

ELEC6027 - VLSI Design Project : Programmers Guide

Team R4

25th April, 2014

Todo list

■ HSL - this doesn't sound right. Maybe "transfer of program flow". Not sure on the use of the word "control" but i know it is the technical term	6
■ HSL - this is a bit too short. Surely there is more to say about it? .	6
■ HSL - I remember Iain saying something about the instruction for- mats being called A1 / A2. I don't see a problem personally as I can't remember exactly what he said!	7
■ surely this we can just use the XOR instruction	55
■ Maybe change to IEEE symbols if we have time	56
■ A register window could also be done for this section too	57
■ Sim.py needs a fair bit of change. If we have time, this could be altered to use Iains dir structure. This section highlights how to use his script and our assembler.	57
■ Make these more accurate when AJR has finished playing around .	59
■ how to run scan path sim too	59
■ need to mention extracting? i wouldn't say so as it should already be done and that can't really change.	59
■ and TCL?	59
■ check that the dissembler works in all the simulations (it doesn't...) .	59

Contents

1	Introduction	5
2	Architecture	5
3	Register Description	5
4	Instruction Set	6
4.1	General Instruction Formatting	7
4.2	ADD Add Word	9
4.3	ADDI Add Immediate	10
4.4	ADDIB Add Immediate Byte	11
4.5	ADC Add Word With Carry	12
4.6	ADCI Add Immediate With Carry	13
4.7	NEG Negate Word	14

4.8	SUB	Subtract Word	15
4.9	SUBI	Subtract Immediate	16
4.10	SUBIB	Subtract Immediate Byte	17
4.11	SUC	Subtract Word With Carry	18
4.12	SUCI	Subtract Immediate With Carry	19
4.13	CMP	Compare Word	20
4.14	CMPI	Compare Immediate	21
4.15	AND	Logical AND	22
4.16	OR	Logical OR	23
4.17	XOR	Logical XOR	24
4.18	NOT	Logical NOT	25
4.19	NAND	Logical NAND	26
4.20	NOR	Logical NOR	27
4.21	LSL	Logical Shift Left	28
4.22	LSR	Logical Shift Right	29
4.23	ASR	Arithmetic Shift Right	30
4.24	LDW	Load Word	31
4.25	STW	Store Word	32
4.26	LUI	Load Upper Immediate	33
4.27	LLI	Load Lower Immediate	34
4.28	BR	Branch Always	35
4.29	BNE	Branch If Not Equal	36
4.30	BE	Branch If Equal	37
4.31	BLT	Branch If Less Than	38
4.32	BGE	Branch If Greater Than Or Equal	39
4.33	BWL	Branch With Link	40
4.34	RET	Return	41
4.35	JMP	Jump	42
4.36	PUSH	Push From Stack	43
4.37	POP	Pop From Stack	44
4.38	RETI	Return From Interrupt	45
4.39	ENAI	Enable Interrupts	46
4.40	DISI	Disable Interrupts	47
4.41	STF	Store Status Flags	48
4.42	LDF	Load Status Flags	49
5	Programming Tips		50

6	Assembler	50
6.1	Instruction Formatting	50
6.2	Assembler Directives	51
6.3	Running The Assembler	51
6.4	Error Messages	53
7	Programs	53
7.1	Multiply	54
7.2	Factorial	55
7.3	Random	55
7.4	Interrupt	56
8	Simulation	57
8.1	Running the simulations	57
A	Code Listings	60
A.1	Multiply	60
A.2	Multiply Unrolled	61
A.3	Factorial	66
A.4	Random	70
A.5	Interrupt	71

1 Introduction

Lorem Ipsum. . .

2 Architecture

Lorem Ipsum. . .

3 Register Description

Lorem Ipsum. . .

4 Instruction Set

The complete instruction set architecture includes a number of instructions for performing calculations on data, memory access, transfer of control within a program and interrupt handling.

All instructions implemented by this architecture fall into one of 6 groups, categorized as follows:

- Data Manipulation - Arithmetic, Logical, Shifting
- Byte Immediate - Arithmetic, Byte Load
- Data Transfer - Memory Access
- Control Transfer - (Un)conditional Branching
- Stack Operations - Push, Pop
- Interrupts - Enabling, Status Storage, Returning

There is only one addressing mode associated with each instruction, generally following these groupings:

- Data Manipulation - Register-Register, Register-Immediate
- Byte Immediate - Register-Immediate
- Data Transfer - Base Plus Offset
- Control Transfer - PC Relative, Register-Indirect, Base Plus Offset
- Stack Operations - Register-Indirect Preincrement/Postdecrement
- Interrupts - Register-Indirect Preincrement/Postdecrement

HSL
- this doesn't sound right. Maybe "transfer of program flow". Not sure on the use of the word "control" but i know it is the technical term

HSL - this is a bit too short. Surely there is more to say about it?

4.1 General Instruction Formatting

HSL - I remember Iain saying something about the instruction formats being called A1 / A2. I don't see a problem personally as I can't remember exactly what he said!

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	

Instruction Field Definitions

Opcode: Operation code as defined for each instruction

Rd: Destination Register

Ra: Source register 1

Rb: Source register 2

immX: Immediate value of length X

Cond.: Branching condition code as defined for branch instructions

ICond.: Interrupt instruction code as defined for interrupt instructions

LS: 0=Load Data, 1=Store Data

U: 1=PUSH, 0=POP

L: 1=Use Link Register, 0=Use GPR

Pseudocode Notation

Symbol	Meaning
\leftarrow, \rightarrow	Assignment
Result[x]	Bit x of result
Ra[$x : y$]	Bit range from x to y of register Ra
$+Ra$	Positive value in Register Ra
$-Ra$	Negative value in Register Ra
$<$	Numerically greater than
$>$	Numerically less than
$<<$	Logical shift left
$>>$	Logical shift right
$>>>$	arithmetic shift right
Mem[val]	Data at memory location with address val
$\{x, y\}$	Contatenation of x and y to form a 16-bit value
$(cond)?$	Operation performed if $cond$ evaluates to true
$!$	Bitwise Negation

Use of the word UNPREDICTABLE indicates that the resultant flag value after operation execution will not be indicative of the ALU result. Instead its value will correspond to the result of an undefined arithmetic operation and as such should not be used.

4.2 ADD

Add Word

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	Rd			Ra			Rb		X	X	

Syntax

ADD Rd, Ra, Rb

eg. ADD R5, R3, R2

Operation

$Rd \leftarrow Ra + Rb$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +Rb \text{ and } -\text{Result}) \text{ or } (-Ra \text{ and } -Rb \text{ and } +\text{Result}) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or } (\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Ra] is added to the 16-bit word in GPR[Rb] and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.3 ADDI

Add Immediate

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	Rd			Ra			imm5				

Syntax

ADDI Rd, Ra, #imm5

eg. ADDI R5, R3, #7

Operation

$Rd \leftarrow Ra + \#imm5$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +\#imm5 \text{ and } -\text{Result}) \text{ or}$
 $(-Ra \text{ and } -\#imm5 \text{ and } +\text{Result}) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or}$
 $(\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Ra] is added to the sign-extended 5-bit value given in the instruction and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.4 ADDIB

Add Immediate Byte

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	Rd			imm8							

Syntax

ADDIB Rd, #imm8

eg. ADDIB R5, #93

Operation

$Rd \leftarrow Rd + \#imm8$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Rd \text{ and } +\#imm8 \text{ and } -\text{Result}) \text{ or}$
 $(-Rd \text{ and } -\#imm8 \text{ and } +\text{Result}) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or}$
 $(\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Rd] is added to the sign-extended 8-bit value given in the instruction and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.5 ADC

Add Word With Carry

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0 0 1 0 0					Rd			Ra			Rb		X X		

Syntax

ADC Rd, Ra, Rb

eg. ADC R5, R3, R2

Operation

$Rd \leftarrow Ra + Rb + C$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +(Rb+CFlag) \text{ and } -Result) \text{ or}$
 $(-Ra \text{ and } -(Rb+CFlag) \text{ and } +Result) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (Result > 2^{16} - 1) \text{ or}$
 $(Result < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Ra] is added to the 16-bit word in GPR[Rb] with the added carry in set according to the Carry flag from previous operation, and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.6 ADCI

Add Immediate With Carry

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	Rd			Ra			imm5				

Syntax

ADCI Rd, Ra, #imm5

eg. ADCI R5, R4, #7

Operation

$Rd \leftarrow Ra + \#imm5 + C$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +(\#imm5 + CFlag) \text{ and } -Result) \text{ or}$
 $(-Ra \text{ and } -(\#imm5 + CFlag) \text{ and } +Result) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (Result > 2^{16} - 1) \text{ or}$
 $(Result < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Ra] is added to the sign-extended 5-bit value given in the instruction with carry in set according to the Carry flag from previous operation, and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.7 NEG

Negate Word

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	Rd			Ra			Rb		X	X	

Syntax

NEG Rd, Ra

eg. NEG R5, R3

Operation

$Rd \leftarrow 0 - Ra$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +Rb \text{ and } -\text{Result}) \text{ or}$
 $(-Ra \text{ and } -Rb \text{ and } +\text{Result}) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or}$
 $(\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Ra] is added to the 16-bit word in GPR[Rb] and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.8 SUB

Subtract Word

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	Rd			Ra			Rb		X	X	

Syntax

SUB Rd, Ra, Rb

eg. SUB R5, R3, R2

Operation

$Rd \leftarrow Ra - Rb$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +Rb \text{ and } -\text{Result}) \text{ or}$
 $(-Ra \text{ and } -Rb \text{ and } +\text{Result}) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or}$
 $(\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Rb] is subtracted from the 16-bit word in GPR[Ra] and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.9 SUBI

Subtract Immediate

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0 1 1 1 0					Rd			Ra			imm5				

Syntax

SUBI Rd, Ra, #imm5

eg. SUBI R5, R3, #7

Operation

$Rd \leftarrow Ra - \#imm5$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +\#imm5 \text{ and } -\text{Result}) \text{ or } (-Ra \text{ and } -\#imm5 \text{ and } +\text{Result}) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or } (\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The sign extended 5-bit value given in the instruction is subtracted from the 16-bit word in GPR[Ra] and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.10 SUBIB

Subtract Immediate Byte

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	Rd			imm8							

Syntax

SUBIB Rd, #imm8

eg. SUBIB R5, #93

Operation

$Rd \leftarrow Rd - \#imm8$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Rd \text{ and } +\#imm8 \text{ and } -\text{Result}) \text{ or}$
 $(-Rd \text{ and } -\#imm8 \text{ and } +\text{Result}) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or}$
 $(\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 8-bit immediate value given in the instruction is subtracted from the 16-bit word in GPR[Rd] and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.11 SUC

Subtract Word With Carry

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	Rd			Ra			Rb			X	X

Syntax

SUC Rd, Ra, Rb

eg. SUC R5, R3, R2

Operation

$Rd \leftarrow Ra - Rb - C$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +(Rb-CFlag) \text{ and } -Result) \text{ or}$
 $(-Ra \text{ and } -(Rb-CFlag) \text{ and } +Result) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (Result > 2^{16} - 1) \text{ or}$
 $(Result < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Rb] is subtracted from the 16-bit word in GPR[Rd] with the subtracted carry in set according to the Carry flag from previous operation, and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.12 SUCI

Subtract Immediate With Carry

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	Rd			Ra			imm5				

Syntax

SUCI Rd, Ra, #imm5

eg. SUCI R5, R4, #7

Operation

$Rd \leftarrow Ra - \#imm5 - C$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +(\#imm5-CFlag) \text{ and } -Result) \text{ or}$

$(-Ra \text{ and } -(\#imm5-CFlag) \text{ and } +Result) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (Result > 2^{16} - 1) \text{ or}$

$(Result < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 5-bit immediate value in instruction is subtracted from the 16-bit word in GPR[Ra] with the subtracted carry in set according to the Carry flag from previous operation, and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.13 CMP

Compare Word

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0 0 1 1 1					Rd			Ra			Rb		X X		

Syntax

CMP Ra, Rb

eg. CMP R3, R2

Operation

$Ra - Rb$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{if } (+Ra \text{ and } +Rb \text{ and } -\text{Result}) \text{ or}$
 $(-Ra \text{ and } -Rb \text{ and } +\text{Result}) \text{ then } 1, \text{ else } 0$

$C \leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or}$
 $(\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0$

Description

The 16-bit word in GPR[Rb] is subtracted from the 16-bit word in GPR[Ra] and the status flags are updated without saving the result.

Addressing Mode: Register-Register.

4.14 CMPI

Compare Immediate

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	Rd			Ra			imm5				

Syntax

CMPI Ra, #imm5

eg. CMPI R3, #7

Operation

Ra - #imm5

N \leftarrow if Result < 0 then 1, else 0

Z \leftarrow if Result = 0 then 1, else 0

V \leftarrow if (+Ra and +#imm5 and -Result) or

(-Ra and -#imm5 and +Result) then 1, else 0

C \leftarrow if (Result $> 2^{16} - 1$) or

(Result $< -2^{16}$) then 1, else 0

Description

The sign extended 5-bit value given in the instruction is subtracted from the 16-bit word in GPR[Ra] and the status flags are updated without saving the result.

Addressing Mode: Register-Immediate.

4.15 AND

Logical AND

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	Rd			Ra			Rb		X	X	

Syntax

AND Rd, Ra, Rb

eg. AND R5, R3, R2

Operation

$Rd \leftarrow Ra \text{ AND } Rb$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The logical AND of the 16-bit words in GPR[Ra] and GPR[Rb] is performed and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.16 OR

Logical OR

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	Rd			Ra			Rb		X	X	

Syntax

OR Rd, Ra, Rb

eg. OR R5, R3, R2

Operation

$Rd \leftarrow Ra \text{ OR } Rb$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The logical OR of the 16-bit words in GPR[Ra] and GPR[Rb] is performed and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.17 XOR

Logical XOR

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	Rd			Ra			Rb		X	X	

Syntax

XOR Rd, Ra, Rb

eg. XOR R5, R3, R2

Operation

$Rd \leftarrow Ra \text{ XOR } Rb$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The logical XOR of the 16-bit words in GPR[Ra] and GPR[Rb] is performed and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.18 NOT

Logical NOT

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	Rd			Ra			Rb		X	X	

Syntax

NOT Rd, Ra

eg. NOT R5, R3

Operation

$Rd \leftarrow \text{NOT } Ra$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The logical NOT of the 16-bit word in GPR[Ra] is performed and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.19 NAND

Logical NAND

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	Rd			Ra			Rb		X	X	

Syntax

NAND Rd, Ra, Rb

eg. NAND R5, R3, R2

Operation

$Rd \leftarrow Ra \text{ NAND } Rb$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The logical NAND of the 16-bit words in GPR[Ra] and GPR[Rb] is performed and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.20 NOR

Logical NOR

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	Rd			Ra			Rb		X	X	

Syntax

NOR Rd, Ra, Rb

eg. NOR R5, R3, R2

Operation

$Rd \leftarrow Ra \text{ NOR } Rb$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The logical NOR of the 16-bit words in GPR[Ra] and GPR[Rb] is performed and the result is placed into GPR[Rd].

Addressing Mode: Register-Register.

4.21 LSL

Logical Shift Left

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	Rd			Ra			0	imm4			

Syntax

LSL Rd, Ra, #imm4

eg. LSL R5, R3, #7

Operation

$Rd \leftarrow Ra \ll \#imm4$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The 16-bit word in GPR[Ra] is shifted left by the 4-bit amount specified in the instruction, shifting in zeros, and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.22 LSR

Logical Shift Right

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	Rd			Ra			0	imm4			

Syntax

LSR Rd, Ra, #imm4

eg. LSR R5, R3, #7

Operation

$Rd \leftarrow Ra \gg \#imm4$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The 16-bit word in GPR[Ra] is shifted right by the 4-bit amount specified in the instruction, shifting in zeros, and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.23 ASR

Arithmetic Shift Right

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	Rd			Ra			0	imm4			

Syntax

ASR Rd, Ra, #imm4

eg. ASR R5, R3, #7

Operation

$Rd \leftarrow Ra \ggg \#imm4$

$N \leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0$

$Z \leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

Description

The 16-bit word in GPR[Ra] is shifted right by the 4-bit amount specified in the instruction, shifting in the sign bit of Ra, and the result is placed into GPR[Rd].

Addressing Mode: Register-Immediate.

4.24 LDW

Load Word

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	Rd				Ra						imm5

Syntax

LDW Rd, [Ra, #imm5]

eg. LDW R5, [R3, #7]

Operation

$Rd \leftarrow \text{Mem}[Ra + \#imm5]$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Data is loaded from memory at the resultant address from addition of GPR[Ra] and the 5-bit immediate value specified in the instruction, and the result is placed into GPR[Rd].

Addressing Mode: Base Plus Offset.

4.25 STW

Store Word

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	Rd			Ra			imm5				

Syntax

STW Rd, [Ra, #imm5]

eg. STW R5, [R3, #7]

Operation

Mem [Ra + #imm5] \leftarrow Rd

N \leftarrow N

Z \leftarrow Z

V \leftarrow V

C \leftarrow C

Description

Data in GPR[Rd] is stored to memory at the resultant address from addition of GPR[Ra] and the 5-bit immediate value specified in the instruction.

Addressing Mode: Base Plus Offset.

4.26 LUI

Load Upper Immediate

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	Rd			imm8							

Syntax

LUI Rd #imm8

eg. LUI R5, #93

Operation

$Rd \leftarrow \{\#imm8, 0\}$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

The 8-bit immediate value provided in the instruction is loaded into the top half in GPR[Rd], setting the bottom half to zero.

Addressing Mode: Register-Immediate.

4.27 LLI

Load Lower Immediate

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	Rd			imm8							

Syntax

LLI Rd #imm8

eg. LLI R5, #93

Operation

$Rd \leftarrow \{Rd[15:8], \#imm8\}$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

The 8-bit immediate value provided in the instruction is loaded into the bottom half in GPR[Rd], leaving the top half unchanged.

Addressing Mode: Register-Immediate.

4.28 BR

Branch Always

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	imm8							

Syntax

BR LABEL

eg. BR .loop

Operation

$PC \leftarrow PC + \#imm8$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Unconditionally branch to the resultant address from addition of PC and the 8-bit immediate value specified in the instruction. LABEL can be both a symbolic name or a numeric value, and is capable of jumping forwards or backwards.

Addressing Mode: PC Relative.

4.29 BNE

Branch If Not Equal

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	0	imm8							

Syntax

BNE LABEL

eg. BNE .loop

Operation

$PC \leftarrow PC + \#imm8$ (z==0)?

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Conditionally branch to the resultant address from addition of PC and the 8-bit immediate value specified in the instruction if zero status flag (Z) equals zero. LABEL can be both a symbolic name or a numeric value, and is capable of jumping forwards or backwards.

Addressing Mode: PC Relative.

4.30 BE

Branch If Equal

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	1	imm8							

Syntax

BE LABEL

eg. BE .loop

Operation

$PC \leftarrow PC + \#imm8 \text{ (z==1)?}$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Conditionally branch to the resultant address from addition of PC and the 8-bit immediate value specified in the instruction if zero status flag (Z) equals one. LABEL can be both a symbolic name or a numeric value, and is capable of jumping forwards or backwards.

Addressing Mode: PC Relative.

4.31 BLT

Branch If Less Than

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	imm8							

Syntax

BLT LABEL

eg. BLT .loop

Operation

$PC \leftarrow PC + \#imm8 \text{ (n\&!v OR !n\&v)?}$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Conditionally branch to the resultant address from addition of PC and the 8-bit immediate value specified in the instruction if negative status flag and overflow status flag are not equivalent. LABEL can be both a symbolic name or a numeric value, and is capable of jumping forwards or backwards.

Addressing Mode: PC Relative.

4.32 BGE

Branch If Greater Than Or Equal

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1	imm8							

Syntax

BGE LABEL

eg. BGE .loop

Operation

$PC \leftarrow PC + \#imm8 \text{ (n\&v OR !n\&!v)?}$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Conditionally branch to the resultant address from addition of PC and the 8-bit immediate value specified in the instruction if negative status flag and overflow status flag are equivalent. LABEL can be both a symbolic name or a numeric value, and is capable of jumping forwards or backwards.

Addressing Mode: PC Relative.

4.33 BWL

Branch With Link

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	imm8							

Syntax

BWL LABEL

eg. BWL .loop

Operation

$LR \leftarrow PC + 1; PC \leftarrow PC + \#imm8$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Save the current program counter (PC) value plus one to the link register. Then unconditionally branch to the resultant address from addition of PC and the 8-bit immediate value specified in the instruction. LABEL can be both a symbolic name or a numeric value, and is capable of jumping forwards or backwards.

Addressing Mode: PC Relative.

4.34 RET

Return

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	imm8							

Syntax

RET

eg. RET

Operation

$PC \leftarrow LR$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Unconditionally branch to the address stored in the link register (LR).

Addressing Mode: Register-Indirect.

4.35 JMP

Jump

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	1	imm8							

Syntax

JMP Ra, #imm5

eg. JMP R3, #7

Operation

$PC \leftarrow Ra + \#imm5$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Unconditionally jump to the resultant address from the addition of GPR[Ra] and the 5-bit immediate value specified in the instruction.

Addressing Mode: Base Plus Offset.

4.36 PUSH

Push From Stack

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	L	X	X	Ra			0	0	0	0	1

Syntax

PUSH Ra

eg. PUSH R3

PUSH RL

eg. PUSH RL

Operation

$\text{Mem}[\text{R7}] \leftarrow \text{reg}; \text{R7} \leftarrow \text{R7} - 1$

$\text{N} \leftarrow \text{N}$

$\text{Z} \leftarrow \text{Z}$

$\text{V} \leftarrow \text{V}$

$\text{C} \leftarrow \text{C}$

Description

‘reg’ corresponds to either a GPR or the link register, the contents of which are stored to the stack using the address stored in the stack pointer (R7). Then Decrement the stack pointer by one.

Addressing Modes: Register-Indirect, Postdecrement.

4.37 POP

Pop From Stack

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	L	X	X		Ra		0	0	0	0	1

Syntax

POP Ra

POP RL

eg. POP R3

eg. POP RL

Operation

$R7 \leftarrow R7 + 1$; $\text{Mem}[R7] \leftarrow \text{reg}$;

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Increment the stack pointer by one. Then ‘reg’ corresponds to either a GPR or the link register, the contents of which are retrieved from the stack using the address stored in the stack pointer (R7).

Addressing Modes: Register-Indirect, Preincrement.

4.38 RETI

Return From Interrupt

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	0	1	1	1	X	X	X	X	X

Syntax

RETI

eg. RETI

Operation

$PC \leftarrow \text{Mem}[R7]$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Restore program counter to its value before interrupt occurred, which is stored on the stack, pointed to by the stack pointer (R7). This must be the last instruction in an interrupt service routine.

Addressing Mode: Register-Indirect.

4.39 ENAI

Enable Interrupts

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	1	1	1	1	X	X	X	X	X

Syntax

ENAI

eg. ENAI

Operation

Set Interrupt Enable Flag

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Turn on interrupts by setting interrupt enable flag to true (1).

4.40 DISI

Disable Interrupts

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	0	1	1	1	X	X	X	X	X

Syntax

DISI

eg. DISI

Operation

Reset Interrupt Enable Flag

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Turn off interrupts by setting interrupt enable flag to false (0).

4.41 STF

Store Status Flags

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	1	1	1	1	X	X	X	X	X

Syntax

STF

eg. STF

Operation

$\text{Mem} [\text{R7}] \leftarrow \{12\text{-bit } 0, Z, C, V, N\}; \text{R7} \leftarrow \text{R7} - 1;$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

Description

Store contents of status flags to stack using address held in stack pointer (R7). Then decrement the stack pointer (R7) by one.

Addressing Modes: Register-Indirect, Postdecrement.

4.42 LDF

Load Status Flags

Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	0	0	1	1	1	X	X	X	X	X

Syntax

LDF

eg. LDF

Operation

$R7 \leftarrow R7 + 1$

$N \leftarrow \text{Mem}[R7][0]$

$Z \leftarrow \text{Mem}[R7][3]$

$V \leftarrow \text{Mem}[R7][1]$

$C \leftarrow \text{Mem}[R7][2]$

Description

Increment the stack pointer (R7) by one. Then load content of status flags with lower 4 bits of value retrieved from stack using address held in stack pointer (R7).

Addressing Modes: Register-Indirect, Preincrement.

5 Programming Tips

Lorem Ipsum...

6 Assembler

The current instruction set architecture includes an assembler for converting assembly language into hex. This chapter outlines the required formatting and available features of this assembler.

6.1 Instruction Formatting

Each instruction must be formatted using the following syntax, here “[...]” indicates an optional field:

```
[.LABELNAME] MNEMONIC, OPERANDS, ..., :[COMMENTS]
```

eg. `.loop ADDI, R5, R3, #5 :Add 5 to R3`

Comments may be added by preceding them with either `:` or `;`

Accepted general purpose register values are: R0, R1, R2, R3, R4, R5, R6, R7, SP. These can be upper or lower case and SP is equivalently evaluated to R7.

Branch instructions can take either a symbolic or numeric value. Where a numeric must be relative and between -32 and 31 for a JMP instruction, or between -128 and 127 for any other branch type. If the branch exceeds the accepted range, the assembler will flag an error message.

All label names must begin with a ‘.’ while `.ISR/.isr` and `.define` are special cases used for the interrupt service routine and variable definitions respectively.

Instruction-less or comments only lines are allowed within the assembly file.

Special Case Label

The `.ISR/.isr` label is reserved for the interrupt service routine and may be located anywhere within the file but must finish with a `'RETI'` instruction and be no longer than 126 lines of code. Branches may occur within the ISR, but are not allowed into this subroutine with the exception of a return from a separate subroutine.

6.2 Assembler Directives

Symbolic label names are supported for branch-type instructions. Following the previous syntax definition for `'LABELNAME'`, they can be used instead of numeric branching provided they branch no further than the maximum distance allowed for the instruction used. Definitions are supported by the assembler. They are used to assign meaningful names to the GPRs to aid with programming. Definitions can occur at any point within the file and create a mapping from that point onwards. Different names can be assigned to the same register, but only one is valid at a time.

The accepted syntax for definitions is:

```
.define NAME REGISTER
```

6.3 Running The Assembler

The assembler is a python executable and is run by typing `“./assemble.py”`. Alternatively, the assembler can be placed in a folder on the users path and executed by running `“assemble.py”`. It supports Python versions 2.4.3 to 2.7.3. A help prompt is given by the script if the usage is not correct, or given a `-h` or `--help` argument.

By default, the script will output the assembled hex to a file with the same name, but with a `‘.hex’` extension in the same directory. The user can specify a different file to use by using a `-o filename.hex` or `--output=filename.hex` argument to the script. The output file can also be a relative or absolute path to a different directory.

The full usage for the script is seen in listing 1. This includes the basic rules for writing the assembly language and a version log.

Listing 1: Assembler help prompt

```

1 $> assemble.py
2 Usage: assemble.py [-o outfile] input
3
4 —Team R4 Assembler Help—
5 ———Version:
6   1 (CMPI addition onwards)
7   2 (Changed to final ISA, added special case I's and error
8     checking)
9   3 (Ajr changes – Hex output added, bug fix)
10  4 (Added SP symbol)
11  5 (NOP support added, help added) UNTESTED
12  6 (Interrupt support added [ENAI, DISI, RETI])
13  7 (Checks for duplicate Labels)
14  8 (Support for any ISR location & automated startup code entry)
15  9 (Support for .define)
16 10 (Changed usage)
17   Current is most recent iteration
18   Commenting uses : or ;
19   Labels start with '.': SPECIAL .ISR/.isr -> Interrupt Service
20     Routine)
21     SPECIAL .define -> define new name for
22     General Purpose Register, .define NAME R0-R7/SP
23   Instruction Syntax: .[LABELNAME] MNEUMONIC, OPERANDS, ..., :[
24     COMMENTS]
25   Registers: R0, R1, R2, R3, R4, R5, R6, R7==SP
26   Branching: Symbolic and Numeric supported
27
28   Notes:
29   Input files are assumed to end with a .asm extension
30   Immediate value sizes are checked
31   Instruction-less lines allowed
32   .ISR may be located anywhere in file
33   .define may be located anywhere, definition valid from location
34     in file onwards, may replace existing definitions
35
36   Options:
37   -h, --help          show this help message and exit
38   -o FILE, --output=FILE
39                       output file for the assembled output

```

6.4 Error Messages

Code	Description
ERROR1	Instruction mnemonic is not recognized
ERROR2	Register code within instruction is not recognized
ERROR3	Branch condition code is not recognised
ERROR4	Attempting to branch to undefined location
ERROR5	Instruction mnemonic is not recognized
ERROR6	Attempting to shift by more than 16 or perform a negative shift
ERROR7	Magnitude of immediate value for ADDI, ADCI, SUBI, SUCI, LDW or STW is too large
ERROR8	Magnitude of immediate value for CMPI or JMP is too large
ERROR9	Magnitude of immediate value for ADDIB, SUBIB, LUI or LLI is too large
ERROR10	Attempting to jump more than 127 forward or 128 backwards
ERROR11	Duplicate symbolic link names
ERROR12	Illegal branch to ISR
ERROR13	Multiple ISRs in file
ERROR14	Invalid formatting for .define directive

7 Programs

Every example program in this section uses R7 as a stack pointer which is initialised to the by the program to 0x07D0 using the LUI and LLI instructions.

7.1 Multiply

The code for the multiply program is held in Appendix A.1 listing 7. A sixteen bit number is read from input switches and then split in to lower and upper bytes which are then multiplied. The resulting sixteen bit word is written to the LEDs before reaching a terminating loop.

The subroutine operation is described using C in listing 2. If the result is greater than or equal to 2^{16} the subroutine will fail and return zero; The lowest bit of the multiplier control the accumulator and the overflow check. The multiplier is shifted right and the quotient is shifted left at every iteration. Equation (1) formally describes the result of algorithm disregarding physical limitations.

Listing 2: Multiply Subroutine

```

1  uint16_t multi(uint16_t mul, quo){
2      uint32_t A;
3      uint16_t M,Q,i;
4      A = 0; M = mul; Q = quo;
5      for (i=0;i<16;i++){
6          if (M && 0x0001){           // LSb
7              A = A + Q;
8              if (A > 0xFFFF){       // Larger than 16 bits?
9                  return 0;         // Fail
10             }
11         }
12         Q = Q << 1;
13         M = M >> 1;
14     }
15     return A;                       // Bottom 16 bits
16 }
```

$$A = M \times Q = \sum_{i=0}^{\infty} 2^i M_i Q \text{ where } M_i \in \{0,1\} \quad (1)$$

The multiply code has also been optimised by unrolling the loop contents and is held in Appendix A.2 listing 8. In implementation a trade off between code size and execution time is made by loop unrolling the eight stages. This creates scope for optimisation in operations contained in the loop, doesn't use a counter and requires less branch operations.

7.2 Factorial

The code for the factorial program is held in Appendix A.3 listing 9. It is possible to calculate the factorial of any integer value between 0 and 8 inclusive. The main body of code masks the value read from the input switches so only acceptable values are passed to subroutine. The factorial subroutine is called which in turn calls the multiply subroutine discussed in section 7.1. The result is calculated recursively as described using C in listing 3.

Listing 3: Recursive Factorial Subroutine

```
1 uint16_t fact(uint16_t x){
2     if(x == 0){
3         return 1;                // 0! = 1
4     }else{
5         return multi(x, fact(x-1)); // Recursive
6     }
7 }
```

7.3 Random

The code for the random program is held in Appendix A.4 listing 10. A random series of numbers is achieved by simulating a 16 bit linear feedback shift register. This produces a new number every 16 sixteen clock cycles so in this case a simulation subroutine is called 16 times. A seed taken from switches is passed to the first subroutine call then using BWL instructions the parameter is altered and passed to the next subroutine call. No more PUSH or POP operations are performed. A load from the stack pointer is used write a new random number to LEDs. All contained within an unconditional branch.

An 2 input XOR gate is simulated by using masking the register value the comparing against inputs 00 and 11. These would return zero so only a shift is performed. If this is not true then a shift is performed followed by an OR operation with 0x8000 therefore feeding back a value to the top of the shift register. This is described using C in listing 4.

surely
this
we
can
just
use
the
XOR
in-
struc-
tion

Listing 4: Linear Feedback Shift Register Subroutine

```
1 uint16_t rand(uint16_t last){
2     uint16_t next, test;
3     next = last >> 1;                // Shift reg
```

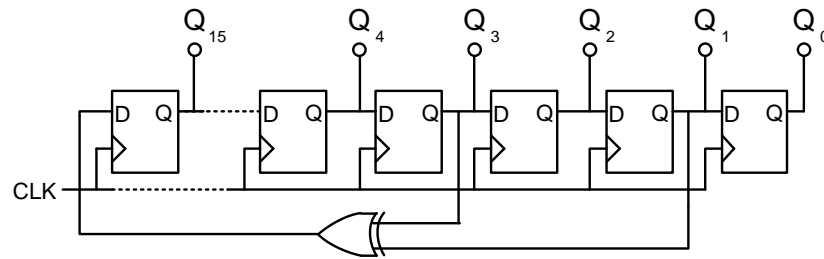


Figure 1: 16 Bit Linear Feedback Shift Register.

Maybe change to IEEE symbols if we have time

```

4     test = last & 0x000A;                // Bits 4 and 1
5     if((test == 0x0000)|(test == 0x000A)){ // XOR test
6         return next;
7     }
8     return next | 0x8000;                // Feedback to top
9 }

```

7.4 Interrupt

The code for the interrupt program is held in Appendix A.5 listing 11. This is the most complex example and makes use of both the multiply and factorial subroutines in sections 7.1 and 7.2 respectively.

Listing 5: Serial Device Interrupt Service Request

```

1  isr () {
2
3  }
4
5  void main () {
6
7
8  }

```


8 Simulation

8.1 Running the simulations

A register window could also be done for this section too

Sim.py needs a fair bit of change. If we have time, this could be altered to use Iains dir structure. This section highlights how to use his script and our assembler.

How to run for each of the behavioural, extracted and mixed

Before the simulator is invoked, the assembler should be invoked. This is discussed in section 6.3. It can be done from within the programs directory (`/design/fcde/verilog/programs`) by running, for example `assemble.py multiply`

The script “simulate” is an executable shell script. It is run from the terminal in the directory `/design/fcde/verilog`. This supports running simulations of a full verilog model, cross simulation and a fully extracted simulation. Usage is as follows:

```
./simulate type program [definitions]
```

The ‘type’ can be one of the following: *behavioural*, *mixed*, *extracted*. ‘Program’ is a relative path to the assembled hex file, usually located in the programs folder. Extra definitions can also be included to set the switch value or serial data input.

The serial data file used is located in the programs directory. This is a hex file with white space separated values of the form “ time data”. The data is then sent at the time to the processor by the serial module. An example serial data hex file is shown in listing 6.

Listing 6: Example serial data file

```
1 // Hex file to specify serial data input
2 //
3 //
4 248    7
5 48F    6
6 6D6    5
7 91D    4
8 B64    7
9 DAB    5
```

10	36B1	3
11	6D61	2

Below is a complete list of commands to run all programs on all versions of the processor. 'Number' is a user defined decimal value to set the switches.

- ./assembler/assemble.py programs/multiply.asm
./simulate behavioural programs/multiply.hex +define+switch_value=number
- ./assembler/assemble.py programs/multiply.asm
./simulate mixed programs/multiply.hex +define+switch_value=number
- ./assembler/assemble.py programs/multiply.asm
./simulate extracted programs/multiply.hex +define+switch_value=number
- ./assembler/assemble.py programs/random.asm
./simulate behavioural programs/random.hex +define+switch_value=number
- ./assembler/assemble.py programs/random.asm
./simulate mixed programs/random.hex +define+switch_value=number
- ./assembler/assemble.py programs/random.asm
./simulate extracted programs/random.hex +define+switch_value=number
- ./assembler/assemble.py programs/factorial.asm
./simulate behavioural programs/factorial.hex +define+switch_value=number
- ./assembler/assemble.py programs/factorial.asm
./simulate mixed programs/factorial.hex +define+switch_value=number
- ./assembler/assemble.py programs/factorial.asm
./simulate extracted programs/factorial.hex +define+switch_value=number
- ./assembler/assemble.py programs/interrupt.asm
./simulate behavioural programs/interrupt.hex programs/serial_data.hex
- ./assembler/assemble.py programs/interrupt.asm
./simulate mixed programs/interrupt.hex programs/serial_data.hex

Table 1: Clock cycles required for each program to run

Make these more accurate when AJR has finished playing around

Program	Clock Cycles
Multiply	900
Factorial	6000
Random	
Interrupt	30000

- ./assembler/assemble.py programs/interrupt.asm
./simulate extracted programs/interrupt.hex programs/serial_data.hex

how to run scan path sim too

need to mention extracting? i wouldn't say so as it should already be done and that can't really change.

The number of clock cycles for each program to fully run is shown in table 1. Factorial run time is given for an input of 8 and is the worst case. Interrupt is dependant on the serial data input and the time is given for the serial data file mentioned above.

and TCL?

A dissembler is also implemented in system verilog to aid debugging. It is an ASCII formatted array implemented at the top level of the simulation. It is capable of reading the instruction register with in the design, and reconstructing the assembly language of the instruction. It will show the opcode, register addresses and immediate values. It is automatically included by the TCL script. The TCL script also opens a waveform window and adds important signals.

check that the dissembler works in all the simulations (it doesn't...)

A Code Listings

All code listed in this section is passed to the assembler *as is* and has been verified using the final design of the processor.

A.1 Multiply

Listing 7: multiply.asm

```
1      LUI SP, #7      ; Init SP
2      LLI SP, #208
3      LUI R0, #8      ; SWs ADDR
4      LLI R0, #0
5      LDW R0, [R0, #0] ; READ SWs
6      LUI R1, #0
7      LLI R1, #255    ; 0x00FF in R1
8      AND R1, R0, R1  ; Lower byte SWs in R1
9      LSR R0, R0, #8  ; Upper byte SWs in R0
10     SUB R2, R2, R2   ; Zero required
11     PUSH R0          ; Op1
12     PUSH R1          ; Op2
13     PUSH R2          ; Place holder is zero
14     BWL .multi      ; Run Subroutine
15     POP R1           ; Result
16     ADDIB SP, #2     ; Dummy pop
17     LUI R4, #8
18     LLI R4, #1       ; Address of LEDS
19     STW R1, [R4, #0] ; Result on LEDS
20 .end BR .end        ; Finish loop
21 .multi PUSH R0
22     PUSH R1
23     PUSH R2
24     PUSH R3
25     PUSH R4
26     PUSH R5
27     PUSH R6
28     LDW R0, [SP, #8] ; R0 - Multiplier
29     LDW R1, [SP, #9] ; R1 - Quotient
30     SUB R2, R2, R2   ; R2 - Accumulator
31     ADDI R3, R2, #1  ; R3 - Compare 1/0
32     SUB R4, R4, R4   ; R4 - Loop counter
33 .lpMul AND R6, R0, R3 ; R6 - Cmp var
34     CMPI R6, #0
35     BE .sh
```

```

36      SUB R3,R3,R3
37      ADD R2,R2,R1      ; A = A + Q
38      ADCI R3,R3,#1
39      CMPI R3,#2
40      BE .over          ; OV
41 .sh      LUI R5,#128
42          LLI R5,#0      ; 0x8000
43          AND R5,R5,R1
44          CMPI R5,#0
45          BE .shift
46          CMPI R0,#0
47          BNE .over      ; And M != 0
48 .shift   LSL R1,R1,#1   ; Q = Q << 1
49          LSR R0,R0,#1   ; M = M >> 1
50          ADDIB R4,#1    ; i++
51          CMPI R4,#15
52          BNE .lpMul
53 .done    STW R2,[SP,#7] ; Res on stack frame
54          POP R6
55          POP R5
56          POP R4
57          POP R3
58          POP R2
59          POP R1
60          POP R0
61          RET
62 .over    SUB R2,R2,R2    ; OV – RET 0
63          BR .done

```

A.2 Multiply Unrolled

Listing 8: multiplyUnroll.asm

```

1      LUI SP, #7          ; Init SP
2      LLI SP, #208
3      LUI R0, #8          ; SWs ADDR
4      LLI R0, #0
5      LDW R0,[R0,#0]      ; READ SWs
6      LUI R1, #0
7      LLI R1, #255        ; 0x00FF in R1
8      AND R1,R0,R1        ; Lower byte SWs in R1
9      LSR R0,R0,#8        ; Upper byte SWs in R0
10     SUB R2,R2,R2        ; Zero required
11     PUSH R0              ; Op1

```

```

12      PUSH R1          ; Op2
13      PUSH R2          ; Place holder is zero
14      BWL .multi       ; Run Subroutine
15      POP R1           ; Result
16      ADDIB SP,#2      ; Dummy pop
17      LUI R4, #8
18      LLI R4, #1       ; Address of LEDS
19      STW R1,[R4,#0]   ; Result on LEDS
20      .end            BR .end      ; Finish loop
21      .multi          PUSH R0
22                      PUSH R1
23                      PUSH R2
24                      PUSH R3
25                      PUSH R4
26                      PUSH R5
27                      PUSH R6
28      LDW R2,[SP,#8]   ; R2 - Multiplier
29      LDW R3,[SP,#9]   ; R3 - Quotient
30      SUB R4,R4,R4     ; R4 - Accumulator
31      ADDI R6,R4,#1    ; R6 - Compare 1/0
32      LUI R5,#128
33      LLI R5,#0        ; R5 - 0x8000
34      AND R1,R2,R6     ; Stage 1, R1 - cmp
35      CMPI R1,#0       ; LSb ?
36      BE .sh1
37      ADD R4,R4,R3     ; (LSb == 1)?
38      .sh1            AND R0,R5,R3
39                      CMPI R0,#0
40                      BNE .over1
41                      LSL R3,R3,#1
42                      LSR R2,R2,#1
43                      AND R1,R2,R6 ; Stage 2
44                      CMPI R1,#0
45                      BE .sh2
46      ADD R4,R4,R3
47      .sh2            AND R0,R5,R3
48                      CMPI R0,#0
49                      BNE .over1
50                      LSL R3,R3,#1
51                      LSR R2,R2,#1
52                      AND R1,R2,R6 ; Stage 3
53                      CMPI R1,#0
54                      BE .sh3
55      ADD R4,R4,R3
56      .sh3            AND R0,R5,R3

```

```

57      CMPI R0,#0
58      BNE .over1
59      LSL R3,R3,#1
60      LSR R2,R2,#1
61      AND R1,R2,R6      ; Stage 4
62      CMPI R1,#0
63      BE .sh4
64      ADD R4,R4,R3
65 .sh4  AND R0,R5,R3
66      CMPI R0,#0
67      BNE .over1
68      LSL R3,R3,#1
69      LSR R2,R2,#1
70      AND R1,R2,R6      ; Stage 5
71      CMPI R1,#0
72      BE .sh5
73      ADD R4,R4,R3
74 .sh5  AND R0,R5,R3
75      CMPI R0,#0
76      BNE .over1
77      LSL R3,R3,#1
78      LSR R2,R2,#1
79      AND R1,R2,R6      ; Stage 6
80      CMPI R1,#0
81      BE .sh6
82      ADD R4,R4,R3
83 .sh6  AND R0,R5,R3
84      CMPI R0,#0
85      BNE .over1
86      LSL R3,R3,#1
87      LSR R2,R2,#1
88      AND R1,R2,R6      ; Stage 7
89      CMPI R1,#0
90      BE .sh7
91      ADD R4,R4,R3
92 .sh7  AND R0,R5,R3
93      CMPI R0,#0
94      BNE .over1
95      LSL R3,R3,#1
96      LSR R2,R2,#1
97      AND R1,R2,R6      ; Stage 8
98      CMPI R1,#0
99      BE .sh8
100     ADD R4,R4,R3
101     BR .sh8

```

```

102 .over1  BR .over
103 .sh8    AND R0,R5,R3
104        CMPI R0,#0
105        BNE .over
106        LSL R3,R3,#1
107        LSR R2,R2,#1
108        AND R1,R2,R6      ; Stage 9
109        CMPI R1,#0
110        BE .sh9
111        SUB R6,R6,R6
112        ADD R4,R4,R3
113        ADCI R6,R4,#1
114        CMPI R6,#2
115        BNE .over
116 .sh9    AND R0,R5,R3
117        CMPI R0,#0
118        BNE .over
119        LSL R3,R3,#1
120        LSR R2,R2,#1
121        AND R1,R2,R6      ; Stage 10
122        CMPI R1,#0
123        BE .sh10
124        SUB R6,R6,R6
125        ADD R4,R4,R3
126        ADCI R6,R4,#1
127        CMPI R6,#2
128        BNE .over
129 .sh10   AND R0,R5,R3
130        CMPI R0,#0
131        BNE .over
132        LSL R3,R3,#1
133        LSR R2,R2,#1
134        AND R1,R2,R6      ; Stage 11
135        CMPI R1,#0
136        BE .sh11
137        SUB R6,R6,R6
138        ADD R4,R4,R3
139        ADCI R6,R4,#1
140        BNE .over
141 .sh11   AND R0,R5,R3
142        CMPI R0,#0
143        BNE .over
144        LSL R3,R3,#1
145        LSR R2,R2,#1
146        AND R1,R2,R6      ; Stage 12

```



```

147      CMPI R1,#0
148      BE .sh12
149      SUB R6,R6,R6
150      ADD R4,R4,R3
151      ADCI R6,R4,#1
152      BNE .over
153 .sh12  AND R0,R5,R3
154      CMPI R0,#0
155      BNE .over
156      LSL R3,R3,#1
157      LSR R2,R2,#1
158      AND R1,R2,R6      ; Stage 13
159      CMPI R1,#0
160      BE .sh13
161      SUB R6,R6,R6
162      ADD R4,R4,R3
163      ADCI R6,R4,#1
164      BNE .over
165 .sh13  AND R0,R5,R3
166      CMPI R0,#0
167      BNE .over
168      LSL R3,R3,#1
169      LSR R2,R2,#1
170      AND R1,R2,R6      ; Stage 14
171      CMPI R1,#0
172      BE .sh14
173      SUB R6,R6,R6
174      ADD R4,R4,R3
175      ADCI R6,R4,#1
176      BNE .over
177 .sh14  AND R0,R5,R3
178      CMPI R0,#0
179      BNE .over
180      LSL R3,R3,#1
181      LSR R2,R2,#1
182      AND R1,R2,R6      ; Stage 15
183      CMPI R1,#0
184      BE .sh15
185      SUB R6,R6,R6
186      ADD R4,R4,R3
187      ADCI R6,R4,#1
188      BNE .over
189 .sh15  AND R0,R5,R3
190      CMPI R0,#0
191      BNE .over

```

```

192     LSL R3,R3,#1
193     LSR R2,R2,#1
194     AND R1,R2,R6      ; Stage 16
195     CMPI R1,#0
196     BE .sh16
197     SUB R6,R6,R6
198     ADD R4,R4,R3
199     ADCI R6,R4,#1
200     BNE .over
201 .sh16 STW R4,[SP,#7]   ; Res on stack frame
202     POP R6
203     POP R5
204     POP R4
205     POP R3
206     POP R2
207     POP R1
208     POP R0
209     RET
210 .over SUB R4,R4,R4
211     BR .sh16

```

A.3 Factorial

Listing 9: factorial.asm

```

1     LUI R7, #7
2     LLI R7, #208
3     LUI R0, #8      ; Address in R0
4     LLI R0, #0
5     LDW R0,[R0,#0]   ; Read switches into R0
6     LUI R1,#0        ; Calculate only 8 or less
7     LLI R1,#8
8     CMP R1,R0
9     BE .do
10    SUBIB R1,#1
11    AND R0,R0,R1
12 .do  PUSH R0          ; Pass para
13     BWL .fact        ; Run Subroutine
14     POP R0           ; Para overwritten with result
15     LUI R4, #8
16     LLI R4, #1      ; Address of LEDS
17     STW R0,[R4,#0]  ; Result on LEDS
18 .end  BR .end        ; finish loop
19 .fact PUSH R0

```

```

20     PUSH R1
21     PUSH LR
22     LDW R1,[SP,#3] ; Get para
23     ADDIB R1,#0
24     BE .retOne ; 0! = 1
25     SUBI R0,R1,#1
26     PUSH R0 ; Pass para
27     BWL .fact ; The output remains on the stack
28     PUSH R1 ; Pass para
29     SUBIB SP,#1 ; Placeholder
30     BWL .multi
31     POP R1 ; Get res
32     ADDIB SP,#2 ; POP x 2
33     STW R1,[SP,#3]
34     POP LR
35     POP R1
36     POP R0
37     RET
38 .retOne ADDIB R1,#1 ; Avoid jump checking
39     STW R1,[SP,#3]
40     POP LR
41     POP R1
42     POP R0
43     RET
44 .multi PUSH R0
45     PUSH R1
46     PUSH R2
47     PUSH R3
48     PUSH R4
49     PUSH R5
50     PUSH R6
51     LDW R2,[SP,#8] ; R2 - Multiplier
52     LDW R3,[SP,#9] ; R3 - Quotient
53     SUB R4,R4,R4 ; R4 - Accumulator
54     ADDI R6,R4,#1 ; R6 - Constant 1
55     SUB R5,R5,R5 ; R5 - Constant 0
56     SUB R0,R0,R0 ; R0 - C check
57     AND R1,R2,R6 ; Stage 1, R1 - cmp
58     CMPI R1,#0 ; LSb ?
59     BE .sh1
60     ADD R4,R4,R3 ; (LSb == 1)?
61 .sh1 LSL R3,R3,#1
62     LSR R2,R2,#1
63     AND R1,R2,R6 ; Stage 2
64     CMPI R1,#0

```

```

65      BE .sh2
66      ADD R4,R4,R3
67 .sh2  LSL R3,R3,#1
68      LSR R2,R2,#1
69      AND R1,R2,R6      ; Stage 3
70      CMPI R1,#0
71      BE .sh3
72      ADD R4,R4,R3
73 .sh3  LSL R3,R3,#1
74      LSR R2,R2,#1
75      AND R1,R2,R6      ; Stage 4
76      CMPI R1,#0
77      BE .sh4
78      ADD R4,R4,R3
79 .sh4  LSL R3,R3,#1
80      LSR R2,R2,#1
81      AND R1,R2,R6      ; Stage 5
82      CMPI R1,#0
83      BE .sh5
84      ADD R4,R4,R3
85 .sh5  LSL R3,R3,#1
86      LSR R2,R2,#1
87      AND R1,R2,R6      ; Stage 6
88      CMPI R1,#0
89      BE .sh6
90      ADD R4,R4,R3
91 .sh6  LSL R3,R3,#1
92      LSR R2,R2,#1
93      AND R1,R2,R6      ; Stage 7
94      CMPI R1,#0
95      BE .sh7
96      ADD R4,R4,R3
97 .sh7  LSL R3,R3,#1
98      LSR R2,R2,#1
99      AND R1,R2,R6      ; Stage 8
100     CMPI R1,#0
101     BE .sh8
102     ADD R4,R4,R3
103 .sh8  LSL R3,R3,#1
104     LSR R2,R2,#1
105     AND R1,R2,R6      ; Stage 9
106     CMPI R1,#0
107     BE .sh9
108     ADD R4,R4,R3
109     ADCI R0,R5,#0

```

```

110      CMPI R0,#0
111      BNE .over
112 .sh9   LSL R3,R3,#1
113      LSR R2,R2,#1
114      AND R1,R2,R6      ; Stage 10
115      CMPI R1,#0
116      BE .sh10
117      ADD R4,R4,R3
118      ADCI R0,R5,#0
119      CMPI R0,#0
120      BNE .over
121 .sh10  LSL R3,R3,#1
122      LSR R2,R2,#1
123      AND R1,R2,R6      ; Stage 11
124      CMPI R1,#0
125      BE .sh11
126      ADD R4,R4,R3
127      ADCI R0,R5,#0
128      CMPI R0,#0
129      BNE .over
130 .sh11  LSL R3,R3,#1
131      LSR R2,R2,#1
132      AND R1,R2,R6      ; Stage 12
133      CMPI R1,#0
134      BE .sh12
135      ADD R4,R4,R3
136      ADCI R0,R5,#0
137      CMPI R0,#0
138      BNE .over
139 .sh12  LSL R3,R3,#1
140      LSR R2,R2,#1
141      AND R1,R2,R6      ; Stage 13
142      CMPI R1,#0
143      BE .sh13
144      ADD R4,R4,R3
145      ADCI R0,R5,#0
146      CMPI R0,#0
147      BNE .over
148 .sh13  LSL R3,R3,#1
149      LSR R2,R2,#1
150      AND R1,R2,R6      ; Stage 14
151      CMPI R1,#0
152      BE .sh14
153      ADD R4,R4,R3
154      ADCI R0,R5,#0

```

```

155      CMPI R0,#0
156      BNE .over
157 .sh14  LSL R3,R3,#1
158      LSR R2,R2,#1
159      AND R1,R2,R6      ; Stage 15
160      CMPI R1,#0
161      BE .sh15
162      ADD R4,R4,R3
163      ADCI R0,R5,#0
164      CMPI R0,#0
165      BNE .over
166 .sh15  LSL R3,R3,#1
167      LSR R2,R2,#1
168      AND R1,R2,R6      ; Stage 16
169      CMPI R1,#0
170      BE .sh16
171      ADD R4,R4,R3
172      ADCI R0,R5,#0
173      CMPI R0,#0
174      BNE .over
175 .sh16  STW R4,[SP,#7]  ; Res on stack frame
176      POP R6
177      POP R5
178      POP R4
179      POP R3
180      POP R2
181      POP R1
182      POP R0
183      RET
184 .over  SUB R4,R4,R4
185      STW R4,[SP,#7]  ; Res on stack frame
186      POP R6
187      POP R5
188      POP R4
189      POP R3
190      POP R2
191      POP R1
192      POP R0
193      RET

```

A.4 Random

Listing 10: random.asm

```

1      LUI      SP,#7          ; Init SP
2      LLI      SP,#208
3      LUI      R0,#8          ; SW Address in R0
4      LLI      R0,#0
5      LDW      R1,[R0,#0]     ; Read switches into R1
6      ADDIB    R0,#1          ; Address of LEDS in R0
7      PUSH     R1
8      .reset   SUB      R4,R4,R4 ; Reset Loop counter
9      .loop    BWL      .rand
10     CMPI     R4,#15
11     BE       .write
12     ADDIB    R4,#1          ; INC loop counter
13     BR       .loop
14     .write   LDW      R1,[SP,#0] ; No POP as re-run
15             STW      R1,[R0,#0] ; Result on LEDS
16             BR       .reset
17     .rand    PUSH     R0      ; LFSR Sim
18             PUSH     R1      ; Protect regs
19             PUSH     R2
20             LDW      R0,[SP,#3] ; Last reg value
21             LSL      R1,R0,#2 ; Shift Bit 4 <- 2
22             XOR      R1,R0,R1 ; XOR Gate
23             LSR      R0,R0,#1 ; Shifted reg
24             LUI      R2,#0
25             LLI      R2,#8
26             AND      R1,R2,R1 ; Mask off Bit 4
27             CMPI     R1,#0
28             BNE      .done
29             LUI      R1,#128
30             LLI      R1,#0
31             OR       R0,R0,R1 ; OR with 0x8000
32     .done    STW      R0,[SP,#3]
33             POP      R2
34             POP      R1
35             POP      R0
36             RET

```

A.5 Interrupt

Listing 11: interrupt.asm

```

1      DISI          ; Reset is off anyway
2      LUI R7, #7
3      LLI R7, #208

```

```

4      LUI R0, #2      ; R0 is read ptr    0x0200
5      LLI R0, #0
6      ADDI R1,R0,#2   ; 0x0202
7      STW R1,[R0,#0]  ; Read ptr set to   0x0202
8      STW R1,[R0,#1]  ; Write ptr set to  0x0202
9      LUI R0,#160     ; Address of Serial control reg
10     LLI R0,#1
11     LUI R1,#0
12     LLI R1,#1      ; Data to enable ints
13     STW R1,[R0,#0]  ; Store 0x001 @ 0xA001
14     ENAI
15     BR .main
16 .isr  DISI
17     STF             ; Keep flags
18     PUSH R0         ; Save only this for now
19     LUI R0,#160
20     LLI R0,#0
21     LDW R0,[R0,#0]  ; R1 contains read serial data
22     ENAI           ; Don't miss event
23     PUSH R1
24     PUSH R2
25     PUSH R3
26     PUSH R4
27     LUI R1,#2
28     LLI R1,#0
29     LDW R2,[R1,#0]  ; R2 contains read ptr
30     ADDI R3,R1,#1
31     LDW R4,[R3,#0]  ; R4 contain the write ptr
32     SUBIB R2,#1     ; Get out if W == R - 1
33     CMP R4,R2
34     BE .isrOut
35     ADDIB R2,#1
36     LUI R1,#2
37     LLI R1,#2
38     CMP R2,R1
39     BNE .write
40     ADDIB R1,#3
41     CMP R4,R1
42     BE .isrOut
43 .write STW R0,[R4,#0] ; Write to buffer
44     ADDIB R4,#1
45     LUI R1,#2
46     LLI R1,#6
47     CMP R1,R4
48     BNE .wrapW

```



```

49      SUBIB R4,#4
50 .wrapW STW R4,[R3,#0] ; Inc write ptr
51 .isrOut POP R4
52      POP R3
53      POP R2
54      POP R1
55      POP R0
56      LDF
57      RETI
58 .main  LUI R0, #2      ; Read ptr address in R0
59      LLI R0, #0
60      LDW R2,[R0,#0]   ; Read ptr in R2
61      LDW R3,[R0,#1]   ; Write ptr in R3
62      CMP R2,R3
63      BE .main        ; Jump back if the same
64      LDW R3,[R2,#0]   ; Load data out of buffer
65      ADDIB R2,#1      ; Inc read ptr
66      SUB R0,R0,R0
67      LUI R0,#2
68      LLI R0,#6
69      SUB R0,R0,R2
70      BNE .wrapR
71      SUBIB R2,#4
72 .wrapR LUI R0, #2      ; Read ptr address in R0
73      LLI R0, #0
74      STW R2,[R0,#0]   ; Store new read pointer
75      SUB R4,R4,R4
76      LLI R4,#15
77      AND R3,R4,R3
78      CMPI R3,#8
79      BE .do
80      LLI R4,#7
81      AND R3,R3,R4
82 .do    PUSH R3
83      BWL .fact
84      POP R3
85      LUI R4,#8
86      LLI R4,#1      ; Address of LEDs
87      STW R3,[R4,#0] ; Put factorial on LEDs
88      BR .main        ; look again
89 .fact  PUSH R0
90      PUSH R1
91      PUSH LR
92      LDW R1,[SP,#3]   ; Get para
93      ADDIB R1,#0

```

```

94         BE .retOne          ; 0! = 1
95         SUBI R0,R1,#1
96         PUSH R0             ; Pass para
97         BWL .fact           ; The output remains on the stack
98         PUSH R1             ; Pass para
99         SUBIB SP,#1         ; Placeholder
100        BWL .multi
101        POP R1               ; Get res
102        ADDIB SP,#2         ; POP x 2
103        STW R1,[SP,#3]
104        POP LR
105        POP R1
106        POP R0
107        RET
108    .retOne ADDIB R1,#1      ; Avoid jump checking
109        STW R1,[SP,#3]
110        POP LR
111        POP R1
112        POP R0
113        RET
114    .multi PUSH R0
115        PUSH R1
116        PUSH R2
117        PUSH R3
118        PUSH R4
119        PUSH R5
120        PUSH R6
121        LDW R2,[SP,#8]      ; R2 - Multiplier
122        LDW R3,[SP,#9]      ; R3 - Quotient
123        SUB R4,R4,R4        ; R4 - Accumulator
124        ADDI R6,R4,#1        ; R6 - Constant 1
125        SUB R5,R5,R5        ; R5 - Constant 0
126        SUB R0,R0,R0        ; R0 - C check
127        AND R1,R2,R6        ; Stage 1, R1 - cmp
128        CMPI R1,#0          ; LSb ?
129        BE .sh1
130        ADD R4,R4,R3        ; (LSb == 1)?
131    .sh1  LSL R3,R3,#1
132        LSR R2,R2,#1
133        AND R1,R2,R6        ; Stage 2
134        CMPI R1,#0
135        BE .sh2
136        ADD R4,R4,R3
137    .sh2  LSL R3,R3,#1
138        LSR R2,R2,#1

```

```

139      AND R1,R2,R6      ; Stage 3
140      CMPI R1,#0
141      BE .sh3
142      ADD R4,R4,R3
143 .sh3  LSL R3,R3,#1
144      LSR R2,R2,#1
145      AND R1,R2,R6      ; Stage 4
146      CMPI R1,#0
147      BE .sh4
148      ADD R4,R4,R3
149 .sh4  LSL R3,R3,#1
150      LSR R2,R2,#1
151      AND R1,R2,R6      ; Stage 5
152      CMPI R1,#0
153      BE .sh5
154      ADD R4,R4,R3
155 .sh5  LSL R3,R3,#1
156      LSR R2,R2,#1
157      AND R1,R2,R6      ; Stage 6
158      CMPI R1,#0
159      BE .sh6
160      ADD R4,R4,R3
161 .sh6  LSL R3,R3,#1
162      LSR R2,R2,#1
163      AND R1,R2,R6      ; Stage 7
164      CMPI R1,#0
165      BE .sh7
166      ADD R4,R4,R3
167 .sh7  LSL R3,R3,#1
168      LSR R2,R2,#1
169      AND R1,R2,R6      ; Stage 8
170      CMPI R1,#0
171      BE .sh8
172      ADD R4,R4,R3
173 .sh8  LSL R3,R3,#1
174      LSR R2,R2,#1
175      AND R1,R2,R6      ; Stage 9
176      CMPI R1,#0
177      BE .sh9
178      ADD R4,R4,R3
179      ADCI R0,R5,#0
180      CMPI R0,#0
181      BNE .over
182 .sh9  LSL R3,R3,#1
183      LSR R2,R2,#1

```

```

184      AND R1,R2,R6      ; Stage 10
185      CMPI R1,#0
186      BE .sh10
187      ADD R4,R4,R3
188      ADCI R0,R5,#0
189      CMPI R0,#0
190      BNE .over
191 .sh10  LSL R3,R3,#1
192        LSR R2,R2,#1
193        AND R1,R2,R6      ; Stage 11
194        CMPI R1,#0
195        BE .sh11
196        ADD R4,R4,R3
197        ADCI R0,R5,#0
198        CMPI R0,#0
199        BNE .over
200 .sh11  LSL R3,R3,#1
201        LSR R2,R2,#1
202        AND R1,R2,R6      ; Stage 12
203        CMPI R1,#0
204        BE .sh12
205        ADD R4,R4,R3
206        ADCI R0,R5,#0
207        CMPI R0,#0
208        BNE .over
209 .sh12  LSL R3,R3,#1
210        LSR R2,R2,#1
211        AND R1,R2,R6      ; Stage 13
212        CMPI R1,#0
213        BE .sh13
214        ADD R4,R4,R3
215        ADCI R0,R5,#0
216        CMPI R0,#0
217        BNE .over
218 .sh13  LSL R3,R3,#1
219        LSR R2,R2,#1
220        AND R1,R2,R6      ; Stage 14
221        CMPI R1,#0
222        BE .sh14
223        ADD R4,R4,R3
224        ADCI R0,R5,#0
225        CMPI R0,#0
226        BNE .over
227 .sh14  LSL R3,R3,#1
228        LSR R2,R2,#1

```

```

229      AND R1,R2,R6      ; Stage 15
230      CMPI R1,#0
231      BE .sh15
232      ADD R4,R4,R3
233      ADCI R0,R5,#0
234      CMPI R0,#0
235      BNE .over
236 .sh15  LSL R3,R3,#1
237        LSR R2,R2,#1
238        AND R1,R2,R6      ; Stage 16
239        CMPI R1,#0
240        BE .sh16
241        ADD R4,R4,R3
242        ADCI R0,R5,#0
243        CMPI R0,#0
244        BNE .over
245 .sh16  STW R4,[SP,#7]    ; Res on stack frame
246        POP R6
247        POP R5
248        POP R4
249        POP R3
250        POP R2
251        POP R1
252        POP R0
253        RET
254 .over  SUB R4,R4,R4
255        STW R4,[SP,#7]    ; Res on stack frame
256        POP R6
257        POP R5
258        POP R4
259        POP R3
260        POP R2
261        POP R1
262        POP R0
263        RET

```