EE2016 - MICROPROCESSORS

LAB #6 B44

EE21b098 : Sailendra naga kumar

EE21b015: Ananya das

CONTENTS

Particular Pg No

Aim

To

(a) learn advanced ARM instructions, conditional execution etc

(b) go through example programs in Welsh and

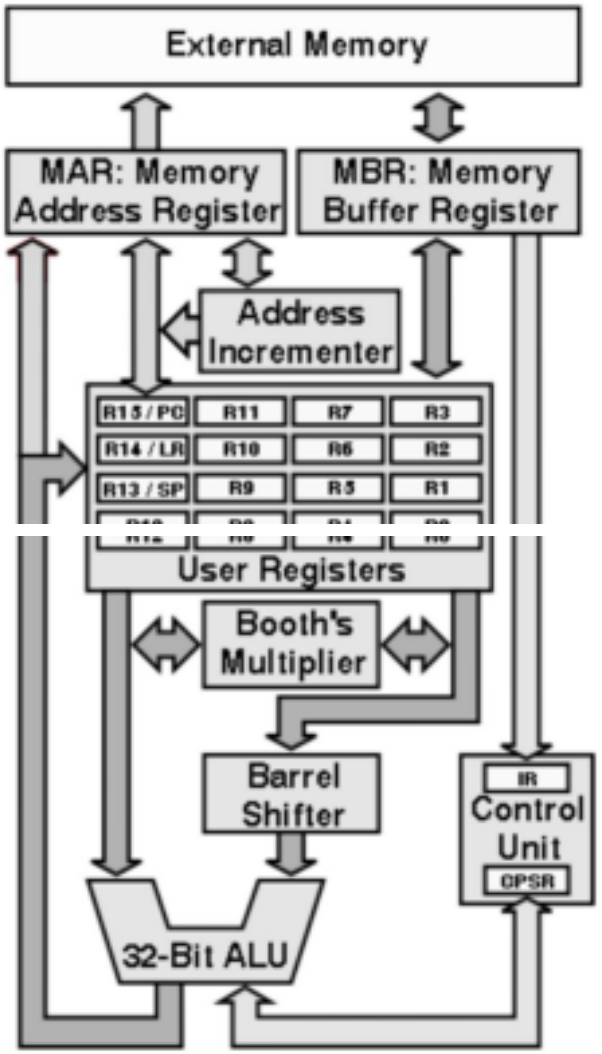
(c) write assembly language programs for the given set of problems at the end of this document.

Equipment Required

The list of equipment, components required are:

1. A PC with Window OS
2. KEIL Micro-vision V5 IDE for ARM

ARM Processor



Review of ARM Instruction Sets

Operation Mnemonic	Meaning	Operation Mnemonic	Meaning
ADC	Add with Carry	ORR	Logical OR
ADD	Add	RSB	Reverse Subtract
AND	Logical AND	RSC	Reverse Subtract with Carry
B	Unconditional	SBC	Subtract with Carry

	Branch		
Bcc	Branch on Condition	SMLAL	Mult Accum Signed Long
BIC	Bit Clear	SMULL	Multiply Signed Long
BL	Branch and Link	STM	Store Multiple
CMP	Compare	STR	Store Register (Word)
EOR	Exclusive OR	STRB	Store Register (Byte)
LDM	Load Multiple	SUB	Subtract
LDR	Load Register (Word)	SWI	Software Interrupt
LDRB	Load Register (Byte)	SWP	Swap Word Value
MLA	Multiply Accumulate	SWPB	Swap Byte Value
MOV	Move	TEQ	Test Equivalence
MRS	Load SPSR or CPSR	TST	Test
MSR	Store to SPSR or CPSR	UMLAL	Mult Accum Unsigned Long
MUL	Multiply	UMULL	Multiply Unsigned Long
MVN	Logical NOT		

Mnemonic	Condition	Mnemonic	Condition
CS	Carry S et	CC	Carry Clear
EQ	Equal (Zero Set)	NE	Not Equal (Zero Clear)
VS	Overflow Set	VC	Overflow Clear
GT	Greater Than	LT	Less Than
GE	Greater Than or Equal	LE	Less Than or Equal
PL	Plus (Positive)	MI	Minus (Negative)
HI	Higher Than	LO	Lower Than (aka CC)
HS	Higher or Same (aka CS)	LS	Lower or Same

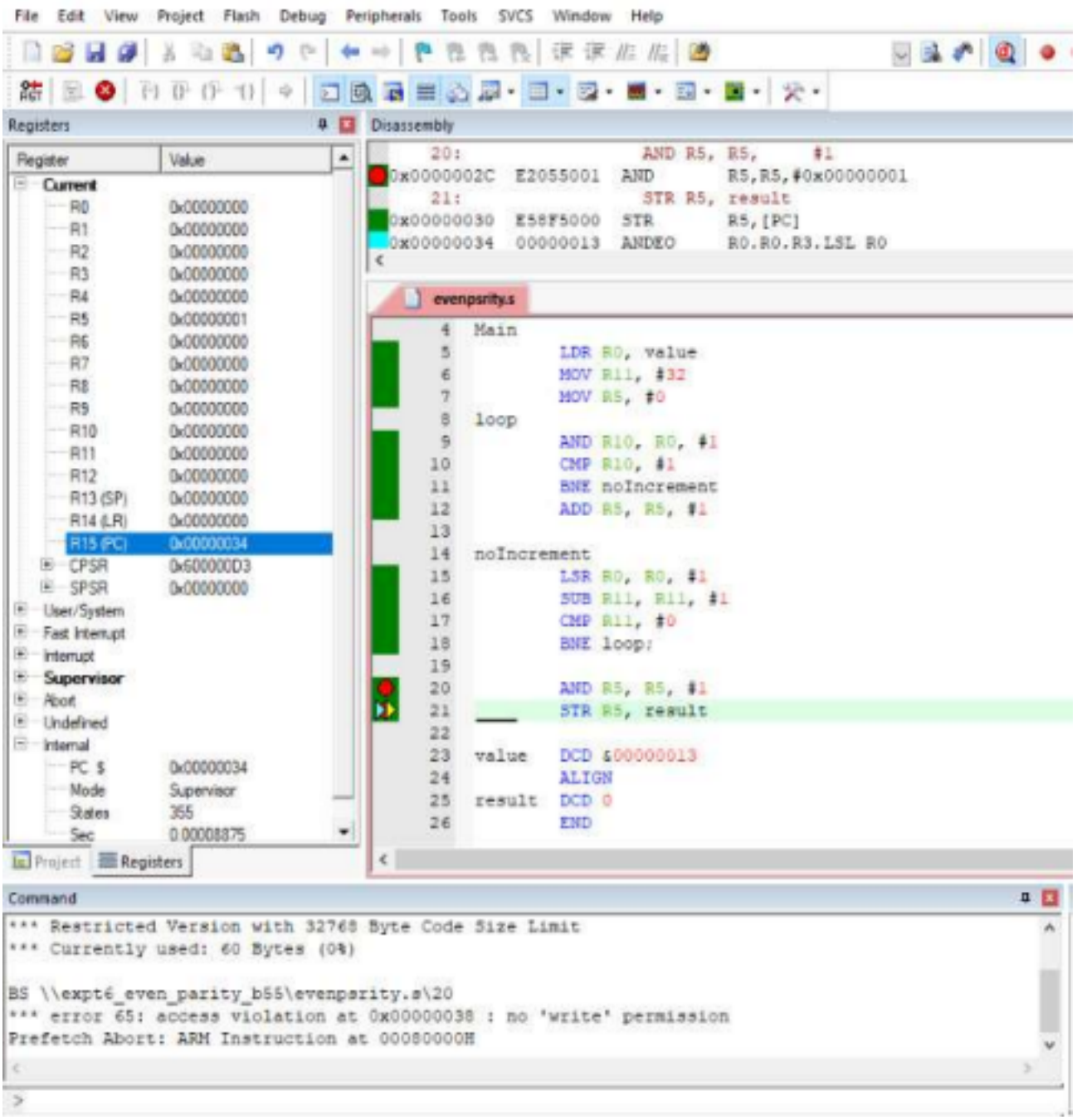
1) Even Parity Bit Generator

Given a 32-bit number, generate an even parity bit for that (32-bit) word. 1.1) CODE

```
AREA program, CODE, READONLY
```

```
 ENTRY
Main
 LDR R0, value
 MOV R11, #32
 MOV R5, #0
loop
 AND R10, R0, #1
 CMP R10, #1
 BNE noIncrement
 ADD R5, R5, #1

noIncrement
 LSR R0, R0, #1
 SUB R11, R11, #1
 CMP R11, #0
 BNE loop;

 AND R5, R5, #1
 STR R5, result

value DCD &00000013
 ALIGN
result DCD 0
 END
```

1.2) CODE EXECUTION



1.3) Explanation

- Input the number into register R0
- Perform Bitwise AND to extract the LSB and check if the bit is 1
- Keep the count of number of 1's in R5
- Left shift R0 and perform the experiment until all 32 bits are checked
- If number of 1's is ODD, the parity bit is 1, if the number of 1's is EVEN, the parity bit is 0. It is generated by performing Bitwise AND of number of 1's with 0x01

1.4) Inference

Here the input number is taken as 0x13 (binary: 10011). The number of 1's is odd. Hence the parity bit is 1. This is generated and stored in R5 as can be seen in the image.

2) ASCII Message

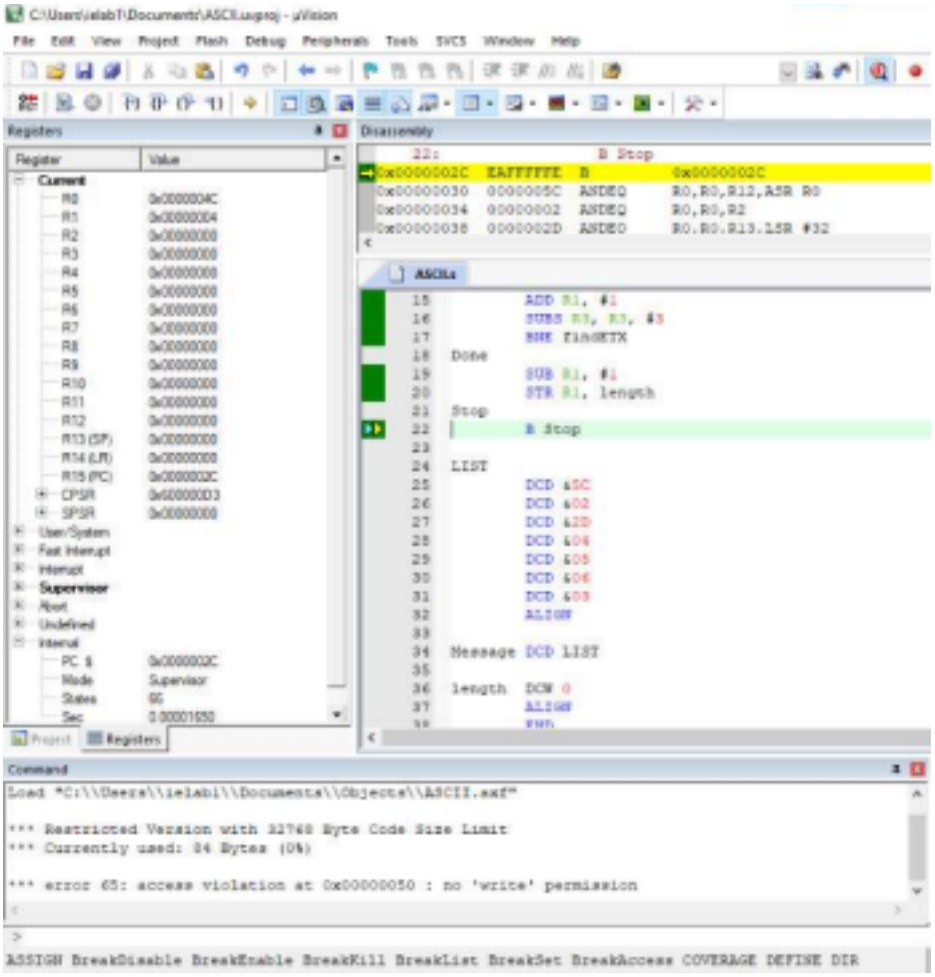
Determine the length of an ASCII message. All characters are 7-bit ASCII with MSB = 0. The string of characters in which the message is embedded has a starting address which is contained in the START variable. The message itself starts with an ASCII STX (Start of Text) character (0x02) and ends with ETX (End of Text) character (0x03). Save the length of the message, the number of characters between the STX and the ETX markers (but not including the markers) in the LENGTH variable

2.1) CODE

```
 TTL wordCount
 AREA Program, CODE, READONLY
 ENTRY
Main
 LDR R0, Message
 EOR R1, R1, R1

findSTX
 LDR R3, [R0], #4
 SUBS R3, R3, #2
 BNE findSTX

findETX
 LDR R3,[R0], #4
 ADD R1, #1
 SUBS R3, R3, #3
 BNE findETX
Done
 SUB R1, #1
 STR R1, length
Stop
 B Stop
```

```
LIST
 DCD &5C
 DCD &02
 DCD &2D
 DCD &04
 DCD &05
                DCD &06
 DCD &03
 ALIGN
Message DCD LIST

length DCW 0
 ALIGN
 END
```

2.2) Code execution



2.3) Explanation

We are given a series of messages which has an ASCII instruction in it. The START of ASCII message is indicated by ASCII STX(0X02) and the END of the ASCII message is indicated by ASCII ETX(0x03)

- o We first find the STX and ETX position of the ASCII message in the series of words. o Execute a loop and find 0x02 in the list
- o Keep a count of number of ASCII instructions in the register R1.
- o Execute a loop and keep incrementing R1 until you find ETX(0x03) in the list

Hence the number of words between 0x02 and 0x03 are found which is the number of ASCII instructions in the given ASCII message.

2.4) Inference

Here, our list consists of

DCD &5C

DCD &02

DCD &2D

DCD &04

```
DCD &05
        DCD &06
DCD &03
```

The 0x02 is found at 2nd position and 0x03 is found at 7th position. Hence the number of ASCII instructions are 4 which are 2D, 04, 05, 06 respectively. Hence they are counted and the number of instructions are stored in register R1 as 4 as shown in the image.

3) Tracking Bit patterns (01111110)

Given a sequence of 32-bit words (sequentially arranged) of length 8 (32 bytes or 256 bits), identify and track special bit patterns of 01111110 in the sequence (if at all appears in the sequence). [This special bit sequence is called “framing bits”, which corresponds to HDLC protocol]. Note that this special bit pattern may start at any bit, not necessarily at byte boundaries. Framing bits, allow the digital receiver to identify the start of the frame (from the stream of bits received).

3.1) CODE

```
AREA Program, CODE, READONLY
 ENTRY
Main
 LDR R0, =Bytes
 LDR R1, =BytesEnd
```

```
 ADD R1, R1, #1
 LDR R3, =0 ; number of sequences   LDR R4, =0 ; progress in sequence
LDR R5, =0 ; amount shifted
 LDRB R7, [R0], #1 ; current byte
 LSL R7, #24 ; put this byte at the left

load_byte
 LDRB R8, [R0], #1 ; load next byte to the right of R7   LSL R8, #16
 ORR R7, R8
 LDR R5, =0

shift
 MOV R9, R7, LSR #31 ; look at leftmost bit in R9   CMP R4, #0
 BEQ startseq
 CMP R4, #7
 BEQ endseq
 CMP R9, #1
          BEQ incstate
```



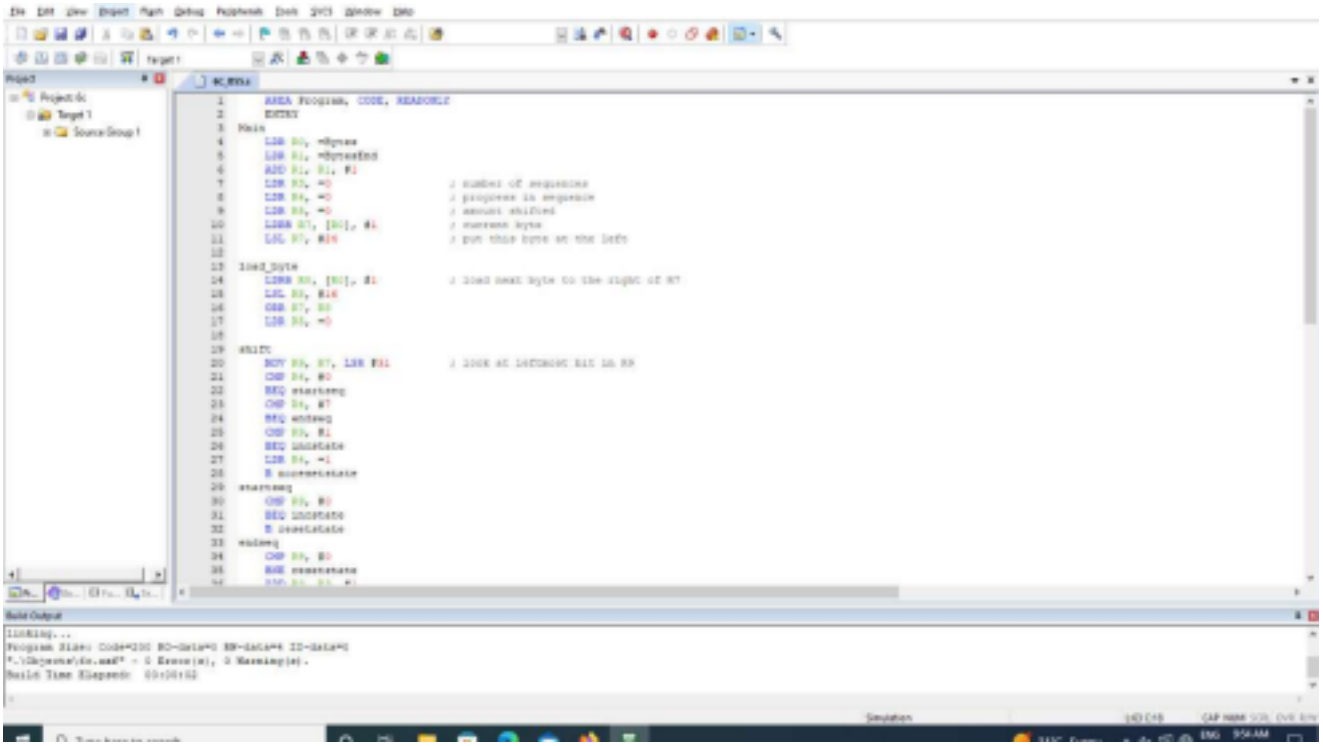
```
          BEQ incstate
 LDR R4, =1
 B noresetstate  startseq
 CMP R9, #0
   BEQ    incstate          B
resetstate  endseq
 CMP R9, #0
 BNE resetstate   ADD R3,
R3, #1   B resetstate
incstate
 ADD R4, R4, #1   CMP R4,
#8
 BNE noresetstate  resetstate
```

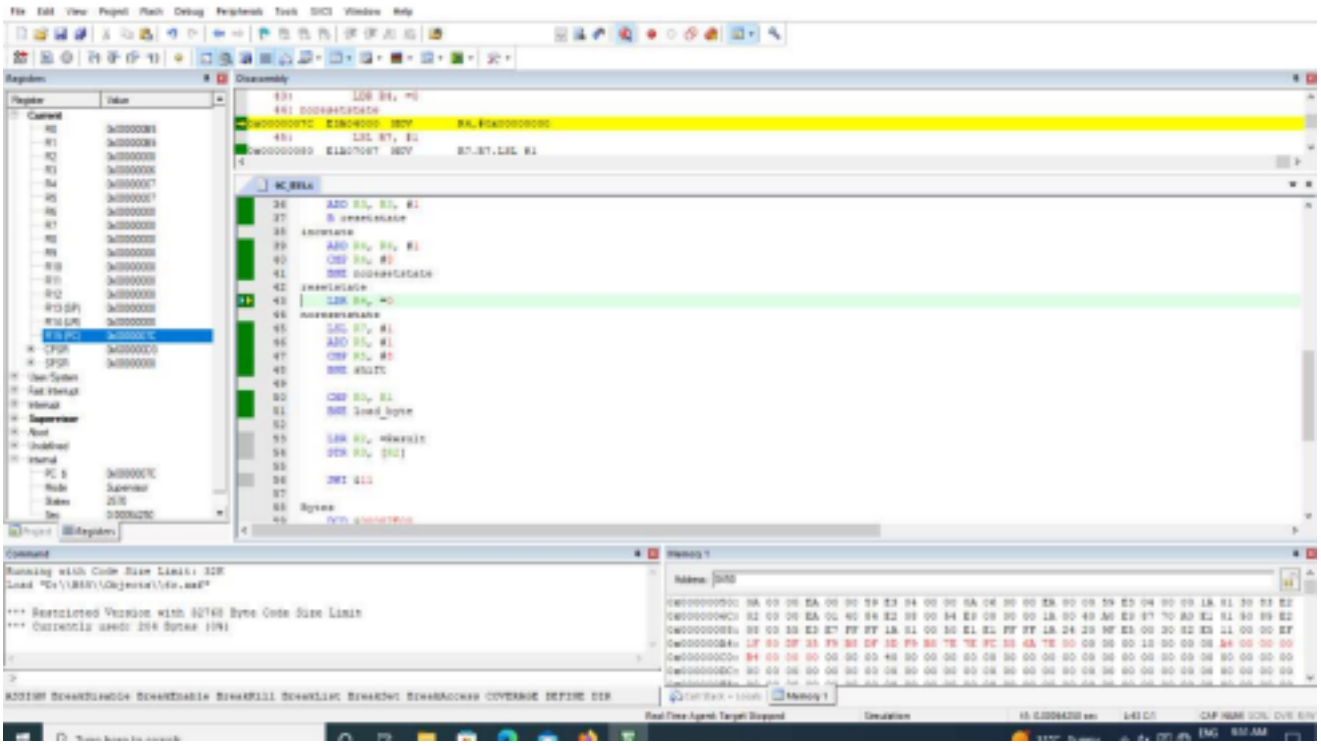
```
 LDR R4, =0
noresetstate
 LSL R7, #1
 ADD R5, #1
 CMP R5, #8
 BNE shift

 CMP R0, R1
 BNE load_byte

 LDR R2, =Result   STR R3,
[R2]

 SWI &11
```



```
 DCD &00007E00
 DCD &007E0000
 DCD &7E000001
 DCD &1007E002
 DCD &00007E00
 DCD &00120000
 DCD &7E000001
 DCD &10000002
BytesEnd DCD 0

 Length DCD (BytesEnd - Bytes)

 AREA DataRAM, DATA, READWRITE
Result DCD 0
 align

 END
```

3.2) Code execution





3.3) Explanation

We are given a list of 8 32-bit words which has 01111110 series in it. We execute a programme to count the number of occurrences of this series in our given list.

- Assign R3 as the register which stores the number of occurrences of

01111110 o Load the first byte into R7 and check each bit by executing a loop

o Keep a track of number of bits that occur correctly in the sequence in register R4 o If the 8 bits in the sequence are correctly obtained, increment R3 and clear R4. Repeat the procedure until all the 256 bits are checked. Hence the number of occurrences is stored in R3

3.4) Inference

We have input the 32-bit 8 length list as shown below

```
DCD &00007E00
DCD &007E0000
DCD &7E000001
DCD &1007E002
DCD &00007E00
DCD &00120000
DCD &7E000001
DCD &10000002
```

This has the series 01111110 6 times in it. Hence the occurrence has been stored in register R3 as shown in the figure.