ELEC6027 - VLSI Design Project : Programmers Guide

Team R4

April 6, 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 fall within. Data manipulation instructions have either a register register or register immediate addressing mode for performing arithmetic, logical and shifting type 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.

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 General Purpose Register

Pseudocode Notation

Symbol	Meaning
$\leftarrow$, $\rightarrow$	Assignment
Ra[x]	Bit x of register Ra
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

4.1 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 ← Ra + Rb
 N ← Rd[31]
 Z ← if Rd = 0 then 1, else 0
 V ← if (+Ra, +Rb, -Result) or
         (-Ra, -Rb, +Result) then 1, else 0
 C ← if (Result > $2^{15}$) 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.

The addressing mode of this instruction is Register Register.

4.2 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

$\text{Rd} \leftarrow \text{Ra} + \#\text{imm5}$

$\text{N} \leftarrow \text{Rd}[31]$

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

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

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

$\text{C} \leftarrow \text{if (Result} > 2^{15}\text{) or}$

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

Description

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

The addressing mode of this instruction is Register Immediate.

4.3 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 ← Rd + #imm8
 N ← Rd[31]
 Z ← if Rd = 0 then 1, else 0
 V ← if (+Rd, +#imm8, -Result) or
        (-Rd, -#imm8, +Result) then 1, else 0
 C ← if (Result > $2^{15}$) or
        (Result < $-2^{16}$) 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.

The addressing mode of this instruction is Register Immediate.

4.4 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

$\text{Rd} \leftarrow \text{Ra} + \text{Rb} + \text{C}$

$\text{N} \leftarrow \text{Rd}[31]$

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

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

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

$\text{C} \leftarrow \text{if (Result} > 2^{15}\text{) or}$

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

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.

The addressing mode of this instruction is Register Register.

4.5 ADCI Add Immediate With Carry

Format

<table>
<tr><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>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td colspan="3">Rd</td><td colspan="3">Ra</td><td colspan="5">imm5</td></tr>
</table>

Syntax

ADCI Rd, Ra, #imm5 eg. ADCI R5, R4, #7

Operation

```
$\text{Rd} \leftarrow \text{Ra} + \#\text{imm5} + \text{C}$
$\text{N} \leftarrow \text{Rd}[31]$
$\text{Z} \leftarrow \text{if Rd} = 0 \text{ then 1, else 0}$
$\text{V} \leftarrow$ if (+Ra, +(#imm5+CFlag), -Result) or
        (-Ra, -(#imm5+CFlag), +Result) then 1, else 0
$\text{C} \leftarrow$ if (Result $> 2^{15}$) 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.

The addressing mode of this instruction is Register Immediate.

4.6 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 $\leftarrow$ Rd[31]

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

V $\leftarrow$ 0

C $\leftarrow$ if (Result $> 2^{15}$) 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.

The addressing mode of this instruction is Register Register.

4.7 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 $\leftarrow$ Rd[31]
 Z $\leftarrow$ if Rd $=$ 0 then 1, else 0
 V $\leftarrow$ if (+Ra, +Rb, -Result) or
          (-Ra, -Rb, +Result) then 1, else 0
 C $\leftarrow$ if (Result $> 2^{15}$) 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.

The addressing mode of this instruction is Register Register.

4.8 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 $\leftarrow$ Rd[31]

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

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

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

C $\leftarrow$ if (Result $> 2^{15}$) 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.

The addressing mode of this instruction is Register Immediate.

4.9 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

```
$Rd \leftarrow Rd - \#imm8$
 $N \leftarrow Rd[31]$
 $Z \leftarrow$ if $Rd = 0$ then 1, else 0
 $V \leftarrow$ if (+Rd, +#imm8, -Result) or
         (-Rd, -#imm8, +Result) then 1, else 0
 $C \leftarrow$ if (Result $> 2^{15}$) 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.

The addressing mode of this instruction is Register Immediate.

4.10 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 $\leftarrow$ Rd[31]

Z $\leftarrow$ if Rd $=$ 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 $> 2^{15}$) 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.

The addressing mode of this instruction is Register Register.

4.11 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 $\leftarrow$ Rd[31]

Z $\leftarrow$ if Rd $=$ 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 $> 2^{15}$) 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.

The addressing mode of this instruction is Register Immediate.

4.12 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 ← Rd[31]
Z ← if Rd = 0 then 1, else 0
V ← if (+Ra, +Rb, -Result) or
      (-Ra, -Rb, +Result) then 1, else 0
C ← if (Result > $2^{15}$) 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.

The addressing mode of this instruction is Register Register.

4.13 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 ← Rd[31]
Z ← if Rd = 0 then 1, else 0
V ← if (+Ra, +#imm5, -Result) or
      (-Ra, -#imm5, +Result) then 1, else 0
C ← if (Result > $2^{15}$) 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.

The addressing mode of this instruction is Register Immediate.

4.14 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 ← Ra AND Rb

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Register Register.

4.15 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 ← Ra OR Rb
N ← N
Z ← Z
V ← V
C ← C

Description

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

The addressing mode of this instruction is Register Register.

4.16 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 ← Ra XOR Rb

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Register Register.

4.17 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 ← NOT Ra

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Register Register.

4.18 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 ← Ra NAND Rb

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Register Register.

4.19 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 ← Ra NOR Rb

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Register Register.

4.20 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 ← Ra << #imm4

N ← N

Z ← Z

V ← V

C ← C

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.

The addressing mode of this instruction is Register Immediate.

4.21 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 ← Ra >> #imm4
N ← N
Z ← Z
V ← V
C ← C

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.

The addressing mode of this instruction is Register Immediate.

4.22 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 $\leftarrow$ N

Z $\leftarrow$ Z

V $\leftarrow$ V

C $\leftarrow$ C

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.

The addressing mode of this instruction is Register Immediate.

4.23 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 ← Mem[Ra + #imm5]

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Base Plus Offset.

4.24 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 → Mem[Ra + #imm5]

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Base Plus Offset.

4.25 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 ← {#imm8, 0}
N ← N
Z ← Z
V ← V
C ← C

Description

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

The addressing mode of this instruction is Register Immediate.

4.26 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 ← {Rd[15:8], #imm8}

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Register Immediate.

4.27 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 ← PC + #imm8

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is PC Relative.

4.28 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 ← PC + #imm8 (z==0)?

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is PC Relative.

4.29 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 ← PC + #imm8 (z==1)?
N ← N
Z ← Z
V ← V
C ← C

Description

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

The addressing mode of this instruction is PC Relative.

4.30 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 ← PC + #imm8 (n&!v OR !n&v)?

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is PC Relative.

4.31 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 ← PC + #imm8 (n&v OR !n&!v)?

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is PC Relative.

4.32 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 ← PC + 1; PC ← PC + #imm8

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is PC Relative.

4.33 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 ← LR

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Register Indirect.

4.34 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 ← Ra + #imm5

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Base Plus Offset.

4.35 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			X	X	X	X	X

Syntax

PUSH Ra	eg. PUSH R3
PUSH RL	eg. PUSH RL

Operation

reg → Mem[R7]; R7 ← R7 - 1

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing mode of this instruction is Register Indirect Postdecrement.

4.36 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			X	X	X	X	X

Syntax

POP Ra	eg. POP R3
POP RL	eg. POP RL

Operation

R7 ← R7 + 1; Mem[R7] ← reg;

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing modes of this instruction are Register Indirect Preincrement.

4.37 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

$$
\begin{aligned}
PC &\leftarrow \text{Mem}[R7] \\
N &\leftarrow N \\
Z &\leftarrow Z \\
V &\leftarrow V \\
C &\leftarrow C
\end{aligned}
$$

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.

The addressing mode of this instruction is Register Indirect.

4.38 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 ← 1

N ← N

Z ← Z

V ← V

C ← C

Description

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

4.39 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 ← 0

N ← N

Z ← Z

V ← V

C ← C

Description

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

4.40 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] ← {12-bit 0, Z, C, V, N}; R7 ← R7 - 1;

N ← N

Z ← Z

V ← V

C ← C

Description

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

The addressing more of this instruction is Register Indirect Postdecrement.

4.41 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 ← R7 + 1; {Z, C, V, N} ← Mem[R7][3:0]

N ← N

Z ← Z

V ← V

C ← 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).

The addressing more of this instruction is 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] MNEUMONIC, 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 exceed 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. Branches may occur moving out of this subroutine, if within an accepted distance, but are not allowed into this subroutine with the exception of a RET call.

Outputted machine code requires the ISR to be in a specific location in memory. This is automatically accounted for by the assembler as well as including some initialization code before main program execution. This start-up code includes initialization of the stack pointer to 2047.

6.2 Assembler Directives

Currently there is one supported assembler directive for assigning meaningful names to each of the general purpose registers. Definitions can occur at any point within the file and create a valid equivalence 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 defintions 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 "python 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.

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

Lorem Ipsum...

8 Simulation

Lorem Ipsum...