CT1 Exercises Data Transfer

1. What is a load/store architecture?

   Data processing is between registers. Transfer of data from and to the external memory is done using load (memory to register) or store (register to memory) instructions.

2. What is the difference between a MOV and a MOVS instruction?

   MOV Transfer does NOT affect flags (low and high registers)
   MOVS Transfer affects flags (only low registers)

3. Which data transfer instructions should you use if at least one high register is an operand?

   MOV works for all registers (but only for reg/reg transfers)

4. List different ways of initializing a low register with an immediate value. What are the advantages/disadvantages?

   MOVS <Rd>,#<imm8> (value to load is in instruction)
   Less memory space but limited to 8-bit values.

   LDR <Rt>,[PC,#<imm>] (use PC/offset combination to point to the value to load).
   Can be used to load larger values (up to 32-bit). Takes more space in memory.

5. What is a pseudo-Instruction? Explain what is done with a <LDR Rn, = literal> pseudo-instruction.

   A pseudo-Instruction does not directly translate in machine code. It is an instruction that is interpreted (or expanded)by the assembler (the tool) to generate the needed machine code instruction(s).

   When we run the assembler (the tool), the <LDR Rn, = literal> instruction is decomposed into

   - **a place reservation of the type DCD and an indirect load of the type LDR <Rt>,[PC,#<imm>]to get the contents of the reserved position and write it in Rn**

     Literal can be an address (a position in the memory)
     Literal can be a constant (the value is known at assembling time)

6. Consider the following assembled listing.

   a) For all the LDR instructions, calculate the value of imm and write the instructions in the form LDR <Rt>,[PC,#<imm>]

   b) Replace yyyy with the appropriate machine code

   c) Do you understand what the <B loop> instruction does? If not, look this up in the assembly document on OLAT.

   d) What do you think will happen during the program execution if there was no <B loop> instruction after ADD R0,R1?

```
                CONST_VALUE_X   EQU         123456
00000000
00000000  yyyy  loop    LDR         R0, =0xFF55AAB0
00000002  yyyy          LDR         R1, =CONST_VALUE_X
00000004  yyyy          LDR         R2, myval2
00000006  yyyy          LDR         R3, myval3
00000008  yyyy          LDR         R4, myval4
0000000A  yyyy          LDR         R5, myval5
0000000C  yyyy          LDR         R6, myval6
0000000E  yyyy          LDR         R7, myval7
00000010  4408          ADD         R0, R1
00000012  E7F5          B           loop
00000014
00000014  33445566
                myval3  DCD         0x33445566
00000018  0000FFAA
                myval4  DCD         0x00FFAA
0000001C  00110044
                myval5  DCD         0x110044
00000020  00AABBCC
                myval7  DCD         0xAABBCC
00000024  7788ABCD
                myval6  DCD         0x7788ABCD
00000028  33445555
                myval2  DCD         0x33445555
0000002C
0000002C
0000002C
0000002C
                FF55AAB0
                0001E240

                480A        LDR         R0,[PC,#40]  ;
                490B        LDR         R1,[PC,#44]  ;
                4A08        LDR         R2,[PC,#32]  ;
                4B03        LDR         R3,[PC,#12]  ;
                4C03        LDR         R4,[PC,#12]  ;
                4D04        LDR         R5,[PC,#16]  ;
                4E05        LDR         R6,[PC,#20]  ;
                4F04        LDR         R7,[PC,#16]  ;
                <B loop> branches unconditionally to loop. Without this,
                the CPU will continue to fetch and execute and take the
                following contents of memory as code and try to interpret
                and execute it. Dangerous!
```

7. Consider the following listing.
   a) Compute on the basis of the opcode (marked in blue) the position where data is read to load registers
   b) The opcodes for <LDR R0, lit1> and <LDR R0, =lit1> are the same. But where are the operands? Explain the differences between the 2.
   c) Suppose that the first operation (0x00000000 in the listing) is effectively at address 0x08000008 after linking and loading. Compute the contents of the registers after each instruction is executed once.

```
00000000  4804  again  LDR   R0, lit1
00000002  480A         LDR   R0, =lit1
00000004  4A04         LDR   R2, lit2
00000006  4A0A         LDR   R2, =lit2
00000008  4B04         LDR   R3, lit3
0000000A  4C04         LDR   R4, lit3
0000000C  4E09         LDR   R6, =lit4
0000000E  4F09         LDR   R7, =lit4
00000010  4408         ADD   R0, R1
00000012  E7F5         B     again
00000014
00000014  00000001
          xxx  DCD    0x01
00000018  00000002
          xxx  DCD    0x02
0000001C  00000003
          xxx  DCD    0x03
00000020  00000004
          xxx  DCD    0x04
00000024  00000005
          xxx  DCD    0x05
00000028  00000006
          xxx  DCD    0x06
0000002C
0000002C
0000002C
0000002C
          00000000
          00000000
          00000000
          00000000
```

The code in the processor's memory

```
0x08000008 4804      LDR      r0,[pc,#16]  ; @0x0800001C
0x0800000A 480A      LDR      r0,[pc,#40]  ; @0x08000034
0x0800000C 4A04      LDR      r2,[pc,#16]  ; @0x08000020
0x0800000E 4A0A      LDR      r2,[pc,#40]  ; @0x08000038
0x08000010 4B04      LDR      r3,[pc,#16]  ; @0x08000024
0x08000012 4C04      LDR      r4,[pc,#16]  ; @0x08000024
0x08000014 4E09      LDR      r6,[pc,#36]  ; @0x0800003C
0x08000016 4F09      LDR      r7,[pc,#36]  ; @0x0800003C
0x08000018 4408      ADD      r0,r0,r1
0x0800001A E7F5      B        Again (0x08000008)
0x0800001C 0001      DCW      0x0001       ; This is lit1
0x0800001E 0000      DCW      0x0000
0x08000020 0002      DCW      0x0002       ; This is lit2
0x08000022 0000      DCW      0x0000
0x08000024 0003      DCW      0x0003       ; This is lit3
0x08000026 0000      DCW      0x0000
0x08000028 0004      DCW      0x0004
0x0800002A 0000      DCW      0x0000
0x0800002C 0005      DCW      0x0005
0x0800002E 0000      DCW      0x0000
0x08000030 0006      DCW      0x0006
0x08000032 0000      DCW      0x0000
0x08000034 001C      DCW      0x001C       ; Address of lit1 stored
0x08000036 0800      DCW      0x0800
0x08000038 0020      DCW      0x0020       ; Address of lit2 stored
0x0800003A 0800      DCW      0x0800
0x0800003C 0028      DCW      0x0028
0x0800003E 0800      DCW      0x0800
0x08000040 0000      DCW      0x0000
0x08000042 0000      DCW      0x0000
```

<LDR R0, lit1>
The value at position label lit1 is loaded in R0

<LDR R0, =lit1>
Of the form LDR r0, =label
(Room is made in the literal pool to store the address of lit1. That address is then loaded in R0 when code is executed)

<LDR R0, =const> (e.g. in previous exercise LDR   R0, =0xFF55AAB0)
Of the form LDR r0, = constant
(The constant is placed in a literal pool and loaded in R0)

8. Whenever possible, work out the contents of registers or memory positions that have changed.

again	LDR	R0,=0xFF	R0 = 0x000000FF
	LDR	R1, lit1	R1 = 0x0000EFAA
	LDR	R2, lit2	R2 = 0x00012345
	LDR	R3,=0x55AA	R3 = 0x000055AA
	MOV	R4, R1	R4 = 0x0000EFAA
	MOV	R4, R2	R4 = 0x00012345
	MOV	R6, R3	R6 = 0x000055AA
	MOVS	R7, #04	R7 = 0x00000004
	LDR	R0,=lit1	R0 = address of lit1
	LDR	R1,=lit2	R1 = address of lit2
	LDR	R2,=lit3	R2 = address of lit3
	LDRB	R5,[R2]	R5 = 0x00000097
	LDRH	R6,[R2,#2]	R6 = 0x0000008F
	LDR	R2,[R0]	R2 = 0x0000EFAA
	LDR	R3,[R0,#4]	R3 = 0x00012345
	STR	R3,[R1]	lit2 = 0x00012345
	STR	R2,[R1,#8]	lit4 = 0x0000EFAA
	STR	R4,[R1,R7]	lit3 = 0x00012345
	LDR	R5,=lit5	R5 = address of lit5
	MOVS	R0,#0	R0 = 0x00000000
	ADDS	R7,R0,#1	R7 = 0x00000001
	LDRSB	R6,[R5,R0]	R6 = 0xFFFFFF89
	LDRSH	R6,[R5,R7]	problem. Unaligned memory access
	B again		
lit1	DCD 0xEFAA		
lit2	DCD 0x12345		
lit3	DCD 0x8F1097		
lit4	DCD 0xFF76552F		
lit5	DCD 0xAA654389		
lit6	DCD 0x0165		
Var1	DCD 0x23		
Var2	DCD 0x24		
Var3	DCD 0x23455678		
Var4	DCD 0xE4568900		

9. Write down the assembly instructions to perform the following actions.
   a) Copy contents of R1 in R3 (flags unchanged)

```
MOV R3,R1
```

   b) Initialize R0 with 0xAA (flags unchanged)

```
LDR R0,=0xAA (000000AA will be placed in literal pool and
loaded in R0 when the instruction executes)
You cannot use a MOVS here, because it modifies flags.
To load an immediate value in the register of a cortex M0
without modifying the flags, you can use a LDR instruction.
```

   c) Initialize R1 with 234 (flags modified)

```
MOVS R1,#234
```

   d) Initialize R4 with 0x55AACC

```
LDR R4,=0x55AACC
```

   e) Copy contents of R9 in R3.

```
MOV R3,R9
```

   f) Initialize R10 with 0x345678

```
LDR R0,= 0x345678 ;R0 as example. Another low register is ok
MOV R10,R0
```

   g) Copy contents of R8 in R9

```
MOV R9,R8
```

10. Code the following C programs in assembly

a)

Code this in assembly

```
// C-Code
int x;
int y;
int *xp;

void main(void) {

    x = 3;
    xp = &x;
    y = *xp;

}
```

```
See following pages
```

b)

Code this in assembly

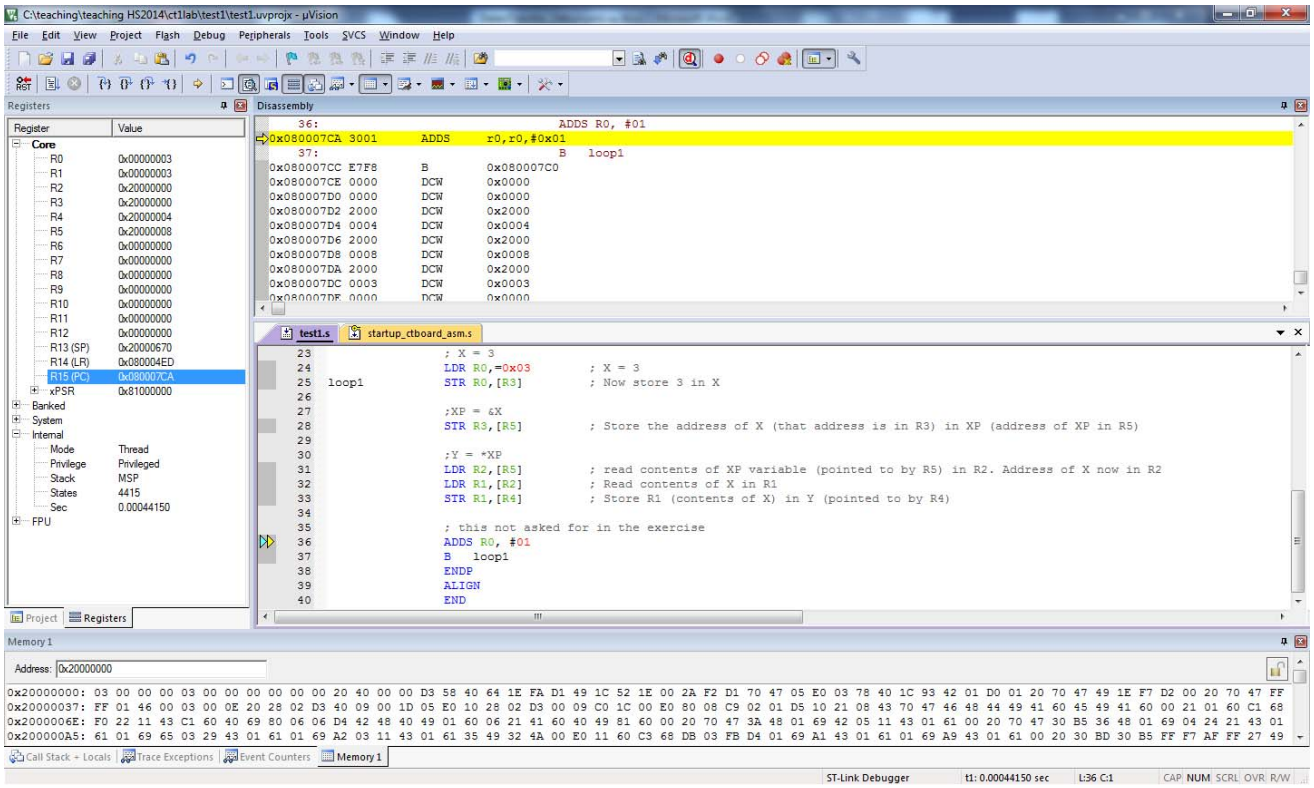
```
// C-Code
char demoArray[2];
char *xp;

void main(void) {
    demoArray[0] = 10;
    demoArray[1] = 11;
    xp = demoArray;
    *xp = 111;
    xp++;
    *xp = 112;
}
```

```
Possible solution, as given by the C compiler.
This solution does not use pseudo instructions.
Before looking at this, make sure that you understand
the solution of 10a

    22:     demoArray[0] = 10;
0x08000254 200A      MOVS     r0,#0x0A
0x08000256 4909      LDR      r1,[pc,#36]  ; @0x0800027C
0x08000258 7008      STRB     r0,[r1,#0x00]
    23:     demoArray[1] = 11;
0x0800025A 200B      MOVS     r0,#0x0B
0x0800025C 7048      STRB     r0,[r1,#0x01]
    24:     xp = demoArray;
0x0800025E 4608      MOV      r0,r1
0x08000260 4907      LDR      r1,[pc,#28]  ; @0x08000280
0x08000262 6008      STR      r0,[r1,#0x00]
    25:     *xp = 111;
0x08000264 206F      MOVS     r0,#0x6F
0x08000266 6809      LDR      r1,[r1,#0x00]
0x08000268 7008      STRB     r0,[r1,#0x00]
    26:     xp++;
0x0800026A 4805      LDR      r0,[pc,#20]  ; @0x08000280
0x0800026C 6800      LDR      r0,[r0,#0x00]
0x0800026E 1C40      ADDS     r0,r0,#1
0x08000270 4903      LDR      r1,[pc,#12]  ; @0x08000280
0x08000272 6008      STR      r0,[r1,#0x00]
    27:     *xp = 112;
0x08000274 2070      MOVS     r0,#0x70
0x08000276 6809      LDR      r1,[r1,#0x00]
0x08000278 7008      STRB     r0,[r1,#0x00]
    28: }
0x0800027A 4770      BX       lr ;Not part of solution
0x0800027C 0000      DCW      0x0000
0x0800027E 2000      DCW      0x2000
0x08000280 0004      DCW      0x0004
0x08000282 2000      DCW      0x2000
```

Possible solution for exercise 10a

```
                        AREA myvar, DATA, READWRITE

                ;definition of variables in RAM space. Not asked for in the exercise
                X               DCD             0
                Y               DCD             0
                XP              DCD             0

                AREA myCode, CODE, READONLY

                THUMB


main            PROC
                EXPORT main
                ;This program is written in order to make it easy to understand
                ; Efficiency is not the important aspect in this implementation

                ; First point to variables using registers
                LDR R3,=X       ;reserve place to store the address of X in literal pool. Initialise R3 with address of X
                LDR R4,=Y       ;reserve place to store the address of Y in literal pool. Initialise R4 with address of Y
                LDR R5,=XP      ;reserve place to store the address of XP in literal pool. Initialise R5 with address of XP

                ; X = 3
                LDR R0,=0x03    ; X = 3
loop1           STR R0,[R3]     ; Now store 3 in X

                ;XP = &X
                STR R3,[R5]     ; Store the address of X (that address is in R3) in XP (address of XP in R5)

                ;Y = *XP
                LDR R2,[R5]     ; read contents of XP variable (pointed to by R5) in R2. Address of X now in R2
                LDR R1,[R2]     ; Read contents of X in R1
                STR R1,[R4]     ; Store R1 (contents of X) in Y (pointed to by R4)

                ; this not asked for in the exercise
                ADDS R0, #01
                B   loop1
                ENDP
                ALIGN
                END
```

Details to the solution (given to help you see what is happening).




```
 19:                    LDR R3,=X            0x080007B8 4B05    LDR    r3,[pc,#20]  ; @0x080007D0
 20:                    LDR R4,=Y            0x080007BA 4C06    LDR    r4,[pc,#24]  ; @0x080007D4
 21:                    LDR R5,=XP           0x080007BC 4D06    LDR    r5,[pc,#24]  ; @0x080007D8
 22:
 23:                    ; X = 3
 24:                    LDR R0,=0x03         0x080007BE 4807    LDR    r0,[pc,#28]  ; @0x080007DC
 25: loop1              STR R0,[R3]          0x080007C0 6018    STR    r0,[r3,#0x00]
 26:
 27:                    ;XP = &X
 28:                    STR R3,[R5]          0x080007C2 602B    STR    r3,[r5,#0x00]
 29:
 30:                    ;Y = *XP
 31:                    LDR R2,[R5]          0x080007C4 682A    LDR    r2,[r5,#0x00]
 32:                    LDR R1,[R2]          0x080007C6 6811    LDR    r1,[r2,#0x00]
 33:                    STR R1,[R4]          0x080007C8 6021    STR    r1,[r4,#0x00]

 34:
 35:                    ; this not asked for in the exercise
 36:                    ADDS R0, #01         0x080007CA 3001    ADDS   r0,r0,#0x01

 37:                    B   loop1            0x080007CC E7F8    B      0x080007C0

0x080007CE 0000     DCW     0x0000
0x080007D0 0000     DCW     0x0000
0x080007D2 2000     DCW     0x2000
0x080007D4 0004     DCW     0x0004
0x080007D6 2000     DCW     0x2000
0x080007D8 0008     DCW     0x0008
0x080007DA 2000     DCW     0x2000
0x080007DC 0003     DCW     0x0003
0x080007DE 0000     DCW     0x0000
0x080007E0 0800     DCW     0x0800
```

Another possible solution for exercise 10a.

As given by the C compiler when code was written in C.

Note the MOVS. (This is not always good because it affects flags. So it should be used carefully)

```
This solution does not use pseudo instructions
It is here for you to see what the compiler could do
In a normal solution in assembler, it is better to use pseudo
instructions in order to create the literal pool. See
solution above.
The lines in blue are only there for help.


              x = 3;
0x08000254 2003      MOVS     r0,#0x03
0x08000256 4905      LDR      r1,[pc,#20]  ; @0x0800026C
0x08000258 6008      STR      r0,[r1,#0x00]
              xp = &x;
0x0800025A 4608      MOV      r0,r1
0x0800025C 4904      LDR      r1,[pc,#16]  ; @0x08000270
0x0800025E 6008      STR      r0,[r1,#0x00]
              y = *xp;
0x08000260 4608      MOV      r0,r1
0x08000262 6800      LDR      r0,[r0,#0x00]
0x08000264 6800      LDR      r0,[r0,#0x00]
0x08000266 4903      LDR      r1,[pc,#12]  ; @0x08000274
0x08000268 6008      STR      r0,[r1,#0x00]

0x0800026C 0000      DCW      0x0000
0x0800026E 2000      DCW      0x2000
0x08000270 0008      DCW      0x0008
0x08000272 2000      DCW      0x2000
0x08000274 0004      DCW      0x0004
0x08000276 2000      DCW      0x2000
```