1200). Totalpoängen per kursmodul är 8 poäng (6 moduler totalt). För del I förväntas det endast korta svar på frågorna. Endast ett fåtal meningar krävs.
- **Del II: Avancerade delen:** Det maximala antalet poäng för del II är 50 poäng. I svaren för denna del krävs att studenten diskuterar, analyserar och konstruerar. Vidare kräver svaren till dessa frågor tydliga motiveringar.

Betyg

För att erhålla godkänt betyg (A, B, C, D eller E) krävs att man får godkänt på del I. För IS1500-studenter krävs det minst 2 poäng på varje modul (exklusive bonuspoäng) samt 36 poäng eller mer på del I (inklusive bonus poäng) för att få godkänt på tentamen. För IS1200-studenter krävs det minst 2 poäng på varje modul (exklusive bonuspoäng) samt 30 poäng eller mer på frågorna 1, 2, 4, 5 och 6 på del I (inklusive bonus poäng) för att bli godkänd.

Betygsskala (För både IS1200 och IS1500):

- A: 41-50 poäng på del II samt genomförandet av ett avancerat project.
- B: 31-40 poäng på del II samt genomförandet av ett avancerat project.
- C: 21-30 poäng på del II
- D: 11-20 poäng på del II
- E: 0-10 poäng på del II
- FX: Minst 36 poäng (för IS1500) eller minst 30 poäng (för IS1200) på del I och som mest en modul med mindre än 2 poäng.
- F: i övriga fall

Resultat

- Resultaten kommer att meddelas senast 2018-06-26.
- Om en student får FX är det möjligt att begära en komplementerande examination. Examinatorn bestämmer om examinationen är muntlig eller skriftlig. Denna examination måste begäras via epost av studenten. Skicka epost till `dbro@kth.se` senast 2018-07-17.

Part I: Fundamentals

1. **Module 1: C and Assembly Programming**

   (a) Write down the machine code for the following 32-bit MIPS instruction. Answer as a hexadecimal number. (3 points)

   ```
   addi   $t3,$t8,-23
   ```

   (b) Consider the following C program code.

   ```
   int final[4];
   int list[] = {1,2,4,10,10,20};

   int *p = final;

   void boo(int *some, int dir){
     for(int i=0; i<4; i++){
       *p = *(some+1)*4 + i;
       some = some + dir;
       p++;
     }
   }

   void foo(){
     boo(list, 2);
   }
   ```

   When function `foo` returns, what are then the values of the elements in array `final`? For each of the elements in the array, answer with the correct integer number, or with `unknown` if it is not possible to determine the exact value with the given information. (4 points)

   (c) Consider again the program code in the previous exercise. Modify the code line with the function call to `boo` inside function `foo`, so that no array out-of-bound access can occur. Answer with the single modified line of code. (1 points)

2. **Module 2: I/O Systems**

(a) Consider the following C code that writes and reads to and from memory mapped I/O.

```
volatile int* switches = (volatile int*) 0x1b430010;
volatile int* leds = (volatile int*) 0xabbb8050;
int temp = ((*switches) >> 6) & 0x3;
*leds = (*leds & 0xff9f) | XXX;
```

From the C code above it is possible to derive that some bits are read from the 16-bit switch port and used to turn on or off some LEDs, without affecting the other LEDs on the 16-bit LED port. One missing expression is marked as `XXX`. Write out what expression `XXX` should be, so that the LEDs are correctly turned on or off.
(2 points)

(b) Translate the complete C code above into MIPS assembly code, including the missing C code expression. You are not allowed to use pseudo instructions. Note that a partial solution may give certain points. (4 points)

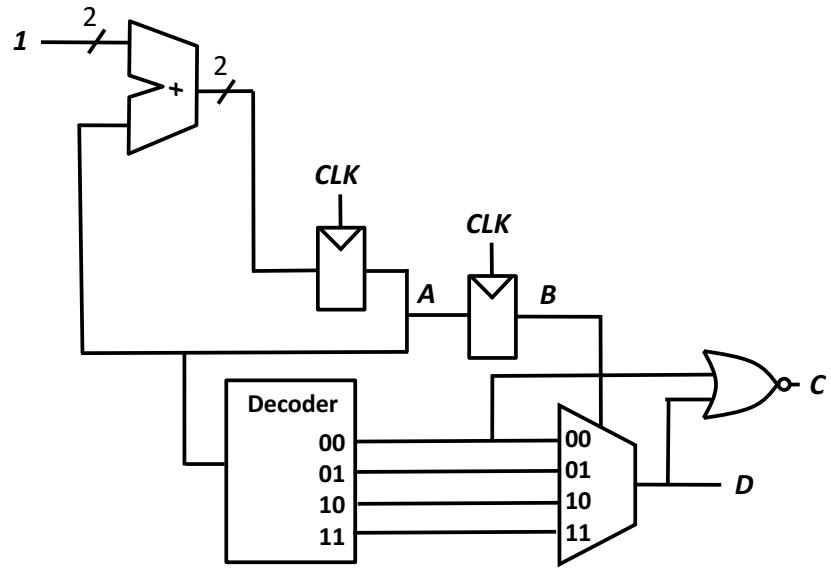
(c) For each of the following two statements, state if the statement is true or false. If you claim that the statement is false, answer shortly why the statement is false (2 sentences). You do not need to give a motivation if the statement is true. (2 points)

i. Direct memory access (DMA) is a techinque in the C programming language to get direct access to the main memory using pointers.

ii. An asynchronous bus does not need a clock signal between the sender and the receiver.

3. **Module 3: Logic Design (for IS1500 only)**

(a) Consider the following digital circuit.



The following table shows the different signals A, B, C, and D, after 1, 2, 3, and 4 clock ticks.

Tick	A	B	C	D
1	2	3		
2				
3				
4				

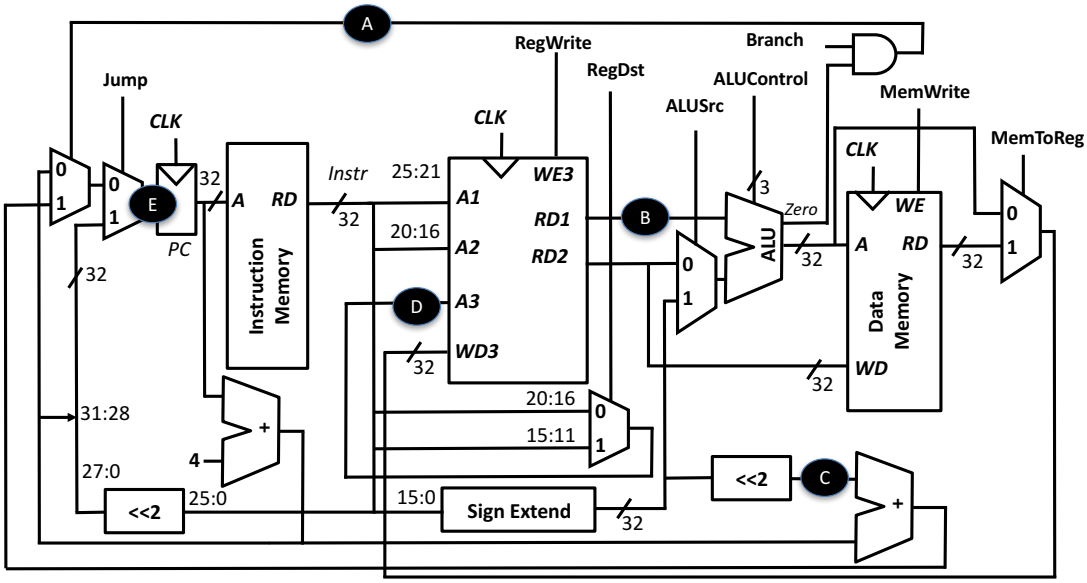
Note that the correct values after clock tick 1 have already been filled in for signals A and B. Copy the table and fill in the missing values. Answer with decimal numbers. (5 points)

(b) Suppose a register file has 3 read ports and 1 write port. The address signals for all the read ports have a bus width of 12 bits. It is possible to read 8 bits concurrently from each read port. What is the capacity of the register file, i.e., how many bytes can the register file store in total? What is the address bus width of the write port, if all byte positions in the register file can be written to?

(3 points)

4. **Module 4: Processor Design**

(a) Consider the following data path for a single-cycle 32-bit MIPS processor:



Suppose instruction

```
lw  $s1, -13($a0)
```

is currently being executed and that this instruction is stored at address `0x40002230` in memory. What are then the signal values for A, B, C, D, and E? Answer `unknown` for a signal value, if it is not possible to determine an exact value with the given information. Note that signal E is the signal between the output from the multiplexer and the input to the register. Answer with hexadecimal numbers.
(5 points)

(b) Consider the following MIPS code:

```
        lw      $t1,4($s0)
        slt     $t1,$s0,$t2
        beq     $t1,$s1,next
        add     $t1,$s0,$s1
next:
```

Assume that the code is executed on a 32-bit 5-stage pipelined MIPS processor and that the comparison in the `beq` instruction is done in the decode stage. Explain which data hazard or data hazards exist(s) in the code. State explicitly which instruction(s) and which registers are involved. State how the hazard or hazards can be solved. (3 points)

5. **Module 5: Memory Hierarchy**

(a) A data cache for a 32-bit processor has the capacity of 16384 bytes. The tag size is 20 bits and each block is 16 bytes. How many sets are there in the cache, what is the associativity of the cache, and how many valid bits exist in the cache? (3 points)

(b) Consider the following 32-bit MIPS assembly code.

```
lui     $t0,0x1001
lw      $t1,0x0($t0)
lw      $t1,0x8($t0)
lw      $t1,0x14($t0)
```

Suppose that the processor has separate data and instruction caches. Both caches are direct mapped with a capacity of 4096 bytes. The instruction cache has a block size of 8 bytes, and the data cache has a block size of 16 bytes. Assume that both caches are empty before the four instructions are executed (all valid bits in the cache are set to zero). The `lui` instruction is stored in the instruction memory at address `0x40000004`.

i. What is the instruction cache miss rate?

ii. What is the data cache miss rate?

iii. Is temporal locality, or spatial locality, or both kinds of locality utilized?

(5 points)

6. **Module 6: Parallel Processors and Programs**

   (a) Suppose a program can be divided into a sequential part A and another part B. Assume further that part B is embarrassingly parallelizable, meaning that N cores gives N times improvement. If part B represents 25% of the computation time when $N = 1$, what is then the maximal speedup that can be achieved? (3 points)

   (b) For each of the following three statements, state if the statement is true or false. If you claim that the statement is false, answer shortly why the statement is false (max two sentences). You do not need to give a motivation if the statement is true.

      i. Single Instruction Single Data (SISD) is a very unusual approach and there have been extremely few examples of SISD processors in the past.

      ii. Moore's law means that the number of transistors doubles approximately every second year.

      iii. Multiple issue is one form of instruction-level parallelism (ILP).

      (3 points)

   (c) The advanced vector extension (AVX) instruction `vaddpd` can operate on multiple double precision floating-point numbers in parallel. How many double-precision floating-point operations can this instruction perform in parallel, assuming AVX with 256 bits registers? (2 points)

Part II: Advanced

7. For each of the following three items, clearly explain: i) what the concepts mean, ii) the main differences, and iii) the similarities.

   (a) N-way set associative cache vs. management of pages in virtual memories

   (b) Advanced Vector Extension (AVX) vs. processors with multiple issue

   (c) hardware multithreading vs. software threads in an operating system

   (15 points)

8. Consider the following 32-bit MIPS assembly code.

```
foo:
  addi  $v0,$0,0
  addi  $t1,$0,0
outer:
  addi  $t2,$0,0
  slt   $t0,$t1,$a1
  beq   $t0,$0,exit
inner:
  slt   $t0,$t2,$t1
  beq   $t0,$0,skip
body:
  mul   $t0,$t1,$a1
  add   $t0,$t0,$t2
  sll   $t0,$t0,2
  add   $t0,$t0,$a0
  lw    $t3,0($t0)
  add   $v0,$v0,$t3
  addi  $t2,$t2,1
  j     inner
skip:
  addi  $t1,$t1,1
  j     outer
exit:
  jr    $ra
```

(a) Write down a C code function that corresponds to the MIPS code. The C function must fulfill the following requirements:

- The C code must be at most 8 lines long.
- The C code should not include any other control structures than `for` loops and a `return` statement. If there is only one statement, do not use curly brackets in the body of `for` loops.
- The function parameters should be given reasonable names, that is, they should not be named after the MIPS register names.
- You are not allowed to use array indexing (using square brackets). You must use pointers.
- The function signature must make it impossible for the function body to accidentally update the memory.

(10 points)

(b) If the new C function is called using expression `foo(mtx,4)` and `mtx` is defined as below, what is then the return value?
(5 points)

```
int mtx[] = {
  1,  2,  3,  4,
  5,  6,  7,  8,
  9, 10, 11, 12,
 13, 14, 15, 16
};
```

9. Consider the MIPS assembly code in Exercise 8 again.

   Assume that function `foo` executes on a 32-bit MIPS processor with a 5-stage pipeline *without* branch delay slots, with policy *branch predict not taken*. Assume that the `mul` instruction takes the same number of clock cycles as other R instructions and that the comparison in the `beq` instruction takes place in the decode stage. Both data and control hazards can take place, and they are handled using forwarding, stalling, and flushing.

   The processor has separate data and instrution caches, each with a capacity of 4096 bytes and a block size of 16 bytes. Assume a cache miss penalty of 10 clock cycles, and no penalty for a hit. Both caches are direct mapped. Assume that the first assembly instruction (`addi`) of the function is located at memory address `0x40002200`. Before the function is executed, assume that both caches are empty (all valid bits are 0).

   This task concerns the problem of computing worst-case and best-case execution numbers where some of the input to the function are unknown. This means that you should think of the situation (possible initial value in registers and possible cache states) that gives either the best-case or worst-case scenarios. Assume register `$a1` can be initiated to one of the possible values 0, 1, 2, 3, and 4, and that we cannot know anything about any other register values or memory content. Given these constraints, compute the

   (a) the best-case and worst-case number of instruction cache misses that can occur during execution of the function when it returns and terminates correctly. (4 points)

   (b) the best-case and worst-case number of data cache misses that can occur during execution of the function when it returns and terminates correctly. (6 points)

   (c) worst-case number of clock cycles that can occur during execution of the function when it returns and terminates correctly. The total number of clock cycles are counted from the fetch of the first instruction, until all 5 stages are completed by the last return (`jr`) instruction. For instance, if a function would only consist of one `jr $ra` instruction, the total number of cycles is 6 (5-stage pipeline + flushing 1 cycle). (10 points)

   Note that for certain input values, such as unaligned memory accesses, the function will not terminate correctly. These incorrect cases must be excluded when computing the best- and worst-case numbers. You must write out and explain your solution in detail.

Del I: Grundläggande

1. **Modul 1: C-programmering och assemblerspråk**

   (a) Skriv ner maskinkoden för följande 32-bitars MIPS-instruktion. Svara med ett hexadecimalt tal. (3 poäng)

   ```
   addi   $t3,$t8,-23
   ```

   (b) Betrakta följande C-programkod.

   ```
   int final[4];
   int list[] = {1,2,4,10,10,20};

   int *p = final;

   void boo(int *some, int dir){
     for(int i=0; i<4; i++){
       *p = *(some+1)*4 + i;
       some = some + dir;
       p++;
     }
   }

   void foo(){
     boo(list, 2);
   }
   ```

   När funktionen `foo` returnerat, vad är då värdena på elementen i arrayen `final`? För vart och ett av elementen i arrayen, svara med det korrekta heltalet, eller med `okänd` om det inte går att avgöra det exakta värdet utifrån den information som ges i uppgiften. (4 poäng)

   (c) Betrakta återigen programkoden i föregående övning. Ändra programraden där funktionen `boo` anropas inuti funktionen `foo`, så att ingen referens utanför arraygränserna kan ske. Svaret ska bara innehålla den ensamma ändrade programraden. (1 poäng)

2. **Modul 2: In- och utmatningssystem**

   (a) Betrakta följande C-kod som skriver till och läser från minnesmappade in- och utenheter.

```
volatile int* switches = (volatile int*) 0x1b430010;
volatile int* leds = (volatile int*) 0xabbb8050;
int temp = ((*switches) >> 6) & 0x3;
*leds = (*leds & 0xff9f) | XXX;
```

   Utifrån C-koden här ovanför går det att dra slutsatsen att vissa bitar läses från en 16-bitars switch-port, och att bitarna används för att tända eller släcka vissa lysdioder, utan att påverka andra lysdioder på en 16-bitars LED-port. Det fattas ett uttryck på ett ställe som markerats med `XXX`. Skriv ut vad uttrycket `XXX` borde vara, så att lysdioderna tänds och släcks på rätt sätt. (2 poäng)

   (b) Översätt den kompletta C-koden här ovanför till MIPS-assemblerkod, inklusive det saknade C-kodsuttrycket. Du får inte använda pseudoinstruktioner. Observera att en ofullständig lösning kan ge vissa poäng. (4 poäng)

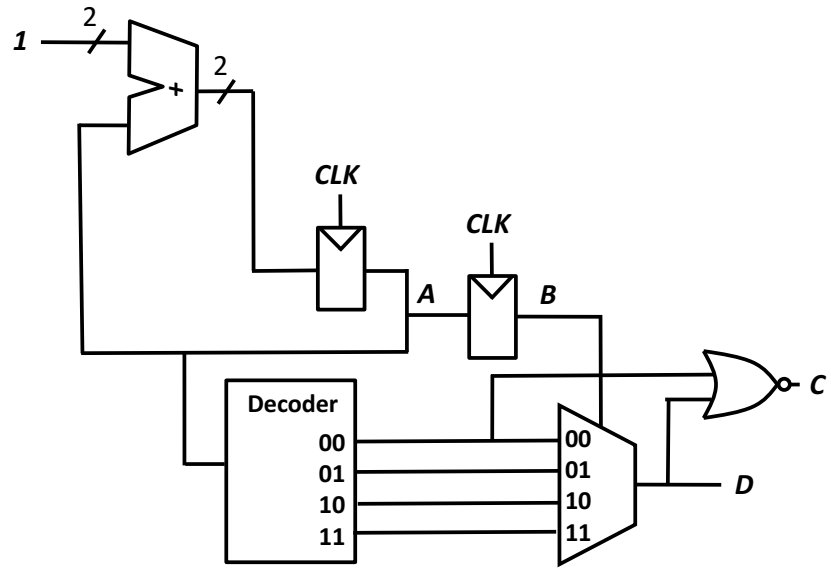
   (c) För vart och ett av följande två påståenden, ange om påståendet är sant eller falskt. Om du hävdar att påståendet är falskt, svara kort (med högst 2 meningar) varför påståendet är falskt. Om du hävdar att påståendet är sant behöver du inte ge någon motivering. (2 poäng)

      i. Direkt minnesåtkomst (Direct Memory Access, DMA) är en teknik i programspråket C, för att få direkt åtkomst till primärminnet genom användning av pekare.

      ii. En asynkron buss behöver inte ha någon klocksignal mellan sändare och mottagare.

3. **Modul 3: Digital Design (endast för IS1500)**

(a) Betrakta följande digitala krets.



Följande tabell visar de olika signalerna A, B, C och D, efter 1, 2, 3 och 4 klockningar (clock ticks).

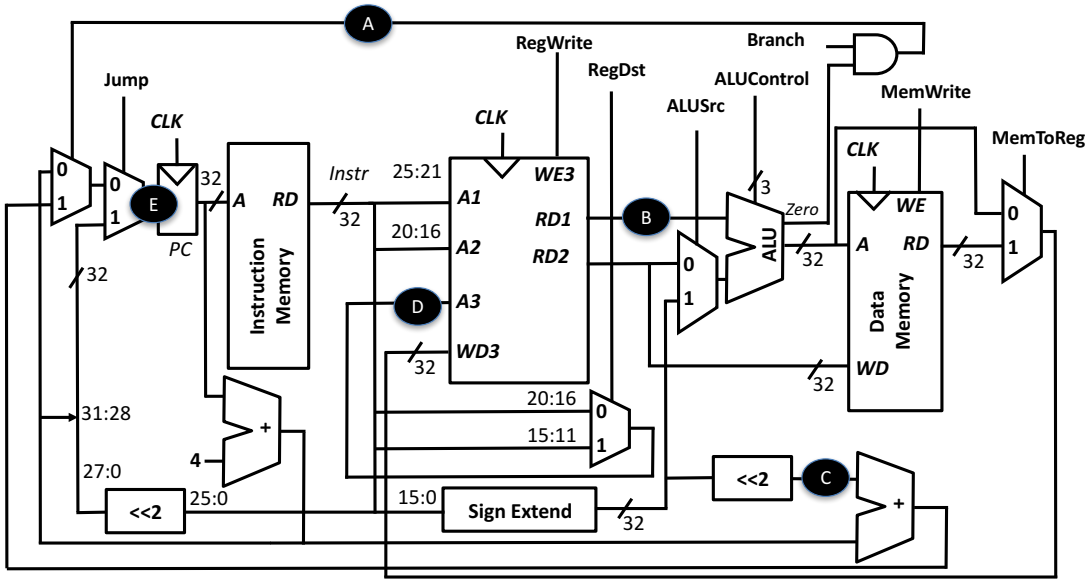
Tick	A	B	C	D
1	2	3		
2				
3				
4				

Observera att de korrekta värdena efter klockning 1 redan fyllts i för signalerna A och B. Kopiera tabellen och fyll i de värden som saknas. Svara med decimala tal. (5 poäng)

(b) Anta att en registerfil (register file) har 3 läsportar och 1 skrivport. Adressignalerna för alla läsportarna har en bussbredd på 12 bitar. Det går att läsa 8 bitar samtidigt från varje läsport. Vilken kapacitet har registerfilen, det vill säga hur många bytes kan registerfilen lagra totalt? Vad är bredden på adressbussen för skrivporten, om man antar att alla byte-positioner i registerfilen går att skriva till? (3 poäng)

4. **Modul 4: Processorkonstruktion**

(a) Betrakta följande dataväg för en 1-cykels 32-bitars MIPS-processor:



Anta att instruktionen

```
lw  $s1, -13($a0)
```

just håller på att exekveras och att den instruktionen är lagrad på adress `0x40002230` i minnet. Vad är då signalvärdena för A, B, C, D och E? Svara `okänt` för ett signalvärde om det inte går att avgöra ett exakt värde utifrån den givna informationen. Observera att signalen E är signalen mellan multiplexerns utgång och ingången till registret. Svara med hexadecimala tal. (5 poäng)

(b) Betrakta följande MIPS-kod:

```
        lw      $t1,4($s0)
        slt     $t1,$s0,$t2
        beq     $t1,$s1,next
        add     $t1,$s0,$s1
next:
```

Anta att koden exekveras på en 32-bitars MIPS-processor med 5-stegs pipeline, och att jämförelsen i instruktionen `beq` görs i avkodningssteget (the decode stage). Förklara vilken eller vilka datahazard(-er) som finns i koden. Ange uttryckligen vilka instruktion(-er) och vilka register som är inblandade. Ange hur hazarden eller hazarderna kan lösas. (3 poäng)

5. **Modul 5: Minneshierarkier**

   (a) Ett datacacheminne för en 32-bitars processor har en kapacitet på 16384 byte. Adresslappen (tag) är 20 bitar lång, och varje block är 16 byte. Hur många set (rader) finns det i cacheminnet, vad är associativiteten hos cacheminnet, och hur många giltigbitar (valid bits) existerar i cacheminnet? (3 poäng)

   (b) Betrakta nedanstående assemblerkod för 32-bitars MIPS.

```
lui     $t0,0x1001
lw      $t1,0x0($t0)
lw      $t1,0x8($t0)
lw      $t1,0x14($t0)
```

   Anta att processorn har separata instruktions- och datacacheminnen. Båda cacheminnena är direktmappade med en kapacitet på 4096 byte. Instruktionscacheminnet har en blockstorlek på 8 byte, och datacacheminnet har en blockstorlek på 16 byte. Anta att båda cacheminnena är tomma innan de fyra instruktionerna exekveras (alla giltigbitar i cacheminnet är satta till noll). Instruktionen `lui` finns lagrad i instruktionsminnet på adress `0x40000004`.

   i. Vad är misskvoten (miss rate) i instruktionscacheminnet?

   ii. Vad är misskvoten (miss rate) i datacacheminnet?

   iii. Används tidslokalitet (temporal locality), eller rumslokalitet (spatial locality), eller båda sorternas lokalitet?

   (5 poäng)

6. **Modul 6: Parallella processorer och program**

   (a) Anta att ett program kan delas i en sekventiell del A och en annan del B. Anta vidare att del B är naturligt parallelliserbar, vilket innebär att N kärnor ger N gångers förbättring. Om del B representerar 25% av beräkningstiden när $N = 1$, vad är då den maximala förbättring (speedup) som kan uppnås? (3 poäng)

   (b) För vart och ett av följande tre påståenden, ange om påståendet är sant eller falskt. Om du hävdar att påståendet är falskt, svara kort (med högst 2 meningar) varför påståendet är falskt. Om du hävdar att påståendet är sant behöver du inte ge någon motivering. (2 poäng)

      i. Single Instruction Single Data (SISD) är ett väldigt ovanligt konstruktionssätt och det har funnits extremt få exempel på SISD-processorer tidigare.

      ii. Moores lag betyder att antalet transistorer fördubblas ungefär vartannat år.

      iii. Multiple issue är en form av instruktionsnivåparallellitet (ILP).

      (3 poäng)

   (c) Instruktionen `vaddpd` är en del av Advanced Vector Extension (AVX). Instruktionen `vaddpd` kan parallellbehandla flera flyttal, där varje flyttal har dubbel precision. Hur många flyttalsoperationer kan denna instruktion utföra parallellt, om vi antar att AVX har 256-bitars register? (2 poäng)

Del II: Avancerat

7. För var och en av följande tre punkter, förklara tydligt: i) vad begreppen betyder, ii) de huvudsakliga skillnaderna, och iii) likheterna.

   (a) N-vägs associativt cacheminne, jämfört med hanteringen av sidor i virtuellt minne

   (b) Advanced Vector Extension (AVX), jämfört med processorer med multiple issue

   (c) hårdvarumultitrådning, jämfört med mjukvarutrådar i ett operativsystem

   (15 poäng)

8. Betrakta följande 32-bitars MIPS-assemblerkod.

```
foo:
  addi  $v0,$0,0
  addi  $t1,$0,0
outer:
  addi  $t2,$0,0
  slt   $t0,$t1,$a1
  beq   $t0,$0,exit
inner:
  slt   $t0,$t2,$t1
  beq   $t0,$0,skip
body:
  mul   $t0,$t1,$a1
  add   $t0,$t0,$t2
  sll   $t0,$t0,2
  add   $t0,$t0,$a0
  lw    $t3,0($t0)
  add   $v0,$v0,$t3
  addi  $t2,$t2,1
  j     inner
skip:
  addi  $t1,$t1,1
  j     outer
exit:
  jr    $ra
```

   (a) Skriv ner en C-kodsfunktion som motsvarar MIPS-koden. C-funktionen måste uppfylla följande krav:

   - C-koden får vara högst 8 rader lång.
   - C-koden bör inte innehålla andra kontrollstrukturer än `for`-slingor och en `return`-sats. Om det bara finns en sats så ska krullparenteser inte användas i kroppen till `for`-loopar.
   - Funktionsparametrarna ska ha rimliga namn. Det innebär att de inte får ha namn efter namnen på MIPS-registren.
   - Du får inte använda array-indexering (med hakparenteser). Du måste använda pekare.
   - Funktionssignaturen ska göra det omöjligt för funktionskroppen att oavsiktligt uppdatera minnet.

   (10 poäng)

   (b) Om den nya C-funktionen anropas med uttrycket `foo(mtx,4)` och `mtx` är definierat som här nedanför, vad blir då returvärdet? (5 poäng)

```
int mtx[] = {
  1,  2,  3,  4,
  5,  6,  7,  8,
  9, 10, 11, 12,
 13, 14, 15, 16
};
```

9. Betrakta återigen MIPS-assemblerkoden i Övning 8.

   Anta att funktionen `foo` exekveras på en 32-bitars MIPS-processor med en 5-stegs pipeline *utan* hoppluckor (without branch delay slots), med policy *branch predict not taken*. Anta att instruktionen `mul` tar samma antal klockcykler som andra instruktioner av R-typ, och att jämförelsen i instruktionen `beq` sker i avkodningssteget (the decode stage). Både data- och kontrollhazarder han inträffa, och de hanteras med forwarding, stalling och tömning (flushing).

   Processorn har separata data- och instruktionscacheminnen, vart och ett med en kapacitet på 4096 byte och en blockstorlek på 16 byte. Anta att extratiden vid miss (miss penalty) är 10 klockcykler och att det inte blir någon extratid vid en träff (hit). Båda cacheminnena är direktmappade. Anta att den första assemblerinstruktionen (`addi`) i funktionen finns på minnesadress `0x40002200`. Innan funktionen exekveras, anta att båda cacheminnena är tomma (alla giltigbitar är 0).

   Denna uppgift berör problemet med att beräkna sämsta möjliga (worst-case) och bästa möjliga (best-case) exekveringstal, där en del input till funktionen är okänd. Detta innebär att du bör tänka på situationen (möjliga startvärden i register och möjliga cacheminnestillstånd) som ger antingen bästa möjliga eller sämsta möjliga scenario. Anta att register `$a1` kan initialiseras till ett av de möjliga värdena 0, 1, 2, 3 och 4, samt att vi inte kan veta någonting om något annat registervärde eller minnesinnehåll. Givet dessa begränsningar, beräkna

   (a) bästa möjliga antal (best-case number) och sämsta möjliga antal (worst-case number) missar i instruktionscacheminnet som kan inträffa under exekveringen av funktionen när den returnerar och avslutas korrekt. (4 poäng)

   (b) bästa möjliga antal (best-case number) och sämsta möjliga antal (worst-case number) missar i datacacheminnet som kan inträffa under exekveringen av funktionen när den returnerar och avslutas korrekt. (6 poäng)

   (c) sämsta möjliga antal klockcykler som kan inträffa under exekveringen av funktionen när den returnerar och avslutas korrekt. Det totala antalet klockcykler beräknas från hämtningen av den första instruktionen, fram tills alla 5 pipelinestegen har avslutats av den sista returinstruktionen (`jr`). Om en funktion till exempel skulle bestå endast av en instruktion `jr $ra`, så skulle det totala antalet cykler vara 6 (5-stegs pipeline + 1 cykels tömning). (10 poäng)

   Observera att för vissa invärden, som till exempel icke-alignade minnesåtkomster (unaligned memory accesses), kommer funktionen inte att avslutas korrekt. Dessa felaktiva fall ska uteslutas vid beräkning av bästa möjliga antal (best-case number) och sämsta möjliga antal (worst-case number). Du ska skriva ut och förklara din lösning i detalj.

MIPS Reference Sheet

David Broman, KTH Royal Institute of Technology
Version 1.15, March13, 2018

INSTRUCTIONS (SUBSET)

Name (format, op, funct)	Syntax	Operation
add (R,0,32)	`add rd,rs,rt`	reg(rd) := reg(rs) + reg(rt);
add immediate (I,8,na)	`addi rt,rs,imm`	reg(rt) := reg(rs) + signext(imm);
add immediate unsigned (I,9,na)	`addiu rt,rs,imm`	reg(rt) := reg(rs) + signext(imm);
add unsigned (R,0,33)	`addu rd,rs,rt`	reg(rd) := reg(rs) + reg(rt);
and (R,0,36)	`and rd,rs,rt`	reg(rd) := reg(rs) & reg(rt);
and immediate (I,12,na)	`andi rt,rs,imm`	reg(rt) := reg(rs) & zeroext(imm);
branch on equal (I,4,na)	`beq rs,rt,label`	if reg(rs) == reg(rt) then PC = BTA else NOP;
branch on not equal (I,5,na)	`bne rs,rt,label`	if reg(rs) != reg(rt) then PC = BTA else NOP;
jump and link register (R,0,9)	`jalr rs`	$ra := PC + 4; PC := reg(rs);
jump register (R,0,8)	`jr rs`	PC := reg(rs);
jump (J,2,na)	`j label`	PC := JTA;
jump and link (J,3,na)	`jal label`	$ra := PC + 4; PC := JTA;
load byte (I,32,na)	`lb rt,imm(rs)`	reg(rt) := signext(mem[reg(rs) + signext(imm)]$_{7:0}$);
load byte unsigned (I,36,na)	`lbu rt,imm(rs)`	reg(rt) := zeroext(mem[reg(rs) + signext(imm)]$_{7:0}$);
load upper immediate (I,15,na)	`lui rt,imm`	reg(rt) := concat(imm, 16 bits of 0);
load word (I,35,na)	`lw rt,imm(rs)`	reg(rt) := mem[reg(rs) + signext(imm)];
multiply, 32-bit result (R,28,2)	`mul rd,rs,rt`	reg(rd) := reg(rs) * reg(rt);
nor (R,0,39)	`nor rd,rs,rt`	reg(rd) := not(reg(rs) | reg(rt));
or (R,0,37)	`or rd,rs,rt`	reg(rd) := reg(rs) | reg(rt);
or immediate (I,13,na)	`ori rt,rs,imm`	reg(rt) := reg(rs) | zeroext(imm);
set less than (R,0,42)	`slt rd,rs,rt`	reg(rd) := if reg(rs) < reg(rt) then 1 else 0;
set less than unsigned (R,0,43)	`sltu rd,rs,rt`	reg(rd) := if reg(rs) < reg(rt) then 1 else 0;
set less than immediate (I,10,na)	`slti rt,rs,imm`	reg(rt) := if reg(rs) < signext(imm) then 1 else 0;
set less than immediate unsigned (I,11,na)	`sltiu rt,rs,imm`	reg(rt) := if reg(rs) < signext(imm) then 1 else 0; (inequality < compares using unsigned values)
shift left logical (R,0,0)	`sll rd,rt,shamt`	reg(rd) := reg(rt) << shamt;
shift left logical variable (R,0,4)	`sllv rd,rt,rs`	reg(rd) := reg(rt) << reg(rs$_{4:0}$);
shift right arithmetic (R,0,3)	`sra rd,rt,shamt`	reg(rd) := reg(rt) >>> shamt;
shift right logical (R,0,2)	`srl rd,rt,shamt`	reg(rd) := reg(rt) >> shamt;
shift right logical variable (R,0,6)	`srlv rd,rt,rs`	reg(rd) := reg(rt) >> reg(rs$_{4:0}$);
store byte (I,40,na)	`sb rt,imm(rs)`	mem[reg(rs) + signext(imm)]$_{7:0}$:= reg(rt)$_{7:0}$;
store word (I,43,na)	`sw rt,imm(rs)`	mem[reg(rs) + signext(imm)] := reg(rt);
subtract (R,0,34)	`sub rd,rs,rt`	reg(rd) := reg(rs) - reg(rt);
subtract unsigned (R,0,35)	`subu rd,rs,rt`	reg(rd) := reg(rs) - reg(rt);
xor (R,0,38)	`xor rd,rs,rt`	reg(rd) := reg(rs) ^ reg(rt);
xor immediate (I,14,na)	`xori rt,rs,imm`	reg(rt) := reg(rs) ^ zeroext(imm);

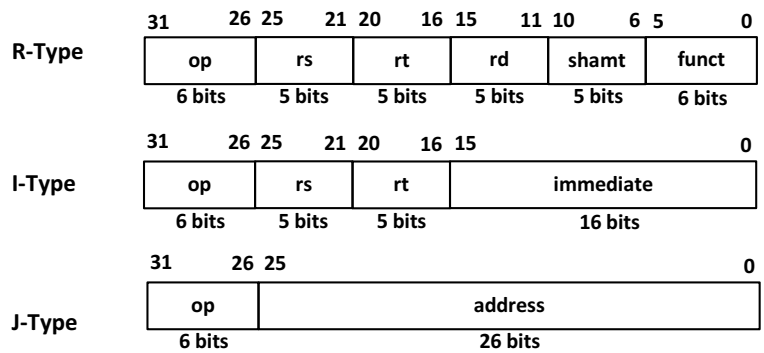
PSEUDO INSTRUCTIONS (SUBSET)

Name	Example	Equivalent Basic Instructions
load address	`la $t0,label`	`lui $at,hi-bits-of-address` `ori $t0,$at,lower-bits-of-address`
load immediate	`li $t0,0xabcd1234`	`lui $at,0xabcd` `ori $t0,$at,0x1234`
branch if less or equal	`ble $t0,$t1,label`	`slt $at,$t1,$t0` `beq $at,$zero,label`
move	`move $t0,$t1`	`add $t0,$t1,$zero`
no operation	`nop`	`sll $zero,$zero,0`

ASSEMBLER DIRECTIVES (SUBSET)

data section	`.data`
ASCII string declaration	`.ascii "a string"`
word alignment	`.align 2`
word value declaration	`.word 99`
byte value declaration	`.byte 7`
global declaration	`.global foo`
allocate X bytes of space	`.space X`
code section	`.text`

INSTRUCTION FORMAT



REGISTERS

Name	Number	Description
$0, $zero	0	constant value 0
$at	1	assembler temp
$v0	2	function return
$v1	3	function return
$a0	4	argument
$a1	5	argument
$a2	6	argument
$a3	7	argument
$t0	8	temporary value
$t1	9	temporary value
$t2	10	temporary value
$t3	11	temporary value
$t4	12	temporary value
$t5	13	temporary value
$t6	14	temporary value
$t7	15	temporary value
$s0	16	saved temporary
$s1	17	saved temporary
$s2	18	saved temporary
$s3	19	saved temporary
$s4	20	saved temporary
$s5	21	saved temporary
$s6	22	saved temporary
$s7	23	saved temporary
$t8	24	temporary value
$t9	25	temporary value
$k0	26	reserved for OS
$k1	27	reserved for OS
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Definitions

- Jump to target address: JTA = concat((PC + 4)$_{31:28}$, address(label), 00_2)
- Branch target address: BTA = PC + 4 + signext(imm) * 4

Clarifications

- All numbers are given in decimal form (base 10).
- Function signext(x) returns a 32-bit sign extended value of x in two's complement form.
- Function zeroext(x) returns a 32-bit value, where zero are added to the most significant side of x.
- Function concat(x, y, …, z) concatenates the bits of expressions x, y, …, z.
- Subscripts, for instance $X_{8:2}$, means that bits with index 8 to 2 are spliced out of the integer X.
- Function *address*(x) is the 26-bit address field value of the J-Type instruction for an address label x.
- NOP and na mean "no operation" and "not applicable", respectively.
- shamt is an abbreviation for "shift amount", i.e. how many bits that should be shifted.
- `addu` and `addiu` are misnamed *unsigned* because an add operation handles both signed and unsigned numbers in the same way. The term unsigned is actually used to describe that the instruction does not throw overflow exceptions.