A. Extend Equation 3.1 from two dimensions to three to provide a formula for the location of array element A[i][j][k].

B. Use your reverse engineering skills to determine the values of R, S, and T based on the assembly code.

3.65 ♦
The following code transposes the elements of an $M \times M$ array, where $M$ is a constant defined by #define:

```c
void transpose(long A[M][M]) {
    long i, j;
    for (i = 0; i < M; i++)
        for (j = 0; j < i; j++) {
            long t = A[i][j];
            A[i][j] = A[j][i];
            A[j][i] = t;
        }
}
```

When compiled with optimization level `-O1`, `gcc` generates the following code for the inner loop of the function:

```c
.L6:
movq (%rdx), %rcx
movq (%rax), %rsi
movq %rsi, (%rdx)
movq %rcx, (%rax)
addq $8, %rdx
addq $120, %rax
cmpq %rdi, %rax
jne .L6
```

We can see that `gcc` has converted the array indexing to pointer code.

A. Which register holds a pointer to array element $A[i][j]$?

B. Which register holds a pointer to array element $A[j][i]$?

C. What is the value of $M$?

3.66 ♦
Consider the following source code, where `NR` and `NC` are macro expressions declared with `#define` that compute the dimensions of array $A$ in terms of parameter $n$. This code computes the sum of the elements of column $j$ of the array.
A. We can see on line 2 of function eval that it allocates 104 bytes on the stack. Diagram the stack frame for eval, showing the values that it stores on the stack prior to calling process.

B. What value does eval pass in its call to process?

C. How does the code for process access the elements of structure arguments?

D. How does the code for process set the fields of result structure r?

E. Complete your diagram of the stack frame for eval, showing how eval accesses the elements of structure r following the return from process.

F. What general principles can you discern about how structure values are passed as function arguments and how they are returned as function results?

3.68 In the following code, A and B are constants defined with #define:

typedef struct {
  int x[A][B]; /* Unknown constants A and B */
  long y;
} str1;

typedef struct {
  char array[B];
  int t;
  short s[A];
  long u;
} str2;

void setVal(str1 *p, str2 *q) {
  long v1 = q->t;
  long v2 = q->u;
  p->y = v1+v2;
}

Gcc generates the following code for setVal:

    void setVal(str1 *p, str2 *q)
      p in %rdi, q in %rsi
    1  setVal:
    2  movslq 8(%rsi), %rax
    3  addq 32(%rsi), %rax
4    movq  %rax, 184(%rdi)
5    ret

What are the values of A and B? (The solution is unique.)

3.69  
You are charged with maintaining a large C program, and you come across the following code:

typedef struct {
    int first;
    a_struct a[CNT];
    int last;
} b_struct;

void test(long i, b_struct *bp)
{
    int n = bp->first + bp->last;
    a_struct *ap = &bp->a[i];
    ap->x[ap->idx] = n;
}

The declarations of the compile-time constant CNT and the structure a_struct are in a file for which you do not have the necessary access privilege. Fortunately, you have a copy of the .o version of code, which you are able to disassemble with the OBDUMP program, yielding the following disassembly:

```
void test(long i, b_struct *bp)
  i in %rdi, bp in %rsi
0000000000000000 <test>:  
  0:  8b 8e 20 01 00 00  mov  0x120(%rsi),%ecx  
  3:  03 0e  add  (%rsi),%ecx  
  6:  48 8d 04 bf  lea  (%rdi,%rdi,4),%rax  
  9:  48 8d 04 c6  lea  (%rsi,%rax,8),%rax  
  c:  48 8b 50 08  mov  0x8(%rax),%rdx  
  f:  48 63 c9  movslq %ecx,%rcx  
  12: 48 89 4c d0 10  mov  %rcx,0x10(%rax,%rdx,8)  
  15: c3  retq
```

Using your reverse engineering skills, deduce the following:

A. The value of CNT.
B. A complete declaration of structure a_struct. Assume that the only fields in this structure are idx and x, and that both of these contain signed values.