Άσκηση 7:
Πρόγραμμα Διπλά Συνδεδεμένης Λίστας και Διαδικασιών

Προθεσμία έως Παρασκευή 19 Μαρτίου 2010, ώρα 23:59 (βδομάδα 6)

[**Up** - Table of Contents] [printer version - **PDF**]
[**Prev** - 6. Procedure Call] [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 μιάς διπλά συνδεδεμένης λίστας (doubly linked list). Κάθε κόμβος (δομή δεδομένων) μας θα αποτελείται από τρεις λέξεις (των 32 bits καθεμία): έναν ακέραιο "data" που θα περιέχει την "πληροφορία χρήστη", έναν δείκτη σύνδεσης (pointer) "nxtPtr" που θα περιέχει τη διεύθυνση του επόμενου κόμβου στη λίστα (στον τελευταίο κόμβο της λίστας, nxtPtr=0) και έναν δείκτη σύνδεσης (pointer) "prvPtr" που θα περιέχει τη διεύθυνση του προηγούμενου κόμβου στη λίστα (στον πρώτο κόμβο της λίστας, prvPtr=0). Τα τρία στοιχεία (λέξεις) της δομής μας θα βρίσκονται σε διαδοχικές θέσεις (λέξεις) της μνήμης και με τη σιερά "data", "nxtPtr", "prvPtr". Επομένως, κάθε δομή (κόμβος) μας θα έχει μέγεθος 12 (3×4 bytes). Διεύθυνση μιάς δομής είναι η διεύθυνση του πρώτου στοιχείου της, δηλαδή του στοιχείου με "μηδενικό offset", που για μας είναι το "data". Άρα, το δεύτερο στοιχείο της δομής μας, ο "nxtPtr", βρίσκεται στη διεύθυνση που προκύπτει προσθέτοντας 4×1=4 στη διεύθυνση της δομής (κόμβου) και το τρίτο στοιχείο της δομής μας, ο "prvPtr", βρίσκεται στη διεύθυνση που προκύπτει προσθέτοντας 4×2=8 στη διεύθυνση της δομής (κόμβου).

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 (κατά το χρόνο εκτέλεσης, run-time). Γιά το σκοπό αυτό θα χρησιμοποιήσετε την κλήση συστήματος (system call) "sbrk" (set break). Η κλήση αυτή "σπρώχνει" πιο πέρα (προς αύξουσες διευθύνσεις μνήμης) το σημείο "break", το οποίο είναι το όριο δηλαδή πριν από το οποίο οι διευθύνσεις μνήμης που γεννά το πρόγραμμα είναι νόμιμες, και 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". Η κλήση συστήματος "sbrk" περιγράφεται στις σελίδες 112-113 του βιβλίου (Β' τόμος Ελληνικής έκδοσης), και λειτουργεί κατ' αναλογία με τις κλήσεις 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, και σε κάθε περίπτωση στον SPIM, στον οποίο η "sbrk" επιστρέφει πάντα μία διεύθυνση πολλαπλάσιο του 4 –σημειώστε πάντως ότι αυτό δεν ισχύει γενικότερα σε πραγματικούς επεξεργαστές).

Άσκηση 7.3: Κατασκευή και Σάρωση Συνδεδεμένης Λίστας

Γράψτε και τρέξτε στον SPIM, σε Assembly του MIPS, ένα πρόγραμμα που πρώτα θα κατασκευάζει και θα γεμίζει με θετικούς ακέραιους αριθμούς μία διπλά συνδεδεμένη λίστα (doubly linked list), θα την σαρώνει επαναληπτικά, τυπώνοντας κάθε φορά ένα διαφορετικό υποσύνολο των στοιχείων της, και θα υλοποιεί διαγραφές υποσυνόλων των στοιχείων της λίστας. Συγκεκριμένα, το πρόγραμμα πρέπει να τυπώνει από το πρώτο προς το τελευταίο και από το τελευταίο προς το πρώτο όσα στοιχεία της λίστας είναι **μικρότερα** από ένα δοθέντα θετικό αριθμό και μεγαλύτερα του 0 και να διαγράφει επίσης από το πρώτο προς το τελευταίο και από το τελευταίο προς το πρώτο όσα στοιχεία της λίστας είναι **μεγαλύτερα** από ένα δοθέντα θετικό αριθμό. Το πρόγραμμά σας θα κρατάει στον καταχωρητή $s0 (δηλαδή τον $16) τον pointer στην αρχή (στον πρώτο κόμβο) της λίστας, και θα αποτελείται από τα εξής κομμάτια:

7.3(a): Κατασκευή της Λίστας. Χρησιμοποιήστε τον καταχωρητή $s1 ($17) σαν pointer στην ουρά (στον τελευταίο κόμβο) της λίστας. Η αρχικοποίηση γίνεται ζητώντας και παίρνοντας έναν κόμβο από το λειτουργικό σύστημα. Ζητάτε από τον χρήστη να εισάγει έναν ακέραιο από την κονσόλα. Εάν ο ακέραιος είναι μικρότερος ή ίσος του 0 η διαδικασία κατασκευής της λίστας τερματίζει. Αλλιώς, γράφετε τον ακέραιο που έδωσε η χρήστης στο πεδίο "data" και θέτετε nxtPtr=0 (εφόσον ο πρώτος κόμβος της λίστας είναι και ο τελευταίος κόμβος μετά την αρχικοποίηση) και prvPtr=0 (εφόσον δεν υπάρχει "προηγούμενος" κόμβος στη λίστα). Επίσης, θέτετε τις τιμές των καταχωρητών $s0 και $s1 να δείχνουν στον πρώτο κόμβο που κατασκευάσατε, δηλαδή περιέχουν τη διεύθυνσή του. Μετά, μπείτε στο βρόχο ανάγνωσης στοιχείων και κατασκευής της λίστας. Σε κάθε ανακύκλωση αυτού του βρόχου:

1. Διαβάζουμε έναν ακέραιο αριθμό από την κονσόλα.
2. Εάν ο αριθμός αυτός είναι μικρότερος ή ίσος του 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 στην ουρά της λίστας, έτσι ώστε ο δείκτης prvPtr του νέου κόμβου να δείχνει στη διεύθυνση του προηγούμενου κόμβου στη λίστα, ο δείκτης nxtPtr του προηγούμενου κόμβου στη λίστα να δείχνει στη διεύθυνση του νέου κόμβου και ο δείκτης nxtPtr του νέου κόμβου να περιέχει το 0 (εφόσον ο νέος κόμβος είναι πλέον ο τελευταίος στη λίστα).
6. Βάζουμε στον καταχωρητή $s1 τη διεύθυνση του νέου κόμβου, δηλαδή, χρησιμοποιούμε τον καταχωρητή $s1 για να δείχνουμε στον τελευταίο κόμβο στη λίστα.

7.3(b): Σάρωση της Λίστας. Το δεύτερο μέρος του προγράμματος θα διαβάζει έναν μη ανητικό αριθμό και θα τυπώνει όσα στοιχεία της λίστας είναι **μικρότερα** από αυτόν τον αριθμό. Σας ζητείται να υλοποιήσετε **δύο** τρόπους σάρωσης της λίστας: Από την αρχή προς το τέλος, χρησιμοποιώντας τον καταχωρητή $s0 για να βρίσκετε αμέσως την αρχή της λίστας; και από το τέλος προς την αρχή χρησιμοποιώντας τον καταχωρητή $s1 για να βρίσκετε αμέσως το τέλος της λίστας. Δώστε προσοχή και στις δύο περιπτώσεις, στον τρόπο με τον οποίο βρίσκετε την "άκρη" της λίστας, δηλαδή το τελευταίο στοιχείο στην πρώτη περίπτωση και το πρώτο στοιχείο στη δεύτερη περίπτωση. Το μέρος αυτό του προγράμματος κάνει τα εξής:

1. Διαβάζει έναν ακέραιο αριθμό από την κονσόλα και τον αποθηκεύει στον καταχωρητή $s2 ($18). Ελέγχει εάν ο αριθμός είναι μικρότερος του 0 και σε αυτή την περίπτωση επιστρέφει στην αρχή του κώδικα που ζητά έναν ακέραιο από την κονσόλα (δηλ. περιμένουμε το χρήστη να δώσει ένα μη αρνητικό ακέραιο). Εάν ο χρήστης δώσει τον αριθμό 0 η διαδικασία τερματίζεται. Εάν όχι τότε εκτελεί τα παρακάτω βήματα.
2. Εάν η σάρωση γίνεται από την αρχή προς το τέλος της λίστας αρχικοποιεί τον καταχωρητή $s3 ($19) σαν δείκτη (pointer) σάρωσης, να δείχνει στον πρώτο κόμβο της λίστας (τον ξέρουμε από τον καταχωρητή $s0 ($16)). Εάν η σάρωση γίνεται από το τέλος προς την αρχή της λίστας αρχικοποιεί τον καταχωρητή $s3 ($19) σαν δείκτη (pointer) σάρωσης, να δείχνει στον τελευταίο κόμβο της λίστας (τον ξέρουμε από τον καταχωρητή $s1 ($17)).
3. Μπαίνει σε ένα βρόχο, σε κάθε ανακύκλωση του οποίου:
    i. ελέγχει αν τα "data" του κόμβου όπου δείχνει ο $s3 ($19) είναι ή όχι μικρότερα από το περιεχόμενο του $s2 ($18),
    ii. αν είναι μικρότερα τα τυπώνει,
    iii. ελέγχει αν υπάρχει ή όχι επόμενος (nxtPtr, σάρωση προς τα εμπρός) ή προηγούμενος (prvPtr, σάρωση προς τα πίσω) κόμβος στη λίστα,
    iv. αν δεν υπάρχει βγαίνει από το βρόχο,
    v. αν υπάρχει, προχωρεί τον $s3 ($19) να δείξει σε αυτόν τον επόμενο κόμβο (nxtPtr, σάρωση προς τα εμπρός) εάν αυτός υπάρχει ή στον προηγούμενο κόμβο (prvPtr, σάρωση προς τα πίσω) κόμβο εάν αυτός υπάρχει, και επιστρέφει στην αρχή του βρόχου.
4. Μετά την έξοδο του βρόχου, επιστρέφει (πάντα) στην αρχή του δεύτερου μέρους του προγράμματος, γιά να ζητήσει μιά νέα τιμή και να ξανατυπώσει τα μεγαλύτερα από αυτή στοιχεία ή να εξέρθει από τη διαδικασία.

7.3(c): Διαγραφή στοιχείων λίστας. Το τρίτο μέρος του προγράμματος θα διαβάζει έναν μη αρνητικό αριθμό από την κονσόλα και θα διαγράφει όλα τα στοιχεία της λίστας που είναι **μεγαλύτερα** από αυτό τον αριθμό. Σας ζητείται και πάλι να υλοποιήσετε **δύο** τρόπους σάρωσης της λίστας για διαγραφή στοιχείων: Από την αρχή προς το τέλος, χρησιμοποιώντας τον καταχωρητή $s0 για να βρίσκετε αμέσως την αρχή της λίστας; και από το τέλος προς την αρχή χρησιμοποιώντας τον καταχωρητή $s1 για να βρίκσετε αμέσως το τέλος της λίστας. Δώστε προσοχή και στις δύο περιπτώσεις, στον τρόπο με τον οποίο βρίσκετε την "άκρη" της λίστας, δηλαδή το τελευταίο στοιχείο στην πρώτη περίπτωση και το πρώτο στοιχεί στη δεύτερη περίπτωση. Το μέρος αυτό του προγράμματος κάνει τα εξής:

1. Διαβάζει έναν ακέραιο αριθμό από την κονσόλα και τον αποθηκεύει στον καταχωρητή $s2 ($18). Ελέγχει εάν ο αριθμός είναι μικρότερος του 0 και σε αυτή την περίπτωση επιστρέφει στην αρχή του κώδικα που ζητά έναν ακέραιο από την κονσόλα (δηλ. περιμένουμε το χρήστη να δώσει ένα μη αρνητικό ακέραιο). Εάν ο χρήστης δώσει τον αριθμό 0 η διαδικασία τερματίζεται. Εάν όχι τότε εκτελεί τα παρακάτω βήματα.
2. Εάν η σάρωση γίνεται από την αρχή προς το τέλος της λίστας αρχικοποιεί τον καταχωρητή $s3 ($19) σαν δείκτη (pointer) σάρωσης, να δείχνει στον πρώτο κόμβο της λίστας (τον ξέρουμε από τον καταχωρητή $s0 ($16)). Εάν η σάρωση γίνεται από το τέλος προς την αρχή της λίστας αρχικοποιεί τον καταχωρητή $s3 ($19) σαν δείκτη (pointer) σάρωσης, να δείχνει στον τελευταίο κόμβο της λίστας (τον ξέρουμε από τον καταχωρητή $s1 ($17)).
3. Μπαίνει σε ένα βρόχο, σε κάθε ανακύκλωση του οποίου:
    i. ελέγχει αν τα "data" του κόμβου όπου δείχνει ο $s3 ($19) είναι ή όχι μεγαλύτερα από το περιεχόμενο του $s2 ($18),
    ii. αν είναι μικρότερα ή ίσα από το περιεχόμενο του $s2 τα τυπώνει και προχωρά στο βήμα iii., αλλιώς διαγράφει τον κόμβο από την λίστα ως εξής:
        1. θέτει τον δείκτη nxtPtr του προηγούμενου κόμβου στη λίστα, αν αυτός υπάρχει (μας το δείχνει ο δείκτης prvPtr του κόμβου που πρόκειται να διαγράψουμε) στην τιμή του δείκτη nxtPtr του κόμβου που πρόκειται να διαγράψουμε,
        2. θέτει τον δείκτη prvPtr του κόμβου που έπεται στη λίστα, αν αυτός υπάρχει (μας τον δείχνει ο δείκτης nxtPtr του κόμβου που πρόκειται να διαγράψουμε) στην τιμή του δείκτη prvPtr του κόμβου που πρόκειται να διαγράψουμε,
        3. αλλάζει τον δείκτη που περιέχει ο $s3 ($19) ώστε να δείχνει στον επόμενο κόμβο στη λίστα από αυτόν που πρόκειται να διαγράψουμε αν κάνουμε σάρωση προς τα εμπρός (nxtPtr του κόμβου που πρόκειται να διαγράψουμε) ή στον προηγούμενο κόμβο στη λίστα από αυτόν που πρόκειται να διαγράψουμε αν κάνουμε σάρωση προς τα πίσω (prvPtr του κόμβου που πρόκειται να διαγράψουμε).
        4. θέτει τους δείκτες prvPtr και nxtPtr του κόμβου που πρόκειται να διαγράψουμε στο 0, ολοκληρώνοντας έτσι τη διαγραφή.
        5. εάν υπάρχει επόμενος κόμβος (δηλαδή ο $s3 έχει τιμή διαφορετική του 0 στη σάρωση προς τα εμπρός) ή προηγούμενος κόμβος (δηλαδή ο $s3 έχει τιμή διαφορετική του 0 στη σάρωση προς τα πίσω) επιστρέφει στο βήμα i. (αρχή του βρόχου), αλλιώς βγαίνει από το βρόχο. (**δώστε ιδιαίτερη προσοχή στην υλοποίηση της ανακύκλωσης, εφόσον η ανακύκλωση γίνεται με διαφορετικό τρόπο στην περίπτωση που γίνεται διαγραφή από την περίπτωση που δεν γίνεται διαγραφή**).
    iii. ελέγχει αν υπάρχει ή όχι επόμενος (nxtPtr != 0, σάρωση προς τα εμπρός) ή προηγούμενος (prvPtr != 0, σάρωση προς τα πίσω) κόμβος στη λίστα,
    iv. αν δεν υπάρχει βγαίνει από το βρόχο,
    v. αν υπάρχει, προχωρεί τον $s3 ($19) να δείξει σε αυτόν τον επόμενο (nxtPtr, σάρωση προς τα εμπρός) ή προηγούμενο (prvPtr, σάρωση προς τα πίσω) κόμβο και επιστρέφει στην αρχή του βρόχου.

7.3(d): Χρήση υπορουτινών.

- Μετατρέψτε το βήμα 7.3(a)(1) σε μια διαδικασία "read_int" η οποία δεν παίρνει καμία παράμετρο εισόδου, διαβάζει έναν ακέραιο αριθμό από την κονσόλα, και επιστρέφει τον αριθμό που διάβασε.
- Μετατρέψτε το βήμα 7.3(a)(3) σε μια διαδικασία "node_alloc" η οποία δεν παίρνει καμία παράμετρο εισόδου, ζητάει από το σύστημα να δεσμεύσει ένα κόμβο για τη λίστα, και επιστρέφει τη διεύθυνση της μνήμης που έχει δεσμευθεί, ώστε το πρόγραμμα που καλεί τη node_alloc να εισάγει τον κόμβο στη λίστα.
- Αλλάξτε το πρόγραμμα του 7.3(b) ώστε να χρησιμοποιεί τις ρουτίνες read_int και node_alloc. Στη χρήση καταχωρητών από τις read_int και node_alloc, όπως και το πρόγραμμα σας που τις καλεί πρέπει να τηρήσετε τις συμβάσεις χρήσης καταχωτητών. Οι read_int 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 print_node που κάνει ότι περιγράφουν τα βήματα 7.3(b)(3ii), 7.3(c)(3ii). Η υπορουτίνα θα παίρνει ως είσοδο τη διεύθυνση ενός κόμβου και δεν θα επιστρέφει τίποτε.
- Γράψτε μια υπορουτίνα search_list_forward που υλοποιεί όλο το βήμα 7.3(b)(3) με σάρωση προς τα εμπρός. Η ρουτίνα θα παίρνει ως πρώτη είσοδο τη διεύθυνση του πρώτου κόμβου τις λίστας (δείκτης σάρωσης) και σαν δεύτερη είσοδο την τιμή με την οποία πρέπει να συγκρίνει τα περιεχόμενα της λίστας. Στη συνέχεια θα υλοποιεί τα βήματα (i,ii,iii,iv,v) χρησιμοποιώντας τη συνάρτηση print_node για

το βήμα (ii).

- Γράψτε μια υπορουτίνα search_list_backward που υλοποιεί όλο το βήμα 7.3(b)(3) με σάρωση προς τα πίσω. Η ρουτίνα θα παίρνει ως πρώτη είσοδο τη διεύθυνση του τελευταίου κόμβου τις λίστας (δείκτης σάρωσης) και σαν δεύτερη είσοδο την τιμή με την οποία πρέπει να συγκρίνει τα περιεχόμενα της λίστας. Στη συνέχεια θα υλοποιεί τα βήματα (i,ii,iii,iv,v) χρησιμοποιώντας τη συνάρτηση print_node για το βήμα (ii).
- Γράψτε μια υπορουτίνα delete_list_forward που υλοποιεί όλο το βήμα 7.3(c)(3) με σάρωση προς τα εμπρός. Η ρουτίνα θα παίρνει ως πρώτη είσοδο τη διεύθυνση του πρώτου κόμβου τις λίστας (δείκτης σάρωσης) και σαν δεύτερη είσοδο την τιμή με την οποία πρέπει να συγκρίνει τα περιεχόμενα της λίστας για να αποφασίσει αν πρέπει να τα διαγράψει. Στη συνέχεια θα υλοποιεί τα βήματα (i,ii,iii,iv,v) χρησιμοποιώντας τη συνάρτηση print_node για το βήμα (ii).
- Γράψτε μια υπορουτίνα delete_list_backward που υλοποιεί όλο το βήμα 7.3(c)(3) με σάρωση προς τα πίσω. Η ρουτίνα θα παίρνει ως πρώτη είσοδο τη διεύθυνση του τελευταίου κόμβου τις λίστας (δείκτης σάρωσης) και σαν δεύτερη είσοδο την τιμή με την οποία πρέπει να συγκρίνει τα περιεχόμενα της λίστας για να αποφασίσει αν πρέπει να τα διαγράψει. Στη συνέχεια θα υλοποιεί τα βήματα (i,ii,iii,iv,v) χρησιμοποιώντας τη συνάρτηση print_node για το βήμα (ii).
- Αλλάξτε το πρόγραμμα σας του μέρους 7.3(b) ώστε να καλεί τις search_list_forward, search_list_backward (οι οποίες θα καλούν 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. Τέλος, πρέπει να καλέσετε και τις δύο διαδικασίες (σάρωση προς τα εμπρός και μετά σάρωση προς τα πίσω).
- Αλλάξτε το πρόγραμμα σας του μέρους 7.3(c) ώστε να καλεί τις delete_list_forward, delete_list_backward (οι οποίες θα καλούν 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. Τέλος, πρέπει να καλέσετε και τις δύο διαδικασίες (σάρωση προς τα εμπρός και μετά σάρωση προς τα πίσω).
- Τελικά το συνολικό πρόγραμμα σας θα πρέπει να έχει μια main που αποτελείται απο τρία μέρη: Το πρώτο μέρος της θα υλοποιεί το 7.3(a) με χρήση της node_alloc και read_int, το δεύτερο θα υλοποιεί το 7.3(b) με χρήση των read_int, search_list_forward, search_list_backward και print_node και το τρίτο μέρος θα υλοποιεί το 7.3(c) με χρήση των read_int, delete_list_forward, delete_list_backward.

Τρόπος Παράδοσης: Παραδώστε ηλεκτρονικά τον κώδικά σας, σε ένα αρχείο "**`ask7.s`**", και **πέντε στιγμιότυπα** της εκτέλεσής του στον SPIM, "**`ask7.1.jpg, ask7.2.jpg, ask7.3.jpg, ask7.4.jpg, ask7.5.jpg`**". Στα στιγμιότυπα αυτά θα πρέπει να δείχνετε τα περιεχόμενα της μνήμης και τα περιεχόμενα της κονσόλας μετά την κατασκευή μίας μικρής λίστας (ask7.1.jpg), τα περιεχόμενα της κονσόλας και της μνήμης μετά από μία σάρωση προς τα εμπρός για την άσκηση 7.3(b) (ask7.2.jpg), τα περιεχόμενα της κονσόλας και της μνήμης μετά από μία σάρωση προς τα πίσω για την άσκηση 7.3(b) (ask7.3.jpg), τα περιεχόμενα της κονσόλας και της μνήμης μετά από μία διαγραφή με σάρωση προς τα εμπρός για την άσκηση 7.3(c) (ask7.4.jpg) και τα περιεχόμενα της κονσόλας και της μνήμης μετά από μία διαγραφή με σάρωση προς τα πίσω για την άσκηση 7.3(c) (ask7.5.jpg). Παραδώστε ένα ολοκληρωμένο πρόγραμμα με όλες τις διαδικασίες (node_alloc, read_int, print_int, search_list_forward, search_list_backward) και τις κλήσεις τους και όχι στιγμιότυπα του προγράμματος στα οποία δεν έχετε συμπεριλάβει τις λειτουργίες του προγράμματος σε διαδικασίες. Επίσης, θα εξεταστείτε προφορικά σε αυτή την άσκηση. Βάλτε τα έξι παραπάνω αρχεία στο αρχείο "**`ask7.zip`**".

Για να παραδώσετε την άσκηση χρησιμοποιήστε την ακόλουθη διαδικασία. 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 με το αρχείο που σας ζητάει η άσκηση (ask7.zip). 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 νέα διαδικασία submit είναι διαθέσιμες εκτελώντας:

```
man submit
```

σε κάποιο από τα μηχανήματα Linux του Τμήματος. Σημειώστε ότι η απομακρυσμένη πρόσβαση στα μηχανήματα του Τμήματος γίνεται από 1.2.2010 με νέο ασφαλή τρόπο μέσω gateways και με χρήση tunnel.

Για την προφορική εξέταση 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 λίγων λεπτών που είναι διαθέσιμα για προφορικές συνεντεύξεις με βοηθούς. Επιλέξτε ένα διάστημα και πατήστε "Book". Οι βοηθοί που θα εξετάσουν την άσκηση είναι οι Μιχάλης Αλβανός (alvanos) και Νίκος Φτυλιτάκης (ftylitak). Η προφορική εξέταση θα γίνει στα γραφεία των μεταπτυχιακών στο υπόγειο των λευκών κτιρίων. **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 - **PDF**]
[8. Introduction to Verilog - **Next**]

Up to the Home Page of CS-225