CS2323 Computer Architecture 2019

Homework 2

- Your submission should be named as RollNumber_CA_HW2.pdf. For example, if your roll number is cs16mtech11075, then your submission should be cs16mtech11075_CA_HW2.pdf. Except pdf, no other format is acceptable. **10 marks will be deducted for not following these instructions or if you submit a zipped file.**
- If you submit hand-written solution after scanning, make sure all the text is legible.
- The reasoning for obtaining the answer should be clearly shown to obtain full marks. At the same time, be concise.
- There are 3 bonus marks for writing your solution in Latex. You need to include the following line at the beginning of your solution (i.e., in your *.tex file) to get the bonus:

This document is generated by \LaTeX

- Late Submission Penalty: 20% for each late day (including weekend)

==========

Q1. For an FP number with total T bits and E exponent bits and 1 sign bit,

(a) Find the generic expression for the smallest and largest normal value, and smallest and largest denormal value. You need to find these only for positive values. (6marks)

Bias = $2^{(E-1)}-1$

Smallest normal value: $2^{2-(2\text{^}(E-1))}$

Largest normal value: $2^{(2\text{^}(E-1))}$

Smallest denormal value: $2^{(E+3-T-2\text{^}(E-1))}$

Largest denormal value: $2^{2-2\text{^}(E-1)}-2^{3-T+E-2\text{^}(E-1)}$

(b) Now find the specific values of these for FP16 and bfloat16 (see slide 3 of L07). (4 marks)

FP 16		
	Normal	Denormal
Largest	$(2-2^{-10})*2^{15}$	$2^{-14}-2^{-24}$
Smallest	2^{-14}	2^{-24}

bfloat16		
	Normal	Denormal
Largest	$(2-2^{-7})*2^{127}$	$2^{-126}-2^{-133}$
Smallest	2^{-126}	2^{-133}

(c) Now consider only normal numbers. Find the difference between the smallest and second smallest number of FP16 and bfloat16. (3 marks)

Considering only positive numbers:

For FP16 : 2^{-24}

For bfloat16: 2^{-133}

(d) From (b) and (c), what can you say about relative pros and cons of FP16 and bfloat16. (2 marks).

bfloat16 has larger range than FP16, and more representations for normal values.

FP16 has better precision as the scale of values increases and more representations for denormal values.

(e) How does range of bfloat16 compare with that of FP32? (1 mark)

Range of bfloat16 is almost same as that of FP32 (which is really good although bfloat16 has less number of total bits!)

If someone wrote this: "Range of FP32 is slightly larger than the range of bfloat16", this is also correct.

We need not worry about range of denormal numbers in them.

Q2. (2 marks) A processor uses 48 bit virtual address. It has 2GB physical memory and the memory addresses are defined at the level of each byte. Page size is 2KB. A processor has only one level of TLB which has 64 entries. Find out the size of TLB (excluding valid bits, etc).

Page number = 48-11 = 37 bits

Frame number = 31-11 = 20 bits

Total bits in each entry = 57 bits. Total size of TLB = 57*64 bits

Q3. (1 mark) Find out the reach of this DTLB:

PageSize	Entries	associativity
4KB	128	4-way
2MB	32	8-way
2GB	8	fully-associative

Answer = 4KB * 128 + 2MB*32+2GB*8

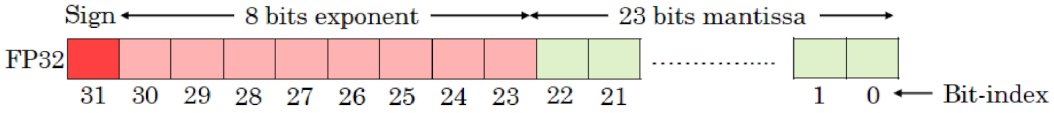
Associativity has no use here, since number of entries is already given.

Q4. (2 marks) Consider a TLB which has 4 ports. The processor has a frame size of 1KB. In a given cycle, the addresses coming to the four ports of TLB are: 0x4795BA21, 0x4795BB21, 0x5795BA21 and 0x4785BA21. Find out how many unique accesses can be sent to TLB if it uses intra-cycle compaction technique to save energy.

Address	Page number	Page offset
0x4795BA21	10001111001010110 1110	1000100001
0x4795BB21	10001111001010110 1110	1100100001
0x5795BA21	10101111001010110 1110	1000100001
0x4785BA21	10001111000010110 1110	1000100001

So, only three unique accesses need to be sent.

Q5. Consider FP32. We give index to its bits as shown in figure below.



Now consider 9 MSBs of FP32 representation of various powers of 2 shown below

2^k for $k \in [11, -15]$ and 31st-23rd bits of corresponding FP32 representation

2^k	31st-23rd bits	2^k	31st-23rd bits	2^k	31st-23rd bits	2^k	31st-23rd bits	2^k	31st-23rd bits
2^{11}	0 10001010	2^5	0 10000100	2^{-1}	0 01111110	2^{-7}	0 01111000	2^{-13}	0 01110010
2^{10}	0 10001001	2^4	0 10000011	2^{-2}	0 01111101	2^{-8}	0 01110111	2^{-14}	0 01110001
2^9	0 10001000	2^3	0 10000010	2^{-3}	0 01111100	2^{-9}	0 01110110	2^{-15}	0 01110000
2^8	0 10000111	2^2	0 10000001	2^{-4}	0 01111011	2^{-10}	0 01110101		
2^7	0 10000110	2^1	0 10000000	2^{-5}	0 01111010	2^{-11}	0 01110100		
2^6	0 10000101	2^0	0 01111111	2^{-6}	0 01111001	2^{-12}	0 01110011		

(a) [1 mark] Consider three deep neural networks (DNNs): AlexNet, GoogleNet and ResNet50 (don't worry what they are). It is given that nearly all the weights of these DNNs are in range [2^−13:2^−2]. If the weights are stored in FP32 format, is there something common about bits [30,29,28] of all the weights? If yes, write that.

All these numbers have their [30,29,28] bits as 011.

(b) [1 mark] If the activations of some deep neural networks are in the range [2^1:2^11], what can you say about the value of bits [30,29,28]?

All these numbers have their [30,29,28] bits as 100.

Q6. [2 mark] Consider the following code:

```
 for (i = 0; i < N; i = i+1)
 for (j = 0; j < N; j = j+1)
   a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
 for (j = 0; j < N; j = j+1)
   d[i][j] = a[i][j] + c[i][j];
```

Assume that the array a, b, c, d are integer arrays and the L1 cache block size is 4B. Is it possible to rewrite the above code to reduce the number of L1 cache misses? If so, write so. You are not allowed to change the data-layout in memory. You can only change the code.

Perform loop-fusion

```
for (i = 0; i < N; i = i+1)
for (j = 0; j < N; j = j+1){
a[i][j] = 1/b[i][j] * c[i][j];
d[i][j] = a[i][j] + c[i][j];
}
```

Q7. [5 marks] Consider invalidating snooping protocol with 3 CPUs: C1, C2 and C3. Consider a memory location `L', where the value 5 is stored.

In the following table, fill the cells with activity or values stored. If a particular cache or the memory does not cache/store the location L, leave it blank. We will ignore the `activity on bus' column; so you can write something here or leave it blank as you wish.

		Value stored at location L in			
CPU Activity	activity on bus	C1's cache	C2's cache	C3's cache	Memory
C1 reads L					
C2 reads L					
C2 writes 2 to L					
C3 reads L					
C3 writes 14 to L					
C2 reads L					
C1 writes 24 to L					

Here is the solution:

CPU Activity	Activity on bus	C1's cache	C2's cache	C3's cache	Memory
					5
C1 reads L	Cache miss for L	5			5
C2 reads L	Cache miss for L	5	5		5
C2 writes 2 to L	Invalidate for L		2		2
C3 reads L	Cache miss for L		2	2	2
C3 writes 14 to L	Invalidate for L			14	14
C2 reads L	Cache miss for L		14	14	14
C1 writes 24 to L	Invalidate for L	24			24

Q8. [2 marks] Consider the following code where a, b, c and d are integer arrays, such that their starting address in virtual memory are 5678, 9100, 5900 and 9969, respectively. Each array has 24 elements. System page size is 1000B (for sake of simplicity). There are no registers in the processor.

```
int main(){
 for(int i=0;i<24; i++)
 cout << a[i] + b[i]+ c[i]+d[i];}
```

Is it possible to rewrite the code to improve TLB efficiency. If so, write it, otherwise, write ``NO".

Arrays a[ ] and c[ ] are on the same page. Arrays b[ ] and d[ ] are also on the same page. TLB accesses can be minimized with translation-reuse by modifying the code to:

```
int main(){
 for(int i=0;i<24; i++)
 cout << a[i] + c[i]+ b[i]+d[i];}
```

Q9. [2 marks] For the program below, which variables (if any) show temporal and spatial locality?

```
int main() {
long int N=100000; int array[N];
for(int i=0;i<N; i++)
  array[i]=0;
}
```

i and N shows temporal locality while array shows spatial locality.

Q10. [1 mark] Person1 runs a program two times on his laptop. He sees that first time, the program ran slower than the second time. Can you think of any reason. Exclude random factors such as noise. (hint: cache).

When the program runs for the second time, lot of memory accesses, which would have been compulsory misses during the first run, may become cache hit. This decrease in compulsory cache misses is the reason why it ran faster the second time.