8 Bit MIPS (One Cycle per Instruction) CPU Design

2022-2023

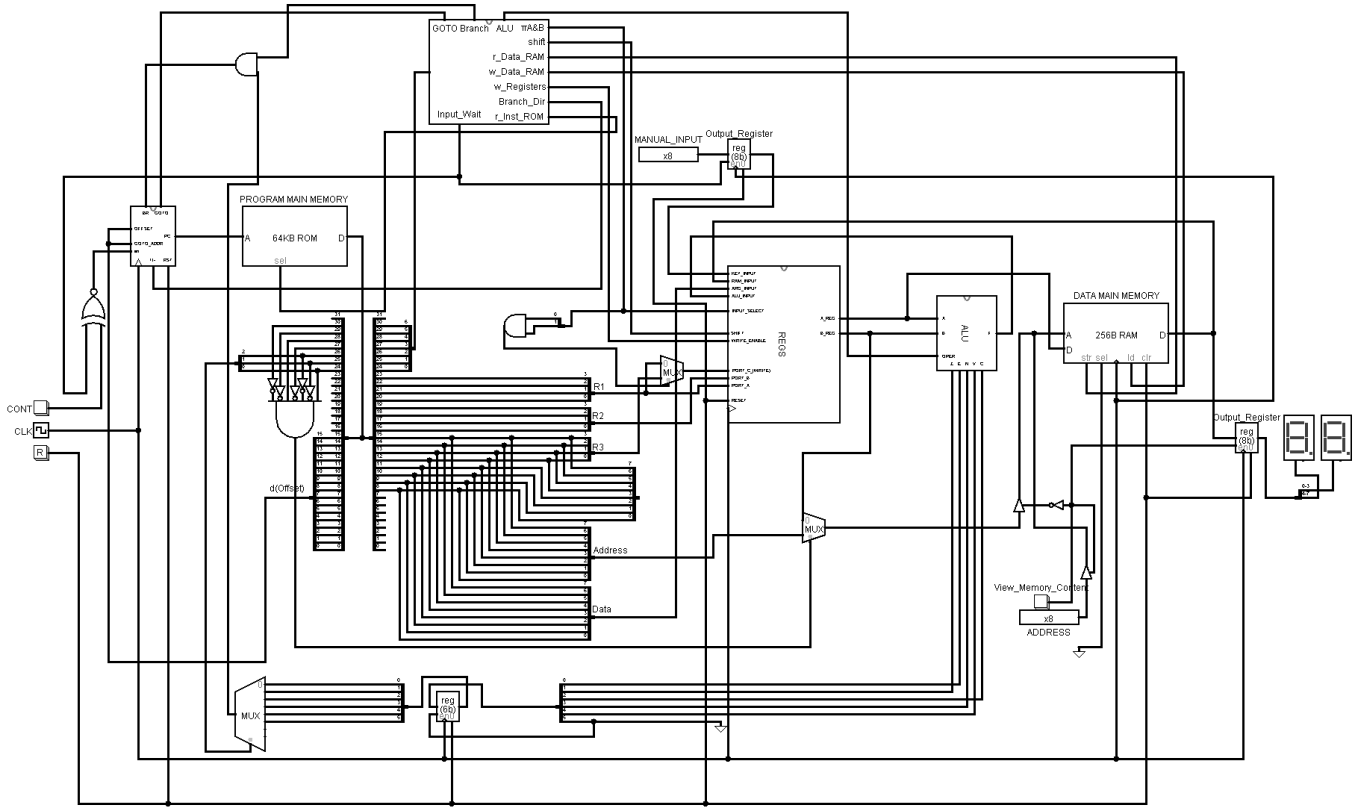
ΠΑΡΑΣΚΕΥΟΠΟΥΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

8 Bit CPU Design in Logisim

Μια Κεντρική Μονάδα Επεξεργασίας 8 Bits, όπου κάθε εντολή εκτελείται σε ένα κύκλο ρολογιού.

Η CPU διαθέτει ξεχωριστή μνήμη προγράμματος χωρητικότητας 256 Kbytes (16-Bit Address, 32-Bit Data)και μνήμη αποθήκευσης δεδομένων χωρητικότητας 256 Bytes.

Για την υλοποίηση του επεξεργαστή βασίστηκα στο βιβλίο Αρχιτεκτονική Υπολογιστών (Εκδόσεις Παναγιώτα Παπακωνσταντίνου) του καθηγητή Δημήτριου Νικολού.



Ο μικροεπεξεργαστής με όλα τα modules

Επεξήγηση των επιμέρους Modules

Η CPU αποτελείται από τα εξής modules τα οποία θα αναλυθούν στη συνέχεια:

1) ALU Module
2) Addition/Subtraction Module
3) Register File Module
4) Control Unit Module
5) microProgram Counter Module

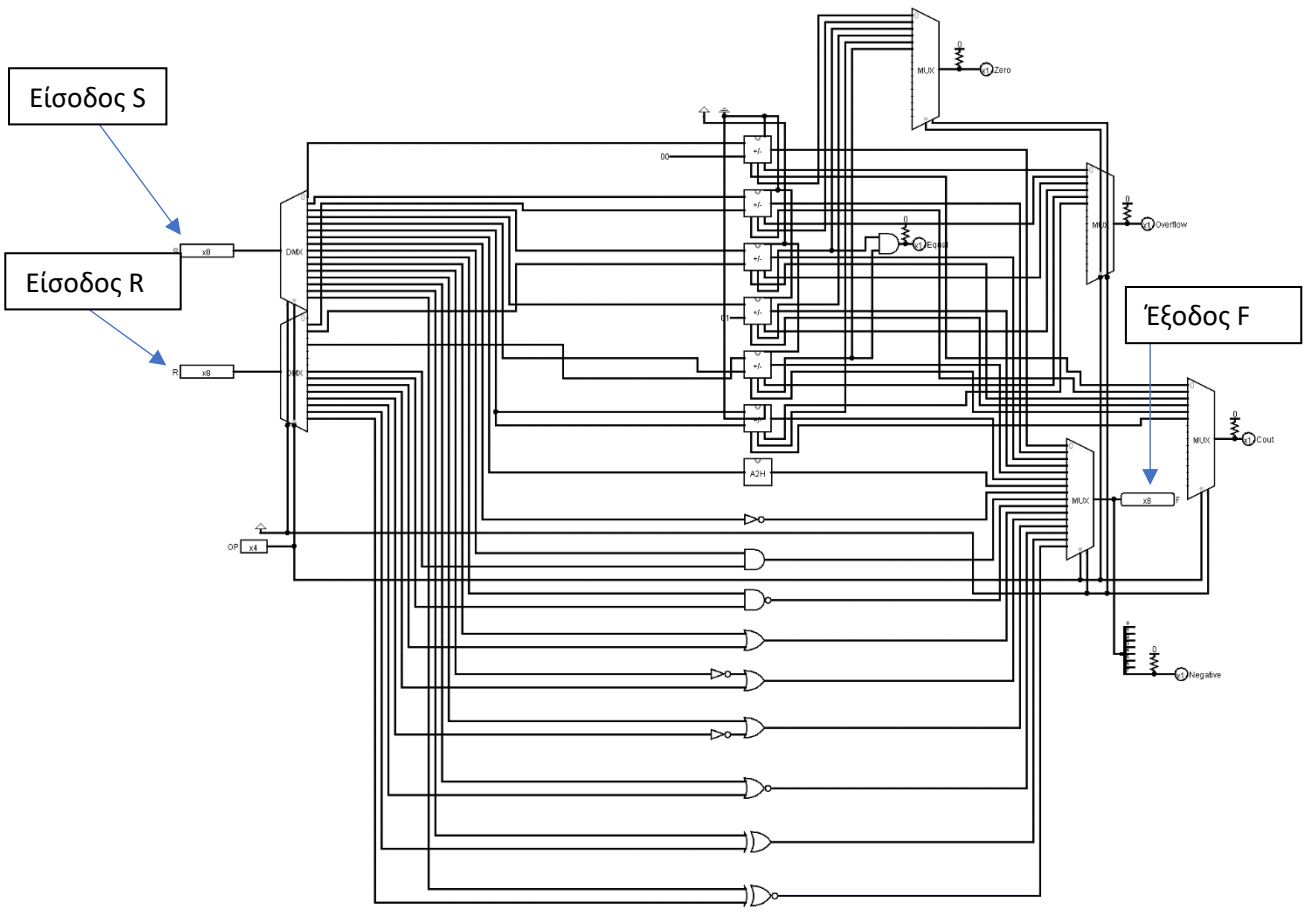
1) ALU Module

Η Αριθμητική και Λογική μονάδα έχει σχεδιαστεί με σκοπό την εκτέλεση απλών πράξεων μεταξύ 2 αριθμών 8-Bit. Οι πράξεις που μπορούν να εκτελεστούν συνοψίζονται στον παρακάτω πίνακα (με κόκκινο χρώμα σημειώνονται τα ορίσματα της κάθε εντολής):

Instruction	COMMENTS	PSEUDOCODE
COPY R1,R3	COPY REGISTER R1 TO R3	R3=R1+0
ADD R1,R2,R3	ADD R1 TO R2 AND STORE TO R3	R3=R1+R2
SUB R1,R2,R3	SUB R2 FROM R1 AND STORE TO R3	R3=R1-R2
ADI R1,R3	INCREMENT R1 AND STORE TO R3	R3=R1+1
SUBR R1,R2,R3	SUB R1 FROM R2 AND STORE TO R3	R3=R2-R1
LSR R1,R3	ADD R1 TO R1 AND STORE TO R3	R3=R1+R1
MUL R1,R2,R3	MULTIPLY R1 BY R2 AND STORE TO R3	R3=R1*R2
NOT R1,R3	1's COMPLEMENTOF R1 STORED TO R3	R3=~R1
AND R1,R2,R3	R1 AND R2 STORE TO R3	R3=R1 AND R2
NAND R1,R2,R3	R1 NAND R2 STORE TO R3	R3=R1 NAND R2
OR R1,R2,R3	R1 OR R2 STORE TO R3	R3=R1 OR R2
NOTOR R1,R2,R3	~R1 OR R2 STORE TO R3	R3=(~R1) OR R2
ORNOT R1,R2,R3	R1 OR ~R2 STORE TO R3	R3=R1 OR (~R2)
NOR R1,R2,R3	R1 NOR R2 STORE TO R3	R3=R1 NOR R2
XOR R1,R2,R3	R1 XOR R2 STORE TO R3	R3=R1 XOR R2
XNOR R1,R2,R3	R1 XNOR R2 STORE TO R3	R3=R1 XNOR R2

Για την υλοποίηση της ALU χρησιμοποιήθηκαν 2 αποπλέκτες (1-16) για την οδήγηση των εισόδων στα micromodules υπεύθυνα για την κάθε αριθμητική ή λογική πράξη και 1 πολυπλέκτης (16-1) για την οδήγηση του αποτελέσματος της κάθε πράξης στην έξοδο. Επίσης γίνεται η

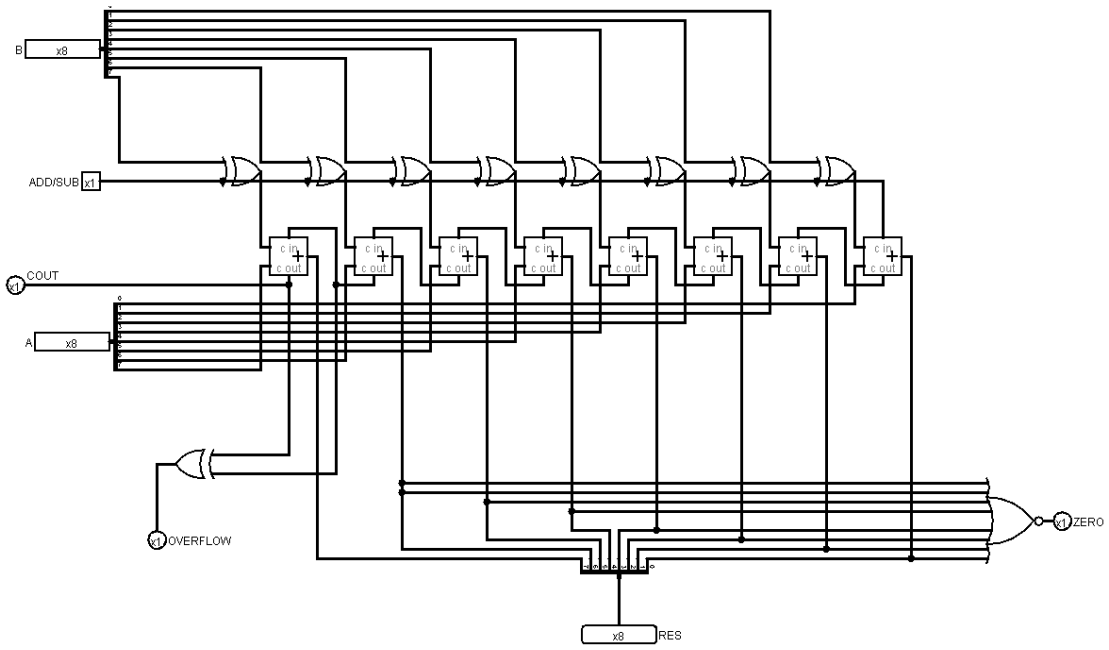
χρήση πολυπλεξίας για την σωστή έκφραση των C(arry), (o)V(erflow) και Z(ero) Flags όπου προκύπτουν. Να σημειωθεί πως σε περίπτωση η σημαία Z ενεργοποιείται σε περίπτωση μηδενικού αποτελέσματος, ωστόσο στην περίπτωση όπου αυτό προκύπτει από αφαίρεση ενεργοποιείται η ξεχωριστή σημαία E(qual) η οποία αποτελεί ένδειξη πως 2 αριθμοί είναι ίσοι.



Η Αριθμητική-Λογική Μονάδα

2) Adder/Subtractor Module

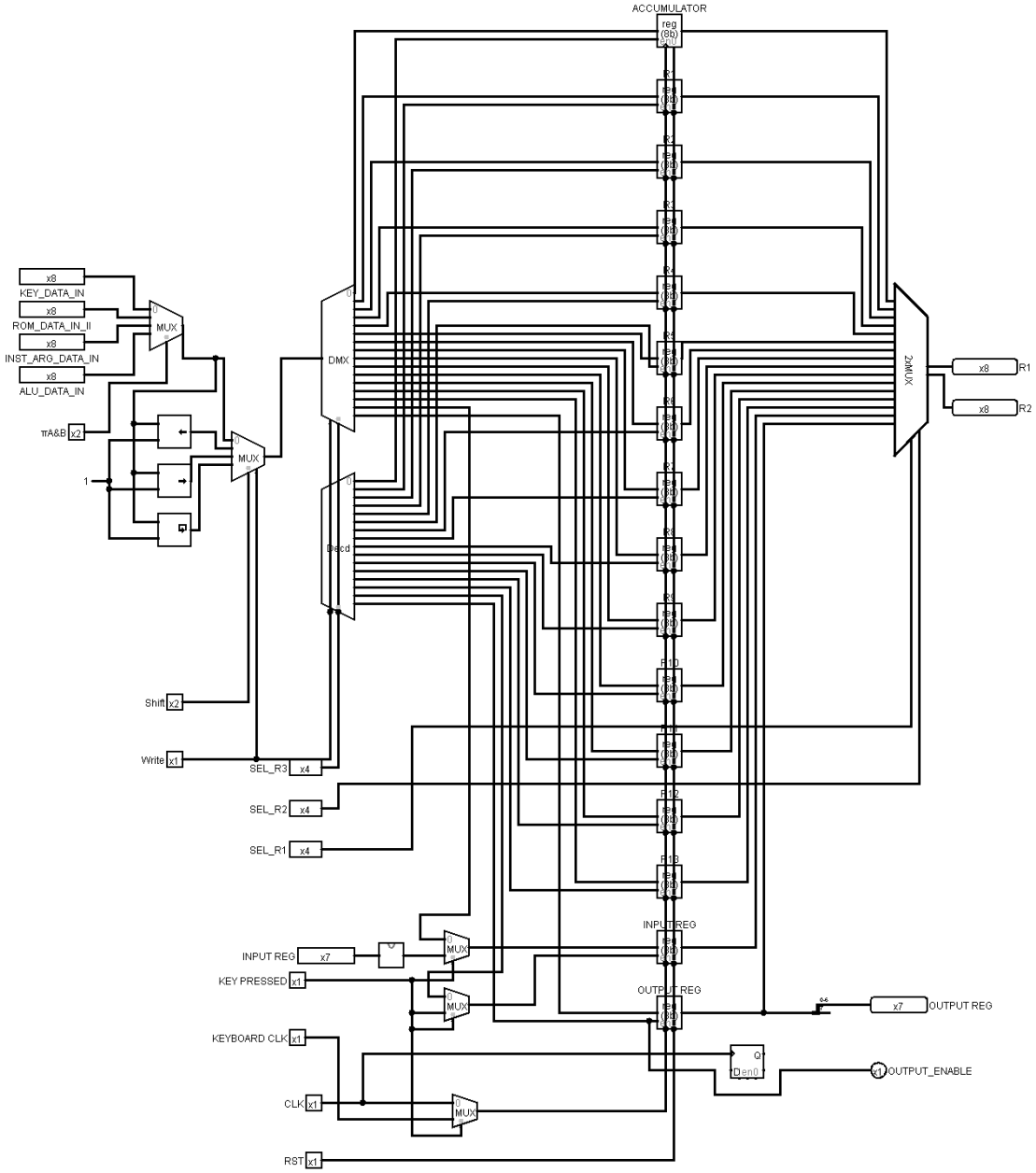
Για την υλοποίηση της μονάδας πρόσθεσης-αφαίρεσης, χρησιμοποιήθηκε το παρακάτω κύκλωμα, στο οποίο χρησιμοποιούμε XOR πύλες για την ελεγχόμενη αντιστροφή της εισόδου B. Από το κύκλωμα αυτό παράγονται οι σημαίες Overflow ($V=C_{n-1} \oplus C_{n-2}$), $Carry_{out}$ και Zero ($Z=(S_{n-1}+S_{n-2}\ldots+S_1+S_0)'$)



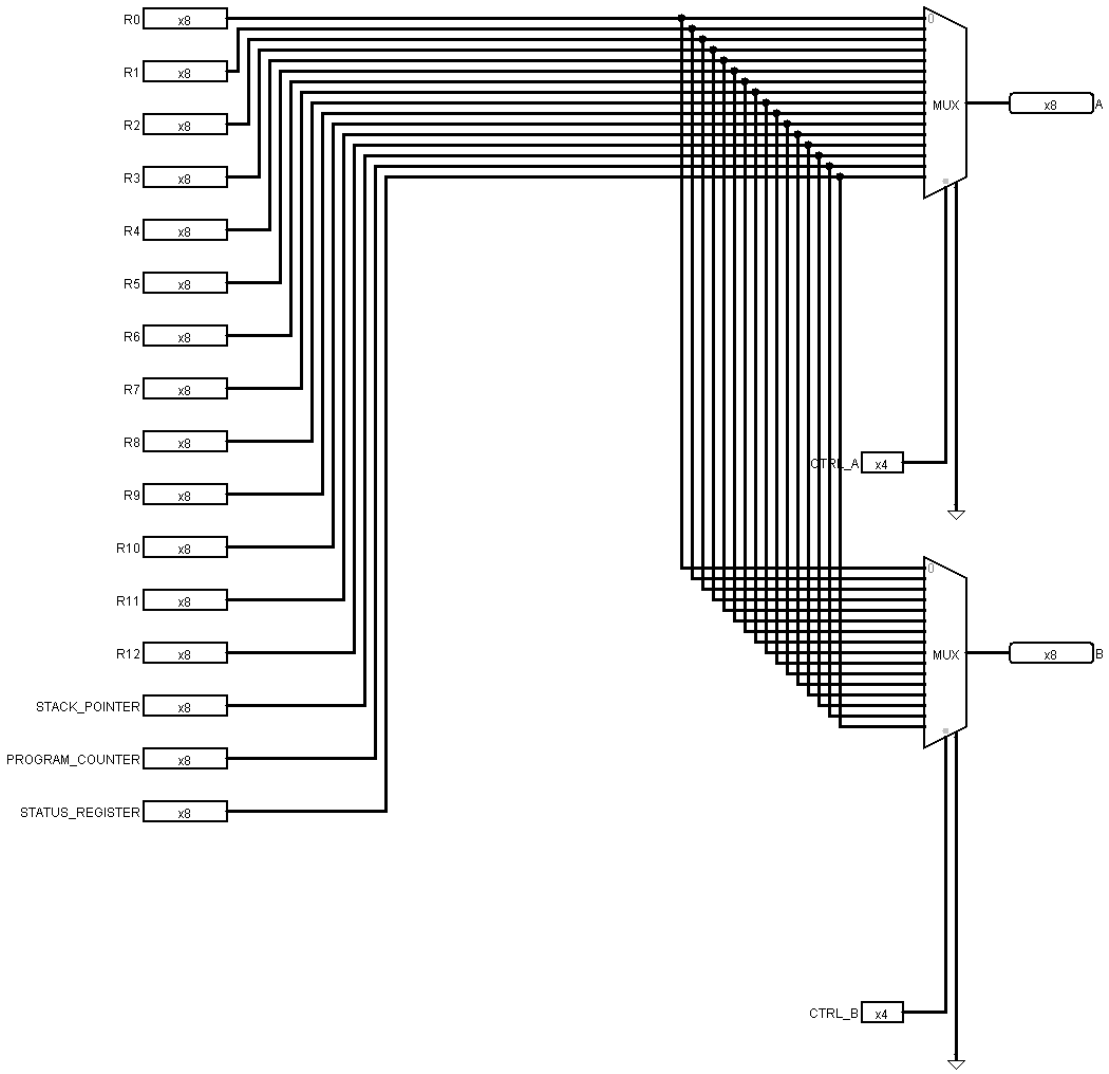
3) Register File

Στη συνέχεια σχεδιάστηκε το Register File με 16 καταχωρητές των 8bit, το οποίο διαθέτει 2 read-ports και 1 write-port. Η είσοδος εγγραφής μπορεί να ολισθηθεί ανάλογα με την τιμή του σήματος “SHIFT” Στους καταχωρητές περιέχονται και δύο ειδικοί καταχωρητές, ο input register ο οποίος συνδέεται άμεσα με το πληκτρολόγιο (ωστόσο ο προγραμματιστής μπορεί να τον χρησιμοποιήσει και ως καταχωρητή γενικού σκοπού) και μπορεί να πάρει την είτε hex δεδομένα (με μετατροπή από ASCII σε hex τιμή -ο χρήστης πρέπει να πληκτρολογήσει το hex δεδομένων με πατημένο το πλήκτρο SHIFT-)είτε ASCII χαρακτήρες κατά την εκτέλεση της εντολής SCANCH η οποία δημιουργεί ένα interrupt μέχρι ο χρήστης να πληκτρολογήσει 1 χαρακτήρα στο πληκτρολόγιο. Στη συνέχεια υπάρχει και ο output register ο οποίος είναι συνδεδεμένος άμεσα με το τερματικό το οποίο χρησιμοποιούμε ως έξοδο. Στους καταχωρητές μπορούμε να γράψουμε από την 1) έξοδο της Αριθμητικής-Λογικής Μονάδας, 2) Από την Μνήμη Δεδομένων (ξεχωριστή από την μνήμη Εντολών), 3) Από όρισμα της εντολής η

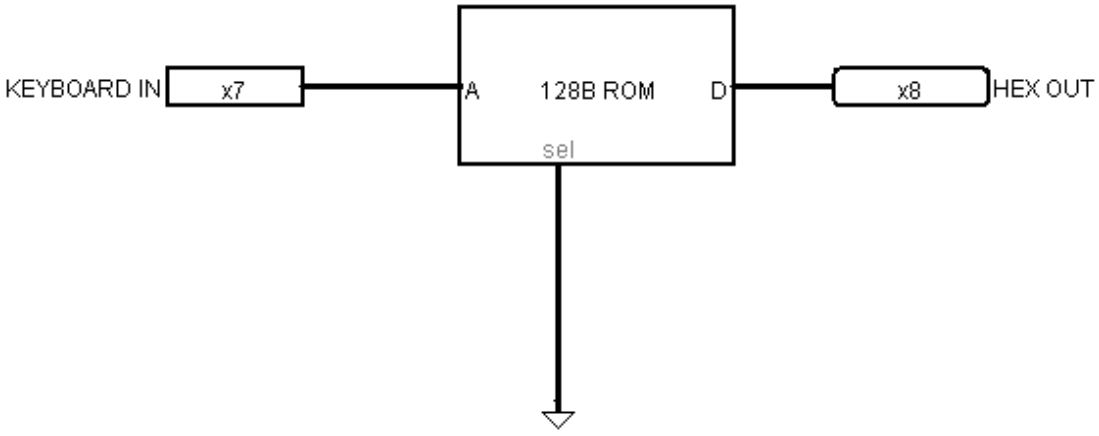
οποίας εκτελείται εκείνη την στιγμή και τέλος 4) Από το πληκτρολόγιο.



To Register File



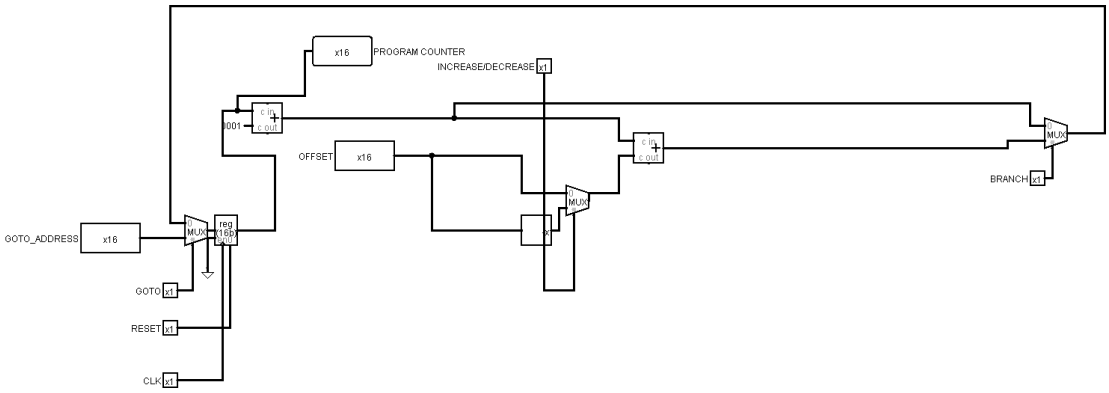
Ο σχεδιασμός του διπλού πολυπλέκτη για την υλοποίηση της διπλής πόρτας διαβάσματος



Το κύκλωμα της μετατροπής από ASCII σε HEX

4) MicroProgram Counter Module

Για την σχεδίαση του Counter χρησιμοποιήθηκε η τακτική του Count=Count+1 δηλαδή ο Counter είναι συνδεδεμένος μόνιμα με έναν Adder ο οποίος έχει την μία του τιμή συνδεδεμένη με την σταθερά 0x1. Στη συνέχεια είναι συνδεδεμένος με έναν ακόμα αθροιστή ο οποίος ενεργοποιείται βάση του σήματος Branch και λαμβάνοντας ένα όρισμα της εντολής μεταβάλει κατά αυτή την τιμή τον μετρητή προγράμματος (είτε θετικά είτε αρνητικά). Επίσης ο μετρητής προγράμματος έχει την δυνατότητα μεταφοράς σε συγκεκριμένη διεύθυνση η οποία επίσης δίνεται ως όρισμα εντολής



6) Control Unit Module

Στο σημείο αυτό έπρεπε να σχεδιαστεί ένα κύκλωμα για των έλεγχο όλων των υπόλοιπων κυκλωμάτων. Αυτό έγινε με την χρήση μιας μνήμης ROM με διευθύνσεις των 7 bit και περιεχόμενο των 16 bit. Αυτό έγινε για απλοποίηση του κυκλώματος καθώς η υλοποίηση με PLA θα δυσκόλευε αρκετά την όλη διαδικασία. Τα σήματα που χρησιμοποιούμε για να ελέγξουμε τις υπομονάδες του μικροεπεξεργαστή είναι:

1) **πA&B** (2 bits): Αυτό το σήμα ελέγχει την είσοδο των καταχωρητών, δηλαδή από ποιο module θα εγγραφεί πληροφορία στους καταχωρητές.

00	Είσοδος από Πληκτρολόγιο
01	Είσοδος από μνήμη Δεδομένων
10	Είσοδος από όρισμα εντολής
11	Είσοδος από αποτέλεσμα (έξοδο) ALU

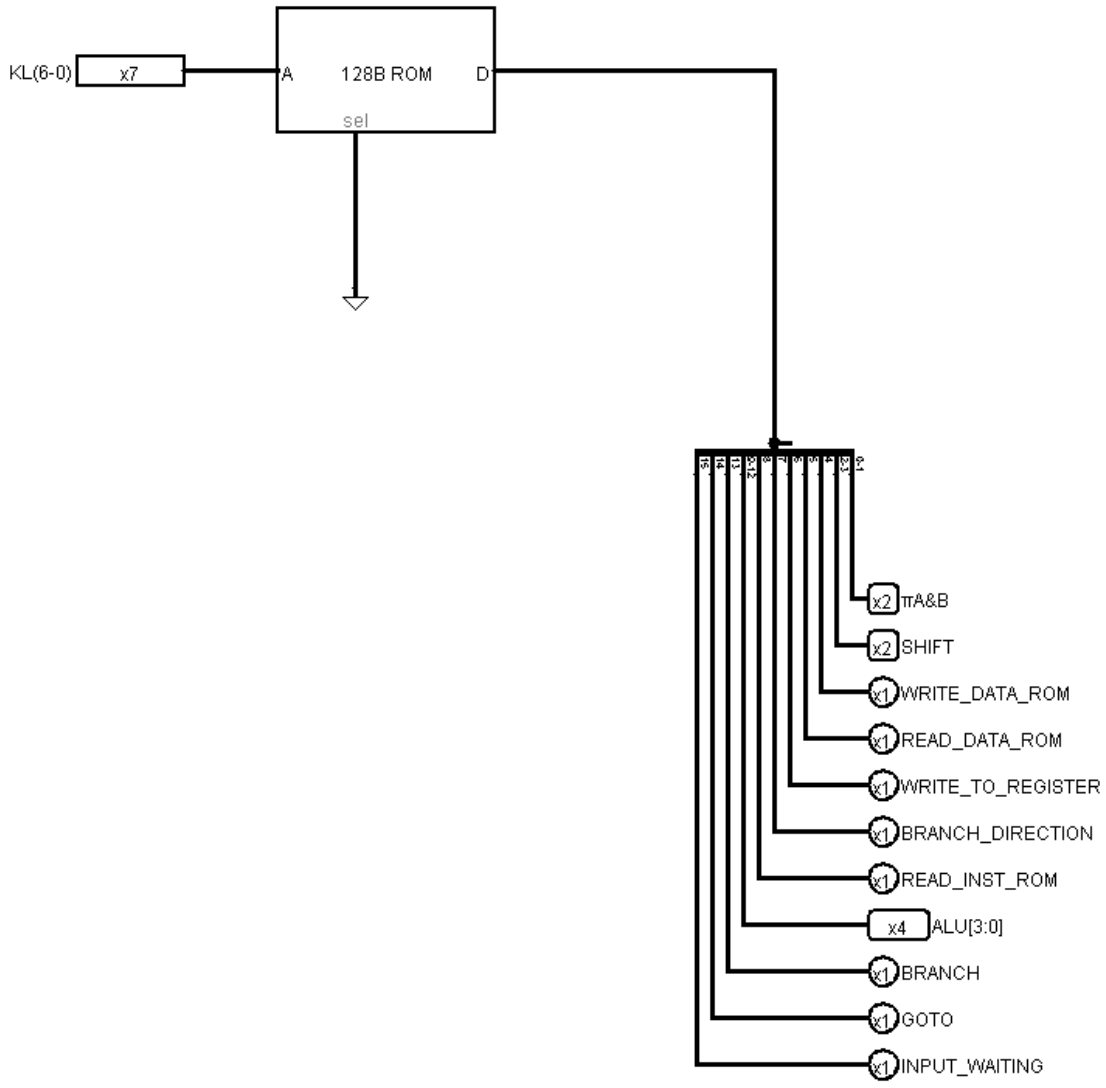
2) **SHIFT** (2 Bits): Αυτό το σήμα ελέγχει αν θα υπάρξει ολίσθηση πριν από την εγγραφή στο Register File καθώς και το είδος της.

00	Καμία Ολίσθηση
01	Λογική Ολίσθηση Αριστερά κατά μία θέση
10	Λογική Ολίσθηση Δεξιά κατά μία θέση
11	Κυκλική Ολίσθηση Δεξιά κατά μία θέση

3) **WRITE_DATA_RAM** (1 Bit): Το σήμα αυτό ελέγχει αν θα ενεργοποιηθεί η είσοδος εγγραφής της μνήμης δεδομένων (1 για εγγραφή στη μνήμη δεδομένων)
4) **READ_DATA_RAM** (1 Bit): Το σήμα αυτό ελέγχει την ενεργοποίηση της εξόδου της μνήμης δεδομένων για διάβασμα (1 για διάβασμα της μνήμης δεδομένων)
5) **WRITE_TO_REGISTER** (1 Bit): Το σήμα αυτό ενεργοποιεί την εγγραφή στο Register File (1 για Εγγραφή στο RegFile)
6) **BRANCH_DIRECTION** (1 Bit): Το σήμα αυτό σε περίπτωση Branch (μεταβολή του μετρητή προγράμματος με πρόσθεση ή αφαίρεση ενός αριθμού 8bit) ελέγχει την κατεύθυνση του Branch, δηλαδή αν θα γίνει πρόσθεση ή αφαίρεση στο περιεχόμενο του μετρητή προγράμματος. (1 για αφαίρεση και 0 για πρόσθεση)
7) **READ_INST_ROM** (1 Bit): Το σήμα αυτό ενεργοποιεί το διάβασμα της μνήμης εντολών.
8) **ALU** (4 Bit): Το σήμα αυτό ελέγχει την πράξη που θα εκτελεστεί από την ALU. (Γ: Καταχωρητής αποθήκευσης προγράμματος |A,B: Είσοδοι ALU)

0000	Γ=A+0
0001	Γ=A+B
0010	Γ=A-B
0011	Γ=A+1
0100	Γ=B-A
0101	Γ=A+A
0110	Γ=A*B
0111	Γ=A’
1000	Γ=A·B (AND)
1001	Γ=(A·B)’ (NAND)
1010	Γ=A+B (OR)
1011	Γ=A’+B
1100	Γ=A+B’
1101	Γ=(A+B)’ (NOR)
1110	Γ=A⊕B (XOR)
1111	Γ= (A⊕B)’ (XNOR)

9) **BRANCH** (1 Bit): Όταν το σήμα αυτό είναι ενεργοποιημένο τότε ανάλογα με το σήμα **BRANCH_DIRECTION** προστίθεται ή αφαιρείται μία ποσότητα από τον μετρητή προγράμματος.
10) **GOTO** (1 Bit): Όταν το σήμα αυτό είναι ενεργοποιημένο ο μετρητής προγράμματος μεταφέρεται στην διεύθυνση που δείχνει το όρισμα της εντολής που εκτελείται την δεδομένη στιγμή
11) **INPUT_WAITING** (1 Bit): Αυτό το σήμα ενεργοποιεί τον καταχωρητή δεδομένων εισόδου για χειροκίνητη είσοδο δεδομένων μέσω switches ενώ σταματά και τον μProgram Counter ώστε να δώσει χρόνο στον χρήστη να εισάγει τα δεδομένα.



Οι εντολές είναι των 32bit ενώ περιέχουν και τα ορίσματα τους.

	7				6				5				4				3				2				1				0			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COPY	x	0	0	0	0	0	0	0	r1				x	x	x	x	r3				x	x	x	x	x	x	x	x	x	x	x	x
ADD	x	0	0	0	0	0	0	1	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
SUB	x	0	0	0	0	0	1	0	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
ADI	x	0	0	0	0	0	1	1	r1				x	x	x	x	r3				x	x	x	x	x	x	x	x	x	x	x	x
SUBR	x	0	0	0	0	1	0	0	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
LSR	x	0	0	0	0	1	0	1	r1				x	x	x	x	r3				x	x	x	x	x	x	x	x	x	x	x	x
MUL	x	0	0	0	0	1	1	0	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
NOT	x	0	0	0	0	1	1	1	r1				x	x	x	x	r3				x	x	x	x	x	x	x	x	x	x	x	x
AND	x	0	0	0	1	0	0	0	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
NAND	x	0	0	0	1	0	0	1	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
OR	x	0	0	0	1	0	1	0	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
NOTOR	x	0	0	0	1	0	1	1	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
ORNOT	x	0	0	0	1	1	0	0	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
NOR	x	0	0	0	1	1	0	1	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
XOR	x	0	0	0	1	1	1	0	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
XNOR	x	0	0	0	1	1	1	1	r1				r2				r3				x	x	x	x	x	x	x	x	x	x	x	x
LOAD	x	0	0	1	0	0	0	0	r1				r2				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
LOADADR	x	0	0	1	0	0	0	1	r1				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
LOADLIT	x	0	0	1	0	0	1	1	r1				x	x	x	x	data								x	x	x	x	x	x	x	x
LSLOAD	x	1	0	1	0	0	0	1	r1				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
RSLOAD	x	1	0	1	0	0	1	0	r1				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
CRSLOAD	x	1	0	1	0	0	1	1	r1				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
STORE	x	1	0	0	0	0	0	0	r1				r2				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
NOP	x	1	1	1	1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
BRE	x	0	1	1	0	0	0	0	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
BRZ	x	0	1	1	0	0	0	1	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
BRC	x	0	1	1	0	0	1	0	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
BRN	x	0	1	1	0	0	1	1	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
BRO	x	0	1	1	0	1	0	0	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
JMP	x	0	1	1	0	1	0	1	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
GOTO	x	0	1	1	0	1	1	0	x	x	x	x	x	x	x	x	address								x	x	x	x	x	x	x	x
MBRE	x	1	1	1	0	0	0	0	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
MBRZ	x	1	1	1	0	0	0	1	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
MBRC	x	1	1	1	0	0	1	0	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
MBRN	x	1	1	1	0	0	1	1	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
MBRO	x	1	1	1	0	1	0	0	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x
MJMP	x	1	1	1	0	1	0	1	x	x	x	x	x	x	x	x	d								x	x	x	x	x	x	x	x

INSTRUCTION SET

Instruction	COMMENTS	PSEUDOCODE
COPY	COPY REGISTER R1 TO R3	R3=R1+0
ADD	ADD R1 TO R2 AND STORE TO R3	R3=R1+R2
SUB	SUB R2 FROM R1 AND STORE TO R3	R3=R1-R2
ADI	INCREMENT R1 AND STORE TO R3	R3=R1+1
SUBR	SUB R1 FROM R2 AND STORE TO R3	R3=R2-R1
LSR	ADD R1 TO R1 AND STORE TO R3	R3=R1+R1
MUL	MULTIPLY R1 BY R2 AND STORE TO R3	R3=R1*R2
NOT	1's COMPLEMENTOF R1 STORED TO R3	R3=~R1
AND	R1 AND R2 STORE TO R3	R3=R1 AND R2
NAND	R1 NAND R2 STORE TO R3	R3=R1 NAND R2
OR	R1 OR R2 STORE TO R3	R3=R1 OR R2
NOTOR	~R1 OR R2 STORE TO R3	R3=~R1 OR R2
ORNOT	R1 OR ~R2 STORE TO R3	R3=R1 OR ~R2
NOR	R1 NOR R2 STORE TO R3	R3=R1 NOR R2
XOR	R1 XOR R2 STORE TO R3	R3=R1 XOR R2
XNOR	R1 XNOR R2 STORE TO R3	R3=R1 XNOR R2
LOAD	CONTENT OF RAM ADDRESSED BY R2 STORED TO R1	R1=($R2)
LOADADR	CONTENT OF RAM ADDRESSED BY ARG. STORED TO R1	R1=($ARG)
LOADLIT	LITERAL NUMBER (ARGUMENT) STORED TO R1	R1=ARG
LSLOAD	COPY REGISTER R1 TO R3 LEFT SLIDED	R3=R1<<1
RSLOAD	COPY REGISTER R1 TO R3 RIGHT SLIDED	R3=R1>>1
CRSLOAD	COPY REGISTER R1 TO R3 CIRCULAR RIGHT SLIDED	R3=R1>>1, CIRCULAR
STORE	STORE R1 TO RAM ADDRESSED BY R2	STORE R1 TO ($R2)
NOP	NO OPERATION	
BRE	BRANCH IF EQUAL FLAG-ADD d ARG TO PC	PC=PC+d, IF(R1.EQUAL==TRUE)
BRZ	BRANCH IF ZERO FLAG-ADD d ARG TO PC	PC=PC+d, IF(R1.ZERO==TRUE)
BRC	BRANCH IF CARRY FLAG-ADD d ARG TO PC	PC=PC+d, IF(R1.CARRY==TRUE)
BRN	BRANCH IF NEGATIVE FLAG-ADD d ARG TO PC	PC=PC+d, IF(R1.NEGATIVE==TRUE)
BRO	BRANCH IF OVERFLOW FLAG-ADD d ARG TO PC	PC=PC+d, IF(R1.OVERFLOW==TRUE)
JMP	BRANCH-ADD d ARG TO PC	PC=PC+d
GOTO	GO TO ADDRESS d	PC=d
MBRE	BRANCH IF ZERO FLAG-SUB d ARG FROM PC	PC=PC-d, IF(R1.EQUAL==TRUE)
MBRZ	BRANCH IF CARRY FLAG-SUB d ARG FROM PC	PC=PC-d, IF(R1.ZERO==TRUE)
MBRC	BRANCH IF NEGATIVE FLAG-SUB d ARG FROM PC	PC=PC-d, IF(R1.CARRY==TRUE)

MBRN	BRANCH IF OVERFLOW FLAG-SUB d ARG FROM PC	PC=PC-d, IF(R1.NEGATIVE==TRUE)
MBRO	BRANCH-SUB d ARG FROM PC	PC=PC-d, IF(R1.OVERFLOW==TRUE)
MJMP	BRANCH IF EQUAL FLAG-SUB d ARG FROM PC	PC=PC-d
SCANCH		
PRINTCH	COPY REGISTER R1 TO OUTPUT REGISTER	OREG=R1+0

Bugs: Στις εντολές Branch ο μετρητής προγράμματος πάντα θα πρέπει να υπολογίζουμε -1 από αυτό που θέλουμε γιατί ανεξαρτήτως του αν θα κάνει Branch ο μετρητής προγράμματος θα αυξάνεται πάντα κατά 1. Δηλαδή αν θέλουμε για παράδειγμα να πάμε 3 θέσεις μνήμης πίσω θα πρέπει να υπολογίσουμε ότι ο PC θα κάνει το +1 πρώτα άρα θα πρέπει να κάνουμε Branch κατά 4 θέσεις μνήμης (Άρα JMP, -0x4, για παράδειγμα) και αν πάλι θέλουμε να πάμε 3 θέσεις μπροστά θα πρέπει να υπολογίσουμε ότι ο PC πάλι θα κάνει το +1 ούτως ή άλλως οπότε θα κάνουμε Branch κατά 2 θέσεις μνήμης (Άρα JMP, 0x2, για παράδειγμα).

Compiler

Ο μικροεπεξεργαστής συνοδεύεται από μία βιβλιοθήκη για ανάπτυξη προγραμμάτων, ο οποίος παράγει το αρχείο-πρόγραμμα το οποίο θα πρέπει να φορτωθεί στη μνήμη προγράμματος για να εκτελεστεί.(Ο κώδικας της βιβλιοθήκης παρατίθεται παρακάτω).

```
#ifndef _MY8BIT_H
#define _MY8BIT_H

#include <iostream>
#include <fstream>
#include <cstring>

#define     R0 "0"
#define     R1 "1"
#define     R2 "2"
#define     R3 "3"
#define     R4 "4"
#define     R5 "5"
#define     R6 "6"
#define     R7 "7"
#define     R8 "8"
#define     R9 "9"
#define     R10 "a"
#define     R11 "b"
#define     R12 "c"
```

```
#define     R13 "d"
#define     R14 "e"
#define     R15 "f"
#define Rout "e"
#define Rin "f"

using namespace std;



class Program{
      public:
      ofstream file;
      int count =0;
      string filename;

      Program(){
            cout<<"Name of the Program? (no spaces PLEASE!!)"<<endl;
            cin>>filename;
            file.open(filename);
      file<<"v2.0 raw\n";
      }

void COPY(string RA, string RC);
void ADD(string RA, string RB, string RC);
void SUB(string RA, string RB, string RC);
void ADI(string RA, string RC);
void SUBR(string RA, string RB, string RC);
void LSR(string RA, string RC);
void MUL(string RA, string RB, string RC);
void NOT(string RA, string RC);
void AND(string RA, string RB, string RC);
void NAND(string RA, string RB, string RC);
void OR(string RA, string RB, string RC);
void NOTOR(string RA, string RB, string RC);
void ORNOT(string RA, string RB, string RC);
void NOR(string RA, string RB, string RC);
void XOR(string RA, string RB, string RC);
void XNOR(string RA, string RB, string RC);
void LOAD(string RA, string RB);
void LOADADR(string RA, string RB, string ADDRESS);
void LOADLIT(string RA,  string DATA);
void LSLOAD(string RA, string RC);
void RSLOAD(string RA, string RC);
void CRSLOAD(string RA, string RC);
void STORE(string RA, string RB);
void NOP();
void BRE(string D);
void BRZ(string D);
void BRC(string D);
void BRN(string D);
void BRO(string D);
void JMP(string D);
void GOTO(string D);
```

```
void MBRE(string D);
void MBRZ(string D);
void MBRC(string D);
void MBRN(string D);
void MBRO(string D);
void MJMP(string D);
void SCANCH();
void PRINTCH(string RA);
void end();
void PRINT(string s);

};

void Program::COPY(string RA, string RC){
      file<<"00"<<RA<<"0"<<RC<<"000 ";
      count++;
      if(count%8==0){
      file<<"\n";
      }
}

void Program::ADD(string RA, string RB, string RC){
      file<<"01"<<RA<<RB<<RC<<"000 ";
      count++;
      if(count%8==0){
      file<<"\n";
      }
}

void Program::SUB(string RA, string RB, string RC){
      file<<"02"<<RA<<RB<<RC<<"000 ";
      count++;
      if(count%8==0){
      file<<"\n";
      }
}

void Program::ADI(string RA, string RC){
      file<<"03"<<RA<<"0"<<RC<<"000 ";
      count++;
      if(count%8==0){
      file<<"\n";
      }
}

void Program::SUBR(string RA, string RB, string RC){
      file<<"04"<<RA<<RB<<RC<<"000 ";
      count++;
      if(count%8==0){
      file<<"\n";
      }
}

void Program::LSR(string RA, string RC){
```

```
        file<<"05"<<RA<<"0"<<RC<<"000 ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}

void Program::MUL(string RA, string RB, string RC){
        file<<"06"<<RA<<RB<<RC<<"000 ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}

void Program::NOT(string RA, string RC){
        file<<"07"<<RA<<"0"<<RC<<"000 ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::AND(string RA, string RB, string RC){
        file<<"08"<<RA<<RB<<RC<<"000 ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::NAND(string RA, string RB, string RC){
        file<<"09"<<RA<<RB<<RC<<"000 ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::OR(string RA, string RB, string RC){
        file<<"0a"<<RA<<RB<<RC<<"000 ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::NOTOR(string RA, string RB, string RC){
        file<<"0b"<<RA<<RB<<RC<<"000";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::ORNOT(string RA, string RB, string RC){
        file<<"0c"<<RA<<RB<<RC<<"000 ";
        count++;
        if(count%8==0){
```

```
	file<<"\n";
	}
}
void Program::NOR(string RA, string RB, string RC){
	file<<"0d"<<RA<<RB<<RC<<"000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::XOR(string RA, string RB, string RC){
	file<<"0e"<<RA<<RB<<RC<<"000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::XNOR(string RA, string RB, string RC){
	file<<"0f"<<RA<<RB<<RC<<"000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}

void Program::LOAD(string RA, string RB){
	file<<"10"<<RA<<RB<<"0000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::LOADADR(string RA, string RB, string ADDRESS){
	file<<"11"<<RA<<RB<<ADDRESS<<"00 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::LOADLIT(string RA,  string DATA){
	file<<"13"<<RA<<"0"<<DATA<<"00 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::LSLOAD(string RA, string RC){
	file<<"51"<<RA<<"0"<<RC<<"000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::RSLOAD(string RA, string RC){
```

```
	file<<"52"<<RA<<"0"<<RC<<"000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::CRSLOAD(string RA, string RC){
	file<<"53"<<RA<<"0"<<RC<<"000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}

void Program::STORE(string RA, string RB){
	file<<"40"<<RA<<RB<<"0000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}

void Program::NOP(){
	file<<"7f"<<"000000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}

void Program::BRE(string D){
	file<<"30"<<"00"<<D<<" ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::BRZ(string D){
	file<<"31"<<"00"<<D<<" ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::BRC(string D){
	file<<"32"<<"00"<<D<<" ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::BRN(string D){
	file<<"33"<<"00"<<D<<" ";
	count++;
```

```
        if(count%8==0){
        file<<"\n";
        }
}
void Program::BRO(string D){
        file<<"34"<<"00"<<D<<" ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::JMP(string D){
        file<<"35"<<"00"<<D<<" ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::GOTO(string D){
        file<<"36"<<"00"<<D<<" ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::MBRE(string D){
        file<<"70"<<"00"<<D<<" ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::MBRZ(string D){
        file<<"71"<<"00"<<D<<" ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::MBRC(string D){
        file<<"72"<<"00"<<D<<" ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::MBRN(string D){
        file<<"73"<<"00"<<D<<" ";
        count++;
        if(count%8==0){
        file<<"\n";
        }
}
void Program::MBRO(string D){
```

```
	file<<"74"<<"00"<<D<<" ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::MJMP(string D){
	file<<"75"<<"00"<<D<<" ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}

void Program::SCANCH(){
	file<<"23"<<"0"<<"e"<<"0000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}

void Program::PRINTCH(string RA){
	file<<"22"<<RA<<"0"<<"f"<<"000 ";
	count++;
	if(count%8==0){
	file<<"\n";
	}
}
void Program::end(){
	file.close();
	float perc;
	perc = ((float)count/65536)*100;
	system("cls");
	cout<<"Succesful Compilation!!!\n"<<endl;
	cout<<count<<"B of total 65.536B (64KB) ["<<perc<<"%] of
memory Used"<<endl;

}


#endif
```

Επόμενα βήματα για το Project

Στο επόμενο διάστημα αναμένεται το Project να αναβαθμιστεί με νέες λειτουργίες και χαρακτηριστικά ώστε να προσεγγίζει καλύτερα τους εκπαιδευτικούς στόχους του μαθήματος.

Για παράδειγμα:

1. Μετατροπή σε 32-Bit CPU.

2. Αλλαγή εκτέλεσης εντολών από "Εκτέλεση κάθε εντολής σε ένα κύκλο ρολογιού" σε "Εκτέλεση κάθε εντολής σε περισσότερους από ένα κύκλους ρολογιού".

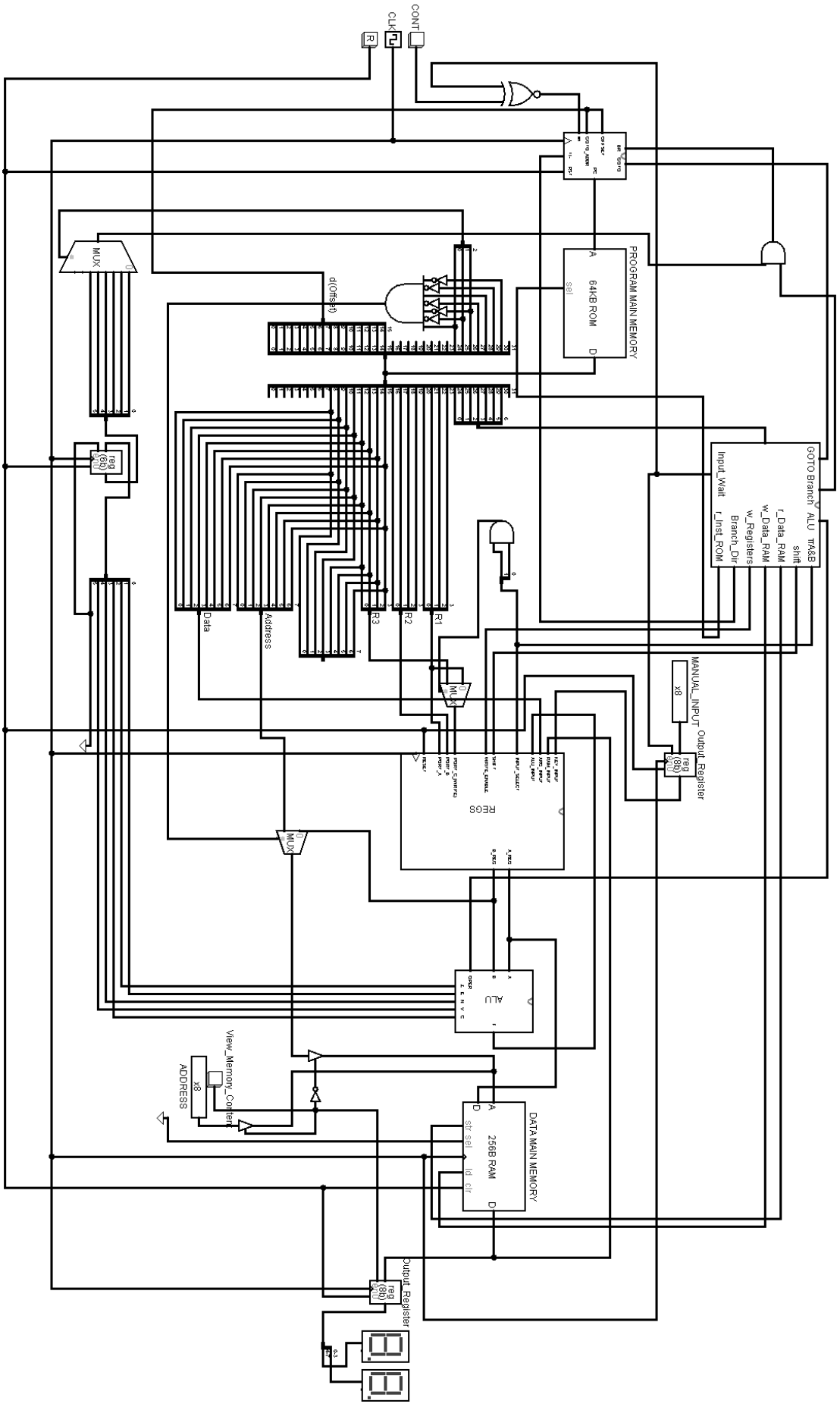
3. Μετατροπή της Κύριας Μνήμης σε Κοινή Μνήμη Δεδομένων και Εντολών.

4. Κατασκευή νέου Compiler βασιζόμενου στα εργαλεία Bison και Flex για ευκολότερο προγραμματισμό της CPU.

5. Τελειοποίηση της CPU με βάση τα ζητούμενα του αντίστοιχου καθηγητή.

6. Δημιουργία φυλλαδίου εργαστηριακών ασκήσεων σε συνεργασία με τον αντίστοιχο καθηγητή.

(7.** Έρευνα για την δημιουργία υποτυπώδους Κρυφής Μνήμης ενός επιπέδου για καλύτερη κατανόηση του αντίστοιχου κεφαλαίου.**)


CONT
CLK
PROGRAM MAIN MEMORY
64KB ROM
sel
A
D
MUX
d(Offset)
reg (8b)
Data
Address
R3
R2
R1
GOTO Branch
ALU
mA&B
shift
r_Data_RAM
w_Data_RAM
w_Registers
Branch_Dir
r_Inst_ROM
Input_Wait
MANUAL_INPUT
x8
Output_Register
REGS
ALU
View_Memory_Content
ADDRESS
DATA MAIN MEMORY
256B RAM
str sel
ld clr
s3