Σχεδίαση και υλοποίηση ενός μικροεπεξεργαστή βασισμένο στην μικροαρχιτεκτονική ARMv6 σε γλώσσα περιγραφής υλικού (verilog)

Παναγιώτης Σκριμπόνης ΑΕΜ:1256
skrimpon@inf.uth.gr

Εισαγωγή

Ο σκοπός αυτής της εργασίας ήταν η σχεδίαση και υλοποίηση ενός μικροεπεξεργαστή βασισμένο στην μικρόαρχιτεκτονική ARMv6 σε γλώσσα περιγραφής υλικού (verilog), καθώς και ενός compiler ο οποίος θα μπορούσε να διαβάζει την assembly και να βγάζει γλώσσα μηχανής σε γλώσσα C++. Ο μικροεπεξεργαστής μπορεί να εκτελεί αριθμητικές/λογικές εντολές, εντολές πολλαπλασιασμού/διαίρεσης, εντολές branch/jump, καθώς και load/store.

Μικροεπεξεργαστής

Ο Μικροεπεξεργαστής υποστηρίζει το εξής σύνολο εντολών:

Εντολές αριθμητικής, λογικής, σύγκρισης,ελέγχου και μεταφοράς																															
Condition				0	0	I	Opcode				S	Rn				Rd				Operand 2											
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
Εντολές πολλαπλασιασμού και πολλαπλασιασμού-πρόσθεσης																															
Condition				0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm			
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
Εντολές διαίρεσης και διαίρεσης-πρόσθεσης																															
Condition				1	1	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm			
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
Εντολές απλής λέξης ή byte από/προς τη μνήμη																															
Condition				0	1	I	P	U	B	W	L	Rn				Rd				Offset											
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
Εντολές διακλάδωσης και διακλάδωσης-σύνδεσης																															
Condition				1	0	1	K	Offset																							
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

Ολες οι εντολές στην κωδικοποίηση τους έχουν ένα πεδίο συνθήκης συφωνα με το οποίο αποφασίζεται από τον μικροεπεξεργαστή εαν θα εκτελεστεί η εντολή:

Πεδίο Συνθήκης				**Επίθεμα Συνθήκης**	**Όνομα**	**Έλεγχος Κωδικού Συνθήκης**
b_{31}	**b_{30}**	**b_{29}**	**b_{28}**			
0	0	0	0	EQ	Equal (zero)	Z = 1
0	0	0	1	NE	Not Equal to Zero	Z = 0
0	0	1	0	CS	Carry set	C = 1
0	0	1	1	CC	Carry clear	C = 0
0	1	0	0	MI	Minus	N = 1
0	1	0	1	PL	Plus	N = 0
0	1	1	0	VS	Overflow	V = 1
0	1	1	1	VC	No Overflow	V = 0
1	0	0	0	HI	Unsigned Higher	$\bar{C} \vee Z = 0$
1	0	0	1	LS	Unsigned Lower or same	$\bar{C} \vee Z = 1$
1	0	1	0	GE	Signed greater than or equal	$\bar{N} \oplus V = 0$
1	0	1	1	LT	Signed less than	$\bar{N} \oplus V = 1$
1	1	0	0	GT	Signed greater than	$Z \vee (\bar{N} \ominus V) = 0$
1	1	0	1	LE	Signed less than or equal	$Z \vee (\bar{N} \ominus V) = 1$
1	1	1	0	AL	Always	
1	1	1	1		Not used	

Αριθμητικες και Λογικές Εντολές

Οι αριθμητικές και λογικές πράξεις, όπως επίσης και οι πράξεις σύγκρισης, ελέγχου και μεταφοράς επιτελούνται από εντολές της μορφής A. Το πρώτο όρισμα εμπεριέχεται στον καταχωρητή Rn. Το δεύτερο όρισμα είτε περιέχεται στον καταχωρητή Rm, είτε είναι ένα μη προσημασμένο 8-bit άμεσο όρισμα, κάτι που καταδικνείεται από το I (25-bit). Το αποτέλεσμα της πράξης καθορίζεται από το opcode(24,…,21bit) και τοποθετείται στον καταχωρητή Rd. Εαν το S(20-bit) είναι 1, οι σημαίες των κωδικών συνθήκης θα επηρεαστούν από το αποτέλεσμα, αλλιώς αν το S είναι 0 θα παραμείνουν ως έχουν.
Για να σχεδιαστεί το μονοπάτι αυτό χρησιμοποιήθηκαν οι συνδιαστικοί αλγόριθμοι του Barrel Shifter καθως και του Carry Look-Ahead Adder.

Opcode				**Όνομα**	**Εντολή**	**Αλλάζουν οι εξής σημαίες**			
B_{24}	**b_{23}**	**b_{22}**	**b_{21}**			**N**	**Z**	**V**	**C**
0	0	0	0	AND	$Rd \leftarrow [Rn] \wedge Oper2$	X	X		X
0	0	0	1	EOR	$Rd \leftarrow [Rn] \oplus Oper2$	X	X		X
0	0	1	0	SUB	$Rd \leftarrow [Rn] - Oper2$	X	X	X	X
0	0	1	1	RSB	$Rd \leftarrow Oper2 - [Rn]$	X	X	X	X
0	1	0	0	ADD	$Rd \leftarrow [Rn] + Oper2$	X	X	X	X
0	1	0	1	ADC	$Rd \leftarrow [Rn] + Oper2 + [C]$	X	X	X	X
0	1	1	0	SBC	$Rd \leftarrow [Rn] - Oper2 + [C] - 1$	X	X	X	X
0	1	1	1	RSC	$Rd \leftarrow Oper2 - [Rn] + [C] - 1$	X	X	X	X
1	0	0	0	TST	$[Rn] \wedge Oper2$	X	X		X
1	0	0	1	TEQ	$[Rn] \oplus Oper2$	X	X		X
1	0	1	0	CMP	$[Rn] - Oper2$	X	X	X	X
1	0	1	1	CMN	$[Rn] + Oper2$	X	X	X	X
1	1	0	0	ORR	$Rd \leftarrow [Rn] \vee Oper2$	X	X		X
1	1	0	1	MOV	$Rd \leftarrow Oper2$	X	X		X
1	1	1	0	BIC	$Rd \leftarrow [Rn] \wedge \neg Oper2$	X	X		X
1	1	1	1	MVN	$Rd \leftarrow \neg Oper2$	X	X		X

Εντολές Πολλαπλασιασμού

Στο μονοπάτι αυτό χρησιμοποιήθηκαν οι συνδιαστικοί αλγόριθμοι του πολλαπλασιασμού Booth με 3-bit έλεγχο και του Carry Look-Ahead Adder, για την υποστήριξη των εντολών

A	**Όνομα**	**Εντολή**	**Αλλάζουν οι εξής σημαίες**			
b_{21}			**N**	**Z**	**V**	**C**
0	MUL	$Rd \leftarrow [Rm] * [Rs]$	X	X		X
1	MLA	$Rd \leftarrow [Rm] * [Rs] + [Rn]$	X	X		X

Εντολές Διαίρεσης

Στο μονοπάτι αυτό χρησιμοποιήθηκαν οι συνδιαστικοί αλγόριθμοι της διαίρεσης χωρίς επαναφορά καθως και του Carry Look-Ahead Adder, για την υποστήριξη των εντολών:

A	**Όνομα**	**Εντολή**	**Αλλάζουν οι εξής σημαίες**			
b_{21}			**N**	**Z**	**V**	**C**
0	DIV	$Rd \leftarrow [Rm]/[Rs]$	X	X		X
1	DVA	$Rd \leftarrow [Rm]/[Rs] + [Rn]$	X	X		X

Εντολές Φορτωσης και αποθήκευσης δεδομενων

B L		Όνομα	Εντολή
b_{22}	b_{20}		
0	0	LDR	$Rd \leftarrow [EA]$
0	1	LDB	$Rd \leftarrow [EA]$
1	0	SDR	$[EA] \leftarrow Rd$
1	1	SDB	$[EA] \leftarrow Rd$

Compiler

Για να φτιάξουμε τον μεταγλωτιστή χρησιμοποιήσαμε τον πίνακα με τα opcodes της κάθε εντολής για να κάνουμε parse ένα αρχείο με assebly, το output χρησιμοποιείται για να κάνει initiliaze την μνήμη

Future Work

Ως μελλοντική δουλειά θα ήθελα να αλλάξω το interface με το οποίο ο επεξεργαστής κάνει προσπέλαση εντολές και δεδομένα. Ειδικά για τα δεδομένα θα ήθελα να βάλω ένα AXI4-Master Interface έτσι ώστε να μπορεί να έχει κάποια AXI4-slaves. Ιδανικά θα ήθελα να μπορώ να τον τρέξω σε μια πλατφόρμα επαναδιατασόμενης λογικής (FPGA). Επίσης, επειδή τα μονοπάτια δεδομένων είναι ανεξάρτητα μεταξύ τους θα ήταν ενδιαφέρον να εφαρμόσω των αλγόριθμο του Tomasulo.