CSPB 2400 - Park - Computer Systems

<u>Dashboard</u> / My courses / <u>2241:CSPB 2400</u> / <u>26 February - 3 March</u>

/ Exam #2: Machine Level Representation of Programs [Section 100] (Remotely Proctored)

Started on Wednesday, 28 February 2024, 7:16 PM

State Finished

Completed on Wednesday, 28 February 2024, 8:37 PM

Time taken 1 hour 20 mins

Marks 91.50/100.00

Grade 9.15 out of 10.00 (92%)

Question 1

Correct

Mark 3.00 out of 3.00

Determine the appropriate instruction suffix based on the operands.





Your answer is correct.

Because this refers to 64-bit register values (e.g. %rcx), you would use movq.

The correct answer is:

Determine the appropriate instruction suffix based on the operands.

mov[q] %rcx, (%rsp)

Question 2
Correct
Mark 4.00 out of 4.00
For the following instruction:
addq 4(%rdx),%rax
what is the size of the addition results in bytes?
Select one:
a. 1 Byte
b. 2 Bytes
c. 4 Bytes
⊙ d. 8 Bytes ✓
Your answer is correct.
The correct answer is: 8 Bytes
Correct Mark 4.00 out of 4.00 If %rbx=6 and %rsi=1, what is the value of %r12 after executing
leaq 2(%rbx, %rsi), %r12
You can use an expression is that's useful.
9
Your last answer was interpreted as follows: 9
Correct answer, well done. The value computed by
leaq 2(%rbx, %rsi), %r12
is "%rbx+%rsi + 2" or $6+1+2=9$.
A correct answer is 9, which can be typed in as follows: 9

Correct

Mark 4.00 out of 4.00

If %r9=9, what is the value of %r12 after executing

leaq 1(, %r9, 4), %r12

You can use an expression is that's useful.

37

Your last answer was interpreted as follows: 37

Correct answer, well done.

The value computed by

leaq 1(, %r9, 4), %r12

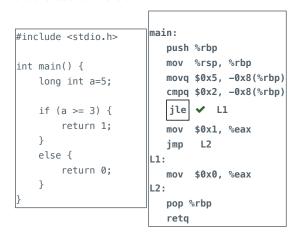
is "%r9 * 4 + 1" or 9*4 + 1 = 37.

A correct answer is 37, which can be typed in as follows: 37

Correct

Mark 6.00 out of 6.00

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.





Your answer is correct.

The correct answer is:

```
#include <stdio.h>
                        main:
                           push %rbp
int main() {
                           mov %rsp, %rbp
    long int a=5;
                           movq $0x5, -0x8(%rbp)
                           cmpq $0x2, -0x8(%rbp)
    if (a >= 3) {
                           [jle] L1
        return 1;
                           mov $0x1, %eax
                           jmp
                                 L2
    else {
                        L1:
        return 0;
                           mov $0x0, %eax
    }
                        L2:
                           pop %rbp
                           retq
```

Correct

Mark 6.00 out of 6.00

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.

```
int a,b,x,y;
                                foo:
int foo() {
                                           y, %eax
                                   movl
 if (a
                b) {
                                           b, %edx
                                   movl
                                           %edx, a
                                   cmpl
    return x;
                                   jne
                                           L3
  } else {
                                   movl
                                           x, %eax
     return y;
                                L3:
                                   ret
```



Your answer is correct.

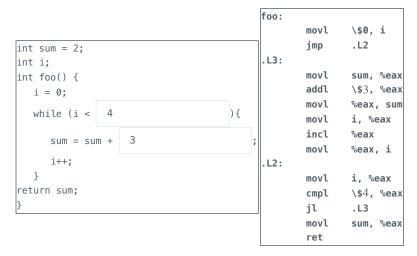
The correct answer is:

```
int a,b,x,y;
                                       y, %eax
int foo() {
                               movl
  if (a [==] b) {
                                       b, %edx
                               movl
    return x;
                               cmpl
                                       %edx, a
   } else {
                               jne
                                       L3
     return y;
                               movl
                                       x, %eax
                            L3:
                               ret
```

Correct

Mark 4.00 out of 4.00

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.



Your last answer was interpreted as follows: 4

Your last answer was interpreted as follows: 3

Correct answer, well done.

Correct answer, well done.

Correct answer, well done.

The **jl** instruction jumps to the start of the loop if the condition is true. The comparison is checking if "i <=4". The value of **sum** is incremented by 3 each time through the loop.

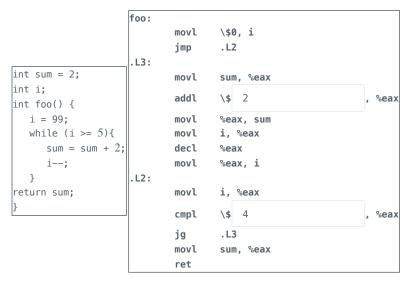
A correct answer is 4, which can be typed in as follows: 4

A correct answer is 3, which can be typed in as follows: 3

Correct

Mark 4.00 out of 4.00

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.



Your last answer was interpreted as follows: 4

Your last answer was interpreted as follows: 2

Correct answer, well done.

Correct answer, well done.

Correct answer, well done.

The **jg** instruction jumps to the start of the loop if the condition is true. The comparison is checking if "i > 4" which is the same as "i >= 5". The value of **sum** is incremented by 2 each time through the loop.

A correct answer is 2, which can be typed in as follows: 2

A correct answer is 4, which can be typed in as follows: 4

Correct

Mark 4.00 out of 4.00

If we compile the following C function, we get the assembly code shown below. Fill in the missing values in the assembly to match the C code.

```
int ii;
int limit;
int foo() {
  int sum = 0;
  for (int i = ii; i \le 6; i += 2) {
   sum += bar(sum,i);
  }
  return sum;
}
 foo:
 .LFB0:
                 pushq
                          %rbx
                                        movl
                                                \$0, sum
                                                                   movl
                                                                            ii, %ebx
                                                                                              cmpl
                                                                                                       \$
  6
                           %ebx
                                         jg
                                                  .L2 .L3:
                                                                     movl
                                                                              %ebx, %esi
                                                                                                  movl
                                                                                                           sum, %edi
call
        bar
                     addl
                              %eax, sum
                                                  addl
                                                          \$
                                                              2
                                                                                        %ebx
                                                                                                       cmpl
                                                                                                               \$
                                                                              sum, %eax
                           %ebx
                                         jι
                                                  .L3 .L2:
                                                                     movl
                                                                                                 popq
                                                                                                          %rbx
                                                                                                                        ret
```

Your last answer was interpreted as follows: 2

Your last answer was interpreted as follows: 7

Your last answer was interpreted as follows: 6

Correct answer, well done.

Correct answer, well done.

Correct answer, well done.

Correct answer, well done.

The body of the **for** loop should not be executed if $\mathbf{i} > 6$. The first \mathbf{jg} jumps around the loop (skipping the body), and thus the comparison is against 6. The loop increments by 2 each time. Since the **for** loop has been transformed into a **while** loop, we need to keep executing the loop while $\mathbf{i} < \mathbf{e}$, or, equivalently, $\mathbf{i} < \mathbf{f}$, which is what the last \mathbf{jl} does.

A correct answer is 6, which can be typed in as follows: 6

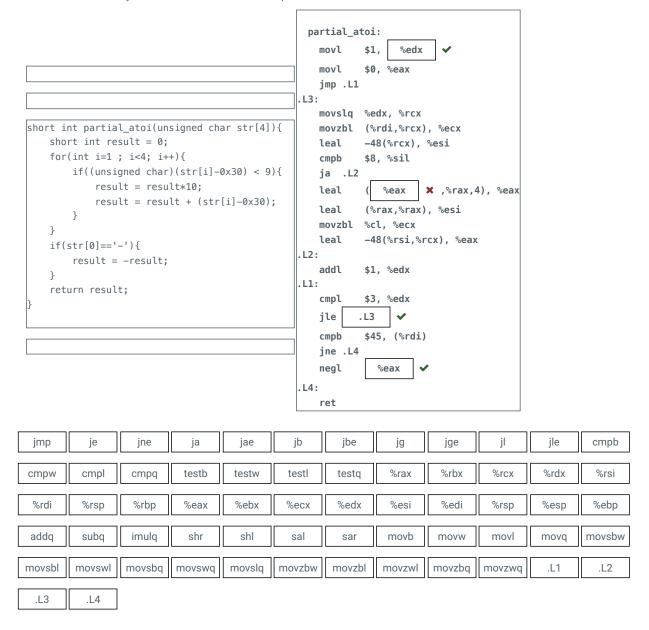
A correct answer is 2, which can be typed in as follows: 2

A correct answer is 7, which can be typed in as follows: 7

Partially correct

Mark 4.50 out of 6.00

This question is designed to test your knowledge of loops and conditionals in assembly. Based on the C code shown here, fill in the five blanks in the assembly below. Each blank is worth four points.



Your answer is partially correct.

You have correctly selected 3.

The correct answer is:

This question is designed to test your knowledge of loops and conditionals in assembly. Based on the C code shown here, fill in the five blanks in the assembly below. Each blank is worth four points.

```
partial_atoi:
                                                           $1, [%edx]
                                                   movl
                                                   movl
                                                           $0, %eax
                                                   jmp .L1
                                               .L3:
                                                   movslq %edx, %rcx
short int partial_atoi(unsigned char str[4]){
                                                   movzbl (%rdi,%rcx), %ecx
   short int result = 0;
                                                   leal
                                                           -48(%rcx), %esi
    for(int i=1; i<4; i++){
                                                           $8, %sil
                                                   cmpb
       if((unsigned char)(str[i]-0x30) < 9){
                                                   ja .L2
           result = result*10;
                                                   leal
                                                           ([%rax],%rax,4), %eax
           result = result + (str[i]-0x30);
                                                           (%rax,%rax), %esi
                                                   leal
                                                   movzbl %cl, %ecx
                                                           -48(%rsi,%rcx), %eax
                                                   leal
   if(str[0]=='-'){
                                               .L2:
        result = -result;
                                                   addl
                                                           $1, %edx
                                               .L1:
   return result;
                                                   cmpl
                                                           $3, %edx
                                                   jle [.L3]
                                                   cmpb
                                                           $45, (%rdi)
                                                   jne .L4
                                                   negl
                                                           [%eax]
                                                .L4:
                                                   ret
```

Correct

Mark 6.00 out of 6.00

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.

```
foo:
                                 cmpl
                                         $1, %edi
                                         .L3
                                 jle
int foo(int a) {
                                 pushl
                                          %edi
  int rval;
                                 leal
                                         -1(%rdi),%edi
  if (a <= 1)
                                 call
                                         foo
    return 1;
                                          %edi
                                 popl
  rval = foo(a
                                 addl
                                         %edi, %eax
  return rval+a;
                                 jmp
                                         .L2
                              .L3:
}
                                 movl
                                         $1, %eax
                              .L2:
                                 ret
```



Your answer is correct.

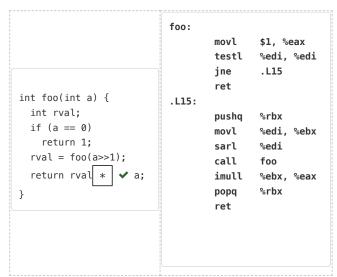
The correct answer is:

```
foo:
                                      $1, %edi
                              cmpl
                                      .L3
                              jle
int foo(int a) {
                              pushl
                                       %edi
  int rval;
                                      -1(%rdi),%edi
                              leal
  if (a <= 1)
                              call
                                      foo
    return 1;
                              popl
                                       %edi
  rval = foo(a[-]1);
                              addl
                                      %edi, %eax
  return rval+a;
                              jmp
                                      .L2
                           .L3:
}
                              movl
                                      $1, %eax
                           .L2:
                              ret
```

Correct

Mark 6.00 out of 6.00

The assembly code on the right partially implements the C function shown on the left. Fill in the missing instruction to correctly implement the C function on the left.







Your answer is correct.

The correct answer is:

```
foo:
                                  movl
                                           $1, %eax
                                  testl
                                           %edi, %edi
                                  jne
                                           .L15
                                  ret
int foo(int a) {
                          .L15:
  int rval;
                                  pushq
                                          %rbx
  if (a == 0)
                                  movl
                                           %edi, %ebx
    return 1;
                                  sarl
                                           %edi
  rval = foo(a>>1);
                                  call
                                           foo
  return rval[*]a;
                                  imull
                                           %ebx, %eax
}
                                           %rbx
                                  popq
                                  ret
```

Correct

Mark 6.00 out of 6.00

Given the following assembly code:

```
rfun:
       movl
               $0, %eax
       testl
               %edi, %edi
               .L30
       jе
               %rbx
       pushq
       movl
               %edi, %ebx
               $2, %edi
       sarl
       movslq %edi, %rdi
                rfun
       call
       movslq %ebx, %rbx
       subq
                %rax, %rbx
               %rbx, %rax
       movq
               %rbx
       popq
.L30:
        rep ret
```

Fill in the blanks by dragging the appropriate entries below:

```
Your answer is correct.
```

The correct answer is:

Given the following assembly code:

```
rfun:
        movl
                $0, %eax
        testl
                %edi, %edi
                .L30
        jе
               %rbx
        pushq
                %edi, %ebx
       movl
                $2, %edi
        sarl
       movslq %edi, %rdi
                rfun
       call
       movslq %ebx, %rbx
        subq
                %rax, %rbx
       movq
                %rbx, %rax
       popq
                %rbx
.L30:
        rep ret
```

Fill in the blanks by dragging the appropriate entries below:

```
long rfun([int] x){
  if ( [x == 0] ) return 0;
  [int] nx = [x >> 2];
  long rv = rfun(nx);
  return [x - rv];
}
```

Correct

Mark 6.00 out of 6.00

Given the following C code:

```
long rfun(long x){
  if ( x < 0 ) return 0;
  long nx = x >> 2;
  long rv = rfun(nx);
  return x * rv;
}
```

Fill in the blanks by dragging the appropriate entries below:



testq %rdi, %rdi	cmpq \$100, %rdi	subq %rax, %rbx	addq %rbx, %rax	testl %edi, %edi	unsigned int	imulq %rdi, %rax
je .L6	js .L6	leaq (%rdi,%	ordi), %rdi sarq	\$2, %rdi s	hrl \$2, %edi	sarq \$3, %rdi
shra \$2 %rdi	i sarl \$2 %e	di ia l	6	J L	JL	

Your answer is correct.

The correct answer is:

Given the following C code:

```
long rfun(long x){
  if ( x < 0 ) return 0;
  long nx = x >> 2;
  long rv = rfun(nx);
  return x * rv;
}
```

Fill in the blanks by dragging the appropriate entries below:

```
rfun:
       movl
               $0, %eax
       [testq %rdi, %rdi]
        [js .L6]
              %rbx
       pushq
               %rdi, %rbx
       movq
               $2, %rdi]
        [sarq
       call
               rfun
        [imulq
               %rbx, %rax]
               %rbx
       popq
.L6:
       rep ret
```

Partially correct

Mark 2.00 out of 6.00

You've been given the following code: callee: movslq %esi, %rsi addq \$2, %rsi movl \$41, (%rdi,%rsi,4) movl 0(%rdi, %rsi, 4), %eax ret caller: $subq \ \$64$, %rsp movl %edi, %esi movl \\$22, (%rsp) movl \\$6, 4(%rsp) movl \\$6, 8(%rsp) movl \\$12, 12(%rsp) movl \\$3, 16(%rsp) movq %rsp, %rdi call callee addq \$64, %rsp Assume that %**rsp** = 4000_{10} on entry to function **caller**, which was called with argument **y=**1. What is the value of **%rsp** when executing the first instruction of **callee**? Your last answer was interpreted as follows: 22 You should enter the address as a decimal value and you may use an expression if it's useful. Into what memory location is 41 written in function **callee**? 3954 Your last answer was interpreted as follows: 3954 You should enter the address as a decimal value and you may use an expression if it's useful. What value is at address 3940? 6 Your last answer was interpreted as follows: 6 You should enter the address as a decimal value and you may use an expression if it's useful. Your answer is partially correct. Incorrect answer.

This code is compiled from the following C code

Incorrect answer.

Correct answer, well done.

The stack starts at address 4000. When routine **caller** is entered, 64 bytes are allocated for local variables by subtracting 64 from **%rsp**, leaving **%rsp** at 3936. An array of 5 integers named **buffer** is allocated at the new bottom of stack, 3936. Following that, a **call** instruction is executed, further reducing the stack value by 8 to 3928 on entry to **callee**. Routine **callee** is called with value 1 and 3936 (the pointer to the start of the array), and it updates **buffer**[1+2] = 41.

After execution, the entries of the array **buffer** are [22,6,6,41,3].

A correct answer is 3928, which can be typed in as follows: 3928

A correct answer is 3948, which can be typed in as follows: 3948

A correct answer is 6, which can be typed in as follows: 6

Partially correct

Mark 3.00 out of 6.00

The following C program

```
int a = /* insert your answer here */;
int b = /* insert your answer here */;

int all_the_money(int x, volatile int *buffer)
{
    printf("You got all the money!\n");
    exit();
}

int get_money(int x, volatile int* buffer, int y)
{
    buffer[x] = y;
}

int main(int argc, char **argv)
{
    int buffer[8];
    int money = get_money(a, buffer, b);
    printf("you got %d\n", money);
}
```

is compiled to the following:

```
all_the_money:
6aa: 48 83 ec 08
                              sub
                                     $0x8,%rsp
6ae: 48 8d 3d df 00 00 00
                              lea
                                     0xdf(%rip),%rdi # get stdin
6b5: e8 b6 fe ff ff
                              calla 570
6ba: 48 83 c4 08
                              add
                                     $0x8,%rsp
6be:
      c3
                              retq
main:
6bf:
      48 83 ec 28
                              sub
                                     $0x28,%rsp
6c3: 48 89 e6
                                     %rsp,%rsi
                              mov
6c6: 8b 15 4c 09 20 00
                                     b,%edx
                              mov
6cc: 8b 3d 4a 09 20 00
                              mov
                                     a,%edi
6d2: e8 22 00 00 00
                              callq 6f9
6d7: 89 c2
                                     %eax,%edx
                              mov
6d9: 48 8d 35 cb 00 00 00
                                     0xcb(%rip),%rsi # get stdin
                              lea
6e0: bf 01 00 00 00
                              mov
                                     $0x1,%edi
6e5: b8 00 00 00 00
                              mov
                                     $0x0,%eax
      e8 91 fe ff ff
6ea:
                              callq 580 <__printf_chk@plt>
6ef:
      b8 00 00 00 00
                              mov
                                     $0x0,%eax
6f4:
      48 83 c4 28
                              add
                                     $0x28,%rsp
6f8: c3
                              retq
get money:
6f9:
      48 63 ff
                              movslq %edi,%rdi
       48 8d 04 be
6fc:
                              lea
                                     (%rsi,%rdi,4),%rax
700:
       89 10
                                     %edx,(%rax)
                              mov
702:
       с3
                              retq
```

- A) In order to have this program call **all_the_money**, what should be the value of a in DECIMAL?
- B) In order to have this program call all_the_money, what should be the value of b 0x6aa

0x6aa

×

Procedure main allocates a stack frame of 0x28 or 40 bytes. This is done by the sub \$0x28,%rsp instruction

Then, arguments are prepared for the call to **get_money** in registers %rdi (a), %rsi (buffer) and %edx (b). The value for the address of **buffer** is the top of stack immediately prior to the function call. This means that the **buffer** array takes 8*4 bytes of a total of 40 allocates bytes of space in the stack frame. The other 8 bytes are not used.

Then, the call to get_money pushes an additional 8 bytes onto the stack.

If we diagram the stack it would look something like the following. We'll use U.R.A to mean the upper 4 bytes of a return address and L.R.A. to mean the lower 4 bytes of a return address. Since our PC values are small, the URA's will be zero. The column on the right shows how we can index the **buffer** variable to modify the corresponding location on the stack.

U.R.A. to main	buffer[11
L.R.A. to main	buffer[10
not used	buffer[9]
not used	buffer[8]
Α	buffer[7]
1	buffer[6]
	buffer[5]
buffer array	buffer[4]
	buffer[3]
I	buffer[2]
T	buffer[1]
V	buffer[0]
U. R. A. to get_money	buffer[-1]
L. R.A. to get_money	buffer[-2]

We are now ready for our attack. Our goal is to call **all_the_money** at address 0x6aa. To do this, we need to change either one of the L.R.A. values. We can do this one of two ways:

- 1. Change the return address of the call to main to 0x6aa. To do this we would set buffer[10] = 0x6aa.
- 2. Or, change the return address of the call to get_money to 0x6aa. To do this we would set buffer[-2] = 0x6aa.

Both of these are acceptable solutions.

Question 1	7
------------	---

Correct

Mark 4.00 out of 4.00

An array A is declared:

int A[2][6];

What is sizeof(A)?

48

Your last answer was interpreted as follows: 48

Correct answer, well done.

The sizeof(A) is the size of the total array. Each array element is 4 bytes and there are 2 rows of 6 columns, or 12 total elements. Thus, sizeof(A) = 48.

A correct answer is 48, which can be typed in as follows: 48

Ouestion 18

Correct

Mark 5.00 out of 5.00

An array A is declared:

#define L 6

#define M 2

#define N 3

double A[L][M][N];

Assuming the starting address of A is 200. What is &A[5] [1] [1]?

472

You can use an expression if that is useful.

Your last answer was interpreted as follows: 472

Correct answer, well done.

The array $\bf A$ consists of 6 planes each having 2 rows of 3 columns.

Each plane contains M*N*8 = 48 bytes. Thus, the start of plane #5 is 5*48 = 240.

Each column contains N*8=24 bytes. Thus, the start of row #1 is 1*24 = 24.

We then add in 8*1 = 8 bytes to get to the start of column #1.

Thus, &A[5][1][1] is 200+240+24+8=472.

A correct answer is 472, which can be typed in as follows: 472

Correct

Mark 5.00 out of 5.00

Assume common data sizes (char = 1 byte, short = 2, int = 4, long = 8, float = 4, double = 8) and that alignment requirements follow the data size.
struct {
} datum;
What is the offset of i[1] relative to &datum? 4
Your last answer was interpreted as follows: 4
Correct answer, well done.
What is the offset of c[3] relative to &datum?
Your last answer was interpreted as follows: 11
Correct answer, well done.
What is the offset of d relative to &datum?
Your last answer was interpreted as follows: 16
Correct answer, well done.
A correct answer is 4, which can be typed in as follows: 4
A correct answer is 11, which can be typed in as follows: 11
A correct answer is 16 , which can be typed in as follows: 16

Correct

Mark 5.00 out of 5.00

Assume common data sizes (char = 1 byte, short = 2, int = 4, long = 8, float = 4, double = 8) and that alignment requirements follow the data size.

struct {
 char c[2];
 int [[2];
 double d[5];
} datum[5];

What is the offset of datum[2].c[0] relative to &datum? 112

Your last answer was interpreted as follows: 112

Correct answer, well done.

What is the offset of datum[2].i[1] relative to &datum? 120

Your last answer was interpreted as follows: 120

Correct answer, well done.

What is the offset of datum[2].d[0] relative to &datum? 128

Your last answer was interpreted as follows: 128

Correct answer, well done.

The character array 'c' starts at the beginning of the struct and the offset relative to the struct is 0. The integer array 'i' starts at offset 4 and the double array 'd' starts at offset 16. The size of each struct is 56 and thus the offset to the beginning of datum[2] is 112. From there, you add the offset of each field multiplied by the index for that field. E.g. datum[2].i[1] is 112+4+4*1.

A correct answer is 112, which can be typed in as follows: 112

A correct answer is 120, which can be typed in as follows: 120

A correct answer is 128, which can be typed in as follows: 128