

# ELEC6027 - VLSI Design Project : Programmers Guide

Team R4

23<sup>rd</sup> 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, moving data between external memory and general purpose registers, transfer of control within a program and interrupt handling. It is based around a RISC architecture and as such has a highly orthogonal formatting of bit fields within the instruction code.

All instruction implemented by the architecture fall into one of 6 groups:

- Data Manipulation
- Byte Immediate
- Data Transfer
- Control Transfer
- Stack Operations
- Interrupts

Each instruction has only one addressing mode associated with it, determined by which group it falls within. Data manipulation instructions have either a register-register or register-immediate addressing mode for performing arithmetic, logic or shift operations. Byte immediate instructions have a register-immediate addressing mode for arithmetic and load immediate type operations. Data transfer instructions have a base plus offset addressing mode for accessing external memory using an address stored in a GPR. Control transfer instructions have PC relative, register indirect and base plus offset addressing modes for changing the value of the program counter. Stack operations have register indirect preincrement or register indirect postdecrement addressing modes for accessing external memory and adjusting the stack pointer value. While interrupt operations have register indirect with postdecrement or preincrement addressing modes for restoring program counter and accessing the stack.

## 4.1 General Instruction Formatting

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

$Rd \leftarrow Ra + Rb$

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

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

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

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

### 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

$Rd \leftarrow Ra + \#imm5$

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

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

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

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

#### 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

$Rd \leftarrow Rd + \#imm8$

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

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

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

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

### 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

$Rd \leftarrow Ra + Rb + C$

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

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

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

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

### 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

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

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

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

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

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

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

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

### 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

$Rd \leftarrow 0 - Ra$

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

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

$V \leftarrow 0$

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

### 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

$Rd \leftarrow Ra - Rb$

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

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

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

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

### 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

$Rd \leftarrow Ra - \#imm5$

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

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

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

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

### 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

$Rd \leftarrow Rd - \#imm8$

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

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

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

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

### 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

$Rd \leftarrow Ra - Rb - C$

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

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

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

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

### Description

The 16-bit word in GPR[Rb] is subtracted from the 16-bit word in GPR[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 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

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

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

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

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

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

### 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, +Rb, -Result) or  
(-Ra, -Rb, +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, +#imm5, -Result) or  
(-Ra, -#imm5, +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

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

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

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

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

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

### Description

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

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

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

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

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

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

### Description

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

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

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

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

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

### Description

The 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

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

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

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

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

### Description

The 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

$PC \leftarrow PC + \#imm8$

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

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

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

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

### Description

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

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

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

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

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

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

### 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

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

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

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

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

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

### 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

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

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

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

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

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

### Description

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

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

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

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

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

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

### Description

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

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

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

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

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

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

#### Description

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

$PC \leftarrow LR$

$N \leftarrow \text{UNPREDICTABLE}$

$Z \leftarrow \text{UNPREDICTABLE}$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

### 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

$PC \leftarrow Ra + \#imm5$

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

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

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

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

### Description

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

reg  $\rightarrow$  Mem[R7]; R7  $\leftarrow$  R7 - 1

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

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

V  $\leftarrow$  UNPREDICTABLE

C  $\leftarrow$  UNPREDICTABLE

### 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

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

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

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

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

### Description

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

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

$N \leftarrow \text{UNPREDICTABLE}$

$Z \leftarrow \text{UNPREDICTABLE}$

$V \leftarrow \text{UNPREDICTABLE}$

$C \leftarrow \text{UNPREDICTABLE}$

### Description

Restore program counter to its value before interrupt occurred, which is stored on the stack, pointed to be 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

IntEn Flag  $\leftarrow$  1

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

IntEn Flag  $\leftarrow$  0

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

$R7 \leftarrow R7 + 1; \{Z, C, V, N\} \leftarrow \text{Mem}[R7][3:0]$

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

### 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 symbolic sequences into machine code. 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.



## 7.1 Multiply

The code for the multiply program is held in appendix A.1 listing 2. Capable of performing multiplication of two unsigned eight bit numbers to produce a single sixteen bit output. The eight bit numbers are read from the sixteen input switches and then split in to lower and upper bytes which are then multiplied. The resulting sixteen bit word is placed upon the LEDs before reaching a terminating loop. A shift and add algorithm is implemented and a trade of between code size and execution time is made by loop unrolling the sixteen stages.

The subroutine is described as C in listing 1. Immediately the two operands are compared against  $0xFF00$ . This checks the input parameters are only 8 bits long otherwise 0 is returned as 16 bits is the maximum length output. In every loop the state of the lowest bit in the multiplier byte controls the accumulator. The multiplier is shifted right and the quotient is shifted left but no shifts occur in the last stage.

```
1 uint16_t multi(uint16_t mul, quo){
2     uint16_t A,M,Q,i;
3     if ((mul && 0xFF00) || (quo && 0xFF00)){
4         return 0;
5     }
6     A = 0; M = mul; Q = quo;
7     for (i=0;i<7;i++){
8         if (M && 0x0001){
9             A = A + Q;
10        }
11        Q = Q << 1;
12        M = M >> 1;
13    }
14    if (M && 0x0001){
15        A = A + Q;
16    }
17    return A;
18 }
```

Listing 1: Shift and Add Subroutine

$$A = 2^7 M_7 Q + \quad (1)$$

## 7.2 Factorial

The code for the factorial program is held in appendix A.2 listing 3.

## 7.3 Random

The code for the random program is held in appendix A.3 listing 4.

## 7.4 Interrupt

The code for the interrupt program is held in appendix A.4 listing 5.

# 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

```
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 R1
22           PUSH R2
23           PUSH R3
24           PUSH R4
25           PUSH R6
26           LDW R2,[SP,#6] ; R2 - Multiplier
27           LDW R3,[SP,#7] ; R3 - Quotient
28           SUB R4,R4,R4   ; R4 - Accumulator
29           LUI R6,#255    ; If larger than 8 bits
30           LLI R6,#0
31           AND R1,R6,R2
32           CMPI R1,#0
33           BNE .sh8       ; Fail
34           AND R1,R6,R3
35           CMPI R1,#0
36           BNE .sh8       ; Fail
37           ADDI R6,R4,#1   ; R6 - Constant 1
38           AND R1,R2,R6   ; Stage 1, R1 - cmp
39           CMPI R1,#0     ; LSb ?
40           BE .sh1
41           ADD R4,R4,R3    ; (LSb == 1)?
42 .sh1      LSL R3,R3,#1
43           LSR R2,R2,#1
44           AND R1,R2,R6    ; Stage 2
45           CMPI R1,#0
46           BE .sh2
47           ADD R4,R4,R3
48 .sh2      LSL R3,R3,#1
49           LSR R2,R2,#1
50           AND R1,R2,R6    ; Stage 3
51           CMPI R1,#0
52           BE .sh3

```

```

53      ADD R4,R4,R3
54 .sh3  LSL R3,R3,#1
55      LSR R2,R2,#1
56      AND R1,R2,R6      ; Stage 4
57      CMPI R1,#0
58      BE .sh4
59      ADD R4,R4,R3
60 .sh4  LSL R3,R3,#1
61      LSR R2,R2,#1
62      AND R1,R2,R6      ; Stage 5
63      CMPI R1,#0
64      BE .sh5
65      ADD R4,R4,R3
66 .sh5  LSL R3,R3,#1
67      LSR R2,R2,#1
68      AND R1,R2,R6      ; Stage 6
69      CMPI R1,#0
70      BE .sh6
71      ADD R4,R4,R3
72 .sh6  LSL R3,R3,#1
73      LSR R2,R2,#1
74      AND R1,R2,R6      ; Stage 7
75      CMPI R1,#0
76      BE .sh7
77      ADD R4,R4,R3
78 .sh7  LSL R3,R3,#1
79      LSR R2,R2,#1
80      AND R1,R2,R6      ; Stage 8
81      CMPI R1,#0
82      BE .sh8
83      ADD R4,R4,R3
84 .sh8  STW R4,[SP,#5]    ; Res on stack frame
85      POP R6
86      POP R4
87      POP R3
88      POP R2
89      POP R1
90      RET

```

Listing 2: multiply.asm

## A.2 Factorial

```

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 from fact to multi remains
    on the stack
28     PUSH R1          ; Pass para
29     BWL .multi
30     POP R1           ; Get res
31     ADDIB SP,#1      ; POP
32     STW R1,[SP,#3]
33     POP LR
34     POP R1
35     POP R0
36     RET
37 .retOne ADDIB R1,#1  ; Trade off code size to avoid jump
    checking
38     STW R1,[SP,#3]
39     POP LR
40     POP R1
41     POP R0
42     RET
43 .multi PUSH R2        ; R2 is M
44     PUSH R3          ; R3 is Q

```

```

45      PUSH R4          ; R4 Is ACC
46      PUSH R6          ; R6 is 1
47      PUSH R1          ; R1 is temp
48      LDW R2,[SP,#5]
49      LDW R3,[SP,#6]
50      SUB R4,R4,R4
51      LUI R6,#0
52      LLI R6,#1        ; load 1 into R6 for compare
53      AND R1,R2,R6     ; Loop unroll for maximum fastness
54      CMPI R1,#0
55      BE .sh1
56      ADD R4,R4,R3
57 .sh1  LSL R3,R3,#1
58      LSR R2,R2,#1
59      AND R1,R2,R6
60      CMPI R1,#0
61      BE .sh2
62      ADD R4,R4,R3
63 .sh2  LSL R3,R3,#1
64      LSR R2,R2,#1
65      AND R1,R2,R6
66      CMPI R1,#0
67      BE .sh3
68      ADD R4,R4,R3
69 .sh3  LSL R3,R3,#1
70      LSR R2,R2,#1
71      AND R1,R2,R6
72      CMPI R1,#0
73      BE .sh4
74      ADD R4,R4,R3
75 .sh4  LSL R3,R3,#1
76      LSR R2,R2,#1
77      AND R1,R2,R6
78      CMPI R1,#0
79      BE .sh5
80      ADD R4,R4,R3
81 .sh5  LSL R3,R3,#1
82      LSR R2,R2,#1
83      AND R1,R2,R6
84      CMPI R1,#0
85      BE .sh6
86      ADD R4,R4,R3
87 .sh6  LSL R3,R3,#1
88      LSR R2,R2,#1
89      AND R1,R2,R6

```

```

90      CMPI R1,#0
91      BE .sh7
92      ADD R4,R4,R3
93 .sh7  LSL R3,R3,#1
94      LSR R2,R2,#1
95      AND R1,R2,R6
96      CMPI R1,#0
97      BE .sh8
98      ADD R4,R4,R3
99 .sh8  LSL R3,R3,#1
100     LSR R2,R2,#1
101     AND R1,R2,R6
102     CMPI R1,#0
103     BE .sh9
104     ADD R4,R4,R3
105 .sh9  LSL R3,R3,#1
106     LSR R2,R2,#1
107     AND R1,R2,R6
108     CMPI R1,#0
109     BE .sh10
110     ADD R4,R4,R3
111 .sh10 LSL R3,R3,#1
112     LSR R2,R2,#1
113     AND R1,R2,R6
114     CMPI R1,#0
115     BE .sh11
116     ADD R4,R4,R3
117 .sh11 LSL R3,R3,#1
118     LSR R2,R2,#1
119     AND R1,R2,R6
120     CMPI R1,#0
121     BE .sh12
122     ADD R4,R4,R3
123 .sh12 LSL R3,R3,#1
124     LSR R2,R2,#1
125     AND R1,R2,R6
126     CMPI R1,#0
127     BE .sh13
128     ADD R4,R4,R3
129 .sh13 LSL R3,R3,#1
130     LSR R2,R2,#1
131     AND R1,R2,R6
132     CMPI R1,#0
133     BE .sh14
134     ADD R4,R4,R3

```

```

135 .sh14    LSL R3,R3,#1
136         LSR R2,R2,#1
137         AND R1,R2,R6
138         CMPI R1,#0
139         BE .sh15
140         ADD R4,R4,R3
141 .sh15    LSL R3,R3,#1
142         LSR R2,R2,#1
143         AND R1,R2,R6
144         CMPI R1,#0
145         BE .sh16
146         ADD R4,R4,R3
147 .sh16    LSL R3,R3,#1
148         LSR R2,R2,#1
149         STW R4,[SP,#5]
150         POP R1
151         POP R6
152         POP R4
153         POP R3
154         POP R2
155         RET

```

Listing 3: factorial.asm

### A.3 Random

```

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, #8
7      LLI R1, #1      ; Address of LEDS
8      PUSH R0
9 .loop BWL .rand      ; 1
10     BWL .rand        ; 2
11     BWL .rand        ; 3
12     BWL .rand        ; 4
13     BWL .rand        ; 5
14     BWL .rand        ; 6
15     BWL .rand        ; 7
16     BWL .rand        ; 8
17     BWL .rand        ; 9
18     BWL .rand        ; 10

```



```

19      BWL .rand      ; 11
20      BWL .rand      ; 12
21      BWL .rand      ; 13
22      BWL .rand      ; 14
23      BWL .rand      ; 15
24      BWL .rand      ; 16
25      LDW R0,[SP,#0]  ; No POP as re-run
26      STW R0,[R1,#0] ; Result on LEDS
27      BR .loop
28 .rand  LDW R2,[SP,#0] ; Linear feedback shift register sim
29      LUI R3,#0      ; Three
30      LLI R3,#3
31      AND R4,R3,R2    ; Bottom two bits of input
32      LSR R5,R2,#1
33      CMP R4,R3       ; Three
34      BE .done
35      SUB R3,R3,R3
36      CMP R4,R3       ; Zero
37      BE .done
38      LUI R3,#128
39      LLI R3,#0
40      OR R5,R5,R3
41 .done  STW R5,[SP,#0]
42      RET

```

Listing 4: random.asm

## A.4 Interrupt

```

1      DISI           ; Reset is off anyway
2      LUI R7, #7
3      LLI R7, #208
4      LUI R0, #1      ; R0 is read ptr    0x0100
5      ADDI R1,R0,#2    ; 0x0102
6      STW R1,[R0,#0]   ; Read ptr set to 0x0102
7      STW R1,[R0,#1]   ; Write ptr set to 0x0102
8      LUI R0,#160      ; Address of Serial control reg
9      LLI R1,#01       ; Data to enable ints
10     STW R1,[R0,#1]    ; Store 0x001 @ 0xA001
11     LLI R3,#18        ; main line -1 in .main
12     ENAI
13     BR .main
14 .isr  DISI
15     STF           ; Keep flags

```

```

16      PUSH R0          ; Save only this for now
17      LUI R0,#160
18      LLI R0,#0
19      LDW R0,[R0,#0]   ; R1 contains read serial data
20      ENAI
21      PUSH R1
22      PUSH R2
23      PUSH R3
24      PUSH R4
25      LUI R1,#1
26      LLI R1,#0
27      LDW R2,[R1,#0]   ; R2 contains read ptr
28      ADDI R3,R1,#1
29      LDW R4,[R3,#0]   ; R4 contain the write ptr
30      SUBIB R2,#1      ; Get out if W == R - 1
31      CMP R4,R2
32      BE .isrOut
33      ADDIB R2,#1
34      LUI R1,#1
35      LLI R1,#2
36      CMP R2,R1
37      BNE .write
38      ADDIB R1,#3
39      CMP R4,R1
40      BE .isrOut
41 .write STW R0,[R4,#0] ; Write to buffer
42      ADDIB R4,#1
43      LUI R1,#1
44      LLI R1,#6
45      CMP R1,R4
46      BNE .wrapW
47      SUBIB R4,#4
48 .wrapW STW R4,[R3,#0] ; Inc write ptr
49 .isrOut POP R4
50      POP R3
51      POP R2
52      POP R1
53      POP R0
54      LDF
55      RETI
56 .main  LUI R0, #1      ; Read ptr address in R0
57      LLI R0, #0
58      LDW R2,[R0,#0]   ; Read ptr in R2
59      LDW R3,[R0,#1]   ; Write ptr in R3
60      CMP R2,R3

```

```

61      BE .main          ; Jump back if the same
62      LDW R3,[R2,#0]    ; Load data out of buffer
63      ADDIB R2,#1       ; Inc read ptr
64      SUB R0,R0,R0
65      LUI R0,#1
66      LLI R0,#6
67      SUB R0,R0,R2
68      BNE .wrapR
69      SUBIB R2,#4
70 .wrapR  LUI R0, #1      ; Read ptr address in R0
71          LLI R0, #0
72          STW R2,[R0,#0] ; Store new read pointer
73          SUB R4,R4,R4
74          LLI R4,#15
75          AND R3,R4,R3
76          CMPI R3,#8
77          BE .do
78          LLI R4,#7
79          AND R3,R3,R4
80 .do     PUSH R3
81          BWL .fact
82          POP R3
83          LUI R4,#8
84          LLI R4,#1      ; Address of LEDs
85          STW R3,[R4,#0] ; Put factorial on LEDs
86          BR .main       ; look again
87 .fact   PUSH R0
88          PUSH R1
89          PUSH LR
90          LDW R1,[SP,#3] ; Get para
91          ADDIB R1,#0
92          BE .retOne     ; 0! = 1
93          SUBI R0,R1,#1
94          PUSH R0        ; Pass para
95          BWL .fact      ; The output from fact to multi remains
on the stack
96          PUSH R1        ; Pass para
97          BWL .multi
98          POP R1         ; Get res
99          ADDIB SP,#1    ; POP
100         STW R1,[SP,#3]
101         POP LR
102         POP R1
103         POP R0
104         RET

```

```

105 .retOne ADDIB R1,#1      ; Trade off code size to avoid jump
    checking
106     STW R1,[SP,#3]
107     POP LR
108     POP R1
109     POP R0
110     RET
111 .multi PUSH R2           ; R2 is M
112     PUSH R3             ; R3 is Q
113     PUSH R4             ; R4 is ACC
114     PUSH R6             ; R6 is 1
115     PUSH R1             ; R1 is temp
116     LDW R2,[SP,#5]
117     LDW R3,[SP,#6]
118     SUB R4,R4,R4
119     LUI R6,#0
120     LLI R6,#1           ; load 1 into R6 for compare
121     AND R1,R2,R6        ; Loop unroll for maximum fastness
122     CMPI R1,#0
123     BE .sh1
124     ADD R4,R4,R3
125 .sh1  LSL R3,R3,#1
126     LSR R2,R2,#1
127     AND R1,R2,R6
128     CMPI R1,#0
129     BE .sh2
130     ADD R4,R4,R3
131 .sh2  LSL R3,R3,#1
132     LSR R2,R2,#1
133     AND R1,R2,R6
134     CMPI R1,#0
135     BE .sh3
136     ADD R4,R4,R3
137 .sh3  LSL R3,R3,#1
138     LSR R2,R2,#1
139     AND R1,R2,R6
140     CMPI R1,#0
141     BE .sh4
142     ADD R4,R4,R3
143 .sh4  LSL R3,R3,#1
144     LSR R2,R2,#1
145     AND R1,R2,R6
146     CMPI R1,#0
147     BE .sh5
148     ADD R4,R4,R3

```

```

149 .sh5    LSL R3,R3,#1
150        LSR R2,R2,#1
151        AND R1,R2,R6
152        CMPI R1,#0
153        BE .sh6
154        ADD R4,R4,R3
155 .sh6    LSL R3,R3,#1
156        LSR R2,R2,#1
157        AND R1,R2,R6
158        CMPI R1,#0
159        BE .sh7
160        ADD R4,R4,R3
161 .sh7    LSL R3,R3,#1
162        LSR R2,R2,#1
163        AND R1,R2,R6
164        CMPI R1,#0
165        BE .sh8
166        ADD R4,R4,R3
167 .sh8    LSL R3,R3,#1
168        LSR R2,R2,#1
169        AND R1,R2,R6
170        CMPI R1,#0
171        BE .sh9
172        ADD R4,R4,R3
173 .sh9    LSL R3,R3,#1
174        LSR R2,R2,#1
175        AND R1,R2,R6
176        CMPI R1,#0
177        BE .sh10
178        ADD R4,R4,R3
179 .sh10   LSL R3,R3,#1
180        LSR R2,R2,#1
181        AND R1,R2,R6
182        CMPI R1,#0
183        BE .sh11
184        ADD R4,R4,R3
185 .sh11   LSL R3,R3,#1
186        LSR R2,R2,#1
187        AND R1,R2,R6
188        CMPI R1,#0
189        BE .sh12
190        ADD R4,R4,R3
191 .sh12   LSL R3,R3,#1
192        LSR R2,R2,#1
193        AND R1,R2,R6

```

```

194      CMPI R1,#0
195      BE .sh13
196      ADD R4,R4,R3
197 .sh13  LSL R3,R3,#1
198      LSR R2,R2,#1
199      AND R1,R2,R6
200      CMPI R1,#0
201      BE .sh14
202      ADD R4,R4,R3
203 .sh14  LSL R3,R3,#1
204      LSR R2,R2,#1
205      AND R1,R2,R6
206      CMPI R1,#0
207      BE .sh15
208      ADD R4,R4,R3
209 .sh15  LSL R3,R3,#1
210      LSR R2,R2,#1
211      AND R1,R2,R6
212      CMPI R1,#0
213      BE .sh16
214      ADD R4,R4,R3
215 .sh16  LSL R3,R3,#1
216      LSR R2,R2,#1
217      STW R4,[SP,#5]
218      POP R1
219      POP R6
220      POP R4
221      POP R3
222      POP R2
223      RET

```

Listing 5: interrupt.asm