ΜΥΥ-402 Αρχιτεκτονική Υπολογιστών
MIPS, ροή προγράμματος

Αρης Ευθυμίου

Το σημερινό μάθημα

- Εντολές διακλάδωσης
  - με συνθήκη, άλματα
- Μετατροπή σε assembly if-then, if-then-else
- Επαναλήψεις σε assembly
  - while, do-while
- Λογικές πράξεις
  - and, or, nor
  - ολισθήσεις: sll, srl, sra
- Σύγκριση ανισότητας
  - slt
  - Υλοποίηση διακλαδώσεων με συνθήκη ανισότητας

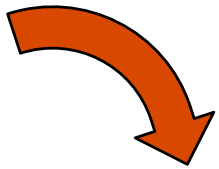
Λήψη αποφάσεων

- Εντολές για λήψη απόφασης ανάλογα με τα αποτελέσματα υπολογισμών
- Χειρισμός αποφάσεων σε προγράμματα:
  - εκτελούνται διαφορετικές εντολές ανάλογα με την απόφαση
  - γίνεται αλλαγή στη ροή εκτέλεσης
- Σε όλες τις γλώσσες προγραμματισμού η βασική δομή είναι εντολές-if
- Δύο μορφές:
  - if (συνθήκη) ακολουθία εντολών
  - if (συνθήκη) ακολουθία εντολών Α else ακολουθία εντολών Β

Αλλαγή ροής σε assembly

- Η assembly δεν έχει σημάδια για αρχή – τέλος ακολουθίας εντολών
  - χρησιμοποιεί ετικέτες (διευθύνσεις για γλώσσα μηχανής)

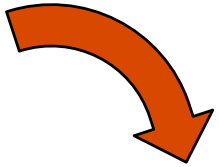
```
if (συνθήκη)
    εντολή 1
    εντολή 2
εντολή 3
```



```
            if (αντίθετη συνθήκη) goto Label_skip
            εντολή 1
            εντολή 2
Label_skip:
            εντολή 3
```

if-then-else σε assembly

```
if (συνθήκη)
    εντολή 1
    εντολή 2
else
    εντολή 3
εντολή 4
```



```
                if (αντίθετη συνθήκη) goto Label_else
                εντολή 1
                εντολή 2
                goto Label_skip
Label_else:
                εντολή 3
Label_skip:
                εντολή 4
```

Εντολές διακλάδωσης

- Οι εντολές που αλλάζουν τη ροή εκτέλεσης (control flow)
  - Αγγλικός όρος: branch instructions
- Διακλάδωση με συνθήκη (conditional branch)
  - στον MIPS: `beq καταχ.1, καταχ.2, ετικέτα`
    - αν τιμή καταχ.1 ίση με την τιμή καταχ.2, πήγαινε στην ετικέτα
    - branch if equal
  - υπάρχει και η εντολή bne
    - branch if not equal
- Υποχρεωτική διακλάδωση (unconditional branch)
  - συχνά λέγεται άλμα (jump)
  - στον MIPS: `j ετικέτα`
    - πήγαινε στην ετικέτα

Παράδειγμα

```
if (i == j)
    f = g+h
else
    f = g-h
```

i → $s3 j → $s4

f → $s0

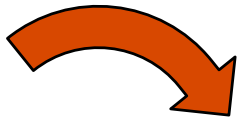
g → $s1 h → $s2

```
        bne   $s3, $s4, else
        add   $s0, $s1, $s2
        j     skip
else:
        sub   $s0, $s1, $s2
skip:
        . . . . .
```

Πολλαπλές if-then-else

- Πώς μεταφράζεται;

```
if (Σ1)
    E1
else if (Σ2)
    E2
else
    E3
```



```
if (Σ1)
    E1
else
    if (Σ2)
        E2
    else
        E3
```

Ex

Πολλαπλές if-then-else 2/2

```
                if (αντίθετη Σ1) goto Label_else1
                E1
                goto Label_out
Label_else1:
                Ex
Label_out:
```

```
                if (αντίθετη Σ1) goto Label_else1
                E1
                goto Label_out
Label_else1:
                if (αντίθετη Σ2) goto Label_else2
                E2
                goto Label_out
Label_else2:
                E3
Label_out:
```

Επαναλήψεις - loops

- **Δεν** υπάρχουν ειδικές εντολές για επαναλήψεις
  - υλοποιούνται με διακλαδώσεις, άλματα προς τα πίσω (πάνω)

```
sum = 0
while (i != 0)
    sum = sum + i
    i = i - 1
```

sum → `$s0` i → `$s1`

```
        add   $s0, $zero, $zero  # sum = 0
loop:
        beq   $s1, $zero, exitLoop
        add   $s0, $s0, $s1      # sum = sum + i
        addi  $s1, $s1, -1       # i = i - 1
        j     loop
exitLoop:
        . . . . .
```

Είδη επαναλήψεων

- Τρία είδη:
  - while (συνθήκη) ακολουθία εντολών
  - for (εντολή Α; συνθήκη; εντολή Β) ακολουθία εντολών
  - do ακολουθία εντολών while (συνθήκη)

- Η for είναι ισοδύναμη με τη while:

```
εντολή Α
while (συνθήκη)
    ακολουθία εντολών
    εντολή Β
```

Επανάληψη: do - while

```
sum = 0
do
    sum = sum + i
    i = i - 1
while (i != 0)
```

sum → `$s0` i → `$s1`

```
        add   $s0, $zero, $zero
loop:
        add   $s0, $s0, $s1
        addi  $s1, $s1, -1
        bne   $s1, $zero, loop
        # --- end of loop here ---
        . . . . .
```

«Λογικές πράξεις» στον MIPS

- Χρήσιμες για επεξεργασία μερικών bit μιας λέξης
  - εξαγωγή, εισαγωγή πεδίων σε λέξεις (**bit masking**)

Logical operations	Java operators	MIPS instructions
Bit-by-bit AND	&	`and`
Bit-by-bit OR	|	`or`
Shift left	<<	`sll`
Shift right	>>>	`srl`

Ολίσθηση στον MIPS

- Αριστερή ολίσθηση `sll` (shift left logical)
  - `sll καταχ. προορισμού, καταχ. πηγής, σταθερά`
  - Παράδειγμα: `sll $s0, $s1, 5  # $s0 = $s1 << 5`
  - Δεν επιτρέπεται καταχωρητής στη θέση της σταθεράς
    - διαφορετική εντολή
  - Η σταθερά ολίσθησης είναι **μικρή** (και θετική πάντα)
  - Συμπληρ. με 0 από δεξιά. Τα τελευταία αριστερά bit χάνονται!
  - Ισοδυναμεί με πολλαπλασιασμό με δύναμη του 2.
- Παράδειγμα: `sll $s0, $s1, 5` με $s1 = 0x2 = 2_{ten}

χάνονται!

  - `$s1 = 0000 0000 0000 0000 0000 0000 0000 0010`
  - Αποτέλεσμα $s0 = 0x40 = 64_{ten}
  - `$s0 = 0000 0000 0000 0000 0000 0000 0100 0000`

5 θέσεις

Δεξιά ολίσθηση

- Λογική δεξιά ολίσθηση `srl` (shift right logical)
  - παρόμοια με αριστερή: συμπληρώνονται 0 από αριστερά
  - ισοδυναμεί με διαίρεση με δύναμη του 2 (πηλίκο μόνο)
  - ποιό είναι το υπόλοιπο;
  - δεν δουλεύει σωστά για αρνητικούς αριθμούς
    - γίνονται θετικοί!
- Αριθμητική δεξιά ολίσθηση `sra` (shift right arithmetic)
  - συμπληρώνονται με ό,τι τιμή έχει το αριστερότερο bit
  - η διαίρεση είναι λίγο παράξενη για αρνητικούς αριθμούς:

$-25_{ten} / 16_{ten} = 1110\ 0111_{two} >> 4 = 1111\ 1110_{two} = -2$

-25/16 = -1.56 = -2 + 7/16

«Μεταβλητή» Ολίσθηση

- Στις προηγούμενες εντολές ολίσθησης, ο αριθμός θέσεων ολίσθησης ήταν σταθερός
- Για μεταβλητό αριθμό θέσεων: sllv, srlv ,srav
  - `sllv καταχ. προορισμού, καταχ. πηγής1, κ. πηγής2`
  - Παράδειγμα: `sllv $s0, $s1, $s3  # $s0 = $s1 << $s3`
- Αν η τιμή του καταχωρητή πηγής 2 είναι μεγαλύτερη του 31, ή αρνητικός αριθμός, ...

Λογικές πράξεις

- Οι γνωστές πράξεις AND, OR, XOR, …
  - εντολές MIPS: and, or, xor
- Οι αντίστοιχες εντολές του MIPS **δουλεύουν ανά bit** (bitwise): η πράξη εκτελείται μεταξύ των αντίστοιχων bits εισόδου
- Παράδειγμα AND:

```
1011 0110 1010 0100 0011 1101 1001 1010
---------------------------------------
0000 0000 0000 0000 0000 1111 1111 1111
0000 0000 0000 0000 0000 1101 1001 1010
```

Χρήση λογικών πράξεων

- Παρατηρείστε:
  - AND ενός bit με το 0, δίνει αποτέλεσμα 0
  - AND ενός bit με το 1, δίνει αποτέλεσμα το αρχικό bit
- Αρα μπορούμε να απομονώσουμε πεδία βάζοντας στα τμήματα που θέλουμε 1 και στα υπόλοιπα 0
- Ο αριθμός με τα 1 και 0 στις κατάλληλες θέσεις ονομάζεται μάσκα (mask) γιατί μπορεί να «κρύψει» τις τιμές της άλλης εισόδου της λογικής πράξης

Χρήση λογικών πράξεων 2

- AND
  - απομόνωση πεδίων
  - καθαρισμός (μηδενισμός) πεδίων
- OR
  - τοποθέτηση 1 σε πεδία

- Οι εντολές λογικών πράξεων έχουν παραλλαγές με σταθερά
  - `andi`, `ori`

Μάσκες - παραδείγματα

- «Εξαγωγή» (extraction) του 2ου byte από μία λέξη

```
srl  $t0, $s0, 8
andi $t0, $t0, 0xff  # clear all upper bytes
```

- Ελεγχος αν ο αριθμός στο `$s0` είναι άρτιος

```
andi $t0, $s0, 0x1
beq  $t0, $zero, άρτιος
```

- «Επόμενος» περιττός (αν είναι ήδη περιττός, δεν αλλάζει)

```
ori  $t0, $s0, 0x1
```

  - μπορεί να γίνει υπερχείλιση;

Παράδειγμα

Μετατροπή σε assembly:

```
while (save[i] == k)
   i += 1;
```

save[] πίνακας (array) λέξεων,
με αρχική διεύθυνση στον $s6
i→ $s3, k → $s5

```
loop:
        # calculate address of save[i]
        # get/load value of save[i] into regX
        bne   regX, k, exitLoop
        addi  i, i, 1
        j     loop
exitLoop:
```

Παράδειγμα

Μετατροπή σε assembly:

```
while (save[i] == k)
   i += 1;
```

save[] πίνακας (array) λέξεων,
με αρχική διεύθυνση στον $s6
i→ $s3, k → $s5

```
loop:
        sll   $t1, $s3, 2    # calculate offset: i*4
        add   $t1, $t1, $s6  # add to base of save[]
        lw    $t0, 0($t1)
        bne   $t0, $s5, exitLoop
        addi  $s3, $s3, 1
        j     loop
exitLoop:
```

Συγκρίσεις ανισότητας

- Μερικές φορές δεν αρκεί έλεγχος ισότητας ή ανισότητας
  - αν και είναι ο πιο συνηθισμένος τύπος ελέγχου
- Χρειάζεται να ελεγθεί αν ένας αριθμός είναι μικρότερος
- Εντολή MIPS slt – set on less than
  - μη την μπερδεύετε με τις εντολές ολίσθησης (sll, srl)
  - Σύνταξη: `slt rdest, rsrc1, rsrc2`

    Αν τιμή του rsc1 < τιμή rsrc2,

    rdest ← 1,

    αλλιώς rdest ← 0

Διακλάδωση με συνθήκη <

Μετατροπή σε assembly:

```
if (g >= h)
   εντολή 1
εντολή 2
```

g→ $s0, h → $s1

```
        slt   $t0, $s0, $s1     # t0 = (g < h)
        bne   $t0, $zero, less  # if (t0==1) goto less
        εντολή 1
less:
        εντολή 2
```

Συνθήκες ανισότητας

- Δεν υπάρχει εντολή set if greater than
  - απλά αλλάζει η σειρά των καταχωρητών

```
if (g < h)
   εντολή 1
εντολή 2
```

1

```
      if (g >= h) goto skip
      εντολή 1
skip:
      εντολή 2
```

2

```
      if not (g < h) goto skip
      εντολή 1
skip:
      εντολή 2
```

3

```
        slt   $t0, $s0, $s1    # t0 = (g < h)
        beq   $t0, $zero, geq  # if (t0==0) goto geq
        εντολή 1
geq:
        εντολή 2
```

Ψευτοεντολές διακλάδωσης

- Υπάρχουν ψευτο-εντολές σύγκρισης-διακλάδωσης
  - blt – branch if less than
  - ble – branch if less or equal
  - …
- Μην τις χρησιμοποιείτε στις ασκήσεις!
  - η παραλλαγή του Mars του μαθήματος, δεν τις επιτρέπει
  - είναι σημαντικό να κατανοήσετε ότι με 1 εντολή σύγκρισης και τις 2 παραλλαγές της branch μπορούν να γίνουν τα πάντα
  - επίσης σημαντικό να σκέφτεται κανείς με «αρνητική λογική»

Σταθερές σε συγκρίσεις

- Πολύ συχνά χρειάζονται συγκρίσεις με σταθερές
- Εντολή: `slti rdest, rsrc1, immediate`

- Προσοχή: η σταθερά είναι πάντα η **δεύτερη πηγή**
  - π.χ. `slti $t0, $s0, 5   #  t0 = s0 < 5`

Συγκρίσεις με απρόσημους

- Οι εντολές που είδαμε είναι για ακέραιους με πρόσημο
  - συμπλήρωμα ως προς 2
- Δεν δουλεύουν σωστά με απρόσημους (θετικούς) αριθμούς
  - $\texttt{ffffffff}_{\text{hex}}$ < $\texttt{00000001}_{\text{hex}}$, γιατί (-1) < 1)
  - αλλά αν είναι θετικοί, $\texttt{ffffffff}_{\text{hex}}$ > $\texttt{00000001}_{\text{hex}}$
- Παραλλαγές εντολών με u στο τέλος:
  - `sltu`, `sltiu`

Περίληψη

- Εντολές διακλάδωσης
  - με συνθήκη, άλματα
- Μετατροπή σε assembly if-then, if-then-else
- Επαναλήψεις σε assembly
- Λογικές πράξεις
  - and, or, xor
  - ολισθήσεις: sll, srl, sra
- Σύγκριση ανισότητας
  - slt
  - Υλοποίηση διακλαδώσεων με συνθήκη ανισότητας

Επόμενη φορά

- Κωδικοποίηση εντολών – γλώσσα μηχανής