

ELEC6027 - VLSI Design Project : Programmers Guide

Team R4

24th April, 2014

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 controlHSL - 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 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 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 if Result < 0 then 1, else 0

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

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

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

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

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

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

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

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

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

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

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

Description

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 reads a `'asm'` file and outputs a `'hex'` file in hexadecimal format. It is run by typing `"/assemble filename"` at the command line when in the directory of both the assembler executable and the program assembly file. `"filename"` does not have to include the `.asm` file extension. The outputted file is saved to the same directory as the input file.

HSL: I'm going to add an option parser to make the UI a bit easier. This section is likely to change a fair amount

Typing `-h` or `--help` instead of the file name will bring up the help menu with version information and basic formatting support.

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. It is possible a stack is not required in which case no initialisation is needed and R7 can be used as a general purpose register.

surely
this
should
be in
the
reg-
ister
de-
scrip-
tion
sec-
tion

7.1 Multiply

The code for the multiply program is held in Appendix A.1 listing 5. 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 1. 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. 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.

Listing 1: Shift and Add 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 }
```

$$A = M \times Q = \sum_{i=0}^7 2^i M_i Q \text{ where } M_i \in \{0,1\} \quad (1)$$

7.2 Factorial

The code for the factorial program is held in Appendix A.2 listing 6. 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 2.

Listing 2: Recursive Factorial Subroutine

```

1 uint16_t multi(uint16_t mul, quo);    // Prototpye
2
3 uint16_t fact(uint16_t x){
4     if(x == 0){
5         return 1;                    // 0! = 1
6     }else{
7         return multi(x, fact(x-1));  // Recurrusive
8     }
9 }

```

7.3 Random

The code for the random program is held in Appendix A.3 listing 7. 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 3.

Listing 3: Linear Feedback Shift Register Subroutine

```

1 uint16_t rand(uint16_t last){
2     uint16_t next, test;
3     next = last >> 1;                // Shift reg
4     test = last & 0x000A;             // Bits 4 and 1
5     if((test == 0x0000) | (test == 0x000A)){ // XOR test

```

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

```

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.4 listing 8. 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 4: Serial Device Interrupt Service Request

```

1 uint16_t multi(uint16_t mul, quo);    // Prototpye
2
3 uint16_t fact(uint16_t x);           // Prototpye#
4
5 isr () {
6
7 }
8
9 void main () {
10
11
12 }

```

8 Simulation

8.1 Running the simulations

Describe sim.py

- What it does, why it is needed

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

- NEED TO CHANGE SIM.PY TO RUN USING IAINS STRUCTURE

(/home/user/design/fcde...)

- Clock cycles for each of the programs

- Register window - need to do one. Description of also.

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 5: 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 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 - Constant 1
32 SUB R5, R5, R5   ; R5 - Constant 0
33 SUB R0, R0, R0   ; R0 - C check
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  LSL R3,R3,#1
39     LSR R2,R2,#1
40     AND R1,R2,R6 ; Stage 2
41     CMPI R1,#0
42     BE .sh2
43     ADD R4,R4,R3
44 .sh2  LSL R3,R3,#1
45     LSR R2,R2,#1
46     AND R1,R2,R6 ; Stage 3
47     CMPI R1,#0
48     BE .sh3
49     ADD R4,R4,R3
50 .sh3  LSL R3,R3,#1
51     LSR R2,R2,#1
52     AND R1,R2,R6 ; Stage 4
53     CMPI R1,#0
54     BE .sh4
55     ADD R4,R4,R3
56 .sh4  LSL R3,R3,#1
57     LSR R2,R2,#1
58     AND R1,R2,R6 ; Stage 5
59     CMPI R1,#0
60     BE .sh5
61     ADD R4,R4,R3
62 .sh5  LSL R3,R3,#1
63     LSR R2,R2,#1
64     AND R1,R2,R6 ; Stage 6
65     CMPI R1,#0
66     BE .sh6
67     ADD R4,R4,R3
68 .sh6  LSL R3,R3,#1
69     LSR R2,R2,#1
70     AND R1,R2,R6 ; Stage 7
71     CMPI R1,#0
72     BE .sh7
73     ADD R4,R4,R3
74 .sh7  LSL R3,R3,#1
75     LSR R2,R2,#1
76     AND R1,R2,R6 ; Stage 8
77     CMPI R1,#0
78     BE .sh8
79     ADD R4,R4,R3
80 .sh8  LSL R3,R3,#1

```

```

81     LSR R2,R2,#1
82     AND R1,R2,R6 ; Stage 9
83     CMPI R1,#0
84     BE .sh9
85     ADD R4,R4,R3
86     ADCI R0,R5,#0
87     CMPI R0,#0
88     BNE .over
89 .sh9  LSL R3,R3,#1
90     LSR R2,R2,#1
91     AND R1,R2,R6 ; Stage 10
92     CMPI R1,#0
93     BE .sh10
94     ADD R4,R4,R3
95     ADCI R0,R5,#0
96     CMPI R0,#0
97     BNE .over
98 .sh10 LSL R3,R3,#1
99     LSR R2,R2,#1
100    AND R1,R2,R6 ; Stage 11
101    CMPI R1,#0
102    BE .sh11
103    ADD R4,R4,R3
104    ADCI R0,R5,#0
105    CMPI R0,#0
106    BNE .over
107 .sh11 LSL R3,R3,#1
108    LSR R2,R2,#1
109    AND R1,R2,R6 ; Stage 12
110    CMPI R1,#0
111    BE .sh12
112    ADD R4,R4,R3
113    ADCI R0,R5,#0
114    CMPI R0,#0
115    BNE .over
116 .sh12 LSL R3,R3,#1
117    LSR R2,R2,#1
118    AND R1,R2,R6 ; Stage 13
119    CMPI R1,#0
120    BE .sh13
121    ADD R4,R4,R3
122    ADCI R0,R5,#0
123    CMPI R0,#0
124    BNE .over
125 .sh13 LSL R3,R3,#1

```

```

126     LSR R2,R2,#1
127     AND R1,R2,R6 ; Stage 14
128     CMPI R1,#0
129     BE .sh14
130     ADD R4,R4,R3
131     ADCI R0,R5,#0
132     CMPI R0,#0
133     BNE .over
134 .sh14 LSL R3,R3,#1
135     LSR R2,R2,#1
136     AND R1,R2,R6 ; Stage 15
137     CMPI R1,#0
138     BE .sh15
139     ADD R4,R4,R3
140     ADCI R0,R5,#0
141     CMPI R0,#0
142     BNE .over
143 .sh15 LSL R3,R3,#1
144     LSR R2,R2,#1
145     AND R1,R2,R6 ; Stage 16
146     CMPI R1,#0
147     BE .sh16
148     ADD R4,R4,R3
149     ADCI R0,R5,#0
150     CMPI R0,#0
151     BNE .over
152 .sh16 STW R4,[SP,#7] ; Res on stack frame
153     POP R6
154     POP R5
155     POP R4
156     POP R3
157     POP R2
158     POP R1
159     POP R0
160     RET
161 .over SUB R4,R4,R4
162     STW R4,[SP,#7] ; Res on stack frame
163     POP R6
164     POP R5
165     POP R4
166     POP R3
167     POP R2
168     POP R1
169     POP R0
170     RET

```

A.2 Factorial

Listing 6: 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.3 Random

Listing 7: random.asm

```

1     LUI R7, #7 ; Init SP
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, #8
7     LLI R1, #1 ; CONSTANT – Address of LEDS
8     LUI R2,#0
9     LLI R2,#10 ; CONSTANT – 0x000A
10    LUI R3,#128
11    LLI R3,#0 ; CONSTANT – 0x8000
12    PUSH R0
13 .loop BWL .rand ; 1
14    BWL .rand ; 2
15    BWL .rand ; 3
16    BWL .rand ; 4
17    BWL .rand ; 5
18    BWL .rand ; 6
19    BWL .rand ; 7
20    BWL .rand ; 8
21    BWL .rand ; 9
22    BWL .rand ; 10
23    BWL .rand ; 11

```

```

24    BWL .rand      ; 12
25    BWL .rand      ; 13
26    BWL .rand      ; 14
27    BWL .rand      ; 15
28    BWL .rand      ; 16
29    LDW R0,[SP,#0]  ; No POP as re-run
30    STW R0,[R1,#0]  ; Result on LEDS
31    BR .loop
32    .rand    LDW R4,[SP,#0]  ; Linear feedback shift register sim
33    LSR R5,R4,#1
34    AND R4,R2,R4    ; Mask bits one and three
35    BE .done        ; Both zero
36    CMP R4,R2
37    BE .done        ; Both one
38    OR R5,R5,R3
39    .done STW R5,[SP,#0]
40    RET

```

A.4 Interrupt

Listing 8: 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