Array Indexing

Take this C code as example:

```
// this line is defined globally
int globalArr[5];
// these lines are part of some function
int b;
int localArr[5];
globalArr[2] = 12;
localArr[0] = 3;
localArr[4] = 18;
b = localArr[4];
```

Generated assembly code:

```
movl    $12, globalArr+8
movl    $3, -24(%ebp)
movl    $18, -8(%ebp)
movl    -8(%ebp), %eax
movl    %eax, -4(%ebp)
```

Location of local variables of the stack (local variables are explained here (memorymanagement.html))

```
b => -4(%ebp)
localArr => -24(%ebp) to -4(%ebp)
        [excluding the memory location pointed by -4(%ebp) ]
        The localArr of length 5; each element in the array, is of 4 bytes.
        So total space by local var = 20 bytes.
```

The use of registers as temporary memory is described here (arithmeticop.html#tempVaribaleUsage)

Comments on generated assembly code:

```
# globalArr[2] = 12;
# globarArr + 8 means memory starting at the 8th byte of global var.
# because each element is of 4 bytes so the 3rd element will start at
# byte offset 8
movl $12, globalArr+8

# localArr[0] = 3. The localArr start at -24(%ebp) so the localArr[0]
# will be at byte offset 0
   movl    $3, -24(%ebp)

# localArr[4] = 18. localArr[4] will at byte offset 16. (-24 + 16 = -8)
   movl    $18, -8(%ebp)

# tmp = localArr[4]
   movl    -8(%ebp), %eax

# b = tmp
   movl     %eax, -4(%ebp)
```

Array indexing using pointers

Take this C code as example:

```
// This line is defined globally.
int globalArr[5];
// these lines are part of some function
int b;
int *ptr = globalArr;
b = ptr[4];
```

Generated assembly code:

```
movl    $globalArr, -8(%ebp)
movl    -8(%ebp), %eax
addl    $16, %eax
movl    (%eax), %eax
movl    %eax, -4(%ebp)
```

Location of local variables of the stack (local variables are explained here (memorymanagement.html))

```
b => -4(%ebp)
ptr => -8(%ebp)
```

The use of registers as temporary memory is described here (arithmeticop.html#tempVaribaleUsage)
Comments on generated assembly code:

```
# ptr = globalArr
movl    $globalArr, -8(%ebp)

# tmp = ptr
movl    -8(%ebp), %eax

# tmp = tmp + 16. The tmp contains a pointer. so the next
# instruction is to move the pointer by 16 bytes.
addl    $16, %eax

# tmp = *tmp
movl    (%eax), %eax

# b = tmp;
movl    %eax, -4(%ebp)
```

‹ Pointer Dereferencing (/cin/pointer.html)
up (/cin/cin.html)
Function Pointer › (/cin/functionpointer.html)