INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
Solution for Long test 1 - 2022

Date: 23-02-22 **Timing: 8 am -9:30 am ; server closes at 9:40 am; Total Marks = __**
Subject No : CS60003 **HIGH PERFORMANCE COMPUTER ARCHITECTURE**
Department/Centre/School : Computer Science and Engineering
Answer all questions. In case of reasonable doubt, make assumptions and state them upfront. Marks will be deducted for claims without proper reasoning. Write your answers (with all analysis, justification, calculation etc) in loose sheets, scan them and upload in moodle within the allotted time

1. A processor has 32KB instruction cache and 32 KB data cache. Also, assume the following quantities
   - 36% of instructions are of type load/store
   - 74% of memory references are instruction references
   - Misses per 1000 instructions:
     1. 16 KB instruction cache: 3.82
     2. 16 KB data cache: 40.9
   - Hit time: 1 clock cycle
   - Miss penalty: 100 clock cycles

   a. Compute the overall miss rate of this split cache design. b. Compute the Average memory access time (AMAT). **[4+4]**

   Solution:

```
Miss rate of i-cache =     3.82/1000 ~ 0.004
Miss rate of d-cache = 40.9/260 ~ 0.1573
Overall miss rate = 0.74 * 0.004 + 0.26 * 0.1573 = 0.043858
AMAT = %instr*(hit-time + i-cache-miss-rate*miss-penalty) +
%mem*(hit-time + d-cache-miss-rate*miss-penalty)
= 0.74*(1+ 0.004*100)+0.26*(1+0.1573*100)=0.74*1.4 +0.26*16.73=5.3858
```

```
Miss rate of i-cache =     3.82/1000 ~ 0.004
Miss rate of d-cache = 40.9/360 ~ 0.114
Overall miss rate = 0.74 * 0.004 + 0.36 * 0.114 = 0.044
AMAT = %instr*(hit-time + i-cache-miss-rate*miss-penalty) +
%mem*(hit-time + d-cache-miss-rate*miss-penalty)
= 0.74*(1+ 0.004*100)+0.36*(1+0.114*100)=0.74*1.4 +0.36*12.4=5.5
```

2. Observe the instruction sequence given below and list all dependencies of type RAW, WAW and WAR. Instruction format is (operation, destination, source1, source2)
   I0: add r1 r2 r3
   I1: sub r3 r1 r2
   I2: add r4 r1 r3
   I3: mul r1 r2 r3 **[7]**
   Solution :
   RAW:
   I0->I1, I1->I2,I1->I3,

WAW:
I0− >I3
WAR:
I0->I1, I1->I3, I2->I3,

3. Explain clearly what is the advantage of the following code transformation in terms of reduction in cache misses. Assume x is an integer array, block size in cache = 10×sizeof(int), cache type = direct mapped, cache miss latency = 10 ms, hit latency = 1 ms, no of blocks in cache = 10.

```
/* Before */
for (j = 0; j < 100; j = j+1)
       for (i = 0; i < 5000; i = i+1)
              x[i][j] = 2 * x[i][j];


/* After */
for (i = 0; i < 5000; i = i+1)
      for (j = 0; j < 100; j = j+1)
             x[i][j] = 2 * x[i][j];
```

[7]

Solution:

Before: x[0][0] x[1,0] ... x[4999][0] each access will lead to cache miss.

Total penalty = 100 * 5000 * miss-penalty = 5000000 ms

After:

Block size = 10 * size of(int). Hence, x[0][0] x[0,1] ... x[0][9] are brought in each cache miss.

Hence, miss-rate = 0.1
hit-rate = 1 - 0.1 = 0.9

Total penalty = (5000 * hit-rate * 100 * hit-latency) + (5000 * miss-rate * 100 * miss-penalty)
= 5000 * .9 * 100 * 1 + 5000 * .1 * 100 * 10 = 500000 + 450000 = 950000

speed up = 5000000/950000 = 500/95 = 100/19 = 5.26

4. Consider a 1-bit and a 2-bit predictor. In the table below, indicate the prediction for the branch and whether the prediction is correct . For the 1-bit predictor use the notation "T" for taken, and "N" for not taken states. For the 2-bit predictor, use "ST" for strongly-taken state, "WT" for weakly-taken state, "SN" for strongly-not-taken state, and "WN" for weakly-not-taken state. Assuming the first entry in the table, fill in the predictor's state after branch resolution for each of the remaining branch outcomes.

[5]

Solution:

	1 bit predictor			2 bit predictor		
Outcome	**Prediction**	**Next State**	**Correct?**	**Prediction**	**Next State**	**Correct?**
Taken	N	T	No	N	ST/WT/WN	No
Taken						
Not taken						
Not Taken						
Not Taken						
Taken						
Not Taken						
Taken						
Taken						
Not Taken						

	1 bit predictor			2 bit predictor		
Outcome	**Prediction**	**Next State**	**Correct?**	**Prediction**	**Next State**	**Correct?**
Taken	N	T	No	N	WT	No
Taken	T	T	Yes	T	ST	Yes
Not taken	T	N	No	T	WT	No
Not Taken	N	N	Yes	T	WN	No
Not Taken	N	N	Yes	N	SN	Yes
Taken	N	T	No	N	WN	No
Not Taken	T	N	No	N	SN	Yes
Taken	N	T	No	N	WN	No
Taken	T	T	Yes	N	WT	No
Not Taken	T	N	No	T	WN	No

Note: In the case of 2-bit predictor, if the first entry to the table is either WT/ST, the second entry to the table remains the same for both the cases. This is because that a branch with ST as its next state on receiving another 'Taken' outcome shall continue to remain in the ST state.

	1 bit predictor			2 bit predictor		
Outcome	**Prediction**	**Next State**	**Correct?**	**Prediction**	**Next State**	**Correct?**
Taken	N	T	No	N	WN	No
Taken				N	WT	No
Not taken				T	WN	No
Not Taken				N	SN	Yes
Not Taken				N	SN	Yes
Taken				N	WN	No
Not Taken				N	SN	Yes
Taken				N	WN	No
Taken				N	WT	No
Not Taken				T	WN	No

5. Assume an application where the execution of floating-point instructions on a certain processor P consumes 60% of the total runtime. Moreover, let's assume that 25% of the floating-point time is spent in square root calculations.

   a. Based on the initial research, the design team of the next-generation processor $P2$ believes that it could either improve the performance of all floating point instructions by a factor of 1.5 or alternatively speed up the square root operation by a factor of 8. From which design alternative would the aforementioned application benefit the most?

   b. Instead of waiting for the next processor generation, the developers of the application decide to parallelize the code. What speedup can be achieved on a 16-CPU system, if 90% of the code is perfectly parallelized? What fraction of the code has to be parallelized to get a speedup of 10?

[2+3]

Solution:

a. **Amdahl's Law**: $S_p(n) := \frac{1}{\beta+(1-\beta)/p}$

- Improvement 1 (all fp instructions sped up by a factor 1.5). Sequential part (β): 0.4, $p = 1.5$.

The application would observe a total speedup of:

$$S_p(n) = \frac{1}{\beta + (1-\beta)/p} = \frac{1}{0.4 + (1-0.4)/1.5} = 1.25$$

- Improvement 2 (square root instructions sped up by factor of 8). Sequential part (β): 0.4 + 0.45, $p = 8$. The application would observe a total speedup of:

$$S_p(n) = \frac{1}{\beta + (1-\beta)/p} = \frac{1}{0.85 + (1-0.85)/8} = 1.15$$

Thus, the application would benefit the most from the first improvement.

b. **Parallelization of code**: The speedup achieved on a 16-CPU system is:

$$S_p(n) = \frac{1}{\beta + (1-\beta)/p} = \frac{1}{0.1 + (1-0.1)/16} = 6.4$$

To attain a speedup of 10, a 96% of the code would need to be perfectly parallelizable. This value is obtained by solving the equation:

$$10 = \frac{1}{\beta + (1-\beta)/16}$$

6. Assume a 3-bit global register (initialized to all 0's) and a two-level Global–Global (global register and global PHT) scheme.

   a. How many entries are there in the PHT?

   b. Show the contents of the global register(GR) and of the PHT (where each entry is initialized to the weak not-taken case), after the following sequence of branch executions has taken place: 1 taken (T) branch, 3 not-taken (NT) branches, 1 T branch, 3 NT branches, 1 T branch. With *pred* being the prediction and *actual* the actual outcome as per the given string, show your results for each branch in the table below in the form (GR) (*pred*) (*Updated* GR) (*actual*) (*correct?*) (PHT). Assume a speculative update for GR and a nonspeculative one for the PHT that occurs before the next branch prediction. On misprediction, restore GR to (001). **[1 + 4]**

GR	***pred***	***Updated* GR**	***actual***	**correct?**	**PHT**
000	N	001	T	No	01, 01, 01, 01, 01, 01, 01, 01

Solution:

a. The global register stores 3-bit history. Therefore, the Pattern History Table (PHT) will have $2^3 = 8$ entries.

b. The table is filled up as discussed in the class. [**Note:** The updates in the PHT has been done in the next row, in a non-speculative manner. You can update the PHT in the same row as well.]

GR	***pred***	***Updated* GR**	***actual***	**correct?**	**PHT**
000	N	000	T	No	00, 00, 00, 00, 00, 00, 00, 00
001	N	010	N	Yes	01, 00, 00, 00, 00, 00, 00, 00
010	N	100	N	Yes	01, 00, 00, 00, 00, 00, 00, 00
100	N	000	N	Yes	01, 00, 00, 00, 00, 00, 00, 00
000	N	000	T	No	01, 00, 00, 00, 00, 00, 00, 00
001	N	010	N	Yes	10, 00, 00, 00, 00, 00, 00, 00
010	N	100	N	Yes	10, 00, 00, 00, 00, 00, 00, 00
100	N	000	N	Yes	10, 00, 00, 00, 00, 00, 00, 00
000	T	001	T	Yes	10, 00, 00, 00, 00, 00, 00, 00
001	-	-	-	-	11, 00, 00, 00, 00, 00, 00, 00