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

2016-06-09

Suggested Solutions

Part I: Fundamentals

1. Module 1: C and Assembly Programming
   (a) Short answer:

   ```c
   int foo(int* x, int y){
   int tmp = *x + y;
   *x = tmp;
   return tmp;
   }
   ```

   Max 5 points. 1 point for correct function signature. 1 point for reading the correct value from memory using a pointer. 1 point for correctly adding up the numbers using the second parameter. 1 point for writing back the result to memory using a pointer. 1 point for returning the result.

   (b) Short answer: 0xac8bfff8

   Advice: Encode the number by using the MIPS reference sheet. Note that the instruction is an I-type instruction. Start by encoding the number into a binary number. Finally, convert the number into a hexadecimal number.

   Max 3 points. Remove one point for each incorrect byte number.

2. Module 2: I/O Systems
   (a) Short answer:

   ```mips
   led3:
   lui $t1,0xccbb
   lw $t0,0($t1)
   sll $a0,$a0,3
   andi $t0,$t0,0xfff7
   or $t0,$t0,$a0
   sw $t0,0($t1)
   jr $ra
   ```

   Max 5 points. 1 point for correct lui instruction. 1 point for correct shifting. 2 point for correct masking (or and andi instructions). 1 point for correct store word instruction and return instruction.
(b) Short answer: One possible solution is to set the period register to 40 000 and to use the prescale value 1 : 2.

Elaborated answer: There are many possible solutions to this exercise. For instance, if we set the period register to 40 000, it means that we will trigger the interrupt 8 000 000/40 000 = 200 times every second. If we then set the prescale value to be 1 : 2 it will be triggered 100 times every second, which means that it will be triggered every 1/100 = 0.01s, which is the same as every 10ms. Note that we need to use a prescale factor because the value 80 000 does not fit into a 16-bit register. An alternative solution can for instance be to set the period register to 10 000 and the prescale factor to 1 : 8. This gives the same result.

Max 3 points. 3 points if both the prescale and the period values are correct. Note that there are several correct solutions, but the two values need to be consistent to give any points on the exercise.

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

(a) Short answer:

![Decoder Circuit Diagram](image)

Comment: Note that you do not need to use all components.
Max 3 points. One point for each correct output signal.

(b) Short answer: This is a sequential logic circuit because the circuit contains a register, which is a form of memory.
Max 2 points. 1 point if the answer is sequential logic circuit and 1 point for a good motivation.

(c) Short answer: A register file.
Max 1 point. One point if the answer is correct. 0 points for just the answer register because this is not the same thing as a register file.

(d) Short answer: A flip-flop is edge-triggered, whereas a D latch is level-sensitive, i.e., the latch is transparent when the clock signal is 1.
Max 2 points. 1 point for answering that a flip-flop is edge triggered and 1 point for answering that the D latch is level-sensitive.
4. **Module 4: Processor Design**

(a) Short answers:
   (i) \( Y = 0x07 \). An or instruction.
   (ii) \( Y = 0xfe \). A sub instruction.
   (iii) \( Y = 0x01 \). A s1t instruction.

Max 3 points. 1 point for each correct answer (both the number and the instruction).

(b) Short answer: \( A = 0x11118888, F = 0x2, \) and \( Y = 0x11118883 \).

Comment: The function that the ALU is performing is an add operation \( (F = 0x2) \) because the instruction is performing an addition of the immediate value and the register value \( $s0 \) when calculating the final address.

Note that \( A \) must be the register value and not the index value, because \( A \) is the first argument in an R-instruction. To see this, compare which the arguments that would be used in a sub instruction.

In this exercise, we assumed that the registers were 32-bit. For the given answer above, we assumed that the ALU also was 32-bit, which was not completely clear in the exercise. Hence, a solution that only uses the 8 least significant bits will also be counted as a correct solution. In this case, \( A = 0x88, F = 0x2, \) and \( Y = 0x83 \).

Max 3 points. 1 point for each correct answer.

(c) Short answer: Register \( \$t0 \). The hazard is solved using forwarding and stalling.

Max 2 points. 1 point for the register and 1 point for correct answer (the combination of forwarding and stalling).

5. **Module 5: Memory Hierarchy**

(a) Short answer: The cache hit rate is \( \frac{1}{3} \).

Elaborated answer: The capacity is 4096 bytes and we are only reading \( 4 \cdot 6 = 24 \) bytes of instructions. Hence, there will be no conflict. We know that the block size is 8 bytes, which means that two instructions will be fetched if we get a cache miss (each instruction takes 4 bytes in a 32-bit MIPS processor).

We have in total \( 4096/8 = 512 \) sets. Hence, the set bit field is 9 bits. The byte offset field is 3 bits since the block size is 8 bytes. We now consider each of the six instruction fetches individually. Note that we write out the set field value and the byte field value in binary format.

i. Address \( 0x4000ff04 \). Set field 111100000. Byte field 100. Hence, we get a cache miss and we load the two instructions in the set.

ii. Address \( 0x4000ff08 \). Set field 111100001. Byte field 000. We are accessing a new set. Consequently, a new cache miss.

iii. Address \( 0x4000ff0c \). Set field 111100001. Byte field 100. We are accessing the same set as for the instruction at address \( 0x4000ff08 \). Hence, we get a cache hit.

iv. Address \( 0x4000ff10 \). Set field 111100010. Byte field 000. We are accessing a new set. Consequently, a new cache miss.
v. Address 0x4000ff14. Set field 111100010. Byte field 100. We are accessing the same set as for the instruction at address 0x4000ff10. Hence, we get a cache hit.

vi. Address 0x4000ff18. Set field 111100011. Byte field 000. We are accessing a new set. A new cache miss.

We have in total 6 memory accesses, 4 cache misses and 2 cache hits. Hence, the cache hit rate is \( \frac{2}{6} = \frac{1}{3} \).

Max 3 points. 3 points for the correct answer, else 0 points.

(b) Short answer: 32768 bytes.

Elaborated answer: The byte offset field is 3 bits \( (2^3 = 8) \). Hence, the set field is 32 − 19 − 3 = 10 bits. This means that there must be \( 2^{10} = 1024 \) sets. Since it is a 4-way set associative cache, we have \( 1024 \cdot 4 = 4096 \) blocks. One cache block is 8 bytes large. Hence, the total capacity is \( 4096 \cdot 8 = 32768 \) bytes.

Max 3 points. 3 points for correct answer, else 0 points.

(c) Short answer: It can result in both an instruction cache miss and a data cache miss.

The load word instruction can result in a data cache miss if the data that is addressed by register \( \text{t1} \) is not in the data cache. Also, when the \( \text{lw} \)-instruction is fetched before it is executed, fetching the instruction can result in a instruction cache miss.

Max 2 points. 1 point if the correct answer that it can result in both an instruction cache and a data cache miss. 1 point of the motivation is OK.

6. Module 6: Parallel Processors and Programs

(a) Short answer:

- False. VLIW uses static scheduling that is done by the compiler at compile time. The processor is not performing the scheduling.
- True.
- False. A semaphore is used for synchronization in concurrent programming.
- False. Hardware multithreading means that the processor switches between different hardware threads. The technique does not fetch multiple instructions. Instead, the technique can improve performance by hiding latencies.
- True.

Max 5 points. 1 point for each correct answer, including motivation if the answer is false.

(b) Short answer: The speedup is 2.

Elaborated answer: In this exercise you should use Amdahl’s law. We will use milliseconds (ms) in all the calculations. In the case of 2 cores, Amdahl’s law gives:

\[
1.5 = \frac{\frac{1200}{x}}{\frac{1200}{x} + \frac{1200}{1200-x}}
\]

where \( x \) is the amount of time it takes to execute the parallel part of the program. If we solve for \( x \), we get \( x = 800 \). We can now compute the speedup for 4 cores directly:

\[
\frac{\frac{1200}{800}}{\frac{1200}{800} + \frac{1200}{1200-800}} = 2.
\]

Max 3 points. 3 points if the correct answer is given, else 0 points.
Part II: Advanced

7. The following program shows a corrected version of foo.

```c
void foo(int* d, int s){
    int t;
    for(int i = 0; i < s; i++){
        for(int j = i+1; j < s; j++){
            if(d[i] > d[j]){
                t = d[i];
                d[i] = d[j];
                d[j] = t;
            }
        }
    }
}
```

(a) The first error that gives the compilation error is that parameter pointer d on line 1 in the incorrect code is const. This means that the pointer is pointing to a memory area that is not allowed to be updated in function foo. However, on lines 7 and 8, the function is writing to that memory content. Hence, the const keyword should be removed.

The second error is on line 4. The comparison should be j < s. Otherwise, there will be an out-of-bound error when accessing array d.

The third error is on lines 6 to 8. There should be curly brackets around these lines of code. Right now, only line 6 is part of the if-expression.

Max 9 points. For each error, give 1 point if the correct line and item are given, 1 point if it is explained what the error means, and 1 point for explaining how it can be solved.

(b) The purpose of the foo-function is to sort an array of integers. Hence, after executing a correct version of the foo-function, the output will be:

2, 3, 8, 12, 32, 58, 99,

Max 4 points. 2 points for pointing out that the foo-function sorts the array and 2 points for giving the correct output.

(c) The data cache is direct mapped. Hence, we have a block size of 4096/512 = 8 bytes. Since there are 512 sets, the set field size of an address is 9 bits. The byte offset field is 3 bits. If we consider address 0x40003320 and the fact that our array is 7 \cdot 4 = 28 bytes long, we can see that there cannot be any conflicting reads or writes that invalidate any of the cache blocks reading or writing to the array. If we add value 28 (in decimal) to the address, we get address 0x4000333C. Since the tag contains the 32 − 9 − 3 = 19 most significant bits, we will always use the same tag for all accesses. Hence, there will be no data conflict and once a block is loaded into the cache, it will stay there during the whole execution of the function.

We can directly see that we will access all the elements in the array. Because the block size is 8 bytes, we will load 2 integers at each cache miss. We get therefore 4 cache misses in total. The next step is to calculate the total number of memory accesses. In the inner loop, we will always do two memory accesses in the guard of
the if-statement. The outer loop that uses variable $i$ will perform 7 iterations. The inner loop that uses variable $j$ will first do 6 iterations, then 5, 4, 3, 2, and finally 1 iteration. This means that the if-statement will be executed $6+5+4+3+2+1 = 21$ times (note that the last time the inner for-loop is executed, the if-statement will not be executed). Hence, for the guard of the if-statement, we will have $21 \cdot 2 = 42$ memory accesses.

If the body of the if-statement is executed, we will have 4 memory accesses. However, to know if the body is executed, we need to look at how the elements in the array are swapped.

- Array at start: $\{12, 8, 2, 58, 3, 99, 32\}$
- After outer loop 1: $\{2, 12, 8, 58, 3, 99, 32\}$. Results in 2 swaps.
- After outer loop 2: $\{2, 3, 12, 58, 8, 99, 32\}$. Results in 2 swaps.
- After outer loop 3: $\{2, 3, 8, 58, 12, 99, 32\}$. Results in 1 swap.
- After outer loop 4: $\{2, 3, 8, 12, 58, 99, 32\}$. Results in 1 swap.
- After outer loop 4: $\{2, 3, 8, 12, 32, 99, 58\}$. Results in 1 swap.
- After outer loop 4: $\{2, 3, 8, 12, 32, 58, 99\}$. Results in 1 swap.

In total, we will perform 8 swaps. This means that the body of the if-statement will do $8 \cdot 4 = 32$ memory accesses. Hence, in total, we have $42 + 32 = 74$ memory accesses. Because we had 4 misses, we have $74 - 4 = 70$ cache hits. As a consequence, the cache hit rate is $\frac{70}{74}$.

Max 7 points. 7 points if the correct result. Max 3 points if only the cache misses were calculated correctly.

8. We do not provide any actual concrete solution for this exercise because the explanations can vary significantly. Please see the lecture slides and the course book for more information about the different concepts.

Max 20 points. At most 5 points for each item. To get full points, all questions need to be discussed in detail, the concepts need to be explained, and a clear example must be given for each item.
9. The following MIPS code shows a possible solution:

```mips
fact:
    beq $zero, $a0, turn
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $a0, 0($sp)
    addi $a0, $a0, -1
    jal fact
    lw $a0, 0($sp)
    mulu $v0, $v0, $a0
    lw $ra, 4($sp)
    addi $sp, $sp, 8
    jr $ra

turn:
    addi $v0, $zero, 1
    jr $ra
```

Max 10 points. Up to 5 points for correct handling of the stack. Up to 5 points for implementing the recursion correctly. Note that there are several solutions that are correct. If the program is using instruction `mul` instead of `mulu` it is also counted as a valid solution. Zero points are given for a solution that is not recursive.