|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | ***Portland State University*** [Maseeh College of Engineering and Computer Science](http://pdx.edu/cecs/) 1559170_300.jpg | | ECE 371 - PROJECT REPORT | | Project 1: ARRAYS AND PROCEDURES | | **11/15/2014** | |
|  |
| **Instructor: Douglas V.Hall**  **TA: Leela Yadlapalli** |
| Students: Hai Dang Hoang  PSUID: 946749560  Email: [danghai@pdx.edu](mailto:danghai@pdx.edu) |

1. **Introduction:**

This program stimulate one common protocol for transferring data (ATM) Asynchronous

Transfer Mode. ATM computer networks transmit and receive data in 53-byte packets. The first five bytes that are sent out in a packet are a header that contains the destination address. Assume that ATM packet is organize Big-Endian order. This program will compute the checksum for the first four bytes of the header and compare it with the checksum stored in the fifth byte of the header. If the computed checksum is different from that stored in the header, execution should return to the mainline immediately with an error code of 1 in the appropriate register. If the checksum computed for the first four byte is correct, the procedure should copy the 48-byte data block from the packet to a new buffer (array) in Little-Endian order and return to the mainline with an error code of 0 in the appropriate register.

1. **Standard Program Structure and Algorithm:**

**2.1 a) Mainline:**

Initialize the stack

Initialize pointer to Packets array

Initialize pointer to StorePackets array

Call procedure Checksum ( Return R0 = 1 Error, R0 = 0 No Error)

Compare R0 ( Error) with 0

If R0 equals 0

Call MoveData Procedure

End Program

**b) Flowchart Mainline:**

Mainline:

Start

Initial Stack

Initial Pointer

checksum

MoveData

End

ReturnValue = 0

No

Yes

**2.2 a) Checksum procedure:**

Save used registers and return address on stack (push R3-R5, R14 on stack)

Initialize loop counter (R3) equals 4

Initialize sum of the first 4 bytes (R5) equals 0

Repeat

Read one byte value from Package array and increment pointer by one byte

Add the byte to Sum register ( R5 )

Decrement loop counter

Until 4 values done

Read byte value (the 5th byte of from Package array) => Checksum byte and increment pointer

Operate AND of Sum with 0xFF to only get 8 bits

Compare Checksum byte with Sum

If Sum equals Checksum

R0 ( Error code ) = 0 ( No error and allow to move data)

Else

R0 ( Error code) = 1

Restore registers from stack and return to mainline

**b) Flowchart Checksum procedure:**

Start

Store Register into Stack and R14

Initial Pointer

Initial Counter 4

Initial Sum = 0

Counter = 0

No

Yes

Add to Sum

Decrement Counter

Load Checksum byte

AND it with 0xFF

CMP Checksum == Sum

No Yes

R0 = 1

R0 = 0

Restore Register from Stack

And it with 0xFF

Go to mainline

And it with 0xFF

End

**2.3a) MoveData procedure:**

Save used registers and return address on stack (push R3-R5, R14 on stack)

Initialize loop first counter for 12 times ( 12 x 4 = 48 bytes data)

Repeat

Initial second counter equals 4 ( 1 word) @ StoreStack @

Repeat

Read one byte value from Package array and increment pointer by one byte

Decrement Stack pointer by one

Store value into stack

Decrement second counter by one

Until 4 values done

Initial counter 2 equals 4 again @ Backup Stack @

Repeat

Pop 1 byte value out of stack

Store it into StorePackets and increment pointer of StorePacket by one

Increment stack pointer by one byte

Decrement the second counter by one

Until 4 values done

Until 12 times done

Restore registers from stack and return to mainline

1. **MoveData procedure:**

Start

Store Register into Stack and R14

Initial Pointer

Initial counter = 12

Counter = 0

Yes

No

Initial counter 2 =4

No

Counter 2 = 0

Get value byte

Decrement Stack Pointer

Store into Stack

Decrement Counter 2

Yes

Initial counter 2 =4

Counter 2 = 0

No

Pop value from stack

Yes

End

Go to mainline

And it with 0xFF

Restore Register from Stack

And it with 0xFF

Increment Stack Pointer

Store into StorePacket

Decrement Counter 2

Decrement Counter

1. **Testing and Debugging:**

There are two parts that I need to test in order to make sure the program work correctly. The first part is Checksum procedure and the second part is MoveData procedure.

**3.1 Checksum procedure:**

There are three situations that take place in the program for testing:

+ Sum of the first four bytes exactly equals the checksum in the Packets array.

+The lowest byte of the sum equals the checksum.

+The lowest byte of the sum is different from the checksum

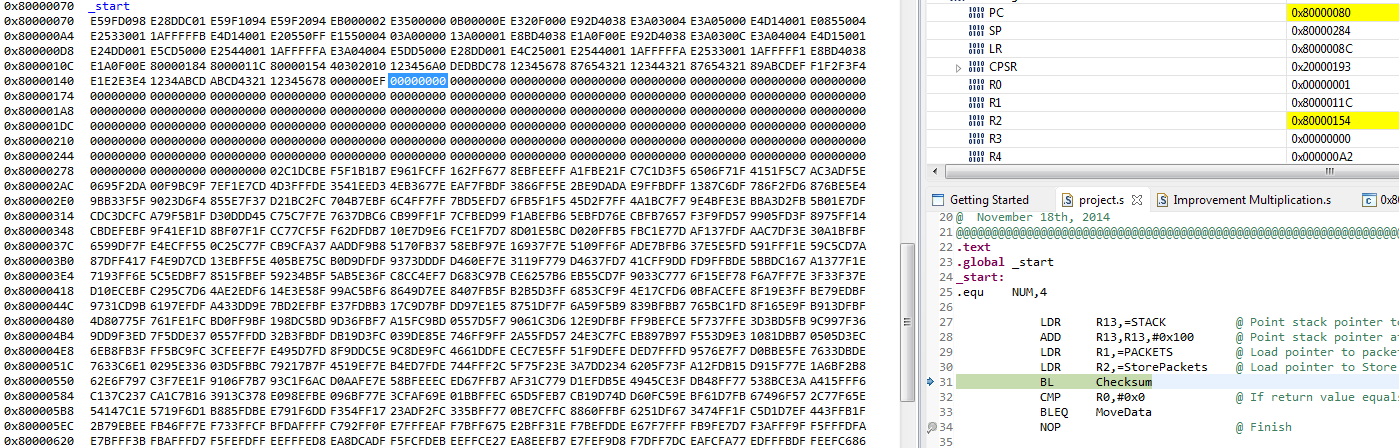
1. **Sum of the first 4 bytes exactly equals the checksum:**

Data in StorePackets Array (0x80000154)

( Address: 0x800011C)

Data in Packets Array

( Address: 0x800011C)



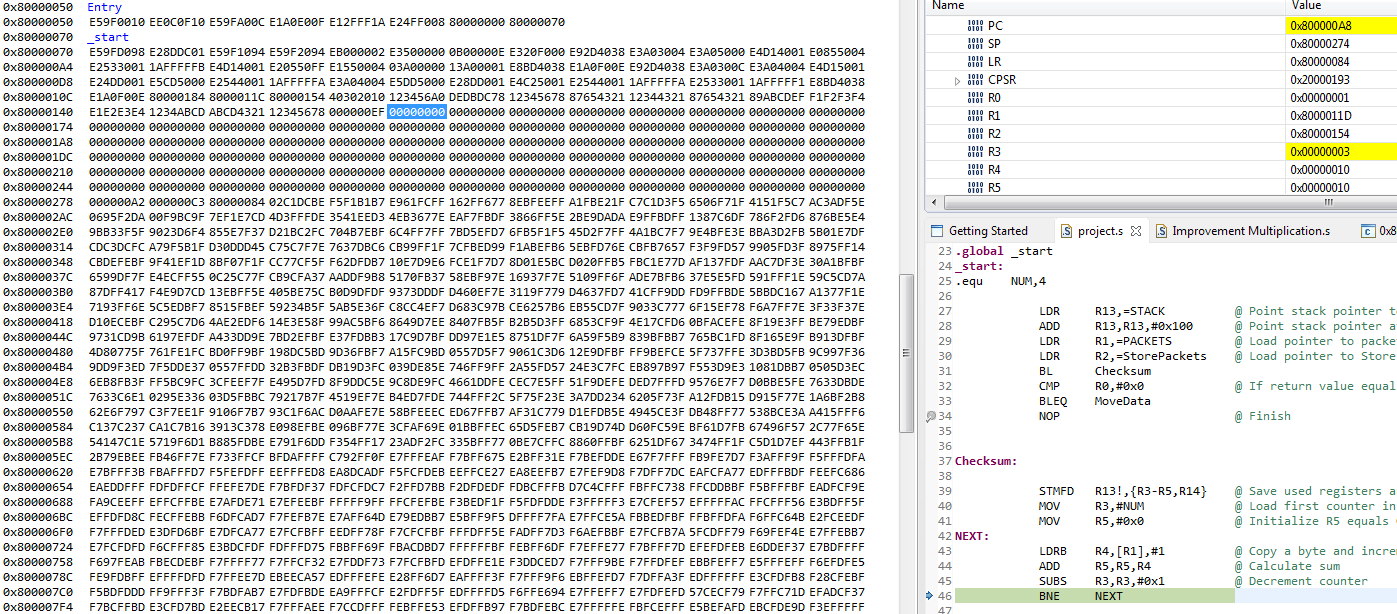
R3: Counter decrement

R4: Store value from packets array

R5: Sum

1st byte of the Packets array

Checksum (5th byte)

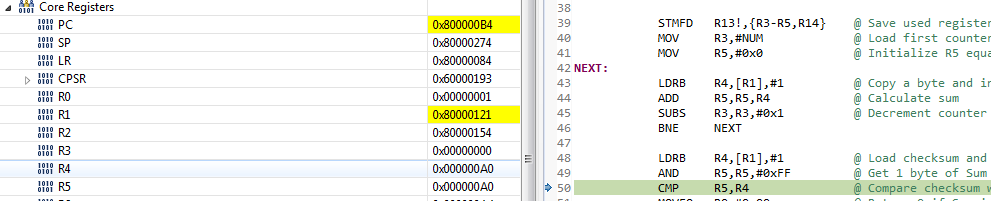


The program will add first 4 bytes and stored in R5 when R3 counter equals 0.

R5 = 0x10 + 0x20 + 0x30 + 0x40 = 0xA0

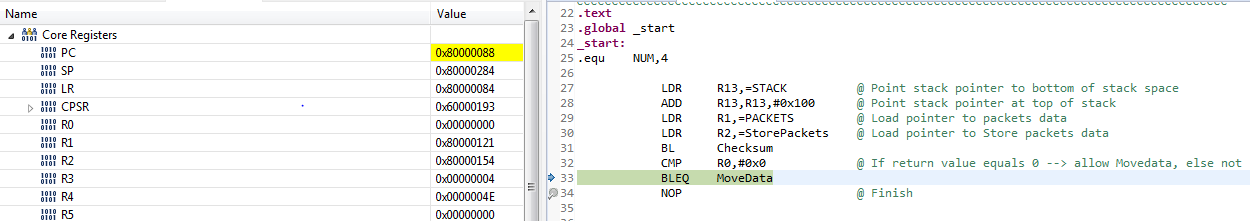
Lowest byte of the sum of the first 4 bytes (R5 store) and the checksum in the 5th byte of the Packets ( R4 store)

R3 stored 0 indicating that all 4 bytes are added



Since the lowest byte of the sum of is A0 which equals the checksum which is the 5th byte of the Packet array. R0 store the error code and the procedure MoveData will be processed.

Error code of 0 is store at R0



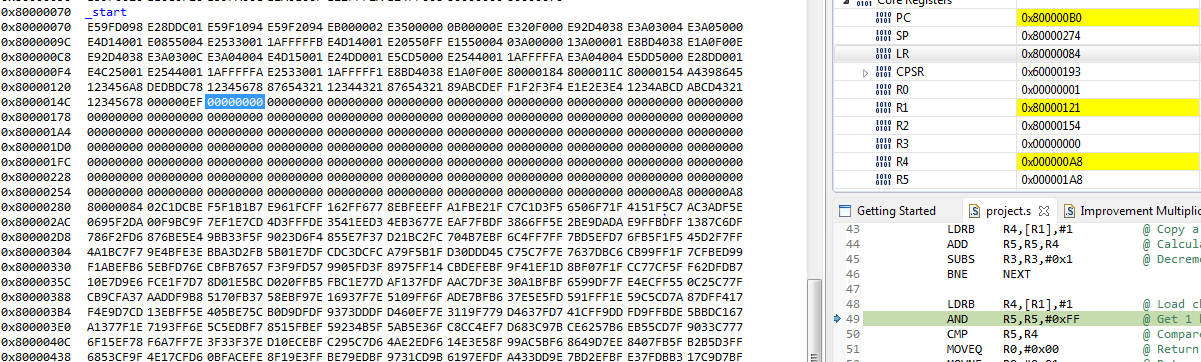
Because R0 stores error code which equals 0, the procedure MoveData will be processed. Specifically, the instruction “BLEQ MoveData” will be processed. Therefore, the checksum operates correctly.

1. **Lowest byte of the sum equals the checksum**

In this situation, we need to have AND operation after computing the sum of the first 4 bytes to get the 8 bits. Since the lowest byte equals the checksum, error code of 0 is stored in R0 and the procedure MoveData still is processed as same as in the first situation.

The sum of those 4 bytes is 1A8

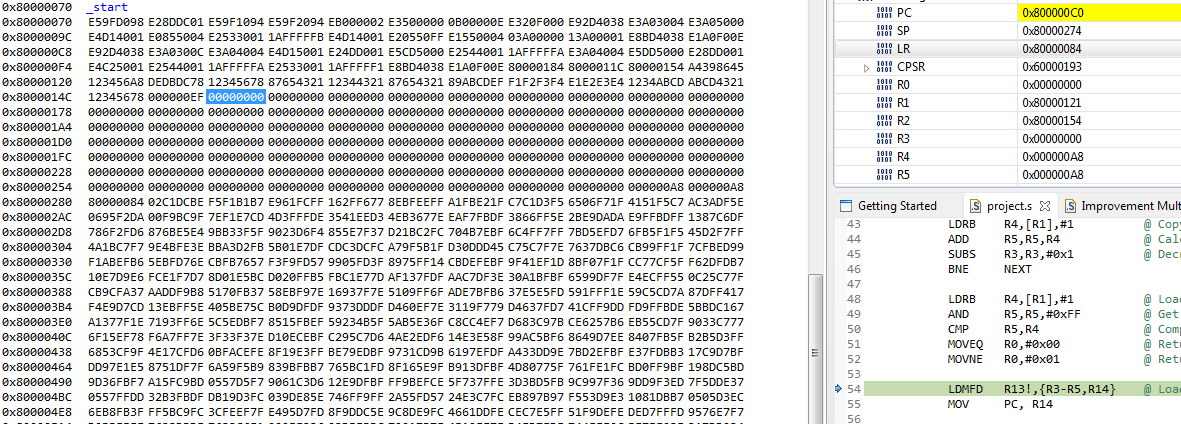
The first 4 bytes are 0x45, 0x86, 0x39 and 0xA4



Sum of the first 4 byte is 1A8. We need to do AND operation between the sum and 0xFF and store the result back to R5. R5 still stores A8 which is the lowest byte of the sum of the first 4 bytes.

R0 still stores Error code which equals 0

The result stored in R5 after the AND operation is the same checksum’s value stored in PACKAGE array ( 0xA8)

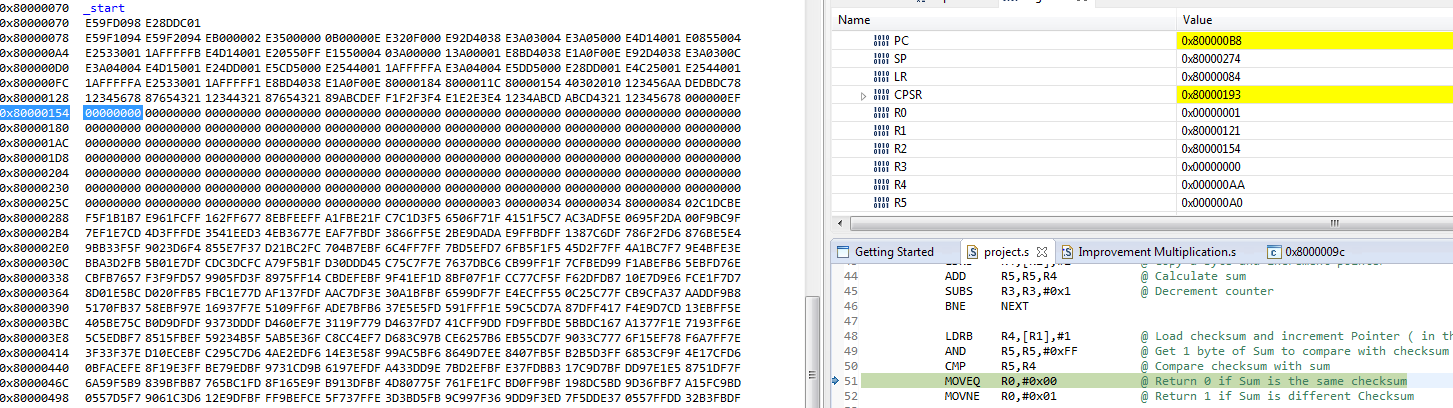


Sum of the first 4 bytes are still equal the checksum. Therefore, the error code of 0 will be stored in R0 and the procedure MoveData will be processed.

1. **The lowest byte of the sum is different from the checksum**

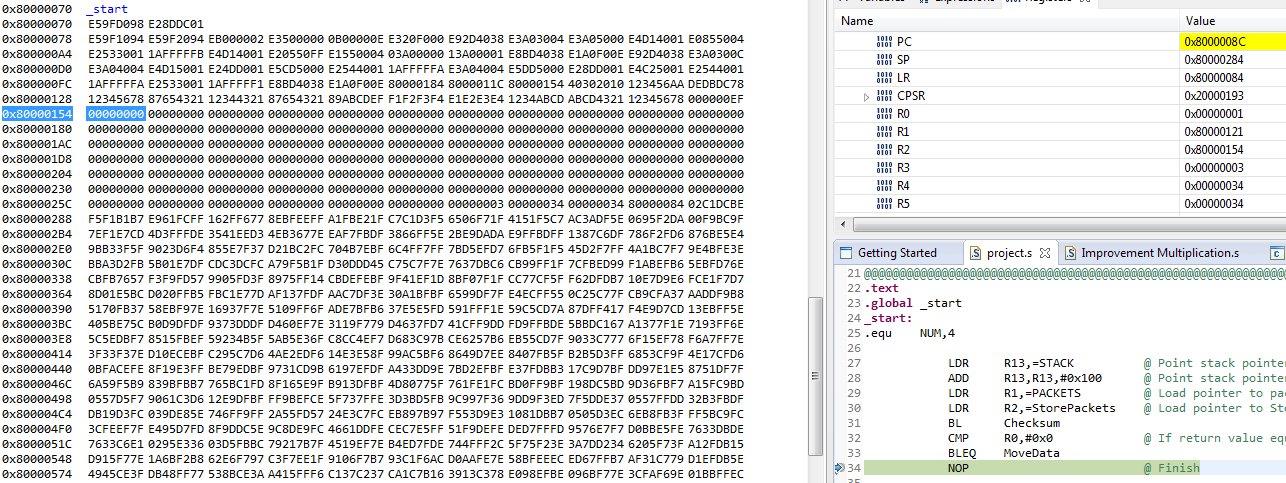
In this situation, since the lowest byte of the sum of the first 4 bytes is different from checksum’s value which is stored in the 5th byte of the PACKAGE array, the execution will return to the mainline with the error code of 1 in resistor R0 and the program finish.

Sum of the first 4 bytes are store in R5 is different from the checksum stored in R4



48 data bytes are not transfer because the error code stored in R0 is 1

Error code 1 is stored at R0



**3.2 MoveData procedure:**

**a) Discussion:**

Big Endian Byte Order: The most significant byte ( the “big end” ) of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory.

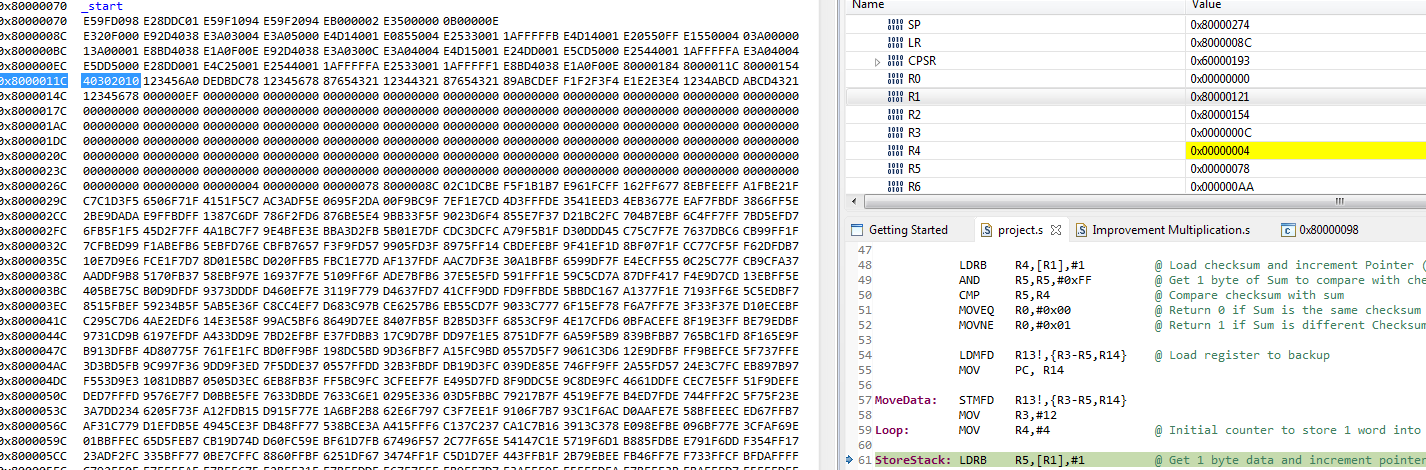
Little Endian Byte Order: The least significant byte ( the “Little end”) of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory.

In my algorithm, I use stack to convert Big endian byte order to Little Endian byte order because Stack operates LIFO ( Last in First out). Specifically, I push 4 bytes from Packet into stack ( Label: StoreStack in the code), and then I pop 4 bytes from the stack and store into StorePacket ( Label: BackupStack). Next, I move to other words in Packet

R3 is the first counter 12 times.

R4 is the second counter 4 times ( Store value into Stack)

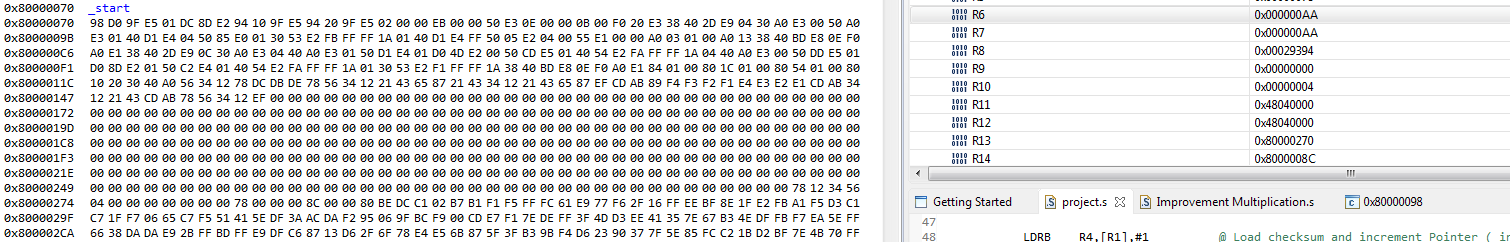
12x4= 48 data bytes



Push 4 values from Packets into Stack, and then restart the second counter equal 4 for popping value out of stack in order to store StorePacket array.

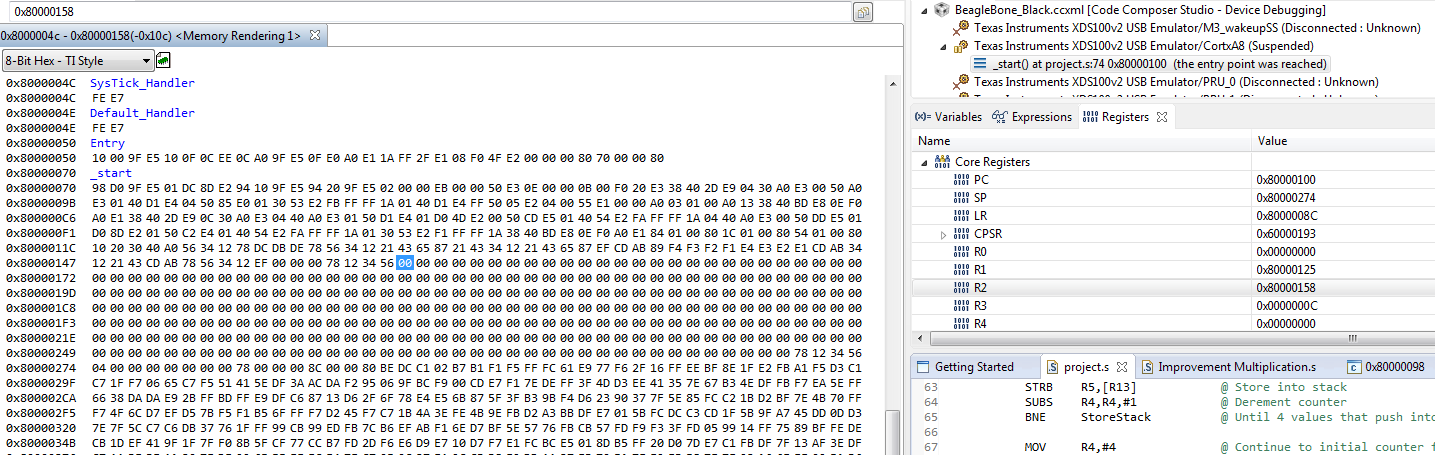
4 bytes data after pushing into Stack. The order will be reversed after popping out of stack

4 bytes data in Packet



Initial address of R2

4 value bytes are stored into R2 with Little Endian order

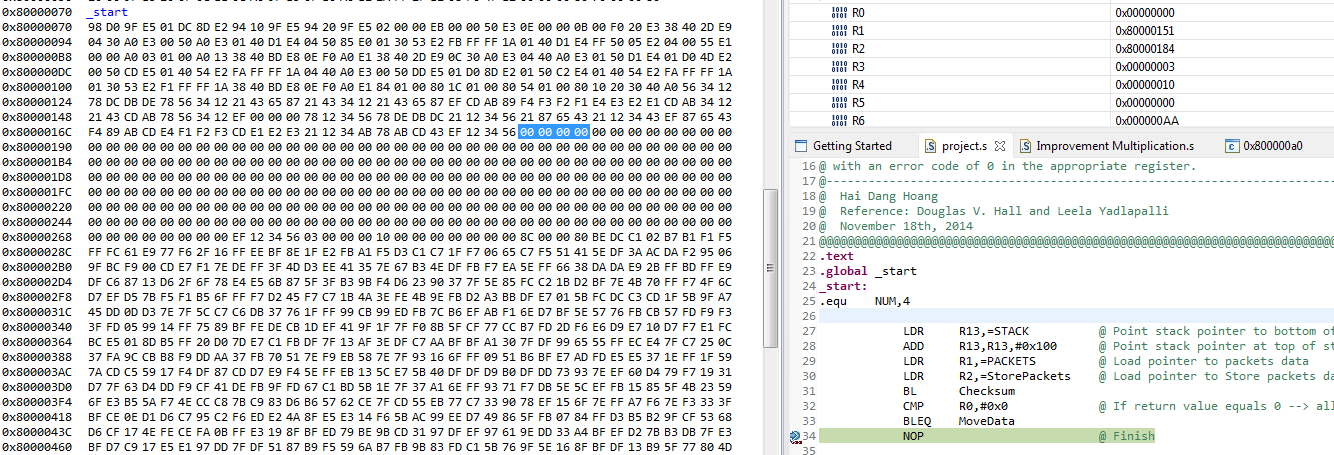


After successfully copying the 1st word, the first counter R3 decrement by one and do the same until 12 times done. Therefore, the program transfers 48 bytes (or 12 words) successfully from Packets.

Final results of the StorePacket after all 48 data bytes are transfered

First byte data of Packets

R0 = 0 ( No error code)



1. **Source code:**

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ ECE 371 – PROGRAMMING PROJECT I

@------------------------------------------------------------------------------------

@ ARRAY AND PROCEDURES

@ This program stimulate one common protocol for transferring data (ATM) Asynchronous

@ Transfer Mode. ATM computer networks transmit and receive data in 53-byte packets.

@ The first five bytes that are sent out in a packet are a header that contains the

@ destination address. Assume that ATM packet is organize Big-Endian order. This

@ program will compute the checksum for the first four bytes of the header and

@ compare it with the checksum stored in the fifth byte of the header. If the

@ computed checksum is different from that stored in the header, execution should

@ return to the mainline immediately with an error code of 1 in the appropriate

@ register. If the checksum computed for the first four byte is correct, the

@ procedure should copy the 48-byte data block from the packet to a new buffer @ (array) in Little-Endian order and return to the mainline with an error code of 0

@ in the appropriate register.

@------------------------------------------------------------------------------------

@ Hai Dang Hoang

@ Reference: Douglas V. Hall and Leela Yadlapalli

@ November 18th, 2014

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**.text**

**.global** \_start

**\_start:**

.equ NUM,4

LDR R13,=STACK @ Point stack pointer to bottom of stack space

ADD R13,R13,#0x100 @ Point stack pointer at top of stack

LDR R1,=PACKETS @ Load pointer to packets data

LDR R2,=StorePackets @ Load pointer to Store packets data

BL Checksum

CMP R0,#0x0 @ R0 = 0 -> Allow to move data

BLEQ MoveData

NOP @ Finish

**Checksum:**

STMFD R13!,{R3-R5,R14} @ Save used registers and Return address on stack

MOV R3,#NUM @ Load first counter in R3

MOV R5,#0x0 @ Initialize R5 equals 0(R5 Register store sum )

**NEXT:**

LDRB R4,[R1],#1 @ Copy a byte and increment pointer

ADD R5,R5,R4 @ Calculate sum

SUBS R3,R3,#0x1 @ Decrement counter

BNE NEXT

LDRB R4,[R1],#1 @ Load checksum and increment Pointer

@ in this time, pointer point to data

AND R5,R5,#0xFF @ Get 1 byte of Sum to compare with checksum

CMP R5,R4 @ Compare checksum with sum

MOVEQ R0,#0x00 @ Return 0 if Sum is the same checksum

MOVNE R0,#0x01 @ Return 1 if Sum is different Checksum

LDMFD R13!,{R3-R5,R14} @ Load register to backup

MOV PC, R14 @ Return to mainline

**MoveData:**

STMFD R13!,{R3-R5,R14} @ Save used registers and Return address on stack MOV R3,#12 @ Initial the first counter

**Loop:** MOV R4,#4 @ Initial counter to store 1 word into stack

**StoreStack:**

LDRB R5,[R1],#1 @ Get 1 byte data and increment pointer

SUB R13,R13,#1 @ Decrement stack

STRB R5,[R13] @ Store into stack

SUBS R4,R4,#1 @ Derement counter

BNE StoreStack @ Until 4 values that push into stack

MOV R4,#4 @ Initial counter for popping 1 word out of stack

**BackupStack:**

LDRB R5,[R13] @ Pop 1 byte data from stack

ADD R13,R13,#1 @ Increment stack

STRB R5,[R2],#1 @ Store 1 byte into storePacket

SUBS R4,R4,#1 @ Decrement counter

BNE BackupStack @ Until 4 values that pop out of stack

SUBS R3,R3,#1 @ End while until 12 times done

BNE Loop

LDMFD R13!,{R3-R5,R14} @ Load register to backup

MOV PC,R14 @ Return to mainline

**.data**

**PACKETS:** @Big Endian

**.word** 0x40302010, 0x123456A0 @ 4 header bytes, the checksum and the three data

**.word** 0xDEDBDC78, 0x12345678, 0x87654321, 0x12344321 @ data bytes

**.word** 0x87654321, 0x89ABCDEF, 0xF1F2F3F4, 0xE1E2E3E4 @ data bytes

**.word** 0x1234ABCD, 0xABCD4321, 0x12345678, 0xEF @ data bytes

**StorePackets:** @ Little Endian

**.word** 0x0, 0x0, 0x0, 0x0, 0x0, 0x0

**.word** 0x0, 0x0, 0x0, 0x0, 0x0, 0x0

**STACK:** .rept 256

**.byte** 0x00

.endr

.end

1. **Conclusion:**

Based on the results from testing and debugging, the project is done. Specifically, the program calculates the lowest byte of the sum of the first four bytes then compares it with the fifth byte stored in Packets. If they are equal, the program will start moving the data to the new packet with the error code of 0. If they are different, the data bytes are not allow to move and error code display 1. This project will give me an introduction to one part of simple, low-level packet processing and some experience using many of the basic techniques in ARM assembly language programming. Specifically, I gain experience with a procedure call and return, passing parameters to procedures, processing arrays with pointer, auto-increment and auto-decrement addressing, masking, big-endian and little-endian data storage, conditional branches, converting an algorithm to assembly language, using the stack, and the program development cycle.

1. **Signed Statement**

I developed and wrote this program by myself with no help from anyone except the instructor and the T.A. and I did not give help to anyone else