Part I: Fundamentals

1. Module 1: C and Assembly Programming

   (a) Short answer:

   ```c
   s = s + *buf;  //Code line 1
   buf++;         //Code line 2
   sum(bufptr,&res,buflen); //Code line 3
   ```

   Note that code lines 1 and 2 can also be written on one line of code:
   ```c
   s = s + *buf++;  
   ```
   or, alternatively, as:
   ```c
   s = s + *(buf+i);  
   ```

   Note also that it is perfectly fine to use `data` directly, instead of `bufptr`.
   ```c
   sum(data,&res,buflen);
   ```

   Max 5 points. 2 points if code line 1 is completely correct. Note that you can also write `s += *buf`. 1 point if code line 2 is correct. 2 points if code line 3 is completely correct and 1 point if it is correct besides the reference to `res`.

   (b) Short answer: 0x8d110008

   Elaborated answer: We need to use the load word instruction, which uses `$t0` for the base address, with index 8. Hence, we need to construct the machine code for instruction:

   ```
   lw  $s1,8($t0)
   ```

   Max 3 points. 3 points if the whole machine code is correct. 1 point if only the 16 least significant bits are correct.

2. Module 2: I/O Systems

   (a) Short answer:

   ```
   readval:
   lui     $t0,0xffff
   lw      $v0,0x44($t0)
   srl     $v0,$v0,4
   andi    $v0,$v0,0xff
   jr       $ra
   ```
Max 4 points. 1 point for setting up and using the correct address (can also be done by lui followed by ori). 1 point for loading the correct value using lw. 1 point for shifting and masking correctly (using srl together with and or orandi). 1 point for declaring the label correctly, using jr correctly, and returning the value in $v0. If for instance register $s0 is used without pushing the existing value on the stack, one point should be removed from the overall score. The minimal number of points for this exercise is zero.

(b) Short answer: True, True
Max 2 points. One point for each correct statement.

(c) Short answer: 120 MHz
Elaborated answer: Let $f$ be the clock frequency of the clock source. Without prescaling option, the timer would be triggered $\frac{f}{30\,000}$ times every second, and with the prescaling option 1 : 4, the timer would be triggered $\frac{f}{30\,000\cdot 4}$ times every second. Now, we know that the timer is triggered every millisecond, which means that is will be triggered 1000 times every second. As a consequence, we have the equation $\frac{f}{30\,000\cdot 4} = 1000$. Hence, we have $f = 30\,000\cdot 4\cdot 1000 = 120\,000\,000$ Hz = 120 MHz.
Max 2 points. Two points for correct answer. You may answer in Hz, kHz, or MHz.

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

(a) Short answer:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$U_{pre}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Max 4 points. One point for each correct row.

(b) Short answer: $\overline{A} \overline{B} \overline{C} + \overline{A} B C + A B C$
Elaborated answer: The solution can be obtained directly by writing the boolean expression in sum-of-products form. The answer can be further simplified to e.g., $B C + \overline{A} B \overline{C}$ by using a combination of the laws of distributivity, complements, and identity. Any valid boolean notation is accepted (including C-expressions).
Max 2 points. Two points for any valid boolean expression. That is, the expression does not need to be simplified.

(c) Short answer: 2048 bytes.
Elaborated answer: The address width is 10 bits, which means that the register file can store $2^{10} = 1024$ registers. Each register can store 16 bits = 2 bytes. Hence, in total, we have $1024 \cdot 2 = 2048$ bytes.
Max 2 points. Two points for a correct answer.
4. Module 4: Processor Design

(a) Short answer:
   i. Control hazard
   ii. 0x4
   iii. no

Elaborated answer:
   i. Control hazard. There are no data dependencies between the different registers. Hence, we do not have any data hazards.
   ii. 0x4. Since we have a pipelined processor with branch delay slots, the instruction coming after the branch will always be executed, regardless if the branch is taken or not. The ori instruction writes value 0x4 to $s0, regardless if the branch is taken or not.
   iii. Since we do not have any data hazards, there is no need to use forwarding.

Max 3 points. One point for each correct answer.

(b) Short answer: $A = 0xb, B = 0x305502d4, C = unknown, D = 0xffffffffe4, and $t3 = unknown.$

Elaborated answer: Signal $A$ is the register number of the destination register. In this case, this is register $t3$, which means that $A = 11$, i.e., the same as $A = 0xb$. Signal $B$ is the result of the ALU operation. For the load byte instruction, it adds together the immediate value -7 and the base address, which is located in register $s0$. Hence $B = 0x305502db - 0x7 = 0x305502d4$. Signal $C$ is the value that is read from memory. We cannot know this value based on the provided information. Hence, $C = unknown$. Signal $D$ is the immediate value, shifted two bits to the left. First, we need to know the hexadecimal value for -7. We can get this value by performing two's complement. It is enough if we perform two's complement on a byte, shift it, and then sign extend it to a word. The binary value for 7 is 00000111. If we invert all bits and add one, we get 11111001. We should also shift left by two bits, which gives 11100100. This is the same as the byte 0xe4. Finally, we know that all higher bits should be 1, which means that $D = 0xffffffffe4$. Finally, after that the load instruction has executed, register $t3$ will contain the value read from memory. However, with the provided information, we do not know this value. Hence, $t3 = unknown$.

Note that the given datapath can handle lw instructions, but only partially handle lb instructions. The reason is that a load byte instruction also needs to perform sign extension on the 8 least significant bits of the returned word. However, since signal $C$ is unknown anyway, this fact does not change the answers to the exercise.

Max 5 points. One point for each correct value. In the question, we asked for hexadecimal numbers, but if the numbers are correctly given as binary or decimal numbers, it will also be counted. However, in such a case, it must be clear which base is used.
5. Module 5: Memory Hierarchy

(a) Short answer:
   i. 8 bytes.
   ii. For the selected set, the valid bit will be set to value 1, the tag will be updated, and the cache block will be updated.

   Clarification: The block size is \( \frac{1024}{128} = 8 \) bytes. Note that when a cache miss occurs, a whole cache block is always fetched from the main memory.

   Max 3 points. 1 point for the correct number of bytes. 2 points if all three parts are enumerated (valid bit, tag, and data block). 1 point if only two items are given.

(b) Short answer: The block size is 16 bytes, there are 256 valid bits, and it is a 2-way set associative cache.

   Elaborated answer: The byte offset field is \( 32 - 7 - 21 = 4 \) bits. Hence, the block size is 16 bytes. There are in total \( \frac{4096}{16} = 256 \) blocks and there is one valid bit associated with each block. Hence, the cache contains 256 valid bits. Finally, the set field is 7 bits, which means that we have \( 2^7 = 128 \) sets. Hence, \( N = \frac{256}{128} = 2 \). That is, it is a 2-way set associative cache.

   Max 3 points. One point for each correct answer.

(c) Short answer: The cache hit rate is 97%.

   Elaborated answer: We have \( \frac{2048}{16} = 128 \) sets and the first instruction starts at the even address 0x00400200. Hence, we know that after the first iteration, we will always get cache hits. In the first iteration, for the 10 memory accesses, we will get the sequence: miss, hit, hit, hit, miss, hit, hit, miss, hit. Hence, we have 7 cache hits during the first iteration. We have in total \( 10 \cdot 10 = 100 \) memory accesses, one access for each instruction fetch. We have \( 7 + 9 \cdot 10 = 97 \) cache hits. Hence, the cache hit rate is \( \frac{97}{100} \), or 97%.

   Max 2 points. Two points for the correct answer, otherwise zero points.

6. Module 6: Parallel Processors and Programs

(a) Short answer:
   i. False. Weak scaling means that the data size is also increased when the number of processors/cores are increased.
   ii. True.
   iii. True.
   iv. True.
   v. False. Hyper-threading is Intel’s name and implementation of simultaneous multithreading (SMT).

   Max 5 points. One point for each correct answer, together with relevant explanations for the false answers.
(b) Short answer: The speedup would be 6.

Elaborated answer: Let $T$ be the time when the program executes on one core, and let $p$ be the proportion of the time that is sequential, which cannot be parallelized. Using Amdahl’s law, we have for the 4-core machine with a speedup of 3 the following equation:

$$3 = \frac{T}{\frac{T(1-p)}{4} + T \cdot p} \quad (1)$$

If we simplify the right hand side by getting rid of $T$, and solve the equation for $p$, we get $p = \frac{1}{9}$. We can now use the law again, and compute the speedup for 16 cores:

$$\frac{T}{\frac{T(\frac{1}{16})}{16} + T \cdot \frac{1}{9} \cdot \frac{16}{9}} = 6 \quad (2)$$

Hence, we estimate that the speedup will be 6 when the program is executed on 16 cores.

Max 3 points. Three points for the correct result value, otherwise zero points.
Part II: Advanced

7. Potential answers:

- A direct mapped cache vs. a 4-way set associative cache.
  
  (a) Differences: A direct mapped cache has only one way, in contrast to a 4-way set associative cache. This means that a direct mapped cache can only hold one cache block for each of the sets in the cache memory at the same time. If the same set is accessed again with a different tag, that cache line is evicted. By contrast, a 4-way associative cache can hold 4 different cache blocks at the same time, each identified with its tag.
  
  (b) Pros and cons: A benefit with a direct mapped cache is its simplicity and low requirement of hardware, compared to a set associative cache. With the same cache capacity, a direct mapped cache can also hold more sets than a 4-way cache. However, a direct mapped cache can only hold one block for each set. This means, for instance, that two instructions that are accessing the same set with different tags, can evict each other. On the other hand, a 4-way set associative cache can store several blocks within the same set. If a set is full, a replacement policy is used (such as least recently used).

- A 30 MHz single-cycle processor without a pipeline vs. a 5-stage pipelined processor running at 80 MHz.
  
  (a) Differences: A single-cycle processor executes one instruction each cycle, whereas a pipelined processor may execute one instruction in one or more clock cycles. A pipelined processor can typically run at a higher clock frequency because the critical path in the processor is shorter (for each stage).
  
  (b) Pros and cons: The purpose of introducing a pipeline in a processor is to be able to shorten the critical path in the processor, thus enabling higher clock frequencies. However, by introducing pipeline stages, we introduce both data hazards and control hazards. The problem with hazards is that it can increase the CPI (clock cycles per instruction). However, as long as the data hazards can be solved using forwarding, and not stalls, the performance penalty is not too big. Likewise, control hazards can be mitigated by using clever branch predictors. The pros with pipelined processors, as with the 80 MHz processor in this case, is that it increases speed by increasing clock frequency. A benefit with single-cycle processors is the simplicity, both in terms of hardware, and for the purpose of teaching. However, single-cycle processors are not really used in real processors today.

- SIMD instructions vs. Multicore support.
  
  (a) Differences: SIMD (Single instruction, multiple data) instructions make use of data-level parallelism. These instructions perform the same kind of instruction (single instruction) on multiple data. In contrast, a multicore processor can handle both task-level parallelism and data-level parallelism.
  
  (b) Pros and cons: SIMD and multicore (MIMD) complement each other. For instance, a multicore processor typically has SIMD instructions that can execute
in parallel on each core. A benefit with multicore (MIMD) is that it is general and can handle both task-level and data-level parallelism. A benefit with SIMD instructions is that the communication cost between instructions are low, compared to the communication cost between cores. Communication costs can also vary quite a lot for multicores due to caches. A downside with SIMD is that only certain applications can make use of these instructions. Another downside with both SIMD instructions and multicore is that it is still fairly hard to really utilize and program these kind of system features efficiently.

Max 15 points. 5 points for each item. Within each item, we give max 2 points for explaining the differences between the concepts and max 3 points for giving relevant pros and cons.

8. Since the capacity is 4096 bytes and the block size is 16 bytes, the number of sets are 4096/16 = 256. The function contains in total 13 instructions, which means that the highest instruction address that will be accessed is 0x0040000c + 13 \cdot 4 = 0x00400040. Hence, there will not be any conflicts in the instruction cache. Also, we can notice that everytime we call fact recursively, we decrement the stack pointer with 8. We will do that 4 times. Hence, the data cache will not have any conflicts either.

Instruction cache: There is one memory access for each instruction that is executed. The code (6 instructions) up to and including the jal instruction is executed 4 times because of the recursive call. The fifth time, the branch is taken, and it jumps to turn, followed by two more instructions. This means that in the fifth iteration, there are 3 memory accesses. Then, when the functions return, the code (5 instructions) between jal and the turn label, will be executed 4 times. Hence, we have 6 \cdot 4 + 3 + 5 \cdot 4 = 47 instruction cache memory accesses.

The first instruction starts at address 0x0040000c. This means that we get a cache miss on instruction beq and then a new cache miss on the first addi. We will then get a new cache miss on jal. Note, however, that we have now loaded all instructions between jal and (including) the second lw-instruction. The next cache miss will be when we branch to turn. When this happen, we load the last code block. Hence, after that, we do not have any more cache misses. In total, we have 4 cache misses. As a consequence, the instruction cache miss rate is \(\frac{4}{47}\).

Data cache: We can directly see the we will call recursively fact four times, which means that we will perform 8 store word operations, and 8 load word operations. Hence, in total, we have 16 data accesses. When the function is first called, the stack pointer points to 0x7fffffff. After subtraction by 8, it points to 0x7fffffff4. The two first store words access the memory at addresses 0x7fffffff and 0x7fffffff8. This means that we have a cache miss in the first access. In the next call, we will access addresses 0x7fffffffec and 0x7fffffff0. Hence, we get another miss for the first one that uses set 0xfe, whereas the set 0xff is already loaded. In the third round, we access 0x7fffffff4 0x7fffffff. Hence, no cache misses. Finally, we access 0x7fffffffdec 0x7fffffff0, which gives one more cache miss. Consequently, we have in total 3 cache misses and 16 memory accesses, which gives data cache miss rate: \(\frac{3}{16}\). To conclude, the instruction cache miss rate is \(\frac{4}{47}\) and the data cache miss rate is \(\frac{3}{16}\).
Max 15 points. 8 points for calculating the data cache miss rate, and 7 points to calculate
the instruction cache miss rate.

9. The C function can be implemented as follows:

```c
void move(int *x, int *y, int *dirx, int *diry){
    if(*x==127)
        *dirx = -1;
    if(*y==63)
        *diry = -1;
    if(*x==0)
        *dirx = 1;
    if(*y==0)
        *diry = 1;
    *x += *dirx;
    *y += *diry;
}
```

The MIPS assembly function can be implemented as follows:

```assembly
.globl drawpix
drawpix:
    sll $a1,$a1,4  # Multiply Y by 16
    srl $t0,$a0,3  # Divide by 8 (for X)
    add $a1,$a1,$t0  # Compute the final byte index
    sll $t1,$t0,3
    sub $a0,$a0,$t1  # Compute the reminder, the bit index
    addi $t0,$zero,128  # Set the left most pixel in a byte to 1.
    srlv $t0,$t0,$a0  # Compute pixel to draw
    lui $t1,0x9000  # Get address to screen
    add $t1,$t1,$a1
    lbu $t2,0($t1)  # Load current pixel from the screen
    or $t2,$t2,$t0  # Update the new pixel in a temp variable
    sb $t2,0($t1)  # Update the new pixel on screen
    jr $ra
```

Max 20 points. 8 points for the C function. 12 points for the MIPS function. Remove one
point for every mistake in the code.