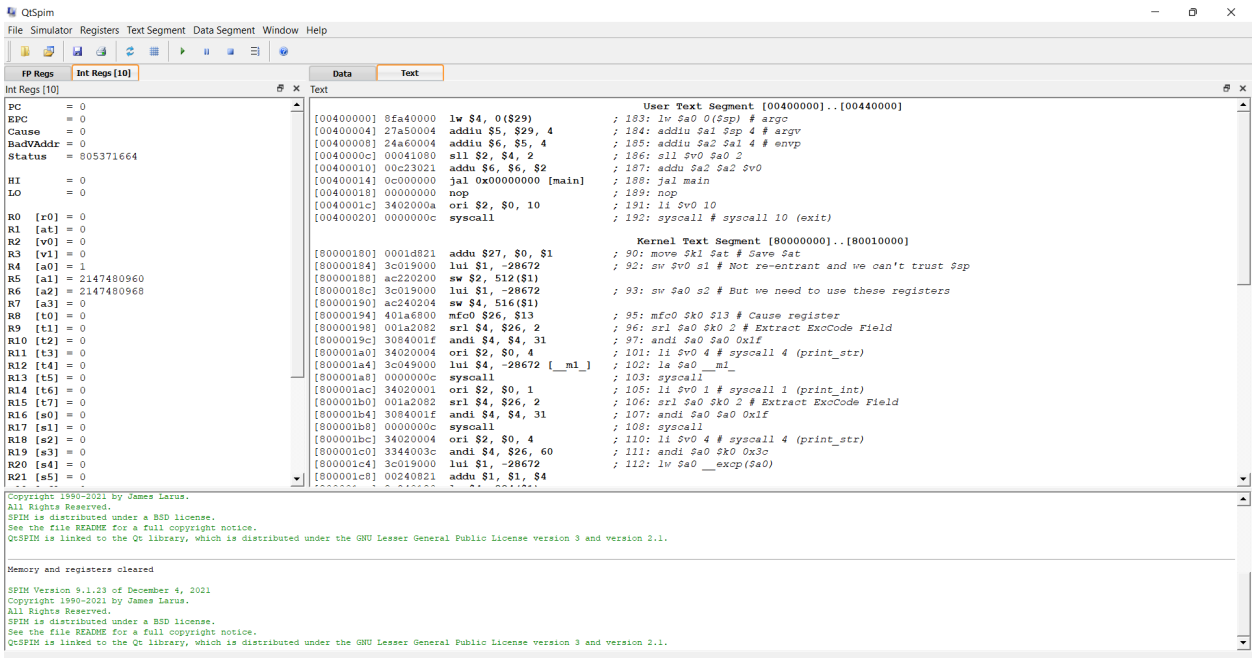
Lab1 Introduction to QtSpim

Name-Pranav Tambe
Roll No- 2106339



Interface before loading any file

Some common features

.text this tells assembler to switch to the text segment or succeeding lines contains instructions
.globl main this is assembler directive
main: start of code section
for comments
Register 0 stores value 0 initially

Q.1 Add numbers 100 and -82 and store the result in register $10.(Hint: Find 2's complement of 82 and add it to 100)

```
[00400024] 34080064  ori $8, $0, 100
OR with 0
[00400028] 2009ffae  addi $9, $0, -82
immidiate with reg 0
[0040002c] 01095020  add $10, $8, $9
```

Code from first question in stimulator after loading the file in the simulator

Analysis of code section

Instruction1 ori $8, $0, 0x64 #100

ori==OR immediate operation between value stored in register no 0 which is 0 and hexadecimal representation of 100 which is ox64 is stored in register 8.

So now register 8 has 100 in hexadecimal

Instruction2 addi $9, $0, 0xFFFFFFAE

Then addi == add immediate takes value from register 0 and adds with given value and stores in register 10 which is 16's complement of -82

Calculated by oxffffffff-ox52+ox1==oxffffae

Instruction3 add $10, $8, $9

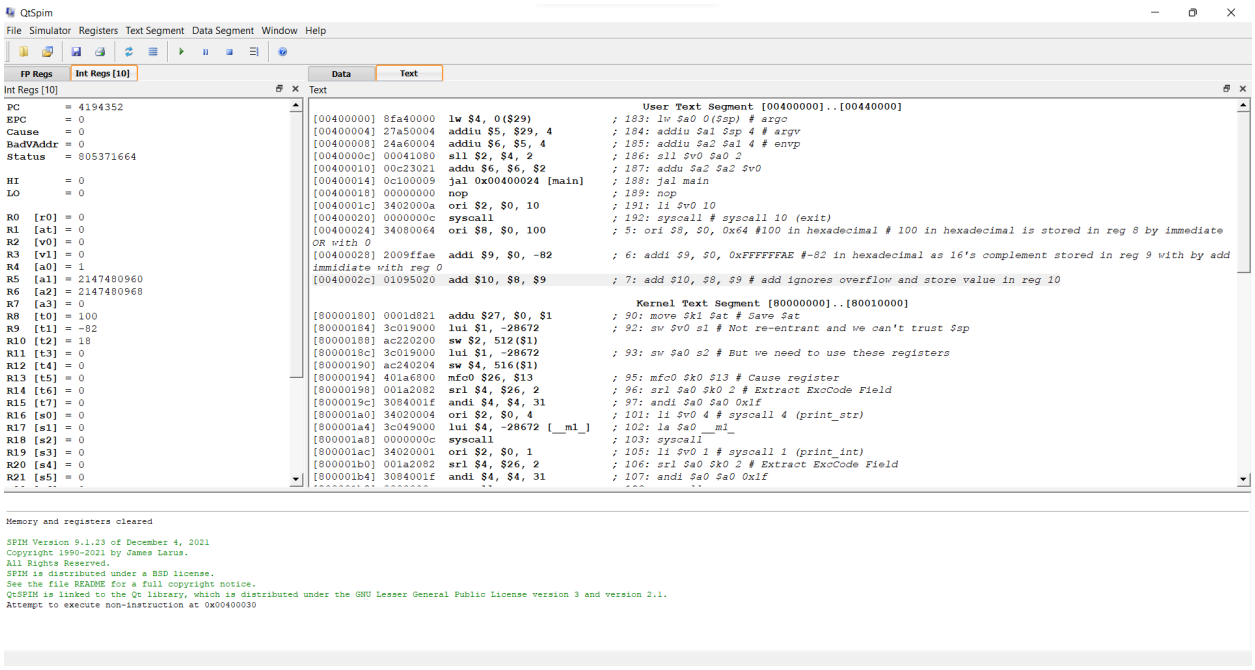
add == addition of two values from registers and stored in destination register 10

```
PC         = 4194352
EPC        = 0
Cause      = 0
BadVAddr   = 0
Status     = 805371664

HI         = 0
LO         = 0

R0   [r0] = 0
R1   [at] = 0
R2   [v0] = 0
R3   [v1] = 0
R4   [a0] = 1
R5   [a1] = 2147480960
R6   [a2] = 2147480968
R7   [a3] = 0
R8   [t0] = 100
R9   [t1] = -82
R10  [t2] = 18
R11  [t3] = 0
R12  [t4] = 0
R13  [t5] = 0
R14  [t6] = 0
R15  [t7] = 0
R16  [s0] = 0
R17  [s1] = 0
R18  [s2] = 0
R19  [s3] = 0
R20  [s4] = 0
R21  [s5] = 0
```

In screenshot values are in decimal 100 -82==18 in register 10



Q.2 b) Store FFFF in $8. Left shift it 2 times and store the result in memory location 0x10000000.

```
[00400024] 3408ffff  ori $8, $0, -1
value in reg 8
[00400028] 00084880  sll $9, $8, 2
0s from left and stores in reg 9
[0040002c] 3c0a1000  lui $10, 4096
[00400030] ad490000  sw $9, 0($10)
10 ,value in () is treated as memory address
```

Code from the second question in stimulator after loading the file in the simulator

Analysis of code section

Instruction1 ori $8, $0, 0xFFFF

ori==OR immediate operation between value stored in register no 0 which is 0 and hexadecimal oxFFFF which is stored in register 8.

So now register 8 has oxFFFFin hexadecimal

Instruction2 sll $9, $8,2

Then sll==shift logical left takes value from register 8 and left shifts the given value and stores in register 9

Instruction3 add $10, $0, 0x10000000

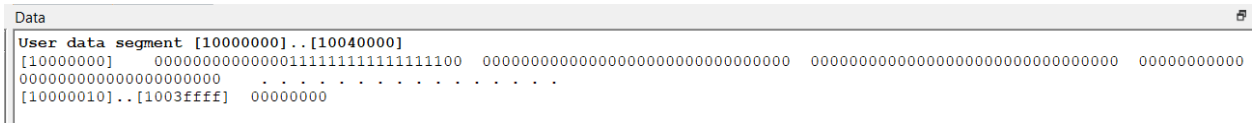
add == addition takes value from register 0 and adds with given value and stores in register 10 which is an memory address

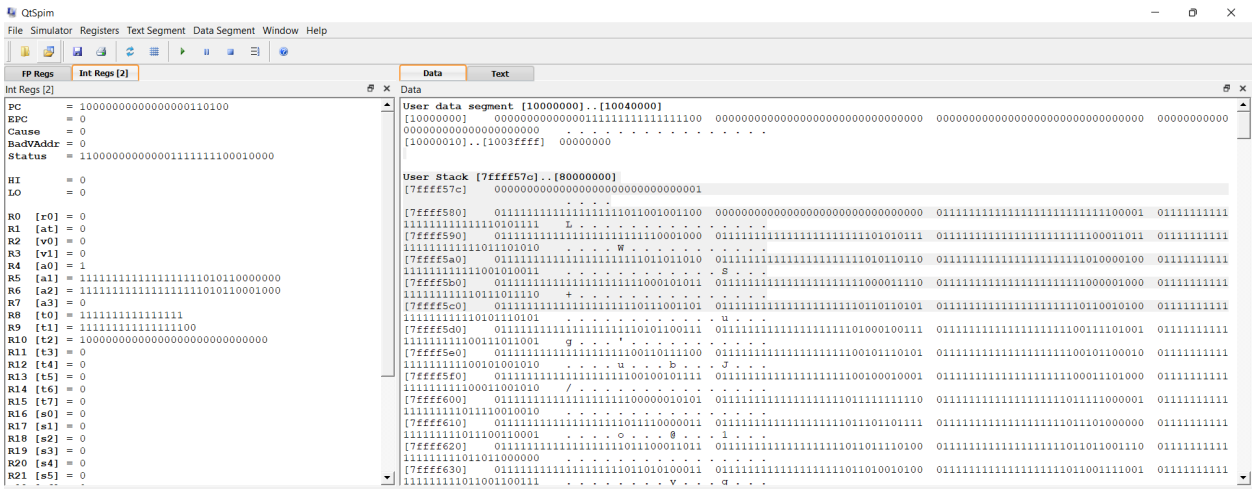
Instruction4 sw $9,($10)
sw==store word (word==16 bits) takes two argument first as a source register then the destination address where we have to store the source value
()==parenthesis tells that it's a memory address
Imm is 0 by default

In screenshot we can see the value stored at memory location
Left shift of 1111111111111111 two times 1111111111111111100 in register 9

```
R0   [r0] = 0
R1   [at] = 0
R2   [v0] = 0
R3   [v1] = 0
R4   [a0] = 1
R5   [a1] = 11111111111111111110101100000000
R6   [a2] = 11111111111111111110101100001000
R7   [a3] = 0
R8   [t0] = 1111111111111111
R9   [t1] = 11111111111111100
R10  [t2] = 10000000000000000000000000000000
R11  [t3] = 0
R12  [t4] = 0
R13  [t5] = 0
R14  [t6] = 0
R15  [t7] = 0
R16  [s0] = 0
```

Value 00000000000000111111111111111100 stored at memory location 0x10000000.





Q.3) Evaluate the expression (2x+3) ^2 where x is the content in register $10 based on exercise (a). Store the result in register $13.

```
[00400024] 34080064  ori $8, $0, 100
by immediate OR with 0
[00400028] 2009ffae  addi $9, $0, -82
9 with by add immidiate with reg 0
[0040002c] 01095020  add $10, $8, $9
[00400030] 014a5821  addu $11, $10, $10
reg 11
[00400034] 216c0003  addi $12, $11, 3
[00400038] 018c0019  multu $12, $12
in LO
[0040003c] 00006812  mflo $13
```

Code from the third question in stimulator after loading the file in the simulator

Analysis of code section

First three instructions are from part 1

We have 18 stored in register 10

Instruction4 addu $11, $10,$10

2*x

addu == addition unsigned takes value from register 10 and adds withitself and stores in register 11 so,now register 10 has 36 stored in it.

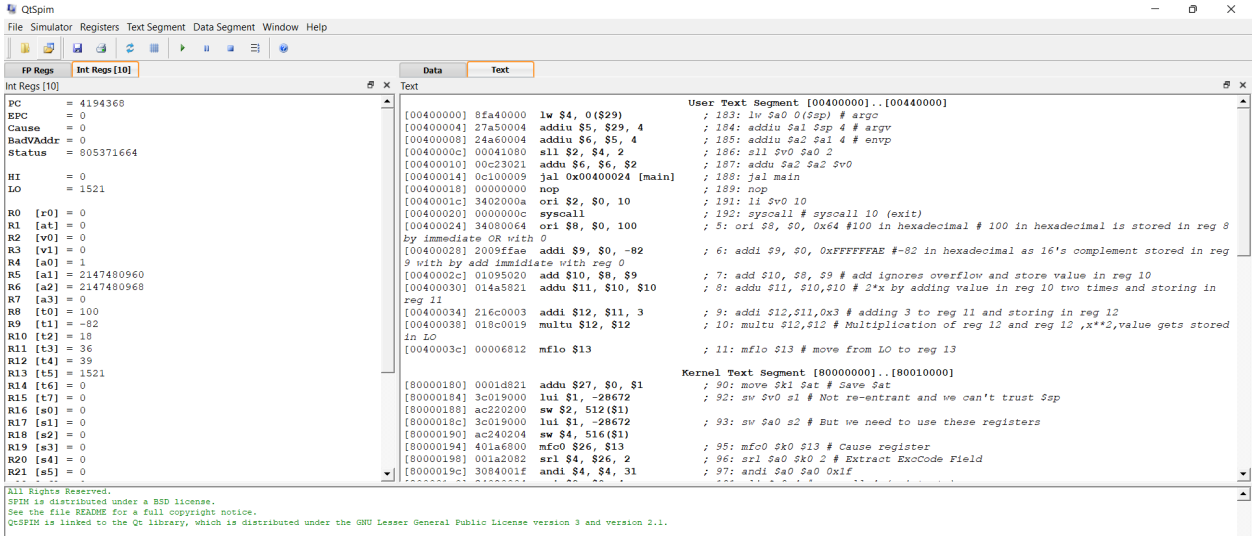
Instruction5 addi $12,$11,0x3
2*x+3
Then addi == add immediate takes value from register 11 and adds with given value which is 3 and stores in register 12 so now register 12 has 39 stored

Instruction6 multu $12,$12
(2*x+3)^2
mult == multiplication takes value from register 12 and multiplies with itself now the value generated by multiplication is stored in LO register
Values generated by multiplication or division are stored in HO and Lo register
first 16 bits in LO and if there are more bits then those 16 bits are stored in HO

Instruction6 mflo $13
So we need to move those bits from there to the destination register by mflo (Move from lo) or by mfhi(Move from HI)
So now register 13 has value 1521 which is square of 39

```
HI        = 0
LO        = 1521

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 2147480960
R6  [a2] = 2147480968
R7  [a3] = 0
R8  [t0] = 100
R9  [t1] = -82
R10 [t2] = 18
R11 [t3] = 36
R12 [t4] = 39
R13 [t5] = 1521
R14 [t6] = 0
```


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [10]
Int Regs [10]
PC = 4194368
EPC = 0
Cause = 0
BadVAddr = 0
Status = 805371664
HI = 0
LO = 1521
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 2147480960
R6 [a2] = 2147480968
R7 [a3] = 0
R8 [t0] = 100
R9 [t1] = -82
R10 [t2] = 18
R11 [t3] = 36
R12 [t4] = 39
R13 [t5] = 1521
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
Data
Text
Text
User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 34080064 ori $8, $0, 100 ; 5: ori $8, $0, 0x64 #100 in hexadecimal # 100 in hexadecimal is stored in reg 8 by immediate OR with 0
[00400028] 2009ffae addi $9, $0, -82 ; 6: addi $9, $0, 0xFFFFFFAE #-82 in hexadecimal as 16's complement stored in reg 9 with by add immidiate with reg 0
[0040002c] 01095020 add $10, $8, $9 ; 7: add $10, $8, $9 # add ignores overflow and store value in reg 10
[00400030] 014a5821 addu $11, $10, $10 ; 8: addu $11, $10,$10 # 2*x by adding value in reg 10 two times and storing in reg 11
[00400034] 216c0003 addi $12, $11, 3 ; 9: addi $12,$11,0x3 # adding 3 to reg 11 and storing in reg 12
[00400038] 018c0019 multu $12, $12 ; 10: multu $12,$12 # Multiplication of reg 12 and reg 12 ,x**2,value gets stored in LO
[0040003c] 00006812 mflo $13 ; 11: mflo $13 # move from LO to reg 13
Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save $at
[80000184] 3c019000 lui $1, -28672 ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672 ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204 sw $4, 516($1)
[80000194] 401a6800 mfc0 $26, $13 ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082 srl $4, $26, 2 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f andi $4, $4, 31 ; 97: andi $a0 $a0 0x1f
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.