Μεταφραστής γλώσσας CutePy σε Assembly RISC-V

Αλέξανδρος Αγαπητός Χριστοδούλου ΑΜ: 4839

Βασίλειος Ιωάννης Μπουζαμπαλίδης ΑΜ: 4744

ΜΥΥ802- Μεταφραστές

Ακαδημαϊκό Έτος 2023-2024

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY OF IOANNINA

Περιεχόμενα

Εισαγωγή

Η παρούσα αναφορά περιγράφει τη διαδικασία υλοποίησης μεταφραστή της γλώσσας CutePy, σε όλα της τα στάδια και παρέχει εξηγήσεις του κώδικα που έχει υλοποιηθεί σε γλώσσα Python. Η αναφορά είναι δομημένη σε διάφορες ενότητες, καθεμία από τις οποίες αντιστοιχεί σε ένα στάδιο της υλοποίησης.

Η ανάλυση της λειτουργίας του προγράμματος πραγματοποιείται μέσω tests της γλώσσας cutePy.

Οι δοκιμές αυτές καθιστούν σαφή τον χειρισμό τόσο των ορθά γραμμένων, όσο και εκείνων που περιέχουν σκόπιμα λάθη στους κανόνες της γλώσσας. Τέτοια tests χρησιμοποιούνται στα πρώτα δύο στάδια, δηλαδή στον **λεκτικό** και τον **συντακτικό** αναλυτή, ώστε να καταδειχθεί πώς ο κώδικας αντιμετωπίζει όλες τις περιπτώσεις.

Σε επόμενα στάδια μετάφρασης, που αναφέρονται στον **ενδιάμεσο κώδικα**, τον **πίνακα συμβόλων** και τον **τελικό κώδικα**, έχει χρησιμοποιηθεί μία δοκιμή που είναι ικανή να επιβεβαιώσει την ορθή λειτουργία καθενός από αυτά.

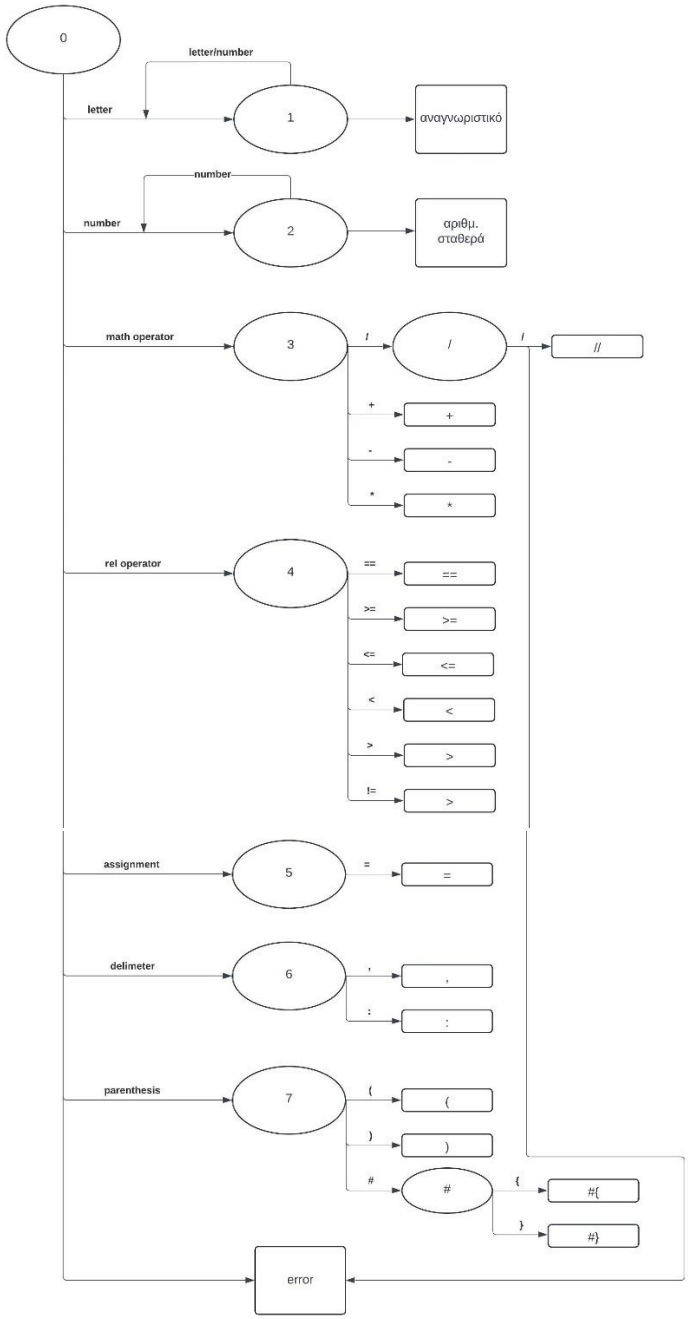
Όσων αφορά τις δοκιμές για τον έλεγχο των σταδίων του μεταφραστή έχουν γίνει τα εξής βήματα:

1. Στον παραδοτέο κώδικα συμπεριλήφθηκαν 5 tests συνολικά.
2. Στην αναφορά, οι δοκιμές που συμπεριλαμβάνουν ορθά γραμμένα test σε γλώσσα CutePy είναι ίδια με αυτά του παραδοτέου κώδικα για διευκόλυνση της διασταύρωσης αποτελεσμάτων.
3. Τα test που ελέγχουν τις περιπτώσεις σφάλματος στον **λεκτικό** και **συντακτικό** αναλυτή δεν συμπεριλήφθηκαν στα παραδοτέα, αλλά χρησιμοποιούνται στην αναφορά ως ένδειξη ορθής λειτουργίας των παραπάνω.

Το προγραμματιστικό περιβάλλον που χρησιμοποιήθηκε είναι το **Visual Studio Code**.

Λεκτικός Αναλυτής

Αρχικά παρουσιάζεται το αυτόματο καταστάσεων που χειρίζεται ο **λεκτικός** αναλυτής, για τις περιπτώσεις που αναγνωρίζει η γραμματική CutePy ως λεκτικές μονάδες:



Η πρώτη πειραματική δοκιμή γίνεται στο test **max3** στο οποίο έχει προστεθεί ο χαρακτήρας “$”, ο οποίος δεν υποστηρίζεται από την γραμματική CutePy.

```
2
3  def max3(x,y,z):
4      #{
5          #int m
6          global counterFunctionCalls
7          counterFunctionCalls = counterFunctionCalls + 1
8          if x>y and x>z:
9              $ = x
10
11         elif y>x and y>z:
12             m = y
13         else:
14             m = z
15
16         return m
17     #}
```

Όπως ορθά φαίνεται παρακάτω, ο λεκτικός αναλυτής αναγνωρίζει το σφάλμα και επιστρέφει μήνυμα λάθους, ενημερώνοντας το χρήστη ότι ο χαρακτήρας δεν υποστηρίζεται.

```
-----------------------
Lectical Analysis
-----------------------
def  family: "keyword", line: 1
max3  family: "id", line: 1
(  family: "groupSymbol", line: 1
x  family: "id", line: 1
,  family: "delimiters", line: 1
y  family: "id", line: 1
,  family: "delimiters", line: 1
z  family: "id", line: 1
)  family: "groupSymbol", line: 1
:  family: "delimiters", line: 1
#{  family: "groupSymbol", line: 2
#int  family: "keyword", line: 3
m  family: "id", line: 3
global  family: "keyword", line: 4
counterFunctionCalls  family: "id", line: 4
counterFunctionCalls  family: "id", line: 5
=  family: "assignment", line: 5
counterFunctionCalls  family: "id", line: 5
+  family: "mathOperator", line: 5
1  family: "number", line: 5
if  family: "keyword", line: 6
x  family: "id", line: 6
>  family: "relOperator", line: 6
y  family: "id", line: 6
and  family: "bool_operator", line: 6
x  family: "id", line: 6
>  family: "relOperator", line: 6
z  family: "id", line: 6
:  family: "delimiters", line: 6
Error: Illegal character used: "$" character not supported, line: 7
```

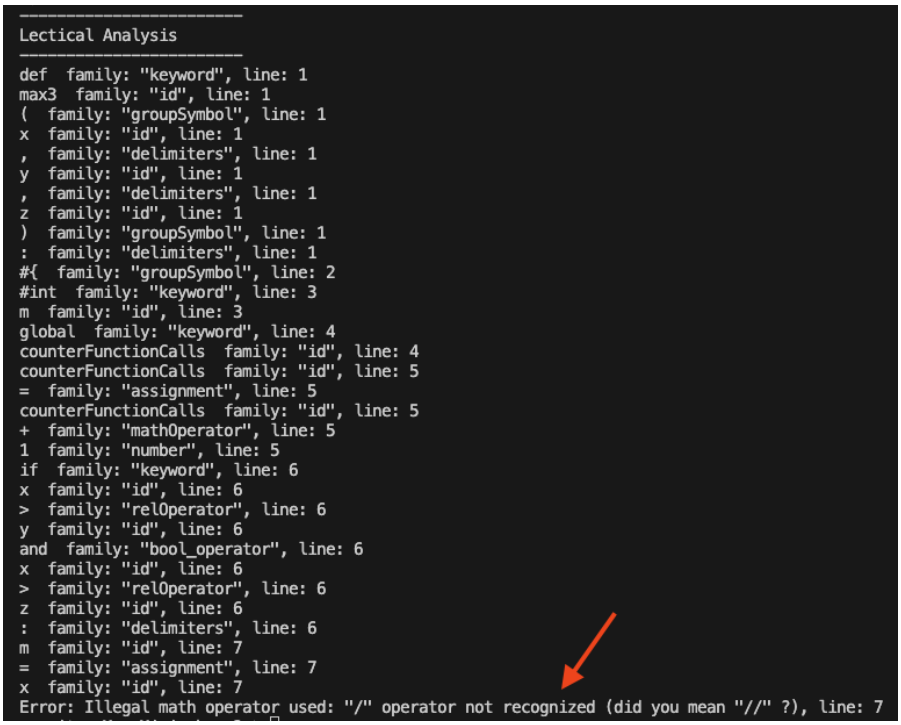
Η επόμενη πειραματική δοκιμή ελέγχει την εσφαλμένη χρήση του χαρακτήρα “/” ως μονή εμφάνιση που δεν υποστηρίζεται από τη γλώσσα, ενώ η διπλή εμφάνιση “//” υποστηρίζεται. Θα γίνει με βάση το ίδιο test **max3** για διευκόλυνση.

```
def max3(x,y,z):
#{
    #int m
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    if x>y and x>z:
        m = x/z

    elif y>x and y>z:
        m = y
    else:
        m = z

    return m
#}
```

Όπως ορθά φαίνεται παρακάτω, ο λεκτικός αναλυτής αναγνωρίζει το σφάλμα και επιστρέφει μήνυμα λάθους, ενημερώνοντας το χρήστη ότι ο τελεστής δεν αναγνωρίζεται. Επίσης, τον ρωτάει για πιθανό σφάλμα εκ παραδρομής και του προσφέρει πιθανή λύση.



Η τελευταία πειραματική δοκιμή για τον **λεκτικό** αναλυτή θα γίνει, με σκοπό τον έλεγχο με ορθή χρήση της γραμματικής για ίδιο test **max3.**

```
def max3(x,y,z):
#{
    #int m
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    if x>y and x>z:
        m = x

    elif y>x and y>z:
        m = y
    else:
        m = z

    return m
#}
```

Παρακάτω φαίνεται ότι ο λεκτικός αναλυτής δεν επέστρεψε κανένα σφάλμα, αναγνώρισε όλες τις λεκτικές μονάδες και εκτελέστηκε επιτυχώς.

```
Lectical Analysis
def  family: "keyword", line: 1
max3  family: "id", line: 1
(  family: "groupSymbol", line: 1
x  family: "id", line: 1
,  family: "delimiters", line: 1
y  family: "id", line: 1
,  family: "delimiters", line: 1
z  family: "id", line: 1
)  family: "groupSymbol", line: 1
:  family: "delimiters", line: 1
#{  family: "groupSymbol", line: 2
#int  family: "keyword", line: 3
m  family: "id", line: 3
global  family: "keyword", line: 4
counterFunctionCalls  family: "id", line: 4
counterFunctionCalls  family: "id", line: 5
=  family: "assignment", line: 5
counterFunctionCalls  family: "id", line: 5
+  family: "mathOperator", line: 5
1  family: "number", line: 5
if  family: "keyword", line: 6
x  family: "id", line: 6
>  family: "relOperator", line: 6
y  family: "id", line: 6
and  family: "bool_operator", line: 6
x  family: "id", line: 6
>  family: "relOperator", line: 6
z  family: "id", line: 6
:  family: "delimiters", line: 6
m  family: "id", line: 7
=  family: "assignment", line: 7
x  family: "id", line: 7
elif  family: "keyword", line: 9
y  family: "id", line: 9
>  family: "relOperator", line: 9
x  family: "id", line: 9
and  family: "bool_operator", line: 9
y  family: "id", line: 9
>  family: "relOperator", line: 9
z  family: "id", line: 9
:  family: "delimiters", line: 9
m  family: "id", line: 10
=  family: "assignment", line: 10
y  family: "id", line: 10
else  family: "keyword", line: 11
:  family: "delimiters", line: 11
m  family: "id", line: 12
=  family: "assignment", line: 12
z  family: "id", line: 12
return  family: "keyword", line: 14
m  family: "id", line: 14
#}  family: "groupSymbol", line: 15
```

Συντακτικός Αναλυτής

Για τον έλεγχο του **συντακτικού** αναλυτή θα παρατεθούν 3 tests. Τα 2 από αυτα θα ελέγχουν περιπτώσεις εσφαλμένης χρήσης της CutePy και το τελευταίο θα είναι πλήρως λειτουργικό.
Οι περιπτώσεις λάθους ελέγχονται μέσα στο test **max3.**

Για το πρώτο test θα ελέγξουμε αν ο **συντακτικός** αναλυτής αναγνωρίζει τιμή που είναι πάνω από το επιτρεπτό όριο ακεραίου που επιτρέπει η γραμματική.

```
def max3(x,y,z):
#{
    #int m
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    if x>y and x>z:
        m = 32768

    elif y>x and y>z:
        m = y
    else:
        m = z

    return m
#}
```

Όπως ορθά φαίνεται παρακάτω, ο **συντακτικός** αναγνωρίζει το σφάλμα, σταματά και δεν προχωράει σε επόμενο έλεγχο, εκτυπώνοντας μήνυμα λάθους στην οθόνη που υποδηλώνει ότι η τιμή που καταχωρήθηκε είναι εκτός ορίων.

```
-----------------------
Syntax Analysis
-----------------------

SyntaxError: invalid syntax
Int "32768" out of bounds, line: 7
```

Στη συνέχεια θα ελέγξουμε τη λανθασμένη δήλωση μεταβλητής με τη χρήση “==” το οποίο αποτελεί Boolean operator και δεν θα έπρεπε να συνεχίζει σε επόμενο έλεγχο. Η σωστή ανάθεση μεταβλητής θα πρέπει να γίνεται με τον τελεστή “=” όπως δηλώνει η CutePy

```
def max3(x,y,z):
#{
    #int m
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    if x>y and x>z:
        m == x

    elif y>x and y>z:
        m = y
    else:
        m = z

    return m
#}
```

Όπως ορθά φαίνεται παρακάτω, ο **συντακτικός** αναγνωρίζει το σφάλμα, σταματά και δεν προχωράει σε επόμενο έλεγχο, εκτυπώνοντας μήνυμα λάθους στην οθόνη και ενημερώνει τον χρήστη για πιθανό λάθος απροσεξίας, δίνοντας μια πιθανή λύση.

```
-------------------------
Syntax Analysis
-------------------------

SyntaxError: invalid syntax
Expected assignment operator "=" for declaring variable, line: 7
```

Η τελευταία πειραματική δοκιμή για τον **συντακτικό** αναλυτή θα γίνει, με σκοπό τον έλεγχο με ορθή χρήση της γραμματικής για ίδιο test με αυτό στα παραδοτέα **IsPrime.**

```
def isPrime(x):
#{
    ## declarations for isPrime ##
    #int i

    def divides(x,y):
    #{
        ## body of divides ##
        global counterFunctionCalls
        counterFunctionCalls = counterFunctionCalls + 1
        if y == (y//x)*x:
            return 1
        else:
            return 0
    #}

    ## body of isPrime ##
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    i = 2
    while i < x :
    #{
        if divides(i,x) == 1:
            return 0
        i = i + 1
    #}
    return 1
#}
```

Σε αυτό τον έλεγχο όλες οι λειτουργίες που επιτελούνται είναι αναγνωρίσιμες από τη γραμματική, και ο **συντακτικός** αναλυτής δεν επιστρέφει κανένα μήνυμα λάθους, όπως είναι αναμενόμενο. Τέλος όταν όλα πάνε καλά εμφανίζεται μήνυμα “Compilation Done” που σηματοδοτεί το τέλος της ανάλυσης.

```
-------------------------
Syntax Analysis
-------------------------
Compilation Done
```

Ενδιάμεσος Κώδικας

Ο έλεγχος του **ενδιάμεσου** κώδικα θα γίνει με χρήση του test **IsPrime,** το οποίο θα υπάρχει και στα παραδοτέα test. Σε αυτή την ενότητα θα αναλύσουμε τη διαδικασία δημιουργίας τετράδων και τι εξυπηρετεί η καθεμία, ώστε να διαπιστώσουμε ότι ο ενδιάμεσος κώδικας έχει υλοποιηθεί σωστά. Οι εξηγήσεις βασίζονται πάνω στην ανάλυση που γίνεται στο test που προαναφέρθηκε και ακολουθεί την ορθή εκτέλεση του **λεκτικού** και **συντακτικού** αναλυτή.
Το αρχείο που αποθηκεύεται ο ενδιάμεσος κώδικας ονομάστηκε “**inter_code.int”**.

****Σημείωση****
Σε περίπτωση απουσίας **main** συνάρτησης δημιουργείται ενδιάμεσος κώδικας, αλλά επειδή η συνάρτηση προς έλεγχο δεν καλείται σε κάποια **main**, στην αρχή της παραγωγής **ενδιάμεσου** κώδικα προστίθεται μία τετράδα **jump, _, _, last_line_of_code**.

Παρατίθεται ο κώδικας του **IsPrime** χωρίς την **main** για να δείξουμε τα προαναφερθέντα:

```
def isPrime(x):
#{
    ## declarations for isPrime ##
    #int i

    def divides(x,y):
    #{
        ## body of divides ##
        global counterFunctionCalls
        counterFunctionCalls = counterFunctionCalls + 1
        if y == (y//x)*x:
            return 1
        else:
            return 0
    #}

    ## body of isPrime ##
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    i = 2
    while i < x :
    #{
        if divides(i,x) == 1:
            return 0
        i = i + 1
    #}
    return 1
#}
```

Στην ετικέτα 100 παρατηρείται η τετράδα **jump, _ ,_ ,128** που εμποδίζει να τρέχει το test.

```
100: jump, _, _, 128
101: begin_block, divides, _, _
102: +, counterFunctionCalls, 1, T$0
103: =, T$0, _, counterFunctionCalls
104: //, y, x, T$1
105: *, T$1, x, T$2
106: ==, y, T$2, 108
107: jump, _, _, 109
108: ret, 1, _, _
109: ret, 0, _, _
110: end_block, divides, _, _
111: begin_block, isPrime, _, _
112: +, counterFunctionCalls, 1, T$3
113: =, T$3, _, counterFunctionCalls
114: =, 2, _, i
115: <, i, x, 117
116: jump, _, _, 127
117: par, i, CV, divides
118: par, x, CV, divides
119: par, T$4, RET, divides
120: call, _, _, divides
121: ==, T$4, 1, 123
122: jump, _, _, 127
123: ret, 0, _, _
124: +, i, 1, T$5
125: =, T$5, _, i
126: jump, _, _, 115
127: ret, 1, _, _
128: end_block, isPrime, _, _
```

Στη συνέχεια παρατίθεται ο κώδικας του **IsPrime** μαζί με την **main** που την καλεί:

```
def isPrime(x):
#{
    ## declarations for isPrime ##
    #int i

    def divides(x,y):
    #{
        ## body of divides ##
        global counterFunctionCalls
        counterFunctionCalls = counterFunctionCalls + 1
        if y == (y//x)*x:
            return 1
        else:
            return 0
    #}

    ## body of isPrime ##
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    i = 2
    while i < x :
    #{
        if divides(i,x) == 1:
            return 0
        i = i + 1
    #}
    return 1
#}

#def main
#int i
counterFunctionCalls = 0

i = int(input())
print(i)


i = 1600
while i<=2000:
#{
    i = i + 400
#}

i=1
while i<=12:
#{
    print(isPrime(i))
    i = i + 1
#}

print(counterFunctionCalls)
```

Τα αποτελέσματα εκτέλεσης εμφανίζονται στο αρχείο του ενδιάμεσου κώδικα που θα παραθέσουμε και θα εξηγήσουμε παρακάτω:

```
100: jump, _, _, main
101: begin_block, divides, _, _
102: +, counterFunctionCalls, 1, T$0
103: =, T$0, _, counterFunctionCalls
104: //, y, x, T$1
105: *, T$1, x, T$2
106: ==, y, T$2, 108
107: jump, _, _, 109
108: ret, 1, _, _
109: ret, 0, _, _
110: end_block, divides, _, _
111: begin_block, isPrime, _, _
112: +, counterFunctionCalls, 1, T$3
113: =, T$3, _, counterFunctionCalls
114: =, 2, _, i
115: <, i, x, 117
116: jump, _, _, 127
117: par, i, CV, divides
118: par, x, CV, divides
119: par, T$4, RET, divides
120: call, _, _, divides
121: ==, T$4, 1, 123
122: jump, _, _, 127
123: ret, 0, _, _
124: +, i, 1, T$5
125: =, T$5, _, i
126: jump, _, _, 115
127: ret, 1, _, _
128: end_block, isPrime, _, _
129: begin_block, main, _, _
130: =, 0, _, counterFunctionCalls
131: in, _, _, i
132: out, i, _, _
133: =, 1600, _, i
134: <=, i, 2000, 136
135: jump, _, _, 139
136: +, i, 400, T$6
137: =, T$6, _, i
138: jump, _, _, 134
139: =, 1, _, i
140: <=, i, 12, 142
141: jump, _, _, 149
142: par, i, CV, isPrime
143: par, T$7, RET, isPrime
144: call, _, _, isPrime
145: out, T$7, _, _
146: +, i, 1, T$8
147: =, T$8, _, i
148: jump, _, _, 140
149: out, counterFunctionCalls, _, _
150: halt, _, _, _
151: end_block, main, _, _
```

Εξήγηση κάθε τετράδας που δημιουργείται

101: begin_block, divides, _, _

Έναρξη νέας συνάρτησης με όνομα "divides".

102: +, counterFunctionCalls, 1, T$0

Αναγνώριση της global μεταβλητής και πρόσθεση του αριθμού 1 και αποθήκευση του αποτελέσματος στον προσωρινή μεταβλητή T$0.

103: =, T$0, _, counterFunctionCalls

Αντιγραφή της τιμής από τη μεταβλητή T$0 στην μεταβλητή counterFunctionCalls.

104: //, y, x, T$1

Μετάβαση στο if block και διαίρεση της τιμής του y με την τιμή του x και αποθήκευση του αποτελέσματος στην προσωρινή μεταβλητή T$1.

105: *, T$1, x, T$2

Πολλαπλασιασμός της τιμής της T$1 με την τιμή του x και αποθήκευση του αποτελέσματος στη καινούργια μεταβλητή T$2.

106: ==, y, T$2, 108

Έλεγχος αν η τιμή του y είναι ίση με την τιμή της μεταβλητής T$2 που είχε αποθηκεύσει το αποτέλεσμα των πράξεων. Αν ναι, γίνεται άλμα στην εντολή με ετικέτα 108.

107: jump, _, _, 109

Αν δεν ισχύει το if, άλμα στην εντολή με ετικέτα 109.

108: ret, 1, _, _

Επιστροφή της τιμής 1 από τη συνάρτηση αν είχε μπει στο if.

109: ret, 0, _, _

Επιστροφή της τιμής 0 από τη συνάρτηση αν είχε μπει στο else.

110: end_block, divides, _, _

Τέλος της συνάρτησης "divides".

111: begin_block, isPrime, _, _

Έναρξη νέας συνάρτησης με όνομα με όνομα "isPrime".

112: +, counterFunctionCalls, 1, T$3

Εξηγήθηκε. Διαφορά η αποθήκευση στην T$3.

113: =, T$3, _, counterFunctionCalls

Εξηγήθηκε.

114: =, 2, _, i

Ανάθεση της τιμής 2 στη μεταβλητή i.

115: <, i, x, 117

Είσοδος στο while block και έλεγχος αν η τιμή του i είναι μικρότερη από την τιμή του x. Αν ναι, γίνεται άλμα στην εντολή 117.

116: jump, _, _, 127

Αν δεν ισχύει το while, άλμα στην εντολή 127.

117: par, i, _, CV, divides

Η τιμή του i είναι παράμετρος για την κλήση της συνάρτησης divides. Γίνεται πέρασμα με τιμή, γι αυτό υπάρχει το CV σαν τρίτο τελούμενο.

118: par, x, _, CV, divides

Η τιμή του x είναι παράμετρος για την κλήση της συνάρτησης divides. Γίνεται πέρασμα με τιμή, γι αυτό υπάρχει το CV σαν τρίτο τελούμενο.

119: par, T$4, _, RET, divides

Παράμετρος για την επιστροφή της τιμής από την κλήση της συνάρτησης divides που αποθηκεύεται στη μεταβλητή T$4.

120: call, _, _, divides

Κλήση της συνάρτησης divides.

121: ==, T$4, 1, 123

Είσοδος στο if block και έλεγχος αν η τιμή της μεταβλητής T$4 είναι ίση με 1. Αν ναι, γίνεται άλμα στην εντολή 123.

122: jump, _, _, 127

Αν δεν ισχύει το if, άλμα στην εντολή 127.

123: ret, 0, _, _

Επιστροφή της τιμής 0 από τη συνάρτηση.

124: +, i, 1, T$5

Πρόσθεση της τιμής του i με 1 και αποθήκευση του αποτελέσματος στη προσωρινή μεταβλητή T$5.

125: =, T$5, _, i

Αντιγραφή της τιμής από τη μεταβλητή T$5 στη μεταβλητή i ώστε να κάνει increment κατα 1.

126: jump, _, _, 115

Άλμα στην εντολή 115 για να ξαναγίνει έλεγχος, καθώς βρισκόμαστε σε while block.

127: ret, 1, _, _

Επιστροφή της τιμής 1 από τη συνάρτηση.

128: end_block, isPrime, _, _

Τέλος της συνάρτησης "isPrime".

129: begin_block, main, _, _

Έναρξη της συνάρτησης "main".

130: =, 0, _, counterFunctionCalls

Αναγνώριση της μεταβλητής counterFunctionCalls και ανάθεση της τιμής 0 σε αυτή.

131: in, _, _, i

Είσοδος τιμής χρήστη και αποθήκευσή της στη μεταβλητή i.

132: out, _, _, i

Εκτύπωση στην οθόνη της τιμής της μεταβλητής i.

133: =, 1600, _, i

Ανάθεση της τιμής 1600 στη μεταβλητή i.

134: <=, i, 2000, 136

Είσοδος στο while block και έλεγχος αν η τιμή του i είναι μικρότερη ή ίση με 2000. Αν ναι, γίνεται άλμα στην εντολή 136.

135: jump, _, _, 139

Αν δεν ισχύει η while, άλμα στην εντολή 139.

136: +, i, 400, T$6

Πρόσθεση της τιμής του i με 400 και αποθήκευση του αποτελέσματος στην προσωρινή μεταβλητή T$6.

137: =, T$6, _, i

Αντιγραφή της τιμής από τη μεταβλητή T$6 στη μεταβλητή i ώστε να κάνει increment κατα 400.

138: jump, _, _, 134

Άλμα στην εντολή 134 για να ξαναγίνει έλεγχος, καθώς βρισκόμαστε σε while block.

139: =, 1, _, i

Ανάθεση της τιμής 1 στη μεταβλητή i.

140: <=, i, 12, 142

Είσοδος στο while block και έλεγχος αν η τιμή του i είναι μικρότερη ή ίση με 12. Αν ναι, γίνεται άλμα στην εντολή 142.

141: jump, _, _, 149

Αν δεν ισχύει η while, άλμα στην εντολή 149.

142: par, i, _, CV, isPrime

Η τιμή του i είναι παράμετρος για την κλήση της συνάρτησης isPrime. Γίνεται πέρασμα με τιμή, γι αυτό υπάρχει το CV σαν τρίτο τελούμενο.

143: par, T$7, _, RET, isPrime

Παράμετρος για την επιστροφή της τιμής από την κλήση της συνάρτησης isPrime που αποθηκεύεται στη μεταβλητή T$7.

144: call, _, _, isPrime

Κλήση της συνάρτησης isPrime.

145: out, T$7, _, _

Έκτύπωση στην οθόνη της τιμής της μεταβλητής T$7.

146: +, i, 1, T$8

Πρόσθεση της τιμής του i με 1 και αποθήκευση του αποτελέσματος στη προσωρινή μεταβλητή T$8.

147: =, T$8, _, i

Αντιγραφή της τιμής από τον καταχωρητή T$8 στη μεταβλητή i για να κάνει increment κατα 1.

148: jump, _, _, 140

Άλμα στην εντολή 140 για να ξαναγίνει έλεγχος, καθώς βρισκόμαστε σε while block.

149: out, counterFunctionCalls, _, _

Εκτύπωση στην οθόνη της τιμής της μεταβλητής counterFunctionCalls.

150: halt, _, _, _

Τερματισμός της εκτέλεσης του προγράμματος.

151: end_block, main, _, _

Τέλος της συνάρτησης "main".

Πίνακας Συμβόλων

Ο **πίνακας συμβόλων** περιλαμβάνει πληροφορίες για κάθε επίπεδο του κώδικα, τις μεταβλητές που δηλώνονται σε αυτό και τις θέσεις (διευθύνσεις) στη μνήμη που χρησιμοποιούνται για την αποθήκευσή τους. Παρουσιάζεται ο πίνακας συμβόλων παρακάτω:

```
(0) isPrime | (1) x integer cv 12 | i integer 16 | divides | (2) y integer cv 12 | counterFunctionCalls integer 16 | T$0 integer 20 | T$1 integer 24 | T$2 integer 28 |
(0) isPrime | (1) x integer cv 12 | i integer 16 | divides | counterFunctionCalls integer 20 | T$3 integer 24 | T$4 integer 28 | T$5 integer 32 |
(0) isPrime | i integer 12 | counterFunctionCalls integer 16 | T$6 integer 20 | T$7 integer 24 | T$8 integer 28 |
```

Η ανάλυση του πίνακα συμβόλων φαίνεται παρακάτω:

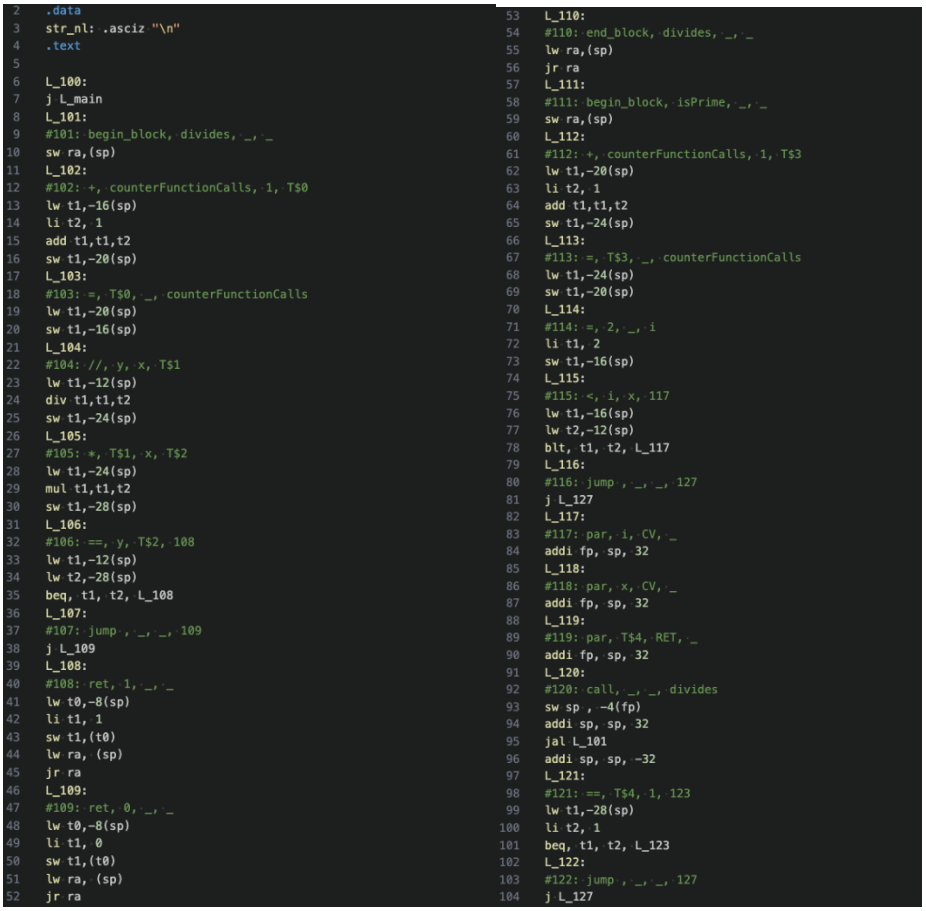
- Ως πρώτη μεταβλητή του πίνακα συμβόλων, ορίζεται η **isPrime** στο επίπεδο 0.
- Η συνάρτηση **isPrime** βρίσκεται στο δεύτερο επίπεδο, το επίπεδο 1.
- Η παράμετρος **x** δηλώνεται και καταλαμβάνει τη θέση 12.
- Η τοπική μεταβλητή **i** της **isPrime** καταλαμβάνει τη θέση 16.
- Η συνάρτηση **divides** δηλώνεται και δημιουργείται νέο επίπεδο, το επίπεδο 2, ως μέρος του πλαισίου (frame) της **isPrime**.
- Η μεταβλητή **y** καταλαμβάνει τη θέση 12 και η **counterFunctionCalls** καταλαμβάνει τη θέση 16.
- Οι προσωρινές μεταβλητές **T$0, T$1, T$2** χρησιμοποιούνται για αποθήκευση ενδιάμεσων αποτελεσμάτων και καταλαμβάνουν τις θέσεις 20, 24, 28 αντίστοιχα.
- Διαγράφεται το επίπεδο 2 και στη συνέχεια του προγράμματος, η συνάρτηση **isPrime** συνεχίζει να εκτελείται.
- Η **counterFunctionCalls** καταλαμβάνει τη θέση 20.
- Οι νέες προσωρινές μεταβλητές **T$3, T$4, T$5** καταλαμβάνουν τις θέσεις 24, 28, 32 αντίστοιχα.
- Διαγράφεται το επίπεδο 1 και συνεχίζει το πρόγραμμα να αποθηκεύει μεταβλητές στο επίπεδο της main (επίπεδο 0).
- Η μεταβλητή **i** καταλαμβάνει τη θέση 12.
- Η **counterFunctionCalls** καταλαμβάνει τη θέση 16.
- Οι νέες προσωρινές μεταβλητές **T$6, T$7, T$8** καταλαμβάνουν τις θέσεις 20, 24, 28 αντίστοιχα.

Τελικός Κώδικας

Σε αυτή την ενότητα, θα εξετάσουμε τον **τελικό κώδικα** assembly που παράγεται από την αρχική γραμματική CutePy. Το test είναι ίδιο με αυτό που χρησιμοποιήθηκε στον **ενδιάμεσο κώδικα** και τον **πίνακα συμβόλων.** Ο κώδικας assembly είναι το αποτέλεσμα της μεταγλώττισης του κώδικα του test σε εντολές χαμηλού επιπέδου, οι οποίες μπορούν να εκτελεστούν απευθείας από τον επεξεργαστή.

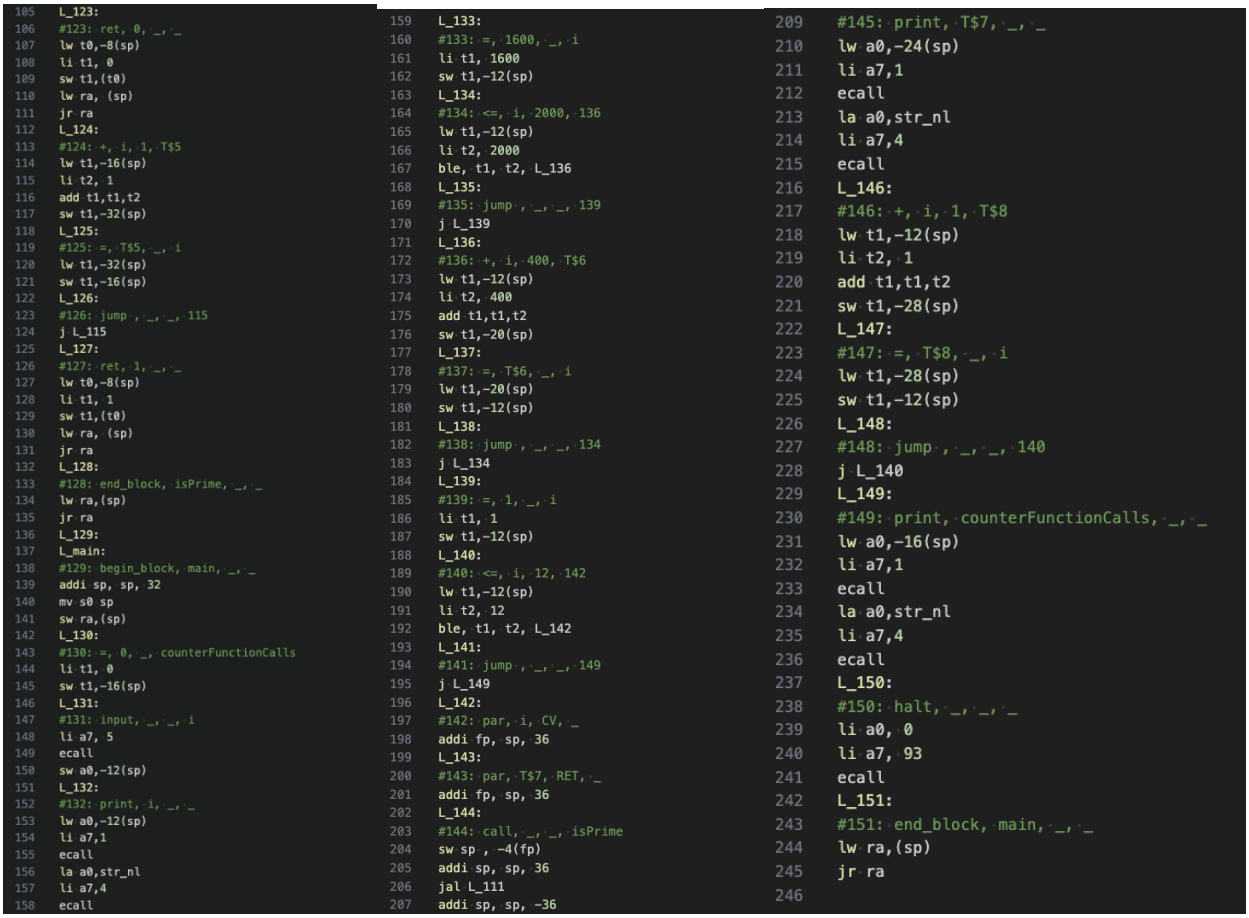
Παρακάτω παρουσιάζεται αναλυτικά ο **κώδικας assembly**, συνοδευόμενος από επεξηγήσεις για κάθε τμήμα του, ώστε να γίνει κατανοητός ο τρόπος με τον οποίο υλοποιούνται οι διάφορες λειτουργίες του προγράμματος.

Κώδικας assembly:



```
.data
str_nl: .asciz "\n"
.text

L_100:
j L_main
L_101:
#101: begin_block, divides, _, _
sw ra,(sp)
L_102:
#102: +, counterFunctionCalls, 1, T$0
lw t1,-16(sp)
li t2, 1
add t1,t1,t2
sw t1,-20(sp)
L_103:
#103: =, T$0, _, counterFunctionCalls
lw t1,-20(sp)
sw t1,-16(sp)
L_104:
#104: //, y, x, T$1
lw t1,-12(sp)
div t1,t1,t2
sw t1,-24(sp)
L_105:
#105: *, T$1, x, T$2
lw t1,-24(sp)
mul t1,t1,t2
sw t1,-28(sp)
L_106:
#106: ==, y, T$2, 108
lw t1,-12(sp)
lw t2,-28(sp)
beq, t1, t2, L_108
L_107:
#107: jump , _, _, 109
j L_109
L_108:
#108: ret, 1, _, _
lw t0,-8(sp)
li t1, 1
sw t1,(t0)
lw ra, (sp)
jr ra
L_109:
#109: ret, 0, _, _
lw t0,-8(sp)
li t1, 0
sw t1,(t0)
lw ra, (sp)
jr ra
L_110:
#110: end_block, divides, _, _
lw ra,(sp)
jr ra
L_111:
#111: begin_block, isPrime, _, _
sw ra,(sp)
L_112:
#112: +, counterFunctionCalls, 1, T$3
lw t1,-20(sp)
li t2, 1
add t1,t1,t2
sw t1,-24(sp)
L_113:
#113: =, T$3, _, counterFunctionCalls
lw t1,-24(sp)
sw t1,-20(sp)
L_114:
#114: =, 2, _, i
li t1, 2
sw t1,-16(sp)
L_115:
#115: <, i, x, 117
lw t1,-16(sp)
lw t2,-12(sp)
blt, t1, t2, L_117
L_116:
#116: jump , _, _, 127
j L_127
L_117:
#117: par, i, CV, _
addi fp, sp, 32
L_118:
#118: par, x, CV, _
addi fp, sp, 32
L_119:
#119: par, T$4, RET, _
addi fp, sp, 32
L_120:
#120: call, _, _, divides
sw sp , -4(fp)
addi sp, sp, 32
jal L_101
addi sp, sp, -32
L_121:
#121: ==, T$4, 1, 123
lw t1,-28(sp)
li t2, 1
beq, t1, t2, L_123
L_122:
#122: jump , _, _, 127
j L_127
```



```
L_123:
#123: ret, 0, _, _
lw t0,-8(sp)
li t1, 0
sw t1,(t0)
lw ra, (sp)
jr ra
L_124:
#124: +, i, 1, T$5
lw t1,-16(sp)
li t2, 1
add t1,t1,t2
sw t1,-32(sp)
L_125:
#125: =, T$5, _, i
lw t1,-32(sp)
sw t1,-16(sp)
L_126:
#126: jump , _, _, 115
j L_115
L_127:
#127: ret, 1, _, _
lw t0,-8(sp)
li t1, 1
sw t1,(t0)
lw ra, (sp)
jr ra
L_128:
#128: end_block, isPrime, _, _
lw ra,(sp)
jr ra
L_129:
L_main:
#129: begin_block, main, _, _
addi sp, sp, 32
mv s0 sp
sw ra,(sp)
L_130:
#130: =, 0, _, counterFunctionCalls
li t1, 0
sw t1,-16(sp)
L_131:
#131: input, _, _, i
li a7, 5
ecall
sw a0,-12(sp)
L_132:
#132: print, i, _, _
lw a0,-12(sp)
li a7,1
ecall
la a0,str_nl
li a7,4
ecall
L_133:
#133: =, 1600, _, i
li t1, 1600
sw t1,-12(sp)
L_134:
#134: <=, i, 2000, 136
lw t1,-12(sp)
li t2, 2000
ble, t1, t2, L_136
L_135:
#135: jump , _, _, 139
j L_139
L_136:
#136: +, i, 400, T$6
lw t1,-12(sp)
li t2, 400
add t1,t1,t2
sw t1,-20(sp)
L_137:
#137: =, T$6, _, i
lw t1,-20(sp)
sw t1,-12(sp)
L_138:
#138: jump , _, _, 134
j L_134
L_139:
#139: =, 1, _, i
li t1, 1
sw t1,-12(sp)
L_140:
#140: <=, i, 12, 142
lw t1,-12(sp)
li t2, 12
ble, t1, t2, L_142
L_141:
#141: jump , _, _, 149
j L_149
L_142:
#142: par, i, CV, _
addi fp, sp, 36
L_143:
#143: par, T$7, RET, _
addi fp, sp, 36
L_144:
#144: call, _, _, isPrime
sw sp , -4(fp)
addi sp, sp, 36
jal L_111
addi sp, sp, -36
#145: print, T$7, _, _
lw a0,-24(sp)
li a7,1
ecall
la a0,str_nl
li a7,4
ecall
L_146:
#146: +, i, 1, T$8
lw t1,-12(sp)
li t2, 1
add t1,t1,t2
sw t1,-28(sp)
L_147:
#147: =, T$8, _, i
lw t1,-28(sp)
sw t1,-12(sp)
L_148:
#148: jump , _, _, 140
j L_140
L_149:
#149: print, counterFunctionCalls, _, _
lw a0,-16(sp)
li a7,1
ecall
la a0,str_nl
li a7,4
ecall
L_150:
#150: halt, _, _, _
li a0, 0
li a7, 93
ecall
L_151:
#151: end_block, main, _, _
lw ra,(sp)
jr ra
```

Θα εξηγηθούν αναλυτικά οι ετικέτες μία προς μία για κατανόηση και έλεγχο του **τελικού κώδικα.** Στον κώδικα υπάρχουν σχόλια για κάθε ετικέτα που υποδεικνύουν την αντίστοιχη εντολή του **ενδιάμεσου κώδικα**, δείχνοντας τον τρόπου που μετατράπηκε σε assembly.

Αρχικά η

.data

str_nl: .asciz "\n"

Δηλώνει τη σταθερά str_nl που περιέχει τον χαρακτήρα νέας γραμμής (\n).

Ετικέτα L_100: Εκτέλεση άλματος στην ετικέτα L_main, η οποία σηματοδοτεί την αρχή του κύριου προγράμματος.

Ενότητα divides (μπλοκ 101-110)

L_101: Στην αρχή της κλήσης της συνάρτησης divides, αποθηκεύουμε την τιμή του καταχωρητή ra στη στοίβα για να διατηρήσουμε τη διεύθυνση επιστροφής της συνάρτησης.

L_102:

- Φορτώνουμε την τιμή της μεταβλητής counterFunctionCalls από τη θέση -16(sp) στον καταχωρητή t1 με την εντολή lw t1, -16(sp).
- Φορτώνουμε την τιμή 1 στον καταχωρητή t2 με την εντολή li t2, 1.
- Προσθέτουμε τις τιμές των t1 και t2 με την εντολή add t1, t1, t2 και αποθηκεύουμε το αποτέλεσμα στον καταχωρητή t1.
- Αποθηκεύουμε το αποτέλεσμα στον καταχωρητή t1 στη θέση -20(sp) με την εντολή sw t1, -20(sp).

L_103:

- Φορτώνουμε την τιμή της προσωρινής μεταβλητής T$0 από τη θέση -20(sp) στον καταχωρητή t1 με την εντολή lw t1, -20(sp).
- Αποθηκεύουμε την τιμή στον καταχωρητή t1 στη μεταβλητή counterFunctionCalls στη θέση -16(sp) με την εντολή sw t1, -16(sp).

L_104:

- Φορτώνουμε την τιμή της παραμέτρου y από τη θέση -12(sp) στον καταχωρητή t1 με την εντολή lw t1, -12(sp).
- Εκτελούμε ακέραια διαίρεση του t1 (y) με τον t2 (x) και αποθηκεύουμε το αποτέλεσμα στον t1 με την εντολή div t1, t1, t2.
- Αποθηκεύουμε το αποτέλεσμα στον t1 στη θέση -24(sp) με την εντολή sw t1, -24(sp).

L_105:

- Φορτώνουμε την τιμή της προσωρινής μεταβλητής T$1 από τη θέση -24(sp) στον καταχωρητή t1 με την εντολή lw t1, -24(sp).
- Εκτελούμε πολλαπλασιασμό του t1 με τον t2 (x) και αποθηκεύουμε το αποτέλεσμα στον t1 με την εντολή mul t1, t1, t2.
- Αποθηκεύουμε το αποτέλεσμα στον t1 στη θέση -28(sp) με την εντολή sw t1, -28(sp).

L_106:

- Φορτώνουμε την τιμή της παραμέτρου y από τη θέση -12(sp) στον καταχωρητή t1 με την εντολή lw t1, -12(sp).
- Φορτώνουμε την τιμή της προσωρινής μεταβλητής T$2 από τη θέση -28(sp) στον καταχωρητή t2 με την εντολή lw t2, -28(sp).
- Εκτελούμε έλεγχο αν οι τιμές των t1 και t2 είναι ίσες με την εντολή beq t1, t2, L_108. Αν είναι ίσες, πηγαίνουμε στην ετικέτα L_108.

L_107: Εκτελούμε άλμα στην ετικέτα L_109 με την εντολή j L_109.

L_108:

- Φορτώνουμε τη διεύθυνση επιστροφής από τη θέση -8(sp) στον καταχωρητή t0 με την εντολή lw t0, -8(sp).
- Φορτώνουμε την τιμή 1 στον καταχωρητή t1 με την εντολή li t1, 1.
- Αποθηκεύουμε την τιμή 1 στη διεύθυνση επιστροφής με την εντολή sw t1, (t0).
- Φορτώνουμε την τιμή του καταχωρητή ra από τη στοίβα με την εντολή lw ra, (sp).
- Εκτελούμε άλμα επιστροφής με την εντολή jr ra.

L_109:

- Φορτώνουμε τη διεύθυνση επιστροφής από τη θέση -8(sp) στον καταχωρητή t0 με την εντολή lw t0, -8(sp).
- Φορτώνουμε την τιμή 0 στον καταχωρητή t1 με την εντολή li t1, 0.
- Αποθηκεύουμε την τιμή 0 στη διεύθυνση επιστροφής με την εντολή sw t1, (t0).
- Φορτώνουμε την τιμή του καταχωρητή ra από τη στοίβα με την εντολή lw ra, (sp).
- Εκτελούμε άλμα επιστροφής με την εντολή jr ra.

L_110: Τέλος του μπλοκ divides, επιστρέφουμε τον έλεγχο στην καλούσα συνάρτηση φορτώνοντας τον καταχωρητή ra από τη στοίβα με την εντολή lw ra, (sp) και εκτελώντας άλμα επιστροφής με την εντολή jr ra.

Ενότητα isPrime (μπλοκ 111-128)

L_111: Στην αρχή της κλήσης της συνάρτησης isPrime, αποθηκεύουμε την τιμή του καταχωρητή ra στη στοίβα για να διατηρήσουμε τη διεύθυνση επιστροφής της συνάρτησης..

L_112:

- Φορτώνουμε την τιμή της μεταβλητής counterFunctionCalls από τη θέση -20(sp) στον καταχωρητή t1 με την εντολή lw t1, -20(sp).
- Φορτώνουμε την τιμή 1 στον καταχωρητή t2 με την εντολή li t2, 1.
- Προσθέτουμε τις τιμές των t1 και t2 με την εντολή add t1, t1, t2 και αποθηκεύουμε το αποτέλεσμα στον καταχωρητή t1.
- Αποθηκεύουμε το αποτέλεσμα στον καταχωρητή t1 στη θέση -24(sp) με την εντολή sw t1, -24(sp).

L_113:

- Φορτώνουμε την τιμή της προσωρινής μεταβλητής T$3 από τη θέση -24(sp) στον καταχωρητή t1 με την εντολή lw t1, -24(sp).
- Αποθηκεύουμε την τιμή στον καταχωρητή t1 στη μεταβλητή counterFunctionCalls στη θέση -20(sp) με την εντολή sw t1, -20(sp).

L_114:

- Φορτώνουμε την τιμή 2 στον καταχωρητή t1 με την εντολή li t1, 2.
- Αποθηκεύουμε την τιμή 2 στη μεταβλητή i στη θέση -16(sp) με την εντολή sw t1, -16(sp).

L_115:

- Φορτώνουμε την τιμή της μεταβλητής i από τη θέση -16(sp) στον καταχωρητή t1 με την εντολή lw t1, -16(sp).
- Φορτώνουμε την τιμή της παραμέτρου x από τη θέση -12(sp) στον καταχωρητή t2 με την εντολή lw t2, -12(sp).
- Εκτελούμε έλεγχο αν η τιμή του t1 είναι μικρότερη από την τιμή του t2 με την εντολή blt t1, t2, L_117. Αν είναι μικρότερη, πηγαίνουμε στην ετικέτα L_117.

L_116: Εκτελούμε άλμα στην ετικέτα L_127 με την εντολή j L_127.

L_117 έως L_119: Προετοιμασία παραμέτρων.

L_120:

- Αποθηκεύουμε τον δείκτη στοίβας sp στη θέση -4(fp) με την εντολή sw sp, -4(fp).
- Αυξάνουμε τον δείκτη στοίβας sp κατά 32 θέσεις με την εντολή addi sp, sp, 32.
- Εκτελούμε άλμα και σύνδεση στην ετικέτα L_101 με την εντολή jal L_101.
- Μειώνουμε τον δείκτη στοίβας sp κατά 32 θέσεις με την εντολή addi sp, sp, -32.

L_121:

- Φορτώνουμε την τιμή της προσωρινής μεταβλητής T$4 από τη θέση -28(sp) στον καταχωρητή t1 με την εντολή lw t1, -28(sp).
- Φορτώνουμε την τιμή 1 στον καταχωρητή t2 με την εντολή li t2, 1.
- Εκτελούμε έλεγχο αν οι τιμές των t1 και t2 είναι ίσες με την εντολή beq t1, t2, L_123. Αν είναι ίσες, πηγαίνουμε στην ετικέτα L_123.

L_122: Εκτελούμε άλμα στην ετικέτα L_127 με την εντολή j L_127.

L_123:

- Φορτώνουμε τη διεύθυνση επιστροφής από τη θέση -8(sp) στον καταχωρητή t0 με την εντολή lw t0, -8(sp).
- Φορτώνουμε την τιμή 0 στον καταχωρητή t1 με την εντολή li t1, 0.
- Αποθηκεύουμε την τιμή 0 στη διεύθυνση επιστροφής με την εντολή sw t1, (t0).
- Φορτώνουμε την τιμή του καταχωρητή ra από τη στοίβα με την εντολή lw ra, (sp).
- Εκτελούμε άλμα επιστροφής με την εντολή jr ra.

L_124:

- Φορτώνουμε την τιμή της μεταβλητής i από τη θέση -16(sp) στον καταχωρητή t1 με την εντολή lw t1, -16(sp).
- Φορτώνουμε την τιμή 1 στον καταχωρητή t2 με την εντολή li t2, 1.
- Προσθέτουμε τις τιμές των t1 και t2 με την εντολή add t1, t1, t2 και αποθηκεύουμε το αποτέλεσμα στον καταχωρητή t1.
- Αποθηκεύουμε το αποτέλεσμα στον καταχωρητή t1 στη θέση -32(sp) με την εντολή sw t1, -32(sp).

L_125:

- Φορτώνουμε την τιμή της προσωρινής μεταβλητής T$5 από τη θέση -32(sp) στον καταχωρητή t1 με την εντολή lw t1, -32(sp).
- Αποθηκεύουμε την τιμή στον καταχωρητή t1 στη μεταβλητή i στη θέση -16(sp) με την εντολή sw t1, -16(sp).

L_126: Εκτελούμε άλμα πίσω στην ετικέτα L_115 με την εντολή j L_115 για να επανελέγξουμε την συνθήκη του while loop.

L_127:

- Φορτώνουμε τη διεύθυνση επιστροφής από τη θέση -8(sp) στον καταχωρητή t0 με την εντολή lw t0, -8(sp).
- Φορτώνουμε την τιμή 1 στον καταχωρητή t1 με την εντολή li t1, 1.
- Αποθηκεύουμε την τιμή 1 στη διεύθυνση επιστροφής με την εντολή sw t1, (t0).
- Φορτώνουμε την τιμή του καταχωρητή ra από τη στοίβα με την εντολή lw ra, (sp).
- Εκτελούμε άλμα επιστροφής με την εντολή jr ra.

L_128: Τέλος του μπλοκ isPrime, επιστρέφουμε τον έλεγχο στην καλούσα συνάρτηση φορτώνοντας τον καταχωρητή ra από τη στοίβα με την εντολή lw ra, (sp) και εκτελώντας άλμα επιστροφής με την εντολή jr ra.

Ενότητα main (μπλοκ 129-151)

L_129:

- Αρχικοποιεί το stack frame της κύριας συνάρτησης main, αυξάνοντας τον δείκτη στοίβας sp κατά 32 θέσεις με την εντολή addi sp, sp, 32.
- Μεταφέρει την τιμή του sp στον καταχωρητή s0 με την εντολή mv s0, sp.
- Αποθηκεύει την τιμή του καταχωρητή ra στη στοίβα με την εντολή sw ra, (sp).

L_130:

- Φορτώνει την τιμή 0 στον καταχωρητή t1 με την εντολή li t1, 0.
- Αποθηκεύει την τιμή 0 στη μεταβλητή counterFunctionCalls στη θέση -16(sp) με την εντολή sw t1, -16(sp).

L_131:

- Θέτει την τιμή 5 στον καταχωρητή a7 με την εντολή li a7, 5 για να ζητήσει είσοδο από τον χρήστη.

- Εκτελεί την εντολή ecall για να διαβάσει την είσοδο και αποθηκεύει την τιμή στη μεταβλητή i στη θέση -12(sp) με την εντολή sw a0, -12(sp).

L_132:

- Φορτώνει την τιμή της μεταβλητής i από τη θέση -12(sp) στον καταχωρητή a0 με την εντολή lw a0, -12(sp).
- Θέτει την τιμή 1 στον καταχωρητή a7 με την εντολή li a7, 1 για να ζητήσει εκτύπωση της τιμής του a0.
- Εκτελεί την εντολή ecall για να εκτυπώσει την τιμή του a0.
- Φορτώνει τη διεύθυνση της σταθεράς str_nl στον καταχωρητή a0 με την εντολή la a0, str_nl.
- Θέτει την τιμή 4 στον καταχωρητή a7 με την εντολή li a7, 4 για να ζητήσει εκτύπωση της νέας γραμμής.
- Εκτελεί την εντολή ecall για να εκτυπώσει τη νέα γραμμή.

L_133:

- Φορτώνει την τιμή 1600 στον καταχωρητή t1 με την εντολή li t1, 1600.
- Αποθηκεύει την τιμή 1600 στη μεταβλητή i στη θέση -12(sp) με την εντολή sw t1, -12(sp).

L_134:

- Φορτώνει την τιμή της μεταβλητής i από τη θέση -12(sp) στον καταχωρητή t1 με την εντολή lw t1, -12(sp).
- Φορτώνει την τιμή 2000 στον καταχωρητή t2 με την εντολή li t2, 2000.
- Εκτελεί έλεγχο αν η τιμή του t1 είναι μικρότερη ή ίση με την τιμή του t2 με την εντολή ble t1, t2, L_136. Αν είναι μικρότερη ή ίση, πηγαίνουμε στην ετικέτα L_136.

L_135: Εκτελεί άλμα στην ετικέτα L_139 με την εντολή j L_139.

L_136:

- Φορτώνει την τιμή της μεταβλητής i από τη θέση -12(sp) στον καταχωρητή t1 με την εντολή lw t1, -12(sp).
- Φορτώνει την τιμή 400 στον καταχωρητή t2 με την εντολή li t2, 400.
- Προσθέτει τις τιμές των t1 και t2 με την εντολή add t1, t1, t2 και αποθηκεύει το αποτέλεσμα στον καταχωρητή t1.
- Αποθηκεύει το αποτέλεσμα στον καταχωρητή t1 στη θέση -20(sp) με την εντολή sw t1, -20(sp).

L_137:

- Φορτώνει την τιμή της προσωρινής μεταβλητής T$6 από τη θέση -20(sp) στον καταχωρητή t1 με την εντολή lw t1, -20(sp).
- Αποθηκεύει την τιμή στον καταχωρητή t1 στη μεταβλητή i στη θέση -12(sp) με την εντολή sw t1, -12(sp).

L_138: Εκτελεί άλμα πίσω στην ετικέτα L_134 με την εντολή j L_134 για να επανελέγξει την συνθήκη του while loop.

L_139:

- Φορτώνει την τιμή 1 στον καταχωρητή t1 με την εντολή li t1, 1.
- Αποθηκεύει την τιμή 1 στη μεταβλητή i στη θέση -12(sp) με την εντολή sw t1, -12(sp).

L_140:

- Φορτώνει την τιμή της μεταβλητής i από τη θέση -12(sp) στον καταχωρητή t1 με την εντολή lw t1, -12(sp).
- Φορτώνει την τιμή 12 στον καταχωρητή t2 με την εντολή li t2, 12.
- Εκτελεί έλεγχο αν η τιμή του t1 είναι μικρότερη ή ίση με την τιμή του t2 με την εντολή ble t1, t2, L_142. Αν είναι μικρότερη ή ίση, πηγαίνουμε στην ετικέτα L_142.

L_141: Εκτελεί άλμα στην ετικέτα L_149 με την εντολή j L_149.

L_142: Αυξάνει τον δείκτη πλαισίου fp κατά 36 θέσεις με την εντολή addi fp, sp, 36.

L_143: Αυξάνει τον δείκτη πλαισίου fp κατά 36 θέσεις με την εντολή addi fp, sp, 36.

L_144:

- Αποθηκεύει τον δείκτη στοίβας sp στη θέση -4(fp) με την εντολή sw sp, -4(fp).
- Αυξάνει τον δείκτη στοίβας sp κατά 36 θέσεις με την εντολή addi sp, sp, 36.
- Εκτελεί άλμα και σύνδεση στην ετικέτα L_111 με την εντολή jal L_111.
- Μειώνει τον δείκτη στοίβας sp κατά 36 θέσεις με την εντολή addi sp, sp, -36.

L_145:

- Φορτώνει την τιμή της προσωρινής μεταβλητής T$7 από τη θέση -24(sp) στον καταχωρητή a0 με την εντολή lw a0, -24(sp).
- Θέτει την τιμή 1 στον καταχωρητή a7 με την εντολή li a7, 1 για να ζητήσει εκτύπωση της τιμής του a0.
- Εκτελεί την εντολή ecall για να εκτυπώσει την τιμή του a0.
- Φορτώνει τη διεύθυνση της σταθεράς str_nl στον καταχωρητή a0 με την εντολή la a0, str_nl.
- Θέτει την τιμή 4 στον καταχωρητή a7 με την εντολή li a7, 4 για να ζητήσει εκτύπωση της νέας γραμμής.
- Εκτελεί την εντολή ecall για να εκτυπώσει τη νέα γραμμή.

L_146:

- Φορτώνει την τιμή της μεταβλητής i από τη θέση -12(sp) στον καταχωρητή t1 με την εντολή lw t1, -12(sp).
- Φορτώνει την τιμή 1 στον καταχωρητή t2 με την εντολή li t2, 1.
- Προσθέτει τις τιμές των t1 και t2 με την εντολή add t1, t1, t2 και αποθηκεύει το αποτέλεσμα στον καταχωρητή t1.
- Αποθηκεύει το αποτέλεσμα στον καταχωρητή t1 στη θέση -28(sp) με την εντολή sw t1, -28(sp).

L_147:

- Φορτώνει την τιμή της προσωρινής μεταβλητής T$8 από τη θέση -28(sp) στον καταχωρητή t1 με την εντολή lw t1, -28(sp).
- Αποθηκεύει την τιμή στον καταχωρητή t1 στη μεταβλητή i στη θέση -12(sp) με την εντολή sw t1, -12(sp).

L_148: Εκτελεί άλμα πίσω στην ετικέτα L_140 με την εντολή j L_140 για να επανελέγξει την συνθήκη του while loop.

L_149:

- Φορτώνει την τιμή της μεταβλητής counterFunctionCalls από τη θέση -16(sp) στον καταχωρητή a0 με την εντολή lw a0, -16(sp).
- Θέτει την τιμή 1 στον καταχωρητή a7 με την εντολή li a7, 1 για να ζητήσει εκτύπωση της τιμής του a0.
- Εκτελεί την εντολή ecall για να εκτυπώσει την τιμή του a0.
- Φορτώνει τη διεύθυνση της σταθεράς str_nl στον καταχωρητή a0 με την εντολή la a0, str_nl.
- Θέτει την τιμή 4 στον καταχωρητή a7 με την εντολή li a7, 4 για να ζητήσει εκτύπωση της νέας γραμμής.
- Εκτελεί την εντολή ecall για να εκτυπώσει τη νέα γραμμή.

L_150:

- Θέτει την τιμή 0 στον καταχωρητή a0 με την εντολή li a0, 0 για να ζητήσει τερματισμό του προγράμματος.
- Θέτει την τιμή 93 στον καταχωρητή a7 με την εντολή li a7, 93.
- Εκτελεί την εντολή ecall για να τερματίσει το πρόγραμμα.

L_151: Τέλος του μπλοκ main, επιστρέφουμε τον έλεγχο στον καλούντα φορτώνοντας τον καταχωρητή ra από τη στοίβα με την εντολή lw ra, (sp) και εκτελώντας άλμα επιστροφής με την εντολή jr ra.

Μετά από compilation που έγινε στο περιβάλλον RISC-V, παρατηρούμε ότι το πρόγραμμα έτρεξε κανονικά:

```
Assemble: assembling /Users/agapitos/Documents/Universita/8o Semestre/metafrastes/2024/phase2/ascode.asm

Assemble: operation completed successfully.
```

Δοκιμαστικοί Έλεγχοι

Σε αυτή την ενότητα θα παραθέσουμε τον κώδικα ενός μεγάλου test, το οποίο θα συμπεριληφθεί και στα παραδοτέα. Ο **ενδιάμεσος κώδικας** θα παρατεθεί ολόκληρος, όπως και ο **πίνακας συμβόλων**. Ο **τελικός κώδικας** δεν θα παρατεθεί ολόκληρος με στιγμιότυπα οθόνης, καθώς είναι πολύ μεγάλος και δεν θα είναι ευδιάκριτος.

Κώδικας test3.cpy

test.cpy > ...

```
#int counterFunctionCalls

def max3(x,y,z):
#{
    #int m
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    if x>y and x>z:
        m = x
    elif y>x and y>z:
        m = y
    else:
        m = z
    return m
#}


def isPrime(x):
#{
    ## declarations for isPrime ##
    #int i

    def divides(x,y):
    #{
        ## body of divides ##
        global counterFunctionCalls
        counterFunctionCalls = counterFunctionCalls + 1
        if y == (y//x)*x:
            return 1
        else:
            return 0
    #}

    ## body of isPrime ##
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    i = 2
    while i < x :
    #{
        if divides(i,x) == 1:
            return 0
        i = i + 1
    #}
    return 1
#}

def quad(x):
#{
    #int y

    ## nested function sqr ##
    def sqr(x):
    #{
        ## body of sqr ##
        global counterFunctionCalls
        counterFunctionCalls = counterFunctionCalls + 1
        return x*x
    #}

    ## body of quad ##
    global counterFunctionCalls
    x=3
    counterFunctionCalls = counterFunctionCalls + 1
    y = sqr(x)*sqr(x)
    return y
#}

#def main
#int i
counterFunctionCalls = 0

i = int(input())
print(i)


i = 1600
while i<=2000:
#{
    i = i + 400
#}
print(quad(3))

i=1
while i<=12:
#{
    print(isPrime(i))
    i = i + 1
#}

print(counterFunctionCalls)
```

Ενδιάμεσος Κώδικας test3.cpy

inter_code.int

```
100: jump, _, _, main
101: begin_block, max3, _, _
102: +, counterFunctionCalls, 1, T$0
103: =, T$0, _, counterFunctionCalls
104: >, x, y, 106
105: jump, _, _, 110
106: >, x, z, 108
107: jump, _, _, 110
108: =, x, _, m
109: jump, _, _, 117
110: >, y, x, 112
111: jump, _, _, 116
112: >, y, z, 114
113: jump, _, _, 116
114: =, y, _, m
115: jump, _, _, 117
116: =, z, _, m
117: ret, m, _, _
118: end_block, max3, _, _
119: begin_block, divides, _, _
120: +, counterFunctionCalls, 1, T$1
121: =, T$1, _, counterFunctionCalls
122: //, y, x, T$2
123: *, T$2, x, T$3
124: ==, y, T$3, 126
125: jump, _, _, 127
126: ret, 1, _, _
127: ret, 0, _, _
128: end_block, divides, _, _
129: begin_block, isPrime, _, _
130: +, counterFunctionCalls, 1, T$4
131: =, T$4, _, counterFunctionCalls
132: =, 2, _, i
133: <, i, x, 135
134: jump, _, _, 145
135: par, i,  CV, divides
136: par, x,  CV, divides
137: par, T$5,  RET, divides
138: call, _, _, divides
139: ==, T$5, 1, 141
140: jump, _, _, 145
141: ret, 0, _, _
142: +, i, 1, T$6
143: =, T$6, _, i
144: jump, _, _, 133
145: ret, 1, _, _
146: end_block, isPrime, _, _
147: begin_block, sqr, _, _
148: +, counterFunctionCalls, 1, T$7
149: =, T$7, _, counterFunctionCalls
150: *, x, x, T$8
151: =, T$8, _, T$9
152: ret, T$9, _, _
153: end_block, sqr, _, _
154: begin_block, quad, _, _
155: =, 3, _, x
156: +, counterFunctionCalls, 1, T$10
157: =, T$10, _, counterFunctionCalls
158: par, x,  CV, sqr
159: par, T$11,  RET, sqr
160: call, _, _, sqr
161: par, x,  CV, sqr
162: par, T$12,  RET, sqr
163: call, _, _, sqr
164: *, T$11, T$12, T$13
165: =, T$13, _, y
166: ret, y, _, _
167: end_block, quad, _, _
168: begin_block, main, _, _
169: =, 0, _, counterFunctionCalls
170: in, _, _, i
171: out, i, _, _
172: =, 1600, _, i
173: <=, i, 2000, 175
174: jump, _, _, 178
175: +, i, 400, T$14
176: =, T$14, _, i
177: jump, _, _, 173
178: par, 173,  CV, quad
179: par, T$15,  RET, quad
180: call, _, _, quad
181: out, T$15, _, _
182: =, 1, _, i
183: <=, i, 12, 185
184: jump, _, _, 192
185: par, i,  CV, isPrime
186: par, T$16,  RET, isPrime
187: call, _, _, isPrime
188: out, T$16, _, _
189: +, i, 1, T$17
190: =, T$17, _, i
191: jump, _, _, 183
192: out, counterFunctionCalls, _, _
193: halt, _, _, _
194: end_block, main, _, _
```

Πίνακας Συμβόλων test3.cpy

symbol_table.sym

```
(0) counterFunctionCalls integer 12 | max3 |  (1) x integer cv 12 | y integer cv 16 | z integer cv 20 | m integer 24 | T$0 integer 28 |
(0) counterFunctionCalls integer 12 | max3 | isPrime |  (1) x integer cv 12 | i integer 16 | divides |  (2) y integer cv 12 | T$1 integer 16 | T$2 integer 20 | T$3 integer 24 |
(0) counterFunctionCalls integer 12 | max3 | isPrime |  (1) x integer cv 12 | i integer 16 | divides | T$4 integer 20 | T$5 integer 24 | T$6 integer 28 |
(0) counterFunctionCalls integer 12 | max3 | isPrime | quad |  (1) x integer cv 12 | y integer 16 | sqr |  (2) T$7 integer 12 | T$8 integer 16 | T$9 integer 20 |
(0) counterFunctionCalls integer 12 | max3 | isPrime | quad |  (1) x integer cv 12 | y integer 16 | sqr | T$10 integer 20 | T$11 integer 24 | T$12 integer 28 | T$13 integer 32 |
(0) counterFunctionCalls integer 12 | max3 | isPrime | quad | i integer 16 | T$14 integer 20 | T$15 integer 24 | T$16 integer 28 | T$17 integer 32 |
```

Ενδεικτικός Τελικός Κώδικας test3.cpy

ascode.asm

```
.data
str_nl: .asciz "\n"
.text

L_100:
j L_main
L_101:
#101: begin_block, max3, _, _
sw ra,(sp)
L_102:
#102: +, counterFunctionCalls, 1, T$0
lw t1,-12(gp)
li t2, 1
add t1,t1,t2
sw t1,-28(sp)
L_103:
#103: =, T$0, _, counterFunctionCalls
lw t1,-28(sp)
sw t1,-12(gp)
L_104:
#104: >, x, y, 106
lw t1,-12(sp)
lw t2,-16(sp)
bgt, t1, t2, L_106
L_105:
#105: jump , _, _, 110
j L_110
L_106:
#106: >, x, z, 108
lw t1,-12(sp)
lw t2,-20(sp)
bgt, t1, t2, L_108
L_107:
#107: jump , _, _, 110
j L_110
L_108:
#108: =, x, _, m
lw t1,-12(sp)
sw t1,-24(sp)
L_109:
#109: jump , _, _, 117
j L_117
L_110:
#110: >, y, x, 112
lw t1,-16(sp)
lw t2,-12(sp)
bgt, t1, t2, L_112
L_111:
#111: jump , _, _, 116
j L_116
L_112:
#112: >, y, z, 114
lw t1,-16(sp)
lw t2,-20(sp)
bgt, t1, t2, L_114
L_113:
#113: jump , _, _, 116
j L_116
L_114:
#114: =, y, _, m
lw t1,-16(sp)
sw t1,-24(sp)
L_115:
#115: jump , _, _, 117
j L_117
L_116:
#116: =, z, _, m
lw t1,-20(sp)
sw t1,-24(sp)
L_117:
#117: ret, m, _, _
lw t0,-8(sp)
lw t1,-24(sp)
sw t1,(t0)
lw ra, (sp)
jr ra
L_118:
#118: end_block, max3, _, _
lw ra,(sp)
jr ra
L_119:
#119: begin_block, divides, _, _
sw ra,(sp)
L_120:
#120: +, counterFunctionCalls, 1, T$1
lw t1,-12(gp)
li t2, 1
add t1,t1,t2
sw t1,-16(sp)
L_121:
#121: =, T$1, _, counterFunctionCalls
lw t1,-16(sp)
sw t1,-12(gp)
L_122:
#122: //, y, x, T$2
lw t1,-12(sp)
div t1,t1,t2
sw t1,-20(sp)
L_123:
#123: *, T$2, x, T$3
lw t1,-20(sp)
mul t1,t1,t2
sw t1,-24(sp)
L_124:
#124: ==, y, T$3, 126
lw t1,-12(sp)
lw t2,-24(sp)
#184: jump , _, _, 192
j L_192
L_185:
#185: par, i, CV, _
addi fp, sp, 32
L_186:
#186: par, T$16, RET, _
addi fp, sp, 32
L_187:
#187: call, _, _, isPrime
sw sp , -4(fp)
addi sp, sp, 32
jal L_129
addi sp, sp, -32
L_188:
#188: print, T$16, _, _
lw a0,-28(sp)
li a7,1
ecall
la a0,str_nl
li a7,4
ecall
L_189:
#189: +, i, 1, T$17
lw t1,-16(sp)
li t2, 1
add t1,t1,t2
sw t1,-32(sp)
L_190:
#190: =, T$17, _, i
lw t1,-32(sp)
sw t1,-16(sp)
L_191:
#191: jump , _, _, 183
j L_183
L_192:
#192: print, counterFunctionCalls, _, _
lw a0,-12(sp)
li a7,1
ecall
la a0,str_nl
li a7,4
ecall
L_193:
#193: halt, _, _, _
li a0, 0
li a7, 93
ecall
L_194:
#194: end_block, main, _, _
lw ra,(sp)
jr ra
```

Στη συνέχεια προχωράμε στο περιβάλλον RISC-V, τρέχουμε τον **τελικό κώδικα** που παράχθηκε και παρατηρούμε ότι γίνεται compile.

```
Assemble: assembling G:\10th_semester\Compilers\phase2\riscv1.asm

Assemble: operation completed successfully.
```

Παρ' ότι ο ενδιάμεσος και ο τελικός κώδικας φαίνεται να παράγονται σωστά, όπως εξηγήθηκε παραπάνω, εκτελούμε run στον RISC-V και παρατηρούμε το εξής σφάλμα:

```
Error in G:\10th_semester\Compilers\phase2\riscv1.asm line 234: Runtime exception at 0x004001f0: address out of range 0x00000000

Go: execution terminated with errors.
```