



ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αδηνών

ΙΔΡΥΘΕΝ ΤΟ 1837

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Αρχιτεκτονικής Υπολογιστών I

Η αρχιτεκτονική RISC-V και οι
προσομοιωτές RARS και QtRVsim

Εαρινό Εξάμηνο 2024-2025

© Εργαστήριο Αρχιτεκτονικής Υπολογιστών ΕΚΠΑ,
cal.di.uoa.gr,
Έκδοση 1.3 – 7 Απριλίου 2025

Περιεχόμενα

1 Ο προσομοιωτής RARS	7
E1 Εισαγωγή στον προσομοιωτή RARS	8
E1.1 Φόρτωση προγραμμάτων στον RARS	8
E1.2 Εκτέλεση προγραμμάτων στον RARS	11
E1.3 Διαδικασία αποσφαλμάτωσης στον RARS	11
E2 Διαχείριση μνήμης και κλήσεις συστήματος	14
E2.1 Φόρτωση και αποθήκευση αριθμών	14
E2.2 Διευθύνσεις μνήμης και δεδομένα	15
E2.3 Ακολουθία αριθμών στη μνήμη	16
E2.4 Διάβασμα και εκτύπωση ενός ακέραιου αριθμού	16
E3 Αριθμητικές και Λογικές Πράξεις	17
E3.1 Ο κώδικας Gray	17
E3.2 Μετατροπή από big-endian σε little-endian	18
E3.3 Πρόσθεση και Υπερχείλιση	18
E4 Έλεγχος Ροής Προγράμματος	21
E4.1 Ωφέλιμα bytes	21
E4.2 Απόσταση Hamming	21
E4.3 Μετατροπή πεζών γραμμάτων σε κεφαλαία	21
E5 Διαδικασίες και Στοίβα	22
E5.1 Πράξεις με συμβολοσειρές	22
E5.2 Ακολουθία Fibonacci	22
E6 Χειρισμός Κρατούμενου, Πολλαπλασιασμός	24
E6.1 Τελικό κρατούμενο	24
E6.2 Πρόσθεση 64-bit αριθμών	25
E6.3 Πολλαπλασιασμός	26
E7 Αριθμητική Κινητής Υποδιαστολής	28
E7.1 Διάβασμα και εκτύπωση ενός αριθμού κινητής υποδιαστολής	28
E7.2 Πράξεις με αριθμούς κινητής υποδιαστολής	28
E7.3 Υπολογισμός του παραγοντικού	29
E7.4 Υλοποίηση εντολής κινητής υποδιαστολής	29
2 Ο προσομοιωτής QtRVsim	31
E8 Εισαγωγή στον προσομοιωτή QtRVsim	32
E8.1 Τα βασικά του QtRVsim	32
E8.2 Προσομοίωση επεξεργαστή ενός κύκλου	34
E9 Προσομοίωση επεξεργαστή με διοχέτευση	36
E9.1 Διοχέτευση με μονάδα ελέγχου κινδύνων	36

E9.1.1	Κίνδυνοι Δεδομένων	36
E9.1.2	Κίνδυνοι Ελέγχου	37
E9.2	Διοχέτευση χωρίς μονάδα ελέγχου κινδύνων	38
E10	Βελτίωση ενός επεξεργαστή με διοχέτευση	39
E10.1	Περιορισμός των κινδύνων της διοχέτευσης	39
E10.1.1	Μονάδα ελέγχου κινδύνων με προώθηση	39
E10.1.2	Πρόβλεψη διακλάδωσης	40
E10.2	Σύγκριση επεξεργαστών ενός κύκλου και με διοχέτευση	42
A'	Επισκόπηση του επεξεργαστή RISC-V	45
A'.1	Τύποι δεδομένων	46
A'.2	Οργάνωση της μνήμης	47
A'.3	Διάταξη μνήμης	48
A'.4	Οι καταχωρητές του επεξεργαστή	49
A'.5	Μορφή και βασικά χαρακτηριστικά προγραμμάτων	51
A'.5.1	Σχόλια	51
A'.5.2	Οδηγίες προς τον συμβολομεταφραστή	51
A'.5.3	Δηλώσεις δεδομένων	51
A'.5.3.1	Δήλωση ακεραίων	53
A'.5.3.2	Δήλωση συμβολοσειράς	53
A'.5.3.3	Δήλωση Αριθμών Κινητής Υποδιαστολής	54
A'.5.4	Ο κώδικας	54
A'.5.4.1	Η οδηγία .text	54
A'.5.4.2	Ετικέτες (Labels)	55
A'.5.5	Πρότυπο Προγραμμάτων	55
B'	Κλήσεις συστήματος (System Calls)	57
B'.1	Υποστηριζόμενες κλήσεις συστήματος	58
B'.2	Χρήση των κλήσεων συστήματος	60
Γ'	Εγκατάσταση και Εκτέλεση του προσομοιωτή RARS	61
G'.1	Εγκατάσταση του προσομοιωτή RARS	62
G'.1.1	Εγκατάσταση της Java	62
G'.2	Εκτέλεση του προσομοιωτή RARS	63

Μέρος 1

Ο προσομοιωτής RARS

Οι προσομοιωτές (simulators) είναι προγράμματα λογισμικού (software) που προσεγγίζουν με όσο δυνατόν πιο ακριβή τρόπο τη λειτουργία ενός πραγματικού επεξεργαστή ή ακόμα και ενός ολόκληρου υπολογιστικού συστήματος. Ο προσομοιωτής RARS προσομοιώνει τη συμπεριφορά ενός επεξεργαστή RISC-V σε ό,τι αφορά τα περιεχόμενα των καταχωρητών και της μνήμης μετά την εκτέλεση κάθε εντολής. Είναι ένας προσομοιωτής επιπέδου αρχιτεκτονικής (Architecture-Level Simulator) ή προσομοιωτής επιπέδου εντολής (Instruction-Level Simulator). Ο προσομοιωτής RARS (**R**I**S**C-**V** **A**ssembler and **R**untime **S**imulator) εκτελεί τη συμβολομετάφραση (assembling) των εντολών συμβολικής γλώσσας (assembly language) σε γλώσσα μηχανής (machine language) και προσομοιώνει την εκτέλεση προγραμμάτων RISC-V. Ο κύριος στόχος του είναι να αποτελέσει ένα αποτελεσματικό περιβάλλον ανάπτυξης για όσους ξεκινούν την ενασχόλησή τους με την αρχιτεκτονική και το σύνολο εντολών των επεξεργαστών RISC-V.



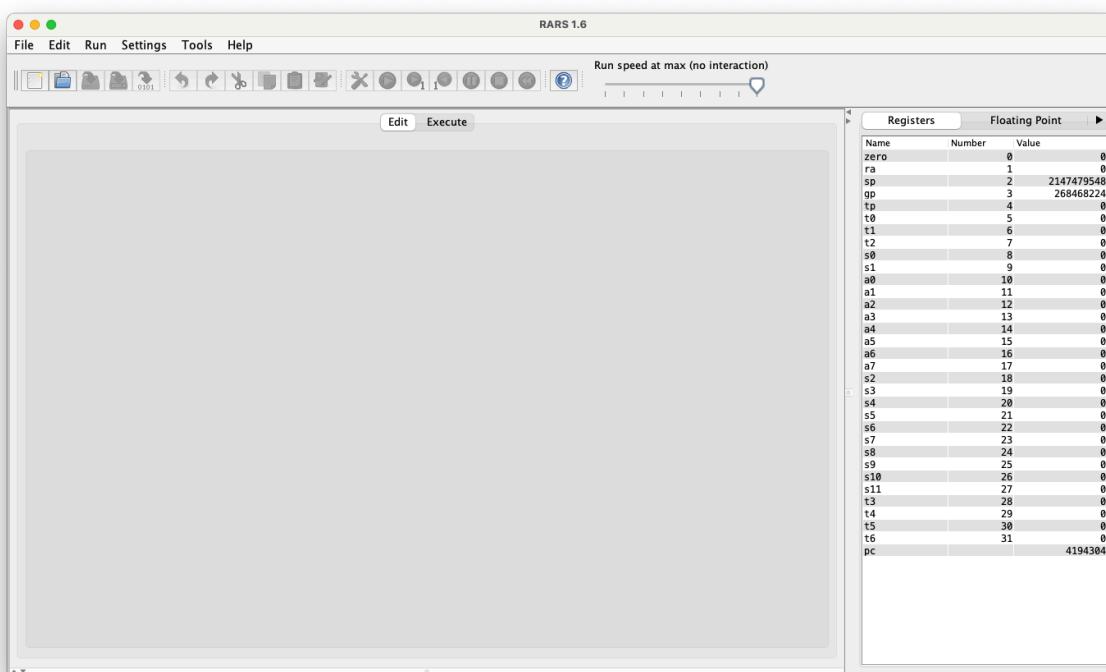
Ε1 Εισαγωγή στον προσομοιωτή RARS

Σκοπός του πρώτου εργαστηριακού μαθήματος είναι η εξοικείωση με το περιβάλλον του προσομοιωτή (simulator) και συμβολομεταφραστή (assembler) RARS ο οποίος προσομοιώνει την εκτέλεση προγραμμάτων σε συμβολική γλώσσα (assembly) επεξεργαστών RISC-V. Θα δούμε τα βήματα της προσομοίωσης, όπως επίσης και τα βασικά μέρη ενός προγράμματος συμβολικής γλώσσας RISC-V όπως για παράδειγμα οδηγίες προς τον συμβολομεταφραστή, ετικέτες, ψευδοεντολές, είσοδο/έξοδο στην κονσόλα, τερματισμό του προγράμματος, καθώς και κλήσεις συστήματος. Επιπλέον, θα επισημάνουμε βασικές μεθόδους αποσφαλμάτωσης (debugging) βασισμένες σε λειτουργίες του προσομοιωτή RARS.

Εγκαταστήστε τον προσομοιωτή RARS και την απαιτούμενη έκδοση Java σύμφωνα με το παράρτημα [Γ'.1](#).

Ε1.1 Φόρτωση προγραμμάτων στον RARS

Κατεβάστε το αρχείο **lab1.s** από το eClass του μαθήματος και εκκινήστε τον προσομοιωτή RARS βάσει των οδηγιών στο Παράρτημα [Γ'.2](#). Το γραφικό περιβάλλον του προσομοιωτή παρουσιάζεται στην Εικόνα 1.



Εικόνα 1: Η αρχική οθόνη του RARS

Πατώντας File → Open ή Ctrl+O ή το εικονίδιο ανοίγει ένα πλαίσιο διαλόγου για το άνοιγμα ενός αρχείου. Βρείτε και επιλέξτε το αρχείο lab1.s.

Ο κώδικας βρίσκεται στο τμήμα Edit της οθόνης, το οποίο δίνει τη δυνατότητα για αλλαγές αν το επιθυμείτε, όπως φαίνεται στην Εικόνα [2](#).



The screenshot shows the RARS 1.6 debugger interface. The assembly code in the main window is as follows:

```

1 .text
2 .globl __start
3 __start:
4 la t2, str      # t2 points to the string
5 li t1, 0        # t1 holds the count
6 nextCh:
7 lb t0, 0(t2)   # get a byte from string
8 beqz t0, strEnd # zero means end of string
9 addi t1, t1, 1  # increment count
10 addi t2, t2, 1 # move pointer one character
11 j nextCh       # go round the loop again
12 strEnd:
13 la a0, ans    # system call to print out
14 li a7, 4       # a message
15 ecall
16 mv a0, t1      # system call to print out
17 li a7, 1       # the length
18 ecall
19 la a0, endl   # system call to print out
20 li a7, 4       # end of line
21 ecall
22 li a7, 10     # au revoir ...
23 ecall
24
25
26 .data
27 str: .asciz "hello world"
28 ans: .asciz "Length is "
29 endl: .asciz "\n"
30

```

The Registers window shows the following values:

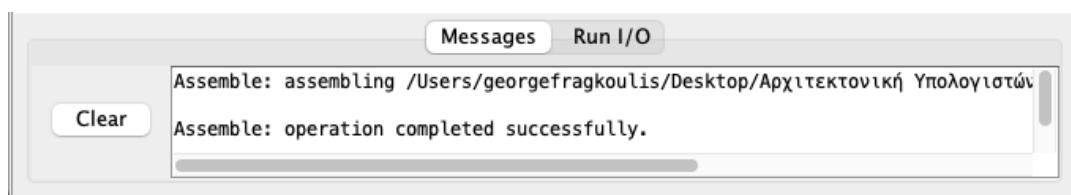
Name	Number	Value
zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	0
s1	9	0
a0	10	0
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194304

The Messages window shows:

- Assemble: assembling /Users/georgefragkoulis/Desktop/Αρχιτεκτονική Υπολογιστών
- Assemble: operation completed successfully.

Εικόνα 2: Επεξεργασία κώδικα στον RARS

Πατώντας Run → Assemble ή F3 ή το εικονίδιο πραγματοποιείται η συμβολομετάφραση του κώδικα. Εδώ θα εμφανιστούν τα συντακτικά λάθη του κώδικα. Σε περίπτωση επιτυχούς συμβολομετάφρασης θα εμφανιστεί μήνυμα στο τμήμα Messages όπως εκείνο στην Εικόνα 3.



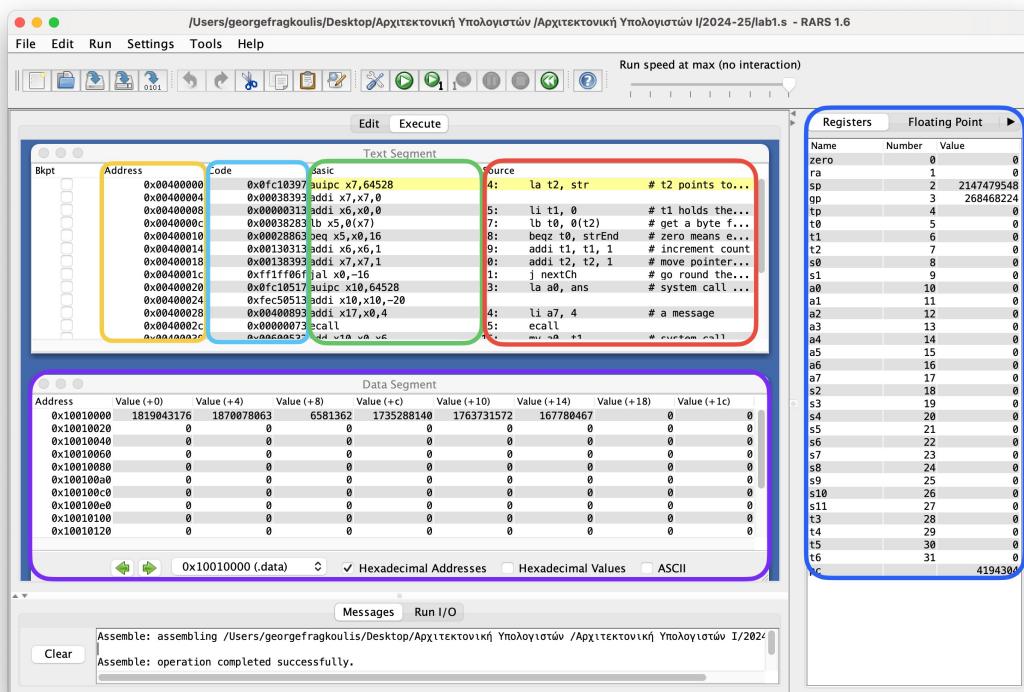
Εικόνα 3: Μήνυμα επιτυχούς συμβολομετάφρασης στον RARS

Μετά από την συμβολομετάφραση, ανοίγει το παράθυρο Execute, όπως παρουσιάζεται στην Εικόνα 4. Παρατηρείστε την προσεκτικά. Περιλαμβάνει όλες τις



πληροφορίες που σχετίζονται με τον κώδικα και την εκτέλεση του. Αναλυτικότερα:

- Στο **κόκκινο** πλαίσιο υπάρχει το τμήμα του κώδικα του αρχείου lab1.s που υπάρχει κάτω από την οδηγία .text. Οι αριθμοί που υπάρχουν αριστερά αντιστοιχούν στον αριθμό των γραμμών του αρχείου lab1.s.
- Πιο αριστερά, στο **πράσινο** πλαίσιο, υπάρχουν οι πραγματικές εντολές που θα εκτελεστούν οι οποίες είναι περισσότερες από αυτές που υπάρχουν στο αρχείο που φορτώθηκε διότι στο αρχείο υπάρχουν ψευδοεντολές. Για παράδειγμα, η πρώτη εντολή, στη γραμμή 4, είναι η la t2, str, η οποία αντιστοιχεί σε δύο πραγματικές εντολές, τις auipc x7, 64528 και addi x7, x7, 0. Όπως βλέπετε δεξιά, στους καταχωρητές (**μπλε** πλαίσιο), ο καταχωρητής t2 έχει αριθμό 7 και επομένως αντιστοιχεί στον x7.
- Το **γαλάζιο** πλαίσιο περιλαμβάνει τις διευθύνσεις των εντολών που θα εκτελεστούν σε δεκαεξαδικό σύστημα αρίθμησης. Η κάθε διεύθυνση έχει μέγεθος 4 bytes και απέχουν μεταξύ τους 4 bytes.
- Το **κίτρινο** πλαίσιο περιλαμβάνει τη δεκαεξαδική αναπαράσταση της κωδικοποίησης (encoding) των εντολών που θα εκτελεστούν.
- Στο **μωβ** πλαίσιο, υπάρχει το τμήμα δεδομένων του προγράμματος. Στη στήλη Address αντιστοιχούν οι διευθύνσεις μνήμης των δεδομένων, ενώ στις διπλανές στήλες υπάρχουν τα δεδομένα, με την κάθε διπλανή στήλη να περιέχει 4 bytes δεδομένων. Τα δεδομένα που αποθηκεύονται εδώ είναι εκείνα που δηλώνονται κάτω από την οδηγία .data και υπάρχει δυνατότητα αναπαράστασης σε δεκαδικό σύστημα (προεπιλογή) ή σε δεκαεξαδικό σύστημα ή σε ASCII χαρακτήρες.

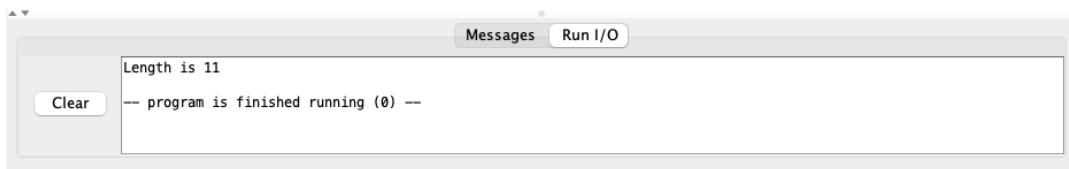


Εικόνα 4: Παράθυρο εκτέλεσης κώδικα RARS



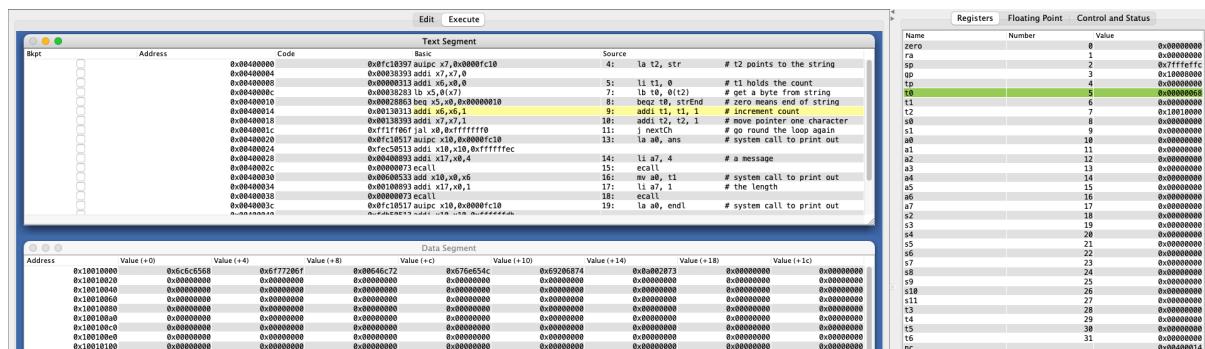
Ε1.2 Εκτέλεση προγραμμάτων στον RARS

Ο RARS παρέχει τρεις τρόπους εκτέλεσης ενός προγράμματος. Πρώτον, **εξολοκλήρου** εκτέλεση (Πίνακας 1), δεύτερον **βηματική εκτέλεση** (Πίνακας 1) και τρίτον εκτέλεση με συγκεκριμένη διεκπαιρεωτική ικανότητα (**εντολές/δευτερόλεπτο**). Τρέξτε ολόκληρο το πρόγραμμα lab1.s και παρατηρείστε την έξοδο του, η οποία αποτελεί ένα μήνυμα στο χρήστη (Εικόνα 5).



Εικόνα 5: Παράθυρο για είσοδο/έξοδο στον RARS

Παρατηρείστε ότι τα περιεχόμενα των καταχωρητών έχουν αλλάξει. Προκειμένου να το διαπιστώσετε καλύτερα μπορείτε να επαναφέρετε τη μνήμη και τους καταχωρητές στην αρχική κατάσταση πατώντας (Πίνακας 1) και στη συνέχεια τρέξτε τον κώδικα βηματικά. Τώρα, υπάρχουν χρωματικές ενδείξεις τόσο για την **εντολή που πρόκειται να εκτελεστεί** αλλά και για τους **καταχωρητές** και τη **θέση μνήμης** που αλλάζει η τιμή τους (Εικόνα 6).



Εικόνα 6: Βηματική εκτέλεση κώδικα στον RARS

Ε1.3 Διαδικασία αποσφαλμάτωσης στον RARS

Υπάρχουν δύο είδη σφαλμάτων, τα συντακτικά και τα λογικά λάθη. Τα συντακτικά λάθη εντοπίζονται κατά τη συμβολομετάφραση και πρέπει να επιλυθούν προκειμένου να εκτελεστεί ο κώδικας.

Δυσκολότερα αντιμετωπίζονται τα λογικά λάθη που προκύπτουν από τη λανθασμένη υλοποίηση του αλγορίθμου στον κώδικα και έχουν ως απόρροια μη αναμενόμενα αποτελέσματα κατά την εκτέλεση.

Γενικά, για την διόρθωση λογικών λαθών χρησιμοποιούμε:

- Βηματική εκτέλεση του προγράμματος παρατηρώντας τις τιμές τόσο των καταχωρητών όσο και της μνήμης (Εικόνα 6) και συγκρίνοντας τες με τις αναμενόμενες.



- Σημεία διακοπής. Ένα σημείο διακοπής (breakpoint) είναι μια θέση στο πρόγραμμα, όπου η εκτέλεση θα διακοπεί πριν εκτελεστεί η εντολή που βρίσκεται στη θέση αυτή. Όταν το πρόγραμμα σταματήσει στο σημείο αυτό, η κατάσταση του προγράμματος μπορεί να ελεγχθεί με την προβολή των περιεχομένων των καταχωρητών και της μνήμης.

Κατεβάστε το αρχείο **lab1_errors.s** από το eClass του μαθήματος, φορτώστε το στον προσωμοιωτή RARS και δοκιμάστε να το συμβολομεταφράσετε. Το αποτέλεσμα που θα πάρετε θα είναι παρόμοιο με αυτό της Εικόνας 7. Παρατηρείστε ότι αν πατήσετε πάνω σε κάποιο σφάλμα, θα χρωματιστεί και η αντίστοιχη γραμμή που το περιέχει.

The screenshot shows the RARS 1.6 simulator interface. The assembly code window displays the `lab1_errors.s` file. The code includes instructions like `la t2, str`, `addi t7, t7, 1`, and `mv a0, t2`. The Registers window shows various CPU registers (zero through t6, plus s0-s10, a1-a3, and pc) with their current values. The Messages window shows three error messages indicating type errors at specific assembly lines. The Run speed at max (no interaction) button is checked.

```
1 .text
2 .globl __start
3 __start:
4 la t2, str      # t2 points to the string
5 li t7, 0        # t7 holds the count
6 nextch:
7 lb t0, 0(t2)   # get a byte from string
8 begz t0, strEnd # zero means end of string
9 addi t7, t7, 1   # increment count
10 add t2, t2, 1    # move pointer one character
11 j nextch       # go round the loop again
12 strEnd:
13 la a0, ans     # system call to print out
14 li a7, 4        # a message
15 ecall
16 mv a0, t2       # system call to print out
17 li a7, 1        # the length
18 ecall
19 la a0, endl     # system call to print out
20 li a7, 4        # end of line
21 ecall
22 li a7, 10       # au revoir ...
23 ecall
24
25
26 .data
27 str: .asciz "hello world"
28 ans: .asciz "Length is "
29 endl: .asciz "\n"
30
```

Line: 5 Column: 45 Show Line Numbers

Messages Run I/O

Ιερά Μητρόπολη Καρδίτσας / Αρχιτεκτονική Υπολογιστών I/2024-25/lab1_errors.s
Ιερά Μητρόπολη Καρδίτσας / Αρχιτεκτονική Υπολογιστών I/2024-25/lab1_errors.s line 5 column 8: "t7": operand is of incorrect type
Ιερά Μητρόπολη Καρδίτσας / Αρχιτεκτονική Υπολογιστών I/2024-25/lab1_errors.s line 9 column 10: "t7": operand is of incorrect type
Ιερά Μητρόπολη Καρδίτσας / Αρχιτεκτονική Υπολογιστών I/2024-25/lab1_errors.s line 10 column 17: "1": operand is of incorrect type

Εικόνα 7: Αναγνώριση συντακτικών λαθών στον RARS

Τα πρώτα δύο σφάλματα είναι ίδια και αναφέρονται στη αναγνώριση λανθασμένου τελεστέος (`t7`). Κοιτώντας τους καταχωρητές θα διαπιστώσετε ότι δεν υπάρχει τέτοιος καταχωρητής αφού οι προσωρινοί καταχωρητές είναι από `t0-t6`. Αλλάξτε το πρόγραμμα έτσι ώστε να μην χρησιμοποιείται ο `t7` αλλά ο `t1`.

Το επόμενο σφάλμα αναφέρεται πάλι σε λανθασμένο τελεστέο. Αυτή τη φορά, υπάρχει πρόβλημα σχετικά με την σταθερά 1 δεδομένου ότι χρησιμοποιείται η εντολή `add`, η οποία απαιτεί τρεις καταχωρητές για τελεστέους. Αλλάξτε την εντολή `add` σε `addi` και επαναλάβετε την μετάφραση.

Τρέξτε το πρόγραμμα χωρίς διακοπή (green circle). Τι παρατηρείτε;



Δοκιμάστε να βάλετε σημείο διακοπής (breakpoint) στην εντολή με διεύθυνση 0x00400020 (auipc x10, 0x0000fc10) επιλέγοντας το τετράγωνο δίπλα από τη διεύθυνση (Εικόνα 8).

Τρέξτε το πρόγραμμα εξ ολοκλήρου και επιβεβαιώστε ότι η εκτέλεση θα σταματήσει στο σημείο που επιλέξατε. Στο συγκεκριμένο σημείο εκτέλεσης ο καταχωρητής t1 θα πρέπει να περιέχει την τιμή 11. Τι παρατηρείτε; Τρέξτε βηματικά το υπόλοιπο πρόγραμμα παρατηρώντας τις αλλαγές στις τιμές των καταχωρητών και προσπαθήστε να το διορθώσετε.

Text Segment			
Bkpt	Address	Code	Source
	0x00400000	0xfc10297 auipc x7,0x0000fc10	4: la t2, str # t2 points to the string
	0x00400004	0x00038393 addi x7,x7,0	5: li t1, 0 # t7 holds the count
	0x00400008	0x00000013 addi x6,x0,0	7: lb t0, 0(t2) # get a byte from string
	0x0040000c	0x00038283 lb x5,0(x7)	8: beqz t0, strEnd # zero means end of string
	0x00400010	0x00028863 beq x5,x0,0x00000010	9: addi t1, t1, 1 # increment count
	0x00400014	0x00130313 addi x6,x6,1	10: addi t2, t2, 1 # move pointer one character
	0x00400018	0x00138393 addi x7,x7,1	11: j nextCh # go round the loop again
	0x0040001c	0xf1fff06f jal x0,0xfffffffff0	13: la a0, ans # system call to print out
	0x00400020	0xfc10517 auipc x10,0x0000fc10	
	0x00400024	0xec50513 addi x10,x10,0xfffffffec	
	0x00400028	0x00400093 addi x17,x0,4	14: li a7, 4 # a message
	0x0040002c	0x00000073 ecall	15: ecall
	0x00400030	0x00700533 addi x10,x0,x7	16: mv a0, t2 # system call to print out
	0x00400034	0x00100093 addi x17,x0,1	17: li a7, 1 # the length
	0x00400038	0x00000073 ecall	18: ecall
	0x0040003c	0xfc10517 auipc x10,0x0000fc10	19: la a0, endl # system call to print out
	0x00400040	0x00000000 addi x0,x0,0xxxxxxxx	

Εικόνα 8: Σημείο διακοπής στον RARS



Ε2 Διαχείριση μνήμης και κλήσεις συστήματος

Σε αυτή την εργαστηριακή άσκηση θα ασχοληθούμε με τη συγγραφή και ανάλυση απλών προγραμμάτων σε συμβολική γλώσσα RISC-V, αξιοποιώντας τον προσομοιωτή RARS για να μελετήσουμε τις λειτουργίες των βασικών εντολών και μηχανισμών της αρχιτεκτονικής RISC-V. Θα εστιάσουμε στη δήλωση και προσπέλαση των δεδομένων της μνήμης καθώς και στη χρήση καταχωρητών για την αποθήκευση και επεξεργασία τιμών. Επιπλέον, θα μελετήσουμε την αλληλεπίδραση μεταξύ δεδομένων και εντολών για την υλοποίηση βασικών λειτουργιών όπως η φόρτωση, η πρόσθεση και η αποθήκευση δεδομένων. Τέλος, θα δούμε τον τρόπο εισαγωγής αριθμού από το χρήστη και την εκτύπωσή του.

E2.1 Φόρτωση και αποθήκευση αριθμών

Κατεβάστε από το eClass του μαθήματος το πρότυπο προγραμμάτων συμβολικής γλώσσας RISC-V (template.s). Μετονομάστε το αρχείο σε lab2_ex1.s και ανοίξτε το για να ξεκινήσετε τη συγγραφή του κώδικα σας για αυτή την άσκηση.

Θεωρήστε το τμήμα δεδομένων που παρουσιάζεται στον Κώδικα 1. Όπως βλέπετε, έχουν δηλωθεί δύο αριθμοί *A* και *B*.

- α'.** Να αντιγράψετε τον Κώδικα 1 στο τμήμα δεδομένων του προγράμματός σας.
- β'.** Στη συνέχεια, στο κυρίως πρόγραμμα (στο τμήμα κώδικα – κάτω από την ετικέτα `_start:`) χρησιμοποιήστε το ζευγάρι εντολών που αποτελείται από την ψευδοεντολή `la` (load address) και την πραγματική εντολή `lw` (load word) για να φορτώσετε στους καταχωρητές `s0, s1` τους αριθμούς *A* και *B* από τη μνήμη δεδομένων.
- γ'.** Υπολογίστε το άθροισμα των δύο αριθμών και αποθηκεύστε το αποτέλεσμα στον καταχωρητή `s2`.
- δ'.** Αποθηκεύστε το αποτέλεσμα στη θέση μνήμης που δηλώνει η ετικέτα `sum` χρησιμοποιώντας το ζευγάρι εντολών που αποτελείται από την ψευδοεντολή `la` (load address) και την πραγματική εντολή αποθήκευσης στη μνήμη δεδομένων `sw` (store word).
- ε'.** Φορτώστε και εκτελέστε βηματικά το πρόγραμμά σας στον RARS. Παρατηρήστε τις αλλαγές στα περιεχόμενα των καταχωρητών που χρησιμοποιήσατε όπως φαίνονται στο υπο-παράθυρο καταχωρητών του RARS καθώς εκτελούνται (βηματικά) οι εντολές, όπως και τη μνήμη δεδομένων στην οποία φαίνεται το άθροισμα των αριθμών *A* και *B*.

```
1     .data
2 A:      .word 2
3 B:      .word 3
4 sum:    .word 0
```

Κώδικας 1: Τμήμα δεδομένων για την Άσκηση E2.1



E2.2 Διευθύνσεις μνήμης και δεδομένα

Κατεβάστε από το eClass του μαθήματος το πρότυπο προγραμμάτων συμβολικής γλώσσας RISC-V (template.s). Μετονομάστε το αρχείο τοπικά στον υπολογιστή σας σε lab2_ex2.s και ανοίξτε το για να ξεκινήσετε τη συγγραφή του κώδικα σας για αυτή την άσκηση.

Θεωρήστε το τμήμα δεδομένων που παρουσιάζεται στον Κώδικα 2. Όπως βλέπετε, έχουν δηλωθεί 4 διαδοχικά bytes και μια λέξη (word) εκφρασμένη σε δεκαεξαδικό σύστημα. Επομένως, έχουμε συνολικά $4 + 4 = 8$ bytes στη μνήμη δεδομένων.

- α'.** Να αντιγράψετε τον Κώδικα 2 στο τμήμα δεδομένων του προγράμματός σας.
- β'.** Φορτώστε και συμβολομεταφράστε το πρόγραμμά σας στον RARS. Παρατηρήστε τη μνήμη δεδομένων στον RARS και συμπληρώστε τον Πίνακα 1α'.
- γ'.** Φορτώστε τη λέξη που ξεκινά από τη διεύθυνση 0x10010000 στον καταχωρητή s0 και συμπληρώστε τον Πίνακα 1β'.
- δ'.** Τι σχέση έχει το περιεχόμενο της θέσης μνήμης 0x10010000 με το Byte 0 του καταχωρητή s0;
Ο RISC-V χρησιμοποιεί την οργάνωση «μικρού άκρου» (little-endian) με αποτέλεσμα το λιγότερο σημαντικό byte (Byte 0) να βρίσκεται στη μικρότερη διεύθυνση μνήμης. Για παραπάνω λεπτομέρειες μπορείτε να ανατρέξετε στο Παράρτημα A'.2.
- ε'.** Εντοπίστε τα περιεχόμενα των διευθύνσεων 0x10010003 και 0x10010007. Συγκρίνετε τα bytes θεωρώντας ότι αναπαριστούν προσημασμένους χαρακτήρες (signed char). Άλλάζει το αποτέλεσμα εάν τα συγκρίνουμε ως απροσήμαστους χαρακτήρες (unsigned char); Αιτιολογείστε την απάντησή σας.

```

1 .data
2 Bytes: .byte 0x11, 0x22, 0x33, 0x44
3 Word: .word 0x88776655

```

Κώδικας 2: Τμήμα δεδομένων για την Άσκηση E2.2

Διεύθυνση Byte Δεδομένα

0x10010000
0x10010001
0x10010002
0x10010003
0x10010004
0x10010005
0x10010006
0x10010007

(α') Μνήμη δεδομένων

Byte 3 Byte 2 Byte 1 Byte 0

(β') Καταχωρητής s0

Πίνακας 1: Ζητούμενα της Άσκησης E2.2



E2.3 Ακολουθία αριθμών στη μνήμη

Θεωρείστε τον Κώδικα 3. Η εντολή `lw t0, 4(t0)` υπάρχει k φορές.

- α'. Τρέξτε τον Κώδικα 3 με k εντολές `lw t0, 4(t0)` κάθε φορά, $0 \leq k \leq 4$, και καταγράψτε το k -στό περιεχόμενο του `s0` μετά το πέρας κάθε εκτέλεσης.
- β'. Τρέξτε βηματικά την καθεμιά από τις $k+1$ προσομοιώσεις και παρατηρήστε τις αλλαγές στα περιεχόμενα των καταχωρητών που σας ενδιαφέρουν. Σας θυμίζει κάτι αυτή η συμπεριφορά;
- γ'. Επεκτείνετε τον Κώδικα 3 προσθέτοντας **μόνο** εντολές `sw` και `addi`, έτσι ώστε η ακολουθία a_k που αποτυπώνει το k -στό περιεχόμενο του καταχωρητή `s0` μετά το πέρας της κάθε εκτέλεσης να ισούται με $a_k = \{1, 3, 5, 12, 7, 9\}$.

```
1 .text
2 .globl __start
3
4 __start:
5   la t0, Elements
6
7   lw t0, 4(t0)
8   lw t0, 4(t0)
9   .
10  .
11  .
12  lw t0, 4(t0)
13
14  lw s0, 0(t0)
15  li a7, 10
16  ecall
17
18  .data
19 Elements: .word 1, 0x10010020, 9, 0
20        .word 5, 0x10010018, 7, 0x10010008
21        .word 3, 0x10010010, 0, 0
```

Κώδικας 3: Ο κώδικας της Άσκησης E2.3

E2.4 Διάβασμα και εκτύπωση ενός ακέραιου αριθμού

Η κονσόλα αποτελεί τη διεπαφή αλληλεπίδρασης του χρήστη με το πρόγραμμα. Να γράψετε πρόγραμμα σε συμβολική γλώσσα RISC-V χρησιμοποιώντας το πρότυπο προγραμμάτων (`template.s`), στο οποίο θα διαβάζετε έναν ακέραιο αριθμό από την κονσόλα (καρτέλα Run I/O του RARS) και θα τον εκτυπώνετε. Μπορείτε να συμβουλευτείτε το Παράρτημα B' για πληροφορίες σχετικά με τις κλήσεις συστήματος και το πως αυτές χειρίζονται στον προσομοιωτή RARS.



Ε3 Αριθμητικές και Λογικές Πράξεις

Σε αυτή την εργαστηριακή άσκηση μελετώνται οι εντολές αριθμητικών πράξεων ακεραίων αριθμών του επεξεργαστή RISC-V και οι εντολές λογικών πράξεων μεταξύ λέξεων. Ειδικά για τις αριθμητικές πράξεις πρόσθεσης-αφαίρεσης θα εξετασθεί και η συμπεριφορά τους σε ακραίες περιπτώσεις όπως στην περίπτωση της υπερχείλισης (overflow).

E3.1 Ο κώδικας Gray

Να γράψετε ένα πρόγραμμα με όνομα lab3_ex1.s σε συμβολική γλώσσα RISC-V που να μετατρέπει έναν δυαδικό αριθμό των 32-bit στον αντίστοιχο του σε κωδικοποίηση Gray. Ο αλγόριθμος μετατροπής ενός δυαδικού αριθμού με n bits σε αναπαράσταση κώδικα Gray δίνεται από την Εξίσωση 1.

$$g_i = \begin{cases} b_i, & i = n - 1 \\ b_{i+1} \oplus b_i, & i = 0, 1, \dots, n - 2 \end{cases}, \quad (1)$$

όπου \oplus είναι η λογική πράξη XOR, b_i είναι το bit i -στής τάξης του δυαδικού αριθμού και g_i το bit i -στής τάξης του αντίστοιχου Gray αριθμού (το i να λαμβάνει τιμές από 0 μέχρι και $n - 1$).

Χρησιμοποιήστε μόνο λογικές πράξεις και ολισθήσεις για να υλοποιήσετε τον παραπάνω αλγόριθμο (δε χρειάζεται βρόχος επανάληψης – τους βρόχους θα τους μελετήσουμε αργότερα).

Binary	Hex	Gray
0000	0	0000
0001	1	0001
0010	2	0011
0011	3	0010
0100	4	0110
0101	5	0111
0110	6	0101
0111	7	0100

Binary	Hex	Gray
1000	8	1100
1001	9	1101
1010	A	1111
1011	B	1110
1100	C	1010
1101	D	1011
1110	E	1001
1111	F	1000

Πίνακας 2: Ο κώδικας Gray για 4 bit

Υπόδειξη: Ο αριθμός θα εισάγεται απευθείας σε ένα καταχωρητή. Αυτό γίνεται με διπλό κλικ στο περιεχόμενο του επιθυμητού καταχωρητή στο υπο-παράθυρο των καταχωρητών του RARS. Βεβαιωθείτε ότι τα περιεχόμενα των καταχωρητών εμφανίζονται σε δεκαεξαδική μορφή ώστε να είναι εύκολη η μετατροπή τους σε δυαδικό σύστημα (από το μενού Settings πρέπει να είναι ενεργοποιημένη η επιλογή Values displayed in hexadecimal).

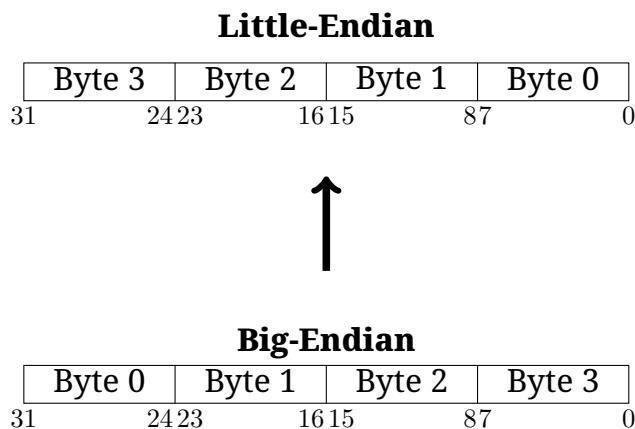
Πειραματιστείτε με μικρούς αριθμούς και συμβουλευτείτε τον παραπάνω πίνακα αντιστοίχισης για τον κώδικα Gray των 4 bit για να επιβεβαιώσετε το αποτέλε-



σμα, παρατηρώντας τον Gray κωδικοποιημένο αριθμό που θα βρίσκεται σε έναν καταχωρητή της επιλογής σας.

E3.2 Μετατροπή από big-endian σε little-endian

Όπως έχουμε μάθει, κατά κανόνα οι επεξεργαστές της αρχιτεκτονικής RISC-V λειτουργούν χρησιμοποιώντας το μοντέλο μνήμης little-endian (μικρού άκρου) για την οργάνωση των byte. Υποθέστε ότι ένας τέτοιος RISC-V επεξεργαστής επικοινωνεί με διάφορες άλλες μηχανές, κάποιες από τις οποίες χρησιμοποιούν το μοντέλο μνήμης big-endian (μεγάλου άκρου) για την οργάνωση των byte. Τα δεδομένα που προέρχονται από τις άλλες μηχανές είναι λέξεις των 32 bit και εισάγονται σε έναν καταχωρητή, όχι όμως με τη μορφή που τα χρειάζεται ο RISC-V. Σε αυτή την άσκηση θέλουμε με προγραμματιστικό τρόπο σε συμβολική γλώσσα RISC-V να μετατρέψουμε τα δεδομένα που λαμβάνονται από τη «λανθασμένη» μορφή big-endian στη μορφή little-endian σε έναν άλλο καταχωρητή, όπως φαίνεται στην Εικόνα 9.



Εικόνα 9: Σύγκριση big-endian και little-endian

ΣΗΜΑΝΤΙΚΟ. Να σημειωθεί ότι μέσα σε κάθε byte η σειρά των bit είναι η σωστή (δηλαδή το byte 0 αποθηκεύει τα bit 7-0, το byte 1 αποθηκεύει τα bit 15-8, κ.λπ.) Να γράψετε το σχετικό πρόγραμμα μετατροπής σε συμβολική γλώσσα RISC-V, χωρίς να χρησιμοποιήσετε καθόλου προσπέλαση στη μνήμη, παρά μόνο με λογικές πράξεις μεταξύ καταχωρητών και αυτό για λόγους ταχύτητας επεξεργασίας.

E3.3 Πρόσθεση και Υπερχείλιση

α'. Θεωρήστε τα ζευγάρια προσημασμένων δυαδικών αριθμών των 32-bit σε αναπαράσταση συμπληρώματος ως προς 2 της Εικόνας 10. Κάντε τις προσθέσεις που φαίνονται σε δυαδικό και στο δεκαδικό σύστημα και συμπληρώστε το αποτέλεσμα στα αντίστοιχα κουτάκια παρακάτω. Επίσης συμπληρώστε το σχετικό bit V της υπερχείλισης (1 αν έχει συμβεί υπερχείλιση και 0 αν δεν έχει συμβεί). Επιπλέον συμπληρώστε τα κρατούμενα που δίνει η πρόσθεση στα bit τελευταίας και προτελευταίας τάξης, δηλαδή τα C_{32} και C_{31} , αντίστοιχα.



ΠΡΟΣΟΧΗ. Τα bit C_{32} , C_{31} και V δεν υπάρχουν στον RISC-V αλλά τα χρησιμοποιούμε εδώ απλά ως συμβολισμό της άσκησης.

- β'.** Ελέγξτε κατά πόσον το αποτέλεσμα στο δεκαδικό σύστημα είναι ίδιο με αυτό της δυαδικής αναπαράστασης.
- γ'.** Αρχικά, εισάγετε τα παραπάνω ζευγάρια αριθμών στο τμήμα δεδομένων. Προτείνεται να εισάγετε τα δεδομένα ως λέξεις και να χρησιμοποιήσετε μια ετικέτα (label) για κάθε έναν αριθμό που θα δηλώσετε, ώστε να είναι ευκολότερη η χρήση των αριθμών μετέπειτα από το πρόγραμμά σας.
- δ'.** Να γράψτε ένα πρόγραμμα σε συμβολική γλώσσα RISC-V το οποίο θα φορτώνει από τη μνήμη ένα-ένα το κάθε ζευγάρι αριθμών σε δύο καταχωρητές, θα τους προσθέτει με την εντολή add και θα αποθηκεύει το αποτέλεσμα σε έναν τρίτο καταχωρητή. Αυτό θα το κάνετε και για τα τέσσερα ζευγάρια αριθμών αλλά χρησιμοποιώντας διαφορετικούς καταχωρητές για κάθε αποτέλεσμα. Ουσιαστικά, το πρόγραμμα που θα γράψετε θα εκτελεί τις ίδιες πράξεις που κάνατε στα προηγούμενα υπο-ερωτήματα «με το χέρι».
- Υπόδειξη:** Βεβαιωθείτε ότι τα περιεχόμενα των καταχωρητών εμφανίζονται σε δεκαεξαδική μορφή ώστε να είναι εύκολη η μετατροπή τους σε δυαδικό σύστημα (από το μενού Settings πρέπει να είναι ενεργοποιημένη η επιλογή Values displayed in hexadecimal).
- ε'.** Επεκτείνετε τη λειτουργία του προγράμματος που γράψατε στο προηγούμενο υποερώτημα έτσι ώστε να εκτυπώνει τα τέσσερα αποτελέσματα των προσθέσεων στην οθόνη (στην καρτέλα Run I/O του RARS).
- στ'.** Πού φαίνεται αν έχει συμβεί υπερχείλιση;
- ζ'.** Ποια λογική συνάρτηση μεταξύ των C_{31} και C_{32} δίνει την υπερχείλιση; Γιατί να επιλεγεί να υλοποιηθεί ένας τέτοιος μηχανισμός ανίχνευσης παρά ένας συγκριτής των εισόδων με το άθροισμα;

**Δυαδική Αναπαράσταση**

+	00111111 11111111 11111111 11111111		
=	00111111 11111111 11111111 11111111		
31 24 23 16 15 8 7 0			

 V C_{32} C_{31} **Δεκαδική**

+1, 073, 741, 823
+1, 073, 741, 823
=

Δυαδική Αναπαράσταση

+	01111111 11111111 11111111 11111111		
=	00000000 00000000 00000000 00000001		
31 24 23 16 15 8 7 0			

 V C_{32} C_{31} **Δεκαδική**

+2, 147, 483, 647
+1
=

Δυαδική Αναπαράσταση

+	10000000 00000000 00000000 00000000		
=	11111111 11111111 11111111 11111111		
31 24 23 16 15 8 7 0			

 V C_{32} C_{31} **Δεκαδική**

-2, 147, 483, 648
-1
=

Δυαδική Αναπαράσταση

+	11111111 11111111 11111111 11111111		
=	11111111 11111111 11111111 11111111		
31 24 23 16 15 8 7 0			

 V C_{32} C_{31} **Δεκαδική**

-1
-1
=

Εικόνα 10: Παραδείγματα προσημασμένων αριθμών για ανίχνευση υπερχείλισης



Ε4 Έλεγχος Ροής Προγράμματος

Σε αυτή την εργαστηριακή άσκηση θα ασχοληθούμε με τη χρήση εντολών διακλαδώσεων υπό συνθήκη (conditional branches) και αλμάτων, προκειμένου να επιτευχθούν γνωστές δομές ελέγχου ροής των γλωσσών προγραμματισμού υψηλού επιπέδου, όπως διακλαδώσεις if–then–else, switch–case, και δομές επανάληψης (βρόχοι) for loop, while loop, until loop, κλπ.

Ε4.1 Ωφέλιμα bytes

Να γράψετε ένα πρόγραμμα σε συμβολική γλώσσα RISC-V το οποίο ο χρήστης θα πληκτρολογεί έναν προσημασμένο ακέραιο αριθμό και θα εμφανίζονται στην οθόνη τα παρακάτω μηνύματα, ανάλογα με τον αριθμό των bytes που απαιτούνται για την αναπαράσταση:

- «The number fits in one byte»
- «The number fits in two bytes»
- «The number fits in four bytes»

Θα υλοποιήσετε 2 παραλλαγές.

- α'.** Θα εμφανίζονται μηνύματα για όσες περιπτώσεις ισχύουν. Για παράδειγμα, αν δόθηκε ο ακέραιος 127 θα εμφανιστούν και τα τρία μηνύματα. Εδώ ουσιαστικά πρόκειται για τη δομή switch της γλώσσας C χωρίς τη χρήση break ανάμεσα στους κλάδους.
- β'.** Θα εμφανίζεται μήνυμα μόνο για μια περίπτωση. Για παράδειγμα, αν δόθηκε ο ακέραιος 128 θα εμφανιστεί μόνο το μήνυμα «The number fits in two bytes». Πρόκειται για τη δομή switch της γλώσσας C με χρήση break ανάμεσα στους κλάδους.

Ε4.2 Απόσταση Hamming

Η απόσταση Hamming (Hamming Distance) είναι ένα μέτρο που χρησιμοποιείται για τη σύγκριση δύο δυαδικών ακολουθιών ίδιου μήκους, υπολογίζοντας τον αριθμό των θέσεων στις οποίες τα αντίστοιχα bit τους διαφέρουν. Για παράδειγμα η απόσταση Hamming των ακολουθιών 11001010 και 11010010 είναι 2. Να γράψετε ένα πρόγραμμα σε συμβολική γλώσσα RISC-V το οποίο θα υπολογίζει την απόσταση Hamming δύο λέξεων των 32 bit που είναι αποθηκευμένες στη μνήμη.

Ε4.3 Μετατροπή πεζών γραμμάτων σε κεφαλαία

Γράψτε ένα πρόγραμμα που θα δέχεται σαν είσοδο από το χρήστη μία ακολουθία χαρακτήρων και θα τυπώνει στην έξοδο την ίδια ακολουθία με κεφαλαία γράμματα (όπου υπάρχουν χαρακτήρες 'a'-'z' θα αντικαθίστανται με τους 'A' - 'Z').



E5 Διαδικασίες και Στοίβα

Σε αυτή την εργαστηριακή άσκηση θα ασχοληθούμε με τη χρήση της στοίβας για τη δημιουργία διαδικασιών (procedures) και θα αναπτυχθούν δύο προσεγγίσεις για τον υπολογισμό της ακολουθίας Fibonacci.

E5.1 Πράξεις με συμβολοσειρές

Αρκετές γλώσσες προγραμματισμού, όπως η Java και η Python, παρέχουν τη συνάρτηση `split()` η οποία διαχωρίζει μια συμβολοσειρά βάσει ενός διαχωριστή (delimiter). Το πρόγραμμα **lab5_ex1.s** που υπάρχει στο eClass του μαθήματος υλοποιεί μια περίπτωση της `split()`. Συγκεκριμένα, διαβάζει μια συμβολοσειρά από το χρήστη και καλεί τη διαδικασία `find` που είναι υπεύθυνη για την εύρεση ενός χαρακτήρα μέσα σε μια συμβολοσειρά. Έπειτα, αντικαθιστά τα κοινά σημεία με τον χαρακτήρα `NULL` και εκτυπώνει το κάθε μέρος χωριστά.

Καλείστε να υλοποιήσετε τη διαδικασία `find`, η οποία θα δέχεται τρία ορίσματα: τη διεύθυνση μιας συμβολοσειράς `str1`, έναν χαρακτήρα `ch` και τη διεύθυνση ενός πίνακα `A`. Η διαδικασία θα εντοπίζει τις θέσεις στις οποίες εμφανίζεται ο χαρακτήρας `ch` μέσα στην `str1` και θα αποθηκεύει στον πίνακα `A` τους δείκτες των εμφανίσεων (ξεκινώντας από το 0) χρησιμοποιώντας **αποκλειστικά** αποθηκευμένους (`s`) καταχωρητές για ενδιάμεσες πράξεις. Επίσης, θα αποθηκεύει στον καταχωρητή `a1` την τιμή 0 αν δεν εντόπισε καμία θέση.

Τρέξτε το πρόγραμμα με βηματικό τρόπο για να παρακολουθείτε τη στοίβα να «μεγαλώνει» και να «μικραίνει», δηλαδή να προστίθενται σε αυτή και να αφαιρούνται από αυτή στοιχεία. Παρακολουθήστε τα περιεχόμενα του καταχωρητή δείκτη στοίβας `sp`.

E5.2 Ακολουθία Fibonacci

Είναι γνωστό ότι πολλοί μαθηματικοί ορισμοί βασίστηκαν σε στοιχεία της φύσης. Η ακολουθία Fibonacci, για παράδειγμα, βασίστηκε στον αριθμό από ζεύγη κουνελιών που θα παραχθούν από ένα αρχικό ζεύγος μέσα σε n μήνες. Επιπλέον, παρατηρείται σε σπειροειδή σχέδια διαφόρων φυτών, όπως ανανάδες, κουκουνάρια, κ.α.

Η μαθηματική σχέση που εκφράζει την ακολουθία Fibonacci περιγράφεται στην Εξίσωση 2.

$$f(n) = \begin{cases} n, & n \in \{0, 1\} \\ f(n - 1) + f(n - 2), & n > 1 \end{cases} \quad (2)$$

Παρατηρήστε ότι ο ορισμός γίνεται αναδρομικά: Ο n -οστός όρος υπολογίζεται ως το άθροισμα των δύο προηγούμενων όρων της ακολουθίας.

α'. Κατεβάστε το πρόγραμμα **lab5_ex2_fib_rec.s** που θα βρείτε στο eClass του μαθήματος. Περιλαμβάνει την κύρια μέθοδο, η οποία καλεί τη διαδικασία



`fib_rec` και αποθηκεύει τους πρώτους 40 όρους της ακολουθίας Fibonacci στη μνήμη. Καλείστε να γράψετε τη διαδικασία `fib_rec` η οποία υπολογίζει το n -οστό όρο της ακολουθίας Fibonacci χρησιμοποιώντας την Εξίσωση 2.

Για να ελέγξετε την ορθότητα του προγράμματος σας, αλλάξτε την κύρια διαδικασία ώστε να υπολογίζει μόνο τους 5 πρώτους όρους της ακολουθίας. Πιθανόν να χρειαστεί και βηματική εκτέλεση του κώδικα σας ώστε να ανακαλύψετε λογικά σφάλματα.

Αφού πειστείτε ότι το πρόγραμμα σας δουλεύει για τους 5 πρώτους όρους της ακολουθίας, επαναφέρετε την κύρια μέθοδο στην αρχική της κατάσταση, και δοκιμάστε να τρέξετε το πρόγραμμα σας για 40 όρους. Τι παρατηρείτε; Μπορείτε να το εξηγήσετε;

β'. Στη συνέχεια, κατεβάστε το πρόγραμμα `lab5_ex2_fib_memo.s`.

Σε αυτό το ερώτημα καλείστε να γράψετε μια πιο αποδοτική έκδοση έναντι της απλής αναδρομικής του προηγούμενο ερωτήματος χρησιμοποιώντας την τεχνική του memoization («απομνημόνευση»).

Το memoization δουλεύει ως εξής: Οι τιμές της ακολουθίας που ήδη υπολογίστηκαν αποθηκεύονται στην μνήμη, ώστε να αποφεύγεται ο επανυπολογισμός τους σε περίπτωση που χρειαστεί να χρησιμοποιηθούν ξανά.

Παρατηρήστε ότι η κύρια μέθοδος αποθηκεύει στην μνήμη τους δύο πρώτους όρους της ακολουθίας και στη συνέχεια καλεί τη μέθοδο `fib_memo` για να υπολογίσει τους όρους 2 έως και 40.

Συμπληρώστε τη μέθοδο `fib_memo` με τέτοιο τρόπο ώστε οι τιμές της ακολουθίας που ήδη υπάρχουν στον πίνακα να επιστρέφονται αμέσως, χωρίς περαιτέρω αναδρομικές κλήσεις.

γ'. Συμπληρώστε τον Πίνακα 3, καταγράφοντας πόσες εντολές εκτελούνται για να υπολογιστούν οι πρώτοι 5, 10 και 20 όροι της ακολουθίας με κάθε μία από τις δύο μεθόδους που υλοποιήσατε στα προηγούμενα ερωτήματα. Για να βλέπετε το πλήθος των εντολών που εκτελέστηκαν μπορείτε να ανοίξετε το παράθυρο με τα στατιστικά των εντολών πατώντας Tools→Instruction Statistics, και να επιλέξετε Connect to Program πριν εκτελέσετε τον κώδικα σας. Για να επαναφέρετε τα στατιστικά σε μηδενικά χρησιμοποιήστε το Reset.

Μέθοδος	5 όροι	10 όροι	20 όροι
----------------	---------------	----------------	----------------

<code>fib_rec</code>

<code>fib_memo</code>

Πίνακας 3: Πλήθος εντολών για την εκτέλεση



Ε6 Χειρισμός Κρατούμενου, Πολλαπλασιασμός

Σε αυτή την εργαστηριακή άσκηση μελετώνται οι εντολές αριθμητικών πράξεων ακεραίων αριθμών του επεξεργαστή RISC-V. Πιο συγκεκριμένα, θα εστιάσουμε στην εξαγωγή του τελευταίου κρατούμενου και θα το χρησιμοποιήσουμε για να υλοποιήσουμε την 64-bit πρόσθεση, ενώ θα μελετήσουμε και τον πολλαπλασιασμό μεταξύ ακέραιων αριθμών.

Ε6.1 Τελικό κρατούμενο

Το ζητούμενο αυτής της άσκησης είναι ένας τρόπος παραγωγής κρατούμενου από τον αθροιστή 32 bit του RISC-V επειδή αυτό δεν προσφέρεται από τον συγκεκριμένο μικροεπεξεργαστή. Υποθέτουμε ότι ο αθροιστής της ALU του RISC-V αποτελείται από 32 πλήρεις αθροιστές (full adders) του ενός bit συνδεδεμένους μεταξύ τους με τα κρατούμενα εισόδου και εξόδου. Ο πίνακας αλήθειας ενός πλήρους αθροιστή για το bit i παρουσιάζεται στον Πίνακα 4. Θεωρείστε ότι τα A_i και B_i είναι τα δύο bit i -τάξης που πρόκειται να προστεθούν, C_i είναι το κρατούμενο που δίνει ο αθροιστής που προσθέτει τα A_{i-1} και B_{i-1} , S_i είναι το άθροισμα ενώ C_{i+1} είναι το κρατούμενο που παράγει ο αθροιστής που προσθέτει τα A_i και B_i .

Είσοδοι			Έξοδοι	
C_i	A_i	B_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Πίνακας 4: Πίνακας αληθείας για το bit i ενός πλήρους αθροιστή

Για να υπολογίσουμε το C_{32} , χρειαζόμαστε τα A_{31}, B_{31}, C_{31} . Τα A_{31}, B_{31} μπορούμε να τα βρούμε πολύ εύκολα, κάτι που δεν ισχύει για το C_{31} . Θέτοντας $i = 30$, για να υπολογίσουμε το C_{31} , χρειαζόμαστε τα A_{30}, B_{30}, C_{30} . Αναδρομικά, φτάνουμε μέχρι και το bit 0 για το οποίο ισχύει $C_0 = 0$.

Υπάρχει τρόπος να εξαλείψουμε την αναδρομή απαιτώντας σαν είσοδο το άθροισμα S_i δεδομένου ότι για τον υπολογισμό του αθροίσματος λαμβάνονται υπόψιν και τα προηγούμενα κρατούμενα C_0, C_1, \dots, C_i . Συμπληρώστε τον Πίνακα 5 που εκφράζει τον πίνακα αληθείας του C_{i+1} όταν είναι γνωστά τα A_i, B_i και S_i (αντί για το C_i).

Κατόπιν, γράψτε ένα πρόγραμμα σε συμβολική γλώσσα RISC-V που να υπολογίζει το τελικό κρατούμενο μίας πρόσθεσης (C_{32}) μεταξύ δύο καταχωρητών και να το αποθηκεύει σε ένα bit κάποιου άλλου καταχωρητή. Οι αριθμοί που θα προστεθούν να εισάγονται στη μνήμη δεδομένων. Προκειμένου να διαπιστώσετε την



Είσοδοι			Έξοδος
S_i	A_i	B_i	C_{i+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Πίνακας 5: Πίνακας αληθείας για το κρατούμενο C_{i+1}

ορθότητα του προγράμματός σας, προτείνεται να χρησιμοποιήσετε τα ζευγάρια αριθμών της Άσκησης [E3.3](#).

E6.2 Πρόσθεση 64-bit αριθμών

Στην άσκηση αυτή θα δημιουργήσετε ένα πρόγραμμα που θα μπορεί να κάνει πρόσθεση δύο προσημασμένων αριθμών A και B των 64 bit ο καθένας. Αρχικά οι αριθμοί θα βρίσκονται στη μνήμη δεδομένων και η διάταξή τους θα ακολουθεί το μοντέλο little-endian, δηλαδή η λέξη με τα λιγότερο σημαντικά bit θα βρίσκεται σε χαμηλότερη διεύθυνση όπως φαίνεται και στο σχήμα παρακάτω. Το αποτέλεσμα της άθροισης S θα αποθηκεύεται και αυτό στη μνήμη όπως δείχνει η Εικόνα 11. Εκμεταλλευτείτε την προηγούμενη άσκηση ώστε το πρόγραμμά σας να χρησιμοποιεί το κρατούμενο C_{32} στην πρόσθεση των bit στη θέση 32 (δηλαδή να «μεταφερθεί» το κρατούμενο από τη λέξη που αποθηκεύει τα 32 χαμηλότερα bit στη λέξη που αποθηκεύει τα 32 υψηλότερα bit).

Data Segment	Byte and Word Address
:	
S (bits 63...32)	0x10010014
S (bits 31...0)	0x10010010
B (bits 63...32)	0x1001000C
B (bits 31...0)	0x10010008
A (bits 63...32)	0x10010004
A (bits 31...0)	0x10010000

Εικόνα 11: Τμήμα δεδομένων της Άσκησης [E6.2](#)

Πειραματιστείτε με τα ζευγάρια δυαδικών αριθμών της Εικόνας 12 και αφού συμπληρώσετε τα αθροίσματα συγκρίνετε τα με τα αποτελέσματα που δίνει το πρόγραμμά σας.

**Δυαδική Αναπαράσταση**

	01111111 11111111 11111111 11111111	11111111 11111111 11111111 11111111	A
+	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000001	B
=			S
	63 54 53 48 47 40 39 32 31	24 23 16 15 8 7 0	

Δυαδική Αναπαράσταση

	00000000 00000000 00000000 00000000	11111111 11111111 11111111 11111111	A
+	11111111 11111111 11111111 11111111	00000000 00000000 00000000 00000000	B
=			S
	63 54 53 48 47 40 39 32 31	24 23 16 15 8 7 0	

Εικόνα 12: Παραδείγματα πρόσθεσης 64-bit αριθμών

E6.3 Πολλαπλασιασμός

Σε αυτή την άσκηση θα εξετάσουμε τη διαφοροποίηση των τεσσάρων πραγματικών εντολών πολλαπλασιασμού ακεραίων `mul` (multiply), `mulh` (multiply high), `mulhu` (multiply high unsigned) και `mulhsu` (multiply high signed/unsigned) που διαθέτει ο επεξεργαστής RISC-V. Η διαφορά τους βρίσκεται στο διαφορετικό τρόπο με τον οποίο μεταφράζουν τους τελεστέους της πράξης του πολλαπλασιασμού αλλά και η διαφορά στο αποτέλεσμα που δίνουν οι εντολές.

- α'.** Να γράψετε ένα πρόγραμμα σε συμβολική γλώσσα RISC-V το οποίο αρχικά θα δεσμεύει στη μνήμη δεδομένων 2 δεκαεξαδικές τιμές μεγέθους λέξης 32 bit η καθεμία, χρησιμοποιώντας την οδηγία `.word` (έστω ότι τα ονόματα των δύο αυτών τιμών είναι x και y). Επίσης, θα δεσμεύει χώρο για τέσσερις ακόμα λέξεις.
- Υπόδειξη:** Η δέσμευση του χώρου μπορεί να γίνει απλά, δεσμεύοντας τέσσερις λέξεις με την οδηγία `.word` και ως αρχική τιμή το 0.
- β'.** Στη συνέχεια, φορτώστε τις δύο δεκαεξαδικές τιμές που ορίσατε σε δύο καταχωρητές. Έπειτα, χρησιμοποιήστε τις εντολές `mul`, `mulh`, `mulhu`, και `mulhsu` για να πολλαπλασιάσετε τους αριθμούς που φορτώσατε στους καταχωρητές και αποθηκεύστε τα αποτελέσματα στις τέσσερις διαδοχικές θέσεις μνήμης που δεσμεύσατε στο υποερώτημα **α'**.
- γ'.** Πειραματιστείτε με τα ζευγάρια αριθμών της Εικόνας 13. Φορτώστε και εκτελέστε το πρόγραμμά σας στον RARS και εκτελέστε το πρόγραμμα ξεχωριστά για κάθε ζευγάρι τιμών και συμπληρώστε τα αποτελέσματα της κάθε εντολής.
- δ'.** Εξηγείστε τη λειτουργία των εντολών `mulh`, `mulhu`, και `mulhsu` (στη γενική τους μορφή, ανεξάρτητα από τα αποτελέσματα που πήρατε).
- ε'.** Παρατηρείστε ότι στις περιπτώσεις των αριθμών **13β'** και **13γ'** τα αποτελέσματα των εντολών `mulh`, `mulhu`, και `mulhsu` δεν είναι πάντα ίδια. Για κάθε ένα από τα παραπάνω 3 ζευγάρια αριθμών, εξηγείστε πότε τα αποτελέσματα των εντολών `mulh`, `mulhu`, και `mulhsu` που πήρατε από την εκτέλεση του προγράμματος θα ήταν σωστά για κάθε μια από τις παρακάτω συνθήκες.



- Τα ζευγάρια αριθμών που πολλαπλασιάζονται είναι πάντα προσημασμένοι αριθμοί
 - Τα ζευγάρια αριθμών που πολλαπλασιάζονται είναι πάντα μη προσημασμένοι αριθμοί
 - Στα ζευγάρια αριθμών που πολλαπλασιάζονται ο πολλαπλασιαστέος είναι προσημασμένος ενώ ο πολλαπλασιαστής είναι μη προσημασμένος

Υπόδειξη: Θα διευκολύνετε τη σκέψη σας αν υπολογίσετε τα αποτελέσματα των αριθμών στο δεκαδικό σύστημα αρίθμησης και τα συγκρίνετε με τα αποτελέσματα του RARS επίσης στο δεκαδικό σύστημα αρίθμησης, για κάθε μια από τις τρεις παραπάνω συνθήκες. Θυμηθείτε ότι το αποτέλεσμα των εντολών `mulh`, `mulhu`, και `mulhsu` αφορά τα bit 32 έως 64 (δηλαδή, τα 32 περισσότερο σημαντικά bit του αποτελέσματος).

00000000	00000000	00000000	00001010	.
00000000	00000000	00000000	00001000	
				mul
				mulh
				mulhu
				mulhsu
31	24 23	16 15	8 7	0
(α') 0x00000000a · 0x00000008				
00000000	00000000	00000000	00000010	.
11111111	11111111	11111111	11111111	
				mul
				mulh
				mulhu
				mulhsu
31	24 23	16 15	8 7	0
(β') 0x00000002 · 0xffffffff				
10000000	00000000	00000000	00000000	.
11000000	00000000	00000000	00000000	
				mul
				mulh
				mulhu
				mulhsu
31	24 23	16 15	8 7	0
(γ') 0x80000000 · 0xc0000000				

Εικόνα 13: Παραδείγματα πολλαπλασιασμού



E7 Αριθμητική Κινητής Υποδιαστολής

Σε αυτή την εργαστηριακή άσκηση θα μελετήσουμε τους αριθμούς κινητής υποδιαστολής απλής ακρίβειας. Αναλυτικότερα, θα εστιάσουμε σε βασικές πράξεις μεταξύ των αριθμών κινητής υποδιαστολής, στον τρόπο αναπαράστασης και στους περιορισμούς που υπάρχουν ως προς την ακρίβεια. Επιπλέον, θα φανεί η ανάγκη τόσο επιλογής τους έναντι ακεραίων αριθμών αλλά και της σημασίας των επεκτάσεων της αρχιτεκτονικής RISC-V.

E7.1 Διάβασμα και εκτύπωση ενός αριθμού κινητής υποδιαστολής

Γράψτε ένα πρόγραμμα σε συμβολική γλώσσα RISC-V που θα διαβάζει από το παράθυρο της κονσόλας έναν αριθμό απλής ακρίβειας με τη χρήση της κλήσης συστήματος *ReadFloat*. Κατόπιν χρησιμοποιήστε την κλήση *PrintFloat* για να τον εμφανίσετε στην οθόνη. Για περισσότερες πληροφορίες σχετικές με τις κλήσεις συστήματος μπορείτε να ανατρέξετε στο Παράρτημα [B'](#).

- α'. Πειραματιστείτε με αριθμούς που είναι δυνάμεις ή αθροίσματα δυνάμεων του 2, π.χ. -0.75 , 1.0 , 1.5 , κ.α.
- β'. Πειραματιστείτε με τυχαίους αριθμούς. Τι παρατηρείτε;

E7.2 Πράξεις με αριθμούς κινητής υποδιαστολής

Γράψτε ένα πρόγραμμα σε συμβολική γλώσσα RISC-V που να ορίζει με τη χρήση του τμήματος δεδομένων έξι σταθερές : 0.0 , $+\infty$, $-\infty$, $+NaN$, και δύο αριθμούς κινητής υποδιαστολής, έναν θετικό x και έναν αρνητικό y . Ενδεχομένως να σας ενδιαφέρει η υποδομή αναπαράστασης 32-bit αριθμών κινητής υποδιαστολής με βάση το πρότυπο IEEE 754 που διαθέτει ο RARS (Tools → Floating Point Representation) και το Παράρτημα [A'5.3.3](#). Το πρόγραμμα θα πρέπει να εκτελεί τις πράξεις που φαίνονται στον Πίνακα 6 και να εμφανίζει τα αποτελέσματά τους. Συμπληρώστε τον Πίνακα 6 σύμφωνα με τα αποτελέσματα που πήρατε.

Πράξη	Αποτέλεσμα
$x - y$	
$x \cdot y$	
$y / 0$	
$0 / 0$	
$(+\infty) / (-\infty)$	
$+\infty + (-\infty)$	
$x + (+NaN)$	

Πίνακας 6: Παραδείγματα πράξεων με αριθμούς κινητής υποδιαστολής



E7.3 Υπολογισμός του παραγοντικού

- α'.** Κατεβάστε από το eClass του μαθήματος το πρόγραμμα **lab7_ex3.s** και αποθηκεύστε το τοπικά στον υπολογιστή σας. Στη συνέχεια ανοίξτε το και μελετήστε το περιεχόμενό του. Περιλαμβάνει τον υπολογισμό του παραγοντικού με επαναληπτικό τρόπο. Φορτώστε το στον προσομοιωτή RARS και τρέξτε το. Τι παρατηρείτε;
- β'.** Άλλάξτε το προηγούμενο πρόγραμμα έτσι ώστε ο υπολογισμός να γίνεται με αριθμητική κινητής υποδιαστολής απλής ακρίβειας. Τρέξτε το πρόγραμμα για $n > 20$ και παρατηρήστε τα αποτελέσματα.

E7.4 Υλοποίηση εντολής κινητής υποδιαστολής

Η αρχιτεκτονική RISC-V διακρίνεται για τον αρθρωτό της σχεδιασμό (modular design), ο οποίος βασίζεται σε μια κύρια αρχιτεκτονική (base ISA), πάνω στην οποία μπορούν να προστεθούν διάφορες επεκτάσεις (ISA extensions).

Ο ρόλος των επεκτάσεων είναι να μειώνουν τον αριθμό των εντολών που απαιτούνται για την εκτέλεση σύνθετων λειτουργιών, επιταχύνοντας έτσι την εκτέλεση των προγραμμάτων. Για παράδειγμα, η επέκταση κινητής υποδιαστολής απλής ακρίβειας RV32F της αρχιτεκτονικής RISC-V παρέχει εξειδικευμένες εντολές που μπορούν να αντικαταστήσουν πολλές ξεχωριστές εντολές σύγκρισης, διακλαδώσεων και αριθμητικών πράξεων. Όταν αυτές οι εντολές υλοποιούνται σε υλικό, μειώνεται τόσο ο χρόνος εκτέλεσης όσο και η κατανάλωση ενέργειας.

Σε αυτή την άσκηση θα δούμε τον τόπο με τον οποίο μια εντολή της επέκτασης RV32F (KY απλής ακρίβειας του RISC-V) θα έπρεπε να υλοποιηθεί με συνδυασμό άλλων εντολών αν δεν υπήρχε ως μέρος της επέκτασης RV32F. Συγκεκριμένα, το ζητούμενο της άσκησης είναι να υλοποιήσετε ένα πρόγραμμα σε συμβολική γλώσσα RISC-V το οποίο θα εξομοιώνει τη λειτουργία της εντολής `f1t.s` που ανήκει στην επέκταση RV32F.

Κατεβάστε από το eClass του μαθήματος το πρόγραμμα **lab7_ex4.s**. Στη συνέχεια ανοίξτε το και μελετήστε το περιεχόμενό του. Αυτό το πρόγραμμα φορτώνει απλώς 2 αριθμούς σε αναπαράσταση IEEE 754 από τη μνήμη δεδομένων και εκτελεί την εντολή `f1t.s`, η οποία συγκρίνει τους αριθμούς κινητής υποδιαστολής που βρίσκονται στους `ft1` και `ft2` και θέτει την τιμή 1 στον (ακέραιο) καταχωρητή `t3` αν ισχύει $ft1 < ft2$, αλλιώς θέτει την τιμή 0 στον `t3`.

Το πρόγραμμα που θα υλοποιήσετε θα πρέπει να εξομοιώνει αυτή τη λειτουργία αποκλειστικά με ακέραιες εντολές και καταχωρητές (εντολές δηλαδή που ανήκουν στην κύρια αρχιτεκτονική RV32I). Συγκεκριμένα, θέλουμε το πρόγραμμα να φορτώνει δύο αριθμούς σε αναπαράσταση IEEE 754 από τη μνήμη δεδομένων και να ελέγχει αν ο πρώτος αριθμός είναι μικρότερος από το δεύτερο. Αν ναι, τότε θα θέτει την τιμή 1 στον καταχωρητή `t4`, αλλιώς θα θέτει την τιμή 0 στον `t4`. Την υλοποίηση του προγράμματος μπορείτε να την κάνετε στο αρχείο που μόλις κατεβάσατε ώστε να επαληθεύσετε τη λειτουργία του συγκρίνοντας το αποτέλεσμα του `t4` (που προέκυψε από την εκτέλεση του δικού σας προγράμματος) με τον `t3` (που προέκυψε από την εκτέλεση της `f1t.s`). Πόσες εντολές χρησιμοποιήσατε για να εξομοιώσετε τη λειτουργία της `f1t.s`;

Μέρος 2

Ο προσομοιωτής QtRVsim

Εκτός από τη γνώση των εντολών συμβολικής γλώσσας του RISC-V βασικός στόχος του εργαστηρίου αποτελεί η εξοικείωσή σας σχετικά με τον τρόπο εκτέλεσης των εντολών στο υλικό ενός RISC-V επεξεργαστή. Ο QtRVsim προσομοιώνει έναν RISC-V επεξεργαστή ενός κύκλου (*single cycle*) ή με διοχέτευση (*pipeline*), και αναπαριστά σε κάθε κύκλο τη λειτουργικότητα από διάφορα τμήματα του hardware που απαιτούνται για την εκτέλεση ενός προγράμματος, όπως για παράδειγμα τα διάφορα σήματα ελέγχου που έχουμε συζητήσει, τα περιεχόμενα των καταχωρητών και των μνημών, κ.ά.

Ο προσομοιωτής QtRVsim υποστηρίζει ένα υποσύνολο των βασικών ακέραιων εντολών RISC-V, ενώ δεν υποστηρίζει αριθμούς κινητής υποδιαστολής. Επίσης, παρέχει ένα μικρό σύνολο από κλήσεις συστήματος.



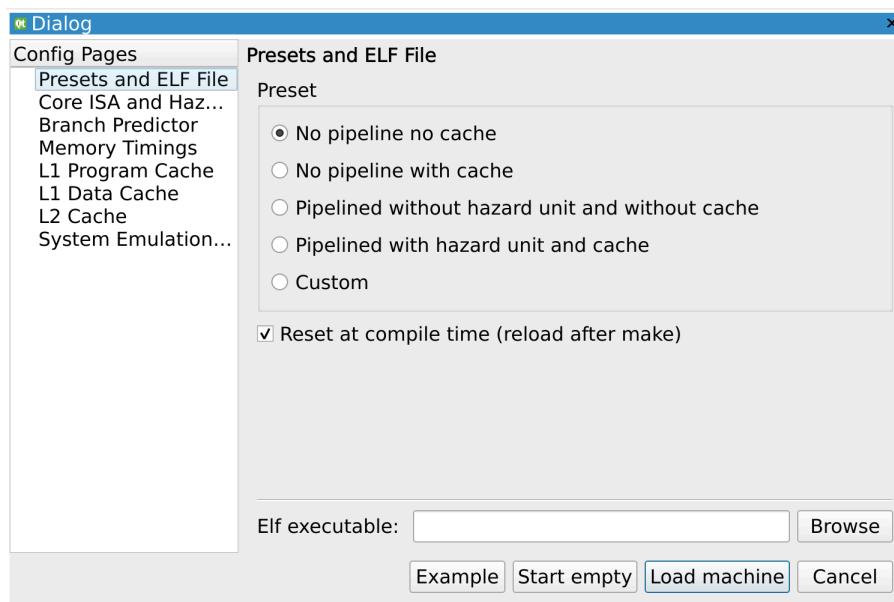
E8 Εισαγωγή στον προσομοιωτή QtRVsim

E8.1 Τα βασικά του QtRVsim

Υπάρχει διαθέσιμη online έκδοση του QtRVsim **εδώ**. Πατώντας το σύνδεσμο, θα εμφανιστεί ένα παράθυρο στο κέντρο της οθόνης σας παρόμοιο με αυτό της Εικόνας 14.

Στην κατηγορία **Presets and ELF File** παρατηρούμε διάφορες διαμορφώσεις που προσφέρει ο προσομοιωτής. Κάθε διαμόρφωση αντιστοιχεί σε ένα μοντέλο επεξεργαστή RISC-V. Για παράδειγμα, μία απλή διαμόρφωση χωρίς διοχέτευση και κρυφές μνήμες (No pipeline no cache, δηλαδή έναν απλό επεξεργαστή ενός κύκλου ρολογιού) μέχρι και διαμόρφωση με διοχέτευση, μνήμες και πρόβλεψη διακλάδωσης. Επιπλέον, επειδή τα προγράμματα που θα δοκιμαστούν στον προσομοιωτή θα είναι γραμμένα από εσάς, δεν υπάρχει ανάγκη για elf εκτελέσιμο.

Οι υπόλοιπες κατηγορίες διαμόρφωσης αφορούν τις επεκτάσεις του συνόλου εντολών RISC-V που υποστηρίζει ο προσομοιωτής, καθώς και διάφορα στοιχεία που αφορούν την εκτέλεση με διοχέτευση (**Hazards, Branch Predictor**) και με μνήμες (**Memory Timings, L1 Program & Data Cache, L2 Cache**).



Εικόνα 14: Παράθυρο διαμόρφωσης του προσομοιωτή QtRVsim

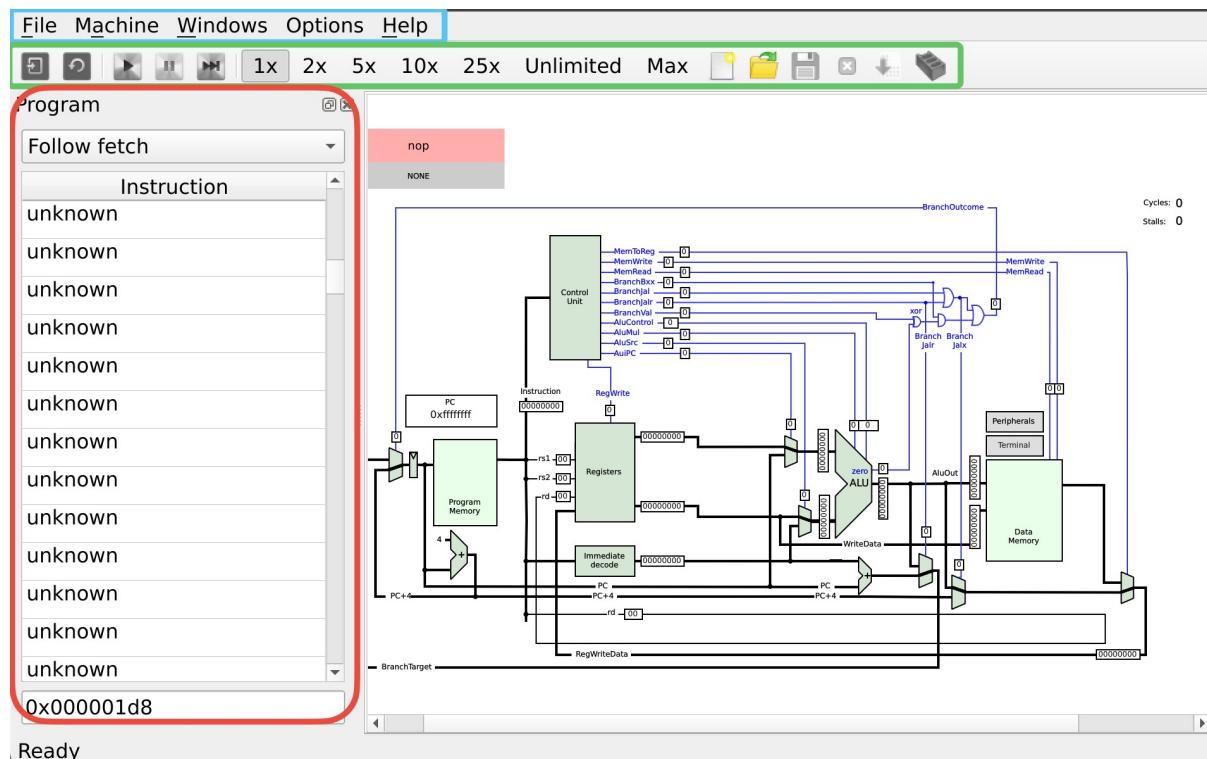
Πατώντας **Start empty** φορτώνονται οι επιλογές μας σχετικά με τον RISC-V επεξεργαστή που θέλουμε να προσομοιώσουμε. Το γραφικό περιβάλλον εκτέλεσης του προσομοιωτή παρουσιάζεται στην Εικόνα 15. Στο κέντρο παρατηρούμε τη διαδρομή δεδομένων ενός κύκλου ρολογιού καθώς και τα δομικά στοιχεία που την απαρτίζουν (λ.χ. οι μνήμες, το αρχείο καταχωρητών κ.α.). Με έντονο μαύρο χρώμα ή με μπλε περιγράφονται τα διάφορα σήματα ενώ στο **κόκκινο** πλαίσιο υπάρχει ένα παράθυρο (**Program**) στο οποίο εμφανίζονται οι εντολές που θα εκτελεστούν. Πάνω, στο **πράσινο** πλαίσιο, υπάρχουν εικονίδια σχετικά με την προσομοίωση. Αναλυτικότερα:



- : Νέα διαμόρφωση (επανεμφάνιση του παραθύρου της Εικόνας 14)
- : Επανεκκίνηση της προσομοίωσης. Η διαμόρφωση παραμένει ίδια
- : Εκκίνηση προσομοίωσης
- : Παύση προσομοίωσης
- : Εκτέλεση επόμενης εντολής (Single Step)
- 1x - Max** : Ταχύτητα εκτέλεσης προσομοίωσης
- : Νέο αρχείο πηγαίου κώδικα
- : Φόρτωση αρχείου πηγαίου κώδικα
- : Αποθήκευση αρχείου πηγαίου κώδικα
- : Κλείσιμο αρχείου πηγαίου κώδικα
- : Μεταγλώττιση αρχείου πηγαίου κώδικα
- : Χτίσιμο εκτελέσιμου. Χρησιμοποιείται μόνο αν έχουμε elf εκτελέσιμο

Επιπλέον, στο **γαλάζιο** πλαίσιο, υπάρχουν επιλογές του κεντρικού μενού. Αναλυτικότερα:

- **File:** Περιέχει όλες τις επιλογές σχετικά με την προσομοίωση και τα αρχεία πηγαίου κώδικα. Τα περισσότερα εικονίδια θα τα βρείτε σε αυτή την κατηγορία
- **Windows:** Περιέχει όλα τα παράθυρα με βασικά στοιχεία του hardware. Τα σημαντικότερα παράθυρα είναι εκείνα που περιέχουν πληροφορίες σχετικά με τους Registers, το Program και τη Memory
- **Options:** Δίνει τη δυνατότητα εμφάνισης των αριθμών των γραμμών στα αρχεία πηγαίου κώδικα
- **Help:** Παρέχει πληροφορίες σχετικά με τον προσομοιωτή



Εικόνα 15: Γραφικό περιβάλλον του προσομοιωτή QtRVsim



E8.2 Προσομοίωση επεξεργαστή ενός κύκλου

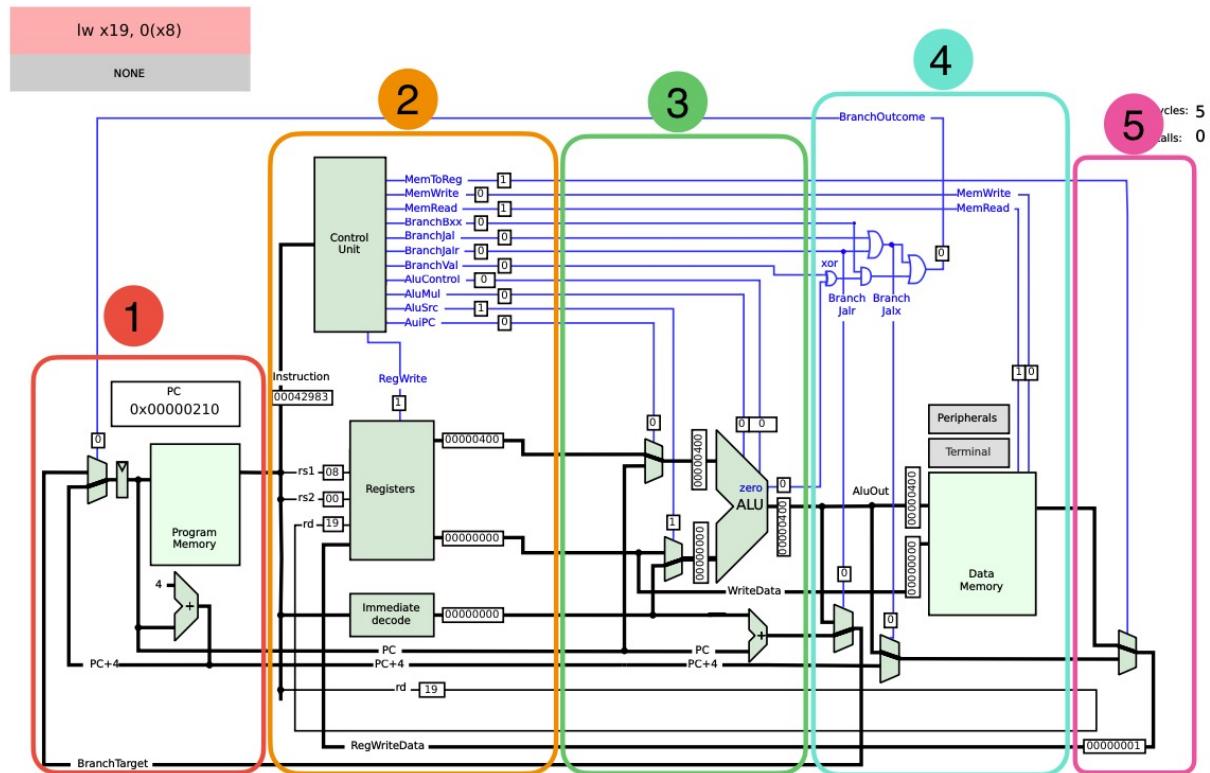
Κατεβάστε το αρχείο **Lab8.s** από το eClass του μαθήματος. Ακολουθήστε προσεκτικά τα παρακάτω βήματα.

- α'. Ανοίξτε τον προσομοιωτή QtRVsim.
- β'. Επιλέξτε στο παράθυρο της Εικόνας 14 την επιλογή «No pipeline no cache». Πατήστε Start empty.
- γ'. Φορτώστε τον κώδικα πατώντας το εικονίδιο . Έπειτα, μεταγλωττίστε τον πατώντας το εικονίδιο .

Πραγματοποιήστε βηματική εκτέλεση () μέχρι και την εντολή `lw x19, 0(x8)`. Συμπληρώστε την παρακάτω άσκηση βάσει της Εικόνας 16 και της περιγραφής που ακολουθεί.

- α'. Το σήμα `rs1` έχει τιμή _____, ενώ το `rd` _____.
- β'. Οι δύο τιμές που βρίσκονται στην έξοδο του αρχείου καταχωρητών είναι ίσες με _____ και _____, οι οποίες οφείλονται στα περιεχόμενα των καταχωρητών _____ και _____.
- γ'. Το σήμα ελέγχου `AluPC` ισούται με _____ δίνοντας στην έξοδο του πρώτου (μετρώντας από πάνω προς τα κάτω) πολυπλέκτη στην είσοδο της ALU την τιμή _____ αντί της τιμής _____ που είναι η τιμή του _____.
- δ'. Η πράξη που κάνει η ALU είναι _____. Για να δώσει 1 το σήμα `zero` στην έξοδο της ALU θα έπρεπε η δεύτερη (μετρώντας από πάνω προς τα κάτω) είσοδος της να είχε τιμή _____. Στην περίπτωση αυτή, το σήμα `BranchOutcome` θα είχε τιμή _____ αφού _____.
- ε'. Η τιμή που γράφεται στον καταχωρητή προορισμού _____ είναι _____ και προκύπτει από τη _____. Αυτό ρυθμίζεται από το σήμα _____ που έχει τιμή _____. Στην περίπτωση που είχε αντίθετη τιμή, θα γραφόταν η τιμή _____.

Η εκτέλεση μιας εντολής σε έναν επεξεργαστή ενός κύκλου μπορεί να χωριστεί σε 5 διακριτά μέρη όπως φαίνεται στην Εικόνα 16. Το μέρος 1 περιλαμβάνει το διάβασμα του μετρητή προγράμματος (PC) που περιέχει τη διεύθυνση που χρησιμοποιείται για την πρόσβαση στη μνήμη εντολών του επεξεργαστή (Fetch). Στο επόμενο μέρος 2 (Decode), η εντολή αποκωδικοποιείται και διαβάζονται οι καταχωρητές προέλευσης. Η δεκαεξαδική της μορφή βρίσκεται στο σήμα `Instruction`, το οποίο οδηγείται στην Μονάδα Ελέγχου (Control Unit) και διαχωρίζεται στα σήματα ελέγχου μπροστά από το αρχείο καταχωρητών (Registers) και στην αποκωδικοποίηση του immediate πεδίου. Ανοίγοντας το παράθυρο Registers (Windows → Registers) βλέπετε τις τιμές των καταχωρητών. Σειρά έχει το μέρος 3 και η εκτέλεση της εντολής (Execute), στην οποία μετέχει η ALU. Στην είσοδο της λαμβάνει το περιεχόμενο του καταχωρητή `rs1` και του immediate πεδίου για να υπολογίσει τη διεύθυνση, στην οποία γίνεται πρόσβαση για ανάγνωση δεδομένων στο επόμενο μέρος 4 Mem. Τέλος, τα δεδομένα γράφονται στον καταχωρητή προορισμού `rd` στο τελευταίο μέρος 5 WB.



Εικόνα 16: Εκτέλεση της μιας εντολής `lw` σε επεξεργαστή ενός κύκλου

Εκτελέστε το υπόλοιπο πρόγραμμα και επαληθεύστε τα αποτελέσματα τόσο στους καταχωρητές όσο στη μνήμη (Windows → Memory και βάζετε τη διεύθυνση που υποδεικνύει η εντολή `.org` πριν το `.data`). Πόσες εντολές και πόσοι κύκλοι εκτελέστηκαν; Τι παρατηρείτε;



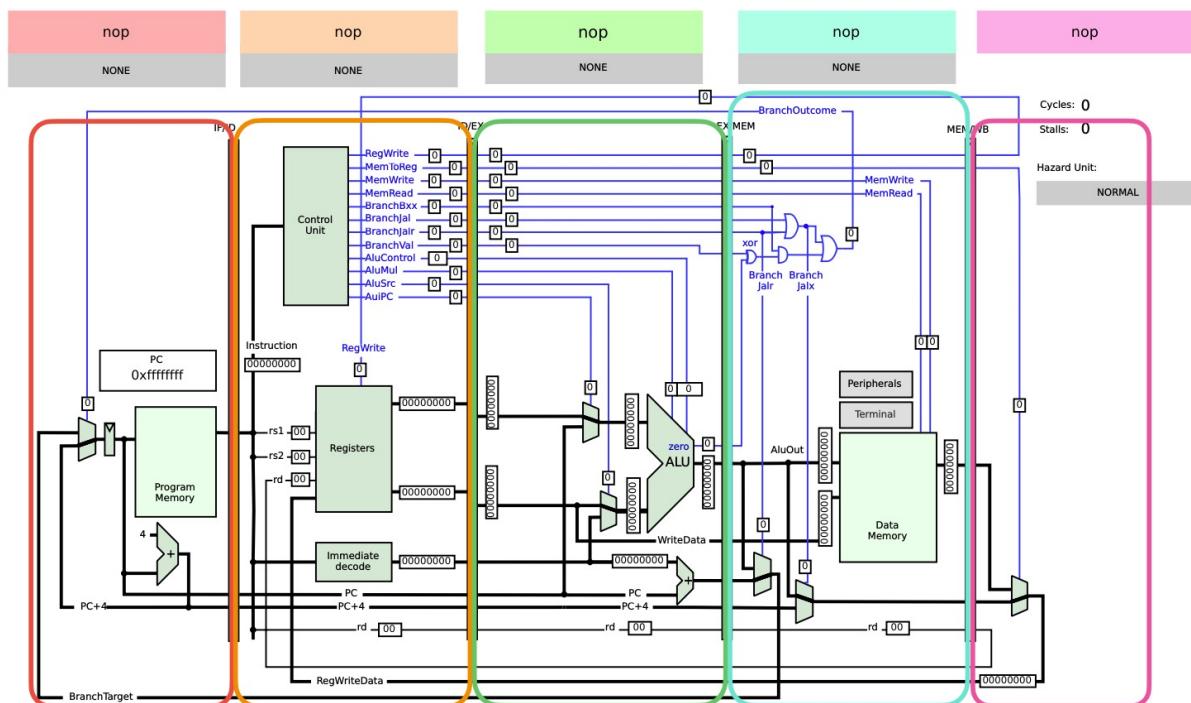
Ε9 Προσομοίωση επεξεργαστή με διοχέτευση

E9.1 Διοχέτευση με μονάδα ελέγχου κινδύνων

Κατεβάστε το αρχείο **lab9.s** από το eClass του μαθήματος. Ακολουθήστε προσεκτικά τα παρακάτω βήματα.

- α'. Ανοίξτε τον προσομοιωτή QtRVsim.
- β'. Επιλέξτε στο παράθυρο της Εικόνας 14 την επιλογή «Custom».
- γ'. Επιλέξτε την κατηγορία «Core ISA and Hazards» και ενεργοποιήστε το «Pipelined» και το «Hazard Unit». Έπειτα, ενεργοποιήστε την επιλογή «Stall when hazard is detected». Πατήστε Start empty.
- δ'. Φορτώστε τον κώδικα και μεταγλωττίστε τον.

Παρατηρήστε της Εικόνας 17. Για διευκόλυνση σας, έχουν σχεδιαστεί τα 5 στάδια της διοχέτευσης, **Fetch**, **Decode**, **Execute**, **Mem** και **WB**. Τώρα, σε έναν κύκλο ρολογιού εκτελούνται 5 εντολές, μία σε κάθε στάδιο.



Εικόνα 17: Αναπαράσταση ενός επεξεργαστή με διοχέτευση στον QtRVsim

E9.1.1 Κίνδυνοι Δεδομένων

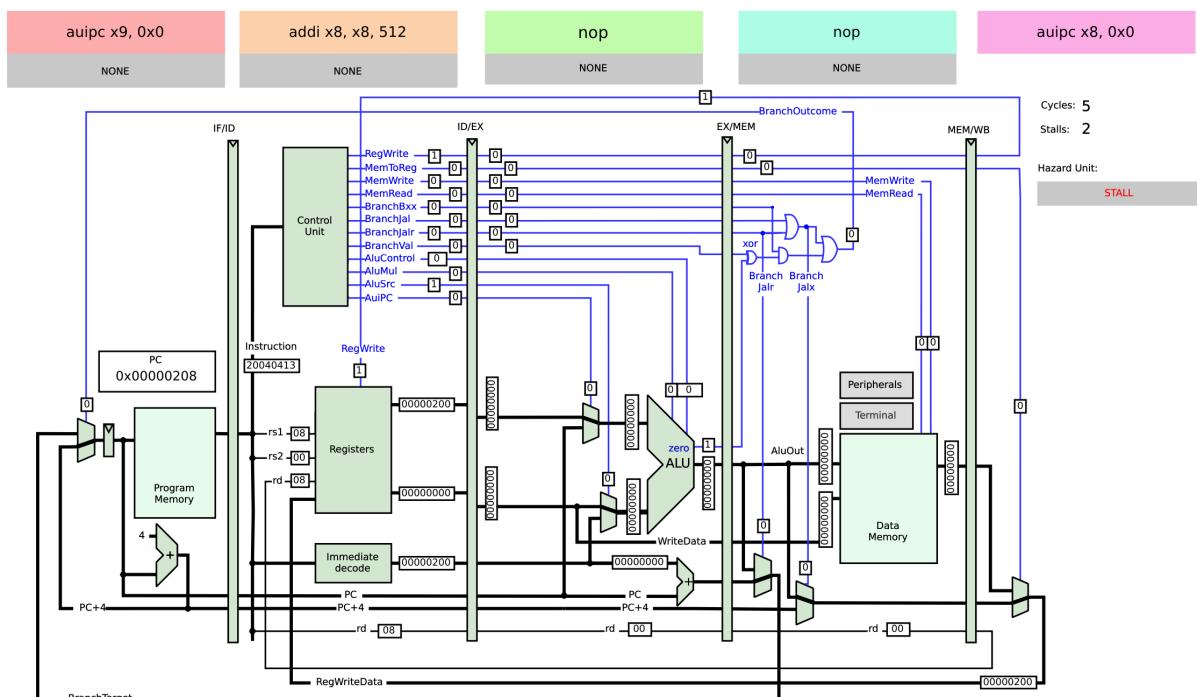
Εκτελέστε βηματικά το πρόγραμμα μέχρι και τον 5^ο κύκλο, δηλαδή μέχρι και την ολοκλήρωση της πρώτης εντολής auipc x8, 0x0.

Σε περίπτωση που θέλετε να τρέξετε από την αρχή το πρόγραμμα, αρκεί μόνο η μεταγλώττιση του κώδικα (→).

Παρατηρήστε ότι ενώ η ALU είναι ελεύθερη στον κύκλο 4 (η εντολή auipc x8,



0x0 βρίσκεται στο στάδιο **Mem**) δεν εκτελείται η επόμενη εντολή addi x8, x8, 512 αλλά εισάγεται μια εντολή nop (φυσαλίδα, ισοδύναμη με την εντολή addi x0, x0, 0x0). Αυτό οφείλεται στο γεγονός ότι η μονάδα ελέγχου κινδύνων έχει εντοπίσει την **εξάρτηση** των δύο εντολών (ο καταχωρητής προορισμού της πρώτης συμμετέχει και ως καταχωρητής προέλευσης της δεύτερης) με αποτέλεσμα να εισάγεται το ισοδύναμο μιας εντολής nop μέχρι η εντολή auipc x8, 0x0 να ολοκληρωθεί και να έχει γραφτεί στον καταχωρητή x8 η νέα τιμή με σκοπό να χρησιμοποιηθεί στην εντολή addi x8, x8, 512 που τον χρειάζεται. Επομένως, η εντολή addi x8, x8, 512 αντί να εκτελεστεί στον κύκλο 4, εκτελείται στον κύκλο 6, δηλαδή με **2 κύκλους καθυστέρησης** (Εικόνα 18). Αυτή η καθυστέρηση προστίθεται κάθε φορά που υπάρχουν **κίνδυνοι δεδομένων** (*Data Hazards*) και το υλικό του επεξεργαστή δεν διαθέτει δυνατότητα προώθησης (forwarding).



Εικόνα 18: Εντοπισμός ενός κινδύνου δεδομένων και καθυστέρηση

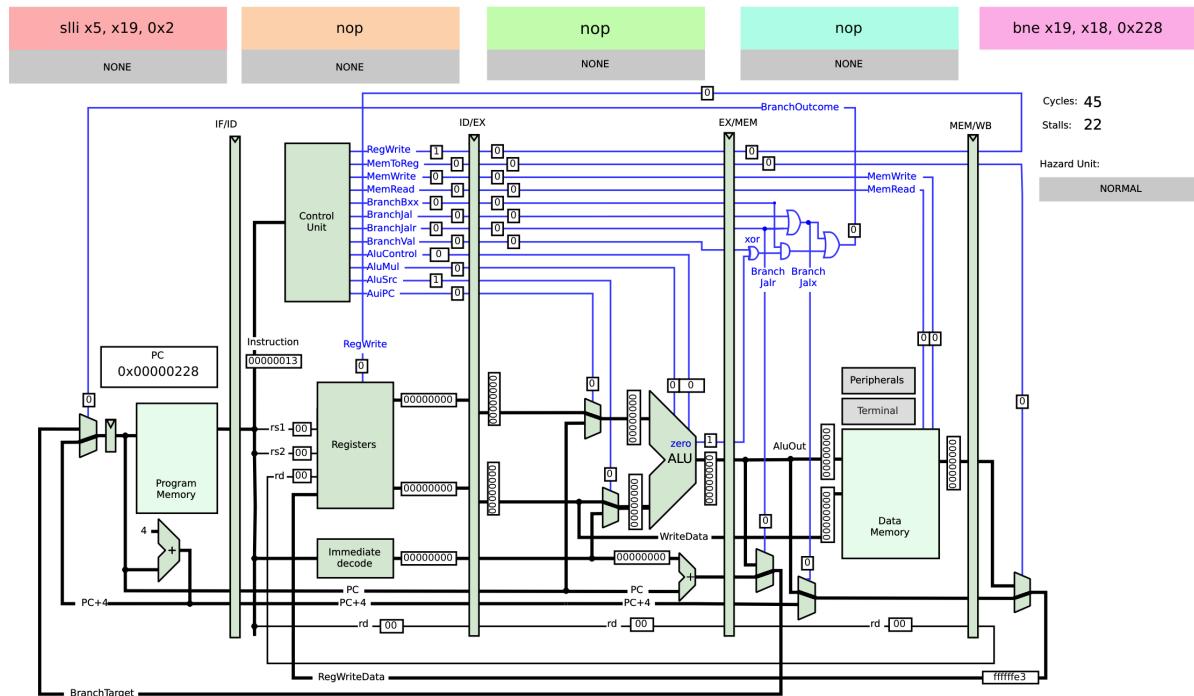
E9.1.2 Κίνδυνοι Ελέγχου

Μεγεθύνετε το παράθυρο Program στα αριστερά σας και θέστε ένα Breakpoint στην εντολή `lwne x19, x18, 0x228` κάνοντας διπλό click στο πλαίσιο BP. Πατώντας τώρα εκκίνηση προσομοίωσης (▶), θα υπάρχει παύση κατά την προσκόμιση (**Fetch**) αυτής της εντολής. Εκτελέστε τη βηματικά μέχρι την ολοκλήρωσή της, δηλαδή μέχρι και το στάδιο **WB**. Παρατηρήστε ότι προσκομίζονται οι επόμενες εντολές και αντικαθίστανται με nop όταν η εντολή διακλάδωσης φτάσει στο στάδιο **Mem**. Παρόλο που η έκβαση της συνθήκης υπολογίζεται στο τέλος του σταδίου **Execute**, χρησιμοποιείται σε κύκλωμα που βρίσκεται στο στάδιο **Mem** (πάνω από τη μνήμη δεδομένων) με σκοπό να αποφασιστεί αν θα τεθεί στον μετρητή προγράμματος η διεύθυνση προορισμού (λαμβανόμενη - taken - διακλάδωση) ή η διεύθυνση της επόμενης εντολής (μη λαμβανόμενη - not taken - διακλάδωση).



Επομένως, για επεξεργαστή που χρησιμοποιεί την ALU για τον υπολογισμό της συνθήκης (όπως συμβαίνει στο μοντέλο του QtRVsim), εισάγονται **3 κύκλοι καθυστέρησης** (Εικόνα 19).

ΠΡΟΣΟΧΗ. Αυτοί οι κύκλοι δεν υπολογίζονται σαν stalls. Προστίθενται κάθε φορά που υπάρχουν **κίνδυνοι ελέγχου** (*Control Hazards*).



Εικόνα 19: Εντοπισμός ενός κινδύνου ελέγχου και καθυστέρηση

Τρέξτε το υπόλοιπο πρόγραμμα και επαληθεύστε τα αποτελέσματα. Τι παρατηρείτε;

E9.2 Διοχέτευση χωρίς μονάδα ελέγχου κινδύνων

Θα δοκιμάσουμε το ίδιο πρόγραμμα αλλά σε έναν διαφορετικό επεξεργαστή που υποστηρίζει διοχέτευση αλλά δεν διαθέτει μονάδα ελέγχου κινδύνων.

- α'. Δημιουργήστε νέα προσομοίωση πατώντας το
- β'. Στην κατηγορία «Presets and ELF File» της Εικόνας 14 επιλέξτε «Custom».
- γ'. Επιλέξτε την κατηγορία «Core ISA and Hazards» και ενεργοποιήστε το «Pipelined» και απενεργοποιήστε το «Hazard Unit». Πατήστε Start empty.
- δ'. Φορτώστε τον κώδικα και μεταγλωττίστε τον.

Τρέξτε το πρόγραμμα. Τι παρατηρείτε;

Η λανθασμένη εκτέλεση του προγράμματος οφείλεται στο γεγονός ότι **οι κίνδυνοι δεδομένων δεν αντιμετωπίζονται**. Αλλάζοντας **μόνο** τον κώδικα, προσπαθήστε να πάρετε τα σωστά αποτελέσματα στον συγκεκριμένο επεξεργαστή.



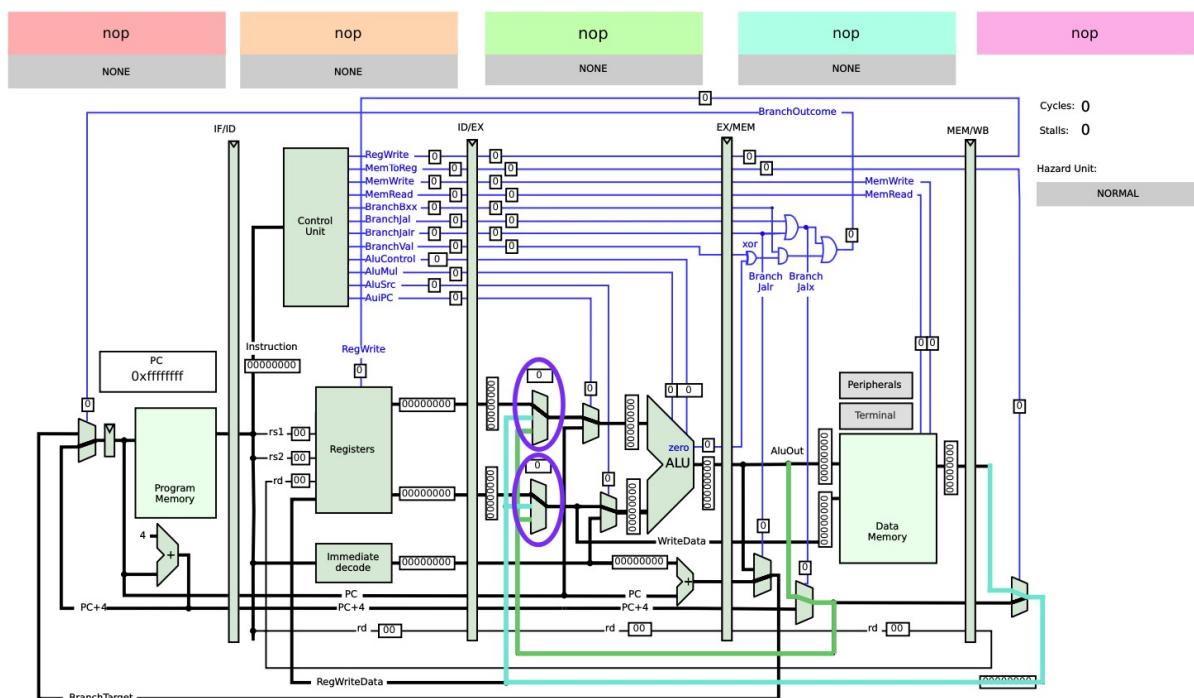
Ε10 Βελτίωση ενός επεξεργαστή με διοχέτευση

Ε10.1 Περιορισμός των κινδύνων της διοχέτευσης

Οι nop εντολές που προστίθενται προκειμένου να αποφευχθούν οι κίνδυνοι δεδομένων και ελέγχου μπορούν να περιοριστούν σε αρκετά μεγάλο βαθμό με επιπλέον λογική και hardware.

Ε10.1.1 Μονάδα ελέγχου κινδύνων με προώθηση

Η ιδέα της προώθησης (*forwarding*) βασίστηκε στο γεγονός ότι κύκλοι καθυστέρησης οφείλονται στην περίπτωση που ορισμένοι καταχωρητές προέλευσης έχουν μη ανανεωμένες τιμές και αναμένουν τη νέα τους τιμή να εγγραφεί, ενώ η ανανεωμένη τιμή είναι έτοιμη σε προηγούμενο στάδιο, είτε στο στάδιο **Execute**, αν αυτή προκύπτει ως έξοδος από την ALU, είτε στο στάδιο **Mem**, αν αυτή είναι αποτέλεσμα δεδομένων που προέρχονται από τη μνήμη.



Εικόνα 20: Αναπαράσταση ενός επεξεργαστή με διοχέτευση και μονάδα ανίχνευσης κινδύνων

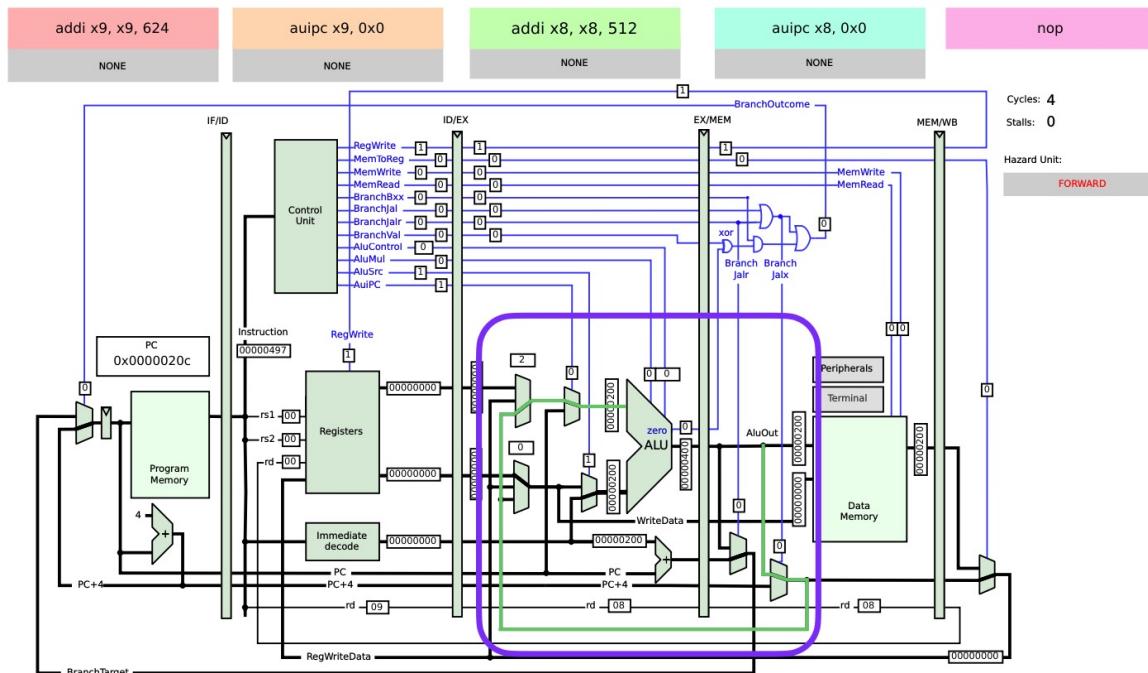
Δημιουργήστε μια προσομοίωση ενός επεξεργαστή με διοχέτευση πατώντας «Stall or forward when hazard is detected» κάτω από το «Hazard Unit».

Παρατηρήστε τη νέα αναπαράσταση του επεξεργαστή με διοχέτευση στην Εικόνα 20. Οι σημειωμένοι πολυπλέκτες και τα σήματα αποτελούν το επιπλέον υλικό που χρειάζεται προκειμένου η ALU να μη λαμβάνει τιμές στην είσοδο της μόνο από καταχωρητές ή το immediate πεδίο αλλά και από την έξοδο της ή την έξοδο της μνήμης δεδομένων - αυτές οι νέες είσοδοι υλοποιούν τον μηχανισμό της προώθησης.



Μη ξεχνάτε ότι δεν μας απασχολεί το γεγονός ότι οι τιμές προέρχονται από ένα μετέπειτα στάδιο αλλά από έναν προηγούμενο κύκλο!

Φορτώστε και εκτελέστε βηματικά το **lab9.s** μέχρι και τον 5^ο κύκλο, δηλαδή μέχρι και την ολοκλήρωση της πρώτης εντολής auipc x8, 0x0. Παρατηρήστε τους πολυπλέκτες μπροστά από την ALU στον κύκλο 4. Στην Εικόνα 21 παρουσιάζεται η προώθηση από την έξοδο της ALU στην πρώτη είσοδο της ALU. Στον κύκλο 3 υπολογίζεται το αποτέλεσμα της auipc x8, 0x0 που είναι η τιμή 0x200, η οποία προωθείται στην πρώτη είσοδο της ALU στον 4^ο κύκλο.



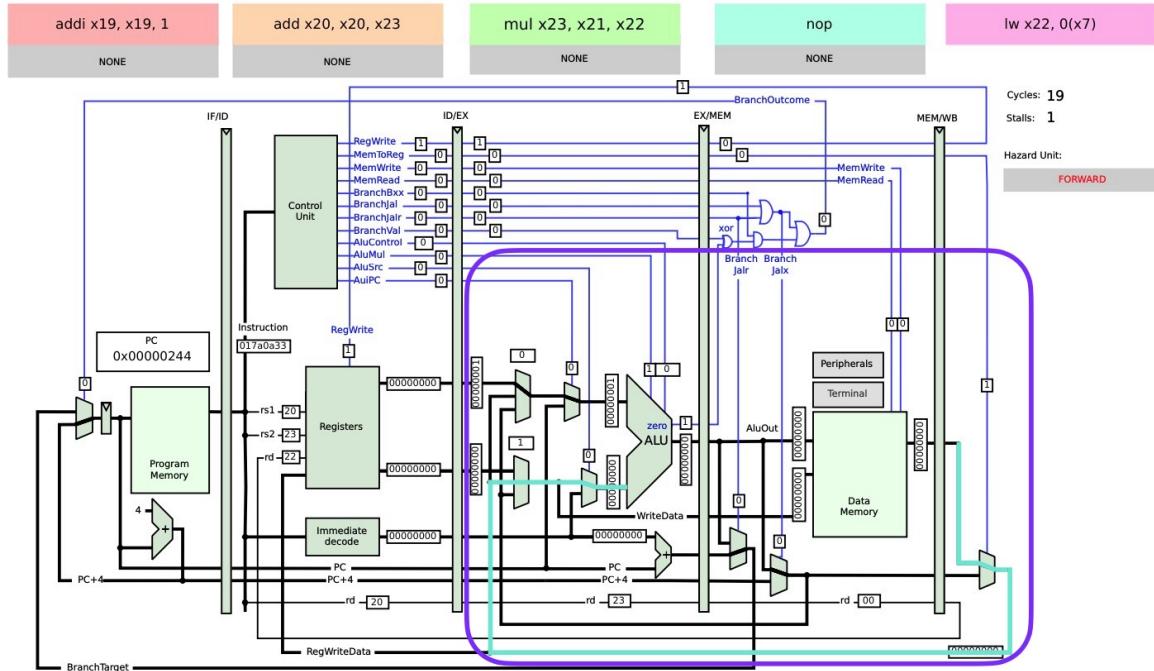
Εικόνα 21: Επίλυση κινδύνου δεδομένων με προώθηση από την ALU

Επιπρόσθετα, εκτελέστε βηματικά μέχρι τον κύκλο 15 (προσκόμιση της εντολής `lw x22, 0(x7)`) και παρατηρήστε τη διαδρομή δεδομένων μέχρι και την ολοκλήρωσή της. Προστίθεται μόνο **1 κύκλος καθυστέρησης** διότι υπάρχει εξάρτηση από την αμέσως επόμενη εντολή `mul x23, x21, x22` που χρειάζεται την ανανεωμένη τιμή του καταχωρητή `x22` στον κύκλο 18 στο στάδιο **Execute** αλλά υπολογίζεται στον ίδιο κύκλο στο στάδιο **Mem**. Επομένως, πρέπει να περιμένουμε μέχρι τον κύκλο 19 για να εκτελεστεί η εντολή `mul x23, x21, x22`. Η προώθηση από τη μνήμη στον 19° κύκλο παρουσιάζεται στην Εικόνα 22. Φορτώνεται το 0 από τη μνήμη δεδομένων, η οποία προωθείται στη δεύτερη είσοδο της ALU.

Ε10.1.2 Πρόβλεψη διακλάδωσης

Μια τεχνική προκειμένου να αποφευχθούν οι κύκλοι που οφείλονται στην καθυστέρηση υπολογισμού του αποτελέσματος μιας διακλάδωσης αποτελεί η πρόβλεψη του. Σε περίπτωση λανθασμένης πρόβλεψης πληρώνονται οι κύκλοι καθυστέρησης.

Η στατική πρόβλεψη διακλάδωσης υποθέτει ότι κάθε διακλάδωση έχει σταθερό αποτέλεσμα, δηλαδή είτε είναι πάντα αληθής είτε πάντα ψευδής. Μια εύκολη



Εικόνα 22: Προώθηση από την μνήμη δεδομένων στην ALU στην περίπτωση κινδύνου φόρτωσης-χρήσης

λύση αποτελεί η πρόβλεψη μη λήψης (predict not taken), στην οποία εκτελούνται οι αμέσως επόμενες εντολές από μια διακλάδωση.

ΠΡΟΣΟΧΗ. Ο προσομοιωτής QtRVsim δεν υλοποιεί στατική πρόβλεψη διακλάδωσης.

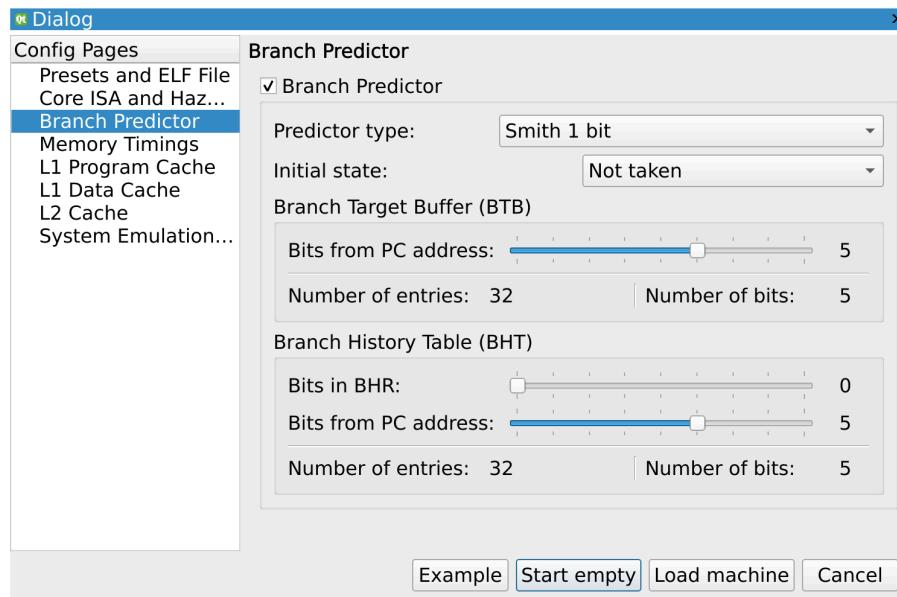
Θα ασχοληθούμε με τη δυναμική πρόβλεψη διακλάδωσης, η οποία προβλέπει την έκβαση μιας διακλάδωσης βάσει προηγούμενων αποτελεσμάτων. Το ιστορικό αποθηκεύεται στον Branch History Table (BHT), ενώ ο προορισμός της στον Branch Target Buffer (BTB).

Δημιουργήστε μια προσομοίωση ενός επεξεργαστή με διοχέτευση, με προώθηση και με πρόβλεψη διακλάδωσης πατώντας στην κατηγορία «Branch Predictor» και επιλέγοντας τύπο «Smith 1». Επιπλέον, θα χρησιμοποιήσουμε 5 bits από τον PC τόσο για το BTB, όσο για το BHT. Για διευκόλυνση σας, παρατίθεται η Εικόνα 23.

Παρατηρείστε πως διαχειρίζεται ο επεξεργαστής τις διακλαδώσεις (είτε βηματικά, είτε με breakpoint στην προηγούμενη εντολή της διακλάδωσης addi x19, x19, 1 και βηματική εκτέλεση ύστερα) ανοίγοντας τα παράθυρα History table (Windows → Branch Predictor (History table)) και Target table (Windows → Branch Predictor (Target table)).

ΠΡΟΣΟΧΗ. Τα στατιστικά του branch predictor δεν υπολογίζονται σωστά με breakpoint στις διακλαδώσεις.

Στην Εικόνα 24 παρουσιάζονται οι περιπτώσεις που η πρόβλεψη είναι ψευδής και αληθής. Στη μοναδική διακλάδωση του προγράμματος, ο branch predictor που ορίσαμε αποτυγχάνει 2 φορές, μια την πρώτη φορά αφού δεν υπάρχει εγγραφή για αυτή την διακλάδωση και μια στην τελευταία επανάληψη αφού η συνθήκη μετατράπηκε από αληθής σε ψευδής. Μπορείτε να πειραματιστείτε με διαφορε-



Εικόνα 23: Παράδειγμα διαμόρφωσης branch predictor

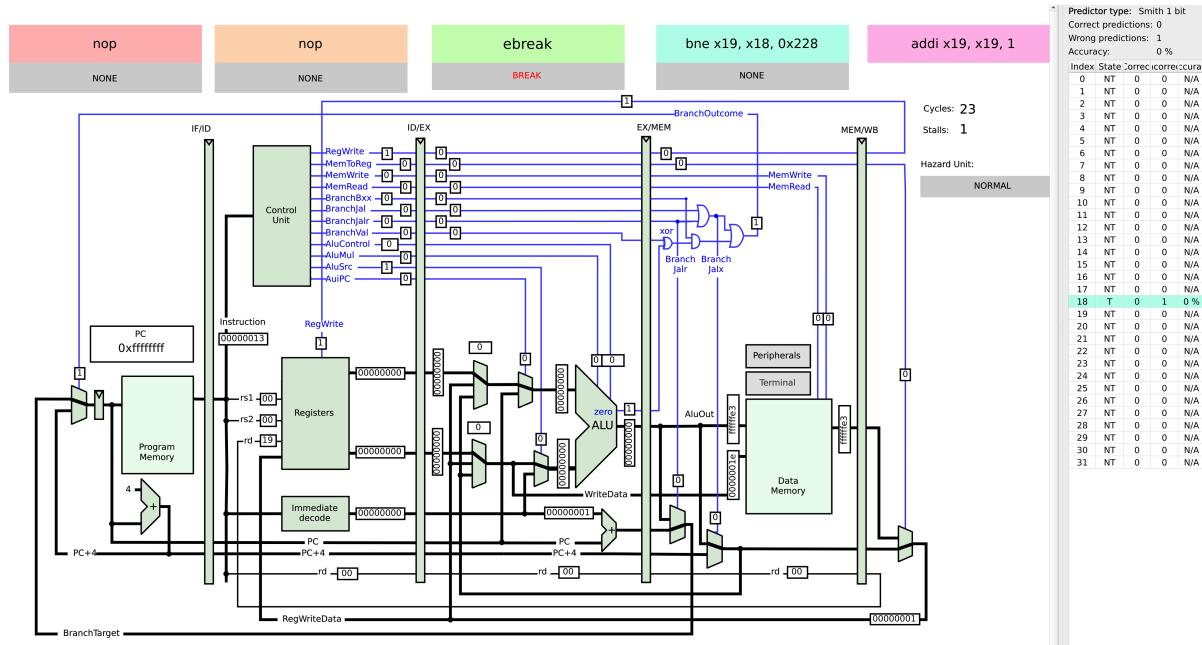
τικούς branch predictors.

Ε10.2 Σύγκριση επεξεργαστών ενός κύκλου και με διοχέτευση

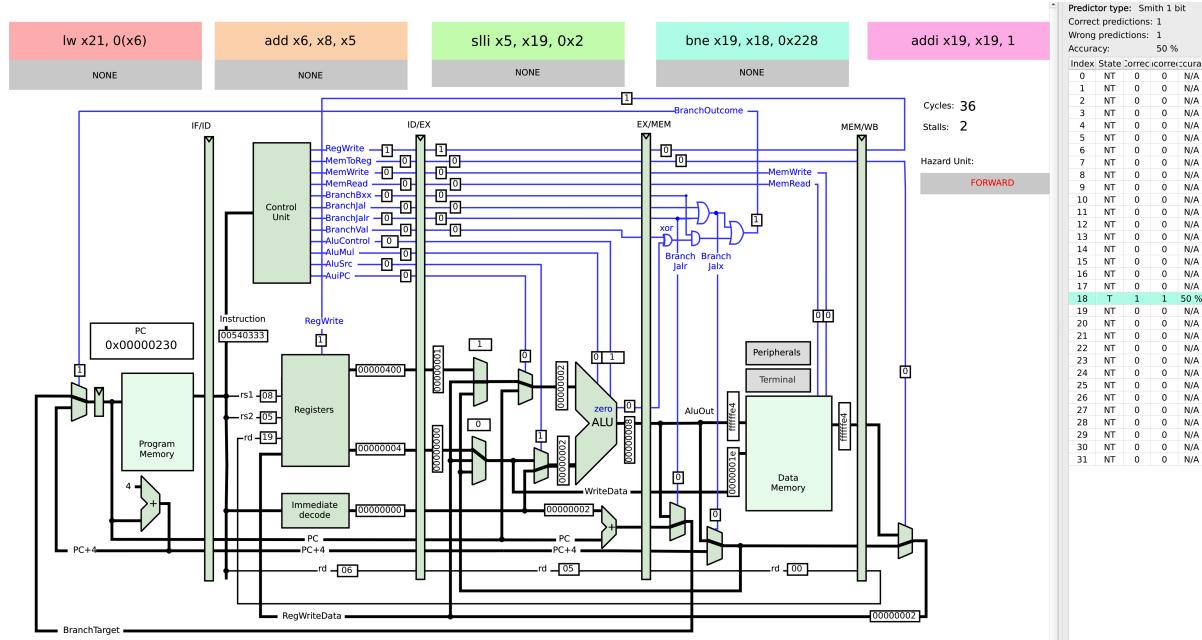
Γράψτε πρόγραμμα σε συμβολική γλώσσα RISC-V το οποίο θα ταξινομεί έναν πίνακα 100 στοιχείων με τη μέθοδο της φυσαλίδας (bubblesort). Δοκιμάστε να το εκτελέσετε στον προσομοιωτή QtRVsim σε τρεις προσομοιώσεις επεξεργαστών:

- α'. Σ' έναν επεξεργαστή ενός κύκλου
- β'. Σ' έναν επεξεργαστή με διοχέτευση και μονάδα ελέγχου κινδύνων χωρίς προώθηση και πρόβλεψη διακλάδωσης
- γ'. Σ' έναν επεξεργαστή με διοχέτευση και μονάδα ελέγχου κινδύνων με προώθηση και πρόβλεψη διακλάδωσης

Ποιος επεξεργαστής χρειάζεται περισσότερους κύκλους για να εκτελέσει το πρόγραμμα; Μπορείτε να βγάλετε συμπέρασμα ποιος επεξεργαστής είναι ο πιο αργός στην εκτέλεση του προγράμματός σας; Αιτιολογείστε την απάντησή σας.



(α') Παράδειγμα αποτυχημένης πρόβλεψης διακλάδωσης



(β') Παράδειγμα επιτυχημένης πρόβλεψης διακλάδωσης

Εικόνα 24: Πρόβλεψη διακλάδωσης στον QtRVsim

Παράρτημα A'

Επισκόπηση του επεξεργαστή RISC-V



A'.1 Τύποι δεδομένων

Οι βασικοί τύποι δεδομένων ενός επεξεργαστή RISC-V περιλαμβάνουν ακεραίους (integer), πραγματικούς (κινητής υποδιαστολής – floating-point), και χαρακτήρες (characters). Ο επεξεργαστής RISC-V υποστηρίζει τα εξής μεγέθη για την αποθήκευση δεδομένων: byte (8 bit), halfword (16 bit), word (32 bit). Οι αριθμοί κινητής υποδιαστολής μπορεί να είναι απλής ακρίβειας (single precision – 32-bit) ή διπλής ακρίβειας (double precision – 64-bit). Οι χαρακτήρες (characters) έχουν μέγεθος 1 byte, ενώ οι συμβολοσειρές (strings) αποτελούνται από πολλά bytes στη σειρά (ακολουθία από bytes). Συνοπτικά, ο επεξεργαστής RISC-V υποστηρίζει τους τύπους δεδομένων που παρουσιάζονται στον Πίνακα 7.

Τύπος Δεδομένων	Μέγεθος
byte	8-bit ακέραιος
halfword	16-bit ακέραιος
word	32-bit ακέραιος
float	32-bit αριθμός κινητής υποδιαστολής
double	64-bit αριθμός κινητής υποδιαστολής

Πίνακας 7: Τύποι δεδομένων και μεγέθη

Ο τύπος «halfword» συχνά αναφέρεται και ως «half».



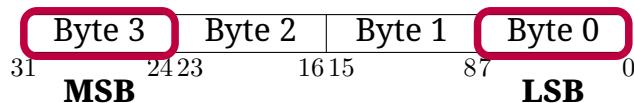
A'.2 Οργάνωση της μνήμης

Η μνήμη (memory) μπορεί να αναπαρασταθεί ως μια σειρά από bytes, καθένα από τα οποία διαθέτει τη δική του διεύθυνση (byte addressable memory – μνήμη διευθυνσιοδοτούμενη κατά byte). Αυτό σημαίνει ότι **κάθε μια διεύθυνση μνήμης αποθηκεύει ένα byte πληροφορίας**. Όμως, για να αποθηκευτεί μια ολόκληρη λέξη (word) στη μνήμη, χρειάζονται 4 bytes (= 32 bits), τα οποία χρησιμοποιούν 4 (συνεχόμενες) διευθύνσεις μνήμης. Όταν πρέπει να αποθηκευτεί στη μνήμη μια ποσότητα που αποτελείται από πολλά bytes (π.χ., ένας ακέραιος αριθμός), **είναι απαραίτητο να καθοριστεί η σειρά με την οποία διευθυνσιοδοτούνται (ή αριθμούνται) τα επιμέρους bytes μέσα σε αυτή την ποσότητα**.

Big-endian: Το περισσότερο σημαντικό byte μίας λέξης (Most Significant Byte – MSB) αποθηκεύεται **στη μικρότερη διεύθυνση μνήμης**.

Little-endian: Το λιγότερο σημαντικό byte μίας λέξης (Least Significant Byte – LSB) αποθηκεύεται **στη μικρότερη διεύθυνση μνήμης**.

Ο επεξεργαστής RISC-V ακολουθεί την οργάνωση μνήμης little-endian. Ανεξαρτήτως της οργάνωσης της μνήμης, σε μια λέξη των 32 bits, το περισσότερο σημαντικό byte αναπαρίσταται πάντα στα αριστερά της λέξης ενώ το λιγότερο σημαντικό byte αναπαρίσταται στα δεξιά της λέξης, όπως φαίνεται στην Εικόνα 25.



Εικόνα 25: Το περισσότερο & το λιγότερο σημαντικό byte μίας λέξης

Ας υποθέσουμε ότι θέλουμε να αποθηκεύσουμε στη μνήμη τις λέξεις (32 bit η κάθε μια): 0x11223344, 0x5566, 58, και 3500000.

Παρατηρείστε ότι οι δύο πρώτες λέξεις είναι στο δεκαεξαδικό σύστημα αρίθμησης, ενώ οι δύο τελευταίες είναι στο δεκαδικό σύστημα αρίθμησης. Συνεπώς, το 58₁₀ είναι το 0x0000003A (σε 32-bit) στο δεκαεξαδικό, και το 3500000₁₀ είναι το 0x003567E0 (σε 32-bit) στο δεκαεξαδικό. Στην Εικόνα 26 φαίνεται ένα παράδειγμα αποθήκευσης των τεσσάρων αυτών λέξεων στη μνήμη όταν η οργάνωση που ακολουθεί είναι little-endian (Εικόνα 26α') ή big-endian (Εικόνα 26β').

Little-Endian				Big-Endian			
Διεύθυνση μνήμης	Τιμή	Διεύθυνση μνήμης	Τιμή	Διεύθυνση μνήμης	Τιμή	Διεύθυνση μνήμης	Τιμή
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0x1001000C	E0 67 35 00	0x1001000F	0x1001000C	00 35 67 E0	00 00 00 3A	0x1001000F	0x1001000F
0x10010008	3A 00 00 00	0x1001000F	0x10010008	00 00 55 66	00 00 00 3A	0x10010007	0x10010007
0x10010004	66 55 00 00	0x10010007	0x10010004	11 22 33 44	11 22 33 44	0x10010003	0x10010003
0x10010000	44 33 22 11	0x10010003	0x10010000				

(α') little-endian

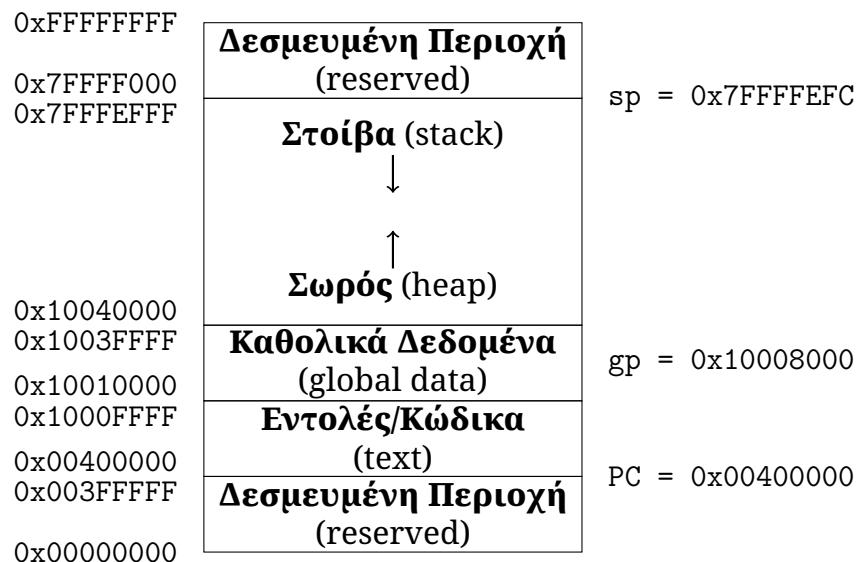
(β') big-endian

Εικόνα 26: Διευθυνσιοδότηση little-endian & big-endian



A'.3 Διάταξη μνήμης

Η γενική διάταξη τη μνήμης για ένα πρόγραμμα παρουσιάζεται στην Εικόνα 27.



Εικόνα 27: Διάταξη μνήμης (memory layout)

Η δεσμευμένη περιοχή δεν είναι διαθέσιμη για τα προγράμματα χρήστη (user programs). Στην περιοχή των εντολών (που παραδοσιακά ονομάζεται text – κείμενο) αποθηκεύεται ο κώδικας μηχανής, και μπορεί να φιλοξενήσει μέχρι 252MB κώδικα. Στην περιοχή των καθολικών δεδομένων (global data) αποθηκεύονται οι μεταβλητές που είναι ορατές από όλες οι συναρτήσεις ενός προγράμματος. Οι καθολικές μεταβλητές ορίζονται κατά την εκκίνηση, πριν ξεκινήσει η εκτέλεση του προγράμματος. Το μέγεθος των καθολικών δεδομένων είναι αρκετά μεγάλο ώστε να αποθηκεύει μέχρι 192KB. Οι καθολικές μεταβλητές προσπελάζονται χρησιμοποιώντας τον καθολικό δείκτη (global pointer, gp), ο οποίος αρχικοποιείται στην τιμή 0x100080000 (δηλαδή ακριβώς στη μέση της περιοχής των καθολικών δεδομένων). Σε αντίθεση με τον δείκτη στοίβας (stack pointer, sp), ο gp δεν αλλάζει περιεχόμενο κατά την εκτέλεση του προγράμματος. Οποιαδήποτε καθολική μεταβλητή μπορεί να προσεγγιστεί με θετική ή αρνητική σχετική απόσταση (offset) από τον gp. Στον σωρό (heap) αποθηκεύονται τα δυναμικά δεσμευμένα δεδομένα, ενώ στη στοίβα τα στατικά δεδομένα των συναρτήσεων (π.χ., οι εσωτερικές μεταβλητές). Η στοίβα ξεκινάει από την υψηλότερη διεύθυνση μνήμης και «μεγαλώνει» (προστίθενται στοιχεία) προς τη χαμηλότερη¹.

¹Σημειώστε ότι τα μεγέθη και οι διευθύνσεις μνήμης της παραπάνω διάταξης μνήμης που φαίνεται στην Εικόνα 27 είναι ενδεικτικές και ακολουθούν την ίδια διάταξη που χρησιμοποιεί και ο προσομοιωτής RARS



A'.4 Οι καταχωρητές του επεξεργαστή

Ένας καταχωρητής είναι μια θέση αποθήκευσης που βρίσκεται στο εσωτερικό του ίδιου του επεξεργαστή (ξεχωριστά από τη μνήμη) και είναι ορατή στον προγραμματιστή συμβολικής γλώσσας δηλαδή κάθε καταχωρητής αναφέρεται ρητά με το όνομά του. Όλες οι αριθμητικές και λογικές πράξεις που εκτελούνται από έναν επεξεργαστή RISC-V χρησιμοποιούν απαραιτήτως καταχωρητές. Οι εντολές της αρχιτεκτονικής RISC-V δεν επιτρέπουν αριθμητικές ή λογικές πράξεις μεταξύ καταχωρητή και μνήμης, αλλά μόνο μεταξύ καταχωρητών.

Ένας επεξεργαστής RISC-V έχει 32 καταχωρητές για ακέραιους αριθμούς (integer registers) (από x0 μέχρι x31), μεγέθους 32 bit ο καθένας αν πρόκειται για την έκδοση RV32I, ή 64 bit ο καθένας αν πρόκειται για την έκδοση RV64I. Εφόσον σε μια συγκεκριμένη σχεδίαση υλοποιείται η επέκταση της αρχιτεκτονικής του συνόλου εντολών για τους αριθμούς κινητής υποδιαστολής (δηλαδή, RV32I{F,D,Q}), τότε η συγκεκριμένη σχεδίαση περιλαμβάνει επίσης 32 καταχωρητές για πραγματικούς αριθμούς κινητής υποδιαστολής (floating-point registers), μεγέθους 32 bit ο καθένας (από f0 μέχρι f31).

Κάποιοι από τους ακέραιους καταχωρητές χρησιμοποιούνται για ειδικούς σκοπούς. Για παράδειγμα, ο καταχωρητής x2 χρησιμοποιείται αποκλειστικά ως δείκτης στοίβας (stack pointer), και αναφέρεται και με το όνομα sp.

Οι διαθέσιμοι ακέραιοι καταχωρητές (integer registers) και οι καταχωρητές κινητής υποδιαστολής (floating-point registers) ενός RISC-V επεξεργαστή και η τυπική χρήση τους συνοψίζεται στον Πίνακα 8.

Ο μετρητής προγράμματος (program counter, pc) είναι ένας καταχωρητής ειδικού σκοπού, σε αντίθεση με όλους τους άλλους καταχωρητές που φαίνονται στον Πίνακα , οι οποίοι είναι όλοι καταχωρητές γενικού σκοπού (general-purpose registers). Ο pc περιέχει πάντα τη διεύθυνση της επόμενης προς εκτέλεση εντολής και ενημερώνεται αυτόματα από τον ίδιο τον επεξεργαστή μετά από την εκτέλεση κάθε εντολής. Αυτός ο καταχωρητής δεν είναι προσπελάσιμος απευθείας από τα προγράμματα χρηστών δηλαδή δεν μπορεί κάποια εντολή να του αλλάξει ρητά την τιμή χρησιμοποιώντας το όνομά του.



Καταχωρητής (Register)	Όνομα	Χρήση Καταχωρητή
Ακέραιοι Καταχωρητές (Integer Registers)		
x0	zero	Η σταθερή τιμή 0 (hard-wired)
x1	ra	Διεύθυνση επιστροφής (return address)
x2	sp	Δείκτης στοίβας (stack pointer)
x3	gp	Καθολικός δείκτης (global pointer)
x4	tp	Δείκτης νήματος (thread pointer)
x5-x7	t0-t2	Προσωρινοί (temporary) καταχωρητές
x8	s0/fp	Αποθηκευμένος (saved) καταχωρητής / Δείκτης πλαισίου (frame pointer)
x9	s1	Αποθηκευμένος (saved) καταχωρητής
x10-x11	a0-a1	Ορίσματα συναρτήσεων / τιμές επιστροφής
x12-x17	a2-a7	Ορίσματα συναρτήσεων
x18-x27	s2-s11	Αποθηκευμένοι καταχωρητές
x28-x31	t3-t6	Προσωρινοί καταχωρητές
pc	-	Μετρητής προγράμματος (Program Counter)
Καταχωρητές Κινητής Υποδιαστολής-KY-(Floating-Point Registers)		
f0-f7	ft0-ft7	Προσωρινοί καταχωρητές KY
f8-f9	fs0-fs1	Αποθηκευμένοι καταχωρητές KY
f10-f11	fa0-fa1	Ορίσματα KY / Τιμές για αποτελέσματα συναρτήσεων KY
f12-f17	fa2-fa7	Ορίσματα KY
f18-f27	fs2-fs11	Αποθηκευμένοι καταχωρητές KY
f28-f31	ft8-ft11	Προσωρινοί καταχωρητές KY

Πίνακας 8: Καταχωρητές ειδικού και γενικού σκοπού του επεξεργαστή RISC-V



A'.5 Μορφή και βασικά χαρακτηριστικά προγραμμάτων

Τα προγράμματα σε συμβολική γλώσσα (assembly) RISC-V διαθέτουν μια σειρά χαρακτηριστικών και απαιτήσεων. Αυτό αρχικά περιλαμβάνει ένα σωστά διαμορφωμένο αρχείο πηγαίου κώδικα συμβολικής γλώσσας. Ένα σωστά διαμορφωμένο αρχείο πηγαίου κώδικα αποτελείται από δύο κύρια μέρη: (α') το τμήμα δεδομένων (data segment – όπου τοποθετούνται τα δεδομένα), και (β') το τμήμα του κειμένου/κώδικα (text segment – όπου τοποθετείται ο κώδικας). Οι επόμενες ενότητες συνοψίζουν τις απαιτήσεις μορφοποίησης και εξηγούν κάθε ένα από τα μέρη του πηγαίου κώδικα και αναλύουν όλα τα χαρακτηριστικά που μπορεί να έχει ένα πρόγραμμα σε συμβολική γλώσσα RISC-V.

A'.5.1 Σχόλια

Ο χαρακτήρας # αντιπροσωπεύει μια γραμμή σχολίων². Οτιδήποτε πληκτρολογείται μετά τον χαρακτήρα # είναι σχόλιο. Οι κενές γραμμές γίνονται αποδεκτές χωρίς να χρειάζεται η χρήση του #.

A'.5.2 Οδηγίες προς τον συμβολομεταφραστή

Μια οδηγία (directive) είναι ένα μήνυμα προς τον συμβολομεταφραστή (assembler), ή τον προσομοιωτή, που του υποδεικνύει κάτι που χρειάζεται να γνωρίζει για να πραγματοποιήσει τη διαδικασία μετάφρασης από τη συμβολική γλώσσα RISC-V σε κώδικα μηχανής (machine code). Αυτό περιλαμβάνει την επισήμανση του σημείου όπου δηλώνονται τα δεδομένα ή/και ο ορισμός του κώδικα. Οι οδηγίες δεν είναι εκτελέσιμες δηλώσεις. Οι οδηγίες ξεκινούν με μια τελεία (.) και απαιτούνται για τον ορισμό των δηλώσεων έναρξης και λήξης του τμήματος δεδομένων/εντολών, τον καθορισμό της έναρξης και λήξης των διαδικασιών/λειτουργιών, κ.λπ. Για παράδειγμα, .data ή .text. Ο Πίνακας 9 επεξηγεί τις βασικές διαθέσιμες οδηγίες προς τον συμβολομεταφραστή.

A'.5.3 Δηλώσεις δεδομένων

Τα δεδομένα πρέπει να δηλώνονται στην ενότητα του τμήματος δεδομένων, δηλαδή αμέσως μετά την οδηγία .data. Όλες οι μεταβλητές και οι σταθερές τοποθετούνται σε αυτήν την ενότητα. Τα ονόματα μεταβλητών (ή αλλιώς το όνομα της ετικέτας) πρέπει να ξεκινούν με ένα γράμμα που ακολουθείται από γράμματα ή αριθμούς (συμπεριλαμβανομένων ορισμένων ειδικών χαρακτήρων όπως η κάτω παύλα (underscore) «_»), και πρέπει να τερματίζονται με μια «:» (άνω κάτω

²Στο βιβλίο «Οργάνωση και Σχεδίαση Υπολογιστών, 2η έκδοση RISC-V: Η Διασύνδεση Υλικού και Λογισμικού» των D. Patterson και J. Hennessy, 2024, ο συμβολισμός που χρησιμοποιείται για τη δήλωση σχολίων είναι //. Στον προσομοιωτή RARS που χρησιμοποιούμε η δήλωση των σχολίων γίνεται με τον χαρακτήρα #, όπως ακριβώς αναφέρεται σε αυτό το φυλλάδιο.



Όνομα Οδηγίας	Περιγραφή
.data	Οι δηλώσεις των τιμών που βρίσκονται κάτω από αυτή την οδηγία αποθηκεύονται στο τμήμα δεδομένων στην επόμενη διαθέσιμη διεύθυνση μνήμης
.text	Οι δηλώσεις των στοιχείων (εντολές) που βρίσκονται κάτω από αυτή την οδηγία αποθηκεύονται στο τμήμα κειμένου (εντολών) στην επόμενη διαθέσιμη διεύθυνση
.globl Label	Δηλώνει τις αναφερόμενες ετικέτες ως καθολικές για να επιτρέπεται η αναφορά τους και από άλλα αρχεία. Χρησιμοποιείται για τη δήλωση της κύριας διαδικασίας του προγράμματος
.align N	Ευθυγράμμιση των στοιχείων δεδομένων στη συγκεκριμένη οριογραμμή (όπου N μπορεί να είναι: 0=byte, 1=μισή λέξη, 2=λέξη, 3=διπλή λέξη)
.eqv A B	Αντικαθιστά τον δεύτερο τελεστέο (B) με τον πρώτο (A). Ο πρώτος τελεστέος είναι σύμβολο, ο δεύτερος είναι έκφραση (όπως το #define)

Πίνακας 9: Βασικές διαθέσιμες οδηγίες προς τον συμβολομεταφραστή και η περιγραφή τους

τελεία). Οι ορισμοί μεταβλητών πρέπει να περιλαμβάνουν το όνομα (ετικέτα), τον τύπο δεδομένων και την αρχική τιμή για τη μεταβλητή. Ο τύπος δεδομένων πρέπει να έχει μπροστά μια τελεία (.). Η γενική μορφή είναι:

<Όνομα_Μεταβλητής/Ετικέτα>: .<Τύπος_Δεδομένου> <Τιμή>

Οι τύποι δεδομένων που υποστηρίζονται παρουσιάζονται στον Πίνακα 10.

Όνομα Οδηγίας / Τύπος δεδομένου	Περιγραφή
.byte	Μεταβλητή μεγέθους 8-bit
.half	Μεταβλητή μεγέθους 16-bit
.word	Μεταβλητή μεγέθους 32-bit
.ascii	Συμβολοσειρά ASCII χαρακτήρων
.asciz	Συμβολοσειρά ASCII χαρακτήρων που τελειώνει με τον χαρακτήρα NULL (\0)
.float	Αριθμός κινητής υποδιαστολής μεγέθους 32-bit
.double	Αριθμός κινητής υποδιαστολής μεγέθους 64-bit
.space n	Δέσμευση n bytes μη αρχικοποιημένης μνήμης

Πίνακας 10: Οδηγίες προς τον συμβολομεταφραστή για τύπους δεδομένων (data types)



A'5.3.1 Δήλωση ακέραιων

Οι τιμές ακέραιων ορίζονται με τις οδηγίες `.word`, `.half` ή `.byte`. Για την αναπαράσταση των αρνητικών τιμών χρησιμοποιείται το συμπλήρωμα ως προς δύο. Παραδείγματος χάριν, οι δηλώσεις του Κώδικα 4 χρησιμοποιούνται για να ορίσουν τις ακέραιες μεταβλητές `var1w`, `var2w`, και `var3w` ως τιμές λέξεων μεγέθους 32 bit και να τις αρχικοποιούν στις τιμές 350000, 0x3f45a και -434, αντίστοιχα.

```
1 var1w:    .word 350000
2 var2w:    .word 0x3f45a
3 var3w:    .word -434
```

Κώδικας 4: Δηλώσεις λέξεων (4 bytes)

Οι δηλώσεις του Κώδικα 5 χρησιμοποιούνται για να ορίσουν τις ακέραιες μεταβλητές `var1b` και `var2b` ως τιμές μεγέθους 8-bit και να τις αρχικοποιήσουν σε 19 και -53, αντίστοιχα.

```
1 var1b:    .byte 19
2 var2b:    .byte -53
```

Κώδικας 5: Δηλώσεις byte

A'5.3.2 Δήλωση συμβολοσειράς

Μια συμβολοσειρά είναι μια σειρά χαρακτήρων μεγέθους byte, τυπικά τερματισμένων με ένα byte NULL (0x00) με βάση τον ορισμό της γλώσσας C. Οι συμβολοσειρές ορίζονται με τις οδηγίες `.ascii` ή `.asciz`. Οι χαρακτήρες αντιτροσωπεύονται χρησιμοποιώντας τυπικούς ASCII χαρακτήρες (η κωδικοποίηση ASCII φαίνεται στον Πίνακα 11). Οι ειδικοί χαρακτήρες νέας γραμμής `\\n` και tab `\\t` μπορούν να συμμετέχουν σε συμβολοσειρές. Η δήλωση του Κώδικα 6 χρησιμοποιείται για τον καθορισμό μιας συμβολοσειράς με όνομα `msg` και την αρχικοποίησή του σε "Hello World".

```
1 msg:    .asciz "Hello World\\n"
```

Κώδικας 6: Δηλώσεις string

Σε αυτό το παράδειγμα, η συμβολοσειρά τερματίζεται με τον χαρακτήρα NULL. Το NULL είναι ένας μη εκτυπώσιμος ASCII χαρακτήρας και χρησιμοποιείται για να επισημάνει το τέλος της συμβολοσειράς με βάση τον ορισμό της C.

Ο τερματισμός NULL είναι τυπικός και απαιτείται από την κλήση συστήματος εκτύπωσης συμβολοσειρών για να λειτουργήσει σωστά (βλ. Πίνακας 12). Για να ορίσετε μια συμβολοσειρά με πολλές γραμμές, ο τερματισμός του NULL θα απαιτηθεί μόνο στην τελική ή την τελευταία γραμμή.



#	Char	#	Char										
20	κενό	30	0	40	@	50	P	60	‘	70	p		
21	!	31	1	41	A	51	Q	61	a	71	q		
22	”	32	2	42	B	52	R	62	b	72	r		
23	#	33	3	43	C	53	S	63	c	73	s		
24	\$	34	4	44	D	54	T	64	d	74	t		
25	%	35	5	45	E	55	U	65	e	75	u		
26	&	36	6	46	F	56	V	66	f	76	v		
27	‘	37	7	47	G	57	W	67	g	77	w		
28	(38	8	48	H	58	X	68	h	78	x		
29)	39	9	49	I	59	Y	69	i	79	y		
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z		
2B	+	3B	;	4B	K	5B	[6B	k	7B	{		
2C	,	3C	<	4C	L	5C	\	6C	l	7C			
2D	-	3D	=	4D	M	5D]	6D	m	7D	}		
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~		
2F	/	3F	?	4F	O	5F	_	6F	o				

Πίνακας 11: Κωδικοποίηση ASCII

A'.5.3.3 Δήλωση Αριθμών Κινητής Υποδιαστολής

Οι τιμές κινητής υποδιαστολής (floating-point) ορίζονται με τις οδηγίες `.float` (32-bit) ή `.double` (64-bit) ή `.word` για να αναπαράσταση ενός 32-bit αριθμού κινητής υποδιαστολής με το πρότυπο IEEE-754. Η μορφή κινητής υποδιαστολής IEEE-754 χρησιμοποιείται για την εσωτερική αναπαράσταση πραγματικών αριθμών.

```
1 pi: .float    3.1415
2 dfp: .double   9.257841
3 f: .word      0x3fc00000 # 1.5
```

Κώδικας 7: Δηλώσεις αριθμών κινητής υποδιαστολής

Για παράδειγμα, οι δηλώσεις του Κώδικα 7 χρησιμοποιούνται για τον ορισμό των μεταβλητών κινητής υποδιαστολής ρι μιας τιμής 32-bit αρχικοποιημένης στο 3.1415 και dfp μιας τιμής 64-bit κινητής υποδιαστολής αρχικοποιημένης στο 9.257841. Επιπλέον, ορίζεται η μεταβλητή f και αρχικοποιείται στο 1.5 με βάση το πρότυπο IEEE-754.

A'.5.4 Ο κώδικας

A'.5.4.1 Η οδηγία `.text`

Ο κώδικας πρέπει να γράφεται αμέσως μετά την οδηγία `.text`. Επιπλέον, υπάρχουν ορισμένες βασικές απαιτήσεις για την ονομασία της «κύριας» διαδικασίας



(της πρώτης διαδικασίας που πρέπει να εκτελεστεί και να είναι εμφανής από όλα τα αρχεία κώδικα του ίδιου προγράμματος σε περίπτωση που υπάρχουν). Η οδηγία `.globl <name>` χρησιμοποιείται για τον ορισμό του ονόματος της κύριας διαδικασίας. Επίσης, η κύρια διαδικασία πρέπει να ξεκινά με μια ετικέτα με το όνομα της διαδικασίας.

A'.5.4.2 Ετικέτες (Labels)

Οι ετικέτες είναι θέσεις κώδικα (άρα και διευθύνσεις μνήμης), που συνήθως χρησιμοποιούνται ως όνομα συνάρτησης / διαδικασίας ή ως στόχος για μια εντολή διακλάδωσης. Επίσης, χρησιμοποιούνται και στο τμήμα δεδομένων για να δηλώσουν ονόματα στα δηλωθέντα δεδομένα και να είναι ευκολότερη η διαχείρισή τους. Η πρώτη χρήση μιας ετικέτας είναι η κύρια θέση εκκίνησης του προγράμματος, η οποία αποτελεί ειδική απαίτηση για τα προγράμματα σε συμβολικής γλώσσας RISC-V. Οι κανόνες για μια ετικέτα έχουν ως εξής:

- Πρέπει να ξεκινάει με ένα γράμμα ή «_» (κάτω παύλα)
- Μπορεί να ακολουθείται από γράμματα, αριθμούς ή «_» (κάτω παύλα)
- Πρέπει να τερματίζεται με «:» (άνω κάτω τελεία)
- Μπορεί να οριστεί μόνο μια φορά με το ίδιο όνομα

Μερικά παραδείγματα μιας ετικέτας περιλαμβάνουν:

α'. main:
 β'. Foo:
 γ'. _BAR:
 δ'. var12F:

Στους χαρακτήρες σε μια ετικέτα γίνεται διάκριση πεζών/κεφαλαίων (case-sensitive). Ως εκ τούτου, το Loop: και loop: είναι διαφορετικές ετικέτες.

A'.5.5 Πρότυπο Προγραμμάτων

Ο Κώδικας 8 ένα πολύ βασικό πρότυπο για τα προγράμματα συμβολικής γλώσσας RISC-V. Αυτό το γενικό πρότυπο θα χρησιμοποιηθεί για όλα τα προγράμματα και συνίσταται να το χρησιμοποιείτε κι εσείς ως βάση για τα προγράμματά. Μπορείτε να το βρείτε στο eClass του μαθήματος με όνομα **template.s** και προτείνεται να το χρησιμοποιείτε για την ανάπτυξη του κώδικα όλων των ασκήσεων του μαθήματος.



```
1 # Name and general description of the program
2
3 #----- text segment -----#
4
5 .text           # Program code goes in this section
6 .globl __start
7
8 __start:        # Execution starts here
9
10
11 # Your program code goes here...
12
13
14 li a7, 10      # Done, terminate program
15 ecall          # au revoir...
16
17
18 #----- data segment -----#
19
20 .data           # Data declarations go in this section
21
22
23 #----- End of File -----#
```

Κώδικας 8: Πρότυπο προγραμμάτων σε RISC-V

Παράρτημα Β'

Κλήσεις συστήματος (System Calls)

Το λειτουργικό σύστημα πρέπει να παρέχει ορισμένες βασικές υπηρεσίες σχετικές με λειτουργίες που ένα πρόγραμμα χρήστη δεν μπορεί να εκτελέσει εύκολα από μόνο του. Ορισμένα βασικά παραδείγματα περιλαμβάνουν τις λειτουργίες εισόδου (κυρίως από το πληκτρολόγιο) και εξόδου (κυρίως στην κονσόλα/οθόνη). Αυτές οι λειτουργίες ονομάζονται κλήσεις συστήματος (system calls). Ο προσωμοιωτής RARS παρέχει μια σειρά υπηρεσιών παρόμοιων με το λειτουργικό σύστημα, χρησιμοποιώντας την εντολή ecall (environment call).



B'.1 Υποστηριζόμενες κλήσεις συστήματος

Για να ζητήσει το πρόγραμμά σας μια συγκεκριμένη κλήση συστήματος από τον προσομοιωτή RARS, πρέπει αρχικά να τοποθετείται ο «κωδικός κλήσης» στον καταχωρητή a7. Εκτός αυτού και ανάλογα με τη συγκεκριμένη κλήση συστήματος που ζητείται, ενδέχεται να χρειαστούν πρόσθετες πληροφορίες και ορισματα που τοποθετούνται στους καταχωρητές a0-a6. Κάθε φορά που εκτελείται η εντολή ecall, εξετάζει την τιμή που περιέχει εκείνη τη στιγμή ο καταχωρητής a7 ώστε να εκτελέσει την αντίστοιχη κλήση συστήματος που υποδεικνύεται από τον συγκεκριμένο αριθμό. Ως εκ τούτου, πριν εκτελεστεί η εντολή ecall θα πρέπει να είμαστε βέβαιοι ότι ο καταχωρητής a7 περιέχει τον απαιτούμενο κωδικό της κλήσης συστήματος. Οι βασικές κλήσεις συστήματος του RISC-V παρατίθεται στον Πίνακα 12. Μπορείτε να ανατρέξετε στο Help του RARS για περισσότερες κλήσεις συστήματος.



Κλήση	Κωδικός (στον α7)	Είσοδος	Εξοδος
Εκτύπωση ακεραίου (32-bit) (<i>PrintInt</i>)	1	a0←ο ακέραιος που θα τυπωθεί	
Εκτύπωση αριθμού KY απλής ακρίβειας (32-bit) (<i>PrintFloat</i>)	2	fa0←ο 32-bit αριθμός KY που θα τυπωθεί	
Εκτύπωση αριθμού KY διπλής ακρίβειας (64-bit) (<i>PrintDouble</i>)	3	fa0←ο 64-bit αριθμός KY που θα τυπωθεί	
Εκτύπωση συμβολοσειράς στην κονσόλα (<i>PrintString</i>)	4	a0←η διεύθυνση έναρξης της συμβολοσειράς που τερματίζεται με NULL	
Ανάγνωση ακεραίου από την κονσόλα (32-bit) (<i>ReadInt</i>)	5		a0→ο 32-bit ακέραιος που δόθηκε από τον χρήστη
Ανάγνωση αριθμού KY απλής ακρίβειας (32-bit) (<i>ReadFloat</i>)	6		fa0→ο 32-bit αριθμός KY που δόθηκε από τον χρήστη
Ανάγνωση αριθμού KY διπλής ακρίβειας (64-bit) (<i>ReadDouble</i>)	7		fa0→ο 64-bit αριθμός KY που δόθηκε από τον χρήστη
Ανάγνωση συμβολοσειράς από την κονσόλα (<i>ReadString</i>)	8	a0←η διεύθυνση έναρξης του χώρου αποθήκευσης της συμβολοσειράς a1←το μέγεθος του χώρου	
Δέσμευση μνήμης στο σωρό (<i>Sbrk</i>)	9	a0←ο αριθμός των bytes που θα δεσμευτούν	a0→η διεύθυνση έναρξης της δεσμευμένης μνήμης
Τερματισμός προγράμματος με τιμή 0 (<i>Exit</i>)	10		
Εκτύπωση χαρακτήρα στην κονσόλα (<i>PrintChar</i>)	11	a0←ο χαρακτήρας που θα τυπωθεί (λαμβάνεται υπόψη μόνο το LSB)	
Ανάγνωση χαρακτήρα από την κονσόλα (<i>ReadChar</i>)	12		a0→ο χαρακτήρας που δόθηκε από τον χρήστη
Εκτύπωση ακεραίου στο δεκαεξαδικό (<i>PrintIntHex</i>)	34	a0←ο ακέραιος που θα τυπωθεί	
Εκτύπωση μη προσημασμένου ακεραίου (<i>PrintIntUnsigned</i>)	36	a0←ο ακέραιος που θα τυπωθεί	

Πίνακας 12: Λίστα με τις υποστηριζόμενες κλήσεις συστήματος



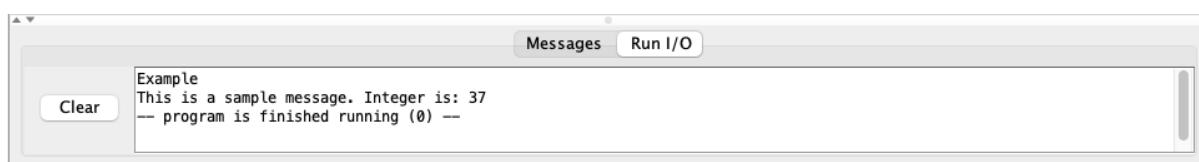
B'.2 Χρήση των κλήσεων συστήματος

Ο Κώδικας 9 παρουσιάζει τις βασικές κλήσεις συστήματος για την εκτύπωση μιας συμβολοσειράς και ενός ακεραίου.

```
1 # Example program to display a string and an integer
2 # It demonstrates the use of system calls
3
4 #----- text segment -----
5
6     .text                      # Program code goes in this section
7     .globl __start
8
9 __start:                      # Execution starts here
10    la    a0, msg               # Address of NULL terminated string
11    li    a7, 4                # Call code for PrintString
12    ecall                     # Environment call
13
14    li    a7, 1                # Call code for PrintInt
15    la    t0, number            # Value for print integer
16    lw    a0, 0(t0)             # Environment call
17    ecall                     # Environment call
18
19    li    a7, 10               # Done, terminate program
20    ecall                     # au revoir...
21
22 #----- data segment -----
23
24     .data
25     msg:      .ascii  "Example\n"
26                 .asciz  "This is a sample message. Integer is: "
27     number:   .word   37
28
29 #----- End of File -----#
```

Κώδικας 9: Παράδειγμα προγράμματος που εμφανίζει στην οθόνη (κονσόλα) μια συμβολοσειρά και έναν ακέραιο

Σημειώστε ότι σε αυτό το παράδειγμα ο ορισμός της συμβολοσειράς εξασφαλίζει τον τερματικό χαρακτήρα NULL (μέσω της οδηγίας .asciz) όπως απαιτείται από την κλήση συστήματος. Η έξοδος του παραδείγματος θα εμφανιστεί στο παράθυρο της κονσόλας όπως παρουσιάζεται στην Εικόνα 28.



Εικόνα 28: Η κονσόλα του RARS μετά το πέρας της εκτέλεσης του Κώδικα 9

Παράρτημα Γ'

**Εγκατάσταση και Εκτέλεση του προ-
σομοιωτή RARS**



Γ'.1 Εγκατάσταση του προσομοιωτή RARS

Ο προσομοιωτής RARS προσφέρεται ελεύθερα στο διαδίκτυο και θα τον χρησιμοποιήσουμε στα πλαίσια του μαθήματος της Αρχιτεκτονικής Υπολογιστών I. Είναι διαθέσιμος για Windows, Linux, και macOS δεδομένου ότι είναι γραμμένος σε Java. Ανεξάρτητα με το λειτουργικό σύστημα που χρησιμοποιείτε, αρκεί να κατεβάσετε και να εκτελέσετε τον προσομοιωτή. Μπορείτε να επισκεφτείτε το [GitHub](#) του RARS και να κατεβάσετε απευθείας από τη διεύθυνση που φιλοξενεί το εκτελέσιμο **αρχείο**. Στα πλαίσια του μαθήματος θα χρησιμοποιήσουμε την πιο πρόσφατη έκδοση του RARS που είναι η έκδοση 1.6.

Γ'.1.1 Εγκατάσταση της Java

Για να εγκαταστήσετε και να εκτελέσετε τον RARS θα χρειαστεί να έχετε εγκατεστημένη τη Java (τουλάχιστον Java 8) στο μηχάνημα σας. Αν δεν την έχετε ήδη, μπορείτε να επισκεφτείτε τον επίσημο ιστότοπο της **Oracle**, μέσω του συνδέσμου και να επιλέξτε το σωστό εκτελέσιμο ανάλογα το λειτουργικό σας σύστημα (.exe, .rpm, .deb, .dmg) για να την εγκαταστήσετε. Μετά τη λήψη του πακέτου, ακολουθήστε την τυπική διαδικασία εγκατάστασης για το συγκεκριμένο λειτουργικό σύστημα που χρησιμοποιείτε. Αυτό συνήθως περιλαμβάνει διπλό κλικ στο πακέτο εγκατάστασης που έχετε λάβει, και έπειτα ακολουθήστε τις οδηγίες εγκατάστασης. Ενδεχομένως να χρειαστείτε δικαιώματα διαχειριστή για να εκτελέσετε την εγκατάσταση. Επιπλέον, ορισμένες εγκαταστάσεις ενδέχεται να απαιτούν πρόσβαση στο Internet κατά τη διάρκεια της εγκατάστασης.



Γ'.2 Εκτέλεση του προσομοιωτή RARS

Στον Κώδικα 10 παρουσιάζεται ο τρόπος εκτέλεσης του προσομοιωτή RARS σε Περιβάλλον Linux. Η δεύτερη εντολή είναι απαραίτητη σε ορισμένες Linux διανομές για να τίθεται κατάλληλα η ανάλυση που χρησιμοποιεί η εφαρμογή του προσομοιωτή και να εμφανίζονται σωστά τα περιεχόμενα της εφαρμογής ειδικά σε οθόνες με υψηλή ανάλυση.

```
user@linux:~$ java -jar rars1_6.jar
user@linux:~$ java -Dsun.java2d.uiScale=3.0 -jar rars1_6.jar
```

Κώδικας 10: Εκτέλεση RARS σε Linux

Σε περίπτωση λειτουργικού macOS, εκτελέστε τις εντολές που περιγράφονται στον Κώδικα 11. Σε περιβάλλον macOS δεν έχει εντοπιστεί κάποια ιδιαιτερότητα σχετικά με την εκτέλεση του προσομοιωτή. Μπορείτε απλώς να εκτελέσετε την παραπάνω εντολή ή να κάνετε διπλό κλικ στο rars1_6.jar αρχείο του RARS που κατεβάσατε.

```
user@macos ~ % java -jar rars1_6.jar
```

Κώδικας 11: Εκτέλεση RARS σε macOS

Τέλος, σε περιβάλλον Windows, υποθέτουμε ότι έχετε κατεβάσει στο rars1_6.jar στον δίσκο σας και συγκεκριμένα στον C:\ Ανοίξτε την εφαρμογή Command Prompt (cmd) πληκτρολογώντας cmd στο πεδίο αναζήτησης των Windows. Στη γραμμή εντολών πληκτρολογήστε την εντολή του Κώδικα 12.

```
C:\> java -jar rars1_6.jar
```

Κώδικας 12: Εκτέλεση RARS σε Windows

Εναλλακτικά, μπορείτε να δημιουργήσετε μια συντόμευση (shortcut) στην επιφάνεια εργασίας και να εκτελέσετε τον RARS απευθείας πατώντας διπλό κλικ πάνω στο εικονίδιο του RARS (π.χ., rars1_6.jar).