

ΥΣ03 Σχεδίαση Ψηφιακών Συστημάτων

Project 1: Μεθοδολογία σχεδίασης επεξεργαστή ενός κύκλου

Κρανίτης Νεκτάριος, Βασιλόπουλος Διονύσης

Χειμερινό Εξάμηνο
2024 – 2025

Project 1

Πίνακας Περιεχομένων

Εισαγωγικά στοιχεία.....	5
0-1. Γενική ροή σχεδίασης του επεξεργαστή.....	5
Βήμα 1: Ανάλυση των απαιτήσεων του επεξεργαστή.....	6
1-1. Απαιτήσεις που απορρέουν από την εφαρμογή της μικροαρχιτεκτονικής ARM	6
1-2. Τα 5 βήματα εκτέλεσης της εντολής στη μικροαρχιτεκτονική ARM	7
1-3. Το σύνολο εντολών που πρόκειται να υλοποιηθεί.....	8
Βήμα 2: Σχεδίαση των ψηφιακών δομικών στοιχείων της διαδρομής δεδομένων.....	13
2-1. Προσκόμιση εντολής και υπολογισμός της επόμενης διεύθυνσης (PC+4)	13
2-2. Ανάγνωση 2 καταχωρητών από το αρχείο καταχωρητών (συμπεριλαμβάνεται και ο μετρητής προγράμματος) και επέκταση προσήμου/μηδενός.....	13
2-3. Εκτέλεση πράξεων στη μονάδα ALU.....	14
2-4. Ανάγνωση ή εγγραφή στη μνήμη δεδομένων.....	14
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	14
Βήμα 3: Σχεδίαση της διαδρομής δεδομένων (datapath).....	15
Βήμα 4: Σχεδίαση της μονάδας ελέγχου.....	17
4-1. Σχεδίαση του αποκωδικοποιητή εντολής (InstrDec).....	17
4-2. Σχεδίαση του αποκωδικοποιητή των σημάτων έγκρισης εγγραφής (WELogic)	18
4-3. Σχεδίαση της λογικής επιλογής διεύθυνσης επόμενης εντολής (PCLogic)	19
4-4. Σχεδίαση της λογικής ελέγχου συνθήκης (CONDLogic)	19
4-5. Σχεδίαση της μονάδας ελέγχου (control)	20
Βήμα 5: Σχεδίαση του επεξεργαστή (processor)	21
Βήμα 6: Επαλήθευση της ορθής σχεδίασης του επεξεργαστή (processor).....	22
6-1. Επαλήθευση της ορθής σχεδίασης της μονάδας ALU (ALU)	22
6-2. Επαλήθευση της ορθής σχεδίασης του αρχείου καταχωρητών (RF)	22
6-3. Επαλήθευση της ορθής σχεδίασης της μονάδας ελέγχου (control).....	22
6-4. Επαλήθευση της ορθής σχεδίασης του επεξεργαστή (processor)	22
Βήμα 7: Παράδοση τεχνικής αναφοράς της σχεδίασης του επεξεργαστή.....	25
7-1. Περιγραφή των στοιχείων και της δομής του επεξεργαστή.....	25
7-2. Επαλήθευση της ορθής σχεδίασης και λειτουργίας του επεξεργαστή (processor).26	26
7-3. Ανάλυση των αποτελεσμάτων της σύνθεσης και της υλοποίησης του επεξεργαστή (processor)	26

Project 1

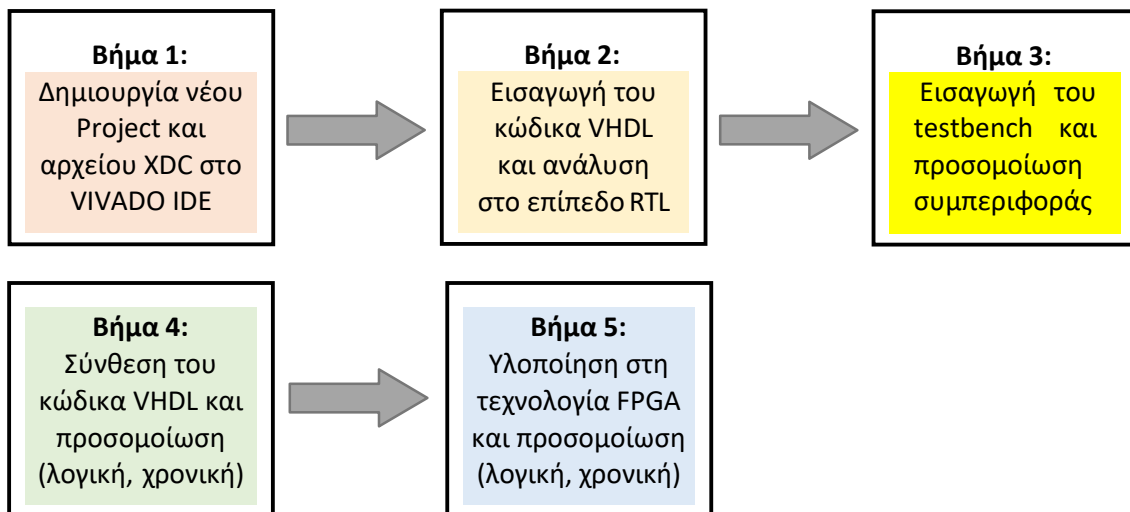
Εισαγωγικά στοιχεία

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

Αρχικά, αναλύουμε τις απαιτήσεις του επεξεργαστή που απορρέουν από την εφαρμογή της μικροαρχιτεκτονικής ARM, από τα 5 βήματα εκτέλεσης της εντολής στη μικροαρχιτεκτονική ARM (με τις αντίστοιχες λειτουργίες που εκτελούνται ανά βήμα) και από το σύνολο εντολών που πρόκειται να υλοποιηθεί. Στη συνέχεια, προσδιορίζουμε τα ψηφιακά δομικά στοιχεία της διαδρομής δεδομένων που απαιτούνται σε κάθε βήμα εκτέλεσης της εντολής και προχωράμε στη σχεδίαση της διαδρομής δεδομένων του επεξεργαστή ενός κύκλου 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	$\bar{Z}N\oplus\bar{V}$
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 ενός κύκλου απαρτίζεται από δύο διακριτές μονάδες που αλληλοεπιδρούν μεταξύ τους:

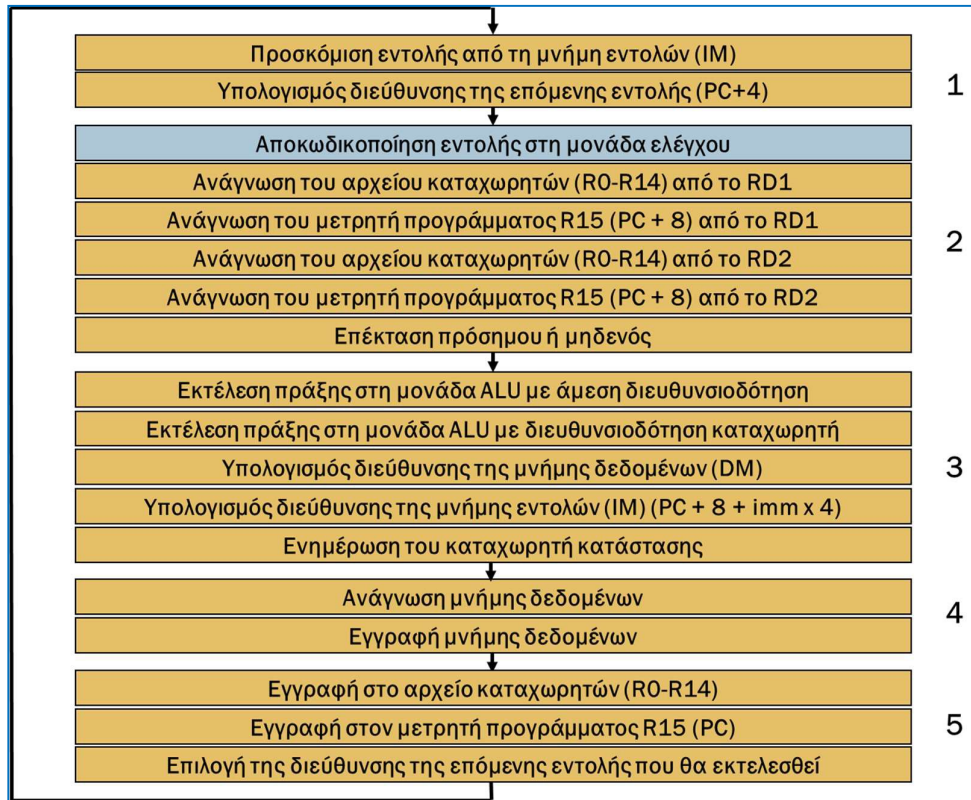
- τη **διαδρομή δεδομένων** (datapath) των 32 bit, που εκτελεί λειτουργίες με λέξεις δεδομένων και περιέχει υπομονάδες όπως: διακριτές μνήμες εντολών (ROM 64 x 32) και δεδομένων (RAM 32 x 32) μεγέθους 32 bit (αρχιτεκτονική Harvard), αρχιτεκτονικούς καταχωρητές, μονάδα ALU για την εκτέλεση των πράξεων 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), ως συνδυαστική λογική, που αποκωδικοποιεί την εντολή και για κάθε εντολή υπό συνθήκη παράγει τα κατάλληλα σήματα ελέγχου των υπομονάδων της διαδρομής δεδομένων, λαμβάνοντας υπόψη τις τιμές των σημαίων (N, Z, C, V).

1-1.6. Στη μικροαρχιτεκτονική ARM του επεξεργαστή ενός κύκλου ολόκληρη η εντολή εκτελείται σε **έναν κύκλο ρολογιού** (CLK), όπου η περίοδος του CLK προσδιορίζεται από την εντολή **LDR** που έχει τη μεγαλύτερη καθυστέρηση διάδοσης από όλες τις εντολές του επεξεργαστή.

1-2. Τα 5 βήματα 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.



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)

Τελεστές 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)

Τελεστές που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστές στον καταχωρητή προέλευσης Rn,
- ο τελεστές στον καταχωρητή προέλευσης Rd,
- ο μη προσημασμένος άμεσος τελεστές 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 σημαίων $SR = (N, Z, C, V)$ (S = 1)

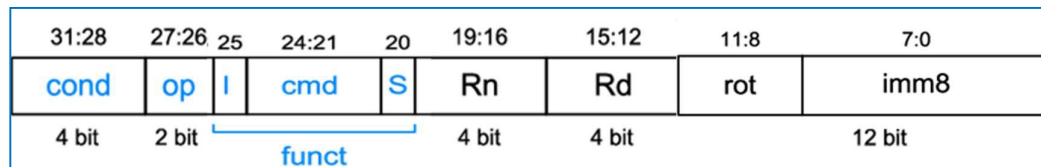
Τελεστές που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστές στον καταχωρητή προέλευσης Rn,
- ο τελεστές στον καταχωρητή προορισμού Rd,
- ο μη προσημασμένος άμεσος τελεστές 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)):



Project 1

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 σημαίων SR = (N, Z, C, V) (S = 1)

Τελεστές που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστής στον καταχωρητή προέλευσης Rn,
- ο τελεστής στον καταχωρητή προέλευσης Rm,
- ο τελεστής 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)):

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
cond	op 00	I	cmd	S	Rn	Rd	shamt5=0	sh 00	0	Rm
4 bit	2 bit	func			4 bit	4 bit	12 bit			

1-3.5. Η εντολή **CMP**

Κώδικας συμβολικής γλώσσας και περιγραφή στο επίπεδο RTL:

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

Τελεστές που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- ο τελεστής στον καταχωρητή προέλευσης Rn,
- ο τελεστής στον καταχωρητή προέλευσης Rm (I = 0),
- ο μη προσημασμένος άμεσος τελεστής 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):

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
cond	op	I	cmd	S	Rn	Rd	rot	imm8
4 bit	2 bit	func			4 bit	4 bit	12 bit	

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
cond	op 00	I	cmd	S	Rn	Rd	shamt5=0	sh 00	0	Rm
4 bit	2 bit	func			4 bit	4 bit	12 bit			

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)

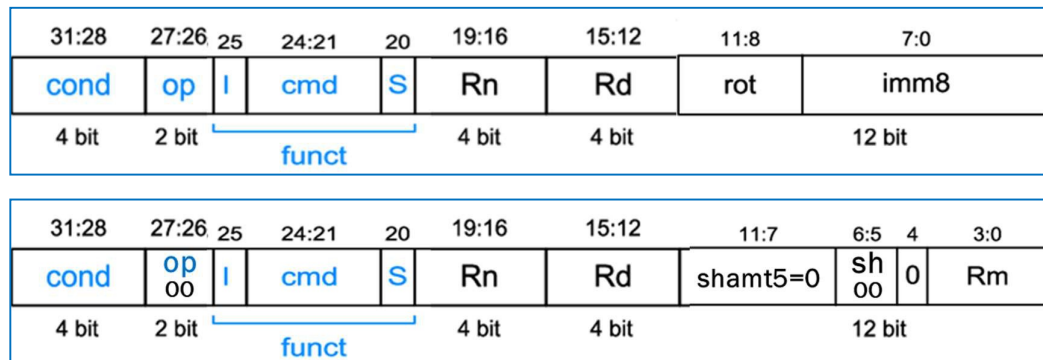
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- 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:

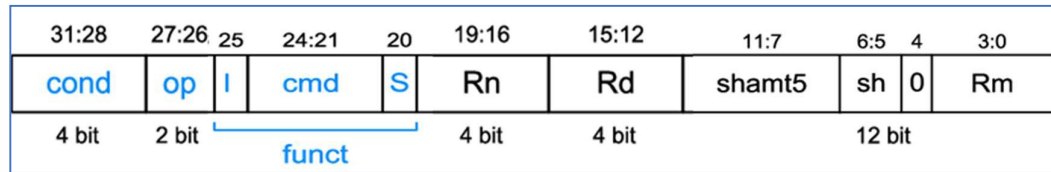
- ο τελεστέος στον καταχωρητή προέλευσης Rm (I = 0),
- ο τελεστέος στον καταχωρητή προορισμού Rd,
- η ποσότητα ολίσθησης shamt5 των 5 bits.

Πράξεις που εκτελούνται στη μονάδα ALU:

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

Project 1

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 ; $PC = BTA = PC + 8 + imm24 \times 4$ (L = 0)
- BL label ; $PC = BTA = PC + 8 + imm24 \times 4$; R14 = LR = PC + 4 (L = 1)

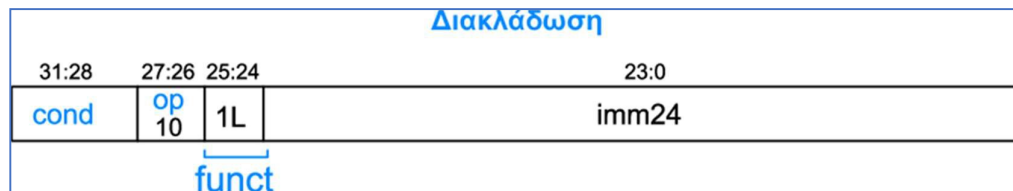
Τελεστέοι που χρησιμοποιούνται κατά την εκτέλεση της εντολής:

- 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 (σταθερά στο 1 στην περίπτωση του επεξεργαστή ενός κύκλου) 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.

Project 1

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 τελεστής 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 έξοδου.

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.**

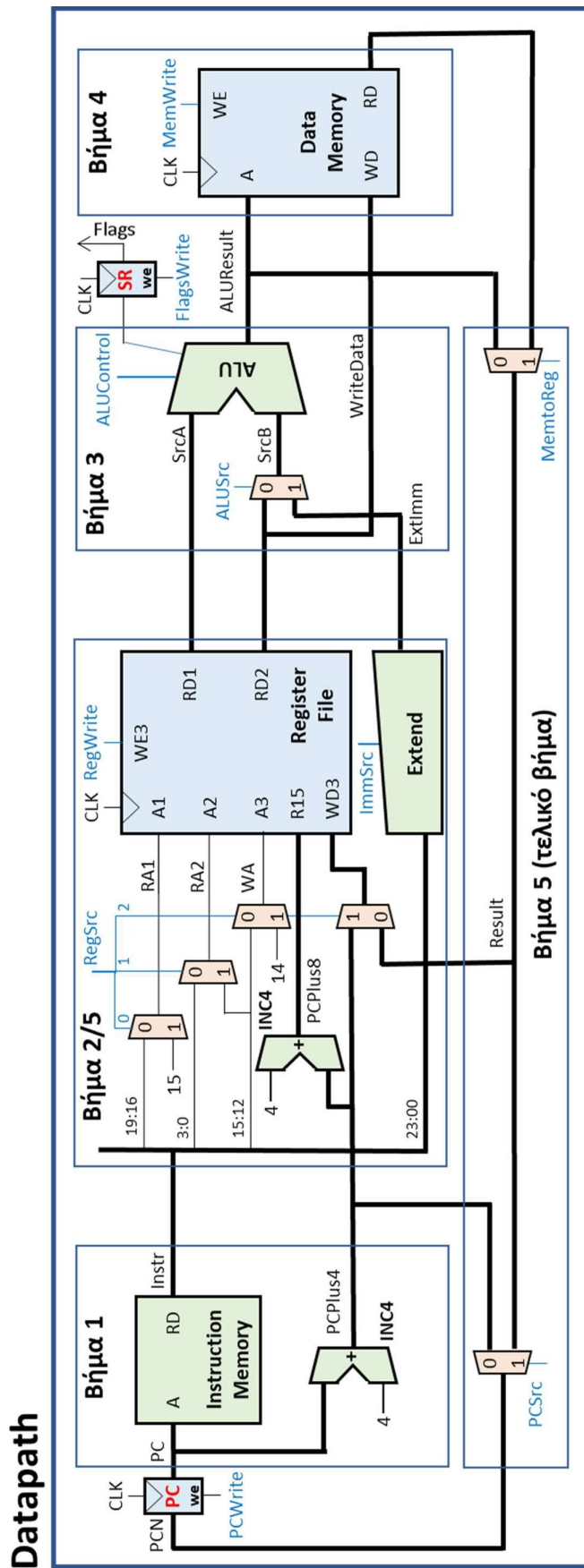
2-5.1. Παραμετροποιημένος πολυπλέκτης 2 σε 1 των N bit (με αρχική τιμή N = 32) για να χρησιμοποιηθεί στην επιλογή των δεδομένων που εγγράφονται ετεροχρονισμένα στον καταχωρητή προορισμού Rd (R0-R15) του αρχείου καταχωρητών (**mux2to1**). Επιλέγει μεταξύ του αποτελέσματος ALUResult της πράξης που εκτελείται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

Η σχεδίαση της διαδρομής δεδομένων (**datapath**) του επεξεργαστή ενός κύκλου ολοκληρώνεται με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom-up, όπως αυτή αποτυπώνεται στο ακόλουθο σχηματικό διάγραμμα.



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 ενός κύκλου 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

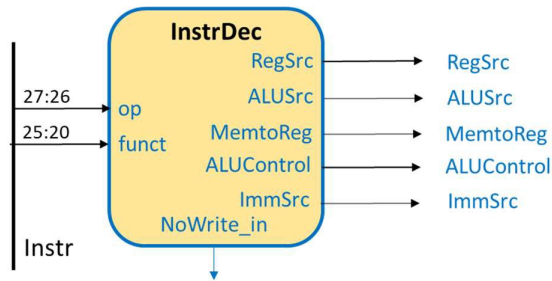
Δίδεται ως παράδειγμα ο πίνακας αλήθειας για τις εντολές LDR, STR και B.

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.

Project 1

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



4-2. Σχεδίαση του αποκωδικοποιητή των σημάτων έγκρισης εγγραφής (**WELogic**).

4-2.1. Συμπλήρωση του πίνακα αλήθειας του αποκωδικοποιητή των σημάτων έγκρισης εγγραφής (**WELogic**).

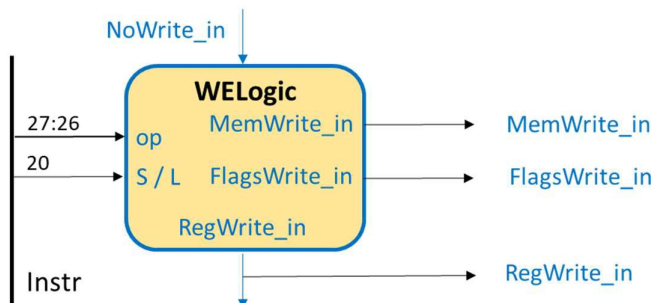
Είσοδοι, τα πεδία *op* και *S/L* της εντολής, καθώς και το εσωτερικό σήμα *NoWrite_in*. Έξοδοι, τα εσωτερικά σήματα *RegWrite_in*, *FlagsWrite_in* και *MemWrite_in*, που ελέγχονται από το εσωτερικό σήμα *CondEx_in*, το οποίο εμποδίζει την εγγραφή στο αρχείο καταχωρητών (*RegWrite* = 0), στον καταχωρητή κατάστασης (*FlagsWrite* = 0) και στη μνήμη δεδομένων (*MemWrite* = 0), όταν *CondEx_in* = 0.

Δίδεται ως παράδειγμα ο πίνακας αλήθειας για εντολές επεξεργασίας δεδομένων, μνήμης και διακλάδωσης.

Τύπος	Instr _{27:26} <i>op</i>	Instr ₂₀ <i>S / L</i>	No Write _{in}	Reg Write _{in}	Mem Write _{in}	Flags Write _{in}
DP	00	0	0	1	0	0
DP	00	1	0	1	0	1
CMP	00	1	1	0	0	1
LDR	01	1	0	1	0	0
STR	01	0	0	0	1	0
B	10	X	0	0	0	0

4-2.2. Με βάση τον πίνακα αλήθειας γίνεται η σχεδίαση του αποκωδικοποιητή των σημάτων έγκρισης εγγραφής (**WELogic**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής *case*).

4-2.3. Το σχηματικό διάγραμμα του αποκωδικοποιητή των σημάτων έγκρισης εγγραφής (**WELogic**) φαίνεται στη συνέχεια.



4-3. Σχεδίαση της λογικής επιλογής διεύθυνσης επόμενης εντολής (PCLogic).

4-3.1. Συμπλήρωση του πίνακα αλήθειας της λογικής επιλογής διεύθυνσης επόμενης εντολής (**PCLogic**) που ενεργοποιεί το εσωτερικό σήμα PCSrc_in, είτε όταν απαιτείται ετεροχρονισμένη εγγραφή στον καταχωρητή R15 (PC) (δηλαδή όταν Rd = 15 και RegWrite_in = 1), είτε όταν εκτελείται εντολή διακλάδωσης (op[1] = 1).

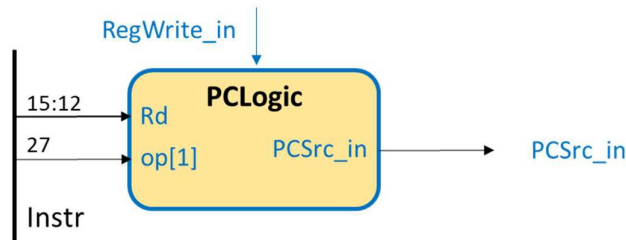
Είσοδοι, τα πεδία Rd και op[1] της εντολής καθώς και το εσωτερικό σήμα RegWrite_in. Έξοδος, το εσωτερικό σήμα PCSrc_in που ελέγχεται από το εσωτερικό σήμα CondEx_in, το οποίο παίρνει την τιμή 0, όταν CondEx_in = 0.

Δίδεται ως παράδειγμα ο πίνακας αλήθειας για εντολές επεξεργασίας δεδομένων, μνήμης και διακλάδωσης.

Τύπος	Instr _{27:26} op	Instr _{15:12} Rd	Reg Write_in	PCSrc_in
DP	00	0000 1110	1	0
DP	00	1111	1	1
CMP	00	XXXX	0	0
LDR	01	0000 1110	1	0
LDR	01	1111	1	1
STR	01	XXXX	0	0
B	10	XXXX	0	1

4-3.2. Με βάση τον πίνακα αλήθειας γίνεται η σχεδίαση της λογικής επιλογής διεύθυνσης επόμενης εντολής (**PCLogic**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case).

4-3.3. Το σχηματικό διάγραμμα της λογικής επιλογής διεύθυνσης επόμενης εντολής (**PCLogic**) φαίνεται στη συνέχεια.



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

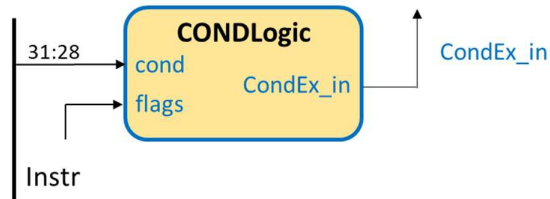
4-4.1. Συμπλήρωση του πίνακα αλήθειας της λογικής ελέγχου συνθήκης (**CONDLogic**), που ελέγχει εάν ικανοποιείται η συνθήκη που ορίζεται στο πεδίο cond της εντολής με βάση τις τρέχουσες τιμές των σημαίων N, Z, C, V (flags).

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.

Project 1

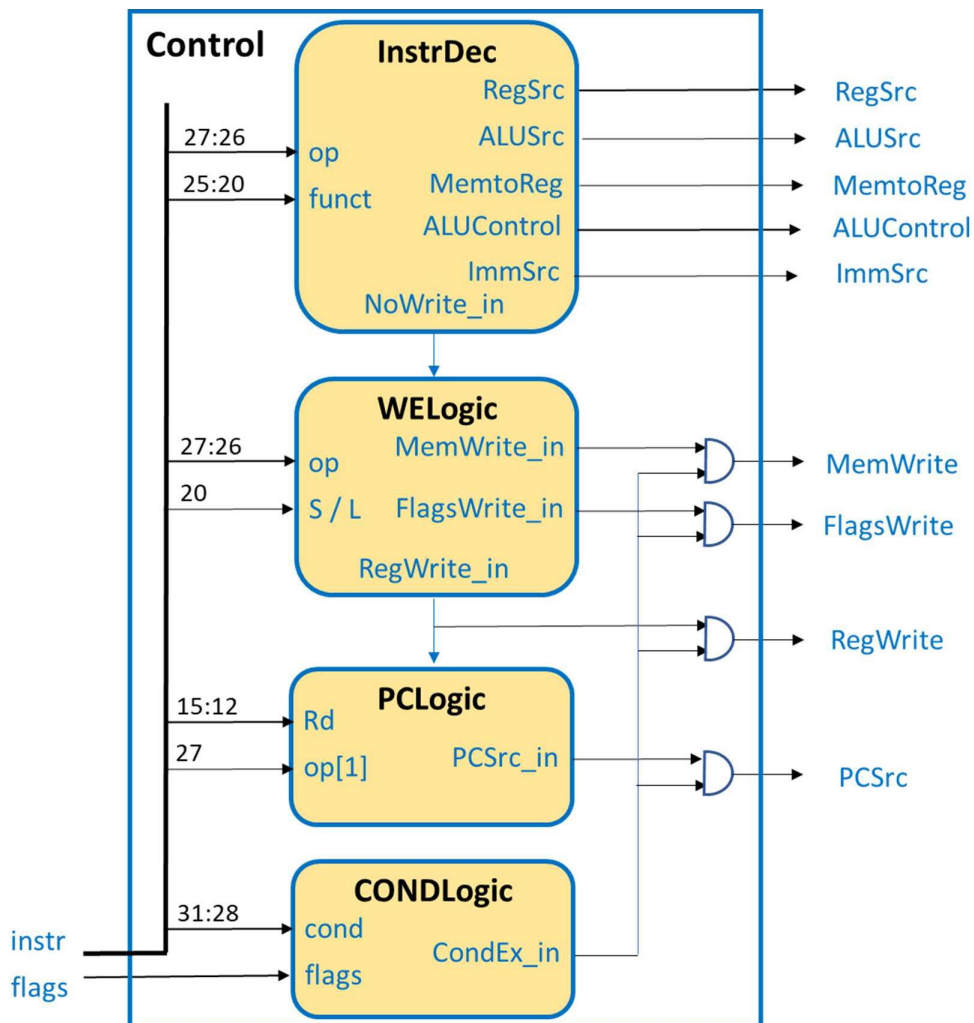
4-4.2. Με βάση τον πίνακα της παραγράφου 1-1.4, όπου φαίνονται τα **μνημονικά συνθήκης** με τις **εξισώσεις Boole** των σημαιών που τις ικανοποιούν, γίνεται η σχεδίαση της λογικής ελέγχου συνθήκης (**CONDLogic**) σε γλώσσα VHDL (περιγραφή συμπεριφοράς με τη χρήση της εντολής case).

4-4.3. Το σχηματικό διάγραμμα της λογικής ελέγχου συνθήκης (**CONDLogic**) φαίνεται στη συνέχεια.



4-5. Σχεδίαση της μονάδας ελέγχου (control)

Η σχεδίαση της μονάδας ελέγχου (**control**) του επεξεργαστή ενός κύκλου ολοκληρώνεται με τη χρήση περιγραφής δομής στη γλώσσα VHDL ακολουθώντας την ιεραρχική προσέγγιση bottom-up, όπως αυτή αποτυπώνεται στο ακόλουθο σχηματικό διάγραμμα (σε αυτό το επίπεδο εισάγονται και οι απαραίτητες πύλες AND).

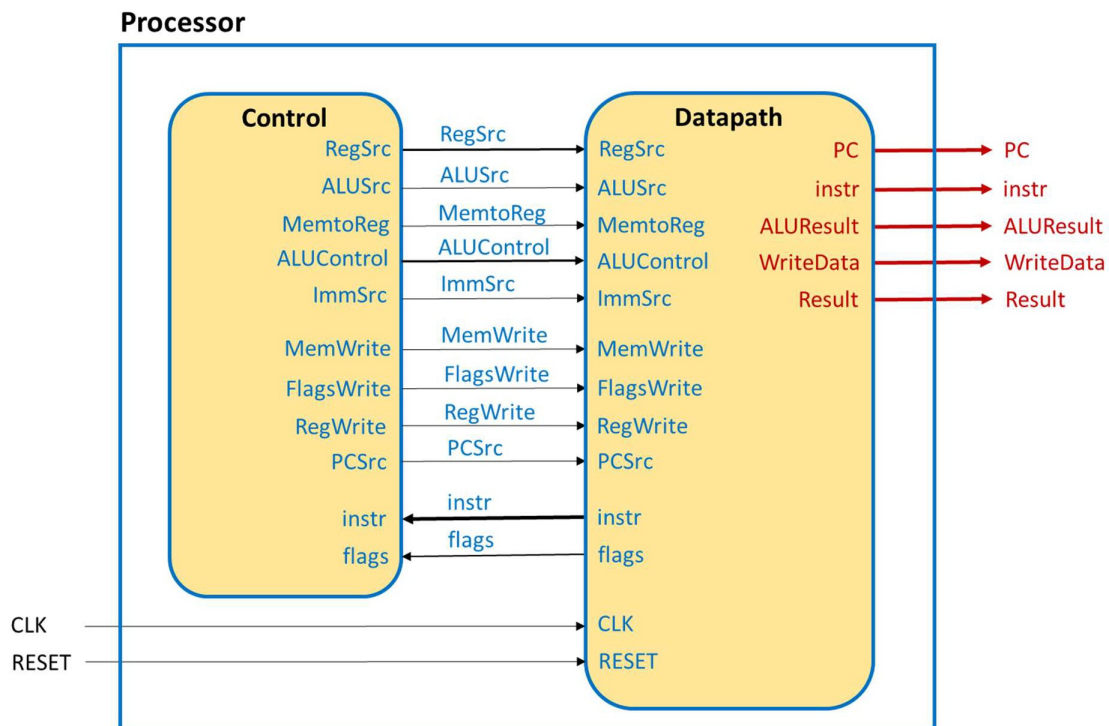


Βήμα 5: Σχεδίαση του επεξεργαστή (processor)

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 1, 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 λειτουργιών του **Vivado IDE** δημιουργώντας και το απαραίτητο πρόγραμμα δοκιμής (**ALU_TB**), που εκτελεί όλες τις πράξεις. Χρησιμοποιείτε 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 λειτουργιών 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). Χρησιμοποιείτε 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 λειτουργιών 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. Χρησιμοποιείτε ως βάση τον αθροιστή, αλλά χωρίς καταχωρητές εισόδου και εξόδου.

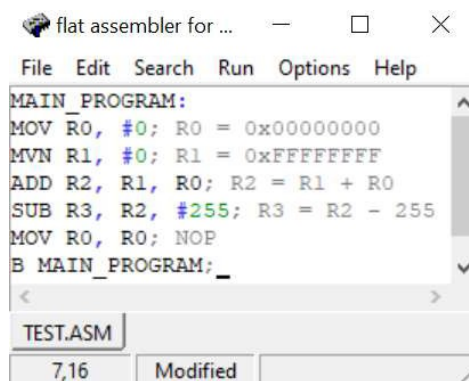
6-4. Επαλήθευση της ορθής σχεδίασης του επεξεργαστή (**processor**).

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 λειτουργιών 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 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 **FASMARM.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**).



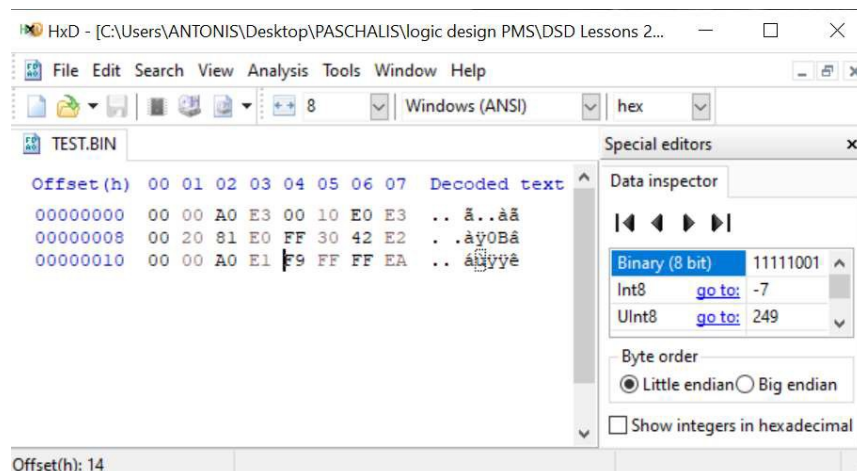
```

flat assembler for ...
File Edit Search Run Options Help
MAIN_PROGRAM:
MOV R0, #0; R0 = 0x00000000
MVN R1, #0; R1 = 0xFFFFFFFF
ADD R2, R1, R0; R2 = R1 + R0
SUB R3, R2, #255; R3 = R2 - 255
MOV R0, R0; NOP
B MAIN_PROGRAM; _
TEST.ASM
7,16 Modified

```

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.



```

HxD - [C:\Users\ANTONIS\Desktop\PASCHALIS\logic design PMS\DSD Lessons 2...
File Edit Search View Analysis Tools Window Help
TEST.BIN
Offset(h) 00 01 02 03 04 05 06 07 Decoded text
00000000 00 00 A0 E3 00 10 E0 E3 .. ä..ää
00000008 00 20 81 E0 FF 30 42 E2 .. àÿ0Bâ
00000010 00 00 A0 E1 F9 FF FF EA .. äÿÿÿè
Special editors
Data inspector
Binary (8 bit) 11111001
Int8 go to: -7
UInt8 go to: 249
Byte order
Little endian Big endian
Show integers in hexadecimal
Offset(h): 14

```

Project 1

Κάθε 4 συνεχόμενα byte (1 byte = 2 δεκαεξασδικά 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.vhc
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"EAF99999", 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 έναν κύκλο, 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 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	FFFFFFF		FFFFFFF	
3	ADD R2, R1, R0	0x08	E0812000	FFFFFFF	00000000	FFFFFFF	
4	SUB R3, R2, #255	0x0C	E24230FF	FFFFFF00		FFFFFF00	
5	MOV R0, R0	0x10	E1A00000	00000000	00000000	00000000	
6	B MAIN_PROGRAM	0x14	EAF	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 απλοποιημένη τεχνική αναφορά η οποία συμπεριλαμβάνει τα ακόλουθα:

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

7-1.1. Παραθέστε το σχηματικό διάγραμμα στο επίπεδο RTL (elaborated design) της διαδρομής δεδομένων (**datapath**) του επεξεργαστή. Περιγράψτε τα στοιχεία (components) που χρησιμοποιείτε (λειτουργία, ποια δεδομένα ανταλλάσσουν με τα component με τα οποία αλληλοεπιδρούν).

7-1.2. Παραθέστε το σχηματικό διάγραμμα στο επίπεδο RTL (elaborated design) της μονάδας ελέγχου (**control**) του επεξεργαστή. Περιγράψτε όλες τις υπομονάδες της μονάδας ελέγχου (λειτουργία, τα δεδομένα που ανταλλάσσουν με τα component με τα οποία αλληλοεπιδρούν) συμπληρώνοντας τους αντίστοιχους πίνακες αληθείας. Εξηγήστε για κάθε εντολή πως προκύπτουν οι αντίστοιχες τιμές στους πίνακες αληθείας. Η ανάλυση θα γίνει ανά πίνακα αληθείας.

7-1.3. Περιγράψτε τη δομή του επεξεργαστή (**processor**) στο ανώτερο ιεραρχικά επίπεδο με στοιχεία τη διαδρομή δεδομένων και τη μονάδα ελέγχου (λειτουργία, τα δεδομένα ανταλλάσσουν με τα δύο component). Παραθέστε το προκύπτον σχηματικό διάγραμμα στο επίπεδο RTL (elaborated design).

7-1.4. Περιγράψτε τη λειτουργία της ALU που έχετε σχεδιάσει (RTL διάγραμμα, περιγραφή/documentation του κώδικα, πόρους που χρησιμοποιεί στο στάδιο του implementation, μέγιστη ταχύτητα λειτουργίας,...).

7-1.5. Βρείτε τη μέγιστη συχνότητα λειτουργίας στο επίπεδο του επεξεργαστή (**processor**),

προσδιορίζοντας επίσης 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. Παραθέστε τα διαγράμματα χρονισμού των προσομοιώσεων της ALU (behavioral και post implementation timing) και τεκμηριώστε την ορθή του σχεδίαση και λειτουργία της ALU για κάθε πράξη που υλοποιεί.

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.

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)

Επιπλέον 10% στην πιο γρήγορη υλοποίηση.