

Microprocessor and Assembly Language Lab 02

Kazi Shadman Sakib, FH-97

February 02, 2022

1 Task 01

1.1 Detailed Code Explanation

To solve this problem, I have first put the data in memory in the form of constants, where $X = 9$, $Y = 8$ and $Z = 5$ by using EQU (equate) Opcode. The equate directive (EQU) is used to substitute values for symbols or labels. Then in the main function, I have used the Opcode MOV to load the value of constants X, Y and Z in registers r4, r3 and r2 respectively, thus $r4 = X$, $r3 = Y$, $r2 = Z$. We simply then add the values in the registers r2 and r3, and store the result in register r0 by using the Opcode ADD. Lastly, we again use the Opcode ADD to add the values in the registers r0 and r4, and store the final result in the register r0.

1.2 Screenshot of Debugger

1.2.1 After the Code has been Loaded

F:\3-1\Microprocessor Lab\Codes\Lab_02_Problem01\Lab02_Problem01.uvprojx - µVision [Non-Commercial Use License]

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

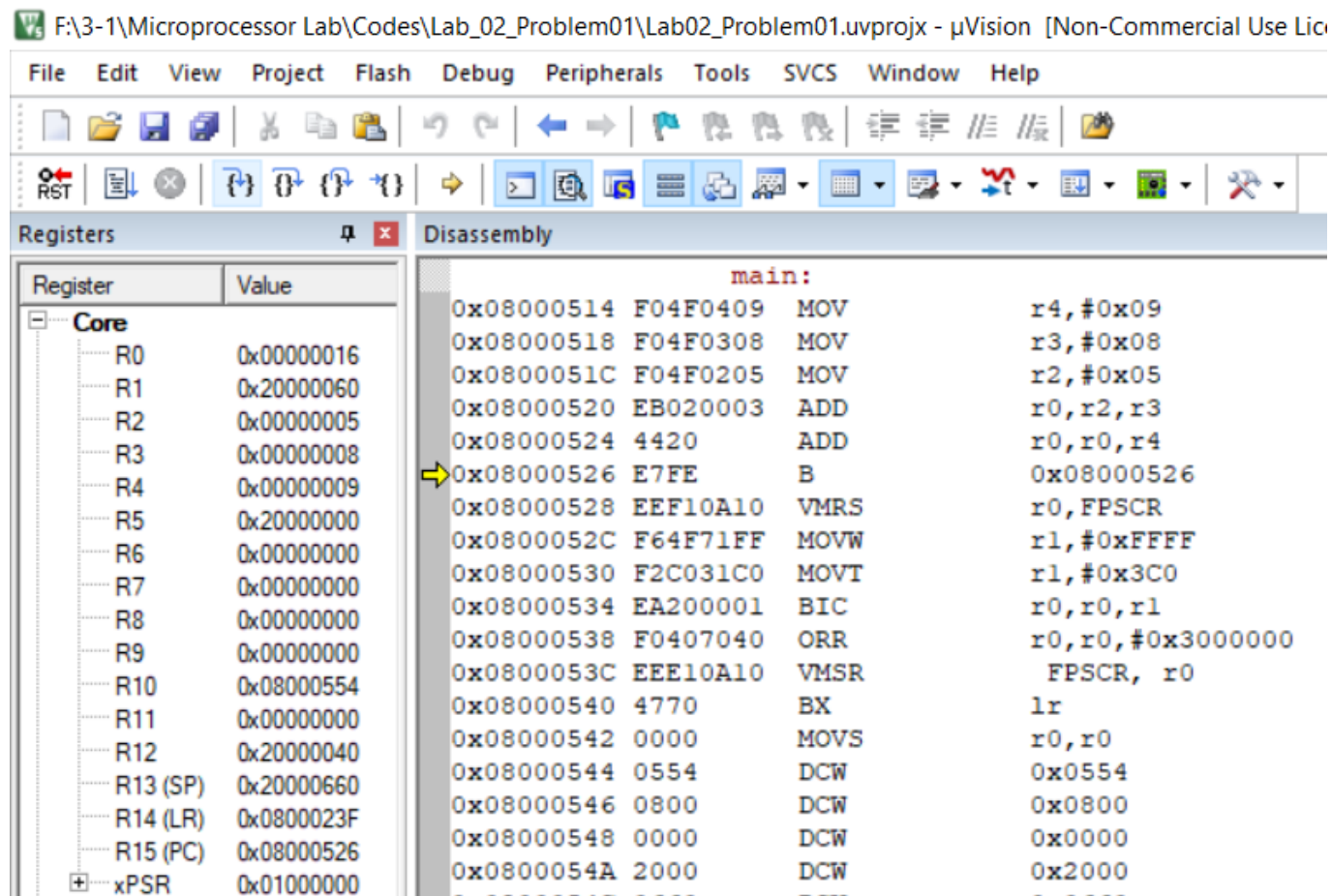
Registers

Register	Value
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 (SP)	0x20000610
R14 (LR)	0x0800035B
R15 (PC)	0x08000446
xPSR	0x01000000

Disassembly

Address	Label	Opcode	Comment
0x08000446	BEAB	BKPT	0xAB
0x08000448	B108	CBZ	r0, 0x0800044E
0x0800044A	2000	MOVS	r0, #0x00
0x0800044C	BD1C	POP	{r2-r4, pc}
0x0800044E	4620	MOV	r0, r4
0x08000450	BD1C	POP	{r2-r4, pc}
0x08000452	4770	BX	lr
0x08000454	4770	BX	lr
0x08000456	4770	BX	lr
0x08000458	B510	PUSH	{r4, lr}
0x0800045A	F000F809	BL.W	0x08000470 __rt_SIGRTMEM_inner
0x0800045E	E8BD4010	POP	{r4, lr}
0x08000462	F3AF8000	NOP.W	
0x08000466	2800	CMP	r0, #0x00
0x08000468	D001	BEQ	0x0800046E
0x0800046A	F7FFBF49	B.W	0x08000300 _sys_exit
0x0800046E	4770	BX	lr
0x08000470	B510	PUSH	{r4, lr}
0x08000472	2801	CMP	r0, #0x01
0x08000474	D005	BEQ	0x08000482

1.2.2 After the Code has been Executed



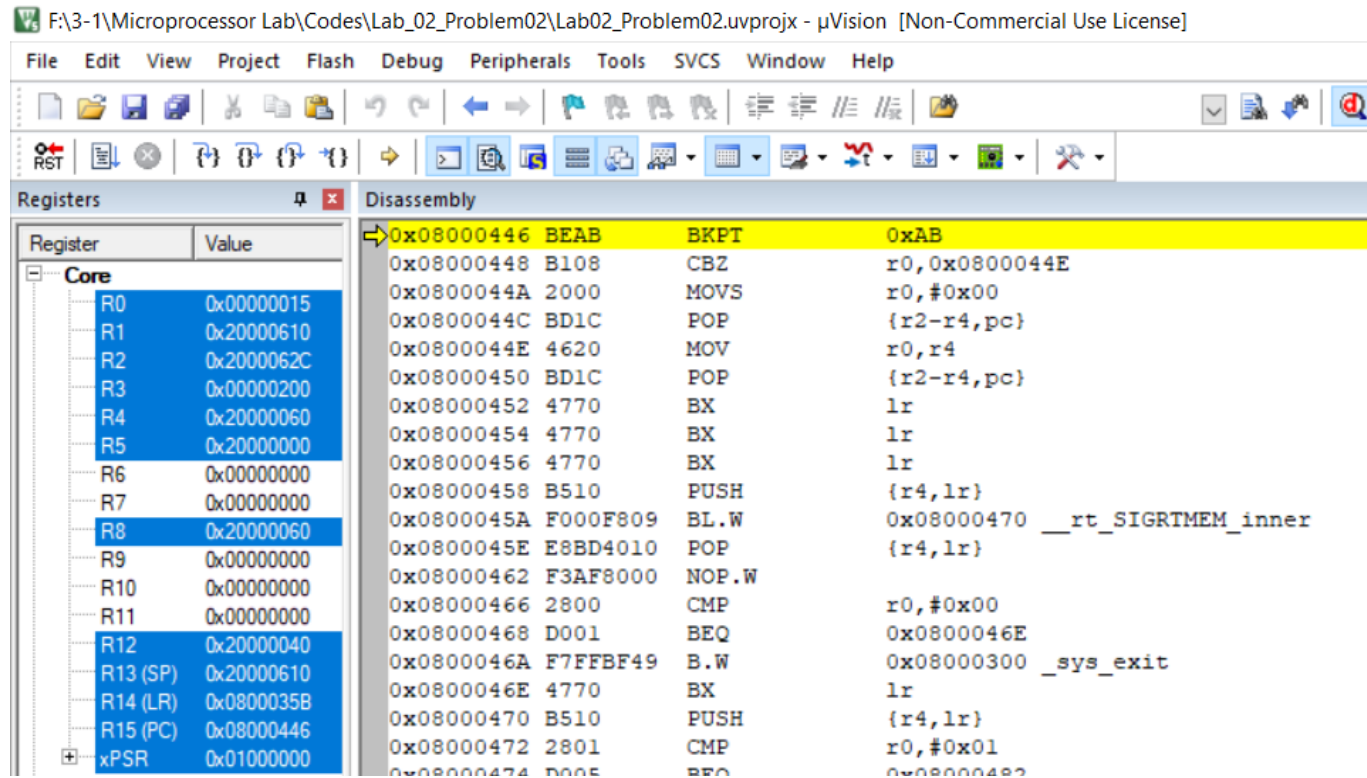
2 Task 02

2.1 Detailed Code Explanation

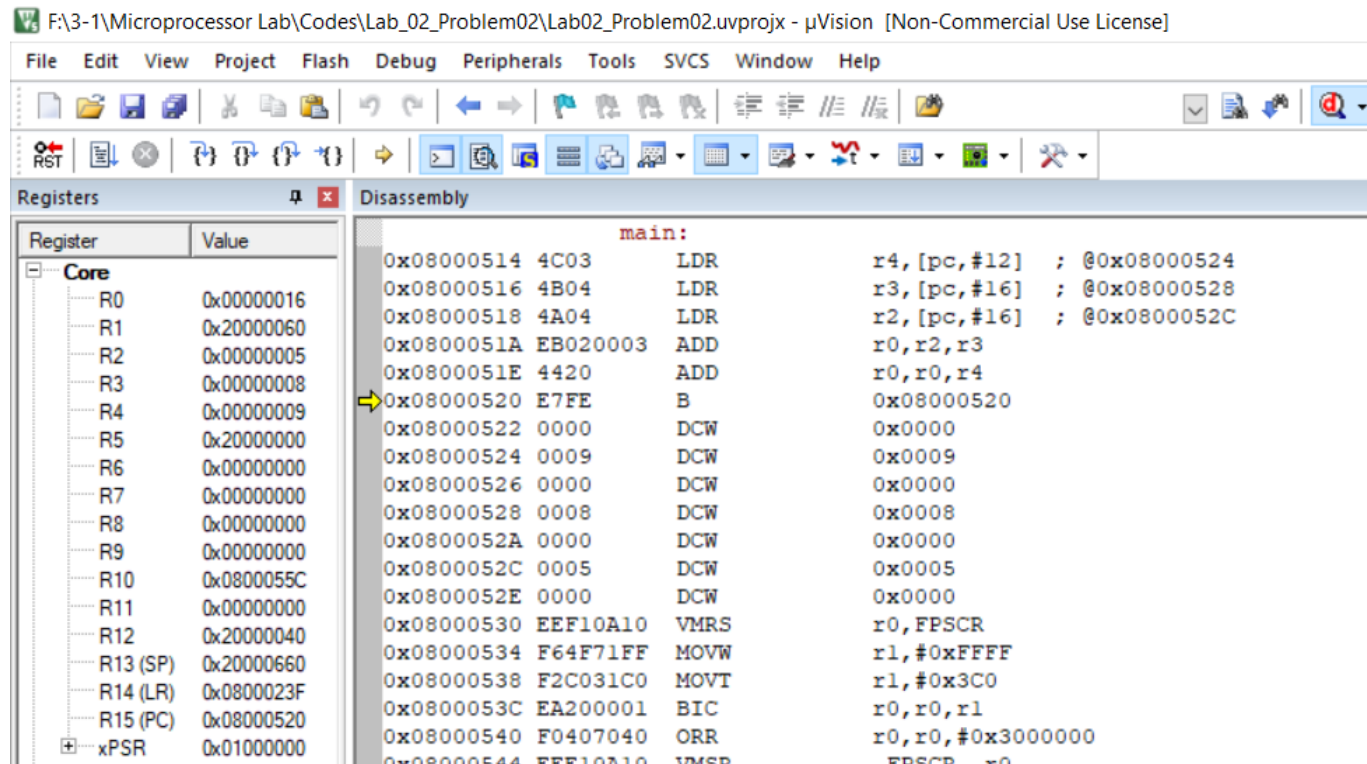
To solve this problem, I have first created three variables X, Y and Z with their initial values as $X = 9$, $Y = 8$ and $Z = 5$ by using the Opcode DCD. The DCD directive allocates one or more words of memory, aligned on four-byte boundaries, and defines the initial run-time contents of the memory. After this, in the main function I have used the Opcode LDR to load the register r4, r3 and r2 with the content of memory location of X, Y and Z respectively, thus $r4 = X$, $r3 = Y$ and $r2 = Z$. We simply then add the values in the registers r2 and r3, and store the result in register r0 by using the Opcode ADD. Lastly, we again use the Opcode ADD to add the values in the registers r0 and r4, and store the final result in the register r0.

2.2 Screenshot of Debugger

2.2.1 After the Code has been Loaded



2.2.2 After the Code has been Executed



3 Task 03

3.1 Detailed Code Explanation

To solve this problem, I have first used the Opcode MOVW to load only 16 bits [15:0] to the registers r1 and r2 with the hexa-decimal values 0xFFFF. Then I have used the Opcode ADD to find the addition of the two 16 bit variables. Lastly, I have used the Opcode MOVT to load 0's in the upper 16 bit [31:16] of the final result, which is in register r0. Opcode MOVT writes to Rd[31:16], without affecting Rd[15:0]. Thus the required result is in the register r0.

3.2 Screenshot of Debugger

3.2.1 After the Code has been Loaded

F:\3-1\Microprocessor Lab\Codes\Lab_02_Problem03\Lab02_Problem03.uvprojx - µVision [Non-Commercial Use License]

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

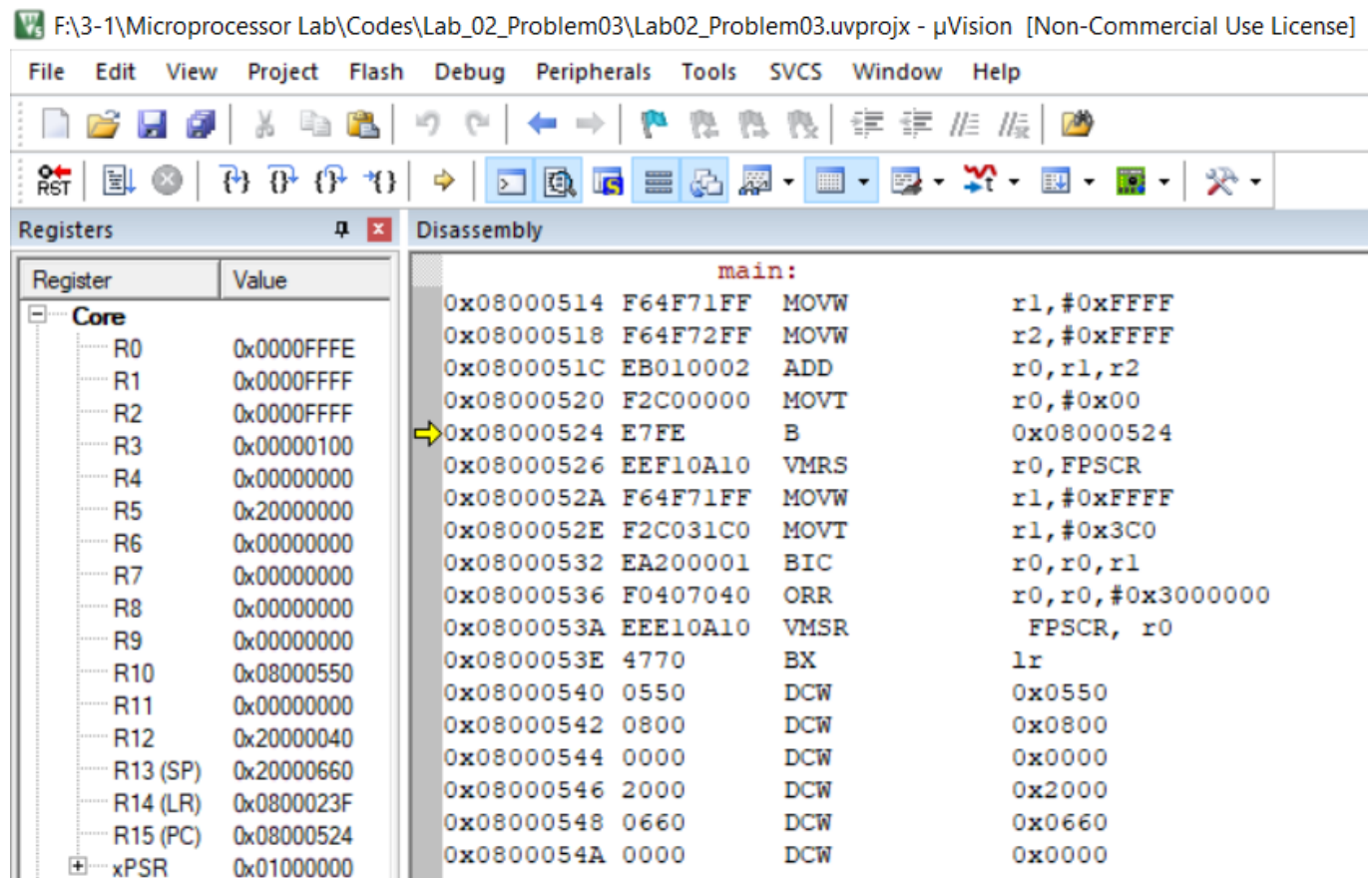
Registers

Register	Value
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 (SP)	0x20000610
R14 (LR)	0x08000358
R15 (PC)	0x08000446
xPSR	0x01000000

Disassembly

Address	Hex	Op	Comment
0x08000446	BEAB	BKPT	0xAB
0x08000448	B108	CBZ	r0,0x0800044E
0x0800044A	2000	MOVS	r0,#0x00
0x0800044C	BD1C	POP	{r2-r4,pc}
0x0800044E	4620	MOV	r0,r4
0x08000450	BD1C	POP	{r2-r4,pc}
0x08000452	4770	BX	lr
0x08000454	4770	BX	lr
0x08000456	4770	BX	lr
0x08000458	B510	PUSH	{r4,lr}
0x0800045A	F000F809	BL.W	0x08000470 __rt_SIGRTMEM_inner
0x0800045E	E8BD4010	POP	{r4,lr}
0x08000462	F3AF8000	NOP.W	
0x08000466	2800	CMP	r0,#0x00
0x08000468	D001	BEQ	0x0800046E
0x0800046A	F7FFBF49	B.W	0x08000300 _sys_exit
0x0800046E	4770	BX	lr
0x08000470	B510	PUSH	{r4,lr}
0x08000472	2801	CMP	r0,#0x01
0x08000474	D005	BEQ	0x08000482

3.2.2 After the Code has been Executed



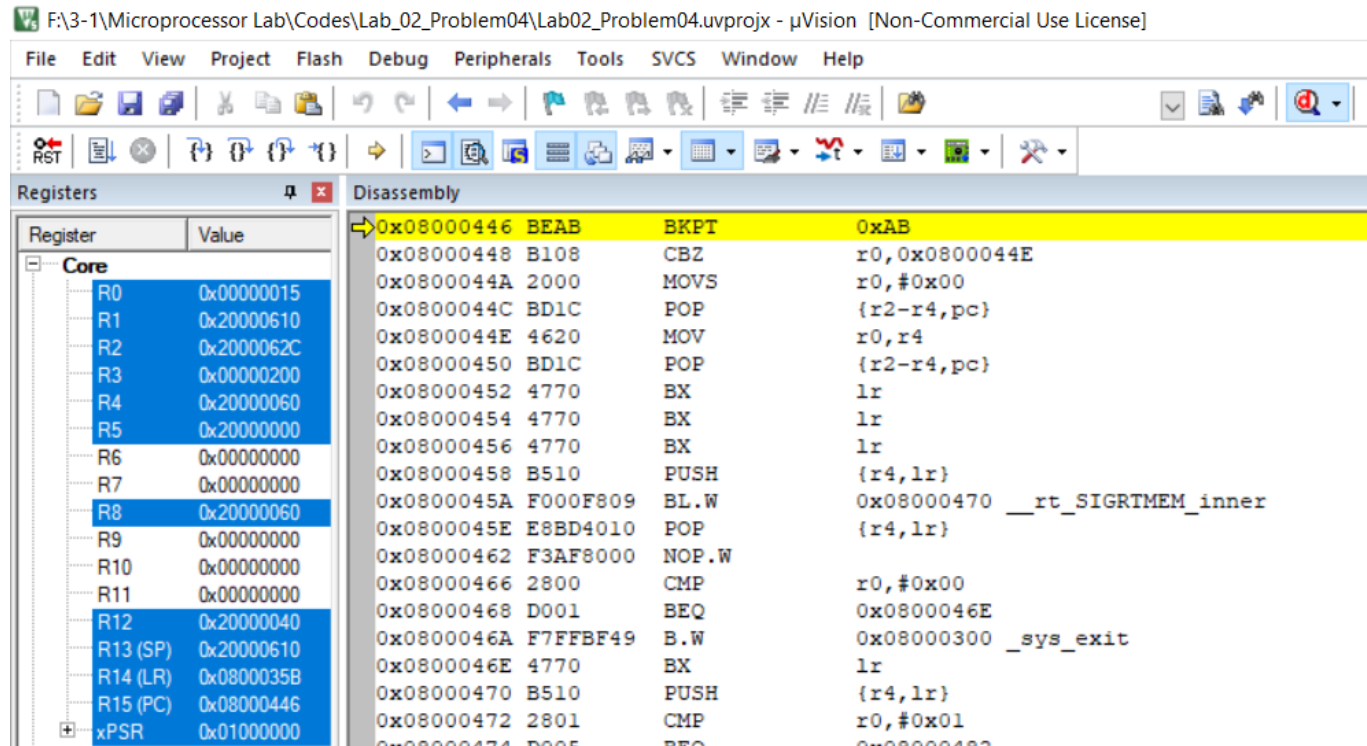
4 Task 04

4.1 Detailed Code Explanation

To solve this problem, I have first created two integer variables value1 and value2, and assigned their initial values as value1 = 8 and value2 = 9 by using the Opcode DCD. The DCD directive allocates one or more words of memory, aligned on four-byte boundaries, and defines the initial run-time contents of the memory. After this, in the main function I have used the Opcode LDR to load the register r1 and r2 with the content of memory location of value1 and value2 respectively, thus r1 = value1 and r2 = value2. To compare the values of these two registers I have used the Opcode CMP. The Opcode CMP compares the two operands. The BLT (branch less than) instruction is one of several conditional branch instructions. The conditional transfer instruction BLT transfers control to the statement labeled "done" if the contents of r1 contains the smaller integer number. Otherwise, the next instruction is executed. At "done", register r0 will always contain the smaller of the two integer numbers as the final result.

4.2 Screenshot of Debugger

4.2.1 After the Code has been Loaded



4.2.2 After the Code has been Executed

