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

2019-06-05

08.00-13.00

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

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

Instructions in English

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

The exam consists of two parts:

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

Grades

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

Grading scale (For both IS1200 and IS1500):

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

Results

- The result will be announced at latest 2019-06-26.
- If a student received grade FX, it is possible to request a complementary examination. The examiner decides if it is oral or in written form. Such complementary examination must be requested by the student. Please send an email to `dbro@kth.se` at latest 2019-07-17.

Instruktioner på Svenska

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

Tentamen består av två delar:

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

Betyg

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

Betygsskala (För både IS1200 och IS1500):

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

Resultat

- Resultaten kommer att meddelas senast 2019-06-26.
- Om en student får FX är det möjligt att begära en komplementerande examination. Examinatorn bestämmer om examinationen är muntlig eller skriftlig. Denna examination måste begäras via epost av studenten. Skicka epost till `dbro@kth.se` senast 2019-07-17.

Part I: Fundamentals

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

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

```
int x = 4;
int y = 15;
int *k = &x;
int z;
int *p;
*k = *k * (*k + 1);
y = x + y;
k = &z;

int a[] = {7,9,3,8};
p = a;
printf("%x\n",(unsigned int)p);

x = *k + 1;
p = p + 2;
z = *p - 4;

printf("%d␣%d␣%x␣%x\n",x,y,z,(unsigned int)p);
```

   Suppose that the first `printf` statement prints out the following to standard output:

```
5a9b1740
```

   What is printed out to standard output when the second `printf` statement executes? If one of the four values cannot be determined with the given information, write `unknown` instead of a concrete value. Note that the first two values are printed out in decimal form (using `%d`) and the third and fourth values are printed out in hexadecimal form (using `%x`). (4 points)

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

```
addi    $t0,$zero,0x55ff
lui     $t0,0x32a1
```

   i. What is the machine code of the `lui` 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 hexadecimal value (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.

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

(a) Assume that the following C function is compiled and executed on a 32-bit MIPS processor.

```
int getvals(){

  int *pushbutton = (int*) 0x10010040;
  int *switches = (int*) 0x10010050;
  int v=0;

  while(*pushbutton & 1){
    v = (*switches >> 4) & 0xf;
  }

  return v;
}
```

The platform that executes the code has memory mapped IO, where the status of 2 push buttons can be read at address `0x10010040` and the status of 8 switches at address `0x10010050`. Bit value 1 means that the button is pushed or that the switch is up. A bit value 0 means that a button is not pushed and that the switch is down. Button 1 is located at bit index 0 and Button 2 at bit index 1. Switch number 1 is located at bit index 4 and switch number 8 is at bit index 11. The bit at index 0 is the least significant bit.

i. The C code is missing a keyword to work correctly. Which keyword? (1 point)

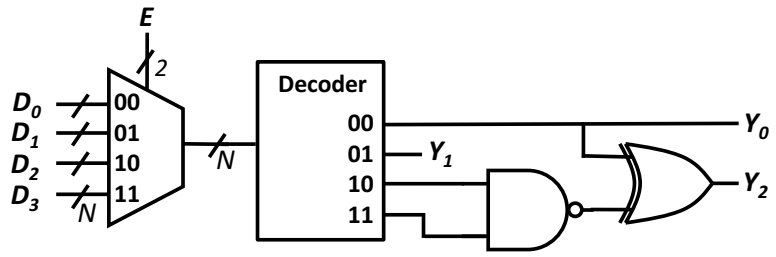
ii. Write down the complete MIPS assembly code that corresponds to the C function (assuming that the missing keyword has been added). (5 points)

iii. What will trigger the function to exit? (1 point)

iv. What value will be returned? (1 point)

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

(a) Consider the combinational logic diagram below. Assume that $D_0 = 1$, $D_1 = 3$, $D_2 = 2$, $D_3 = 0$, 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? (4 points)



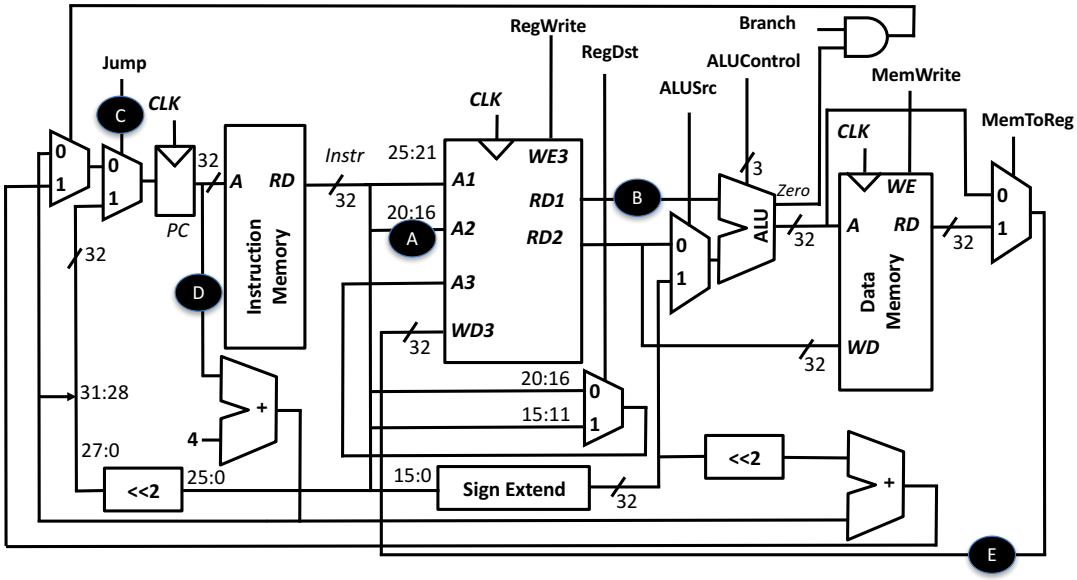
(b) Consider the following truth table, where A, B, and C are boolean variables. Write down a boolean algebra expression, using A, B, and C, that correctly represents the values of Y. The solution needs to be correct, but does not have to be in a simplified form. (2 points)

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(c) Assume we have a register file with three read ports and one write port. The width of the address signals is 6 bits. The register file can in total store 2048 bits. How many bits can then be written in parallel using the write port? (2 points)

4. **Module 4: Processor Design**

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



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

```
addi    $t0,$zero,3
sll     $t1,$t0,3
sub     $s0,$t1,$t0
```

and that the processor is currently executing the `sub` instruction. 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) Consider the following assembly code that is executing on a 5-stage pipelined 32-bit MIPS processor.

```
        addi    $t0,$zero,8         # F D E M W
        lw      $t0,-8($zero)       #   F D E M W
        beq     $t0,$zero,skip      #     F D E M W
        addi    $t0,$zero,5         #       F D E M W
        add     $t0,$t2,$t1         #         F D E M W
skip:
```

Assume that the processor is using branch delay slots and that the branch comparing operation is performed in the decode stage.

i. Between which instructions is there a data hazard?

ii. Is stalling needed to solve the hazard? If yes, how many clock cycles of delay are needed?

iii. If the branch is taken, what is then the value of register `$t0` when the program counter reaches label `skip`? Give a specific value, or state `unknown` if it is not possible to determine a specific value.

(3 points)

5. **Module 5: Memory Hierarchy**

(a) Suppose you have a 2-way set associative cache on a 32-bit MIPS processor. The cache has a capacity of 2048 bytes and there are 128 sets.

i. What is the block size in bytes?

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

iii. What is the tag size?

(3 points)

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

```
add     $t2,$zero,0x2210
lui     $t3,0x3300
or      $t3,$t3,$t2
lw      $t4,4($t3)
lw      $t4,8($t3)
```

The processor has separate data and instruction caches. The data cache has a capacity of 2048 bytes and there are 256 blocks in the data cache. The instruction cache has the capacity of 1024 bytes and the block size in the instruction cache is 16 bytes. The first assembly instruction in the listing is located at memory address `0x00030200`. 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 instruction cache show temporal locality, spatial locality, or both? (1 point)

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

   (a) Assume that a program can be divided into a sequential part and another part that is embarrassingly parallelizable. The latter means that N cores give N times improvement. If the parallelizable part represents 60% of the computation time when $N = 1$, what is then the maximal speedup that can be achieved? (3 points)

   (b) For each of the following 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. SIMD (single instruction, multiple data) corresponds to the notion of data-level parallelism.

      ii. Software threads are scheduled by the operating system.

      iii. A semaphore is a clever way to avoid the cache coherence problem.

      iv. Advanced Vector Extension (AVX) is a SIMD standard used in x86 processors.

      v. Very Long Instruction Word (VLIW) processors require that instruction scheduling is done statically, at compile time.

      (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) SIMD vs. multicore

   (b) Instruction-level parallelism vs. data-level parallelism

   (c) Hardware multithreading vs. multithreading in software

   (15 points)

8. Consider the following C program:

```
char* keys[100];
int values[100];
int elements = 0;

char* s1 = "Sara";
char* s2 = "Anders";
char* s3 = "Eva";

int find(const char* key){
  for(int i=0; i != elements; i++){
    if(match(*(keys + i),key)){
      return *(values + i);
    }
  }
  return -1;
}

int main(){
  insert(s1,43);
  insert(s2,35);
  insert(s3,21);
  printf("Anders:␣%d\n", find("Anders"));
  printf("Eva:␣%d\n", find("Eva"));
  printf("Kalle:␣%d\n", find("Kalle"));
  printf("Sara:␣%d\n", find("Sara"));
  return 0;
}
```

The main program illustrates a simple (not optimal) approach of how key/value pairs can be inserted into a map, and how the values can be returned for a given key. The keys are pointers to null-terminated C strings, and the values are integers. Only the `find` function is defined as C code. The `insert` and `match` functions are defined in 32-bit MIPS assembly code below.

```
# Function: insert
insert:
        la      $t2,elements
        lw      $t3,0($t2)
        sll     $t0,$t3,2

        la      $t1,keys
        add     $t1,$t1,$t0
        sw      $a0,0($t1)

        la      $t1,values
        add     $t1,$t1,$t0
        sw      $a1,0($t1)

        addi    $t3,$t3,1
        sw      $t3,0($t2)
        jr      $ra
```

```
# Function: match
match:
        lb      $t0,0($a0)
        lb      $t1,0($a1)
        beq     $t0,$zero,exit
        beq     $t1,$zero,exit
        bne     $t0,$t1,retfalse
        addi    $a0,$a0,1
        addi    $a1,$a1,1
        j       match
exit:
        bne     $t0,$zero,retfalse
        bne     $t1,$zero,retfalse
        addi    $v0,$zero,1
        jr      $ra
retfalse:
        addi    $v0,$zero,0
        jr      $ra
```

Note how the `find` function makes use of the `match` function. Translate both the `insert` function and the `match` function into C code. Your code must use pointers correctly and you are not allowed to use array syntax (`[ ]`).
(18 points)

9. Consider the program in Exercise 8 again. Translate the C function `find` into a MIPS assembly function. Note that the calling conventions must be implemented correctly.
(17 points)

Del I: Grundläggande

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

   (a) Förutsätt att följande C-kod exekveras på en 32-bitars MIPS-processor

```
int x = 4;
int y = 15;
int *k = &x;
int z;
int *p;
*k = *k * (*k + 1);
y = x + y;
k = &z;

int a[] = {7,9,3,8};
p = a;
printf("%x\n",(unsigned int)p);

x = *k + 1;
p = p + 2;
z = *p - 4;

printf("%d␣%d␣%x␣%x\n",x,y,z,(unsigned int)p);
```

   Anta att den första `printf`-satsen skriver ut följande på standard-output:

```
5a9b1740
```

   Vad skrivs ut på standard-output när den andra `printf`-satsen exekveras? Om ett av de fyra värdena inte kan avgöras utifrån den givna informationen, skriv `okänd` i stället för ett bestämt värde. Observera att de första två värdena skrivs ut i decimal form (med `%d`) och att de tredje och fjärde värdena skrivs ut hexadecimalt (med `%x`). (4 poäng)

   (b) Anta att en 32-bitars MIPS-processor exekverar följande assemblerinstruktioner:

```
addi    $t0,$zero,0x55ff
lui     $t0,0x32a1
```

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

      ii. Vad är värdet i register `$t0` efter att de två assemblerinstruktionerna har exekverats? Svara med ett hexadecimalt värde. (1 poäng)

   Observera att om en registerposition inte används i maskinkoden så ska noll-bitar användas i sådana positioner.

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

(a) Anta att följande C-funktion kompileras och exekveras på en 32-bitars MIPS-processor.

```
int getvals(){

  int *pushbutton = (int*) 0x10010040;
  int *switches = (int*) 0x10010050;
  int v=0;

  while(*pushbutton & 1){
    v = (*switches >> 4) & 0xf;
  }

  return v;
}
```

Plattformen som exekverar koden har minnesmappad in- och utmatning (memory-mapped I/O), där status för 2 tryckknappar kan läsas på adress `0x10010040` och status för 8 strömbrytare på adress `0x10010050`. En bit med värdet 1 betyder att knappen är intryckt eller att strömbrytaren är uppe. En bit med värdet 0 betyder att en knapp inte är intryckt och att strömbrytaren är nere. Knapp 1 finns på index 0 och Knapp 2 på index 1. Strömbrytare nummer 1 är placerad på bit-index 4 och strömbrytare nummer 8 är på bit-index 11. Biten med index 0 är den minst signifikanta biten.

i. C-koden saknar ett nyckelord för att fungera korrekt. Vilket nyckelord? (1 poäng)

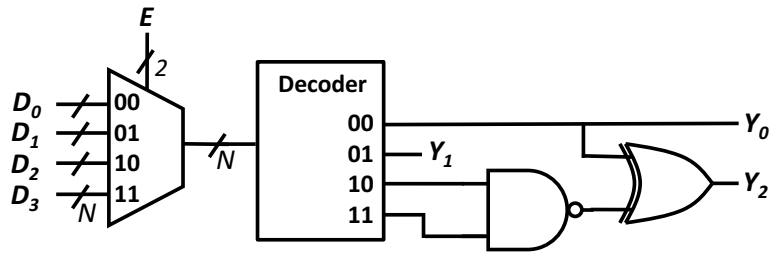
ii. Skriv ner den kompletta MIPS-assemblerkoden som motsvarar C-funktionen (förutsatt att det saknade nyckelorder har lagts till). (5 poäng)

iii. Vad kommer att trigga funktionen att avslutas? (1 poäng)

iv. Vilket värde kommer att returneras? (1 poäng)

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

(a) Betrakta det kombinatoriska schemat här nedanför. Anta att $D_0 = 1$, $D_1 = 3$, $D_2 = 2$, $D_3 = 0$ och $E = 3$. Vad blir då värdena på signalerna Y_0, Y_1 och Y_2 och vad är storleken på bit-bussen N? (4 poäng)



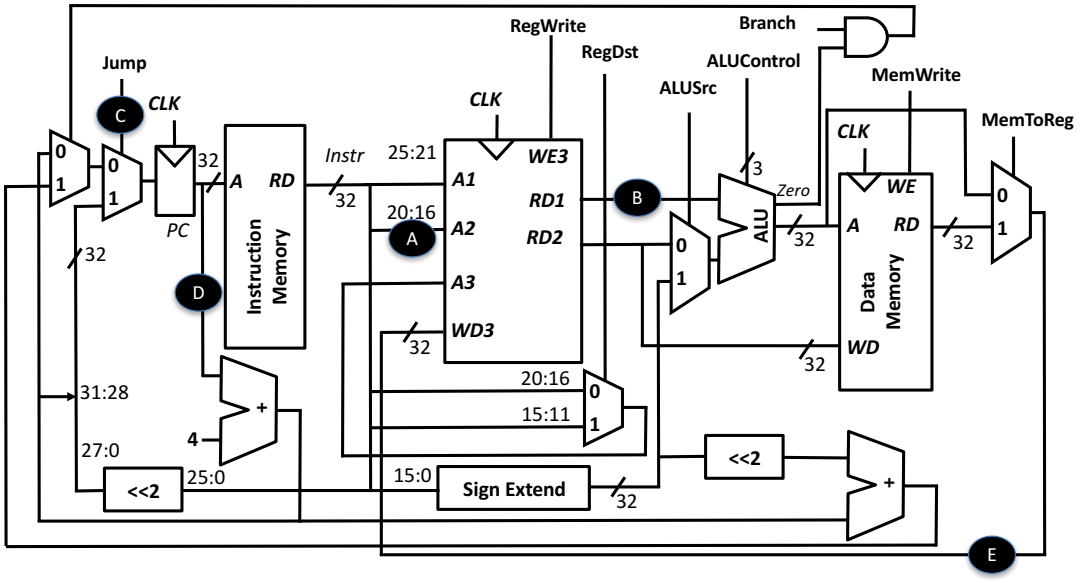
(b) Betrakta följande sanningstabell, där A, B och C är booleska variabler. Skriv ner ett uttryck i boolesk algebra, innehållande A, B och C, som representerar värdena för Y på ett korrekt sätt. Lösningen måste vara korrekt men behöver inte vara i förenklad form. (2 poäng)

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(c) Anta att vi har en registeruppsättning (register file) med tre läsportar och en skrivport. Bredden på vardera adress-signal är 6 bitar. Registeruppsättningen kan totalt lagra 2048 bitar. Hur många bitar kan då skrivas parallellt med skrivporten?
(2 poäng)

4. Modul 4: Processorkonstruktion

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



Anta att de första två raderna av följande kod har exekverats

```
addi    $t0,$zero,3
sll     $t1,$t0,3
sub     $s0,$t1,$t0
```

och att processorn för närvarande exekverar instruktionen `sub`. Vad blir då värdena på signalerna A, B, C, D och E? Svara antingen med hexadecimala tal, eller skriv `okänd` för ett signalvärde om det inte går att bestämma ett exakt värde utifrån den givna informationen. (5 poäng)

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

```
        addi    $t0,$zero,8         # F D E M W
        lw      $t0,-8($zero)       #   F D E M W
        beq     $t0,$zero,skip      #     F D E M W
        addi    $t0,$zero,5         #       F D E M W
        add     $t0,$t2,$t1         #         F D E M W
skip:
```

Anta att processorn har hoppfördröjning (branch delay slots) och att jämförelseoperationen vid hopp (branch) utförs i avkodningssteget (decode).

i. Mellan vilka instruktioner finns det en data-hazard?

ii. Behövs stalling för att lösa hazarden? Om ja, hur många klockcyklers fördröjning behövs?

iii. Om hoppet tas (the branch is taken), vad blir då värdet i register `$t0` när programräknaren (program counter) når läget `skip`? Ange ett specifikt värde, eller ange `okänd` om det inte går att bestämma ett specifikt värde.

(3 poäng)

5. **Modul 5: Minneshierarkier**

(a) Anta att du har ett 2-vägsassociativt cacheminne (2-way set-associative cache) till en 32-bitars MIPS-processor. Cacheminnet har en storlek (capacity) på 2048 byte och det finns 128 set (128 sets).

i. Vad är blockstorleken räknad i byte?

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

iii. Vad är storleken hos adressetiketten (tag)?

(3 poäng)

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

```
add     $t2,$zero,0x2210
lui     $t3,0x3300
or      $t3,$t3,$t2
lw      $t4,4($t3)
lw      $t4,8($t3)
```

Processorn har separata cacheminnen för data och instruktioner. Datacacheminnet har storleken (capacity) 2048 byte och det finns 256 block i cacheminnet. Instruktionscacheminnet har storleken (capacity) 1024 byte och blockstorleken i instruktionscacheminnet är 16 byte. Den första assemblerinstruktionen i listningen finns på minnesadress `0x00030200`. Vi antar att alla giltigbitar (valid-bits) är nollställda i båda cacheminnena innan exekveringen av koden.

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

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

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

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

   (a) Anta att ett program kan delas i en sekventiell del och en annan del som är generande parallelliserbar (embarrassingly parallelizable). Det senare innebär att N kärnor ger N gångers förbättring. Om den parallelliserbara delen representerar 60% av körtiden när $N = 1$, vad är då den maximala uppsnabbning (speedup) som kan uppnås? (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 varför påståendet är falskt (med högst två meningar). Om påståendet är sant behöver du inte ge någon motivering.

      i. SIMD (Single Instruction, Multiple Data) motsvarar begreppet dataparallellitet (data-level parallelism).

      ii. Mjukvarutrådar schemaläggs av operativsystemet.

      iii. En semafor är ett fiffigt sätt att undvika problemet med cachekoherens (the cache coherence problem).

      iv. Advanced Vector Extension (AVX) är en SIMD-standard som används i x86-processorer.

      v. Processorer av typen Very Long Instruction Word (VLIW) kräver att instruktionsschemaläggning (instruction scheduling) görs statiskt, vid kompileringstillfället.

      (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) SIMD jämfört med flerkärnighet (multicore)

   (b) Instruktionsnivåparallellitet (instruction-level parallelism) jämfört med dataparallellitet (data-level parallelism)

   (c) Hårdvarumultitrådning jämfört med multitrådning i programvara

   (15 poäng)

8. Betrakta följande C-program:

```
char* keys[100];
int values[100];
int elements = 0;

char* s1 = "Sara";
char* s2 = "Anders";
char* s3 = "Eva";

int find(const char* key){
  for(int i=0; i != elements; i++){
    if(match(*(keys + i),key)){
      return *(values + i);
    }
  }
  return -1;
}

int main(){
  insert(s1,43);
  insert(s2,35);
  insert(s3,21);
  printf("Anders:␣%d\n", find("Anders"));
  printf("Eva:␣%d\n", find("Eva"));
  printf("Kalle:␣%d\n", find("Kalle"));
  printf("Sara:␣%d\n", find("Sara"));
  return 0;
}
```

Huvudprogrammet (main) illustrerar ett enkelt (inte optimal) sätt för hur par av nycklar och värden (key/value pairs) kan sättas in i en mappning (map), och hur värdena kan returneras för en given nyckel (key). Nycklarna är pekare till noll-terminerade C-strängar (null-terminated C strings), och värdena är heltal. Endast funktionen `find` är definierad som C-kod. Funktionerna `insert` och `match` är definierade här nedanför, i 32-bitars assemblerkod för MIPS.

```
# Function: insert
insert:
        la      $t2,elements
        lw      $t3,0($t2)
        sll     $t0,$t3,2

        la      $t1,keys
        add     $t1,$t1,$t0
        sw      $a0,0($t1)

        la      $t1,values
        add     $t1,$t1,$t0
        sw      $a1,0($t1)

        addi    $t3,$t3,1
        sw      $t3,0($t2)
        jr      $ra
```

```
# Function: match
match:
        lb      $t0,0($a0)
        lb      $t1,0($a1)
        beq     $t0,$zero,exit
        beq     $t1,$zero,exit
        bne     $t0,$t1,retfalse
        addi    $a0,$a0,1
        addi    $a1,$a1,1
        j       match
exit:
        bne     $t0,$zero,retfalse
        bne     $t1,$zero,retfalse
        addi    $v0,$zero,1
        jr      $ra
retfalse:
        addi    $v0,$zero,0
        jr      $ra
```

Observera hur funktionen `find` använder funktionen `match`. Översätt både funktionen `insert` och funktionen `match` till C-kod. Din kod måste använda pekare korrekt och du får inte använda array-syntax (`[ ]`).

(18 poäng)

9. Betrakta återigen programmet i uppgift 8. Översätt C-funktionen `find` till en funktion i MIPS-assembler. Observera att anropskonventionerna (calling conventions) måste implementeras korrekt.
(17 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);

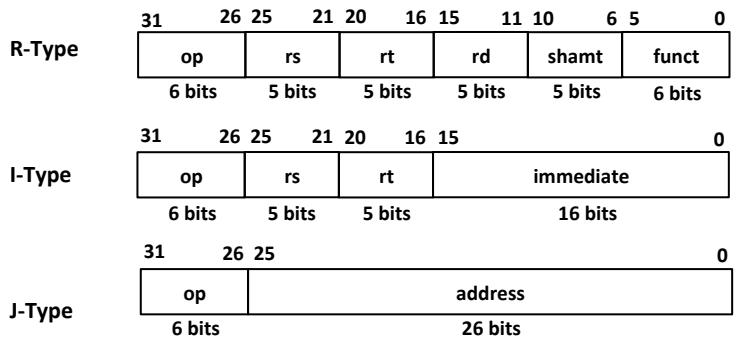
PSEUDO INSTRUCTIONS (SUBSET)

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.