2a

i)

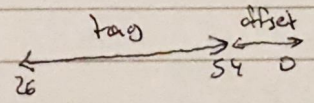
Assembly	Type	Expression	Value
movl 8(%rbx), %eax	int	N[2]	M[X_N + 8]
lea 8(%rbx,%rdx,4), %rax	int*	N + i + 2	X_N + 4·i + 8
movl 12(%rbx,%rdx,8), %eax	int	N[2*i + 3]	M[X_N + 8·i + 12]
lea 20(%rbx,%rdx,4), %rax	int*	N + i + 5	X_N + 4·i + 20

ii)

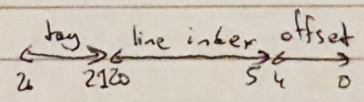
```
short compute (short x, short y) {
    if (x <= 0)
        return 0;
    else if (y <= 0)
        return x;

short compute (short x, short y) {
    if (x <= 0)
        return 0;
    if (y <= 0)
        return x;
    if (x <= y)
        increment(&x, y); // y is zero extended for some reason,
    else                  // so I assume it is passed as an int
        x = x <<y;        // argument

    return x;
}
```

b.

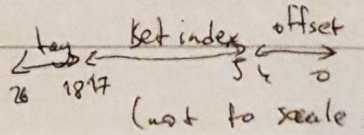
a. Assuming "fully associative".
bits 0 to 4: offset within block
bits 5 to 26: tag bits



b. Direct cache
bits 0 to 4: offset within block
bits 5 to 20: line index
bits 21 to 26: tag bits



c. 8-way set-associative
bits 0 to 4: offset within block
bits 5 to 17: set index
bits 18 to 26: tag bits



(not to scale!!!)

2b cont.

Access 1:
addr = 0, block = 0, not yet in cache, miss
Access 2:
addr = 32, block = 1, not yet in cache, miss
Access 3:
addr = 8, block = 0, in cache, hit
Access 4:
addr = 32, block = 1, in cache, hit
Access 5:
addr = 64, block = 2, ~~in cache~~ not yet in cache, miss

Overall three misses and two hits,
miss rate = 3/(2+3) = 3/5 = 0.6

Note that
1) cache is big enough to fit everything that was loaded
2) associativity of the cache does not affect the outcome (directly mapped lines are always available)