Vidya Akavoor

CAS CS 210 - Computer Systems

Fall 2016

PROBLEM SET 2 (PS2) (ASSEMBLY LANGUAGE AND PROGRAM REPRESENTATIONS)
OUT: SEPTEMBER 27
DUE: OCTOBER 18, 1:30 PM

NO LATE SUBMISSIONS WILL BE ACCEPTED

imul = signed mult
test = logical and
sal = arith left shift
sar = " right
shr = shift right logical

Problem 1: 6 Points

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

```
foo1:
        lea     0x3(%rdi),%eax      3+x
        test    %edi,%edi
        cmovns  %edi,%eax
        sar     $0x2,%eax
        retq

foo2:
        lea     0xf(%rdi),%edx      15+x
        lea     0x16(%rdi),%eax     22+x
        test    %edx,%edx
        cmovns  %edx,%eax
        sar     $0x3,%eax
        retq

foo3:
        mov     %edi,%eax
        shr     $0x1f,%eax
        retq

foo4:
        mov     $0x0,%eax
        retq

foo5:
        lea     (%rdi,%rdi,2),%eax  rdi+2rdi
        shl     $0x2,%eax
        retq

foo6:
        mov     %edi,%eax
        sar     $0x1f,%eax
        retq
```

```
int choice1(int x)
{
  return x / 4;
}

int choice2(int x)
{
  return 12 * x;
}

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

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

int choice5(int x)
{
  return (x + 15) /8;
}

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

Fill in your answers here:

foo1 corresponds to choice 1.

foo2 corresponds to choice 5.

foo3 corresponds to choice 4.

foo4 corresponds to choice 3.

foo5 corresponds to choice 2.

foo6 corresponds to choice 6.

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 * 65;
}

int fun5(int a)
{
    return a * 33;
}
```

```
mov    %edi,%eax
shl    $0x5,%eax
add    %edi,%eax
retq
```

multiply by 32
add 1 more
*33

Which of the functions compiled into the assembly code shown? fun5

B: 3 Points

Consider the following C functions and assembly code:

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

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

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

```
                    b   a
        test    %esi,%edi
        je      .L0
        mov     %edi,%eax
        retq
.L0:
        mov     %esi,%eax
        retq
```

Which of the functions compiled into the assembly code shown? fun 4

C: Points 3

Consider the following C functions and assembly code:

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

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

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

```
                    movslq %esi,%rsi   convert idx to long
                    mov    (%rdi,%rsi,8),%rcx
                    mov    (%rdx),%rax
                    cmp    %rax,%rcx
                    jle    .L1
                    mov    %rcx,(%rdx)
                    jmp    .L2
.L1:
                    add    %rax,%rax
                    mov    %rax,(%rdx)
.L2:
                    mov    (%rdx),%rax
                    retq
```

Which of the functions compiled into the assembly code shown? funA

rcx = 8*rsi + rdi a[idx]
rax = *b
if rcx ≤ rax
    2*rax
    rdx = rax
else
  rdx = rcx
rax = rdx
ret rax

Problem 3: 10 Points

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

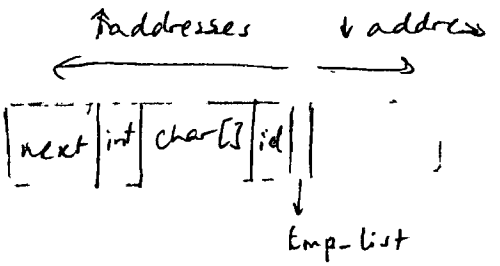
```
1  bar:                3 + rdi
2          lea     (%rdi,%rdi,2),%eax
3          mov     $0x0,%edx
4          jmp     .L2
5  .L3:                5 + rdx + rax
6          lea     0x5(%rdx,%rax,1),%eax
7          lea     0x3(%rdx),%ecx   3 + rdx
8          imul    %ecx,%eax        eax = eax * ecx
9          add     $0x1,%edx        edx += 1
10 .L2:
11         cmp     %edi,%edx        edx - edi
12         jl      .L3
13         retq
```

edi = x
edx = i
eax = val

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

```
int bar(int x)
{
    int i;
    int val = ____x*3____;

    for( ___i=0___; ___i<x___; i++ ) {
        ___val = 5+i+val;___
        ___val = val*(3+i)___;

    }
    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 objdump output

```
<mystery>:
000000000040061f <mystery>:
  40061f:       48 8b 15 1a 0a 20 00    mov    0x200a1a(%rip),%rdx  # 601040 <Emp_list>
  400626:       b8 00 00 00 00          mov    $0x0,%eax
  40062b:       eb 07                   jmp    400634 <mystery+0x15>
  40062d:       03 42 58                add    0x58(%rdx),%eax  --> y to sala
  400630:       48 8b 52 60             mov    0x60(%rdx),%rdx
  400634:       48 85 d2                test   %rdx,%rdx
  400637:       75 f4                   jne    40062d <mystery+0xe>
  400639:       c3                      retq
```

Assuming `&Emp_list` is `0x601040` 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) "load rdx with the employee id" and 2) "test if the list is empty". Note: Use x86_64 alignment rules thus pointers are 8 bytes in size and 8 bytes aligned.

Address	Explanation
40061f	initialize rdx to value of Emp_list
400626	initialize eax to 0, this will be incremented in the salaries then returned
40062b	jump to test if we are in loop (which sums salaries by moving the pointer rdx to each employee in Emp_List)
40062d	Look at addresses above rdx to get salary and add to eax
400630	look at addresses above rdx to get to next Emp. (i.e. look at the next Emp in list)
400634	test to see if there are anymore Emps left (the loop will continue if there are)
400637	if test indicates there are Emps, jump to loop; else continue to return
400639	return

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

The mystery function sums the salaries of all the employees stored in Emp_list and returns the sum.

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     0x0000000000400545 <+0>:     sub    $0x18,%rsp
3     0x0000000000400549 <+4>:     movl   $0x0,0xc(%rsp)
4     0x0000000000400551 <+12>:    mov    $0x400620,%esi
5     0x0000000000400556 <+17>:    mov    %rsp,%rdi
6     0x0000000000400559 <+20>:    callq  0x400530 <bar>
7     0x000000000040055e <+25>:    mov    0xc(%rsp),%edx
8     0x0000000000400562 <+29>:    lea    0xc(%rsp),%rsi
9     0x0000000000400567 <+34>:    mov    $0x400631,%edi
10    0x000000000040056c <+39>:    mov    $0x0,%eax
11    0x0000000000400571 <+44>:    callq  0x400410 <printf@plt>
12    0x0000000000400576 <+49>:    add    $0x18,%rsp
13    0x000000000040057a <+53>:    retq
```

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 0x000000000040055e.

```
pc  = 0x0000000000400545
rsp = 0x00007fffffffe018
```

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 has values 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 then we would have to write down :

```
0x0000000bfffec10:    04030201
0x0000000bfffec14:    ????0605
```

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

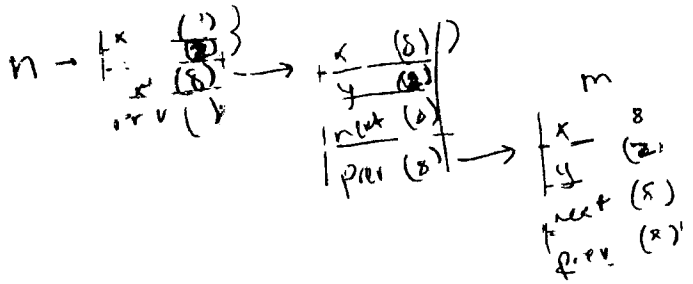
Address	int hex value	Description
rsp → 0x7fffffffe000	0x6c6c6548	stores Hell which is 1ST 4 bytes of "Hello World???"
0x7fffffffe004	0x6f57206f	next 4 bytes of string above
0x7fffffffe008	0x3f646c72	next 4 bytes
rsp+12 → 0x7fffffffe00c	0x003f3f3f	corresponds to i; overwritten by next 4 bytes because of bar
0x7fffffffe010		
0x7fffffffe014		
0x7fffffffe018	0x00400584	low 32 bits of return address
0x7fffffffe01c	0x00000000	high 32 bits of return address

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.

Part B

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

0x7fffffffe00c 0x003f3f3f



Problem 6: 10 Points

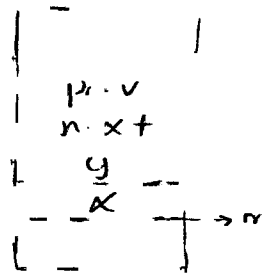
Consider the following incomplete definition of a C struct along with the incomplete code for a function `func` given below.

```
typedef struct node {

  double          x;

  unsigned short  y;

  struct node *next;

  struct node *prev;

} node_t;
```

```
node_t n;

void func() {

  node_t *m;

  m = n.next->prev;

  m->y *= 8;

  return;
}
```

When this C code was compiled on an IA-64 machine running Linux, the following assembly code was generated for function `func`.

```
func:
  movq n+16(%rip),%rax
  movq 24(%rax),%rax
  shlq $0x3,8(%rax)
  retq
```



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. Remember that pointers on x86_64 are 8 byte aligned.

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