## Test Case 1 – while { while .. if { while

### Source code:

begIn

inTeger a;

a:=0;

inteGer b;

b:=0;

integer c;

c:=0;

while a<6 comment: test while{ while ..if { while ...;

do

beGin

while b<5

do

Begin

writeln(c);

c:=c+1;

b:=b+1;

end;

a:=a+1;

if a=5

then

begIn

writeln("Surprise!");

while a>(b-10)

do

begIn

writeln(b);

b:=b+1;

end;

end;

b:=0;

end;

end.

### Symbol Table

Entire ST:

Lexeme Type Offset

A INTEGER 0

B INTEGER -8

C INTEGER -16

### MIPS code

#Prolog:

.text

.globl main

main:

move $fp $sp

#End of Prolog

# Read 0 into $fp pos -4

li $t0 0

sw $t0 -4($fp)

# Assign operation

lw $t0 -4($fp)

sw $t0 0($fp)

# Read 0 into $fp pos -12

li $t0 0

sw $t0 -12($fp)

# Assign operation

lw $t0 -12($fp)

sw $t0 -8($fp)

# Read 0 into $fp pos -20

li $t0 0

sw $t0 -20($fp)

# Assign operation

lw $t0 -20($fp)

sw $t0 -16($fp)

# Remember the start position for re-looping.

TopWhileLabel0:

# Read 6 into $fp pos -24

li $t0 6

sw $t0 -24($fp)

# Value in fp 0 and in fp -24

# will make a comparasion(<) and store the logical result in t0

lw $t0 0($fp)

lw $t1 -24($fp)

slt $t0 $t0 $t1

sw $t0 -28($fp)

# Check the expression to determine whether to execute the loop.

beq $t0 $zero EndWhileLabel0

# Remember the start position for re-looping.

TopWhileLabel1:

# Read 5 into $fp pos -32

li $t0 5

sw $t0 -32($fp)

# Value in fp -8 and in fp -32

# will make a comparasion(<) and store the logical result in t0

lw $t0 -8($fp)

lw $t1 -32($fp)

slt $t0 $t0 $t1

sw $t0 -36($fp)

# Check the expression to determine whether to execute the loop.

beq $t0 $zero EndWhileLabel1

# Print value from $fp pos -16

lw $a0 -16($fp)

li $v0 1

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read 1 into $fp pos -40

li $t0 1

sw $t0 -40($fp)

# Value in $fp pos -16 + value in pos -40

# And store in $fp pos -44

lw $t0 -16($fp)

lw $t1 -40($fp)

add $t0 $t0 $t1

sw $t0 -44($fp)

# Assign operation

lw $t0 -44($fp)

sw $t0 -16($fp)

# Read 1 into $fp pos -48

li $t0 1

sw $t0 -48($fp)

# Value in $fp pos -8 + value in pos -48

# And store in $fp pos -52

lw $t0 -8($fp)

lw $t1 -48($fp)

add $t0 $t0 $t1

sw $t0 -52($fp)

# Assign operation

lw $t0 -52($fp)

sw $t0 -8($fp)

# Loop in fact is the "j" to a circular expression.

j TopWhileLabel1

# The loop's break point starts from below label.

EndWhileLabel1:

# Read 1 into $fp pos -40

li $t0 1

sw $t0 -40($fp)

# Value in $fp pos 0 + value in pos -40

# And store in $fp pos -44

lw $t0 0($fp)

lw $t1 -40($fp)

add $t0 $t0 $t1

sw $t0 -44($fp)

# Assign operation

lw $t0 -44($fp)

sw $t0 0($fp)

# Read 5 into $fp pos -48

li $t0 5

sw $t0 -48($fp)

# Value in fp 0 and in fp -48

# will make a comparasion(=) and store the logical result in t0

lw $t0 0($fp)

lw $t1 -48($fp)

seq $t0 $t0 $t1

sw $t0 -52($fp)

lw $t0 -52($fp)

beq $t0 $zero EndIfLabel0

# Print string from SLabel0

la $a0 SLabel0

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Remember the start position for re-looping.

TopWhileLabel2:

# Read 10 into $fp pos -56

li $t0 10

sw $t0 -56($fp)

# Value in $fp pos -8 - value in pos -56

# And store in $fp pos -60

lw $t0 -8($fp)

lw $t1 -56($fp)

sub $t0 $t0 $t1

sw $t0 -60($fp)

# Value in fp 0 and in fp -60

# will make a comparasion(>) and store the logical result in t0

lw $t0 0($fp)

lw $t1 -60($fp)

sgt $t0 $t0 $t1

sw $t0 -64($fp)

# Check the expression to determine whether to execute the loop.

beq $t0 $zero EndWhileLabel2

# Print value from $fp pos -8

lw $a0 -8($fp)

li $v0 1

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read 1 into $fp pos -68

li $t0 1

sw $t0 -68($fp)

# Value in $fp pos -8 + value in pos -68

# And store in $fp pos -72

lw $t0 -8($fp)

lw $t1 -68($fp)

add $t0 $t0 $t1

sw $t0 -72($fp)

# Assign operation

lw $t0 -72($fp)

sw $t0 -8($fp)

# Loop in fact is the "j" to a circular expression.

j TopWhileLabel2

# The loop's break point starts from below label.

EndWhileLabel2:

EndIfLabel0:

# Read 0 into $fp pos -56

li $t0 0

sw $t0 -56($fp)

# Assign operation

lw $t0 -56($fp)

sw $t0 -8($fp)

# Loop in fact is the "j" to a circular expression.

j TopWhileLabel0

# The loop's break point starts from below label.

EndWhileLabel0:

#PostLog:

li $v0 10

syscall

.data

SLabel0: .asciiz "Surprise!"

CR: .asciiz "\n"

TL: .asciiz "TRUE"

FL: .asciiz "FALSE"

### SPIM output

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

Surprise!

5

6

7

8

9

10

11

12

13

14

25

26

27

28

29

## Test case 2 – Any other test case not included in test case 1

### Source code

begin

logical c;

c:= false or true;

writeln(c);

logical a;

a:= 1 and 0;

writeln(a);

inteGer b;

inteGer d;

inteGer f;

b:=((3+2)\*7+55/3-6 div 3);

writeln(b);

d:=99 rem 10;

f:=1;

begin

inTeger f;

f:=(d+10)\*(1-3);

writeln(f); comment: should print local f;

end;

read(f); comment: read into global f;

writeln("f now is");

writeln(f);

writeln("f >=10?");

writeln(!(f<10));

end.

### Symbol Table

Entire ST:

Lexeme Type Offset

C LOGICAL 0

A LOGICAL -16

B INTEGER -32

D INTEGER -36

F INTEGER -40

F INTEGER -112

### MIPS code

#Prolog:

.text

.globl main

main:

move $fp $sp

#End of Prolog

# Read FALSE into $fp pos -4

# 1 as TRUE and 0 as FALSE

li $t0 0

sw $t0 -4($fp)

# Read TRUE into $fp pos -8

# 1 as TRUE and 0 as FALSE

li $t0 1

sw $t0 -8($fp)

# Value in $fp pos -4 OR value in pos -8

# And store in $fp pos -12

lw $t0 -4($fp)

lw $t1 -8($fp)

or $t0 $t0 $t1

sw $t0 -12($fp)

# Assign operation

lw $t0 -12($fp)

sw $t0 0($fp)

# Print logical from $fp pos 0

lw $t0 0($fp)

la $a0 FL

beq $t0 $zero EndIfLabel0

la $a0 TL

EndIfLabel0:

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read 1 into $fp pos -20

li $t0 1

sw $t0 -20($fp)

# Read 0 into $fp pos -24

li $t0 0

sw $t0 -24($fp)

# Spcial code gen for "and" operator.

# The result could be stored to any temporary register instead of lo or hi.

lw $t0 -20($fp)

lw $t1 -24($fp)

and $t0 $t0 $t1

sw $t0 -28($fp)

# Assign operation

lw $t0 -28($fp)

sw $t0 -16($fp)

# Print logical from $fp pos -16

lw $t0 -16($fp)

la $a0 FL

beq $t0 $zero EndIfLabel1

la $a0 TL

EndIfLabel1:

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read 3 into $fp pos -44

li $t0 3

sw $t0 -44($fp)

# Read 2 into $fp pos -48

li $t0 2

sw $t0 -48($fp)

# Value in $fp pos -44 + value in pos -48

# And store in $fp pos -52

lw $t0 -44($fp)

lw $t1 -48($fp)

add $t0 $t0 $t1

sw $t0 -52($fp)

# Read 7 into $fp pos -56

li $t0 7

sw $t0 -56($fp)

# Value in $fp pos -52 \* value in pos -56

# And read lo into $t0, then store in $fp pos -60

lw $t0 -52($fp)

lw $t1 -56($fp)

mult $t0 $t1

mflo $t0

sw $t0 -60($fp)

# Read 55 into $fp pos -64

li $t0 55

sw $t0 -64($fp)

# Read 3 into $fp pos -68

li $t0 3

sw $t0 -68($fp)

# Value in $fp pos -64 / value in pos -68

# And read lo into $t0, then store in $fp pos -72

lw $t0 -64($fp)

lw $t1 -68($fp)

div $t0 $t1

mflo $t0

sw $t0 -72($fp)

# Read 6 into $fp pos -76

li $t0 6

sw $t0 -76($fp)

# Read 3 into $fp pos -80

li $t0 3

sw $t0 -80($fp)

# Value in $fp pos -76 DIV value in pos -80

# And read lo into $t0, then store in $fp pos -84

lw $t0 -76($fp)

lw $t1 -80($fp)

div $t0 $t1

mflo $t0

sw $t0 -84($fp)

# Value in $fp pos -72 - value in pos -84

# And store in $fp pos -88

lw $t0 -72($fp)

lw $t1 -84($fp)

sub $t0 $t0 $t1

sw $t0 -88($fp)

# Value in $fp pos -60 + value in pos -88

# And store in $fp pos -92

lw $t0 -60($fp)

lw $t1 -88($fp)

add $t0 $t0 $t1

sw $t0 -92($fp)

# Assign operation

lw $t0 -92($fp)

sw $t0 -32($fp)

# Print value from $fp pos -32

lw $a0 -32($fp)

li $v0 1

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read 99 into $fp pos -96

li $t0 99

sw $t0 -96($fp)

# Read 10 into $fp pos -100

li $t0 10

sw $t0 -100($fp)

# Value in $fp pos -96 REM value in pos -100

# And read hi into $t0, then store in $fp pos -104

lw $t0 -96($fp)

lw $t1 -100($fp)

div $t0 $t1

mfhi $t0

sw $t0 -104($fp)

# Assign operation

lw $t0 -104($fp)

sw $t0 -36($fp)

# Read 1 into $fp pos -108

li $t0 1

sw $t0 -108($fp)

# Assign operation

lw $t0 -108($fp)

sw $t0 -40($fp)

# Read 10 into $fp pos -116

li $t0 10

sw $t0 -116($fp)

# Value in $fp pos -36 + value in pos -116

# And store in $fp pos -120

lw $t0 -36($fp)

lw $t1 -116($fp)

add $t0 $t0 $t1

sw $t0 -120($fp)

# Read 1 into $fp pos -124

li $t0 1

sw $t0 -124($fp)

# Read 3 into $fp pos -128

li $t0 3

sw $t0 -128($fp)

# Value in $fp pos -124 - value in pos -128

# And store in $fp pos -132

lw $t0 -124($fp)

lw $t1 -128($fp)

sub $t0 $t0 $t1

sw $t0 -132($fp)

# Value in $fp pos -120 \* value in pos -132

# And read lo into $t0, then store in $fp pos -136

lw $t0 -120($fp)

lw $t1 -132($fp)

mult $t0 $t1

mflo $t0

sw $t0 -136($fp)

# Assign operation

lw $t0 -136($fp)

sw $t0 -112($fp)

# Print value from $fp pos -112

lw $a0 -112($fp)

li $v0 1

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read something into $fp pos -40

li $v0 5

syscall

sw $v0 -40($fp)

# Print string from SLabel0

la $a0 SLabel0

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Print value from $fp pos -40

lw $a0 -40($fp)

li $v0 1

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Print string from SLabel1

la $a0 SLabel1

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read 10 into $fp pos -112

li $t0 10

sw $t0 -112($fp)

# Value in fp -40 and in fp -112

# will make a comparasion(<) and store the logical result in t0

lw $t0 -40($fp)

lw $t1 -112($fp)

slt $t0 $t0 $t1

sw $t0 -116($fp)

lw $t0 -116($fp)

xor $t0 $t0 1

sw $t0 -116($fp)

# Print logical from $fp pos -116

lw $t0 -116($fp)

la $a0 FL

beq $t0 $zero EndIfLabel2

la $a0 TL

EndIfLabel2:

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

#PostLog:

li $v0 10

syscall

.data

SLabel0: .asciiz "f now is"

SLabel1: .asciiz "f >=10?"

CR: .asciiz "\n"

TL: .asciiz "TRUE"

FL: .asciiz "FALSE"

### SPIM output – red indicates as input

TRUE

FALSE

51

-38

99

f now is

99

f >=10?

TRUE

## Test Case 3 – Wrong test case

### Source code

begin

inteGer a; ,@& comment: no , allowed;

d:=9;

if a then a:=9;

logical A;

while 9 do ;

end.

### Result

Entire ST:

Lexeme Type Offset

A INTEGER 0

Non fatal errors:

D is refered without declaration.On line 3

Type not match. LHS:U RHS:I. On line 3

If occurred a logical error.On line 4

A tries to decalre twice.On line 5

While occurred a Logical error.On line 6

Lexical errors:

At Line 2 Column 13 : Illegal char ','

At Line 2 Column 14 : Illegal char '@'

At Line 2 Column 15 : Illegal char '&'

## Test Case 4 – Test the given number is prime

### Source code

begin

Integer a;

integer b;

integer c;

logicAl isPrime;

isPrime:=true;

writeln("Input 1 positive integer for testing if prime");

read(a);

b:=2;

c:=b\*b;

while !(c>a) do

begin

integer result;

result:=a rem b;

if result=0 then

begin

writeln("Not a prime!");

isPrime:=false;

end;

b:=b+1;

c:=b\*b;

end;

if isprime then

writeln("Is a prime!");

end.

### Symbol Table

Entire ST:

Lexeme Type Offset

A INTEGER 0

B INTEGER -4

C INTEGER -8

ISPRIME LOGICAL -12

RESULT INTEGER -32

### MIPS code

#Prolog:

.text

.globl main

main:

move $fp $sp

#End of Prolog

# Read TRUE into $fp pos -16

# 1 as TRUE and 0 as FALSE

li $t0 1

sw $t0 -16($fp)

# Assign operation

lw $t0 -16($fp)

sw $t0 -12($fp)

# Print string from SLabel0

la $a0 SLabel0

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read something into $fp pos 0

li $v0 5

syscall

sw $v0 0($fp)

# Read 2 into $fp pos -20

li $t0 2

sw $t0 -20($fp)

# Assign operation

lw $t0 -20($fp)

sw $t0 -4($fp)

# Value in $fp pos -4 \* value in pos -4

# And read lo into $t0, then store in $fp pos -24

lw $t0 -4($fp)

lw $t1 -4($fp)

mult $t0 $t1

mflo $t0

sw $t0 -24($fp)

# Assign operation

lw $t0 -24($fp)

sw $t0 -8($fp)

# Remember the start position for re-looping.

TopWhileLabel0:

# Value in fp -8 and in fp 0

# will make a comparasion(>) and store the logical result in t0

lw $t0 -8($fp)

lw $t1 0($fp)

sgt $t0 $t0 $t1

sw $t0 -28($fp)

lw $t0 -28($fp)

xor $t0 $t0 1

sw $t0 -28($fp)

# Check the expression to determine whether to execute the loop.

beq $t0 $zero EndWhileLabel0

# Value in $fp pos 0 REM value in pos -4

# And read hi into $t0, then store in $fp pos -36

lw $t0 0($fp)

lw $t1 -4($fp)

div $t0 $t1

mfhi $t0

sw $t0 -36($fp)

# Assign operation

lw $t0 -36($fp)

sw $t0 -32($fp)

# Read 0 into $fp pos -40

li $t0 0

sw $t0 -40($fp)

# Value in fp -32 and in fp -40

# will make a comparasion(=) and store the logical result in t0

lw $t0 -32($fp)

lw $t1 -40($fp)

seq $t0 $t0 $t1

sw $t0 -44($fp)

lw $t0 -44($fp)

beq $t0 $zero EndIfLabel0

# Print string from SLabel1

la $a0 SLabel1

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

# Read FALSE into $fp pos -48

# 1 as TRUE and 0 as FALSE

li $t0 0

sw $t0 -48($fp)

# Assign operation

lw $t0 -48($fp)

sw $t0 -12($fp)

EndIfLabel0:

# Read 1 into $fp pos -48

li $t0 1

sw $t0 -48($fp)

# Value in $fp pos -4 + value in pos -48

# And store in $fp pos -52

lw $t0 -4($fp)

lw $t1 -48($fp)

add $t0 $t0 $t1

sw $t0 -52($fp)

# Assign operation

lw $t0 -52($fp)

sw $t0 -4($fp)

# Value in $fp pos -4 \* value in pos -4

# And read lo into $t0, then store in $fp pos -56

lw $t0 -4($fp)

lw $t1 -4($fp)

mult $t0 $t1

mflo $t0

sw $t0 -56($fp)

# Assign operation

lw $t0 -56($fp)

sw $t0 -8($fp)

# Loop in fact is the "j" to a circular expression.

j TopWhileLabel0

# The loop's break point starts from below label.

EndWhileLabel0:

lw $t0 -12($fp)

beq $t0 $zero EndIfLabel1

# Print string from SLabel2

la $a0 SLabel2

li $v0 4

syscall

# Print a carriage return.

la $a0 CR

li $v0 4

syscall

EndIfLabel1:

#PostLog:

li $v0 10

syscall

.data

SLabel0: .asciiz "Input 1 positive integer for testing if prime"

SLabel1: .asciiz "Not a prime!"

SLabel2: .asciiz "Is a prime!"

CR: .asciiz "\n"

TL: .asciiz "TRUE"

FL: .asciiz "FALSE"

### SPIM output – red indicates as input

(1)Input 1 positive integer for testing if prime

96

Not a prime!

Not a prime!

Not a prime!

Not a prime!

Not a prime!

(2) Input 1 positive integer for testing if prime

97

Is a prime!

(3) Input 1 positive integer for testing if prime

17

Is a prime!

(4) Input 1 positive integer for testing if prime

21

Not a prime!

## Test case 5 – Lexical Analyzer test

### Input code (Illegal for parser, skip parse.)

begin

a :=3;

integER b\_3

string a33 comment : , is illegal;

logical d,e,f;d3\_ true : = comment: illegal": =";

a=a+b;@ & commeNt:@ & also illegal;

c<(2+"hello");

d>false;

!e;

if d!=e then

while a= b rem c do 3 div 3;

c:=2+3/4\*5;

1 or 2; 3 and 4;

write read writeln

end.

### Analyzer output

7 BEGIN 1 A 19 :=

2 3 16 ; 3 INTEGER

1 B\_3 3 STRING 1 A33

3 LOGICAL 1 D 1 E

1 F 16 ; 1 D3\_

2 TRUE 6 = 1 A

6 = 1 A 4 +

1 B 16 ; 1 C

6 < 14 ( 2 2

4 + 2 "hello" 15 )

16 ; 1 D 6 >

2 FALSE 16 ; 17 !

1 E 16 ; 9 IF

1 D 6 != 1 E

10 THEN 11 WHILE 1 A

6 = 1 B 5 REM

1 C 12 DO 2 3

5 DIV 2 3 16 ;

1 C 19 := 2 2

4 + 2 3 5 /

2 4 5 \* 2 5

16 ; 2 1 4 OR

2 2 16 ; 2 3

5 AND 2 4 16 ;

13 WRITE 13 READ 13 WRITELN

8 END 18 . -2 EOF$

At Line 5 Column 11 : Illegal char ','

At Line 5 Column 13 : Illegal char ','

At Line 5 Column 26 : Illegal char ':'

At Line 6 Column 8 : Illegal char '@'

At Line 6 Column 10 : Illegal char '&'