Solution

Problem 1: (18 points)

[1]	111101	[2]	110101	[3]	011010
[4]	111	[5]	010000	[6]	1
[7]	0	[8]	100000	[9]	1

Problem 2: (10 points)

[1]	0x00000104	[2]	0x12345677
[3]	0x00000108	[4]	0x00220020
[5]	%ecx	[6]	0x0000010C
[7]	0x00000108	[8]	0x10011105
[9]	%eax/%al	[10]	0x000001D4/0xD4

Problem 3: (12 points)

[1]	20	[2]	0x804a062
[3]	0x804a064	[4]	0x804a068
[5]	0x804a06c	[6]	0x804a06c

Problem 4: (20 points)

1	[1]	0xbfda8204	[2]	0xbfda8208	[3]	0xbfda820c
2	[1]	$0x4	[2]	0x8048400	[3]	%dx
	[4]	$0x2	[5]	$0x8	[6]	$0x4
3	[1]	03070306				

Problem 5: (20 points)

1 [1] ch >= '0' && ch <= '9' [2] '9' - digit
[3] 'B' [4] 'C' [5] L4 [6] L8
[7] L8 [8] L3 [9] $65 [10] $5
[11] L4 [12] *L6(,%eax,4)

2 CD8AF2

3 Like the 'Jump Table' scheme, we can construct an array to map the given char to the encrypted one. Sample code is as follow:

```
static int map[] = {
   '9', '8', '7', '6', '5', '4', '3', '2', '1', '0',
   'A', 'A', 'A', 'A', 'A', 'A', 'A', 'F', 'C', 'B', 'A',
   'D', 'E'
};

int encrypt(int ch) {
   if (ch >= '0' && ch <= 'F')
      return map[ch – '0'];
   else
      return 'A';
}
```

Problem 6: (20 points)

1 [1] $0x4 [2] 80483bc
[3] %eax [4] %ebx
[5] 0x8(%ebp) [6] %eax
[7] (%eax,%ebx,1) or (%ebx, %eax, 1)
or (%eax,%ebx) or (%ebx,%eax)
[8] $0x14 [9] %ebx

2 [1] 0xbfdaff20 [2] 0xbfdaff30
[3] 0x4 [4] 0xbfdafefc
[5] 0xbfdaff18 [6] 0x3
[7] 0x2 [8] 0xbfdafef8

3 fib1(4): 5