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

2019-03-15

08.00-13.00

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

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

Instructions in English

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

The exam consists of two parts:

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

Grades

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

Grading scale (For both IS1200 and IS1500):

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

Results

- The result will be announced at latest 2019-04-05.
- 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 2019-04-26.

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 2019-04-05.
- 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 2019-04-26.

Part I: Fundamentals

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

   (a) Assume that a 32-bit MIPS processor executes the following assembly instructions:

```
addi    $t1,$zero,-32
sra     $t0,$t1,2
```

   i. What is the machine code of the `sra` instruction above? Answer as a 32-bit hexadecimal value. (3 points)

   ii. What is the value of register `$t0` after executing these two instructions? Answer as a positive or negative decimal number. (1 point)

   Note: if a register position is not used in the encoding of the machine code, zero bits shall be used in such positions.

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

```
int lst[] = {7,8,10,0};

void foo(int *p){
  while(*p != 0){
  // ---------------
  // Code line 1
  // Code line 2
  // ---------------
  }
}

void bar(){
  int *b = lst;
  foo(b);
}
```

   The program is started by calling function `bar`. Function `foo` is missing two lines of C code. The assembly code corresponding to these two lines of C code is as follows:

```
lw      $t0,0($a0)
addi    $t0,$t0,2
sw      $t0,0($a0)
addi    $a0,$a0,4
```

   - Write down the two lines of C code that correspond to the assembly code. You are not allowed to make use of array index syntax (using brackets [ ]). (3 points)
   - Write down the content of the array `lst` after that the function `bar` has finished its execution. (1 point)

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

(a) For each of the following four 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 2 sentences). You do not need to give a motivation if the statement is true. (4 points)

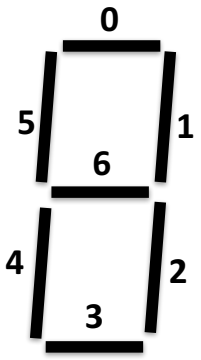
i. Suppose you are configuring a 16-bit timer running at 100 Mhz. You would like your program to write out a message to the standard output every 10th second. It is clear to you that you cannot configure the period register so that the timer triggers every 10th second. Even if this limitation exists, it is still possible to use the 16-bit timer to solve the described task using software counters.

ii. The `volatile` keyword in the C programming language means that the specified memory segment is volatile and that the data will be lost if the computer is powered off.

iii. A key aspect of a synchronous bus compared to an asynchronous bus is that a synchronous bus is using a clock signal.

iv. When interrupts are used in an embedded system, it means that a program can be interrupted, but it *cannot* jump back to the original program after that the interrupt is over.

(b) Consider the 7-segment LED display below, where each LED is numbered 0 through 6.

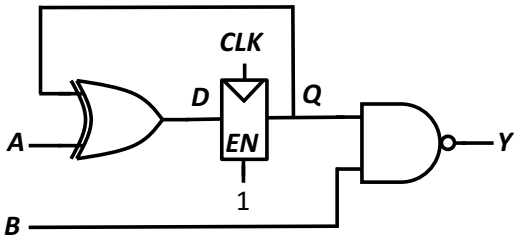


Your task is to write down the 32-bit MIPS assembly code that displays number 3 on the 7-segment display. That is, the LEDs with numbers 5 and 4 should be turned off and the rest of the LEDs should be turned on.

The LED lights are memory mapped to address `0x95003f00`. The seven bits with indices 3 to 9 represent the 7-segment lights 0 to 6. For instance, if the bit with index 3 is set to one, then the LED light with number 0 is turned on. If the bit with index 9 is set to zero, then the LED light with number 6 on the display is turned off. No other bits of the port are allowed to be modified. This means that you must read the current bit values of the port and only modify the relevant bits. You are not allowed to use any pseudo instructions. (4 points)

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

(a) Consider the circuit and the truth table below.



clock	*A*	*B*	*Q*	*D*	*Y*
init	1	0	0		
1st	1	1			

The first line of the truth table is the initial state before a clock tick. The second line is the stabilized values after the first clock tick. Copy the truth table and fill in the missing values. (4 points)

(b) Prove the correctness of the following boolean algebra theorem by using *proof by perfect induction* (also called *proof by exhaustion*).
(3 points)

$$C \cdot \overline{A \cdot B} = C \cdot \overline{A} + C \cdot \overline{B} \tag{1}$$

(c) Which of the following alternatives is correct? Note that only *one* alternative is correct. You do not have to motivate your answer. Just write down one of the alternatives i), ii), iii), or iv). (1 point)

i. When the enable signal of a tristate buffer is not enabled, the output is said to be floating.

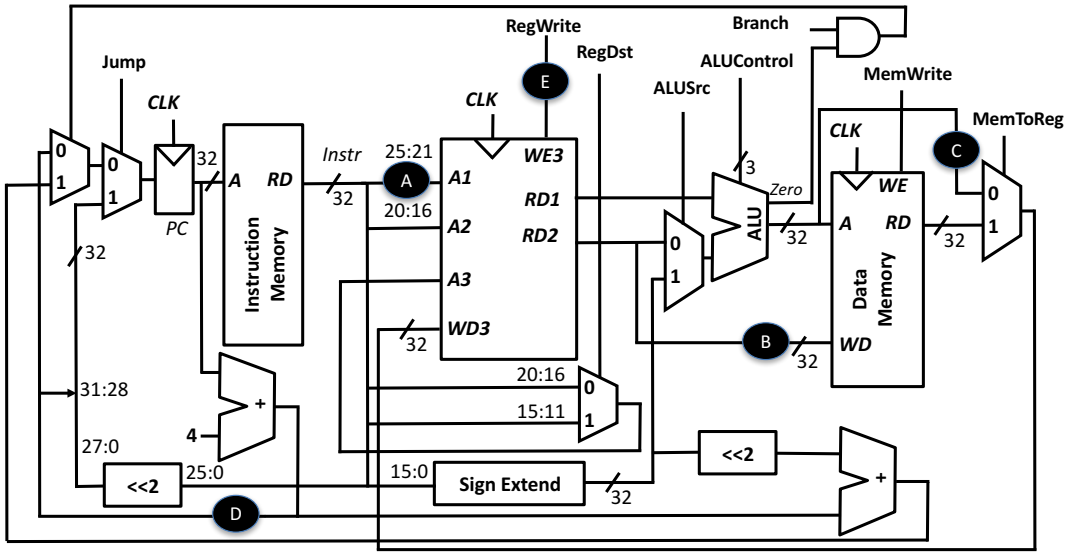
ii. A multiplexer has a one-hot output signal. That is, for any given input, one and only one output signal is high at the same time on a multiplexer.

iii. A carry propagate adder (CPA) cannot be used when adding together signed integer values.

iv. A combinatorial circuit is a circuit that depends on both the current and prior input values.

4. **Module 4: Processor Design**

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



Suppose the following instruction is executing.

```
sw      $t8,-8($a0)
```

Before the execution of the `sw` starts, register `$t8` contains value `0x2f` and register `$a0` contains value `0x3500`. What are then the signal values for A, B, C, D, and E when the `sw` 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. (5 points)

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

```
        addi    $t0,$zero,0xff20        # F D E M W
        sll     $t2,$t0,16              #   F D E M W
        addi    $t3,$zero,0xf           #     F D E M W
        lw      $t1,8($t2)              #       F D E M W
        bne     $t0,$zero,foo           #         F D E M W
        xori    $t1,$zero,1             #           F D E M W
foo:
```

For each data hazard found in the above code, answer the following questions:

i. Between which instructions does the hazard occur?

ii. Which register is involved in the hazard?

iii. How can the hazard be solved in the best possible way (forwarding, stalling, or both forwarding and stalling)?

(3 points)

5. **Module 5: Memory Hierarchy**

   (a) Suppose you have a direct mapped data cache on a 16-bit processor with block size 8 bytes. Inside the cache, at set number 31, the valid bit is 1 and the tag is of size 6 bits and has the binary value `100101`. In all other sets, the valid bits are set to 0.

      i. How many sets are there in the cache? (1 point)

      ii. What is the capacity of the cache in bytes? (1 point)

      iii. State an address where a load word instruction would result in a cache hit. Answer as a hexadecimal number. (2 points)

   (b) Suppose we have an instruction cache with 4096 sets and a block size of 16 bytes. For a 32-bit MIPS processor, 7 instructions are executed in sequence. That is, none of the 7 instructions are branch or jump instructions. The first instruction in the sequence is located at address `0x4000301C`. Assume that all valid bits are zero before executing the code sequence.

      i. For each of the seven instructions, state if there will be a cache hit or a cache miss when the instruction is executed (3 points).

      ii. What is the cache hit rate when executing the 7 instructions? Answer as a rational number. (1 point)

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

   (a) Suppose you developed a program that takes 10s to execute on one core. You have used Amdahl's law and calculated that the theoretical limitation for speedup of your program is 5. Assume that doubling the number of cores half the execution time for the part that can be parallelized. What would then the speedup be if the program is executed on 4 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. A Linux operating system running on a processor with just one core is a good example of running programs with concurrency, but without parallelism.

      ii. A *multiplexer* can be used to avoid that several different threads in a multi-threaded program access the same resource.

      iii. MapReduce is a programming model that is commonly used for parallel computations on cluster systems.

      iv. False sharing is a problematic situation on a multicore system, where different cores invalidate the same part of the cache, without actually accessing the same data.

      v. One important component of simultaneous multithreading (SMT) is fine-grained multithreading.

      (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) MIMD vs. data-level parallelism.

   (b) Polling of timers vs. using timers with interrupts.

   (c) Instruction-level parallelism vs. hardware multi-threading.

   (15 points)

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

```
.data
lst:    .word   0,0,0,0,0,0,0,0
s1:     .asciiz  "A"

.text
        la      $a0,lst
        la      $t1,s1
        sw      $t1,0($a0)
        jal     foo
stop:   j       stop


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

        lw      $t0,0($a0)
        bne     $t0,$zero,else

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

else:   addi    $s0,$zero,0

while:  lb      $t1,0($t0)
        beq     $t1,$zero,exit
        addi    $s0,$s0,1
        addi    $t0,$t0,1
        j       while

exit:   addi    $a0,$a0,4
        jal     foo
        add     $v0,$v0,$s0
        lw      $s0,0($sp)
        lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra
```

Suppose the above assembly program is running on a 32-bit 5-stage pipelined MIPS processor without branch delay slots. The `bne` and `beq` comparisons are done in the execute stage. The processor has separate data and instruction caches. Both caches are direct mapped, each with a capacity of 4096 bytes. The instruction cache has a block size of 16 bytes, whereas the data cache has a block size of 8 bytes. A cache miss gives a 10 clock cycles penalty and a cache hit gives no penalty. The caches have write-back policies, which mean that a write cache miss always gives a penalty, whereas write cache hits give no penalties. The processor can handle both forwarding and stalling. Stalling in the pipeline and cache miss penalty are independent. The assembler directive `.asciiz` means that the string is null terminated.

The first code between `.text` and label `foo` is initialization code and should not be included when computing cache misses or clock cycles below. That is, assume that the program starts when function `foo` is called the first time and that it ends when the function returns from this first call. However, note that the initialization actually takes place before the call to `foo`, which changes the values in the `.data` section. Assume further that both caches are empty before the first call to `foo` (i.e. that the caches are cleared). Assume also that `foo`, `lst`, and `s1` are memory word aligned. The stack pointer `$sp` points to address `0x70000000` before the function `foo` is called.

(a) Calculate the instruction cache miss rate. (5 points)

(b) Calculate the data cache miss rate. (5 points)

(c) Calculate the total number of clock cycles, including cache misses and hazards that cause stalling (due to data hazards) or bubbles in the pipeline (due to control hazards). Note that you should view the number of clock cycles as the number of instruction fetches. That is, if `foo` only contained one return instruction `foo: jr $ra`, the total number of clock cycles would be 1 clock cycle for the fetch, plus additional clock cycles for instruction cache misses. (8 points)

9. Consider the following partially implemented C program.

```
#include <stdio.h>

char *s1 = "This";
char *s2 = "is";
char *s3 = "fun";
char *lst[4];

int main(){
  lst[0] = s1;
  lst[1] = s2;
  lst[2] = s3;
  lst[3] = 0;

  printf("Length␣%d\n", foo(lst));
}
```

Function `main` calls function `foo`, which was defined as an assembler program in exercise 8.

(a) Translate the `foo` function into a clear C function. Note that the C function should make use of pointers and recursion. It should contain one `if-then-else` statement and one `while` loop. (13 points)

(b) If the above C program is executed using function `foo`, what is then printed to standard output? Explain shortly what the program is doing. (4 points)

Del I: Grundläggande

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

   (a) Antag att en 32-bitars MIPS-processor exekverar följande assemblerinstruktioner:

   ```
   addi    $t1,$zero,-32
   sra     $t0,$t1,2
   ```

   i. Vad är maskinkoden för instruktionen `sra` ovan? Svara med ett 32-bitars hexadecimalt tal. (3 poäng)

   ii. Vad är värdet i register `$t0` efter att dessa två instruktioner exekverats? Svara med ett positivt eller negativt decimalt tal. (1 poäng)

   Observera att om en registerposition inte används i maskinkoden så ska nollställda bitar användas i de positionerna.

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

```
int lst[] = {7,8,10,0};

void foo(int *p){
  while(*p != 0){
  // ---------------
  // Code line 1
  // Code line 2
  // ---------------
  }
}

void bar(){
  int *b = lst;
  foo(b);
}
```

   Programmet startas genom att funktionen `bar` anropas. Funktionen `foo` saknar två rader C-kod. Den assemblerkod som motsvarar dessa två rader C-kod är som följer:

```
lw      $t0,0($a0)
addi    $t0,$t0,2
sw      $t0,0($a0)
addi    $a0,$a0,4
```

   - Skriv de två rader C-kod som motsvarar assemblerkoden. Du får inte använda syntaxen för arrayindexering (med hakparenteser [ ]). (3 poäng)
   - Skriv ner innehållet i arrayen `lst` efter att funktionen `bar` har slutat exekveras. (1 poäng)

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

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

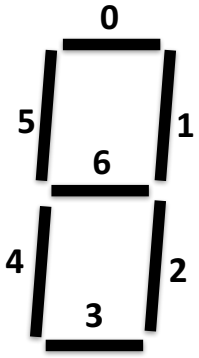
i. Anta att du konfigurerar en 16-bitars timer som går med 100 MHz. Du vill att ditt program ska skriva ut ett meddelande på standard-output var 10:e sekund. Det är uppenbart att du inte kan ställa in periodregistret så att timern triggas var 10:e sekund. Även med denna begränsning så går det att använda 16-bitars-timern för att lösa den beskrivna uppgiften med programvaruräknare.

ii. Nyckelordet `volatile` i programmeringsspråket C innebär att det specificerade minnessegmentet är volatilt och att data kommer att tappas om datorn stängs av.

iii. En nyckelaspekt hos en synkron buss, jämförd med en en asynkron buss, är att den synkrona bussen använder en klocksignal.

iv. När avbrott används i ett inbyggt system så betyder det att ett program kan avbrytas, men kan *inte* hoppa tillbaka till originalprogrammet när avbrottet är över.

(b) Här nedanför visas en 7-segments lysdiodsdisplay, där varje lysdiod är numrerad från 0 till 6.

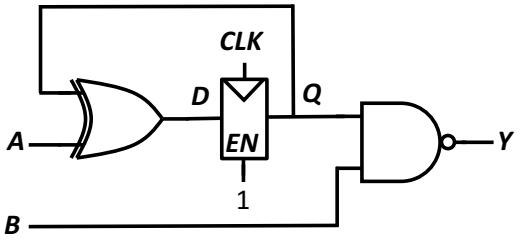


Din uppgift är att skriva den 32-bitars MIPS-assemblerkod som visar siffran 3 på 7-segmentsdisplayen. Det betyder att lysdioderna med numren 5 och 4 ska vara släckta och att resten av lysdioderna ska vara tända.

Lysdioderna är minnesmappade till adressen `0x95003f00`. De sju bitar som har index 3 till 9 representerar 7-segments-lysdioderna 0 till 6. Om till exempel biten med index 3 ettställs, så tänds lysdioden med nummer 0. Om biten med index 9 nollställs, så släcks lysdioden med nummer 6 i displayen. Inga andra bitar i porten får ändras. Det betyder att du måste läsa de aktuella bitvärdena som porten har, och bara ändra de relevanta bitarna. Du får inte använda någon pseudoinstruktion.
(4 poäng)

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

(a) Betrakta kretsen och sanningstabellen här nedanför.



clock	*A*	*B*	*Q*	*D*	*Y*
init	1	0	0		
1st	1	1			

Första raden i sanningstabellen anger starttillståndet innan någon klockpuls. Andra raden anger det stabiliserade värdet efter första klockpulsen. Kopiera sanningstabellen och fyll i de värden som fattas. (4 poäng)

(b) Använd *perfekt induktion* (även kallat *proof by exhaustion*) för att visa att följande sats från den Booleska algebran är korrekt. (3 poäng)

$$C \cdot \overline{A \cdot B} = C \cdot \overline{A} + C \cdot \overline{B} \tag{2}$$

(c) Vilket av följande alternativ är korrekt? Observera att endast *ett* alternativ är korrekt. Du behöver inte motivera ditt svar. Skriv bara ner ett av alternativen i), ii), iii) eller iv). (1 poäng)

i. När enable-signalen till en tristate-buffert inte är aktiv, så säger man att utgången flyter.

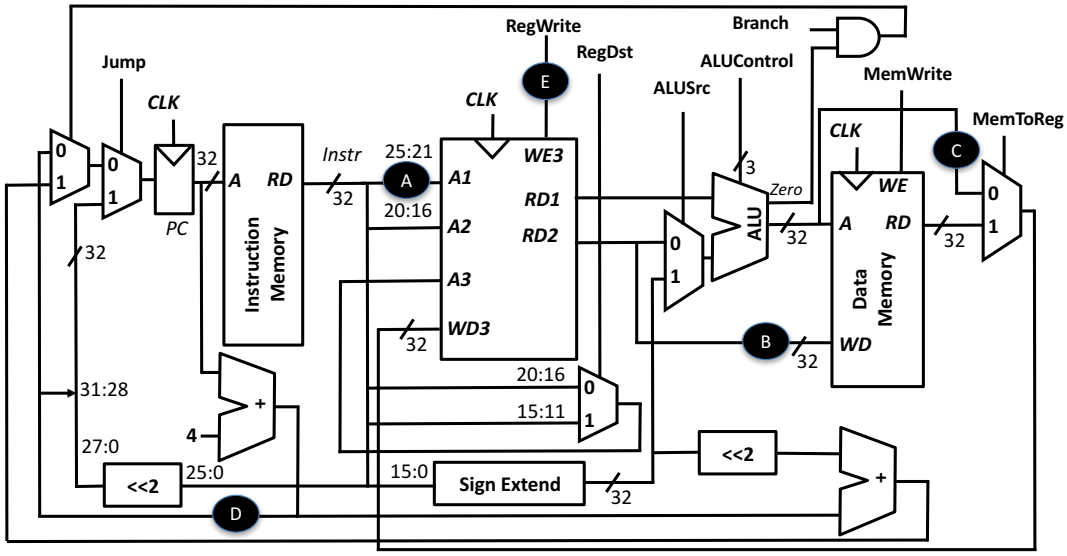
ii. En multiplexer har en utsignal av typen ”one-hot”. Det innebär att för vilken som helst insignal så är endast en utgång i taget aktiv på en multiplexer.

iii. En carry-propagate-adderare (CPA) kan inte användas för att addera heltalsvärden med tecken (signed integer values).

iv. En kombinatorisk krets är en krets som beror både på de nuvarande och på de tidigare ingångsvärdena.

4. **Modul 4: Processorkonstruktion**

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



Anta att följande instruktion exekveras.

```
sw      $t8,-8($a0)
```

Innan exekveringen av instruktionen `sw` så innehåller register `$t8` värdet `0x2f` och register `$a0` innehåller värdet `0x3500`. Vad blir då signalvärdena för A, B, C, D och E medan instruktionen `sw` exekveras? Svara antingen med hexadecimala tal, eller skriv `okänd` för ett signalvärde om det inte går att avgöra ett exakt värde utifrån den givna informationen. (5 poäng)

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

```
        addi    $t0,$zero,0xff20        # F D E M W
        sll     $t2,$t0,16              #   F D E M W
        addi    $t3,$zero,0xf           #     F D E M W
        lw      $t1,8($t2)              #       F D E M W
        bne     $t0,$zero,foo           #         F D E M W
        xori    $t1,$zero,1             #           F D E M W
foo:
```

För varje data-hazard i koden ovan, besvara följande frågor:

i. Mellan vilka instruktioner uppträder hazarden?

ii. Vilket register är inblandat i hazarden?

iii. Hur kan hazarden lösas på bästa möjliga sätt (forwarding, stalling eller både forwarding och stalling)?

(3 poäng)

5. **Modul 5: Minneshierarkier**

   (a) Anta att du har ett direktmappat datacacheminne på en 16-bitars processor. Blockstorleken är 8 byte. I cacheminnet, på rad nummer 31 (set number 31), är giltigbiten ettställd och adressetiketten (tag) har storleken 6 bitar och det binära värdet `100101`. På alla andra rader (in all other sets) så är giltigbiten satt till 0.

      i. Hur många rader (how many sets) finns i cacheminnet? (1 poäng)

      ii. Vad är storleken (capacity) på cacheminnet, räknat i byte? (1 poäng)

      iii. Ange en adress för vilken en load-word-instruktion skulle ge en träff (hit) i cacheminnet. Svara med ett hexadecimalt tal. (2 poäng)

   (b) Anta att vi har ett instruktionscacheminne med 4096 rader (4096 sets) och en blockstorlekt på 16 byte. En sekvens med 7 stycken instruktioner för en 32-bitars MIPS-processor utförs i följd. Ingen av de 7 instruktionerna är en hoppinstruktion. Första instruktionen i sekvensen är placerad på adress `0x4000301C`. Anta att alla giltigbitar är noll innan kodsekvensen exekveras.

      i. För var och en av de sju instruktionerna, ange om det kommer att bli träff eller miss i cacheminnet när instruktionen exekveras. (3 poäng)

      ii. Vad blir träffsäkerheten (hit rate) i cacheminnet när de 7 instruktionerna exekveras? Svara med ett rationellt tal. (1 poäng)

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

   (a) Anta att du utvecklat ett program som tar 10s att exekvera på en kärna. Du har använt Amdahls lag och räknat ut att den teoretiska gränsen för uppsnabbningen (speedup) för ditt program är 5. Antag även att om antalet processorkärnor dubbleras så halveras exekveringstiden för den delen som går att parallellisera. Vad blir då uppsnabbningen om programmet exekveras på 4 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. Exekvering av operativsystemet Linux på en processor med endast en kärna är ett bra exempel på körning av program med parallellitet i form av ”concurrency”, men utan det slags parallellitet som kallas ”parallelism” på engelska.

      ii. En *multiplexer* kan användas för att undvika att flera olika trådar i ett flertrådat program använder samma resurs.

      iii. MapReduce är en programmeringsmodell som ofta används för parallella beräkningar i klustersystem.

      iv. Falsk delning (false sharing) är en problematisk situation i ett flerkärnigt system, där olika kärnor invaliderar samma del av cacheminnet utan att faktiskt referera samma data.

      v. En viktigt komponent i samtidig flertrådighet (simultaneous multithreading, SMT) är finkorning flertrådighet (fine-grained multithreading).

      (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) MIMD jämfört med dataparallellitet (data-level parallelism).

   (b) Pollning av timers jämfört med timerstyrt avbrott.

   (c) Instruktionsnivåparallellitet (instruction-level parallelism) jämfört med hårdvaruflertrådighet (hardware multi-threading).

   (15 poäng)

8. Betrakta nedanstående 32-bitars MIPS-assemblerkod:

```
.data
lst:    .word   0,0,0,0,0,0,0,0
s1:     .asciiz  "A"

.text
        la      $a0,lst
        la      $t1,s1
        sw      $t1,0($a0)
        jal     foo
stop:   j       stop


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

        lw      $t0,0($a0)
        bne     $t0,$zero,else

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

else:   addi    $s0,$zero,0

while:  lb      $t1,0($t0)
        beq     $t1,$zero,exit
        addi    $s0,$s0,1
        addi    $t0,$t0,1
        j       while

exit:   addi    $a0,$a0,4
        jal     foo
        add     $v0,$v0,$s0
        lw      $s0,0($sp)
        lw      $ra,4($sp)
        addi    $sp,$sp,8
        jr      $ra
```

Anta att ovanstående assemblerprogram körs på en 32-bitars MIPS-processor, med 5-stegs pipeline utan fördröjt hopp (without branch delay slots). Jämförelserna `bne` och `beq` görs i exekveringssteget (the execute stage). Processorn har separata cacheminnen för instruktioner och data. Båda cacheminnena är direktmappade, och vart och ett av dem har storleken (capacity) 4096 byte. Instruktionscacheminnet har blockstorleken 16 byte, medan datacacheminnet har blockstorleken 8 byte. En cachemiss medför 10 cyklers extratid (a 10 clock-cycle penalty) och en cacheträff medför ingen extratid. Cacheminnen har write-back policies, vilket betyder att en skrivning med resulterande cachemiss alltid ger extratid (penalty), medan en skrivning med resulterande cachehit aldrig ger extratid. Processorn kan hantera både stalling och forwarding. Stalling i pipelinen och extratid vid cachemiss är oberoende. Assemblerdirektivet `.asciiz` betyder att strängen är nollter-

minerad (null-terminated).

Den första koden mellan `.text` och läget `foo` är initialiseringskod, som inte ska tas med när cachemissar eller klockcykler beräknad här nedanför. Det innebär att du ska anta att programmet startar när funktionen `foo` anropas första gången, och att det slutar när funktionen returnerar från detta första anrop. Notera dock att initialiseringen faktiskt sker innan anropet till `foo`, vilket ändrar värdena i sektionen `.data`. Anta vidare att båda cacheminnena är tomma innan det första anropet till `foo`, det vill säga att cachminnena har nollställts. Anta även att `foo`, `lst` och `s1` är minnesordriktade (memory word aligned). Stackpekaren `$sp` pekar på adressen `0x70000000` innan funktionen `foo` anropas.

(a) Beräkna misskvoten (miss rate) i instruktionscacheminnet. (5 poäng)

(b) Beräkna misskvoten (miss rate) i datacacheminnet. (5 poäng)

(c) Beräkna det totala antalet klockcykler, inklusive cachemissar och hazarder som orsakar stalling (på grund av datahazarder) eller bubblor i pipelinen (på grund av kontrollhazarder, control hazards). Observera att du ska se antalet klockcykler som antalet instruktionshämtningar. Som exempel, anta att `foo` bara skulle ha innehållit en returinstruktion `foo: jr $ra`. I så fall skulle det totala antalet klockcykler vara 1 cykel för instruktionshämtningen, plus klockcykler för missar i instruktionscacheminnet. (8 poäng)

9. Betrakta nedanstående delvis implementerade C-program.

```
#include <stdio.h>

char *s1 = "This";
char *s2 = "is";
char *s3 = "fun";
char *lst[4];

int main(){
  lst[0] = s1;
  lst[1] = s2;
  lst[2] = s3;
  lst[3] = 0;

  printf("Length␣%d\n", foo(lst));
}
```

Funktionen `main` anropar funktionen `foo`, som definierades som ett assemblerprogram i uppgift 8.

(a) Översätt funktionen `foo` till en tydlig C-funktion. Observera att C-funktionen ska använda pekare och rekursion. Den ska innehålla en sats av typen om-så-annars (`if-then-else`) och en `while`-slinga. (13 poäng)

(b) Om ovanstående C-program exekveras med funktionen `foo`, vad skrivs då ut på standard-output? Förklara kortfattat vad programmet gör. (4 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);

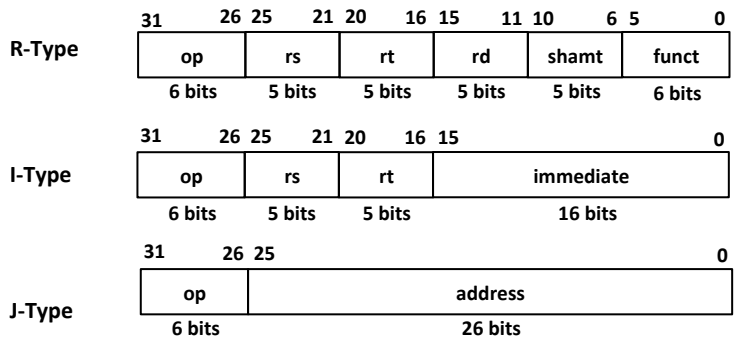
PSEUDO INSTRUCTIONS (SUBSET)

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.