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

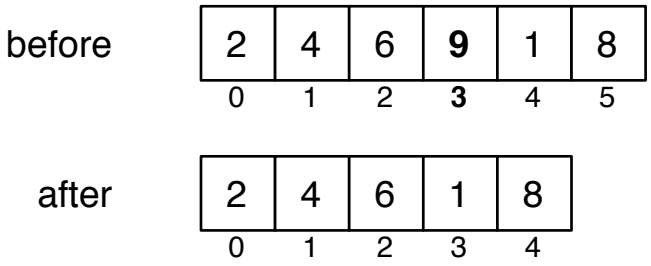
(a) Translate each of the pseudo-code statements below into ARM Assembly Language. Assume that variables f, i and j correspond to registers R4, R5 and R6 respectively. A is a one-dimensional array of 16-bit unsigned values with a starting address contained in R10. B is a 16×16 (two-dimensional) array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

CS1022 Tutorial #1 SOLUTION
Hilary Term Weeks 2 and 3
Addressing Modes and Arrays

1 Addressing Modes

(a) (i) Load a word from memory at address 0x41000080 + 8 = 0x41000088 into register R0.

(ii) Load a word from memory at address 0x41000080. R1 is updated with 0x41000080 + 4 = 0x41000084.

(iii) Store a word from R0 to memory at 0x41000080 + (0x42 <<2) = 0x41000188

(iv) Load a byte from memory at address 0x41000080 + 1 = 0x41000081 into register R0. R1 is updated with 0x41000081.

The above assumes the effects of each instruction execution are not cumulative – each instruction is executed independently. Solutions may alternatively assume that the effects are cumulative, leading to different results.

(b) Instruction counts based on 10 halfwords. Use LDRH and STRH to load/store halfword values. Add 2 to the address of a halfword to obtain the address of the next halfword in memory. Similarly, when using Scaled Register Offset addressing, we need to multiply the index by 2 (LSL by 1 bit) to obtain the address of a halfword element (contrast with 4 (LSL by 2 bits) for an array of word-size values.)

(i) Immediate Offset (83 instructions)

```
1          MOV     R4, #0
2  L3      CMP     R4, R1
3          BHS     L4
4          LDRSH   R5, [R0]
5          MUL     R6, R5, R5
6          STRH    R6, [R0]
7          ADD     R0, R0, #2
8          ADD     R4, R4, #1
9          B       L3
10 L4
```

(ii) Immediate Offset and Register Offset (84 instructions)

```
1          MOV     R4, #0
2          MOV     R7, #0
3  L5      CMP     R4, R1
4          BHS     L6
5          LDRSH   R5, [R0, R7]
6          MUL     R6, R5, R5
7          STRH    R6, [R0, R7]
8          ADD     R7, R7, #2
```

```
9           ADD     R4, R4, #1
10          B       L5
11  L6
```

(iii) Immediate Offset and Scaled Register Offset (73 instructions)

```
1           MOV     R4, #0
2   L7      CMP     R4, R1
3           BHS     L8
4           LDRSH   R5, [R0, R4, LSL #1]
5           MUL     R6, R5, R5
6           STRH    R6, [R0, R4, LSL #1]
7           ADD     R4, R4, #1
8           B       L7
9   L8
```

(iv) Immediate Offset and Pre- or Post-indexed (73 instructions)

```
1           MOV     R4, #0
2   L9      CMP     R4, R1
3           BHS     L10
4           LDRSH   R5, [R0]
5           MUL     R6, R5, R5
6           STRH    R6, [R0], #2
7           ADD     R4, R4, #1
8           B       L9
9   L10
```

2 Arrays

(a) (i)

```
1           LDRH    R4, [R10, R5, LSL #1]
```

(ii)

```
1           ADD     R8, R6, #2
2           LDRH    R7, [R10, R8, LSL #1]
3           STRH    R7, [R10, R6, LSL #1]
```

(iii)

```
1           MOV     R7, R5, LSL #4          ; idx = i * 16
2           ADD     R7, R7, R6              ; idx += j
3           LDR     R8, [R11, R7, LSL #2]
4
5           MOV     R7, R6, LSL #4          ; idx = j * 16
6           ADD     R7, R7, R5              ; idx += i
7           LDR     R9, [R11, R7, LSL #2]
8
9           MUL     R4, R8, R9
```

(b) Remove

```
curIdx = N - 1;
remembered = array[curIdx];
while (--curIdx >= remIdx) {
    tmp = array[curIdx];
    array[curIdx] = remembered;
    remembered = tmp;
}
```

```
1          SUB     R4, R3, #1              ; curIdx = N - 1
2          LDR     R5, [R0, R4, LSL #2]    ; remembered = Memory.Byte[curIdx]
3  whDn
4          SUB     R4, R4, #1              ; while (--curIdx
5          CMP     R4, R1                  ;     >= remIdx) {
6          BLO     eWhDn                   ;
7          LDR     R6, [R0, R4, LSL #2]    ;   tmp = array[tmpidx]
8          STR     R5, [R0, R4, LSL #2]    ;   array[idx] = remembered
9          MOV     R5, R6                  ;   remembered = tmp
10         B       whDn                    ; }
11 eWhDn
```

(c) Matrix Multiplication

```
1          MOV     r4, #0                  ; i = 0;
2  L1
3          CMP     r4, r3                  ; while (i < N)
4          BHS     L6                      ; {
5          MOV     r5, #0                  ;  j = 0;
6  L2
7          CMP     r5, r3                  ;  while (j < N)
8          BHS     L5                      ;  {
9          MOV     r7, #0                  ;   r = 0;
10         MOV     r6, #0                  ;   k = 0;
11 L3
12         CMP     r6, r3                  ;   while (k < N)
13         BHS     L4                      ;   {
14         MUL     r8, r4, r3              ;    idx = (i * N)
15         ADD     r8, r8, r6              ;            + k;
16         LDR     r9, [r1, r8, LSL #2]    ;    tmpA = Memory.Byte[pA + (idx * 4)];
17         MUL     r8, r6, r3              ;    idx = (k * N)
18         ADD     r8, r8, r5              ;            + j;
19         LDR     r10, [r1, r8, LSL #2]   ;    tmpB = Memory.Byte[pB + (idx * 4)];
20         MUL     r9, r10, r9             ;    tmpA = tmpA * tmpB;
21         ADD     r7, r7, r9              ;    r = r + tmpA;
22         ADD     r6, r6, #1              ;    k = k + 1;
23         B       L3                      ;   }
24 L4
25         MUL     r8, r4, r3              ;   idx = (i * N)
26         ADD     r8, r8, r5              ;           + j;
27         STR     r7, [r0, r8, LSL #2]    ;   Memory.Byte[pZ + (idx * 4)];
28         ADD     r5, r5, #1              ;   j = j + 1;
29         B       L2                      ;  }
30 L5
31         ADD     r4, r4, #1              ;  i = i + 1;
32         B       L1                      ; }
33 L6
```