Suggested Solutions

Part I: Fundamentals

1. Module 1: C and Assembly Programming
   
   (a) Short answer: Register $s0 gets the updated value 0x12a3ff11.

   Elaborated answer: The machine code 0x014b8026 corresponds to the assembly instruction `xor $s0,$t2,$t3`. This means that only $s0 is updated.

   Max 3 points. One point for stating that register $s0 is the updated register. Two points if the register value is completely correct.

   (b) Short answer: 23, 26, unknown, 8, unknown

   Advice: To understand the solution, please compile and test the program. Note that we cannot know what p1 points to, since it is pointing outside the array. Note also that it is not possible to know the pointer value p2.

   Max 5 points. One point for each correct answer.

2. Module 2: I/O Systems
   
   (a) Short answer: `*switches & 0x80`

   Max 2 points. One point for correct dereferencing of pointer. One point for the use of bitwise and, together with correct mask value. Note that a solution that combines shifting and masking can also be correct.

   (b) Short answer:

   ```
   lui $t0,0xabab
   loop:
   lw $t1,0x20($t0)
   andi $t2,$t1,0x80
   beq $t2,$zero,exit
   srl $t1,$t1,4
   andi $t1,$t1,1
   sw $t1,0x40($t0)
   j loop
   exit:
   ```

   Max 4 points. One point for correct handling of load word, including correct handling of addresses. One point for correct handling of the while statement, including guard, labels, and conditional branch. One point for correct shifting and masking before writing to the LED lights. One point for correct use of store word.
(c) Short answer:
   i. True.
   ii. True.
Max 2 points. One point for each correct answer.

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

(a) Short answer: 1024 bytes.
Each register value has the size of 4 bytes (32 bits). We have in total $2^8 = 256$ such values. Hence, the register file can hold $4 \cdot 256 = 1024$ bytes in total.
Max 2 points. Two points for the correct answer. One point if the number is correct, but the unit is incorrect.

(b) Short answer: 0xfec
Elaborated answer: Value 20 is first converted to the binary number 0000 0001 0100. We first invert all bits 1111 1110 1011, and then add one, which gives 1111 1110 1100. That is, 0xfec in hexadecimal form.
Max 2 points. Two points for the correct answer.

(c) Short answer:
\[
\begin{array}{cccccccc}
A_1 & A_0 & B & C_1 & C_0 & Y_1 & Y_0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]
Max 4 points. One point for each correct row.

4. Module 4: Processor Design

(a) Short answer: $A = 8$, $B = 16$, $C = 0$, $D = \text{unknown}$, and $E = 0x40001488$.
Elaborated answer: The destination register is $t0$, which means that signal $A = 8$. The $rs$ field at bits 25:21 is in this case $s0$, which means that $B = 16$. Since this is not a branch instruction, $C = 0$. The value that is being written to the register is the sum of the immediate value and the value of register $s0$. However, because we do not know the value of $s0$, signal $D = \text{unknown}$. Finally, if we take the immediate value $0x121$, and shift it two bits to the left, we get $0x484$. Hence, by adding the program counter plus four, we get $E = 0x40001488$. Note that the answers can be given as either binary, decimal, or hexadecimal values.
Max 5 points. One point for each correct signal value.

(b) Short answer: addi $t3, s1, 0$. The first hazard can be solved using a combination of forwarding and stalling, whereas the second hazard can be solved using only forwarding.
Elaborated answer: If we choose an instruction that reads from $s1$, we get an hazard that needs to be solved using forwarding and stalling. Also, if we write to register $t3$, we get a data hazard to the \texttt{xor} instruction. Note that there are several correct solutions for this exercise.
Max 3 points. One point if the instruction causes one hazard and two points if it causes both hazards. One point if the description of forwarding and stalling is correct.

5. Module 5: Memory Hierarchy

(a) Short answer: The tag field is 6 bits and the set field is 5 bits.
Elaborated answer: The byte offset field is \( n = 5 \) bits, where \( 2^n = 32 \). There are \( 4096/4/32 = 32 \) sets, which means that the set field is 5 bits. Hence, the tag field is \( 16 - 5 - 5 = 6 \) bits.
Max 2 points. One point for each correct number.

(b) Short answer: The instruction cache miss rate is 5% and the data cache miss rate is 25%. The instruction cache utilizes both temporal and spatial locality, whereas the data cache can only utilize spatial locality.
Elaborated answer: The first instruction starts at address \( 0\times00401008 \). This means that we first get a cache miss, followed by a cache hit for the instruction cache. Then, when the third instruction is loaded, we access a new block, which results in a cache miss. For the last two instructions, we access the same block and we will have cache hits. The cache hits in the first iteration are due to spatial locality. In the next 7 loop iterations, we will get instruction cache hits due to temporal locality. We have in total \( 8 \cdot 5 = 40 \) instruction memory accesses and there are in total 2 instruction cache misses. Hence, we have \( \frac{2}{40} = \frac{1}{20} = 5\% \) instruction miss rate.
The first time the load byte instruction is executed, it access address \( 0\times60000002 \) (note the subtraction \(-4\) in the instruction), which will result in a cache miss. Since the block size is 8 bytes, and we have accessed the third byte in the block, this will follow by 5 data cache hits. In the 7th loop iteration, a new cache block is accessed, which will lead to a second cache miss. The last memory access is in the same block, which means that the last access is a hit. Hence, there are in total 8 data memory accesses and 2 cache misses. The data cache miss rate is therefore \( \frac{2}{8} = 25\% \).
Max 6 points. Two points for correct instruction cache miss rate and two points for correct data cache miss rate. One point for the correct locality of the instruction cache and one point for the locality of the data cache.

6. Module 6: Parallel Processors and Programs

(a) Short answer: 6 cores are needed.
Elaborated answer: By using Amdahl’s law, we can compute that the sequential part of the program takes \( 40/10 = 4 \) seconds. This means that the part that can be parallelized takes \( 40 - 4 = 36 \) seconds when executed on one core. By Amdahl’s law, we have

\[
4 = \frac{40}{\frac{36}{n} + 4}
\]

where \( n \) is the number of cores needed for the speedup of 4. Hence, we can directly compute that \( n = 6 \).
Max 3 points. Three points for the correct result value, otherwise zero points.
(b) Short answer:

i. True.

ii. False. Register renaming is used for solving hazards due to out-of-order execution.

iii. False. Today, it is very common with many more pipeline stages, e.g., 14 pipeline stages for an Intel Core i5.

iv. False. AVX is used for data-level parallelism.

v. False. Fine-grained multithreading is not related to the cache coherence problem. The cache coherence problem appears in a multicore processor with separate caches.

Max 5 points. One point for each correct answer, together with relevant explanations for the false answers.
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 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) A possible solution of foo1 in MIPS code.

   ```mips
   foo1:  
   addi $a0, $a0, 4
   loop:
   lw $t0, 0($a0)
   beq $t0, $zero, exit
   lw $t1, -4($a0)
   sub $t0, $t0, $t1
   sll $t0, $t0, 2
   add $t0, $t0, $a1
   lw $t1, 0($t0)
   addi $t1, $t1, 1
   sw $t1, 0($t0)
   addi $a0, $a0, 4
   j loop
   exit:
   jr $ra
   ```

   Max 8 points. Max two points for initiating $a0 correctly before the loop and to return correctly using jr. Max two points for handling the while-loop correctly, including the lw, to branch out of the loop correctly, and to increment the address correctly (adding by 4). Max two points for calculating the address to be modified. Finally, max two points for loading, incrementing, and storing back correctly.

(b) A possible solution for foo2 in C.

```c
void foo2(int* lst, int n){
    int i, j;
    int next = 2;
    for(i=0; i<n; i++){
        for(j=2; j<next; j++){
            if((next/j)*j == next){
                j=2;
                next++;
            }
        }
        lst[i] = next;
        next++;
    }
    lst[i] = 0;
}
```

Max 12 points. Two points for a completely correct signature. Max four points for correctly writing out the outer for-loop, and max six points for the inner for-loop. Remove one point for each mistake within each of the correcting parts. For instance, the points for the inner loop cannot be less than 0. Zero points in total for a solution that exceeds the 20 lines limit.
9. The function is called with \( n \) equal to 2. First, we need to count the number of instructions that are executed. The easiest way to do this is to write down the register values for each step in the program. By doing this, we can count that the program is executing 39 instructions in total.

The function contains in total 27 instruction (addi at label \texttt{foo2} to the \texttt{jr} instruction). Since the first instruction starts at address 0x00400108, we can easily see that the program will access four consecutive instruction cache blocks in total, where the first block starts at address 0x00400100 (each block can hold 8 instructions). The instruction cache has \( 4096/32 = 128 \) sets, which means that all four consecutive blocks can be in the cache at the same time. Hence, there will in total be four instruction cache misses, one for each miss in one of the four blocks. Hence, the extra instruction cache miss penalty is \( 4 \cdot 10 = 40 \) cycles.

By following the program flow, we can see that there are in total three memory accesses, using the \texttt{sw} instruction. The first time it writes a value to the address of \texttt{data1}, the second time to the address of \texttt{data1}+4, and the third time to the address of \texttt{data1}+8. The exercise does not say which address \texttt{data1} points to. However, since the block size is 8 bytes, it gives the same number of cache hits and misses regardless if the first write is at the first word of a block (miss, hit, miss), or the second word of the block (miss, miss, hit). In either case, we get two data cache misses, which gives the extra penalty of 20 clock cycles.

In summer, executing 39 instructions on a single-cycle processor takes 39 clock cycles, plus 40 cycles penalty for instruction cache misses, and 20 cycles penalty for data cache misses. This means that the total number of cycles are \( 39 + 40 + 20 = 99 \) clock cycles. Hence, the CPI is \( \frac{99}{39} \).

Max 10 points. Max five points for calculating the right number of instructions, and max five points for the correct number of cycles. In the latter case, we give max three points for calculating the data cache miss penalty, and max two points for calculating the instruction cache miss penalty.

10. Function \texttt{foo1} goes through a null-terminated array of integers and counts the frequency of the distances between consecutive integers in the array. The resulting frequency table is stored in the array, which is pointed to by \texttt{*dist}.

Function \texttt{foo2} generates a list of prime numbers. The first parameter of the function is a pointer that points to where the output list will be stored. The second parameter is an integer that states how many prime numbers that should be generated.

Function \texttt{foo3} first clears \( n \) number of elements (sets the array elements to zero). The two last lines first generates \( n \) number of prime numbers and then counts the frequency of distances between these prime numbers.

Max 5 points. Max two points for explaining \texttt{foo1}, max two points for explaining \texttt{foo2}, and max one point for \texttt{foo3}.  

6