5. Suppose a computer with following features:

- ✧ Memory accesses are to **2-byte words**.
- ✧ It's a **little endian** type machine.
- ✧ Physical addresses are **12-bits** wide.
- ✧ The cache uses **LRU replacement policy**.

The contents of the cache are as follows. (Hexadecimal):

2-way Set Associative									
	Line 0					Line1				
Set Index	Tag	Valid	B0	B1	B2	B3	Tag	Valid	B0	B1	B2	B3
0	C7	1	86	30	3F	10	05	1	40	67	C2	3B
1	45	1	06	78	07	C5	38	0	00	BC	0B	37
2	91	1	60	4F	E0	23	F0	0	C0	88	30	12
3	06	0	B7	26	2D	47	32	1	12	08	7B	AD

Given the following fields of physical address:

- ✧ CO: Byte offset within the cache line
- ✧ CI: The cache (set) index
- ✧ CT: The cache tag

1. Indicate the length of every field of above cache. (1' * 3 = 3')

	Fields	Length (bit)
Cache	CT	[8]
	CI	[2]
	CO	[2]

2. List all of the hex physical address range that will hit in **Set 1**.(4')

0100 0101 0100 ~ 0100 0101 0111 (0x454 ~ 0x457)

3. Suppose a program running on this machine and access the memory (load) as the following trace, fill the blanks in the table. If there is a cache miss, enter "-" in the "Value Returned" row, otherwise fill in the true value **in hex** the cache returns. (8')

Binary Address	Hit or Miss?	Value Returned
1100 0111 0000	[H]	[0x3086]
1100 1000 0010	[M]	--
0011 1000 0110	[M]	--
1001 0001 1000	[H]	[0x4F60]

4. Consider the program execution on above machine. You are given the following definitions:

```
struct node{
    short id;
    short v;
};
struct node tree[4][4];
int i, j;
```

- *sizeof*(short) is equal to **2**.
- The array **tree** is allocated at physical address **0**.
- The cache above is now cleaned up (empty now).
- The only memory accesses are to the entries of the array **tree**.

  (**i** and **j** are stored in registers).

Please determine the cache performance of the following codes

```
for(i=0; i<4; i++){
     for(j=0; j<4; j++){
         tree[i][j].id = i * j;
         tree[i][j].v = i + j;
     }
}
```

1) What is the total number of writes that ***miss*** in the cache? **16** (1')

2) What is the ***miss rate***? **1/2**. (2')

If we change the code (the cache is still empty)

```
for(i=0; i<4; i++){
    for(j=0; j<4; j++){
        tree[i][j].id = i * j;
    }
}
for(i=0;i<4;i++){
    for(j=0;j<4;j++){
        tree[i][j].v = i + j;
    }
}
```

3) What is the total number of writes that ***miss*** in the cache? **32**.(1')

4) What is the ***miss rate***? **1**. (2')

5) After finish running above code, fill all the content of **Set 1** in hex(8')

2-way Set Associative				
	Line 0		Line1	
Set Index	Tag Valid	B0 B1 B2 B3	Tag Valid	B0 B1 B2 B3
1	[02] [1]	[02 00 03 00]	[03] [1]	[03 00 04 00]