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

Suggested Solutions

Part I: Fundamentals[1]

In part I, on the real exam, only short answers are expected. The elaborated answers given here are just for your information, and are not needed on the real exam.

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

   (a) Short answer: Register `$t2` with value 48.

   Elaborated answer: The assembly code is:

   ```
   sllv  $t2,$t0,$s1
   ```

   Shifting the value of `$t0` three bits to the left is the same as $6 \cdot 8 = 48$.

   Max 4 points. Two points for the correct register and two points for the correct value. The answer can be given in decimal form (48) or in hexadecimal form (0x30).

   (b) Short answer: `56e11884, -4, 12, 13`

   Max 4 points. One point for each correct number.

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

   (a) Short answer:

```
twinkle:
  lui   $t0,0xb800          # Setup port address
  ori   $t0,$t0,0x4d00

  add   $t1,$zero,$ra       # Save the return address
  addi  $t2,$zero,0x100     # Initiate 1 to LED 3 (index 8)
  addi  $t3,$zero,0x0       # Set sum up variable to zero
loop:
  sw    $t2,0($t0)          # Turn on / off the LED 3
  xori  $t2,$t2,0x100       # Flip the bit

  jal   pause               # Call pause
  add   $t3,$t3,$v0         # Add together time values

  addi  $a0,$a0,-1
```

[1] Update 2019-02-04: After careful analysis, the examiner decided to lower the number of points for pass for this specific exam in the fundamental part with 4 points, and with 4 points for all grades in the advanced part. The reason was that the exam was more time consuming and difficult compared to previous years.

```
bne   $a0,$zero,loop      # Repreat

addi  $v0,$t3,0           # Set return value
jr    $t1                 # Return using saved address
```

Max 6 points.

- 1 point for setting up the port address correctly.
- 1 point for writing correctly to the port address using `sw`.
- 1 point for correctly flipping the bit with `xori`, including initiating the register correctly (it is fine if the LED starts as either being turned on or off).
- 1 point for calling pause correctly and adding together time values, including initializing the sum together register.
- 1 point for looping correctly the number of times the argument states.
- 1 point for i) returning correctly by also saving the return address (`ra` is overwritten by calling `pause`), and ii) for returning the correct value in `$v0` and not destroying the values in the `s` registers.

Note that nothing is mentioned about constraints on the input value. Hence, the function only needs to behave well for positive input values larger than zero[2].

(b) Short answer: $75\,000\,000$

Elaborated answer: Using prescaling `1:4` and the knowledge that the clock ticks $100\,000\,000$ times very second, the timer will tick $100\,000\,000/4 = 25\,000\,000$ times every second. Since we want the period to be 3 seconds, the period register must be set to $25\,000\,000 \cdot 3 = 75\,000\,000$.

Max 2 points. Two points for the correct value.

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

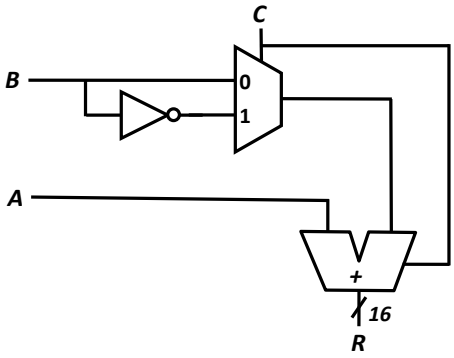
(a) Short answers:

- 64 bits
- `unknown`
- 0
- 7

Max 4 points. 1 point for each correct answer.

[2]Update 2019-06-04: A valid solution can also include a solution for the case of a zero value input, but it is not required for full points. Also, note that the above solution does not preserve register `$a0` when function `pause` is called. A more complete solution would also preserve `$a0`. However, this is not required to get full points on this exercise.

(b) Short answers:



Max 4 points. 1 point for drawing the carry propagate adder correctly, including the 16-bit bus. 1 point for getting the multiplexer correct, including the inverter signal from signal B. 1 point for getting signal C to the carry-in signal to the adder. 1 final point if everything else is correct, i.e., that the circuit works exactly as specified. Remove this last point if there are some minor errors. If the circuit only performs the addition, max 1 point can be given in total. If the circuit only performs subtraction, max 2 points can be given in total. It is OK to use other symbols than the standard symbols, as long as it is clearly described what they mean.

4. **Module 4: Processor Design**

(a) Short answer: $A =$ `0xfffffffec`, $B =$ `0x3ff8`, $C =$ `0x12`, $D =$ `0x40224`, and $E =$ `0x4000`.

Max 5 points. One point for each correct answer.

(b) Short answer:

i. One data hazard and one control hazard.

ii. The `lw` and `beq` instructions.

iii. 3 clock cycles.

Elaborated answer: It is 3 clock cycles because: i) one penalty clock cycle if the branch is taken, and ii) two clock cycle stalls are needed before the memory result in the `lw` instruction can be forwarded to the decode stage for the `beq` instruction.

Max 3 points. One point for each correct answer.

5. **Module 5: Memory Hierarchy**

(a) Short answer: i) 512 valid bits, and ii) 22 bits

Elaborated answer:

i. There are as many valid bits as there are blocks. We have $4096/8 = 512$ blocks. Hence, there are 512 valid bits.

ii. If we divide the number of blocks with the number of ways, we get the number of sets. Hence, we have $512/4 = 128$ sets. The set field requires 7 bits because $2^7 = 128$. In the same way, the byte index field size is 3 bits because $2^3 = 8$. Hence, the tag field size is $32 - 7 - 3 = 22$ bits.

Max 2 points. One point for each correct answer.

(b) Short answers: i) 60%, ii) 10%, iii) spatial locality, and iv) both temporal and spatial locality.

Elaborated answer:

i. The loop loops 5 times. Hence, there are in total 5 memory access to the data memory via the `lw` instruction. The data block size is 8 bytes and the first memory access is at address `0x3b2214` (note the index address at the `lw` instruction). Since the memory address is incremented by 4 in each iteration, we get the following sequence: 1. cache miss, 2. cache miss, 3. cache hit, 4. cache miss, 5. cache hit. Hence, we have 3 cache misses and 5 memory accesses. We get a data cache miss rate of $\frac{3}{5}$ or 60%.

ii. There are 5 instructions before the loop and there are 5 instructions inside the loop, and the loop loops 5 times. Hence, $5 + 5 \cdot 5 = 30$ instructions need to be loaded, and we therefore have 30 memory accesses from the instruction memory. The capacity of the instruction cache is 4096 bytes and we have 256 blocks. Hence, the block size is $4096/256 = 16$ bytes. The first memory access is at address `0x00040100`. Hence, we get a cache miss when we access `lui`. The following three accesses are cache hits. Then, when we load the `addi` instruction before the `loop` label, we get a cache miss again. We get 3 more cache hits, and then a cache miss again the last time. After that we, only get cache hits. Hence, we have an instruction cache miss rate of $\frac{3}{30} = \frac{1}{10}$ or 10%.

iii. The data instructions never read from the same memory address. Every second memory access, we get a hit. Hence, it uses spatial locality but not temporal locality.

iv. In the instruction cache case, we can observe both temporal and spatial locality. We always read 4 instructions, which are used when executing several instructions in a row. We can also see temporal locality, since we are executing the same loop 5 times.

Max 6 points. For the two first questions: two points each for a correct answer. Zero points if not correct. For the two last questions: One point each for a correct answer. No explanation is needed.

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

(a) Short answers:

i. True.

ii. False. Hardware threads are scheduled by the hardware and not by the operating system.

iii. False. Hyper-threading is another name for simultaneous multithreading, whereas VLIW is a form of multiple issue.

iv. True.

v. False. SIMD is used when there is data-level parallelism.

Max 5 points. One point for each correct answer.

(b) Short answers: 5 seconds.

Elaborated answer: For 4 cores, we have the speedup $\frac{12}{6} = 2$. Hence, if we let x be the time of the original execution time that is possible to parallelize, we have Amdahl's law:

$$2 = \frac{12}{\frac{x}{4} + 12 - x} \tag{1}$$

If we solve for x, we get $x = 8$. Finally, the estimated execution time for 8 cores are:

$$\frac{8}{8} + 12 - 8 = 5 \text{ seconds} \tag{2}$$

Part II: Advanced

7. The complete solution is omitted. Please see the lecture slides and the course book.

   Max 15 points. For each of the three items, max three points for the explanations of the two concepts, max one point for at least one difference, and max one point for at least one similarity. That is, max five points for each of the three items.

8. (a)
```
int boo(int i, int k){
  if(i == -1 || vlist[k] < vlist[i]){
    plist[k] = i;
    return k;
  }
  int next = boo(plist[i],k);
  plist[i] = next;
  return i;
}


int foo(int p){
  for(int k=0; k<5; k++){
    p = boo(p,k);
  }
  return p;
}
```

   (b) The following line is printed to standard output:

```
1, 2, 7, 35, 100,
```

   The functions shows a simple variant of insertion sort, implemented as a linked list using array indices. Function `foo` is the main sort function and function `boo` is the insertion function.

   A detailed solution is omitted.

   Max 25 points in total. 15 points for the translation to C and 10 points for the explanation.

9. First, we can observe that both arrays `plist` and `vlist` start on even addresses and that the block size is 8 bytes. We only need to consider the memory accesses to `sw` and `lw`. This solution assumes a write-back policy, but a solution that assumes write-through is also considered correct if the solution is described correctly.

   We can first analyze the flow of the program. In the loop of `foo`, function `boo` will first be called with the second argument with value 0, and then with value 1. We need to analyze both the first and second time boo is called, because the content of the cache is affected in the first call. We can also observe that we do not need to consider the pushing to the stack in `foo`, since it affects one block (8 bytes) that will not affect `boo`.

   The first time `boo` is called, `$a0=-1` and `$a1=0`. A complete cache block is fetched when storing 8 bytes on the stack. Since `$a0` has value -1, it jumps to `ortrue`. The `sw` stores to the first word in array `plist`. Hence, the first 8 bytes in the `plist` array are loaded to the cache. This is followed by popping from the cache (8 bytes). The function `boo` returns value 0 in `$v0`. Note that in this first call, we are only interested in if there are some blocks fetched to the cache. We have not started to count memory accesses yet. We can conclude that the first 8 bytes on the stack (stack pointer -8) and the first 8 bytes of `plist` are in the cache.

   The second time `boo` is called, its arguments are `$a0=0` and `$a1=1`. The first two `sw` instructions result in two memory accesses, but since this block is already in the cache, there are no cache misses. This time the first argument is not -1. Hence, we have two memory accesses to `vlist` (the two `lw` instructions before label `ortrue`). Note that the `vlist` address is $4 \cdot 5 = 20$ bytes after the `plist` address, which means that the first element of `vlist` is at address `0x02030014`. This will result in two memory accesses, where both are cache misses (two different blocks). In the next call, we have a load word from `plist`, at index 0, which is a hit (because of the last call to `boo`). In the first part of the second call to `boo`, we have $2 + 2 + 1 = 5$ memory accesses and 2 cache misses.

   In the recursive call to `boo`, we have two more memory accesses via the stack, where the first one is a miss and the second one a hit. This time, `boo` is called with `$a0=-1` and `$a1=1`. Since the first argument is -1, it jumps to `ortrue`, stores at the index 4 in `plist`, which is one memory access with a cache hit. It pops from the stack, another two memory accesses, and two cache hits. In total, during the recursive call, we have $2 + 1 + 2 = 5$ memory accesses and 1 cache miss.

   When the recursive call returns, it stores a word into index 0 in array `plist`, which is a cache hit. It then returns, by popping the stack using two `lw` instructions. Hence, there are two more memory accesses with two cache hits. In this last part of the second call to `foo`, we have $1 + 2 = 3$ memory accesses and 0 cache misses.

   In summary, we have $5 + 5 + 3 = 13$ memory accesses, and $2 + 1 = 3$ cache misses. Hence, the data cache miss rate for calling `boo` with the second argument 1, is $\frac{3}{13}$.

   Max 10 points in total.

Correction of the Exam

The examiner David Broman authored the exam questions, the correction guidelines, and the suggested solutions. The following persons took part in the correction: Saranya Natarajan, Viktor Palmkvist, Elias Castegren, Daniel Lundén, and Fredrik Lundevall.