ELEC6027 - VLSI Design Project : Programmers Guide

Team R4

21st 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

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

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

$$\begin{aligned}
\text{Rd} &\leftarrow \text{Ra} + \text{Rb} \\
\text{N} &\leftarrow \text{if Result} < 0 \text{ then 1, else 0} \\
\text{Z} &\leftarrow \text{if Result} = 0 \text{ then 1, else 0} \\
\text{V} &\leftarrow \text{if (+Ra, +Rb, -Result) or} \\
&\qquad \text{(-Ra, -Rb, +Result) then 1, else 0} \\
\text{C} &\leftarrow \text{if (Result} > 2^{16} - 1\text{) or} \\
&\qquad \text{(Result} < -2^{16}\text{) then 1, else 0}
\end{aligned}$$

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

$$
\begin{aligned}
\text{Rd} &\leftarrow \text{Ra} + \#\text{imm5} \\
\text{N} &\leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0 \\
\text{Z} &\leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0 \\
\text{V} &\leftarrow \text{if (+Ra, +\#imm5, -Result) or} \\
&\qquad \text{(-Ra, -\#imm5, +Result) then } 1, \text{ else } 0 \\
\text{C} &\leftarrow \text{if (Result} > 2^{16} - 1) \text{ or} \\
&\qquad (\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0
\end{aligned}
$$

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

$$\text{Rd} \leftarrow \text{Rd} + \#\text{imm8}$$

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

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

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

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

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

$Rd \leftarrow Ra + Rb + C$

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

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

$V \leftarrow$ if (+Ra, +(Rb+CFlag), -Result) or

(-Ra, -(Rb+CFlag), +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[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 $\leftarrow$ if Result $<$ 0 then 1, else 0

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

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

(-Ra, -(#imm5+CFlag), +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[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 $\leftarrow$ if Result <0 then 1, else 0

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

V $\leftarrow$ 0

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

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

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

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

V $\leftarrow$ if (+Ra, +(Rb-CFlag), -Result) or

(-Ra, -(Rb-CFlag), +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[Rb] 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 $\leftarrow$ if Result $<$ 0 then 1, else 0

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

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

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

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

(Result $< -2^{16}$) then 1, 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 AND Rb

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

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 OR Rb

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

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 XOR Rb

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

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 $\leftarrow$ NOT Ra

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

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 NAND Rb

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

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 NOR Rb

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

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

$Rd \leftarrow Ra << \#imm4$

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

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

$V \leftarrow$ UNPREDICTABLE

$C \leftarrow$ UNPREDICTABLE

Description

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 >> #imm4

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

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 >>> #imm4

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

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

$$\begin{aligned}
\text{Rd} &\leftarrow \text{Mem[Ra + \#imm5]} \\
\text{N} &\leftarrow \text{if Result} < 0 \text{ then 1, else 0} \\
\text{Z} &\leftarrow \text{if Result} = 0 \text{ then 1, else 0} \\
\text{V} &\leftarrow \text{if (+Ra, +\#imm5, -Result) or} \\
&\qquad \text{(-Ra, -\#imm5, +Result) then 1, else 0} \\
\text{C} &\leftarrow \text{if (Result} > 2^{16} - 1\text{) or} \\
&\qquad \text{(Result} < -2^{16}\text{) then 1, else 0}
\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

$V \leftarrow$ UNPREDICTABLE

$C \leftarrow$ 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

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

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

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

$V \leftarrow$ UNPREDICTABLE

$C \leftarrow$ 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

$$PC \leftarrow PC + \#imm8$$

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

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

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

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

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

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

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 (n&!v OR !n&v)?

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 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 (n&v OR !n&!v)?

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 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

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

$N \leftarrow$ 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

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 $\leftarrow$ UNPREDICTABLE

Z $\leftarrow$ UNPREDICTABLE

V $\leftarrow$ UNPREDICTABLE

C $\leftarrow$ 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

$$
\begin{aligned}
PC &\leftarrow Ra + \#imm5 \\
N &\leftarrow \text{if Result} < 0 \text{ then } 1, \text{ else } 0 \\
Z &\leftarrow \text{if Result} = 0 \text{ then } 1, \text{ else } 0 \\
V &\leftarrow \text{if } (+Rd, +\#imm8, \text{-Result}) \text{ or} \\
&\qquad (-Rd, -\#imm8, +\text{Result}) \text{ then } 1, \text{ else } 0 \\
C &\leftarrow \text{if } (\text{Result} > 2^{16} - 1) \text{ or} \\
&\qquad (\text{Result} < -2^{16}) \text{ then } 1, \text{ else } 0
\end{aligned}
$$

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 — eg. POP R3

POP RL — eg. POP RL

Operation

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

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

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

$V \leftarrow$ UNPREDICTABLE

$C \leftarrow$ UNPREDICTABLE

Description

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

$PC \leftarrow Mem[R7]$

$N \leftarrow$ UNPREDICTABLE

$Z \leftarrow$ UNPREDICTABLE

$V \leftarrow$ UNPREDICTABLE

$C \leftarrow$ UNPREDICTABLE

Description

Restore program counter to its value before interrupt occured, 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

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

N $\leftarrow$ N

Z $\leftarrow$ Z

V $\leftarrow$ V

C $\leftarrow$ C

Description

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

Addressing Modes: Register Indirect, Postdecrement.

4.42 LDF — Load Status Flags

Format

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

Syntax

LDF — eg. LDF

Operation

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

$N \leftarrow N$

$Z \leftarrow Z$

$V \leftarrow V$

$C \leftarrow C$

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 mneumonic 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 mneumonic 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 algorithm is described using C in listing 1. In the 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 (multiplied by two).

```c
unsigned short multi(unsigned char mul, unsigned char quo){
    unsigned char M,Q,i;
    unsigned short A;
    A = 0; M = mul; Q = quo;
    for(i=0;i<16;i++){
        if(M[0] == 1){
            A = A + Q;
        }
        Q = Q << 1;
        M = M >> 1;
    }
    return A;
}
```

Listing 1: shiftAndAdd.c

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 verfied using the final design of the processor.

A.1 Multiply

```
        LUI SP, #7        ; Init SP
        LLI SP, #208
        LUI R0, #8        ; SWs ADDR
        LLI R0, #0
        LDW R0,[R0,#0]    ; READ SWs
        LUI R1, #0
        LLI R1, #255      ; 0x00FF in R1
        AND R1,R0,R1      ; Lower byte SWs in R1
        LSR R0,R0,#8      ; Upper byte SWs in R0
        PUSH R0
        PUSH R1
        BWL .multi        ; Run Subroutine
        POP R1            ; Result
        ADDIB SP,#1       ; Duummy pop
        LUI R4, #8
        LLI R4, #1        ; Address of LEDS
        STW R1,[R4,#0]    ; Result on LEDS
.end    BR .end           ; Finish loop
.multi  LDW R2,[SP,#0]    ; R2 - Multiplier
        LDW R3,[SP,#1]    ; R3 - Quotient
        SUB R4,R4,R4      ; R4 - Accumulator
        LUI R6,#0         ; R6 - Constant 1
```

```
        LLI R6,#1       ; R1 - CMP var
        AND R1,R2,R6    ; Stage 1
        CMPI R1,#0      ; LSb ?
        BE .sh1
        ADD R4,R4,R3    ; (LSb == 1)?
.sh1    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6    ; Stage 2
        CMPI R1,#0
        BE .sh2
        ADD R4,R4,R3
.sh2    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6    ; Stage 3
        CMPI R1,#0
        BE .sh3
        ADD R4,R4,R3
.sh3    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6    ; Stage 4
        CMPI R1,#0
        BE .sh4
        ADD R4,R4,R3
.sh4    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6    ; Stage 5
        CMPI R1,#0
        BE .sh5
        ADD R4,R4,R3
.sh5    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6    ; Stage 6
        CMPI R1,#0
        BE .sh6
        ADD R4,R4,R3
.sh6    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6    ; Stage 7
        CMPI R1,#0
        BE .sh7
        ADD R4,R4,R3
.sh7    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6    ; Stage 8
        CMPI R1,#0
```

```
        BE .sh8
        ADD R4,R4,R3
.sh8    STW R4,[SP,#0]    ; Res on stack frame
        RET
```

Listing 2: multiply.asm

A.2 Factorial

```
        LUI R7, #7
        LLI R7, #208
        LUI R0, #8        ; Address in R0
        LLI R0, #0
        LDW R0,[R0,#0]    ; Read switches into R0
        LUI R1,#0         ; Calculate only 8 or less
        LLI R1,#8
        CMP R1,R0
        BE .do
        SUBIB R1,#1
        AND R0,R0,R1
.do     PUSH R0           ; Pass para
        BWL .fact         ; Run Subroutine
        POP R0            ; Para overwritten with result
        LUI R4, #8
        LLI R4, #1        ; Address of LEDS
        STW R0,[R4,#0]    ; Result on LEDS
.end    BR .end           ; finish loop
.fact   PUSH R0
        PUSH R1
        PUSH LR
        LDW R1,[SP,#3]    ; Get para
        ADDIB R1,#0
        BE .retOne        ; 0! = 1
        SUBI R0,R1,#1
        PUSH R0           ; Pass para
        BWL .fact         ; The output from fact to multi remains
    on the stack
        PUSH R1           ; Pass para
        BWL .multi
        POP R1            ; Get res
        ADDIB SP,#1       ; POP
        STW R1,[SP,#3]
        POP LR
        POP R1
```

```
        POP R0
        RET
.retOne ADDIB R1,#1      ; Trade off code size to avoid jump checking
        STW R1,[SP,#3]
        POP LR
        POP R1
        POP R0
        RET
.multi  PUSH R2          ; R2 is M
        PUSH R3          ; R3 is Q
        PUSH R4          ; R4 Is ACC
        PUSH R6          ; R6 is 1
        PUSH R1          ; R1 is temp
        LDW R2,[SP,#5]
        LDW R3,[SP,#6]
        SUB R4,R4,R4
        LUI R6,#0
        LLI R6,#1        ; load 1 into R6 for compare
        AND R1,R2,R6     ; Loop unroll for maximum fastness
        CMPI R1,#0
        BE .sh1
        ADD R4,R4,R3
.sh1    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6
        CMPI R1,#0
        BE .sh2
        ADD R4,R4,R3
.sh2    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6
        CMPI R1,#0
        BE .sh3
        ADD R4,R4,R3
.sh3    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6
        CMPI R1,#0
        BE .sh4
        ADD R4,R4,R3
.sh4    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6
        CMPI R1,#0
```

```
            BE .sh5
            ADD R4,R4,R3
.sh5        LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh6
            ADD R4,R4,R3
.sh6        LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh7
            ADD R4,R4,R3
.sh7        LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh8
            ADD R4,R4,R3
.sh8        LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh9
            ADD R4,R4,R3
.sh9        LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh10
            ADD R4,R4,R3
.sh10       LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh11
            ADD R4,R4,R3
.sh11       LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh12
            ADD R4,R4,R3
.sh12       LSL R3,R3,#1
```

```
          LSR  R2,R2,#1
          AND  R1,R2,R6
          CMPI R1,#0
          BE   .sh13
          ADD  R4,R4,R3
.sh13     LSL  R3,R3,#1
          LSR  R2,R2,#1
          AND  R1,R2,R6
          CMPI R1,#0
          BE   .sh14
          ADD  R4,R4,R3
.sh14     LSL  R3,R3,#1
          LSR  R2,R2,#1
          AND  R1,R2,R6
          CMPI R1,#0
          BE   .sh15
          ADD  R4,R4,R3
.sh15     LSL  R3,R3,#1
          LSR  R2,R2,#1
          AND  R1,R2,R6
          CMPI R1,#0
          BE   .sh16
          ADD  R4,R4,R3
.sh16     LSL  R3,R3,#1
          LSR  R2,R2,#1
          STW  R4,[SP,#5]
          POP  R1
          POP  R6
          POP  R4
          POP  R3
          POP  R2
          RET
```

Listing 3: factorial.asm

A.3 Random

```
          LUI R7,  #7
          LLI R7,  #208
          LUI R0,  #8          ; Address in R0
          LLI R0,  #0
          LDW R0,[R0,#0]       ; Read switches into R0
          LUI R1,  #8
          LLI R1,  #1          ; Address of LEDS
```

```
        PUSH R0
.loop   BWL .rand       ; 1
        BWL .rand       ; 2
        BWL .rand       ; 3
        BWL .rand       ; 4
        BWL .rand       ; 5
        BWL .rand       ; 6
        BWL .rand       ; 7
        BWL .rand       ; 8
        BWL .rand       ; 9
        BWL .rand       ; 10
        BWL .rand       ; 11
        BWL .rand       ; 12
        BWL .rand       ; 13
        BWL .rand       ; 14
        BWL .rand       ; 15
        BWL .rand       ; 16
        LDW R0,[SP,#0]  ; No POP as re-run
        STW R0,[R1,#0]  ; Result on LEDS
        BR .loop
.rand   LDW R2,[SP,#0]  ; Linear feedback shift register sim
        LUI R3,#0       ; Three
        LLI R3,#3
        AND R4,R3,R2    ; Bottom two bits of input
        LSR R5,R2,#1
        CMP R4,R3       ; Three
        BE .done
        SUB R3,R3,R3
        CMP R4,R3       ; Zero
        BE .done
        LUI R3,#128
        LLI R3,#0
        OR R5,R5,R3
.done   STW R5,[SP,#0]
        RET
```

Listing 4: random.asm

A.4 Interrupt

```
        DISI            ; Reset is off anyway
        LUI R7, #7
        LLI R7, #208
        LUI R0, #1      ; R0 is read ptr    0x0100
```

```
        ADDI R1,R0,#2     ; 0x0102
        STW R1,[R0,#0]    ; Read ptr set to  0x0102
        STW R1,[R0,#1]    ; Write ptr set to 0x0102
        LUI R0,#160       ; Address of Serial control reg
        LLI R1,#01        ; Data to enable ints
        STW R1,[R0,#1]    ; Store 0x001 @ 0xA001
        LLI R3,#18        ; main line -1 in .main
        ENAI
        BR .main
.isr    DISI
        STF               ; Keep flags
        PUSH R0           ; Save only this for now
        LUI R0,#160
        LLI R0,#0
        LDW R0,[R0,#0]    ; R1 contains read serial data
        ENAI
        PUSH R1
        PUSH R2
        PUSH R3
        PUSH R4
        LUI R1,#1
        LLI R1,#0
        LDW R2,[R1,#0]    ; R2 contains read ptr
        ADDI R3,R1,#1
        LDW R4,[R3,#0]    ; R4 contain the write ptr
        SUBIB R2,#1       ; Get out if W == R - 1
        CMP R4,R2
        BE .isrOut
        ADDIB R2,#1
        LUI R1,#1
        LLI R1,#2
        CMP R2,R1
        BNE .write
        ADDIB R1,#3
        CMP R4,R1
        BE .isrOut
.write  STW R0,[R4,#0]    ; Write to buffer
        ADDIB R4,#1
        LUI R1,#1
        LLI R1,#6
        CMP R1,R4
        BNE .wrapW
        SUBIB R4,#4
.wrapW  STW R4,[R3,#0]    ; Inc write ptr
.isrOut POP R4
```

```
        POP R3
        POP R2
        POP R1
        POP R0
        LDF
        RETI
.main   LUI R0, #1        ; Read ptr address in R0
        LLI R0, #0
        LDW R2,[R0,#0]    ; Read ptr in R2
        LDW R3,[R0,#1]    ; Write ptr in R3
        CMP R2,R3
        BE .main          ; Jump back if the same
        LDW R3,[R2,#0]    ; Load data out of buffer
        ADDIB R2,#1       ; Inc read ptr
        SUB R0,R0,R0
        LUI R0,#1
        LLI R0,#6
        SUB R0,R0,R2
        BNE .wrapR
        SUBIB R2,#4
.wrapR  LUI R0, #1        ; Read ptr address in R0
        LLI R0, #0
        STW R2,[R0,#0]    ; Store new read pointer
        SUB R4,R4,R4
        LLI R4,#15
        AND R3,R4,R3
        CMPI R3,#8
        BE .do
        LLI R4,#7
        AND R3,R3,R4
.do     PUSH R3
        BWL .fact
        POP R3
        LUI R4,#8
        LLI R4,#1         ; Address of LEDs
        STW R3,[R4,#0]    ; Put factorial on LEDs
        BR .main          ; look again
.fact   PUSH R0
        PUSH R1
        PUSH LR
        LDW R1,[SP,#3]    ; Get para
        ADDIB R1,#0
        BE .retOne        ; 0! = 1
        SUBI R0,R1,#1
        PUSH R0           ; Pass para
```

```
        BWL .fact       ; The output from fact to multi remains on the stack
        PUSH R1         ; Pass para
        BWL .multi
        POP R1          ; Get res
        ADDIB SP,#1     ; POP
        STW R1,[SP,#3]
        POP LR
        POP R1
        POP R0
        RET
.retOne ADDIB R1,#1     ; Trade off code size to avoid jump checking
        STW R1,[SP,#3]
        POP LR
        POP R1
        POP R0
        RET
.multi  PUSH R2         ; R2 is M
        PUSH R3         ; R3 is Q
        PUSH R4         ; R4 Is ACC
        PUSH R6         ; R6 is 1
        PUSH R1         ; R1 is temp
        LDW R2,[SP,#5]
        LDW R3,[SP,#6]
        SUB R4,R4,R4
        LUI R6,#0
        LLI R6,#1       ; load 1 into R6 for compare
        AND R1,R2,R6    ; Loop unroll for maximum fastness
        CMPI R1,#0
        BE .sh1
        ADD R4,R4,R3
.sh1    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6
        CMPI R1,#0
        BE .sh2
        ADD R4,R4,R3
.sh2    LSL R3,R3,#1
        LSR R2,R2,#1
        AND R1,R2,R6
        CMPI R1,#0
        BE .sh3
        ADD R4,R4,R3
.sh3    LSL R3,R3,#1
```

```
            LSR  R2,R2,#1
            AND  R1,R2,R6
            CMPI R1,#0
            BE   .sh4
            ADD  R4,R4,R3
.sh4        LSL  R3,R3,#1
            LSR  R2,R2,#1
            AND  R1,R2,R6
            CMPI R1,#0
            BE   .sh5
            ADD  R4,R4,R3
.sh5        LSL  R3,R3,#1
            LSR  R2,R2,#1
            AND  R1,R2,R6
            CMPI R1,#0
            BE   .sh6
            ADD  R4,R4,R3
.sh6        LSL  R3,R3,#1
            LSR  R2,R2,#1
            AND  R1,R2,R6
            CMPI R1,#0
            BE   .sh7
            ADD  R4,R4,R3
.sh7        LSL  R3,R3,#1
            LSR  R2,R2,#1
            AND  R1,R2,R6
            CMPI R1,#0
            BE   .sh8
            ADD  R4,R4,R3
.sh8        LSL  R3,R3,#1
            LSR  R2,R2,#1
            AND  R1,R2,R6
            CMPI R1,#0
            BE   .sh9
            ADD  R4,R4,R3
.sh9        LSL  R3,R3,#1
            LSR  R2,R2,#1
            AND  R1,R2,R6
            CMPI R1,#0
            BE   .sh10
            ADD  R4,R4,R3
.sh10       LSL  R3,R3,#1
            LSR  R2,R2,#1
            AND  R1,R2,R6
            CMPI R1,#0
```

```
            BE .sh11
            ADD R4,R4,R3
.sh11       LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh12
            ADD R4,R4,R3
.sh12       LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh13
            ADD R4,R4,R3
.sh13       LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh14
            ADD R4,R4,R3
.sh14       LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh15
            ADD R4,R4,R3
.sh15       LSL R3,R3,#1
            LSR R2,R2,#1
            AND R1,R2,R6
            CMPI R1,#0
            BE .sh16
            ADD R4,R4,R3
.sh16       LSL R3,R3,#1
            LSR R2,R2,#1
            STW R4,[SP,#5]
            POP R1
            POP R6
            POP R4
            POP R3
            POP R2
            RET
```

Listing 5: interrupt.asm