Instruction Set Summary

	Mnemonic	Syntax	Semantics	Flags	Encoding	Opcode	Cond.
1	ADD	ADD Rd, Ra, Rb	Rd ← Ra + Rb	c,v,n,z	A	00100	-
2	ADDI	ADDI Rd, Ra, #imm5	Rd ← Ra + imm5	c,v,n,z	A	00101	-
3	ADDIB	ADDIB Rd, #imm8	Rd ← Rd + imm8	c,v,n,z	B	11000	-
4	ADC	ADC Rd, Ra, Rb	Rd ← Ra + Rb + c	c,v,n,z	A	00110	-
5	ADCI	ADCI Rd, Ra, #imm5	Rd ← Ra + imm5 + c	c,v,n,z	A	00111	-
6	NEG	NEG Rd	Rd ← 0 - Rd	c,v,n,z	A	01000	-
7	SUB	SUB Rd, Ra, Rb	Rd ← Ra - Rb	c,v,n,z	A	01001	-
8	SUBI	SUBI Rd, Ra, #imm5	Rd ← Ra - imm5	c,v,n,z	A	01010	-
9	SUBIB	SUBIB Rd, #imm8	Rd ← Rd - imm8	c,v,n,z	B	11001	-
10	SUC	SUC Rd, Ra, Rb	Rd ← Ra - Rb - NOT c	c,v,n,z	A	01011	-
11	SUCI	SUCI Rd, Ra, #imm5	Rd ← Ra - imm5 - NOT c	c,v,n,z	A	01100	-
12	CMP	CMP Ra, Rb	Ra - Rb	c,v,n,z	A	01101	-
13	AND	AND Rd, Ra, Rb	Rd ← Ra AND Rb	z	A	10000	-
14	OR	OR Rd, Ra, Rb	Rd ← Ra OR Rb	z	A	10001	-
15	XOR	XOR Rd, Ra, Rb	Rd ← Ra XOR Rb	z	A	10010	-
16	NOT	NOT Rd, Ra	Rd ← NOT Ra	z	A	10011	-
17	NAND	NAND Rd, Ra, Rb	Rd ← Ra NAND Rb	z	A	10100	-
18	NOR	NOR Rd, Ra, Rb	Rd ← Ra NOR Rb	z	A	10101	-
19	LSL	LSL Rd, Ra, #imm4	Rd ← Ra << imm4	-	A	00001	-
20	LSR	LSR Rd, Ra, #imm4	Rd ← Ra >> imm4	-	A	00010	-
21	ASR	ASR Rd, Ra, #imm4	Rd ← Ra >> imm4	-	A	00011	-
22	LDW	LDW Rd, [Ra,Ro]	Rd ← Mem[Ra + Ro]	-	C	11101	-
23	SDW	SDW Rd, [Ra,Ro]	Mem[Ra + Ro] ← Rd	-	C	11101	-
24	LUI	LUI Rd, #imm8	Rd[15:8] ← imm8	-	B	11010	-
25	LLI	LLI Rd, #imm8	Rd[7:0] ← imm8	-	B	11011	-
26	BR	BR LABEL	PC ← PC + imm8	-	D	11111	000
27	BNE	BNE LABEL	(z==0)? PC ← PC + imm8	-	D	11111	110
28	BE	BE LABEL	(z==1)? PC ← PC + imm8	-	D	11111	111
29	BLT	BLT LABEL	(n&~v OR ~n&v)? PC ← PC + imm8	-	D	11111	100
30	BGE	BGE LABEL	(n&v OR ~n&~v)? PC ← PC + imm8	-	D	11111	101
31	BWL	BWL LABEL	LR ← PC PC ← PC + LABEL	-	D	11111	011
32	RET	RET	PC ← LR	-	D	11111	010
33	ABR	ABR Ra	PC ← Ra	-	D	11111	001
34	PUSH	PUSH {Ra, RL}	SP ← SP + 1 Update stack	-	E	11100	-
35	POP	POP {Ra, RL}	SP ← SP - 1 Update stack	-	E	11100	-

Opcodes 00000, 11000 to 11011, 11110 are UNDEFINED

Bit encoding has been optimized so that subsections can be identified by first 2/3 bits to aid controller simplicity.

General Instruction Formatting

	Instruction Type	Sub-Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	**Data Manipulation**	**Register**	Opcode					X	X	Rb			Ra			Rd		
		Immediate						imm4/5										
B	**Byte Immediate**		Opcode					imm8								Rd		
C	**Data Transfer**		1	1	1	0	1	X	LS	Ro			Ra			Rd		
D	**Control Transfer**	**Others**	1	1	1	1	1	imm8								Cond.		
		Absolute						X	X	X	X	X	Ra					
E	**Stack Operations**		1	1	1	0	0	L	X	X	X	X	X	X	X	Ra		

LS: 1 = Load Data, 0 = Store Data No. = 1 or 2 or 3 L = Link Register

Example Coding

Data Manipulation

These operations are performed by the Arithmetic Logic Unit and examples are shown below.

1	ADD R5, R3, R4	R5 ← R3 + R4
2	ADDI R5, R3, #9	R5 ← R3 + 9
4	ADC R5, R3, R4	R5 ← R3 + R4 + c
5	ADCI R5, R3, #9	R5 ← R3 + 9 + c
6	NEG R5	R5 ← 0 - R5
7	SUB R5, R3, R4	R5 ← R3 - R4
8	SUBI R5, R3, #9	R5 ← R3 - 9
10	SUC R5, R3, R4	R5 ← R3 - R4 - NOT c
11	SUCI R5, R3, #9	R5 ← R3 - 9 - NOT c
12	CMP R3, R4	R3 - R4

13	AND R5, R3, R4	R5 ← R3 AND R4
14	OR R5, R3, R4	R5 ← R3 OR R4
15	XOR R5, R3, R4	R5 ← R3 XOR R4
16	NOT R5, R3	R5 ← NOT R3
17	NAND R5, R3, R4	R5 ← R3 NAND R4
18	NOR R5, R3, R4	R5 ← R3 NOR R4
19	LSL R5, R3, #3	R5 ← R3 << 3
20	LSR R5, R3, #3	R5 ← R3 >> 3
21	ASR R5, R3, #3	R5 ← R3 >> 3

The value ‘c’ corresponds to the carry bit flag in the ALU from the previous calculation.
CMP is a comparison instruction for performing a subtraction without saving the result. The updated status flags can then be used for a conditional branch.

Byte Immediate

These instructions ADD/SUB an 8-bit immediate value from the given register, replacing the result back in that register. Alternatively, the same formatting is used for loading the upper/lower byte of a register with an 8-bit immediate value.

3	ADDIB R5, #150	R5 ← R5 + 150
9	SUBIB R5, #150	R5 ← R5 - 150
24	LUI R5, #150	R5[15:8] ← 150
25	LLI R5, #150	R5[7:0] ← 150

Data Transfer

When loading data, the value at the memory location held in Ra, adds an offset held in Ro, and replaces the returned value in register Rd. When storing data, the same functionality is used, only with data transferring in opposite direction.

22	LDW R5, [R3, R2]	R5 ← Mem[R3 + R2]
23	SDW R5, [R3, R2]	Mem[R3 + R2] ← R5

Control Transfer

This set of instructions adjust the value of the program counter by a relative amount determined by the location of the given label. Conditions are as follows:

- BR – Branch Always – Unconditionally branch to the stated location
- BNE – Branch if not equal – Conditionally branch if zero status flag (z) equals zero
- BE – Branch if equal – Conditionally branch if zero status flag (z) equals one
- BLT – Branch if < – Conditionally branch if negative status flag (n) equals one
- BGE – Branch if ≥ – Conditionally branch if negative status flag (n) equals zero
- BWL – Branch with link – Unconditionally branch to stated location, saving PC to link register (LR)
- RET – Return – Unconditionally jump to the value stored in the link register (LR)
- ABR – Absolute Branch – Unconditionally branch to the location held in register Ra

Stack Operations

These operations are for popping or pushing either a general purpose register or the link register onto the system stack, useful for context saving when an interrupt occurs. PUSH increments stack pointer (SP) and POP decrements stack pointer (SP) for a top-down growing stack. If the 'L' bit is set, the link register value will be used instead of the value in register Ra.