



# HACETTEPE UNIVERSITY COMPUTER ENGINEERING DEPARTMENT

ADVANCED COMPUTER ARCHITECTURE  
PROJECT

Ahmet Burak KOÇ | CMP634 | 15 November 2020

# 1) Arrays using for loops

Converting C code fragment to MIPS code is conducted. In MIPS code, multiplication instructions (mult/mul) are not allowed to be used.

## C CODE FRAGMENT

Below C code fragment is to be converted.

```
int A[4];
int i;
int diff;
for(i=0; i<3; i++)
{
    diff = A[i+1] - A[i];
    if (diff > 0)
        A[i] = 5*A[i];
    else
        A[i+1] = -5*A[i];
}
```

## MIPS CODE

For given C code fragment, MIPS code is written as below.

```
.data
##; Input data
A: .word 8, 6, 4, 2

.text

##      ;Main function registers
#      Base address: $t0
#      A(i)           : $s0
#      A(i+1)         : $s1
#      diff           : $s2
##      ;For loop registers
#      slt_res_for    : $t1
#      i              : $t2
#      i_limit        : $t3
##      ;Multiplication loop registers
#      slt_res_mult   : $t4
#      j              : $t5
#      multiplier     : $t6
#      product        : $s3
##      ;If registers
#      slt_res_if     : $t7

main:
    la $t0, A           # $t0, A base address
    addi $s2, $0, 0     # $s2, diff = 0
    addi $t2, $0, 0     # $t2, i = 0
    addi $t3, $0, 3     # $t3, i_limit = 3

##; for loop begin##
loop_for:
    slt $t1, $t2, $t3   # $t1 = 1 if i is less than i_limit=3
    beq $t1, $0, done_for # jump done_for if $t1 = 0 (condition is false)
    lw  $s0, 0($t0)     # $s0, load A(i)
    lw  $s1, 4($t0)     # $s1, load A(i+1)
```

```

        sub $s2, $s1, $s0          # A(i+1)-A(i) -> diff

##; calculation 5*A[i] begin##
        addi $t5, $0, 0            # j=0
        addi $t6, $0, 5            # multiplier=5
        addi $s3, $0, 0            # product=0
loop_mult:
        slt $t4, $t5, $t6          # slt_res_mult=1 if j is less than multiplier=5
        beq $t4, $0, done_mult      # jump done_mult slt_res_mult=0
        add $s3, $s3, $s0           # product=product+A(i)
        addi $t5, $t5, 1            # increment j by 1
        j loop_mult                # jump loop_mult
done_mult:
##; calculation 5*A(i) end##

##; if/else begin ##
        slt $t7, $0, $s2           # slt_res_if=1 if 0 is less than diff
        beq $t7, $0, else_op        # if diff is less than 0 continue, else jump else_op
if_op:
        sw $s3, 0($t0)             # store product --> A(i)
        j done_if                  # jump done_if
else_op:
        sub $s3, $0, $s3           # product = -product
        sw $s3, 4($t0)             # store product --> A(i+1)
done_if:
##; if/else end##

        addi $t0, $t0, 4           # increment array address by 4 (index by 1)
        addi $t2, $t2, 1           # increment i by 1
        j loop_for                 # jump loop_for
done_for:
##; for loop end##

endloop:
        li $v0, 10                 # terminate program run and
        syscall                     # Exit

```

## OPTIMISATION

For multiplication function addition in for loop is used which is not efficient way of multiplication but one can easily change of multiplier value. For more efficient multiplication with 5, it can be replaced with

```

sll $s3, $s0, 2          # product=4*A(i)
add $s3, $s3, $s0        # product = product + A(i) = 5*A(i)

```

## TESTS & SCREENSHOTS

For given C code fragment, MIPS code is written as below. The program is tested for the following input values:

Test 1: A={2,4,6,8}

Test 2: A={8,6,4,2}

Test 3: A={2,2,6,4}

For each test, screenshots of the memory before and after running the code are given in Figure 1, Figure 2, Figure 3, Figure 4, Figure 5 and Figure 6. **Values are in Hexadecimal form.**

Before test 1, A contains {2, 4, 6, 8} in decimal. After test 1, A contains {10, 20, 30, 8} in decimal.

Before test 2, A contains {8, 6, 4, 2} in decimal. After test 1, A contains {8, -200, 4, -20} in decimal.

Before test 3, A contains {2, 2, 6, 4} in decimal. After test 1, A contains {2, -50, 6, -30 } in decimal.

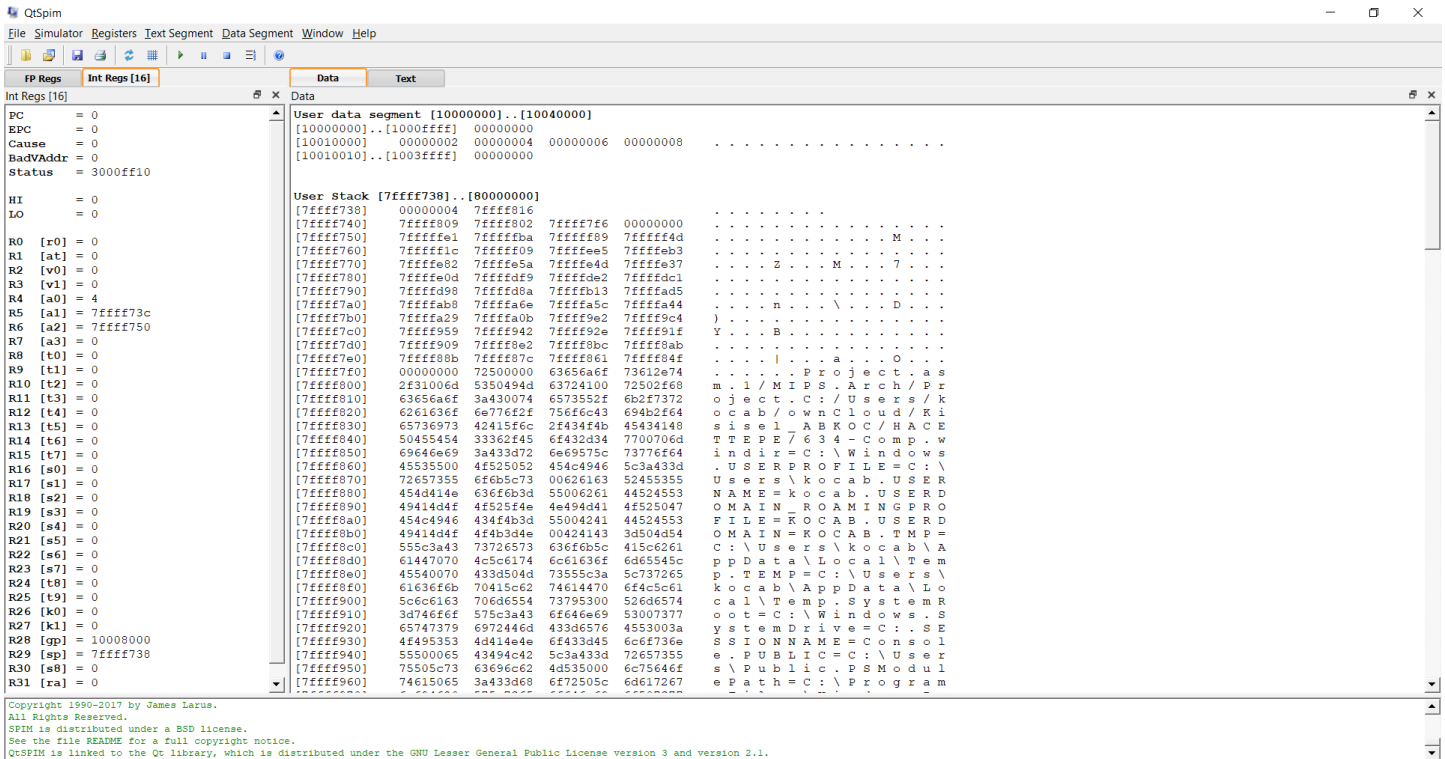


Figure 1: Memory and Register Values (hex) before Test 1:  $A=\{2,4,6,8\}$

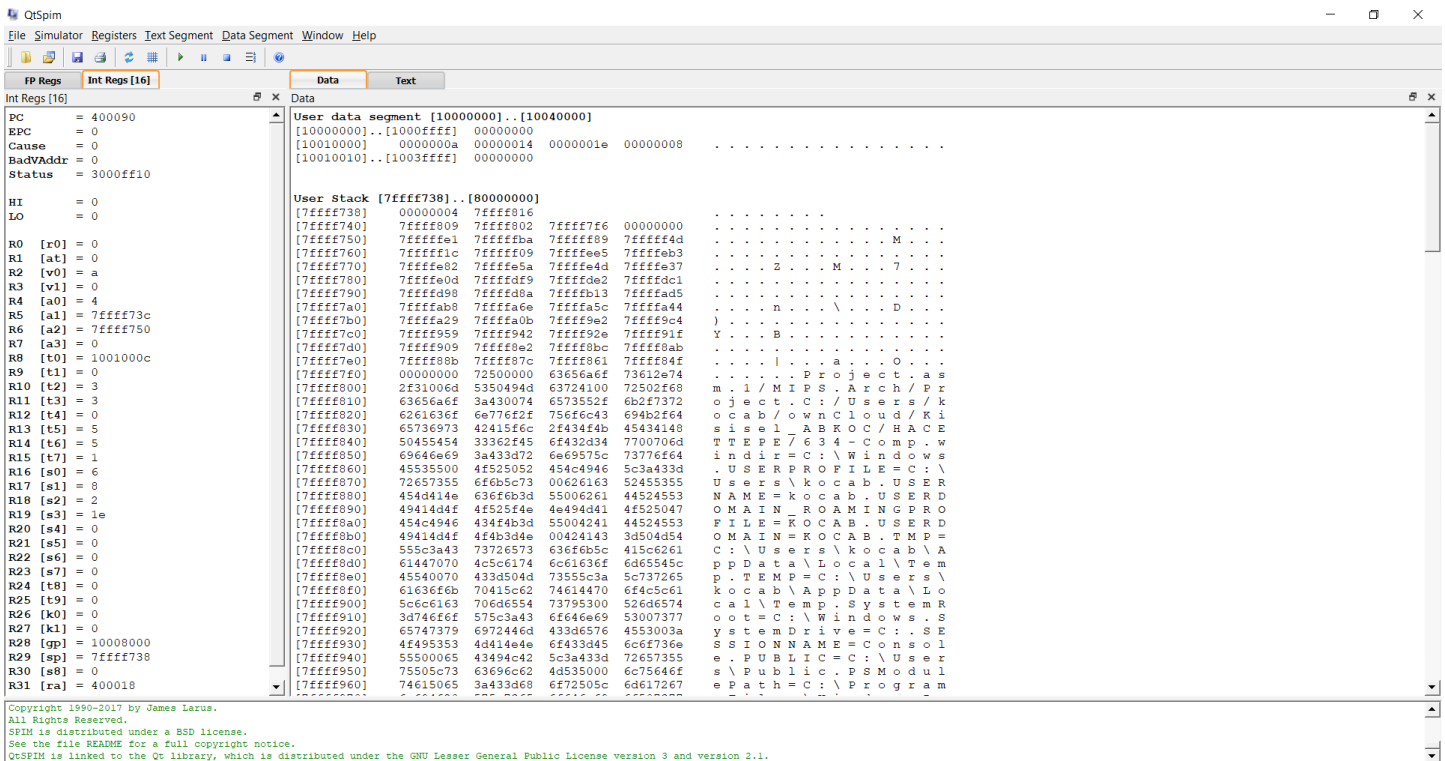


Figure 2: Memory and Register Values (hex) after Test 1:  $A=\{2,4,6,8\}$

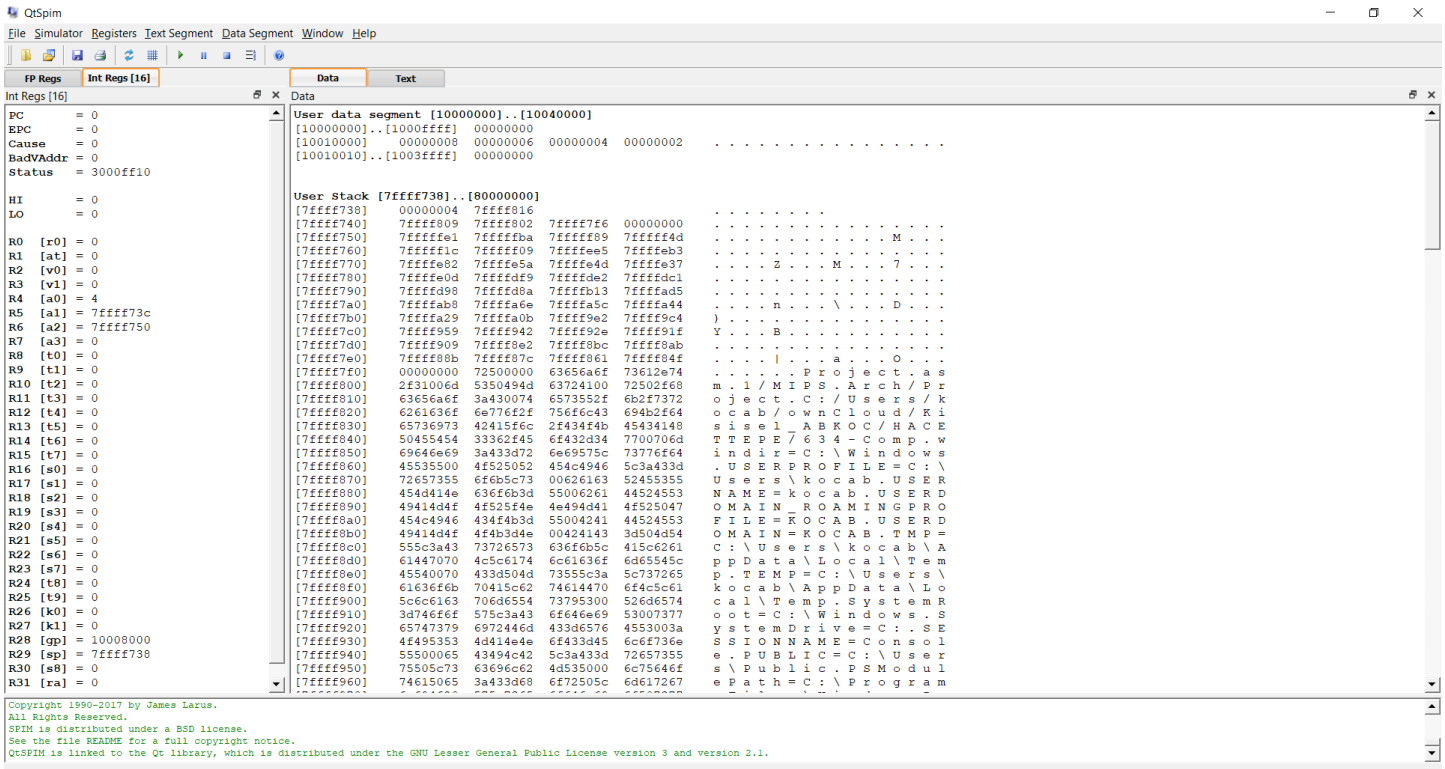


Figure 3: Memory and Register Values (hex) before Test 2: A={8,6,4,2}

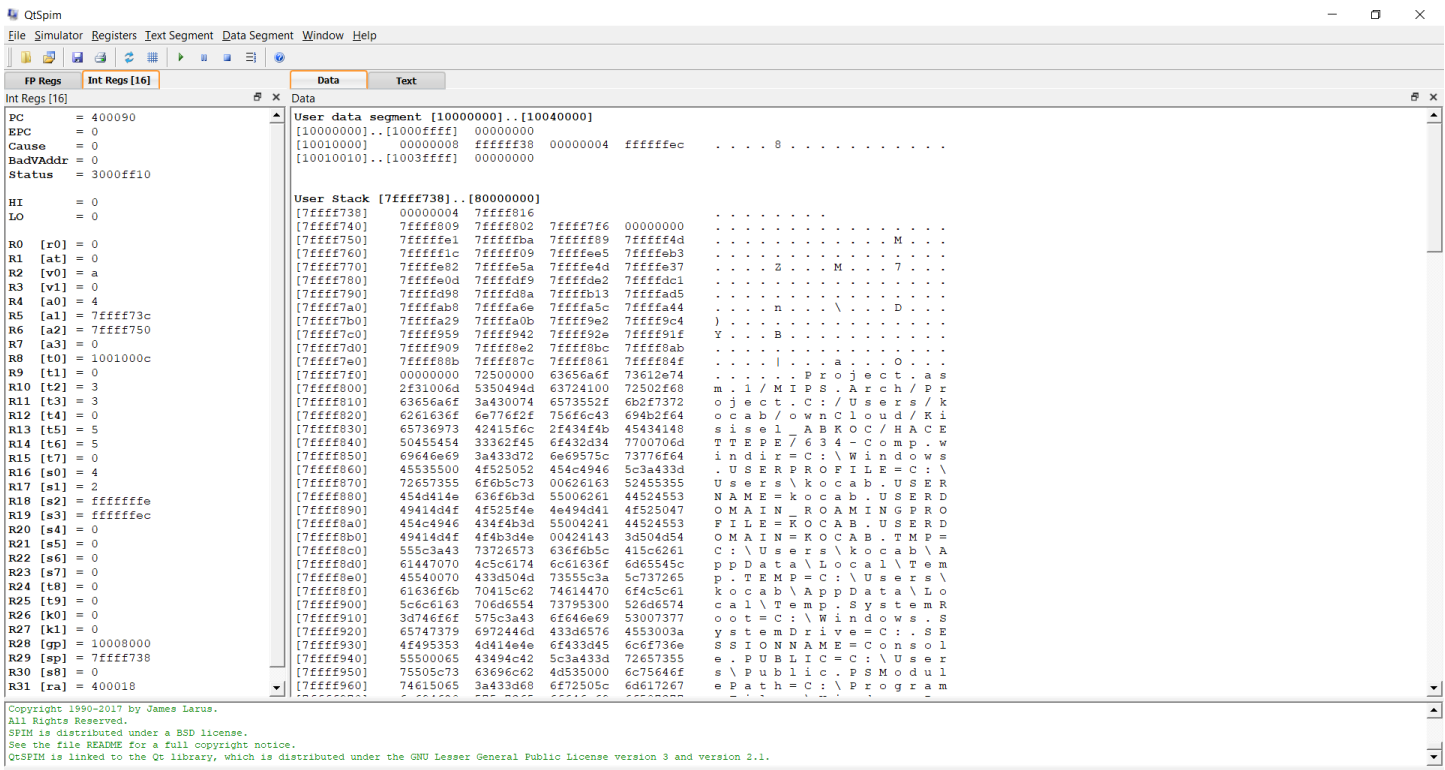


Figure 4: Memory and Register Values (hex) after Test 2: A={8,6,4,2}

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs	Int Regs [16]	Data	Text
<p>Int Regs [16]</p> <p>PC = 0</p> <p>EPC = 0</p> <p>Cause = 0</p> <p>BadVAddr = 0</p> <p>Status = 3000ff10</p> <p>HI = 0</p> <p>LO = 0</p> <p>R0 [x0] = 0</p> <p>R1 [at] = 0</p> <p>R2 [v0] = 0</p> <p>R3 [v1] = 0</p> <p>R4 [a0] = 4</p> <p>R5 [a1] = 7ffff73c</p> <p>R6 [a2] = 7ffff750</p> <p>R7 [a3] = 0</p> <p>R8 [t0] = 0</p> <p>R9 [t1] = 0</p> <p>R10 [t2] = 0</p> <p>R11 [t3] = 0</p> <p>R12 [t4] = 0</p> <p>R13 [t5] = 0</p> <p>R14 [t6] = 0</p> <p>R15 [t7] = 0</p> <p>R16 [s0] = 0</p> <p>R17 [s1] = 0</p> <p>R18 [s2] = 0</p> <p>R19 [s3] = 0</p> <p>R20 [s4] = 0</p> <p>R21 [s5] = 0</p> <p>R22 [s6] = 0</p> <p>R23 [s7] = 0</p> <p>R24 [t8] = 0</p> <p>R25 [t9] = 0</p> <p>R26 [k0] = 0</p> <p>R27 [k1] = 0</p> <p>R28 [gp] = 10008000</p> <p>R29 [sp] = 7ffff738</p> <p>R30 [s8] = 0</p> <p>R31 [ra] = 0</p>			
<p>User data segment [10000000]..[10040000]</p> <p>[10000000]..[1000ffff] 00000000</p> <p>[10010000] 00000002 00000002 00000006 00000004 . . . . .</p> <p>[10010010]..[1003ffff] 00000000</p>			
<p>User Stack [7ffff738]..[80000000]</p> <p>[7ffff738] 00000004 7ffff816 . . . . .</p> <p>[7ffff740] 7ffff809 7ffff802 7ffff7f6 00000000 . . . . .</p> <p>[7ffff750] 7ffff8e1 7ffff8ba 7ffff809 7ffff8fd . . . . .</p> <p>[7ffff760] 7ffff81c 7ffff809 7ffff8e5 7ffff8b3 . . . . .</p> <p>[7ffff770] 7ffff8e2 7ffff85a 7ffff8e4 7ffff837 . . . . .</p> <p>[7ffff780] 7ffff80d 7ffff8f9 7ffff8e2 7ffff8d1 . . . . .</p> <p>[7ffff790] 7ffff8d9 7ffff8ba 7ffff8b3 7ffff8d5 . . . . .</p> <p>[7ffff7a0] 7ffff8ab 7ffff8a6 7ffff8a5 7ffff8a4 . . . . .</p> <p>[7ffff7b0] 7ffff8a29 7ffff8a0b 7ffff8e2 7ffff8c4 . . . . .</p> <p>[7ffff7c0] 7ffff859 7ffff842 7ffff82e 7ffff81f . . . . .</p> <p>[7ffff7d0] 7ffff809 7ffff802 7ffff8bc 7ffff8ab . . . . .</p> <p>[7ffff7e0] 7ffff88b 7ffff87c 7ffff861 7ffff84f . . . . .</p> <p>[7ffff7f0] 00000000 72500000 63656a6f 73612e74 . . . . .</p> <p>[7ffff800] 2f31006d 5350494d 63724100 72502f68 . . . . .</p> <p>[7ffff810] 63656a6f 3a430074 6573552f 6b2f7372 . . . . .</p> <p>[7ffff820] 6261636f 6e776f2f 756f6c43 694b2f64 . . . . .</p> <p>[7ffff830] 65736973 42415f6c 2f434f4b 45434148 . . . . .</p> <p>[7ffff840] 50455454 33362f45 6f432d34 7700706d . . . . .</p> <p>[7ffff850] 69646e69 3a433d72 6e69575c 73776f64 . . . . .</p> <p>[7ffff860] 45535500 4f525052 454c4946 5c3a433d . . . . .</p> <p>[7ffff870] 72657355 6f6b5c73 00626163 52455355 . . . . .</p> <p>[7ffff880] 454d414e 636f6b3d 55006261 44524553 . . . . .</p> <p>[7ffff890] 49414d4f 4f525f4e 4e494d41 4f525047 . . . . .</p> <p>[7ffff8a0] 454c4946 434f4b3d 55004241 44524553 . . . . .</p> <p>[7ffff8b0] 49414d4f 4f4b3d4e 00424143 3d504d54 . . . . .</p> <p>[7ffff8c0] 555c3a43 73726573 636f6b5c 415c6261 . . . . .</p> <p>[7ffff8d0] 61447070 4c5c6174 6c61636f 6d65545c . . . . .</p> <p>[7ffff8e0] 45540070 433d504d 73555c3a 5c737265 . . . . .</p> <p>[7ffff8f0] 61636f6b 70415c62 74614470 6f4c5c61 . . . . .</p> <p>[7ffff900] 5c6c6163 706d6554 73795300 526d6574 . . . . .</p> <p>[7ffff910] 3d746f6f 575c3a43 6f646e69 53007377 . . . . .</p> <p>[7ffff920] 65747379 6972446d 433d657e 4553003a . . . . .</p> <p>[7ffff930] 4f495353 4d414e4e 6f433d45 6c6f736e . . . . .</p> <p>[7ffff940] 55500065 43494c42 5c3a433d 72657355 . . . . .</p> <p>[7ffff950] 75505c73 63696c62 4d535000 6c75646f . . . . .</p> <p>[7ffff960] 74615065 3a433d68 6f72505c 6d617267 . . . . .</p>			

Copyright 1990-2017 by James Larus.  
All Rights Reserved.  
SPIM is distributed under a BSD license.  
See the file README for a full copyright notice.  
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

Figure 5: Memory and Register Values (hex) before Test 3:  $A=\{2,2,6,4\}$

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs	Int Regs [16]	Data	Text
<p>Int Regs [16]</p> <p>PC = 400090</p> <p>EPC = 0</p> <p>Cause = 0</p> <p>BadVAddr = 0</p> <p>Status = 3000ff10</p> <p>HI = 0</p> <p>LO = 0</p> <p>R0 [x0] = 0</p> <p>R1 [at] = 0</p> <p>R2 [v0] = a</p> <p>R3 [v1] = 0</p> <p>R4 [a0] = 4</p> <p>R5 [a1] = 7ffff73c</p> <p>R6 [a2] = 7ffff750</p> <p>R7 [a3] = 0</p> <p>R8 [t0] = 1001000c</p> <p>R9 [t1] = 0</p> <p>R10 [t2] = 3</p> <p>R11 [t3] = 3</p> <p>R12 [t4] = 0</p> <p>R13 [t5] = 5</p> <p>R14 [t6] = 5</p> <p>R15 [t7] = 0</p> <p>R16 [s0] = 6</p> <p>R17 [s1] = 4</p> <p>R18 [s2] = ffffffff</p> <p>R19 [s3] = ffffffff</p> <p>R20 [s4] = 0</p> <p>R21 [s5] = 0</p> <p>R22 [s6] = 0</p> <p>R23 [s7] = 0</p> <p>R24 [t8] = 0</p> <p>R25 [t9] = 0</p> <p>R26 [k0] = 0</p> <p>R27 [k1] = 0</p> <p>R28 [gp] = 10008000</p> <p>R29 [sp] = 7ffff738</p> <p>R30 [s8] = 0</p> <p>R31 [ra] = 400018</p>			
<p>User data segment [10000000]..[10040000]</p> <p>[10000000]..[1000ffff] 00000000</p> <p>[10010000] 00000002 ffffffff 00000006 ffffffff . . . . .</p> <p>[10010010]..[1003ffff] 00000000</p>			
<p>User Stack [7ffff738]..[80000000]</p> <p>[7ffff738] 00000004 7ffff816 . . . . .</p> <p>[7ffff740] 7ffff809 7ffff802 7ffff7f6 00000000 . . . . .</p> <p>[7ffff750] 7ffff8e1 7ffff8ba 7ffff809 7ffff8fd . . . . .</p> <p>[7ffff760] 7ffff81c 7ffff809 7ffff8e5 7ffff8b3 . . . . .</p> <p>[7ffff770] 7ffff8e2 7ffff85a 7ffff8e4 7ffff837 . . . . .</p> <p>[7ffff780] 7ffff80d 7ffff8f9 7ffff8e2 7ffff8d1 . . . . .</p> <p>[7ffff790] 7ffff8d9 7ffff8ba 7ffff8b3 7ffff8d5 . . . . .</p> <p>[7ffff7a0] 7ffff8ab 7ffff8a6 7ffff8a5 7ffff8a4 . . . . .</p> <p>[7ffff7b0] 7ffff8a29 7ffff8a0b 7ffff8e2 7ffff8c4 . . . . .</p> <p>[7ffff7c0] 7ffff859 7ffff842 7ffff82e 7ffff81f . . . . .</p> <p>[7ffff7d0] 7ffff809 7ffff802 7ffff8bc 7ffff8ab . . . . .</p> <p>[7ffff7e0] 7ffff88b 7ffff87c 7ffff861 7ffff84f . . . . .</p> <p>[7ffff7f0] 00000000 72500000 63656a6f 73612e74 . . . . .</p> <p>[7ffff800] 2f31006d 5350494d 63724100 72502f68 . . . . .</p> <p>[7ffff810] 63656a6f 3a430074 6573552f 6b2f7372 . . . . .</p> <p>[7ffff820] 6261636f 6e776f2f 756f6c43 694b2f64 . . . . .</p> <p>[7ffff830] 65736973 42415f6c 2f434f4b 45434148 . . . . .</p> <p>[7ffff840] 50455454 33362f45 6f432d34 7700706d . . . . .</p> <p>[7ffff850] 69646e69 3a433d72 6e69575c 73776f64 . . . . .</p> <p>[7ffff860] 45535500 4f525052 454c4946 5c3a433d . . . . .</p> <p>[7ffff870] 72657355 6f6b5c73 00626163 52455355 . . . . .</p> <p>[7ffff880] 454d414e 636f6b3d 55006261 44524553 . . . . .</p> <p>[7ffff890] 49414d4f 4f525f4e 4e494d41 4f525047 . . . . .</p> <p>[7ffff8a0] 454c4946 434f4b3d 55004241 44524553 . . . . .</p> <p>[7ffff8b0] 49414d4f 4f4b3d4e 00424143 3d504d54 . . . . .</p> <p>[7ffff8c0] 555c3a43 73726573 636f6b5c 415c6261 . . . . .</p> <p>[7ffff8d0] 61447070 4c5c6174 6c61636f 6d65545c . . . . .</p> <p>[7ffff8e0] 45540070 433d504d 73555c3a 5c737265 . . . . .</p> <p>[7ffff8f0] 61636f6b 70415c62 74614470 6f4c5c61 . . . . .</p> <p>[7ffff900] 5c6c6163 706d6554 73795300 526d6574 . . . . .</p> <p>[7ffff910] 3d746f6f 575c3a43 6f646e69 53007377 . . . . .</p> <p>[7ffff920] 65747379 6972446d 433d657e 4553003a . . . . .</p> <p>[7ffff930] 4f495353 4d414e4e 6f433d45 6c6f736e . . . . .</p> <p>[7ffff940] 55500065 43494c42 5c3a433d 72657355 . . . . .</p> <p>[7ffff950] 75505c73 63696c62 4d535000 6c75646f . . . . .</p> <p>[7ffff960] 74615065 3a433d68 6f72505c 6d617267 . . . . .</p>			

Copyright 1990-2017 by James Larus.  
All Rights Reserved.  
SPIM is distributed under a BSD license.  
See the file README for a full copyright notice.  
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

Figure 6: Memory and Register Values (hex) after Test 3:  $A=\{2,2,6,4\}$

## 2) Function calls

Converting C code fragment to MIPS code is conducted. In MIPS code, multiplication instructions (mult/mul) are allowed to be used.

### C CODE FRAGMENT

Below C code fragment is to be converted.

```
int main() {
    int a;
    int b;
    int result = 0;

    if(a == b)
        result = 8*(a + b);
    else
        result = compare(a, b);
    return result;
}

int compare(int a, int b)
{
    if(a<b)
        return punish(a, b);
    else
        return award(a, b);
}

int punish(int a, int b)
{ return (a-b)*2;}

int award(int a, int b)
{ return (a+b)*4;}
```

### MIPS CODE

MIPS code is written in modular structure, called functions does not modify saved registers or temporary registers but uses only argument and return registers.

If callee function calls another function (i.e. compare function) it saves function return address register for backup.

For given C code fragment, MIPS code is written as below.

```
.data
##; Input data
A: .word 5
B: .word 3
##; Output Data
Res: .word 0

.text
##      ;Base address registers
#      a_addr      : $t0
#      b_addr      : $t1
#      result_addr  : $t2
##      ;Main function registers
#      a           : $s0
#      b           : $s1
#      result      : $s2
#      arg1        : $a0
#      arg2        : $a1
#      v0          : $v0      ;return value modified after called functions are returned
##      ;Compare function registers
#      slt_compare  : $t3
#      arg1        :      ;defined in main function
```

```

#      arg2      :      ;defined in main function
#      ra_mem    : $t4  ;return address backup
##      ;Punish function registers
#      arg1      :      ;defined in main function
#      arg2      :      ;defined in main function
#      v0        :      ;modified when punish() is called
##      ;Award function registers
#      arg1      :      ;defined in main function
#      arg2      :      ;defined in main function
#      v0        :      ;modified when award() is called
#

###; Main function ##
main:
    la $t0, A          # $t0, A base address
    la $t1, B          # $t1, B base address
    la $t2, Res        # $t2, Res base address
    lw $s0, 0($t0)      # take input a
    lw $s1, 0($t1)      # take input b
    addi $s2, $0, 0     # result=0
    bne $s0, $s1, else_main # jump else_main if a=b is false
    add $s2, $s0, $s1    # result=a+b
    sll $s2, $s2, 3     # result = 8*result (result=8*(a+b))
    j done_main_if      # jump done_main_if
else_main:
    add $a0, $s0, $0     # arg1=a
    add $a1, $s1, $0     # arg2=b
    jal compare          # call compare(), $ra keeps PC address
    add $s2, $v0, $0     # result = compare(a,b)
done_main_if:
    addi $v0, $s2, 0     # return result (v0 contains result)
    sw $v0, 0($t2)      # store return value in data section
    j end               # call program ending function
##; Main function end ##

###; Compare function ## only arguments are used, return address is backed up
compare:
    slt $t3, $a0, $a1    # slt_compare=1 if a is less than b
    add $t4, $ra, $0     # store $ra in ra_mem
    beq $t3, 0, else_compare # jump else_compare if condition is false
    jal punish          # call punish()
    add $ra, $t4, $0     # backup $ra from ra_mem
    j $ra               # return caller function
else_compare:
    jal award           # call award()
    add $ra, $t4, $0     # backup $ra from ra_mem
    j $ra               # return caller function
##; Compare function end ##

###; Punish function ## only arguments are used, return register modified
punish:
    sub $v0, $a0, $a1    # return v0=a-b
    sll $v0, $v0, 1      # return v0=2*v0
    j $ra               # return caller function
##; Punish function end ##

###; Award function ## only arguments are used, return register modified
award:
    add $v0, $a0, $a1    # return v0=a+b
    sll $v0, $v0, 2      # return v0=4*v0
    j $ra               # return caller function
##; Award function end ##

end:
    li $v0, 10          # terminate program run and
    syscall             # Exit

```



## TESTS & SCREENSHOTS

For given C code fragment, MIPS code is written as below. The program is tested for the following input values:

Test 1: a=3, b=3

Test 2: a=3, b=5

Test 3: a=5, b=3

For each test, screenshots of the memory before and after running the code are given in Figure 7, Figure 8, Figure 9, Figure 10, Figure 11 and Figure 12. **Values are in Hexadecimal form.**

In test 1 with input values a=3 and b=3, result is 48 in decimal and 00000030 in hexadecimal.

In test 2 with input values a=3 and b=5, result is -4 in decimal and ffffffff in hexadecimal.

In test 3 with input values a=5 and b=3, result is 32 in decimal and 00000020 in hexadecimal.

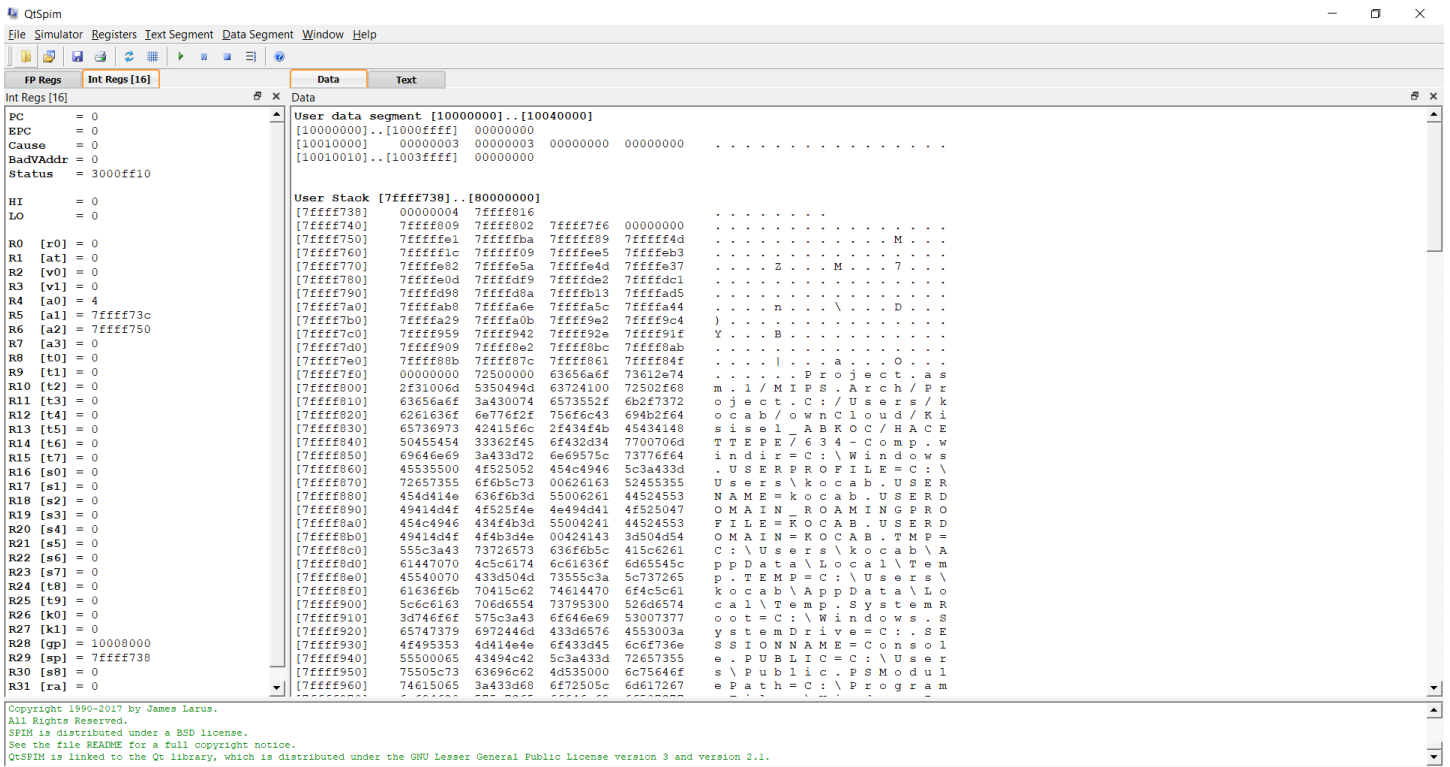


Figure 7: Memory and Register Values (hex) before Test 1:  $a=3$ ,  $b=3$

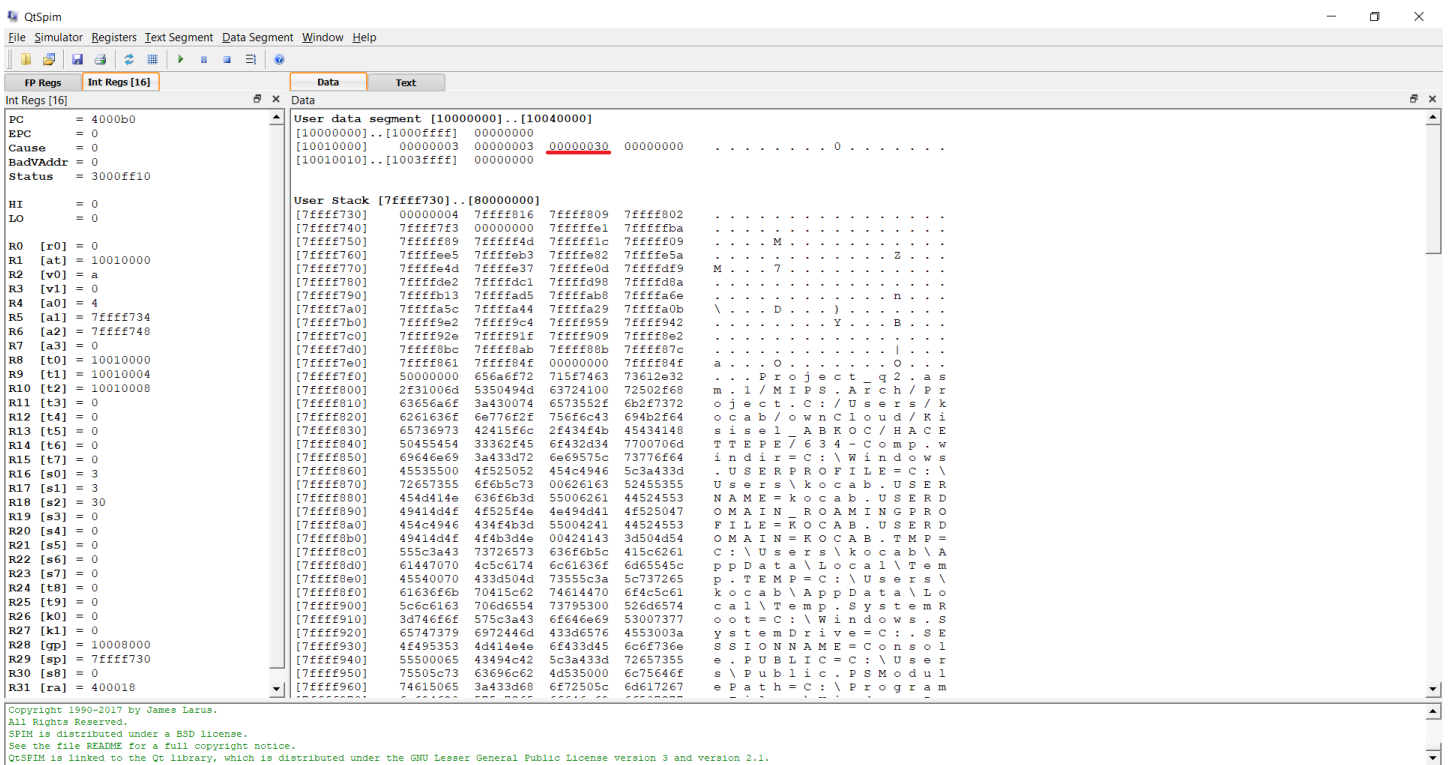


Figure 8: Memory and Register Values (hex) after Test 1:  $a=3$ ,  $b=3$  (return result in data.res)

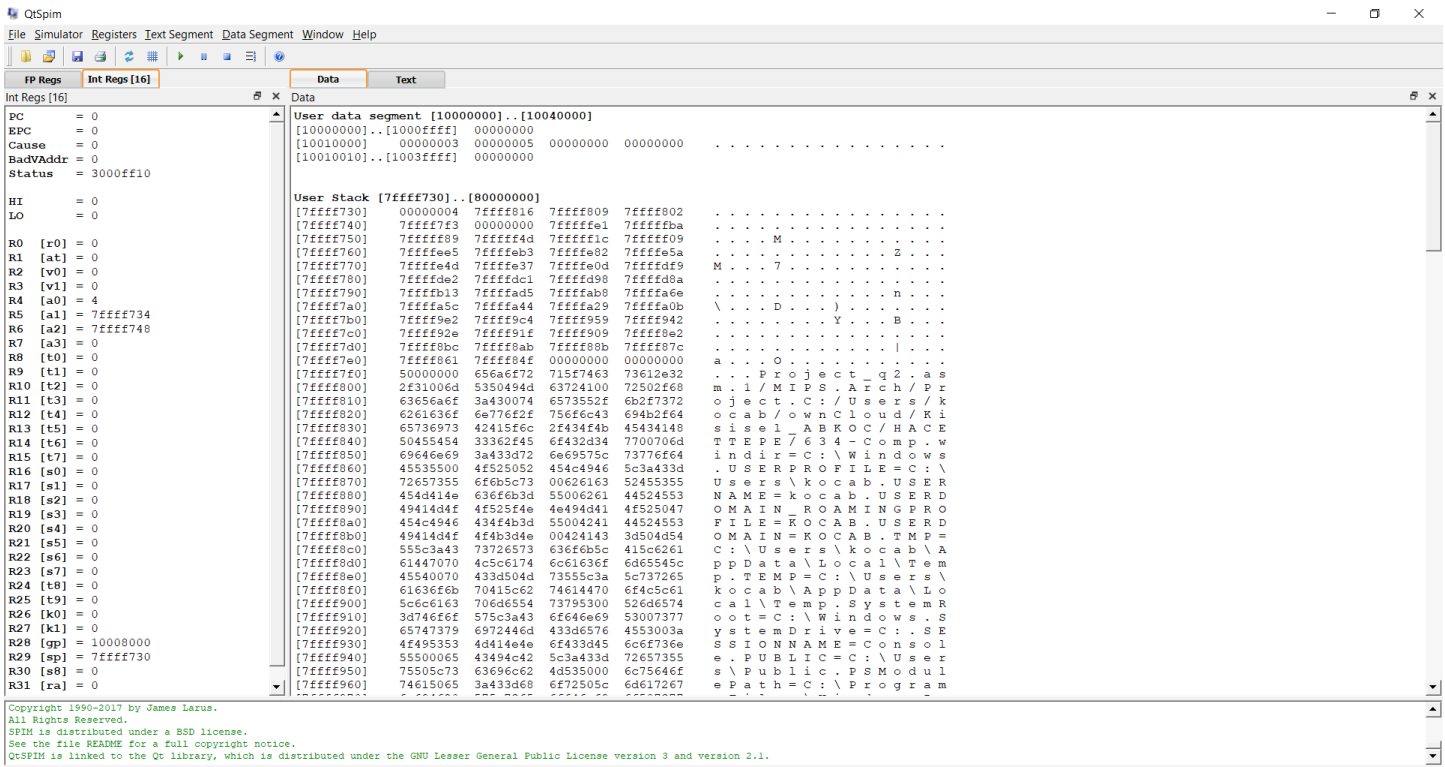


Figure 9: Memory and Register Values (hex) before Test 2:  $a=3$ ,  $b=5$

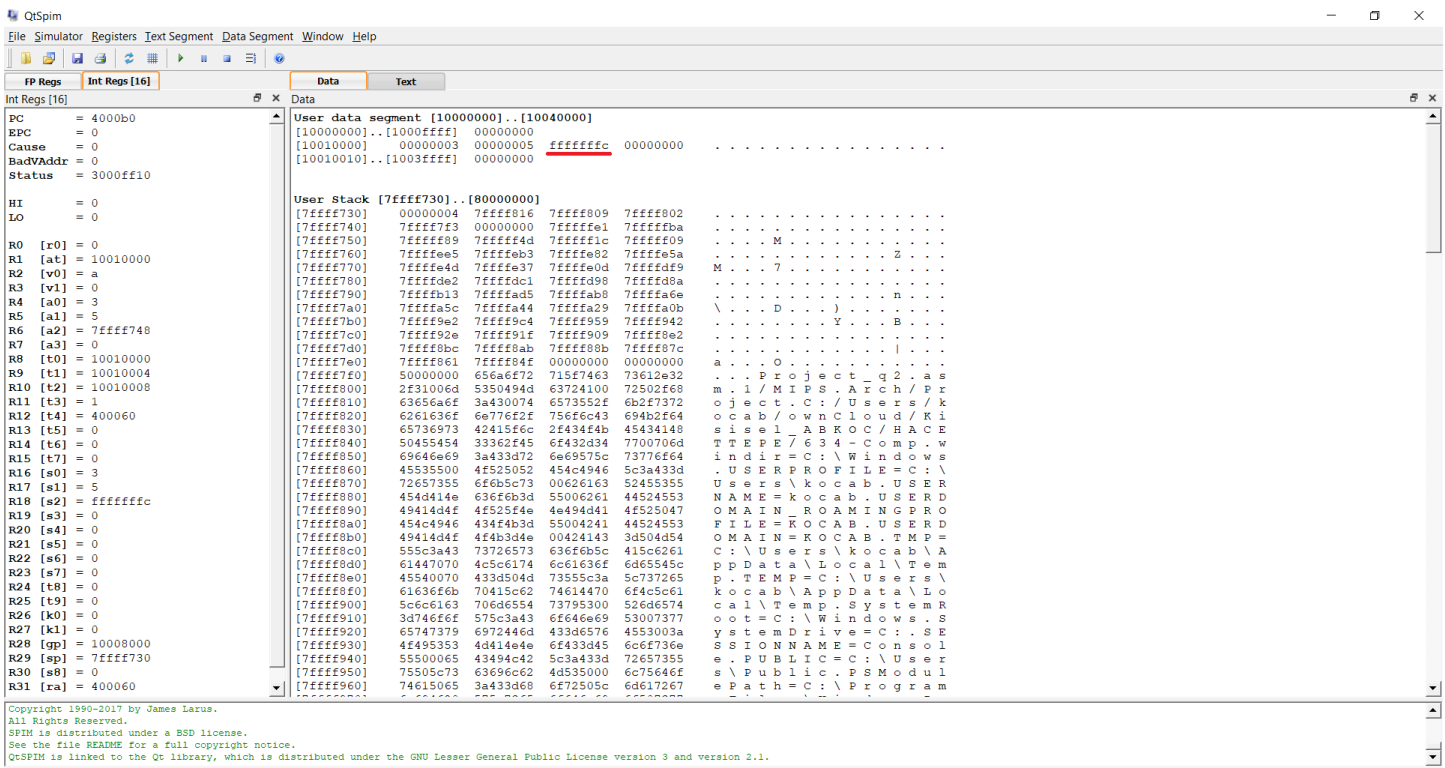


Figure 10: Memory and Register Values (hex) after Test 2:  $a=3$ ,  $b=5$  (return result in data.res)

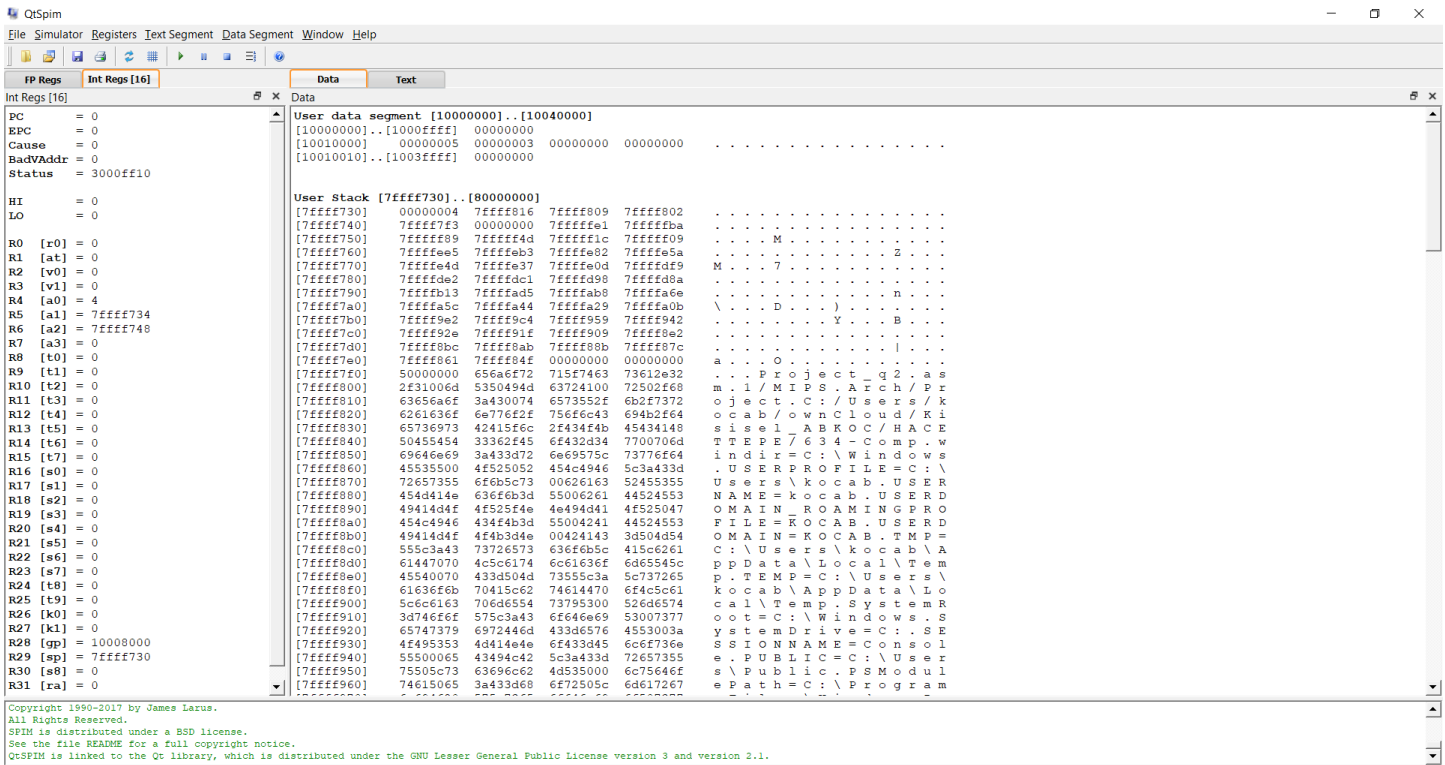


Figure 11: Memory and Register Values (hex) before Test 3:  $a=5$ ,  $b=3$

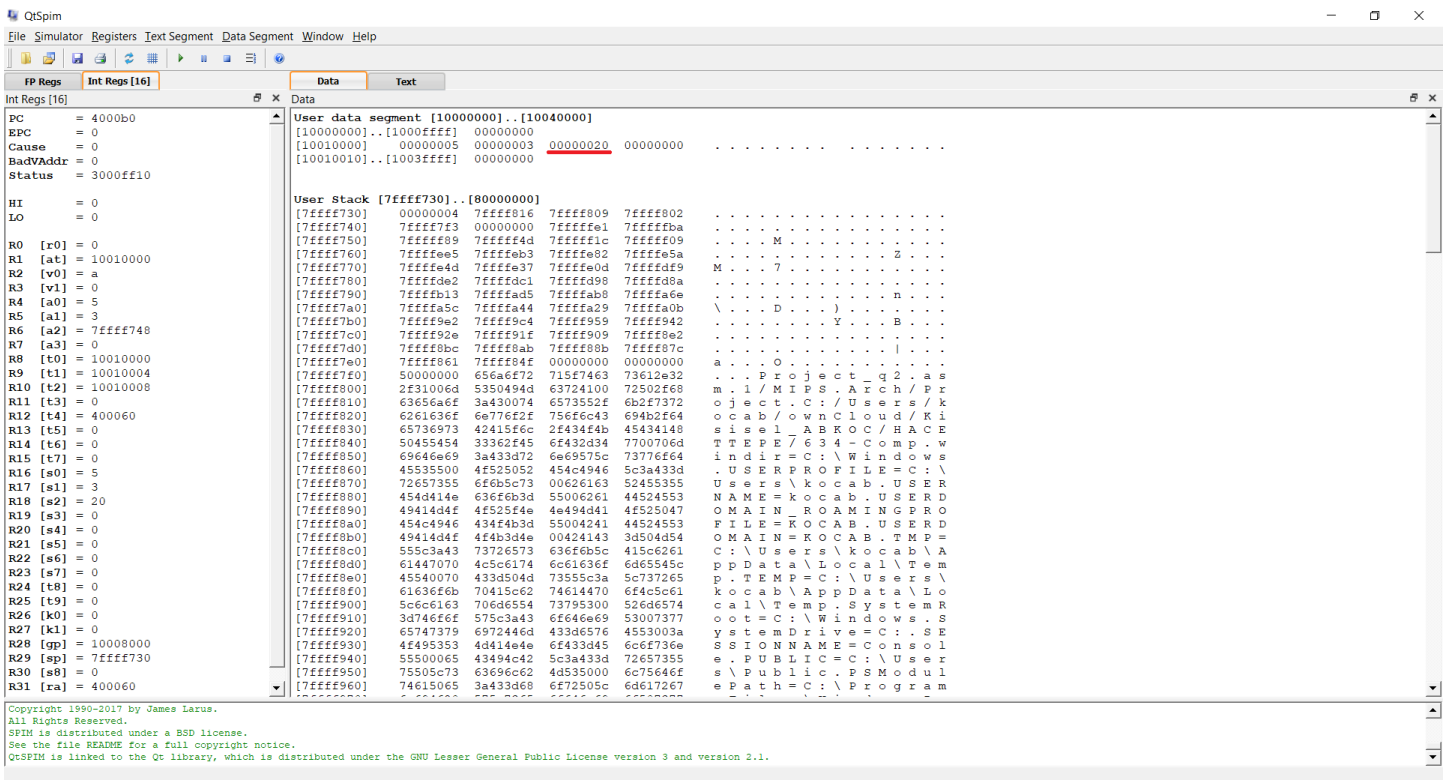


Figure 12: Memory and Register Values (hex) before Test 3:  $a=5$ ,  $b=3$  (return result in data.res)