Πανεπιστήμιο Θεσσαλίας
Τμήμα Πληροφορικής

Οργάνωση Η/Υ

Ενότητα 4η: Μικροπρογραμματισμένη Μονάδα Ελέγχου

Άσκηση 1:

Θεωρήστε την απλοποιημένη αρχιτεκτονική MIPS πολλαπλών κύκλων μηχανής ανά κύκλο εντολής με έλεγχο μικροπρογραμματισμένης λογικής. Προσθέστε σε αυτή την αρχιτεκτονική τις ακόλουθες εντολές έμμεσου άλματος με σύνδεση:

```
jmal $rt,off($rs)
jalm $rt,off($rs)
```

από τις οποίες η πρώτη εκτελεί άλμα σε διεύθυνση που διαβάζεται από τη μνήμη με σύνδεση σε κάποιον καταχωρητή, ενώ η δεύτερη εκτελεί άλμα σε διεύθυνση που περιέχεται σε καταχωρητή με σύνδεση στη μνήμη. Και για τις δύο εντολές, η διεύθυνση μνήμης, όπου είτε περιέχεται η διεύθυνση προορισμού του άλματος είτε αποθηκεύεται η παρούσα τιμή του μετρητή προγράμματος, υπολογίζεται σαν το άθροισμα του περιεχομένου του καταχωρητή $rs με τη μετατόπιση off της λέξης εντολής. Ο καταχωρητής $rt στην πρώτη εντολή αποτελεί τη θέση αποθήκευσης της τιμής του μετρητή προγράμματος, ενώ στη δεύτερη εντολή παρέχει τη διεύθυνση προορισμού του άλματος.
Να γράψετε το συντομότερο δυνατό μικροκώδικα. Εξηγήστε όποιες τροποποιήσεις απαιτούνται στη ΜΕΔ για την υποστήριξη των πιο πάνω εντολών, περιγράφοντας τη ροή πληροφορίας που συμβαίνει στη ΜΕΔ από την εκτέλεσή τους.

Απάντηση:

Για να προσθέσουμε στη ΜΕΔ MIPS πολλαπλών κύκλων μηχανής ανά κύκλο εντολής τις εντολές jmal και jalm, θα πρέπει να εξετάσουμε ποιες μικρολειτουργίες απαιτούνται γι' αυτές, αν οι μικρολειτουργίες αυτές είναι ήδη υλοποιημένες, και αν όχι, να τις προσθέσουμε με κατάλληλη τροποποίηση των δρόμων ροής πληροφορίας, ή κάποια άλλη τροποποίηση.
Είναι φανερό ότι δεν απαιτείται καμία τροποποίηση για την υποστήριξη των φάσεων ανάκλησης και αποκωδικοποίησης των νέων εντολών. Τα σήματα ελέγχου, και άρα ο μικροκώδικας, παραμένουν ως έχουν για τις φάσεις αυτές.

Μετά την αποκωδικοποίηση της εντολής jmal, οπότε θα έχει διαβαστεί και ο φάκελος καταχωρητών, πρέπει να προσπελάσουμε τη μνήμη για να διαβάσουμε τη διεύθυνση προορισμού του άλματος. Άρα, θα πρέπει πρώτα να υπολογίσουμε τη διεύθυνση προσπέλασης, κάτι που θα γίνει με το γνωστό τρόπο πρόσθεσης του περιεχομένου του καταχωρητή A με την προέκταση προσήμου της μετατόπισης off, αφού η διευθυνσιοδότηση μνήμης που γίνεται στην εντολή είναι η ίδια με τις κλασικές εντολές φόρτωσης και αποθήκευσης της αρχιτεκτονικής MIPS. Η μικροεντολή επομένως που ακολουθεί τη φάση αποκωδικοποίησης θα είναι ταυτόσημη με αυτή της φάσης εκτέλεσης των εντολών φόρτωσης και αποθήκευσης. Μάλιστα, για εξοικονόμηση μνήμης ελέγχου, μπορούμε να χρησιμοποιήσουμε την ίδια μικροεντολή, και να αυξήσουμε τις επιλογές άλματος μικροκώδικα από αυτήν προς τις φάσεις προσπέλασης μνήμης των εντολών φόρτωσης και αποθήκευσης ή προς την επόμενη φάση της εντολής jmal.
Στη συνέχεια μπορούμε να προσπελάσουμε τη μνήμη και να φέρουμε στον καταχωρητή DR τη διεύθυνση προορισμού του άλματος, από τη διεύθυνση μνήμης που λαμβάνεται από τον καταχωρητή C. Η αντίστοιχη μικροεντολή είναι παρόμοια με αυτή της φάσης προσπέλασης μνήμης της εντολής φόρτωσης. Και αυτή η μικροεντολή θα μπορούσε να είναι η ίδια με εκεί-

νη της εντολής φόρτωσης, κάτι που όμως αυξάνει την πολυπλοκότητα των αλμάτων στο μικροκώδικα, αφού θα πρέπει να προσθέσουμε ένα νέο άλμα από τη μικροεντολή προσπέλασης μνήμης της εντολής φόρτωσης προς την κατάλληλη επόμενη μικροεντολή.
Στον επόμενο κύκλο μηχανής της εντολής jmal, θα πρέπει να μεταφέρουμε το περιεχόμενο του DR στο μετρητή προγράμματος (PC). Ενώ μέχρι τώρα δεν κάναμε τροποποιήσεις στη ΜΕΔ, παρατηρούμε ότι εδώ θα χρειαστούμε ένα δρόμο μεταφοράς πληροφορίας εύρους 32 bits από τον DR προς τον PC, με κατάλληλη νέα επιλογή στην είσοδο του τελευταίου. Κάνουμε λοιπόν αυτή την προσθήκη στη ΜΕΔ, και προσθέτουμε και τη νέα επιλογή στον πολυπλέκτη που τροφοδοτεί τον PC, παρέχοντας στο μικροκώδικα αυτή την επιλογή.
Μέχρι τώρα έχουμε υλοποιήσει το ένα από τα δύο μέρη της εντολής jmal, το άλμα, και απομένει η σύνδεση. Παρατηρήστε ότι η υλοποίηση που κάναμε οδήγησε στο μικρότερο δυνατό αριθμό κύκλων μηχανής, εφ' όσον οι εξαρτήσεις μεταξύ των μικρολειτουργιών αποτρέπουν κάποια σύμπτυξή τους σε ακόμα μικρότερο αριθμό κύκλων.
Σύνδεση σημαίνει αποθήκευση της παλαιάς τιμής του PC, ώστε ο κώδικας να μπορεί με κατάλληλη άλλη εντολή άλματος να επιστρέψει στην εντολή που ακολουθεί την παρούσα. Έτσι, η σύνδεση δε μπορεί να γίνει μετά τον τελευταίο κύκλο μηχανής που περιγράψαμε, επειδή ο PC αλλάζει τιμή, και αν επιχειρήσουμε σύνδεση μετά από αυτόν, θα αποθηκεύσουμε στο φάκελο καταχωρητών λανθασμένη διεύθυνση. Η αποθήκευση επομένως θα πρέπει να γίνει σε κάποιον από τους κύκλους που έχουμε ήδη περιγράψει, και βέβαια μετά την αποκωδικοποίηση της εντολής. Αν αφήσουμε τις φάσεις υπολογισμού της διεύθυνσης προσπέλασης και της προσπέλασης μνήμης ως έχουν, θα προσθέσουμε τη σύνδεση στη τελευταία φάση που περιγράψαμε, αυτή της αποθήκευσης στον PC της διεύθυνσης προορισμού του άλματος. Θα πρέπει δε να προσθέσουμε στη ΜΕΔ το δρόμο από τον PC προς το φάκελο καταχωρητών, με κατάλληλη επιλογή στην είσοδο δεδομένων. Έτσι, στη φάση αυτή, ταυτόχρονα με τη μεταφορά του περιεχομένου του DR στον PC, έχουμε και τη μεταφορά της προηγούμενης τιμής του PC στο φάκελο καταχωρητών. Οι δύο αποθηκεύσεις, στον PC και στο φάκελο καταχωρητών, δεν αλληλεπιδρούν, διότι συμβαίνουν ταυτόχρονα στο τέλος του κύκλου μηχανής.

Όσο αφορά την εντολή jalm τώρα, πρέπει να παρατηρήσουμε ότι δε μπορούμε να μεταφέρουμε στο μετρητή προγράμματος τη διεύθυνση προορισμού του άλματος από τον καταχωρητή B όπου αποθηκεύεται από το φάκελο καταχωρητών, πριν η προηγούμενη τιμή του PC σταλεί στη μνήμη για σύνδεση. Επειδή δε η προσπέλαση της μνήμης προϋποθέτει τον υπολογισμό της διεύθυνσης προσπέλασης, γίνεται φανερό ότι η πρώτη φάση της εντολής jalm μετά την αποκωδικοποίησή της θα είναι και πάλι ο υπολογισμός της διεύθυνσης αυτής με την πρόσθεση στην ΑΛΜ του περιεχομένου του A με την προέκταση προσήμου της μετατόπισης off. Άρα η μικροεντολή που ακολουθεί τη φάση αποκωδικοποίησης θα είναι και πάλι ταυτόσημη ή η ίδια με αυτή της φάσης εκτέλεσης των εντολών φόρτωσης και αποθήκευσης.
Μετά τον υπολογισμό της διεύθυνσης προσπέλασης, μπορεί να γίνει η προσπέλαση της μνήμης για τη σύνδεση, την αποθήκευση δηλαδή στη μνήμη του περιεχομένου του PC. Γι' αυτή τη μικρολειτουργία, χρειαζόμαστε ένα δρόμο από τον PC προς την είσοδο δεδομένων της μνήμης, με προσθήκη πολυπλέκτη πριν από την είσοδο αυτή, και κατάλληλων σημάτων ελέγχου για την επιλογή μεταξύ του B, που αποτελούσε μέχρι τώρα τη μοναδική πηγή δεδομένων προς τη μνήμη, και του PC. Το πεδίο MEM του μικροκώδικα θα πρέπει τώρα να καθορίζει και την επιλογή αυτή για προσπελάσεις εγγραφής της μνήμης.
Από τη στιγμή που έχει γίνει η σύνδεση για την εντολή jalm, ο PC μπορεί να πάρει τη νέα τιμή από τον καταχωρητή B. Φυσικά, για τη μικρολειτουργία αυτή απαιτείται και ο αντίστοιχος δρόμος από τον B προς τον PC, με ανάλογη επιλογή στην είσοδο του PC. Επειδή μια νέα τιμή εμφανίζεται στο τέλος του κύκλου μηχανής, συμφέρει η μικρολειτουργία αυτή να εκτελείται ταυτόχρονα με την αποθήκευση στη μνήμη της παρούσας τιμής του PC, ώστε και γι' αυτή την εντολή να επιτυγχάνεται ο ελάχιστος αριθμός κύκλων μηχανής. Με τους δύο δρόμους που προσθέσαμε, από τον PC προς την είσοδο δεδομένων της μνήμης και από τον καταχωρητή B προς τον PC, μπορούμε να υλοποιήσουμε μετά τον υπολογισμό της διεύθυνσης προσπέλασης στην ίδια φάση τόσο τη μεταφορά του περιεχομένου του PC προς τη μνήμη, όσο και τη μεταφορά του περιεχομένου του B προς τον PC. Όπως και προηγουμένως, οι δύο αποθηκεύσεις, στη μνήμη και στον PC, συμβαίνουν ταυτόχρονα και δεν αλληλεπιδρούν.

Βέβαια, για να είναι σωστή η αποθήκευση στον PC, πρέπει να επαληθεύσουμε ότι ο B συνεχίζει και σε αυτή τη φάση να περιέχει τη διεύθυνση προορισμού άλματος. Επειδή ο καταχωρητής B δεν έχει επίτρεψη εγγραφής, γράφεται σε κάθε κύκλο μηχανής, οπότε η τιμή που μας ενδιαφέρει θα έχει αποθηκευτεί σε αυτόν τον αμέσως προηγούμενο κύκλο. Εφόσον ούτε ο IR έχει μεταβληθεί, ούτε έχει μεσολαβήσει εγγραφή στο φάκελο καταχωρητών, η ανάγνωση του τελευταίου που δίνει τιμή στον B θα επιστρέψει όντως τη διεύθυνση προορισμού άλματος.

Σύμφωνα με τα παραπάνω, ο συνολικός μικροκώδικας των εντολών jmal και jalm απαιτεί 2 κοινές μικροεντολές για την ανάκληση και την αποκωδικοποίηση, και μετά άλλες 5 μικροεντολές, 3 για την πρώτη και 2 για τη δεύτερη εντολή, όπως φαίνεται στον ακόλουθο πίνακα:

Ετικέτα	**Άλμα**	**ΑΛΜ**	**ΑΛΜ1**	**ΑΛΜ2**	**ΦΚ**	**ΜΕΜ**	**PC=**
Fetch:	seq	add	PC	4		readIR	ΑΛΜ
	disp	add	PC	προ(IR)$_{<<2}$			
…							
JMAL:	seq	add	A	προ(IR)			
	seq					readDR	
	j Fetch				writePC		DR
JALM:	seq	add	A	προ(IR)			
	j Fetch					writePC	B

όπου οι ετικέτες JMAL και JALM δείχνουν την πρώτη μικροεντολή των εντολών jmal και jalm, αντίστοιχα, που ακολουθεί τις δύο πρώτες φάσεις, οι οποίες είναι κοινές για όλες τις εντολές. Με το όνομα “writePC” στο πεδίο ΦΚ δώσαμε συμβολικά τις τιμές των σημάτων ελέγχου για επιλογή του PC στην είσοδο δεδομένων του φακέλου καταχωρητών, για επιλογή του πεδίου rt της λέξης εντολής στην είσοδο διεύθυνσης αποθήκευσης του φακέλου καταχωρητών, και για ενεργοποίηση του σήματος επίτρεψης εγγραφής του φακέλου καταχωρητών. Με το όνομα “writePC” στο πεδίο MEM δώσαμε συμβολικά στο μικροκώδικα τις τιμές των σημάτων για επιλογή του PC στην είσοδο δεδομένων της μνήμης, και για την ενεργοποίηση του σήματος εγγραφής της μνήμης. Αντίστοιχο όνομα “writeB” για το ίδιο πεδίο πρέπει να αντικαταστήσει το όνομα “write” στην εντολή αποθήκευσης.

Άσκηση 2:

Στην προσπάθειά μας για εξοικονόμηση χώρου στη μνήμη ελέγχου, κωδικοποιούμε τις μικρολειτουργίες σε διαφορετικά πεδία της μικροεντολής. Η κωδικοποίηση αυτή γίνεται με τους εξής περιορισμούς:
(α) Οι μικρολειτουργίες που εμφανίζονται στην ίδια μικροεντολή κωδικοποιούνται σε διαφορετικά πεδία, ώστε να μπορούν να εκτελούνται ταυτόχρονα.
(β) Κάθε μικρολειτουργία κωδικοποιείται σε ένα και μόνο πεδίο, το ίδιο για όλες τις μικροεντολές στις οποίες αυτή εμφανίζεται.
(γ) Κάθε πεδίο κωδικοποιεί και την κενή μικρολειτουργία.
Προτείνετε μια μέθοδο κωδικοποίησης των μικρολειτουργιών μιας μονάδας επεξεργασίας δεδομένων, η οποία να ικανοποιεί τους πιο πάνω περιορισμούς, και ταυτόχρονα να μεγιστοποιεί την εξοικονόμηση χώρου στη μνήμη ελέγχου.
Παρουσιάστε ένα παράδειγμα εφαρμογής της μεθόδου σας.

Απάντηση:

Έστω n ο συνολικός αριθμός των μικρολειτουργιών της μονάδας επεξεργασίας δεδομένων και k ο μέγιστος αριθμός μικρολειτουργιών σε μια μικροεντολή. Για να μην περιορίσουμε τη μέγιστη παραλληλία στην εκτέλεση μικρολειτουργιών, θα πρέπει σύμφωνα με τον περιορισμό (α) ο αριθμός πεδίων της μικροεντολής να είναι τουλάχιστον ίσος με k.

Η μοναδικότητα εμφάνισης κάθε μικρολειτουργίας σε κάποιο πεδίο της μικροεντολής, γεγονός που ικανοποιεί τον περιορισμό (β), επιτυγχάνεται με την πράξη του διαμερισμού. Υποθέτουμε έτσι ότι διαμερίζουμε τις n μικρολειτουργίες σε m σύνολα για κωδικοποίηση σε m πεδία της μικροεντολής με τέτοιον τρόπο, ώστε κανένα ζεύγος μικρολειτουργιών του ίδιου συνόλου να μην εμφανίζεται στην ίδια μικροεντολή, σύμφωνα με τον περιορισμό (α). Εάν ο αριθμός των μικρολειτουργιών σε ένα από τα σύνολα της διαμέρισης είναι q_i, τότε για να κωδικοποιηθούν αυτές οι μικρολειτουργίες σε ένα πεδίο, απαιτούνται $C_i = \lceil \log_2(q_i+1) \rceil$ δυαδικά ψηφία, όπου το 1 προστίθεται για να συμπεριληφθεί η κενή μικρολειτουργία, όπως μας επιβάλλει ο περιορισμός (γ). Το σύνολο των ψηφίων που απαιτούνται για μια συγκεκριμένη διαμέριση είναι:

$$C = \sum_{i=1}^{m} C_i = \sum_{i=1}^{m} \lceil \log_2(q_i+1) \rceil$$

Μία μέθοδος που μας δίνει τη διαμέριση εκείνη που ελαχιστοποιεί το C είναι η εξαντλητική μέθοδος. Αυτή βρίσκει όλες τις δυνατές διαμερίσεις των n μικρολειτουργιών σε m σύνολα που ικανοποιούν τον περιορισμό (α), και στη συνέχεια υπολογίζει την τιμή C για κάθε διαμέριση, ώστε να βρει τη ζητούμενη. Σύμφωνα με τα παραπάνω, και λόγω της ικανοποίησης του περιορισμού (α), θα έχουμε $m \geq k$. Άρα η εξαντλητική μέθοδος θα δοκιμάσει όλες τις δυνατές διαμερίσεις, με m=k, k+1, …, n-1, n.

Μια πιο πρακτική μέθοδος περιγράφεται παρακάτω και ολοκληρώνεται σε τέσσερα βήματα:

1. Εύρεση όλων των διαφορετικών συνόλων μικρολειτουργιών S_i, καθένα από τα οποία περιλαμβάνει το μέγιστο αριθμό μικρολειτουργιών *συμβιβαστών* μεταξύ τους, δηλαδή μικρολειτουργιών που δε μπορούν να ανήκουν στην ίδια μικροεντολή.
2. Εύρεση όλων των διαφορετικών συνδυασμών B_j των συνόλων S_i, έτσι ώστε κάθε συνδυασμός B_j να έχει τον ελάχιστο αριθμό συνόλων S_i και κάθε μικρολειτουργία να υπάρχει σε ένα τουλάχιστον σύνολο S_i του συνδυασμού B_j. Ένας τέτοιος συνδυασμός ονομάζεται *ελάχιστη κάλυψη*.
3. Προσδιορισμός όλων των δυνατών τρόπων διαγραφής μικρολειτουργιών από τα σύνολα S_i ενός συνδυασμού B_j, έτσι ώστε κάθε συνδυασμός συνόλων να περιέχει κάθε μικρολειτουργία ακριβώς μία φορά. Έτσι προκύπτουν νέοι συνδυασμοί M_t.
4. Εκτίμηση του κόστους C_t κάθε συνδυασμού M_t και επιλογή του συνδυασμού με το ελάχιστο κόστος.

Το βήμα 1 εκτελείται ως εξής:
Αρχικά υποθέτουμε ότι έχουμε n σύνολα, τα οποία αποτελούν τα σύνολα του επιπέδου 1. Κάθε σύνολο σ' αυτό το επίπεδο περιέχει μία ακριβώς μικρολειτουργία.
Εάν έχουμε τα σύνολα του επιπέδου i, τότε τα σύνολα του επιπέδου i+1 βρίσκονται με τον ακόλουθο τρόπο: Σε κάθε σύνολο του προηγούμενου επιπέδου προσθέτουμε μια νέα μικρολειτουργία, έτσι ώστε τα νέα σύνολα να εξακολουθήσουν να περιέχουν συμβιβαστές μικρολειτουργίες. Δοκιμάζουμε κάθε διαθέσιμη μικρολειτουργία, και εάν βρήκαμε κάποια νέα σύνολα, διαγράφουμε όλα τα υποσύνολα αυτών.
Το βήμα 2 εκτελείται με τη μέθοδο ελαχιστοποίησης λογικής συνάρτησης ως εξής:
Σχεδιάζουμε έναν πίνακα όπου οι γραμμές αντιστοιχούν στα σύνολα S_i και οι στήλες στις μικρολειτουργίες. Στη συνέχεια συμπληρώνουμε τον πίνακα σημειώνοντας για κάθε γραμμή τις μικρολειτουργίες που περιέχονται στο αντίστοιχο σύνολο. Τέλος βρίσκουμε κάθε ελάχιστο συνδυασμό συνόλων που να καλύπτουν όλες τις μικρολειτουργίες. Στη διαδικασία αυτή, κάθε συνδυασμός B_j περιέχει υποχρεωτικά τα σύνολα που έχουν αποκλειστικότητα στην κάλυψη κάποιων μικρολειτουργιών.
Το βήμα 3 εκτελείται εξαντλητικά για την εύρεση των νέων συνδυασμών M_t.

Ας θεωρήσουμε το επόμενο παράδειγμα, στο οποίο 6 μικροεντολές μ_1, μ_2, …, μ_6 ενεργοποιούν 7 μικρολειτουργίες c_1, c_2, …, c_7, με τον ακόλουθο τρόπο:

μ_1:c_1, c_4

μ_2:c_2, c_3, c_7
μ_3:c_1, c_2, c_3, c_4
μ_4:c_3, c_4, c_5
μ_5: c_1, c_6
μ_6: c_2, c_5

Θα ακολουθήσουμε την παραπάνω μέθοδο για την εύρεση της βέλτιστης κωδικοποίησης της μικροεντολής σε πεδία.

Βήμα 1:
Επίπεδο 1: {~~c_1~~}, {~~c_2~~}, {~~c_3~~}, {~~c_4~~}, {~~c_5~~}, {~~c_6~~}, {~~c_7~~}
Επίπεδο 2: {~~c_1, c_5~~}, {~~c_1, c_7~~}, {c_2, c_6}, {c_3, c_6}, {~~c_4, c_6~~}, {~~c_4, c_7~~}, {~~c_5, c_6~~}, {~~c_5, c_7~~}, {~~c_6, c_7~~}
Επίπεδο 3: {c_1, c_5, c_7}, {c_4, c_6, c_7}, {c_5, c_6, c_7}
Για παράδειγμα, από το σύνολο {c_1} μπορούμε να πάρουμε τα σύνολα {c_1, c_2}, {c_1, c_3}, {c_1, c_4}, {c_1, c_5}, {c_1, c_6} και {c_1, c_7}. Από αυτά τα 6 σύνολα, μόνο τα {c_1, c_5} και {c_1, c_7} περιέχουν συμβιβαστές μικρολειτουργίες, γι' αυτό τα υπόλοιπα απορρίπτονται. Παρόμοια, από το σύνολο {c_1, c_5} μπορούμε να πάρουμε τα σύνολα {c_1, c_5, c_6} και {c_1, c_5, c_7}. Όμως στο πρώτο από αυτά, οι μικρολειτουργίες c_1 και c_6 απαντώνται στη μικροεντολή μ_5 κι επομένως είναι μη συμβιβαστές. Επιλέγουμε έτσι το δεύτερο.

Βήμα 2:
Ο πίνακας συνόλων/μικρολειτουργιών θα είναι:

Σύνολα	**Μικρολειτουργίες**						
	c_1	c_2	c_3	c_4	c_5	c_6	c_7
$S_1 = \{c_2, c_6\}$		x				x	
$S_2 = \{c_3, c_6\}$			x			x	
$S_3 = \{c_1, c_5, c_7\}$	x				x		x
$S_4 = \{c_4, c_6, c_7\}$				x		x	x
$S_5 = \{c_5, c_6, c_7\}$					x	x	x

Εύκολα μπορούμε να διαπιστώσουμε ότι τα σύνολα S_1, S_2, S_3 και S_4 λαμβάνονται υποχρεωτικά στους ζητούμενους συνδυασμούς B_j, διότι είναι τα μόνα που καλύπτουν τις μικρολειτουργίες c_2, c_3, c_1 και c_4 αντίστοιχα. Επειδή οι υπόλοιπες μικρολειτουργίες καλύπτονται από τα σύνολα αυτά, κι επειδή ζητούμε τις *ελάχιστες* καλύψεις, θα έχουμε μόνο έναν αποδεκτό συνδυασμό B_1, ο οποίος θα περιλαμβάνει τα 4 πρώτα σύνολα:

$B_1 = \{S_1, S_2, S_3, S_4\}$

Ο εναλλακτικός συνδυασμός $B_2 = \{S_1, S_2, S_3, S_4, S_5\}$ δεν αποτελεί ελάχιστη κάλυψη, εφ' όσον $B_1 \subset B_2$.

Βήμα 3,4:
Αναγράφουμε τα 4 σύνολα και παίρνουμε όλους τους δυνατούς συνδυασμούς υποσυνόλων, έτσι ώστε να πάρουμε νέους συνδυασμούς που να αποτελούν τις διαμερίσεις M_t των 7 μικρολειτουργιών. Ταυτόχρονα, υπολογίζουμε το κόστος C_t κάθε διαμέρισης M_t και το συμπληρώνουμε στον πίνακα. Παρατηρούμε ότι υπάρχουν 4 διαμερίσεις με ελάχιστο κόστος ίσο με 6 ψηφία ανά μικροεντολή:

Διαμέριση	**B_1**				**Κόστος**
	S_1	S_2	S_3	S_4	
M_1	$\{c_2\}$	$\{c_3\}$	$\{c_1, c_5\}$	$\{c_4, c_6, c_7\}$	6
M_2	$\{c_2\}$	$\{c_3\}$	$\{c_1, c_5, c_7\}$	$\{c_4, c_6\}$	6
M_3	$\{c_2\}$	$\{c_3, c_6\}$	$\{c_1, c_5\}$	$\{c_4, c_7\}$	7
M_4	$\{c_2\}$	$\{c_3, c_6\}$	$\{c_1, c_5, c_7\}$	$\{c_4\}$	6
M_5	$\{c_2, c_6\}$	$\{c_3\}$	$\{c_1, c_5\}$	$\{c_4, c_7\}$	7
M_6	$\{c_2, c_6\}$	$\{c_3\}$	$\{c_1, c_5, c_7\}$	$\{c_4\}$	6

Αν επιλέξουμε τη διαμέριση M_1 για την κωδικοποίηση που ζητάμε, θα έχουμε 4 πεδία. Το πρώτο θα κωδικοποιεί τη μικρολειτουργία c_2 με το ψηφίο 1 και την κενή με το ψηφίο 0. Το δεύτερο θα κωδικοποιεί τη μικρολειτουργία c_3 με το ψηφίο 1 και την κενή με το ψηφίο 0. Το τρίτο θα κωδικοποιεί τη μικρολειτουργία c_1 με τα δύο ψηφία 01, την c_5 με τα δύο ψηφία 10 και την κενή με τα ψηφία 00. Το τελευταίο πεδίο κωδικοποιεί τη μικρολειτουργία c_4 με τα δύο ψηφία 01, την c_6 με τα ψηφία 10, την c_7 με τα ψηφία 11 και την κενή με τα ψηφία 00.

Οι μικροεντολές που μας δίνονται θα κωδικοποιηθούν επομένως ως εξής:

μ_1:	0	0	01	01
μ_2:	1	1	00	11
μ_3:	1	1	01	01
μ_4:	0	1	10	01
μ_5:	0	0	01	10
μ_6:	1	0	10	00

Υποθέτουμε ότι τους κωδικούς πεδίων μικρολειτουργιών συνοδεύουν τα κατάλληλα πεδία άλματος, τα οποία δε μας απασχολούν στην άσκηση αυτή.

Η παραπάνω μέθοδος βρίσκει τη βέλτιστη κωδικοποίηση μικρολειτουργιών για εξοικονόμηση χώρου στη μνήμη ελέγχου και μπορεί να επεκταθεί προς δύο κατευθύνσεις:
Α. Εάν έχουμε άνω φράγμα στον αριθμό ψηφίων της κάθε μικροεντολής, θα πρέπει πιθανά να τροποποιήσουμε τον περιορισμό (α), ώστε να επιτρέπεται στο ίδιο πεδίο κωδικοποίηση μικρολειτουργιών από την ίδια μικροεντολή. Το αποτέλεσμα αυτής της τροποποίησης είναι η απώλεια παραλληλίας στην εκτέλεση, και αύξηση του αριθμού των μικροεντολών. Στην περίπτωση αυτή, ο υπολογισμός του κόστους θα πρέπει να συνυπολογίζει τον αριθμό των μικροεντολών, και η μέθοδος θα πρέπει να μας δίνει κάποιο συνδυασμό μέγιστης εξοικονόμησης χώρου με ελάχιστο αριθμό μικροεντολών.
Β. Εάν έχουμε περιορισμούς στους χρόνους εκτέλεσης των μικρολειτουργιών, είναι πιθανό να μη μπορούμε να επιτρέψουμε κάποιες μικρολειτουργίες να ανήκουν σε πεδία με μεγάλη κωδικοποίηση (δηλαδή κωδικοποίηση με πολλά ψηφία). Αυτό συμβαίνει επειδή ο χρόνος αποκωδικοποίησης αυξάνεται με μεγαλύτερη κωδικοποίηση, και μπορεί για αργές μικρολειτουργίες να αυξήσει το χρόνο κύκλου μηχανής. Επομένως, θα τροποποιήσουμε το βήμα 1, έτσι ώστε η πρόσθεση μιας μικρολειτουργίας σε κάποιο σύνολο να μη γίνεται, όταν το μέγεθος κωδικοποίησης γίνεται έτσι απαγορευτικό για αυτήν ή για κάποια από όσες είναι ήδη στο σύνολο.

Άσκηση 3:

Η αρχιτεκτονική 80x86 διαθέτει την εντολή MOVS, η οποία μεταφέρει μια συμβολοσειρά (string) από μια περιοχή μνήμης σε κάποια άλλη.
Έστω ότι θέλετε να διαθέσετε την πιο πάνω εντολή σε μικροπρογραμματισμένη ΜΕΔ MIPS, με τη μορφή:

```
movs ($rt), off($rs)
```

όπου οι καταχωρητές $rs και $rt περιέχουν τις αρχικές διευθύνσεις ανάγνωσης και εγγραφής της συμβολοσειράς αντίστοιχα, με τον $rs να δέχεται και μια σταθερά μετατόπισης off.
Γράψτε έναν όσο πιο σύντομο μικροκώδικα μπορείτε για την υλοποίηση της παραπάνω εντολής, τροποποιώντας τη ΜΕΔ, ώστε να υποστηρίζει τη ροή πληροφορίας που χρειάζεστε. Δε μπορείτε όμως να προσθέσετε νέους καταχωρητές ειδικού σκοπού, ούτε να τροποποιήσετε τις εισόδους στους ήδη υπάρχοντες. Η είσοδος διευθύνσεων της ΜΔΜ να γίνεται μόνο από τον C. Τέλος, αποτίμηση συνθηκών να γίνεται μόνο στην ΑΛΜ.
Υποθέστε ότι η συμβολοσειρά τελειώνει με το χαρακτήρα '\0'.
Με την ολοκλήρωση της εντολής, οι $rs και $rt μπορούν να έχουν αλλαγμένη τιμή.

Απάντηση:

Η εντολή μεταφοράς συμβολοσειράς μεταφέρει κάθε ψηφιολέξη αυτής, μέχρι να βρει τη σταθερά 0. Αν είχαμε τη δυνατότητα να σχεδιάσουμε τη ΜΕΔ από την αρχή, ώστε να επιλέξουμε τις υπομονάδες που θα μας έδιναν την καλύτερη δυνατή υλοποίηση, θα επιλέγαμε: (α) δύο ανεξάρτητους καταχωρητές δείκτες, οι οποίοι να αποθηκεύουν τις διευθύνσεις ανάγνωσης και εγγραφής της συμβολοσειράς με δυνατότητα αυτόματης αύξησης της τιμής τους κατά 1, (β) δυνατότητα ελέγχου μηδενικής τιμής πάνω στην έξοδο της μνήμης, που σε κάθε ανάγνωση φορτώνει ένα χαρακτήρα της συμβολοσειράς.
Επειδή δεν έχουμε την ευχέρεια ευρείας παρέμβασης στην οργάνωση της ΜΕΔ, αλλά ουσιαστικά μόνο δυνατότητα τροποποίησης των υπαρχόντων δρόμων ροής πληροφορίας, θα πρέπει λίγο-πολύ να στηριχτούμε στις μικρολειτουργίες που υλοποιούν τις κοινές εντολές MIPS. Έτσι, αντί να προσπαθήσουμε να γράψουμε απ' ευθείας το ζητούμενο μικροκώδικα, είναι ευκολότερο να ξεκινήσουμε από τον αντίστοιχο κώδικα σε συμβολική γλώσσα με κοινές εντολές MIPS και από αυτόν να προχωρήσουμε βήμα-βήμα προς τον τελικό μικροκώδικα.

Επομένως, θα ακολουθήσουμε την πιο κάτω διαδικασία για την εύρεση του ζητούμενου μικροκώδικα:
Α. Ανάπτυξη κώδικα συμβολικής γλώσσας MIPS που υλοποιεί την αντίστοιχη μεταφορά.
Β. Εύρεση μικροκώδικα που αντιστοιχεί σε μετάφραση αυτού του κώδικα στις αντίστοιχες μικροεντολές.
Γ. Ανάλυση της ροής πληροφορίας που γίνεται κατά την εκτέλεση της εντολής και εισαγωγή τροποποιήσεων στη ΜΕΔ για να υποστηρίζει αυτή τη ροή, λαμβάνοντας υπ' όψη τους περιορισμούς που μας δίνονται.
Δ. Σύμπτυξη του παραπάνω μικροκώδικα με δεδομένους τους νέους δρόμους ροής πληροφορίας, ώστε να πάρουμε έναν όσο το δυνατό συντομότερο τελικό μικροκώδικα.

Αναλυτικά:

Α. Ο κώδικας σε συμβολική γλώσσα MIPS που μεταφέρει μια συμβολοσειρά από τη περιοχή μνήμης που αρχίζει στη διεύθυνση που περιέχεται στον καταχωρητή $rs με μετατόπιση off στην περιοχή μνήμης που αρχίζει στη διεύθυνση που περιέχεται στον καταχωρητή $rt, μπορεί να είναι ο ακόλουθος:

```
        lb    $tmp, off($rs)
        beq   $tmp, $0, End
Loop:   sb    $tmp, 0($rt)
        addi  $rs, $rs, 1
        addi  $rt, $rt, 1
        lb    $tmp, off($rs)
        bne   $tmp, $0, Loop
End:    sb    $tmp, 0($rt)
```

όπου ο καταχωρητής $tmp είναι κάποιος διαθέσιμος καταχωρητής γενικού σκοπού, τον οποίο χρησιμοποιούμε για την προσωρινή αποθήκευση ενός χαρακτήρα της συμβολοσειράς.
Ο παραπάνω κώδικας ξεκινάει με την ανάγνωση του πρώτου χαρακτήρα της συμβολοσειράς από την αντίστοιχη διεύθυνση ανάγνωσης. Στη συνέχεια ελέγχει αν αυτός έχει τιμή 0 (ή ισοδύναμα το χαρακτήρα '\0'), το οποίο θα σημαίνει ότι η συμβολοσειρά είναι η "" – δηλαδή η κενή συμβολοσειρά. Αν ο έλεγχος αποδειχτεί αληθής, ο κώδικας εκτελεί άλμα στην εντολή που βρίσκεται στην ετικέτα End, διαφορετικά συνεχίζεται με την αποθήκευση του χαρακτήρα στην αντίστοιχη διεύθυνση αποθήκευσης. Οι καταχωρητές $rs και $rt αυξάνονται κατά 1, ώστε να πάμε στις επόμενες διευθύνσεις ανάγνωσης και αποθήκευσης αντίστοιχα, και διαβάζουμε τον επόμενο χαρακτήρα της συμβολοσειράς. Τέλος, επανελέγχουμε για μηδενική τιμή του χαρακτήρα, και σε αρνητικό συμπέρασμα – δηλαδή μη μηδενική τιμή – επαναλαμβάνουμε τη διαδικασία με την εντολή που βρίσκεται στην ετικέτα Loop. Διαφορετικά, ο κώδικας μεταφοράς ολοκληρώνεται με την έξοδο από το βρόχο και εκτέλεση της εντολής στην ετικέτα End, η οποία είναι μια ακόμα εντολή αποθήκευσης που αποθηκεύει τον τελικό χαρακτήρα '\0'.

Παρατηρήστε ότι οι καταχωρητές $rs και $rt χρησιμοποιούνται σα δείκτες και αλλάζουν τιμή μεταξύ διαδοχικών επαναλήψεων, κάτι που μας επιτρέπει η εκφώνηση.

Β. Για την εύρεση του μικροκώδικα που προκύπτει από τη μετάφραση καθεμιάς από τις παραπάνω εντολές, θα θεωρήσουμε τη μορφή της μικροεντολής MIPS που είδαμε στο μάθημα.
Πιο συγκεκριμένα, στη μορφή αυτή οι μικρολειτουργίες είναι χωρισμένες σε ομάδες, με κάθε ομάδα να αναφέρεται είτε στις διαφορετικές μικρολειτουργίες που εκτελούνται στο ίδιο σημείο κάποιας υπομονάδας της ΜΕΔ, είτε στις διαφορετικές επιλογές κάποιας εισόδου μιας υπομονάδας της ΜΕΔ. Κάθε ομάδα μικρολειτουργιών κωδικοποιείται στο ίδιο πεδίο της μικροεντολής.
Η κωδικοποίηση αυτή δεν εξασφαλίζει τη μέγιστη εξοικονόμηση χώρου στη μνήμη ελέγχου, αλλά είναι απλή και επιτυγχάνει πλήρη παραλληλία στην εκτέλεση των μικρολειτουργιών μιας μικροεντολής, εφ' όσον οι μικρολειτουργίες κάθε ομάδας είναι εξ ορισμού συμβιβαστές (η μια αποκλείει την άλλη).
Ο συνολικός μικροκώδικας που προκύπτει από τη μετάφραση κάθε εντολής ξεχωριστά – επαναλαμβάνοντας εντολές όπου χρειάζεται – είναι:

Εντολή	Ετικέτα	Άλμα	ΑΛΜ	ΑΛΜ1	ΑΛΜ2	ΦΚ	MEM	PC
	Fetch:	seq	add	PC	4		readIR	ΑΛΜ
		disp	add	PC	προ(IR)$_{<<2}$			
lb $tmp,off($rs)	LB:	seq	add	A	προ(IR)			
		seq					readbDR	
		j Fetch				writeDR		
beq $tmp,$0,End	BEQ:	j Fetch	sub	A	B			C(Z==1)
Loop: sb $tmp,0($rt)	SB:	seq	add	A	προ(IR)			
		j Fetch					writeb	
addi $rs,$rs,1	ADDI:	seq	add	A	προ(IR)			
		j Fetch				writeC		
addi $rt,$rt,1	ADDI:	seq	add	A	προ(IR)			
		j Fetch				writeC		
lb $tmp,off($rs)	LB:	seq	add	A	προ(IR)			
		seq					readbDR	
		j Fetch				writeDR		
bne $tmp,$0,Loop	BNE:	j Fetch	sub	A	B			C(Z==0)
End: sb $tmp,0($rt)	SB:	seq	add	A	προ(IR)			
		j Fetch					writeb	

όπου οι δύο πρώτες μικροεντολές αντιστοιχούν στις κοινές φάσεις ανάκλησης και αποκωδικοποίησης των εντολών MIPS. Στις μικροεντολές με προσπέλαση στη μνήμη έχουμε προσθέσει στις αντίστοιχες μικρολειτουργίες την ένδειξη b για το μέγεθος δεδομένου.

Γ. Για να προχωρήσουμε προς την εντολή movs, πρέπει να κατανοήσουμε ότι ο τελικός μικροκώδικας θα αντιστοιχεί σε μία μόνο εντολή, οπότε θα περνάει μία μοναδική φορά από τις φάσεις ανάκλησης και αποκωδικοποίησης. Από εκεί και πέρα, μια σειρά από μικροεντολές θα εκτελούν την movs, με τη βοήθεια των υπομονάδων και καταχωρητών της ΜΕΔ, με τον περιορισμό ότι η τιμή του IR θα παραμένει σταθερή, όσο είμαστε στην ίδια εντολή, όπως θα παραμένει σταθερή και η τιμή του PC, δεδομένου ότι η movs δεν είναι εντολή άλματος. Άρα ο παραπάνω μικροκώδικας θα τροποποιηθεί με μια διαδικασία ενοποίησης, στην οποία (α) θα αφαιρεθεί το άλμα στη διεύθυνση Fetch από όλες τις μικροεντολές πλην της τελευταίας που εκτελείται, και (β) κάθε άλμα σε επίπεδο εντολής που προηγουμένως τροποποιούσε τον PC πρέπει να γίνει άλμα σε επίπεδο μικροεντολής που τροποποιεί μόνο τον μPC.
Σαν αποτέλεσμα των παραπάνω, όλες οι μικρολειτουργίες που χρησιμοποιούν τον IR πρέπει να διαβάζουν απ' αυτόν τα πεδία της τελικής εντολής movs.
Πιο συγκεκριμένα, οι μόνοι καταχωρητές γενικού σκοπού που μπορούν να προσπελαστούν είναι αυτοί που δηλώνονται στα πεδία rs και rt του IR, δηλαδή ο καταχωρητής με διεύθυνση IR^{25-21} για τον $rs και ο καταχωρητής με διεύθυνση IR^{20-16} για τον $rt. Σε κάθε ανάγνωση του

ΦΚ, ο πρώτος διαβάζεται στον καταχωρητή ειδικού σκοπού Α και ο δεύτερος στον καταχωρητή ειδικού σκοπού Β. Από την άλλη μεριά, όμως, όσο αφορά την εγγραφή στο ΦΚ, βλέπουμε ότι η ροή πληροφορίας που έχουμε στις δύο εντολές addi μας επιβάλλει να μπορούμε να εισάγουμε στο ΦΚ νέα τιμή τόσο για τον καταχωρητή $rs όσο και για τον καταχωρητή $rt. Επομένως, θα προσθέσουμε στην είσοδο διευθύνσεων του ΦΚ τη δυνατότητα επιλογής διεύθυνσης εγγραφής και από το πεδίο $IR^{25\text{-}21}$, επιπλέον του πεδίου $IR^{20\text{-}16}$ που ήδη επιλέγεται.
Επειδή η εντολή movs είναι κωδικοποιημένη με σταθερά μετατόπισης, το μόνο άλλο πεδίο του IR που μπορούμε να χρησιμοποιήσουμε είναι το πεδίο σταθεράς $IR^{15\text{-}0}$. Αυτό λοιπόν θα περιέχει τη μετατόπιση που προστίθεται στο περιεχόμενο του καταχωρητή $rs. Οι περιπτώσεις στις οποίες χρειαζόμαστε κάποια άλλη σταθερά από τη λέξη εντολής, όπως εδώ συμβαίνει στις επιμέρους εντολές sb, addi, beq και bne, πρέπει να αντιμετωπιστούν διαφορετικά. Η πιο απλή λύση, την οποία και θα υιοθετήσουμε, είναι να υποθέσουμε ότι η είσοδος ΑΛΜ2 μπορεί να επιλέξει, εκτός των Β και προ(IR), απ' ευθείας και τις σταθερές που θέλουμε, οι οποίες εδώ είναι οι 0 και 1.
Η εντολή movs δεν μας παρέχει τον καταχωρητή γενικού σκοπού $tmp που χρησιμοποιήσαμε στον αρχικό κώδικα MIPS. Επομένως, θα πρέπει να αντικαταστήσουμε τον $tmp με κάποιον από τους καταχωρητές ειδικού σκοπού της ΜΕΔ. Εφόσον η τιμή που διαβάζουμε από τη μνήμη τοποθετείται αυτόματα στον DR, είναι βολικό στη θέση του $tmp να χρησιμοποιήσουμε τον καταχωρητή DR.
Ο καταχωρητής PC δε μπορεί να χρησιμοποιηθεί στις εντολές beq και bne. Τα αντίστοιχα άλματα μεταφέρονται στο πιο χαμηλό επίπεδο των μικροεντολών, όπου θα πρέπει να εισάγουμε δύο συνθήκες άλματος, τις ΑΛΜΖ και ΑΛΜΝΖ, για μηδενική και μη μηδενική έξοδο της ΑΛΜ, αντίστοιχα. Το κύκλωμα μPC δέχεται τις δύο συνθήκες, και εάν είναι αληθείς, θέτει στον μPC την κατάλληλη νέα διεύθυνση μικροκώδικα. Όπως με το μικροκώδικα εντολών διακλάδωσης MIPS για τον PC, έτσι και εδώ για τον μPC ο έλεγχος των παραπάνω συνθηκών γίνεται στον ίδιο κύκλο μηχανής με την πράξη της σύγκρισης στην ΑΛΜ.
Συμπληρώνοντας την ανάλυση ροής πληροφορίας για τη νέα εντολή, παρατηρούμε τα εξής:

1. Η ΑΛΜ εκτελεί πράξεις τόσο πάνω στην τιμή του καταχωρητή $rs, όσο και πάνω στην τιμή του καταχωρητή $rt. Επειδή ο πρώτος διαβάζεται στον Α και ο δεύτερος διαβάζεται στον Β, θα προσθέσουμε τη δυνατότητα επιλογής της εισόδου ΑΛΜ1 και από τον Β.
2. Ο καταχωρητής DR χρησιμοποιείται από την ΑΛΜ για σύγκριση με το 0. Θα συνδέσουμε έτσι την έξοδο του DR με την ΑΛΜ, και θα υποθέσουμε ότι η ΑΛΜ επιλέγει και τον DR για την είσοδο ΑΛΜ1.
3. Η τιμή του DR χρησιμοποιείται και για επακόλουθη εγγραφή στη μνήμη. Άρα, θα χρειαστεί να προσθέσουμε στη ΜΔΜ τη δυνατότητα επιλογής εισόδου δεδομένων εγγραφής και από τον DR. Παρόλο που η τιμή εγγραφής μπορεί να ληφθεί και από τον C μετά την αφαίρεση του 0 από την τιμή του DR, ο C πρέπει να παρέχει τη διεύθυνση προσπέλασης, και δε μπορεί να παρέχει ταυτόχρονα και τα δεδομένα εγγραφής! Ούτως ή άλλως, ούτε ο DR ούτε ο C δεν είναι συνδεμένοι με την είσοδο δεδομένων εγγραφής της ΜΔΜ, κι επομένως θα πρέπει να τροποποιήσουμε τους δρόμους ροής πληροφορίας προς τη μνήμη, είτε για τη μία είτε για την άλλη περίπτωση.

Με βάση την παραπάνω ανάλυση παίρνουμε τον ακόλουθο μικροκώδικα:

Εντολή	Ετικέτα	Άλμα	ΑΛΜ	ΑΛΜ1	ΑΛΜ2	ΦΚ	ΜΕΜ	PC
	Fetch:	seq	add	PC	4		readIR	ΑΛΜ
		disp	add	PC	προ(IR)$_{<<2}$			
lb $tmp,off($rs)	MOVS:	seq	add	A	προ(IR)			
		seq					readbDR	
beq $tmp,$0,End		brΑΛΜΖ End	sub	DR	0			
Loop: sb $tmp,0($rt)	Loop:	seq	add	B	0			
		seq					writebDR	
addi $rs,$rs,1		seq	add	A	1			
		seq				write1C		
addi $rt,$rt,1		seq	add	B	1			
		seq				write2C		

lb $tmp,off($rs)		seq	add	A	προ(IR)			
		seq					readbDR	
bne $tmp,$0,Loop		brΑΛΜΝΖ Loop	sub	DR	0			
End: sb $tmp,0($rt)	End:	seq	add	B	0			
		j Fetch					writebDR	

Η ενοποίηση των εντολών στον μικροκώδικα της εντολής movs απαλείφει την επανάληψη των φάσεων ανάκλησης και αποκωδικοποίησης για κάθε εντολή και την περιορίζει στην αρχή της νέας εντολής.
Ειδικότερα, όσο αφορά τη φάση αποκωδικοποίησης, δεν απαιτείται ο υπολογισμός της διεύθυνσης προορισμού άλματος στην ΑΛΜ για τις επιμέρους εντολές μετά την πρώτη, εφ' όσον ενδιάμεσα άλματα με συνθήκη υλοποιούνται σε επίπεδο μικροεντολών, όπως ήδη αναφέραμε, και στο μικροκώδικα η διεύθυνση προορισμού υπολογίζεται με ανεξάρτητα κυκλώματα. Η μικρολειτουργία άλματος disp μας οδηγεί στην πρώτη μικροεντολή της εκτέλεσης της νέας εντολής movs, με ετικέτα MOVS.
Στον παραπάνω πίνακα έχουμε σημειώσει με write1C και με write2C την εγγραφή από τον C στο ΦΚ με διεύθυνση εγγραφής IR^{25-21} και IR^{20-16} αντίστοιχα.
Παρατηρήστε ότι οι ετικέτες Loop και End που αρχικά αναφέρονταν σε εντολές, τώρα θα γίνουν ετικέτες στις αντίστοιχες μικροεντολές.

Ο μικροκώδικας που δώσαμε είναι πιθανό να μην είναι ακόμα σωστός, λόγω λανθασμένου χρονισμού σε εξαρτήσεις μεταξύ μικρολειτουργιών. Τέτοιο θέμα προκύπτει όταν ένα δεδομένο παράγεται σε υπομονάδα της ΜΕΔ, αλλά δε χρησιμοποιείται από κάποια εξαρτημένη μικρολειτουργία τη στιγμή που πρέπει, αλλά χρησιμοποιείται είτε νωρίτερα, είτε αργότερα. Έτσι, οποτεδήποτε συμβαίνει εγγραφή σε κάποιον καταχωρητή ειδικού σκοπού που δε διαθέτει επίτρεψη εγγραφής, το δεδομένο που αποθηκεύεται σε αυτόν θα πρέπει να καταναλώνεται στον αμέσως επόμενο κύκλο μηχανής. Αν κάτι τέτοιο δεν είναι εφικτό, θα χρειαστεί να προσθέσουμε σε αυτόν επίτρεψη εγγραφής.
Παρόμοιο θέμα εμφανίζεται όταν στο μικροκώδικα υπάρχει ενδιάμεση εγγραφή στο ΦΚ, οπότε οι νέες τιμές περνάνε στους Α και Β – αν βέβαια ο καταχωρητής που γράφεται ταυτίζεται με κάποιον από τους δύο καταχωρητές ανάγνωσης – μετά την εγγραφή. Θα επανέλθουμε στο χρονισμό των μικρολειτουργιών στο επόμενο βήμα, αφού η όποια πιθανή σύμπτυξη θα αλλάξει τις σχετικές θέσεις των μικρολειτουργιών που απαιτούν διόρθωση χρονισμού.

Δ. Ο τελικός μικροκώδικας βρίσκεται με σύμπτυξη διαδοχικών μικροεντολών, οι οποίες εκτελούν μικρολειτουργίες που ανήκουν σε διαφορετικά πεδία, και οι οποίες ακολουθούν υποχρεωτικά η μία την άλλη, χωρίς όμως η δεύτερη να είναι εξαρτημένη από την πρώτη, δηλαδή καμία μικρολειτουργία της δεύτερης μικροεντολής να μη χρησιμοποιεί το αποτέλεσμα μιας μικρολειτουργίας της πρώτης.
Έτσι, η τελευταία μικροεντολή της πρώτης από τις δύο εντολές sb μπορεί να συμπτυχθεί με την πρώτη μικροεντολή της επόμενης εντολής addi, αφού η μεν εκτελεί μόνο εγγραφή στη μνήμη, η δε εκτελεί μόνο πρόσθεση στην ΑΛΜ, ενώ η ροή από τη μία φτάνει στην άλλη χωρίς διακλάδωση.
Φαινομενικά, η τελευταία μικροεντολή της πρώτης από τις δύο addi μπορεί να συμπτυχθεί με την πρώτη μικροεντολή της δεύτερης addi, επειδή οι εκτελούμενες μικρολειτουργίες ανήκουν σε διαφορετικά πεδία, και επειδή η ροή από τη μία φτάνει πάντα στην άλλη. Όμως, η πρώτη μικροεντολή γράφει το ΦΚ, και συγκεκριμένα τον καταχωρητή $rs. Υποθέτοντας ότι σε κάθε κύκλο συμβαίνει ανάγνωση του ΦΚ, στο τέλος του κύκλου στον οποίο γίνεται η εγγραφή, ο καταχωρητής ειδικού σκοπού Α – που λαμβάνει την τιμή του $rs – θα πάρει από τον ΦΚ την τιμή που μόλις γράφτηκε. Επειδή η δεύτερη μικροεντολή χρησιμοποιεί τον καταχωρητή Β για την πρόσθεση, δεν υπάρχει πρόβλημα, εκτός εάν οι καταχωρητές $rs και $rt συμπίπτουν. Σε μια τέτοια περίπτωση, θα υπάρχει εξάρτηση μεταξύ των μικρολειτουργιών εγγραφής του ΦΚ και επιλογής του Β στο πεδίο ΑΛΜ1, οπότε η σύμπτυξη θα παραβιάσει την εξάρτηση αυτή. Κοιτάζοντας όμως προσεκτικότερα, θα διαπιστώσουμε ότι η τήρηση της εξάρτησης σε μια τέτοια περίπτωση θα οδηγήσει σε αύξηση του ίδιου καταχωρητή δύο φορές, μία από την πρώτη και μία από τη δεύτερη addi! Παρόλο που ο αρχικός κώδικας MIPS αυτό ακριβώς κά-

νει, το σωστό είναι η αύξηση να γίνεται μία φορά αν οι δύο καταχωρητές συμπίπτουν, οπότε με σύμπτυξη των δύο μικροεντολών – οπότε δεν προλαβαίνει να διαβαστεί ο Β με τη νέα τιμή – η δεύτερη αύξηση θα γίνει στην παλιά τιμή του $rt, και στην ουσία θα ακυρώνει την πρώτη. Επομένως θα πρέπει να συμπτύξουμε τις δύο μικροεντολές, όχι μόνο για βελτίωση του μικροκώδικα, αλλά και για ορθή εκτέλεσή του! Φυσικά αν οι δύο καταχωρητές δε συμπίπτουν, η σύμπτυξη δεν έχει καμία επίδραση στα αποτελέσματα των δύο προσθέσεων.
Προχωρώντας πιο κάτω, βλέπουμε ότι το ίδιο ακριβώς συμβαίνει μεταξύ της τελευταίας μικροεντολής της δεύτερης addi και της πρώτης μικροεντολής της εντολής lb που ακολουθεί. Και εδώ έχουμε μια εγγραφή του ΦΚ που ακολουθείται από χρήση του καταχωρητή ειδικού σκοπού Α. Ο ΦΚ όμως γράφει τον καταχωρητή $rt, ο οποίος κανονικά περνάει στον Β, κι επομένως παραβίαση εξάρτησης σε σύμπτυξη θα συμβεί μόνο αν οι δύο καταχωρητές $rs και $rt συμπίπτουν. Τώρα όμως, η τιμή που χρειαζόμαστε έχει ήδη γραφτεί στο ΦΚ από την πρώτη addi, και αν δεν είχαμε κάνει την προηγούμενη σύμπτυξη, θα διαβάζαμε από τον ΦΚ την λανθασμένη, διπλά αυξημένη τιμή του καταχωρητή, την οποία θα έγραφε στο ΦΚ η δεύτερη addi! Εδώ βέβαια, λόγω της υποχρεωτικής προηγούμενης σύμπτυξης, η τιμή που διαβάζεται είναι η σωστή, ακόμα κι αν δε συμπτύξουμε τις δύο μικροεντολές addi και lb, αφού όταν οι δύο καταχωρητές συμπίπτουν, οι δύο διαδοχικές addi θα γράφουν στο ΦΚ την ίδια τιμή. Εμείς θα επιλέξουμε τη σύμπτυξη για βελτίωση της απόδοσης του μικροκώδικα.

Γενικά, δε μπορεί να γίνει σύμπτυξη μεταξύ δύο μικροεντολών, όταν υπάρχει κάποια εξάρτηση μεταξύ τους. Σε τέτοια περίπτωση όμως, μπορούμε να εξετάσουμε εάν κάποια *μερική* σύμπτυξη μπορεί να παράγει πιο σύντομο μικροκώδικα, εάν δηλαδή μπορούμε να μεταφέρουμε μέρος μιας μικροεντολής σε προηγούμενη και το υπόλοιπο σε επόμενη. Οι μικρολειτουργίες κάποιας μικροεντολής που πρέπει να ενεργοποιούνται στον ίδιο κύκλο μηχανής δε μπορούν να διαχωριστούν για μερική σύμπτυξη. Για παράδειγμα, δε μπορούμε να διαχωρίσουμε μια επιλογή του πεδίου ΑΛΜ1 από την αντίστοιχη επιλογή του πεδίου ΑΛΜ2.
Η μόνη περίπτωση μικρολειτουργιών από τον παραπάνω μικροκώδικα που δεν απαιτείται να ενεργοποιούνται στον ίδιο κύκλο μηχανής είναι στη μικροεντολή ανάκλησης, και συγκεκριμένα η μικρολειτουργία του πεδίου MEM σε σχέση με τις υπόλοιπες που σχετίζονται με την αύξηση του PC. Όμως, δε μπορεί να γίνει μερική σύμπτυξη με την επόμενη μικροεντολή, επειδή τόσο η ανάγνωση της μνήμης όσο και η αύξηση του PC έχουν αποτέλεσμα που χρησιμοποιείται σε αυτήν.

Τελικά δεν προκύπτει καμία άλλη σύμπτυξη στον μικροκώδικα, κι έτσι καταλήγουμε στον παρακάτω μικροκώδικα της εντολής movs:

Ετικέτα	**Άλμα**	**ΑΛΜ**	**ΑΛΜ1**	**ΑΛΜ2**	**ΦΚ**	**MEM**	**PC**
Fetch:	seq	add	PC	4		readIR	ΑΛΜ
	disp	add	PC	$\pi\rho o(IR)_{<<2}$			
MOVS:	seq	add	A	$\pi\rho o(IR)$			
	seq					readbDR	
	brΑΛΜΖ End	sub	DR	0			
Loop:	seq	add	B	0			
	seq	add	A	1		writebDR	
	seq	add	B	1	write1C		
	seq	add	A	$\pi\rho o(IR)$	write2C		
	seq					readbDR	
	brΑΛΜΝΖ Loop	sub	DR	0			
End:	seq	add	B	0			
	j Fetch					writebDR	

Ο χρονισμός στον παραπάνω μικροκώδικα μας επιβάλλει την εισαγωγή επίτρεψης εγγραφής στον καταχωρητή ειδικού σκοπού DR. Διαφορετικά, σύμφωνα με όσα αναφέραμε και νωρίτερα, η χρήση του ως πηγή δεδομένων προς αποθήκευση στη μνήμη θα ήταν σωστή, μόνο αν η αποθήκευση γινόταν αμέσως μετά την αντίστοιχη φόρτωση. Χωρίς επίτρεψη εγγραφής, το περιεχόμενο του DR αλλάζει σε κάθε κύκλο μηχανής. Έτσι, αν στον κύκλο που προηγείται

της αποθήκευσης δεν υπάρχει φόρτωση, η τιμή που αποθηκεύεται είναι απροσδιόριστη. Επειδή στον παραπάνω μικροκώδικα οι αποθηκεύσεις δεν ακολουθούν άμεσα τις φορτώσεις, ούτε θα μπορούσαμε να τις κάνουμε να ακολουθούν, επειδή πρέπει να μεσολαβεί η αποτίμηση της συνθήκης τερματισμού της εντολής movs, δεν έχουμε άλλη λύση από την προσθήκη στον DR σήματος επίτρεψης εγγραφής. Το σήμα αυτό θα ενεργοποιείται, κάθε φορά που έχουμε φόρτωση από τη μνήμη, κι έτσι το περιεχόμενο του DR θα διατηρείται μέχρι την επόμενη φόρτωση, και θα μπορεί να χρησιμοποιηθεί από όσες ενδιάμεσες μικροεντολές το χρειάζονται.

Ο παραπάνω μικροκώδικας προκύπτει για τους περιορισμούς που δίνονται στην εκφώνηση της άσκησης. Ας δούμε στη συνέχεια, πώς θα διαμορφωνόταν ο μικροκώδικας, αν είχαμε μεγαλύτερη ελευθερία σε τροποποιήσεις στη ΜΕΔ.
Κατ' αρχήν, ας θεωρήσουμε τη δυνατότητα ελέγχου μηδενικής τιμής κατ' ευθείαν πάνω στα δεδομένα που φορτώνονται από τη μνήμη, πριν αυτά αποθηκευτούν στον καταχωρητή DR. Σε τέτοια περίπτωση, δεν είναι αναγκαίο να χρησιμοποιηθεί η ΑΛΜ για την αποτίμηση της συνθήκης. Έτσι, μπορούμε με κατάλληλη σύμπτυξη να τοποθετήσουμε τις αποθηκεύσεις αμέσως κάτω από τις φορτώσεις, και να αποφύγουμε την προσθήκη επίτρεψης εγγραφής στον DR, επιτυγχάνοντας μάλιστα αρκετά πιο σύντομο μικροκώδικα. Υποθέτοντας ότι ο έλεγχος μηδενικής τιμής γίνεται στην έξοδο της μνήμης με δύο συμπληρωματικές συνθήκες άλματος ΜΔΜΖ (μηδενική τιμή από μνήμη) και ΜΔΜΝΖ (μη μηδενική τιμή από μνήμη), ο μικροκώδικας διαμορφώνεται ως εξής:

Ετικέτα	**Άλμα**	**ΑΛΜ**	**ΑΛΜ1**	**ΑΛΜ2**	**ΦΚ**	**ΜΕΜ**	**PC**
Fetch:	seq	add	PC	4		readIR	ΑΛΜ
	disp	add	PC	$\pi\rho o(IR)_{<<2}$			
MOVS:	seq	add	A	$\pi\rho o(IR)$			
	brMΔMZ End	add	B	0		readbDR	
Loop:	seq	add	A	1		writebDR	
	seq	add	B	1	write1ΑΛΜ		
	seq	add	A	$\pi\rho o(IR)$	write2ΑΛΜ		
	brMΔMNZ Loop	add	B	0		readbDR	
End:	fetch					writebDR	

Παρατηρήστε ότι ο υπολογισμός της διεύθυνσης αποθήκευσης τώρα γίνεται στις μικροεντολές που ελέγχουν τη συνθήκη μηδενικής τιμής, αφού δεν απαιτείται η ΑΛΜ για τον έλεγχο, και από την άλλη η ροή ελέγχου είναι τέτοια, που όποιο και να είναι το αποτέλεσμα των διακλαδώσεων, ακολουθεί πάντα εγγραφή στη μνήμη.
Παρόμοια, αν μπορούμε να έχουμε επιλογή διεύθυνσης στη ΜΔΜ και από τον B, δε χρειάζεται να προσθέσουμε το περιεχόμενο του B με το 0, ώστε να λάβουμε τη διεύθυνση από τον C. Επιπλέον, μπορούμε να λάβουμε το δεδομένο προς αποθήκευση από τον C, όπου βρίσκεται μετά τον έλεγχο μηδενικής τιμής, και πάλι αποφεύγοντας την επίτρεψη εγγραφής στον DR. Θα πρέπει όμως να έχουμε σύνδεση τόσο του B προς την είσοδο διευθύνσεων, όσο και του C – αντί του DR – προς την είσοδο δεδομένων της ΜΔΜ. Με αυτή τη δυνατότητα, αλλά χωρίς τη δυνατότητα ελέγχου στην έξοδο της μνήμης, ο μικροκώδικας γίνεται:

Ετικέτα	**Άλμα**	**ΑΛΜ**	**ΑΛΜ1**	**ΑΛΜ2**	**ΦΚ**	**ΜΕΜ**	**PC**
Fetch:	seq	add	PC	4		readIR	ΑΛΜ
	disp	add	PC	$\pi\rho o(IR)_{<<2}$			
MOVS:	seq	add	A	$\pi\rho o(IR)$			
	seq					readbDR	
	brAΛMZ End	sub	DR	0			
Loop:	seq	add	A	1		writebC(B)	
	seq	add	B	1	write1ΑΛΜ		
	seq	add	A	$\pi\rho o(IR)$	write2ΑΛΜ		
	seq					readbDR	
	brAΛMNZ Loop	sub	DR	0			
End:	fetch					writebC(B)	

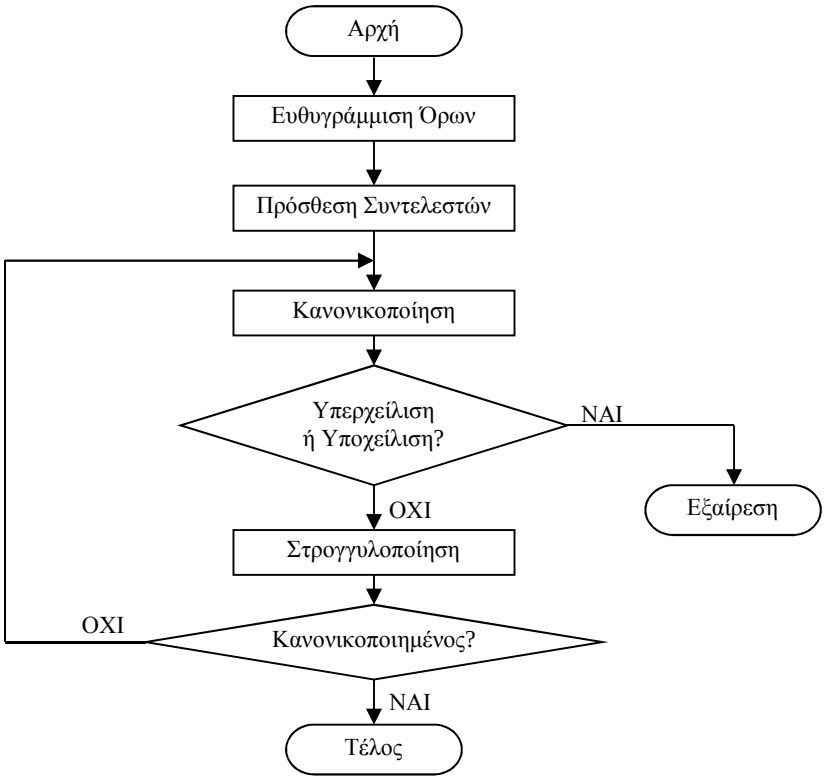
όπου με writebC(B) συμβολίσαμε την ενεργοποίηση εγγραφής μιας ψηφιολέξης στη μνήμη από τον C και με διεύθυνση που παρέχει ο B.
Μπορείτε να βρείτε τη μορφή που θα έπαιρνε ο μικροκώδικας, αν είχαμε ταυτόχρονα και τις δύο πιο πάνω τροποποιήσεις της ΜΕΔ. Ακόμα και έτσι, καταφέραμε να μειώσουμε το σώμα του βρόχου περίπου στο μισό του αρχικού, κάνοντας την εντολή movs σημαντικά πιο γρήγορη. Αυτό δείχνει πώς μικρές βελτιώσεις σε μια ΜΕΔ μπορούν να επιφέρουν μεγάλες βελτιώσεις στην απόδοση των επεξεργαστών.

Ας σημειώσουμε τέλος, ότι αν δε μας επιτρεπόταν αλλαγή στις τιμές των δύο καταχωρητών $rs και $rt, θα έπρεπε να χρησιμοποιήσουμε άλλους καταχωρητές ειδικού σκοπού για προσωρινή αποθήκευση των διευθύνσεων προσπέλασης μνήμης. Αν δεν μας αρκούσαν οι ήδη υπάρχοντες A, B, C και DR, θα έπρεπε να εισάγουμε νέους καταχωρητές και να δημιουργήσουμε τις κατάλληλες συνδέσεις με πολυπλέκτες και αντίστοιχα σήματα ελέγχου. Επιπλέον, είναι πιθανό να έπρεπε να τροποποιήσουμε τη μορφή της μικροεντολής, ώστε να περιλαμβάνει και πεδία διαχείρισης αυτών των καταχωρητών.
Αν για παράδειγμα θέλαμε να χρησιμοποιήσουμε τους A και B για την αποθήκευση των δύο διευθύνσεων προσπέλασης μνήμης, θα εισάγαμε δύο πολυπλέκτες επιλογής εισόδου στους A και B, ώστε να μπορούν να λάβουν τιμή και από την έξοδο της ΑΛΜ εκτός από τον ΦΚ, και να αποθηκεύουν έτσι τα αποτελέσματα της αύξησης κατά 1 των δύο διευθύνσεων της προηγούμενης επανάληψης. Οι δύο καταχωρητές θα έπρεπε να διαθέτουν επίτρεψη εγγραφής, ώστε να μη χάνεται η τιμή που εγγράφεται σε αυτούς από την έξοδο της ΑΛΜ σε επόμενους κύκλους μηχανής. Επιπλέον, οι A και B λαμβάνουν τιμή από το ΦΚ κατά τη φάση αποκωδικοποίησης, κάτι που θα έπρεπε να αναφέρεται στο μικροκώδικα. Έτσι, η μικροεντολή θα έπρεπε να περιλαμβάνει δύο πρόσθετα πεδία για την επιλογή εισόδου των A και B, τα οποία αν δεν είναι κενά να υπονοούν και επίτρεψη εγγραφής.
Θεωρώντας τον πρώτο τελικό μικροκώδικα που γράψαμε, λαμβάνοντας υπόψη τα παραπάνω, θα καταλήγαμε στον ακόλουθο μικροκώδικα:

Ετικέτα	**Άλμα**	**ΑΛΜ**	**ΑΛΜ1**	**ΑΛΜ2**	**A**	**B**	**ΦΚ**	**MEM**	**PC**
Fetch:	seq	add	PC	4				readIR	ΑΛΜ
	disp	add	PC	προ(IR)$_{<<2}$	ΦΚ	ΦΚ			
MOVS:	seq	add	A	προ(IR)					
	seq							readbDR	
	brΑΛΜΖ End	sub	DR	0					
Loop:	seq	add	B	0					
	seq	add	A	1	ΑΛΜ			writebDR	
	seq	add	B	1		ΑΛΜ			
	seq	add	A	προ(IR)					
	seq							readbDR	
	brΑΛΜΝΖ Loop	sub	DR	0					
End:	seq	add	B	0					
	j Fetch							writebDR	

Άσκηση 4:

Θεωρήστε τον αλγόριθμο πρόσθεσης αριθμών κινητής υποδιαστολής του προτύπου IEEE 754 που δίνεται στην επόμενη σελίδα. Ορίστε μια αρχιτεκτονική που εκτελεί την πράξη του αλγορίθμου αυτού και περιγράψτε τις μικρολειτουργίες της. Σχεδιάστε μια μικροπρογραμματισμένη μονάδα ελέγχου γι' αυτή την αρχιτεκτονική, δίνοντας το μικροκώδικα και τις αντίστοιχες συνθήκες άλματος.



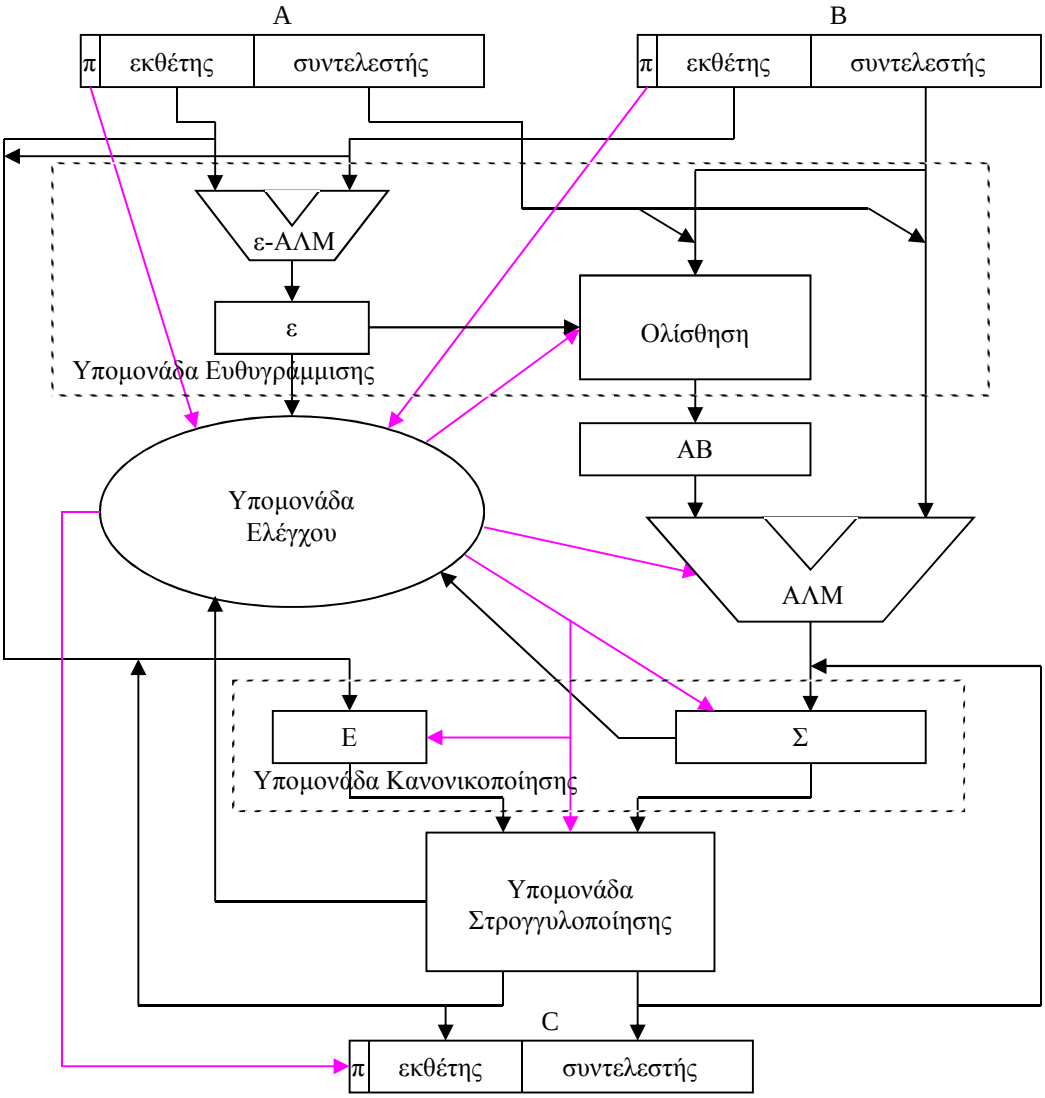
Απάντηση:

Υποθέτουμε ότι η ζητούμενη μονάδα πρόσθεσης διαβάζει τους προσθετέους από δύο καταχωρητές κινητής υποδιαστολής Α και Β, και επιστρέφει το άθροισμα σε έναν τρίτο C. Θα αποτελείται δε από τις ακόλουθες 5 υπομονάδες:

1. Υπομονάδα Ευθυγράμμισης. Αυτή παίρνει τους δύο προσθετέους και τους ευθυγραμμίζει. Ειδικότερα, συγκρίνει με μια πράξη αφαίρεσης τους δύο εκθέτες και ολισθαίνει τον συντελεστή του αριθμού με το μικρότερο εκθέτη τόσες θέσεις δεξιά, όση είναι η απόλυτη τιμή της διαφοράς, συνυπολογίζοντας στην ολίσθηση και το ψηφίο 1 του ακέραιου μέρους του αριθμού. Ο μεγαλύτερος από τους δύο εκθέτες θα είναι και ο αρχικός εκθέτης του αθροίσματος.
2. ΑΛΜ. Αυτή εκτελεί τις πράξεις της πρόσθεσης και της αφαίρεσης, ανάλογα με το σχετικό πρόσημο των δύο προσθετέων. Δέχεται ως εισόδους τους δύο συντελεστές μετά την ευθυγράμμιση, συνυπολογίζοντας στην πράξη και το πιθανό ψηφίο 1 του ακέραιου μέρους των αριθμών.
3. Υπομονάδα Κανονικοποίησης. Αυτή κανονικοποιεί το άθροισμα, δεδομένου ότι η έξοδος της ΑΛΜ μπορεί να μην αντιστοιχεί σε κανονικοποιημένο αριθμό κινητής υποδιαστολής. Ειδικότερα, διακρίνει δύο περιπτώσεις μη κανονικοποιημένου αποτελέσματος: (α) Εάν το ακέραιο μέρος του αθροίσματος είναι τα δύο ψηφία 10 ή 11, ολισθαίνει το συντελεστή δεξιά, και ταυτόχρονα αυξάνει τον εκθέτη κατά 1. (β) Εάν το ακέραιο μέρος του αθροίσματος είναι τα δύο ψηφία 00 και το άθροισμα δεν είναι το 0, ολισθαίνει το συντελεστή τόσες θέσεις αριστερά, μέχρι να εμφανιστεί 1 στο ακέραιο μέρος του αθροίσματος. Ταυτόχρονα μειώνει αντίστοιχα τον εκθέτη.
4. Υπομονάδα Στρογγυλοποίησης. Αυτή στρογγυλοποιεί το κανονικοποιημένο άθροισμα, μια που η προηγούμενη φάση παράγει αριθμό μεγαλύτερης ακρίβειας από την αναπαράσταση που έχουμε. Ο ακριβής αλγόριθμος στρογγυλοποίησης δε μας απασχολεί. Το αποτέλεσμά του όμως μπορεί να μην είναι κανονικοποιημένο, οπότε θα πρέπει να το ξαναστείλουμε πίσω στην υπομονάδα κανονικοποίησης, πριν πάρουμε το τελικό αποτέλεσμα.
5. Υπομονάδα Ελέγχου. Αυτή ενεργοποιεί όλες τις μικρολειτουργίες της μονάδας μέσω μικροπρογραμματισμένης λογικής. Έτσι, επιλέγει το μεγαλύτερο εκθέτη και τους αντίστοι-

χους συντελεστές για την υπομονάδα ευθυγράμμισης, ελέγχει το πρόσημο των προσθετέων και αποφασίζει για τη σωστή πράξη μεταξύ των συντελεστών, ελέγχει την υπομονάδα κανονικοποίησης με βάση το ακέραιο μέρος του αποτελέσματος της ΑΛΜ και αποφασίζει για υπερχείλιση ή υποχείλιση, ελέγχει το αποτέλεσμα της στρογγυλοποίησης και αποφασίζει για πιθανή επανάληψη της κανονικοποίησης, ενώ τέλος παράγει και το τελικό πρόσημο του αθροίσματος.

Η αρχιτεκτονική της ζητούμενης μονάδας απεικονίζεται στο διάγραμμα που ακολουθεί:



Στο διάγραμμα αυτό παρατηρούμε τα εξής:

1. Έχουμε 4 καταχωρητές ειδικού σκοπού για την αποθήκευση ενδιάμεσων αποτελεσμάτων. (α) Ο καταχωρητής ε αποθηκεύει το αποτέλεσμα της αφαίρεσης των δύο εκθετών. (β) Ο καταχωρητής ΑΒ αποθηκεύει το αποτέλεσμα της ολίσθησης του συντελεστή που αντιστοιχεί στο μικρότερο εκθέτη κατά τη φάση της ευθυγράμμισης. (γ) Ο καταχωρητής Ε αποθηκεύει τον προσωρινό εκθέτη του αθροίσματος. Μπορεί να αυξάνει ή να μειώνει το περιεχόμενό του κατά 1 με κατάλληλο σήμα ελέγχου. (δ) Ο καταχωρητής Σ αποθηκεύει το προσωρινό συντελεστή του αθροίσματος. Λειτουργεί ως καταχωρητής ολίσθησης, ολισθαίνοντας το περιεχόμενό του κατά μία θέση δεξιά ή αριστερά με κατάλληλο σήμα ελέγχου.
2. Έχουμε 5 σημεία επιλογής πληροφορίας, στα οποία τοποθετούμε πολυπλέκτες 2×1: (α) Ο ολισθητής της υπομονάδας ευθυγράμμισης δέχεται είσοδο από έναν από τους Α και Β. (β) Η ίδια επιλογή γίνεται και για τη δεύτερη είσοδο της ΑΛΜ. (γ) Ο δρόμος από την εί-

σοδο προς τον καταχωρητή Ε μεταφέρει τον έναν από τους δύο εκθέτες εισόδου. (δ) Η είσοδος του Ε μπορεί να γίνει είτε από τους Α και Β, είτε από την έξοδο εκθέτη της υπομονάδας στρογγυλοποίησης. (ε) Η είσοδος στο Σ μπορεί να γίνει είτε από την ΑΛΜ είτε από την έξοδο συντελεστή της υπομονάδας στρογγυλοποίησης.

3. Υποθέτουμε ότι οι υπομονάδες εκτέλεσης πράξεων ε-ΑΛΜ και ΑΛΜ παράγουν το αποτέλεσμά τους στη μορφή πρόσημο/μέτρο. Τα δύο πρόσημα εξόδου στέλνονται στην υπομονάδα ελέγχου, το πρώτο για την επιλογή εκθετών και συντελεστών στην υπομονάδα ευθυγράμμισης, το δεύτερο για τον υπολογισμό του τελικού προσήμου του αθροίσματος.
4. Σε κάθε πράξη, η ΑΛΜ συμπεριλαμβάνει το ακέραιο ψηφίο των προσθετέων. Αν αυτοί είναι ευθυγραμμισμένοι, το άθροισμα (σε περίπτωση πράξης πρόσθεσης) δίνει κρατούμενο εξόδου, το οποίο αποθηκεύουμε στον Σ σα δεύτερο και σημαντικότερο ψηφίο του ακέραιου μέρους του αποτελέσματος. Αυτό θα απαλειφτεί με την κανονικοποίηση.
5. Η αρχή της πρόσθεσης σηματοδοτείται από τη ΜΕ της ΚΜΕ με την εγγραφή των καταχωρητών Α και Β. Το τέλος της πρόσθεσης σηματοδοτείται είτε με την εγγραφή στον C, είτε με την ένδειξη εξαίρεσης (υπερχείλιση ή υποχείλιση). Σε καθεμιά από τις δύο περιπτώσεις έχουμε επιστροφή ελέγχου στην ΚΜΕ.

Οι μικρολειτουργίες (σήματα ελέγχου) της αρχιτεκτονικής δίνονται στον ακόλουθο πίνακα:

α/α	**Μικρολειτουργία**	**Περιγραφή**
1	ε-sub	Εκτέλεση της πράξης: Εκθέτης(Α) – Εκθέτης(Β)
2	Ε=Α	Επιλογή του Εκθέτης(Α) για είσοδο του Ε
3	Ε=Β	Επιλογή του Εκθέτης(Β) για είσοδο του Ε
4	ΟΛ1 = Α	Επιλογή του Συντελεστής(Α) για την είσοδο 1 του Ολισθητή
5	ΟΛ1 = Β	Επιλογή του Συντελεστής(Β) για την είσοδο 1 του Ολισθητή
6	ολίσθηση(ε)	Εκτέλεση ολίσθησης με την είσοδο αριθμού ψηφίων ολίσθησης από τον ε
7	ΑΛΜ2 = Α	Επιλογή του Συντελεστής(Α) για την είσοδο 2 της ΑΛΜ
8	ΑΛΜ2 = Β	Επιλογή του Συντελεστής(Β) για την είσοδο 2 της ΑΛΜ
9	πράξη(ΥΕ)	Η ΑΛΜ εκτελεί πράξη που καθορίζεται από την Υπομονάδα Ελέγχου
10	Ε = ΥΣ	Επιλογή της εξόδου εκθέτη της Υπομονάδας Στρογγυλοποίησης για τον Ε
11	Ε++	Αύξηση του Ε κατά 1
12	Ε--	Μείωση του Ε κατά 1
13	Σ = ΑΛΜ	Επιλογή της εξόδου της ΑΛΜ για είσοδο στον Σ
14	Σ = ΥΣ	Επιλογή της εξόδου συντελεστή της ΥΣ για είσοδο στον Σ
15	δ.ολίσθησηΣ	Εκτέλεση δεξιάς ολίσθησης στον Σ
16	α.ολίσθησηΣ	Εκτέλεση αριστερής ολίσθησης στον Σ
17	στρογγυλοποίηση	Εκτέλεση στρογγυλοποίησης
18	πρόσημο	Υπολογισμός του προσήμου του αθροίσματος
19	εγγραφή C	Ενεργοποίηση σήματος εγγραφής του καταχωρητή C
20	Cause = υπερχείλιση	Επιλογή κωδικού υπερχείλισης για τον κώδικα εξαίρεσης
21	Cause = υποχείλιση	Επιλογή κωδικού υποχείλισης για τον κώδικα εξαίρεσης

Οι τελευταίες δύο μικρολειτουργίες αναφέρονται σε έναν καταχωρητή που παρέχει η ΜΕ για τον έλεγχο εξαιρέσεων, τον Cause. Σ' αυτόν υποθέτουμε ότι η μονάδα κινητής υποδιαστολής γράφει έναν κωδικό με την επιστροφή ελέγχου στην ΚΜΕ, όταν θέλει να σηματοδοτήσει κάποια εξαίρεση. Το πώς η ΜΕ της ΚΜΕ αντιμετωπίζει τις εξαιρέσεις δε μας απασχολεί εδώ.
Όπως και σε άσκηση της προηγούμενης ενότητας, έτσι κι τώρα δεν αναφέρθηκαν κάποια σήματα εγγραφής καταχωρητών, επειδή ενεργοποιούνται μέσω άλλων σημάτων. Έτσι, η εγγραφή του ε ενεργοποιείται με τη μικρολειτουργία 1, η εγγραφή του ΑΒ με τη μικρολειτουργία 6, η εγγραφή του Ε με τις μικρολειτουργίες 2, 3 και 10, και τέλος η εγγραφή του Σ με τις μικρολειτουργίες 13 και 14.
Από την άλλη μεριά, επειδή κάποιοι δρόμοι πληροφορίας δεν έχουν σημεία επιλογής, δεν ορίζονται τα αντίστοιχα σήματα εισόδου σε κάποιες υπομονάδες. Για παράδειγμα, δεν αναφέρεται σήμα επιλογής της εισόδου ΑΛΜ1, που είναι συνδεμένη μόνιμα με τον ΑΒ. Παρόμοια, η είσοδος ΟΛ2 του ολισθητή που δέχεται τον αριθμό των ψηφίων ολίσθησης είναι συνδεμένη μόνιμα με τον ε.

Για την υλοποίηση της ΥΕ με μικροπρογραμματισμένη λογική χρειαζόμαστε να επιλέξουμε τη μορφή της μικροεντολής. Επειδή έχουμε μικρό σχετικά αριθμό μικρολειτουργιών, θα χρησιμοποιήσουμε κωδικοποίηση σε οριζόντιες μικροεντολές, δηλαδή με μορφή της λέξης ελέγχου με ένα bit για κάθε μικρολειτουργία. Ακόμα, θα χρησιμοποιήσουμε και δύο πεδία άλματος: ένα πεδίο συνθήκης και ένα πεδίο διεύθυνσης.
Υποθέτοντας ότι στο πεδίο συνθήκης προστίθεται κι ένα bit επιλογής ΑΨ μεταξύ αληθούς και ψευδούς συνθήκης, κι ότι το ψηφίο αυτό μας επιτρέπει επιπλέον να επιλέξουμε (α) μεταξύ άμεσου άλματος και συνέχειας στην επόμενη μικροεντολή και (β) μεταξύ κανονικής εξόδου και εξόδου με εξαίρεση από τη μονάδα πρόσθεσης, οι συνθήκες που χρειαζόμαστε θα είναι όπως δίνονται στον παρακάτω πίνακα:

Κωδικός	**Συνθήκη Άλματος**	**Περιγραφή**
000		Επιστροφή στη ΜΕ (εξαίρεση εάν ΑΨ = 1)
001		Επόμενη μικροεντολή (άμεσο άλμα εάν ΑΨ = 1)
010	ε < 0	Ο καταχωρητής ε έχει αρνητική τιμή
011	ΑΚΕΡ(Σ) = 01	Το ακέραιο μέρος του Σ είναι 01 (κανονικοποιημένη μορφή)
100	ΑΚΕΡ(Σ) = 00	Το ακέραιο μέρος του Σ είναι 00
101	$ΑΚΕΡ_1(Σ) = 1$	Το πιο σημαντικό ψηφίο του ακέραιου μέρους του Σ είναι 1
110	υπερχείλιση	Η κανονικοποίηση οδήγησε σε υπερχείλιση
111	υποχείλιση	Η κανονικοποίηση οδήγησε σε υποχείλιση

Όταν υπάρχει συνθήκη, το ψηφίο ΑΨ του πεδίου συνθήκης επιλέγει το αληθές της συνθήκης όταν έχει τιμή 1 και το ψευδές όταν έχει τιμή 0, αντιστρέφοντας δηλαδή τη συνθήκη. Για παράδειγμα, συμπεριλαμβάνοντας το ψηφίο ΑΨ, πεδίο συνθήκης με τιμή 0101 αντιστοιχεί στη συνθήκη «ε<0», ενώ πεδίο με τιμή 0100 αντιστοιχεί στη συνθήκη «ε≥0». Παρόμοια, πεδίο συνθήκης με τιμή 1101 αντιστοιχεί στη συνθήκη «υπερχείλιση», ενώ πεδίο με τιμή 1100 στη συνθήκη «όχι υπερχείλιση». Όταν δεν υπάρχει συνθήκη, το ψηφίο ΑΨ απλά αυξάνει τις επιλογές στο πεδίο άλματος. Έτσι, η τιμή 0000 αντιστοιχεί σε κανονική επιστροφή ελέγχου στην κεντρική ΜΕ, ενώ η τιμή 0001 αντιστοιχεί σε επιστροφή ελέγχου στην κεντρική ΜΕ με σηματοδότηση εξαίρεσης. Τέλος, η τιμή 0011 αντιστοιχεί σε άμεσο άλμα προς τη διεύθυνση που δίνει το πεδίο διεύθυνσης, ενώ η τιμή 0010 αντιστοιχεί σε μετάβαση στην επόμενη μικροεντολή.

Ο ζητούμενος μικροκώδικας δίνεται στον παρακάτω πίνακα, όπου παραθέσαμε και τις αντίστοιχες μικρολειτουργίες σαν επεξήγηση της λέξης ελέγχου:

	Συνθήκη	**Διεύθυνση**	**Λέξη Ελέγχου**	**Μικρολειτουργίες**
Add:	0010		10000000000000000000	ε-sub
	0101	L1	00000000000000000000	
	0010		01001100000000000000	Ε = Α, ΟΛ1 = Β, ολίσθηση(ε)
	0011	Norm	00000010100010000000	ΑΛΜ2 = Α, πράξη(ΥΕ), Σ = ΑΛΜ
L1:	0010		00110100000000000000	Ε = Β, ΟΛ1 = Α, ολίσθηση(ε)
	0010		00000001100010000000	ΑΛΜ2 = Β, πράξη(ΥΕ), Σ = ΑΛΜ
Norm:	0111	L	00000000000000000000	
	1010	L2	00000000000000000000	
	0010		00000000010001000000	Ε++, δ.ολίσθησηΣ
	1101	EX1	00000000000000000000	
	0011	L+1	00000000100010010000	στρογγυλοποίηση, Ε = ΥΣ, Σ = ΥΣ
L2:	0010		00000000001000100000	Ε--, α.ολίσθησηΣ
	1111	EX2	00000000000000000000	
	1001	L2	00000000000000000000	
L:	0010		00000000100010010000	στρογγυλοποίηση, Ε = ΥΣ, Σ = ΥΣ
	0110	Norm+1	00000000000000000000	
	0000		00000000000000001100	πρόσημο, εγγραφή C
EX1:	0001		00000000000000000010	Cause = υπερχείλιση
EX2:	0001		00000000000000000001	Cause = υποχείλιση