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

2019-01-12

09.00-14.00

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 2019-02-02.
- 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 2019-02-23.

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 2019-02-02.
- 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 2019-02-23.

Part I: Fundamentals

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

   (a) Assume that a MIPS processor's registers contain the following values:

   ```
   $t0 = 0x6
   $t1 = 0x7
   $t2 = 0xaf
   $s0 = 0x23
   $s1 = 0x3
   $s2 = 0x1c
   ```

   Which of the registers changes value and what is the new value of this register after the following machine code has finished executing? (4 points)

   ```
   0x02285004
   ```

   (b) Assume that the following C code executes on a 32-bit MIPS processor.

   ```
   int a=3;
   int b=2;
   int lst[] = {6,13,19,-7,-4,18,4,23};
   int *p1 = &a;
   int *p2 = lst;

   printf("%x\n",(int)p2);

   b = *(p2 + *p1 * b) * a;
   *p1 = *p1 + p2[3];
   p2++;

   printf("%x, %d, %d, %d\n",(int)p2, a, b, *p2);
   ```

   The first print statement prints out the following line:

   ```
   56e11880
   ```

   What does the second print statement print out? (4 points)

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

   (a) Assume that four LED lights can be controlled by a general-purpose digital I/O (GPIO) at port `0xb8004d00`. Each of the LED lights can be turned on by writing bit value 1, and be turned off by writing bit value 0. The LED lights are controlled by writing to the bits at indices 9, 8, 7, and 6, where index 9 represents LED light number 4 and index 6 represents LED light number 1. Bit index 0 is the least significant bit. The port is already configured as an output port. When you write to the port, you do not need to preserve any of the other bits of the register.

   Write a function in MIPS assembler that is called `twinkle`. The assembly function needs to follow C calling conventions in such a way that it is possible to call this assembly function from the C programming language. The function should twinkle LED light number 3 (turn on or off). Inverting the bit in a register when twinkling must be done with the `xori` instruction. All other LED lights should be turned off. In the first iteration, the LED should be turned on. The `twinkle` function has one parameter (unsigned integer), which states how many times the LED light should be turned on or off. For instance, if the argument to the function is 6, then the function should turn on the LED light 3 times, and turn it off 3 times. After these 6 steps, the function should return.

   After each step (either turning on or turning off the LED light), a function `pause` should be called correctly. You can assume that this function already exists and that it does not destroy any `$t` registers. This function pauses the program for a short while, and returns the number of milliseconds that the program was paused. The `twinkle` function should add together the total number of milliseconds that the `twinkle` function has paused, and return this value.

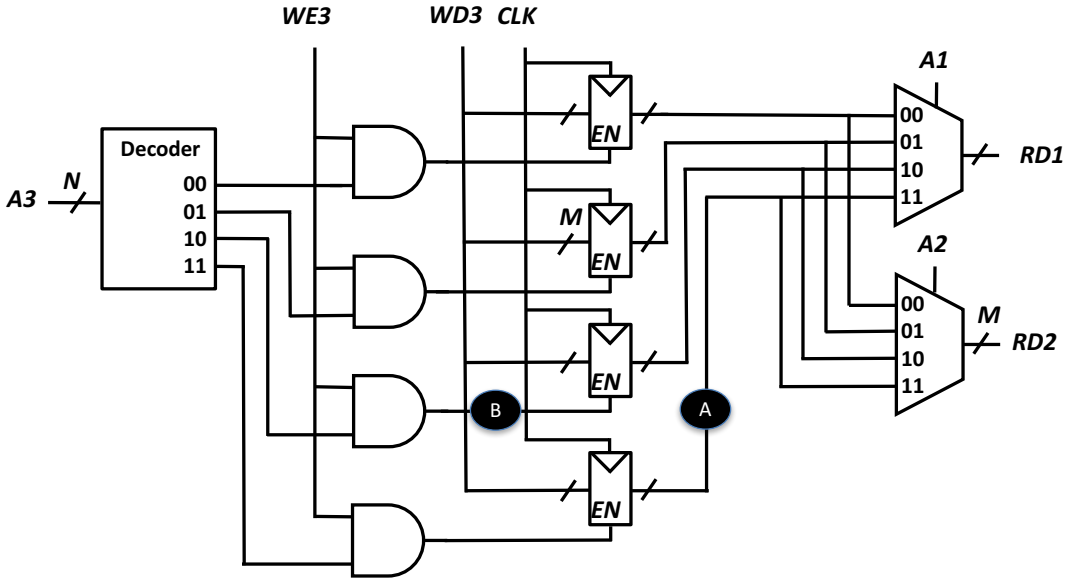
   You are not allowed to use any pseudo assembler instructions or stack memory. You may be given certain points even if the solution is not completely correct.

   (6 points)

   (b) Assume you are setting up a timer on an embedded system with a clock frequency of 100 MHz. The timer is using prescaling `1:4` and it is a 32-bit timer. What value should the period register have, if we want the timer to have a period of 3 seconds? (2 points)

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

(a) 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 256 bits in total. What is then the value of M?

ii. Suppose *A3* $= 3$, *WE3* $= 1$, *WD3* $= 7$, and *CLK* $= 0$ at a specific point in time. What is then the value of A?

iii. Suppose *A3* $= 3$ and *WE3* $= 1$. What is then the value of signal B?

iv. Suppose *WD3* $= 9$, *CLK* $= 0$, *A2* $= 2$, *A3* $= 3$ and all registers hold value 7 in their memories. The register memories are triggered on a raising edge. What is then the value of *RD2* after that the clock switched to *CLK* $= 1$?

(4 points)

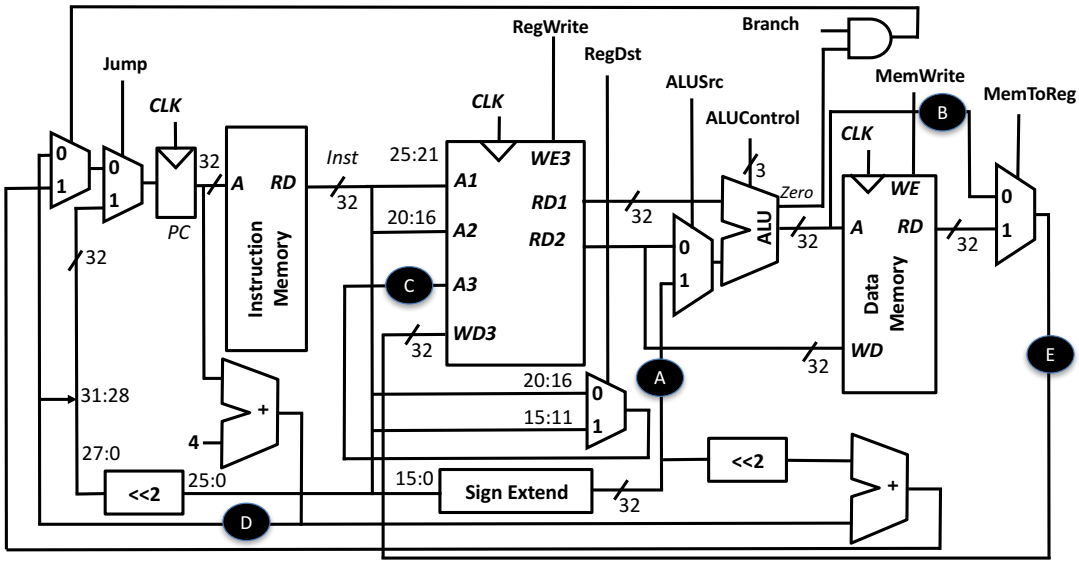
(b) Construct and draw a digital circuit that has three inputs A, B, and C, and one output R. Signals A, B, and R are 16-bit signals, representing numbers in two's complement form. Signal C is a binary signal. The circuit should fulfill the following specification: If $C = 0$ then $R = A + B$ and if $C = 1$ then $R = A - B$.

In your drawing, you are allowed to use the following components: 16-bit carry propagate adder, 2:1 multiplexer, 4:1 multiplexer, 2:4 decoder, inverter, AND-gate, and OR-gate. Note that you do not need to use all these components for a correct solution. You need to write out bus widths (in bits) of signals correctly (when it is not possible to infer).

(4 points)

4. **Module 4: Processor Design**

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



Suppose the first four lines of the following code have been executed

```
addi    $t0,$zero,0x3ff0
addi    $t0,$t0,16
sw      $t0,-8($t0)
addi    $t0,$t0,12
lw      $s2,-20($t0)
```

and that the processor is currently executing the `lw` instruction. The first `addi` instruction is located at memory address `0x00040210`. What are then the signal values for A, B, C, D, and E? 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.
(5 points)

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

```
    lw   $t0,0xff08($zero)
    beq  $t0,$zero,foo
    addi $t0,$zero,5
foo:
    addi $t0,$zero,10
```

Moreover, assume that the comparison for equality in the `beq` instruction is done in the decode stage and that it uses static predict branch not taken.

i. How many data hazards exist and how many control hazards exist?

ii. If there is one or more data hazards, which of the instructions are then involved?

iii. What is the maximal total penalty in clock cycles that can occur when executing the code, when the best solutions for handling the hazards are used?

(3 points)

5. **Module 5: Memory Hierarchy**

(a) Suppose we have a 32-bit MIPS processor with a 4-way set associative cache. The capacity of the cache is 4096 bytes and the block size is 8 bytes.

i. How many valid bits are there in the cache?

ii. What is the tag field size in bits?

(2 points)

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

```
   lui   $t0,0x3b
   ori   $t0,$t0,0x2210
   addi  $t1,$zero,5
   addi  $t2,$zero,0
   addi  $t3,$zero,0
loop:
   lw    $t2,4($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 4096 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 8 bytes. The `lui` instruction is located at memory address `0x00040100`. 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. What is the instruction cache miss rate? (2 points)

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

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

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

   (a) 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 (max two sentences). You do not need to give a motivation if the statement is true.

      i. Advanced Vector Extension (AVX) is designed for programs with data-level parallelism.

      ii. Hardware multithreading is a form of concurrency mechanism inside a process of an operating system.

      iii. Hyper-threading is another name for Very Long Instruction Words (VLIW).

      iv. Instruction-level parallelism is a parallelization approach that is available in almost all modern processors for laptops and workstations.

      v. Single Instruction Multiple Data (SIMD) is often used in applications with task-level parallelism.

      (5 points)

   (b) Suppose you are developing a software system and your manager asks you if it is possible to make it faster by running on a multicore machine. You only have access to a multicore machine with 4 cores, but the manager wants to know how fast it will be if you run the program on an 8 core machine. You know that you cannot give an exact answer without running it on an 8 core machine, but you also realize that you can use Amdahl's law to make an approximation. You make a measurement running on just one core, and the test program then takes 12 seconds to execute. You run it again on 4 cores, and the execution time is then 6 seconds. What is then the estimated execution time for 8 cores? (3 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) SIMD instructions vs. task-level parallelism

   (b) Processors with deep pipelines vs. instruction level parallelism (ILP)

   (c) caches vs. hardware multithreading

   (15 points)

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

```
foo:    addi    $sp,$sp,-8
        sw      $ra,4($sp)
        sw      $s3,0($sp)

        addi    $s3,$zero,0
        addi    $s4,$zero,5

loop:   slt     $t3,$s3,$s4
        beq     $t3,$zero,exit
        addi    $a1,$s3,0
        jal     boo
        addi    $a0,$v0,0
        addi    $s3,$s3,1
        j       loop

exit:   lw      $s3,0($sp)
        lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra


boo:    addi    $sp,$sp,-8
        sw      $ra,4($sp)
        sw      $s0,0($sp)

        addi    $t0,$zero,-1
        beq     $a0,$t0,ortrue

        sll     $t0,$a0,2
        add     $t0,$t0,$s2
        sll     $t1,$a1,2
        add     $t1,$t1,$s2
        lw      $t0,0($t0)
        lw      $t1,0($t1)
        slt     $t2,$t1,$t0
        beq     $t2,$zero,next

ortrue: sll     $t0,$a1,2
        add     $t0,$t0,$s1
        sw      $a0,0($t0)
        addi    $v0,$a1,0
        lw      $s0,0($sp)
        lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra

next:   sll     $t0,$a0,2
        add     $t0,$t0,$s1
        addi    $s0,$a0,0
        lw      $a0,0($t0)
        jal     boo
        sll     $t0,$s0,2
        add     $t0,$t0,$s1
        sw      $v0,0($t0)
```

```
        addi    $v0,$s0,0
        lw      $s0,0($sp)
        lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra
```

Suppose the following two arrays are defined in the C programming language:

```
int  plist[5];
int  vlist[5] = {7,100,2,35,1};
```

Also, assume that register `$s1` contains the address to the first element in array `plist`, and `$s2` contains the address to the first element in array `vlist`. Assume that register `$sp` contains an address to a memory area that is safe to read from and write to. Suppose now that the following C code is executed:

```
  int p = foo(-1);
  while(p != -1){
    printf("%d, ", vlist[p]);
    p = plist[p];
  }
}
```

(a) The assembly code defines two functions called `foo` and `boo`, where `foo` is called from the C code. Translate these two functions to clear and understandable C functions. Your C functions are not allowed to use pointers, only array indexing. To get full points, the C functions must be short, clean, and easy to understand. Each of the C functions should not need more than 10 lines of code. (15 points)

(b) State what is printed to standard output, and give a clear and pedagogical explanation of what happens when executing the program. Use diagrams and figures to make the explanation clear. The answer is expected to be detailed. (10 points)

9. Consider the MIPS code in exercise 8 again. Assume the code is executed on a 32-bit 5-staged MIPS processor with a direct mapped data cache. Assume further that the cache size is 32 KB and that the block size is 8 bytes. The array `plist` starts at address `0x00330000` and the array `vlist` is located directly after `plist` in memory. The `$sp` register contains address `0x02030000` each time `foo` is called. Assume that all valid bits in the cache are set to zero before function `foo` is called. The `boo` function is called several times from `foo`. When `boo` is called with the second argument equals to value 1, what is then the data cache miss rate when executing this call to `boo`?
(10 points)

Del I: Grundläggande

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

   (a) Anta att registren i en MIPS-processor innehåller följande värden:

   ```
   $t0 = 0x6
   $t1 = 0x7
   $t2 = 0xaf
   $s0 = 0x23
   $s1 = 0x3
   $s2 = 0x1c
   ```

   Vilket av registren ändrar innehåll, och vad blir det nya värdet i det registret när följande maskininstruktion har slutat exekveras? (4 poäng)

   ```
   0x02285004
   ```

   (b) Anta att följande C-programkod exekveras på en 32-bitars MIPS-processor.

   ```
   int a=3;
   int b=2;
   int lst[] = {6,13,19,-7,-4,18,4,23};
   int *p1 = &a;
   int *p2 = lst;

   printf("%x\n",(int)p2);

   b = *(p2 + *p1 * b) * a;
   *p1 = *p1 + p2[3];
   p2++;

   printf("%x, %d, %d, %d\n",(int)p2, a, b, *p2);
   ```

   Den första printf-satsen skriver ut följande rad:

   ```
   56e11880
   ```

   Vad skrivs ut av den andra printf-satsen? (4 poäng)

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

   (a) Anta att fyra LED-lampor kan styras med en digital in/ut-port (GPIO, General Purpose I/O) med adress `0xb8004d00`. Var och en av LED-lamporna kan tändas genom att värdet 1 skrivs, och släckas genom att värdet 0 skrivs. LED-lamporna styrs med skrivningar till bitarna med index 9, 8, 7 och 6, där index 9 representerar LED-lampa nummer 4 och index 6 representerar LED-lampa nummer 1. Den bit som har index 0 är den minst signifikanta biten. Porten är redan konfigurerad som utport. När du skriver till porten behöver du inte bevara någon av de andra bitarna i registret.

   Skriv en funktion i MIPS-assembler; funktionen ska heta `twinkle`. Assemblerfunktionen måste följa C-anropskonventionerna så att det går att anropa assemblerfunktionen från programspråket C. Funktionen ska blinka med LED-lampa nummer 3 (det vill säga tända eller släcka den). Att invertera en bit i ett register för att blinka med lampan ska göras med instruktionen `xori`. Alla andra LED-lampor ska vara släckta. Vid den första iterationen ska LED-lampan tändas. Funktionen twinkle har en parameter (positivt heltal), som anger hur många gånger LED-lampan ska tändas eller släckas. Om exempelvis argumentet till funktionen är 6, så ska funktionen tända LED-lampan 3 gånger, och släcka den 3 gånger. Efter dessa 6 steg ska funktionen returnera.

   Efter varje steg (antingen tändning eller släckning av LED-lampan) ska funktionen `pause` anropas korrekt. Du kan anta att denna funktion redan finns, och att den inte förstör några `$t`-register. Funktionen pausar programmet en kort stund, och returnerar antalet millisekunder som programmet var pausat. Funktionen `twinkle` ska addera samman det totala antalet millisekunder som funktionen `twinkle` har pausat, och returnera detta värde.

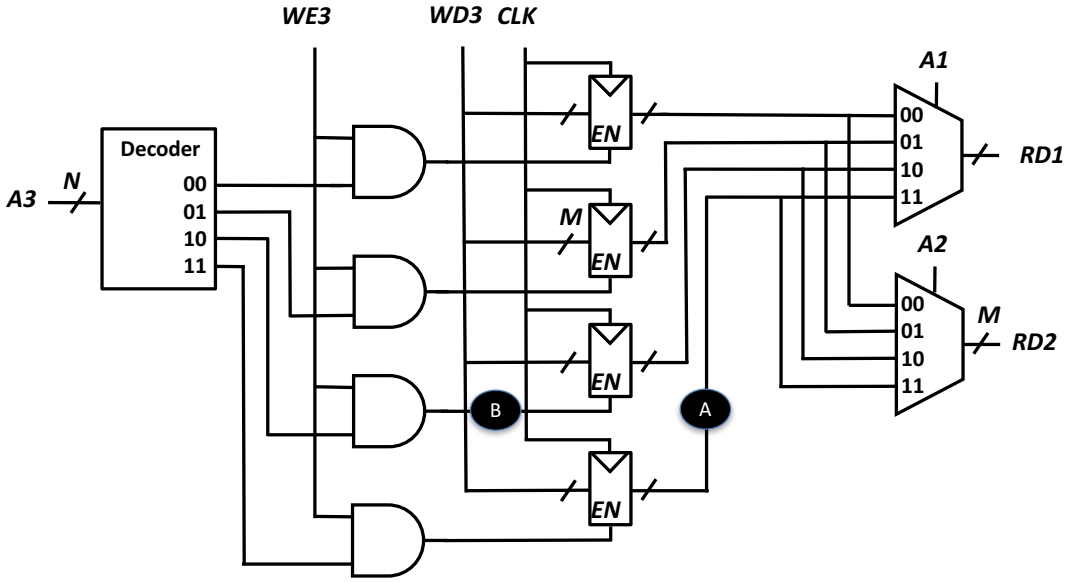
   Du får inte använda några pseudoinstruktioner i assembler, och inte heller något stackutrymme. Du kan få delpoäng även om lösningen inte är fullständigt korrekt.

   (6 poäng)

   (b) Anta att du ska ställa in en timer i ett inbyggt system med klockfrekvensen 100 MHz. Timern använder en skalfaktor (pre-scaling), och faktorn är `1:4`. Timern är en 32-bits timer. Vilket värde ska periodregistret ha, om vi vill att timerns periodtid ska vara 3 sekunder? (2 poäng)

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

(a) 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 256 bitar. Vad är då värdet på M?

ii. Anta att *A3* $= 3$, *WE3* $= 1$, *WD3* $= 7$ och *CLK* $= 0$ vid ett visst tillfälle. Vad är då värdet på A?

iii. Anta att *A3* $= 3$ och *WE3* $= 1$. Vad är då värdet på signal B?

iv. Anta att *WD3* $= 9$, *CLK* $= 0$, *A2* $= 2$, *A3* $= 3$ och att alla register innehåller värdet 7 i sina minnen. Registrens minnen triggas på positiv flank. Vad blir då värdet på *RD2* när clockan har ändrats till *CLK* $= 1$?

(4 poäng)

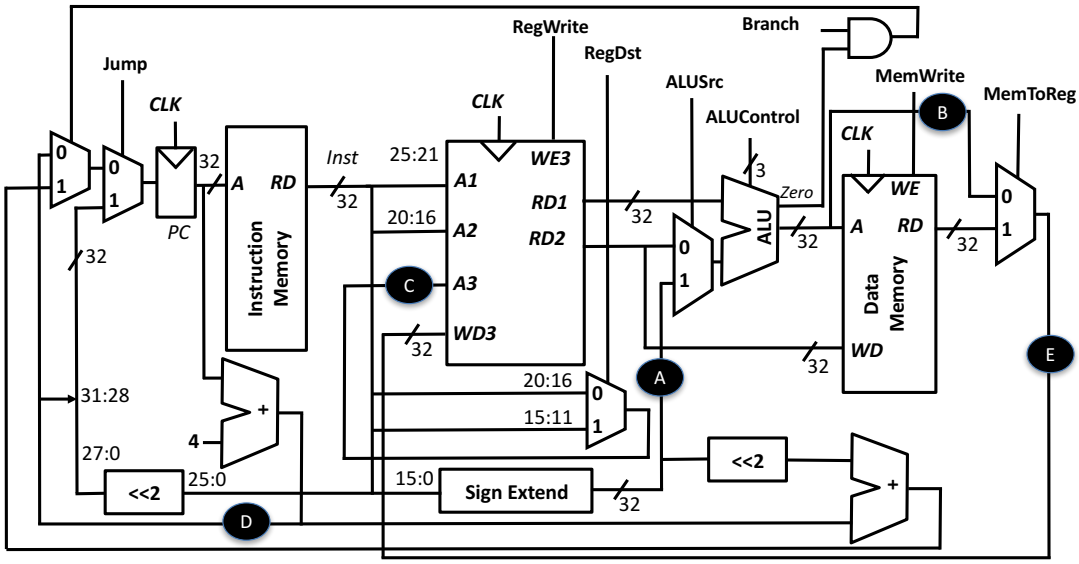
(b) Konstruera och rita en digital krets som har tre ingångar A, B och C, samt en utgång R. Signalerna A, B och R är 16-bitars signaler, som representerar tal i 2-komplementsform. Signalen C är en binär signal. Kretsen ska uppfylla följande specifikation: Om $C = 0$ så ska $R = A + B$ och om $C = 1$ så ska $R = A - B$.

Det är tillåtet att använda följande komponenter: 16-bitars adderare (carry propagate adder), 2:1 multiplexer, 4:1 multiplexer, 2:4-avkodare (decoder), inverterare, OCH-grind, samt ELLER-grind. Notera att du inte behöver använda alla dessa komponenter för att korrekt svar. Du måste skriva ut bussbredder (i bitar) för signaler på ett korrekt sätt (när det inte är möjligt att härleda).

(4 poäng)

4. **Modul 4: Processorkonstruktion**

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



Anta att de fyra första raderna av följande kod har exekverats:

```
addi    $t0,$zero,0x3ff0
addi    $t0,$t0,16
sw      $t0,-8($t0)
addi    $t0,$t0,12
lw      $s2,-20($t0)
```

Anta också att processorn för tillfället håller på att exekvera instruktionen `lw`. Den första `addi`-instruktionen finns på minnesadress `0x00040210`. Vad blir då signalvärdena för A, B, C, D och E? Svara antingen med hexadecimala tal, eller skriv `okänt` för ett signalvärde om 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 en 5-stegs pipeline.

```
    lw   $t0,0xff08($zero)
    beq  $t0,$zero,foo
    addi $t0,$zero,5
foo:
    addi $t0,$zero,10
```

Anta vidare att jämförelsen för likhet i instruktionen beq utförs i avkodningssteget, och att instruktionen statiskt gissar att hopp inte tas (branch not taken).

i. Hur många data-hazarder finns och hur många kontroll-hazarder finns?

ii. Om det finns en eller flera data-hazards, vilka instruktioner är då inblandade?

iii. Vad är den maximala totala extratiden (penalty) räknat i klockcykler, som kan inträffa när koden exekveras, när de bästa lösningarna för att hantera hazarderna används?

(3 poäng)

5. **Modul 5: Minneshierarkier**

(a) Anta att vi har en 32-bitars MIPS-processor med ett 4-vägsassociativt cacheminne. Cacheminnets storlek (capacity) är 4096 byte och blockstorleken är 8 byte.

i. Hur många giltigbitar (valid bits) finns i cacheminnet?

ii. Vad är storleken på adresslappsfältet (tag field), räknat i bitar?

(2 poäng)

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

```
   lui   $t0,0x3b
   ori   $t0,$t0,0x2210
   addi  $t1,$zero,5
   addi  $t2,$zero,0
   addi  $t3,$zero,0
loop:
   lw    $t2,4($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 4096 byte och det finns 256 block i instruktionscacheminnet. Datacacheminnet har en storlek (capacity) på 1024 byte och blockstorleken i datacacheminnet är 8 byte. Instruktionen `lui` finns på minnesadress `0x00040100`. Vi antar att alla giltigbitar är noll i båda fallen, innan koden exekveras.

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

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

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

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

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

   (a) 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. Advanced Vector Extension (AVX) är konstruerad för program med dataparallellitet (data-level parallelism).

      ii. Multitrådning i maskinvara (hardware multithreading) är en sorts parallellitetsmekanism (concurrency mechanism) inuti en process i ett operativsystem.

      iii. Hyper-threading är ett annat namn på VLIW (Very Long Instruction Words).

      iv. Instruktionsnivåparallellitet (instruction-level parallelism) är en parallelliseringsmetod som är tillgänglig i nästan alla processorer för bärbara datorer (laptops) och arbetsstationer.

      v. Single Instruction Multiple Data (SIMD) används ofta i tillämpningar med uppgiftsparallellitet (task-level parallelism).

      (5 poäng)

   (b) Anta att du utvecklar ett programvarusystem, och att din chef frågar om det skulle gå att göra det snabbare om det kördes på en maskin med flera kärnor (multicore machine). Du har bara tillgång till en flerkärnig maskin med 4 kärnor, men chefen vill veta hur fort det skulle gå om du körde programmet på en 8-kärnig maskin. Du vet att du inte kan ge ett exakt svar utan att köra på en 8-kärnig maskin, men du inser också att du kan använda Amdahls lag för att göra en ungefärlig beräkning. Du gör en mätning vid körning på bara en kärna, och testprogrammet tar 12 sekunder att exekvera. Du kör det igen på 4 kärnor, och exekveringstiden är då 6 sekunder. Vad är då den uppskattade exekveringstiden om 8 kärnor används? (3 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) SIMD-instruktioner, jämfört med uppgiftsparallellitet (task-level parallelism)

   (b) Processorer med långa pipelines, jämfört med instruktionsnivåparallellitet (ILP, instruction-level parallelism)

   (c) cacheminnen, jämfört med hårdvarumultitrådning (hardware multithreading)

   (15 poäng)

8. Betrakta följande assemblerkod för en 32-bitars MIPS-processorer:

```
foo:    addi    $sp,$sp,-8
        sw      $ra,4($sp)
        sw      $s3,0($sp)

        addi    $s3,$zero,0
        addi    $s4,$zero,5

loop:   slt     $t3,$s3,$s4
        beq     $t3,$zero,exit
        addi    $a1,$s3,0
        jal     boo
        addi    $a0,$v0,0
        addi    $s3,$s3,1
        j       loop

exit:   lw      $s3,0($sp)
        lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra


boo:    addi    $sp,$sp,-8
        sw      $ra,4($sp)
        sw      $s0,0($sp)

        addi    $t0,$zero,-1
        beq     $a0,$t0,ortrue

        sll     $t0,$a0,2
        add     $t0,$t0,$s2
        sll     $t1,$a1,2
        add     $t1,$t1,$s2
        lw      $t0,0($t0)
        lw      $t1,0($t1)
        slt     $t2,$t1,$t0
        beq     $t2,$zero,next

ortrue: sll     $t0,$a1,2
        add     $t0,$t0,$s1
        sw      $a0,0($t0)
        addi    $v0,$a1,0
        lw      $s0,0($sp)
        lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra

next:   sll     $t0,$a0,2
        add     $t0,$t0,$s1
        addi    $s0,$a0,0
        lw      $a0,0($t0)
        jal     boo
        sll     $t0,$s0,2
        add     $t0,$t0,$s1
        sw      $v0,0($t0)
```

```
        addi    $v0,$s0,0
        lw      $s0,0($sp)
        lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra
```

Anta att följande två arrayer är definierade i programspråket C:

```
int  plist[5];
int  vlist[5] = {7,100,2,35,1};
```

Antag också att register `$s1` innehåller adressen till första elementet i array `plist`, samt att `$s2` innehåller adressen till första elementet i array `vlist`. Anta att register `$sp` innehåller en adress till en minnesarea som går att läsa ifrån och skriva till på ett säkert sätt. Anta nu att följande C-programkod exekveras:

```
  int p = foo(-1);
  while(p != -1){
    printf("%d, ", vlist[p]);
    p = plist[p];
  }
}
```

(a) Assemblerkoden definierar två funktioner med namnen `foo` och `boo`, där `foo` anropas från C-koden. Översätt dessa två funktioner till tydliga och lättförståeliga C-funktioner. Dina C-funktioner får inte använda pekare, utan bara array-indexering. För full poäng ska C-funktionerna vara korta, välskrivna och lätta att förstå. Var och en av C-funktionerna bör inte behöva mer än 10 rader programkod. (15 poäng)

(b) Ange vad som skrivs till stout (standard output), och ge en klar och pedagogisk förklaring till vad som händer när programmet exekveras. Använd diagram och figurer för attgöra förklaringen tydlig. Ditt svar förväntas vara detaljerat. (10 poäng)

9. Betrakta återigen MIPS-programkoden i uppgift 8. Anta att programkoden exekveras på en 32-bitars 5-stegs MIPS-processor med direktmappat datacacheminne. Anta vidare att cacheminnesstorleken är 32 kbyte, och att blockstorleken är 8 byte. Arrayen `plist` börjar på adress `0x00330000` och arrayen `vlist` är placerad direkt efter `plist` i minnet. Register `$sp` innehåller adressen `0x02030000` varje gång `foo` anropas. Anta att alla giltigbitar (valid bits) i cacheminnet är nollställda innan funktionen `foo` anropas. Funktionen `boo` anropas flera gånger från `foo`. Vad blir misskvoten (miss rate) i datacacheminnet när funktionen `boo` exekveras, vid det specifika tillfälle när den anropas med värdet 1 i det andra argumentet? (10 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);

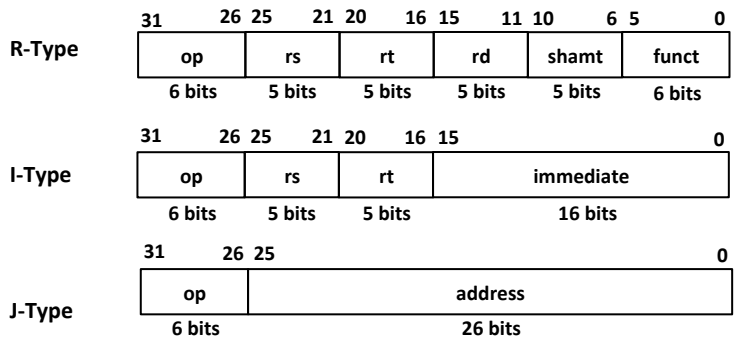
PSEUDO INSTRUCTIONS (SUBSET)

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.