ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ &

ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ

ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση και Προσομοίωση ενός Ομόχειρου Επεξεργαστή Αρχιτεκτονικής MIPS πέντε Βαθμίδων.

Χρήστου Παναγιώτης

Επιτροπή: Αν. Καθηγητής Δ. Πνευματικάτος (επιβλέπων)

Καθηγητής Α. Δόλλας

Επ. Καθηγητής Ι. Παπαευσταθίου

ΧΑΝΙΑ 2006

Θα ήθελα να εκφράσω τις θερμες μου ευχαριστίες στον επιβλέπων καθηγητή Πνευματικάτο Διονύσιο για τη πολύτιμη βοήθεια που μου προσέφερε καθ'όλη τη διάρκεια της παρούσας εργασίας.

Ένα μεγάλο ευχαριστώ επίσης στα παιδιά του εργαστηρίου μικροεπεξεργαστών και υλικού του Π.Κ για την βοήθεια που μου παρείχαν για τη διεκπεραίωση της εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ

1.1 ΙΣΤΟΡΙΚΗ ΑΝΑΣΚΟΠΗΣΗ

Η τεχνολογία των υπολογιστών έχει παρουσιάσει θεαματική πρόοδο τα τελευταία 55 χρόνια, από τότε, δηλαδή, που δημιουργήθηκε ο πρώτος ηλεκτρονικός υπολογιστής γενικής χρήσης. Αυτός ο γρήγορος ρυθμός ανάπτυξης είναι αποτέλεσμα τόσο της προόδου της τεχνολογίας που χρησιμοποιείται για την κατασκευή υπολογιστών, όσο και της καινοτομίας στον σχεδιασμό υπολογιστών. Σήμερα οι ηλεκτρονικοί υπολογιστές χρησιμοποιούνται ευρέως σε όλους τους τομείς της ανθρώπινης ζωής από τα πιο ασήμαντα όπως για λόγους ψυχαγωγίας μέχρι θέματα που αφορούν την ασφάλεια μίας χώρας.

Ο πρώτος ηλεκτρονικός υπολογιστής γενικού σκοπού κατασκευάστηκε κατά την διάρκεια του Β' παγκόσμιου πολέμου από τους J. Presper Eckert και John Mauchly στο Moore school του πανεπιστημίου της Πελσυλβάνιας. Ο ηλεκτρονικός υπολογιστής αυτός ονομάζεται ENIAC (Electronic Numerical Integrator and Calculator) και χρησιμοποιήθηκε από τον στρατό των Η.Π.Α. Το μέγεθος του ENIAC είναι τρεις τάξεις μεγαλύτερο από το μέγεθος των μηχανημάτων που κατασκευάζονται σήμερα. Επίσης έχει 20 καταχωρητές των 10 ψηφίων και υπάρχουν συνολικά 18.000 ηλεκτρονικές λυχνίες. Ο ENIAC ήταν ο πρώτος μέχρι τότε υπολογιστής που μπορούσε να προγραμματιστεί. Ο προγραμματισμός του γινόταν με το χέρι τοποθετώντας καλώδια και ρυθμίζοντας διακόπτες και ήταν πολύ δύσκολος και χρονοβόρος. Τα δεδομένα παρέχονταν σε διάτρητες καρτέλες.

Το 1944, ο John von Neumman γοητεύτηκε από τον ENIAC και πρότεινε έναν υπολογιστή αποθηκευμένων προγραμμάτων το οποίο ονομάζεται EDVAC (Electronic Discrete Variable Automatic Computer). Το 1946, ο Maurice Wilkes ξεκίνησε ένα ερευνητικό έργο στο Cambridge για την ανάπτυξη ενός υπολογιστή αποθήκευσης προγραμμάτων. Το όνομα του υπολογιστή ήταν EDSAC (Electronic Delay Storage

Automatic Computer). Η αρχιτεκτονική του βασιζόταν σε συσσωρευτή και παρέμεινε δημοφιλής έως τις αρχές της δεκαετίας του 1970.

Παράλληλα με την ανάπτυξη του ENIAC, Ο Howard Aiken σχεδίασε στο Harvard έναν ηλεκτρομηχανικό υπολογιστή που ονομάστηκε Mark-I. Στην συνέχεια ακολούθησε ο Mark-II, που ήταν μια μηχανή με ρελέ, και οι Mark-III και Mark-IV, που ήταν μηχανές ηλεκτρονικών λυχνιών και είχαν εξωτερική μνήμη για την αποθήκευση εντολών και δεδομένων. Η αρχιτεκτονική των Mark-III και Mark-IV ερχόταν σε αντίθεση με αυτή των υπολογιστών που αποθήκευαν προγράμματα.

Επίσης σημαντικά ήταν και τα μηχανήματα ειδικού σκοπού που δημιουργήθηκαν στην Μεγάλη Βρετανία και στις Ηνωμένες Πολιτείες. Συγκεκριμένα, στην Μεγάλη Βρετανία η προσπάθεια για αποκρυπτογράφηση των μηνυμάτων που είχαν κωδικοποιηθεί από τη γερμανική μηχανή Enigma είχε σαν αποτέλεσμα να δημιουργηθούν δύο αξιόλογα μηχανήματα. Το πρώτο ονομαζόταν BOMB και δημιουργήθηκε από τον Alan Turing και το δεύτερο ονομαζόταν COLOSSUS και δημιουργοί του ήταν οι Newman και Flowers.

Παρόμοια στις Ηνωμένες Πολιτείες εργασίες για κρυπτανάλυση οδήγησαν στην δημιουργία μιας εταιρείας που ονομαζόταν ERA (Engineering Research Associates) και η οποία υλοποίησε πολλά μηχανήματα. Επίσης στα τέλη της δεκαετίας του 1930 και στις αρχές του 1940 στην Γερμανία δημιουργήθηκαν σημαντικά μηχανήματα ειδικού σκοπού από τον Konrad Zuse. Ο Zuse ήταν ο πρώτος που δημιούργησε σύστημα κινητής υποδιαστολής.

Οι πρώτες εμπορικές αξιοποιήσεις έγιναν το 1949 όπου η Εταιρεία Υπολογιστών των Eckert και Mauchly παρουσίασε τη μηχανή BINAC. Τον Ιούνη του 1951, η εταιρεία Sperry-Rand συγχώνευσε τη μηχανή της εταιρείας Eckert-Mauchly με την ERA και πούλησαν τον πρώτο υπολογιστή τον UNIVAC I. Το 1952 η IBM πούλησε τον πρώτο της υπολογιστή τον IBM 701. Ακολούθησαν και άλλα μηχανήματα από την IBM όπως τα 702, 605,704 και 705. Το 1964 η IBM παρουσίασε τον IBM 360, ο οποίος ήταν ο πρώτος υπολογιστής που πουλιόταν σε μεγάλες ποσότητες και είχε διευθυνσιοδότηση σε byte των 8 bits και καταχωρητές γενικού σκοπού. Επίσης είχε εντολές καταχωρητή-μνήμης και κάποιες μνήμης-μνήμης. Την ίδια χρονική περίοδο η Control Data παρουσίασε τον πρώτο

υπερυπολογιστή, τον CDC 6600. Ο 6600 ήταν ο πρώτος υπολογιστής φόρτωσης-αποθήκευσης γενικής χρήσης.

Το 1978 ο Tanenbaum μελέτησε τις ιδιότητες των γλωσσών υψηλού επιπέδου και διαπίστωσε ότι τα προγράμματα είναι απλά. Αποτέλεσμα των μελετών του ήταν να προτείνει ότι οι αρχιτεκτονικές θα πρέπει να βελτιστοποιούν το μέγεθος του προγράμματος καθώς και την ευκολία της μεταγλώττισης. Η αρχιτεκτονική που παρουσιάστηκε για να ικανοποιήσει τα παραπάνω είναι αυτή του VAX. Η πρώτη μηχανή της VAX ήταν οVAX-11/780.

Τη δεκαετία του 1980, η κατεύθυνση της αρχιτεκτονικής υπολογιστών άρχισε να απομακρύνεται από την παροχή υλικού το οποίο να υποστηρίζει υψηλού επιπέδου για γλώσσες και να προσεγγίζει απλούστερες αρχιτεκτονικές. Αυτό είχε ως αποτέλεσμα τη δημιουργία υπολογιστών περιορισμένου συνόλου εντολών RISC [Patterson και Ditzel 1980]. Οι απλοί υπολογιστές φόρτωσης-αποθήκευσης όπως ο MIPS ονομάζονται κοινώς αρχιτεκτονικές RISC.

Στις μέρες μας τόσο η Intel όσο και η μεγάλη της αντίπαλος η AMD έχουν παρουσιάσει επεξεργαστές με πολύ εξελιγμένες αρχιτεκτονικές και πολλές τεχνολογικές καινοτομίες. Η Intel παρέχει επεξεργαστές Pentium 4 με τεχνολογίες Hyper threading, απλούς Pentium 4, επεξεργαστές 64-bit και τελευταία έχει παρουσιάσει επεξεργαστές διπλοπύρινους των 64-bit οι οποίοι ενσωματώνουν και την τεχνολογία Hyper threading. Από την άλλη η AMD έχει παρουσιάσει τους Athlon, Athlon64 και τελευταία επεξεργαστές διπλοπύρινους των 64-bit οι οποίοι περιέχουν και το memory controller με αποτέλεσμα η επικοινωνία με την μνήμη να γίνεται πιο γρήγορα.

1.2 ΤΟ ΜΟΝΤΕΛΟ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ

Το μοντέλο Προγραμματιστή (Instruction Set Architecture) ουσιαστικά αποκρύπτει «λεπτομέρειες» υλοποίησης ενός υπολογιστή από τον προγραμματιστή του, δίνοντας μόνο το τι εντολές έχει ο υπολογιστής, ένα ακριβή τρόπο προσδιορισμού των ενεργειών κάθε εντολής (π.χ. η ADD $1,$2,$3 θα έχει ως αποτέλεσμα την πρόσθεση των περιεχομένων των

καταχωρητών $2 και $3 και το αποτέλεσμα θα μπει στον καταχωρητή $1). Η περιγραφή αυτή συνήθως είναι σε μία ψευδογλώσσα που λέγεται Register Transfer Language. Επιπλέον, το ISA περιλαμβάνει τον προσδιορισμό κάποιων σημαιών που επηρεάζονται από την εκτέλεση κάποιας εντολής.

Το ISA για πρώτη φορά παρουσιάστηκε στα μέσα της δεκαετίας του 1960 με την εισαγωγή του όρου αρχιτεκτονική υπολογιστών από την IBM. Με την ιδέα αυτή η εταιρεία θα μπορούσε να κατασκευάζει πολλά διαφορετικά μηχανήματα, με διαφορετικό κόστος, απόδοση και τιμή. Τα μηχανήματα αυτά θα ήταν σε επίπεδο κώδικα μηχανής απόλυτα συμβατά μεταξύ τους.

1.3 Η ΤΕΧΝΙΚΗ ΤΗΣ ΟΜΟΧΕΙΡΙΑΣ

Ομοχειρία (Pipeline) είναι η τεχνική υλοποίησης στην οποία η εκτέλεση εντολών επικαλύπτεται. Η ομοχειρία εκμεταλλεύεται τον παραλληλισμό που υπάρχει μεταξύ των διαφόρων ενεργειών που χρειάζονται να γίνουν κατά την εκτέλεση μιας εντολής.

Στην ομοχειρία κάθε βήμα ολοκληρώνει ένα κομμάτι μιας εντολής. Καθένα από τα βήματα ονομάζεται βαθμίδα ή τμήμα ομοχειρίας. Οι βαθμίδες είναι συνδεδεμένες η μια με την άλλη δημιουργώντας μια αλυσίδα ομοχειρίας. Οι βαθμίδες παίρνουν σαν είσοδο τις εντολές τις επεξεργάζονται και δίνουν σαν έξοδο τα αποτελέσματα της επεξεργασίας, τα οποία τα χρησιμοποιούν οι επόμενες βαθμίδες.

Ο χρόνος μετακίνησης μιας εντολής από τη μια βαθμίδα στην επόμενη ονομάζεται κύκλος του επεξεργαστή. Επειδή όλες οι βαθμίδες λειτουργούν παράλληλα η πιο αργή βαθμίδα είναι αυτή που καθορίζει και το κύκλο του επεξεργαστή. Ο κύκλος του επεξεργαστή συνήθως είναι ένας κύκλος ρολογιού.

Ο στόχος του σχεδιαστή είναι να εξισορροπήσει τους χρόνους επεξεργασίας κάθε βαθμίδας ομοχειρίας. Ο χρόνος που χρειάζεται ο επεξεργαστής για την εκτέλεση μιας εντολής για εξισορροπημένες βαθμίδες είναι

$$\frac{\text{Χρόνος ανά εντολή στο μη-ομόχειρο επεξεργαστή}}{\text{αριθμός βαθμίδων ομοχειρίας}}$$

Επομένως, η επιτάχυνση λόγω της ομοχειρίας είναι ίση με τον αριθμό των βαθμίδων της. Η ομοχειρία μειώνει τον αριθμό κύκλων ανά εντολή (CPI).

1.4 ΣΚΟΠΟΣ ΔΙΠΛΩΜΑΤΙΚΗΣ

Η διπλωματική αυτή εργασία έχει ως σκοπό τη δημιουργία ενός πλήρως λειτουργικού επεξεργαστή πέντε βαθμίδων. Πιο συγκεκριμένα, η διπλωματική μας βοήθησε να ασχοληθούμε εκτενέστερα και λεπτομερέστερα με τα βασικά χαρακτηριστικά και τις τεχνικές που χρησιμοποιούνται σε όλους τους σύγχρονους επεξεργαστές στις μέρες μας και να τα υλοποιήσουμε ώστε να τα υποστηρίζει ο δικός μας επεξεργαστής.

Τα γενικά χαρακτηριστικά που υποστηρίζει ο επεξεργαστή μας είναι: α) υποστηρίζει την τεχνική της ομοχειρίας κατά την εκτέλεση των εντολών, β) βασίζεται στην αρχιτεκτονική Μειωμένων Συνόλων Εντολών (RISC) και συγκεκριμένα υποστηρίζει το πρώτο σετ εντολών του MIPS (MIPS I), γ) περιλαμβάνει την υλοποίηση όλων των εντολών του συνεπεξεργαστή μηδέν (Coprocessor 0), δ) υποστηρίζει όλες τις εξαιρέσεις (Excepions) που περιέχονται στο πρώτο σετ εντολών του MIPS καθώς και εξαιρέσεις που προέρχονται από το εξωτερικό περιβάλλον όπως η εξωτερική διακοπή (Interrupt) και οι τρείς εξαιρέσεις που προέρχονται από το TLB και ε) υποστηρίζει την τεχνική της προώθησης δεδομένων (Bypassing) για την αποφυγή όλων των κινδύνων δεδομένων που εμφανίζονται για τις εντολές του πρώτου σετ εντολών του MIPS.

Τέλος, για τον επεξεργαστή μας έχουμε κάνει σύνθεση σε fpga συσκευή με τη βοήθεια του εργαλείου της xilinx. Η σύνθεση που κάναμε μας έδωσε πληροφορίες για την μέγιστη ταχύτητα που μπορεί να πετύχει ο επεξεργαστή μας καθώς και τα ποσοστά χρησιμοποίησης των πόρων της fpga συσκευής όταν υλοποιήσουμε τον επεξεργαστή μας σε μία fpga συσκευή.

Επίσης, με μελέτη των αποτελεσμάτων που δίνει το εργαλείο της σύνθεσης διαπιστώσαμε σε ποια τμήματα του επεξεργαστή μας παρουσιάζονται οι μεγαλύτερες καθυστερήσεις και προτείνουμε μελλοντικές βελτιώσεις σε ορισμένα τμήματα του επεξεργαστή για μείωση των καθυστερήσεων.

1.5 ΔΟΜΗ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Η δομή τη διπλωματικής εργασίας περιλαμβάνει τα εξής κεφάλαια. Το πρώτο κεφάλαιο περιλαμβάνει μία εισαγωγή και μία ιστορική αναδρομή για τους υπολογιστές.

Το δεύτερο κεφάλαιο περιγράφει τα βασικά χαρακτηριστικά και τις αρχές που διέπουν την αρχιτεκτονική του MIPS.

Το τρίτο κεφάλαιο περιγράφει τη βασική ιδέα της ομοχειρίας καθώς και την βασική υλοποίηση ομοχειρίας πέντε βαθμίδων για αρχιτεκτονικές μειωμένου συνόλου εντολών (Reduced Instruction Set Computer). Για κάθε στάδιο παραθέτουμε τα αντίστοιχα μπλοκ διαγράμματα για την καλύτερη κατανόηση της εργασίας. Επίσης, περιλαμβάνει τους κινδύνους (hazards) της ομοχειρίας, τα προβλήματα επιδόσεων που δημιουργούν και πως τους αντιμετωπίζουμε στην εργασία μας παραθέτοντας παράλληλα και τα κατάλληλα παραδείγματα. Τέλος, περιλαμβάνει ποιες εξαιρέσεις υλοποιήσαμε και πως αλληλεπιδρούν με την ομοχειρία.

Το τέταρτο κεφάλαιο περιλαμβάνει τα αποτελέσματα της σύνθεσης με το εργαλείο ISE 7.1.02i της Xilinx και τα διαγράμματα χρονισμού που προκύπτουν από την προσομοίωση καθώς και τις συγκρίσεις αυτών με τα αποτελέσματα που δίνει ο προσομοιωτής PCSPIM ώστε να επιβεβαιώσουμε τη σωστή λειτουργία του επεξεργαστή μας. Με τα τεστ που δημιουργήσαμε προσπαθήσαμε να καλύψουμε ένα μεγάλο εύρος από τις καταστάσεις στις οποίες μπορεί να βρεθεί ο επεξεργαστής. Πρέπει να σημειωθεί ότι έλεγχοι γίνονταν κατά την διάρκεια της υλοποίησης ώστε να διασφαλίζεται η σωστή λειτουργία κάθε κομματιού που δημιουργούσαμε.

Το πέμπτο κεφάλαιο περιλάμβανει τα συμπεράσματα από την εργασία και περαιτέρω βελτιώσεις.

ΚΕΦΑΛΑΙΟ 2

Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ MIPS

2.1 ΕΙΣΑΓΩΓΗ

Στο κεφάλαιο αυτό περιγράφουμε μια αρχιτεκτονική φόρτωσης αποθήκευσης (RISC) και συγκεκριμένα της αρχιτεκτονικής του MIPS. Την αρχιτεκτονική του MIPS τη διαλέξαμε γιατί είναι ένα καλό αρχιτεκτονικό μοντέλο για μελέτη. Το μοντέλο του MIPS δίνει έμφαση στην απλότητα, στην ευχέρεια υλοποίησης, στην ευχέρεια χρήσης, στην ελαχιστοποίηση του κόστους υλοποίησης και στην επίτευξη υψηλών ταχυτήτων. Οι βασικές αρχές που διέπουν την αρχιτεκτονική RISC, και συνεπώς και το αρχιτεκτονικό μοντέλο του MIPS, είναι τα εξής:

- Η ομοιομορφία των λειτουργιών συμβάλλει στην απλότητα υλικού. Όλοι οι τελεστές των αριθμητικών και λογικών πράξεων είναι τρεις και οι μόνες εντολές που επηρεάζουν την μνήμη είναι οι εντολές φόρτωσης και αποθήκευσης. Οι εντολές αυτές μετακινούν δεδομένα ανάμεσα στους καταχωρητές και την μνήμη.
- Οσο μικρότερο το κύκλωμα, τόσο πιο γρήγορο. Όλες οι πράξεις σε δεδομένα εφαρμόζονται σε καταχωρητές και επηρεάζουν όλο το εύρος τους.
- Η καλή σχεδίαση απαιτεί συμβιβασμούς. Οι κατηγορίες μορφοποίησης των εντολών είναι λίγες και όλες οι εντολές έχουν το ίδιο μέγεθος.
- Κάνε γρήγορο ότι είναι συχνό. Υπάρχει τελεστής πηγής που είναι σταθερή ποσότητα για να αποφεύγονται οι προσβάσεις στην μνήμη για τη φόρτωση μιας σταθερής ποσότητας που απαιτείται για την πράξη.

2.2 ΚΑΤΑΧΩΡΗΤΕΣ ΓΙΑ ΤΟΝ MIPS

Οι καταχωρητές του MIPS έχουν τα εξής χαρακτηριστικά:

- 32 καταχωρητές γενικής χρήσης των 32-bit (GPR), που ονομάζονται R0, R1, …, R31.
- 32 καταχωρητές κινητής υποδιαστολής (FPR) που ονομάζονται F0, F1, …, F31, οι οποίοι αποθηκεύουν 32 τιμές απλής ακρίβειας (32-bit) ή 32 τιμές διπλής ακρίβειας (64-bit).
- Ο R0 έχει πάντα τη τιμή μηδέν και ο R31 περιέχει τη διεύθυνση προορισμού όταν η εντολή είναι άλμα και σύνδεση το οποίο χρησιμοποιείται για τις κλήσεις των υποπρογραμμάτων.
- Υπάρχουν και ορισμένοι ειδικοί καταχωρητές που οι τιμές τους μπορούν να μεταφερθούν από και προς του καταχωρητές γενικού σκοπού. Ένας τέτοιος καταχωρητής είναι ο HI register ο οποίος κρατάει τα πιο σημαντικά 32 bits της 64 bits ποσότητας του γινομένου.
- Υπάρχουν και οι καταχωρητές του συνεπεξεργαστή μηδέν (Coprocessor 0). Ένας τέτοιος καταχωρητής είναι ο status register ο οποίος κρατάει πληροφορίες για την κατάσταση του επεξεργαστή.
- Ο μετρητής προγράμματος (PC) είναι ένας ειδικός καταχωρητής ο οποίος έχει 32 bits και δείχνει στο σημείο στο οποίο βρίσκεται η εκτέλεση του προγράμματος.

Register	Symbolic name	Description
$0	z0, ZERO	Always contains 0, no matter what's written to it.
$1	AT	Assembler temporary.
$2	v0	Value 0. Used for computations; function return value is placed here.
$3	v1	Value 1. Used for computations; upper word of 64-bit return value is placed here. Also holds the system call number on syscall entry.
$4	a0	Argument 0. First function argument goes here.
$5	a1	Argument 1. Second function argument goes here.
$6	a2	Argument 2. Third function argument goes here.
$7	a3	Argument 3. Fourth function argument goes here.
$8	t0	General-purpose temporary register.
$9	t1	General-purpose temporary register.
$10	t2	General-purpose temporary register.
$11	t3	General-purpose temporary register.
$12	t4	General-purpose temporary register.
$13	t5	General-purpose temporary register.
$14	t6	General-purpose temporary register.
$15	t7	General-purpose temporary register.
$16	s0	General-purpose saved register.
$17	s1	General-purpose saved register.
$18	s2	General-purpose saved register.
$19	s3	General-purpose saved register.

$20	s4	General-purpose saved register.
$21	s5	General-purpose saved register.
$22	s6	General-purpose saved register.
$23	s7	General-purpose saved register.
$24	t8	General-purpose temporary register.
$25	t9	General-purpose temporary register.
$26	k0	Kernel scratch register.
$27	k1	Kernel scratch register.
$28	gp	Global pointer. Constant for any given process.
$29	sp	Stack pointer.
$30	s8	Saved register #8 - conventionally, but not always, a frame pointer.
$31	ra	Return address of function.

Πίνακας 2.1: Καταχωρητές Γενικού Σκοπού.

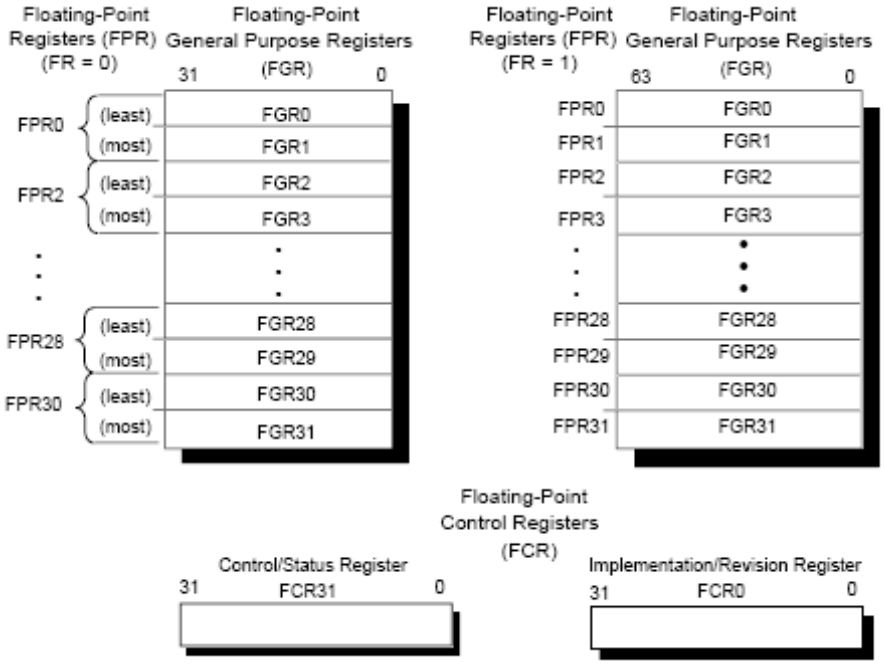
.

Register	**Description**
LO	High-order word of 64-bit multiply result, or remainder of divide result.
HI	Low-order word of 64-bit multiply result, or quotient of divide result.
PC	Program counter.

Πίνακας 2.2: Ειδικοί Καταχωρητές

Register	Symbolic name	Description
cop0 $0	c0_index	TLB entry index register.
cop0 $1	c0_random	TLB randomized access register.
cop0 $2	c0_entrylo	Low-order word of "current" TLB entry.
cop0 $4	c0_context	Page-table lookup address.
cop0 $8	c0_vaddr	Virtual address associated with certain exceptions.
cop0 $10	c0_entryhi	High-order word of "current" TLB entry.
cop0 $12	c0_status	Processor status register.
cop0 $13	c0_cause	Exception cause register.
cop0 $14	c0_epc	PC at which exception occurred.

Πίνακας 2.3: Καταχωρητές Coprocessor 0.



Σχήμα 2.1: Καταχωρητές FPU.

2.3 ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

Οι τύποι δεδομένων για τα ακέραια δεδομένα είναι:

- bytes των 8 bit
- μισές λέξεις των 16 bit (half-word)
- λέξεις των 32 bits (word)
- διπλές λέξεις των 64 bit (double word)

Οι τύποι δεδομένων για δεδομένα κινητής υποδιαστολής είναι:

- 32-bit απλής ακρίβειας (single precision)
- 64-bit διπλής ακρίβειας (double precision)

Οι λειτουργίες του MIPS γίνονται με ακέραιους 32 bit και με αριθμούς κινητής υποδιαστολής των 32 ή των 64 bit. Τα bytes, οι μισές λέξεις και οι λέξεις φορτώνονται σε καταχωρητές γενικού σκοπού είτε με μηδενικά (zero filling), είτε με το bit προσήμου (sign extension) για να γεμίσει όλα τα bits του καταχωρητή γενικού σκοπού.

2.4 Η ΜΝΗΜΗ ΤΟΥ MIPS

2.4.1 Διευθύνσεις byte, περιορισμοί ευθυγράμμισης (ALIGNMENT) και προσπελάσεις μνήμης.

Κάποια σημαντικά χαρακτηριστικά της μνήμης είναι τα παρακάτω:

- Οι τρόποι διευθυνσιοδότησης για μεταφορές δεδομένων του MIPS είναι:
  - Η άμεση με πεδίο 16 bit πραγματοποιείται χρησιμοποιώντας τον καταχωρητή R0 ως καταχωρητή βάσης
  - Η έμμεση πραγματοποιείται τοποθετώντας το μηδέν στο πεδίο εκτόπισης των 16 bit.
- Η μνήμη είναι οργανωμένη σε λέξεις των 32-bits.
- Οι διευθύνσεις μνήμης είναι διευθύνσεις byte (8 bit).

- Οι προσβάσεις στην μνήμη είναι ευθυγραμμισμένες (ALIGNMENT). Μια λέξη καταλαμβάνει τις εξής διευθύνσεις των 4 bytes A_0, A_0+1, A_0+2, A_0+3 όπου A_0 ακέραιο πολλαπλάσιο του 4. Η πρώτη λέξη έχει διεύθυνση 0, η δεύτερη έχει διεύθυνση 4 κ.λ.π. Τα παραπάνω φαίνονται στον παρακάτω πίνακα.
- Οι εντολές φόρτωσης και αποθήκευσης είναι οι μόνες που επηρεάζουν την μνήμη οι οποίες μεταφέρουν δεδομένα από τους καταχωρητές στην μνήμη και από την μνήμη στους καταχωρητές αντίστοιχα. Οι εντολές αυτές φορτώνουν ή αποθηκεύουν ένα μέρος του καταχωρητή των 16 ή 32 bits.

Address (8 bits)	Data(8 bits)
0	A_0
1	A_0+1
2	A_0+2
3	A_0+3
4	
5	2^{nd} word
6	
7	
8	
9	3^{rd} word
10	
11	
12	
13	4^{th} word
14	
15	

Πίνακας 2.4:Η μνήμη του MIPS.

2.4.2 Αρίθμηση των bytes

Για την αποθήκευση μιας ποσότητας στην μνήμη υπάρχουν δύο τρόποι ανάλογα με το ποιο byte περιέχει τα πιο σημαντικά bits της ποσότητας και ποιο τα λιγότερο σημαντικά. Ο ένας τρόπος ονομάζεται Big-Endian και ξεκινά την αρίθμηση των bytes από το "μεγάλο άκρο"(big end) δηλαδή από το περισσότερο σημαντικό byte (MSB). Το περισσότερο σημαντικό byte της ποσότητας έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των bytes του αυξάνουν καθώς προχωράμε προς το λιγότερο σημαντικό byte του (LSB). Ενώ στην Little-Endian τεχνική η αρίθμηση των bytes ξεκινά από το "μικρό άκρο"(little end) δηλαδή από το λιγότερο σημαντικό byte (LSB). Το λιγότερο σημαντικό byte της ποσότητας έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των bytes του αυξάνουν καθώς προχωράμε προς το περισσότερο σημαντικό byte του (MSB). Ο επεξεργαστής μας χρησιμοποιεί την τεχνική Big-Endian.

Διεύθυνση	**Δεδομένα**			
	MS			**LS**
0	0	1	2	3
4	4	5	6	7

Πίνακας 2.5: Αρίθμηση Bytes με Big-Endian.

Διεύθυνση	**Δεδομένα**			
	MS			**LS**
0	3	2	1	0
4	7	6	5	4

Πίνακας 2.6: Αρίθμηση Bytes με Little-Endian.

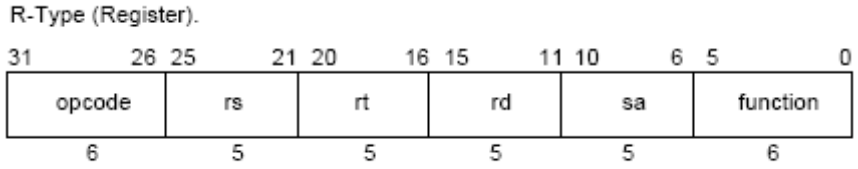
2.5 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΝΟΛΟΥ ΕΝΤΟΛΩΝ ΤΟΥ MIPS

2.5.1 Εισαγωγή

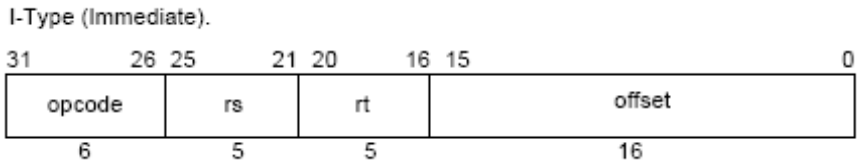
Όλες οι εντολές έχουν πλάτος 32 bits και οι διευθύνσεις στις οποίες βρίσκονται είναι διευθύνσεις λέξεων (δηλαδή είναι πολλαπλάσια του 4). Στον ορισμό όλων των εντολών εκτός των εντολών ελέγχου ροής, υπονοείται και η αύξηση του μετρητής προγράμματος κατά 4 ώστε η εκτέλεση να προχωρήσει στην επόμενη εντολή. Η εκτέλεση των εντολών γίνεται στην σειρά, από μικρότερες διευθύνσεις προς μεγαλύτερες, εκτός εάν η ροή του προγράμματος αλλάξει λόγω εντολής ελέγχου ροής.(branch, ή jump). Όλες οι αριθμητικές και λογικές πράξεις αποθηκεύουν το αποτέλεσμά τους σε καταχωρητή.

Υπάρχουν τρεις διαφορετικές μορφές εντολών που είναι οι εξής:

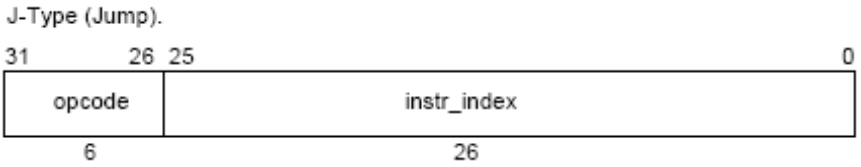
- R-type (Register)



- I-type (Immediate)



- J-type (Jump)



Τα πεδία των εντολών είναι:

- **opcode:** 6 bits κύριος κωδικός εντολής
- **rt:** 5 bits δεύτερος καταχωρητής πηγής /προορισμού
- **rs:** 5 bits πρώτος καταχωρητής πηγής
- **rd:** 5 bits καταχωρητή προορισμού
- **sa:** 5 bits για ολίσθηση
- **function:** 6 bits πράξη για εντολές R-type
- **offset:** 16 bits σταθερές για εντολές I-type
- **instr_index:** 26 bits σταθερές για εντολές J-type

2.5.2 Η μορφή εντολών (R-type)

Αυτός ο τύπος κωδικοποιεί:

- Εντολές που εκτελούν μια αριθμητική ή λογική πράξη ανάμεσα στους καταχωρητές rs και rt. Το αποτέλεσμα τοποθετείται στον καταχωρητή rd.

rd ← rs function rt

- Εντολές ολίσθησης , οι οποίες ολισθαίνουν το καταχωρητή rt αριστερά ή δεξιά κατά μια ποσότητα που ορίζεται είτε από το καταχωρητή sa είτε από το καταχωρητή rs. Το αποτέλεσμα τοποθετείται στον καταχωρητή rd.

rd ← rt shift rs/sa

- Εντολές που γράφουν ή διαβάζουν ειδικούς καταχωρητές.

2.5.3 Η μορφή εντολών (I-type)

Αυτός ο τύπος κωδικοποιεί:

- Φορτώσεις και αποθηκεύσεις bytes, μισών λέξεων και λέξεων. Ο υπολογισμός της διεύθυνσης μνήμης γίνεται με το άθροισμα του καταχωρητή πηγής rs με τη σταθερή offset μετά την επέκταση του προσήμου. Ο καταχωρητής rt στην περίπτωση της αποθήκευσης περιέχει τα

δεδομένα που θα γραφτούν στη μνήμη ενώ στην περίπτωση της φόρτωσης ο rt είναι η πηγή προορισμού για τα δεδομένα που διαβάζονται από τη μνήμη.

rt ← mem[offset+rs]

mem[offset+rs] ←rt

- Εντολές που εκτελούν μια πράξη ανάμεσα στον καταχωρητή rs και την σταθερή offset επεκταμένη ως προς το πρόσημο. Το αποτέλεσμα αποθηκεύεται στο καταχωρητή rt.

Rt ← rs opcode offset

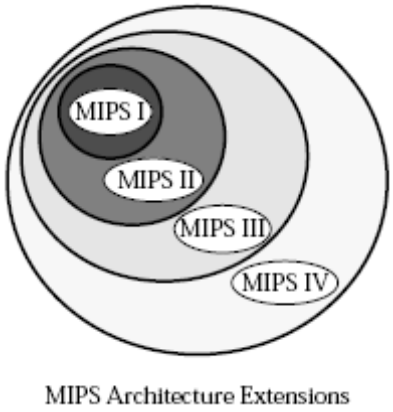
- Εντολές υπό συνθήκη διακλάδωσης. Η σύγκριση γίνεται είτε μεταξύ των καταχωρητών rs και rt είτε μεταξύ του καταχωρητή rs και του μηδέν. Ο υπολογισμός της διεύθυνσης προορισμού γίνεται προσθέτοντας τη σταθερή offset με το μετρητή του προγράμματος.
- Εντολές καταχωρητή άλματος και καταχωρητή άλματος και σύνδεσης. Ο καταχωρητής rs προσδιορίζει τη διεύθυνση προορισμού της διακλάδωσης. Ο καταχωρητής rt και η σταθερά offset είναι μηδέν.

2.5.4 Η μορφή εντολών (J-type)

Αυτός ο τύπος κωδικοποιεί εντολές άλματος και άλματος και σύνδεση. Ο υπολογισμός της διεύθυνσης προορισμού γίνεται ολισθαίνοντας κατά δύο προς τα αριστερά τη ποσότητα instr_index και βάζοντας αριστερά αυτής της ποσότητας τα τέσσερα πιο σημαντικά bit του μετρητή προγράμματος.

2.5.5 Το σύνολο εντολών του MIPS

Η αρχιτεκτονική του MIPS περιέχει τέσσερα σύνολα εντολών. Το πρώτο σέτ εντολών περιέχει τις βασικές εντολές της αρχιτεκτονικής του MIPS. Τα τρία υπερσύνολα που έρχονται να καλύψουν πλήρως την αρχιτεκτονική του MIPS είναι το καθένα συμβατό προς τα πίσω με τα σύνολα εντολών που περικλείουν.

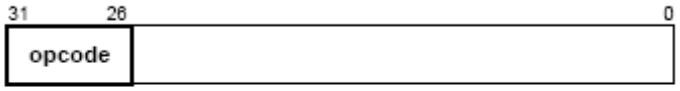


Σχήμα 2.2: Τα τέσσερα σύνολα εντολών και η μεταξύ τους ιεραρχία

Στο σύνολο MIPS I ανήκουν εντολές που εκτελούν φορτώσεις και αποθηκεύσεις δεδομένων από και προς την μνήμη αντίστοιχα, αριθμητικές και λογικές πράξεις, άλματα και διακλαδώσεις, εξαιρέσεις και μεταφορές δεδομένων για τον συνεπεξεργαστή.

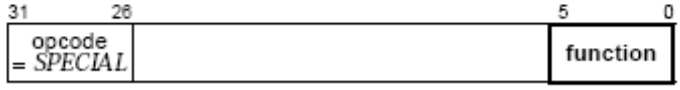
Αναλυτικά οι εντολές που περιλαμβάνει το πρώτο σετ εντολών φαίνονται στο παρακάτω πίνακα:

Table A-37 CPU Instruction Encoding - MIPS I Architecture



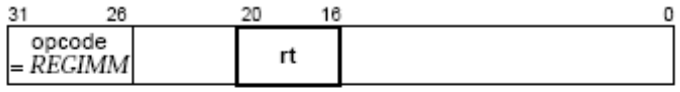
Instructions encoded by opcode field.

opcode bits 31..29 / bits 28..26	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	*SPECIAL* δ	*REGIMM* δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1 001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2 010	*COP0* δ,π	*COP1* δ,π	*COP2* δ,π	*COP3* δ,π,κ	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	LB	LH	LWL	LW	LBU	LHU	LWR	*
5 101	SB	SH	SWL	SW	*	*	SWR	*
6 110	*	LWC1 π	LWC2 π	LWC3 π,κ	*	*	*	*
7 111	*	SWC1 π	SWC2 π	SWC3 π,κ	*	*	*	*



Instructions encoded by function field when opcode field = SPECIAL.

function bits 5..3 / bits 2..0	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1 001	JR	JALR	*	*	SYSCALL	BREAK	*	*
2 010	MFHI	MTHI	MFLO	MTLO	*	*	*	*
3 011	MULT	MULTU	DIV	DIVU	*	*	*	*
4 100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5 101	*	*	SLT	SLTU	*	*	*	*
6 110	*	*	*	*	*	*	*	*
7 111	*	*	*	*	*	*	*	*



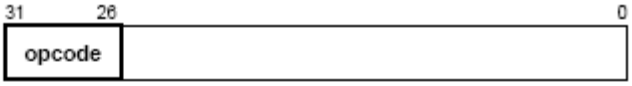
Instructions encoded by the rt field when opcode field = REGIMM.

rt bits 20..19 / bits 18..16	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00	BLTZ	BGEZ	=	=	=	=	=	=
1 01	=	=	=	=	=	=	=	=
2 10	BLTZAL	BGEZAL	=	=	=	=	=	=
3 11	=	=	=	=	=	=	=	=

Πίνακας 2.7:Το πρώτο σετ εντολών (MIPS I).

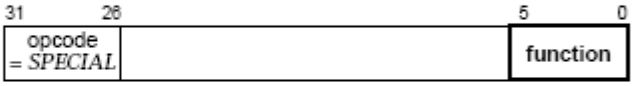
Το σύνολο MIPS II, εκτός από τις εντολές του MIPS I, περιλαμβάνει επιπλέον και μεταφορές από και προς τον συνεπεξεργαστή δεδομένων των 64 bit. Επίσης κάποιες ειδικές διακλαδώσεις αλλά και κάποιες υπό συνθήκη παγίδες (trap). Αναλυτικά οι εντολές φαίνονται στο παρακάτω πίνακα.

Table A-42 CPU Instruction Encoding Changes - MIPS II Revision.

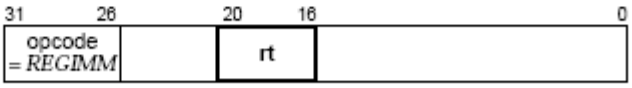


An instruction encoding is shown if the instruction is added in this revision.

opcode bits 31..26	Instructions encoded by opcode field.							
bits	0	1	2	3	4	5	6	7
	000	001	010	011	100	101	110	111
0 000								
1 001								
2 010					BEQL	BNEL	BLEZL	BGTZL
3 011								
4 100								
5 101								ρ
6 110	LL					LDC1 π	LDC2 π	LDC3 π
7 111	SC					SDC1 π	SDC2 π	SDC3 π



function bits 2..0	Instructions encoded by function field when opcode field = SPECIAL.							
bits	0	1	2	3	4	5	6	7
5..3	000	001	010	011	100	101	110	111
0 000								
1 001								SYNC
2 010								
3 011								
4 100								
5 101								
6 110	TGE	TGEU	TLT	TLTU	TEQ		TNE	
7 111								



rt bits 18..16	Instructions encoded by the rt field when opcode field = REGIMM.							
bits	0	1	2	3	4	5	6	7
20..19	000	001	010	011	100	101	110	111
0 00			BLTZL	BGEZL				
1 01	TGEI	TGEIU	TLTI	TLTIU	TEQI		TNEI	
2 10			BLTZALL	BGEZALL				
3 11								

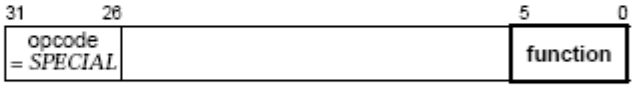
Πίνακας 2.8:Το δεύτερο σετ εντολών (MIPS II)

Το MIPS III, εκτός από τα δύο προηγούμενα σύνολα, περιλαμβάνει εντολές που εκτελούν αριθμητικές πράξεις ανάμεσα σε 64 bit δεδομένα, ολισθήσεις σε δεδομένα των 64 bit καθώς και ορισμένες φορτώσεις δεδομένων των 64 bit από την μνήμη. Αναλυτικά οι εντολές φαίνονται στο παρακάτω πίνακα.

An instruction encoding is shown if the instruction is added or modified in this revision.

Instructions encoded by opcode field.

opcode bits 28..26	0	1	2	3	4	5	6	7
bits 31..29	000	001	010	011	100	101	110	111
0 000								
1 001								
2 010				* (was *COP3*)				
3 011	DADDI	DADDIU	LDL	LDR				
4 100								LWU
5 101					SDL	SDR		
6 110				* (was LWC3)	LLD			LD (was LDC3)
7 111				* (was SWC3)	SCD			SD (was SDC3)

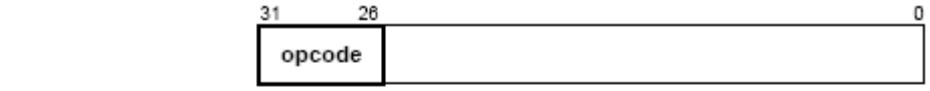


Instructions encoded by function field when opcode field = SPECIAL.

function bits 2..0	0	1	2	3	4	5	6	7
bits 5..3	000	001	010	011	100	101	110	111
0 000								
1 001								
2 010					DSLLV		DSRLV	DSRAV
3 011					DMULT	DMULTU	DDIV	DDIVU
4 100								
5 101					DADD	DADDU	DSUB	DSUBU
6 110								
7 111	DSLL		DSRL	DSRA	DSLL32		DSRL32	DSRA32

Πίνακας 2.9:Το τρίτο σετ εντολών (MIPS III)

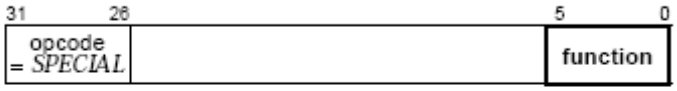
Το MIPS IV, εκτός από τα προηγούμενα τρία σύνολα, περιλαμβάνει εντολές που εκτελούν μεταφορές δεδομένων από και προς τον συνεπεξεργαστή για τη μονάδα κινητής υποδιαστολής. Υπό συνθήκη μεταφορές 64 bit δεδομένων ανάμεσα στους καταχωρητές γενικής χρήσης.



An instruction encoding is shown if the instruction is added or modified in this revision.

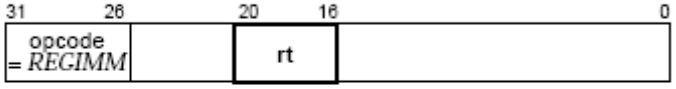
Instructions encoded by opcode field.

opcode bits 28..26 / bits 31..29	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000								
1 001								
2 010				*COP1X* δ,π				
3 011								
4 100								
5 101								
6 110				PREF				
7 111								



Instructions encoded by function field when opcode field = SPECIAL.

function bits 2..0 / bits 5..3	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 000		*MOVCI* δ,μ						
1 001			MOVZ	MOVN				
2 010								
3 011								
4 100								
5 101								
6 110								
7 111								



Instructions encoded by the rt field when opcode field = REGIMM.

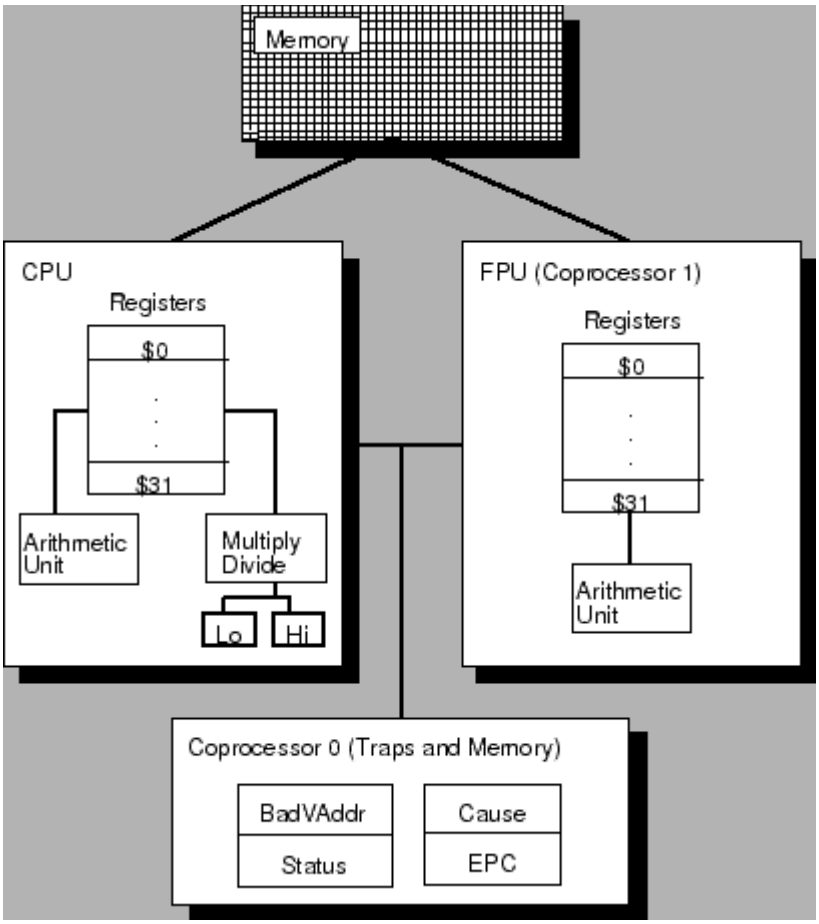
rt bits 18..16 / bits 20..19	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 00								
1 01								
2 10								
3 11								

Πίνακας 2.10:Το τέταρτο σετ εντολών (MIPS IV)

.

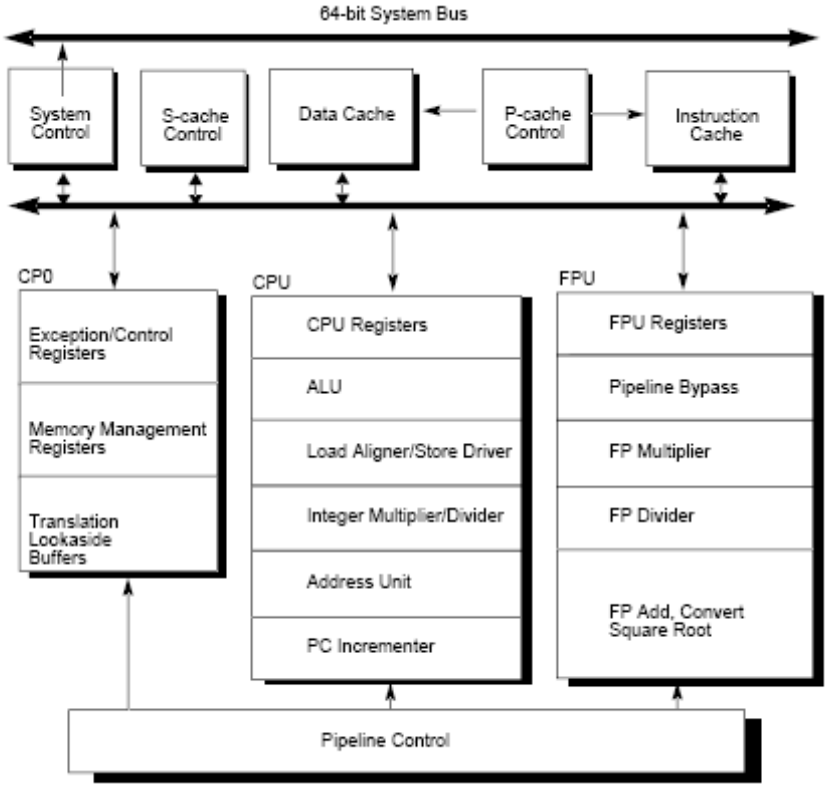
2.6 ΔΙΑΦΟΡΕΣ ΥΛΟΠΟΙΗΣΕΙΣ ΤΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ MIPS.

Στο παρακάτω σχήμα φαίνεται το μπλοκ διάγραμμα μιας απλής υλοποίησης ενός επεξεργαστή αρχιτεκτονικής MIPS. Ο επεξεργαστής αυτός αποτελείται από μια κεντρική μονάδα επεξεργασίας (CPU) και δύο συνεπεξεργαστές οι οποίοι εκτελούν διαφορετικές εργασίες. Η κεντρική μονάδα επεξεργασίας εκτελεί τις αριθμητικές πράξεις μεταξύ των ακεραίων. Ο συνεπεξεργαστής 0 διαχειρίζεται τις εξαιρέσεις και τη εικονική μνήμη του συστήματος. Τέλος, ο συνεπεξεργαστής 1εκτελεί τις πράξεις ανάμεσα σε αριθμούς κινητής υποδιαστολής.

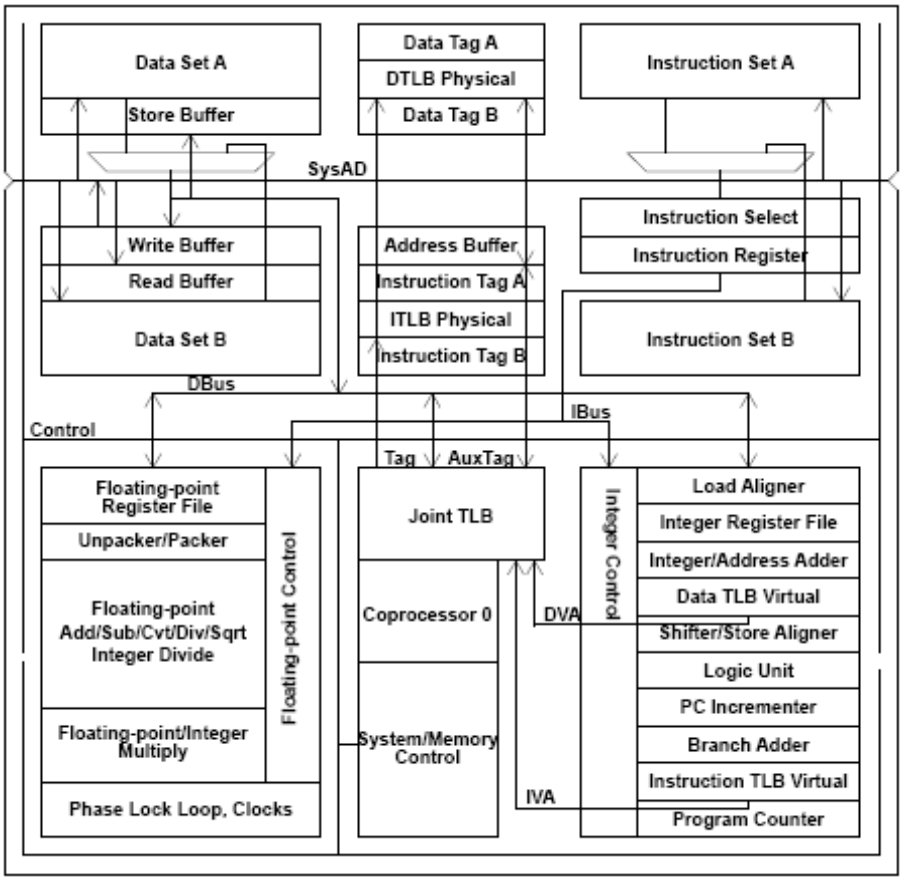


Σχήμα 2.3: Μπλοκ διάγραμμα του MIPS R2000.

Άλλη μια πιο περίπλοκη υλοποίηση είναι ο 64-bit επεξεργαστής R4000 του οποίου το μπλοκ διάγραμμα φαίνεται στο σχήμα 2.3. Και σε αυτό το μπλοκ διάγραμμα διακρίνονται η κεντρική μονάδα επεξεργασίας και οι συνεπεξεργαστές 0 και 1. Επίσης μια 64 bit υλοποίηση είναι ο επεξεργαστής MIPS R4600 του οποίου το μπλοκ διάγραμμα φαίνεται στο σχήμα 2.4.

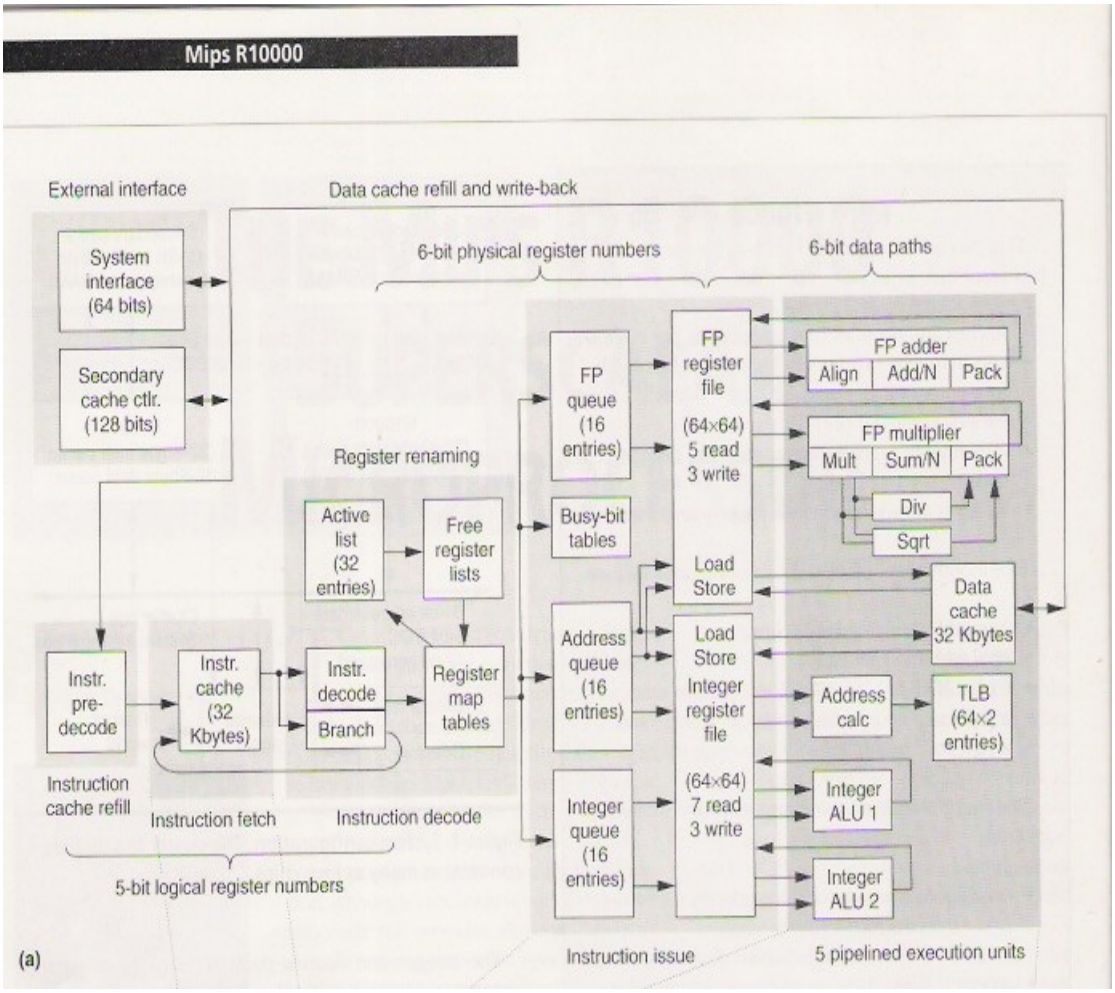


Σχήμα 2.4: Μπλοκ διάγραμμα του MIPS R4000.



Σχήμα 2.5: Μπλοκ διάγραμμα του MIPS R4600.

Τέλος μια άλλη υλοποίηση αρχιτεκτονικής MIPS είναι ο superscalar MIPS R10000 ο οποίος ανακτά και αποκωδικοποιεί τέσσερις εντολές ανά κύκλο και εκτελεί εκτός σειράς τις εντολές με δυναμικό τρόπο. Το μπλοκ διάγραμμα του επεξεργαστή φαίνεται στο παρακάτω σχήμα.



Σχήμα 2.6: Μπλοκ διάγραμμα του SUPERSCALAR MIPS R10000.

ΚΕΦΑΛΑΙΟ 3

ΠΕΡΙΓΡΑΦΗ ΥΛΟΠΟΙΗΣΗΣ ΤΟΥ ΣΥΝΟΛΟΥ ΕΝΤΟΛΩΝ ΤΟΥ MIPS I

3.1 ΕΙΣΑΓΩΓΗ

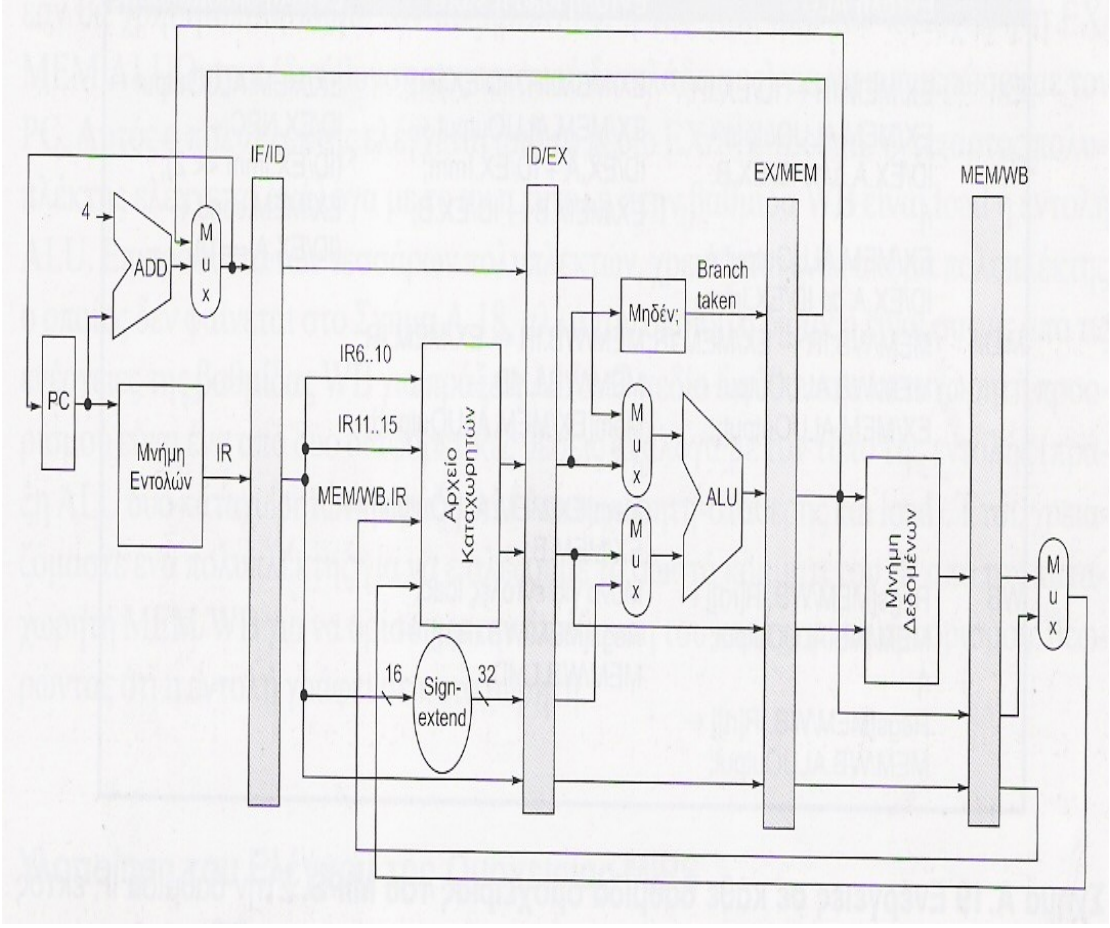
Στο κεφάλαιο αυτό παρουσιάζουμε την υλοποίηση της ομοχειρίας πέντε βαθμίδων και περιγράφουμε αναλυτικά την λειτουργία κάθε βαθμίδας. Η υλοποίηση μας βασίζεται στην αρχιτεκτονική Μειωμένων Συνόλων Εντολών (RISC) και συγκεκριμένα στο πρώτο σετ εντολών του MIPS (MIPS I). Στην συνέχεια παρουσιάζουμε την μονάδα ελέγχου του επεξεργαστή. Επίσης παρουσιάζουμε πως υλοποιήσαμε την προώθηση δεδομένων για την αποφυγή εξαρτήσεων δεδομένων καθώς και τον τρόπο που υποστηρίξαμε τις διακοπές (Interrupts) που περιέχονται στο πρώτο σύνολο του MIPS και τις εξωτερικές διακοπές που προέρχονται από το εξωτερικό περιβάλλον.

3.2 Η ΒΑΣΙΚΗ ΥΛΟΠΟΙΗΣΗ ΟΜΟΧΕΙΡΙΑΣ ΠΕΝΤΕ ΒΑΘΜΙΔΩΝ

Στη βασική ομοχειρία πέντε βαθμίδων κάθε κύκλο ξεκινάει μια νέα εντολή. Επειδή κάθε βαθμίδα ομοχειρίας είναι ενεργή κάθε κύκλο, όλες οι λειτουργίες σε μια βαθμίδα πρέπει να ολοκληρωθούν σε 1 κύκλο και κάθε συνδυασμός λειτουργιών πρέπει να μπορεί να συμβεί ταυτόχρονα. Ακόμα, η ομοχειρία του μονοπατιού δεδομένων απαιτεί το πέρασμα των αποτελεσμάτων μεταξύ βαθμίδων μέσω καταχωρητών. Στο σχήμα 3.1 φαίνεται η ομοχειρία του MIPS με τους κατάλληλους καταχωρητές ομοχειρίας ανάμεσα από τις βαθμίδες.

Όλοι οι καταχωρητές που χρειάζονται για να κρατάνε τις προσωρινές τιμές κατά την διάρκεια μιας εντολής έχουν ενσωματωθεί στους καταχωρητές ομοχειρίας. Οι καταχωρητές ομοχειρίας μεταφέρουν τόσο δεδομένα όσο και bits ελέγχου από την μια βαθμίδα στην

επόμενη. Κάθε τιμή η οποία χρειάζεται αργότερα στην ομοχειρία πρέπει να τοποθετηθεί σε ένα τέτοιο καταχωρητή και να αντιγράφετε από ένα καταχωρητή ομοχειρίας στον επόμενο, μέχρις ότου δεν χρειάζεται πλέον. Κάθε εντολή, όπως είπαμε πρίν, είναι ενεργή σε ακριβώς μία βαθμίδα ομοχειρίας σε κάθε χρονική στιγμή. Έτσι, κάθε ενέργεια λαμβάνει χώρα για την εντολή που βρίσκεται μεταξύ ζευγαριών καταχωρητών ομοχειρίας. Συνεπώς, μπορούμε να θεωρήσουμε τις ενέργειες στην ομοχειρία εξετάζοντας τι συμβαίνει σε κάθε βαθμίδα ανάλογα με τον τύπο της εντολής.



Σχήμα 3.1: Η βασική ομοχειρία πέντε βαθμίδων.

3.2.1 Κύκλος πρώτος: Ανάκληση εντολής (Instruction Fetch)

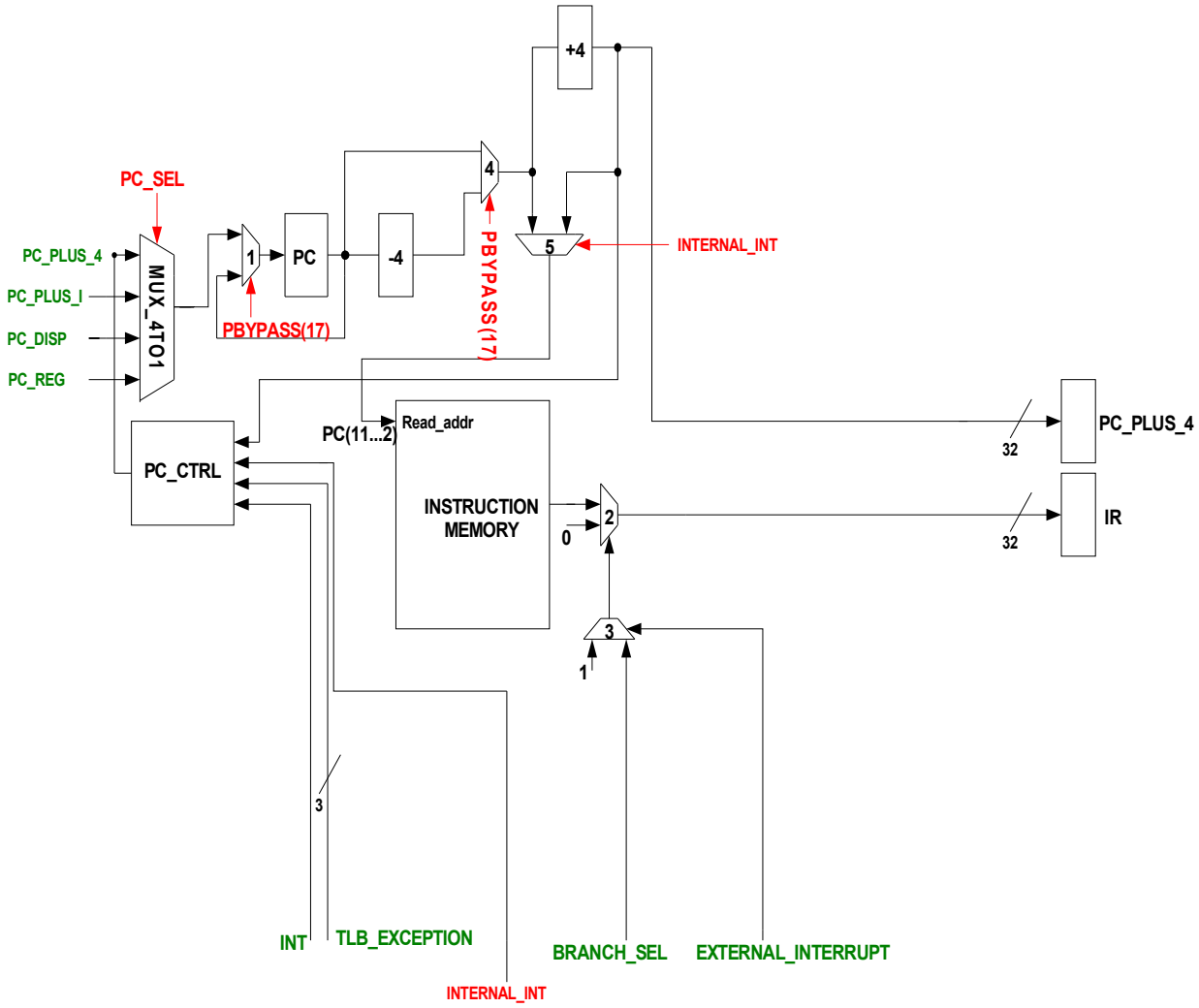
Επιλογή του κατάλληλου μετρητή προγράμματος (Program Counter) μέσω ενός πολυπλέκτη 4 σε 1, αποστολή του σωστού PC στην μνήμη και ανάκληση της εντολής από την μνήμη στον καταχωρητή εντολής (IR). Αύξηση του PC κατά τέσσερα ώστε να δείχνει στην επόμενη ακολουθιακά εντολή, η νέα τιμή του PC τοποθετείται στον καταχωρητή PC_PLUS_4.

Με τον πρώτο και τέταρτο πολυπλέκτη 2 σε 1 (βλέπε σχήμα 3.2) καθυστερούμε την εκτέλεση της εντολής και όλες όσες ακολουθούν τόσους κύκλους ώστε το αποτέλεσμα το οποίο χρειάζεται η υπό εκτέλεση εντολή να είναι διαθέσιμο σε περίπτωση εξαρτήσεων δεδομένων. Ο δεύτερος και τρίτος πολυπλέκτης 2 σε 1 χρησιμεύουν για την καθυστέρηση ενός κύκλου των εντολών στην περίπτωση επιτυχούς διακλάδωσης, ώστε να γίνει γνωστή η διεύθυνση της διακλάδωσης και στην κωδικοποίηση των εξωτερικών εξαιρέσεων. Ο πέμπτος πολυπλέκτης 2 σε 1 χρησιμεύει για τη διακλάδωση της εκτέλεσης του προγράμματος στη διεύθυνση του διαχειριστή εξαιρέσεων (Interrupt handler) όταν έχουμε τις εντολές break ή syscall.

Η μονάδα PC_ctrl έχει σαν εισόδους τα 32 bits του μετρητή προγράμματος τα οποία προέρχονται από την έξοδο του αθροιστή συν τέσσερα, την εξωτερική διακοπή, τις διακοπές που προέρχονται από το TLB και ένα σήμα ενός bit Internal_int που προέρχεται από την μονάδα pipe_control του σταδίου pipe_decstage. Το σήμα Internal_int είναι ένα όταν έχουμε μία από τις εντολές break ή syscall.Ο σκοπός της μονάδας PC_ctrl είναι να ελέγχει αν έχουμε κάποια εξαίρεση (exception) και να ενημερώνει κατάλληλα τον μετρητή προγράμματος ώστε η ροή του προγράμματος να μεταφερθεί στην διεύθυνση του διαχειριστή εξαιρέσεων (Exception Handler).

Η μνήμη που χρησιμοποιήσαμε για την αποθήκευση των εντολών είναι μία ROM των 1024 θέσεων των 32 bits η κάθε μία. Σαν είσοδο η μνήμη παίρνει 10 bits από τα 32 του μετρητή προγράμματος τα οποία προέρχονται από την έξοδο του πέμπτου πολυπλέκτη 2 σε 1 και συγκεκριμένα το $11^{ο}$ έως το $2^{ο}$ bit αυτό γιατί οι διευθύνσεις μνήμης είναι διευθύνσεις byte. Τη μνήμη τη δημιουργήσαμε με τη βοήθεια του Core Generator εργαλείου της Xilinx

ISE 7.1. Τα αποτελέσματα που προκύπτουν σε αυτό το κύκλο τοποθετούνται στους αντίστοιχους προσωρινούς καταχωρητές για χρήση στους επόμενους κύκλους.



Σχήμα 3.2: Βαθμίδα ανάκλησης εντολής.

3.2.2 Κύκλος δεύτερος: Αποκωδικοποίηση εντολής και ανάγνωση καταχωρητών (Instruction Decode)

Η μονάδα Pbypass έχει σαν εισόδους τα 32 bits της εντολής που βρίσκεται σε αυτό το στάδιο, τις 4 διευθύνσεις προορισμού, τα 4 σήματα για κάθε ειδικό καταχωρητή που δείχνουν αν έχουμε εγγραφή αυτών των καταχωρητών και 4 σήματα που δείχνουν αν έχουμε φορτώσεις δεδομένων από την μνήμη. Τα σήματα αυτά προέρχονται από τις τέσσερις προηγούμενες εντολές. Ο σκοπός της μονάδας Pbypass ελέγχει αν έχουμε εξαρτήσεις δεδομένων και στέλνει τα κατάλληλα σήματα στους πολυπλέκτες ώστε να έχουμε τα σωστά δεδομένα τη στιγμή που ζητούνται.

Αποκωδικοποίηση της εντολής στη μονάδα ελέγχου Pipe_control και ενεργοποίηση των κατάλληλων σημάτων ανάλογα με την εντολή. Η μονάδα Pipe_control έχει σαν εισόδους την 32 bits της εντολής που βρίσκεται σε αυτό το στάδιο, την εξωτερική διακοπή και τις διακοπές που προέρχονται από το TLB. Ανάγνωση των καταχωρητών πηγής από το αρχείο καταχωρητών ή από τους ειδικούς καταχωρητές (HI, LO, STATUS, CAUSE, EPC). Το αρχείο καταχωρητών περιέχει 32 καταχωρητές των 32 bits ο καθένας.

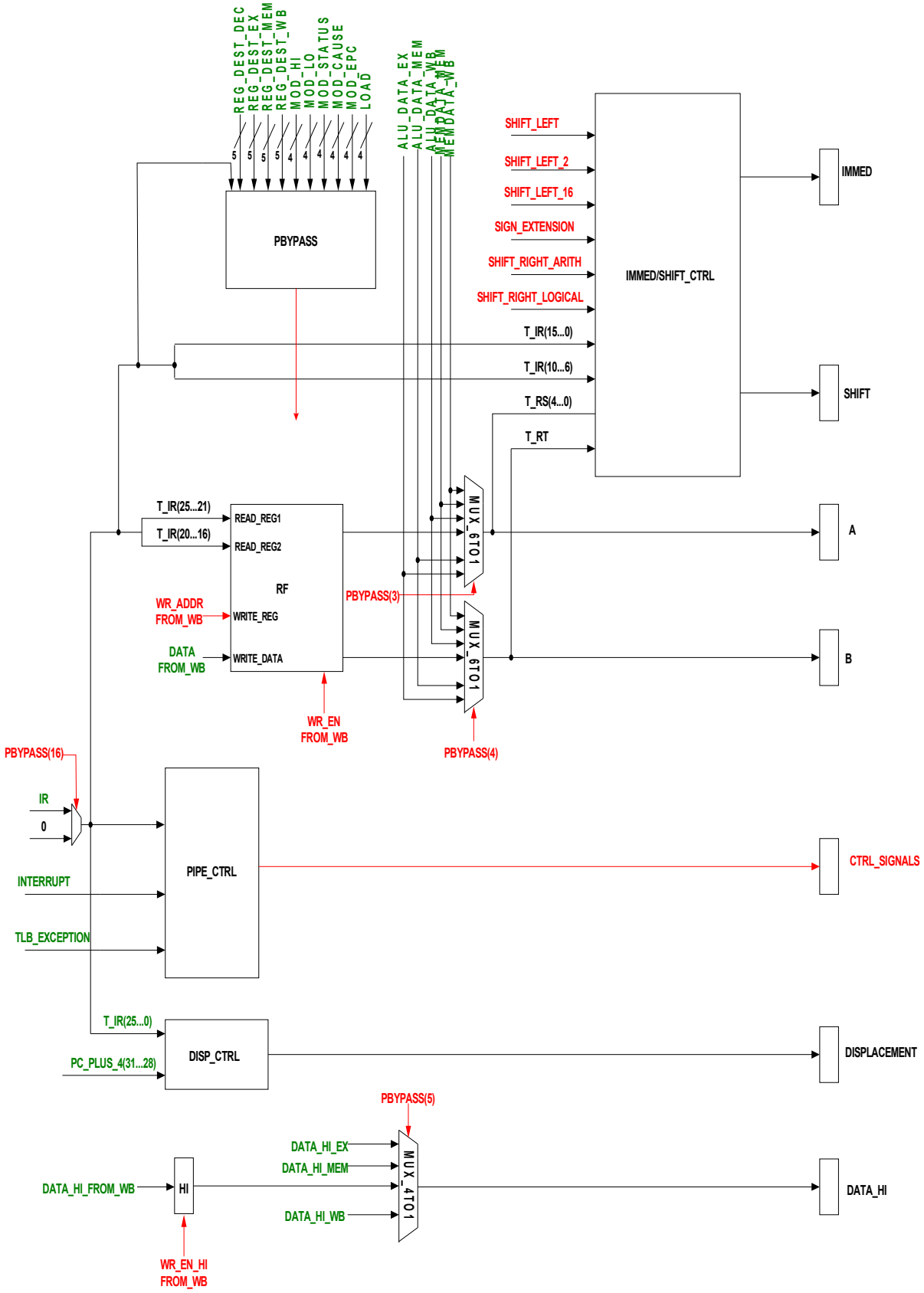
Το αρχείο καταχωρητών έχει σαν εισόδους τα 5 bits για κάθε διεύθυνση των καταχωρητών ανάγνωσης, τα 5 bits της διεύθυνσης του καταχωρητή προορισμού, τα 32 bits των δεδομένων που επρόκειτο να γραφτούν στον καταχωρητή προορισμού και το bit που κάνει δυνατή την εγγραφή στον καταχωρητή προορισμού. Παρακάτω περιγράφουμε ακριβώς πως υλοποιήσαμε το αρχείο καταχωρητών (βλέπε ενότητα 3.2.2.1).

Η μονάδα immed_control έχει σαν εισόδους τα 32 bits των δεδομένων του καταχωρητή ανάγνωσης rt, τα 5 bits της εντολής και τα bits από το $10^{ο}$ εώς το $6^{ο}$ bit, τα 5 πρώτα bits των δεδομένων του καταχωρητή rs και τα σήματα Shift_left_2, Shift_left_16, Shift_left, Shift_Right_Arith, Shift_Right_Logical και Sign_extension. Η μονάδα immed_control παράγει την επέκταση προσήμου (Sign Extension) του πεδίου άμεσης σταθερής της εντολής καθώς και την παραγωγή του αποτελέσματος ολίσθησης για τις αντίστοιχες εντολές ολίσθησης.

Η μονάδα disp_control παίρνει σαν εισόδους τα 26 πρώτα bits από τα 32 bits της εντολής και 4 πιο σημαντικά bits του μετρητή προγράμματος αυξημένος κατά 4 και σαν έξοδο παράγει την διεύθυνση προορισμού διακλάδωσης για ορισμένες εντολές διακλάδωσης.

Επίσης στο στάδιο αυτό έχουμε υλοποιήσει το συνεπεξεργαστή μηδέν. Ο συνεπεξεργαστής αυτός αποτελείται από τρεις καταχωρητές τους Cause register, Status register και EPC. Οι καταχωρητές αυτοί κρατάνε πληροφορίες για την κατάσταση του επεξεργαστή και τον τύπο της εξαίρεσης. Αναλυτικά τους καταχωρητές αυτούς τους περιγράφουμε στην ενότητα 3.5.3. Η επικοινωνία του συνεπεξεργαστή 0 με τον επεξεργαστή γίνεται με τις εντολές mfc0 (move from coprocessor 0) και mtc0 (move to coprocessor 0).

Οι πολυπλέκτες που υπάρχουν σε αυτό το στάδιο χρησιμεύουν για την προώθηση των δεδομένων από τα στάδια Alustage, Memstage και WBstage την κατάλληλη χρονική στιγμή. Τα αποτελέσματα που προκύπτουν σε αυτό το κύκλο τοποθετούνται στους αντίστοιχους προσωρινούς καταχωρητές για χρήση στους επόμενους κύκλους.



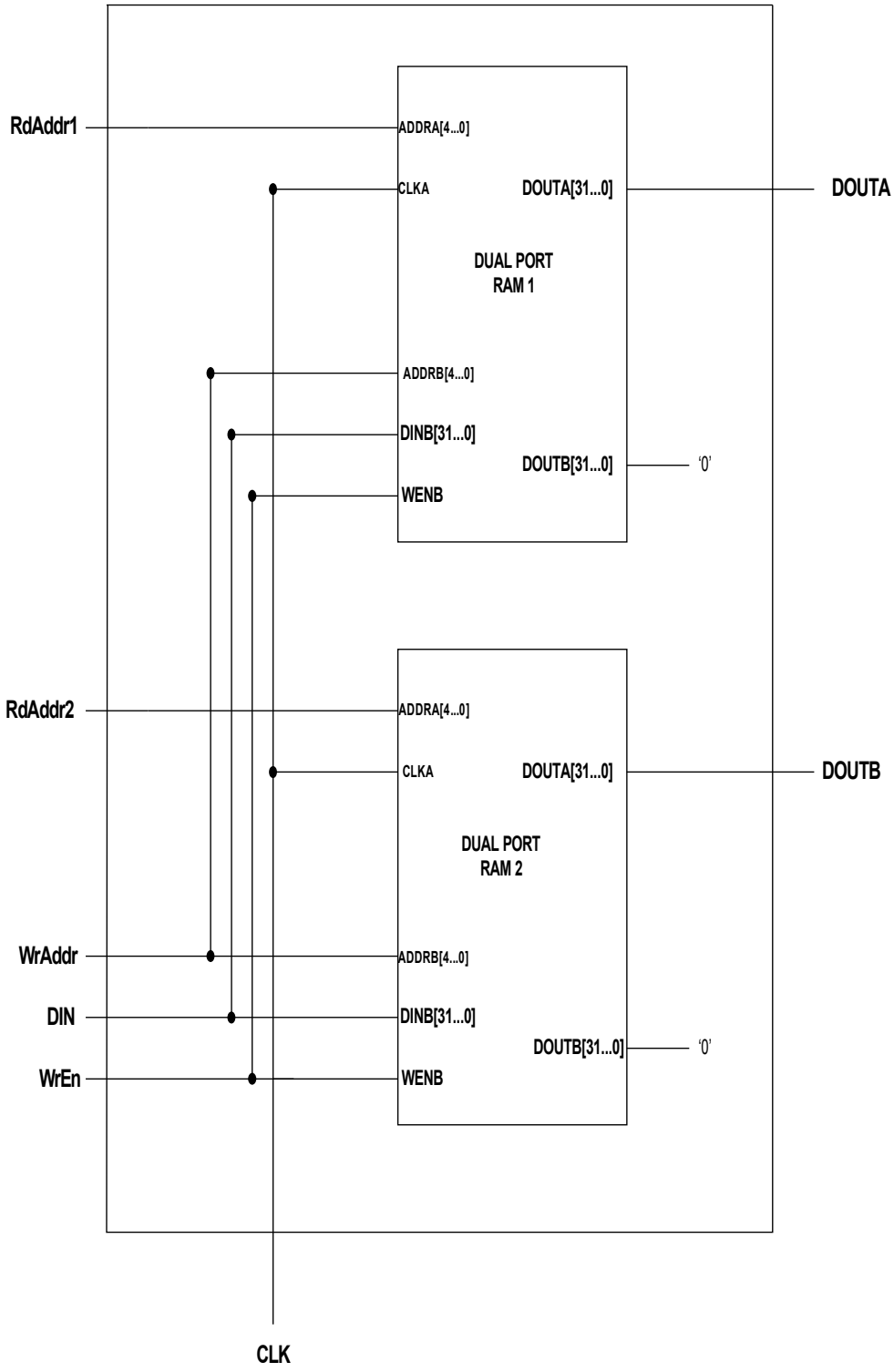
Σχήμα 3.3:Βαθμίδα αποκωδικοποίησης εντολής και ανάγνωσης καταχωρητών.

3.2.2.1 Υλοποίηση αρχείου καταχωρητών

Το αρχείο καταχωρητών αποτελείται από 32 καταχωρητές των 32 bits ο καθένας, δύο θύρες ανάγνωσης και μία θύρα εγγραφής. Η ανάγνωση των δεδομένων από το αρχείο καταχωρητών γίνεται ασύγχρονα και δεν υπάρχει κάποιο σήμα ενεργοποίησης της ανάγνωσης πράγμα που σημαίνει ότι το αρχείο καταχωρητών διαβάζει πάντα από τις δύο θέσεις που υποδεικνύουν οι διευθύνσεις ανάγνωσης. Η εγγραφή των δεδομένων στο αρχείο καταχωρητών γίνεται σύγχρονα όταν το σήμα εγγραφής είναι ενεργοποιημένο. Για την υλοποίηση του χρησιμοποιήθηκαν δύο δίπορτες (dual port) μνήμες 32 θέσεων των 32 bit. Οι μνήμες αυτές δημιουργήθηκαν από τον core generator του ISE 7.1.

Η κάθε μια δίπορτη μνήμη έχει τις εξής εισόδους μια θύρα των 5 bits που υποδεικνύουν την διεύθυνση ανάγνωσης, μια θύρα των 5 bits που υποδεικνύουν την διεύθυνση εγγραφής, μια θύρα των 32 bits που υποδεικνύουν τα δεδομένα που επρόκειτο να εγγραφούν στο αρχείο καταχωρητών, μια θύρα 1 bit που αντιστοιχεί στο ρολόι και μια θύρα 1 bit που αντιστοιχεί στο σήμα ενεργοποίησης της εγγραφής.

Για την επίτευξη του αρχείου καταχωρητών με δύο θύρες ανάγνωσης και μια θύρα εγγραφής χρησιμοποιήσαμε δύο αντίγραφα της δίπορτης μνήμης που περιγράψαμε πιο πάνω. Η πρώτη θύρα κάθε μίας από τις δίπορτες μνήμες χρησιμοποιείται για ανάγνωση μόνο. Έτσι δημιουργούνται οι δύο θύρες ανάγνωσης του αρχείου καταχωρητών. Για την επίτευξη μιας διεύθυνσης εγγραφής, βραχυκυκλώνουμε τις δεύτερες θύρες της κάθε δίπορτης μνήμης στη διεύθυνση εγγραφής δεδομένων. Επίσης βραχυκυκλώνουμε τα δεδομένα εισόδου και τα ρολόγια των μνημών. Η συνδεσμολογία που περιγράψαμε πιο πάνω φαίνεται στο παρακάτω σχήμα.



Σχήμα 3.4: Το αρχείο καταχωρητών με δύο θύρες ανάγνωσης και μια θύρα εγγραφής.

3.2.3 Κύκλος τρίτος: Εκτέλεση αριθμητικών και λογικών πράξεων/ Υπολογισμός διεύθυνσης μνήμης (EX)

Η μονάδα αριθμητικών και λογικών πράξεων (ALU) υπολογίζει ανάλογα με την εντολή τα εξής:

- Υπολογίζει την διεύθυνση για την πρόσβαση στη μνήμη προσθέτοντας τον καταχωρητή βάσης με την σταθερή μετατόπισης (offset).
- Εκτελεί την αριθμητική/ λογική πράξη, που ορίζεται στο πεδίο ορισμού συνάρτησης της εντολής, μεταξύ των δύο καταχωρητών που διαβάστηκαν στο προηγούμενο κύκλο.
- Εκτελεί την αριθμητική/ λογική πράξη, που ορίζεται στον κωδικό εντολής μεταξύ του καταχωρητή A και της σταθερής μετά την επέκταση προσήμου.
- Υπολογίζει την διεύθυνση προορισμού της διακλάδωσης προσθέτοντας το μετρητή προγράμματος αυξημένο κατά τέσσερα στη επεκταμένη σταθερή ολισθημένη προς τα αριστερά κατά δύο.

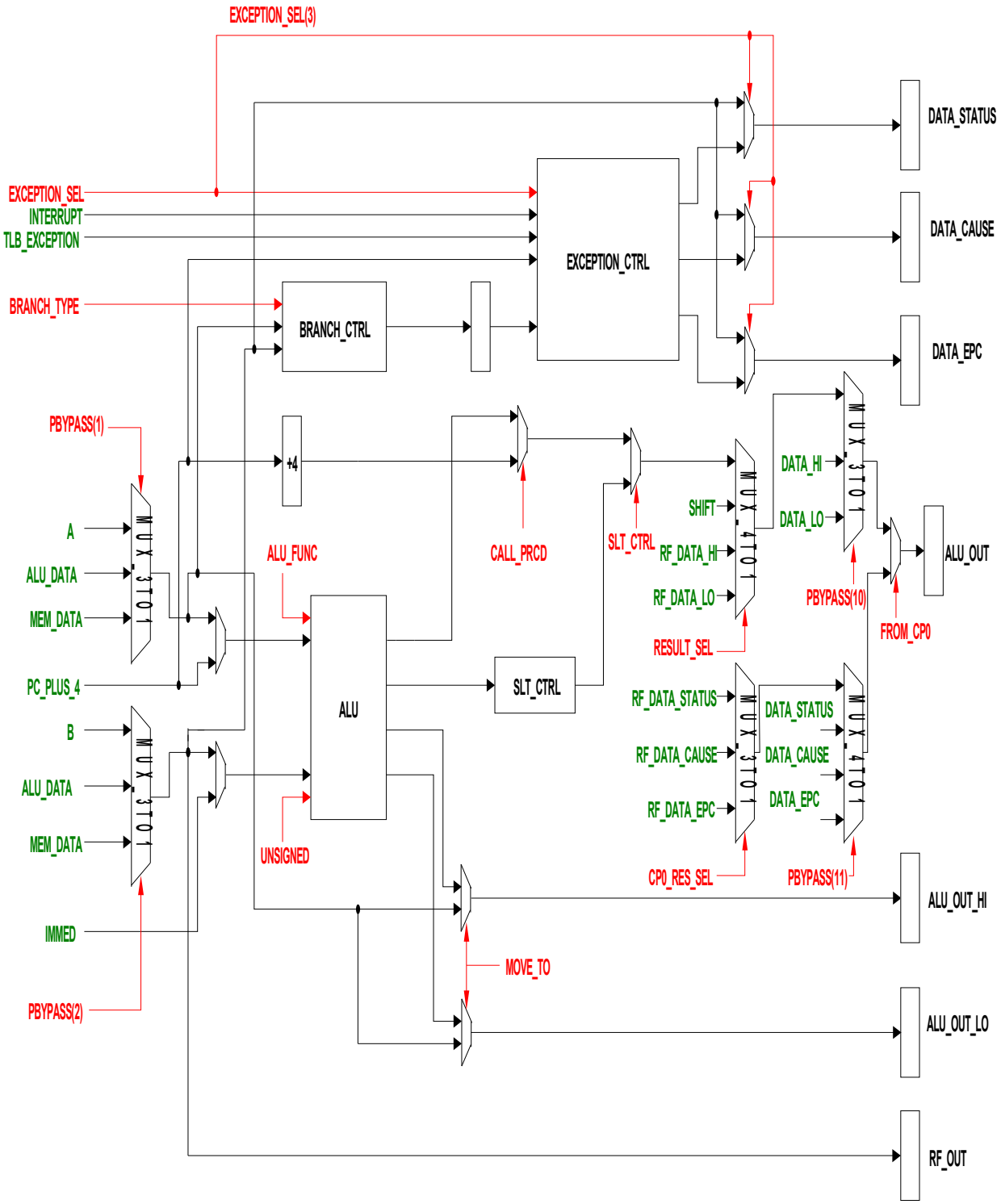
Η μονάδα Branch_control έχει σαν εισόδους τα 4 bits του BranchType, τα 32 bits του καταχωρητή rs ή των προωθημένων δεδομένων από την μνήμη και τα 32 bits του καταχωρητή rt ή των προωθημένων δεδομένων από την μνήμη. Ο σκοπός της μονάδας είναι να ελέγχει αν η συνθήκη της διακλάδωσης είναι επιτυχής και ανάλογα με τον τύπο της διακλάδωσης επιλέγει τη σωστή διεύθυνση προορισμού της διακλάδωσης.

Η μονάδα exceptions_ctrl έχει σαν εισόδους τα 4 bits του exception_sel, τα 3 bits του TLB_exception, ένα bit του Interrupt και ένα bit του BranchDelay. Ο σκοπός της μονάδας είναι να ενημερώνει ανάλογα με τον τύπο της εξαίρεσης και αν αυτή συμβαίνει σε Branch delay slot τις τιμές των καταχωρητών Status,Cause και Epc.

Η μονάδα slt_control έχει σαν είσοδο ένα bit που προέρχεται από την ALU. Ο σκοπός της μονάδας είναι να ελέγχει αν η συνθήκη των εντολών slt, slti, sltu, sltiu είναι αληθής.

Οι πολυπλέκτες που ελέγχονται από τα σήματα PBYPASS(1), PBYPASS(2), PBYPASS(10) και PBYPASS(11) χρησιμοποιούνται για την σωστή προώθηση των

αποτελεσμάτων στο σημείο που τα χρειάζεται η υπό εκτέλεση εντολή. Τα αποτελέσματα που προκύπτουν σε αυτό το κύκλο τοποθετούνται στους αντίστοιχους προσωρινούς καταχωρητές για χρήση στους επόμενους κύκλους.



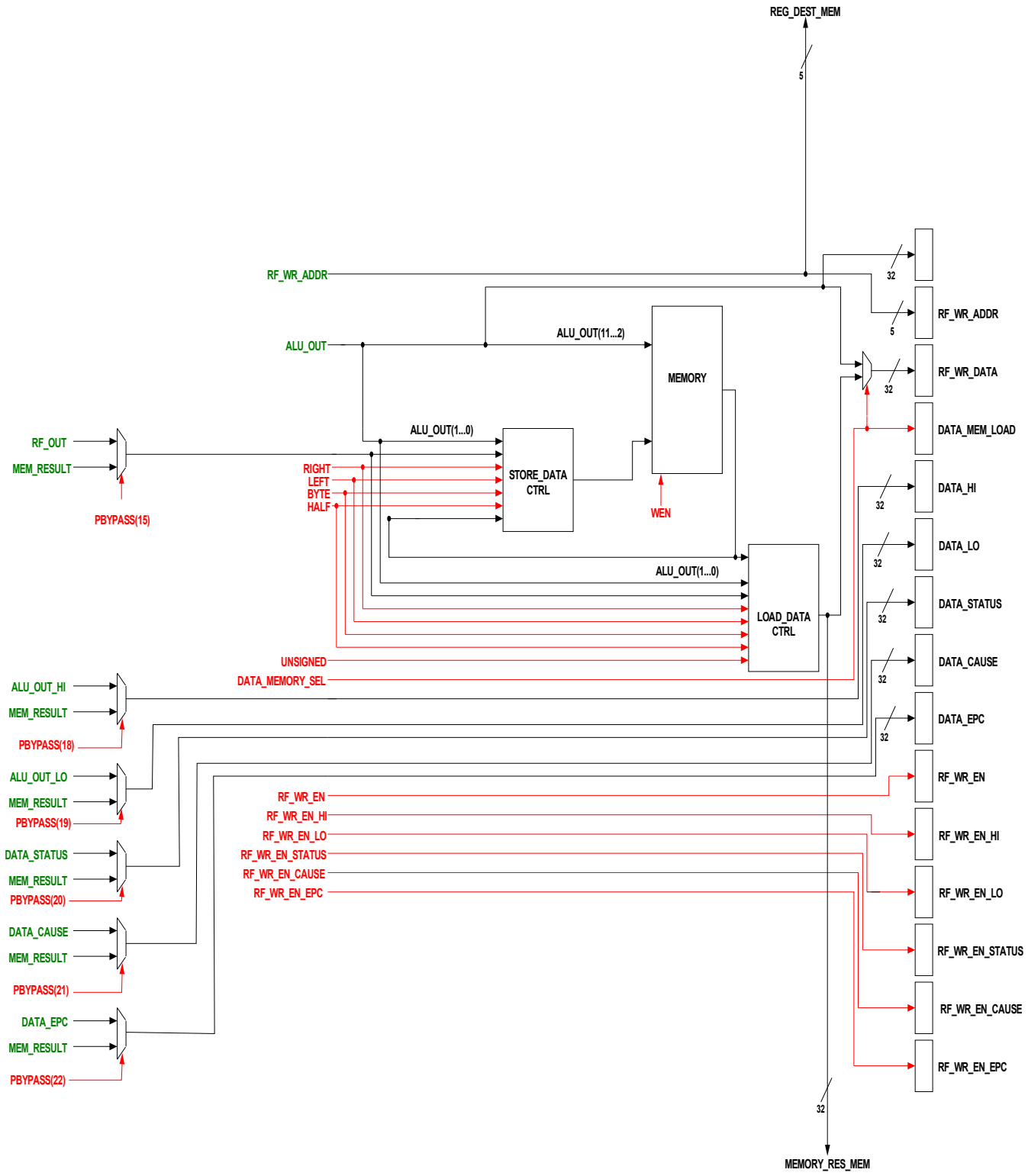
Σχήμα 3.5: Βαθμίδα εκτέλεσης αριθμητικών και λογικών πράξεων.

3.2.4 Κύκλος τέταρτος: Πρόσβαση στη μνήμη (MEM)

Η μνήμη που χρησιμοποιήσαμε σε αυτό το στάδιο είναι μία single port RAM των 1024 θέσεων των 32 bits η κάθε μία. Η διεύθυνση της μνήμης καθορίζεται από την είσοδο των 10 bits από τα 32 της εξόδου της ALU και συγκεκριμένα το 11^{ο} έως το 2^{ο} bit αυτό γιατί οι διευθύνσεις μνήμης είναι διευθύνσεις byte. Επιπλέον έχει σαν είσοδο τα 32 bits δεδομένων που προέρχονται από την μονάδα store_data_ctrl. Τέλος, σήματα εισόδου είναι το ρολόι και το WE. Όταν το WE είναι 1 επιτρέπεται η εγγραφή δεδομένων στην μνήμη. Τη μνήμη τη δημιουργήσαμε με τη βοήθεια του Core Generator εργαλείου της Xilinx ISE 7.1.

Η μονάδα store_data_ctrl έχει σαν εισόδους 2 bits από την έξοδο της ALU, 32 bits δεδομένων του καταχωρητή rt, 32 bits δεδομένων από την έξοδο της μνήμης καθώς και τα σήματα ενός bit το καθένα left, right, half και byte. Ο σκοπός της μονάδας είναι να δημιουργεί τα κατάλληλα δεδομένα που θέλει να αποθηκεύσει η εντολή αποθήκευσης ανάλογα με το είδος της εντολής αποθήκευσης (sw, sb κ.τ.λ.).

Η μονάδα data_mem_ctrl έχει σαν εισόδους 2 bits από την έξοδο της ALU, 32 bits δεδομένων του καταχωρητή rt, 32 bits δεδομένων από την έξοδο της μνήμης καθώς και τα σήματα ενός bit το καθένα left, right, half, byte και unsigned. Ο σκοπός της μονάδας είναι να δημιουργεί τα κατάλληλα δεδομένα που θέλει να φορτώσει η εντολή φόρτωσης στο αρχείο καταχωρητών ανάλογα με το είδος της εντολής φόρτωσης (lw, lb κ.τ.λ.). Τα αποτελέσματα που που προκύπτουν σε αυτό το κύκλο τοποθετούνται στους αντίστοιχους προσωρινούς καταχωρητές για χρήση στους επόμενους κύκλους.



Σχήμα 3.6: Βαθμίδα πρόσβασης στην μνήμη.

3.2.5 Κύκλος πέμπτος: Εγγραφή αποτελέσματος (WB)

Εγγραφή του αποτελέσματος στο αρχείο καταχωρητών ή στους ειδικούς καταχωρητές ανάλογα με το τύπο της εντολής.

3.3 ΥΛΟΠΟΙΗΣΗ ΜΟΝΑΔΑΣ ΕΛΕΓΧΟΥ

Η μονάδα ελέγχου είναι υλοποιημένη σε πολλαπλά στάδια. Στο στάδιο αποκωδικοποίησης της εντολής (decode stage) υπάρχει το κύκλωμα PIPE_CTRL. Η μονάδα PIPE_CTRL έχει σαν εισόδους την 32 bits της εντολής που βρίσκεται σε αυτό το στάδιο, την εξωτερική διακοπή και τις διακοπές που προέρχονται από το TLB. Σκοπός της μονάδας είναι η αποκωδικοποίηση της εντολής και ενεργοποίηση των κατάλληλων σημάτων ανάλογα με την εντολή όπως το σήμα εγγραφής στο αρχείο καταχωρητών, το σήμα εγγραφής στους ειδικούς καταχωρητές, το σήμα εγγραφής στη μνήμη δεδομένων και τα σήματα ελέγχου διάφορων πολυπλεκτών.

Στο ίδιο στάδιο υπάρχει και το κύκλωμα PBYPASS το οποίο έχει σαν σκοπό να ελέγχει αν έχουμε εξαρτήσεις δεδομένων και να στέλνει τα κατάλληλα σήματα στους πολυπλέκτες ώστε να έχουμε τα σωστά δεδομένα τη στιγμή που ζητούνται. Επίσης στο ίδιο κύκλωμα υλοποιείται η μονάδα ανίχνευσης περιορισμών δεδομένων (hazard detection unit), η οποία παράγει τα κατάλληλα σήματα στους πολυπλέκτες ώστε να καθυστερήσει η υπό εκτέλεση εντολή τόσους κύκλους μέχρι να εξαλειφθεί η εξάρτηση δεδομένων επειδή δεν επιλύεται με τη τεχνική της προώθησης δεδομένων.

Επίσης, υπάρχει η μονάδα hit_unit η οποία έχει ως σκοπό να παγώνει την αλυσίδα της ομοχειρίας μέχρις ότου η μνήμη να απαντήσει τα σωστά δεδομένα. Εμείς εδώ έχουμε υλοποιήσει η μνήμη να απαντάει πάντα σε δύο κύκλους.

Στο στάδιο εκτέλεσης αριθμητικών και λογικών πράξεων υπάρχει η μονάδα BRANCH_CTRL.Ο σκοπός της μονάδας είναι να ελέγχει αν η συνθήκη της διακλάδωσης

είναι επιτυχής και ανάλογα με τον τύπο της διακλάδωσης να επιλέγει τη σωστή διεύθυνση προορισμού της διακλάδωσης.

Επίσης στο ίδιο στάδιο υπάρχει η μονάδα EXCEPTION_CTRL. Ο σκοπός της μονάδας είναι να ενημερώνει ανάλογα με τον τύπο της εξαίρεσης και αν αυτή συμβαίνει σε Branch delay slot τις τιμές των καταχωρητών Status,Cause και Epc. Επίσης υπάρχει η μονάδα slt_control η οποία ελέγχει αν η συνθήκη των εντολών slt, slti, sltu, sltiu είναι αληθής.

3.4 ΚΙΝΔΥΝΟΙ ΟΜΟΧΕΙΡΙΑΣ

3.4.1 Εισαγωγή

Οι καταστάσεις που εμποδίζουν την επόμενη εντολή να εκτελεστεί στο προδιαγεγραμμένο κύκλο ονομάζονται κίνδυνοι (hazards). Υπάρχουν τρεις κατηγορίες κινδύνων:

- Δομικοί Κίνδυνοι (Structural Hazards) προέρχονται όταν κάποιος συνδυασμός επικαλυμμένης εκτέλεσης εντολών απαιτεί την ταυτόχρονη χρήση του ίδιου πόρου.
- Κίνδυνοι Δεδομένων (Data Hazards) όταν μια εντολή για την εκτέλεσή της απαιτεί το αποτέλεσμα μιας πρότερης εντολής.
- Κίνδυνοι Ελέγχου (Control Hazards) προέρχονται από την ομόχειρη εκτέλεση των διακλαδώσεων.

Εμείς ασχοληθήκαμε με τους κινδύνους δεδομένων και τους κινδύνους ελέγχου. Για την αποφυγή των κινδύνων απαιτείται η προώθηση κάποιων δεδομένων ή η καθυστέρηση ορισμένων εντολών.

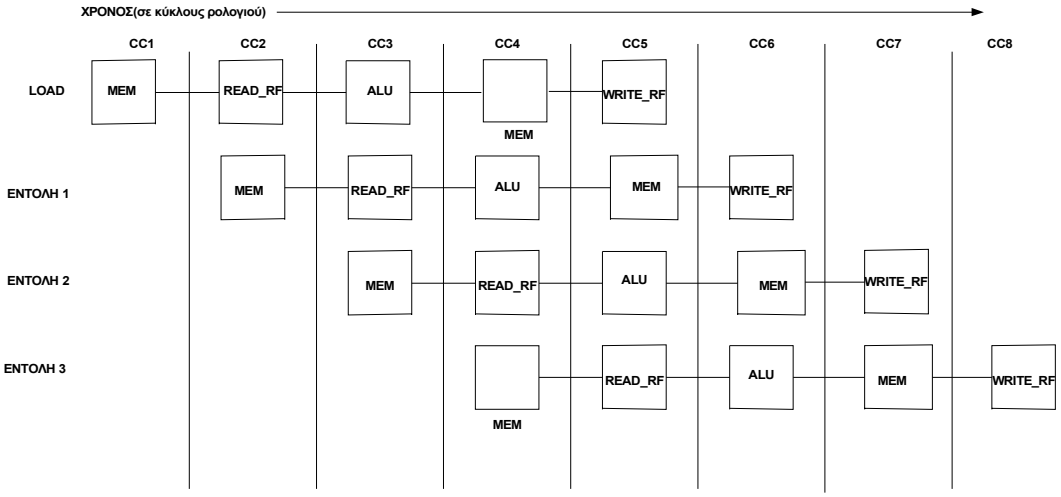
3.4.2 Δομικοί κίνδυνοι

Στον ομόχειρο επεξεργαστή για να υποστηριχθούν όλοι οι πιθανοί συνδυασμοί εντολών απαιτείται η ύπαρξη πολλαπλών αντιγράφων των πόρων και η ομοχειρία των λειτουργικών μονάδων. Εάν κάποιοι συνδυασμοί δεν υποστηρίζονται λόγω σύγκρουσης πόρων, τότε λέμε ότι ο επεξεργαστής έχει δομικό κίνδυνο (structural hazard).

Οι πιο συνηθισμένες περιπτώσεις δομικών κινδύνων προέρχονται από το γεγονός ότι κάποιες λειτουργικές μονάδες δεν είναι πλήρως ομόχειρες. Άλλη μια συνηθισμένη περίπτωση δομικού κινδύνου είναι όταν κάποιος πόρος δεν υπάρχει σε πολλά αντίγραφα ώστε να υποστηρίζει όλους τους δυνατούς συνδυασμούς εντολών στην αλυσίδα ομοχειρίας.

Όταν σε μια ακολουθία εντολών παρουσιαστεί κάποιος δομικός κίνδυνος, τότε έχουμε ανάσχεση της ομόχειρης εκτέλεσης ώστε η απαιτούμενη μονάδα να είναι διαθέσιμη. Τέτοιες ανασχέσεις αυξάνουν το CPI κατά ένα.

Μερικοί ομόχειροι επεξεργαστές μοιράζονται την ίδια μνήμη τόσο για δεδομένα όσο και για εντολές. Αυτό έχει ως αποτέλεσμα την σύγκρουση της εντολής που απαιτεί πρόσβαση στην μνήμη με την ανάκληση μιας επόμενης εντολής όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 3.7: Σύγκρουση κατά την πρόσβαση στην μονάδα της μνήμης

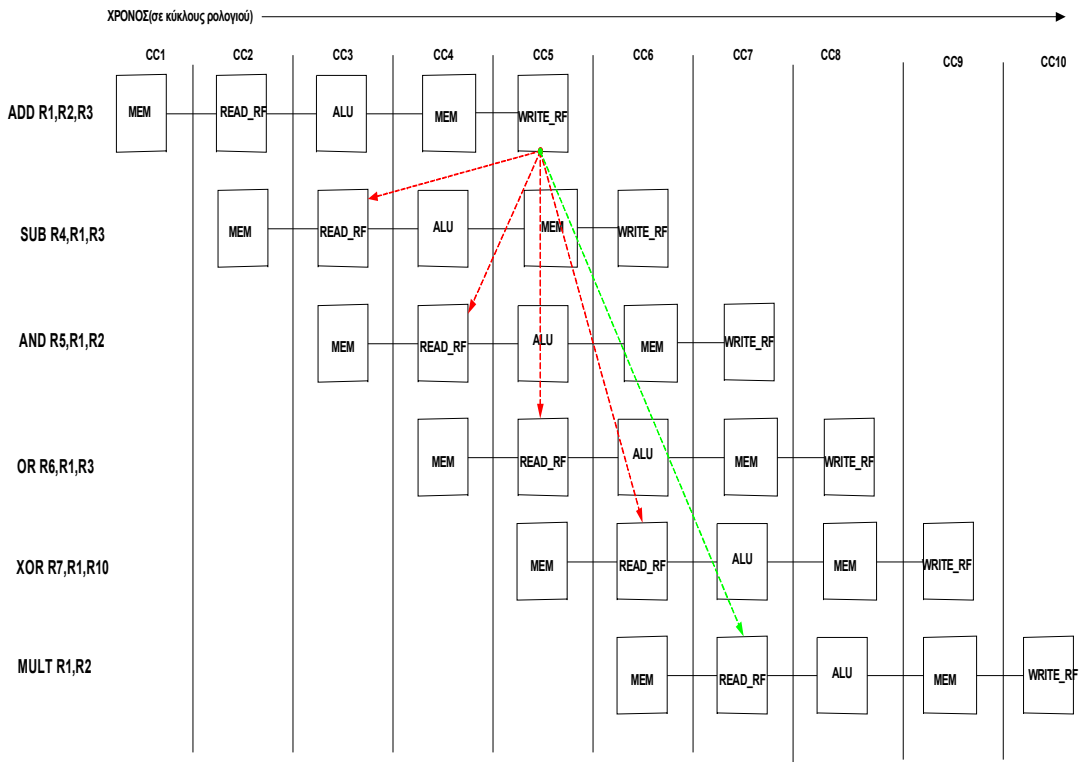
Για την αποφυγή του κινδύνου σταματάμε την ομοχειρία κατά 1 κύκλο κατά τη διάρκεια διεξαγωγής της πρόσβασης δεδομένων. Η ανάσχεση αυτή ονομάζεται φυσαλίδα καθώς διασχίζει την αλυσίδα της ομοχειρίας καταλαμβάνοντας χώρο αλλά μη προσφέροντας χρήσιμη εργασία.

3.4.3 Κίνδυνοι δεδομένων

Οι κίνδυνοι δεδομένων εμφανίζονται διότι η ομόχειρη εκτέλεση αλλάζει τη σειρά προσβάσεων ανάγνωσης-εγγραφής τελεσταίων με αποτέλεσμα να είναι διαφορετική από την ακολουθιακή εκτέλεση σε μία μη ομόχειρη εκτέλεση. Ένα παράδειγμα ομόχειρης εκτέλεσης είναι το παρακάτω:

```
add r1, r2, r3
sub r4, r1,r3
And r5, r1, r2
Or r6, r1, r3
Xor r7, r1, r10
Mult r1,r2
```

Στο παράδειγμα παρατηρούμε ότι όλες οι εντολές που ακολουθούν την add χρησιμοποιούν το αποτέλεσμα r1. Η εντολή add γράφει το αποτέλεσμα στο αρχείο καταχωρητών στον πέμπτο κύκλο, δηλαδή στη βαθμίδα WB, όμως η εντολή sub διαβάζει τον r1 στο δεύτερο κύκλο, στη βαθμίδα ID. Επομένως, η εντολή sub θα διαβάσει λάθος τιμή για τον r1. Το φαινόμενο αυτό ονομάζεται κίνδυνος δεδομένων. Το ίδιο συμβαίνει και για τις εντολές που ακολουθούν μέχρι και τη xor. Η mult διαβάζει τη σωστή τιμή του r1 γιατί η add έχει προλάβει να γράψει το αποτέλεσμα στον r1. Τους ίδιους κινδύνους δεδομένων έχουμε και για τους ειδικούς καταχωρητές HI, LO, CAUSE, STATUS, EPC.



Σχήμα 3.8: Η χρήση του αποτελέσματος της ADD από τις επόμενες 4 εντολές προκαλεί κίνδυνο

3.4.3.1 Ελαχιστοποίηση των κινδύνων δεδομένων με προώθηση δεδομένων

Για την αντιμετώπιση του προβλήματος χρησιμοποιήσαμε την κυκλωματική τεχνική της προώθησης (bypassing). Το κύκλωμα που υλοποιεί αυτή τη τεχνική είναι η μονάδα PBYPASS. Η λειτουργία της τεχνικής αυτής στο συγκεκριμένο παράδειγμα είναι η εξής:

- Το αποτέλεσμα της ALU προωθείται από τους καταχωρητές ομοχειρίας EX/MEM, MEM/WB και WB/DEC στις εισόδους των βαθμίδων DECSTAGE και ALUSTAGE.
- Η μονάδα PBYPASS ελέγχει αν έχουμε εξάρτηση δεδομένων και ενεργοποιεί τα κατάλληλα σήματα ελέγχου των πολυπλεκτών ώστε η εντολή

να διαβάσει το προωθημένο αποτέλεσμα αντί της τιμής που διαβάστηκε από το αρχείο καταχωρητών.

Στον επεξεργαστή μας κάνουμε προώθηση αποτελεσμάτων όχι μόνο από την αμέσως προηγούμενη εντολή αλλά και από εντολή η οποία ξεκίνησε τέσσερις κύκλους νωρίτερα. Πιο συγκεκριμένα οι περιπτώσεις που έχουμε εξαρτήσεις δεδομένων είναι οι παρακάτω:

- Για R-type εντολές στην αποκωδικοποίηση και το ζητούμενο αποτέλεσμα από την ALU.

  DEC/EX. IR[rd]= =IF/DEC. IR[rs] OR

  DEC/EX. IR[rd]= =IF/DEC. IR[rt] OR

  EX/MEM. IR[rd]= =IF/DEC. IR[rs] OR

  EX/MEM. IR[rd]= =IF/DEC. IR[rt] OR

  MEM/WB. IR[rd]= =IF/DEC. IR[rs] OR

  MEM/WB. IR[rd]= =IF/DEC. IR[rt] OR

  WB/DEC. IR[rd]= =IF/DEC. IR[rs] OR

  WB/DEC. IR[rd]= =IF/DEC. IR[rt]

  Σε όλες τις παραπάνω περιπτώσεις είναι δυνατή η προώθηση των δεδομένων.

- Για I-type εντολές στην αποκωδικοποίηση και το ζητούμενο αποτέλεσμα από την ALU.

  DEC/EX. IR[rd]= =IF/DEC. IR[rs] OR

  EX/MEM. IR[rd]= =IF/DEC. IR[rs] OR

  MEM/WB. IR[rd]= =IF/DEC. IR[rs] OR

  WB/DEC. IR[rd]= =IF/DEC. IR[rs]

  Σε όλες τις παραπάνω περιπτώσεις είναι δυνατή η προώθηση των δεδομένων.

- Για R-type εντολές στην αποκωδικοποίηση και το ζητούμενο αποτέλεσμα από την μνήμη.

  DEC/EX. IR[rt]= =IF/DEC. IR[rs] OR

  DEC/EX. IR[rt]= =IF/DEC. IR[rt] OR

  EX/MEM. IR[rt]= =IF/DEC. IR[rs] OR

  EX/MEM. IR[rt]= =IF/DEC. IR[rt] OR

  MEM/WB. IR[rt]= =IF/DEC. IR[rs] OR

  MEM/WB. IR[rt]= =IF/DEC. IR[rt] OR

  WB/DEC. IR[rt]= =IF/DEC. IR[rs] OR

  WB/DEC. IR[rt]= =IF/DEC. IR[rt]

  Μόνο στις έξι τελευταίες περιπτώσεις είναι δυνατή η προώθηση των δεδομένων.

- Για I-type εντολές, εκτός τις εντολές αποθηκεύσεων, στην αποκωδικοποίηση και το ζητούμενο αποτέλεσμα προέρχεται από την μνήμη.

  DEC/EX. IR[rt]= =IF/DEC. IR[rs] OR

  EX/MEM. IR[rt]= =IF/DEC. IR[rs] OR

  MEM/WB. IR[rt]= =IF/DEC. IR[rs] OR

  WB/DEC. IR[rt]= =IF/DEC. IR[rs]

  Μόνο στις τρεις τελευταίες περιπτώσεις είναι δυνατή η προώθηση των δεδομένων.

  Για τις εντολές αποθηκεύσεων έχουμε:

  DEC/EX. IR[rt]= =IF/DEC. IR[rt] OR

  EX/MEM. IR[rt]= =IF/DEC. IR[rt] OR

  MEM/WB. IR[rt]= =IF/DEC. IR[rt] OR

  WB/DEC. IR[rt]= =IF/DEC. IR[rt]

  Σε όλες τις περιπτώσεις είναι δυνατή η προώθηση των δεδομένων.

- Επίσης εξαρτήσεις δεδομένων έχουμε και για τους ειδικούς καταχωρητές όταν συμβαίνουν τα παρακάτω, π.χ. για HI register:

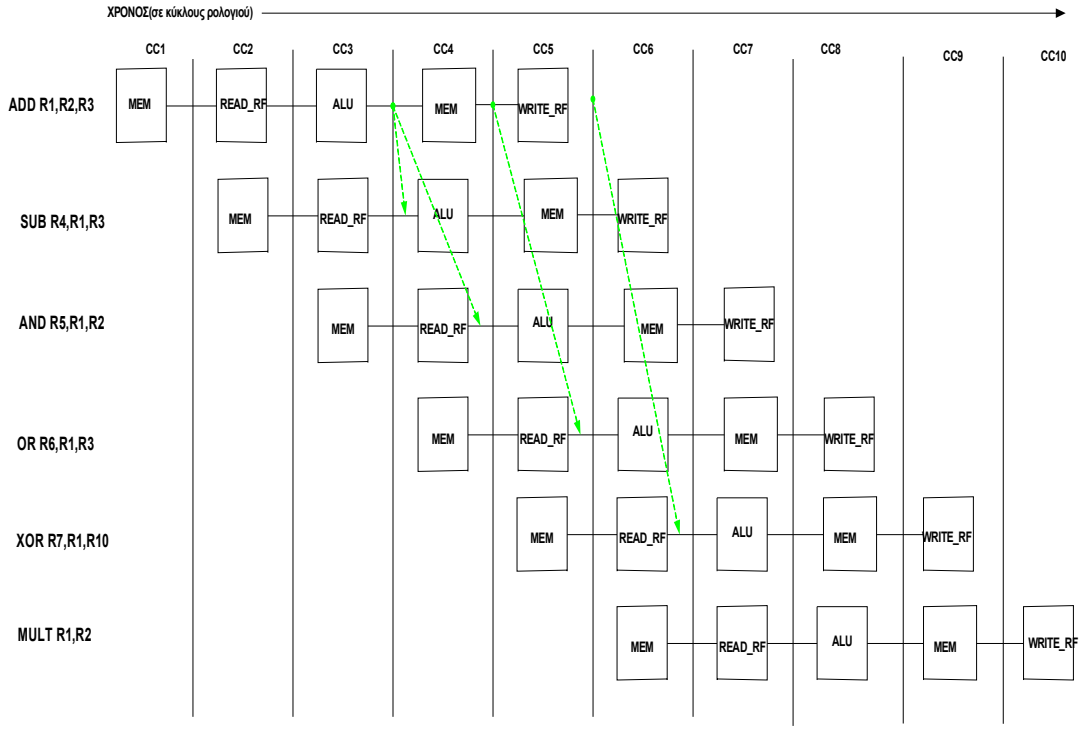
  DEC/EX. IR[hi]= =IF/DEC. IR[hi] OR

  EX/MEM. IR[hi]= =IF/DEC. IR[hi] OR

  MEM/WB. IR[hi]= =IF/DEC. IR[hi] OR

  WB/DEC. IR[hi]= =IF/DEC. IR[hi]

  Σε όλες τις παραπάνω περιπτώσεις είναι δυνατή η προώθηση των δεδομένων.



Σχήμα 3.9: Χρήση μονοπατιών προώθησης για την αποφυγή του κινδύνου και της ανάσχεσης

3.4.3.2 Κίνδυνοι δεδομένων που προκαλούν ανάσχεση

Υπάρχουν όμως περιπτώσεις όπου η προώθηση δεν μπορεί να αντιμετωπίσει όλες τις περιπτώσεις κινδύνων δεδομένων. Ένα τέτοιο παράδειγμα είναι το παρακάτω:

Ld r1, 0(r2)

Sub r3, r1, r4

And r5, r1, r4

Or r7, r1, r6

Xor r8, r1, r10

Η εντολή Ld παράγει το αποτέλεσμα στο τέλος του τέταρτου κύκλου στη βαθμίδα MEMSTAGE, ενώ η εντολή sub θέλει το αποτέλεσμα στην αρχή του τέταρτου κύκλου στη βαθμίδα ALUSTAGE. Το αποτέλεσμα της εντολής Ld δεν μπορούμε να το προωθήσουμε με την τεχνική της προώθησης στο σωστό σημείο τη σωστή στιγμή. Για τις εντολές And, Or και Xor μπορούμε να προωθήσουμε το αποτέλεσμα της εντολής Ld. Για την αντιμετώπιση του κινδύνου που παρουσιάσαμε πιο πάνω, εισάγουμε λογική ανίχνευσης κινδύνων (pipeline interlock logic). Η λογική αυτή ανιχνεύει τους κινδύνους και εισάγει ανασχέσεις ομοχειρίας μέχρις ότου ο κίνδυνος να εξαλειφθεί. Στη συγκεκριμένη περίπτωση καθυστερούμε την εντολή sub και όσες ακολουθούν αυτήν κατά ένα κύκλο. Με τον τρόπο αυτό πετυχαίνουμε τη σωστή προώθηση δεδομένων στην εντολή sub στη βαθμίδα που τα χρειάζεται.

Πιο συγκεκριμένα οι περιπτώσεις που προκαλούν ανάσχεση είναι οι παρακάτω:

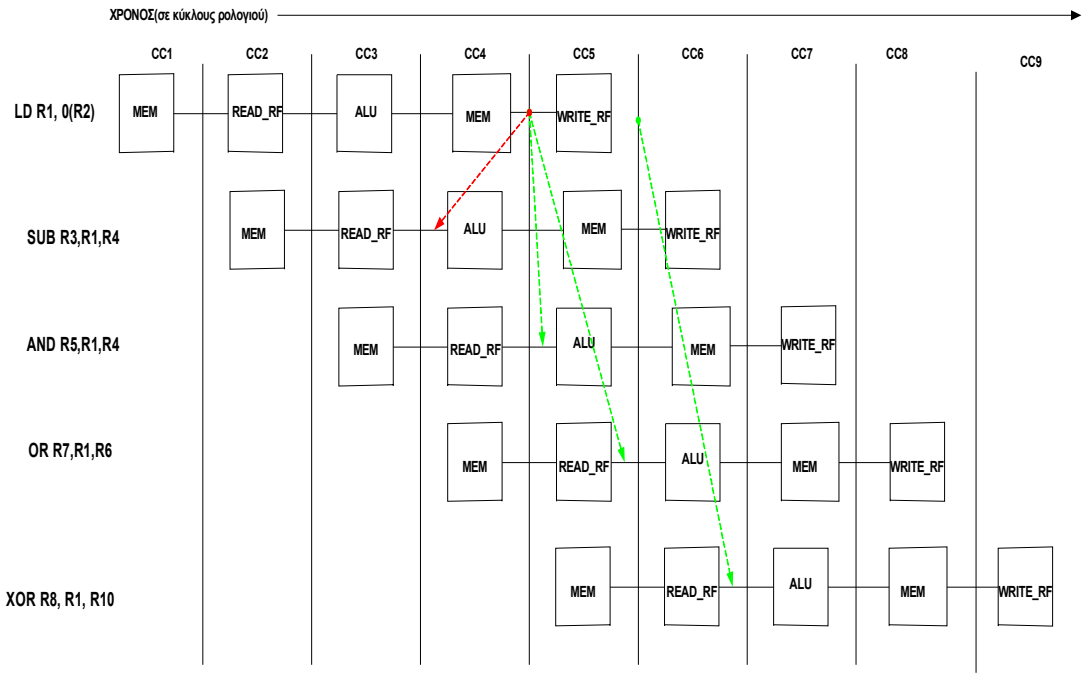
- Για R-type εντολές στην αποκωδικοποίηση και το ζητούμενο αποτέλεσμα από την μνήμη.

  DEC/EX. IR[rt]= =IF/DEC. IR[rs] OR

  DEC/EX. IR[rt]= =IF/DEC. IR[rt]

- Για I-type εντολές, εκτός τις εντολές αποθηκεύσεων, στην αποκωδικοποίηση και το ζητούμενο αποτέλεσμα προέρχεται από την μνήμη.

  DEC/EX. IR[rt]= =IF/DEC. IR[rs]



Σχήμα 3.10: Η εντολή load μπορεί να προωθήσει το αποτέλεσμα στις εντολές and, or και xor, αλλά όχι στη sub.

3.4.3.3 Κίνδυνοι ελέγχου

Οι κίνδυνοι ελέγχου προέρχονται από την ομόχειρη εκτέλεση διακλαδώσεων, οι οποίες αλλάζουν τη τιμή του μετρητή προγράμματος σε διαφορετική από την τρέχουσα τιμή συν τέσσερα. Στον επεξεργαστή μας έχουμε διακλαδώσεις που γίνονται μόνο αν ισχύει η συνθήκη διακλάδωσης και διακλαδώσεις που γίνονται χωρίς τον έλεγχο κάποιας συνθήκης. Στις πρώτες, όταν η συνθήκη είναι μη αληθής η ροή του προγράμματος εκτελείται κανονικά και δεν έχουμε καθυστερήσεις. Εάν όμως η συνθήκη είναι αληθής τότε η εντολή διακλάδωσης θα πρέπει να μεταβάλλει τον μετρητή προγράμματος στην διεύθυνση προορισμού. Επειδή όμως ο υπολογισμός της διεύθυνσης προορισμού γίνεται στην βαθμίδα ALUSTAGE δηλαδή η καθυστέρηση σε αυτή την περίπτωση είναι δύο κύκλους. Για τη μείωση της καθυστέρησης επιτρέπουμε την εκτέλεση της επόμενης ακολουθιακής εντολής μετά την εντολή της διακλάδωσης στην πρώτη θέση καθυστέρησης της διακλάδωσης (branch delay slot). Στην δεύτερη θέση καθυστέρησης της διακλάδωσης έχουμε το πάγωμα (freeze) της ομοχειρίας. Το πάγωμα της ομοχειρίας γίνεται με την ανάκληση της εντολής nop. Το ίδιο συμβαίνει και στην περίπτωση των διακλαδώσεων χωρίς συνθήκη.

Σχήμα 3.11: Η συμπεριφορά του επεξεργαστή στις επιτυχείς διακλαδώσεις.

3.5 ΕΞΑΙΡΕΣΕΙΣ

3.5.1 Εισαγωγή

Για την περιγραφή των περιπτώσεων που μεταβάλλουν την κανονική ροή του προγράμματος σε διάφορους επεξεργαστές χρησιμοποιούνται οι όροι διακοπές (Interrupts), εξαιρέσεις (exceptions) και οι παγίδες (traps). Ο μηχανισμός που χρησιμοποιείται για την αντιμετώπιση αυτών των καταστάσεων είναι ο ίδιος. Η διαφορά στην ονοματολογία οφείλεται στο ότι οι διακοπές δημιουργούνται από εξωτερικά συμβάντα (π.χ. ελεγκτή δίσκου), οι εξαιρέσεις δημιουργούνται για επικοινωνία των προγραμμάτων με το λειτουργικό σύστημα (π.χ. για I/O) και παγίδες δημιουργούνται συνήθως από εσωτερικά σφάλματα στο υλικό ή στο λογισμικό (π.χ. διαίρεση με το μηδέν). Εδώ θα χρησιμοποιήσουμε τον όρο εξαίρεση για να καλύψουμε όλους τους μηχανισμούς, συμπεριλαμβάνοντας τους ακόλουθους:

- Αίτηση συσκευής Εισόδου/Εξόδου (I/O device)
- Κλήση υπηρεσίας του λειτουργικού συστήματος από ένα πρόγραμμα χρήστη
- Ιχνηλάτηση εκτέλεσης προγράμματος (Tracing)
- Διακοπή με αίτηση του χρήστη (Breakpoint)
- Υπερχείλιση ακέραιας αριθμητικής
- Ανωμαλία αριθμητικής κινητής υποδιαστολής
- Σφάλμα σελίδας (Page Fault)
- Μη ευθυγραμμισμένη πρόσβαση μνήμης
- Παραβίαση προστασίας μνήμης
- Χρήση εντολής που δεν έχει οριστεί ή που δεν έχει υλοποιηθεί
- Αστοχία υλικού
- Αστοχία πηγής ρεύματος

Όλα τα παραπάνω απαιτούν ειδικό κώδικα για την εξυπηρέτησή τους που λέγεται interrupt handler και μετά επιστροφή στην κανονική ροή του προγράμματος.

3.5.2 Κατηγορίες εξαιρέσεων

Οι κατηγορίες των εξαιρέσεων ανάλογα με τις ενέργειες που απαιτούνται από το υλικό είναι οι εξής:

- Σύγχρονες/ Ασύγχρονες.
    - Σύγχρονες είναι οι εξαιρέσεις που συμβαίνουν στο ίδιο σημείο του προγράμματος κάθε φορά που εκτελείται και επίσης με τα ίδια δεδομένα και κατανομή μνήμης. Η εξυπηρέτηση τους πρέπει να είναι άμεση γιατί η εκτέλεση του προγράμματος δεν μπορεί να συνεχιστεί. Ένα παράδειγμα είναι οι διακοπές εικονικής μνήμης.
    - Ασύγχρονες είναι οι εξαιρέσεις που δημιουργούνται από εξωτερικές πηγές. Οι διακοπές αυτές εξυπηρετούνται μετά την ολοκλήρωση της τρέχουσας εντολής. Ένα παράδειγμα είναι οι διακοπές από περιφερειακές συσκευές.
- Αιτούμενη από το χρήστη (user requested)/ Καταναγκαστική (coerced).
    - Αιτούμενη από το χρήστη είναι η διαδικασία που αιτεί άμεσα το γεγονός. Αυτό το είδος εξαιρέσεων δεν είναι πραγματικές εξαιρέσεις επειδή είναι προβλέψιμες. Τις χειριζόμαστε όμως σαν εξαιρέσεις επειδή χρησιμοποιούν τους ίδιους μηχανισμούς. Η εξυπηρέτησή τους γίνεται μετά την ολοκλήρωση της εντολής.
    - Οι καταναγκαστικές εξαιρέσεις προκαλούνται από κάποιο γεγονός στο υλικό του συστήματος. Οι εξαιρέσεις αυτές είναι δυσκολότερες στην αντιμετώπιση επειδή δεν είναι προβλέψιμες.
- Αποκρυπτόμενες (User Maskable)/ Μη αποκρυπτόμενες από τον χρήστη (Nonmaskable).
    - Αποκρυπτόμενες είναι οι εξαιρέσεις που μπορούν να αγνοηθούν ή να μείνουν κρυφές από τον χρήστη. Αυτό γίνεται με μια μάσκα που ελέγχει αν το υλικό θα ανταποκριθεί ή όχι στην εξαίρεση.
- Εντός/μεταξύ εντολών.

- Αυτή η κατηγορία εξαρτάται από το κατά πότε γίνεται η εξαίρεση δηλαδή αν γίνεται στην μέση της εκτέλεσης της εντολής ή ανάμεσα στις εντολές. Οι εξαιρέσεις εντός των εντολών είναι συνήθως σύγχρονες. Οι εξαιρέσεις εντός εντολών είναι δυσκολότερες από τις μεταξύ εντολών διότι θα πρέπει η εντολή να σταματήσει και να ξαναεκκινηθεί. Ασύγχρονες εξαιρέσεις εντός εντολών προκύπτουν από καταστροφικές καταστάσεις (π.χ. αστοχία υλικού) και προκαλούν τερματισμό της διεργασίας του χρήστη.

- Συνεχίσιμες (Resume)/ Τερματικές (Terminate).
  - Συνεχίσιμες είναι οι εξαιρέσεις που η εκτέλεση του προγράμματος συνεχίζεται μετά την εξαίρεση. Οι συνεχίσιμες είναι πιο δύσκολες γιατί ο επεξεργαστής πρέπει να επανεκκινήσει την εκτέλεση του προγράμματος.
  - Τερματικές είναι οι εξαιρέσεις που η εκτέλεση του προγράμματος τερματίζεται μετά την εξαίρεση.

3.5.3 Υλοποίηση Εξαιρέσεων στον MIPS

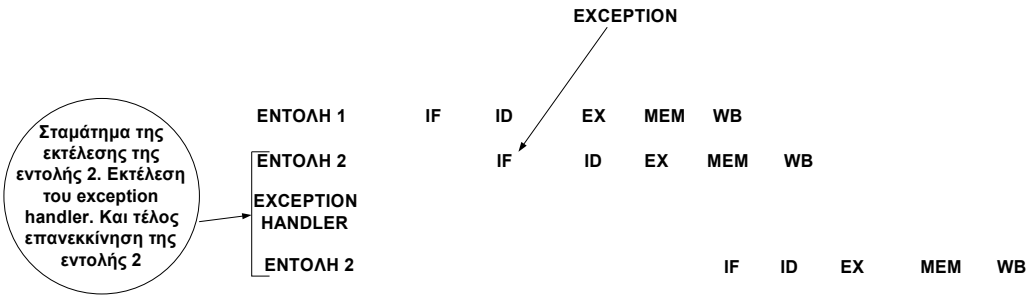
Η διαχείριση των εξαιρέσεων στον MIPS γίνονται από τον συνεπεξεργαστή μηδέν (coprocessor 0). Ο συνεπεξεργαστής είναι μία λογικά αυτόνομη μονάδα με τους δικούς της καταχωρητές αλλά βρίσκεται μαζί με τον κύριο επεξεργαστή. Οι καταχωρητές του συνεπεξεργαστή 0 είναι:

- $12=Status register, ο καταχωρητής αυτός περιέχει πληροφορίες για την κατάσταση του επεξεργαστή (processor mode), την κατάσταση του λειτουργικού (operating mode) και τέλος πληροφορίες για ενεργοποίηση των διακοπών (interrupt enabling).
- $13= Cause register, ο καταχωρητής αυτός περιέχει πληροφορίες για τον τύπο της εξαίρεσης και από που προέρχεται.
- $14=EPC (Exception PC), ο καταχωρητής αυτός περιέχει τη διεύθυνση της διακοπής.

Η επικοινωνία του συνεπεξεργαστή 0 με τον επεξεργαστή γίνεται με τις εντολές mfc0 (move from coprocessor 0) και mtc0 (move to coprocessor 0).

Σε έναν ομόχειρο επεξεργαστή όταν συμβεί μία εξαίρεση ο έλεγχος της ομοχειρίας κάνει τα εξής βήματα για να αποθηκεύσει την κατάσταση της ομοχειρίας με ασφάλεια:

- Εισαγωγή μιας εντολής trap στην βαθμίδα IFSTAGE.
- Απενεργοποίηση όλων των εγγραφών για την εντολή που προκάλεσε την εξαίρεση και όλες όσες ακολουθούν στην αλυσίδα της ομοχειρίας.
- Αποθήκευση του PC στον καταχωρητή EPC αν η εντολή που συμβαίνει η εξαίρεση δεν είναι διακλάδωση. Αν είναι διακλάδωση η εντολή πρέπει να σώσουμε τόσα PCs όσο και το πλήθος των θέσεων καθυστέρησης διακλάδωσης συν ένα.
- Ενημέρωση του καταχωρητή Cause register με την αιτία της εξαίρεσης.
- Απενεργοποίηση των διακοπών.
- Σώσιμο του επιπέδου επεξεργασίας.
- Αλλαγή του επιπέδου επεξεργασίας σε kernel
- Μεταφορά του ελέγχου στον interrupt handler.
- Εξυπηρέτηση της συσκευής που προκαλεί την εξαίρεση.
- Επανεργοποίηση των διακοπών.
- Επιστροφή από την εξαίρεση με την εντολή rfe (return from exception).
- Επαναφορά του επιπέδου επεξεργασίας.
- Διακλάδωση στην εντολή που διακόπηκε με χρήση του καταχωρητή EPC.



Σχήμα 3.12: Αντιμετώπιση εξαιρέσεων στον MIPS

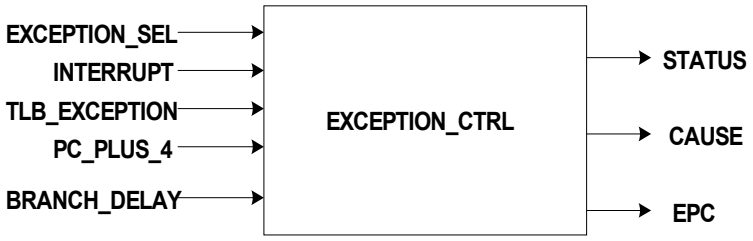
3.5.4 Τι εξαιρέσεις υλοποιήσαμε και με ποιο τρόπο

Στον επεξεργαστή μας υλοποιήσαμε την εξωτερική διακοπή (Interrupt) και τρεις εξαιρέσεις οι οποίες προέρχονται από το TLB. Τα σήματα αυτά προέρχονται από το εξωτερικό περιβάλλον του επεξεργαστή. Επομένως, όταν συμβεί μια εξωτερική διακοπή σταματάμε την ανάκληση της τρέχουσας εντολής στο στάδιο ανάκλησης εντολής και ενημερώνουμε τον μετρητή προγράμματος με τη διεύθυνση του διαχειριστή εξαιρέσεων. Η μονάδα που ενημερώνει τον μετρητή προγράμματος είναι η PC_CTRL. Επίσης, πρέπει να επισημάνουμε ότι αντιμετωπίζουμε τις εξαιρέσεις όπως τις απλές εντολές. Πιο συγκεκριμένα στο στάδιο της αποκωδικοποίησης θεωρούμε ότι οι εξωτερικές εξαιρέσεις έχουν κωδικό εντολής null. Επομένως το γεγονός αυτό σε συνδυασμό με τέσσερα σήματα που δηλώνουν το είδος της εξαίρεσης μας βοηθάει στην αποκωδικοποίηση των εξαιρέσεων και συνεπώς στην ενημέρωση των καταχωρητών του συνεπεξεργαστή μηδέν με τις σωστές τιμές. Η ενημέρωση των καταχωρητών του συνεπεξεργαστή μηδεν γίνεται τον πέμπτο κύκλο. Η μονάδα που δημιουργεί τις σωστές τιμές που πρέπει να γραφτούν στους καταχωρητές του συνεπεξεργαστή μηδέν είναι η EXCEPTION_CTRL και βρίσκεται στο στάδιο εκτέλεσης αριθμητικών και λογικών πράξεων. Ο κώδικας σε vhdl που υλοποιεί τη μονάδα αυτή είναι ο παρακάτω.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity exceptions_ctrl is
port(
                exception_sel:          in std_logic_vector(3 downto 0);
                PC4:            in std_logic_vector(31 downto 0);
                BranchDelay:            in std_logic;
                Interrupt:              in std_logic;
                TLB_Exception: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
                --Ovf:          in std_logic;
                Status:         out std_logic_vector(31 downto 0);
                Cause:          out std_logic_vector(31 downto 0);
                Epc:            out std_logic_vector(31 downto 0)
                );
end exceptions_ctrl;

architecture Behavioral of exceptions_ctrl is

begin
 process(exception_sel,PC4,BranchDelay,Interrupt,TLB_Exception)
 begin
        If(Interrupt='1')then
                if(exception_sel="1100") then
                        if(BranchDelay='0')then
                                Epc<=PC4-4;
                                Status<="00010000000000000000000100000001";
                                Cause<="00000000000000000000001000000000";
                        elsif(BranchDelay='1')then
                                Epc<=PC4-4;
                                Status<="00000000000000000000000100000001";
                                Cause<="10000000000000000000001000000000";
                        end if;
                        else
                                Epc<="00000000000000000000000000000000";
                                Status<="00000000000000000000000000000000";
                                Cause<="00000000000000000000000000000000";
                        end if;
        elsIf(Interrupt='0')then
                if(exception_sel="1101" ) then
                        if(BranchDelay='0')then
                                Epc<=PC4-4;
                                Status<="00010000000000000000001000010001";
                                Cause<="00000000000000000000010000100100";
                        elsif(BranchDelay='1')then
                                Epc<=PC4-4;
                                Status<="00000000000000000000001000010001";
                                Cause<="10000000000000000000010000100100";
                        end if;
                elsif(exception_sel="1110") then
                        if(BranchDelay='0')then
                                Epc<=PC4-4;
                                Status<="00000000000000000000010000010001";
```

```
                    Cause<="00000000000000000000100000100000";
                elsif(BranchDelay='1')then
                    Epc<=PC4-4;
                    Status<="00000000000000000000010000010001";
                    Cause<="10000000000000000000100000100000";
                    end if;
            elsif(exception_sel="1111") then
                if(BranchDelay='0')then
                    Epc<=PC4-4;
                    Status<="00000000000000000000010000010001";
                    Cause<="00000000000000000000100000100011";
                elsif(BranchDelay='1')then
                    Epc<=PC4-4;
                    Status<="00000000000000000000010000010001";
                    Cause<="10000000000000000000100000100011";
                end if;
            elsif(exception_sel="1000" ) then
                if(BranchDelay='0')then
                    Epc<=PC4-4;
                    Status<="00010000000000000000001000010001";
                    Cause<="00000000000000000000010000000100";
                elsif(BranchDelay='1')then
                    Epc<=PC4-4;
                    Status<="00000000000000000000001000010001";
                    Cause<="10000000000000000000010000000100";
                end if;
            elsif(exception_sel="1001") then
                if(BranchDelay='0')then
                    Epc<=PC4-4;
                    Status<="00000000000000000000010000010001";
                    Cause<="00000000000000000000100000001000";
                elsif(BranchDelay='1')then
                    Epc<=PC4-4;
                    Status<="00000000000000000000010000010001";
                    Cause<="10000000000000000000100000001000";
                end if;
            elsif(exception_sel="1010") then
                if(BranchDelay='0')then
                    Epc<=PC4-4;
                    Status<="00000000000000000000010000010001";
                    Cause<="00000000000000000000100000001100";
                elsif(BranchDelay='1')then
                    Epc<=PC4-4;
                    Status<="00000000000000000000010000010001";
                    Cause<="10000000000000000000100000001100";
                end if;
            else
                Epc<="00000000000000000000000000000000";
                Status<="00000000000000000000000000000000";
                Cause<="00000000000000000000000000000000";
            end if;
        end if;
        end process;
end Behavioral;
```

Σχήμα 3.13: Ο κώδικας σε vhdl που υλοποιεί τη μονάδα EXCEPTION_CTRL.



Σχήμα 3.14: Η μονάδα EXCEPTION_CTRL.

3.6 ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο που προηγήθηκε παρουσιάσαμε τη βασική ομοχειρία πέντε βαθμίδων. Περιγράψαμε αναλυτικά τη λειτουργία κάθε βαθμίδας καθώς και της μονάδας ελέγχου του επεξεργαστή. Επίσης, παρουσιάσαμε την τεχνική της προώθησης δεδομένων για την αποφυγή των εξαρτήσεων δεδομένων καθώς και το ποιές εξαρτήσεις και με ποιό τρόπο τις υλοποιήσαμε. Όλη η υλοποίηση της εργασίας έγινε με το εργαλείο ISE 7.1.02i της Xilinx και η γλώσσα προγραμματισμού που χρησιμοποιήσαμε είναι η γλώσσα περιγραφής υλικού VHDL. Μετά την υλοποίηση με τη γλώσσα VHDL σειρά έχουν η σύνθεση του επεξεργαστή σε συσκευή fpga, η εξαγωγή σημαντικών συμπερασμάτων από την σύνθεση, η προσομοίωση της λειτουργίας του καθώς και η επιβεβαίωση της σωστής λειτουργίας του. Όλα αυτά τα περιγράφουμε αναλυτικά στο επόμενο κεφάλαιο.

ΚΕΦΑΛΑΙΟ 4

ΣΥΝΘΕΣΗ ΣΕ FPGA ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ MIPS

4.1 ΕΙΣΑΓΩΓΗ

Στο κεφάλαιο αυτό παρουσιάζουμε τον τρόπο που κάναμε σύνθεση τον επεξεργαστή μας σε μια fpga, τα αποτελέσματα που προκύπτουν από τη σύνθεση καθώς και τα συμπεράσματα που προκύπτουν από την μελέτη των αποτελεσμάτων. Επίσης παρουσιάζουμε τον τρόπο που κάνουμε προσομοίωση, τα αποτελέσματα της προσομοίωσης και επιβεβαιώνουμε τη σωστή λειτουργία του επεξεργαστή συγκρίνοντάς τα με τα αποτελέσματα που μας δίνει ο προσομοιωτής PCSpim.

4.2 ΣΥΝΘΕΣΗ ΣΕ FPGA

Η σύνθεση του επεξεργαστή μας έγινε με το εργαλείο ISE 7.1.02i της Xilinx. Η συσκευή που επιλέξαμε για τη σύνθεση είναι μια Virtex4 xc4vsx35 με speed grade -12. Η συσκευή αυτή είναι η μικρότερη που μπορεί να γίνει τοποθέτηση και δρομολόγηση του επεξεργαστή μας. Τα αποτελέσματα της σύνθεσης μας δίνουν πληροφορίες για τον αριθμό των πόρων που χρησιμοποιούνται από τη συσκευή, όπως τον αριθμό των slice registers, τον αριθμό των 4 input LUTS, τον αριθμό των bonded IOBs κ.α. Στο παρακάτω πίνακα φαίνονται αναλυτικά τα αποτελέσματα που προκύπτουν από την σύνθεση του επ εξεργαστή μας στη συσκευή.

Device Utilization Summary

Logic Utilization	Used	Available	Utilization
Total Number Slice Registers:	1,447	30,720	4%
Number used as Flip Flops:	1,283		
Number used as Latches:	164		
Number of 4 input LUTs:	6,680	30,720	21%
Logic Distribution:			
Number of occupied Slices:	6,118	15,360	39%
Number of Slices containing only related logic:	6,118	6,118	100%
Total Number 4 input LUTs:	9,015	30,720	29%
Number used as logic:	6,680		
Number used as a route-thru:	31		
Number used for Dual Port RAMs:	256		
Number used for 32x1 RAMs:	2,048		
Number of bonded IOBs:	247	448	55%
Number of DSP48s:	8	192	4%

Πίνακας 4.1: Τα αποτελέσματα της σύνθεσης

Επίσης η σύνθεση μας δίνει πληροφορίες για τη μέγιστη ταχύτητα που πετυχαίνει ο επεξεργαστής καθώς και τους χρόνους καθυστέρησης των σημάτων σε κάθε στάδιο. Ένα δείγμα των πληροφοριών της σύνθεσης φαίνεται στο παρακάτω σχήμα.

Timing Summary:

Minimum period: 18.607ns (Maximum Frequency: 53.742MHz)

Max Delay: 18.607ns (Levels of Logic = 26)

Source: **PIPE_ALUSTAGE_1/RFWrAddrNxt_EX_1/stage2/Out1 (FF)**
Destination: **PIPE_IFSTAGE_1/IR_1/stage29/Out1 (FF)**

Σχήμα 4.1: Δείγμα πληροφοριών της σύνθεσης

Από το προηγούμενο σχήμα συμπεραίνουμε ότι η μέγιστη ταχύτητα του επεξεργαστή είναι 53,742 MHz. Επίσης, η μέγιστη χρονική καθυστέρηση είναι 18,607ns και παρατηρείται κατα την προώθηση δεδομένων. Το σήμα που έχουμε την μέγιστη καθυστέρηση είναι αυτό που προέρχεται από την έξοδο του σταδίου εκτέλεσης αριθμητικών πράξεων και πηγαίνει στο στάδιο αποκωδικοποίησης της εντολής στη μονάδα PBYPASS. Το σήμα αυτό χρησιμοποιείται στην συνθήκη ελέγχου για το αν υπάρχει εξάρτηση δεδομένων ανάμεσα σε δύο εντολές.

4.3 ΠΡΟΣΟΜΟΙΩΣΗ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

4.3.1 Η διαδικασία της προσομοίωσης

Στην αρχή δημιουργούμε το αρχείο τύπου COE το οποίο περιλαμβάνει τις εντολές που θα εκτελέσει ο επεξεργαστής. Στη συνέχεια με το αρχείο τύπου COE και μέσα από το εργαλείο ISE 7.1.02i της Xilinx αρχικοποιούμε τη μνήμη εντολών που υπάρχει στο στάδιο IFSTAGE. Επόμενο βήμα είναι η δημιουργία, μέσα από το εργαλείο ISE 7.1.02i της Xilinx, του αρχείου VHDL testbench με το οποίο δίνουμε τιμές στο ρολόι και τις εισόδους του επεξεργαστή. Ένα τέτοιο αρχείο φαίνεται στο σχήμα 4.2.

Επιπλέον μέσα από το περιβάλλον του εργαλείου της Xilinx εκτελούμε δύο είδη προσομοιώσεων το ένα είναι το Behavioral Model και το άλλο είναι Post-Place and Route Vhdl Model. Η διαφορά των δύο προσομοιώσεων είναι ότι το δεύτερο είδος κάνει τοποθέτηση και δρομολόγηση του επεξεργαστή πάνω στη συσκευή που έχουμε επιλέξει στην αρχή για να κάνουμε σύνθεση και μετά κάνει την προσομοίωση. Ενώ το πρώτο μοντέλο κάνει προσομοίωση πάνω στο κώδικα VHDL. Και τα δύο μοντέλα προσομοιώσεων εκτελούνται με το εργαλείο Modelsim SE 6.0a.

Τα αποτελέσματα της προσομοίωσης φαίνονται στα αρχεία outputRF.txt και outputSpecialReg.txt. Τα αρχεία αυτά περιέχουν τα αποτελέσματα του αρχείου καταχωρητών και τα αποτελέσματα των ειδικών καταχωρητών αντίστοιχα.

Τέλος, η επιβεβαίωση των αποτελεσμάτων της προσομοίωσης γίνετα μέσα από τον προσομοιωτή PCSpim 7.2 ελέγχοντας τις τίμες του αρχείου καταχωρητών και των ειδικών καταχωρητών.

Για την επιβεβαίωση της σωστής λειτουργίας του επεξεργαστή ξεκινήσαμε με μικρά προγραμματάκια για κάθε τύπο εντολών. Στη συνέχεια δημιουργήσαμε προγραμματάκια με διάφορες εντολές από κάθε τύπο εντολών. Επιπλέον, δημιουργήσαμε προγραμματάκια με διάφορες εντολές και όλες τις πιθανές εξαρτήσεις δεδομένων.Τέλος, δημιουργήσαμε προγραμματάκια με διάφορες εντολές και πιθανές εξαιρέσεις που μπορούν να συμβούν. Στις επόμενες ενότητες παραθέτουμε ένα πρόγραμμα με αρκετές εντολές και πολλές εξαρτήσεις

δεδομένων καθώς και δύο γνωστά προγράμματα. Τα γνωστά προγράμματα είναι το bubble sort και το Fibonacci τα οποία τα έχουμε τρέξει για διάφορες εισόδους. Το Fibonacci τρέχει σωστά και για n=47, για μεγαλύτερο n ο Fibonacci αριθμός χρειάζεται παραπάνω από 32 bits για να το αναπαραστήσουμε πράγμα το οποίο δεν είναι εφικτό στον δικό μας 32μπιτο επεξεργαστή. Συνολικά έχουμε τρέξει πάνω από 60 διαφορετικά προγράμματα και το μεγαλύτερο πρόγραμμα περιείχε πάνω από 500 εντολές. Όλα τα προγράμματα λειτουργούσαν σωστά.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY pipeline_extra_testbench_vhd IS
END pipeline_extra_testbench_vhd;

ARCHITECTURE behavior OF pipeline_extra_testbench_vhd IS

        -- Component Declaration for the Unit Under Test (UUT)
COMPONENT pipeline_extra
    PORT(
                Clk : IN std_logic;
                Interrupt : IN std_logic;
                ExternalINT : IN std_logic;
                TLB_Exceptions : IN std_logic_vector(2 downto 0);
                PipeRegLdEn : IN std_logic;
                Reset : IN std_logic;
                Instr : OUT std_logic_vector(31 downto 0);
                result : OUT std_logic_vector(31 downto 0);
                PC_sel : OUT std_logic_vector(1 downto 0);
                Displacement1 : OUT std_logic_vector(31 downto 0);
                DataHI : OUT std_logic_vector(31 downto 0);
                DataLO : OUT std_logic_vector(31 downto 0);
                RFEn : OUT std_logic;
                RFEnHI : OUT std_logic;
                RFEnLO : OUT std_logic;
                AddrWr : OUT std_logic_vector(4 downto 0);
                DataSTATUS : OUT std_logic_vector(31 downto 0);
                DataCAUSE : OUT std_logic_vector(31 downto 0);
                DataEPC : OUT std_logic_vector(31 downto 0);
                RFEnSTATUS : OUT std_logic;
                RFEnCAUSE : OUT std_logic;
                RFEnEPC : OUT std_logic;
                Start_read : OUT std_logic
        );
END COMPONENT;
```

```
--Inputs
SIGNAL Clk :  std_logic := '0';
SIGNAL Interrupt :  std_logic := '0';
SIGNAL ExternalINT :  std_logic := '0';
SIGNAL PipeRegLdEn :  std_logic := '0';
SIGNAL Reset :  std_logic := '0';
SIGNAL TLB_Exceptions :  std_logic_vector(2 downto 0) := (others=>'0');

--Outputs
SIGNAL Instr :  std_logic_vector(31 downto 0);
SIGNAL result :  std_logic_vector(31 downto 0);
SIGNAL PC_sel :  std_logic_vector(1 downto 0);
SIGNAL Displacement1 :  std_logic_vector(31 downto 0);
SIGNAL DataHI :  std_logic_vector(31 downto 0);
SIGNAL DataLO :  std_logic_vector(31 downto 0);
SIGNAL RFEn :  std_logic;
SIGNAL RFEnHI :  std_logic;
SIGNAL RFEnLO :  std_logic;
SIGNAL AddrWr :  std_logic_vector(4 downto 0);
SIGNAL DataSTATUS :  std_logic_vector(31 downto 0);
SIGNAL DataCAUSE :  std_logic_vector(31 downto 0);
SIGNAL DataEPC :  std_logic_vector(31 downto 0);
SIGNAL RFEnSTATUS :  std_logic;
SIGNAL RFEnCAUSE :  std_logic;
SIGNAL RFEnEPC :  std_logic;
SIGNAL Start_read :  std_logic;

BEGIN
	uut: pipeline_extra
		PORT MAP(

Clk => Clk,
Interrupt => Interrupt,
ExternalINT => ExternalINT,
TLB_Exceptions => TLB_Exceptions,
PipeRegLdEn => PipeRegLdEn,
Reset => Reset,
Instr => Instr,
result => result,
PC_sel => PC_sel,
Displacement1 => Displacement1,
DataHI => DataHI,
DataLO => DataLO,
RFEn => RFEn,
RFEnHI => RFEnHI,
RFEnLO => RFEnLO,
AddrWr => AddrWr,
DataSTATUS => DataSTATUS,
DataCAUSE => DataCAUSE,
DataEPC => DataEPC,
RFEnSTATUS => RFEnSTATUS,
RFEnCAUSE => RFEnCAUSE,
RFEnEPC => RFEnEPC,
Start_read => Start_read
```

```
                    );
clock_proc: Process
            begin
                        clk <= '0';
                       wait for 50 ns;
                        clk <= '1';
                       wait for 50 ns;
            end process;

tb : PROCESS
            BEGIN
                           reset<='1';
                         Interrupt<='0';
                         wait for 100 ns;
                       PipeRegLdEn<='1';
                           reset<='0';
                         Interrupt<='0';
                    TLB_exceptions<="000";
                       ExternalINT<='0';
                     wait; -- will wait forever
      END PROCESS;
END;
```

Σχήμα 4.2: Το αρχείο VHDL testbench

4.3.2 Παράδειγμα Προσομοίωσης

Για την προσομοίωση του επεξεργαστή μας δημιουργήσαμε ενα αρχείο τύπου COE που περιλαμβάνει ένα μεγάλο εύρος από τις εντολές που υλοποιήσαμε και περιέχει και αρκετές περιπτώσεις εξαρτήσεων δεδομένων. Με το αρχείο αυτό αρχικοποιούμε τη μνήμη εντολών που υπάρχει στο στάδιο IFSTAGE. Οι υπόλοιπες μνήμες είναι όλες αρχικοποιημένες με μηδενικά. Ο παρακάτω πίνακας περιέχει διάφορες εντολές από το πρώτο σετ εντολών του MIPS με εξαρτήσεις δεδομένων:

ΘΕΣΗ ΜΝΗΜΗΣ	ΕΝΤΟΛΗ
1	lui $1, 64
2	ori $2, $1, 132
3	lui $1, 4096
4	Ori $4, $1, 1

5	**ori $6, $0, 6**
6	**add $3, $4, $2**
7	**slt $7, $6, $3**
8	**sub $5, $3, $4**
9	**and $7, $5, $2**
10	**or $8, $3, $5**
11	**mult $7, $5**
12	**mthi $7**
13	**xor $9, $7, $8**
14	**nor $10, $9, $5**
15	**sll $11, $10, 3**
16	**sra $13, $11, 3**
17	**srl $15, $14, 4**
18	**mtlo $16**
19	**sllv $12, $13, $9**
20	**addi $17, $15, 5**
21	**andi $18, $17, 6**
22	**srav $14, $12, $2**
23	**ori $19, $17, 7**
24	**xori $20, $19, 3**
25	**srlv $16, $14, $19**
26	**addiu $21, $14, 4**
27	**subu $22, $20, $21**
28	**addu $23, $20, $21**
29	**sw $23, 3($4)**
30	**ori $2, $0, 2**
31	**lui $1, 4096**
32	**ori $4, $1, 1**
33	**ori $6, $0, 6**
34	**sw $2, 3($4)**
35	**lw $5, 3($4)**
36	**add $7, $6, $5**
37	**sub $16, $7, $5**
38	**and $17, $7, $5**
39	**or $18, $7, $5**
40	**sw $4, 7($4)**
41	**lwl $5, 7($4)**
42	**add $19, $5, $18**
43	**sub $20, $5, $19**
44	**and $21, $5, $19**
45	**or $22, $5, $20**
46	**swr $2, 11($4)**
47	**lw $9, 11($4)**
48	**sw $9, 15($4)**
49	**lw $11, 15($4)**
50	**add $8, $9, $11**
51	**sub $10, $9, $11**
52	**sb $20, 19($4)**
53	**lw $12, 19($4)**
54	**sh $20, 23($4)**
55	**lw $13, 23($4)**
56	**sh $20, 35($4)**
57	**sw $20, 27($4)**

58	**lb $14, 27($4)**
59	**lh $15, 27($4)**
60	**lwl $24, 27($4)**
61	**lwr $25, 27($4)**
62	**ffffffff**
63	**sw $2, 27($4)**
64	**sw $3, 27($4)**
65	**sw $4, 27($4)**
66	**sw $5, 27($4)**
67	**sw $6, 27($4)**
68	**sw $7, 27($4)**
69	**sw $8, 27($4)**
70	**sw $9, 27($4)**
71	**sw $10, 27($4)**
72	**sw $11, 27($4)**
73	**sw $12, 27($4)**
74	**sw $13, 27($4)**
75	**sw $14, 27($4)**
76	**sw $15, 27($4)**
77	**sw $16, 27($4)**
78	**sw $17, 27($4)**
79	**sw $18, 27($4)**
80	**sw $19, 27($4)**
81	**sw $20, 27($4)**
82	**sw $21, 27($4)**
83	**sw $22, 27($4)**
84	**sw $23, 27($4)**
85	**sw $24, 27($4)**
86	**sw $25, 27($4)**
87	**sw $26, 27($4)**
88	**sw $27, 27($4)**
89	**sw $28, 27($4)**
90	**sw $29, 27($4)**
91	**sw $30, 27($4)**
92	**sw $31, 27($4)**
93	**sw $32, 27($4)**
94	**mfhi**
95	**mflo**
96	**mfc0 $5, $12(Status)**
97	**mfc0 $9,$13(Cause)**
98	**mfc0 $13, $14(EPC)**

Πίνακας 4.2: Οι εντολές αρχικοποίησης της μνήμης στο στάδιο IFSTAGE

Ο παρακάτω πίνακας περιέχει τις τιμές των 32 καταχωρητών του αρχείου outputRF.txt καθώς και τις τιμές των ειδικών καταχωρητών του αρχείου outputSpecialReg.txt μετά την εκτέλεση του προγράμματος.

ΚΑΤΑΧΩΡΗΤΗΣ	ΤΙΜΗ
R0	00000000
R1	10000000
R2	00000002
R3	10400085
R4	10000001
R5	01000002
R6	00000006
R7	00000008
R8	00000004
R9	00000002
R10	00000000
R11	00000002
R12	000000F6
R13	0000FFF6
R14	FFFFFFF6
R15	FFFFFFF6
R16	00000006
R17	00000000
R18	0000000A
R19	0100000C
R20	FFFFFFF6
R21	01000000
R22	FFFFFFF6
R23	01F7FFF7
R24	F6000000
R25	FFFFFFF6
R26	00000000
R27	00000000
R28	00000000
R29	00000000
R30	00000000
R31	00000000
HI	00400084
LO	00000000
STATUS	00000000
CAUSE	00000000
EPC	00000000

Πίνακας 4.3: Οι τιμές των 32 καταχωρητών και των ειδικών καταχωρητών μετά την προσομοίωση

Στη συνέχεια κάνουμε την προσομοίωση των ίδιων εντολών με τον προσομοιωτή PCSpim 7.2. Από τη σύγκριση των αποτελεσμάτων διαπιστώνουμε ότι οι τιμές των καταχωρητών είναι ίδιες.

4.3.2 Προσομοίωση του προγράμματος BUBBLE SORT

Το πρόγραμμα bubble sort είναι ένας αλγόριθμος ταξινόμησης. Για παράδειγμα αν έχουμε για είσοδο ένα πίνακα Α με στοιχεία 3,4,10,5,3 ο αλγόριθμος θα δώσει σαν έξοδο τον πίνακα ταξινομημένο κατά αύξουσα σειρά 3,3,4,5,10. Ο κώδικας σε C που υλοποιεί τον αλγόριθμο είναι

```
for (i = 0, i < n – 1; i++)
    for (j = n – 1; j > i;  j--)
        if (x[i] < x[i-1])
            swap(x[i], x[i-1]);
```

.

Ο αντίστοιχος κώδικας σε συμβολική γλώσσα για το PCSpim είναι ο παρακάτω.

```
                .data
                A: .word 4,5,6,7,8,9,10,1,2,3
                .text
                .globl start

start:  li $a0, 10              # parameter n
        sll $a0, $a0, 2         # number of bytes in array A
outer:  sub $t0, $a0, 8         # $t0: j-1
        li $t1, 0               # no swap yet
inner:  lw $t2, A+4($t0)         # $t2 <- A[j]
        lw $t3, A($t0)          # $t3 <- A[j-1]
        bgt $t2, $t3, no_swap   # A[j] <= A[j-1]?
        sw $t2, A($t0)          # A[j-1] <- $t2 \ move bubble
        sw $t3, A+4($t0)        # A[j] <- $t3 / $t2 upwards
        li $t1, 1               # swap occurred
no_swap: sub $t0, $t0, 4        # next array element
        bgez $t0, inner          # more?
        bnez $t1, outer                 # did we swap?
        li $v0, 10               # exit
        syscall
```

Σχήμα 4.3: Ο κώδικας σε συμβολική γλώσσα τουBUBBLE SORT για το PCSpim.

Το αρχείο με το οποίο αρχικοποιούμε τη μνήμη εντολών στο στάδιο της ανάκλησης εντολών περιέχει τις εντολές που φαίνονται στο παρακάτω πίνακα.

ΘΕΣΗ ΜΝΗΜΗΣ	ΕΝΤΟΛΗ
1	ori $4, $0, 10
2	sll $4, $4, 2
3	addi $8, $4, -8
4	ori $9, $0, 0
5	lui $1, 4097
6	addu $1, $1, $8
7	lw $10, 4($1)
8	lui $1, 4097
9	addu $1, $1, $8
10	lw $11, 0($1)
11	slt $1, $11, $10
12	bne $1, $0, 28
13	lui $1, 4097
14	addu $1, $1, $8
15	sw $10, 0($1)
16	lui $1, 4097
17	addu $1, $1, $8
18	sw $11, 4($1)
19	ori $9, $0, 1
20	addi $8, $8, -4
21	bgez $8 -68
22	bne $9, $0, -80
23	ori $2, $0, 10
24	lui $1, 4097
25	addu $1, $1, $8
26	lw $2, 4($1)
27	lw $3, 8($1)
28	lw $4, 12($1)
29	lw $5, 10($1)
30	lw $6, 14($1)
31	lw $7, 18($1)
32	lw $8, 1c($1)
33	lw $9, 20($1)
34	lw $10, 24($1)
35	lw $11, 28($1)
36	ffffffff
37	sw $1, 27($4)
38	sw $2, 27($4)
39	sw $3, 27($4)
40	sw $4, 27($4)
41	sw $5, 27($4)
42	sw $6, 27($4)
43	sw $7, 27($4)
44	sw $8, 27($4)
45	sw $9, 27($4)
46	sw $10, 27($4)

47	**sw $11, 27($4)**
48	**sw $12, 27($4)**
49	**sw $13, 27($4)**
50	**sw $14, 27($4)**
51	**sw $15, 27($4)**
52	**sw $16, 27($4)**
53	**sw $17, 27($4)**
54	**sw $18, 27($4)**
55	**sw $19, 27($4)**
56	**sw $20, 27($4)**
57	**sw $21, 27($4)**
58	**sw $22, 27($4)**
59	**sw $23, 27($4)**
60	**sw $24, 27($4)**
61	**sw $25, 27($4)**
62	**sw $26, 27($4)**
63	**sw $27, 27($4)**
64	**sw $28, 27($4)**
65	**sw $29, 27($4)**
66	**sw $30, 27($4)**
67	**sw $31, 27($4)**
68	**mfhi**
69	**mflo**
70	**mfc0 $5, $12(Status)**
71	**mfc0 $9,$13(Cause)**
72	**mfc0 $13, $14(EPC)**

Πίνακας 4.4: Οι εντολές αρχικοποίησης της μνήμης στο στάδιοIFSTAGE

Τα δεδομένα με τα οποία αρχικοποιούμε τη μνήμη του επεξεργαστή φαίνονται στο παρακάτω πίνακα.

ΘΕΣΗ ΜΝΗΜΗΣ (HEX)	ΔΕΔΟΜΕΝΑ
0	3
4	2
8	1
12	10
10	9
14	8
18	7
1C	6
20	5
24	4

Πίνακας 4.5: Τα δεδομένα αρχικοποίησης της μνήμης στο στάδιοMEMSTAGE.

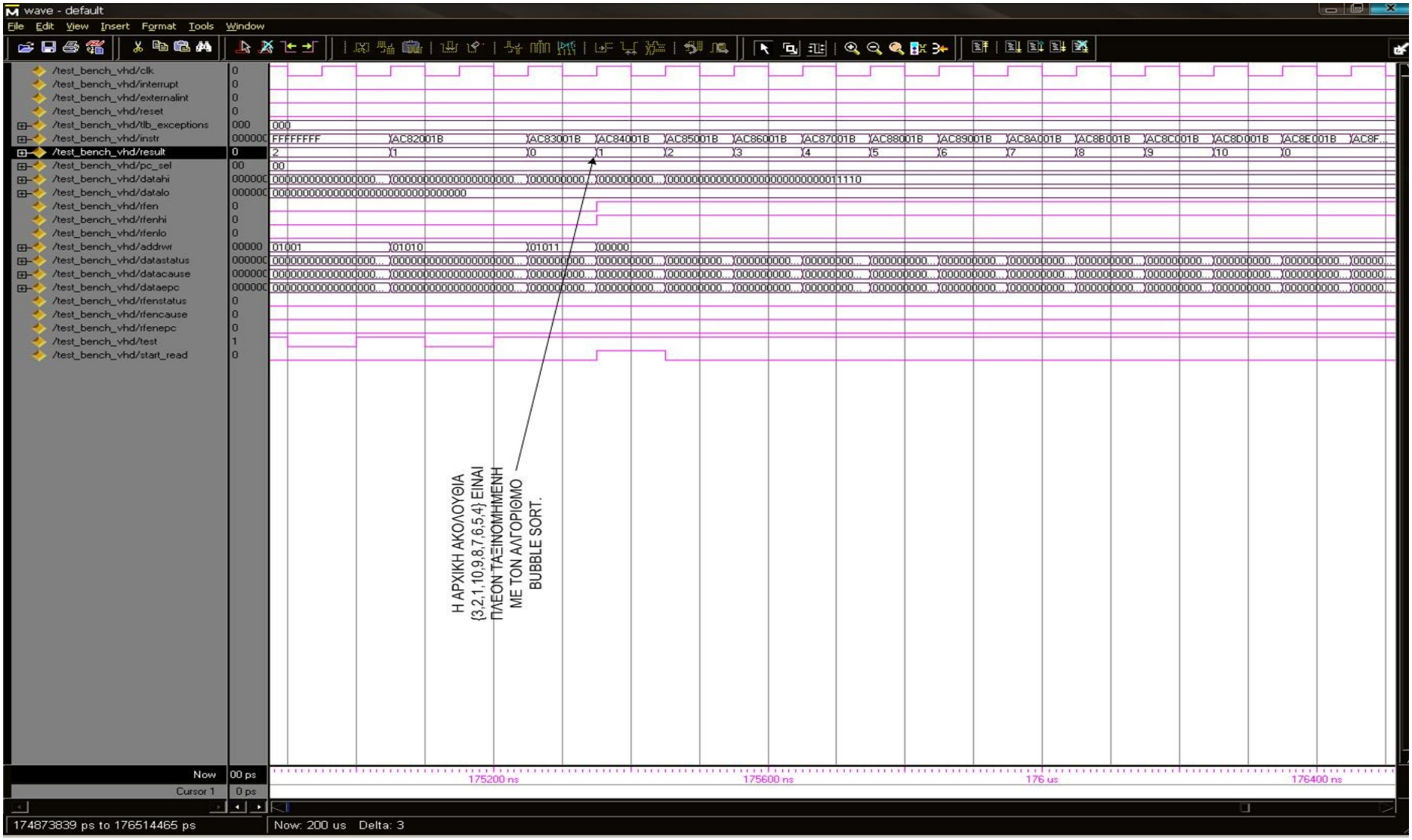
Μετά το τέλος της προσομοίωσης τα δεδομένα στη μνήμη έχουν όπως φαίνονται στο παρακάτω πίνακα.

ΘΕΣΗ ΜΝΗΜΗΣ (HEX)	ΔΕΔΟΜΕΝΑ
0	1
4	2
8	3
12	4
10	5
14	6
18	7
1C	8
20	9
24	10

Πίνακας 4.6: Τα δεδομένα της μνήμης στο στάδιοMEMSTAGE μετά την προσομοίωση

Όπως παρατηρούμε από τον προηγούμενο πίνακα τα δεδομένα που είχαμε βάλει σε μη ταξινομημένη σειρά είναι τώρα ταξινομημένα.

Στο παρακάτω σχήμα βλέπουμε την κυματομορφή της προσομοίωσης για το παράδειγμα της ταξινόμησης των αριθμών με τον αλγόριθμο bubble sort.


wave - default
File Edit View Insert Format Tools Window
/test_bench_vhd/clk
/test_bench_vhd/interrupt
/test_bench_vhd/externalint
/test_bench_vhd/reset
/test_bench_vhd/tlb_exceptions
/test_bench_vhd/instr
/test_bench_vhd/result
/test_bench_vhd/pc_sel
/test_bench_vhd/datahi
/test_bench_vhd/datalo
/test_bench_vhd/rfen
/test_bench_vhd/rfenhi
/test_bench_vhd/rfenlo
/test_bench_vhd/addrwr
/test_bench_vhd/datastatus
/test_bench_vhd/datacause
/test_bench_vhd/dataepc
/test_bench_vhd/rfenstatus
/test_bench_vhd/rfencause
/test_bench_vhd/rfenepc
/test_bench_vhd/test
/test_bench_vhd/start_read
FFFFFFFF
AC820001B
AC830001B
AC840001B
AC850001B
AC860001B
AC870001B
AC880001B
AC890001B
AC8A001B
AC8B001B
AC8C001B
AC8D001B
AC8E001B
2
1
0
1
2
3
4
5
6
7
8
9
10
0
01001
01010
01011
00000
Η ΑΡΧΙΚΗ ΑΚΟΛΟΥΘΙΑ {3,2,1,10,9,8,7,6,5,4} ΕΙΝΑΙ ΠΛΕΟΝ ΤΑΞΙΝΟΜΗΜΕΝΗ ΜΕ ΤΟΝ ΑΛΓΟΡΙΘΜΟ BUBBLE SORT.
Now
Cursor 1
175200 ns
175600 ns
176 us
176400 ns
174873839 ps to 176514465 ps
Now: 200 us Delta: 3

4.3.3 Προσομοίωση του προγράμματος FIBONACCI NUMBERS

Το πρόγραμμα Fibonacci numbers είναι μια άπειρη ακολουθία από θετικούς αριθμούς. Οι αριθμοί ορίζονται ως εξής:

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-2) + fib(n-1) & \text{if } n \geqslant 2 \end{cases}$$

Ένα παράδειγμα είναι το παρακάτω, για n=3:

$$fib(3) = fib(1) + fib(2) = 1 + fib(0) + fib(1) = 1 + 0 + 1 = 2$$

Ο αντίστοιχος κώδικας σε συμβολική γλώσσα για το PCSpim είναι το παρακάτω.

```
.data
result: .word 0x11111111
.text
.globl start
start:
        li $a0, 20 # fib(n): parameter n
        move $v0, $a0 # n < 2 => fib(n) = n
        blt $a0, 2, done
        li $t0, 0 # second last Fib' number
        li $v0, 1 # last Fib' number
fib:    add $t1, $t0, $v0 # compute next Fib' number in
        sequence
        move $t0, $v0 # update second last
        move $v0, $t1 # update last
        sub $a0, $a0, 1 # more work to do?
        bgt $a0, 1, fib # yes: iterate again
done: sw $v0, result # no: store result, done
```

Σχήμα 4.4: Ο κώδικας σε συμβολική γλώσσα του FIBONACCI NUMBER για το PCSpim.

Ο παρακάτω πίνακας περιέχει τις εντολές με τις οποίες αρχικοποιήσαμε τη μνήμη ανάκτησης εντολής. Οι υπόλοιπες μνήμες είναι αρχικοποιημένες με μηδέν.

ΘΕΣΗ ΜΝΗΜΗΣ	ΕΝΤΟΛΗ
1	ori $4, $0, 20
2	addu $2, $0, $4
3	slti $1, $4, 2
4	bne $1, $0, 32
5	ori $8, $0, 0
6	ori $2, $0, 1
7	add $9, $8, $2
8	addu $8, $0, $2
9	addu $2, $0, $9
10	addi $4, $4, -1
11	slti $1, $4, 2
12	beq $1, $0, -24
13	lui $1, 4097
14	ffffffff
15	sw $1, 27($4)
16	sw $2, 27($4)
17	sw $3, 27($4)
18	sw $4, 27($4)
19	sw $5, 27($4)
20	sw $6, 27($4)
21	sw $7, 27($4)
22	sw $8, 27($4)
23	sw $9, 27($4)
24	sw $10, 27($4)
25	sw $11, 27($4)
26	sw $12, 27($4)
27	sw $13, 27($4)
28	sw $14, 27($4)
29	sw $15, 27($4)
30	sw $16, 27($4)
31	sw $17, 27($4)
32	sw $18, 27($4)
33	sw $19, 27($4)
34	sw $20, 27($4)
35	sw $21, 27($4)
36	sw $22, 27($4)
37	sw $23, 27($4)
38	sw $24, 27($4)
39	sw $25, 27($4)
40	sw $26, 27($4)
41	sw $27, 27($4)
42	sw $28, 27($4)
43	sw $29, 27($4)
44	sw $30, 27($4)
45	sw $31, 27($4)
46	mfhi

47	**mflo**
48	**mfc0 $5, $12(Status)**
49	**mfc0 $9,$13(Cause)**
50	**mfc0 $13, $14(EPC)**

Πίνακας 4.7: Οι εντολές αρχικοποίησης της μνήμης στο στάδιοIFSTAGE

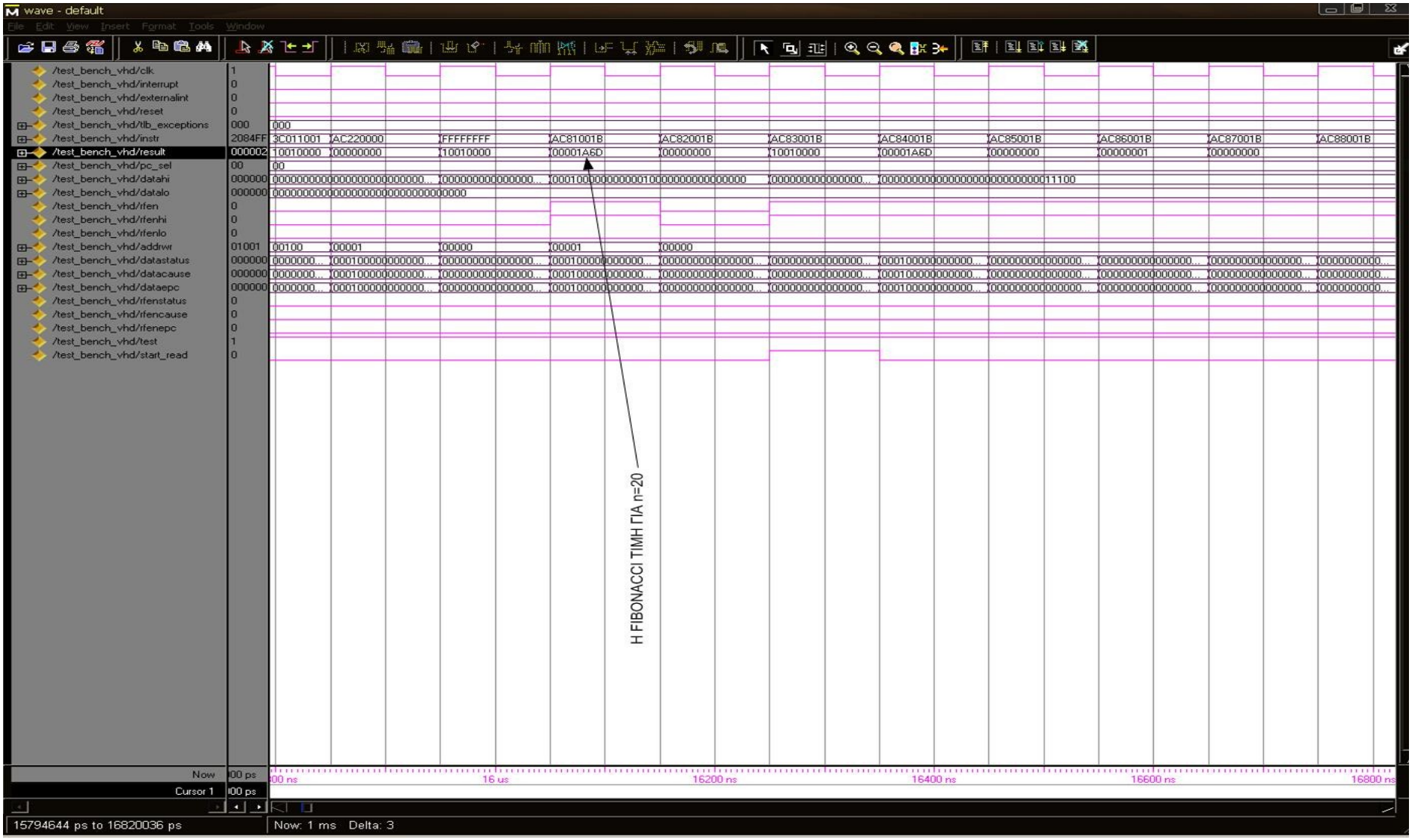
Μετά το τέλος της προσομοίωσης οι τιμές του αρχείου καταχωρητών φαίνονται στον παρακάτω πίνακα.

ΚΑΤΑΧΩΡΗΤΗΣ	ΤΙΜΗ
R0	**00000000**
R1	**10010000**
R2	**00001a6d**
R3	**00000000**
R4	**00000001**
R5	**00000000**
R6	**00000000**
R7	**00000000**
R8	**00001055**
R9	**00001a6d**
R10	**00000000**
R11	**00000000**
R12	**00000000**
R13	**00000000**
R14	**00000000**
R15	**00000000**
R16	**00000000**
R17	**00000000**
R18	**00000000**
R19	**00000000**
R20	**00000000**
R21	**00000000**
R22	**00000000**
R23	**00000000**
R24	**00000000**
R25	**00000000**
R26	**00000000**
R27	**00000000**
R28	**00000000**
R29	**00000000**
R30	**00000000**
R31	**00000000**
HI	**00000000**
LO	**00000000**
STATUS	**00000000**
CAUSE	**00000000**
EPC	**00000000**

Πίνακας 4.8: Οι εντολές αρχικοποίησης της μνήμης στο στάδιοIFSTAGE

Συγκρίνοντας τα αποτελέσματα της προσομοίωσης με αυτά που μας έδωσε ο προσομοιωτής PCSpim παρατηρούμε ότι είναι τα ίδια. Ο καταχωρητής R2 μας δίνει τον ζητούμενο Fibonacci αριθμό. Στο δικό μας παράδειγμα για n=20 έχουμε R2=00001a6d (HEX).

Στο παρακάτω σχήμα βλέπουμε την κυματομορφή της προσομοίωσης για τη fibonacci συνάρτηση για n=20.


wave - default
File Edit View Insert Format Tools Window
/test_bench_vhd/clk
/test_bench_vhd/interrupt
/test_bench_vhd/externalint
/test_bench_vhd/reset
/test_bench_vhd/tlb_exceptions
/test_bench_vhd/instr
/test_bench_vhd/result
/test_bench_vhd/pc_sel
/test_bench_vhd/datahi
/test_bench_vhd/datalo
/test_bench_vhd/rfen
/test_bench_vhd/rfenhi
/test_bench_vhd/rfenlo
/test_bench_vhd/addrwr
/test_bench_vhd/datastatus
/test_bench_vhd/datacause
/test_bench_vhd/dataepc
/test_bench_vhd/rfenstatus
/test_bench_vhd/rfencause
/test_bench_vhd/rfenepc
/test_bench_vhd/test
/test_bench_vhd/start_read
3C011001
AC220000
FFFFFFFF
AC81001B
AC82001B
AC83001B
AC84001B
AC85001B
AC86001B
AC87001B
AC88001B
10010000
00000000
00001A6D
00000001
Η FIBONACCI ΤΙΜΗ ΓΙΑ n=20
Now
Cursor 1
16 us
16200 ns
16400 ns
16600 ns
16800 ns
15794644 ps to 16820036 ps
Now: 1 ms Delta: 3

4.4 ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο αυτό παρουσιάστηκαν η σύνθεση του επεξεργαστή σε μία συσκευή fpga καθώς και τα συμπεράσματα που προκύπτουν από τα αποτελέσματα της σύνθεσης. Επίσης, παρουσιάστηκαν ο τρόπος προσομοίωσης του επεξεργαστή καθώς και η επιβεβαίωση της σωστής λειτουργίας του μέσω του προσομοιωτή PCSpim 7.2. Στο επόμενο κεφάλαιο θα παρουσιαστούν τα σημαντικά στοιχεία της εργασίας, τα συμπεράσματα που προκύπτουν καθώς και περαιτέρω βελτιώσεις που μπορούν να γίνουν στον επεξεργαστή.

ΚΕΦΑΛΑΙΟ 5
ΣΥΜΠΕΡΑΣΜΑΤΑ - ΠΑΡΑΤΗΡΗΣΕΙΣ

5.1 ΑΠΟΤΙΜΗΣΗ ΤΗΣ ΕΡΓΑΣΙΑΣ

Μέσα από την εργασία αυτή ασχοληθήκαμε σε βάθος με διάφορες τεχνικές και χαρακτηριστικά που υπάρχουν σε όλους τους σύγχρονους επεξεργαστές. Συγκεκριμένα υλοποιήσαμε έναν ομόχειρο επεξεργαστή πέντε βαθμίδων αρχιτεκτονικής MIPS. Ο επεξεργαστής αυτός εκτελεί όλες τις εντολές που περιλαμβάνονται στο πρώτο σετ εντολών του MIPS.

Επίσης υλοποιήσαμε τον συνεπεξεργαστή μηδέν ο οποίος έχει τους δικούς του καταχωρητές και διαχειρίζεται τις εξαιρέσεις. Την επικοινωνία ανάμεσα στον επεξεργαστή και τον συνεπεξεργαστή επιτυγχάνεται με την υλοποίηση των εντολών mtc0 και mfc0.

Επίσης ασχοληθήκαμε σε βάθος για το πως αντιμετωπίζουμε τις εξαιρέσεις σε ομόχειρους επεξεργαστές αρχιτεκτονικής μειωμένου συνόλου εντολών. Οι εξαιρέσεις που υλοποιήσαμε είναι αυτές που προέρχονται από τις εντολές break syscall και αυτές που προέρχονται από το εξωτερικό περιβάλλον. Οι εξαιρέσεις που προέρχονται από το εξωτερικό περιβάλλον είναι η εξωτερική διακοπή (Interrupt) και τρείς διακοπές που προέρχονται από το TLB.

Επιπλέον ασχοληθήκαμε εκτενώς με το ποιους κινδύνους (hazards) εμφανίζονται κατά την εκτέλεση των εντολών σε ένα ομόχειρο επεξεργαστή καθώς και με την υλοποίηση διαφόρων τεχνικών για την αποφυγή αυτών των κινδύνων.

Τέλος, έχουμε υλοποιήσει τον απαραίτητο έλεγχο για το αν η κρυφή μνήμη του επεξεργαστή απαντήσει τα σωστά δεδομένα σε δύο κύκλους αντί για ένα.

5.2 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΕΡΑΙΤΕΡΩ ΒΕΛΤΙΩΣΕΙΣ

Σαν αποτέλεσμα όλης αυτής της εργασίας ήταν η υλοποίηση ενός πλήρως λειτουργικού ομόχειρου επεξεργαστή πέντε βαθμίδων εξοπλισμένος με χαρακτηριστικά που υπάρχουν σε όλους τους σύγχρονους επεξεργαστές όπως η τεχνική προώθησης δεδομένων και η υποστήριξη εξαιρέσεων. Οι διαφορές της υλοποίησης αυτής με την αντίστοιχη που είχε γίνει στα πλαίσια του μαθήματος Αρχιτεκτονική Υπολογιστών είναι ότι στην εργασία του μαθήματος υλοποιήσαμε ένα μέρος από τις εντολές που περιλαμβάνονται στο πρώτο σετ εντολών καθώς και μία απλή μορφή της τεχνικής προώθησης δεδομένων. Επόμενως, όλα τα άλλα χαρακτηριστικά που αναφέρθηκαν στην προηγουμένη ενότητα τα υλοποιήσαμε για πρώτη φορά.

Από τα αποτελέσματα της σύνθεσης σε fpga μέσα από το εργαλείο ISE 7.1.02i της Xilinx (βλέπε ενότητα 4.2) παρατηρούμε ότι για την επίτευξη μεγαλύτερης ταχύτητας λειτουργίας αρκεί η ελαχιστοποίηση των χρόνων καθυστέρησης κατά την παραγωγή κάποιων σημάτων από ορισμένες μονάδες. Από μια σύντομη μελέτη που κάναμε στο κώδικα παρατηρήσαμε ότι τις μεγαλύτερες καθυστερήσεις τις έχουμε στις μονάδες PIPE_CTRL και PBYPASS του σταδίου της αποκωδικοποίησης της εντολής. Επομένως, μια βελτίωση της υλοποίησης των μονάδων αυτών θα είχε ως αποτέλεσμα την μείωση των χρόνων καθυστέρησης για τις συγκεκριμένες μονάδες και κατά συνέπεια την αύξηση της ταχύτητας λειτουργίας του επεξεργαστή.

Επίσης το μέσον αλληλεπίδρασης του επεξεργαστή μας με το εξωτερικό περιβάλλον καθώς και οι μηχανισμοί ελέγχου που έχουν υλοποιηθεί κάνουν εφικτή την προσθήκη και την υποστήριξη και άλλων ειδών εξαιρέσεων εκτός από τις υπάρχουσες που υποστηρίζει. Επιπλέον, η σχεδίαση και η λειτουργικότητα του επεξεργαστή είναι τέτοια που επιτρέπει την προσθήκη και επικοινωνία με μια σύγχρονη κρυφή μνήμη δεδομένων.

Τέλος αν στον επεξεργαστή μας προσθέσουμε την υποστήριξη και των άλλων τριών συνόλων εντολών καθώς και την υλοποίηση του συνεπεξεργαστή 1 θα έχουμε τη σχεδίαση

ενός σύγχρονου επεξεργαστή που δεν θα υστερεί σε τίποτα από τους επεξεργαστές που κυκλοφορούν στη σημερινή αγορά.

5.3 ΕΠΙΛΟΓΟΣ

Συνοψίζοντας, παρουσιάστηκαν τα βασικά χαρακτηριστικά της αρχιτεκτονικής του MIPS. Επίσης, παρουσιάστηκε ο τρόπος που υλοποιήσαμε τον ομόχειρο επεξεργαστή πέντε βαθμίδων. Στη συνέχεια, παρουσιάστηκαν ο τρόπος που έγινε η σύνθεση σε συσκευή fpga καθώς και τα αποτελέσματα της σύνθεσης. Επιπλέον, παρουσιάστηκαν ο τρόπος που έγινε η προσομοίωση και τα αποτελέσματά της καθώς και η επιβεβαίωση της σωστής λειτουργίας του επεξεργαστή μέσω του προσομοιωτή PCSpim 7.2. Τέλος, παρουσιάστηκαν τα σημαντικά τεχνικά χαρακτηριστικά που ενσωματώνει ο επεξεργαστή μας καθώς και περαιτέρω βελτιώσεις που μπορούν να γίνουν.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] John L Hennessy, David A. Patterson, Αρχιτεκτονική Υπολογιστών, Τρίτη εκδοση.

[2] Πολυτεχνείο Κρήτης, Σημειώσεις μαθήματος " Οργάνωση Υπολογιστών"

[3] Πολυτεχνείο Κρήτης, Σημειώσεις μαθήματος " Αρχιτεκτονική Υπολογιστών"

[4] John L Hennessy, David A. Patterson, "Computer Organization & Design. The Hardware-Software interface, second edition 1997.

[5] Ηλεκτρονική διεύθυνση www.mips.com

[6] Charles Price, "MIPS IV Instruction Set", revision 3.2, September 1995.

[7] datasheet δίπορτων μνημών από www.xilinx.com.