Final Report
ELEC6027: VLSI Design Project

Format title page

Team R4
Henry Lovett (hl13g10)
Ashley Robinson (ajr2g10)
Martin Wearn (mw20g10)
Anusha Reddy (arr1g13)

24th April, 2014

Todo list

Contents

Chapter 1

Introduction

INCOMPLETE CHAPTER: Introduction

Overview of the report

Chapter 2

Architecture

INCOMPLETE CHAPTER: Architecture

Design of the datapath architecture.
Refer to the research done and how this influenced the design
Incl. diagram

Chapter 3

Instruction Set

Design of instruction set

Allocation of opcodes etc

Refer to research

ISA novelties

In designing the instruction set architecture (ISA) emphasis was put on creating a complete set of basic operations which could be used to implement any program. This gave rise to a RISC based architecture since they have a small number of instructions and are optimized for a smaller chip area.

Expand basic ISA design considerations

Since a 16-bit microprocessor was to be designed, it was decided to base the system on the ARM Thumb architecture. This is a subset of the main 32-bit ARM instruction set which contains a suitably complete set of instructions. However it included a number of operations which take advantage of the ARM's 32-bit datapath, so any high register operations were removed. Change of state, sign extension and debugging instructions, among others, were also removed for simplification or because they were not necessary. This produced the original ISA made up of instructions 1-4, 6-10, 12-16, 20-24, 27-32, 35 and 36 as noted in the summary table in Appendix A. While instructions 5 and 11 were added to support use of carry flag with an immediate value. 17, 18 and 19 are included to form a complete logic set. 25 and 26 are for loading an initial value to any general purpose register. Instruction 33 is included from the SPARC ISA for returning from a procedure. Instruction 34 enables a control jump to anywhere in 2^{16} memory locations. With

instructions 37-41 were added for support of a single interrupt.

	Instruction Type	Sub-Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A1	**Data Manipulation**	**Register**	Opcode					Rd			Ra			Rb			X	X
A2		**Immediate**						Rd			Ra			imm4/5				
B	**Byte Immediate**		Opcode					Rd			imm8							
C	**Data Transfer**		0	LS	0	0	0	Rd			Ra			imm5				
D1	**Control Transfer**	**Others**	1	1	1	1	0	Cond.			imm8							
D2		**Jump**									Ra			imm5				
E	**Stack Operations**		0	U	0	0	1	L	X	X	Ra			0	0	0	0	1
F	**Interrupts**		1	1	0	0	1	ICond.			1	1	1	X	X	X	X	X

Table 3.1: General Instruction Formatting

To promote orthogonality, the instruction formatting for data manipulation operations followed a similar structure to the ARM Thumb, as shown in Table 3.1. Which was adapted to create all other types of formatting, and reordered to ensure immediate values were always on the far right of the instruction. This was to make sign extension of immediate values in the datapath easier since they are always in the same location in the instruction.

Chapter 4

Design and Implementation

INCOMPLETE CHAPTER: Implementation

4.1 Register Block

Design of whole module, including circuit diagram
- Use of hierarchy / blocks - i.e. bit sliced, decoder
- Design of slice,
- Design of decoder,
- Design of block,
- Layout in silicon

4.2 Program Counter

Design of whole module, including circuit diagram
- Use of hierarchy / blocks - i.e. bit sliced, decoder
- Design of slice,
- Design of decoder,
- Design of block,
- Layout in silicon

4.3 Instruction Register

Design of whole module, including circuit diagram
- Use of hierarchy / blocks - i.e. bit sliced, decoder
- Design of slice,
- Design of decoder,
- Design of block,
- Layout in silicon

4.4 Arithmetic Logic Unit

Design of whole module, including circuit diagram
- Use of hierarchy / blocks - i.e. bit sliced, decoder
- Design of slice,
- Design of decoder,
- Design of block,
- Layout in silicon

4.5 Datapath

Design of whole module, including circuit diagram
- Use of hierarchy / blocks - i.e. bit sliced, decoder
- Design of slice,
- Design of decoder (slice 17),
- Design of block,
- Layout in silicon

4.6 Controller

Design of - simple statemachine?
- Control signals - description, use of type defs?
- Description of main states:
- Fetch
- Execute
- Interrupt
- Implementation of interrupts (flags, enable. . .)

Synthesis and layout - I/O config, magic vs Ledit maybe?

4.7 CPU

Overall layout
- pad ring size
- positioning of control and datapath
- power routing
- anything else?

Chapter 5

Testing

INCOMPLETE CHAPTER: Testing

5.1 Register Block

Include Sub tests - of slice and decoder (if app)
- Explain tests - what is done
- why it is done.
- How it verifies everything - why it is complete
- Show simulation results

5.2 Program Counter

Include Sub tests - of slice and decoder (if app)
- Explain tests - what is done
- why it is done.
- How it verifies everything - why it is complete
- Show simulation results

5.3 Instruction Register

Include Sub tests - of slice and decoder (if app)
- Explain tests - what is done

why it is done.
How it verifies everything - why it is complete
Show simulation results

5.4 Arithmetic Logic Unit

Include Sub tests - of slice and decoder (if app)
Explain tests - what is done
why it is done.
How it verifies everything - why it is complete
Show simulation results

5.5 Datapath

Include Sub tests - of slice and decoder (if app)
Explain tests - what is done
why it is done.
How it verifies everything - why it is complete
Show simulation results

5.6 Controller

Include Sub tests - of slice and decoder (if app)
Explain tests - what is done
why it is done.
How it verifies everything - why it is complete
Show simulation results

5.7 CPU

Include Sub tests - of slice and decoder (if app)
Explain tests - what is done
why it is done.
How it verifies everything - why it is complete
Show simulation results

Chapter 6

Conclusion

INCOMPLETE CHAPTER: Conclusion

Generic concluding marks

Appendix A

Instruction Set Summary

INCOMPLETE CHAPTER: Instruction Set Summary

	Mnemonic	Syntax	Semantics	Flags	Encoding	Opcode	Cond.
1	ADD	ADD Rd, Ra, Rb	Rd ← Ra + Rb	c,v,n,z	A	00010	-
2	ADDI	ADDI Rd, Ra, #imm5	Rd ← Ra + imm5	c,v,n,z	A	00110	-
3	ADDIB	ADDIB Rd, #imm8	Rd ← Rd + imm8	c,v,n,z	B	00011	-
4	ADC	ADC Rd, Ra, Rb	Rd ← Ra + Rb + c	c,v,n,z	A	00100	-
5	ADCI	ADCI Rd, Ra, #imm5	Rd ← Ra + imm5 + c	c,v,n,z	A	00101	-
6	NEG	NEG Rd, Ra	Rd ← 0 - Ra	c,v,n,z	A	11010	-
7	SUB	SUB Rd, Ra, Rb	Rd ← Ra - Rb	c,v,n,z	A	01010	-
8	SUBI	SUBI Rd, Ra, #imm5	Rd ← Ra - imm5	c,v,n,z	A	01110	-
9	SUBIB	SUBIB Rd, #imm8	Rd ← Rd - imm8	c,v,n,z	B	01011	-
10	SUC	SUC Rd, Ra, Rb	Rd ← Ra - Rb - c	c,v,n,z	A	01100	-
11	SUCI	SUCI Rd, Ra, #imm5	Rd ← Ra - imm5 - c	c,v,n,z	A	01101	-
12	CMP	CMP Ra, Rb	Rd ← Ra - Rb	c,v,n,z	A	00111	-
13	CMPI	CMPI Ra, #imm5	Rd ← Ra - imm5	c,v,n,z	A	01111	-
14	AND	AND Rd, Ra, Rb	Rd ← Ra `AND` Rb	n,z	A	10000	-
15	OR	OR Rd, Ra, Rb	Rd ← Ra `OR` Rb	n,z	A	10001	-
16	XOR	XOR Rd, Ra, Rb	Rd ← Ra `XOR` Rb	n,z	A	10011	-
17	NOT	NOT Rd, Ra	Rd ← `NOT` Ra	n,z	A	10010	-
18	NAND	NAND Rd, Ra, Rb	Rd ← Ra `NAND` Rb	n,z	A	10110	-
19	NOR	NOR Rd, Ra, Rb	Rd ← Ra `NOR` Rb	n,z	A	10111	-
20	LSL	LSL Rd, Ra, #imm4	Rd ← Ra << imm4	n,z	A	11111	-
21	LSR	LSR Rd, Ra, #imm4	Rd ← Ra >> imm4	n,z	A	11101	-
22	ASR	ASR Rd, Ra, #imm4	Rd ← Ra >>> imm4	n,z	A	11100	-
23	LDW	LDW Rd, [Ra, #imm5]	Rd ← Mem[Ra + imm5]	-	C	00000	-
24	STW	STW Rd, [Ra, #imm5]	Mem[Ra + imm5] ← Rd	-	C	01000	-
25	LUI	LUI Rd, #imm8	Rd ← imm8, 0	-	B	10100	-
26	LLI	LLI Rd, #imm8	Rd ← Rd[15:8], imm8	-	B	10101	-
27	BR	BR LABEL	PC ← PC + imm8	-	D	-	000
28	BNE	BNE LABEL	(z==0)?PC ← PC + imm8	-	D	-	110
29	BE	BE LABEL	(z==1)?PC ← PC + imm8	-	D	-	111
30	BLT	BLT LABEL	(n&∼v `OR` ∼n&v)?PC ← PC + imm8	-	D	-	100
31	BGE	BGE LABEL	(n&v `OR` ∼n&∼v)?PC ← PC + imm8	-	D	-	101
32	BWL	BWL LABEL	LR ← PC + 1; PC ← PC + imm8	-	D	-	011
33	RET	RET	PC ← LR	-	D	-	010
34	JMP	JMP Ra, #imm5	PC ← Ra + imm5	-	D	-	001
35	PUSH	PUSH Ra PUSH LR	Mem[R7] ← Ra; R7 ← R7 - 1; Mem[R7] ← RL; R7 ← R7 - 1;	-	E	-	-
36	POP	POP Ra POP LR	R7 ← R7 + 1; Mem[R7] ← Ra; R7 ← R7 + 1; Mem[R7] ← RL;	-	E	-	-
37	RETI	RETI	PC ← Mem[R7]	-	F	-	000
38	ENAI	ENAI	IntEnFlag ← 1	-	F	-	001
39	DISI	DISI	IntEnFlag ← 0	-	F	-	010
40	STF	STF	Mem[R7] ← Flags; R7 ← R7 - 1;	-	F	-	011
41	LDF	LDF	R7 ← R7 + 1; Mem[R7] ← Flags;	c,v,n,z	F	-	100

Appendix B

Project Management

INCOMPLETE CHAPTER: Project Management

Use of git
regular meetings

Appendix C

Division of Labour

	Task	Percentage Effort on task			
	ECSID:	hl13g10	ajr2g10	mw20g10	arr1g13
1	Initial Design	100	0	0	0
2	Verilog Behavioural Model	100	0	0	0
3	Multiply Program	100	0	0	0
4	Magic Datapath	100	0	0	0
4.1	Registers	100	0	0	0
4.2	Program Counter	100	0	0	0
4.3	Instruction Register	100	0	0	0
4.4	ALU	100	0	0	0
5	Verilog Cross Simulation	100	0	0	0
6	Control Unit Synthesis	100	0	0	0
7	Magic Control Unit	100	0	0	0
8	Final Floorplanning, Placement and Routing	100	0	0	0
9	Factorial Program	100	0	0	0
10	Random Program	100	0	0	0
11	Interrupt Program	100	0	0	0
11	Verilog Final Simulations and Cadence DRC	100	0	0	0
12	Assembler	100	0	0	0
13	Programmer's Guide Documentation	100	0	0	0
13.1	Architecture	100	0	0	0
13.2	Assembler	100	0	0	0
13.3	Instruction Set	100	0	0	0
13.4	Programming Tips	100	0	0	0
13.5	Programs	100	0	0	0
13.6	Register Description	100	0	0	0
13.7	Simulation	100	0	0	0
14	Final Report	100	0	0	0
14.1	Introduction	100	0	0	0
14.2	Architecture	100	0	0	0
14.3	Instruction Set	100	0	0	0
14.4	Implementation	100	0	0	0
14.5	Testing	100	0	0	0
14.6	Conclusion	100	0	0	0
14.7	Project Management	100	0	0	0
	OVERALL EFFORT	100	0	0	0