CAS CS 210 - Computer Systems

Fall 2014

SOLUTIONS: PROBLEM SET 2 (PS2) (ASSEMBLY LANGUAGE AND PROGRAM REPRESENTATIONS)

OUT: SEPTEMBER 25

DUE: OCTOBER 16, 1:30 PM

NO LATE SUBMISSIONS WILL BE ACCEPTED

Problem 1: 6 Points

Match each of the assembler routines on the left with the equivalent C function on the right.

```
foo1:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
shrl $31, %eax
popl %ebp
ret

foo2:
pushl %ebp
movl %esp, %ebp
movl $0, %eax
popl %ebp
ret

foo3:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
addl %eax, %eax
leal 0(,%eax,8), %edx
movl %edx, %ecx
subl %eax, %ecx
movl %ecx, %eax
popl %ebp
ret

foo4:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
addl $13, %eax
leal 3(%eax), %edx
testl %eax, %eax
cmovs %edx, %eax
sarl $2, %eax
popl %ebp
ret

foo5:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
leal 15(%eax), %edx
testl %eax, %eax
cmovs %edx, %eax
sarl $4, %eax
popl %ebp
ret

foo6:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
sarl $31, %eax
popl %ebp
ret
```

```
int choice1(int x) // foo5
{
    return x / 16;
}

int choice2(int x) // foo3
{
    return 14 * x;
}

int choice3(int x) // foo2
{
    return (x << 31) & 1;
}

int choice4(int x) // foo1
{
    return (x < 0);
}

int choice5(int x) // foo4
{
  return (x + 13) /4;
}

int choice6(int x) // foo6
{
    return (x >> 31);
}
```

Fill in your answers here:

foo1 corresponds to choice __choice4.
foo2 corresponds to choice __choice3.
foo3 corresponds to choice __choice2.
foo4 corresponds to choice __choice5.
foo5 corresponds to choice __choice1.
foo6 corresponds to choice __choice6.

Problem 2: 9 Points

A: 3 Points

Consider the following C functions and assembly code:

```
int fun3(int a)
{
    return a * 128;
}

int fun12(int a)
{
    return a * 33;
}

int fun5(int a)
{
    return a * 65;
}
```

```
pushl   %ebp
movl    %esp, %ebp
movl    8(%ebp), %edx
movl    %edx, %eax
sall    $6, %eax
addl    %edx, %eax
popl    %ebp
ret
```

Which of the functions compiled into the assembly code shown?

ANSWER: fun5

B: 3 Points

Consider the following C functions and assembly code:

```
int fun3(int a, int b)
{
  if (a & b)
    return a;
  else
    return b;
}

int fun4(int a, int b)
{
  if (a & b)
    return b;
  else
    return a;
}

int fun5(int a, int b)
{
    if (a < b)
        return b;
    else
        return a;
}
```

```
pushl   %ebp
movl    %esp, %ebp
movl    12(%ebp), %eax
movl    8(%ebp), %edx
andl    %edx, %eax
testl   %eax, %eax
je      .L2
movl    12(%ebp), %eax
jmp     .L3
.L2:
movl    8(%ebp), %eax
.L3:
popl    %ebp
ret
```

Which of the functions compiled into the assembly code shown?

ANSWER: fun3

C: Points 3

Consider the following C functions and assembly code:

```
int funA(int *a, int idx, int *b)
{
  if (a[idx] > *b)
    *b = a[idx];
  else
    *b = 2 * *b;
  return *b;
}

int funB(int *a, int idx, int *b)
{
  if (b[idx] > *a)
    *a = b[idx];
  else
    *a = 2 * *a;
  return *a;
}

int funC(int *a, int idx, int *b)
{
  if (a[idx] > b)
    b = a[idx];
  else
    b = 2 * b;
  return b;
}
```

```
        pushl   %ebp
        movl    %esp, %ebp
        movl    16(%ebp), %eax
        movl    12(%ebp), %ecx
        movl    8(%ebp), %edx
        movl    (%edx,%ecx,4), %ecx
        movl    (%eax), %edx
        cmpl    %edx, %ecx
        jle     .L2
        movl    %ecx, (%eax)
        jmp     .L3
.L2:
        addl    %edx, %edx
        movl    %edx, (%eax)
.L3:
        movl    (%eax), %eax
        popl    %ebp
        ret
```

Which of the functions compiled into the assembly code shown?

ANSWER: funA

Problem 3: 10 Points

Consider the following assembly representation of a function `foo` containing a `for` loop:

```
1  bar:
2          pushl   %ebp
3          movl    %esp, %ebp
4          subl    $16, %esp
5          movl    8(%ebp), %eax
6          addl    %eax, %eax
7          movl    %eax, -4(%ebp)
8          movl    $0, -8(%ebp)
9          jmp     .L2
10 .L3:
11         movl    -8(%ebp), %eax
12         addl    $7, %eax
13         addl    %eax, -4(%ebp)
14         movl    -8(%ebp), %eax
15         leal    5(%eax), %edx
16         movl    -4(%ebp), %eax
17         imull   %edx, %eax
18         movl    %eax, -4(%ebp)
19         addl    $1, -8(%ebp)
20 .L2:
21         movl    -8(%ebp), %eax
22         cmpl    8(%ebp), %eax
23         jl      .L3
24         movl    -4(%ebp), %eax
25         leave
26         ret
```

Fill in the blanks to provide the functionality of the loop:

```
int bar(int x)
{
    int i;
    int val = _______________;

    for( _________; _________; i++ ) {

        ____________________;

        ____________________;

    }
    return val;
}
```

ANSWER:

```
int bar(int x)
{
    int i;
    int val = x * 2;

    for (i=0; i < x; i++) {
        val += (i + 7);
       val *= (i + 5);
    }
    return val;
}
```

Problem 4: Points 12

```
#include <stdio.h>
#include <stdlib.h>

typedef long long Unum;
#define NAMELEN 80

struct Emp {
  Unum          id;
  char          name[NAMELEN];
  int           salary;
  struct Emp *next;
};

struct Emp *Emp_list = 0;

Unum Emp_get_id(struct Emp *emp) { return emp->id; }

void Emp_set_id(struct Emp *emp, Unum id) { emp->id = id; }

void Emp_get_name(struct Emp *emp, char *name) {
  int i;
  for (i=0;i<NAMELEN; i++) name[i] = emp->name[i];
}

void Emp_set_name(struct Emp *emp, char *name) {
  int i;
  for (i=0;i<NAMELEN; i++) emp->name[i] = name[i];
}

int Emp_get_salary(struct Emp *emp) { return emp->salary; }

void Emp_set_salary(struct Emp *emp, int salary) { emp->salary = salary; }

struct Emp * Emp_get_next(struct Emp *emp) { return emp->next; }

void Emp_set_next(struct Emp *emp, struct Emp *next) { emp->next = next; }

void Emp_Emp(struct Emp *emp, Unum id, char *name, int salary) {
  Emp_set_id(emp, id); Emp_set_name(emp, name); Emp_set_salary(emp, salary);
  Emp_set_next(emp, 0);
}

struct Emp *Emp_new() { return malloc(sizeof(struct Emp)); }

void Emp_add(Unum id, char *name, int salary) {
  struct Emp *emp = Emp_new();
  Emp_Emp(emp, id, name, salary);
  Emp_set_next(emp, Emp_list);
  Emp_list = emp;
}
```

Given the above code and the following disassembly

```
08048510 <mystery>:
 8048510: 8b 15 ac 98 04 08     mov    0x80498ac,%edx
 8048516: 31 c0                 xor    %eax,%eax
 8048518: 55                    push   %ebp
 8048519: 89 e5                 mov    %esp,%ebp
 804851b: 85 d2                 test   %edx,%edx
 804851d: 74 0b                 je     804852a <mystery+0x1a>
 804851f: 90                    nop
 8048520: 03 42 58              add    0x58(%edx),%eax
 8048523: 8b 52 5c              mov    0x5c(%edx),%edx
 8048526: 85 d2                 test   %edx,%edx
 8048528: 75 f6                 jne    8048520 <mystery+0x10>
 804852a: 5d                    pop    %ebp
 804852b: c3                    ret
```

Assuming `&Emp_list` is `0x80498ac` fill in the following table. Your explanations should not just be a restatement of the assembly code. Rather the explanation sould be interms of the what the assembly is doing in context of the above 'C' code. Here are two examples of the kind of explanations we are looking for: 1) "save old frame pointer" and 2) "test if the list is empty".

Address	Explanation
8048510	initialize edx to emp_list
8048516	
8048518	
8048519	
804851b	
804851d	
804851f	NOP
8048520	
8048523	
8048526	
8048528	
804852a	restore old frame pointer
804852b	return

What purpose does the mystery function server eg. what is it doing?

ANSWER:

Address	Explanation
8048510	initialize edx to emp_list
8048516	initialize return value to 0
8048518	save old frame pointer
8048519	set current frame pointer
804851b	test if list is empty
804851d	if empty return with 0 value
804851f	NOP
8048520	return value += employee-¿salary
8048523	move to next employee (employee = employee-¿next)
8048526	test if we are at the end of the list
8048528	if not loop back to 8048520
804852a	restore old frame pointer
804852b	return

The mystery function walks the employee list and calculates the total salary of all the employees.

Problem 5: 14 Points

Consider the following C code

```
1  #include <stdio.h>
2
3  void bar(char *buf, char *src)
4  {
5    while (*src) {
6      *buf = *src;
7      buf++; src++;
8    }
9    return;
10 }
11
12 void foo(void)
13 {
14   int i = 0;
15   char buf[4];
16
17   bar(buf, "Hello World!");
18   printf("0x%x 0x%x\n", &i, i);
19
20   return;
21 }
22
23 int main(int argc, char **argv)
24 {
25   foo();
26   return 1;
27 }
```

and the following dissasembly:

```
1  Dump of assembler code for function foo:
2     0x080483ec <+0>:     push   %ebp
3     0x080483ed <+1>:     mov    %esp,%ebp
4     0x080483ef <+3>:     sub    $0x28,%esp
5     0x080483f2 <+6>:     movl   $0x0,-0xc(%ebp)
6     0x080483f9 <+13>:    movl   $0x8048504,0x4(%esp)
7     0x08048401 <+21>:    lea    -0x10(%ebp),%eax
8     0x08048404 <+24>:    mov    %eax,(%esp)
9     0x08048407 <+27>:    call   0x80483c4 <bar>
10    0x0804840c <+32>:    mov    -0xc(%ebp),%eax
11    0x0804840f <+35>:    mov    %eax,0x8(%esp)
12    0x08048413 <+39>:    lea    -0xc(%ebp),%eax
13    0x08048416 <+42>:    mov    %eax,0x4(%esp)
14    0x0804841a <+46>:    movl   $0x8048511,(%esp)
15    0x08048421 <+53>:    call   0x80482f4 <printf@plt>
16    0x08048426 <+58>:    leave
17    0x08048427 <+59>:    ret
```

Use the following table to translate the ASCII characters to their hexadecimal values.

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 space	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

Part A

Given the code, the ascii chart on the previous page, and the following starting values, fill in the following memory diagram with execution proceeding up to 0x0804840c.

```
pc  = 0x080483ed
esp = 0xffffd368
```

Memory values not updated may be left blank. Remember that an int value is 4 bytes located with the least significant byte at the address and the remaining 3 bytes in the successive byte addresses. Eg. If we know that six bytes starting at 0xbfffec10 is 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 then we would have to write down :

```
0xbfffec10:   04030201
0xbfffec14:   ????0605
```

Individual bytes of an int that whose value are unknown should be specifed as `??`.

Address	int hex value	Description
0xffffd36c	0x08048433	return address for call to foo
0xffffd368	0xffffd378	old frame pointer
0xffffd364		
0xffffd360		
0xffffd35c		
0xffffd358		
0xffffd354		
0xffffd350		
0xffffd34c		
0xffffd348		
0xffffd344		
0xffffd340		
0xffffd33c		

In the descriptions be sure to indicate if an address corresponds to a specific variable or argument and its value or if an address is a return address and its value.

ANSWER:

Address	int hex value	Description
0xffffd36c	0x08048433	return address for call to foo
0xffffd368	0xffffd378	old frame pointer
0xffffd364		
0xffffd360	0x21646c72	
0xffffd35c	0x6f57206f	i
0xffffd358	0x6c6c6548	buf[0-4]
0xffffd354		
0xffffd350		
0xffffd34c		
0xffffd348		
0xffffd344	0x08048504	arg 2: string to bar
0xffffd340	0xffffd358	arg 1: buf to bar
0xffffd33c	0x0804840c	return address

Part B

Provide the output from the printf in the `foo` function:

ANSWER: 0xffffd35c 0x6f57206f

Problem 6: 10 Points

Consider the following incomplete definition of a C struct along with the incomplete code for a function `func` given below.

```
typedef struct node {

  ________________ x;

  ________________ y;

  struct node *next;

  struct node *prev;

} node_t;
```

```
node_t n;

void func() {

  node_t *m;

  m = ____________________________;

  m->y /= 16;

  return;
}
```

When this C code was compiled on an IA-32 machine running Linux, the following assembly code was generated for function `func`.

```
func:
  pushl %ebp
  movl n+12,%eax
  movl 16(%eax),%eax
  movl %esp,%ebp
  movl %ebp,%esp
  shrw $4,8(%eax)
  popl %ebp
  ret
```

Given these code fragments, fill in the blanks in the C code given above. Note that there is a unique answer.

The types must be chosen from the following table, assuming the sizes and alignment given.

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
unsigned short	2	2
int	4	4
unsigned int	4	4
double	8	4

```
ANSWER:
double x
unsigned short y
n.next->prev
```