Instruction Set Summary

	Mnemonic	Syntax	Semantics	Flags	Encoding	Opcode	Cond.
1	ADD	ADD Rd, Ra, Rb	Rd ← Ra + Rb	c,v,n,z	A	00010	-
2	ADDI	ADDI Rd, Ra, #imm5	Rd ← Ra + imm5	c,v,n,z	A	00110	-
3	ADDIB	ADDIB Rd, #imm8	Rd ← Rd + imm8	c,v,n,z	B	00011	-
4	ADC	ADC Rd, Ra, Rb	Rd ← Ra + Rb + c	c,v,n,z	A	00101	-
5	ADCI	ADCI Rd, Ra, #imm5	Rd ← Ra + imm5 + c	c,v,n,z	A	00111	-
6	NEG	NEG Rd, Ra	Rd ← 0 - Ra	c,v,n,z	A	11011	-
7	SUB	SUB Rd, Ra, Rb	Rd ← Ra - Rb	c,v,n,z	A	01010	-
8	SUBI	SUBI Rd, Ra, #imm5	Rd ← Ra - imm5	c,v,n,z	A	01110	-
9	SUBIB	SUBIB Rd, #imm8	Rd ← Rd - imm8	c,v,n,z	B	01011	-
10	SUC	SUC Rd, Ra, Rb	Rd ← Ra - Rb - c	c,v,n,z	A	01101	-
11	SUCI	SUCI Rd, Ra, #imm5	Rd ← Ra - imm5 - c	c,v,n,z	A	01111	-
12	CMP	CMP Ra, Rb	Ra - Rb	c,v,n,z	A	11010	-
13	CMPI	CMPI Ra, #imm5	Ra - imm5	c,v,n,z	A	11110	-
14	AND	AND Rd, Ra, Rb	Rd ← Ra AND Rb	z	A	10000	-
15	OR	OR Rd, Ra, Rb	Rd ← Ra OR Rb	z	A	10001	-
16	XOR	XOR Rd, Ra, Rb	Rd ← Ra XOR Rb	z	A	10011	-
17	NOT	NOT Rd, Ra	Rd ← NOT Ra	z	A	10010	-
18	NAND	NAND Rd, Ra, Rb	Rd ← Ra NAND Rb	z	A	10110	-
19	NOR	NOR Rd, Ra, Rb	Rd ← Ra NOR Rb	z	A	10111	-
20	LSL	LSL Rd, Ra, #imm4	Rd ← Ra << imm4	-	A	01100	-
21	LSR	LSR Rd, Ra, #imm4	Rd ← Ra >> imm4	-	A	11100	-
22	ASR	ASR Rd, Ra, #imm4	Rd ← Ra >>> imm4	-	A	10100	-
23	LDW	LDW Rd, [Ra, #imm5]	Rd ← Mem[Ra + imm5]	-	C	00001	-
24	STW	SDW Rd, [Ra, #imm5]	Mem[Ra + imm5] ← Rd	-	C	01001	-
25	LUI	LUI Rd, #imm8	Rd[15:8] ← imm8	-	B	10101	-
26	LLI	LLI Rd, #imm8	Rd[7:0] ← imm8	-	B	11101	-
27	BR	BR LABEL	PC ← PC + imm8	-	D	-	000
28	BNE	BNE LABEL	(z==0)? PC ← PC + imm8	-	D	-	110
29	BE	BE LABEL	(z==1)? PC ← PC + imm8	-	D	-	111
30	BLT	BLT LABEL	(n&~v OR ~n&v)? PC ← PC + imm8	-	D	-	100
31	BGE	BGE LABEL	(n&v OR ~n&~v)? PC ← PC + imm8	-	D	-	101
32	BWL	BWL LABEL	LR ← PC + 1; PC ← PC + imm8	-	D	-	011
33	RET	RET	PC ← LR	-	D	-	010
34	JMP	JMP Ra, #imm5	PC ← Ra + imm5	-	D	-	001
35	PUSH	PUSH Ra PUSH LR	Mem[SP] ← Ra; SP ← SP - 1; Mem[SP] ← RL; SP ← SP - 1;	-	E	-	-
36	POP	POP Ra POP LR	SP ← SP + 1; Ra ← Mem[SP] SP ← SP + 1; RL ← Mem[SP]	-	E	-	-

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>Data Manipulation</td><td>Register</td><td colspan="5" rowspan="2">Opcode</td><td colspan="3" rowspan="2">Rd</td><td colspan="3" rowspan="2">Ra</td><td colspan="3">Rb</td><td>X</td><td>X</td></tr>
<tr><td>A2</td><td></td><td>Immediate</td><td colspan="5">imm4/5</td></tr>
<tr><td>B</td><td>Byte Immediate</td><td></td><td colspan="5">Opcode</td><td colspan="3">Rd</td><td colspan="8">imm8</td></tr>
<tr><td>C</td><td>Data Transfer</td><td></td><td>0</td><td>LS</td><td>0</td><td>0</td><td>1</td><td colspan="3">Rd</td><td colspan="3">Ra</td><td colspan="5">imm5</td></tr>
<tr><td>D1</td><td>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">1</td><td colspan="3" rowspan="2">Cond.</td><td colspan="8">imm8</td></tr>
<tr><td>D2</td><td></td><td>Jump</td><td colspan="3">Ra</td><td colspan="5">imm5</td></tr>
<tr><td>E</td><td>Stack Operations</td><td></td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>U</td><td>L</td><td>X</td><td colspan="3">Ra</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>

LS: 0 = Load Data, 1 = Store Data   U: 1 = PUSH, 0 = POP   L: 1 = Use Link, 0 = Don't use Link

Team R4 – H. Lovett, A. Reddy, A. Robinson, M. Wearn

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	13	CMPI R3, #9	R3 - 9
2	ADDI R5, R3, #9	R5 ← R3 + 9	14	AND R5, R3, R4	R5 ← R3 AND R4
4	ADC R5, R3, R4	R5 ← R3 + R4 + c	15	OR R5, R3, R4	R5 ← R3 OR R4
5	ADCI R5, R3, #9	R5 ← R3 + 9 + c	16	XOR R5, R3, R4	R5 ← R3 XOR R4
6	NEG R5	R5 ← 0 - R5	17	NOT R5, R3	R5 ← NOT R3
7	SUB R5, R3, R4	R5 ← R3 - R4	18	NAND R5, R3, R4	R5 ← R3 NAND R4
8	SUBI R5, R3, #9	R5 ← R3 - 9	19	NOR R5, R3, R4	R5 ← R3 NOR R4
10	SUC R5, R3, R4	R5 ← R3 - R4 - NOT c	20	LSL R5, R3, #3	R5 ← R3 << 3
11	SUCI R5, R3, #9	R5 ← R3 - 9 - NOT c	21	LSR R5, R3, #3	R5 ← R3 >> 3
12	CMP R3, R4	R3 - R4	22	ASR R5, R3, #3	R5 ← R3 >>> 3

The value 'c' corresponds to the carry bit flag in the ALU from the previous calculation.
CMP, CMPI are comparison instructions 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
25	LUI R5, #150	R5[15:8] ← 150
26	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.

23	LDW R5, [R3, #imm5]	R5 ← Mem[R3 + imm5]
24	STW R5, [R3, #imm5]	Mem[R3 + imm5] ← 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 != – Conditionally branch if zero status flag (z) equals zero
- BE – Branch if = – 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)
- JMP – Jump – Unconditionally jump to the location held in register Ra plus an 5-bit offset

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 pre-decrements stack pointer (SP) and POP post-increments stack pointer (SP) for a top-down growing stack. The 'U' bit indicates if a PUSH or POP operation is to be performed. If the 'L' bit is set, the link register value will be used instead of the value in register Ra.

Combined Branching & Stack Example

Below is an example showing how PUSH/POP operations and branches can be used to call a subroutine. ".sub" is a label used in assembly language to refer to a different line of code, it is converted to a relative address by an assembler. Here it is calculated as 4 + 5 = 9, if the destination address was before the calling instruction the relative value would be negative.

	PUSH R1	:Save R1
	PUSH R2	:Save R2
	BWL .sub	:Call subroutine
	POP R2	:Restore R2
	POP R1	:Restore R1
	BR .end	:Branch to end of memory
.sub	PUSH LR	:Save Link Register
	…	:Subroutine does something
	POP LR	:Restore Link Register
	JMP	:Return to where subroutine was called
.end	BR .end	