HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2025

Τμ. Επ. Υπολογιστών
© Πανεπιστήμιο Κρήτης

Σειρά Ασκήσεων 2: Βρόχοι και Επικοινωνία Κονσόλας στον RARS, Προσημασμένοι και Επέκταση Προσήμου

Από 1η για τέλος 2ης εβδομάδας του Εξαμήνου

Βιβλίο: Προσημασμένοι αριθμοί: Διαβάστε την §2.4 (σελίδες 115-121), και ξαναθυμηθείτε το Εργαστήριο 6 της Ψηφιακής Σχεδίασης.

2.1 Ολίγα περί Εντολών Διακλάδωσης υπό Συνθήκη

Όπως λέγαμε και στην §1.3, για να εκτελεστεί ένα πρόγραμμα, οι εντολές του γράφονται στην κεντρική μνήμη η μία "κάτω" από την άλλη, δηλαδή σε συνεχόμενες θέσεις (διευθύνσεις) μνήμης. Μετά την ανάγνωση και εκτέλεση μιας εντολής, ο επεξεργαστής αυξάνει τον PC κατά το μέγεθος της εντολής που εκτελέστηκε (κατά 4 στον βασικό RISC-V, αφού οι εντολές του έχουν μέγεθος 4 Bytes), οπότε αυτός (ο PC) δείχνει στην επόμενη (την "από κάτω") εντολή. Η σειριακή αυτή εκτέλεση εντολών διακόπτεται όταν εκτελείται μιά εντολή **μεταφοράς ελέγχου** (CTI - control transfer instruction). Είδαμε ήδη μία τέτοια, την (ψεύδο)εντολή άλματος `j label` ("jump" to label), που κάνει ώστε η επόμενη εντολή που θα εκτελεστεί να είναι η εντολή στη διεύθυνση μνήμης `label`, αντί να είναι η "από κάτω" εντολή. Με άλλα λόγια, η (ψεύδο)εντολή `j label` φορτώνει τη διεύθυνση `label` στον καταχωρητή PC. Χρησιμοποιώντας αυτήν στην άσκηση 1 φτιάξαμε έναν "άπειρο βρόχο", δηλαδή κάναμε τον υπό προσομοίωση υπολογιστή να εκτελεί συνεχώς το ίδιο "μπλόκ" εντολών.

Γιά να φτιάξουμε έναν κανονικό (όχι άπειρο) βρόχο χρειαζόμαστε μια εντολή **διακλάδωσης υπό συνθήκη** (conditional branch), δηλαδή μια εντολή που μερικές φορές προκαλεί διακλάδωση και μερικές φορές όχι, ανάλογα με το αν ισχύει ή δεν ισχύει κάποια κατάλληλη συνθήκη. Η βασική τέτοια εντολή είναι η **`beq`** (branch if equal): Η εντολή "`beq x26, x27, label`" διαβάζει τους καταχωρητές 26 και 27, και τους συγκρίνει. Εάν τους βρει ίσους (equal) διακλαδίζεται στη θέση label, δηλαδή κάνει τον επεξεργαστή να διαβάσει και εκτελέσει την εντολή από εκείνη τη διεύθυνση σαν επόμενη εντολή. Αλλιώς, δεν κάνει τίποτα το ξεχωριστό, οπότε σαν επόμενη εντολή θα διαβαστεί και εκτελεστεί η "από κάτω" εντολή. Η εντολή **`bne`** (branch if not equal) κάνει τα ανάποδα, δηλαδή διακλαδίζεται εάν βρει τους καταχωρητές άνισους (not equal), αλλιώς συνεχίζει "από κάτω".

2.2 Κώδικας Βρόχου και Εισόδου/Εξόδου Κονσόλας

Για να επικοινωνούν τα προγράμματα με τον έξω κόσμο, ο RARS προσομοιώνει μερικές υποτυπώδεις υπηρεσίες λειτουργικού συστήματος για

είσοδο/έξοδο (I/O) στην "κονσόλα" (ένα απλό τερματικό ASCII). Οι κλήσεις στο λειτουργικό σύστημα λέγονταν παραδοσιακά "system call", αλλά ο RISC-V υιοθετεί το γενικότερο όνομα "environment call (ecall)" διότι μπορούν να χρησιμοποιηθούν και γιά την κλήση άλλων περιβαλλόντων ελέγχου (π.χ. hypervisor κλπ). Δεν είναι ανάγκη προς στιγμήν να καταλάβετε όλες τις λεπτομέρειες του πώς γίνονται αυτές οι κλήσεις –αρκεί να μιμηθείτε το παρακάτω παράδειγμα και να καταλάβετε τις εξηγήσεις που δίνονται κάτω από αυτό. **Μελετήστε και αντιγράψτε** σε ένα αρχείο (π.χ. "`ex02.asm`") τον παρακάτω κώδικα –ή διάφορες παραλλαγές του που προτιμάτε– και τρέξτε τον στον RARS:

```
                # compute s = 1+2+3+...+(n-1),  for n>=2
                # register x26: n
                # register x27: s
                # register x28: i

        .data           # init. data memory with the strings needed:
str_n:  .asciz "n = "
str_s:  .asciz "       s = "
str_nl: .asciz "\n"

        .text           # program memory:

main:                       # (1) PRINT A PROMPT:
        addi    x17, x0, 4      # environment call code for print_string
        la      x10, str_n      # pseudo-instruction: address of string
        ecall                   # print the string from str_n
                            # (2) READ n (MUST be n>=2 --not checked!):
        addi    x17, x0, 5      # environment call code for read_int
        ecall                   # read a line containing an integer
        add     x26, x10, x0    # copy returned int from x10 to n
                            # (3) INITIALIZE s and i:
        add     x27, x0, x0     # s=0;
        addi    x28, x0, 1      # i=1;
loop:                       # (4) LOOP starts here
        add     x27, x27, x28   # s=s+i;
        addi    x28, x28, 1     # i=i+1;
        bne     x28, x26, loop  # repeat while (i!=n)
                            #     LOOP ENDS HERE
                            # (5) PRINT THE RESULT:
        addi    x17, x0, 4      # environment call code for print_string
        la      x10, str_s      # pseudo-instruction: address of string
        ecall                   # print the string from str_s
        addi    x17, x0, 1      # environment call code for print_int
        add     x10, x27, x0    # copy argument s to x10
        ecall                   # print the integer in x10 (s)
        addi    x17, x0, 4      # environment call code for print_string
        la      x10, str_nl     # pseudo-instruction: address of string
        ecall                   # print a new-line
                            # (6) START ALL OVER AGAIN (infinite loop)
        j       main            # unconditionally jump back to main
```

Ο κώδικας αυτός υπολογίζει το άθροισμα s=1+2+3+...+(n-1), για n μεγαλύτερο ή ίσο του 2 --προσοχή: αν δοθεί n μικρότερο του 2, ο κώδικας θα μπει σε (σχεδόν) άπειρο βρόγχο! Η "καρδιά" του κωδικά είναι τα κομμάτια (3) --

αρχικοποιήσεις-- και (4) --βρόγχος υπολογισμού. Προσέξτε τις παρακάτω εξηγήσεις:

- Το κομμάτι κάθε γραμμής μετά το # είναι σχόλια, όπως είπαμε και στην άσκηση 1 (όμως, το βιβλίο χρησιμοποιεί "//" γιά τα σχόλια).
- Οι γραμμές που αρχίζουν με τελεία (".") είναι **οδηγίες** (Directives) προς τον Assembler, και όχι εντολές Assembly του RISC-V. Ο πλήρης κατάλογος των οδηγιών που δέχεται ο RARS βρίσκεται μέσω του Help του RARS, στην καρτέλα RISCV→Directives, σε περίπτωση που θέλετε να τον συμβουλευθείτε.
- Η οδηγία **`.data`** σημαίνει πως ό,τι ακολουθεί είναι δεδομένα (και όχι εντολές), και πρέπει να τοποθετηθούν στο κομμάτι της μνήμης που προορίζεται για αυτά (Data segment) (στον RARS αυτό αρχίζει από τη διεύθυνση 10010000 δεκαεξαδικό).
- Η οδηγία **`.asciz`** σημαίνει να αρχικοποιήσει ο Assembler τις επόμενες θέσεις (bytes) μνήμης με το ASCII string που ακολουθεί, τερματισμένο με ένα NULL byte όπως και στην C. Οι ετικέτες (labels) `str_n`, `str_s`, και `str_nl`, ακολουθούμενες από άνω-κάτω τελεία ":", ορίζουν την κάθε ετικέτα σαν την διεύθυνση μνήμης όπου ο Assembler βάζει το αντίστοιχο string (τη διεύθυνση μνήμης του πρώτου byte του string).
- Η οδηγία **`.text`** σημαίνει, όπως είπαμε και στην άσκηση 1, πως ό,τι ακολουθεί είναι εντολές (και όχι δεδομένα), και πρέπει να τοποθετηθούν στο κομμάτι της μνήμης που προορίζεται για αυτές (Text segment).

- Το κομμάτι **(1)** του κώδικα είναι ένα κάλεσμα περιβάλλοντος (ecall – environment call) (π.χ. λειτουργικού συστήματος) προκειμένου να τυπωθεί το string `str_n` στην κονσόλα "Run I/O", κάτω αριστερά (πρόκειται για το string "n = " που ορίστηκε παραπάνω). Για να καταλάβει το λειτουργικό σύστημα ποιο από όλα τα ecall's ζητάμε, βάζουμε στον καταχωρητή x17 σαν παράμετρο (argument) τον αριθμό 4, που σημαίνει ότι ζητάμε το ecall υπ' αριθμό 4, που είναι το print_string. Τα ecall's που υλοποιεί ο RARS περιγράφονται μέσω του Help του RARS, στην καρτέλα RISCV→Syscalls.
- Επίσης, για να ξέρει το περιβάλλον ποιο string θέλουμε να τυπώσει στην κονσόλα, βάζουμε στον καταχωρητή x10 σαν παράμετρο (argument) τη διεύθυνση μνήμης αυτού του string (δηλ. έναν pointer σε αυτό το string), που στην περίπτωσή μας είναι η ετικέτα `str_n` που ορίσαμε παραπάνω: το "**`la`**" είναι **ψεύδο**εντολή (pseudoinstruction) του Assembler του RARS, και όχι κανονική εντολή του RISC-V, και λέει στον Assembler να γεννήσει μία ή δύο πραγματικές εντολές που τοποθετούν τη διεύθυνση της ετικέτας `str_n` στον καταχωρητή x10, ανάλογα αν η διεύθυνση αυτή χωρά ή όχι στα περιορισμένα bits μιάς σταθεράς "immediate" όπως θα δούμε αργότερα.
- Το κομμάτι **(2)** του κώδικα είναι ένα ανάλογο κάλεσμα (το κάλεσμα υπ' αριθμό 5, δηλαδή read_int), που περιμένει να διαβάσει έναν ακέραιο από την κονσόλα: ο προσομοιωτής θα περιμένει εκεί μέχρι να πληκτρολογήσετε έναν ακέραιο και ένα ENTER (RETURN) στο παράθυρο "Run I/O", κάτω αριστερά. Μέσω της επόμενης εντολής, `add`, ο ακέραιος που επιστρέφει το κάλεσμα (στον καταχωρητή x10) αρχικοποιεί τη μεταβλητή μας *n* (στον καταχωρητή x26).

- Το κομμάτι **(3)** του κώδικα είναι η αρχικοποίηση των μεταβλητών *s* (καταχωρητής x27) και *i* (καταχωρητής x28) πριν μπούμε στο βρόχο.
- Το κομμάτι **(4)** του κώδικα είναι ο κυρίως βρόχος υπολογισμού. Σε κάθε επανάληψή του αυξάνει το *s* κατά *i* και το *i* κατά 1, και στη συνέχεια συγκρίνει το *i* (καταχωρητής x28) με το *n* (καταχωρητής x26) και διακλαδίζεται (πηγαίνει) πίσω στην ετικέτα `loop`, δηλαδή στην αρχή του βρόχου, όσο αυτές οι δύο μεταβλητές δεν είναι ίσες μεταξύ τους, δηλαδή όσο το *i* δεν έφτασε ακόμα το *n*. Αλλιώς, μόλις το (ήδη αυξημένο) *i* γίνει ίσο με *n*, δεν διακλαδιζόμαστε πίσω, αλλά συνεχίζουμε με την επόμενη εντολή, δηλαδή το κομμάτι (5) του κώδικα.
- Το κομμάτι **(5)** του κώδικα είναι τρία καλέσματα περιβάλλοντος για να τυπωθούν: το string `str_s`, το αποτέλεσμα *s*, και το string `str_nl`. Τέλος, η εντολή jump στο (6) μας επιστρέφει πάντα πίσω στο `main`, ώστε το πρόγραμμα να ξανατρέξει από την αρχή και συνεχώς μέχρι να τερματίσετε τον RARS.

Άσκηση 2.3: Τρέξιμο στον RARS

- Ξεκινήστε τον **RARS** με τον τρόπο που είπαμε στην §1.4, και φορτώστε το αρχείο με το παραπάνω πρόγραμμα που γράψατε μέσω του File→Open (πάνω αριστερά). Στη συνέχεια, μεταφράστε το πρόγραμμα μέσω του κουμπιού "Assemble" (εικονίδιο με κατσαβίδι και κλειδί γιά μπουλόνια).
- Μέσω του κουμπιού "**Single step**" ζητήστε single-stepping, δηλαδή να εκτελούνται μια-μια οι εντολές και να τις βλέπετε. Η εκτέλεση αρχίζει στη διεύθυνση "main" (0x00400000). Όταν φτάσετε στο δεύτερο κάλεσμα συστήματος, μην ξεχάσετε να πληκτρολογήσετε έναν ακέραιο μεγαλύτερο ή ίσο του 2 (πρέπει να τον πληκτρολογήσετε *αφού* ο RARS φτάσει εκεί --παλαιότερες πληκτρολογήσεις αγνοούνται).
- Αφού βαρεθείτε να βλέπετε τις εντολές να εκτελούνται μία-μία, ορίστε "**breakpoint(s)**" στο πρόγραμμά σας: τσεκάρετε το κουτί "Bkpt" αριστερά από την επιθυμητή εντολή στο Text segment. Μετά, πείτε στο πρόγραμμα να τρέξει, μέσω του κουμπιού "Run" (δεξιό πράσινο βέλος), οπότε αυτό τρέχει "σιωπηλά" μέχρι να (ξανα)φτάσει αμέσως πριν την εκτέλεση της επόμενης εντολής που έχει οριστεί σαν breakpoint. Έτσι μπορείτε να επιταχύνετε την παρακολούθηση ενός προγράμματος, και να το κάνετε να σταματάει σε "ενδιαφέροντα" ή "ύποπτα" σημεία.
- Τέλος, αφαιρέστε όλα τα breakpoints και τρέξτε το πρόγραμμα κανονικά, οπότε θα βλέπετε μόνο τις εισόδους και εξόδους στην κονσόλα.

Εάν δεν σας δουλεύει ο RARS και πρέπει να χρησιμοποιήστε τον Venus:
Εάν δεν καταφέρατε να εγκαταστήσετε την Java και τον RARS, και θέλετε να κάνετε αυτή την άσκηση με τον www.kvakil.me/venus/ που λέγαμε στην §1.5, θα πρέπει να κάνετε τρείς αλλαγές στο πρόγραμμα σας: Πρώτον, η οδηγία ".asciz" του RARS γράφεται με δύο "i" στον Venus: ".asciiz". Δεύτερον, ο κωδικός του ecall περνιέται μέσω του καταχωρητή x10 στον Venus (αντί μέσω του x17 στον RARS), και το όρισμα (argument) του ecall περνιέται μέσω του x11 στον Venus (αντί μέσω του x10 στον RARS) –δείτε τις οδηγίες στο github.com/kvakil/venus/wiki (item: Environmental Calls). Και τρίτον, ο Venus δεν

έχει ecall γιά ανάγνωση εισόδου (user input), άρα στη θέση του read_int ecall θα πρέπει να γράψετε μιάν εντολή π.χ. `addi x26, x0, 77` που να αρχικοποιεί π.χ. n=77, και κάθε φορά που θέλετε να τρέξετε το πρόγραμμά σας με άλλη τιμή του n να αλλάζετε με τον Editor το "77" μέσα στο πρόγραμμα στη νέα τιμή που θέλετε.

Τρόπος Παράδοσης:
Θα παραδώσετε ηλεκτρονικά ένα στιγμιότυπο της οθόνης καθώς τρέχετε το πρόγραμμα "**RARS**" και αυτό βρίσκεται σ' ένα "ενδιαφέρον" ενδιάμεσο breakpoint της επιλογής σας (όχι στην αρχή και όχι στο τέλος του προγράμματος). Το στιγμιότυπο μπορείτε να το πάρετε π.χ. όπως παρακάτω, θα το ονομάσετε **ex02.jpg**, και θα το παραδώσετε ως εξής:

- Σε μηχανή **Windows**, πατήστε το κουμπί "*Print Screen*" του πληκτρολογίου, και μετά ανοίξτε το *Microsoft Photo Editor* και κάνετε "*paste*". Σώστε την εικόνα αυτή σε μορφή JPEG (.jpg), με το όνομα "**ex02.jpg**".
- Σε μηχανή **MacOS**, πατήστε μαζί τα 3 κουμπιά **Cmd-Shift-4** και έπειτα κρατώντας πατημένο το **Space** κάντε κλικ στο παράθυρο. Το screenshot θα εμφανιστεί στο Desktop by default, αλλιώς στο φάκελο που έχετε ρυθμίσει εσείς να αποθηκεύονται τα screenshots.
- Σε μηχανή **LINUX με X-windows**, ένας τρόπος είναι πάλι το κουμπί "*Print Screen*", που, by default, αποθηκεύει το στιγμιότυπο στο Desktop. Εάν έχετε το (παλαιότερο) πρόγραμμα "*xv*", τότε με την επιλογή του "*Grab*" (σας επιτρέπει να καθορίσετε και τον χρόνο που θα περάσει μέχρι να παρθεί το στιγμιότυπο), διαλέξτε το παράθυρο και σώστε την εικόνα σε μορφή .jpg σε αρχείο με το όνομα "**ex02.jpg**".

Γιά να παραδώσετε τις ασκήσεις σας γενικά και αυτήν εδώ ειδικά, συνδεθείτε σε ένα μηχάνημα Linux του Τμήματος. Ετοιμάστε ένα directory με το(α) αρχείο(α) που σας ζητάει η άσκηση. Ας υποθέσουμε ότι το όνομα του directory είναι `[somepath]/mydir`. Μετακινηθείτε στο directory `[somepath]`, και εκτελέστε την εντολή:

```
turnin ex02@hy225 mydir
```

Η διαδικασία *turnin* θα σας ζητήσει να επιβεβαιώσετε την αποστολή των αρχείων. Περισσότερες πληροφορίες και αναλυτικές οδηγίες για τη διαδικασία *turnin* είναι διαθέσιμες στην ιστοσελίδα https://www.csd.uoc.gr/index.jsp?custom=use_the_turnin ή εκτελώντας **`man turnin`** σε κάποιο από τα μηχανήματα Linux του Τμήματος. Γιά επαλήθευση της υποβολής μπορείτε να εκτελέσετε: **`verify-turnin ex02@hy225`**

2.4 Προσημασμένοι Αριθμοί, Επέκταση Προσήμου

Οι σημερινοί επεξεργαστές παριστάνουν τους προσημασμένους (signed) αριθμούς σε κωδικοποίηση συμπληρώματος ως-προς-2, όπως είχαμε δεί στο μάθημα της Ψηφιακής Σχεδίασης (HY-120, ενότητα 6.3). Σε αυτή την αναπαράσταση, η μετατροπή προσημασμένου (signed) αριθμού από λιγότερα σε περισσότερα bits γίνεται με την εξής τεχνική, που ονομάζεται "*επέκταση προσήμου*" *(sign extension)* και αποδεικνύεται μαθηματικά ως εξής:

Έστω ο προσημασμένος ακέραιος $A_{s,k}$ με *k bits*, τον οποίο θέλουμε να μετατρέψουμε στον ίδιο αριθμό $A_{s,n}$ με *n bits*: $A_{s,n} = A_{s,k}$ όπου n>k. Εάν τα bits του $A_{s,k}$ τα ερμηνεύσουμε σαν μη προσημασμένο (unsigned) ακέραιο, τότε θα μοιάζουν με (θα δηλώνουν) έναν αριθμό που ας το ονομάσουμε $A_{u,k}$ και ομοίως εάν τα bits του $A_{s,n}$ τα ερμηνεύσουμε σαν unsigned τότε θα μοιάζουν με τον $A_{u,n}$. Εάν ο $A_{s,k} = A_{s,n}$ είναι μη αρνητικός (δηλ. θετικός ή μηδέν), τότε, κατά τον ορισμό της κωδικοποίησης συμπληρώματος ως-προς-2, (α) το αριστερό bit τους θα είναι μηδέν, και (β) οι απρόσημες ερμηνίες τους θα είναι: $A_{u,k} = A_{s,k}$ και $A_{u,n} = A_{s,n}$, και αφού $A_{s,n} = A_{s,k}$ τότε θα είναι και: $A_{u,n} = A_{u,k}$. Αυτοί οι δύο τελευταίοι, αφού είνα unsigned ακέραιοι, με n και k bits αντίστοιχα, και είναι ίσοι μεταξύ τους, προκύπτει ότι ο $A_{u,n}$ που έχει περισσότερα bits θα είναι ίδιος με τον $A_{u,k}$ αλλά με n-k μηδενικά προστεθημένα αριστερά από τον $A_{u,k}$.

Αλλιώς, εάν οι $A_{s,n} = A_{s,k}$ είναι αρνητικοί, τότε, κατά τον ορισμό μας, (α) το αριστερό bit τους θα είναι ένα, και (β) οι απρόσημες ερμηνίες τους θα είναι: $A_{u,k} = A_{s,k} + 2^k$ και $A_{u,n} = A_{s,n} + 2^n$. Δεδομένου ότι: $A_{s,n} = A_{s,k}$ προκύπτει ότι θα είναι και: $A_{u,n} - 2^n = A_{u,k} - 2^k$. Επομένως, η αναπαράσταση που ψάχνουμε είναι η: $A_{u,n} = A_{u,k} - 2^k + 2^n = A_{u,k} + (2^n - 2^k) = (2^{n-k} - 1) \cdot 2^k + A_{u,k}$. Σε αυτήν την τελευταία έκφραση, ο μεν αριστερός προσθετέος, $(2^{n-k} - 1) \cdot 2^k$, αποτελείται από (n-k) το πλήθος άσσους (που είναι ο αριθμός $2^{n-k} - 1$) ολισθημένους αριστερά κατά k θέσεις bits (που είναι ο πολλαπλασιασμός επί 2^k), ο δε δεξιός προσθετέος, $A_{u,k}$, είναι τα αρχικά k bits του αρχικού αριθμού που μας δόθηκε. Δεδομένου ότι ο πρώτος προσθετέος έχει όλο μηδενικά στις k δεξιές θέσεις, ο δε δεύτερος προσθετέος έχει όλο μηδενικά στις (n-k) αριστερές θέσεις, το άθροισμά τους θα είναι απλώς η "συγκόλληση" (concatenation) των δύο αυτών ποσοτήτων. Επομένως, η αναπαράσταση με n bits του αρχικού αριθμού που μας δόθηκε θα αποτελείται από τα αρχικά k bits, δεξιά, μαζί με (n-k) άσσους κολλημένους ακριβώς αριστερά τους.

Συνολικά λοιπόν, γιά να μετατρέψουμε ένα προσημασμένο (signed) ακέραιο από k (λιγότερα) bits σε n (περισσότερα) bits, δεν έχουμε παρά να κάνουμε το εξής: Τοποθετούμε αριστερά από τα bits που μας δόθηκαν (n-k) επιπλέον bits τα οποία είναι όλα τους *αντίγραφα του αριστερού* (most significant) bit του αριθμού που μας δόθηκε, δηλαδή αντίγραφα του bit που υποδεικνύει το πρόσημο του δοθέντος αριθμού (0 γιά θετικούς ή το μηδεν, 1 γιά αρνητικούς). Η πράξη αυτή ονομάζεται επομένως, προφανώς, *επέκταση προσήμου* (sign extension).
