DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
TIRUCHIRAPPALLI - 620 015, TAMIL NADU, INDIA

END SEMESTER EXAMINATION
CSPC51 – COMPUTER ARCHITECTURE

Branch/Semester/Sec : CSE/V/A Time: 2 PM to 5 PM
Date : 25.11.2024 Max Marks: 50

Note: Question no 1,2,3 are compulsory and 4 & 5 are Optional

1. a. When making changes to optimize part of a processor, it is often the case that speeding up one type of instruction comes at the cost of slowing down something else. For example, if we put in a complicated fast floating point unit, that takes space, and something might have to be moved farther away from the middle to accommodate it, adding an extra cycle in delay to reach that unit. The basic Amdahl's law equation does not take into account this trade-off.

i) If the new fast floating-point unit speeds up floating-point operations by, on average, 2×, and floating-point operations take 20% of the original program's execution time, what is the overall speedup (ignoring the penalty to any other instructions)?

ii) Now assume that speeding up the floating-point unit slowed down data cache accesses, resulting in a 1.5× slowdown (or 2/3 speedup). Data cache accesses consume 10% of the execution time. What is the overall speedup now?

iii) After implementing the new floating-point operations, what percentage of execution time is spent on floating-point operations? What percentage is spent on data cache accesses?

[5]

b. Assume that we are considering enhancing a machine by adding vector hardware to it. When a computation is run in vector mode on the vector hardware, it is 10 times faster than the normal mode of execution. We call the percentage of time that could be spent using vector mode the percentage of vectorization. Suppose you have measured the percentage of vectorization of the program to be 70%. The hardware design group estimates it can speed up the vector hardware even more with significant additional investment. You wonder whether the compiler crew could increase the percentage of vectorization, instead. What percentage of vectorization would the compiler team need to achieve in order to equal an addition 2× speedup in the vector unit (beyond the initial 10×)?

[5]

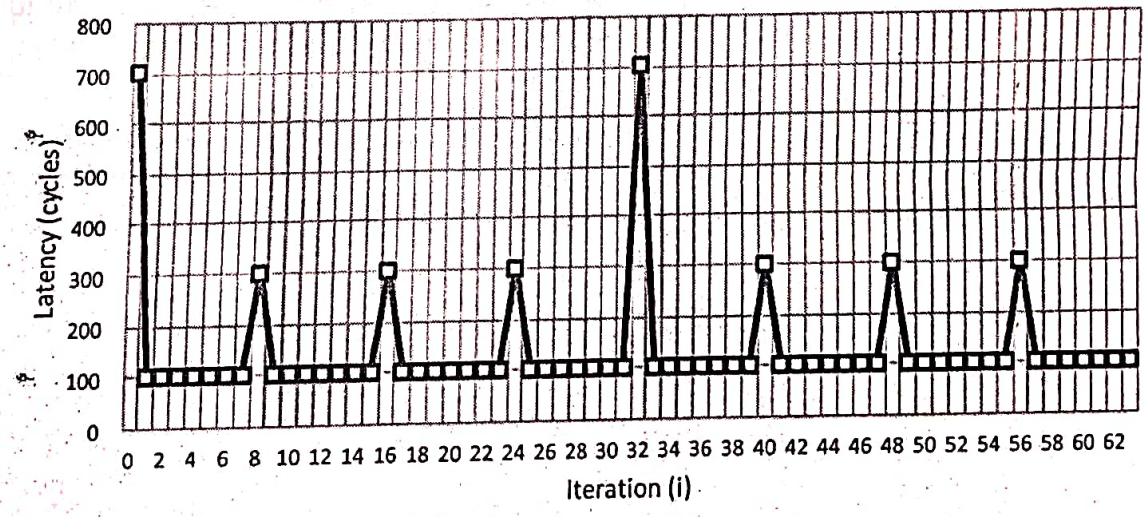
2. a. We are going to micro benchmark the cache hierarchy of a computer with the following two codes. The array data contains 32-bit unsigned integer values. For simplicity, we consider that accesses to the array latency bypass all caches (i.e., latency is not cached). timer() returns a timestamp in cycles.

```
Code Segment:
 j = 0;
     for (i=0; i<size; i+=stride){

          start = timer();
          d = data[i];
          stop = timer();
          latency[j++] = stop - start;
     }
```

(Stride means: Distance between two consecutive memory elements accessed by CPU)

The cache hierarchy has two levels. L1 is a 4kB set associative cache. When we run code segment, we obtain the latency values in the following chart for the first 64 reads to the array data (in the first 64 iterations of the loop) with stride equal to 1.



What are the cache block sizes in L1 and L2? – Explain your answer.

[5]

b. A 16KB direct mapped 256B block unified cache is attached to a 16MB main memory system. The word length as well as instruction length of the processor is 16 bits. Consider a program that consists of a main routine M which in turn calls a subroutine S. M consists of 12 instruction words which are loaded

in the main memory from the address 0x4230FA onwards. The last five instructions of M is a Loop that is iterated 10 times.

The second Instruction in the loop is a call to subroutine S. S consists of 4 instruction words loaded in the main memory from the address 0x70F168. The last instruction of S is a subroutine return back to M. The only two data words that are used by M and S are at addresses 0x748074 and 0x846064. Assume the cache are initially empty. Ignore OS level interruption and subsequent cache impact on context switching.

i)Calculate the address components of the direct mapped unified cache.

ii)Find the number of cache misses occurred during the execution of the program.

iii)How many cache cache block eviction happened during the execution of the program?

iv)List out the block numbers (in decimal) in the cache that are non empty after the execution of the program

[5]

3. a. Determine the total branch penalty for a branch target buffer assuming the Penalty cycles for individual misprediction from the below table.

Instruction in buffer	Prediction	Actual branch	Penalty cycles
Yes	Taken	Taken	0
Yes	Taken	Not Taken	2
No		Taken	2
No		Not Taken	0

Make the Following assumptions about the prediction accuracy and hit rate:

- Prediction accuracy is 90% (for instructions in the buffer).
- Hit rate in the buffer is 90% (for branches predicted taken).

[5]

b. Computers spend most of their time in loops, so multiple loop iterations are great places to speculatively find more work to keep CPU resources busy. Nothing is ever easy, though; the compiler emitted only one copy of that loop's code, so even though multiple iterations are handling distinct data, they will appear to use the same registers. To keep multiple iterations' register usages from colliding, we rename their registers. A compiler could have simply unrolled the loop and used different registers to avoid conflicts, but if we expect our hardware to unroll the loop, it must also do the register renaming. How? Assume your hardware has a pool of temporary registers (call them T registers, and assume that there are 64 of them, T0 through T63) that it can substitute for those registers designated by the compiler. This rename hardware is indexed by the src (source)register designation, and the value in the table is the T register of the last destination that targeted that register.

```
Loop: LD      F4,0(Rx)
I0:   MULTD   F2,F0,F2
I1:   DIVD    F8,F4,F2
I2:   LD      F4,0(Ry)
I3:   ADDD    F6,F0,F4
I4:   SUBD    F8,F8,F6
I5:   SD      F8,0(Ry)
```

Maintain the true dependencies and construct the resulting code

[5]

c. Think about what latency numbers really mean—they indicate the number of cycles a given function requires to produce its output, nothing more. If the overall pipeline stalls for the latency cycles of each functional unit, then you are at least guaranteed that any pair of back-to-back instructions (a "producer" followed by a "consumer") will execute correctly. But not all instruction pairs have a producer/consumer relationship. Sometimes two adjacent instructions have nothing to do with each other. How many cycles would the loop body in the code sequence in given figure below require if the pipeline detected true data dependences and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? Show the code with <stall> inserted where necessary

```
Loop:  LD     F2,0(RX)
I0:    DIVD   F8,F2,F0
I1:    MULTD  F2,F6,F2
I2:    LD     F4,0(Ry)
I3:    ADDD   F4,F0,F4
I4:    ADDD   F10,F8,F2
I5:    ADDI   Rx,Rx,#8
I6:    ADDI   Ry,Ry,#8
I7:    SD     F4,0(Ry)
I8:    SUB    R20,R4,Rx
I9:    BNZ    R20,Loop
```

Latencies beyond single cycle	
Memory LD	+4
Memory SD	+1
Integer ADD, SUB	+0
Branches	+1
ADDD	+1
MULTD	+5
DIVD	+12

to accommodate stated latencies. (Hint: An instruction with latency +2 requires two <stall> cycles to be inserted into the code sequence. Think of it this way: A one-cycle instruction has latency 1 + 0, meaning zero extra wait states. So, latency 1 + 1 implies one stall cycle; latency 1 + N has N extra stall cycles.

[5]

d. Discuss the role Tomasulo's Algorithm in OOO execution with an example and explain the necessity of Commit stage and the role of reorder buffer in hardware based speculation. [Draw diagram and summarize your answer pointwise].

[5]

4. a. Compare the performance of a vector processor with a hybrid system that contains a scalar processor and a GPU-based coprocessor. In the hybrid system, the host processor has superior scalar performance to the GPU, so in this case all scalar code is executed on the host processor while all vector code is executed on the GPU. We will refer to the first system as the vector computer and the second system as the hybrid computer. Assume that your target application contains a vector kernel with an arithmetic intensity of 0.5 FLOPs per DRAM byte accessed; however, the application also has a scalar component which that must be performed before and after the kernel in order to prepare the input vectors and output vectors, respectively. For a sample dataset, the scalar portion of the code requires 400 ms of execution time on both the vector processor and the host processor in the hybrid system. The kernel reads input vectors consisting of 200 MB of data and has output data consisting of 100 MB of data. The vector processor has a peak memory bandwidth of 30 GB/sec and the GPU has a peak memory bandwidth of 150 GB/sec. The hybrid system has an additional overhead that requires all input vectors to be transferred between the host memory and GPU local memory before and after the kernel is invoked. The hybrid system has a direct memory access (DMA) bandwidth of 10 GB/sec and an average latency of 10 ms. Assume that both the vector processor and GPU are performance bound by memory bandwidth. Compute the execution time required by both computers for this application.

[5]

b. Explain the features of vector processor and array processors that are currently used in GPU's.

[5]

5. a. What is the necessity of cache coherence in multiprocessor architecture? Discuss the way it is obtained in symmetric shared memory and distributed memory architecture. Explain any one of them in details.

[5]

b. Suppose you are assigned a job to construct a good design of memory hierarchy for a computer system. So discuss how you will evaluate the performance of your design.

In continuation discuss the optimization techniques that can be employed.

[5]