Κλασσικός Αλγόριθμος Πολλαπλασιασμού Αριθμών

Θεωρούμε ότι στον καταχωρητή $a βρίσκεται ο πολλαπλασιαστέος και στον καταχωρητή $b βρίσκεται ο πολλαπλασιαστής. Εκτός από αυτούς τους 2 καταχωρητές, θα χρησιμοποιήσουμε και τους ακόλουθους:

$mask:	Θα περιέχει την ποσότητα 0x1000 0000 0000 0000 0000 0000 0000 0000 για να ελέγξουμε το πρόσημο των 2 αριθμών
$nega:	Μας δείχνει αν ο $a είναι αρνητικός
$negb:	Μας δείχνει αν ο $b είναι αρνητικός
$neg:	Μας δείχνει αν το γινόμενο $a x $b είναι θετικό ή αρνητικό (ομόσημοι αριθμοί έχουν γινόμενο θετικό)
$testa0:	Περιέχει το LSBit του καταχωρητή $a. Αν η τιμή του είναι 1, τότε προσθέτουμε τον πολλαπλασιαστέο στο γινόμενο
$prod:	Καταχωρητής των 64 bit που περιέχει το αποτέλεσμα του γινομένου των $a και $b.
$count:	Καταχωρητής/μετρητής που αρχικοποιείται με την τιμή 32 και στη συνέχεια μειώνεται σε κάθε βήμα του πολ/μού, μέχρι να γίνει μηδέν.

Κώδικας assembly

		Αρχικά θεωρούμε ότι όλοι οι καταχωρητές είναι ίσοι με μηδέν. Ειδάλλως είναι πολύ εύκολο να μηδενίσουμε ένα καταχωρητή χρησιμοποιώντας την εντολή xor $reg, $reg, $reg	Τμήμα αρχικοποίησης καταχωρητών-τιμών
	addi, $count, $count, 32	$count αρχικοποιείται με την τιμή 32	
	beq $a, $zero, Lfinish beq $b, $zero, Lfinish	Αν ο καταχωρητής $a ή $b περιέχουν μηδέν, τότε το πρόγραμμα τερματίζεται και η σωστή τιμή γινομένου (μηδέν) βρίσκεται ήδη στον καταχωρητή $prod	Τμήμα ελέγχου μηδενικών τιμών στους $a και $b
	addi $mask, $zero, 1 sll $mask, $mask, 31	Δημιουργούμε τη μάσκα $mask η οποία περιέχει ένα άσσο στο MSB του καταχωρητή $mask και όλα τα άλλα bit είναι μηδέν	Τμήμα ελέγχου αρνητικών αριθμών και μετατροπής τους σε θετικούς
	and $nega, $a, $mask and $negb, $b, $mask	Χρησιμοποιώντας τη μάσκα, μπορούμε να ξέρουμε αν οι καταχωρητές $a και $b περιέχουν αρνητική ποσότητα, πληροφορία η οποία φυλάσσεται στους $nega και $negb αντίστοιχα	
	xor $neg, $nega, $negb	Αν $a και $b είναι ομόσημοι, τότε το γινόμενό τους θα είναι θετικό. Δηλ, αν $neg είναι μηδέν, τότε το γινόμενο είναι θετικό. Αν περιέχει 1, τότε το γινόμενο είναι αρνητικό.	
	beq $nega, $zero, Lapos	Αν ο $nega δεν είναι μηδέν, τότε ο $a είναι αρνητικός και θα τον μετατρέψουμε σε θετικό	
	nor $a, $a, $zero addi $a, $a, 1	Invert το περιεχόμενο του καταχωρητή $a Πρόσθεση 1 στον καταχωρητή $a	
Lapos	beq $negb, $zero, Lbpos	Αν ο $negb δεν είναι μηδέν, τότε ο $b είναι αρνητικός και θα τον μετατρέψουμε σε θετικό	
	nor $b, $b, $zero addi $b, $b, 1	Invert το περιεχόμενο του καταχωρητή $a Πρόσθεση 1 στον καταχωρητή $a	
Lbpos	andi $testa0, $a, 1 beq $testa0, $zero, Lnext	Έλεγχος του LSB του $a Αν είναι μηδέν, τότε προχώρα στο επόμενο βήμα, χωρίς να κάνεις καμία πρόσθεση	Κυρίως κομμάτι πολλαπλασιασμού
	add $prod, $prod, $a	Αν το LSB του $a είναι άσσος, τότε πρόσθεσε τον $a στο $prod	
Lnext	sll $a, $a, 1 srl $b , $b, 1	Αριστερή ολίσθηση του $a κατά 1 θέση Δεξιά ολίσθηση του $b κατά 1 θέση	
	sub $count, $count, 1 bne $count, $zero, Lbpos	Μείωση του μετρητή $count κατά 1 Αν ο $count δεν είναι μηδέν, διακλαδώσου στο Lbpos	
	beq $neg, $zero, Lfinish	Αν το αποτέλεσμα πρέπει να είναι θετικό, πήγαινε στο Lfinish	Τμήμα μετατροπής θετικού σε

	nor $prod, $prod, $zero addi $prod, $prod, 1	ειδάλλως μετάτρεψε το $prod σε αρνητικό	αρνητικό
Lfinish	.end		Τέλος προγράμματος

Πολλαπλασιασμός Αριθμών με τον αλγόριθμο Booth

Θεωρούμε ότι στον καταχωρητή $a βρίσκεται ο πολλαπλασιαστέος και στον καταχωρητή $b βρίσκεται ο πολλαπλασιαστής. Το αποτέλεσμα θα αποθηκευτεί σε ένα ζεύγος καταχωρητών {$p1, $p0) των 64 bit. Στο ακόλουθο πίνακα φαίνονται οι καταχωρητές που θα χρησιμοποιήσουμε.

$p0:	Περιέχει τη λιγότερο σημαντική λέξη του γινομένου
$p1:	Περιέχει την περισσότερο σημαντική λέξη του γινομένου
$extra:	Περιέχει το 1 extra bit που χρειαζόμαστε δεξιά του γινομένου στον αλγόριθμο του Booth
$test:	Καταχωρητής για λήψη απόφασης (00, 01, 10, 11)
$r01:	Περιέχει την τιμή 0
$r10:	Περιέχει την τιμή 2
$p1msb:	Το MSBit του καταχωρητή $p1
$p1lsb:	Το LSBit του καταχωρητή $p1. Αυτοί οι δύο καταχωρητές χρησιμεύουν στη δεξιά ολίσθηση του γινομένου {$p1, $p0}.
$count:	Καταχωρητής/μετρητής που αρχικοποιείται με την τιμή 32 και στη συνέχεια μειώνεται σε κάθε βήμα του πολ/μού, μέχρι να γίνει μηδέν.

Κώδικας assembly

	addi $count, $count, 32		
	nor $s, $b, $zero addi $s, $s, 1	Ο καταχωρητής $s περιέχει το -$b (συμπλήρωμα ως προς 2)	
	addi $r01, $zero, 1 addi $r10, $zero, 2	Δημιουργία των τιμών 1 και 2 στους αντίστοιχους καταχωρητές	Τμήμα αρχικοποίησης καταχωρητών-τιμών
	add $p0, $p0, $b	Ο καταχωρητής $p0 περιέχει τον πολλαπλασιαστή $b.	
Lcheck	add $test, $p0, $zero	Μεταφέρω την τιμή του $p0 στον $test	
	sll $test, $test, 31 srl $test, $test, 30	Απομόνωσε το LSB του $test και φέρε το στη 2η θέση (από δεξιά)	Τμήμα ελέγχου τιμών 00, 01, 10, 11
	add $test, $test, $extra	Στον $test υπάρχουν τα 2 bits ελέγχου του αλγόριθμου Booth	
	beq $test, $r01, Ladda beq $test, $r10, Ladds	Διακλαδώσου στην αντίστοιχη περίπτωση	
Lshift	srl $p1msb, $p1, 31		
	sll $p1msb, $p1msb, 31	Στον $p1msb κρατάμε το MSB του $p1 που χρησιμεύει στην αριθμητική ολίσθηση του $p1	Τμήμα δεξιάς ολίσθησης συνόλου καταχωρητών {$p1, $p0, $extra}

		κατά δεξιά (κρατάει το πρόσημο), χωρίς τη χρήση εντολής sra (Shift Right Arithmetic)	
	andi $p1lsb, $p1, 1	Ο $p1lsb κρατάει το LSB του $p1 που θα πάει ως MSB του $p0. Θεωρούμε το ζεύγος {$p1, $p0} ως έναν 64bit καταχωρητή.	
	sll $p1lsb, $p1lsb, 31 srl $p1, $p1, 1 or $p1, $p1, $p1msb	Ολίσθηση του $p1 Ολοκλήρωση της «sra»	
	and $extra, $t0, 1 srl $p0, $p0, 1 or $p0, $p0, $p1lsb		
	sub $count, $count, 1 bne $count, $zero, Lcheck j Lend		Τμήμα Ελέγχου τέλους βημάτων αλγορίθμου
Ladda	add $p1, $p1, $a j Lshift	Πρόσθεσε τον πολλαπλασιαστέο στο γινόμενο	
Ladds	add $p1, $p1, $s j Lshift	Αφαίρεσε τον πολλαπλασιαστέο από το γινόμενο. Ο αντίθετός του υπάρχει ήδη στον $s.	Τμήμα Πρόσθεσης ή Αφαίρεσης Τιμής από το γινόμενο
Lend	.end		