Άσκηση 7:
Πρόγραμμα Δυαδικού Δένδρου και Διαδικασιών

Προθεσμία έως Δευτέρα 16 Μαρτίου 2009, ώρα 23:59 (βδομάδα 7)

Υπενθύμιση Διαγωνισμού Προόδου: Σάββατο, 21 Μαρτίου 2009, ώρα 11:15 - 13:05
(Ο βαθμός της εξέτασης Προόδου αποτελεί το 20% του βαθμού μαθήματος, **οιασδήποτε** περιόδου)

[Up - Table of Contents]
[Prev - 6. Procedure Call]
[printer version - PDF]
[8. Verilog Intro. 1 - Next]

7.1 Δομές Δεδομένων (Data Structures):

Σε αυτή την άσκηση θα χρησιμοποιήσουμε μία δομή δεδομένων (structure) που θα αποτελεί ένα κόμβο ενός δυαδικού δένδρου αναζήτησης (binary search tree). Κάθε κόμβος (δομή δεδομένων) μας θα αποτελείται από τρεις λέξεις (των 32 bits καθεμία): έναν ακέραιο "data" που θα περιέχει την "πληροφορία χρήστη", κι δύο δείκτες απογόνων (pointers) "lPtr" (αριστερός απόγονος) και "rPtr" (δεξιός απόγονος). Σε ένα δυαδικό δένδρο αναζήτησης, κάθε κόμβος έχει το πολύ δύο απογόνους. Ο αριστερός απόγονος, εάν υπάρχει, έχει τιμή data μικρότερη από τον κόμβο-γονέα. Ο δεξιός απόγονος, απόγονος, εάν υπάρχει, έχει τιμή data μεγαλύτερη από τον κόμβο γονέα. Εάν ο αριστερός/δεξιός απόγονος ενός κόμβου δεν υπάρχει, σημαίνει ότι στο δένδρο δεν υπάρχει αποθηκευμένη μικρότερη/μεγαλύτερη τιμή data από αυτή του κόμβου. Στην περίπτως που ο αριστερός/δεξιός απόγονος του κόμβου δεν υπάρχει, υποθέτουμε ότι lPtr=0 ή rPtr=0, αντίστοιχα. Τα τρία στοιχεία (λέξεις) της δομής μας θα βρίσκονται σε διαδοχικές θέσεις (λέξεις) της μνήμης. Επομένως, κάθε δομή (κόμβος) μας θα έχει μέγεθος 3x4=12 bytes. Διεύθυνση μιάς δομής είναι η διεύθυνση του πρώτου στοιχείου της, δηλαδή του στοιχείου με "μηδενικό offset", που γιά μας είναι το "data". Άρα, το δεύτερο στοιχείο της δομής μας, ο "lPtr", βρίσκεται στη διεύθυνση που προκύπτει προσθέτοντας 4x1=4 στη διεύθυνση της δομής (κόμβου) και το τρίτο στοιχείο της δομήας μας, ο "rPtr", βρίσκεται στη διεύθυνση 4x2=8.

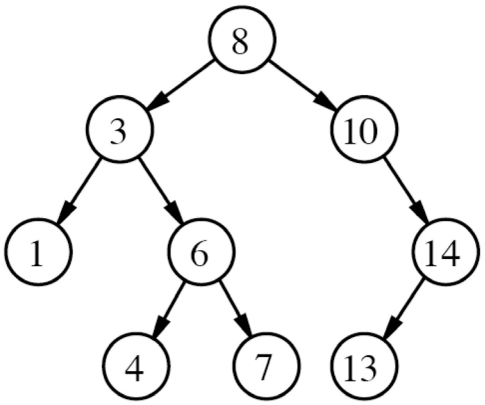
7.2 Δυναμική Εκχώρηση Μνήμης (Dynamic Memory Allocation):

Το πρόγραμμά σας θα ζητάει και θα παίρνει δομές (κόμβους) από το λειτουργικό σύστημα "δυναμικά", την ώρα που τρέχει (σε run-time). Γιά το σκοπό αυτό θα χρησιμοποιήσετε την κλήση συστήματος (system call) "sbrk" (set break). Το κάλεσμα αυτό "σπρώχνει" πιό πέρα (προς αύξουσες διευθύνσεις μνήμης) το σημείο "break", το όριο δηλαδή πριν από το οποίο οι διευθύνσεις μνήμης που γεννά το πρόγραμμα είναι νόμιμες, ενώ μετά από το οποίο (και μέχρι την αρχή της στοίβας) οι διευθύνσεις είναι παράνομες και τυχόν χρήση τους προκαλεί το γνωστό από την C "segmentation violation - core dumped". Το κάλεσμα συστήματος "sbrk" περιγράφεται στις σελίδες 112-113 του βιβλίου (Β' τόμος Ελληνικής έκδοσης), και λειτουργεί κατ' αναλογία με τα άλλα καλέσματα συστήματος (εκτύπωσης και ανάγνωσης) που χρησιμοποιήσατε σε προηγούμενες ασκήσεις. Πριν το καλέσετε, θέτετε τον καταχωρητή $a0 ($4) να περιέχει το πλήθος των νέων bytes που επιθυμείτε (ένας αριθμός). Μετά την επιστροφή του, ο καταχωρητής $v0 ($2) περιέχει τη διεύθυνση του νέου block μνήμης, του ζητηθέντος μεγέθους, που το σύστημα δίνει στο πρόγραμμά σας (έναν pointer). Η επιστρεφόμενη διεύθυνση μνήμης είναι πάντα διάφορη του μηδενός (εκτός --πιθανότατα-- όταν γεμίσει όλη η μνήμη, αλλά δεν χρειάζεται εσείς εδώ να ελέγχετε κάτι τέτοιο), και είναι πάντα ευθυγραμμισμένη σε όρια λέξεων

(πολλαπλάσιο του 4) (τουλάχιστο στη δική μας περίπτωση, που ζητάμε πάντα blocks μεγέθους πολλαπλάσιου του 4, αλλά και σε κάθε περίπτωση).

<u>Άσκηση 7.3:</u> Κατασκευή και Αναζήτηση σε Δυαδικό Δένδρο Αναζήτησης

Γράψτε και τρέξτε στον SPIM, σε Assembly του MIPS, ένα πρόγραμμα που πρώτα θα κατασκευάζει και θα γεμίζει με θετικούς ακέραιους αριθμούς ένα δυαδικό δένδρο αναζήτησης (binary search tree), και στη συνέχεια θα το ψάχνει στο δένδρο αυτό μη αρνητικούς αριθμούς που δίνει ο χρήστης από την κονσόλα. Το πρόγραμμά σας θα κρατάει στον καταχωρητή $s0 (δηλαδή τον $16) τον pointer στη ρίζα του δένδρου. Η ρίζα του δένδρου είναι ο πρώτος κόμβος που εισάγουμε στο δένδρο. Εδώ κάνουμε μία σημαντική απλοποίηση η οποία μπορεί να οδηγήσει σε μη ισορροποημένα δένδρα που γέρνουν προς τα αριστερά ή τα δεξιά ανάλογα με τα δεδομένα που εισάγουμε σε αυτά. Στην άσκηση αυτή θα αγνοήσουμε λειτουργίες εξισορρόπησης των δένδρων. Η εικόνα δεξιά σας δείχνει ένα δυαδικό δένδρο αναζήτησης με 9 κόμβους, βάθος απογόνων 3, και ρίζα το 8. Το πρόγραμμά σας θα αποτελείται από δύο κομμάτια:



7.3(a): Κατασκευή του Δένδρου Δυαδικής Αναζήτησης. Για κάθε κόμβο που θα εισάγετε στο δένδρο, πρέπει να ζητήσετε και να πάρετε χώρο στη μνήμη δεδομένων ένα κόμβο από το λειτουργικό σύστημα. Αρχικοποιείτε τον κόμβο γράφοντας data=0, lPtr=0 και rPtr=0 σε αυτόν, και κάνοντας τον καταχωρητή $s1 να δείχνει σε αυτόν τον κόμβο (δηλ. να περιέχουν τη διεύθυνσή του). Μετά, μπείτε στο βρόχο ανάγνωσης στοιχείων και κατασκευής του δένδρου. Σε κάθε ανακύκλωση αυτού του βρόχου:

1. Διαβάζουμε έναν ακέραιο αριθμό από την κονσόλα.
2. Εάν ο αριθμός αυτός είναι μηδέν, βγαίνουμε από το βρόχο.
3. Ζητάμε έναν νέο κόμβο από το λειτουργικό σύστημα (memory allocation).
4. Τοποθετούμε τον αριθμό που διαβάσαμε στο πεδίο "data" του κόμβου.
5. Συνδέουμε το νέο κόμβο στο δένδρο.

Το βήμα σύνδεσης χρειάζεται περαιτέρω εξήγηση: Ο πρώτος κόμβος που εισάγεται στο δένδρο προφανώς δεν έχει απογόνους. Για να απλοποιήσουμε τη διαδικασία δημιουργίας του πρώτου κόμβου, υποθέτουμε ότι ο καταχωρητής $s0, που είναι ο δείκτης στη ρίζα, αρχικοποιείται με την τιμή 0. Ακολουθούμε την εξής διαδικασία, η οποία μπορεί να εκφραστεί είτε επαναληπτικά είτε με χρήση αναδρομής (η οποία κατά τη γνώμη μας είναι και καλύτερη για την υλοποίηση της συγκεκριμένης δομής):

1. Ελέγχουμε εάν ο δείκτης ρίζας έχει την τιμή 0.
2. Εάν ναι το δένδρο είναι άδειο και θέτουμε τον δείκτη της ρίζας $s0 να δείχνει στον νέο κόμβο που εισάγουμε.
3. Εάν όχι, συγκρίνουμε την τιμή data του νέου κόμβου με αυτή του κόμβου που δείχνει ο δείκτης της ρίζας.
4. Εάν η τιμή data του νέου κόμβου είναι μικρότερη αυτής της ρίζας, διαβάζουμε το δείκτη του αριστερού απογόνου της ρίζας. Εάν ο δείκτης του αριστερού απογόνου έχει την τιμή 0, τότε εκχωρούμε στον δείκτη αυτό τη διεύθυνση του νέου κόμβου που εισάγουμε (την οποία έχουμε στον καταχωρητή $s1). Εάν ο δείκτης του αριστερού απογόνου έχει τιμή διαφορετική του 0 (δηλ. δείχνει σε μία νόμιμη διεύθυνση στο χώρο δεδομένων), τότε θέτουμε τον αριστερό απόγονο σαν ρίζα του (υπο)δένδρου στο οποίο επιχειρούμε να εισάγουμε το νέο κόμβο και επαναλαμβάνουμε τι διαδικασία από το βήμα 1. Υπενθυμίζεται ότι αρχικά η ρίζα του δένδρου είναι στον καταχωρητή $s0.

5. Εάν η τιμή data του νέου κόμβου είναι μεγαλύτερη αυτής της ρίζας, διαβάζουμε το δείκτη του δεξιού απογόνου της ρίζας. Εάν ο δείκτης του δεξιού απογόνου έχει την τιμή 0, τότε εκχωρούμε στον δείκτη αυτό τη διεύθυνση του νέου κόμβου που εισάγουμε (την οποία έχουμε στον καταχωρητή $s1). Εάν ο δείκτης του δεξιού απογόνου έχει τιμή διαφορετική του 0 (δηλ. δείχνει σε μία νόμιμη διεύθυνση στο χώρο δεδομένων), τότε θέτουμε τον δεξιό απόγονο σαν ρίζα του (υπο)δένδρου στο οποίο επιχειρούμε να εισάγουμε το νέο κόμβο και επαναλαμβάνουμε τι διαδικασία από το βήμα 1. Υπενθυμίζεται ότι αρχικά η ρίζα του δένδρου είναι στον καταχωρητή $s0.

7.3(b): Αναζήτηση στο Δένδρο. Το δεύτερο μέρος του προγράμματος θα διαβάζει έναν μη αρνητικό αριθμό, θα αναζητεί τον αριθμό στο δένδρο και θα τυπώνει ένα μήνυμα που να λέει εάν ο αριθμός βρέθηκε ή δεν βρέθηκε. Το μέρος αυτό του προγράμματος κάνει τα εξής:

- Διαβάζει έναν ακέραιο αριθμό από την κονσόλα και τον φυλάει στον καταχωρητή $s1 ($17) (υποτίθεται ότι αυτός είναι μη αρνητικός αριθμός, αλλά δεν χρειάζεται εσείς να το ελέγχετε).
- Αρχικοποιεί τον καταχωρητή $s2 ($18) σαν δείκτη (pointer) που δείχνει στη ρίζα του δένδρου (τον ξέρουμε από τον $s0 ($16)).
- Μπαίνει σ' ένα βρόχο, σε κάθε ανακύκλωση του οποίου:
  i. ελέγχει αν ο δείκτης $s2 είναι 0,
  ii. εάν ο $s2 είναι 0 τερματίζει με μήνυμα "δεν βρέθηκε" και βγαίνει από το βρόχο,
  iii. εάν όχι, ελέγχει αν τα "data" του κόμβου όπου δείχνει ο $s2 ($18) είναι μεγαλύτερα, μικρότερα, ή ίσα από τον $s1 ($17),
  iv. αν είναι μικρότερα θέτει τον $s2 να δείχνει στον δεξιό απόγονο του κόμβου τον οποίο έδειχνε πριν και επαναλαμβάνει τη διαδικασία από το πρώτο βήμα,
  v. αν είναι μεγαλύτερα θέτει τον $s2 να δείχνει στον αριστερό απόγονο του κόμβου τον οποίο έδειχνε πριν και επαναλαμβάνει τη διαδικασία από το πρώτο βήμα,
  vi. εάν είναι ίσα τυπώνει το μήνυμα "βρέθηκε" και βγαίνει από το βρόχο.
- Μετά την έξοδο του βρόχου, επιστρέφει (πάντα) στην αρχή του δεύτερου μέρους του προγράμματος, γιά να ζητήσει μιά νέα τιμή και να την αναζητήσει στο δένδρο.

7.3(c): Χρήση υπορουτινών.

- Μετατρέψτε το βήμα 7.3(a)(1) σε μια υπορουτίνα "read_int" η οποία δεν παίρνει καμία παράμετρο εισόδου, διαβάζει έναν ακέραιο αριθμό, και επιστρέφει τον αριθμό που διάβασε.
- Μετατρέψτε το βήμα 7.3(a)(3) σε μια υπορουτίνα "node_alloc" η οποία δεν παίρνει καμία παράμετρο εισόδου, ζητάει από το σύστημα να δεσμέυσει ένα κόμβο για το δένδρο, και επιστρέφει τη διεύθυνση της μνήμης που έχει δεσμευθεί, ώστε το πρόγραμμα που καλεί τη node_alloc να εισάγει τον κόμβο στο δένδρο.
- Μετατρέψτε το βήμα 7.3(a)(5) σε μια υπορουτίνα "insert_node" η οποία παίρνει δύο εισόδους. Η πρώτη είσοδος είναι ένας δείκτης (pointer) στη ρίζα του δυαδικού δένδρου και η δεύτερη είσοδος είναι ένας δείκτης (pointer) στον κόμβο που θέλουμε να εισάγουμε στο δένδρο. **Προσοχή:** Η υπορουτίνα insert_node υλοποιεί τα βήματα σύνδεσης 1-5 που αναφέρθηκαν παραπάνω αναδρομικά. Δηλαδή, η υπορουτίνα καλεί τον εαυτό της αλλάζοντας την πρώτη είσοδό της έτσι ώστε να δείχνει στη ρίζα του υποδένδρου στο οποίο ο αλγόριθμος "ψάχνει" εκείνη τη στιγμή, για να βρει το σημείο εισαγωγής του νέου κόμβου.
- Αλλάξτε το πρόγραμμα του 7.3(a) ώστε να χρησιμοποιεί τις ρουτίνες read_int, insert_node και node_alloc. Στη χρήση καταχωρητών από τις read_int, insert_node και node_alloc, όπως και το πρόγραμμα σας που τις καλεί πρέπει να τηρήσετε τις συμβάσεις χρήσης καταχωτητών. Οι read_int, insert_node και node_alloc είναι αρκετά απλές και δεν είναι απαραίτητο να έχουν τοπικές μεταβλητές που να αποθηκεύονται στη στοίβα, ωστόσο θα πρέπει να έχουν η καθε μία το δικό της stack frame, π.χ. για την αποθήκευση της

διεύθυνσης επιστροφής.

- Γράψτε μια υπορουτίνα search_tree που κάνει ότι περιγράφουν τα βήματα 7.3(b)(3i) έως 7.3(b)(3vi). Η υπορουτίνα θα πέρνει ως πρώτη είσοδο τη διεύθυνση της ρίζας του δένδρου και σαν δεύτερη είσοδο την τιμή για την οποία πρέπει να ψάξει το δένδρο. Στη συνέχεια θα υλοποιεί τα βήματα (i, ii, iii, iv, v, vi) χρησιμοποιώντας τη συνάρτηση print_node για τα βήματα (ii, vi).
- Αλλάξτε το πρόγραμμα σας του μέρους 7.3(b) ώστε να καλεί την search_tree (η οποία θα καλεί την print_node). Και πάλι πρέπει να χρησιμοποιήσετε τις συμβάσεις χρήσεις των καταχωρητών μεταξύ των ρουτινών και του προγράμματος που τις καλεί. Επίσης η κάθε ρουτίνα θα πρέπει να έχει το δικό της stack frame.
- Τελικά το συνολικό πρόγραμμα σας θα πρέπει να έχει μια main που αποτελείται απο δύο μέρη: Το πρώτο μέρος της θα υλοποιεί το 7.3(a) με χρήση της node_alloc και read_int, ενώ το δεύτερο θα υλοποιεί το 7.3(b) με χρήση των read_int, search_tree, και print_node.

Για να παραδώσετε την άσκηση, συνδεθείτε σε ένα μηχάνημα Linux του τμήματος με το login και password που χρησιμοποιείτε στο Τμήμα. Η λίστα μηχανημάτων Linux του Τμήματος είναι διαθέσιμη εδώ. Ετοιμάστε ένα directory ένα αρχείο ask7.s με τις απαντήσεις σας. Ας υποθέσουμε ότι το όνομα του directory είναι /somepath/mydir. Μετακινηθείτε στο directory somepath και εκτελέστε την εντολή:

```
/usr/local/bin/submit exercise7@hy225 mydir
```

Η διαδικασία submit θα σας ζητήσει να επιβεβαιώσετε την αποστολή των αρχείων. Περισσότερες πληροφορίες και αναλυτικές οδηγίες για τη διαδικασία submit είναι διαθέσιμες εκτελώντας:

```
man submit
```

σε κάποιο από τα μηχανήματα Linux του Τμήματος. Θα εξεταστείτε **προφορικά** για την Άσκηση 7, από βοηθούς του μαθήματος. Για τη διαδικασία αυτή θα κλείσετε ραντεβού με έναν βοηθό χρησιμοποιώντας το web-based εργαλείο Submit-Rendezvous. Αφού κάνετε login με το όνομα και τον κωδικό σας, επιλέγετε το tab "Rendezvous", κατόπιν "HY225 - Examination of exercise 7" και τέλος "Continue". Θα δείτε μία λίστα με διαστήματα 10 λεπτών που είναι διαθέσιμα για προφορικές συνεντεύξεις με βοηθούς. Επιλέξτε ένα διάστημα και πατήστε "Book". **Προσοχή: το web-based εργαλείο θα το χρησιμοποιήσετε μόνο για να κλείσετε ραντεβού και όχι για να υποβάλετε την άσκηση. Για την υποβολή της άσκησης θα χρησιμοποιήσετε το command-line εργαλείο submit και τη διαδικασία που περιγράψαμε παραπάνω.**

[Up: Table of Contents]
[Prev: 6. Procedure Call]
[printer version, in PDF]
[Next: 8. Verilog Intro. 1]

Up to the Home Page of CS-225