ΗΥ 232
Οργάνωση και Σχεδίαση Υπολογιστών

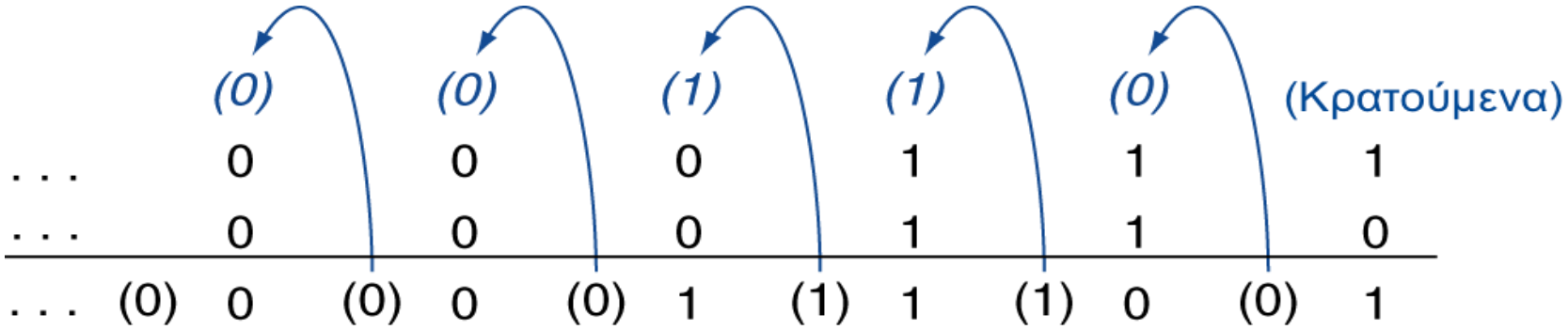
Διάλεξη 6

Αριθμητική Υπολογιστών

Νίκος Μπέλλας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

Πρόσθεση ακεραίων

- Παράδειγμα: 7 + 6



- Παράδειγμα με πολλαπλούς ακεραίους: 10 + 15 + 23 = 48

+	0	0	1	0	1	0
	0	0	1	1	1	1
	0	1	0	1	1	1
	1	1	0	0	0	0

Αφαίρεση ακεραίων

- Προσθέστε το αντίθετο του δεύτερου τελεστέου
- Παράδειγμα: 7 – 6 = 7 + (–6)

  +7: 0000 0000 … 0000 0111
  –6: 1111 1111 … 1111 1010
  +1: 0000 0000 … 0000 0001

- Τόσο η πρόσθεση όσο και η αφαίρεση μπορούν να οδηγήσουν σε υπερχείλιση αποτελέσματος (overflow)

Υπερχείλιση (overflow)

- Υπερχείλιση: κατάσταση μη έγκυρου αποτελέσματος (το αποτέλεσμα μιας πράξης χρειάζεται περισσότερα από τα διαθέσιμα ψηφία για να αναπαρασταθεί)
- Παράδειγμα μη προσημασμένης πρόσθεσης
  - Έστω ότι θέλουμε να προσθέσουμε δύο **απρόσημους** αριθμούς , και να τοποθετήσουμε το αποτέλεσμα σε ένα καταχωρητή των 4-bits.

+14: 1110
+ 3: 0011
+17: **10001**

  - Το πιο σημαντικό bit δεν χωράει στον καταχωρητή → overflow
  - Άρα, όταν το κρατούμενο είναι $c_n = 1$, τότε έχουμε υπερχείλιση στην πρόσθεση

Υπερχείλιση (overflow)

- Παράδειγμα προσημασμένης πρόσθεσης
  - Έστω ότι θέλουμε να προσθέσουμε δύο **προσημασμένους**, θετικούς αριθμούς , και να τοποθετήσουμε το αποτέλεσμα σε ένα καταχωρητή των 5-bits.

+12: 01100
+ 8: 01000
+20: **010100**

  - Ο αριθμός είναι αρνητικός → overflow
  - Άρα, όταν το κρατούμενο είναι $c_n <> c_{n-1}$, τότε έχουμε υπερχείλιση στην πρόσθεση προσημασμένων αριθμών. Εδώ $c_n = 0$, $c_{n-1} = 1$

Υπερχείλιση (overflow)

- Παράδειγμα προσημασμένης πρόσθεσης
  - Έστω ότι θέλουμε να προσθέσουμε δύο **προσημασμένους**, αρνητικούς αριθμούς , και να τοποθετήσουμε το αποτέλεσμα σε ένα καταχωρητή των 5-bits.

  -12: 10100
  - 8: 11000
  -20: **101100**

  - Άρα, όταν το κρατούμενο είναι $c_n <> c_{n-1}$, τότε έχουμε υπερχείλιση στην πρόσθεση προσημασμένων αριθμών. Εδώ $c_n = 1$, $c_{n-1} = 0$
  - Δεν μπορούμε ποτέ να έχουμε υπερχείλιση όταν προσθέτουμε έναν θετικό και έναν αρνητικό αριθμό
  - Δεν μπορούμε ποτέ να έχουμε υπερχείλιση όταν αφαιρούμε έναν θετικό από έναν θετικό αριθμό ή το αντίθετο

Υπερχείλιση (overflow)

- Υπερχείλιση πρόσθεσης A+B μη προσημασμένων αριθμών, n bit:
  - $A+B \geq 2^n \Leftrightarrow c_n = 1$ (c_n : κρατούμενο εξόδου από το MSB)
- Η αφαίρεση A-B ή B-A μη προσημασμένων αριθμών δεν παρουσιάζει υπερχείλιση
- Υπερχείλιση πρόσθεσης A+B προσημασμένων αριθμών:
  - εάν $A\geq 0, B\geq 0$: $A+B \geq 2^{n-1}$ εάν $c_n = 0$: $c_{n-1} = 1$

    εάν $A<0, B<0$: $A+B < -2^{n-1}$ εάν $c_n = 1$: $c_{n-1} = 0$

    εάν $A \cdot B<0$ (περίπτωση χωρίς υπερχείλιση) τότε πάντοτε είναι $c_n=c_{n-1}=0$ ή $c_n=c_{n-1}=1$ (c_{n-1} : κρατούμενο εισόδου προς το MSB)
  - Συνολική συνθήκη υπερχείλισης: MSB(X) = MSB(Y) ≠ MSB(X+Y)
  - Ισοδύναμη συνθήκη υπερχείλισης: (c_n xor c_{n-1}) = 1

Αριθμητικές και συγκριτικές πράξεις στον MIPS

- Αριθμητικές πράξεις μη προσημασμένων αριθμών:

  addu $s0,$s1,$s2 ⇔ $s0 = $s1 + $s2 ; no_overflow

  subu $s0,$s1,$s2 ⇔ $s0 = $s1 - $s2 ; no_overflow

  addiu $s0,$s1,100 ⇔ $s0 = $s1 + 100 ; no_overflow
- Οι addu, subu, addiu παράγουν το ίδιο αποτέλεσμα με τις add, sub, addi αλλά δεν προκαλούν εξαίρεση σε πιθανή υπερχείλιση

  => χρησιμοποιούνται - εκτός από το χειρισμό μη προσημασμένων δεδομένων, π.χ. *διευθύνσεων* - όταν δεν θέλουμε να διακοπεί η ροή του προγράμματος από το λειτουργικό σύστημα σε περίπτωση υπερχείλισης (τότε η ύπαρξη υπερχείλισης θα πρέπει να ελέγχεται μέσα από το ίδιο το πρόγραμμα)
- Συγκριτικές πράξεις μη προσημασμένων αριθμών:

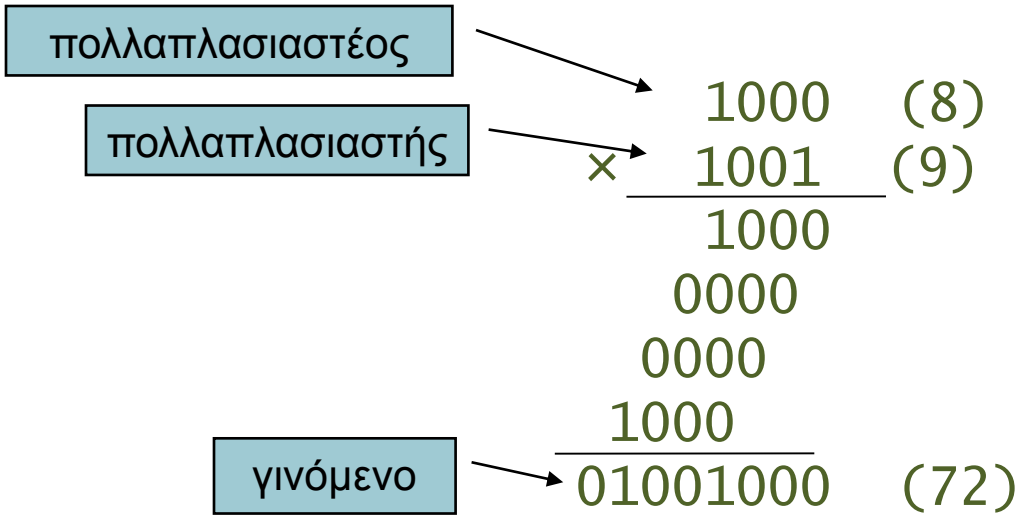
  sltu $s0,$s1,$s2 ⇔ $s0 = ($s1 < $s2)$_u$

  sltiu $s0,$s1,100 ⇔ $s0 = ($s1 < 100)$_u$
- Προσοχή. Στις εντολές addiu και sltiu η 16-bit σταθερά είναι *προσημασμένη* και λαμβάνει επέκταση προσήμου μέχρι τα 32-bits, και μόνο η τελική πράξη της πρόσθεσης ή της σύγκρισης αφορά μη προσημασμένους αριθμούς

Πολλαπλασιασμός και Διαίρεση Ακεραίων

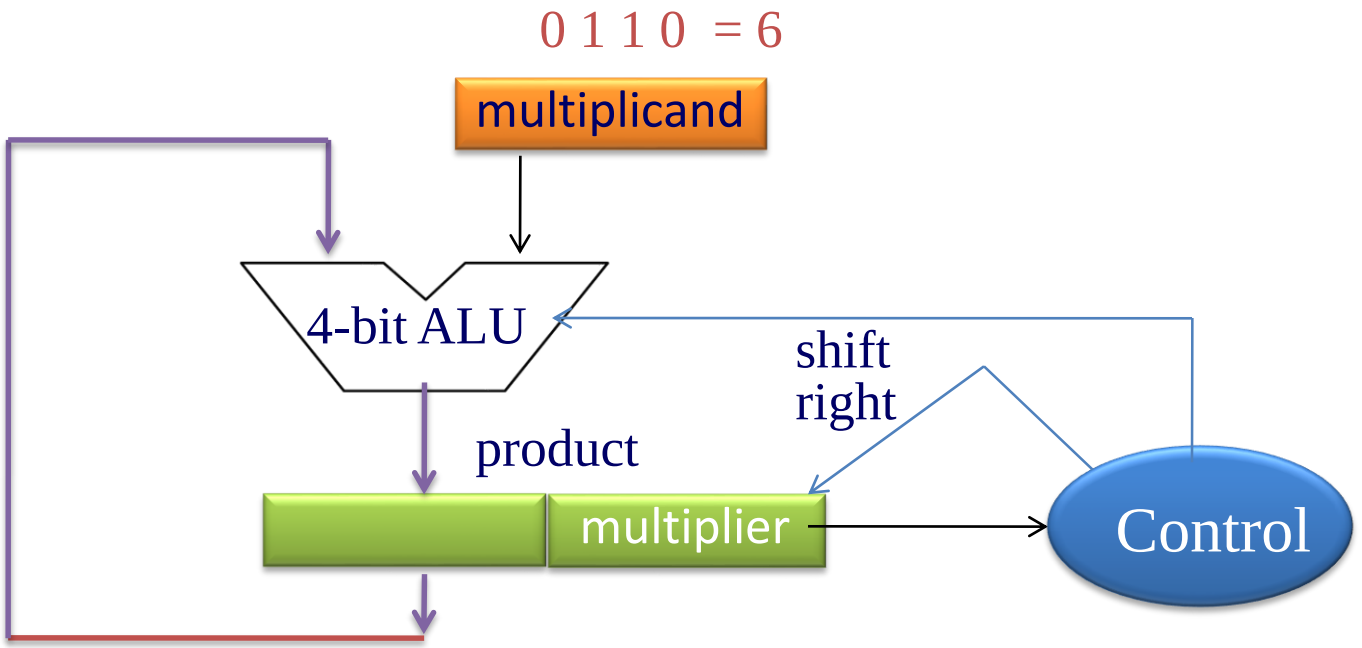
Πολλαπλασιασμός Ακεραίων

Μη προσημασμένοι ακέραιοι: ακριβώς όπως και στον πολλαπλασιασμό στο δεκαδικό σύστημα



Το μήκος του γινομένου είναι όσο το άθροισμα των μηκών των τελεστέων.

Σειριακός πολλαπλασιαστής απρόσημων αριθμών




```
      0 0 0 0      0 1 0 1    = 5
add   0 1 1 0      0 1 0 1
      0 0 1 1  →   0 0 1 0
add   0 0 1 1      0 0 1 0
      0 0 0 1  →   1 0 0 1
add   0 1 1 1      1 0 0 1
      0 0 1 1  →   1 1 0 0
add   0 0 1 1      1 1 0 0
      0 0 0 1  →   1 1 1 0    = 30
```

Προσημασμένος πολλαπλασιασμός

- Ο απλούστερος τρόπος να πολλαπλασιάσουμε δύο προσημασμένους αριθμούς n και m bits:
  1. είναι να τους μετατρέψουμε πρώτα σε θετικούς αριθμούς
  2. να κάνουμε τον πολλαπλασιασμό όπως μεταξύ απρόσημων
  3. να μετατρέψουμε το γινόμενο σε αρνητικό (συμπλήρωμα ως προς 2), εάν ένας από τους πολλαπλασιαστέο ή πολλαπλασιαστή είναι αρνητικός
- Το γινόμενο έχει μέγεθος (n-1)+(m-1) + 1 = n+m-1 bits
  - m-1 είναι το μέγεθος του πολλαπλασιαστέου (χωρίς το πρόσημο)

(-8) x (+10) με m=n=5

1. Πολλ/σμός (+8)x(+10)

2.
```
        01000   (+8)
        01010  (+10)
        00000
       01000
      00000
     01000
    00000
    001010000    (+80)
```

3. Μετατροπή σε -80 = $(110110000)_2$

χρησιμοποιώντας 4+4+1=9 bits

Προσημασμένος πολλαπλασιασμός

- Ο πολλαπλασιασμός με προσημασμένους μπορεί να γίνει και απευθείας με την κατάλληλη επέκταση προσήμου του πολλαπλασιαστέου στα μερικά γινόμενα (βλ. παράδειγμα Α)
- Επίσης έχουμε διαφορετικό χειρισμό στην περίπτωση αρνητικού πολλ/στή.
- Στην περίπτωση αυτή, αφαιρούμε τον πολλ/στέο από τον πολλ/στη στο ν-οστό βήμα (βλ. παράδειγμα Β)

• (-8) x (+10) με m=n=5

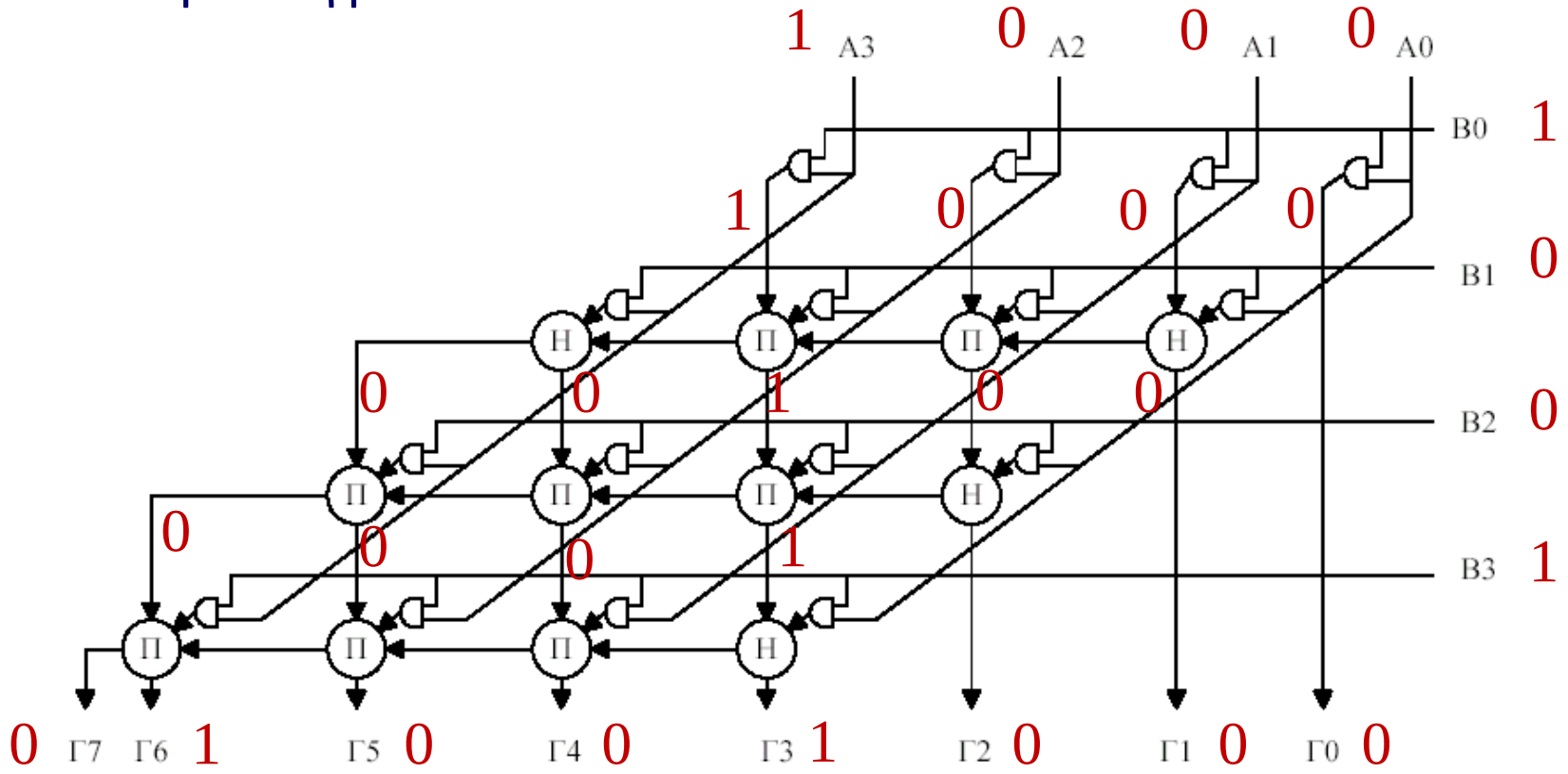
```
          11000   (-8)
          01010  (+10)
         -------
          00000
       11111000
         00000
       111000
      000000
     ---------
      110110000  (-80)
```

•(-8) x (-10) με m=n=5

```
          11000   (-8)
          10110  (-10)
         -------
          00000
       11111000
      11111000
        00000
      001000    (+8)
     ----------
      001010000    (+80)
```

Παράλληλος Πολλαπλασιαστής

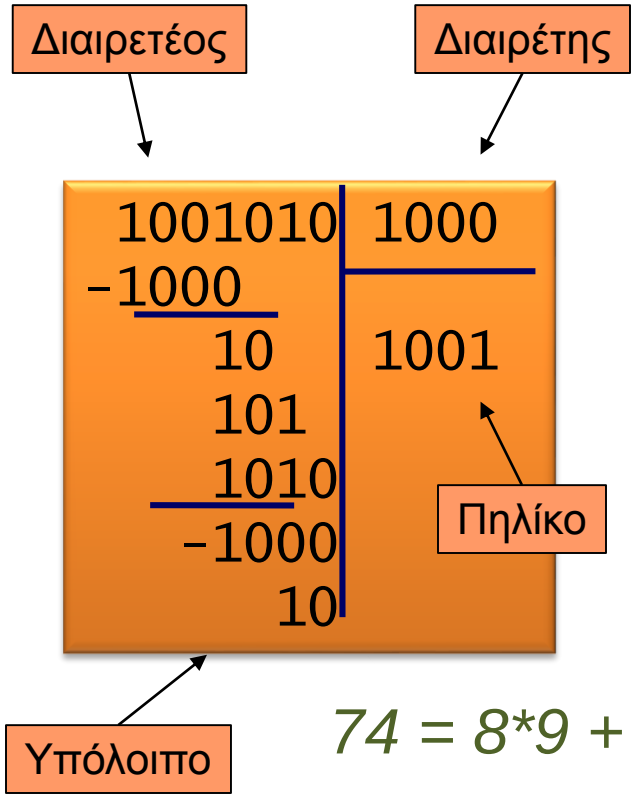
- Χρήση ενός δέντρου (tree) από αθροιστές
- Όλες οι προσθέσεις γίνονται παράλληλα
- Παράδειγμα: 8x9 = 72



Πολλαπλασιασμός στον MIPS

- Δύο νέοι καταχωρητές 32-bit για το γινόμενο
  - HI: Περισσότερο-σημαντικά 32 bits
  - LO: Λιγότερο-σημαντικά 32-bits
- Εντολές
  - `mult $rs, $rt / multu $rs, $rt`
    - γινόμενο 64-bit σε καταχωρητές HI/LO
  - `mfhi $rd / mflo $rd`
    - Μετακίνηση από HI/LO στο rd
    - Μπορεί να ελέγξει την τιμή του HI για να δει αν το γινόμενο υπερχειλίζει τα 32 bits
  - `mul $rd, $rs, $rt`
    - γινόμενο 64-bit σε καταχωρητές HI/LO ΚΑΙ λιγότερο-σημαντικά 32 bits του γινομένου στον $rd

Διαίρεση



$74 = 8*9 + 2$

- Έλεγχος για διαιρέτη 0
- Ο κλασσικός αλγόριθμος
  - Αν ο διαιρέτης χωράει στο διαιρετέο
    - 1 στο πηλίκο, αφαίρεση
  - Αλλιώς
    - 0 στο πηλίκο, κατεβαίνει επόμενο ψηφίο
- Προσημασμένη διαίρεση
  - Κάνουμε διαίρεση χρησιμοποιώντας απόλυτες τιμές
  - Διορθώνουμε το πρόσημο του πηλίκου και του υπολοίπου αναλόγως

- Βασική σχέση της Διαίρεσης:

Διαιρετέος = Πηλίκο*Διαιρέτης + Υπόλοιπο

Προσημασμένη Διαίρεση

- Μετατρέπουμε τον διαιρέτη και διαιρετέο από αρνητικούς σε θετικούς και κάνουμε απρόσημη διαίρεση
- Στο τέλος, κάνουμε αρνητικό το πηλίκο, εάν τα πρόσημα του διαιρέτη και διαιρετέου είναι διαφορετικά
- Ας δούμε την διαίρεση (+-7)/(+-2)
  - 7/2 : Πηλίκο = 3, Υπόλοιπο = 1 (7 = 3*2 + 1)
  - -7/2 : Πηλίκο = -3, Υπόλοιπο = -1 (-7 = (-3)*2 + (-1))
  - 7/-2 : Πηλίκο = -3, Υπόλοιπο = 1 (-7 = (-3)*(-2) + 1)
  - -7/-2 : Πηλίκο = 3, Υπόλοιπο = -1 (-7 = 3*(-2) + (-1))

Διαίρεση στον MIPS

- Εντολές

  div $rs, $rt ⇔ Lo = $rs / $rt ; Hi = $rs mod $rt

  divu $rs, $rt ⇔ Lo = ($rs / $rt)$_u$; Hi = ($s0 mod $rt) $_u$

- Ο Πολλαπλασιασμός και η Διαίρεση αγνοούν τυχόν υπερχείλιση
  - Το software θα πρέπει να ελέγχει για υπερχείλιση
  - Το software θα πρέπει να ελέγχει τον διαιρέτη ώστε να είναι διάφορος του 0.

Αριθμητική Κινητής Υποδιαστολής

Αναπαράσταση πολύ μεγάλων και πολύ μικρών αριθμών

- Πως θα μπορούσαμε να αναπαραστήσουμε την μέση ηλικία της γης?

  4.600.000.000 ή $4,6 \times 10^{9}$ έτη

- ή την μονάδα ατομικής μάζας

  0,000000000000000000000000166 or $1,6 \times 10^{-27}$ Kg

Αυτοί οι αριθμοί δεν μπορούν να αναπαρασταθούν με 32-bit ή 64-bit ακεραίους

- Χρησιμοποιούμε αριθμούς κινητής υποδιαστολής για την αναπαράσταση πραγματικών αριθμών
  - float x, y
  - double x, y

Κινητή υποδιαστολή

- Αναπαράσταση μη-ακεραίων
  - Καθώς και πολύ μικρών/μεγάλων αριθμών
- Παρόμοιο με επιστημονική σημειογραφεία
  - –+987,02 × 10^{9} Όχι ΕΣ
  - 2,34 × 10^{56} ΕΣ, Κ
  - +0,002 × 10^{-4} ΕΣ, Όχι Κ

Επιστημονική σημειογραφία: μόνο ένα ψηφίο πρίν την υποδιαστολή

Κανονικοποιημένος εάν είναι σε επιστημονική σημειογραφία και το ψηφίο αριστερά της υποδιαστολής δεν είναι 0.

- Στο δυαδικό σύστημα χρησιμοποιούμε βάση 2:
  - $\pm 1,xxxxxxx_2 \times 2^{yyyy}$ (Σημαντικό / Significand, Εκθέτης / Exponent)
  - Όταν ο αριθμός είναι κανονικοποιημένος, 1,0 ≤ |significand| < 2,0
  - Significand = “1.xxxx…”

IEEE 754 FP Standard

- Οι περισσότεροι υπολογιστές σήμερα χρησιμοποιούν το standard της IEEE 754 για αναπαράσταση αριθμών κινητής υποδιαστολής
  - Τι είναι η IEEE?
- Αριθμός = $(-1)^{\text{Πρόσημο}} \times (1+\text{Κλάσμα}) \times 2^{\text{Εκθέτης-Πόλωση}}$

  Απλή ακρίβεια : συνολικά 32 bits, Διπλή ακρίβεια : συνολικά 64 bits

  Απλή ακρίβεια: 8 bits
  Διπλή ακρίβεια: 11 bits

  Απλή ακρίβεια: 23 bits
  Διπλή ακρίβεια: 52 bits

- Υπάρχει συμβιβασμός μεταξύ μεγέθους (σε bits) του εκθέτη και του κλάσματος
  - Μεγαλύτερος εκθέτης → μεγαλύτερο εύρος τιμών (range)
  - Μεγαλύτερο κλάσμα → μεγαλύτερη ακρίβεια (precision)

IEEE 754 FP Standard

$$\text{Αριθμός} = (-1)^{\text{Πρόσημο}} \times (1+\text{Κλάσμα}) \times 2^{\text{Εκθέτης-Πόλωση}}$$

- Π: πρόσημο (0 ⇒ μη-αρνητικός, 1 ⇒ αρνητικός)
- Κανονικοποίηση του significand: 1.0 ≤ |significand| < 2.0
  - Πάντα έχει ένα αρχικό bit 1 οπότε δε χρειάζεται να αποθηκευτεί (κρυφό bit)
- Εκθέτης: πραγματικός εκθέτης + Πόλωση (Bias)
  - Εξασφαλίζει ότι ο εκθέτης είναι **απρόσημος**
  - Απλή ακρίβεια: Πόλωση = 127; Διπλή ακρίβεια: Πόλωση = 1023

Παραδείγματα IEEE 754 FP Standard

$$\text{Αριθμός} = (-1)^{\text{Πρόσημο}} \times (1+\text{Κλάσμα}) \times 2^{\text{Εκθέτης-Πόλωση}}$$

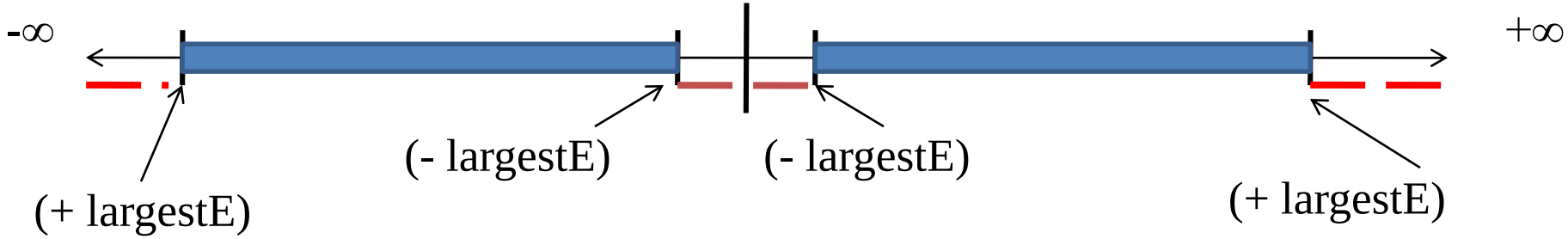
- Παραδείγματα (κανονικοποιημένη μορφή)
- Πρώτα πρέπει να γράψουμε τον αριθμό στην μορφή $\pm 1,xxxxxxx_2 \times 2^{yyyy}$

- $0,5 = 1,0_2 \times 2^{-1} = 1,0_2 \times 2^{126-127} =$ 0 01111110 00000000000000000000000
- $-0,875_{10} \times 2^4 = -0,111_2 \times 2^4 = -1,11_2 \times 2^3 = -1,11_2 \times 2^{130-127} =$
  1 10000010 11000000000000000000000
- Μικρότερος +: 0 00000001 00000000000000000000000 = $1 \times 2^{1-127} = 2^{-126}$
- Zero: 0 00000000 00000000000000000000000 = true 0
- Μεγαλύτερος +: 0 11111110 11111111111111111111111 =
  $(2-2^{-23}) \times 2^{254-127} \sim 2 \times 2^{127} = 2^{128}$

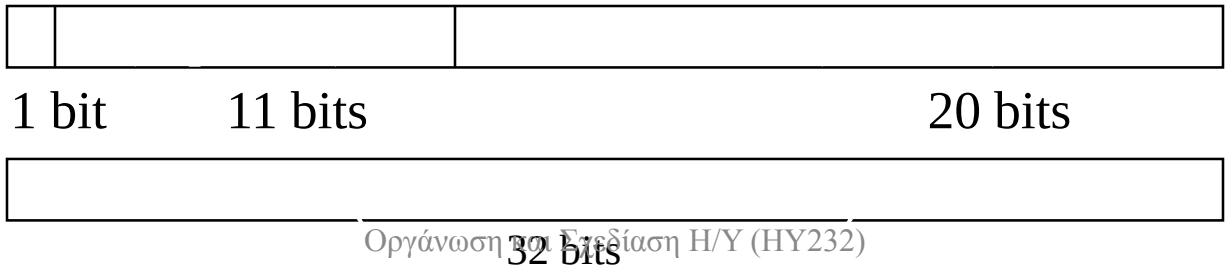
Εξαιρέσεις στους αριθμούς κινητής υποδιαστολής

- Υπερχείλιση (overflow) όταν ο αριθμός που θέλουμε να αναπαραστίσουμε έχει πολύ μεγάλο θετικό εκθέτη που δεν μπορεί να χωρέσει στο πεδίο του εκθέτη

- Ανεπάρκεια (underflow) όταν ο αριθμός που θέλουμε να αναπαραστίσουμε έχει πολύ μεγάλο αρνητικό εκθέτη που δεν μπορεί να χωρέσει στο πεδίο του εκθέτη



- Η αριθμητική διπλής ακρίβειας χρησιμοποιεί 64 bits για να αυξήσει τα bits εκθέτη και κλάσματος και να μειώσει την πιθανότητα overflow ή underflow



Παραδείγματα IEEE 754 FP Standard

- Ποιος αριθμός αναπαρίσταται σε απλή ακρίβεια από το

  1 10000001 01000…00

  – Π = 1

  – Κλάσμα = $01000\ldots00_2$

  – Εκθέτης = $10000001_2 = 129$

- $x = (-1)^1 \times (1 + ,01_2) \times 2^{(129 - 127)}$

  $= (-1) \times 1,25 \times 2^2$

  $= -5,0$

IEEE 754 FP Special Encoding

- Ειδικοί κώδικες στον IEEE 754 FP χρησιμοποιούνται για “ασυνήθιστα» γεγονότα
  - +- άπειρο μετά από διαίρεση με 0
  - NαN (not a number) για άκυρα αποτελέσματα όπως διαίρεση 0/0
  - Το 0 αναπαριστάται με όλα τα bits ίσα με 0 (γιά λόγους απλότητας)

Απλή ακρίβεια		Double Precision		Object Represented
E (8)	F (23)	E (11)	F (52)	
0	0	0	0	true zero (0)
0	nonzero	0	nonzero	± denormalized number
1-254	anything	1-2046	anything	± floating point number
255	0	2047	0	± infinity
255	nonzero	2047	nonzero	not a number (NaN)

Αθροιστής κινητής υποδιαστολής

Παράδειγμα 4-ψήφιου δεκαδικού. Θεωρούμε ότι μόνο 4 σημαντικά ψηφία μπορούν να αποθηκευθούν:

$9{,}999 \times 10^{1} + 1{,}610 \times 10^{-1}$

1. Ευθυγράμμιση υποδιαστολής

   Ολίσθηση αριθμού με μικρότερο εκθέτη

   $9{,}999 \times 10^{1} + 0{,}016 \times 10^{1}$

2. Πρόσθεση significands

   $9{,}999 \times 10^{1} + 0{,}016 \times 10^{1} = 10{,}015 \times 10^{1}$

3. Κανονικοποίηση αποτελέσματος και έλεγχος υπερείλισης ή υποχείλισης

   $10{,}015 \times 10^{1} \rightarrow 1{,}0015 \times 10^{2}$

4. Στρογγυλοποίηση και (αν ξαναχρειαστεί) κανονικοποίηση

   $1{,}002 \times 10^{2}$

Αθροιστής κινητής υποδιαστολής

Παράδειγμα 4-ψήφιου δυαδικού: $(\pm F1 \times 2^{E1}) + (\pm F2 \times 2^{E2}) = \pm F3 \times 2^{E3}$

$0{,}5 + -0{,}4375 = 1{,}000_2 \times 2^{-1} + -1{,}110_2 \times 2^{-2}$

1. Ευθυγράμμιση υποδιαστολής

   Ολίσθηση αριθμού με μικρότερο εκθέτη (E2) κατά E1-E2 θέσεις

   $1{,}000_2 \times 2^{-1} + -0{,}111_2 \times 2^{-1}$

2. Πρόσθεση significands ($\pm F1 + \pm F2 = F3$), θεωρώντας άπειρη ακρίβεια στο κλασματικό μέρος

   $1{,}000_2 \times 2^{-1} + -0{,}111_2 \times 2^{-1} = 0{,}001_2 \times 2^{-1}$

Αθροιστής κινητής υποδιαστολής

3. Κανονικοποίηση αποτελέσματος και έλεγχος overflow/underflow

   $0{,}001_2 \times 2^{-1} = 0{,}01_2 \times 2^{-2} = \ldots = 1{,}000_2 \times 2^{-4}$, χωρίς overlflow/underflow

4. Στρογγυλοποίηση και (αν ξαναχρειαστεί) κανονικοποίηση

   $1{,}000_2 \times 2^{-4}$ (καμμία αλλαγή) $= 0{,}0625$

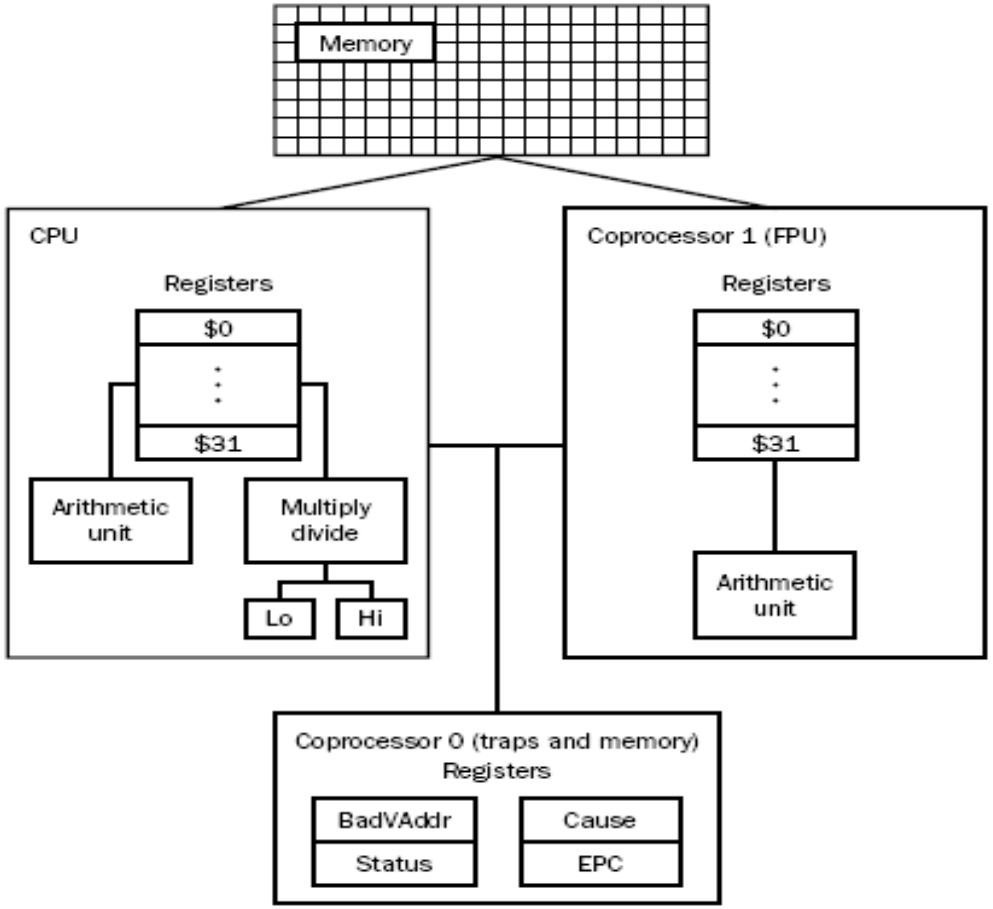
Παραδείγματα

- A + B = 2850,0 + -9840,0 = ? σε IEEE FP 754
  - A = 2850,0 = $(101100100010,0)_2 \times 2^0 = 1,01100100010 \times 2^{11}$
  - B = -9840,0 = $-(10011001110000,0)_2 = -1,0011001110000 \times 2^{13}$
  - $A = 1,01100100010 \times 2^{11} = 0,0101100100010 \times 2^{13}$
  - Πρόσθεση σημαντικών

    0,0101100100010 + -1,0011001110000 = 0,0101100100010 + 10,1100110010000 = 11,0010010110010 = -0,1101101001110
  - Sum = A+B = $-0,1101101001110 \times 2^{13} =$ **$-1,101101001110 \times 2^{12}$**
  - Πραγματικό αποτέλεσμα είναι: 2850,0 + -9840,0 = -6990,0
  - Αποτέλεσμα σε IEEE 754 είναι: $-1,101101001110 \times 2^{12} =$ -1,70654296875 x 4096 = -6990,0
  - Άρα Error = 0

Εντολές κινητής υποδιαστολής στο MIPS

- Το υλικό κινητής υποδιαστολής είναι ο συνεπεξεργαστής 1
  - Επεκτείνει το ISA
- Ξεχωριστοί κατ/τές κινητής υποδιαστολής
  - 32 απλής ακρίβειας: $f0, $f1, … $f31
  - Σε ζεύγη για διπλή ακρίβεια: $f0/$f1, $f2/$f3,



Ακριβής Αριθμητική

- Υπάρχουν άπειροι αριθμοί μεταξύ δύο οποιονδήποτε πραγματικών αριθμών
- Συνεπώς, οι αριθμοί κινητής υποδιαστολής είναι απλά προσεγγίσεις ενός αριθμού που δεν μπορεί να παρασταθεί πραγματικά

- Μικρότερος +: A = 0 00000001 00000000000000000000000 = $1 \times 2^{1-127} = 2^{-126}$
- Αμέσως μεγαλύτερος: B= 0 00000001 00000000000000000000001 = $(1+ 2^{-23}) \times 2^{-126}$
- $\Delta = B-A = 2^{-23} \times 2^{-126} = 2^{-149}$
- Αυτή είναι και η μεγαλύτερη δυνατή ακρίβεια στην αναπαράσταση μεταξύ διαδοχικών αριθμών απλής ακρίβειας
- Αλλά και πάλι ο αριθμοί : $(1+ 2^{-24}) \times 2^{-126}$, $(1+ 2^{-24} +2^{-25}) \times 2^{-126}$ κοκ. δεν μπορούν να αναπαρασταθούν

Ακριβής Αριθμητική

- Η στρογγυλοποίηση (rounding) χρησιμοποιείται για προσέγγιση πραγματικών αριθμών από το πεπερασμένο πλήθος των αριθμών κινητής υποδιαστολής
  - Η στρογγυλοποίηση ενδιάμεσων αποτελεσμάτων στην πρόσθεση και πολλαπλασιασμό μπορεί να επιφέρει χαμηλότερη προσέγγιση
  - Υποθέστε μόνο 3 σημαντικά ψηφία στην παρακάτω πρόσθεση

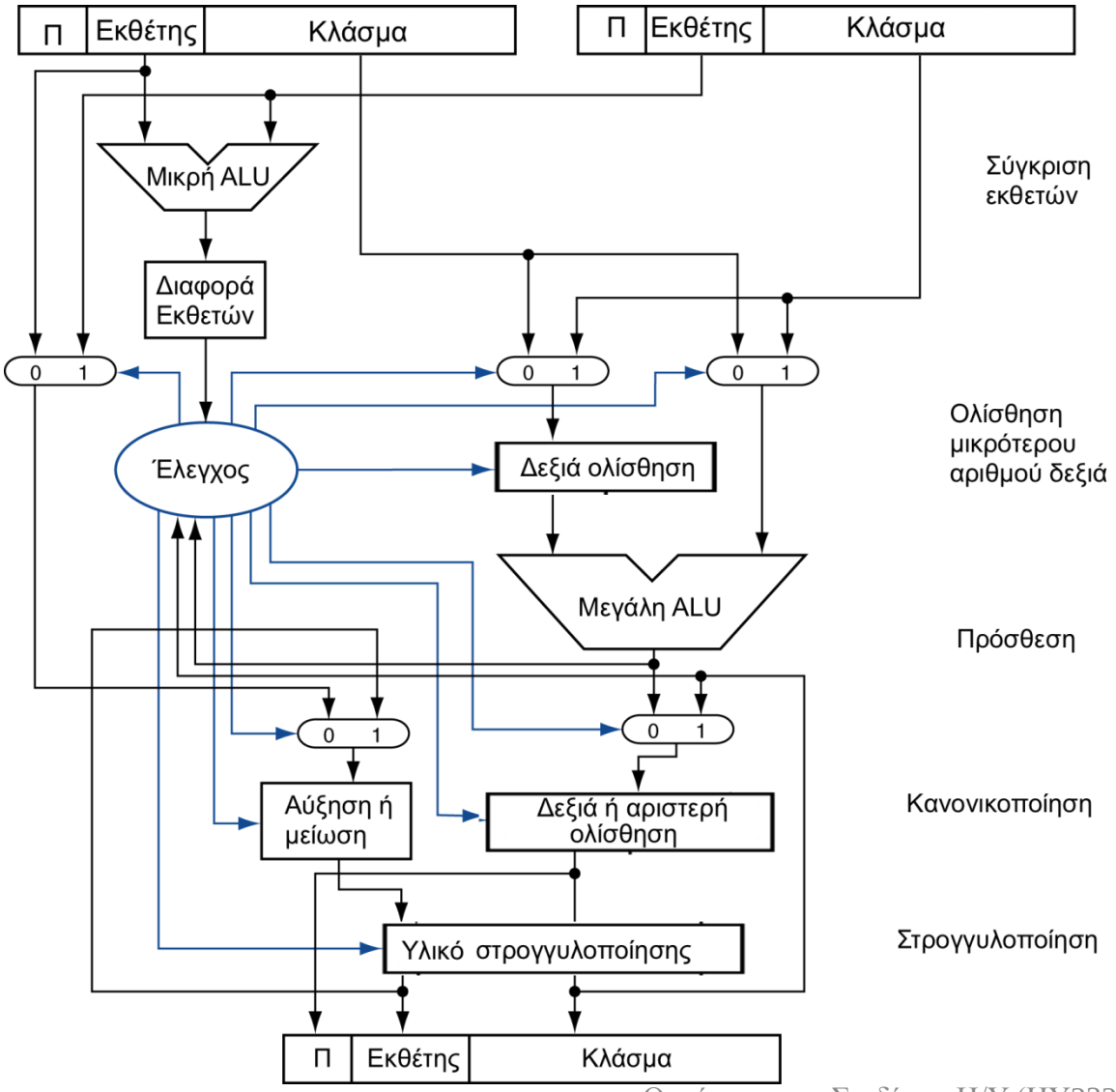
Στρογγυλοποίηση χωρίς extra bits	**Στρογγυλοποίηση με guard και round bits για ενδιάμεσα αποτελέσματα**
$2,56 \times 10^0 + 2,34 \times 10^2$ = $0,0256 \times 10^2 + 2,34 \times 10^2$ (Το 0,0256 πρέπει να στρογγυλοποιηθεί στο 0,02, επειδή τα ψηφία 5,6 χάνονται στην ολίσθηση) Άρα : $0,02 \times 10^2 + 2,34 \times 10^2 =$ **$2,36 \times 10^2$**	$2,56 \times 10^0 + 2,34 \times 10^2$ = $0,0256 \times 10^2 + 2,34 \times 10^2$ = $2,3656 \times 10^2$ Στρογγυλοποίηση στο **$2,37 \times 10^2$** για 3 σημαντικά ψηφία

Προσεταιριστικότητα (Associativity)

- Η πρόσθεση κινητής υποδιαστολής ΔΕΝ είναι προσεταιριστική πράξη
  - x + (y+z) != x+(y+z)
  - X = $-1,5 \times 10^{38}$
  - Y= $1,5 \times 10^{38}$
  - Z = 1,0

x+(y+z)	**(x+y)+z**
Y+Z = $1,5 \times 10^{38} + 1,0 = 1,5 \times 10^{38}$ X+(Y+Z) = $-1,5 \times 10^{38} + 1,5 \times 10^{38}$ = **0,0**	X+Y = $-1,5 \times 10^{38} + 1,5 \times 10^{38}$ = 0,0 (X+Y)+Z = 0,0 + 1,0 = **1,0**

Υλοποίηση αθροιστή κινητής υποδιαστολής



- Πολύ πιο πολύπλοκος από αθροιστή ακεραίων
- Συνήθως απαιτεί πολλαπλούς κύκλους μηχανής
- Τμήμα του Floating Point Unit (FPU) του επεξεργαστή

Πολλαπλασιασμός κινητής υποδιαστολής

Παράδειγμα τετραψήφιου δεκαδικού: : $(\pm F1 \times 2^{E1}) \times (\pm F2 \times 2^{E2}) = \pm F3 \times 2^{E3}$

$(1{,}110 \times 10^{10}) \times (9{,}200 \times 10^{-5})$

1. Πρόσθεση εκθετών

   Νέος Εκθέτης = $10 + -5 = 5$

2. Πολλαπλασιασμός significands

   $1{,}110 \times 9{,}200 = 10{,}212 \Rightarrow 10{,}212 \times 10^{5}$

3. Κανονικοποίηση αποτελέσματος και έλεγχος overflow/underflow

   $1{,}0212 \times 10^{6}$

4. Στρογγυλοποίηση και (αν ξαναχρειαστεί) κανονικοποίηση

   $1{,}021 \times 10^{6}$

5. Προσδιορισμός προσήμου με βάση τα πρόσημα των τελεστέων

   $+1{,}021 \times 10^{6}$

Πολλαπλασιασμός κινητής υποδιαστολής

Παράδειγμα τετραψήφιου δυαδικού (δηλ. μπορούμε να αποθηκέυσουμε μόνο 4 σημαντικά ψηφία σε κάθε χρονική στιγμή)

$0.5 \times -0.4375 = 1{,}000_2 \times 2^{-1} \times -1{,}110_2 \times 2^{-2}$

1. Πρόσθεση εκθετών

   Χωρίς πόλωση: $-1 + -2 = -3$

   Με πόλωση : $-3 + 127 = 124$

2. Πολλαπλασιασμός σημαντικών

   $1{,}000_2 \times 1{,}110_2 = 1{,}110000 = 1{,}110_2 \Rightarrow 1{,}110_2 \times 2^{-3}$

3. Κανονικοποίηση αποτελέσματος και έλεγχος overflow/underflow

   $1{,}110_2 \times 2^{-3}$ (καμία αλλαγή) γιατί -126 <= -3 <= 127

4. Στρογγυλοποίηση και (αν ξαναχρειαστεί) κανονικοποίηση

   $1{,}110_2 \times 2^{-3}$ (καμία αλλαγή)

5. Προσδιορισμός προσήμου $+ \times - \Rightarrow -$

   $-1{,}110_2 \times 2^{-3} = -0{,}21875$

Αριθμητική μονάδα κινητής υποδιαστολής

- Ο πολ/στής είναι παρόμοιας πολυπλοκότητας με τον αθροιστή
  - Αλλά χρησιμοποιεί πολλαπλασιαστή για τα significands αντί για αθροιστή
- Η αριθμητική μονάδα κινητής υποδιαστολής συνήθως κάνει:
  - Πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση, αντίστροφο, ρίζα
  - Μετατροπή από πραγματικό σε ακέραιο και το αντίθετο
- Οι εντολές συνήθως παίρνουν αρκετούς κύκλους

Εντολές κινητής υποδιαστολής στο MIPS

- Οι εντολές κινητής υποδιαστολής λειτουργούν μόνο σε καταχωρητές κινητής υποδιαστολής
  - Το προγράμματα γενικά δεν κάνουν ακέραιες πράξεις σε δεδομένα κινητής υποδιαστολής και αντίστροφα.
- Εντολές φόρτωσης και αποθήκευσης αριθμών κινητής υποδιαστολής

Απλή ακρίβεια

```
lwc1  $f1, 54($s2)      #   $f1 = Memory[$s2+54]
swc1 $f1, 58($s4)       # Memory[$s4+58] = $f1
```

Διπλή ακρίβεια

```
ldc1  $f2, 54($s2)       #   ($f2, $f3) = Memory[$s2+54]
sdc1  $f24, 58($s4)      # Memory[$s4+58] = ($f24, $f25)
```

Εντολές κινητής υποδιαστολής στο MIPS

- Αριθμητική απλής ακρίβειας
  - `add.s`, `sub.s`, `mul.s`, `div.s`
    - `add.s $f0, $f1, $f6`
- Αριθμητική διπλής ακρίβειας
  - `add.d`, `sub.d`, `mul.d`, `div.d`
    - `mul.d $f4, $f4, $f6`
- Σύγκριση απλής & διπλής ακρίβειας
  - `c.`*xx*`.s`, `c.`*xx*`.d` (*xx* είναι `eq`, `lt`, `le`, …)
  - Θέτει ή μηδενίζει το το bit κωδικού συνθήκης FP
    - `c.lt.s $f3, $f4`
- Διακλάδωση ανάλογα με τον κωδικό συνθήκης FP
  - `bc1t`, `bc1f`
    - `bc1t TargetLabel`
- Μετατροπή μεταξύ κινητής υποδιαστολής και ακεραίων
  - `cvt.w.s  # από κιν. υποδιαστολής απλής ακρίβειας σε ακέραιο`
    - `cvt.w.s $f0, $f1`

Παράδειγμα: °F σε °C

- Κώδικας C:

```
float f2c (float fahr) {
  return ((5.0/9.0)*(fahr - 32.0));
}
```

  - `fahr` στον $f12, αποτέλεσμα στον $f0, σταθερές στο global memory space
  - Αντίθετα με τους καταχωρητές ακεραίων, ο $f0 μπορεί να περιέχει έναν αριθμό
  - Υποθέτουμε ότι ο μεταγλωττιστής τοποθετεί τις τρεις σταθερές 5, 9, 32 στο .data section για να είναι προσπελάσιμες από τον καθολικό δείκτη $gp.

Παράδειγμα: °F σε °C

- Μεταγλωττισμένος κώδικας MIPS:

```
f2c: lwc1  $f16, const5($gp)
     lwc2  $f18, const9($gp)
     div.s $f16, $f16, $f18
     lwc1  $f18, const32($gp)
     sub.s $f18, $f12, $f18
     mul.s $f0,  $f16, $f18
     mov.s $f12, $f0
     li $v0, 2
      syscall
     jr    $ra
```

επίσης

```
li       $t0, 5
mtc1     $t0,  $f16
cvt.s.w $f16, $f16
```