Tutorial #1
Hilary Term Weeks 2 and 3
Addressing Modes and Arrays

Work in groups of 5 to 7 students.
Complete the exercises in your own time if necessary.

1 Addressing Modes

(a) For each of the instructions below, predict the precise memory operation that will be performed. Your prediction should include the memory address that is accessed and the new value contained in the base address register, R1, if it has been modified.

Verify your solutions using µVision.

```
1        LDR     R0, [R1, #8]
2        LDRB    R0, [R1, #1]!
3        LDR     R0, [R1], #4
4        STR     R0, [R1, R2, LSL #2]
```

Assume the following initial values in R1 and R2:

`R1=0x40000080, R2=0x00000042`

(b) Write four versions of an ARM Assembly Langauge program to replace each value in an array of 10 signed, halfword-sized values in memory with the square of the value. For example, 5 should be replaced with 25.

Each version of your program should be restricted to using the following addressing modes:

(i) Immediate Offset (e.g. `[Rn]` or `[Rn, #offset]`)

(ii) Immediate Offset and Register Offset (e.g. `[Rn, Rm]`)

(iii) Immediate Offset and Scaled Register Offset (e.g. `[Rn, Rm, LSL #shift]`)

(iv) Immediate Offset and Pre- or Post-indexed (e.g. `[Rn], #offset` or `[Rn,#offset]!`)

(c) For each version of your program above, calculate the number of instructions executed during the execution of the program.

2 Arrays

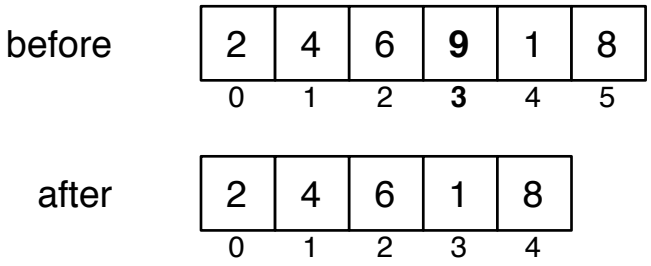
(a) Translate each of the pseudo-code statements below into ARM Assembly Language. Assume that variables f, i and j correspond to registers R4, R5 and R6 respectively. A is a one-dimensional array of 16-bit unsigned values with a starting address contained in R10. B is a 16×16 (two-dimensional) array of 32-bit unsigned values with a starting address contained in R11. (Assume that B is stored in row-major order.)

(i) f = A[i]

(ii) A[j] = A[j+2]

(iii) f = B[i][j] * B[j][i]

(b) Write an ARM Assembly Language program that will remove an array element from a specified index in an array of word-size values. The figure below illustrates an array in which an element is removed from index 3. **When removing the element from the array, your program should move the subsequent elements towards the start of the array to fill the "gap" created in memory.** Assume that the start address of the array is in R0, the index of the element to remove is in R1 and the number of elements in the array is in R2.



Note: a similar problem will be addressed in the forthcoming lab exercise.