# 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**

2020-06-05

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.
- **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 question, 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):

- 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

**Results**

The result will be announced at latest 2020-06-26. 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 2020-07-17 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).
- **Del II: Avancerade delen:** Det finns totalt tre avancerade frågor där det krävs att du diskuterar, analyserar och konstruerar. Vidare krävs motiveringar. Varje fråga 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 2020-06-26. Vid FX är det möjligt att begära en komplementerande examination. Examinatorn bestämmer om examinationen är muntlig eller skriftlig. Studenten måste begära examination senast 2020-07-17 via epost till dbro@kth.se.

# Part I: Fundamentals

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

   (a) Consider the following C code

```
void something(int* foo, int len, int v)
{
  int i;
  for(i=0; i<len; i++){
    -- missing statement #1 --
    -- missing statement #2 --
  }
}
```

   Function `something` has three function parameters: a pointer `foo` that points to an array of integers, a parameter `len` that specifies the length of the array, and a value `v`. The function should iterate over the array and multiply each element in the array with value `v`. For instance, if function `something` is called as follows:

```
int k1[] = {2,2,3,3,10};
something(k1, 5, 4);
```

   the array will be updated and contain values:

```
{8,8,12,12,40}
```

   In the C function, there are two statements missing. Please write down these two statements. Statement #1 should multiply value `v` to the array element that pointer `foo` points to. Statement #2 should increment the pointer correctly. None of these two statements are allowed to use variable `i`. It is only allowed to use pointer arithmetic. It is not allowed to use array syntax (`[]`). (4 points)

   (b) Consider the following MIPS assembly code.

```
        addi    $t3,$zero,220
repeat:
        addi    $t3,$t3,-2
        add     $s3,$s3,$s1
        bne     $t3,$zero,repeat
```

   Write down the machine code for the `bne` instruction given in the code above. Answer as a 32-bit hexadecimal number.
   (4 points)

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

   (a) Consider a 16-bit timer that has seven prescaling options: `1:128`, `1:64`, `1:32`, `1:8`, `1:4`, `1:2`, and `1:1`. The timer is clocked at a frequency of 20 MHz. Assume further that the timer is configured to trigger an interrupt. Decide on a value for the period register and a prescale value such that the interrupt is triggered every 10 ms. (3 points)

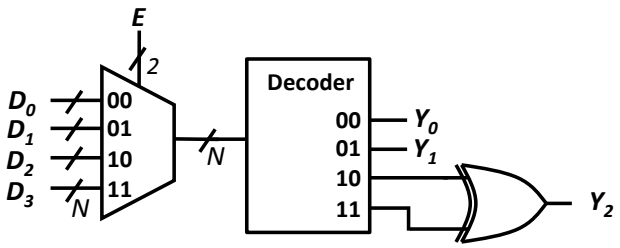
   (b) Consider the following C code that writes and reads to and from memory mapped I/O.

```
volatile int* leds = (volatile int*) 0x1dbb8050;
volatile int* switches = (volatile int*) 0x22430030;
*leds = (*leds & 0xff9f) | ((((*switches) >> 4) & 0x3) << 3);
```

   Translate the complete C code into MIPS assembly code. You are not allowed to use pseudo instructions. Note that a partial solution may give certain points. (5 points)

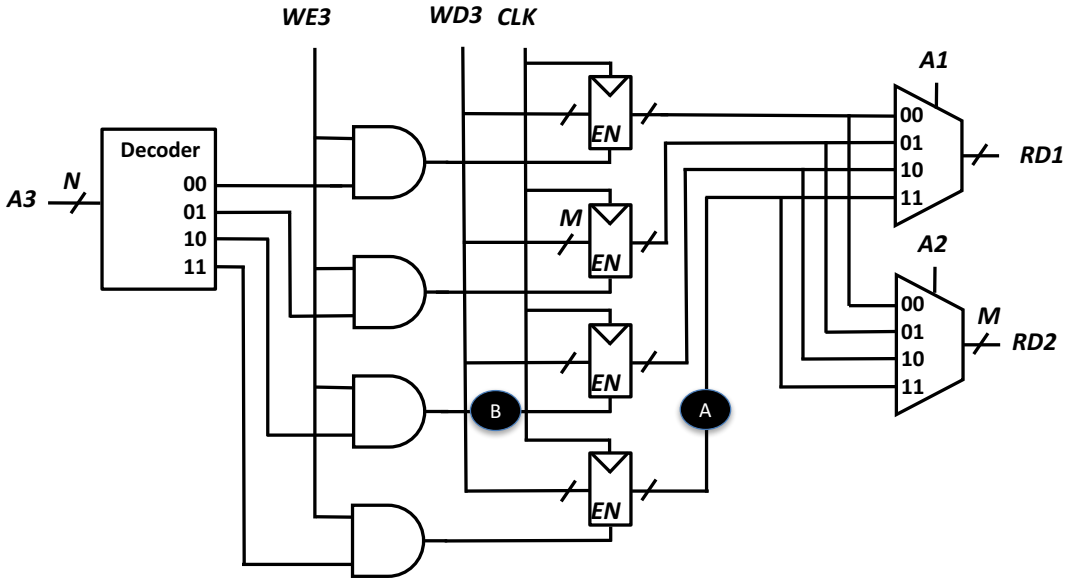
3. **Module 3: Logic Design (for IS1500 only)**

(a) Consider the combinational logic diagram below. Assume that $D_0 = 2$, $D_1 = 3$, $D_2 = 0$, $D_3 = 1$, and $E = 3$. What are the values of signals $Y_0$, $Y_1$, $Y_2$ and what is the size of the bit bus $N$? (2 points)




(b) Assume that a register file has two read ports and two write ports. The data input and output signals have a bus bit width of 32 bits. The address signals are 10 bits wide. In total, how many bytes of data can this register file store? (2 points)

(c) The following digital circuit implements a simple register file.




For each of the questions below, answer with either a number or state `unknown` if it is not possible to give a specific number with the given information. In all cases, you can assume the signals have stabilized.

i. Suppose the register file can store 128 bits in total. What is then the value of $M$?

ii. Suppose *A3* $= 2$, *WE3* $= 0$, *WD3* $= 3$, and *CLK* $= 1$ at a specific point in time. What is then the value of $A$?

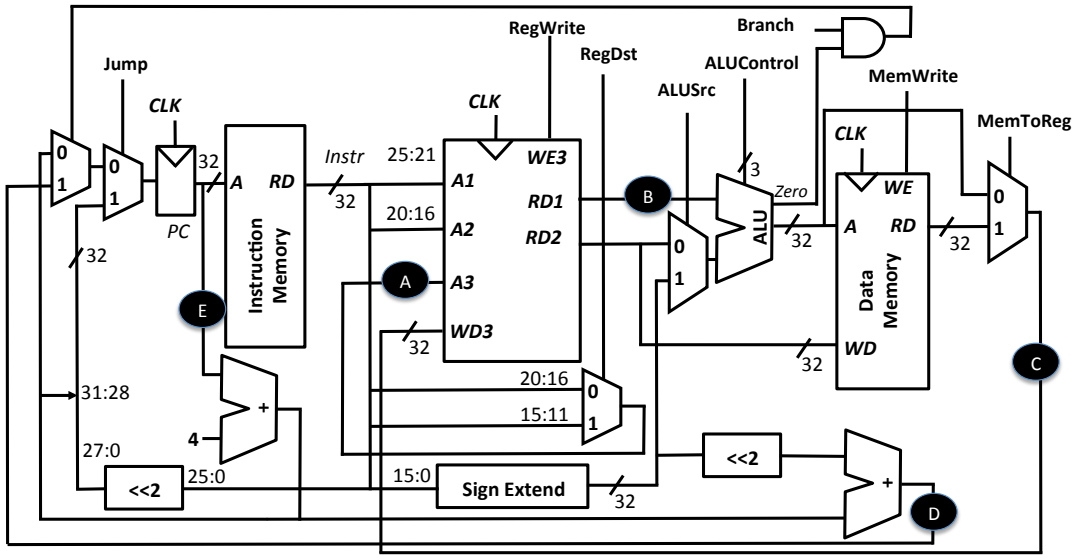
iii. Suppose *A3* $= 2$ and *WE3* $= 1$. What is then the value of signal $B$?

iv. Suppose *WD3* $= 40$, *CLK* $= 0$, *A2* $= 3$, *A3* $= 1$ and all registers hold value 20 in their memories. The register memories are triggered on a rising edge. What is then the value of *RD2* after that the clock switched to *CLK* $= 1$?

(4 points)

4. **Module 4: Processor Design**

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




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

```
lui     $t2,0x3f
ori     $t2,$t2,0x11
lw      $t8,4($t2)
```

Instruction `lui` is located at address `0x40004044`. What are then the signal values for $A$, $B$, $C$, $D$, and $E$ when the `lw` instruction is executed? For each signal, answer with either an integer number or write `unknown` for a signal value where it is not possible to determine an exact value with the given information. (5 points)

(b) Assume that the following code is executed on a 32-bit MIPS processor with a 5-stage pipeline.

```
        addi    $t1,$zero,0x7821
        addi    $t2,$zero,0xab21
        addi    $t0,$zero,0x18
foo:
        lw      $t0,0x1fb0($zero)
        beq     $zero,$t1,foo
```

How many control hazards and how many data hazards exist in the code above? Which register or registers are involved in this hazard or these hazards?
(3 points)

5. **Module 5: Memory Hierarchy**

(a) Assume that we have a processor with a 4-way set associative cache that uses 32-bit addresses. The capacity of the cache is 4096 bytes. The cache block size is 16 bytes. How many bits of the address is then used for the tag field, the set field (also called the index), and the byte offset field? Finally, how many valid bits exist in the cache? (4 points)

(b) Suppose we execute the following code on a 32-bit MIPS processor with direct-mapped caches.

```
   lui   $t0,0xa020
   ori   $t0,$t0,0x1f20
   addi  $t1,$zero,6
   addi  $t2,$zero,0
   addi  $t3,$zero,0
loop:
   lw    $t2,8($t0)
   add   $t3,$t3,$t2
   addi  $t0,$t0,4
   addi  $t1,$t1,-1
   bne   $t1,$zero,loop
```

The processor has separate data and instruction caches. The instruction cache has a capacity of $2048$ bytes and there are $256$ blocks in the instruction cache. The data cache has the capacity of $1024$ bytes and the block size in the data cache is $16$ bytes. The `lui` instruction is located at memory address `0x002c0110`. We assume that all valid bits are zero in both caches before executing the code.

i. What is the data cache miss rate? (2 points)

ii. Does the use of the data cache show temporal locality, spatial locality, or both? (1 point)

iii. Does the use of the instruction cache show temporal locality, spatial locality, or both? (1 point)

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

(a) Suppose you would like to estimate the speedup for a program when it executes on a processor with $32$ cores. Unfortunately, you only have access to a computer with $6$ cores. When you run it on the $6$-core machine, you get a speedup of $4$, compared to if you ran it on only one core. Compute what the speedup would be if you run the program on a machine with $32$ cores. Give an exact answer, as a fractional number. (3 points)

(b) For each of the following five 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. (5 points)

i. Register renaming is a way to solve control hazards in single-cycle processors.

ii. Hyper-threading is Intel's name for simultaneous multithreading (SMT).

iii. AVX (Advanced Vector Extensions) can be applicable in an application that has data-level parallelism.

iv. Software threads are typically used when implementing SIMD (Single instruction, multiple data) instructions.

v. Semaphores can be used to synchronize software threads.

(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) Advanced Vector Extension (AVX) vs. multiple issue

   (b) Hardware multi-threading vs. task-level parallelism

   (c) Instruction-level parallelism vs. multicore

   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).

8. Consider the following C code:

```
#include <stdio.h>

const char* text1 = "This is a text.";
const char* text2 = "Three lines \n and \n  numbers 8812 121.";
```

First, you should create a function called `count` that counts the number of characters and words in one of the strings. Its first parameter is a pointer to the input string. Its second parameter is a pointer to an integer variable. After that the function has finished, the variable that the second parameter points to should contain the number of counted characters that are not white spaces (newline, space, or a tab). The return value should be an integer that is the number of words in the string. Words are separated by white space. Only pointer arithmetic is allowed, i.e. array indexing is not allowed.

You must also write down a `main` function. This function shall call function `count` two times, each time pointing to one of the two strings `text1` and `text2`. As a result of these two function calls, the main function prints out the number of words for the two strings, as well as the total number of characters in the two strings.

Hence, if you run the program, it should output the following:

```
Words in text1: 4
Words in text2: 6
Total chars: 40
```

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).

9. Consider the following MIPS assembly code:

```
.data
.align 2
numbers: .space 40


.align 2
.text

start:
        la      $a0,numbers
        addi    $a1, $0, 10
        jal     foo

foo:
        addi    $sp,$sp,-4
        sw      $a0,0($sp)
loop:
        sw      $a1, 0($a0)
        addi    $a0, $a0, 4
        addi    $a1, $a1, -1
        bne     $a1, $0, loop

        lw      $t3,0($sp)
        addi    $sp,$sp,4
        lw      $t0,0($t3)
        lw      $t1,4($t3)
        add     $v0, $t0, $t1
        jr      $ra
```

The following question is divided into three different sub-problems:

- L1: The program above shall write 10 integers to the memory space allocated at label `numbers`. But, it does not terminate and it writes outside the allocated 40 bytes. Give a detailed explanation of why.
- L2: Suppose the code is running on a MIPS processor with a 2-way instruction cache. The capacity of the cache is 4096 bytes and it has 128 sets. Label `foo` is located at address `0x40001a0c`. Assume that all valid bits are zero in the cache before the code is executed. How many instruction cache misses occur in the above program? Note that you should also include the number of cache misses for the erroneus part of the execution.
- L3: Suppose that the assembly code is executing on a 32-bit MIPS processor with a 5-stage pipeline that supports stalling but not forwarding. The comparison for branch instructions are done in the execute stage. There are no branch delay slots. Note that the `la` instruction is a pseudo instruction and consists of two real instructions. Count the number of clock cycles starting with the fetch cycle of the `addi` instruction after the `foo` label, and ending with (including) the fetch cycle of the `jr` instruction. Do not include any cache aspects in this sub-problem. You need to document all assumptions you make when solving exercise.

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).

# Del I: Grundläggande

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

   (a) Betrakta nedanstående C-kod.

```
void something(int* foo, int len, int v)
{
  int i;
  for(i=0; i<len; i++){
    -- missing statement #1 --
    -- missing statement #2 --
  }
}
```

   Funktionen `something` har tre parametrar. Första parametern `foo` är en pekare till en heltalsvektor (array of int). Andra parametern `len` anger antalet element i heltalsvektorn. Tredje parametern är ett heltalvärde `v`. Funktionen ska iterera över heltalsvektorn och multiplicera `v` med vart och ett av vektorelementen. Exempel: Om `something` anropas såhär:

```
int k1[] = {2,2,3,3,10};
something(k1, 5, 4);
```

   så kommer heltalsvektorn `k1` efteråt att innehålla

```
{8,8,12,12,40}
```

   I C-funktionen saknas två satser, och din uppgift är att skriva dem. Satsen vid ”missing statement #1” ska multiplicera värdet `v` med det vektorelement som pekas ut av pekarvariabeln `foo`. Satsen vid ”missing statement #2” ska öka pekaren så att den har rätt värde till nästa iteration av for-slingan. Observera att ingen av de två satserna får använda variabeln `i`. Det är endast tillåtet att använda pekararitmetik. Det är inte tillåtet att använda array-syntax (`[]`) (4 poäng)

   (b) Betrakta följande assemblerkod för MIPS.

```
        addi    $t3,$zero,220
repeat:
        addi    $t3,$t3,-2
        add     $s3,$s3,$s1
        bne     $t3,$zero,repeat
```

   Skriv ner maskinkoden för `bne`-instruktionen som ges i koden här ovanför. Svara med ett 32-bitars hexadecimalt tal.
   (4 poäng)

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

   (a) Anta att du har en 16-bitars timer med sju möjliga skalfaktorer: `1:128`, `1:64`, `1:32`, `1:8`, `1:4`, `1:2` och `1:1`. Timern klockas med frekvensen 20 MHz. Anta vidare att timern är konfigurerad så att den ger avbrott. Välj ett värde för perioden och ett värde för skalfaktorn, så att avbrott ges med 10 ms mellanrum. (3 poäng)

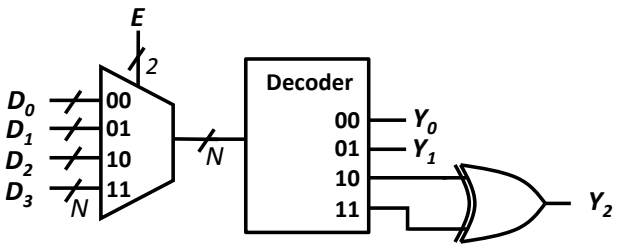
   (b) Betrakta följande C-kod som skriver till och läser från minnesmappade in- och utenheter.

```
volatile int* leds = (volatile int*) 0x1dbb8050;
volatile int* switches = (volatile int*) 0x22430030;
*leds = (*leds & 0xff9f) | ((((*switches) >> 4) & 0x3) << 3);
```

   Översätt den kompletta C-koden här ovanför till MIPS-assemblerkod. Du får inte använda pseudoinstruktioner. Observera att en ofullständig lösning kan ge vissa poäng. (5 poäng)

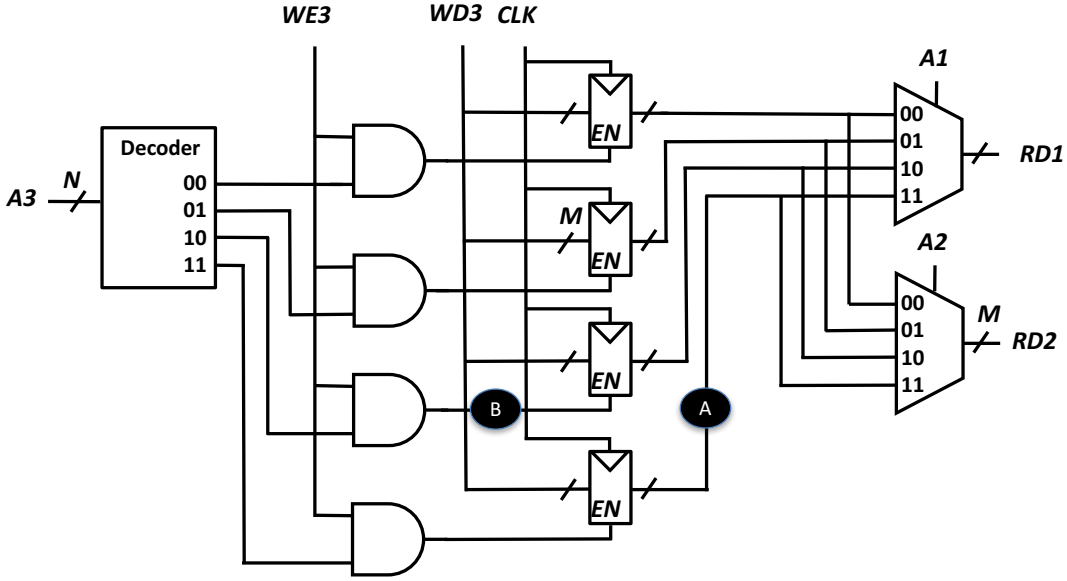
3. **Modul 3: Digital Design (endast för IS1500)**

(a) Betrakta logikschemat här nedanför. Anta att $D_0 = 2$, $D_1 = 3$, $D_2 = 0$, $D_3 = 1$, $E = 3$. Vilka värden får signalerna $Y_0$, $Y_1$ och $Y_2$ och hur många bitar bred är bussen $N$? (2 poäng)




(b) Anta att en registeruppsättning (register file) har två läsportar och två skrivportar. Dataingångarna och utsignalerna har en bussbredd på 32 bitar. Adressignalerna är 10 bitar breda. Hur många byte data totalt kan lagras i registeruppsättningen? (2 poäng)

(c) Följande digitala krets implementerar en enkel registerfil.




För var och en av frågorna nedan, svara med antingen ett tal eller ange `okänt` om det inte går att ange ett specifikt tal utifrån den givna informationen. I samtliga fall kan du anta att signalerna har stabiliserats.

i. Anta att registerfilen kan lagra totalt 128 bitar. Vad är då värdet på $M$?

ii. Anta att *A3* $= 2$, *WE3* $= 0$, *WD3* $= 3$ och *CLK* $= 1$ vid ett visst tillfälle. Vad är då värdet på $A$?

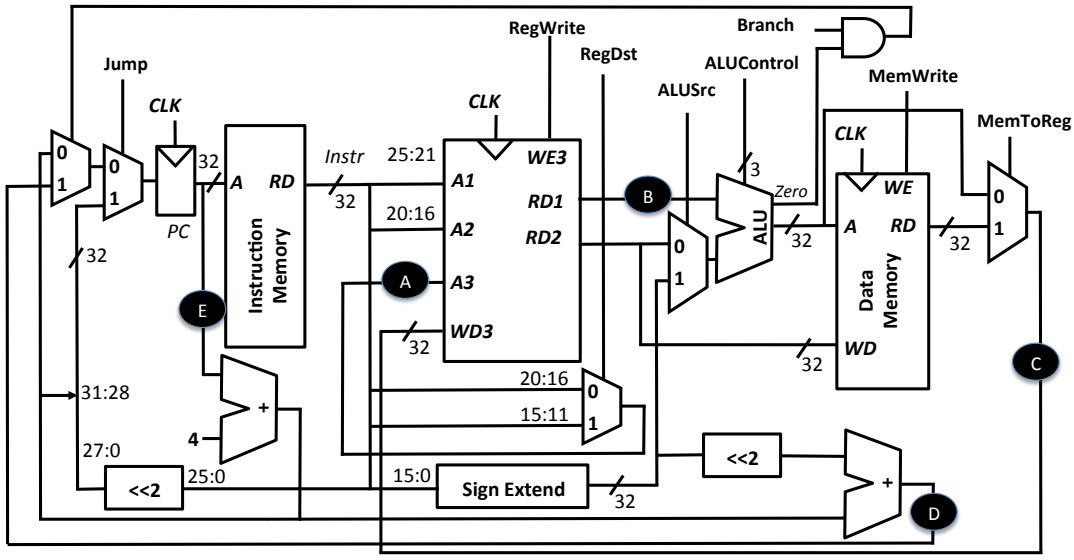
iii. Anta att *A3* $= 2$ och *WE3* $= 1$. Vad är då värdet på signal $B$?

iv. Anta att *WD3* $= 40$, *CLK* $= 0$, *A2* $= 3$, *A3* $= 1$ och att alla register innehåller värdet 20 i sina minnen. Registrens minnen triggas på positiv flank. Vad blir då värdet på *RD2* när clockan har ändrats till *CLK* $= 1$?

(4 poäng)

4. **Modul 4: Processorkonstruktion**

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




Anta att instruktionen `lw` exekveras i följande program:

```
        lui     $t2,0x3f
        ori     $t2,$t2,0x11
        lw      $t8,4($t2)
```

Instruktionen `lui` finns på adress `0x40004044`. Vad är då signalvärdena för $A$, $B$, $C$, $D$ och $E$ när `lw`-instruktionen exekveras? För varje signal, svara med antingen ett heltal eller skriv `okänd` för ett signalvärde där det inte går att avgöra ett exakt värde utifrån den givna informationen. (5 poäng)

(b) Anta att följande kod exekveras på en 32-bitars MIPS-processor med 5-stegs pipeline.

```
        addi    $t1,$zero,0x7821
        addi    $t2,$zero,0xab21
        addi    $t0,$zero,0x18
foo:
        lw      $t0,0x1fb0($zero)
        beq     $zero,$t1,foo
```

Hur många kontrollflödeshazarder (control hazards) och hur många datahazarder (data hazards) finns i koden ovan? Vilket eller vilka register är inblandade i denna hazard eller dessa hazarder?
(3 poäng)

5. **Modul 5: Minneshierarkier**

(a) Antag att vi har en processor med ett 4-vägs associativt cacheminne som använder 32-bitars adresser. Cacheminnets storlek är 4096 bytes och dess blockstorlek 16 bytes. Hur många adressbitar används då för vart och ett av fälten adressetikett (tag), radnummer (index, även kallat set field) och byteoffset? Slutligen, hur många giltigbitar (valid bits) finns i cacheminnet? (4 poäng)

(b) Anta att vi exekverar följande kod på en 32-bitars MIPS-processor med direktmappade cacheminnen.

```
   lui   $t0,0xa020
   ori   $t0,$t0,0x1f20
   addi  $t1,$zero,6
   addi  $t2,$zero,0
   addi  $t3,$zero,0
loop:
   lw    $t2,8($t0)
   add   $t3,$t3,$t2
   addi  $t0,$t0,4
   addi  $t1,$t1,-1
   bne   $t1,$zero,loop
```

Processorn har separata cacheminnen för instruktioner och data. Instruktionscacheminnet har en storlek (capacity) på $2048$ byte och det finns $256$ block i instruktionscacheminnet. Datacacheminnet har en storlek (capacity) på $1024$ byte och blockstorleken i datacacheminnet är $16$ byte. Instruktionen `lui` finns på minnesadress `0x002c0110`. Vi antar att alla giltigbitar är noll i båda fallen, innan koden exekveras.

i. Vad blir misskvoten (miss rate) i datacacheminnet? (2 poäng)

ii. Uppvisar användningen av datacacheminnet tidslokalitet (temporal lokalitet), rumslokalitet (spatial lokalitet), eller både och? (1 poäng)

iii. Uppvisar användningen av instruktionscacheminnet tidslokalitet (temporal lokalitet), rumslokalitet (spatial lokalitet), eller både och? (1 poäng)

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

   (a) Anta att du skulle vilka uppskatta vilken speedup ett program får om det körs på en processor med $32$ kärnor. Tyvärr har du bara tillgång till en dator med $6$ kärnor. När du kör programmet på maskinen med $6$ kärnor får du en speedup på $4$, jämfört med att köra programmet på bara en kärna. Beräkna vad speedup skulle bli om du körde programmet på en maskin med $32$ kärnor. Ge ett exakt svar, som ett bråktal. (3 poäng)

   (b) För var och en av följande fem påståenden, ange om påståendet är sant eller falskt. Om du hävdar att påståendet är falskt, svara kort varför påståendet är falskt (2 meningar). Du behöver inte ge motivering om påståendet är sant. (5 poäng)

      i. Registeromdöpning (renaming) är ett sätt att lösa kontrollhazarder för en encykelprocessor.
      ii. Hyper-threading är Intels namn för simultan multitrådning (simultaneous multithreading, SMT).
      iii. AVX (Advanced Vector Extensions) kan vara tillämpligt i en applikation som har datanivå (data-level) parallellism.
      iv. Mjukvarutrådar används vanligtvis när man implementerar SIMD-instruktioner (Single instruction, multiple data).
      v. Semaforer kan användas för att synkronisera mjukvarutrådar.

      (5 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) Advanced Vector Extension (AVX), jämfört med "multiple issue"

   (b) Hårdvaru-multitrådning jämfört med parallellitet på uppgiftsnivå (task-level parallelism)

   (c) Instruktionsnivåparallellitet (instruction-level parallelism) jämfört med flerkärnighet (multicore)

   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).

8. Betrakta följande C kod:

```
#include <stdio.h>

const char* text1 = "This is a text.";
const char* text2 = "Three lines \n and \n  numbers 8812 121.";
```

Först ska du skapa en funktion som heter `count`. Denna funktion räknar antalet tecken och antalet ord i en av strängarna. Funktionens första parameter är en pekare till inputsträngen. Dess andra parameter är en pekare till en heltalsvariabel. När funktionen avslutats ska heltalsvariablen som den andra parametern pekar på innehålla antalet räknade tecken som inte är white space (ny rad, mellanslag eller tab). Returvärdet ska vara ett heltal som är antalet ord i strängen. Ord separeras med white space. Endast pekararitmetik är tillåtet, dvs array-indexering är inte tillåtet.

Du ska även skriva ner en `main`-funktion. Denna funktionen ska anropa funktionen `count` två gånger. Varje gång den anropas ska den peka på en av de två strängarna `text1` och `text2`. Som ett resultat av dessa två funktionsanrop ska `main`-funktionen skriva ut antalet ord för de två strängarna, samt det totala antalet tecken i de två strängarna.

Således, om man exekverar programmet så ska följande skrivas ut.

```
Words in text1: 4
Words in text2: 6
Total chars: 40
```

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).

9. Betrakta följande MIPS-assembler kod.

```
.data
.align 2
numbers: .space 40


.align 2
.text

start:
        la      $a0,numbers
        addi    $a1, $0, 10
        jal     foo

foo:
        addi    $sp,$sp,-4
        sw      $a0,0($sp)
loop:
        sw      $a1, 0($a0)
        addi    $a0, $a0, 4
        addi    $a1, $a1, -1
        bne     $a1, $0, loop

        lw      $t3,0($sp)
        addi    $sp,$sp,4
        lw      $t0,0($t3)
        lw      $t1,4($t3)
        add     $v0, $t0, $t1
        jr      $ra
```

Följande fråga är uppdelad i tre olika del-problem:

- L1: Ovan program ska skriva 10 heltal till minnesarean som är allokerad vid labeln `numbers`. Men det terminerar inte och programmet skriver utanför de 40 bytes som är allokerade. Ge en detaljerad förklaring till varför.
- L2: Anta att koden körs på en MIPS-processor med en 2-vägs instruktionscache. Cachens kapacitet är 4096 bytes och den har 128 sets (rader). Label `foo` finns på adress `0x40001a0c`. Antag att alla giltigbitar är noll i cachen innan koden exekveras. Hur många instruktionscachemissar uppstår i ovan program? Notera att du även ska inkludera cachemissar för koden som utför den felaktiga exekveringen.
- L3: Antag att assemblerkoden exekverar på en 32-bitars MIPS-processor med en 5-stegs pipeline, vilken stödjer "stalling" men ej "forwarding". Jämförelse av hopp-instruktioner görs i exekveringssteget. Det finns ingen hoppfördröjning (branch delay slots). Notera att `la`-instruktionen är en pseudoinstruktion och består av två riktiga instruktioner. Räkna antalet klockcykler och starta med hämtningscykeln (fetch) av `addi`-instruktionen efter label `foo`. Avsluta med (och inkludera) hämtningscykeln (fetch) av `jr`-instruktionen. Inkludera inga cacheaspekter i detta del-problem. Du måste dokumentera alla antaganden du gör för att lösa uppgiften.

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).

# 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); |

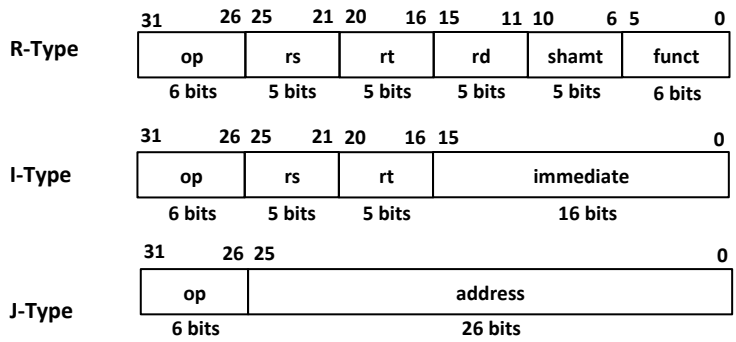
## PSEUDO INSTRUCTIONS (SUBSET)

| Name | Example | Equivalent Basic Instructions |
|---|---|---|
| load address | `la $t0,label` | `lui $at,hi-bits-of-address`<br>`ori $t0,$at,lower-bits-of-address` |
| load immediate | `li $t0,0xabcd1234` | `lui $at,0xabcd`<br>`ori $t0,$at,0x1234` |
| branch if less or equal | `ble $t0,$t1,label` | `slt $at,$t1,$t0`<br>`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.