<u>Solution Paper-I</u>

High Performance Computer Architecture

Question 1:- How to debug a branch predictor?

Answer:-
You can debug by having a very short GHR (3 bits or 4 bits) something like that and print out all the PHT entries (8 or 16 entries) and then check how 2 bit counters and GHR are all updated.

Question 2:- Where do we put the following message?

```
if (KNOB_DEBUG_PRINT.Value()) {
	cout << "ID_STAGE OP " << op->inst_id <<  " is scheduled at cycle " << cycle_count << endl; }
```

When we insert ops into the scheduler or when instructions are removed from the scheduler?

Answer:-
When instructions are removed from the scheduler. Description says that " Add the following code where an op is scheduled. (an instruction will be executed at the following cycle) (An in order processor should print out these messages in order.) "

Question 3:- What is the maximum value of KNOB_GHR_LENGTH?

Answer:-
Since 2^(KNOB_GHR_LENGTH)*2 bit will be the size of g-share branch predictor, any branch predictor which would have more than 1MB is very unpractical due to power and space overhead. Hence, you can assume that the maximum GHR_LENGTH is 32 bits.

Question 4:- What are characteristics of Unpack benchmarks?

Answer:-
The Linpack benchmark is derived from a real application that originated as a collection of linear is subroutines, There is following characteristics of LINPAC benach mark given below

1. This benchmark has a large percentage of floating point operations. Although it do not use division. - .

2. This benchmark does not use any mathematical functions although whetstone benchmark uses mathematical function.

3. It does not have global variables. The operations are being carried out on local variables or an array passed to subroutines as a parameter.

4. This benchmark operates on a two dimensional army. The care must be taken t hiring result comprising that the array size is used.

5. The result for single or double precisiorr operations are both specified in user manual.

6. This benchmark spends a large percentage (over 70%) of execution time within a single function where even a small instruction cache can enhance the performance considerably.

7. This benchmark is based on basic linear algebra subroutines (BLAS) that should be coded in FORTRAN (as in original). Some Vendors present results where the subroutines have been rewriting in assembly language that can make a considerable difference in the performance.

8. Versions of the Lin pack floating-point program are available in FORTRAN, assembly language and C language.

Question 5:- When does the processor insert instructions inside the scheduler and when does it remove instructions?

Answer:-
We remove instructions from the scheduler when an instruction is start to be executed. (i.e., the next cycle all the sources are ready.) As soon as the instruction is removed, we assume that we can insert instructions into the scheduler.

Question 6:- Can we fetch instructions after a branch if a branch is correctly predicted?

Answer:-
This is a very good question. To simplify the homework, we assume that the processor can fetch instructions after a branch if a branch is correctly predicted. Of course, the processor should not fetch more than KNOB_ISSUE_WIDTH number of instructions. If a branch is mispredicted, the processor should not fetch instructions after the mispredicted branch. In a real hardware, the processor can fetch instructions if a branch is not taken. Typically the processor brings a cache block so it can fetch instructions from the same cache block. However, we do not model that behavior in the simulator. If a processor has very aggressive I-cache or trace-cache mechanism, it can fetch instructions across branches.

Question 7:- Do we limit the number of physical register?

Answer:-
We assume that the number of physical register is the double of the number of ROB entries. Hence, there is no case that a processor cannot allocate a physical register.

Question 8:- We use a branch predictor for conditional branches. How about other control flow instructions?

Answer:-
We assume that all other control flow instructions are correctly predicted with other predictor. We just do not model other branch predictor. The simulator simply fetches next correct instructions for other types of control flow instructions.

Question 9:- When does a store instruction actually write a value to the memory system? The MEM stage or the WB stage?

Answer:-
Store instructions can change architectural states. So, a processor must send the store value into the memory system when an instruction is ready to retire. Hence, we model such that a store instruction writes a result in the WB stage. However, the store instruction should check data cache or miss in the MEM stage. Hence, as a simulator's view point, both load and store instructions are equally handled in the MEM stage. We do not add additional timing delay for store instruction in the commit stage.

Question 10:- When should the processor update the register file?

Answer:-
It should update the register file at the commit stage. Note that there are data forwarding logic in the pipeline. We do not check the register valid bits to grade.

Question 11:- Which improvement gives a greater reduction in execution time : One that is used 20 percent of the time but improves performance by a factor of when used, or one what is used 70 percent of time but only improves performance by a factor of 1.3 when used.

Answer:-
Applying Amdahl's Law, we get the following equation for the first improvement:

$$\text{Execution Time}_{(\text{new})} = \text{Execution time}_{(\text{old})} \times \left[\text{Frac}_{(\text{unused})} + \frac{\text{Frac}_{(\text{used})}}{\text{Speed up}_{(\text{used})}} \right]$$

$$= \text{Execution time}_{(\text{old})} \times \left[0.8 + \frac{0.2}{2} \right]$$

$$\text{Execution time}_{(\text{new})} = \text{Execution time}_{(\text{old})} \times 0.9$$

So the execution time with the first improvement is 90 percent of Execution time without the improvement. Plugging the values for the second improvement into Amdahl's law:

$$\text{Execution Time}_{(\text{New})} = \text{Execution time}_{(\text{old})} \times \left[0.3 + \frac{0.7}{1.3} \right]$$

$$\text{Execution Time}_{(\text{new})} = \text{Execution time}_{(\text{old})} \times 0.84$$

This shows that the execution time with the second improvement is 84 percent of the execution time without the improvement. Thus, the second improvement will have a greater impact on overall execution time despite the fact that it given less of an improvement when it is in use.

Question 12:- Briefly explain instruction format.

Answer:-
An instruction contain number of bits in the so that it is being to perform specific operation. Generally an instruction is divided into three fields
Addressing mode: It specifies that how the operands are accessed in an instruction.

Operation code (O): This field specifies the operation that is performed in the operand.

Operand: It specifies the data on which operation is performed.

Question 13:- Differentiate between RISC and CISC.

Answer:-
Difference between RISC and CISC are given below:
1. It means Reduced Instruction set computing.

2. It uses hardwired control unit.

3. RISC requires fewer and limited instructions

4. Example of RISC processors are BM2PO,. SPARC from SO p-ticrosoft ycm, power PC and PA- RISC. —.

CISC:

It means complex instruction set computing.

It uses micro programmed control unit.

ClSC requires wide range of instructions.

These instructions produce more efficient results. -

The example of CISC processor is IBM Z and gital equipment corporation VAX computer.

Question 14:- How pipelining would improve the performance of CPU justify.

Answer:-
Non-pipeline unit that performs the same operation and takes a time equal to (time taken to complete each task). The total time required for n tasks is n t. The speed up of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio

$$S = \frac{nt_n}{(k+n-1)t_p}$$

As the number of tasks increases, n becomes much larger than k — 1, and K + n — I approaches the value of n, where K is segments of pipeline and Ip is time used to execute n tasks. Under this condition, the speed up becomes.

$$S = \frac{t_n}{t_p}$$

The time it takes to process a task is the same in the pipeline and non pipeline circuits. There if t = kt speed reduces to

$$S = \frac{k\cancel{t_p}}{\cancel{t_p}} = K.$$

Maximum speed that a pipeline can provide is K, where K is number of segments in pipeline. Speed of pipeline process is improved the performance of C.P.U. To clarify the meaning of improving the performance of C.P.U. through speed up ratio, consider the following numerical example. Let the time it takes to process sub operation in each segment be equal to 20 ns. Assume that the pipeline k 4 segments and executes n 100 tasks in square. The pipeline system will take (k + n — 1) t = (4 + 99) 20 = 2060 ns to complete. Assuming that t = kt = 4 x 20 = 80ns, a non-pipeline system requires nk tp = 100 x 80 8080 ns to complete tice 100 tasks. The speed up ratio is equal to 8000/2060 88. As the number of tasks increase, the speed up will approach 4, which is equal to the number of segment in pipeline. It we assume that = 60ns, then speed up become=60/3.

Question 15:- Differentiate between computer architecture and computer organization.

Answer:-
Difference between computer architecture and computer organization:

Computer Architecture	Computer Organization
1. It includes emphasis on logical design, computer design and the system design.	It includes emphasis on the system components, circuit design, logical design, structure of instructions, computer arithmetic, processor control, assembly language programming and methods of performance enhancement.
2. It is concerned with the structure and behaviour of computer as seen by user.	Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system.