Solution

Problem 1: (16 points)

[1]	1001	[2]	0000 1000	[3]	0000 1000
[4]	0000	[5]	0000 0001	[6]	1000 0000
[7]	0000 0001	[8]	0		

Problem 2: (10 points)

[1]	%eax	[2]	0x0000 c3d4
[3]	%eax	[4]	0x0000 0204
[5]	%edx	[6]	0x0000 0100
[7]	%edx & %eax	[8]	0x0000 0000 & 0x0000 0204
[9]	0x10c	[10]	0x0000 0006

Problem 3: (15 points)

1	[1]	48	[2]	80
	[3]	24	[4]	40
	[5]	0x8049781	[6]	0x6009e1
	[7]	0x8049788	[8]	0x6009e8
	[9]	0x80497ac	[10]	0x600a28

2. X86:5 bytes; X86-64: 9 bytes

3. X86:4 bytes; X86-64: 8 bytes

Problem 4: (10 points)

1 [1] 3

[2] x 111 yyyyyyy , where x is 0 or 1. $(yyyyyyy)_2$ is non-zero

[3] 0 001 0000000

[4] 0 000 1111111

2 1 101 0011110

3 -0.6640625 * 2(-2) =-0.166015625

Problem 5: (25 points)

1 [1] ‘a’ [2] result += x < i ? x : i
[3] ‘c’ [4] return -1
[5] 10 [6] 5
[7] .L3 [8] .L6
[9] %ebx [10] $5
[11] *.L7(,%ebx,4) [12] %edx
[13] $9 [14] %ecx

2 The purpose is to expand the variable “type” from 1 byte to 4 bytes so that it can be stored in a 32-bit register.

No.

The signed bit of characters ranging from ‘a’ to ‘f’ are all ‘0’. Thus expanding them with ‘0’, or expanding with the signed bit will make no difference. As for the other characters, they will finally jump to “default” even their value changes during the expanding process, so it will make no difference as well.

Problem 6: (24 points)

1 [1] fe830408 [2] 80483e0
[3] 0x8049618 [4] %ebp
[5] 0x80484f0 [6] leave

2 [1] 0xffffdb9c [2] 0xffffdba8
[3] 0x8049614 [4] 0xffffdba8
[5] 0xffffdb80 [6] 0xffffdb98

3 [1] 0x80483fe [2] The starting address of function bar1()
[3] 0xffffdb9c [4] Saved %esp before implicitly calling bar1()
[5] 0x8048415 [6] The starting address of function bar2()
[7] 0xffffdb9c [8] Saved %esp before implicitly calling bar2()

4 Modify the instruction in address 0x80483f7 as below:
lea 0x8049618,%esp