ΥΣ03 Σχεδίαση Ψηφιακών Συστημάτων

Project 2: Μεθοδολογία σχεδίασης επεξεργαστή πολλών κύκλων

Κρανίτης Νεκτάριος, Βασιλόπουλος Διονύσης

Χειμερινό Εξάμηνο
2023 – 2024

Πίνακας Περιεχομένων

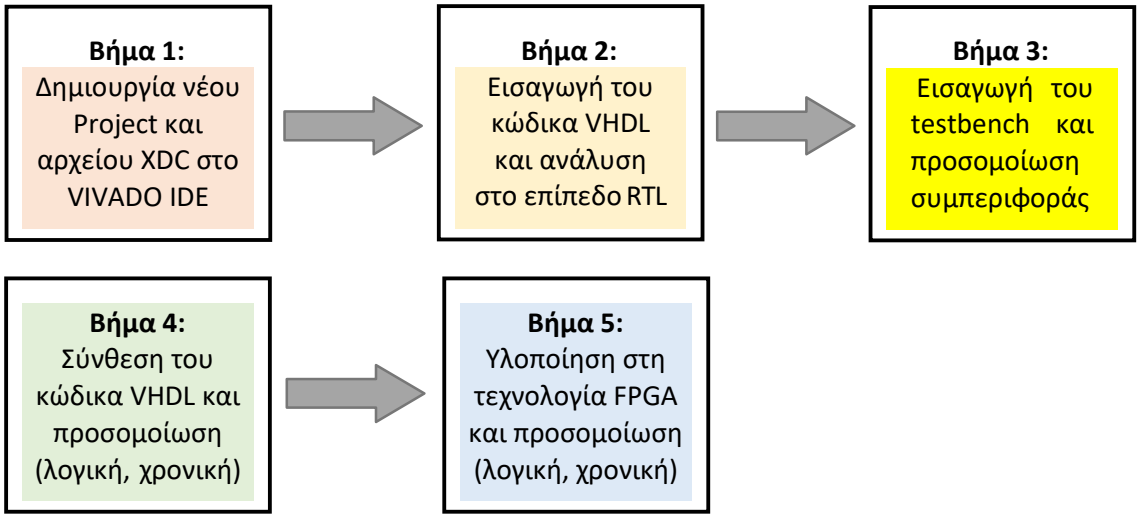
Εισαγωγικά στοιχεία

Στον παρόντα εργαστηριακό οδηγό θα περιγράψουμε αναλυτικά τη μεθοδολογία σχεδίασης της μικροαρχιτεκτονικής ενός απλοποιημένου **επεξεργαστή πολλών κύκλων** αρχιτεκτονικής **ARM** σε τεχνολογία **FPGA** με τη χρήση του εργαλείου **Vivado IDE** (Integrated Development Environment) της Xilinx.

Βασιζόμαστε στη σχεδίαση του επεξεργαστή ενός κύκλου (Project 1). Ξεκινάμε από τη διαδρομή δεδομένων του επεξεργαστή ενός κύκλου, όπου οριοθετούμε τα ψηφιακά δομικά στοιχεία της διαδρομής δεδομένων που απαιτούνται σε κάθε βήμα εκτέλεσης της εντολής (βήματα 1 έως 5) και προχωράμε στη σχεδίαση της διαδρομής δεδομένων του επεξεργαστή πολλών κύκλων τοποθετώντας τους μη αρχιτεκτονικούς ανάμεσα στα υλοποιημένα βήματα ακολουθώντας την ιεραρχική προσέγγιση bottom-up. Μετά, προσδιορίζουμε τα ψηφιακά δομικά στοιχεία της μονάδας ελέγχου και προχωράμε στη σχεδίαση της. Τέλος, στο ανώτερο επίπεδο (top-level) του επεξεργαστή, τοποθετούμε μαζί τη διαδρομή δεδομένων και τη μονάδα ελέγχου, επαληθεύουμε την ορθή σχεδίαση με κατάλληλο πρόγραμμα σε συμβολική γλώσσα ARM, βρίσκουμε τη μέγιστη συχνότητα λειτουργίας και αναλύουμε τις επιδόσεις του επεξεργαστή.

0-1. **Γενική ροή σχεδίασης του επεξεργαστή**

Ακολουθούμε τα Βήματα 1-5 σε όλα τα ιεραρχικά επίπεδα της σχεδίασης του επεξεργαστή: τα δομικά στοιχεία της διαδρομής δεδομένων, τη μονάδα ελέγχου, το ανώτερο επίπεδο του επεξεργαστή.



Βήμα 1: Ανάλυση των απαιτήσεων του επεξεργαστή

1-1. **Απαιτήσεις που απορρέουν από την εφαρμογή της μικροαρχιτεκτονικής ARM**

1-1.1. Κάθε μικροαρχιτεκτονική ARM πρέπει να υλοποιεί όλους τους **αρχιτεκτονικούς καταχωρητές**, που είναι οι εξής:

- **αρχείο καταχωρητών** (register file, RF) με 16 καταχωρητές **R0-R15** των 32 bit,
- **μετρητής προγράμματος** (program counter, PC) που θεωρείται ως ο καταχωρητής **R15** του αρχείου καταχωρητών,
- **καταχωρητής κατάστασης** (status register, SR) των 4 bit για την αποθήκευση των σημαιών **N**, **Z**, **C**, **V**.

1-1.2. Στη μικροαρχιτεκτονική ARM η μνήμη είναι προσπελάσιμη **ανά byte** με λέξεις εντολών/δεδομένων των 32 bit (4 byte). Η διεύθυνση των 32 bit της λέξης ταυτίζεται με τη διεύθυνση του **λιγότερου σημαντικού byte** (least significant byte, LSB) της λέξης και είναι **ευθυγραμμισμένη** (σε πολλαπλάσια του 4). Το περισσότερο σημαντικό byte (most significant byte, **MSB**) της λέξης βρίσκεται στα **αριστερά**, ενώ το λιγότερο σημαντικό byte (least significant byte, **LSB**) της λέξης βρίσκεται στα **δεξιά**.

Για τον προσδιορισμό της διεύθυνσης της λέξης δεδομένων στη μνήμη χρησιμοποιείται ο τρόπος διευθυνσιοδότησης **μνήμης με σχετική απόσταση** (offset addressing) που χρησιμοποιεί έναν καταχωρητή βάσης (base register) που περιέχει τη διεύθυνση βάσης, και μια σχετική απόσταση (offset) ως **μη προσημασμένος ακέραιος των 12 bit** που είναι άμεσα αποθηκευμένος στην ίδια την εντολή. Για να σχηματιστεί η διεύθυνση μνήμης, προσθέτουμε/ αφαιρούμε τη σχετική απόσταση (που μπορεί να είναι 0) στο/από το περιεχόμενο του καταχωρητή βάσης.

1-1.3. Η μικροαρχιτεκτονική ARM υποστηρίζει εντολές που **ενεργοποιούν σημαίες συνθήκης** (*condition flags*) ανάλογα με το αποτέλεσμα μίας πράξης στη μονάδα ALU. Οι εντολές, που έπονται της εντολής που ενεργοποιεί σημαίες συνθήκης, εκτελούνται **υπό συνθήκη**, ανάλογα με τις τιμές των σημαιών.

1-1.4. Η αρχιτεκτονική ARM υποστηρίζει **εντολές υπό συνθήκη**. Το μνημονικό της εντολής ακολουθείται από ένα μνημονικό συνθήκης, το οποίο υποδεικνύει τη συνθήκη (CondEx) που πρέπει να ικανοποιείται για να εκτελεσθεί η συγκεκριμένη εντολή. Το μνημονικό συνθήκης αποθηκεύεται στο πεδίο συνθήκης (cond) της εντολής μεγέθους 4 bit. Στον επόμενο πίνακα φαίνονται τα **μνημονικά συνθήκης** με τις **εξισώσεις Boole** των σημαιών που τις ικανοποιούν.

$cond_{3:0}$	Μνημονικό	Όνομα	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\bar{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\bar{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\bar{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\bar{V}$

$cond_{3:0}$	Μνημονικό	Όνομα	CondEx
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z+\bar{C}$
1010	GE	Signed greater or equal	$\overline{N \oplus V}$
1011	LT	Signed less	$N \oplus V$
1100	GT	Signed greater	$\bar{Z}\,\overline{N \oplus V}$
1101	LE	Signed less or equal	$Z+(N \oplus V)$
1110	AL (ή none)	Always / unconditional	1
1111	none	For unconditional instructions	1

1-1.5. Η μικροαρχιτεκτονική ARM του επεξεργαστή πολλών κύκλων απαρτίζεται από δύο διακριτές μονάδες που αλληλοεπιδρούν μεταξύ τους:

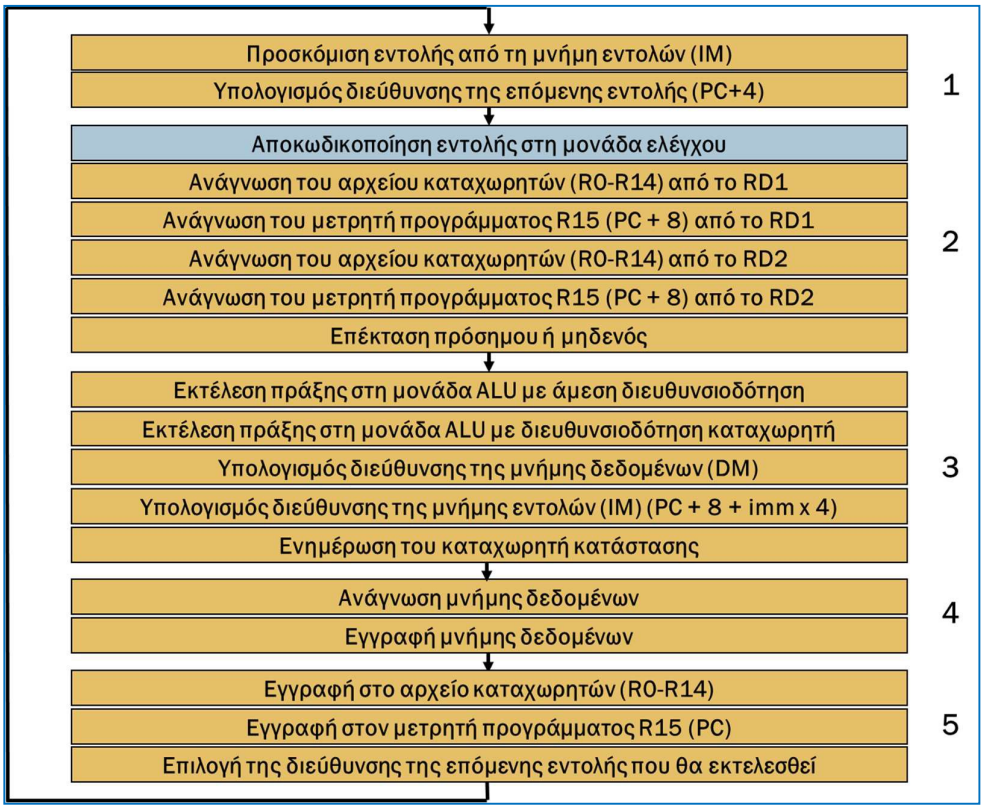
- τη **διαδρομή δεδομένων** (datapath) των 32 bit, που εκτελεί λειτουργίες με λέξεις δεδομένων και περιέχει υπομονάδες όπως: διακριτές μνήμες εντολών (ROM 64 x 32) και δεδομένων (RAM 32 x 32) μεγέθους 32 bit (αρχιτεκτονική Harvard), αρχιτεκτονικούς καταχωρητές (PC και SR) και **μη αρχιτεκτονικούς καταχωρητές** για την εκτέλεση τις εντολής σε πολλούς κύκλους, μονάδα ALU για την εκτέλεση των πράξεων που απορρέουν από το σύνολο εντολών που πρόκειται να υλοποιηθεί, πολυπλέκτες για τον έλεγχο της ροής δεδομένων, επιπλέον συνδυαστική λογική (2 αθροιστές αύξησης κατά 4, μονάδα επέκτασης προσήμου/μηδενός),
- τη **μονάδα ελέγχου** (control unit), που αποκωδικοποιεί την εντολή υπό συνθήκη και περιέχει υπομονάδες συνδυαστικής λογικής, όπως ο αποκωδικοποιητής εντολής και η λογική ελέγχου συνθήκης για την παραγωγή των σημάτων ελέγχου που παραμένουν σταθερά κατά την εκτέλεση της εντολής, καθώς και **μηχανή πεπερασμένων καταστάσεων** (FSM) για την ενεργοποίηση των σημάτων έγκρισης εγγραφής και επιλογής διεύθυνσης επόμενης εντολής σε συγκεκριμένο βήμα (κύκλο), λαμβάνοντας υπόψη τις τιμές των σημαιών.

1-1.6. Στη μικροαρχιτεκτονική ARM του επεξεργαστή πολλών κύκλων ολόκληρη η εντολή εκτελείται σε **3 έως 5 κύκλους ρολογιού** (CLK), όπου η περίοδος του CLK προσδιορίζεται από την **κρίσιμη διαδρομή** (critical path) που ενεργοποιείται κατά την εκτέλεση του πιο αργού βήματος της εντολής μέσω της λογικής της μικροαρχιτεκτονικής πολλών κύκλων.

1-2. **Τα 5 βήματα (κύκλοι) εκτέλεσης της εντολής στη μικροαρχιτεκτονική ARM**

1. Προσκόμιση εντολής και υπολογισμός της επόμενης διεύθυνσης (PC+4)
2. Αποκωδικοποίηση εντολής, ανάγνωση 2 καταχωρητών από το αρχείο καταχωρητών (συμπεριλαμβάνεται και ο μετρητής προγράμματος) και επέκταση προσήμου/μηδενός.
3. Εκτέλεση πράξεων στη μονάδα ALU
4. Ανάγνωση ή εγγραφή στη μνήμη δεδομένων
5. Ετεροχρονισμένη εγγραφή ενός καταχωρητή στο αρχείο καταχωρητών (συμπεριλαμβάνεται και ο μετρητής προγράμματος) και επιλογή της διεύθυνσης της επόμενης εντολής που πρόκειται να εκτελεσθεί.

Στο επόμενο σχήμα φαίνονται οι 18 λειτουργίες που εκτελούνται ανά βήμα (κύκλο).



1-3. **Το σύνολο εντολών που πρόκειται να υλοποιηθεί**

1-3.1. Η εντολή **LDR**

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

- LDR Rd, [Rn, #imm12]; Rd = DM[Rn + #imm12] (U = 1)
- LDR Rd, [Rn, #–imm12]; Rd = DM[Rn – #imm12] (U = 0)

Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστέος στον καταχωρητή προέλευσης Rn,
- ο τελεστέος στον καταχωρητή προορισμού Rd,
- ο μη προσημασμένος άμεσος τελεστέος των 12 bit με επέκταση μηδενός στα 32 bit.

Πράξεις που εκτελούνται στη μονάδα ALU:

- πρόσθεση και αφαίρεση.

Μορφή εντολής ($\bar{I}$ = 0, P = 1, W = 0, B = 0, L = 1):

1-3.2. Η εντολή **STR**

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

- STR Rd, [Rn, #imm12]; DM[Rn + #imm12] = Rd (U = 1)
- STR Rd, [Rn, #–imm12]; DM[Rn – #imm12] = Rd (U = 0)

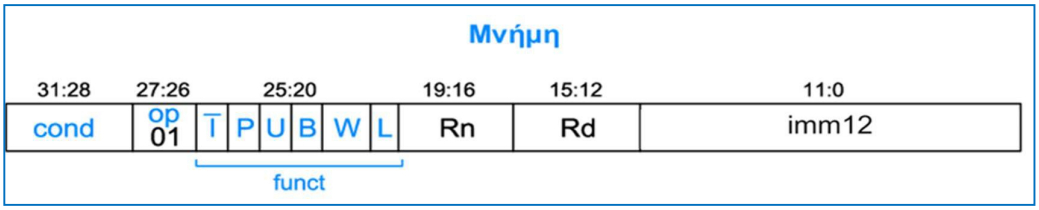
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστέος στον καταχωρητή προέλευσης Rn,
- ο τελεστέος στον καταχωρητή προέλευσης Rd,
- ο μη προσημασμένος άμεσος τελεστέος των 12 bit με επέκταση μηδενός στα 32 bit.

Πράξεις που εκτελούνται στη μονάδα ALU:

- πρόσθεση και αφαίρεση.

Μορφή εντολής ($\bar{I}$ = 0, P = 1, W = 0, B = 0, L = 0):



1-3.3. Οι εντολές επεξεργασίας δεδομένων με άμεση διευθυνσιοδότηση **ALU(S)-I**

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

- ALU(S) Rd, Rn, #imm8; Rd = Rn +/–/and/or/xor imm8
- ενημέρωση σημαιών SR = (N, Z, C, V) (S = 1)

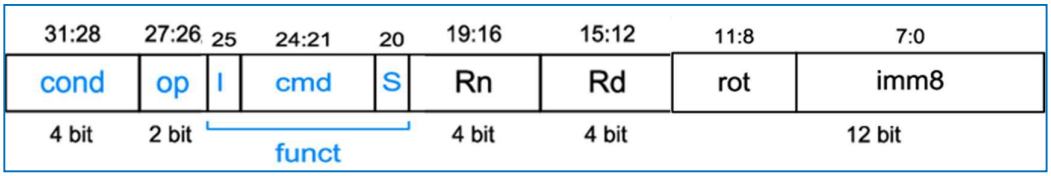
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστέος στον καταχωρητή προέλευσης Rn,
- ο τελεστέος στον καταχωρητή προορισμού Rd,
- ο μη προσημασμένος άμεσος τελεστέος των 12 bit (rot = 0) με επέκταση μηδενός στα 32 bit.

Πράξεις που εκτελούνται στη μονάδα ALU:

- αριθμητικές πράξεις (πρόσθεση, αφαίρεση),
- λογικές πράξεις (and, or, xor)

Μορφή εντολής (I = 1, cmd = 0100 (+), 0010 (–), 0000 (and), 1100 (or), 0001 (xor)):



1-3.4. Οι εντολές επεξεργασίας δεδομένων με διευθυνσιοδότηση καταχωρητή **ALU(S)-R**

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

- ALU(S) Rd, Rn, Rm; Rd = Rn +/–/and/or/xor Rm
- ενημέρωση σημαιών SR = (N, Z, C, V) (S = 1)

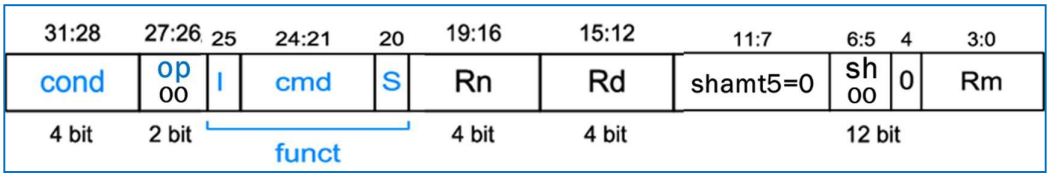
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστέος στον καταχωρητή προέλευσης Rn,
- ο τελεστέος στον καταχωρητή προέλευσης Rm,
- ο τελεστέος στον καταχωρητή προορισμού Rd.

Πράξεις που εκτελούνται στη μονάδα ALU:

- αριθμητικές πράξεις (πρόσθεση, αφαίρεση),
- λογικές πράξεις (and, or, xor)

Μορφή εντολής (I = 0, cmd = 0100 (+), 0010 (–), 0000 (and), 1100 (or), 0001 (xor)):



1-3.5. Η εντολή **CMP**

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

- CMP Rn, #imm8; ενημέρωση σημαιών για Rn – imm8 (I = 1)
- CMP Rn, Rm; ενημέρωση σημαιών για Rn – Rm (I = 0)

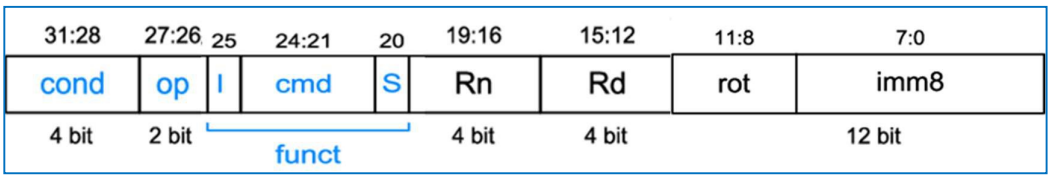
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

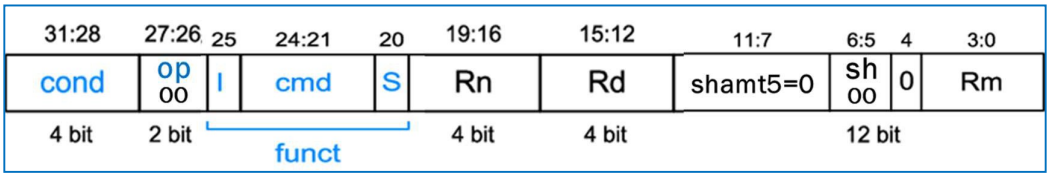
- ο τελεστέος στον καταχωρητή προέλευσης Rn,
- ο τελεστέος στον καταχωρητή προέλευσης Rm (I = 0),
- ο μη προσημασμένος άμεσος τελεστέος των 12 bit (rot = 0) με επέκταση μηδενός στα 32 bit (I = 1).

Πράξεις που εκτελούνται στη μονάδα ALU:

- αφαίρεση.

Μορφές εντολής (S = 1, cmd = 1010):





1-3.6. Οι εντολές μεταφοράς δεδομένων **MOV**, **NOP**, **MVN** (μόνο για S = 0)

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

- MOV Rd, #imm8; Rd = imm8 (I = 1)
- MOV Rd, Rm; Rd = Rm (I = 0)
- NOP = MOV R0, R0; R0 = R0 (I = 0)
- MVN Rd, #imm8; Rd = not imm8 (I = 1)
- MVN Rd, Rm; Rd = not Rm (I = 0)

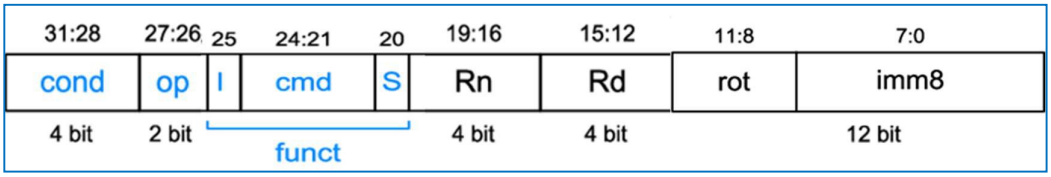
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

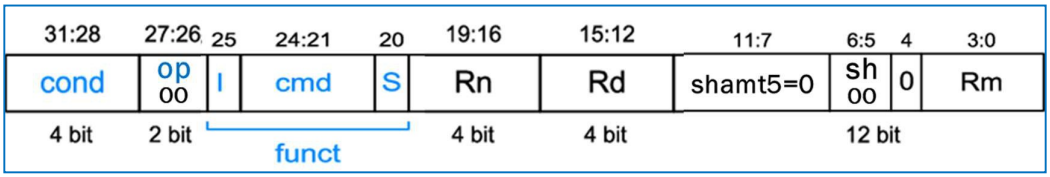
- ο τελεστέος στον καταχωρητή προέλευσης Rm (I = 0),
- ο τελεστέος στον καταχωρητή προορισμού Rd,
- ο μη προσημασμένος άμεσος τελεστέος των 12 bit (rot = 0) με επέκταση μηδενός στα 32 bit (I = 1).

Πράξεις που εκτελούνται στη μονάδα ALU:

- μεταφορά (MOV, NOP),
- αντιστροφή και μεταφορά (MVN).

Μορφές εντολής (S = 0, cmd = 1101 (MOV, NOP), 1111 (MVN), Rn = 0):





1-3.7. Οι εντολές ολίσθησης **LSL**, **LSR**, **ASR**, **ROR** (μόνο για S = 0)

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

- LSL Rd, Rm, #shamt5; Rd = Rm LSL by #shamt5
- LSR Rd, Rm, #shamt5; Rd = Rm LSR by #shamt5
- ASR Rd, Rm, #shamt5; Rd = Rm ASR by #shamt5
- ROR Rd, Rm, #shamt5; Rd = Rm ROR by #shamt5

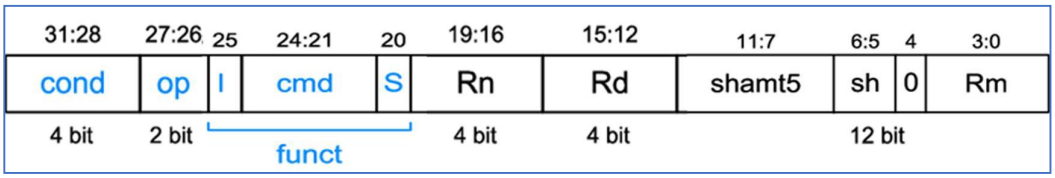
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστέος στον καταχωρητή προέλευσης Rm (I = 0),
- ο τελεστέος στον καταχωρητή προορισμού Rd,
- η ποσότητα ολίσθησης shamt5 των 5 bits.

Πράξεις που εκτελούνται στη μονάδα ALU:

- ολίσθησης (LSL, LSR, ASR),
- περιστροφής (ROR).

Μορφή εντ. (S = 0/1, cmd = 1101, sh = 00 (LSL), 01 (LSR), 10 (ASR), 11 (ROR), Rn = 0):



1-3.8. Οι εντολές διακλάδωσης **B, BL**

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

- B label ; PC = BTA = PC + 8 + imm24 x 4 (L = 0)
- BL label ; PC = BTA = PC + 8 + imm24 x 4; R14 = LR = PC + 4 (L = 1)

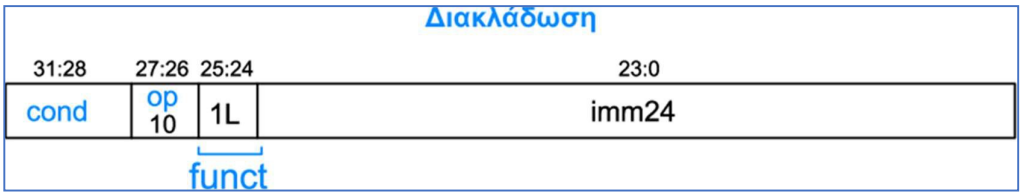
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστέος στον καταχωρητή προορισμού R14 (link register, LR),
- ο προσημασμένος άμεσος τελεστέος των 24 bit, πολλαπλασιασμένος επί 4 και με επέκταση προσήμου στα 32 bit.

Πράξεις που εκτελούνται στη μονάδα ALU:

- πρόσθεση.

Μορφή εντολής (L = 0 (B), 1 (BL)):



Βήμα 2: Σχεδίαση των ψηφιακών δομικών στοιχείων της διαδρομής δεδομένων

Στο κατώτερο ιεραρχικά επίπεδο της σχεδίασης περιγράφεται στη γλώσσα VHDL η συμπεριφορά των ψηφιακών δομικών στοιχείων που απαιτούνται σε κάθε βήμα εκτέλεσης της εντολής. Το μέγεθος των αρτηριών των λειτουργικών μονάδων και των καταχωρητών, καθώς και το μέγεθος των μνημών παραμετροποιείται με τη δήλωση της εντολής generic.

2-1. **Προσκόμιση εντολής και υπολογισμός της επόμενης διεύθυνσης (PC+4)**

2-1.1. Παραμετροποιημένος καταχωρητής των N bit (με αρχική τιμή N = 32) με RESET και WE (συνδέεται με το PCWrite στην περίπτωση του επεξεργαστή πολλών κύκλων) για να χρησιμοποιηθεί ως μετρητής προγράμματος (**program counter, PC**).

2-1.2. Παραμετροποιημένη διάταξη μνήμης ROM με 2^N λέξεις μεγέθους M bit (με αρχικές τιμές N = 6 και M = 32) για να χρησιμοποιηθεί ως μνήμη εντολών (**instruction memory, IM**). Προσοχή! A[N-1:0] = PC[N+1:2].

2-1.3. Παραμετροποιημένος αθροιστής κατά 4 με έξοδο PCPlus4 για τον υπολογισμό της διεύθυνσης της αμέσως επόμενης εντολής PC + 4 (**incrementer by 4, INC4**).

2-2. **Ανάγνωση 2 καταχωρητών από το αρχείο καταχωρητών (συμπεριλαμβάνεται και ο μετρητής προγράμματος) και επέκταση προσήμου/μηδενός**

2-2.1. Παραμετροποιημένο αρχείο καταχωρητών των 2^N καταχωρητών μεγέθους M bit (με αρχικές τιμές N = 4 και M = 32) με δυνατότητα ασύγχρονου διαβάσματος δύο καταχωρητών και σύγχρονης εγγραφής ενός καταχωρητή, όταν RegWrite = 1 (**register file, RF**). Προσοχή! Ο καταχωρητής R15 είναι ο μετρητής προγράμματος PC και δεν υλοποιείται εντός του αρχείου καταχωρητών μαζί με τους υπόλοιπους καταχωρητές R0 έως R14. Όταν διαβάζεται ο R15 επιστρέφει την τιμή PC + 8. Δεν ορίζεται εγγραφή του R15 που να αφορά στο αρχείο καταχωρητών, αφού υλοποιείται ξεχωριστά.

2-2.2. Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στον προσδιορισμό της διεύθυνσης του πρώτου καταχωρητή προέλευσης του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του πεδίου Rn της εντολής (RegSrc[0] = 0) και της σταθερής τιμής 15 (RegSrc[0] = 1). Η έξοδος συνδέεται στην είσοδο A1 του αρχείου καταχωρητών.

2-2.3. Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στον προσδιορισμό της διεύθυνσης του δεύτερου καταχωρητή προέλευσης του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του πεδίου Rm της εντολής (RegSrc[1] = 0) και του πεδίου Rd της εντολής (RegSrc[1] = 1). Η έξοδος συνδέεται στην είσοδο A2 του αρχείου καταχωρητών.

2-2.4. Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στον προσδιορισμό της διεύθυνσης του καταχωρητή προορισμού του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του πεδίου Rd της εντολής (RegSrc[2] = 0) και της σταθερής τιμής 14 (RegSrc[2] = 1). Η έξοδος συνδέεται στην είσοδο A3 του αρχείου καταχωρητών.

2-2.5. Παραμετροποιημένος αθροιστής κατά 4 με είσοδο PCPlus4 και έξοδο PCPlus8 για τον υπολογισμό της διεύθυνσης PC + 8 (**incrementer by 4, INC4**). Η έξοδος συνδέεται στην θύρα R15 του αρχείου καταχωρητών.

2-2.6. Μονάδα επέκτασης μηδενός από τα 12 bit στα 32 Bit (για ImmSrc = 0) ή προσήμου από τα 26 bit (άμεσος τελεστέος των 24 bit πολλαπλασιασμένος επί 4) στα 32 bit (για ImmSrc = 1) (**Extend**).

2-3. **Εκτέλεση πράξεων στη μονάδα ALU**

2-3.1. Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στον προσδιορισμό των δεδομένων της εισόδου SrcB της μονάδας ALU (**mux2to1**). Επιλέγει μεταξύ της θύρας ανάγνωσης A2/RD2 του αρχείου καταχωρητών (ALUSrc = 0) και της εξόδου ExtImm της μονάδας επέκτασης προσήμου-μηδενός (ALUSrc = 1). Η έξοδος συνδέεται στην είσοδο SrcB της μονάδας ALU.

2-3.2. Παραμετροποιημένη μονάδα ALU μεγέθους N bit (με αρχική τιμή N = 32) (**ALU**). Η μονάδα ALU έχει δύο εισόδους SrcA και SrcB, μία έξοδο ALUResult και συμπεριλαμβάνει αθροιστή/αφαιρέτη των N bit, μονάδα λογικών πράξεων των N bit, μονάδα μεταφοράς δεδομένων των N bit, μονάδα ολίσθησης των N bit, πύλη NOR των N bit για τη σημαία Z και τους απαραίτητους πολυπλέκτες 2 σε 1 των N bit. Η μονάδα ALU παράγει τις τιμές των σημαιών N, Z, C, V. Οι σημαίες C και V ενεργοποιούνται (παίρνουν τιμή 1) μόνο όταν εκτελούνται αριθμητικές πράξεις. Το μέγεθος του σήματος ελέγχου ALUControl εξαρτάται από το πλήθος των πράξεων που εκτελούνται στη μονάδα ALU.

2-3.3. Παραμετροποιημένος καταχωρητής των N bit (με αρχική τιμή N = 4) με RESET και WE (FlagsWrite) για να χρησιμοποιηθεί ως καταχωρητής καταστάσεων (**status register, SR**). Χρησιμοποιείται για την αποθήκευση των τιμών σημαιών N, Z, C, V, μόνο όταν το πεδίο S των εντολών επεξεργασίας δεδομένων έχει την τιμή 1.

2-4. **Ανάγνωση ή εγγραφή στη μνήμη δεδομένων**

2-4.1. Παραμετροποιημένη διάταξη μνήμης RAM με 2^N λέξεις μεγέθους M bit (με αρχικές τιμές N = 5 και M = 32) για να χρησιμοποιηθεί ως μνήμη δεδομένων (**data memory, DM**). Προσοχή! A[N-1:0] = ALUResult[N+1:2]. Λόγω του μικρού μεγέθους υλοποιείται ως distributed RAM. Το διάβασμα γίνεται ασύγχρονα, ενώ η εγγραφή γίνεται σύγχρονα, όταν MemWrite = 1. Δεν πρέπει να έχει καταχωρητή εξόδου.

2-5. **Ετεροχρονισμένη εγγραφή ενός καταχωρητή στο αρχείο καταχωρητών (συμπεριλαμβάνεται και ο μετρητής προγράμματος) και επιλογή της διεύθυνσης της επόμενης εντολής που πρόκειται να εκτελεσθεί.**

2-5.1. Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στην επιλογή των δεδομένων που εγγράφονται ετεροχρονισμένα στον καταχωρητή προορισμού Rd (R0-R15) του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του αποτελέσματος ALUResult της πράξης που εκτελείται στη μονάδα ALU (MemtoReg = 0) και της θύρας ανάγνωσης RD της μνήμης δεδομένων

(MemtoReg = 1). Η έξοδος Result χρησιμοποιείται ως είσοδος δεδομένων άλλων πολυπλεκτών 2 σε 1.

2-5.2. Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στην επιλογή των δεδομένων που εγγράφονται ετεροχρονισμένα στον καταχωρητή προορισμού (R0-R14) του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του αποτελέσματος Result (RegSrc[2] = 0) και της τιμής PC + 4 (RegSrc[2] = 1). Η έξοδος συνδέεται στην είσοδο WD3 του αρχείου καταχωρητών. Η εγγραφή γίνεται στους καταχωρητές R0-R14, όταν RegWrite = 1.

2-5.3. Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στην επιλογή της διεύθυνσης της επόμενης εντολής που πρόκειται να εκτελεσθεί (**mux2to1**). Επιλέγει μεταξύ της τιμής PC + 4 (PCSrc = 0) και του αποτελέσματος Result (PCSrc = 1). Η έξοδος συνδέεται στην είσοδο δεδομένων του μετρητή προγράμματος PC. Η τιμή PC + 4 επιλέγεται ως διεύθυνση της επόμενης εντολής που πρόκειται να εκτελεσθεί, όταν εκτελούνται εντολές που δεν αλλάζουν τη ροή του προγράμματος και όταν δεν ικανοποιείται η συνθήκη κατά την εκτέλεση εντολών υπό συνθήκη. Το αποτέλεσμα Result επιλέγεται ως διεύθυνση της επόμενης εντολής που πρόκειται να εκτελεσθεί, είτε όταν ο καταχωρητής προορισμού Rd στις εντολές επεξεργασίας δεδομένων και στην εντολή LDR είναι ο R15, είτε όταν εκτελείται εντολή διακλάδωσης (B, BL).

Βήμα 3: Σχεδίαση της διαδρομής δεδομένων (datapath)

Αρχικά, ολοκληρώνεται η σχεδίαση της διαδρομής δεδομένων (**datapath**) του επεξεργαστή ενός κύκλου με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom-up, όπως αυτή αποτυπώνεται στο ακόλουθο σχηματικό διάγραμμα. Κάθε βήμα της εντολής υλοποιείται ξεχωριστά με **διακριτά στοιχεία** (components) (Σχήμα 1).

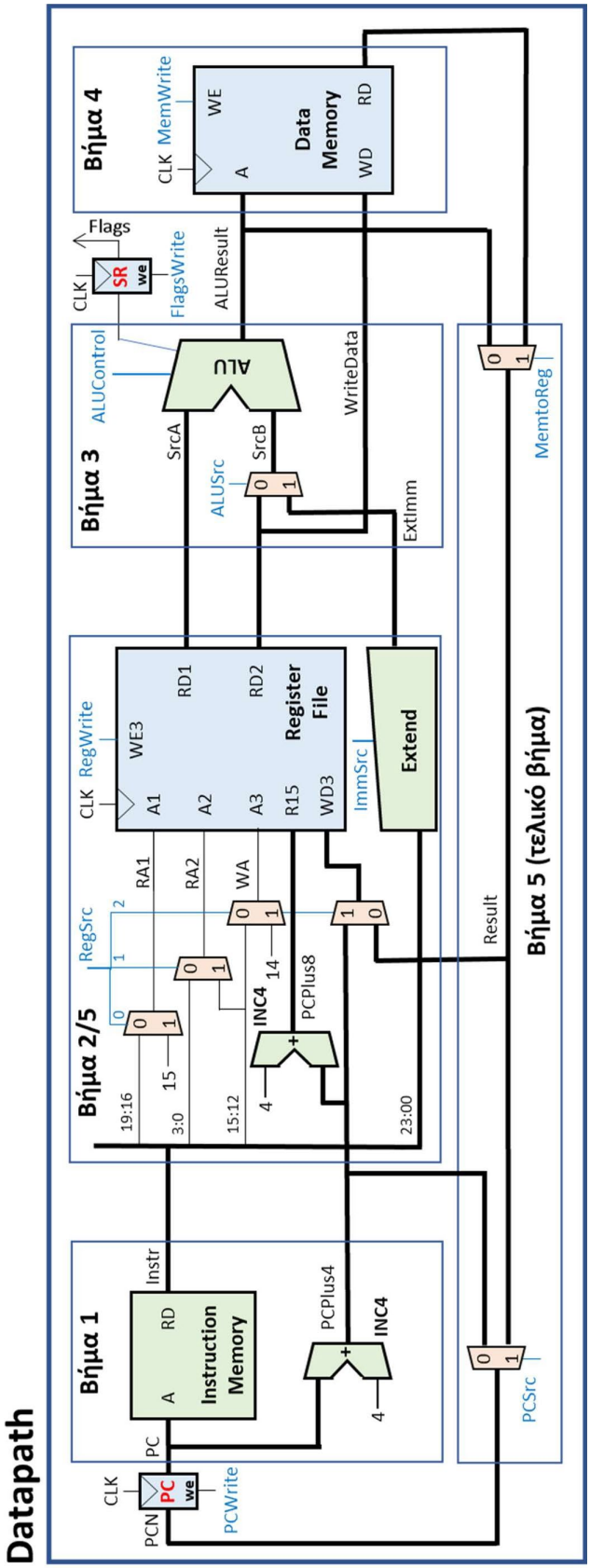
Στη συνέχεια, προστίθενται ανάμεσα στα διακριτά στοιχεία οι απαραίτητοι **μη αρχιτεκτονικοί καταχωρητές**, που επιτρέπουν να εκτελείται η εντολή σε πολλούς κύκλους (Σχήμα 2), ως εξής:

- Ανάμεσα στα διακριτά στοιχεία των βημάτων 1 και 2 προστίθενται οι καταχωρητές: **Instruction Register (IR)** στη θύρα ανάγνωσης (A/RD) της μνήμης εντολών με σήμα έγκρισης εγγραφής IRWrite (που ενεργοποιείται στο Βήμα 1) και **PCPlus4 Register (PCp4)** στην έξοδο PCPlus4 του αθροιστή.
- Ανάμεσα στα διακριτά στοιχεία των βημάτων 2 και 3 προστίθενται οι καταχωρητές: **Register A (A)** στη θύρα ανάγνωσης (A1/RD1) του αρχείου καταχωρητών, **Register B (B)** στη θύρα ανάγνωσης (A2/RD2) του αρχείου καταχωρητών, και **Register I (I)** στην έξοδο ExtImm της μονάδας Extend.
- Ανάμεσα στα διακριτά στοιχεία των βημάτων 3 και 4 προστίθενται οι καταχωρητές: **Memory Address Register (MA)** στην έξοδο ALUResult της μονάδας ALU με σήμα έγκρισης εγγραφής MAWrite (που ενεργοποιείται στο Βήμα 3) και **Memory Write Data Register (WD)** στην έξοδο του Register B (B), που μπορεί και να απαλειφθεί.
- Ανάμεσα στα διακριτά στοιχεία των βημάτων 3 και 5 προστίθεται ο καταχωρητής: **Register S (S)** στην έξοδο ALUResult της μονάδας ALU.
- Ανάμεσα στα διακριτά στοιχεία των βημάτων 4 και 5 προστίθεται ο καταχωρητής: **Memory Read Data Register (RD)** στη θύρα ανάγνωσης (A/RD) της μνήμης δεδομένων. Εάν η υλοποίηση της μνήμης δεδομένων γίνει με Block RAM, τότε ο καταχωρητής RD είναι ήδη ενσωματωμένος σε αυτή και δεν απαιτείται προσθήκη νέου καταχωρητή.

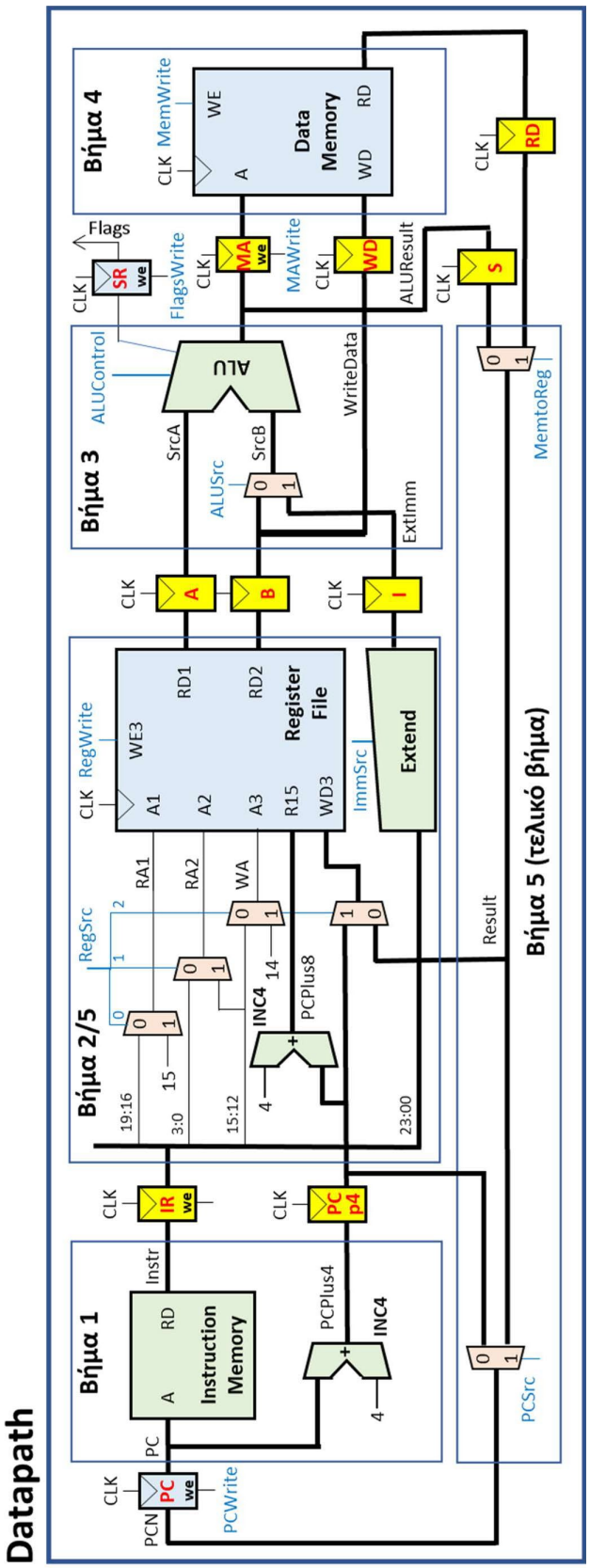
Τέλος, ο αρχιτεκτονικός καταχωρητής **program counter (PC)** τροποποιείται έτσι, ώστε να υποστηρίζει το σήμα έγκρισης εγγραφής PCWrite (που ενεργοποιείται στο Βήμα 5).

Στην παρούσα υλοποίηση, όπως αυτή αποτυπώνεται στο Σχήμα 2, οι εντολές διακλάδωσης (B, BL) εκτελούνται σε 4 κύκλους ρολογιού.

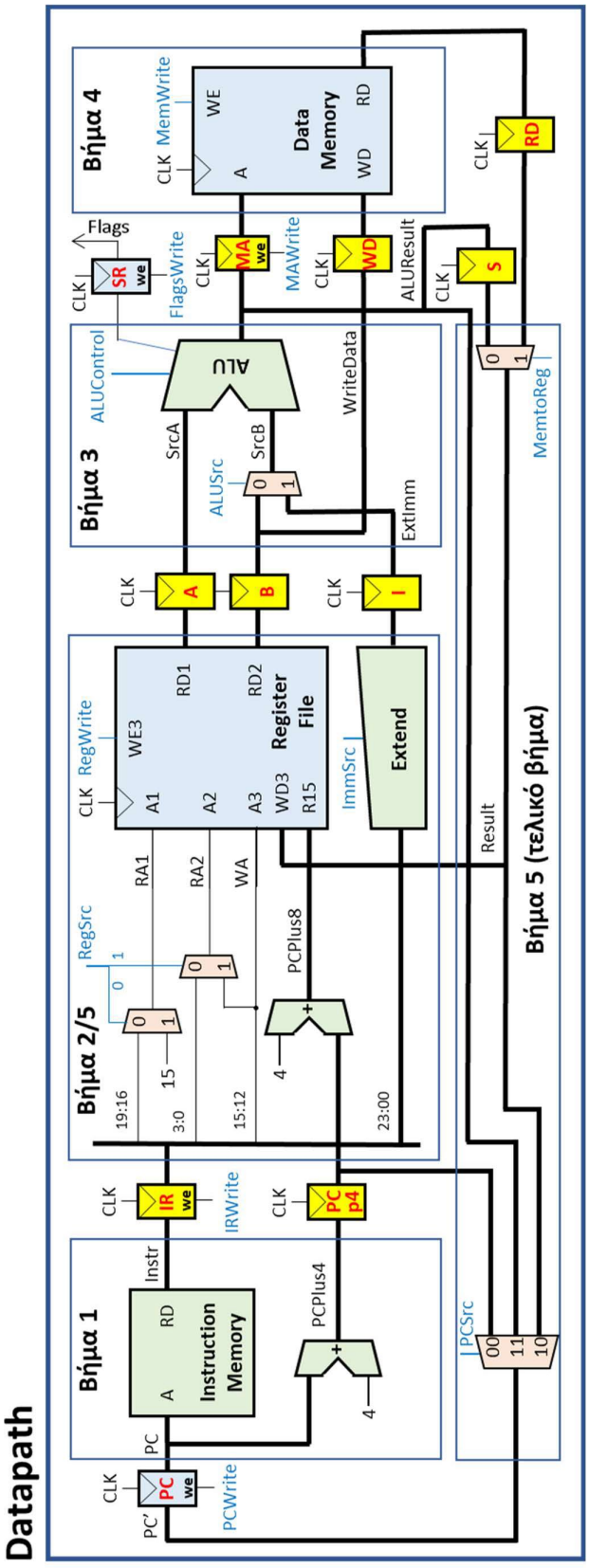
Εάν η έξοδος ALUResult της μονάδας ALU συνδεθεί απ' ευθείας με τον πολυπλέκτη επιλογής διεύθυνσης επόμενης εντολής, **παρακάμπτοντας τον Register S (S)**, τότε οι εντολές διακλάδωσης (B, BL) εκτελούνται μόνο σε 3 κύκλους ρολογιού. Η συγκεκριμένη παράκαμψη απαιτεί την τροποποίηση του πολυπλέκτη επιλογής διεύθυνσης επόμενης εντολής από 2 σε 1 σε 3 σε 1 με αντίστοιχη αύξηση του σήματος επιλογής PCSrc σε 2 bit, όπως φαίνεται στο Σχήμα 3. Ο νέος πολυπλέκτης επιλέγει μεταξύ της τιμής **PC + 4** (PCSrc[1:0] = 00) και του αποτελέσματος **Result**, που είτε αποθηκεύεται προσωρινά στους καταχωρητές **S** (PCSrc = 10, memtoReg = 0) ή **RD** (PCSrc = 10, memtoReg = 1), είτε προέρχεται απ' ευθείας από την έξοδο **ALUResult** της μονάδας ALU (PCSrc = 11).



Σχήμα 1: Διαδρομή δεδομένων ενός κύκλου.



Σχήμα 2: Διαδρομή δεδομένων πολλών κύκλων, όπου οι εντολές διακλάδωσης εκτελούνται σε 4 κύκλους ρολογιού.



Σχήμα 3: Διαδρομή δεδομένων πολλών κύκλων, όπου οι εντολές διακλάδωσης εκτελούνται σε 3 κύκλους ρολογιού.

Βήμα 4: Σχεδίαση της μονάδας ελέγχου

Αρχικά, στο κατώτερο ιεραρχικά επίπεδο της σχεδίασης περιγράφεται στη γλώσσα VHDL η συμπεριφορά των υπομονάδων που απαρτίζουν τη μονάδα ελέγχου. Στη συνέχεια, η σχεδίαση της μονάδας ελέγχου (**control**) του επεξεργαστή πολλών κύκλων ολοκληρώνεται με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom-up.

4-1. **Σχεδίαση του αποκωδικοποιητή εντολής (InstrDec).**

4-1.1. Συμπλήρωση του πίνακα αλήθειας του αποκωδικοποιητή εντολής (**InstrDec**).

Είσοδοι, τα πεδία op και funct της εντολής. Έξοδοι τα σήματα ελέγχου RegSrc[1:0], ALUSrc, ImmSrc, ALUControl[1:0] και MemtoReg, καθώς και το εσωτερικό σήμα NoWrite_in. Απαιτείται τροποποίηση, ώστε να καλύπτονται οι επιπλέον εντολές επεξεργασίας δεδομένων που υλοποιούνται, καθώς και η εντολή BL.

Δίδεται ως παράδειγμα ο πίνακας αλήθειας για τις εντολές ADD, SUB, CMP, AND και ORR.

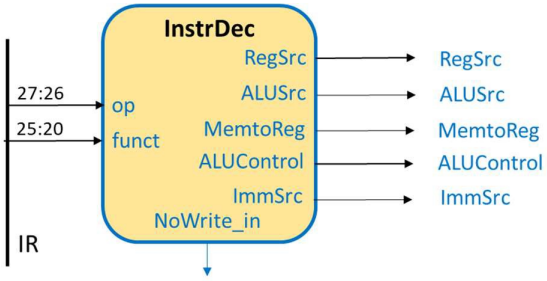
Εντολή	$Instr_{27:26}$ op	$Instr_{25:20}$ funct	Τύπος	RegSrc	ALUSrc	Imm Src	ALU Control	Memto Reg	NoWrite_in
ADD	00	1 0100 X	DP Imm	X0	1	0	00	0	0
ADD	00	0 0100 X	DP Reg	00	0	X	00	0	0
SUB	00	1 0010 X	DP Imm	X0	1	0	01	0	0
SUB	00	0 0010 X	DP Reg	00	0	X	01	0	0
CMP	00	1 1010 1	DP Imm	X0	1	0	01	X	1
CMP	00	0 1010 1	DP Reg	00	0	X	01	X	1
AND	00	1 0000 X	DP Imm	X0	1	0	10	0	0
AND	00	0 0000 X	DP Reg	00	0	X	10	0	0
ORR	00	1 1100 X	DP Imm	X0	1	0	11	0	0
ORR	00	0 1100 X	DP Reg	00	0	X	11	0	0

Δίδεται ως παράδειγμα ο πίνακας αλήθειας για τις εντολές LDR, STR και B.

Εντολή	$Instr_{27:26}$ op	$Instr_{25:20}$ funct	Τύπος	RegSrc	ALUSrc	Imm Src	ALU Control	Memto Reg	NoWrite_in
LDR	01	0 1100 1	M Imm +	X0	1	0	00	1	0
LDR	01	0 1000 1	M Imm –	X0	1	0	01	1	0
STR	01	0 1100 0	M Imm +	10	1	0	00	X	0
STR	01	0 1000 0	M Imm –	10	1	0	01	X	0
B	10	1 0XXX X	B Imm +	X1	1	1	00	0	0

4-1.2. Με βάση τον πίνακα αλήθειας γίνεται η σχεδίαση του αποκωδικοποιητή εντολής (**InstrDec**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case). Προσοχή! Η αξιοποίηση των αδιάφορων τιμών X στην έξοδο οδηγούν σε απλοποίηση του συνδυαστικού κυκλώματος.

4-1.3. Το σχηματικό διάγραμμα του αποκωδικοποιητή εντολής (**InstrDec**) φαίνεται στη συνέχεια.



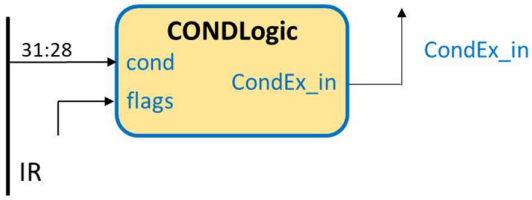
4-2. **Σχεδίαση της λογικής ελέγχου συνθήκης (CONDLogic).**

4-2.1. Συμπλήρωση του πίνακα αλήθειας της λογικής ελέγχου συνθήκης (**CONDLogic**), που ελέγχει εάν ικανοποιείται η συνθήκη που ορίζεται στο πεδίο cond της εντολής με βάση τις τρέχουσες τιμές των σημαιών N, Z, C, V (flags).

Είσοδοι, το πεδίο cond της εντολής και η έξοδος flags του καταχωρητή καταστάσεων. Έξοδος: το σήμα CondEx_in που εγκρίνει την εκτέλεση της εντολής, όταν παίρνει την τιμή 1.

4-2.2. Με βάση τον πίνακα της παραγράφου 1-1.4, όπου φαίνονται τα **μνημονικά συνθήκης** με τις **εξισώσεις Boole** των σημαιών που τις ικανοποιούν, γίνεται η σχεδίαση της λογικής ελέγχου συνθήκης (**CONDLogic**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case).

4-2.3. Το σχηματικό διάγραμμα της λογικής ελέγχου συνθήκης (**CONDLogic**) φαίνεται στη συνέχεια.



4-3. **Σχεδίαση της μηχανής πεπερασμένων καταστάσεων (FSM) τύπου Moore.**

4-3.1. Εύρεση των εισόδων και των εξόδων της μηχανής FSM.

Είσοδοι, το πεδίο op του IR (Instr27:26), το πεδίο S (για εντολές DP) ή L (για εντολές μνήμης) του IR (Instr20), το πεδίο Rd του IR (Instr15:12), καθώς και τα εσωτερικά σήματα NoWrite_in και CondEx_in που παράγονται κατά την αποκωδικοποίηση της εντολής (βήμα 2). Έξοδοι, τα σήματα έγκρισης εγγραφής (PCWrite, IRWrite, RegWrite, FlagsWrite, MAWrite και MemWrite) καθώς και το σήμα επιλογής διεύθυνσης επόμενης εντολής PCSrc[1:0].

4-3.2. Συσχέτιση των βημάτων εκτέλεσης των εντολών με τους απαιτούμενους κύκλους ρολογιού. Αν και κάθε βήμα εκτέλεσης της εντολής εκτελείται σε έναν κύκλο, κατά την εκτέλεση μίας εντολής πιθανώς να μην απαιτούνται όλα τα βήματα ή/και να εκτελούνται παράλληλα στον ίδιο κύκλο, όσα βήματα δεν δημιουργούν εξαρτήσεις.

Στον επόμενο πίνακα φαίνεται η συσχέτιση των βημάτων εκτέλεσης των εντολών με τους απαιτούμενους κύκλους ρολογιού για το σύνολο των εντολών. Σημειώνεται ότι εάν CondEx_in = 0, τότε τερματίζεται πρόωρα η εκτέλεση της εντολής με την εκτέλεση μόνο του βήματος 5 της εντολής (ενεργοποιείται το PCWrite, ενώ όλα τα υπόλοιπα σήματα έχουν την τιμή 0) έτσι, ώστε στον επόμενο κύκλο να εκτελεσθεί το Βήμα 1 της αμέσως επόμενης εντολής στη διεύθυνση PC+4.

Κύκλοι	LDR	STR	DP	CMP	B, BL	No executed
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3, 5	3, 5	5
4	4	4, 5	5			
5	5					

1. Προσκόμιση εντολής και υπολογισμός επόμενης διεύθυνσης (PC+4)
2. Αποκωδικοποίηση εντολής και ανάγνωση αρχείου καταχωρητών
3. Εκτέλεση πράξεων στη μονάδα ALU
4. Ανάγνωση ή εγγραφή στη μνήμη δεδομένων
5. Ετεροχρονισμένη εγγραφή στο αρχείο καταχωρητών και επιλογή της διεύθυνσης της επόμενης εντολής

4-3.3. Προσδιορισμός των απαιτούμενων καταστάσεων με βάση τις τιμές των σημάτων ελέγχου, ανά βήμα εκτέλεσης της εντολής. Για κάθε εντολή που υλοποιείται, δημιουργείται και ένας πίνακας προσδιορισμού καταστάσεων, ο οποίος συμπεριλαμβάνει την τρέχουσα κατάσταση, την επόμενη κατάσταση, τις συνθήκες στην είσοδο (εάν υπάρχουν) και τις αντίστοιχες τιμές των εξόδων της μηχανής FSM.

Ως ενδεικτικά παραδείγματα για την κατανόηση της διαδικασίας παρατίθενται οι αντίστοιχοι πίνακες των εντολών:

LDR:

Βήμα	Curr. State	Next State	op	S / L	Rd	NoWrite_in	Cond Ex_in	IR Write	Reg Write	MA Write	Mem Write	Flags Write	PC Src	PC Write
1	S0	S1	XX	X	X	X	X	1	0	0	0	0	00	0
2	S1	S2a	01	X	X	X	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	X	0	0	0	0	0	0	00	0
3	S2a	S3	XX	1	X	X	X	0	0	1	0	0	00	0
4	S3	S4a	XX	X	Not 15	X	X	0	0	0	0	0	00	0
4	S3	S4b	XX	X	15	X	X	0	0	0	0	0	00	0
5	S4a	S0	XX	X	XX	X	X	0	1	0	0	0	00	1
5	S4b	S0	XX	X	XX	X	X	0	0	0	0	0	10	1
5	S4c	S0	XX	X	XX	X	X	0	0	0	0	0	00	1

STR:

Βήμα	Curr. State	Next State	op	S / L	Rd	NoWrite_in	Cond Ex_in	IR Write	Reg Write	MA Write	Mem Write	Flags Write	PC Src	PC Write
1	S0	S1	XX	X	X	X	X	1	0	0	0	0	00	0
2	S1	S2a	01	X	X	X	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	X	0	0	0	0	0	0	00	0
3	S2a	S4d	XX	0	X	X	X	0	0	1	0	0	00	0
5	S4c	S0	XX	X	XX	X	X	0	0	0	0	0	00	1
4,5	S4d	S0	XX	X	XX	X	X	0	0	0	1	0	00	1

Επεξεργασίας δεδομένων (DP):

Βήμα	Curr. State	Next State	op	S / L	Rd	NoWrite_in	Cond Ex_in	IR Write	Reg Write	MA Write	Mem Write	Flags Write	PC Src	PC Write
1	S0	S1	XX	X	X	X	X	1	0	0	0	0	00	0
2	S1	S2b	00	X	X	0	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	X	0	0	0	0	0	0	00	0
3	S2b	S4a	XX	0	Not 15	X	X	0	0	0	0	0	00	0
3	S2b	S4b	XX	0	15	X	X	0	0	0	0	0	00	0
3	S2b	S4e	XX	1	Not 15	X	X	0	0	0	0	0	00	0
3	S2b	S4f	XX	1	15	X	X	0	0	0	0	0	00	0
5	S4a	S0	XX	X	XX	X	X	0	1	0	0	0	00	1
5	S4b	S0	XX	X	XX	X	X	0	0	0	0	0	10	1
5	S4c	S0	XX	X	XX	X	X	0	0	0	0	0	00	1
5	S4e	S0	XX	X	XX	X	X	0	1	0	0	1	00	1
5	S4f	S0	XX	X	XX	X	X	0	0	0	0	1	10	1

CMP:

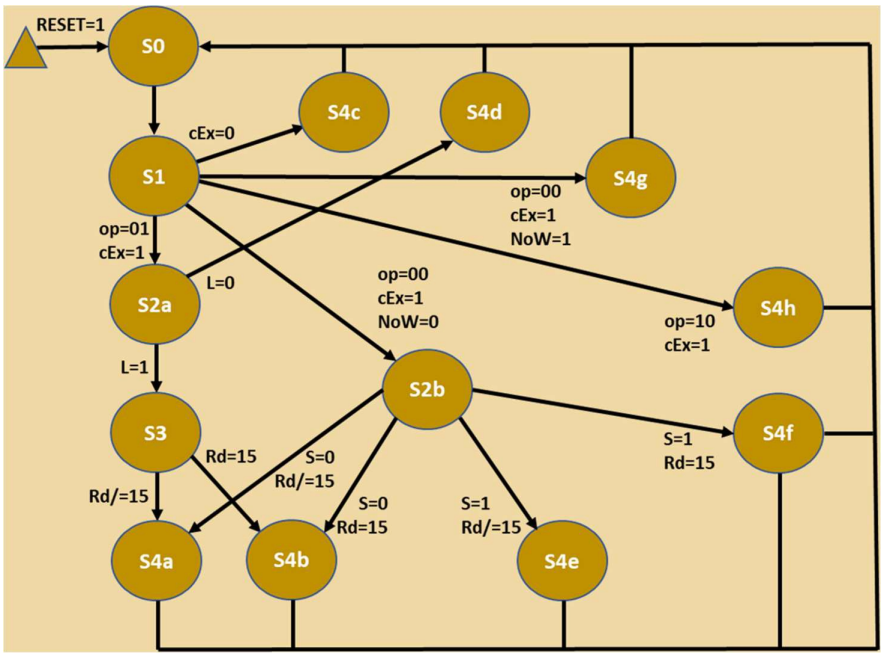
Βήμα	Curr. State	Next State	op	S / L	Rd	NoWrite_in	Cond Ex_in	IR Write	Reg Write	MA Write	Mem Write	Flags Write	PC Src	PC Write
1	S0	S1	XX	X	X	X	X	1	0	0	0	0	00	0
2	S1	S4g	00	X	X	1	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	X	0	0	0	0	0	0	00	0
5	S4c	S0	XX	X	XX	X	X	0	0	0	0	0	00	1
3,5	S4g	S0	XX	X	XX	X	X	0	0	0	0	1	00	1

B:

Βήμα	Curr. State	Next State	op	S / L	Rd	NoWrite_in	Cond Ex_in	IR Write	Reg Write	MA Write	Mem Write	Flags Write	PC Src	PC Write
1	S0	S1	XX	X	X	X	X	1	0	0	0	0	00	0
2	S1	S4h	10	X	X	X	1	0	0	0	0	0	00	0
2	S1	S4c	XX	X	X	X	0	0	0	0	0	0	00	0
5	S4c	S0	XX	X	XX	X	X	0	0	0	0	0	00	1
3,5	S4h	S0	XX	X	XX	X	X	0	0	0	0	0	11	1

4-3.4. Δημιουργία του διαγράμματος μεταβολής κατάστασης. Απαιτείται προσοχή στον προσδιορισμό των συνθηκών εισόδων που καθορίζουν τη μετάβαση από τη μία κατάσταση στην άλλη (δεν απαιτούνται όλες οι είσοδοι).

Ενδεικτικά παρατίθεται το διάγραμμα μεταβολής κατάστασης που αφορά στις εντολές που αναφέρονται στο προηγούμενο βήμα.

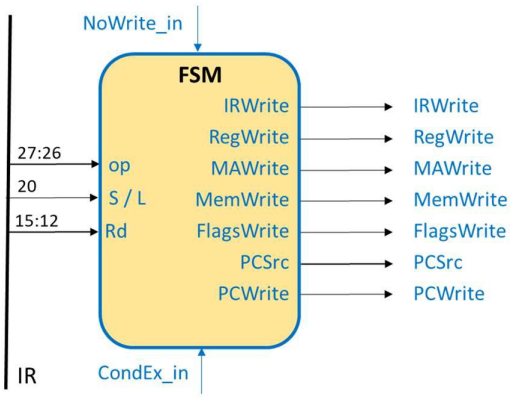


Συντομογραφίες: CondEx_in = cEx, NoWrite_in = NoW.

4-3.5. Με βάση το διάγραμμα μεταβολής κατάστασης γίνεται η σχεδίαση της μηχανής πεπερασμένων καταστάσεων **(FSM)** τύπου Moore σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case).

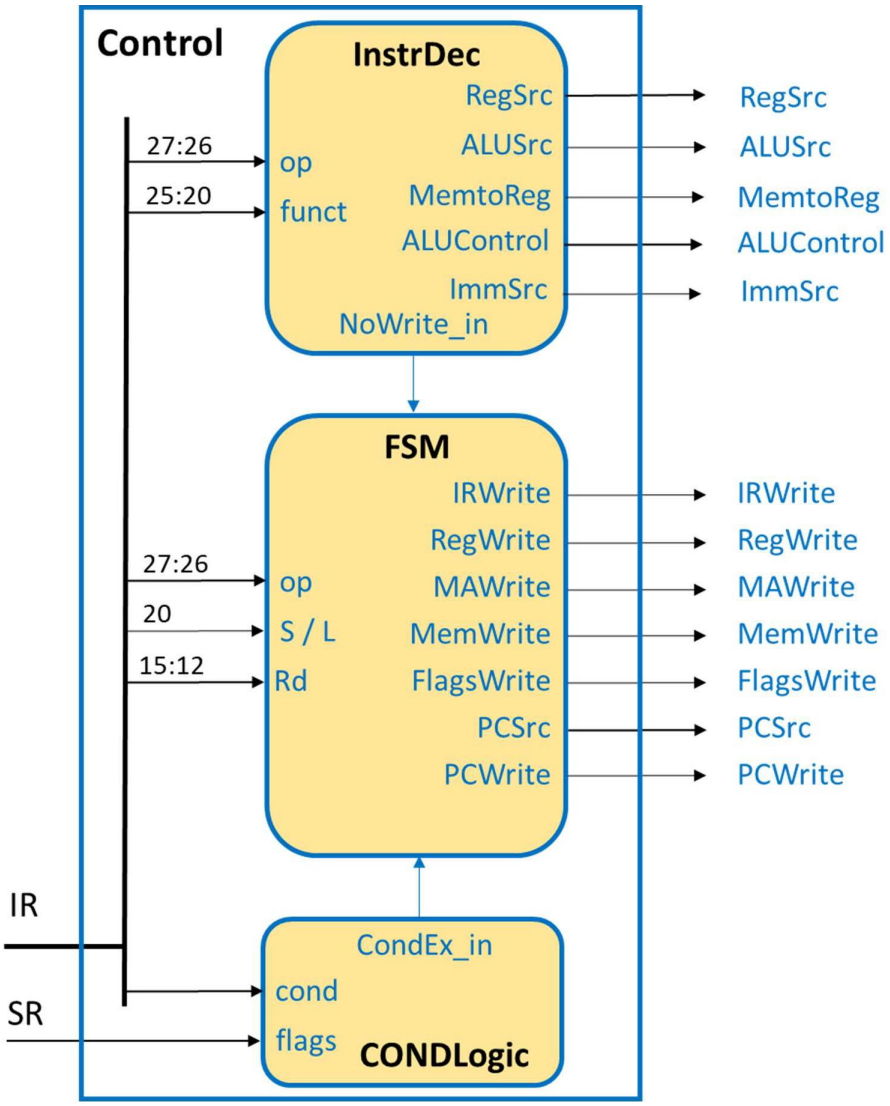
Σημειώνεται ότι παρουσιάζει ενδιαφέρον και η σχεδίαση της μηχανής πεπερασμένων καταστάσεων **(FSM)** τύπου Mealy, η οποία έχει ως αποτέλεσμα στη μείωση των κύκλων της εκτέλεσης των εντολών και συνεπώς στην αύξηση των επιδόσεων του επεξεργαστή πολλών κύκλων.

4-3.6. Το σχηματικό διάγραμμα της μηχανής πεπερασμένων καταστάσεων **(FSM)** φαίνεται στη συνέχεια.



4-4. **Σχεδίαση της μονάδας ελέγχου (control)**

Η σχεδίαση της μονάδας ελέγχου (**control**) του επεξεργαστή πολλών κύκλων ολοκληρώνεται με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom-up, όπως αυτή αποτυπώνεται στο ακόλουθο σχηματικό διάγραμμα.



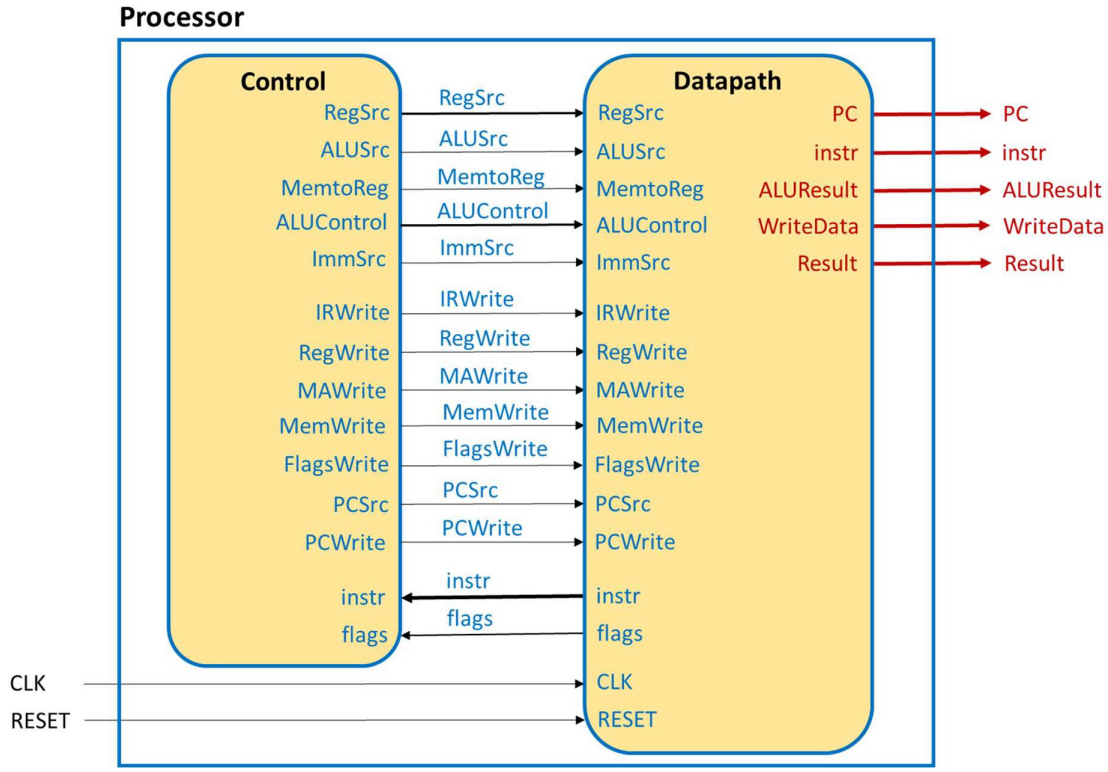
Βήμα 5: Σχεδίαση του επεξεργαστή (processor)

Στο υψηλότερο ιεραρχικά επίπεδο σχεδιάζεται ο επεξεργαστής (**processor**) με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom-up, όπως αυτή αποτυπώνεται στο ακόλουθο σχηματικό διάγραμμα.

Στο πλαίσιο του Project 2, επιλέγεται ο επεξεργαστής να έχει ως εξόδους τις αρτηρίες **PC** και **Instr**, που σχετίζονται με τη μνήμη εντολών, και τις αρτηρίες **ALUResult**, **WriteData** και **Result**, που σχετίζονται με τη μνήμη δεδομένων και τη μονάδα ALU. Οι είσοδοι του επεξεργαστή είναι τα σήματα **CLK** και **RESET**.

Πριν την υλοποίηση θα πρέπει να φορτωθεί το απαραίτητο πρόγραμμα στη μνήμη ROM που να συμπεριλαμβάνει όλες τις εντολές που υλοποιούνται. Προσοχή! Θα πρέπει να μελετηθούν προσεκτικά οποιεσδήποτε ανεπιθύμητες απλοποιήσεις που ενδεχομένως να γίνουν στο στάδιο της υλοποίησης.

Η διασύνδεση της διαδρομής δεδομένων (**datapath**) και της μονάδας ελέγχου (**control**) φαίνεται στο επόμενο σχηματικό διάγραμμα στο επίπεδο του επεξεργαστή (**processor**).



Βήμα 6: Επαλήθευση της ορθής σχεδίασης του επεξεργαστή (processor)

Η επαλήθευση της ορθής σχεδίασης του επεξεργαστή (**processor**) που βασίζεται στην προσομοίωση (simulation based verification) εφαρμόζεται σε επιλεγμένες υπομονάδες της διαδρομής δεδομένων, όπως η μονάδα ALU (**ALU**) και το αρχείο καταχωρητών (**RF**), στη μονάδα ελέγχου (**control**) και στον επεξεργαστή (**processor**) σαν ολότητα.

6-1. **Επαλήθευση της ορθής σχεδίασης της μονάδας ALU (ALU).**

6-1.1. Ακολουθήστε όλα τα Βήματα 1-5 της ιεραρχικής σχεδίασης της οντότητας της μονάδας ALU (**ALU**), όπως αυτά περιγράφονται στον οδηγό χρήσης των βασικών λειτουργειών του **Vivado IDE** δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**ALU_TB**), που εκτελεί όλες τις πράξεις. Χρησιμοποιείστε ως βάση τον αθροιστή με καταχωρητές εισόδου και εξόδου.

6-2. **Επαλήθευση της ορθής σχεδίασης του αρχείου καταχωρητών (RF).**

6-2.1. Ακολουθήστε όλα τα Βήματα 1-5 της ιεραρχικής σχεδίασης της οντότητας του αρχείου καταχωρητών (**RF**), όπως αυτά περιγράφονται στον οδηγό χρήσης των βασικών λειτουργειών του **Vivado IDE** δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**RF_TB**), που γράφει στους καταχωρητές R0-R14 (δεν ορίζεται εγγραφή στον καταχωρητή R15) και διαβάζει όλους τους καταχωρητές R0-R15 (και τον καταχωρητή R15 – απαιτείται κατάλληλη τιμή στην είσοδο R15). Χρησιμοποιείστε ως βάση τον αθροιστή με καταχωρητές εισόδου και εξόδου.

6-3. **Επαλήθευση της ορθής σχεδίασης της μονάδας ελέγχου (control).**

6-3.1. Ακολουθήστε όλα τα Βήματα 1-5 της ιεραρχικής σχεδίασης της οντότητας της μονάδας ελέγχου (**control**), όπως αυτά περιγράφονται στον οδηγό χρήσης των βασικών λειτουργειών του **Vivado IDE** δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**control_TB**), που επιβεβαιώνει την ορθή λειτουργία όλων των εντολών και όλων των διακλαδώσεων των εντολών case και if του κώδικα στη γλώσσα VHDL. Χρησιμοποιείστε ως βάση τη μηχανή πεπερασμένων καταστάσεων (FSM).

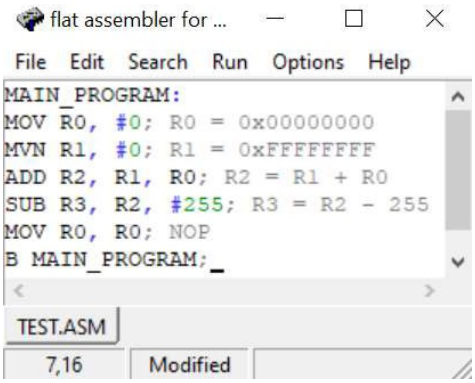
6-4. **Επαλήθευση της ορθής σχεδίασης του επεξεργαστή (processor).**

6-4.1. Ακολουθήστε όλα τα Βήματα 1-5 της ιεραρχικής σχεδίασης της οντότητας του επεξεργαστή (**processor**), όπως αυτά περιγράφονται στον οδηγό χρήσης των βασικών λειτουργειών του Vivado IDE δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**processor_TB**). Προσοχή! Το πλήθος των κύκλων ρολογιού που θα τρέξει ο επεξεργαστής πρέπει να αντιστοιχεί στο πλήθος των κύκλων που απαιτούνται για την εκτέλεση όλων των εντολών που είναι αποθηκευμένες στη μνήμη ROM. Σε κάθε περίπτωση, η τελευταία εντολή του προγράμματός σας επαναφέρει τη συνέχεια της εκτέλεσης του προγράμματος στην πρώτη εντολή του. Να γίνει χρήση και της ψευδοεντολής NOP.

6-4.2. Δημιουργήστε ένα πρόγραμμα σε συμβολική γλώσσα της αρχιτεκτονικής ARM, που να συμπεριλαμβάνει όλες τις εντολές που υλοποιείτε και ενεργοποιεί όλες τις πιθανές ροές δεδομένων και λειτουργίες στη διαδρομή δεδομένων.

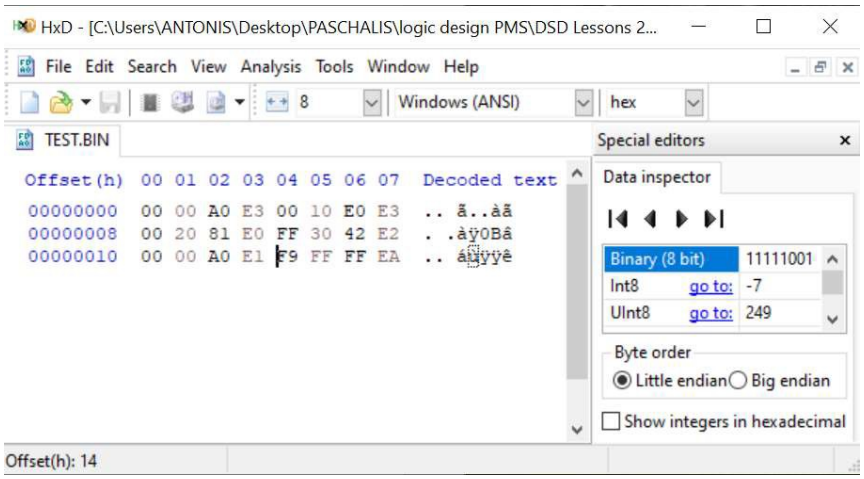
Για τη δημιουργία του προγράμματος μπορείτε να χρησιμοποιήσετε έναν ελεύθερο συμβολομεταφραστή (assembler) της αρχιτεκτονικής ARM, όπως είναι ο **FASMARM** (https://arm.flatassembler.net). Ο FASMARM στην πραγματικότητα είναι cross assembler υπό την έννοια ότι παράγει μεν κώδικα σε γλώσσα μηχανής ARM, ο ίδιος όμως δεν τρέχει σε επεξεργαστή αρχιτεκτονικής ARM αλλά αρχιτεκτονικής X86/X64.

Εκτελέστε την εφαρμογή **FASMWARM.EXE**. Εμφανίζεται το παράθυρο του editor στο οποίο γράφετε τον κώδικα ARM, όπως στο παράδειγμα. Αναφορικά με τα labels στον κώδικα, ο FASMARM έχει μία ιδιαιτερότητα: μετά από κάθε label στην αρχή μίας γραμμής (όχι αν το label αποτελεί μέρος της εντολής, όπως η B) πρέπει να βάζετε το σύμβολο «:». Ο assembler θεωρεί συντακτικό λάθος την απουσία του. Αποθηκεύστε και ονομάστε το πηγαίο αρχείο στη συμβολική γλώσσα της αρχιτεκτονικής ARM επιλέγοντας **file->save as** και δηλώνοντας το όνομα (**TEST.ASM**).



6-4.3. Μετατρέψτε το πρόγραμμά σας σε γλώσσα μηχανής, επιλέγοντας **Run->Compile**. Εάν ο assembler εντοπίσει συντακτικά λάθη, η συμβολομετάφραση αποτυγχάνει και εμφανίζεται ένα αναδυόμενο παράθυρο που επισημαίνει το πρώτο συντακτικό σφάλμα που συναντήθηκε. Μετά την αποσφαλμάτωση του πηγαίου αρχείου .ASM παράγεται το αντίστοιχο αρχείο σε γλώσσα μηχανής που είναι binary (**TEST.BIN**). Φαίνεται στο ίδιο directory με το πηγαίο αρχείο TEST.ASM.

6-4.4. Ανοίξτε το binary αρχείο με έναν ελεύθερο hex editor, όπως είναι ο **HxD**, αφού πρώτα τον εγκαταστήσετε στον υπολογιστή σας.



Κάθε 4 συνεχόμενα byte (1 byte = 2 δεκαεξαδικά ψηφία) αποτελούν μία εντολή, ξεκινώντας από την πρώτη κατά σειρά έως και την τελευταία, όπως αυτές εμφανίζονται στο πηγαίο αρχείο **TEST.ASM**.

6-4.5. Αντιγράψτε τις εντολές σε γλώσσα μηχανής στο αρχείο **ROM_array.vhd** που περιγράφει τη μνήμη εντολών, ως μνήμη ROM. Αντιγράψτε μία-μία τις τετράδες των byte στις κατάλληλες θέσεις του ROM_array ξεκινώντας από την πρώτη θέση. Θα χρειαστεί να αφαιρέσετε με το χέρι τα ενδιάμεσα κενά μεταξύ των διαδοχικών byte και να **αντιστρέψετε** τη σειρά τους, αφού το περισσότερο σημαντικό byte (most significant byte, **MSB**) της λέξης εντολών βρίσκεται στα **αριστερά**, ενώ το λιγότερο σημαντικό byte (least significant byte, **LSB**) της λέξης εντολών βρίσκεται στα **δεξιά**. Προσοχή! Στην περίπτωση που οι εντολές είναι λιγότερες από τη συνολική χωρητικότητα της μνήμης ROM θα πρέπει να γεμίσετε τις υπόλοιπες θέσεις με X"00000000". Για παράδειγμα, για μία μνήμη ROM χωρητικότητας 16 εντολών των 32 bit (N = 4, M = 32) ο κώδικας σε γλώσσα VHDL διαμορφώνεται ως εξής:

ROM_array.vhd

C:/Users/ANTONIS/Xilinx/Projects/DSD/Examples/Examples.srcs/sources_1/new/ROM_array.vhd

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity ROM_array is
35    generic (
36      N : positive := 4;    -- address length
37      M : positive := 32);  -- data word length
38    port (
39      ADDR:      in STD_LOGIC_VECTOR (N-1 downto 0);
40      DATA_OUT: out STD_LOGIC_VECTOR (M-1 downto 0));
41  end ROM_array;
42
43  architecture Behavioral of ROM_array is
44    type ROM_array is array (2**N-1 downto 0)
45      of STD_LOGIC_VECTOR (M-1 downto 0);
46    constant ROM : ROM_array := (
47      X"E3A00000", X"E3E01000", X"E0812000", X"E24230FF",
48      X"E1A00000", X"EAFFFFF9", X"00000000", X"00000000",
49      X"00000000", X"00000000", X"00000000", X"00000000",
50      X"00000000", X"00000000", X"00000000", X"00000000");
51  begin
52    DATA_OUT <= ROM(to_integer(unsigned(ADDR)));
53  end Behavioral;
```

6-4.6. Ολοκληρώστε την επαλήθευση της ορθής σχεδίασης του επεξεργαστή (**processor**) συγκρίνοντας για κάθε εντολή, που εκτελείται σε πολλούς κύκλους, τις τιμές των αρτηριών **PC** και **Instr**, που σχετίζονται με τη μνήμη εντολών, και των αρτηριών **ALUResult**, **WriteData** και **Result**, που σχετίζονται με τη μνήμη δεδομένων και τη μονάδα ALU, που προκύπτουν μετά την προσομοίωση σε συγκεκριμένους κύκλους ρολογιού, με τις αντίστοιχες αναμενόμενες τιμές που φαίνονται στον ακόλουθο πίνακα επαλήθευσης ορθής λειτουργίας του επεξεργαστή και προκύπτουν μετά από μελέτη των εντολών που έχουν αποθηκευτεί στη μνήμη εντολών και έχουν υλοποιηθεί στον επεξεργαστή. Επιπλέον μελετήστε και τιμές των σημαιών N, Z, C, V, ως εσωτερικά σήματα.

α.α.	Εντολή	PC	Instr	ALUResult	WriteData	Result	Σχόλια
1	MOV R0, #0	0x00	E3A00000	00000000		00000000	
2	MVN R1, #0	0x04	E3E01000	FFFFFFFF		FFFFFFFF	
3	ADD R2, R1, R0	0x08	E0812000	FFFFFFFF	00000000	FFFFFFFF	
4	SUB R3, R2, #255	0x0C	E24230FF	FFFFFF00		FFFFFF00	
5	MOV R0, R0	0x10	E1A00000	00000000	00000000	00000000	
6	B MAIN_PROGRAM	0x14	EAFFFFF9	00000000		00000000	PC’ = 0x00

Βήμα 7: Παράδοση τεχνικής αναφοράς της σχεδίασης του επεξεργαστή

Μετά την ολοκλήρωση της σχεδίασης του επεξεργαστή (**processor**) παραδίδετε το project συνοδευόμενο από μία τεχνική αναφορά η οποία συμπεριλαμβάνει τα ακόλουθα:

7-1. **Περιγραφή των στοιχείων και της δομής του επεξεργαστή.**

7-1.1. Περιγράψτε το σύνολο των εντολών που έχετε υλοποιήσει.

7-1.2. Παραθέστε το σχηματικό διάγραμμα στο επίπεδο RTL (elaborated design) της διαδρομής δεδομένων (datapath) του επεξεργαστή. Περιγράψτε τα στοιχεία (components) που χρησιμοποιείτε (λειτουργία, τα δεδομένα ανταλλάσσουν με τα component με τα οποία αλληλοεπιδρούν, αναφέρετε τους μη αρχιτεκτονικούς καταχωρητές).

7-1.3. Περιγράψτε όλες τις υπομονάδες της μονάδας ελέγχου (λειτουργία, τα δεδομένα ανταλλάσσουν με τα component με τα οποία αλληλοεπιδρούν) συμπληρώνοντας τους αντίστοιχους πίνακες αλήθειας. Εξηγήστε για κάθε εντολή πως προκύπτουν οι αντίστοιχες τιμές στου πίνακες αληθείας. Η ανάλυση θα γίνει ανά πίνακα αληθείας.

7-1.4. Περιγράψτε τη δομή της μονάδας ελέγχου (**control**) του επεξεργαστή στο ανώτερο ιεραρχικό επίπεδο και σχολιάστε τα προκύπτοντα σχηματικά διαγράμματα στο επίπεδο RTL (elaborated design).

7-1.5. Περιγράψτε τη μηχανή πεπερασμένων καταστάσεων (FSM) της μονάδας ελέγχου συμπληρώνοντας τους αντίστοιχους πίνακες προσδιορισμού κατάστασης για όλες τις εντολές που υλοποιείτε.

7-1.6. Περιγράψτε τη δομή του επεξεργαστή (**processor**) στο ανώτερο ιεραρχικά επίπεδο με στοιχεία τη διαδρομή δεδομένων και τη μονάδα ελέγχου ελέγχου (λειτουργία, τα δεδομένα ανταλλάσσουν με τα δύο component). Παραθέστε το προκύπτον σχηματικό διάγραμμα στο επίπεδο RTL (elaborated design).

7-1.7. Βρείτε τη μέγιστη συχνότητα λειτουργίας στο επίπεδο του επεξεργαστή (**processor**), προσδιορίζοντας τη χειρότερη κρίσιμη διαδρομή και τη χειρότερη σύντομη διαδρομή.

7-2. **Επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή (processor).**

7-2.1. Δημιουργείστε κατάλληλο πρόγραμμα σε συμβολική γλώσσα αρχιτεκτονικής ARM, που συμπεριλαμβάνει όλες τις υλοποιημένες εντολές και ενεργοποιεί όλες (κατά το δυνατόν) τις πιθανές ροές δεδομένων και λειτουργίες στη διαδρομή δεδομένων. Τεκμηριώστε γιατί το πρόγραμμα σας σε συμβολική γλώσσα επαληθεύει την ορθή σχεδίαση και λειτουργία του επεξεργαστή (**processor**) (δηλαδή, ποιο τμήμα του προγράμματος επαληθεύει ποιες εντολές και με ποιον τρόπο).

7.2.2. Επιπλέον για το αρχείο καταχωρητών και τη μονάδα ALU, παραθέστε τα απαραίτητα προγράμματα δοκιμής και τα διαγράμματα χρονισμού των προσομοιώσεων του behavioral model και του post-implementation model (μόνο χρονική προσομοίωση) και τεκμηριώστε την ορθή τους σχεδίαση και λειτουργία.

7.2.3 Παραθέστε τα διαγράμματα χρονισμού των προσομοιώσεων του behavioral model και του post-implementation model (μόνο χρονική προσομοίωση) για τον

επεξεργαστή και τεκμηριώστε την ορθή του σχεδίαση και λειτουργία.

7-3. **Ανάλυση των αποτελεσμάτων της σύνθεσης και της υλοποίησης του επεξεργαστή (processor).**

7-3.1. Αναλύστε τα αποτελέσματα της σύνθεσης και της υλοποίησης του επεξεργαστή (**processor**) μελετώντας το project summary και το report utilization. Είναι οι χρησιμοποιούμενοι πόροι αυτοί που περιμένατε; Τι κατανάλωση έχει το κύκλωμα;

7-4. **Περιγράψτε τη λειτουργία της εντολής ROR**

Συμβουλευτείτε το αρχείο από το eclass που αναφέρεται στην Μικροαρχιτεκτονική (Κεφάλαιο 7) και

7-4.1 Περιγράψτε συνοπτικά αλλά περιεκτικά τη λειτουργία της εντολής ROR. Ενδεικτικά και μόνο αναφέρονται οι σελίδες 20-30 από τις διαφάνειες του κεφαλαίου 7.

7-4.2 Αναφέρετε την αλληλεπίδραση με το Control Unit (ποια σήματα επηρεάζει, ποιες τιμές τους αποδίδονται).

Προσοχή. Η τεχνική αναφορά θα πρέπει να ακολουθεί πλήρως τη δομή που περιγράφεται στις παραγράφους 7.1. έως και 7.4. ΔΕΝ πρέπει να συγχωνεύσετε τις απαντήσεις σας, για κάθε ενότητα θα πρέπει να υπάρχει διακριτή ανάλυση. Αναμένουμε λοιπόν μια αναφορά 4 κεφαλαίων (χωρίς την εισαγωγή ή πιθανά παραρτήματα). Στον κώδικα αναμένεται εκτός των άλλων να υπάρχει testbench για κάθε σημαντικό module που έχετε δημιουργήσει (CPU, Datapath, Control Unit, ALU, Register File)