SCRAP Assembler! Reference

Benjamin Antczak, Alec Adair, Aaron Herrmann
Team 3'b101

Abstract

This document summarizes the syntax and conventions implemented by the assembler for Team 3'b101's SCRAP (Single Cycle RiSC Arithmatic Processor). The SCRAP Assembler is written in Java and requires the JRE 1.7.

I. Simple Instructions

The following table summarizes the commands that can be implemented via the SCRAP Assembler. Instructions are case-insensitive. Jumping to labels is supported; see section II for details.

Delimiters

Spaces are used to delimit instructions and arguments; new lines are used to delimit new instructions.

Argument Registers

Valid registers are R0 through R15, although by convention R15 is to be zero (or set back to zero after use).

Immediates

Immediates are 8-bit signed unless the instruction is explicitly unsigned or otherwise noted.

TABLE I
Simple Instructions

Instruction	Arguments	Format	Notes
ADD	rsrc, rdest	add [rsrc] [rdest]	[rdest] ← [rsrc] + [rdest]
ADDI	imm, rdest	addi [imm] [rdest]	[rdest] ← [imm] + [rdest]
ADDU	rsrc, rdst	addu [rsrc] [rdest]	[rdest] ← [rsrc] + [rdest]
ADDUI	imm, rdest	addui [imm] [rdest]	[rdest] ← [imm] + [rdest]
SUB	rsrc, rdst	sub [rsrc] [rdest]	[rdest] ← [rsrc] - [rdest]
SUBI	imm, rdest	subi [imm] [rdest]	[rdest] ← [imm] - [rdest]
CMP	rsrc, rdst	cmp [rsrc] [rdest]	sets flags; preserves [rdest]
CMPI	imm, rdest	cmpi [imm] [rdest]	sets flags; preserves [rdest]
AND	rsrc, rdst	and [rsrc] [rdest]	[rdest] ← [rsrc] & [rdest]
ANDI	imm, rdest	and [imm] [rdest]	[rdest] ← [imm] & [rdest]
OR	rsrc, rdst	or [rsrc] [rdest]	[rdest] ← [rsrc] | [rdest]
ORI	imm, rdest	ori [imm] [rdest]	[rdest] ← [imm] | [rdest]
XOR	rsrc, rdst	xor [rsrc] [rdest]	[rdest] ← [rsrc] ^ [rdest]
XORI	imm, rdest	xori [imm] [rdest]	[rdest] ← [imm] ^ [rdest]
MOV	rsrc, rdst	mov [rsrc] [rdest]	[rdest] ← [rsrc]
MOVI	imm, rdest	movi [imm] [rdest]	[rdest] ← [imm]
LSH	ramount, rdest	lsh [imm] [rdest]	[rdest] ← [rdest] << ±1
ASH	ramount, rdest	ash [imm] [rdest]	[rdest] ← [rdest] << ±1 (sign extended)
LOAD	rdest, raddr	load [rdest] [raddr]	[rdest] ← mem[raddr]
STOR	rsrc, raddr	stor [rsrc] [raddr]	mem[raddr] ← [rsrc]

II. Conditional Jump

Jump uses the value set in the ALU's flag registers to determine whether to jump to the labeled address or fall through. Labels begin with periods, and addresses point to instructions immediately following the line defining the label (*e.g.*, ".loopHere" on line eight will point to line nine). Labels are referred to in conditional jumps *without* a period. Jumping also requires two registers: the first to hold the memory address of the label's pointer; the second to hold the label's address. These can be set by using the "Memory Config" button in the assembler. **NOTE: As of this writing, the PC start address changes based on the number of labels a program uses, since the jump table begins at address 0x0.**

TABLE II
JUMP CONDITIONAL USAGE

Instruction	Operation
jcond [condition] [raddr]	*if ([condition])*$\{PC \leftarrow$ *[raddr]*$\}$ *else*$\{PC \leftarrow PC + 1\}$
jcond eq r1	if (eq) $\{$PC $\leftarrow$ [r1]$\}$ else$\{$PC $\leftarrow$ PC + 1$\}$

TABLE III
JUMP CONDITIONAL WITH LABEL USAGE

Instruction	Operation	Resulting Instruction Sequence*
jcond [label] [rsrc1] [rsrc2] [condition]	*if ([rsrc1] [condition] [rsrc2])* $\{PC \leftarrow$ *[label]*$\}$ *else* $\{PC \leftarrow PC + 1\}$	*movi [label pointer] [rx]* *load [ry] [rx]* *comp rsrc1 rsrc2* *jcond [condition] [rx]*
jcond loopHere r1 r2 eq	if (r1 == r2) $\{$PC $\leftarrow$ [.loopHere + 1]$\}$ else $\{$PC $\leftarrow$ PC + 1$\}$	movi [loopHere pointer] [rx] load [ry] [loopHere address] comp [r1] [r2] jcond eq [rx]

*Choose registers [rx] and [ry] via the "Memory Config" button; they default to R13 and R14, respectively.

TABLE IV
CONDITION CODES

Condition	Condition Code *[condition]*	Flag Configuration
Equal	EQ	Z = 1
Not Equal	NE	Z = 0
Greater than or Equal	GE	N = 1 or Z = 1
Carry Set	CS	C = 1
Carry Clear	CC	C = 0
Higher than	HI	L = 1
Lower than or the Same as	LS	L = 0
Lower than	LO	L = 0 and Z = 0
Higher than or the Same as	HS	L = 1 or Z = 1
Greater than	GT	N = 1
Less than or Equal	LE	N = 0
Flag Set	FS	F = 1
Flag Clear	FC	F = 0
Less than	LT	N = 0 and Z = 0

III. Binary Decomposition

TABLE V
INSTRUCTION FORMATS

Instruction	Arguments	Binary
ADD	rsrc, rdest	0000 rdest 0101 rsrc
ADDI	imm, rdest	0101 rdest imm
ADDU	rsrc, rdst	0000 rdest 0110 rsrc
ADDUI	imm, rdest	0110 rdest imm
SUB	rsrc, rdst	0000 rdest 1001 rsrc
SUBI	imm, rdest	1001 rdest imm
CMP	rsrc, rdst	0000 rdest 1011 rsrc
CMPI	imm, rdest	1011 rdest imm
AND	rsrc, rdst	0000 rdest 0001 rsrc
ANDI	imm, rdest	0001 rdest imm
OR	rsrc, rdst	0000 rdest 0010 rsrc
ORI	imm, rdest	0010 rdest imm
XOR	rsrc, rdst	0000 rdest 0011 rsrc
XORI	imm, rdest	0011 rdest imm
MOV	rsrc, rdst	0000 rdest 1101 rsrc
MOVI	imm, rdest	1101 rdest imm
LSH	ramount, rdest	1000 rdest 0100 ramount
ASH	ramount, rdest	1000 rdest 0110 ramount
LOAD	rdest, raddr	0100 rdest 0000 raddr
STOR	rsrc, raddr	0100 rsrc 0100 raddr
BCOND	cond, imm	1100 cond imm
JCOND	cond, rtarget	0100 cond 0000 rtarget

TABLE VI
CONDITION CODES

Condition	Assembly Code	Flags (Input)	Binary
Equal	EQ	Z = 1	0000
Not Equal	NE	Z = 0	0001
Greater than or Equal	GE	N = 1 or Z = 1	1101
Carry Set	CS	C = 1	0010
Carry Clear	CC	C = 0	0011
Higher than	HI	L = 1	0100
Lower than or the Same as	LS	L = 0	0101
Lower that	LO	L = 0 and Z = 0	1010
Higher than or the Same as	HS	L = 1 or Z = 1	1011
Greater than	GT	N = 1	0110
Less than or Equal	LE	N = 0	0111
Flag Set	FS	F = 1	1000
Flag Clear	FC	F = 0	1001
Less than	LT	N = 0 and Z = 0	1100