Exercise 1: Write a function that adds two integers

```
add2ints:
                              pushl
                                      %ebp
                              movl
                                      %esp, %ebp
int add2ints(int A, int B)
                                      8(%ebp), %edx
                              movl
                                      12(%ebp), %eax
                              movl
  int C = A + B;
                                      %edx, %eax
                              addl
  return C;
                                      %ebp, %esp
                              movl
                                      %ebp
                              popl
                              ret
```

Exercise 2:

Write a function that compares two integers and returns 0/1 for mismatch/match

```
compareAB:
                                    pushl %ebp
int compareAB (int A, int B)
                                    movl %esp, %ebp
                                    subl $16, %esp
  int match = 0;
                                    movl $0, -4(%ebp)
                                    movl 8(%ebp), %eax
  if(A==B)
                                    cmpl 12(%ebp), %eax
     match = 1;
                                    ine .L2
                                    movl $1, -4(%ebp)
  return match;
                                 .L2:
                                    movl -4(%ebp), %eax
                                    leave
```

ret

Exercise 2:

Write a function that compares two integers and returns 0/1 for mismatch/match

```
compareAB:
                                    pushl %ebp
int compareAB (int A, int B)
                                    movl %esp, %ebp
                                    subl $16, %esp
  int match = 0;
                                    movl $0, -4(%ebp)
                                    movl 8(%ebp), %eax
  if(A==B)
                                    cmpl 12(%ebp), %eax
     match = 1;
                                    ine .L2
                                    movl $1, -4(%ebp)
  return match;
                                 .L2:
                                    movl -4(%ebp), %eax
                                    leave
                                             movl
                                                   %ebp, %esp
                                    ret
                                                   %ebp
                                             popl
```

Exercise 3:

Write a function that compares two characters and returns 0/1 for mismatch/match

```
compareAB:
                                    pushl %ebp
int compareAB (char A, char B)
                                    movl %esp, %ebp
                                    subl $16, %esp
  int match = 0;
                                    movl 8(%ebp), %edx
                                    movl 12(%ebp), %eax
                                    movl $0, -4(%ebp)
  if(A==B)
                                    cmpb %dl, %al
     match = 1;
                                    ine .L2
                                    movl $1, -4(%ebp)
   return match;
                                 .L2:
                                    movl -4(%ebp), %eax
                                    leave
                                    ret
```

Exercise 4: Write a function that returns the Nth character in a string

Exercise 4: Write a function that returns the Nth character in a string

```
getNthChar:
char getNthChar (char *str, int N)
                                      pushl %ebp
                                            %esp, %ebp
                                      movl
   char c = str[N];
                                      movl 12(%ebp), %edx
   return c;
                                            8(%ebp), %eax
                                      movl
                                            %edx, %eax
                                      addl
                                                (%eax), %eax
                                      movzbl
           (%edx, %eax), %eax
 movzbl
                                      leave
                                      ret
```

Exercise 5: Write a function that returns the Nth integer in an array

```
int getNthInt (int *array, int N)
{
   int c = array[N];
   return c;
}
```

```
getNthInt:

pushl %ebp

movl %esp, %ebp

movl 12(%ebp), %eax

leal 0(,%eax,4), %edx

movl 8(%ebp), %eax

addl %edx, %eax

movl (%eax), %eax

leave

ret
```

Exercise 5: Write a function that returns the Nth integer in an array

```
getNthInt:
int getNthInt (int *array, int N)
                                        pushl %ebp
                                        movl
                                              %esp, %ebp
   int c = array[N];
                                              12(%ebp), %eax
                                        movl
   return c;
                                               0(,%eax,4), %edx
                                        leal
                                               8(%ebp), %eax
                                        movl
     These can be replaced with a
                                               %edx, %eax
                                        addl
     single instruction. How?
                                               (%eax), %eax
                                        movl
                                        leave
                                        ret
```

Exercise 5: Write a function that returns the Nth integer in an array

```
getNthInt:

pushl %ebp

movl %esp, %ebp

movl 12(%ebp), %edx

movl 8(%ebp), %eax

movl (%eax, %edx,4), %eax

leave

ret
```

Exercise 6: Write a function that performs an element-wise addition of two integer arrays

```
void addABarrays (int *A, int *B, int *C, int size)
{
  int i = 0;
  for(i=0;i<size;i++)
   {
      C[i] = A[i]+B[i];
  }
}</pre>
```

```
addABarrays:
   pushl
          %ebp
   movl
          %esp, %ebp
   pushl
          %ebx
          $16, %esp
   subl
          $0, -8(%ebp)
   movl
   jmp.L2
.L3:
           -8(%ebp), %eax
   movl
           0(,%eax,4), %edx
   leal
           16(%ebp), %eax
   movl
   addl
           %edx, %eax
           -8(%ebp), %edx
   movl
           0(,%edx,4), %ecx
   leal
          8(%ebp), %edx
   mov
   addl
           %ecx, %edx
           (%edx), %ecx
   movl
           -8(%ebp), %edx
   movl
           0(,%edx,4), %ebx
   leal
           12(%ebp), %edx
   movl
   addl
           %ebx, %edx
           (%edx), %edx
   movl
   addl
           %ecx, %edx
           %edx, (%eax)
   movl
           $1, -8(%ebp)
   addl
```

```
L2:

movl -8(%ebp), %eax
cmpl 20(%ebp), %eax
jl .L3
addl $16, %esp
popl %ebx
popl %ebp
ret
```

Exercise 7:

Write a function that returns 1 if a character C exists in a string

```
int findCharacter (char *string, int size, char V)
  int i = 0;
  for(i=0;i<size;i++)
     if(string[i]==V)
        return 1;
   return 0;
```

```
findCharacter:
  pushl %ebp
  movl %esp, %ebp
  subl $20, %esp
  movl 16(%ebp), %eax
  movb %al, -20(%ebp)
  movl $0, -4(%ebp)
       .L2
  jmp
.L5:
  movl -4(%ebp), %edx
  movl 8(%ebp), %eax
  addl %edx, %eax
  movzbl (%eax), %eax
  cmpb -20(%ebp), %al
  ine .L3
  movl $1, %eax
       .L4
  jmp
```

```
.L3:
    addl $1, -4(%ebp)
.L2:
    movl -4(%ebp), %eax
    cmpl 12(%ebp), %eax
    jl .L5
    movl $0, %eax
.L4:
    leave
    ret
```

Exercise 8: Write a function that finds the max value in an integer array

```
void findMax (int *array, int size, int * maxValue)
   int i = 0;
   int max = array[i];
   for(i=1;i<size;i++)</pre>
       int v = array[i];
       if(v>max)
          max = v;
   *maxValue = max;
```

```
findMax:
          %ebp
   pushl
          %esp, %ebp
   mov
   subl
          $16, %esp
          $0, -12(%ebp)
   mov
          -12(%ebp), %eax
   movl
          0(,%eax,4), %edx
   leal
          8(%ebp), %eax
   movl
          %edx, %eax
   addl
          (%eax), %eax
   mov
          %eax, -8(%ebp)
   mov
          $1, -12(%ebp)
   movl
   jmp.L2
.L4:
          -12(%ebp), %eax
   movl
          0(,%eax,4), %edx
   leal
          8(%ebp), %eax
   mov
          %edx, %eax
   addl
          (%eax), %eax
   movl
          %eax, -4(%ebp)
   mov
          -8(%ebp), %eax
   cmpl
   ile .L3
   mov
          -4(%ebp), %eax
          %eax, -8(%ebp)
   movl
```

```
.L3:
          $1, -12(%ebp)
   addl
.L2:
           -12(%ebp), %eax
   movl
           12(%ebp), %eax
   cmpl
   il
     .L4
           16(%ebp), %eax
   movl
           -8(%ebp), %edx
   movl
           %edx, (%eax)
   movl
   leave
   ret
```

```
findMax:
   pushl
          %ebp
          %esp, %ebp
   mov
   subl
          $16, %esp
          $0, -12(%ebp)
   mov
          -12(%ebp), %eax
   movl
          0(,%eax,4), %edx
   leal
          8(%ebp), %eax
   mov
          %edx, %eax
   addl
          (%eax), %eax
   movl
          %eax, -8(%ebp)
   mov
          $1, -12(%ebp)
   movl
   jmp.L2
.L4:
          -12(%ebp), %eax
   movl
          0(,%eax,4), %edx
   leal
          8(%ebp), %eax
   movl
   addl
          %edx, %eax
          (%eax), %eax
   movl
          %eax, -4(%ebp)
   mov
          -8(%ebp), %eax
   cmpl
   jle .L3
   mov
          -4(%ebp), %eax
          %eax, -8(%ebp)
   movl
```

```
.L3:
          $1, -12(%ebp)
   addl
.L2:
   mov
          -12(%ebp), %eax
          12(%ebp), %eax
   cmpl
     .L4
          16(%ebp), %eax
   movl
          -8(%ebp), %edx
   mov
          %edx, (%eax)
   movl
   leave
   ret
```

```
movl -12(%ebp), %eax
movl 8(%ebp), %edx
movl 0(%edx,%eax,4), %eax
```

```
unsigned int LUT[4] = {0, 8, 16, 24};
unsigned int getByteByIndex (unsigned int word, int index)
{
   unsigned myByte = (word >> LUT[index]) & 0x000000ff;
   return myByte;
}
```

movl %eax, -4(%ebp)

movl

leave

ret

-4(%ebp), %eax

```
.text
                       getByteByIndex:
                          pushl %ebp
.data
                          movl %esp, %ebp
LUT:
                          subl $16, %esp
   .long
                          movl 12(%ebp), %eax
         8
   .long
                          movl LUT(,%eax,4), %eax
         16
   .long
                          movl 8(%ebp), %edx
   .long
         24
                          movl %eax, %ecx
                          shrl %cl, %edx
                          movl %edx, %eax
                                $255, %eax
                          andl
```

```
.text
                         getByteByIndex:
                             pushl %ebp
  .data
                             movl %esp, %ebp
  LUT:
                             subl $16, %esp
     .long
                             movl 12(%ebp), %eax
           8
     .long
                             movl LUT(,%eax,4), %eax
           16
     .long
                             movl 8(%ebp), %edx
     .long
           24
                             movl %eax, %ecx
                            shrl %cl, %edx
                             movl %edx, %eax
                                   $255, %eax
                             andl
unsigned myByte;
                             movl %eax, -4(%ebp)
                             movl -4(%ebp), %eax
                             leave
```

ret

```
.data
LUT:
.long 0
.long 8
.long 16
.long 24
```

What else can we write differently?

```
.text
getByteByIndex:
   pushl %ebp
   movl %esp, %ebp

movl 12(%ebp), %eax
   movl LUT(,%eax,4), %eax
   movl 8(%ebp), %edx
   movl %eax, %ecx
   shrl %cl, %edx
   movl %edx, %eax
   andl $255, %eax
```

leave ret

```
.text
                            getByteByIndex:
                               pushl %ebp
    .data
                               movl %esp, %ebp
    LUT:
       .long
                               movl 12(%ebp), %eax
             8
       .long
                               movl
                                     LUT(,%eax,4), %eax
             16
       .long
                                     8(%ebp), %edx
                               movl
       .long
             24
                               movl %eax, %ecx
                               shrl %cl, %edx
                               movl %edx, %eax
                               andl
                                     $255, %eax
movzbl %al, %eax -
                               leave
```

leave ret

```
.text
                            getByteByIndex:
                               pushl %ebp
    .data
                               movl %esp, %ebp
    LUT:
       .long
                                     12(%ebp), %eax
             8
                               movl
       .long
                               movl
                                     LUT(,%eax,4), %eax
             16
       .long
                                     8(%ebp), %edx
                               movl
       .long
             24
                               movl %eax, %ecx
                               shrl %cl, %edx
                               movl %edx, %eax
                               andl
                                     $255, %eax
movzbl %al, %eax -
                               leave
             %ebp
                               ret
```

Write a function that performs one of the following operations on two integers A and B based on the opcode:

0: add, 1: sub, 2: and, 3: or

```
int func (int A, int B, unsigned char opcode)
    int result = 0;
    switch(opcode)
         case 0:
         result = A+B;
         break;
         case 1:
         result = A-B;
         break;
         case 2:
         result = A \& B;
         break;
         case 3:
         result = A \mid B;
         break;
         default:
         result = 0;
         break;
    return result;
```

func	\• '•	LYCICI2C TO						
	pushl	%ebp						
	movl	%esp, %ebp						
	subl	\$20, %esp	(.L3:					
	movl	16(%ebp), %eax		movl	8(%ebp), %eax			
	movb	%al, -20(%ebp)		subl	12(%ebp), %eax			
	movl	\$0, -4(%ebp)		movl	%eax, -4(%ebp)			
	movzbl	-20(%ebp), %eax		jmp .L8				
	cmpl	\$1, %eax	(.L6:) .				
	je .L3	d 4 0/		/movl	8(%ebp), %eax			
	cmpl	\$1, %eax		andl	12(%ebp), %eax			
	jg .L4	CO 0/ 00 /		movl	%eax, -4(%ebp)			
	cmpl	\$0, %eax Labels with	1.7	jmp .L8				
.L4:	je .L5	local scope	(.L7:	movd	0/0/ ohp)			
	jmp .L2			/movl orl	8(%ebp), %eax 12(%ebp), %eax			
	<i>)</i> cmpl	\$2, %eax		movl	%eax, -4(%ebp)			
	je .L6	φ2, πουαχ		\jmp .L8	70cax, -4(70cbp)			
	cmpl	\$3, %eax	(.L2:	Jilip .Lo				
	je .L7	70, 700001		movl	\$0, -4(%ebp)			
	jmp .L2		(.L8:		ψο, ι(/σουρ)			
) .			movl	-4(%ebp), %eax			
	movl	8(%ebp), %edx		leave	(17,			
	movl	12(%ebp), %eax		ret				
	addl	%edx, %eax						
	movl	%eax, -4(%ebp)						
	jmp .L8							

func): :	EXELCISE TO					
	pushl	%ebp					
	movl	%esp, %ebp					
	subl	\$20, %esp	.L3:				
	movl	16(%ebp), %eax		movl	8(%ebp), %eax		
	movb	%al, -20(%ebp)		subl	12(%ebp), %eax		
	movl	\$0, -4(%ebp)		movl	%eax, -4(%ebp)		
	movzbl	-20(%ebp), %eax		jmp .L8			
	cmpl	\$1, %eax	.L6:				
	je .L3			movl	8(%ebp), %eax		
	cmpl	\$1, %eax		andl	12(%ebp), %eax		
	jg .L4			movl	%eax, -4(%ebp)		
	cmpl	\$0, %eax		jmp .L8			
	je .L5		.L7:				
	jmp .L2			movl	8(%ebp), %eax		
.L4:	_	ΦΟ 0/		orl .	12(%ebp), %eax		
	cmpl	\$2, %eax		movl	%eax, -4(%ebp)		
	je .L6	ΦΟ 0/		jmp .L8			
	cmpl	\$3, %eax	.L2:	•	^^		
	je .L7			movl	\$0, -4(%ebp)		
ı E.	jmp .L2		.L8:		4/0/ - > 0/		
.L5:	_	9(0(abp) 0(adv		movl	-4(%ebp), %eax		
	movl	8(%ebp), %edx		leave			
	movl	12(%ebp), %eax %edx, %eax		ret			
	addl	•					
	movl	%eax, -4(%ebp)					
	jmp .L8						

```
int DivByPow2 (int N, int shift)
   return N>>shift;
int AddABDiv2 (int A, int B)
   int C = A + B * 8 + 100;
   C = DivByPow2(C,2);
   return C;
```

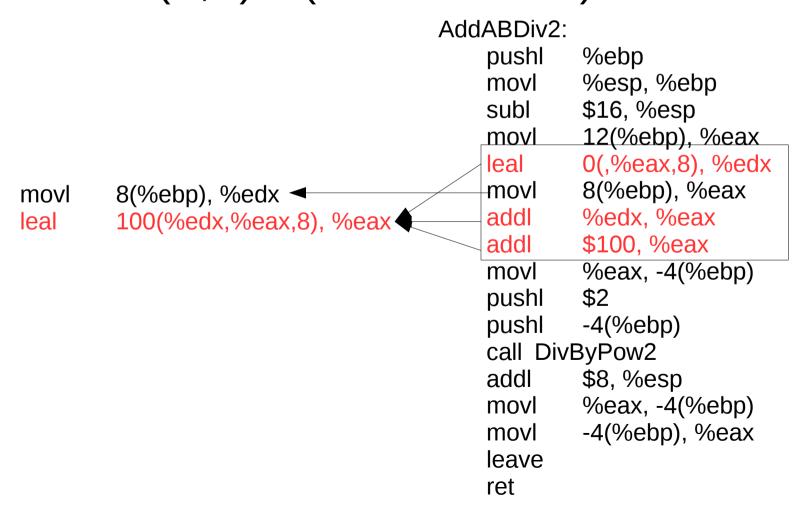
```
AddABDiv2:
                                               pushl
                                                       %ebp
                                               movl
                                                       %esp, %ebp
DivByPow2:
                                                       $16, %esp
                                               subl
    pushl
            %ebp
                                                       12(%ebp), %eax
                                               movl
    movl
            %esp, %ebp
                                                       0(,%eax,8), %edx
                                               leal
    movl
            12(%ebp), %eax
                                                       8(%ebp), %eax
                                               movl
    movl
            8(%ebp), %edx
                                               addl
                                                       %edx, %eax
            %eax, %ecx
    movl
                                                       $100, %eax
                                               addl
            %cl, %edx
    sarl
                                                       %eax, -4(%ebp)
                                               movl
            %edx. %eax
    movl
                                                       $2
                                               pushl
    popl
            %ebp
                                               pushl
                                                       -4(%ebp)
    ret
                                               call DivByPow2
                                               addl
                                                       $8, %esp
                                                       %eax, -4(%ebp)
                                               movl
                                                       -4(%ebp), %eax
                                               movl
                                               leave
                                               ret
```

```
DivByPow2:
   pushl
           %ebp
   movl
           %esp, %ebp
   movl
           12(%ebp), %eax
   movl
           8(%ebp), %edx
   movl
           %eax, %ecx
   sarl
           %cl. %edx
           %edx. %eax
   movl
   popl
           %ebp
   ret
   What can we write
   differently?
```

```
AddABDiv2:
    pushl
            %ebp
    movl
            %esp, %ebp
            $16, %esp
    subl
            12(%ebp), %eax
    movl
            0(,%eax,8), %edx
    leal
            8(%ebp), %eax
    movl
    addl
            %edx, %eax
            $100, %eax
    addl
            %eax, -4(%ebp)
    movl
            $2
    pushl
    pushl
            -4(%ebp)
    call DivByPow2
    addl
            $8, %esp
            %eax, -4(%ebp)
    movl
            -4(%ebp), %eax
    movl
    leave
    ret
```

```
DivByPow2:
   pushl
           %ebp
           %esp, %ebp
   movl
   movl
           12(%ebp), %eax
   movl
           8(%ebp), %edx
   movl
           %eax, %ecx
   sarl
           %cl. %edx
           %edx. %eax
   movl
   popl
           %ebp
   ret
   What can we write
   differently?
```

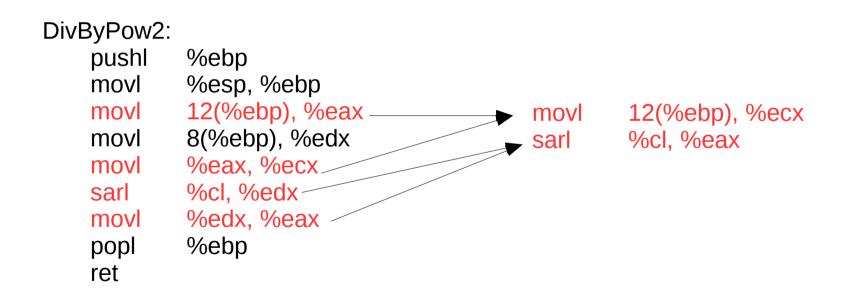
```
AddABDiv2:
    pushl
            %ebp
    movl
            %esp, %ebp
            $16, %esp
    subl
            12(%ebp), %eax
    movl
            0(,%eax,8), %edx
    leal
            8(%ebp), %eax
    movl
    addl
            %edx, %eax
    addl
            $100, %eax
            %eax, -4(%ebp)
    movl
            $2
    pushl
    pushl
            -4(%ebp)
    call DivByPow2
    addl
            $8, %esp
            %eax, -4(%ebp)
    movl
            -4(%ebp), %eax
    movl
    leave
    ret
```



Write a function that divides a number N with a number that is a power of two by receiving the required shift amount as the second argument. Then use it to write a function that takes two arguments A and B and returns f(A,B) = (A + 8*B + 100)/4.

```
DivByPow2:
    pushl
            %ebp
            %esp, %ebp
    movl
    movl
            12(%ebp), %eax
    movl
            8(%ebp), %edx
    movl
            %eax, %ecx
    sarl
            %cl. %edx
            %edx, %eax
    movl
    popl
            %ebp
    ret
```

What can we write differently?



Exercise 12: Write a function that performs a memory copy operation for N bytes.

```
myMemCpy:
   pushl
           %ebp
   movl
           %esp, %ebp
           $16, %esp
   subl
           $0, -4(%ebp)
   movl
   jmp .L2
.L3:
           -4(%ebp), %edx
   movl
           12(%ebp), %eax
   movl
   addl
           %eax. %edx
           -4(%ebp), %ecx
   movl
           8(%ebp), %eax
   movl
   addl
           %ecx, %eax
           (%eax), %eax
   movzbl
   movb
           %al, (%edx)
           $1, -4(%ebp)
   addl
.L2:
           -4(%ebp), %eax
   movl
            16(%ebp), %eax
   cmpl
       .L3
    leave
   ret
```

Write a function that performs a memory copy operation for N bytes. (least number of memory accesses)

```
myMemCpy:
    pushl
            %ebp
            %esp, %ebp
    movl
    subl
            $16, %esp
            $0, -4(%ebp)
    movl
   jmp .L2
.L3:
            -4(%ebp), %edx
    movl
            12(%ebp), %eax
    movl
    addl
            %eax. %edx
            -4(%ebp), %ecx
    movl
            8(%ebp), %eax
    movl
    addl
            %ecx, %eax
    movzbl
            (%eax), %eax
    movb
            %al, (%edx)
            $1, -4(%ebp)
    addl
.L2:
            -4(%ebp), %eax
    movl
            16(%ebp), %eax
    cmpl
        .L3
    leave
    ret
```

Write a function that performs a memory copy operation for N bytes. (least number of memory accesses)

```
myMemCpy:
myMemCpy:
                                          pushl
                                                  %ebp
    pushl
            %ebp
                                                  %esp, %ebp
            %esp, %ebp
                                          movl
    movl
                                                  8(%ebp), %eax
                                          movl
            $16, %esp
    subl
                                                  12(%ebp), %ebx
                                          movl
            $0, -4(%ebp)
    movl
                                                  16(%ebp), %ecx
    jmp .L2
                                          movl
                                          jmp .L2
.L3:
                                      .L3:
            -4(%ebp), %edx
    movl
                                          movzbl
                                                  (%eax), %edx
            12(%ebp), %eax
    movl
                                                  %dl, (%ebx)
                                          movb
    addl
            %eax, %edx
                                                  $1, %eax
                                          addl
            -4(%ebp), %ecx
    movl
                                          addl
                                                  $1, %ebx
            8(%ebp), %eax
    movl
                                          subl
                                                  $1, %ecx
    addl
            %ecx, %eax
                                      .L2:
    movzbl
            (%eax), %eax
                                                  $0. %ecx
                                          cmpl
            %al, (%edx)
    movb
                                          jne .L3
            $1, -4(%ebp)
    addl
                                          leave
.L2:
                                          ret
            -4(%ebp), %eax
    movl
    cmpl
            16(%ebp), %eax
        .L3
    leave
    ret
```

Exercise 13: Write a function that shifts the contents of an integer array one position to the left

```
void shiftInts(int * x, int size)
{
    int i;
    for(i=0;i<size-1;i++)
        x[i]=x[i+1];
}</pre>
```

```
shiftInts:
           %ebp
   pushl
   movl
           %esp, %ebp
           $16, %esp
   subl
           $0, -4(%ebp)
   movl
   jmp .L2
.L3:
           -4(%ebp), %eax
   movl
           0(,%eax,4), %edx
   leal
           8(%ebp), %eax
   movl
           %eax, %edx
   addl
           -4(%ebp), %eax
   movl
   addl
           $1, %eax
           0(,%eax,4), %ecx
   leal
           8(%ebp), %eax
   movl
           %ecx, %eax
   addl
           (%eax), %eax
   movl
           %eax, (%edx)
   movl
   addl
           $1, -4(%ebp)
L2:
           12(%ebp), %eax
   movl
   subl
           $1. %eax
           -4(%ebp), %eax
   cmpl
   jg .L3
   leave
   ret
```

Exercise 13: Write a function that shifts the contents of an integer array one position to the left (less instructions)

```
void shiftInts(int * x, int size)
{
    int i;
    for(i=0;i<size-1;i++)
        x[i]=x[i+1];
}</pre>
```

```
shiftInts:
          %ebp
   pushl
          %esp, %ebp
   movl
          12(%ebp), %eax
   movl
          $1, %eax
   subl
          $0, %ecx
   movl
          8(%ebp), %ebx
   movl
   jmp.L2
.L3:
          4(%ebx), %edx
   movl
          %edx, (%ebx)
   movl
          $4, %ebx
   addl
   addl
          $1, %ecx
.L2:
          %ecx, %eax
   cmpl
   ine .L3
   leave
   ret
```

Assume a function popcnt8() that returns the number of set bits in a byte. Write a function popcntX() that uses popcnt8() to return the total number of set bits in Y bytes.

```
int popcntX (unsigned char * x, int size)
{
    int i=size, j=0, cnt=0;
    while(i!=0)
    {
       cnt += popcnt8(x[j]);
       j++;
       i--;
    }
    return cnt;
}
```

```
popcntX:
   pushl
           %ebp
   movl
           %esp, %ebp
   subl
           $16, %esp
           12(%ebp), %eax
   movl
           %eax, -4(%ebp)
   movl
           $0, -8(%ebp)
   movl
           $0, -12(%ebp)
   movl
   jmp .L5
.L6:
   movl
           -8(%ebp), %edx
           8(%ebp), %eax
   movl
           %edx, %eax
   addl
   movzbl (%eax), %eax
           %eax, -16(%ebp)
   movl
   call popent8
           %eax, -12(%ebp)
   addl
           $1, -8(%ebp)
   addl
           $1, -4(%ebp)
   subl
.L5:
           $0, -4(%ebp)
   cmpl
   jne .L6
           -12(%ebp), %eax
   movl
   leave
   ret
```