PES UNIVERSITY

Department of Computer Science & Engineering

Microprocessor & Computer Architecture Lab

UE23CS251B

WEEK 4 submission

Name of the Student	Pranav Hemanth
SRN	PES1UG23CS433
Section	G
Department	CSE
Campus	RR

Department of Computer Science & Engineering
Microprocessor & Computer Architecture Lab

UE23CS251B

1 Write an ALP using ARM7TDMI to add n numbers bytewise.

. .DATA

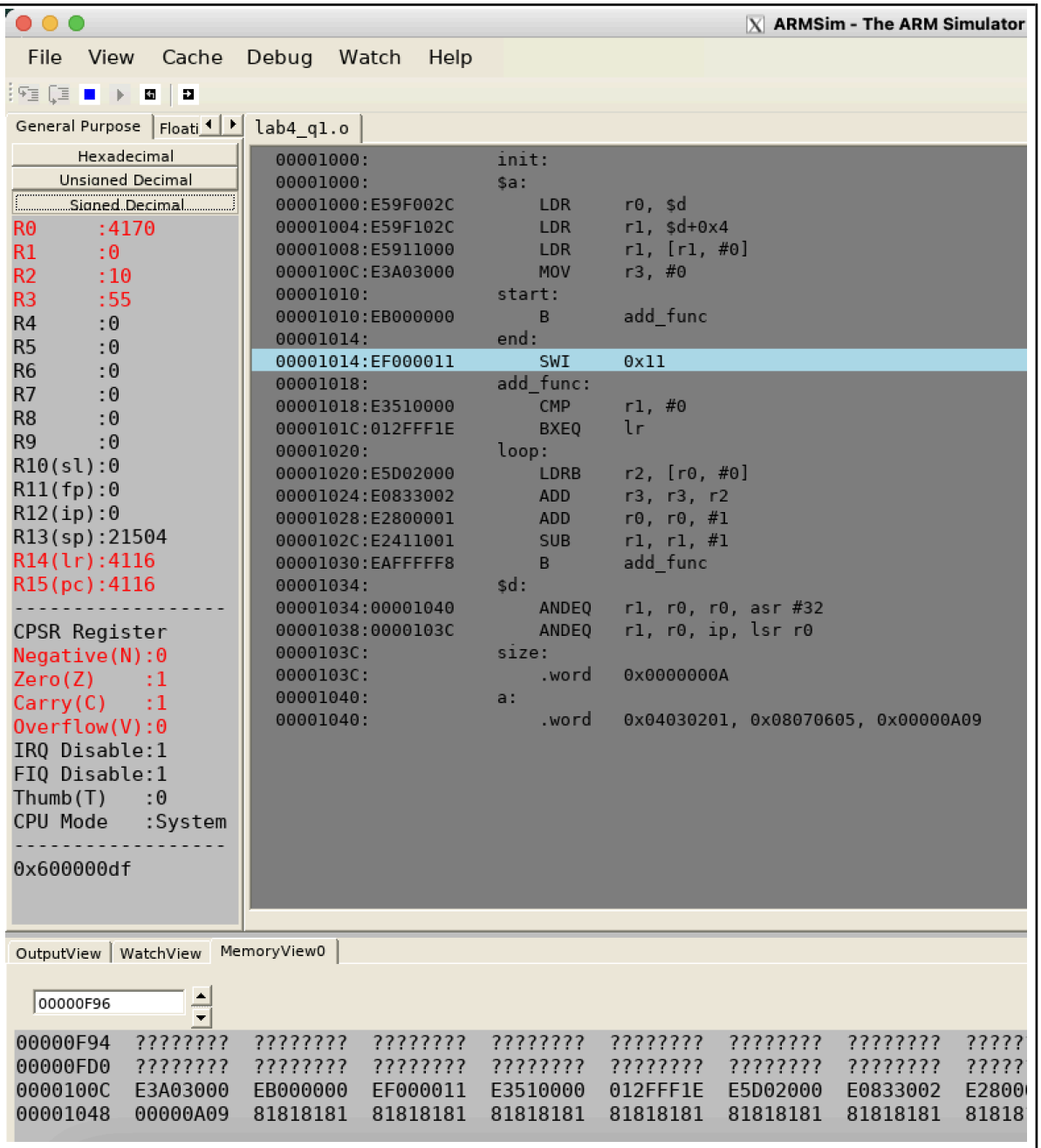
A: .byte 1,2,3,4,5,6,7,8,9,10

Program screen shot:

ARMSim_files > ASM lab4_q1.s

```
1   @ Write an ALP using ARM7TDMI to add n numbers bytewise using BL.
2
3   .data
4
5   size: .word 10
6   A: .byte 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
7
8   .text
9
10  init:
11      LDR R0, =A
12      LDR R1, =size
13      LDR R1, [R1]
14      MOV R3, #0
15
16  start:
17      BL add_func
18
19  end:
20      SWI 0x011
21
22  add_func:
23      CMP R1, #0
24      BXEQ LR
25
26  loop:
27      LDRB R2, [R0]
28      ADD R3, R3, R2
29      ADD R0, R0, #1
30      SUB R1, R1, #1
31      B add_func
```

Screen shot of Register set output:



2 Write an ALP using ARM7TDMI to generate a square given matrix with A

If (i==j) then A[i][j]=5
Otherwise A[i][j]=0
(Note:Any size of matrix can be given as input)

Considering 4X4 matrix

Example :

5	0	0	0
0	5	0	0
0	0	5	0
0	0	0	0

Before:
A:.word 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

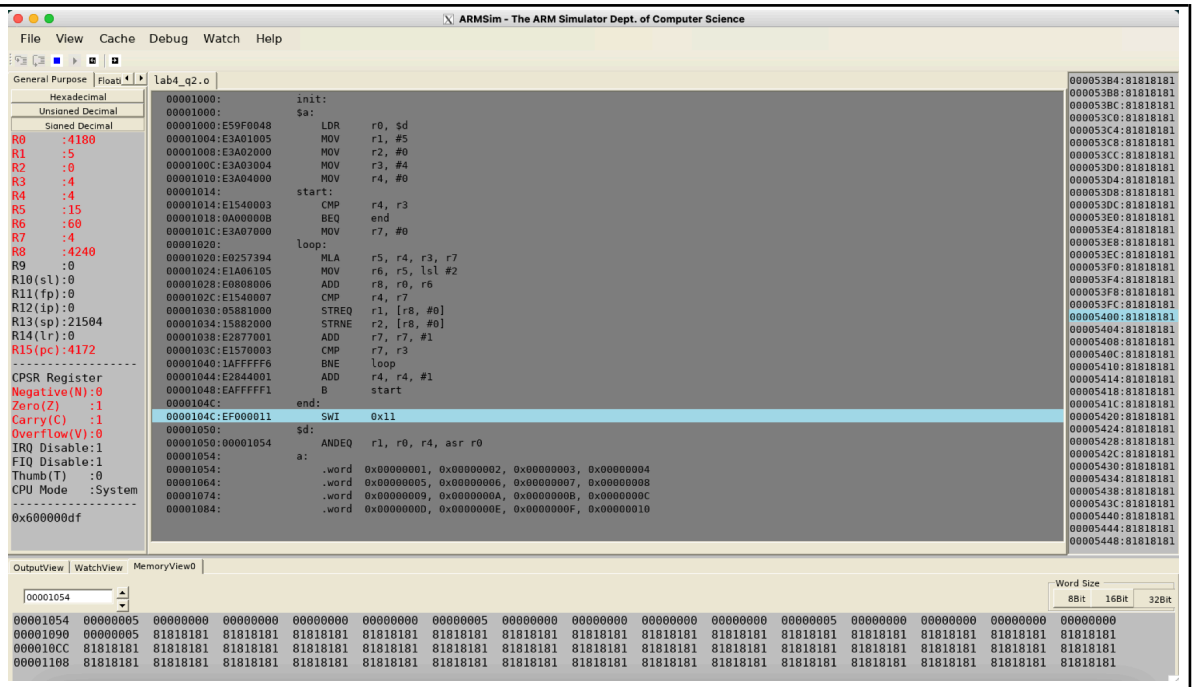
After:
A:.word 5,0,0,0,0,5,0,0,0,0,5,0,0,0,0,5

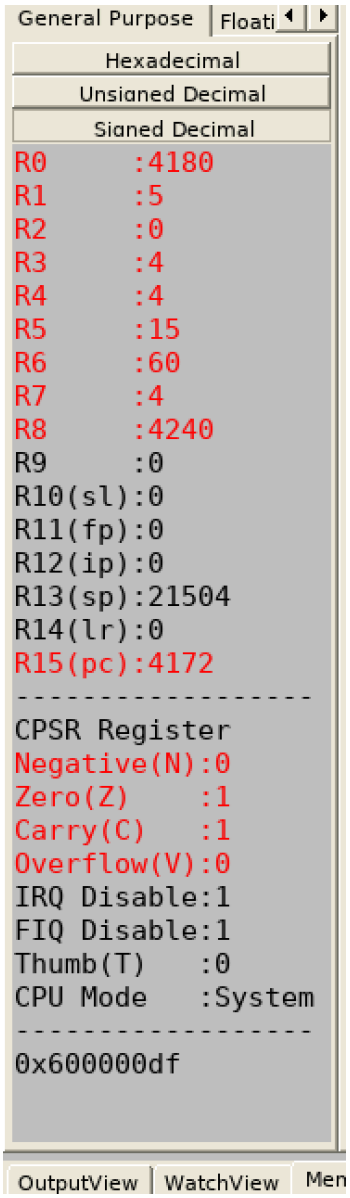
Program screen shot:

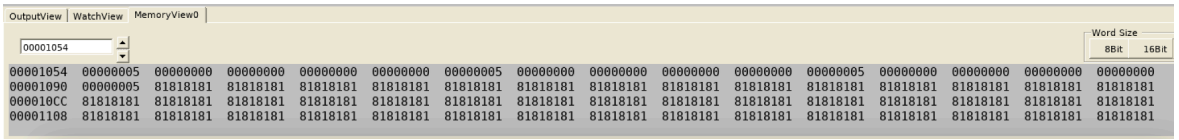
ARMSim_files > lab4_q2.s

```
@ Considering 4X4 matrix

@ Example :     5 0 0 0
@               0 5 0 0
@               0 0 5 0
@               0 0 0 0

@Before:
@A:.word 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

@After:
@A:.word 5,0,0,0,0,5,0,0,0,0,5,0,0,0,0,5

.data
A: .word 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

.text

init:
    LDR R0, =A
    MOV R1, #5       // Value to store in the diagonal
    MOV R2, #0       // Value to store for non-diagonal elements
    MOV R3, #4       // Size of the matrix (4x4)
    MOV R4, #0       // Row index

start:
    CMP R4, R3
    BEQ end
    MOV R7, #0

loop:
    MLA R5, R4, R3, R7
    MOV R6, R5, LSL #2
    ADD R8, R0, R6
    CMP R4, R7          // Check if row index == column index
    STREQ R1, [R8]
    STRNE R2, [R8]
    ADD R7, R7, #1
    CMP R7, R3
    BNE loop
    ADD R4, R4, #1
    B start

end:
    SWI 0x011
```

Screen shot of Register set output and memory location:


ARMSim - The ARM Simulator Dept. of Computer Science
File View Cache Debug Watch Help
General Purpose
lab4_q2.o
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :4180
R1 :5
R2 :0
R3 :4
R4 :4
R5 :15
R6 :60
R7 :4
R8 :4240
R9 :0
R10(sl):0
R11(fp):0
R12(ip):0
R13(sp):21504
R14(lr):0
R15(pc):4172
CPSR Register
Negative(N):0
Zero(Z) :1
Carry(C) :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T) :0
CPU Mode :System
0x600000df
00001000: init:
00001000: $a:
00001000:E59F0048 LDR r0, $d
00001004:E3A01005 MOV r1, #5
00001008:E3A02000 MOV r2, #0
0000100C:E3A03004 MOV r3, #4
00001010:E3A04000 MOV r4, #0
00001014: start:
00001014:E1540003 CMP r4, r3
00001018:0A00000B BEQ end
0000101C:E3A07000 MOV r7, #0
00001020: loop:
00001020:E0257394 MLA r5, r4, r3, r7
00001024:E1A06105 MOV r6, r5, lsl #2
00001028:E0808006 ADD r8, r0, r6
0000102C:E1540007 CMP r4, r7
00001030:05881000 STREQ r1, [r8, #0]
00001034:15882000 STRNE r2, [r8, #0]
00001038:E2877001 ADD r7, r7, #1
0000103C:E1570003 CMP r7, r3
00001040:1AFFFFF6 BNE loop
00001044:E2844001 ADD r4, r4, #1
00001048:EAFFFFF1 B start
0000104C: end:
0000104C:EF000011 SWI 0x11
00001050: $d:
00001050:00001054 ANDEQ r1, r0, r4, asr r0
00001054: a:
00001054: .word 0x00000001, 0x00000002, 0x00000003, 0x00000004
00001064: .word 0x00000005, 0x00000006, 0x00000007, 0x00000008
00001074: .word 0x00000009, 0x0000000A, 0x0000000B, 0x0000000C
00001084: .word 0x0000000D, 0x0000000E, 0x0000000F, 0x00000010
000053B4:81818181
000053B8:81818181
000053BC:81818181
000053C0:81818181
000053C4:81818181
000053C8:81818181
000053CC:81818181
000053D0:81818181
000053D4:81818181
000053D8:81818181
000053DC:81818181
000053E0:81818181
000053E4:81818181
000053E8:81818181
000053EC:81818181
000053F0:81818181
000053F4:81818181
000053F8:81818181
000053FC:81818181
00005400:81818181
00005404:81818181
00005408:81818181
0000540C:81818181
00005410:81818181
00005414:81818181
00005418:81818181
0000541C:81818181
00005420:81818181
00005424:81818181
00005428:81818181
0000542C:81818181
00005430:81818181
00005434:81818181
00005438:81818181
0000543C:81818181
00005440:81818181
00005444:81818181
00005448:81818181
OutputView WatchView MemoryView0
00001054
Word Size
8Bit 16Bit 32Bit
00001054 00000005 00000000 00000000 00000000 00000000 00000005 00000000 00000000 00000000 00000000 00000005 00000000 00000000 00000000 00000000
00001090 00000005 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
000010CC 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181
00001108 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181





3 Write an ALP using ARM7TDMI to convert hexadecimal to decimal.

Input: **0x0124**(Hex)

Conversion Process:

- 0x124⟶ Extract **4**⟶ 0 * 16 + 4 = 4
- 0x124 ⟶ Extract **2** ⟶ 2 * 16^1 + 4 = 36
- 0x124 ⟶ Extract **1** ⟶ 1* 16^2 + 36 =292

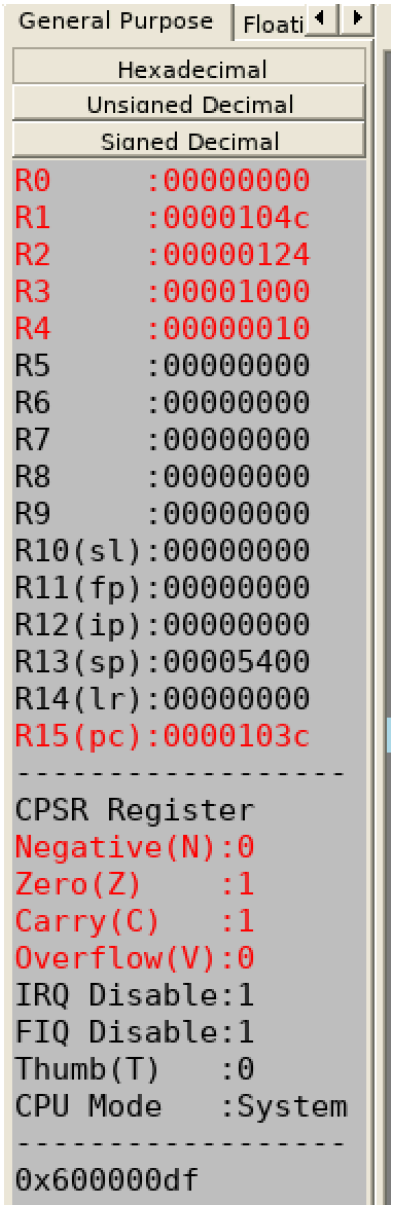
Output: **292(**Decimal)

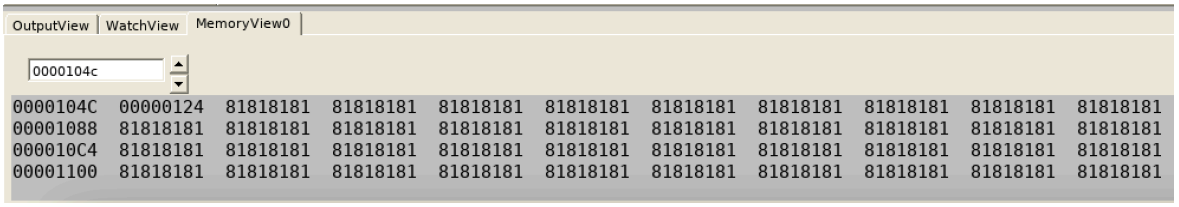
Program screen shot:

ARMSim_files > lab4_q3.s

```
@ Write an ALP using ARM7TDMI to convert hexadecimal to decimal.
@ Input: 0x0124(Hex)
@ Conversion Process:
@ 0x124→ Extract 4→ 0 * 16 + 4 = 4
@ 0x124 → Extract 2 → 2 * 16^1 + 4 = 36
@ 0x124 → Extract 1 → 1* 16^2 + 36 =292
@ Output: 292(Decimal)

.data
hex: .word 0x0124
decimal: .word 0

.text

init:
    LDR R0, =hex
    LDR R0, [R0]
    MOV R2, #0
    MOV R3, #1

loop:
    CMP R0, #0
    BEQ end

    AND R4, R0, #0xF   @ Extract the least significant digit (last 4 bits)
    MUL R4, R3, R4     @ Multiply the extracted digit by the current multiplier (16^0, 16^1, etc.)
    ADD R2, R2, R4

    MOV R0, R0, LSR #4 @ Shift R0 right by 4 bits to process the next digit

    MOV R4, #16
    MUL R3, R4, R3     @ Update the multiplier for the next place value (16^1, 16^2, etc.)

    B loop

end:
    LDR R1, =decimal
    STR R2, [R1]
    SWI 0x011
```

Screen shot of Register set output and memory location:





4 Write an ALP using ARM7TDMI, for the given matrix arranged in Column major order, find the index of an element if coordinates of a matrix is given and also find the address of the indexed element. (Using MLA instruction)

. DATA

A:.WORD 1,2,3,4,5,6,7,8,9

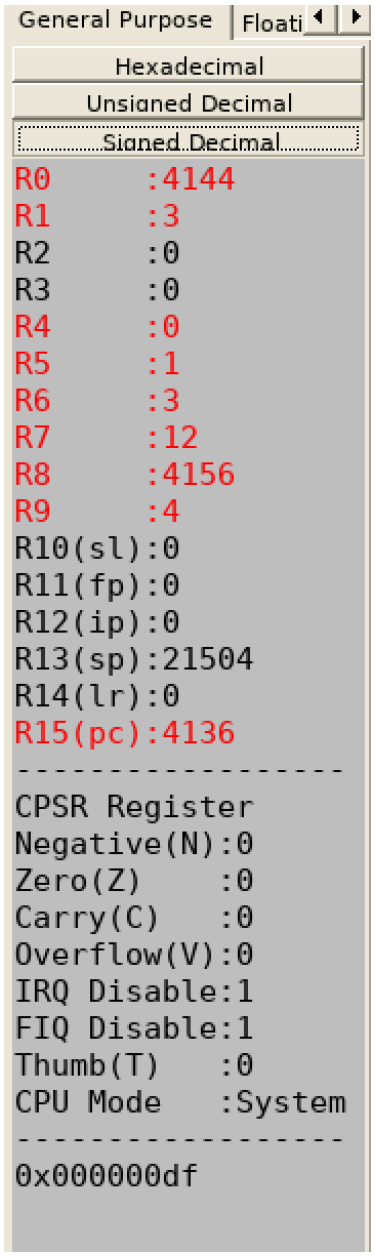
.Index for the column major= y*no of rows+x

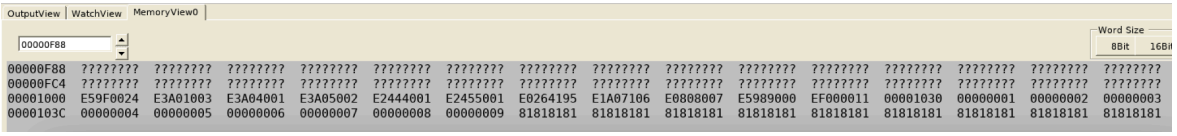
Program screen shot:

ARMSim_files > lab4_q4.s

```
1  @ Write an ALP using ARM7TDMI, for the given matrix arranged in Column major order, find the index of an element
2  @ if coordinates of a matrix is given and also find the address of the indexed element. (Using MLA instruction)
3  @ . DATA
4  @
5  @ A:.WORD 1,2,3,4,5,6,7,8,9
6  @ .Index for the column major=  y*no of rows+x
7
8  .data
9  A: .word 1,2,3,4,5,6,7,8,9
10
11 .text
12
13 init:
14     LDR R0, =A
15     MOV R1, #3          @ Number of rows
16     MOV R4, #1          @ x-coordinate (row index, assuming 1-based indexing)
17     MOV R5, #2          @ y-coordinate (column index, assuming 1-based indexing)
18
19     SUB R4, R4, #1      @ Convert x to 0-based index
20     SUB R5, R5, #1      @ Convert y to 0-based index
21
22 start:
23     MLA R6, R5, R1, R4
24     MOV R7, R6, LSL #2
25     ADD R8, R0, R7
26
27 end:
28     LDR R9, [R8]
29     SWI 0x011
30
```

Screen shot of Register set output and memory location:


General Purpose
Floati
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :4144
R1 :3
R2 :0
R3 :0
R4 :0
R5 :1
R6 :3
R7 :12
R8 :4156
R9 :4
R10(sl):0
R11(fp):0
R12(ip):0
R13(sp):21504
R14(lr):0
R15(pc):4136
CPSR Register
Negative(N):0
Zero(Z) :0
Carry(C) :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T) :0
CPU Mode :System
0x000000df

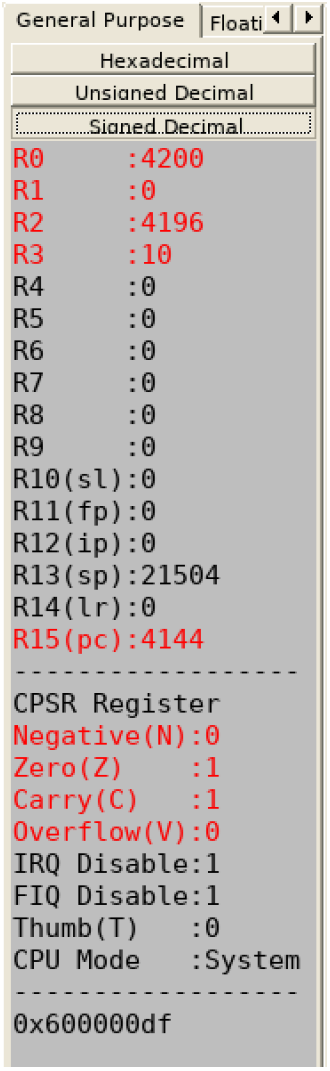

OutputView
WatchView
MemoryView0
00000F88
Word Size
8Bit
16Bit
00000F88 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ????????
00000FC4 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ????????
00001000 E59F0024 E3A01003 E3A04001 E3A05002 E2444001 E2455001 E0264195 E1A07106 E0808007 E5989000 EF000011 00001030 00000001 00000002 00000003
0000103C 00000004 00000005 00000006 00000007 00000008 00000009 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181

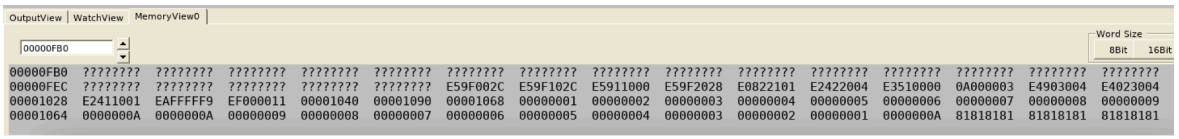
	Assignments Questions
5	Write an ALP using ARM7TDMI to reverse the elements stored in location A with location B Before: A:.word 1,2,3,4,5,6,7,8,9,10 After : A:.word 10,9,8,7,6,5,4,3,2,1 Solution 1: Program screen shot:

ARMSim_files > lab4_q5a.s

```
1   @ Write an ALP using ARM7TDMI to reverse the elements stored in location A with location B
2   @ Before:
3   @ A:.word 1,2,3,4,5,6,7,8,9,10
4   @ After :
5   @ A:.word 10,9,8,7,6,5,4,3,2,1
6
7   .data
8   A:    .word 1,2,3,4,5,6,7,8,9,10
9   B:    .word 0,0,0,0,0,0,0,0,0,0
10  size: .word 10
11
12  .text
13
14  init:
15      LDR R0, =A
16      LDR R1, =size
17      LDR R1, [R1]
18      LDR R2, =B
19
20      ADD R2, R2, R1, LSL #2 @ Move R2 to the end of array B (size * 4)
21      SUB R2, R2, #4      @ Adjust R2 to point to the last element of B
22
23  start:
24      CMP R1, #0
25      BEQ end             @ If size (R1) == 0, we're done
26
27      LDR R3, [R0], #4    @ Load current element from A into R3 and increment R0
28      STR R3, [R2], #-4   @ Store R3 into B and decrement R2 to move backward
29      SUB R1, R1, #1      @ Decrement the size counter
30
31      B start
32
33  end:
34      SWI 0x011
35
```

Screen shot of Register set output:





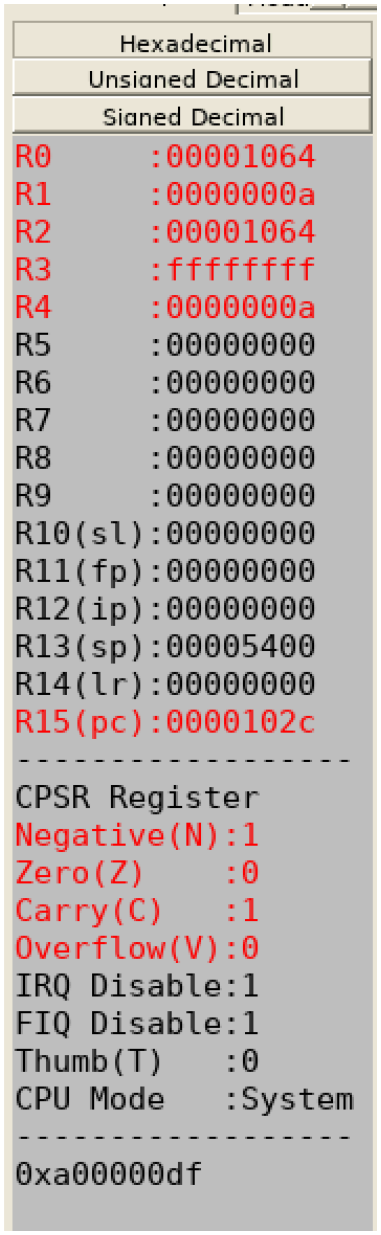
Solution 2:

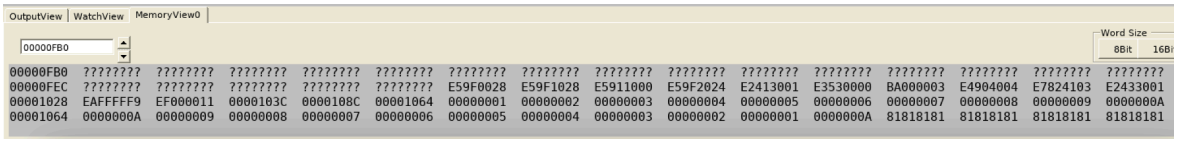
Program screen shot:

ARMSim_files > lab4_q5b.s

```
@ Write an ALP using ARM7TDMI to reverse the elements stored in location A with location B
@ Before:
@ A:.word 1,2,3,4,5,6,7,8,9,10
@ After :
@ A:.word 10,9,8,7,6,5,4,3,2,1

.data
A:    .word 1,2,3,4,5,6,7,8,9,10
B:    .word 0,0,0,0,0,0,0,0,0,0
size: .word 10

.text

init:
    LDR R0, =A
    LDR R1, =size
    LDR R1, [R1]
    LDR R2, =B

    SUB R3, R1, #1      @ Set R3 to (size - 1), used as the reverse index

start:
    CMP R3, #0          @ Check if reverse index (R3) is less than 0
    BLT end             @ If R3 < 0, we are done

    LDR R4, [R0], #4
    STR R4, [R2, R3, LSL #2] @ Store R4 into B at offset (R3 * 4)
    SUB R3, R3, #1

    B start

end:
    SWI 0x011

```

Screen shot of Register set output:





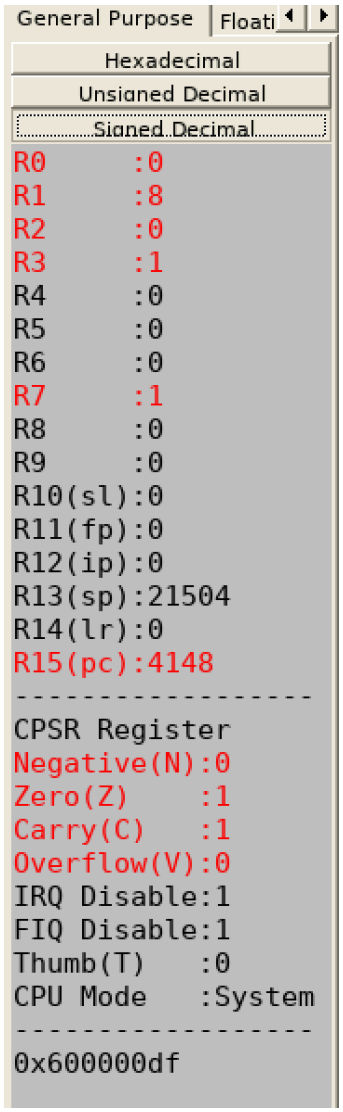
6 Write an ALP using ARM7TDMI to find the largest of all the BCD digits of a given 32bit number. (hint:If R1=17845374 the largest digit is 8

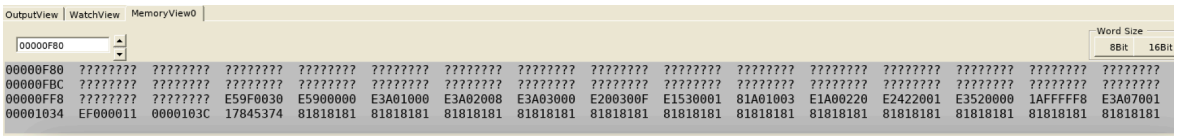
Program screen shot:

ARMSim_files > lab4_q6.s

```
@ Write an ALP using ARM7TDMI to find the largest of all the BCD digits of a given 32bit number.
@ (hint:If R1=17845374 the largest digit is 8

.data
num: .word 0x17845374  @ 32-bit number

.text

init:
    LDR R0, =num
    LDR R0, [R0]
    MOV R1, #0
    MOV R2, #8         @ number of digits in 32bit number
    MOV R3, #0

loop:
    AND R3, R0, #0xF   @ Extract the lsb (last 4 bits)
    CMP R3, R1
    MOVHI R1, R3

    MOV R0, R0, LSR #4 @ Shift R0 right by 4 bits
    SUB R2, R2, #1
    CMP R2, #0
    BNE loop

end:
    MOV R7, #1
    SWI 0x011

```

Screen shot of Register set output:





Extra

a) Write an ALP using ARM7TDMI to copy a block 400 bytes of data from location A to location B if the rate of data transfer rate is 40 bytes, LDM and STM instructions.

and

b) For the same transfer the block with auto-indexing.

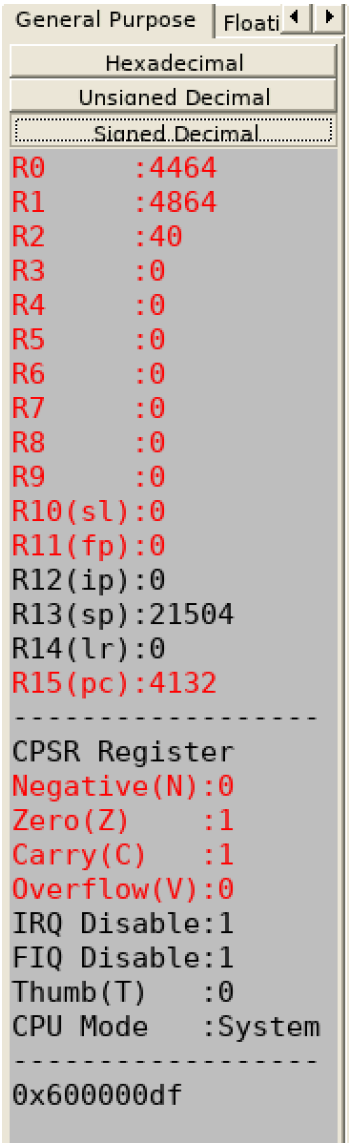
Solution A:

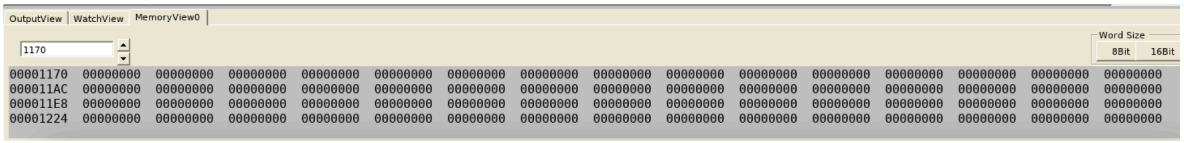
Program screen shot:

ARMSim_files > ASM lab4_q7a.s

```
@ Write an ALP using ARM7TDMI to  copy a block 400 bytes of data from location A to location B
@if the rate of data transfer rate is 40 bytes, LDM and STM instructions.
@ and
@ For the same transfer the block with auto-indexing.

@ Using LDM and STM Instructions

.data
A: .skip 400      @ 400 bytes of data at location A (source)
B: .skip 400      @ 400 bytes of data at location B (destination)

.text
init:
    LDR R0, =A
    LDR R1, =B
    MOV R2, #40     @ Set the data transfer rate to 40 bytes (one LDM/STM operation)
    MOV R3, #10     @ Set the number of transfers

loop:
    LDMIA R0!, {R4-R11} @ Load 8 registers (40 bytes) from source (A) into R4-R11, incrementing R0
    STMIA R1!, {R4-R11} @ Store the 8 registers (40 bytes) to destination (B), incrementing R1
    SUBS R3, R3, #1     @ Decrement the transfer counter
    CMP R3, #0
    BNE loop

end:
    SWI 0x11

```

Screen shot of Register set output:


General Purpose
Floati
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :4464
R1 :4864
R2 :40
R3 :0
R4 :0
R5 :0
R6 :0
R7 :0
R8 :0
R9 :0
R10(sl):0
R11(fp):0
R12(ip):0
R13(sp):21504
R14(lr):0
R15(pc):4132
CPSR Register
Negative(N):0
Zero(Z) :1
Carry(C) :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T) :0
CPU Mode :System
0x600000df


OutputView
WatchView
MemoryView0
1170
Word Size
8Bit
16Bit
00001170 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000011AC 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000011E8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00001224 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

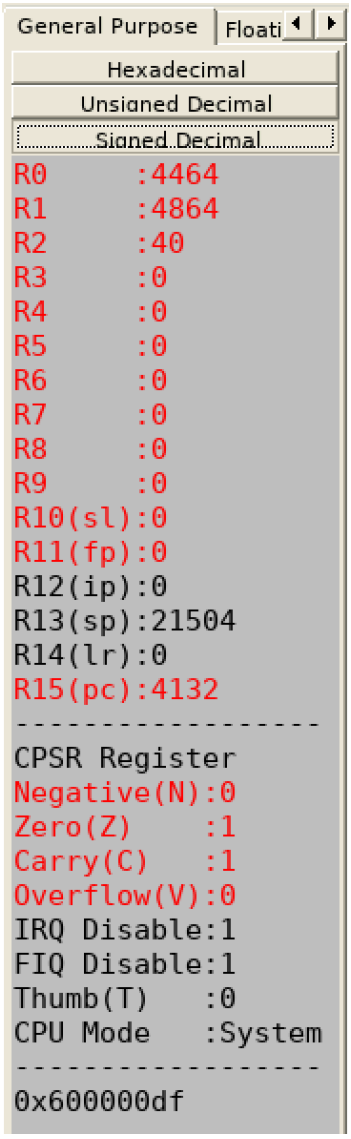
Solution B:

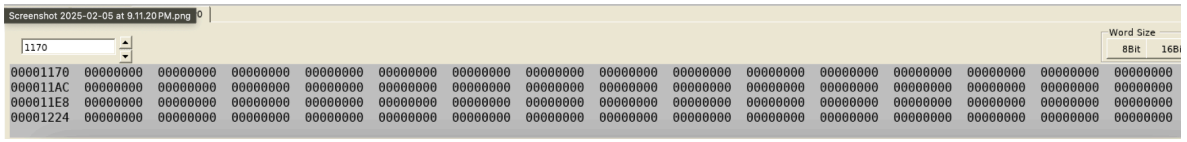
Program screen shot:

ARMSim_files > lab4_q7b.s

```
@ Write an ALP using ARM7TDMI to  copy a block 400 bytes of data from location A to location B
@if the rate of data transfer rate is 40 bytes, LDM and STM instructions.
@ and
@ For the same transfer the block with auto-indexing.

@ Using Auto-Indexing (with LDM and STM)

.data
A: .skip 400      @ 400 bytes of data at location A (source)
B: .skip 400      @ 400 bytes of data at location B (destination)

.text
init:
    LDR R0, =A
    LDR R1, =B
    MOV R2, #40     @ Set the data transfer rate to 40 bytes (one LDM/STM operation)
    MOV R3, #10     @ Set the number of transfers

loop:
    LDMIA R0!, {R4-R11} @ Load 8 registers (40 bytes) from source (A) into R4-R11, incrementing R0
    STMIA R1!, {R4-R11} @ Store the 8 registers (40 bytes) to destination (B), incrementing R1
    SUBS R3, R3, #1
    CMP R3, #0
    BNE loop

end:
    SWI 0x11
```

Screen shot of Register set output:





Q) Write an ALP using ARM7TDMI to multiply two matrices of any valid size and store the result in a matrix.

- The matrices are stored in row-major order.
- The sizes of matrices are predefined (modifiable as needed).
- MLA instruction is used.

Program screen shot:

ARMSim_files > matrix_multiplication.s

```
@ Write an ALP using ARM7TDMI to multiply two matrices of any valid size and store the result in a matrix.
@   •   The matrices are stored in row-major order.
@   •   The sizes of matrices are predefined (modifiable as needed).
@   •   MLA instruction is used.

.data
A:  .word 1, 2, 3, 4, 5, 6    @ 2x3 matrix
B:  .word 7, 8, 9, 10, 11, 12 @ 3x2 matrix
C:  .space 16                 @ 2x2 result matrix (initialized to 0)

rowsA:  .word 2   @ Rows of A
colsA:  .word 3   @ Columns of A (and Rows of B)
colsB:  .word 2   @ Columns of B

    .text
    .global _start

_start:
    LDR R4, =rowsA
    LDR R4, [R4]    @ R4 = rowsA (2)
    LDR R5, =colsA
    LDR R5, [R5]    @ R5 = colsA (3)
    LDR R6, =colsB
    LDR R6, [R6]    @ R6 = colsB (2)

    LDR R0, =A
    LDR R1, =B
    LDR R2, =C

    MOV R7, #0      @ i = 0 (Row index of A)

row_loop:
    CMP R7, R4      @ if i >= rowsA, exit
    BGE end


    MOV R8, #0      @ j = 0 (Column index of B)

col_loop:
    CMP R8, R6      @ if j >= colsB, next row
    BGE next_row

    MOV R9, #0      @ k = 0 (Column index of A / Row index of B)
    MOV R10, #0     @ result = 0

mul_loop:
    CMP R9, R5      @ if k >= colsA, store result
    BGE store_result

    MLA R3, R7, R5, R9  @ R3 = i * colsA + k
    LDR R11, [R0, R3, LSL #2]  @ A[i][k]

    MLA R3, R9, R6, R8  @ R3 = k * colsB + j
    LDR R12, [R1, R3, LSL #2]  @ B[k][j]

    MLA R10, R11, R12, R10     @ result += A[i][k] * B[k][j]

    ADD R9, R9, #1             @ k++
    B mul_loop

store_result:
    MLA R3, R7, R6, R8  @ R3 = i * colsB + j
    STR R10, [R2, R3, LSL #2]  @ Store result in C[i][j]

    ADD R8, R8, #1      @ j++
    B col_loop

next_row:
    ADD R7, R7, #1      @ i++
    B row_loop

end:
    SWI 0x011
```

Screen shot of Register set output: