Exercise 1:
Write a function that adds two integers

```
int add2ints(int A, int B)
{
    int C = A+B;
    return C;
}
```

```
add2ints:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    addl    %edx, %eax
    movl    %ebp, %esp
    popl    %ebp
    ret
```

Exercise 2:
Write a function that compares two integers and returns 0/1 for mismatch/match

```
int compareAB (int A, int B)
{
    int match = 0;

    if(A==B)
        match = 1;

    return match;
}
```

```
compareAB:
    pushl %ebp
    movl  %esp, %ebp
    subl  $16, %esp
    movl  $0, -4(%ebp)
    movl  8(%ebp), %eax
    cmpl  12(%ebp), %eax
    jne .L2
    movl  $1, -4(%ebp)
.L2:
    movl  -4(%ebp), %eax
    leave
    ret
```

Exercise 2:
Write a function that compares two integers and returns 0/1 for mismatch/match

```
int compareAB (int A, int B)
{
    int match = 0;

    if(A==B)
        match = 1;

    return match;
}
```

```
compareAB:
    pushl %ebp
    movl  %esp, %ebp
    subl  $16, %esp
    movl  $0, -4(%ebp)
    movl  8(%ebp), %eax
    cmpl  12(%ebp), %eax
    jne .L2
    movl  $1, -4(%ebp)
.L2:
    movl  -4(%ebp), %eax
    leave
    ret
```

```
movl    %ebp, %esp
popl    %ebp
```

Exercise 3:
Write a function that compares two <u>characters</u> and returns 0/1 for mismatch/match

```
int compareAB (char A, char B)
{
    int match = 0;

    if(A==B)
        match = 1;

    return match;
}
```

```
compareAB:
    pushl %ebp
    movl  %esp, %ebp
    subl  $16, %esp
    movl  8(%ebp), %edx
    movl  12(%ebp), %eax
    movl  $0, -4(%ebp)
    cmpb  %dl, %al
    jne   .L2
    movl  $1, -4(%ebp)
.L2:
    movl  -4(%ebp), %eax
    leave
    ret
```

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

```
char getNthChar (char *str, int N)
{
    char c = str[N];
    return c;
}
```

```
getNthChar:
    pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %edx
    movl    8(%ebp), %eax
    addl    %edx, %eax
    movzbl    (%eax), %eax
    leave
    ret
```

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

```
char getNthChar (char *str, int N)
{
    char c = str[N];
    return c;
}
```

```
movzbl    (%edx, %eax), %eax
```

```
getNthChar:
    pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %edx
    movl    8(%ebp), %eax
    addl    %edx, %eax
    movzbl    (%eax), %eax
    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

```
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
```

These can be replaced with a single instruction. How?

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), %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];
    }
}
```

Exercise 6

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

```
.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;
}
```

Exercise 7

```
findCharacter:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $20, %esp
    movl    16(%ebp), %eax
    movb    %al, -20(%ebp)
    movl    $0, -4(%ebp)
    jmp     .L2
.L5:
    movl    -4(%ebp), %edx
    movl    8(%ebp), %eax
    addl    %edx, %eax
    movzbl     (%eax), %eax
    cmpb    -20(%ebp), %al
    jne .L3
    movl    $1, %eax
    jmp     .L4
.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++)
    {
        int v = array[i];
        if(v>max)
            max = v;
    }
    *maxValue = max;
}
```

Exercise 8

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

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

Exercise 8

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

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

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

Exercise 9:
Write a function that returns the Nth byte in a word (N=0,1,2,3)

```
unsigned int LUT[4] = {0, 8, 16, 24};

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

Exercise 9:
Write a function that returns the Nth byte in a word (N=0,1,2,3)

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

```
.text
getByteByIndex:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        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
        movl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```

Exercise 9:
Write a function that returns the Nth byte in a word (N=0,1,2,3)

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

unsigned myByte;

```
.text
getByteByIndex:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        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
        movl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```

Exercise 9:
Write a function that returns the Nth byte in a word (N=0,1,2,3)

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

```
.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
```

What else can we write differently?

Exercise 9:
Write a function that returns the Nth byte in a word (N=0,1,2,3)

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

```
.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
```

```
movzbl    %al, %eax
```

Exercise 9:
Write a function that returns the Nth byte in a word (N=0,1,2,3)

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

```
.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
```

movzbl %al, %eax

popl %ebp

Exercise 10:

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

Exercise 10

```
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 | B;
		break;

		default:
		result = 0;
		break;
	}
	return result;
}
```

Exercise 10

```
func:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $20, %esp
        movl    16(%ebp), %eax
        movb    %al, -20(%ebp)
        movl    $0, -4(%ebp)
        movzbl  -20(%ebp), %eax
        cmpl    $1, %eax
        je      .L3
        cmpl    $1, %eax
        jg      .L4
        cmpl    $0, %eax
        je      .L5
        jmp     .L2
.L4:
        cmpl    $2, %eax
        je      .L6
        cmpl    $3, %eax
        je      .L7
        jmp     .L2
.L5:
        movl    8(%ebp), %edx
        movl    12(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        jmp     .L8
```

Labels with local scope

```
.L3:
        movl    8(%ebp), %eax
        subl    12(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp     .L8
.L6:
        movl    8(%ebp), %eax
        andl    12(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp     .L8
.L7:
        movl    8(%ebp), %eax
        orl     12(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp     .L8
.L2:
        movl    $0, -4(%ebp)
.L8:
        movl    -4(%ebp), %eax
        leave
        ret
```

Exercise 10

```
func:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $20, %esp
        movl    16(%ebp), %eax
        movb    %al, -20(%ebp)
        movl    $0, -4(%ebp)
        movzbl  -20(%ebp), %eax
        cmpl    $1, %eax
        je  .L3
        cmpl    $1, %eax
        jg  .L4
        cmpl    $0, %eax
        je  .L5
        jmp .L2
.L4:
        cmpl    $2, %eax
        je  .L6
        cmpl    $3, %eax
        je  .L7
        jmp .L2
.L5:
        movl    8(%ebp), %edx
        movl    12(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        jmp .L8
.L3:
        movl    8(%ebp), %eax
        subl    12(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp .L8
.L6:
        movl    8(%ebp), %eax
        andl    12(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp .L8
.L7:
        movl    8(%ebp), %eax
        orl     12(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp .L8
.L2:
        movl    $0, -4(%ebp)
.L8:
        movl    -4(%ebp), %eax
        leave
        ret
```

Exercise 11:

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.

Exercise 11:

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.

```
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;
}
```

Exercise 11:

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

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

Exercise 11:

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

What can we write differently?

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

Exercise 11:

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

What can we write differently?

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

Exercise 11:

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.

```
movl    8(%ebp), %edx
leal    100(%edx,%eax,8), %eax
```

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

Exercise 11:

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

What can we write differently?

Exercise 11:

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

```
movl    12(%ebp), %ecx
sarl    %cl, %eax
```

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

```
void myMemCpy(unsigned char * src,
              unsigned char * dst,
              int size)
{
    int i;
    for(i=0;i<size;i++)
        dst[i] = src[i];
}
```

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

Exercise 12:
Write a function that performs a memory copy operation for N bytes. (least number of memory accesses)

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

Exercise 12:
Write a function that performs a memory copy operation for N bytes. (least number of memory accesses)

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

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

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

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

Exercise 14:

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
    movl    12(%ebp), %eax
    movl    %eax, -4(%ebp)
    movl    $0, -8(%ebp)
    movl    $0, -12(%ebp)
    jmp .L5
.L6:
    movl    -8(%ebp), %edx
    movl    8(%ebp), %eax
    addl    %edx, %eax
    movzbl  (%eax), %eax
    movl    %eax, -16(%ebp)
    call popcnt8
    addl    %eax, -12(%ebp)
    addl    $1, -8(%ebp)
    subl    $1, -4(%ebp)
.L5:
    cmpl    $0, -4(%ebp)
    jne  .L6
    movl    -12(%ebp), %eax
    leave
    ret
```