CE/CZ3001: Advanced Computer Architecture:
Tutorial 5

1. Answer:

a) Like the table shows:

	Always Taken Predictor	Always Not Taken Predictor
i	75%	25%
ii	60%	40%

b) (i) 0% (ii) 20% (only the third T)

c) (i) 75% (ii) 40%

2. Answer:

When X=5, the total number of instructions within the loop=6*5=30. Total number of instructions=31. Total number of stalls= I2 and I3 having RAW dependency. (2 Stalls) I3 and I4 has RAW dependency. (2 stalls) I5 and I7 having RAW dependency. (1 stall)
5 data stalls and 1 control stall
6stalls*5iterations=30, CPI=(31+30)/31=1.9677
When X=100, CPI=(601+600)/601=1.998

After unrolling by 4 and instruction reordering we get (note that each datamemory value is 64 bits and hence the next address is separated by a value of 8)

```
        ADDI X10, X31, 25
loop:   LDUR X1, [X11, #0]
        LDUR X2, [X11, #8]
        LDUR X3, [X11, #16]
        LDUR X4, [X11, #24]
        ADD X1, X1, X12
        ADD X2, X2, X12
        ADD X3, X3, X12
        ADD X4, X4, X12
        SUBI X10, X10, #1
        STUR X1, [X11, #0]
        STUR X2, [X11, #8]
        STUR X3, [X11, #16]
        STUR X4, [X11, #24]
        ADDI X11, X11, #32
        CBNZ X10, loop
```

One stall for control dependency

finish

Total number of instructions= 15*25+1=376

CPI=(376 + 1*25) /376=1.0665

Here we can notice that combined effect of loop unrolling and instruction reordering helped us to improve the performance of the system.

Tutorial-6

1. Answer:

Processor = 32B = 2^5B, Address bits = 5 bits, Cache size = 8B, Block size =1B
Therefore, number of offset bits = log2(1B) = 0 bits
Number of sets = cache size/block size=8, Index =3 bits, Tag=5-3-0=2 bits

Dividing the address as tag, index and offset we can get the following as the hit and miss rate.

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M	M	H	H	M	M	M	M	M	M

2. Answer:

Address =8 bits (of the main memory)

Cache size=16B, Block size=4
Offset = 2
Number of blocks= 16/4=4
Number of sets=2 and Index = 1,Tag= 5

Time	0	1	2	3	4	5	6	7
Access	10001101	10110010	10111111	10001100	10011100	11101001	11111110	11101001
TAG	10001	10110	10111	10001	10011	11101	11111	11101
OFFSET	01	10	11	00	00	01	10	01
INDEX	1	0	1	1	1	0	1	0

In the following each access is represented by a triple: (Tag, LRU Bit, Hit Bit)
LRU bit = 1 if the current block is the least recently used one.
Hit Bit = 1 if the current reference is a hit

Access 0		
Set 0		
Set 1	10001, 0, 0	

Access 1		
Set 0	10110, 0, 0	
Set 1	10001, 0, 0	

Access 2		
Set 0	10110, 0, 0	
Set 1	10001, 1, 0	10111,0, 0

Access 3		
Set 0	10110, 0, 0	
Set 1	10001, 0, 1	10111,1, 0

Hit

Access 4		
Set 0	10110, 0, 0	
Set 1	10001, 1, 0	10011,0, 0

Access 5		
Set 0	10110, 1, 0	11101, 0, 0
Set 1	10001, 1, 0	10011, 0, 0

Access 6		
Set 0	10110, 1, 0	11101, 0, 0
Set 1	11111, 0, 0	10011, 1, 0

Access 7		
Set 0	10110, 1, 0	11101, 0, 1
Set 1	11111, 0, 0	10011, 1, 0

Hit

Hit rate 2/8=0.25

Tutorial-7

1. Answer:

a) Memory-stall cycles = memory accesses/program × miss rate × miss penalty
Total CPI = Original CPI + Mem. Stall cycles for L1 cache miss + Mem. Stall cycles for L2 cache miss = 1.25+ 20% (mem access) *20% (miss rate for L1 cache) *8 (cycle penalty) +20%*20%*10% *30
CPI final =1.25 + (0.32) + (0.12) =1.69

b) To access the main memory 30 cycles are required.
CPI= 1.25 (original CPI) + 20% *20% (miss rate of L1) * 30 cycles (for accessing main memory) = 1.25 + 0.2 * 0.2* 30= 2.45
Speedup = 2.45/1.69 = 1.45X

2. Answer:

For the unconditional branch, the execution will be as follows: there is one stall induced by the unconditional branch.

Instruction	1	2	3	4	5	6	7
I	IF	ID	EX	WB			
I+1		IF	IF	ID	EX	WB	
I+2			Stall	IF	ID	EX	…

For the conditional branch, if the branch is taken, there incurred two stalls.

Instruction	1	2	3	4	5	6	7	8
I	IF	ID	EX	WB				
I+1		IF	ID	IF	ID	EX	WB	
I+2			IF (stall)	stall	IF	ID	EX	WB

Otherwise, if the branch is not taken, there will be no stall.

Instruction	1	2	3	4	5	6	7
I	IF	ID	EX	WB			
I+1		IF	ID	EX	WB		
I+2			IF	ID	EX	WB	…

Therefore, CPI = 1 + 1%*1 + 15%*60%*2 + 0 = 1.19.

Tutorial-8

1. Answer:

```
__global__ void stencil_1d(int *in, int *out) {
      __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
      int gindex = threadIdx.x + blockIdx.x * blockDim.x;
      int lindex = threadIdx.x + RADIUS;

      // Read input elements into shared memory
      temp[lindex] = in[gindex];
      if (threadIdx.x < RADIUS) {
            temp[lindex - RADIUS] = in[gindex - RADIUS];
            temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
      }

      // Synchronize (ensure all the data is available)
      __syncthreads();

      // Apply the stencil
      int result = 0;
      for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
            result += temp[lindex + offset];
      // Store the result
      out[gindex] = result;
}
```