1. A certain benchmark contains 195578 floating-point operations with the details shown below.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>82014</td>
</tr>
<tr>
<td>Subtract</td>
<td>8229</td>
</tr>
<tr>
<td>Multiply</td>
<td>73220</td>
</tr>
<tr>
<td>Divide</td>
<td>21399</td>
</tr>
<tr>
<td>Convert int to fp</td>
<td>6006</td>
</tr>
<tr>
<td>Compare</td>
<td>4710</td>
</tr>
</tbody>
</table>

The benchmark was run on an embedded processor after compilation with optimizations turned on. The embedded processor is based on a current RISC processor that includes floating-point function units, but the embedded processor does not include floating-point for reasons of cost, power consumption, and lack of need for floating-point by the target applications.

The compiler allows floating-point instructions to be calculated with the hardware units or using software routines, depending on compiler flags. The benchmark took 1.08 seconds on the RISC processor and 13.6 seconds using software on its embedded version. Assume that the CPI using the RISC processor was measured to be 10, while that of the embedded version was measured to be 6.

A. What is the total number of instructions executed for both runs?
B. What is the MIPS rating for both runs?
C. On the average, how many integer instructions does it take to perform a floating-point operation in software?


2. Several researchers have suggested that adding a register-memory addressing mode to a load/store machine might be useful. The idea is to replace sequences of

```
LOAD    R1, 0(Rb)
ADD     R2, R2, R1
```

by

```
ADD     R2, 0(Rb)
```

Assume the new instruction will cause the clock cycle to increase by 5%. Use the following instruction frequencies for the gcc benchmark on the load/store machine. The new instruction affects only the clock cycle and not the CPI.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
<td>25.1%</td>
</tr>
<tr>
<td>store</td>
<td>13.2%</td>
</tr>
<tr>
<td>add</td>
<td>19.0%</td>
</tr>
<tr>
<td>sub</td>
<td>2.2%</td>
</tr>
<tr>
<td>mul</td>
<td>0.1%</td>
</tr>
<tr>
<td>compare</td>
<td>6.1%</td>
</tr>
<tr>
<td>load imm.</td>
<td>2.5%</td>
</tr>
<tr>
<td>cond. branch</td>
<td>12.1%</td>
</tr>
<tr>
<td>cond. move</td>
<td>0.6%</td>
</tr>
</tbody>
</table>
jump 0.7%
call 0.6%
return 0.6%
shift 1.1%
and 4.6%
or 8.5%
xor 2.1%
other logic 0.9%

What percentage of the loads must be eliminated for the machine with the new instruction to have at least the same performance?


3. Consider adding a new index addressing mode to MIPS. The addressing mode adds two registers and an 11-bit signed offset to get the effective address (notice that we have borrowed five bits from the immediate field to encode one of the registers). Our compiler will be changed so that code sequences of the form

```
ADD     R1, R1, R2
LW      Rd, 100(R1) /* Could be store also */
```

will be replaced by a load (or store) using the new addressing mode. Use the instruction frequencies from the last question to answer the following questions.

A. Assume that the addressing mode can be used for 10% of the loads and stores. What is the ratio of instruction count on the enhanced MIPS compared to the original MIPS?
B. If the new addressing mode lengthens the clock cycle by 5%, which machine will be faster and by how much?


4. Consider implementing a new instruction known as "find first set bit": ffsb r1, r2 which takes one register operand r2 as source and puts the position of the first least significant set bit in r1. For example, if r2 is 0x1234, after execution of this instruction r1 will contain 0x2 because the first least significant set bit is in position 2 (the least significant bit is in position 0).

A. In the absence of the ffsb instruction, how would you emulate it using the existing 32-bit MIPS ISA which has a branch delay slot? Your code need not be syntactically correct as long as it conveys the idea. Assume that the data path width is 32 bits.
B. Assuming that the bits in the source operand are distributed uniformly at
random, compute the expected number of cycles required to execute the emulated code. Assume that a shift instruction takes three cycles to execute while all other instructions take one cycle.

C. The compiler team tells you that a particular application that does a lot of bit manipulation can have ffsb as 10% of all instructions. The remaining 90% instructions (other than ffsb) exhibit an average CPI of 1.5. The ffsb instruction, when implemented with a highly parallel hardware, takes four cycles to execute. Compute the speedup achieved by implementing ffsb in hardware as compared to emulating it in software for this particular application.

Quadrant#2:

1. Short answer questions (use at most four sentences):
   (A) Why is branch prediction needed when a pipe is deepened or fetch width is increased?
   (B) Why are multiple issue and out-of-order issue important for performance?
   (C) Does it make sense to have branch prediction for a processor that does not have multiple issue or out-of-order issue?
   (D) Why is register renaming helpful?
   (E) Does it make sense to have register renaming for a processor that does not have multiple issue or out-of-order issue?
   (F) In a modern microprocessor there are three basic structures used for branch prediction. What are they? For each of them, also specify the types of branches which use that structure.
   (G) In a processor with register renaming, an instruction can write to the destination physical register as soon as it completes execution and need not wait till it retires. However, there remains a chance that this instruction may never retire. Doesn't it corrupt the register file?
   (H) In a processor with register renaming, a branch usually checkpoints only the map table, and not the values in the register file. Explain why it is sufficient to restore only the map table in case of a branch misprediction.
   (I) In a processor with register renaming, is there any need to have an ROB?
   (J) In a dynamically scheduled out-of-order issue microprocessor, the register file read stage is usually located after the instruction issue stage. Cite at least two reasons why it is not placed before the instruction issue stage.
   (K) A dynamically scheduled out-of-order issue microprocessor has 32 logical registers, 64 physical registers, and a 32-entry ROB. Will there be any stall due to full ROB? Explain.
   (L) We have seen the use of lui followed by xori for loading a 32-bit integer constant into a register in MIPS ISA. It is surprising that MIPS did not include an instruction like lui r2, r1, imm where the imm value is loaded in the upper half of r2 while the lower half of r1 is transferred to the lower half of r2. Is there a reason for omitting such an instruction?
   (M) A SAg branch predictor has h bit history and n entries in the first level BHT. The second level PHT is a bank of 2-bit saturating counters. A GAg predictor of the same total size also employs 2-bit saturating counters.
Compute the history length of the GAg predictor.

2. A microprocessor has a ten-stage pipeline: IF1, IF2, ID, RF1, RF2, EX, MEM1, MEM2, WB1, WB2. Enumerate the bypass paths needed in this processor. How big are the bypass multiplexers?

3. The Intel Pentium 4 has a retirement register alias table~(RRAT) in addition to the register map table. This table is same as the register map table maintained by the renamer, but is updated only when an instruction retires. So when the instruction add r1, r2, r3 retires it updates the physical register number for r1 to its destination. Explain why such a table might be needed. How does the MIPS R10000 function without such a table?

4. Consider the following code segment.

for(i=0;i<1000;i++) {
    if (i%2==0) {
        // S1 (arbitrary non-branch statements, not relevant)
    }
    else if (i%3==0) {
        // S2 (arbitrary non-branch statements, not relevant)
    }
}

Compiling this will generate three major conditional branches. Roughly, the control flow will look like this (this is not syntactically correct MIPS code).

Loop:       r1 = i%2
            bnez r1, label1    // B1
            S1
            j label2

label1:     r2 = i%3
            bnez r2, label2    // B2
            S2

label2:     i++
            slti r3, i, 1000
            bnez r3, Loop     // B3

This code is executed on two different processors with two different branch predictors. The first one has a bimodal predictor with 512 entries and each saturating counter is 2-bit wide. The second one has a SAg predictor where the first level table has 1024 entries and each entry can hold a 10-bit history. The second level table has 1024 saturating counters and each counter is 2-bit wide. Assume that all tables are initialized to zero and that each of the three branches (B1, B2, B3) gets a separate entry in the bimodal predictor and in the first level of SAg predictor. If the most significant bit of a saturating counter is 1, the branch is predicted taken while if it is zero, the branch is predicted not-taken. Assume that the predictors are updated immediately with the correct outcome i.e. assume a single-cycle pipeline. Compute the misprediction rate for each branch for each of the two predictors i.e. you should have six answers (please show all the intermediate steps).

5. Consider the following loop nest written in C.
for (i=0; i<M; i++) {
    // S1 (arbitrary non-branch statements, not relevant)
    for (j=0; j<N; j++) {
        // S2 (arbitrary non-branch statements, not relevant)
    }
}

How many mispredictions will a GAg branch predictor have for the branch corresponding to the inner loop if the history length is K? Propose a new branch predictor to significantly reduce this count while not having a history length bigger than K. Do not use more than ten sentences to describe your design. A diagram would be of great help.

Quadrants #3, #4:

1. Consider the following memory organization of a processor. The virtual address is 40 bits, the physical address is 32 bits, the page size is 8 KB. The processor has a 4-way set associative 128-entry TLB i.e. each way has 32 sets. Each page table entry is 32 bits in size. The processor also has a 2-way set associative 32 KB L1 cache with line size of 64 bytes.

   (A) What is the total size of the page table?

   (B) Clearly show (with the help of a diagram) the addressing scheme if the cache is virtually indexed and physically tagged. Your diagram should show the width of TLB and cache tags.

   (C) How many color bins must be implemented in OS to avoid synonyms? If the cache was 4-way set associative (keeping the total size and line size unchanged) how many color bins would be needed?

   (D) If the cache was physically indexed and physically tagged, what part of the addressing scheme would change?

2. (A) How does out-of-order load issue improve memory-level parallelism (MLP)?

   (B) An issuing load usually searches the store queue for an already store before it to the same address. Why is it needed? Is it needed even if the microprocessor does not issue loads out-of-order, but continues to do speculation on branches?

   (C) In a microprocessor employing out-of-order load issue, an issuing store searches the load queue for an already issued load after it to the same address. Why is it needed? Is it needed even if the microprocessor does not issue loads out-of-order, but continues to do speculation on branches?

3. (A) A set associative cache has longer hit time than an equally sized direct-mapped cache. Why?

   (B) What is a pseudo set associative cache? Why does it in general improve hit time? What is the cost you pay in exchange of that?

   (C) The Alpha 21264 has a virtually indexed virtually tagged instruction cache.
Do you see any security/protection issues with this? If yes, explain and offer a solution. How would you maintain correctness of such a cache in a multi-programmed environment?

(D) Given the ABCs of cache and a program, how would you measure the volume of capacity, conflict, and cold misses?

(E) What is a row buffer in a DRAM? How does the fast page mode access exploit the row buffers?

(F) In the MIPS R10000, the L2 cache tag stores extra two bits from the virtual index of the L1 cache. Why is it needed?

4. The Intel Pentium 4 does not have register map checkpoints. What options does such a processor have for recovering from branch mispredictions and exceptions?

5. Consider the following segment of C code for adding the elements in each column of an NxN matrix A and putting it in a vector x of size N.

```c
for(j=0;j<N;j++) {
  for(i=0;i<N;i++) {
    x[j] += A[i][j];
  }
}
```

Assume that the C compiler carries out a row-major layout of matrix A i.e. A[i][j] and A[i][j+1] are adjacent to each other in memory for all i and j in the legal range and A[i][N-1] and A[i+1][0] are adjacent to each other for all i in the legal range. Assume further that each element of A and x is a floating point double i.e. 8 bytes in size. This code is executed on a modern speculative out-of-order processor with the following memory hierarchy: page size 4 KB, fully associative 128-entry data TLB, 32 KB 2-way set associative single level data cache with 32 bytes line size, 256 MB DRAM. You may assume that the cache is virtually indexed and physically tagged, although this information is not needed to answer this question. For N=8192, compute the following (please show all the intermediate steps). Assume that every instruction hits in the instruction cache. Assume LRU replacement policy for physical page frames, TLB entries, and cache sets.

(A) Number of page faults.

(B) Number of data TLB misses.

(C) Number of data cache misses. Assume that x and A do not conflict with each other in the cache.

(D) At most how many memory operations can the processor overlap before coming
to a halt? Assume that the instruction selection logic (associated with the issue unit) gives priority to older instructions over younger instructions if both are ready to issue in a cycle.