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

2020-01-13

14.00-19.00

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.

Instructions in English

- Allowed aids: One sheet of A4 paper with handwritten notes. You may write on both sides of the paper. You may bring this specific paper home after the exam. It is not a scrap 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:** There are in total three advanced questions, where you should discuss, analyze, or construct, and where the answers require clear motivations. For each exercise, the result can be fail (F), satisfactory (S), good (G), or very Good (VG).

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) [1]:

- A: Passed Part I, advanced project, and three VG **or** two VG and one G on part II.
- B: Passed Part I, advanced project, and one VG and two G **or** two VG and one S on part II.
- C: Passed Part I, and three G **or** two G and one S **or** one VG and two S **or** one VG and one G and one F **or** two VG and one F **or** one VG and one G and one S on part II.
- D: Passed Part I, and three S **or** one G and one S and one F **or** two G and one F **or** one VG and one S and one F **or** one VG and two F **or** one G and two S on part II.
- E: Passed Part I.
- 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

[1] For IS1500, the grading rules changed because of the grading criteria requirements. They have been updated after the exam to give fairer grades (see course PM). The IS1200 retake exam follows the rules from VT 2019.

Results

The result will be announced at latest 2020-02-03. If the students receives FX, it is possible to request a complementary examination. The examiner decides if it is oral or in written form. The student must request such examination at latest 2020-02-24 via email to `dbro@kth.se`.

Instruktioner på Svenska

- Tillåtna hjälpmedel: En A4-sida med handskrivna anteckningar. Det är tillåtet att skriva på båda sidorna. Det är tillåtet att få tillbaka detta papper efter examen. Det är inget kladdpapper.
- 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 finns totalt tre avancerade övningar där det krävs att diskuterar, analyserar och konstruerar. Vidare krävs motiveringar. Varje övning kan resultera i icke godkänt (F), tillfredsställande (S), bra (G) och mycket bra (VG).

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 bonuspoäng) 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 bonuspoäng) för att bli godkänd.

Betygsskala (För både IS1200 och IS1500):

- A: Godkänt del I, avancerat projekt, och tre VG **eller** två VG och ett G på del II.
- B: Godkänt del I, avancerat projekt, och ett VG och två G **eller** två VG och ett S på del II.
- C: Godkänt del I, och tre G **eller** två G och ett S **eller** ett VG och två S **eller** ett VG och ett G och ett F **eller** två VG och ett F **eller** ett VG och ett G och ett S på del II.
- D: Godkänt del I, och tre S **eller** ett G och ett S och ett F **eller** två G och ett F **eller** ett VG och ett S och ett F **eller** ett VG och två F **eller** ett G och två S på del II.
- E: Godkänt del I.
- 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 2020-02-03. Vid FX är det möjligt att begära en komplementerande examination. Examinatorn bestämmer om examinationen är muntlig eller skriftlig. Studenten måste begära examination senast 2020-02-24 via epost till `dbro@kth.se`.

Part I: Fundamentals

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

   (a) Consider the following C program:

```
#include "stdio.h"

int a[] = {7,10,3,5,12,8,17,-8,4,5};
int x[] = {8,2,0};

int sum(int* p1, int* p2, int len){
  int i = 0;
  int s = 0;
  while(i < len){
    -- MISSING CODE --
  }
  return s;
}

int main(){
  int v = sum(a,x,3);
  printf("Sum␣%d\n", v);
  return 0;
}
```

   Function `sum` should sum up the values found using pointer `p1`, where pointer `p2` points to the array of indices used in the summation. That is, the `sum` function should take the first element that `p2` points to (in the first iteration of the loop it is value 8). It uses this index value to look up the value at index 8 in the array that `p1` points to (in this case, value 4). This process continues for three loop iterations. In the second iteration of the loop, index 2 is used, and therefore value 3 is found using `p1`. In the last iteration, index 0 is used, and value 7 is used. The sum that is returned using variable `s` should be $4 + 3 + 7 = 14$. Hence, the `main` function prints out value `14`.

   In the middle of function `sum`, a number of code lines are missing. Please write down these lines of code, such that the program behaves as described above. Please note that you are only allowed to use pointer arithmetic (no usage of array indexing `[]`). You are not allowed to reference arrays `a` and `x` directly; all access must be done using pointers `p1` and `p2`. (5 points)

   (b) What is the machine code of the following 32-bit MIPS assembler instruction? Answer as a 32-bit hexadecimal number. (3 points)

```
lb  $t3, -11($s2)
```

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

(a) Suppose you are developing a very simple pocket calculator that can perform two different tasks, depending on the setting of switch number 4 on the ChipKit Basic I/O shield. If the switch is high (the bit value for the switch is 1) then the calculator should perform the square function (multiply the 3-bit switch value given by switches 1 through 3 by itself). If the switch value bit for switch 4 is zero, the 3-bit value should be doubled (adding the value to itself).

On the Basic I/O shield, the toggle switches are memory mapped to the word address `0xbf8860d0` (bits 11 through 8) and the green LEDs to `0xbf886110` (bits 7 through 0). The output result should be shown as a binary number on the LEDs. Toggle switch number four is located at bit index 11. You can assume that before the program starts, the ports are already configured correctly (switches as input and LEDs as output). Also, assume that the processor is executing without branch delay slots.

The calculator is implemented as an infinite loop, using the template code below.

```
loop:
        lui     $t0,0xbf88
        lw      $t1,0x60D0($t0)
        -- MISSING CODE: PART I --
do_add:
        add     $t1,$t2,$t2
next:
        -- MISSING CODE: PART II --
        j       loop
```

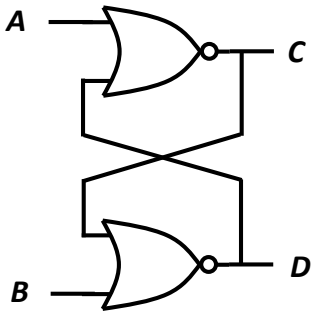
i. Write down the missing code fragment for PART I. This part should include code for extracting the right information from the switches, do correct branching, and perform the square function correctly. Note that the doubling (adding the value to itself) is already implemented in the template. (3 points)

ii. Write down the missing code fragment for PART II. This part should make sure that only the 8 bits related to the LEDs are changed when outputting the result to the memory mapped IO port. You can assume that only the 16 least significant bits contain relevant data, i.e., you do not have to preserve the 16 most significant bits (3 points)

(b) Suppose you are configuring a 32-bit hardware timer that has the clock frequency 4 MHz. You have decided that the period register has value $500\,000$. Which prescaling value should then be used if the timer should have the period of 1s? (2 points)

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

   (a) Consider the following circuit:



   In the cases below, answer if the signals are 0, 1, or `unknown`, where the latter means that it is not possible to determine a concrete value with the given information.

   i. If $A = 1$ and $B = 0$, what are then the values of C and D? (2 points)

   ii. If $A = 0$ and $B = 0$, what are then the values of C and D? (2 points)

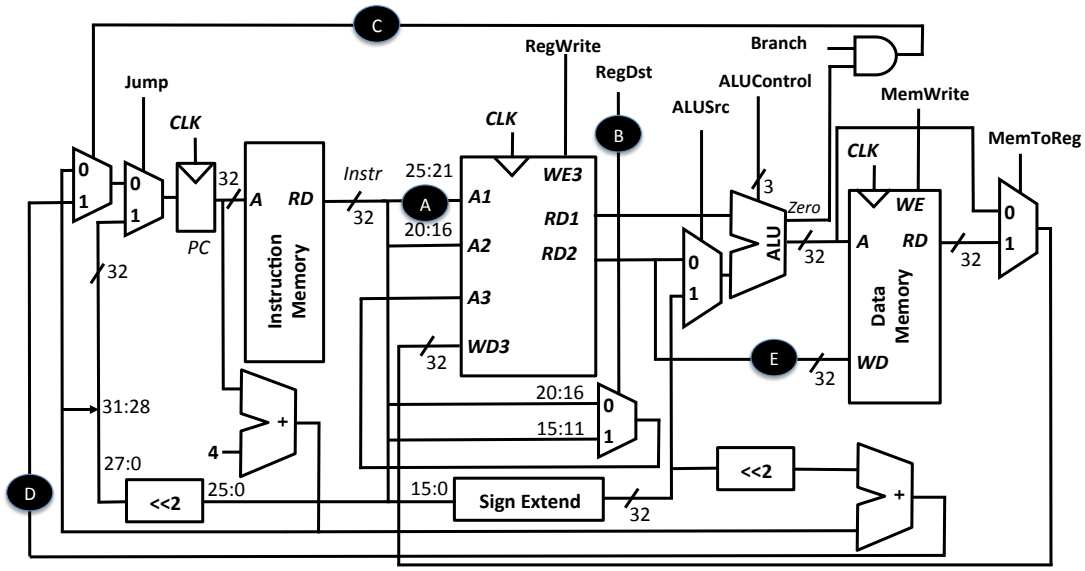
   iii. What is the official name of this circuit? (1 point)

   (b) Suppose you have a register file with one read port and one 16-bit write port. The address signals are 6 bits wide. How many bytes can the register file store in total, i.e., what is the capacity of the whole register file? (2 points)

   (c) Suppose you have a $2:4$ decoder. If the decimal input value is 2, what are then the output signals for outputs Y_0, Y_1, Y_2, and Y_3? (1 point)

4. **Module 4: Processor Design**

(a) Consider the following datapath for a single-cycle 32-bit MIPS processor.



Suppose the `bne` instruction is executing in the following program:

```
        xor     $t1,$t2,$t2
        bne     $t1,$zero,next
        addi    $t0,$t0,1
next:
```

Instruction `bne` is located at address `0x40004008`. What are then the signal values for A, B, C, D, and E when the `bne` instruction is executed? Answer with either hexadecimal numbers, or write `unknown` for a signal value where it is not possible to determine an exact value with the given information, or where the exact signal value does not matter. (5 points)

(b) Consider the following assembly code that is executing on a 5-stage pipelined 32-bit MIPS processors[2].

```
        addi    $t1,$t1,10      # 1: F D E M W
        addi    $t2,$t2,0       # 2:   F D E M W
loop:   lb      $t0,0($s0)      # 3:     F D E M W
        add     $t2,$t0,$t2     # 4:       F D E M W
        addi    $t1,$t1,-1      # 5:         F D E M W
        addi    $s0,$s0,1       # 6:           F D E M W
        bne     $t1,$zero,loop  # 7:             F D E M W
                                #               F D E M W
```

i. List all data hazards that exist. For each data hazard, state between which two instructions the hazard occurs. Assume that the equality check when branching is done in the decode stage. For instance, suppose that there is a data hazard between the two first `addi` instructions, the answer for that hazard would be 1-2 (between instruction 1 and instruction 2). (2 points)

ii. Which of the above data hazards requires stalling? (1 point)

[2] 2020-01-21: Note that the alignment of the F D E M W has been updated (a space was missing).

5. **Module 5: Memory Hierarchy**

   (a) Suppose we have a 32-bit MIPS processor with an instruction cache with capacity 1024 bytes. The tag field size is 23 bits and the block size is 8 bytes.

   - What is the byte offset field size (1 point)
   - What is the set field size? (1 point)
   - What is the associativity of the cache? (1 point)
   - How many valid bits exist in the cache? (1 point)

   (b) Suppose 6 instructions are executed in a row on a 32-bit MIPS processor. The first and the sixth instructions are `lw` instructions. The rest of the instructions are `add` or `sll` instructions. The first `lw` instruction is located at address `0x40001004`. Suppose the processor has separate data and instruction caches, where each cache has a capacity of 4 KiB (4096 bytes). Both caches are direct mapped. The instruction cache has a block size of 8 bytes, and the data cache has a block size of 16 bytes. The first `lw` instruction reads from address `0x80002000`, and the second `lw` instruction reads from address `0x80002010`. Assume that all valid bits in the two caches are zero before the code sequence executes.

   i. What is the instruction cache miss rate? (2 points)
   ii. What is the data cache miss rate? (1 point)
   iii. Do the instruction memory accesses show temporal locality, spatial locality, or both? (1 point)

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

   (a) Suppose you have developed a program that takes 6 seconds to execute on a processor with 1 core. If you execute the program on a processor with 4 cores, the program executes in 3 seconds. What is theoretically the shortest execution time of the program if you had access to an infinite number of cores? (3 points).

   (b) For each of the following five statements, determine 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. Hardware multithreading is typically used for hiding latencies.

      ii. SIMD (single instruction, multiple data) corresponds to the notion of thread-level parallelism.

      iii. Advanced Vector Extension (AVX) is a standard for handling instruction-level parallelism (ILP).

      iv. One way of handling the cache coherency problem is to use the snooping protocol.

      v. MapReduce is an efficient way for handling Very Long Instruction Words (VLIW).

      (5 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) Data-level parallelism. vs multicore

   (b) Hardware multi-threading vs. Advanced Vector Extension (AVX)

   (c) Multiple issue vs. task-level parallelism

   For IS1500 students: The question is graded in three levels S, G, and VG, with the following criteria:

   - Satisfactory (S): Some of the concepts in the question are clearly explained.
   - Good (G): Basically all concepts in the question are clearly explained, and some of the concepts are related to each other, by discussing similarities and differences.
   - Very Good (VG): Basically all concepts in the question are clearly explained, and all of the concepts are related to each other, by discussing similarities and differences.

   If none of the three levels is achieved, the exercise is considered as failed (F).

   For IS1200 students (retake exam), max 15 points. For each of the three items, max three points for the explanations of the two concepts, max one point for at least one difference, and max one point for at least one similarity. That is, max five points for each of the three items.

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

```
bar:    addi    $sp,$sp,-4
        sw      $ra,0($sp)

        lb      $t0,0($a0)
        beq     $t0,$zero,return

        bne     $t0,32,return       # 32 = ascii code for space
        lw      $t0,0($a1)
        addi    $t0,$t0,1
        sw      $t0,0($a1)

        addi    $a0,$a0,1
        jal     bar

return: lw      $t0,0($sp)
        addi    $sp,$sp,4
        jr      $t0


foo:    addi    $t0,$zero,0
        addi    $sp,$sp,-8
        sw      $ra,4($sp)
        sw      $t0,0($sp)

        addi    $a1,$sp,0
        addi    $sp,$sp,-4
        sw      $a0,0($sp)
        jal     bar
        lw      $a0,0($sp)
        addi    $sp,$sp,4

        lw      $t0,0($sp)
while:  add     $t1,$a0,$t0
        lb      $t2,0($t1)
        sb      $t2,0($a0)
        beq     $t2,$zero,exit
        addi    $a0,$a0,1
        j       while

exit:   lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra
```

The assembly code consists of the two functions `foo` and `bar`. Suppose that the following C program is compiled and linked together with the assembly code

```
#include "stdio.h"
char s1[] = "␣␣␣What␣is␣this?␣␣";

int main(){
  foo(s1);
  printf("[%s]",s1);
  return 0;
}
```

Note that the character

`"␣"`

in a C string expression means that there is a space character in the string.

Translate the two MIPS assembly functions `foo` and `bar` into well-formatted and readable C code. You must make use of pointers and pointer arithmetic, and not array indexing (`[]`). Make sure to use local variables as in the assembly code when variables are allocated on the stack. Non recursive and recursive function calls must be preserved.

Grading for IS1500 students: The question is graded in three levels S, G, and VG, with the following criteria:

- Satisfactory (S): Some minor parts of a C or Assembly program are constructed correctly, but there are major errors or missing parts of the program.
- Good (G): The majority of the program is constructed correctly, including the main flow and structure of the program, but there are a number of minor errors within the program.
- Very Good (VG): The program is correct, with basically no errors.

If none of the three levels is achieved, the exercise is considered as failed (F).

For IS1200 students (retake exam), max 20 points.

9. Consider again the assembly code given in exercise 8. Assume that all instructions are real MIPS machine instructions (no pseudo instructions). The following question is divided into three different sub-problems:

   - L1: Analyze functions `foo` and `bar`. Describe shortly what function `foo` is doing. Explain what the output will be for string `s1` after executing `foo(s1);`
   - L2: Suppose the functions are running on a 32-bit MIPS processor with separate instruction and data caches. In the data cache, the block size is 4 bytes, and the capacity 4096 bytes. In the instruction cache, the block size is 16 bytes and the capacity 8192 byte. Both caches are direct mapped. The `bar` label is located at address `0x4000000c` and the stack pointer points to address `0x7fff0018` before function `foo` is entered. Suppose the null-terminated string `s1` is declared using string `"KTH "` instead, and that the first byte of the string is located at address `0x8000fff0`. What are then the total number of instruction cache misses when executing the program? Note that the string has one space character after the text `KTH`. Assume that the cache is empty (all valid bits are zero) before entering function `foo`. This means that the first instruction that is executed is `addi $t0,$zero,0` in function `foo`, and the last instruction that is executed is `jr $ra`.
   - L3: Using the same setup as in (L2) what is then the data cache miss rate? Assume that a `lw` or a `sw` cache miss is always counted as one cache miss, and that executing a `lw` or a `sw` is always counted as one memory access.

   Grading for IS1500 students: The question is graded in three levels S, G, and VG, with the following criteria:

   - Satisfactory (S): The task at level L1 is solved correctly.
   - Good (G): Either the task at level L2 or the task at level L3 is solved correctly.
   - Very Good (VG): Both the tasks at level L2 and L3 are solved correctly.

   If none of the three levels is achieved, the exercise is considered as failed (F).

   For IS1200 students (retake exam), max 15 points. Sub-task L1 gives max 4 points, L2 max 5 points, and L3 max 6 points.

Del I: Grundläggande

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

   (a) Betrakta följande C-program.

```
#include "stdio.h"

int a[] = {7,10,3,5,12,8,17,-8,4,5};
int x[] = {8,2,0};

int sum(int* p1, int* p2, int len){
  int i = 0;
  int s = 0;
  while(i < len){
    -- MISSING CODE --
  }
  return s;
}

int main(){
  int v = sum(a,x,3);
  printf("Sum␣%d\n", v);
  return 0;
}
```

   Funktionen `sum` ska summera de värden som hittas med användning av pekaren `p1`. Pekaren `p2` pekar ut den array med index som används i summeringen. Det innebär att funktionen `sum` ska ta det första element som `p2` pekar på (i första iterationen av slingan är värdet 8). Funktionen använder detta indexvärde för att slå upp värdet på index 8 i arrayen som `p1` pekar på (i detta fall, värdet 4). Denna process fortsätter i tre sling-iterationer. I andra iterationen används index 2, och därför hittas värdet 3 med hjälp av `p1`. I sista iterationen används index 0 och därför värde 7[3]. Summan som returneras med hjälp av variabeln `s` ska alltså vara $4 + 3 + 7 = 14$. Alltså ska funktionen `main` skriva ut värdet 14.

   Mitt i funktionen `sum` saknas ett antal rader programkod. Skriv ner dessa rader programkod, så att programmet uppvisar det beteende som beskrivits här ovanför. Observera att du bara får använda pekararitmetik (ingen användning av array-indexering med `[]`). Du får inte referera arrayerna `a` och `x` direkt; alla referenser måste göras via pekarna `p1` och `p2`. (5 poäng)

   (b) Vad är maskinkoden för följande instruktion i 32-bitars MIPS-assembler? Svara med ett 32-bitars hexadecimalt tal. (3 poäng)

```
lb  $t3, -11($s2)
```

[3] 2020-01-21: denna uppgift har förtydligats då den svenska versionen inte överensstämde med den engelska versionen. Att så var fallet förtydligades dock under tentamen.

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

(a) Anta att du utvecklar en mycket enkel miniräknare, som kan utföra två olika uppgifter beroende på inställningen hos omkopplare nummer 4 på Chipkit Basic I/O Shield. Om omkopplaren är hög (bit-värdet för omkopplaren är 1) så ska miniräknaren utföra kvadratfunktionen (multiplicera det 3-bitars värde som ges av omkopplare 1 till 3 med sig självt). Om omkopplarvärdet för omkopplare 4 är noll, så ska det 3-bitarsvärdet fördubblas (adderas till sig självt).

På Basic I/O Shield är omkopplarna minnesmappade till ordadressen `0xbf8860d0` (bitarna 11 till och med 8) och de gröna lysdioderna till `0xbf886110` (bitarna 7 till och med 0). Ut-resultatet ska visas som ett binärt tal på lysdioderna. Omkopplare nummer 4 finns på bitnummer 11. Du kan anta att portarna är korrekt konfigurerade innan programmet startar (omkopplarna är konfigurerade som input och lysdioderna för output). Antag även att processorn exekverar utan hoppfördröjning (branch delay slots).

Miniräknaren implementeras som en oändlig slinga, med användning av programkoden i nedanstående mall.

```
loop:
        lui     $t0,0xbf88
        lw      $t1,0x60D0($t0)
        -- MISSING CODE: PART I --
do_add:
        add     $t1,$t2,$t2
next:
        -- MISSING CODE: PART II --
        j       loop
```

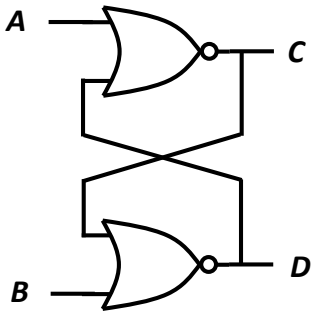
i. Skriv ner det kodavsnitt som saknas vid PART I. Denna del ska omfatta programkod för att extrahera rätt information från omkopplarna, utföra korrekt hopp, och beräkna kvadratfunktionen korrekt. Notera att fördubblingen (addition till sig själv) redan är implementerad i programkodsmallen (3 poäng)

ii. Skriv ner det kodasvnitt som saknas vid PART II. Denna del ska se till att endast de 8 bitar som avser lysdioderna ändras när resultatet matas ut till den minnesmappade in/ut-porten. Du kan anta att endast de 16 minst signifikanta bitarna av porten innehåller relevant data, dvs. du behöver inte bevara de 16 mest signifikanta bitarna. (3 poäng)

(b) Anta att du ska konfigurera en 32-bitars hårdvaru-timer med klockfrekvensen 4 MHz. Du har bestämt att periodregistret har värdet $500\,000$. Vilken skalfaktor (prescaling value) behöver då användas, om timern ska ha perioden 1s? (2 poäng)

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

   (a) Betrakta kretsen här nedanför:



   I de fall som beskrivs här nedanför, ange om signalerna är 0, 1 eller `okänd`, där det sistnämnda betyder att det inte går att avgöra ett bestämt värde utifrån den givna informationen.

   i. Om $A = 1$ och $B = 0$, vad är då värdena på C och D? (2 poäng)

   ii. Om $A = 0$ och $B = 0$, vad är då värdena på C och D? (2 poäng)

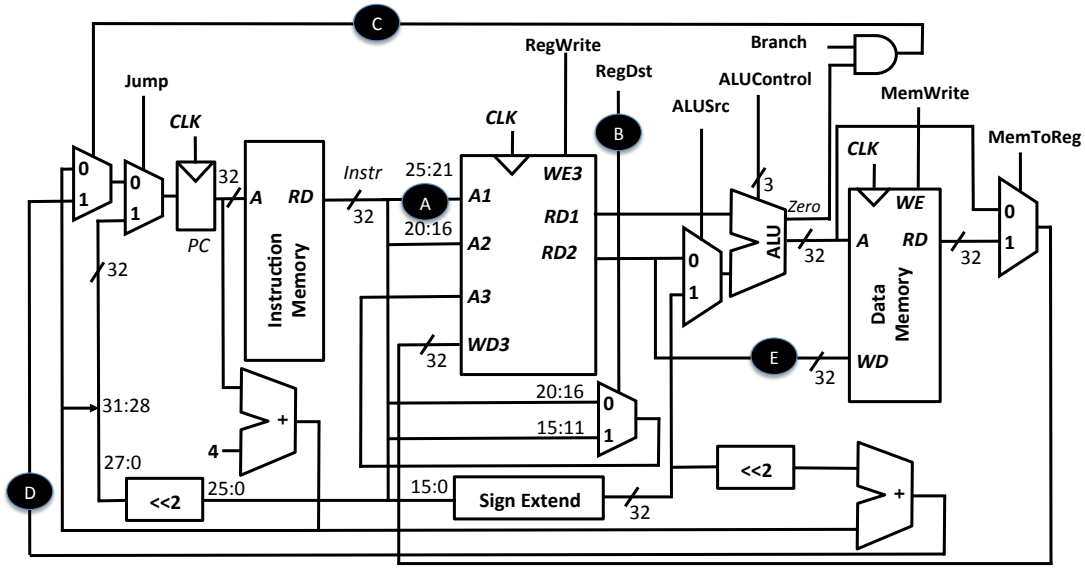
   iii. Vad är det officiella namnet på denna krets? (1 poäng)

   (b) Anta att du har en registeruppsättning (register file) med en läsport och en 16-bitars skrivport. Adressignalerna är 6 bitar breda. Hur många byte kan registeruppsättningen lagra totalt, det vill säga: vad är kapaciteten hos hela registeruppsättningen? (2 poäng)

   (c) Anta att du har en 2:4-avkodare. Om det decimala värdet 2 matas in, vad blir då utsignalerna Y_0, Y_1, Y_2 och Y_3? (1 poäng)

4. **Modul 4: Processorkonstruktion**

(a) Betrakta nedanstående dataväg för en 1-cykels, 32-bitars MIPS-processor.



Anta att instruktionen `bne` exekveras i följande program

```
        xor     $t1,$t2,$t2
        bne     $t1,$zero,next
        addi    $t0,$t0,1
next:
```

Instruktionen `bne` finns på adress `0x40004008`. Vad är då signalvärdena för A, B, C, D och E när `bne`-instruktionen exekveras? Svara med antingen hexadecimala tal, eller skriv `okänd` för ett signalvärde där det inte går att avgöra ett exakt värde utifrån den givna informationen, eller där det exakta värdet på signalen inte spelar någon roll. (5 poäng)

(b) Betrakta nedanstående assemblerkod som exekveras på en 32-bitars MIPS-processor med 5-stegs pipeline.

```
        addi    $t1,$t1,10      # 1: F D E M W
        addi    $t2,$t2,0       # 2:   F D E M W
loop:   lb      $t0,0($s0)      # 3:     F D E M W
        add     $t2,$t0,$t2     # 4:       F D E M W
        addi    $t1,$t1,-1      # 5:         F D E M W
        addi    $s0,$s0,1       # 6:           F D E M W
        bne     $t1,$zero,loop  # 7:             F D E M W
                                #                 F D E M W
```

i. Räkna upp alla data-hazarder som finns. För varje data-hazard, ange mellan vilka två instruktioner hazarden inträffar. Anta att likhetskontrollen vid hopp görs i avkodningssteget (decode). Som exempel, om det finns en data-hazard mellan de två första `addi`-instruktionerna, så skulle svaret för den hazarden vara 1-2 (mellan instruktion 1 och instruktion 2). (2 poäng)

ii. Vilka av ovanstående hazarder kräver stalling? (1 poäng)

5. **Modul 5: Minneshierarkier**

   (a) Anta att vi har en 32-bitars MIPS-processor, med ett instruktionscacheminne med storleken (capacity) 1024 byte. Adressetiketten (tag field) är 23 bitar stor och blockstorleken är 8 byte.

   - Hur stort är byte-offset-fältet? (1 poäng)
   - Hur stort är set-fältet (set field)? (1 poäng)
   - Vad är cacheminnets associativitetsvärde? (1 poäng)
   - Hur många giltigbitar finns i cacheminnet? (1 poäng)

   (b) Anta att 6 instruktioner exekveras i följd på en 32-bitars MIPS-processor. Den första och sjätte instruktionen är `lw`-instruktioner. De övriga instruktionerna är `add` och `sll`. Den första `lw`-instruktionen finns på adress `0x40001004`. Anta att processorn har separata data- och instruktionscacheminnen, där vart och ett av cacheminnena har storleken (capacity) 4 KiB (4096 byte). Båda cacheminnena är direktmappade. Instruktionscacheminnet har en blockstorlek på 8 byte, och datacacheminnet har en blockstorlek på 16 byte. Den första `lw`-instruktionen läser från adress `0x80002000`, och den andra `lw`-instruktionen läser från adress `0x80002010`. Anta att alla giltigbitar i de båda cacheminnena är noll innan kodavsnittet exekveras.

   i. Vad är instruktionscacheminnets misskvot (miss rate)? (2 poäng)
   ii. Vad är datacacheminnets misskvot (miss rate)? (1 poäng)
   iii. Uppvisar instruktionshämtningarna tidslokalitet (temporal locality), rumslokalitet (spatial locality) eller både och? (1 poäng)

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

   (a) Anta att du utvecklat ett program som tar 6 sekunder att exekvera på en processor med 1 kärna. Om du exekverar programmet på en processor med 4 kärnor, så exekveras programmet på 3 sekunder. Vilken är den teoretiska kortaste möjliga exekveringstiden för programmet, om du skulle ha tillgång till oändligt många kärnor? (3 poäng)

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

      i. Hårdvaruflertrådighet (hardware multithreading) används vanligtvis för att dölja fördröjningar (latencies).

      ii. SIMD (Single Instruction, Multiple Data) motsvarar begreppet trådparallellitet (thread-level parallelism).

      iii. Advanced Vector Extension (AVX) är en standard för att hantera instruktionsnivåparallellitet (Instruction-Level Parallelism, ILP).

      iv. Ett sätt att hantera problemet med cachekoherens (the cache coherency problem) är att använda snooping-protokollet (the snooping protocol).

      v. MapReduce är ett effektivt sätt att hantera VLIW (Very Long Instruction Word).

      (5 poäng)

Del II: Avancerat

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

   (a) Dataparallellitet (data-level parallelism), jämfört med flerkärnighet (multicore)

   (b) Multitrådning i hårdvara (hardware multithreading), jämfört med Advanced Vector Extension (AVX)

   (c) Multiple issue, jämfört med parallellitet på uppgiftsnivå (task-level parallelism)

   För studenter som tenterar IS1500: frågan betygsätts i tre nivåer S, G och VG med följande kriterier:

   - Tillfredsställande (Satisfactory, S): Vissa av begreppen i frågan är tydligt förklarade.
   - Bra (Good, G): I grund och botten är alla begrepp i frågan tydligt förklarade, och vissa av begreppen har relaterats till varandra genom att likheter och skillnader har diskuterats.
   - Mycket bra (Very Good, VG): I grund och botten är alla begrepp i frågan tydligt förklarade, och alla begrepp har relaterats till varandra genom att likheter och skillnader har diskuterats.

   Om ingen av de tre nivåerna har uppnåtts, betraktas uppgiften som underkänd (Failed, F).

   För studenter som tenterar IS1200 (omtentamen), högst 15 poäng. För var och en av de tre deluppgifterna ges högst tre poäng för förklaringarna av de två begreppen, högst en poäng för minst en skillnad och högst en poäng för minst en likhet. Det betyder högst fem poäng för var och en av de tre deluppgifterna.

8. Betrakta nedanstående 32-bitars MIPS-assemblerkod:

```
bar:    addi    $sp,$sp,-4
        sw      $ra,0($sp)

        lb      $t0,0($a0)
        beq     $t0,$zero,return

        bne     $t0,32,return       # 32 = ascii code for space
        lw      $t0,0($a1)
        addi    $t0,$t0,1
        sw      $t0,0($a1)

        addi    $a0,$a0,1
        jal     bar

return: lw      $t0,0($sp)
        addi    $sp,$sp,4
        jr      $t0


foo:    addi    $t0,$zero,0
        addi    $sp,$sp,-8
        sw      $ra,4($sp)
        sw      $t0,0($sp)

        addi    $a1,$sp,0
        addi    $sp,$sp,-4
        sw      $a0,0($sp)
        jal     bar
        lw      $a0,0($sp)
        addi    $sp,$sp,4

        lw      $t0,0($sp)
while:  add     $t1,$a0,$t0
        lb      $t2,0($t1)
        sb      $t2,0($a0)
        beq     $t2,$zero,exit
        addi    $a0,$a0,1
        j       while

exit:   lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra
```

Assemblerkoden består av de två funktionerna `foo` och `bar`. Anta att följande C-program kompileras och länkas tillsammans med assemblerkoden.

```
#include "stdio.h"
char s1[] = "␣␣␣What␣is␣this?␣␣";

int main(){
  foo(s1);
  printf("[%s]",s1);
  return 0;
}
```

Observera att tecknet

`"␣"`

i ett stränguttryck i C betyder att det finns ett mellanslagstecken (space character) i strängen.

Översätt de två funktionerna `foo` och `bar` från MIPS-assembler till välformatterad och läslig C-kod. Du måste använda pekare och pekararitmetik, och inte arrayindexering (`[]`). Se till att använda lokala variabler, som i assemblerkoden när variabler allokeras på stacken. Icke-rekursiv och rekursiva funktionsanrop måste bevaras.

Betygsättning för studenter som tenterar IS1500: frågan betygsätts i tre nivåer S, G och VG med följande kriterier:

- Tillfredsställande (Satisfactory, S): Vissa mindre delar av ett program i C eller assembler har konstruerats korrekt, men det finns antingen stora fel i programmet eller stora delar som saknas.
- Bra (Good, G): Huvuddelen av programmet har konstruerats korrekt, inklusive huvudflödet och strukturen i programmet, men det finns ett antal mindre fel i programmet.
- Mycket bra (Very Good, VG): Programmet är korrekt och har i grund och botten inga felaktigheter.

Om ingen av de tre nivåerna har uppnåtts, betraktas uppgiften som underkänd (Failed, F).

För studenter som tenterar IS1200 (omtentamen), högst 20 poäng.

9. Betrakta återigen assemblerkoden som visas i uppgift 8. Anta att alla instruktioner är riktiga MIPS-maskininstruktioner (inga pseudoinstruktioner). Följande fråga är uppdelad i tre olika delproblem:

   - L1: Analysera funktionerna `foo` och `bar`. Beskriv kort vad funktionen `foo` gör. Förklara vad som matas ut (output) för strängen `s1` efter att `foo(s1);` har exekverats.
   - L2: Anta att funktionerna kör på en 32-bitars MIPS-processor med separata cacheminnen för instruktioner och data. I datacacheminnet är blockstorleken 4 byte, och storleken (capacity) 4096 byte. I instruktionscacheminnet är blockstorleken 16 byte och storleken (capacity) 8192 byte. Båda cacheminnena är direktmappade. Läget `bar` är placerat på adressen `0x4000000c` och stackpekaren (the stack pointer) pekar på adress `0x7fff0018` innan programkörningen går in i funktionen `foo`. Anta att den null-terminerade strängen `s1` har deklarerats med användning av strängen `"KTH "` i stället, och att den första byten av strängen har placerats på adress `0x8000fff0`. Vad blir då det totala antalet missar i instruktionscacheminnet när programmet exekveras? Observera att strängen har ett mellanslagstecken (space character) efter texten `KTH`. Anta att cacheminnet är tomt (så att alla giltigbitar är noll) innan programkörningen går in i funktionen `foo`. Detta betyder att den första instruktionen som exekveras är `addi $t0,$zero,0` i funktionen `foo`, och den sista instruktionen som exekveras är `jr $ra`.
   - L3: Med samma egenskaper som i L2, vad blir misskvoten i datacacheminnet? Anta att en `lw` eller en `sw` cache miss alltid räknas som en cache miss och att exekveringen av en `lw` eller en `sw` alltid räknas som en minnesaccess.

   Betygsättning för studenter som tenterar IS1500: frågan betygsätts i tre nivåer S, G och VG med följande kriterier:

   - Tillfredsställande (Satisfactory, S): Uppgiften på nivå L1 har lösts korrekt.
   - Bra (Good, G): Antingen uppgiften på nivå L2 eller uppgiften på nivå L3 har lösts korrekt.
   - Mycket bra (Very Good, VG): Både uppgiften på nivå L2 och uppgiften på nivå L3 har lösts korrekt.

   Om ingen av de tre nivåerna har uppnåtts, betraktas uppgiften som underkänd (Failed, F).

   För studenter som tenterar IS1200 (omtentamen), högst 15 poäng. Delen L1 ger högst 4 poäng, L2 högst 5 poäng och L3 högst 6 poäng.

MIPS Reference Sheet

David Broman, KTH Royal Institute of Technology
Version 1.16, December 21, 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);

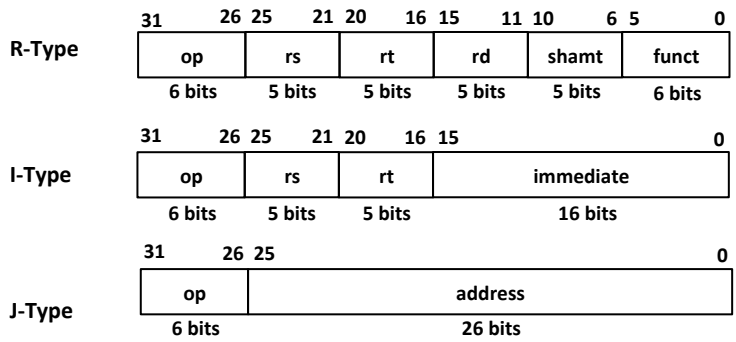
PSEUDO INSTRUCTIONS (SUBSET)

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.