

Microprocessor and Assembly Language Lab 04

Kazi Shadman Sakib, FH-97

March 04, 2022

1 Task i

1.1 Explaining two directives: Data and Align with proper example.

Data: The data directive sets data as the current section. The lines that follow will be assembled into the data section. The data section is normally used to contain tables of data or pre-initialized variables.

Example:

```
AREA test_data, DATA, READWRITE, ALIGN=2
X DCB "Hello",0
Y DCB "World",0
```

Align: The ALIGN directive aligns the current location to a specified boundary by padding with zeros or NOP instructions. Use ALIGN to ensure that your data and code is aligned to appropriate boundaries.

Example:

```
AREA test_data, DATA, READWRITE, ALIGN=2
X DCB "Hello",0
Y DCB "World",0
```

The ALIGN directive tells the assembler that the next instruction is word aligned and offset by 2 bytes. DCB is followed by an instruction, use an ALIGN directive to ensure that the instruction is aligned.

2 Task ii

2.1 Detailed Code Explanation

The MRS instruction is used for copying the values of status register from APSR to general purpose register r0, from IPSR to general purpose register r1 and from EPSR to general purpose register r2. We update the flags by comparing value of 5 and 10 in register r3.

2.2 Detailed Description of the Instruction Used to Design the Program.

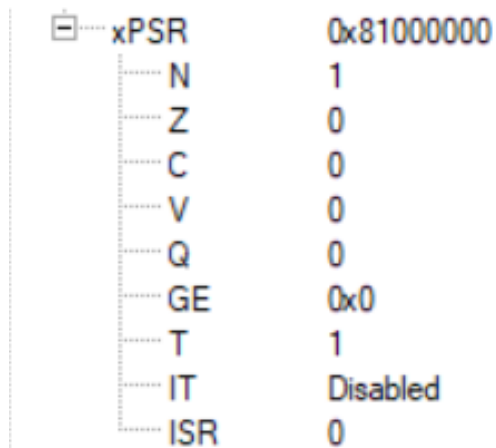
MOV: MOV copies the value specified in the second operand to the the specified register.

CMP: CMP compares the two operands (operand 1 and operand 2) and updates the status flags accordingly.

MRS: MRS copies the contents of a special purpose register to a general purpose register.

2.3 Screenshot of Debugger

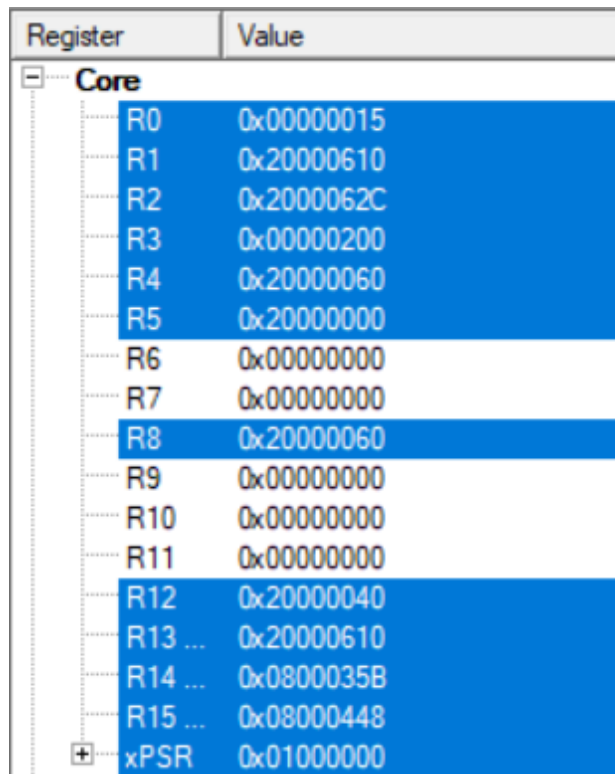
2.3.1 Status of the Status Registers After the Operation



A screenshot of a debugger's status register window. It shows a tree view with 'xPSR' expanded, listing various flags and their values. The flags include N (1), Z (0), C (0), V (0), Q (0), GE (0x0), T (1), IT (Disabled), and ISR (0).

<input checked="" type="checkbox"/> xPSR	0x81000000
N	1
Z	0
C	0
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

2.3.2 After the Code has been Loaded



A screenshot of a debugger's core registers window. It shows a table with two columns: 'Register' and 'Value'. The registers R0 through R15 are listed, along with the xPSR register. The values are hexadecimal. The xPSR register is expanded, showing its value as 0x01000000.

Register	Value
<input checked="" type="checkbox"/> Core	
R0	0x00000015
R1	0x20000610
R2	0x2000062C
R3	0x00000200
R4	0x20000060
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x20000060
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000040
R13 ...	0x20000610
R14 ...	0x0800035B
R15 ...	0x08000448
<input checked="" type="checkbox"/> xPSR	0x01000000

2.3.3 After the Code has been Executed

Register	Value
Core	
R0	0x80000000
R1	0x00000000
R2	0x00000000
R3	0x00000005
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000554
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x0800023F
R15 (PC)	0x08000526
+ xPSR	0x81000000

3 Task iii A

3.1 Detailed Code Explanation

Firstly, We create an area for the variables, and allocate memory for the variables I and S. We load the contents of I and S to the register r1 and r2 respectively. Then we multiply the contents of register r1 with itself to get the square of I and add to the current sum. Then we increment I by 1. Then we check if the value of I is reached, and if not we loop again. When the loop has ended, we store the sum into the memory location of the variable S.

3.2 Detailed Description of the Instruction Used to Design the Program.

DCD: DCD allocates words of memory and defines the initial runtime contents of the memory.

LDR: LDR loads the words to the specified register.

MUL: :MUL multiplies the specified values and places the result in a register.

ADD: :ADD multiplies the specified values and places the result in a register.

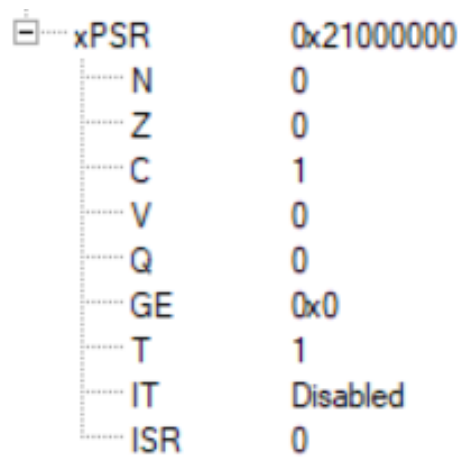
CMP: CMP compares the two operands and updates the status flags.

BLS: BLS branches to specified label if the result of the compared value was lower or same than the other.

STR: STR stores the contents of the specified register to a memory location.

3.3 Screenshot of Debugger

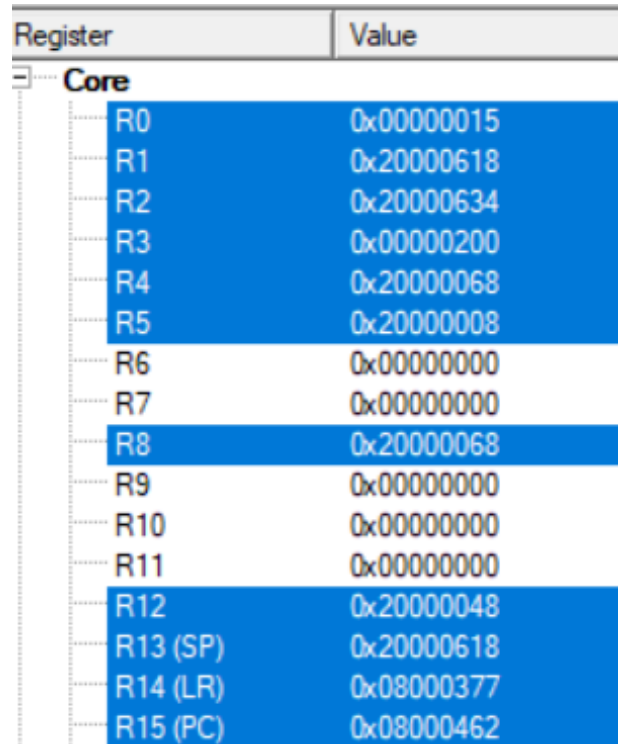
3.3.1 Status of the Status Registers After the Operation



A screenshot of a debugger's status register window. It shows a list of status registers on the left and their corresponding values on the right. The registers are xPSR, N, Z, C, V, Q, GE, T, IT, and ISR. The values are 0x21000000, 0, 0, 1, 0, 0, 0x0, 1, Disabled, and 0 respectively.

xPSR	0x21000000
N	0
Z	0
C	1
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

3.3.2 After the Code has been Loaded



A screenshot of a debugger's core register window. It shows a table with two columns: Register and Value. The registers are R0 through R15, with R13 labeled as SP, R14 as LR, and R15 as PC. The values are 0x00000015, 0x20000618, 0x20000634, 0x00000200, 0x20000068, 0x20000008, 0x00000000, 0x00000000, 0x20000068, 0x00000000, 0x00000000, 0x00000000, 0x20000048, 0x20000618, 0x08000377, and 0x08000462 respectively.

Register	Value
Core	
R0	0x00000015
R1	0x20000618
R2	0x20000634
R3	0x00000200
R4	0x20000068
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x20000068
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000618
R14 (LR)	0x08000377
R15 (PC)	0x08000462

3.3.3 After the Code has been Executed

Register	Value
Core	
R0	0x20000004
R1	0x0000000B
R2	0x00000181
R3	0x00000064
R4	0x00000000
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000590
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000668
R14 (LR)	0x0800025B
R15 (PC)	0x08000548
xPSR	0x21000000

4 Task iii B

4.1 Detailed Code Explanation

We create an area for the variables, and allocate memory for the variables and result. We load the contents of the variable A, B, C into the register r1, r2 and r3 respectively. Then we multiply the contents of each of r1, r2 and r3 to get the squares of the variables, and then we add the contents of r1 and r2 for the sum of their squares. We compare this value against the third square, and if they are equal, we set the value of r5 to 1 and store it to the memory location of the S variable.

4.2 Detailed Description of the Instruction Used to Design the Program.

DCB: DCB allocates bytes of memory and defines the initial runtime contents of the memory.

LDR{B}: LDR loads the words to the specified register. B stands for Byte.

MUL: :MUL multiplies the specified values and places the result in a register.

ADD: :ADD multiplies the specified values and places the result in a register.

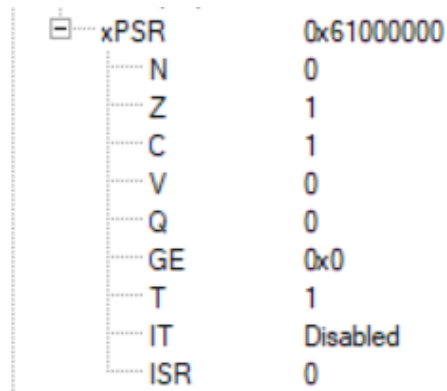
CMP: CMP compares the two operands and updates the status flags.

BEQ: BEQ (short for "Branch if Equal") is the mnemonic for a machine language instruction which branches, or "jumps", to the address specified if, and only if the zero flag is set.

STR: STR stores the contents of the specified register to a memory location.

4.3 Screenshot of Debugger

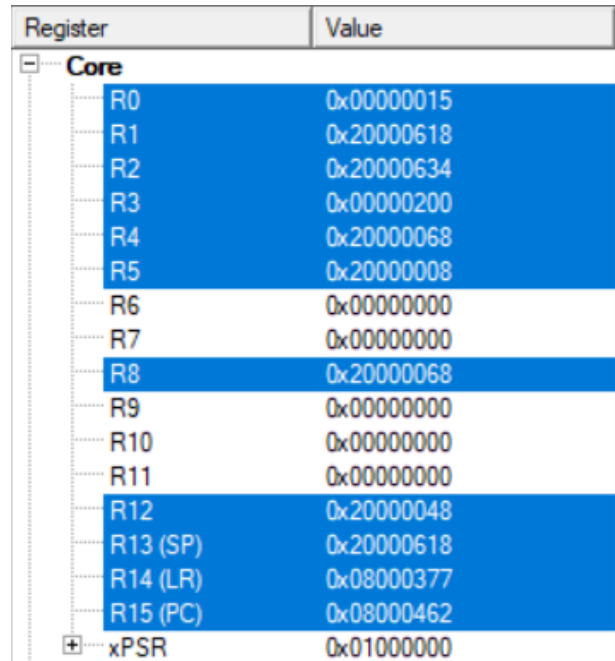
4.3.1 Status of the Status Registers After the Operation



A screenshot of a debugger's status register window. The window is titled 'xPSR' and shows a list of status registers with their corresponding values. The registers are N, Z, C, V, Q, GE, T, IT, and ISR. The values are: N=0, Z=1, C=1, V=0, Q=0, GE=0x0, T=1, IT=Disabled, and ISR=0.

xPSR	0x61000000
N	0
Z	1
C	1
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

4.3.2 After the Code has been Loaded



A screenshot of a debugger's core register window. The window is titled 'Core' and shows a list of registers with their corresponding values. The registers are R0 through R15 and xPSR. The values are: R0=0x00000015, R1=0x20000618, R2=0x20000634, R3=0x00000200, R4=0x20000068, R5=0x20000008, R6=0x00000000, R7=0x00000000, R8=0x20000068, R9=0x00000000, R10=0x00000000, R11=0x00000000, R12=0x20000048, R13 (SP)=0x20000618, R14 (LR)=0x08000377, R15 (PC)=0x08000462, and xPSR=0x01000000.

Register	Value
Core	
R0	0x00000015
R1	0x20000618
R2	0x20000634
R3	0x00000200
R4	0x20000068
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x20000068
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000618
R14 (LR)	0x08000377
R15 (PC)	0x08000462
xPSR	0x01000000

4.3.3 After the Code has been Executed

register	Value
Core	
R0	0x00000001
R1	0x00000009
R2	0x00000010
R3	0x00000019
R4	0x00000019
R5	0x00000001
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x080005A4
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000668
R14 (LR)	0x0800025B
R15 (PC)	0x08000558
xPSR	0x61000000

5 Task iv A

5.1 Detailed Code Explanation

We create an area for the variables, and allocate memory for the strings. We load the address of the first byte of strings X and Y to the registers r0 and r1 respectively. Then we load the current byte of the string X to r2 and increment the stored address to point to the next byte. Then we store the byte loaded to r2, into the address of Y and increment it. We check if the end of the string is reached, and if not then we loop. Otherwise we copy the entire string and end execution.

5.2 Detailed Description of the Instruction Used to Design the Program.

DCB: DCB allocates bytes of memory and defines the initial runtime contents of the memory.

LDR{B}: LDR loads the words to the specified register. B stands for Byte.

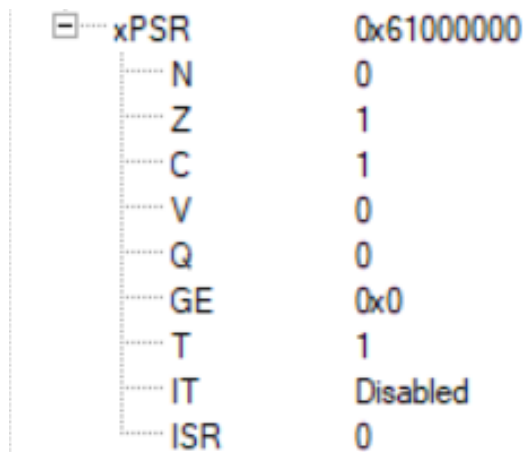
CMP: CMP compares the two operands and updates the status flags.

BNE: BNE branches to specified label if the result of the compared values were not equal.

STR{B}: STR stores the contents of the specified register to a memory location. B stands for Byte.

5.3 Screenshot of Debugger

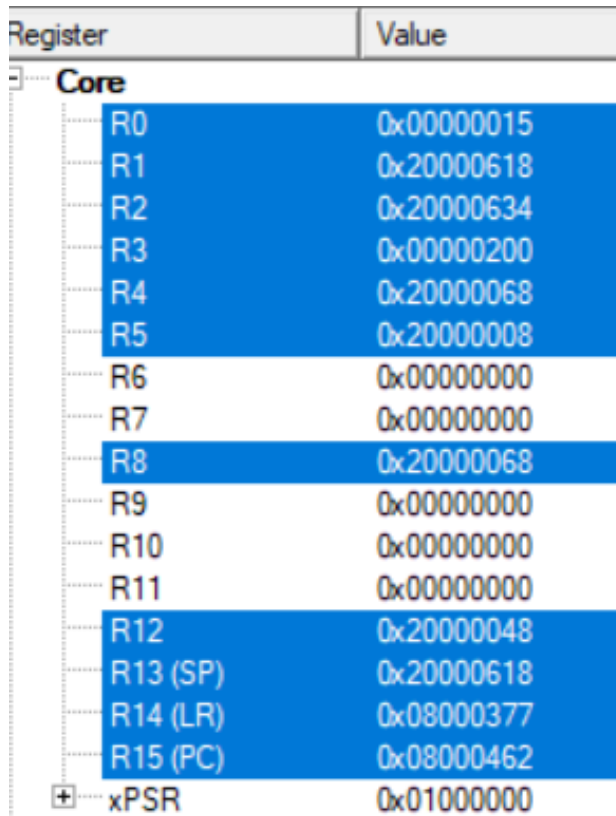
5.3.1 Status of the Status Registers After the Operation



A screenshot of a debugger's status register window. It shows a list of status registers on the left and their corresponding values on the right. The registers are xPSR, N, Z, C, V, Q, GE, T, IT, and ISR. The values are 0x61000000, 0, 1, 1, 0, 0, 0x0, 1, Disabled, and 0 respectively.

<input type="checkbox"/> xPSR	0x61000000
N	0
Z	1
C	1
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

5.3.2 After the Code has been Loaded



A screenshot of a debugger's register window. It shows a table with two columns: Register and Value. The registers are listed under a 'Core' header. The registers are R0 through R15, plus xPSR. The values are 0x00000015, 0x20000618, 0x20000634, 0x00000200, 0x20000068, 0x20000008, 0x00000000, 0x00000000, 0x20000068, 0x00000000, 0x00000000, 0x00000000, 0x20000048, 0x20000618, 0x08000377, 0x08000462, and 0x01000000 respectively.

Register	Value
Core	
R0	0x00000015
R1	0x20000618
R2	0x20000634
R3	0x00000200
R4	0x20000068
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x20000068
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000618
R14 (LR)	0x08000377
R15 (PC)	0x08000462
<input checked="" type="checkbox"/> xPSR	0x01000000

5.3.3 After the Code has been Executed

Register	Value
Core	
R0	0x20000006
R1	0x2000000C
R2	0x00000000
R3	0x00000100
R4	0x00000000
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000588
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000668
R14 (LR)	0x0800025B
R15 (PC)	0x08000540
xPSR	0x61000000

6 Task iv B

6.1 Detailed Code Explanation

We create an area for the variables, and allocate memory for the strings. We load the address of the first bytes strings X and Y to the registers r1 and r2 respectively. We also load the immediate address before the string X in the register r0. Then we load the current or first byte of the string X to r3 and increment the stored address to point to the next byte. We check if the end of the string is reached, and if not then we loop. Otherwise we reach the end of the string and have that address in the register r1. Then we start loading bytes from the last address of the string X in r1 and decrement it until we reach the start of the string, and store the bytes in reverse order in the memory location of Y.

6.2 Detailed Description of the Instruction Used to Design the Program.

DCB: DCB allocates bytes of memory and defines the initial runtime contents of the memory.

LDR{B}: LDR loads the words to the specified register. B stands for Byte.

ADD: :ADD multiplies the specified values and places the result in a register.

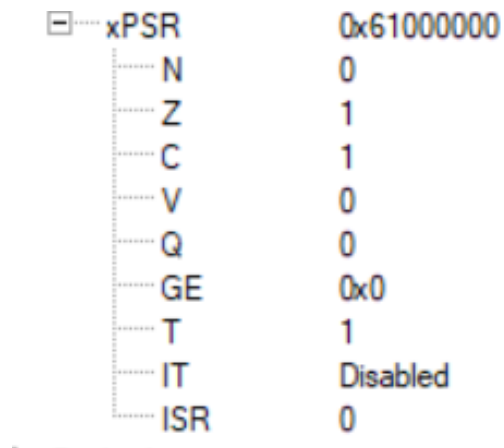
CMP: CMP compares the two operands and updates the status flags.

BNE: BNE branches to specified label if the result of the compared values were not equal.


STR{B}: STR stores the contents of the specified register to a memory location. B stands for Byte.

6.3 Screenshot of Debugger

6.3.1 Status of the Status Registers After the Operation



A screenshot of a debugger's status register window. The window has a minus sign icon in the top-left corner. It lists several status registers with their corresponding values. The registers are xPSR, N, Z, C, V, Q, GE, T, IT, and ISR. The values are 0x61000000, 0, 1, 1, 0, 0, 0x0, 1, Disabled, and 0 respectively.

	xPSR	0x61000000
	N	0
	Z	1
	C	1
	V	0
	Q	0
	GE	0x0
	T	1
	IT	Disabled
	ISR	0

6.3.2 After the Code has been Loaded

Register	Value
Core	
R0	0x00000015
R1	0x20000618
R2	0x20000634
R3	0x00000200
R4	0x20000068
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x20000068
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000618
R14 (LR)	0x08000377
R15 (PC)	0x08000462
xPSR	0x01000000

6.3.3 After the Code has been Executed

Register	Value
Core	
R0	0x1FFFFFFF
R1	0x1FFFFFFF
R2	0x2000000D
R3	0x00000048
R4	0x00000000
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000594
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000668
R14 (LR)	0x0800025B
R15 (PC)	0x0800054E
xPSR	0x61000000

7 Task iv C

7.1 Detailed Code Explanation

We create an area for the variables, and allocate memory for the strings. We load the address of the first bytes of the string X to the register r0. Then we load the current or first byte of the string X to r1 and increment the stored address to point to the next byte. Then we increment the value of r2, which we initialized to 0. We check if the end of the string is reached, and if not then we loop. Otherwise we find the length the entire string and decrement by 1 to remove null character and end execution.

7.2 Detailed Description of the Instruction Used to Design the Program.

DCB: DCB allocates bytes of memory and defines the initial runtime contents of the memory.

LDR{B}: LDR loads the words to the specified register. B stands for Byte.

MOV: MOV copies the value specified in the second operand to the the specified register.

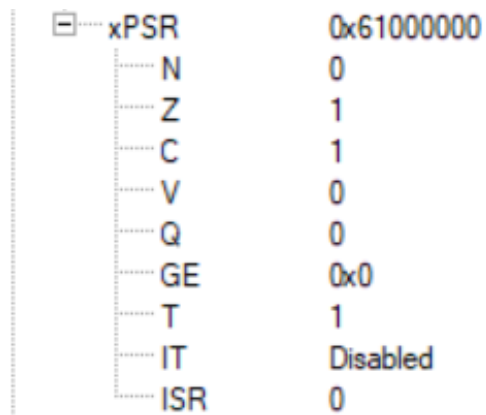
ADD: :ADD multiplies the specified values and places the result in a register.

CMP: CMP compares the two operands and updates the status flags.

BNE: BNE branches to specified label if the result of the compared values were not equal.

7.3 Screenshot of Debugger

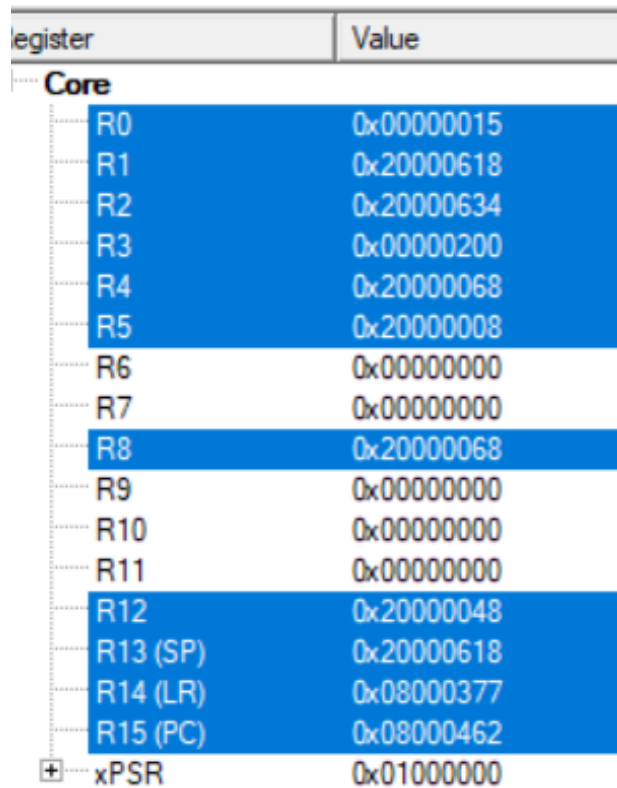
7.3.1 Status of the Status Registers After the Operation



A screenshot of a debugger's status register window. It shows a tree view with 'xPSR' expanded, listing various flags and their values. The flags include N (0), Z (1), C (1), V (0), Q (0), GE (0x0), T (1), IT (Disabled), and ISR (0).

<input type="checkbox"/> xPSR	0x61000000
N	0
Z	1
C	1
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

7.3.2 After the Code has been Loaded



A screenshot of a debugger's register window. It shows a table of registers for a 'Core'. The registers are R0 through R15, plus xPSR. R0-R5, R8, R12, R13 (SP), R14 (LR), and R15 (PC) are highlighted in blue. The xPSR register is at the bottom and is not highlighted.

Register	Value
Core	
R0	0x00000015
R1	0x20000618
R2	0x20000634
R3	0x00000200
R4	0x20000068
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x20000068
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000618
R14 (LR)	0x08000377
R15 (PC)	0x08000462
<input checked="" type="checkbox"/> xPSR	0x01000000

7.3.3 After the Code has been Executed

Register	Value
Core	
R0	0x20000006
R1	0x00000000
R2	0x00000005
R3	0x00000100
R4	0x00000000
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000588
R11	0x00000000
R12	0x20000048
R13 (SP)	0x20000668
R14 (LR)	0x0800025B
R15 (PC)	0x08000546
xPSR	0x61000000

8 Task iv D

8.1 Detailed Code Explanation

We create an area for the variables, and allocate memory for the strings. We load the address of the first bytes strings X and Y to the registers r0 and r1 respectively. Then we load the current byte of X and Y to the registers r2 and r3, and check if any of them have reached the end of the string, and if so, go to exit. Otherwise we compare the values of the two characters, and if they are the same we loop again, otherwise we exit the loop. After exiting, we compare the last two characters, if the loop exited with both being equal, then this last compare would reflect it and update flags accordingly.

8.2 Detailed Description of the Instruction Used to Design the Program.

DCB: DCB allocates words of memory and defines the initial runtime contents of the memory.

LDR{B}: LDR loads the words to the specified register. B stands for Byte.

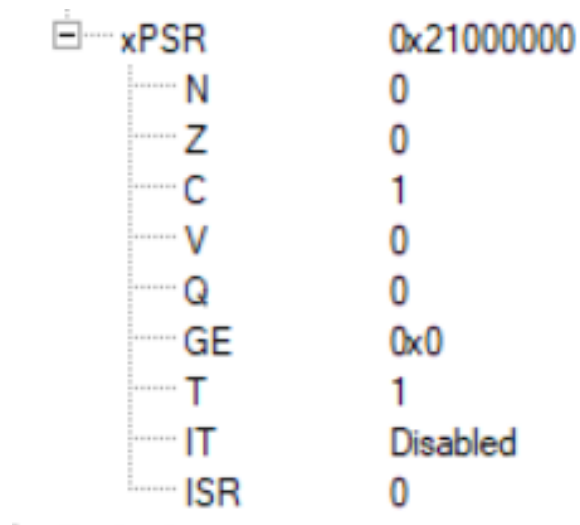
CMP: CMP compares the two operands and updates the status flags.

BEQ: BEQ branches to specified label if the result of the compared values were equal.


BNB: BNE branches to specified label if the result of the compared values were not equal.

8.3 Screenshot of Debugger



8.3.1 Status of the Status Registers After the Operation



A screenshot of a debugger's status register window. The window has a minus icon in the top-left corner. It lists several status registers with their current values. The registers are: xPSR (0x21000000), N (0), Z (0), C (1), V (0), Q (0), GE (0x0), T (1), IT (Disabled), and ISR (0).

	xPSR	0x21000000
	N	0
	Z	0
	C	1
	V	0
	Q	0
	GE	0x0
	T	1
	IT	Disabled
	ISR	0

8.3.2 After the Code has been Loaded

Register	Value
 Core	
R0	0x00000015
R1	0x20000620
R2	0x2000063C
R3	0x00000200
R4	0x20000070
R5	0x20000010
R6	0x00000000
R7	0x00000000
R8	0x20000070
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000050
R13 (SP)	0x20000620
R14 (LR)	0x08000377
R15 (PC)	0x08000462
 xPSR	0x01000000

8.3.3 After the Code has been Executed

Register	Value
Core	
R0	0x20000005
R1	0x2000000B
R2	0x00000064
R3	0x00000000
R4	0x00000000
R5	0x20000010
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000590
R11	0x00000000
R12	0x20000050
R13 (SP)	0x20000670
R14 (LR)	0x0800025B
R15 (PC)	0x0800054A
xPSR	0x21000000

9 Task iv E

9.1 Detailed Code Explanation

We create an area for the variables, and allocate memory for the strings. We load the address of the first bytes strings X and Y to the registers r0 and r1 respectively. Then we load the current or first byte of the string X to r2 and increment the stored address to point to the next byte. We check if the end of the string is reached, and if not then we loop. Otherwise we reach the end of the string and that address is stored in the register r0. Then we load the current or first byte of the string Y to r2 and increment that by 1, and store that byte to the address stored in r0. Then we check if the end of the string Y is reached, otherwise we loop. As a result, the two strings will be concatenated. We create an area for the variables, and allocate memory for the strings.

9.2 Detailed Description of the Instruction Used to Design the Program.

DCB: DCB allocates bytes of memory and defines the initial runtime contents of the memory.

LDR{B}: LDR loads the words to the specified register. B stands for Byte.

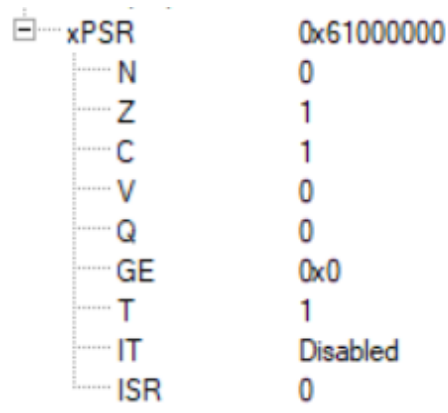
CMP: CMP compares the two operands and updates the status flags.

BNE: BNE branches to specified label if the result of the compared values were not equal.


STR{B}: STR stores the contents of the specified register to a memory location. B stands for Byte.

9.3 Screenshot of Debugger

9.3.1 Status of the Status Registers After the Operation



A screenshot of a debugger's status register window. The window has a minus sign icon in the top-left corner. It lists several status registers with their corresponding values. The registers are: xPSR (0x61000000), N (0), Z (1), C (1), V (0), Q (0), GE (0x0), T (1), IT (Disabled), and ISR (0).

	xPSR	0x61000000
	N	0
	Z	1
	C	1
	V	0
	Q	0
	GE	0x0
	T	1
	IT	Disabled
	ISR	0

9.3.2 After the Code has been Loaded

Register	Value
Core	
R0	0x00000015
R1	0x20000620
R2	0x2000063C
R3	0x00000200
R4	0x20000070
R5	0x20000010
R6	0x00000000
R7	0x00000000
R8	0x20000070
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000050
R13 (SP)	0x20000620
R14 (LR)	0x08000377
R15 (PC)	0x08000462
+ xPSR	0x01000000

9.3.3 After the Code has been Executed

Register	Value
Core	
R0	0x2000000F
R1	0x2000000F
R2	0x00000000
R3	0x00000100
R4	0x00000000
R5	0x20000010
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000590
R11	0x00000000
R12	0x20000050
R13 (SP)	0x20000670
R14 (LR)	0x0800025B
R15 (PC)	0x08000548
xPSR	0x61000000
N	0
Z	1
C	1
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0