



ARM ASSEMBLY LAB-5

By

Swapneel Pimparkar (CS18M516)

CS6620a

Prof. Madhu Mutyam

STATUS UPDATE

The ARM assembly code was written for 32 bit processor and verified using ARMSim simulator successfully.

EXERCISE 1 – CONVERT FROM AN ASCII CHARACTER TO A HEXADECIMAL DIGIT.

Convert the contents of a given A_DIGIT variable from an ASCII character to a hexadecimal digit and store the result in H_DIGIT. Assume that A_DIGIT contains the ASCII representation of a hexadecimal digit (i.e., 7 bits with MSB=0).

The logic to solve this exercise is hand-coded in 32 bit ARM Assembly and verified on ARMSim Simulator.

The logic used is mentioned in the code file itself and all the necessary instructions are supplied with comments.

Various unit tests are also part of the code file itself. Need to uncomment one by one and run. All the unit tests passed including two error cases.

All the output locations are initialized with 0xFF.

SCREENSHOT - TEST 1

The sample output screenshot for the logic is as below for input (which is essentially error case) –

A_DIGIT:

.word 0x74

H_DIGIT:

.word 0xFF

ARMSim# - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView

General Purpose Floating Point

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 : 00000000

R1 : 00000000

R2 : 00000000

R3 : 00000000

R4 : 00000000

R5 : 00000000

R6 : 00001044

R7 : 00001048

R8 : 0000003d

R9 : 00000000

R10 (s1) : 00000000

R11 (fp) : 00000000

R12 (ip) : 00000000

R13 (sp) : 00011400

R14 (lr) : 00000000

R15 (pc) : 0000103c

CPSR Register

Negative (N) : 0

Zero (Z) : 0

Carry (C) : 1

Overflow (V) : 0

IRQ Disable : 1

FIQ Disable : 1

Thumb (T) : 0

CPU Mode : System

0x200000df

CodeView

Lab5_1.o

```
@TEST 5 : Output 0xFF i.e. H_DIGIT = FF is expected. This is error case.
@
@   A_DIGIT:
@       .word 0x4B
@   H_DIGIT:
@       .word 0xFF

.text
.align 2
.global MAIN
.global _OUT
.global _END

@ Program starts here
_MAIN:

00001000:E59F6034    LDR R6, =A_DIGIT    @ Read A_DIGIT address into R6
00001004:E59F7034    LDR R7, =H_DIGIT    @ Read H_DIGIT address into R7. This is where output will be stored
00001008:E5968000    LDR R8, [R6]        @ Load the contents of R6 i.e. A_DIGIT into R8
0000100C:E2488030    SUB R8, #0x30        @ Subtract 0x30 from R8 as first printable in three sets starts wit
@@ in ASCII table as mentioned in logic at the start of this progra
00001010:E358000A    CMP R8, #0xA        @ Compare the subtraction in R8 with 0xA. This is for digits 0-9.
00001014:BAFFFFFFE    BLT _OUT            @ If less than 0xA then jump to _OUT. It means that digit is in the
00001018:E2488007    SUB R8, #7          @ Else subtract 7 from above subtraction (16 - 9)

@ With following series of three comparisons, we can take care of all the three sets of
@ printable ASCII characters within HEX range.
@ i.e. 0-9, A-F and a-f.

0000101C:E3580030    CMP R8, #0x30        @ Compare now again with 0x30 to see if we have any character beyon
@ If R8 is greater than 0x30 then we have error case. Keep output a
@@ To do this, jump to _END.
00001024:E3580020    CMP R8, #0x20        @ Else compare R8 with 0xA. This is to take care of characters in t
@ Jump to _OUT if content in R8 is greater than 0x20.
00001028:CAFFFFFFFE    BGT _OUT
0000102C:E3580010    CMP R8, #0x10        @ Compare R8 with 0x10.
00001030:CAFFFFFFFE    BGT _END            @ Jump to _END if content in R8 is greater than 0xA.

_OUT:
00001034:E208800F    AND R8, R8, #0x0F    @ Clear for MSBs.
00001038:E5878000    STR R8, [R7]        @ Store value in R8 into address pointed by R7 i.e. into H_DIGIT.

_END:

0000103C:00000000    .end
:00000004
```

MemoryView1 MemoryView0

1048

00001048 00FF 0000 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

0000108E 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

000010D4 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

0000111A 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

00001160 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

SCREENSHOT - TEST 2

The sample output screenshot for the logic is as below for input (which is essentially good case for small case letter) –

A_DIGIT:

.word 0x64

H_DIGIT:

.word 0xFF

The screenshot displays the ARMSim# - The ARM Simulator interface. The main window is titled 'CodeView' and shows assembly code for 'Lab5_1.o'. The code includes comments and instructions for reading A_DIGIT and H_DIGIT, performing comparisons, and storing the result in H_DIGIT. The RegistersView window on the left shows the values of various registers, with R15 (pc) highlighted as 0000103c. The MemoryView window at the bottom shows the memory contents at address 1048, displaying a sequence of 0000 and 8181 values.

```

@TEST 5 : Output 0xFF i.e. H_DIGIT = FF is expected. This is error case.
@
@   A_DIGIT:
@       .word 0x4B
@   H_DIGIT:
@       .word 0xFF

.text
.align 2
.global _MAIN
.global _OUT
.global _END

@ Program starts here
_MAIN:

00001000:E59F6034    LDR    R6, =A_DIGIT    @ Read A_DIGIT address into R6
00001004:E59F7034    LDR    R7, =H_DIGIT    @ Read H_DIGIT address into R7. This is where output will be s
00001008:E5968000    LDR    R8, [R6]        @ Load the contents of R6 i.e. A_DIGIT into R8
0000100C:E2488030    SUB    R8, #0x30       @ Subtract 0x30 from R8 as first printable in three sets start
                        @@ in ASCII table as mentioned in logic at the start of this p
00001010:E358000A    CMP    R8, #0xA        @ Compare the subtraction in R8 with 0xA. This is for digits 0
00001014:BAFFFFFE    BLT    _OUT            @ If less than 0xA then jump to _OUT. It means that digit is i
00001018:E2488007    SUB    R8, #7          @ Else subtract 7 from above subtraction (16 - 9)

@ With following series of three comparisons, we can take care of all the three sets o
@ printable ASCII characters within HEX range.
@ i.e. 0-9, A-F and a-f.

0000101C:E3580030    CMP    R8, #0x30       @ Compare now again with 0x30 to see if we have any character
00001020:CAFFFFFFFE    BGT    _END            @ If R8 is greater than 0x30 then we have error case. Keep out
                        @@ To do this, jump to _END.
00001024:E3580020    CMP    R8, #0x20       @ Else compare R8 with 0xA. This is to take care of characters
00001028:CAFFFFFFFE    BGT    _OUT            @ Jump to _OUT if content in R8 is greater than 0x20.
0000102C:E3580010    CMP    R8, #0x10       @ Compare R8 with 0x10.
00001030:CAFFFFFFFE    BGT    _END            @ Jump to _END if content in R8 is greater than 0xA.

_OUT:
00001034:E208800F    AND    R8, R8, #0x0F   @ Clear for MSBs.
00001038:E5878000    STR    R8, [R7]        @ Store value in R8 into address pointed by R7 i.e. into H_DIG

_END:
0000103C:00000000    .end
:00000004

```

RegistersView:

Register	Value
R0	:00000000
R1	:00000000
R2	:00000000
R3	:00000000
R4	:00000000
R5	:00000000
R6	:00001044
R7	:00001048
R8	:0000000d
R9	:00000000
R10 (s1)	:00000000
R11 (fp)	:00000000
R12 (ip)	:00000000
R13 (sp)	:00011400
R14 (lr)	:00000000
R15 (pc)	:0000103c

CPSR Register:

Negative (N) : 0
Zero (Z) : 0
Carry (C) : 1
Overflow (V) : 0
IRQ Disable : 1
FIQ Disable : 1
Thumb (T) : 0
CPU Mode : System

0x200000df

MemoryView1: 1048

MemoryView0:

Address	Value
00001048	000D 0000 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181
0000104E	8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181
00001054	8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

SCREENSHOT- TEST 3

The sample output screenshot for the logic is as below for input (which is essentially good case for decimal digit) –

A_DIGIT:

.word 0x35

H_DIGIT:

.word 0xFF

The screenshot displays the ARMSim# ARM Simulator interface. On the left, the 'RegistersView' panel shows the state of various registers. R15 (pc) is highlighted in red and shows the value 0000103a. Below the registers, the CPSR Register status is shown with fields like Negative (N), Zero (Z), Carry (C), Overflow (V), IRQ Disable, FIQ Disable, Thumb (T), and CPU Mode (System). The main 'CodeView' panel shows assembly code for a program named 'Lab5_1.o'. The code includes comments and instructions for loading A_DIGIT and H_DIGIT into registers, performing subtraction, and conditional branching based on the result. The memory view at the bottom shows a sequence of memory addresses and their corresponding values, with some values highlighted in red.

```
@TEST 5 : Output 0xFF i.e. H_DIGIT = FF is expected. This is error case.
@
@   A_DIGIT:
@
@   H_DIGIT:
@
@       .word 0x4B
@       .word 0xFF

.text
.align 2
.global _MAIN
.global _OUT
.global _END

@ Program starts here
_MAIN:

00001000:E59F6034      LDR    R6, =A_DIGIT      @ Read A_DIGIT address into R6
00001004:E59F7034      LDR    R7, =H_DIGIT      @ Read H_DIGIT address into R7. This is where output will be
00001008:E5968000      LDR    R8, [R6]          @ Load the contents of R6 i.e. A_DIGIT into R8
0000100C:E2488030      SUB    R8, #0x30         @ Subtract 0x30 from R8 as first printable in three sets start
                        @ in ASCII table as mentioned in logic at the start of this
00001010:E358000A      CMP    R8, #0xA          @ Compare the subtraction in R8 with 0xA. This is for digits
00001014:BAFFFFFE      BLT    _OUT              @ If less than 0xA then jump to _OUT. It means that digit is
00001018:E2488007      SUB    R8, #7            @ Else subtract 7 from above subtraction (16 - 9)

                        @ With following series of three comparisons, we can take care of all the three sets
                        @ printable ASCII characters within HEX range.
                        @ i.e. 0-9, A-F and a-f.

0000101C:E3580030      CMP    R8, #0x30         @ Compare now again with 0x30 to see if we have any character
00001020:CAFFFFFE      BGT    _END              @ If R8 is greater than 0x30 then we have error case. Keep on
                        @ To do this, jump to _END.
00001024:E3580020      CMP    R8, #0x20         @ Else compare R8 with 0xA. This is to take care of character
00001028:CAFFFFFE      BGT    _OUT              @ Jump to _OUT if content in R8 is greater than 0x20.
0000102C:E3580010      CMP    R8, #0x10         @ Compare R8 with 0x10.
00001030:CAFFFFFE      BGT    _END              @ Jump to _END if content in R8 is greater than 0xA.

_OUT:
00001034:E208800F      AND    R8, R8, #0x0F     @ Clear for MSBs.
00001038:E5878000      STR    R8, [R7]          @ Store value in R8 into address pointed by R7 i.e. into H_DI

_END:
0000103C:00000000      .end
:00000004
```

SCREENSHOT - TEST 4

The sample output screenshot for the logic is as below for input (which is essentially good case for capital letter 'C') –

A_DIGIT:

.word 0x43

H_DIGIT:

.word 0xFF

ARMSim# - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView

General Purpose Floating Point

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 : 00000000

R1 : 00000000

R2 : 00000000

R3 : 00000000

R4 : 00000000

R5 : 00000000

R6 : 00001044

R7 : 00001048

R8 : 00000000

R9 : 00000000

R10 (s1) : 00000000

R11 (fp) : 00000000

R12 (ip) : 00000000

R13 (sp) : 00011400

R14 (lr) : 00000000

R15 (pc) : 00001030

CPSR Register

Negative (N) : 1

Zero (Z) : 0

Carry (C) : 0

Overflow (V) : 0

IRQ Disable : 1

FIQ Disable : 1

Thumb (T) : 0

CPU Mode : System

0x800000df

CodeView

Lab5_1.o

```

@TEST 5 : Output 0xFF i.e. H_DIGIT = FF is expected. This is error case.
@ A_DIGIT:
@ .word 0x4B
@ H_DIGIT:
@ .word 0xFF

.text
.align 2
.global _MAIN
.global _OUT
.global _END

@ Program starts here
_MAIN:

00001000:E59F6034    LDR R6, =A_DIGIT    @ Read A_DIGIT address into R6
00001004:E59F7034    LDR R7, =H_DIGIT    @ Read H_DIGIT address into R7. This is where output will be
00001008:E5968000    LDR R8, [R6]         @ Load the contents of R6 i.e. A_DIGIT into R8
0000100C:E2488030    SUB R8, #0x30        @ Subtract 0x30 from R8 as first printable in three sets star
                        @@ in ASCII table as mentioned in logic at the start of this

00001010:E358000A    CMP R8, #0xA         @ Compare the subtraction in R8 with 0xA. This is for digits
00001014:BAFFFFFE    BLT _OUT             @ If less than 0xA then jump to _OUT. It means that digit is
00001018:E2488007    SUB R8, #7           @ Else subtract 7 from above subtraction (16 - 9)

@ With following series of three comparisons, we can take care of all the three sets
@ printable ASCII characters within HEX range.
@ i.e. 0-9, A-F and a-f.

0000101C:E3580030    CMP R8, #0x30        @ Compare now again with 0x30 to see if we have any character
00001020:CAFFFFFE    BGT _END             @ If R8 is greater than 0x30 then we have error case. Keep ou
                        @@ To do this, jump to _END.

00001024:E3580020    CMP R8, #0x20        @ Else compare R8 with 0xA. This is to take care of character
00001028:CAFFFFFE    BGT _OUT             @ Jump to _OUT if content in R8 is greater than 0x20.
0000102C:E3580010    CMP R8, #0x10        @ Compare R8 with 0x10.
00001030:CAFFFFFE    BGT _END             @ Jump to _END if content in R8 is greater than 0xA.

_OUT:
00001034:E208800F    AND R8, R8, #0x0F    @ Clear for MSBs.
00001038:E5878000    STR R8, [R7]         @ Store value in R8 into address pointed by R7 i.e. into H_DI

_END:
0000103C:00000000    .end
:00000004

```

MemoryView1 MemoryView0

1048

00001048 000C 0000 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

0000108E 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

000010D4 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

SCREENSHOT - TEST 5

The sample output screenshot for the logic is as below for input (which is essentially error case) –

A_DIGIT:

.word 0x4B

H_DIGIT:

.word 0xFF

ARMSim# - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView CodeView

General Purpose Floating Point

Hexadecimal
Unsigned Decimal
Signed Decimal

R0 : 00000000
R1 : 00000000
R2 : 00000000
R3 : 00000000
R4 : 00000000
R5 : 00000000
R6 : 00001044
R7 : 00001048
R8 : 00000014
R9 : 00000000
R10 (s1): 00000000
R11 (fp): 00000000
R12 (ip): 00000000
R13 (sp): 00011400
R14 (lr): 00000000
R15 (pc): 0000103c

CPSR Register
Negative (N): 0
Zero (Z): 0
Carry (C): 1
Overflow (V): 0
IRQ Disable: 1
FIQ Disable: 1
Thumb (T): 0
CPU Mode: System

0x200000df

Lab5_1.o

```
@TEST 5 : Output 0xFF i.e. H_DIGIT = FF is expected. This is error case.
A_DIGIT:
00001044:0000004B      .word 0x4B
H_DIGIT:
00001048:000000FF      .word 0xFF

.text
.align 2
.global _MAIN
.global _OUT
.global _END

@ Program starts here
_MAIN:

00001000:E59F6034      LDR R6, =A_DIGIT      @ Read A_DIGIT address into R6
00001004:E59F7034      LDR R7, =H_DIGIT      @ Read H_DIGIT address into R7. This is where output will
00001008:E5968000      LDR R8, [R6]          @ Load the contents of R6 i.e. A_DIGIT into R8
0000100C:E2488030      SUB R8, #0x30          @ Subtract 0x30 from R8 as first printable in three sets
                        @@ in ASCII table as mentioned in logic at the start of
00001010:E358000A      CMP R8, #0xA          @ Compare the subtraction in R8 with 0xA. This is for di
00001014:BAFFFFFFFE      BLT _OUT              @ If less than 0xA then jump to _OUT. It means that digi
00001018:E2488007      SUB R8, #7            @ Else subtract 7 from above subtraction (16 - 9)

                        @ With following series of three comparisons, we can take care of all the three
                        @ printable ASCII characters within HEX range.
                        @ i.e. 0-9, A-F and a-f.

0000101C:E3580030      CMP R8, #0x30          @ Compare now again with 0x30 to see if we have any char
00001020:CAFFFFFFFE      BGT _END              @ If R8 is greater than 0x30 then we have error case. Ke
                        @@ To do this, jump to _END.
00001024:E3580020      CMP R8, #0x20          @ Else compare R8 with 0xA. This is to take care of char
00001028:CAFFFFFFFE      BGT _OUT              @ Jump to _OUT if content in R8 is greater than 0x20.
0000102C:E3580010      CMP R8, #0x10          @ Compare R8 with 0x10.
00001030:CAFFFFFFFE      BGT _END              @ Jump to _END if content in R8 is greater than 0xA.

_OUT:
00001034:E208800F      AND R8, R8, #0x0F      @ Clear for MSBs.
00001038:E5878000      STR R8, [R7]          @ Store value in R8 into address pointed by R7 i.e. into

_END:
0000103C:00000000      .end
:00000004
```

MemoryView1 MemoryView0

1048

00001048 00FF 0000 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181
0000108E 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181
000010D4 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181 8181

EXERCISE – CONVERT A GIVEN EIGHT ASCII CHARACTERS TO AN 8-BIT BINARY NUMBER.

Convert a given eight ASCII characters in the variable `STRING` to an 8-bit binary number in the variable `NUMBER`. Clear the byte variable `ERROR` if all the ASCII characters are either ASCII “1” or ASCII “0”; otherwise set `ERROR` to all ones (0xFF).

The logic to solve this exercise is hand-coded in 32 bit ARM Assembly and verified on ARMSim Simulator.

The logic used is mentioned in the code file itself and all the necessary instructions are supplied with comments.

Two unit tests are also part of the code file itself. Need to uncomment and run.

Both the unit tests passed.

SCREENSHOT – TEST 1

The sample output screenshot for the logic is as below for input (which is essentially good case)

STRING:

.ascii "11010101"

NUMBER:

.byte 0x0

ERROR:

.byte 0xFF

The screenshot displays the ARM Simulator interface. The main window is titled 'CodeView' and shows assembly code for a program. The code is organized into sections: `.global _END`, `_MAIN:`, `_LOOP:`, `_SHIFT_AND_ADD:`, and `_END:`. Comments explain the purpose of each instruction, such as loading input string and output addresses, clearing registers, and performing loop operations. The `RegistersView` window on the left shows the state of various registers, including `R0` through `R15`, `CPSR`, and `CPUR`. The `MemoryView` window at the bottom shows the memory layout, with addresses and corresponding data values.

```
.global _END

@ Program starts here
_MAIN:
    LDR R0,=STRING      @Load the address of input string into R0
    LDR R6,=NUMBER       @Load the address of output NUMBER into R6
    LDR R7,=ERROR        @Load the address of output ERROR into R7

    EOR R3, R3, R3       @Clear NUMBER (R3).

    LDR R3, [R6]         @Load the contents of NUMBER into R3.
    MOV R9, #0           @R9 is used as loop counter

_LOOP:
    ADD R9, R9, #1       @Increase the count by 1.
    LDRB R1, [R0], #1    @Load the content of string byte by byte.
    CMP R1, #0x30        @Check if read byte is ASCII "0"
    BEQ _SHIFT_AND_ADD  @If equal then jump to _SHIFT_AND_ADD
    CMP R1, #0x31        @Check if read byte is ASCII "1"
    BEQ _SHIFT_AND_ADD  @If equal then jump to _SHIFT_AND_ADD
    EOR R3, R3, R3       @Clear NUMBER (R3).
    @In fact this is not necessary as ERROR is initialized with 0xFF.
    STRB R3, [R6]        @Store value of NUMBER to its address in memory.
    STRB R8, [R7]        @Store value of ERROR to its address in memory.

    _SHIFT_AND_ADD:
    LSL R3, R3, #1       @Left Shift (logical) by 1 bit. This will make way to set or reset
    AND R1, R1, #0x1     @Clear everything except LSB. This is the bit we desire. Either 0 or
    ORR R3, R3, R1       @OR with shifted contents of R3 to set or reset LSB.
    CMP R9, #8           @Are we done with count?
    BNE _LOOP            @If not, then loop through again.

    MOV R8, #0x00        @Clear ERROR
    STRB R8, [R7]        @Store desired value of ERROR to its address in memory.
    STRB R3, [R6]        @Store desired value of NUMBER to its address in memory.

@ Program ends here
_END:
    .end

00001064:00000000
00000008
00000010
```

RegistersView:

Register	Value
R0	00001078
R1	00000001
R2	00000000
R3	000000d5
R4	00000000
R5	00000000
R6	00001078
R7	00001080
R8	00000000
R9	00000008
R10 (s1)	00000000
R11 (fp)	00000000
R12 (ip)	00000000
R13 (sp)	00011400
R14 (lr)	00000000
R15 (pc)	00001064

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

MemoryView:

Address	Value
00001018	E2899001
0000101C	E4D01001
00001020	E3510030
00001024	0A000006
00001028	E3510031
0000102C	0A000004
00001030	E0233003
00001034	E3A080FF
00001038	E5C63000
0000103C	E5C78000
00001040	EA000007

SCREENSHOT – TEST 2

The sample output screenshot for the logic is as below for input (which is essentially bad case)

STRING:

.ascii " 11078101"

NUMBER:

.byte 0x0

ERROR:

.byte 0xFF

The screenshot displays the ARM Simulator interface. The main window is titled 'CodeView' and shows assembly code for a program. The code is organized into sections: `.global _END`, `_MAIN:`, `_LOOP:`, `_SHIFT_AND_ADD:`, and `_END:`. Comments explain the purpose of each instruction, such as loading input string and output addresses, clearing registers, and performing loop operations. The `RegistersView` window on the left shows the current state of the registers. The `CPSR Register` section indicates the processor mode is `System`. The `MemoryView` window at the bottom shows the memory layout, with addresses and corresponding data values.

RegistersView:

Register	Value
R0	:00001074
R1	:00000037
R2	:00000000
R3	:00000000
R4	:00000000
R5	:00000000
R6	:00001078
R7	:00001080
R8	:000000ff
R9	:00000004
R10 (s1)	:00000000
R11 (fp)	:00000000
R12 (ip)	:00000000
R13 (sp)	:00011400
R14 (lr)	:00000000
R15 (pc)	:00001064

CPSR Register:

- Negative (N): 0
- Zero (Z): 0
- Carry (C): 1
- Overflow (V): 0
- IRQ Disable: 1
- FIQ Disable: 1
- Thumb (T): 0
- CPU Mode: System

MemoryView:

Address	Value
00001018	E2899001
0000101C	E4D01001
00001020	E3510030
00001024	0AFFFFFFE
00001028	E3510031
0000102C	0AFFFFFFE
00001030	E0233003
00001034	E3A080FF
00001038	E5C63000
0000103C	E5C78000
00001040	EAF7FFFE
00001044	E1A03083
00001048	E2011001
0000104C	E1933001
00001050	E3590008
00001054	1AFFFFFFE
00001058	E3A08000
0000105C	E5C78000
00001060	E5C63000

EXERCISE – CONVERT A GIVEN EIGHT ASCII CHARACTERS TO AN 8-BIT BINARY NUMBER.

Convert a given eight-digit packed binary-coded-decimal number in the BCDNUM variable into a 32-bit number in a NUMBER variable.

The logic is of two types. One inherent from ARM and one hand computed.

First is that ARM assembly inherently stores number as hex. So, no extra effort is needed. Just store it after reading back to destination.

Second is convert the packed BCD (i.e. decimal) to hex. This can be achieved using positional logic.

Using the first one for ease and optimal coding.

SCREENSHOT – TEST 1

The sample output screenshot for the logic is as below for input (which is essentially bad case)

BCDNUM:

```
.word 92529673
```

NUMBER:

```
.word 0x0
```

R0 :00001018
R1 :0583e409
R2 :00000000
R3 :00000000
R4 :00000000
R5 :00000000
R6 :0000101c
R7 :00000000
R8 :00000000
R9 :00000000
R10 (s1):00000000
R11 (fp):00000000
R12 (ip):00000000
R13 (sp):00011400
R14 (lr):00000000
R15 (pc):00001010

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

```
@ $Header: $
@ Guide: Prof. Madhumutyam IITM, PACE
@
@ when      who      what, where, why
@ -----
@ 12 Nov 2019  Swapneel Pimparkar  (Bengaluru) First Draft
@=====

@ -----
@ Logic: (1) There are two ways. First is that ARM assembly inherently stores
@ number as hex. So, no extra effort is needed. Just store it after
@ reading back to destination.
@ (2) Second is convert the packed BCD (i.e. decimal) to hex.
@ This can be achieved using positional logic -
@ something like - di * 10 + di + 1.
@
@ NOTE: Because there was no special instruction, using first (and easy) method.
@ -----

_PROG_DATA:
.data

@TEST 1 : This is good case.
BCDNUM:
00001018:0583E409 .word 92529673
NUMBER:
0000101C:00000000 .word 0x0

.text
.align 2
.global _MAIN
.global _END

@ Program starts here
_MAIN:
00001000:E59F0008 LDR R0,=BCDNUM @Load the address of input BCDNUM into R0
00001004:E59F6008 LDR R6,=NUMBER @Load the address of output NUMBER into R6
00001008:E5901000 LDR R1, [R0]
0000100C:E5861000 STR R1, [R6]
_END:
00001010:00000000 .end
:00000004
```

00001018	0583E409	0583E409	81818181	81818181	81818181	81818181	81818181	81818181	81818181
0000105C	81818181	81818181	81818181	81818181	81818181	81818181	81818181	81818181	81818181