1

TR-94032

RWC-1 Assembly Language Manual
Release 0.2

Massively Parallel Software Laboratory

Received February 1995

Tsukuba Research Center, Real World Computing Partnership

Tsukuba Mitsui Building 16F, 1-6-1 Takezono
Tsukuba-shi, Ibaraki 305, JAPAN

*This document was prepared by Tsutomu Horikawa, Kanrikogaku Kenkyusho Ltd., as the part of the contract with Massively Parallel Software Laboratory

Chapter 1

Assembly Language Syntax

In the description of the assembly language syntax in this chapter, brackets ’[’ ... ’]’ indicate optional items. Braces ’{’ ... ’}’ indicate alternate item choices, the individual items being separated by vertical bars ’|’. A star ’*’ indicates that the previous item is to be repeated or more times.

1.1 Syntax Rule

<syntax>	[<statement>] [; [<statement>]*] [! *comment*]
	[! *comment*]
<statement>	[<label> :] [<instruction>]
	[<assign>]
<instruction>	*opcode* [<operand>] [, <operand>]*
<assign>	*symbol* = *constant_expression*
<operand>	{ *register* | *constant_expression* | *address* }

1.2 Lexical Features

Comments

An exclamation mark ’!’ indicates that all subsequent characters, up to the end of the line, are a comment.

Numbers

Decimal, hexadecimal, octal and floating-point constants are supported, and are written as C. Floating-point constants can be coded only in floating-point pseudo operations.

Character

A character is quoted with a single-quote ('), and indicates the ASCII value of a character. A back-slash (\) indicates an escape character.

String

A string is a sequence of ASCII characters, enclosed in double-quotes ("). A back-slash (\) is indicates an escape character.

Escape codes

`\a`	bell (0x07)
`\b`	backspace (0x08)
`\t`	horizontal tab (0x09)
`\n`	newline (0x0a)
`\v`	vertical tab (0x0b)
`\f`	form-feed (0x0c)
`\r`	carriage return (0x0d)
`\\`	backslash (0x5c)
`\`*nnn*	octal value *nnn*

Symbol Names

The syntax of a symbol name is:

{ *letter* | _ | $ | . } { *letter* | _ | $ | . | *digit*}*

Upper-case and lower-case letters are distinct.

A period '.' indicates the current address.

Special Symbols

Special symbol names begin with dollar sign '$', and is distinguished from user symbols.

`$r0...$r31`	general purpose registers
`$aux`	auxiliary accumlator (`$r28`)
`$link`	link register (`$r29`)
`$opadr`	operand address regster (`$r30`)
`$zero`	zero register (`$r31`)

Labels

A label is either a symbol or a single decimal digit. Note that a label is immediately followed by a colon ':'.

Numeric labels can be referenced only in branch operations.

`bra`	*n*`b`	! references the first numeric label *n* prior to the current point
`bra`	*n*`f`	! references the first numeric label *n* subsequent to the current point

Constant Expressions

The following operators are recognized in constant expressions, and have the same precedence as in C.

Binary	
`+`	integer add
`-`	integer subtract
`*`	integer multiply
`/`	integer divide
`%`	modulo
`^`	exclusive or
`<<`	left shift
`>>`	right shift
`&`	bit-wise and
`|`	bit-wise or

Unary	
`+`	no effect
`-`	2's complement
`~`	1's complement

1.3 Addressing

Expression	Comment
constant_expression(register)	register indirect + offset
relocation_symbol+constant_expression	symbol value + offset

Chapter 2

Pseudo Operation

Mnemonic	Description
`.seg` "*string*"	Changes the current segment to *string*. `text`,`data`, `data1` and `bss` are supported.
`.align` *boundary*	Aligns the location counter on a 0-mod-*boundary* boundary.
`.talign`	Advances the location counter to an eight-byte boundary, if not currentry on a eight-byte boundary. Intervening bytes are filled with NOP.
`.skip` *expression*	Increments the location counter by *expression*.
`.global` *symbol*	Assigns global storage class to *symbol*.
`.comm` *symbol*, *expression*	Declares *symbol* and *expression* of a common area.
`.ascii` " *string* "	Generates a sequence of ASCII characters.
`.asciz` " *string* "	Generates a sequence of ASCII characters, terminated by zero.
`.byte` *expression*	Generates an initialized byte by *expression* in the current segment.
`.quart` *expression*	Generates an initialized quartword by *expression* in the current segment.
`.half` *expression*	Generates an initialized halfword by *expression* in the current segment.
`.word` *expression*	Generates an initialized word by *expression* in the current segment.
`.float` *float_value*	Generates an initialized floating-opint value by *float_value* in the current segment.
symbol = *expression*	Assigns the value of *expression* to *symbol*.

Chapter 3

Instruction Set

This chapter describes the instruction set of RWC-1. For futher details of the instruction, see the instruction set manual.

The following table describes the operands.

rd	destination register
rs1,rs2	source register
xim	extended immediate (64 bit)
imm	immediate (46 bit)
him	high-order 18 bits of 64 bit immediate
lim	low-order 46 bits of 64 bit immediate
bra	branch address (15 bit)
jpa	jump address (23 bit)
bcc	branch condition code (eq0,ne0,gt0,lt0,ge0,le0)
lvia	local virtual instruction address (22bit)
lvoa	local virtual operand address offset (10bit)
bn	'n'bit data
rsn	register set number
rn	register number

3.1 Load/Store

Instruction	Mnemonic	Operands
Load byte	`ld.b`	`rd,imm(rs1)`
Load quarter word	`ld.q`	`rd,imm(rs1)`
Load half word	`ld.h`	`rd,imm(rs1)`
Load word	`ld`	`rd,imm(rs1)`
Load unsigned byte	`ld.bu`	`rd,imm(rs1)`
Load unsigned quarter word	`ld.qu`	`rd,imm(rs1)`
Load unsigned half word	`ld.hu`	`rd,imm(rs1)`
Store byte	`st.b`	`imm(rs1),rd`
Store quater word	`st.q`	`imm(rs1),rd`
Store half word	`st.h`	`imm(rs1),rd`
Store word	`st`	`imm(rs1),rd`

3.2 Arithmetic

Instruction	Mnemonic	Operands
Add integer	add	rd,rs1,rs2
		rd,rs1,imm
Add integer with carry in	add.ci	rd,rs1,rs2
Add integer with carry out	add.co	rd,rs1,rs2
Add integer with carry in-out	add.cio	rd,rs1,rs2
Add unsigned integer	add.u	rd,rs1,rs2
Add unsigned integer with carry in	add.ciu	rd,rs1,rs2
Add unsigned integer with carry out	add.cou	rd,rs1,rs2
Add unsigned integer with carry in out	add.ciou	rd,rs1,rs2
Subtract integer	sub	rd,rs1,rs2
		rd,rs1,imm
Subtract integer with carry in	sub.ci	rd,rs1,rs2
Subtract integer with carry out	sub.co	rd,rs1,rs2
Subtract integer with carry in-out	sub.cio	rd,rs1,rs2
Subtract unsigned integer	sub.u	rd,rs1,rs2
Subtract unsigned integer with carry in	sub.ciu	rd,rs1,rs2
Subtract unsigned integer with carry out	sub.cou	rd,rs1,rs2
Subtract unsigned integer with carry in out	sub.ciou	rd,rs1,rs2
Multiply signed	mul	rd,rs1,rs2
Multiply unsigned	mul.u	rd,rs1,rs2
Multiply lower	mul.l	rd,rs1,rs2
Multiply higher	mul.h	rd,rs1,rs2
Multiply higher unsigned	mul.hu	rd,rs1,rs2
Initial stop for divide	divi	rd,rs1,rs2
Divide 8bit	div8	rd,rs1,rs2
Swap quotient and dividend	divs	rd,rs1,rs2
Nomaization of remainder for divide	divr	rd,rs1,rs2
Nomaization of quotient for divide	divq	rd,rs1,rs2
Initial stop for divide unsigned	divi.u	rd,rs1,rs2
Step for divied unsigned	div8.u	rd,rs1,rs2
Compare integer	cmp	rd,rs1,rs2
		rd,rs1,imm
Set immediate to high	seth	rd,him

3.3 Logical operation

Instruction	Mnemonic	Operands
Logical and	`and`	`rd,rs1,rs2`
		`rd,rs1,imm`
Logical or	`or`	`rd,rs1,rs2`
		`rd,rs1,imm`
Logical exclusive or	`xor`	`rd,rs1,rs2`
		`rd,rs1,imm`
Shift left logical	`sll`	`rd,rs1,rs2`
		`rd,rs1,eim`
Shift right arithmetic	`sra`	`rd,rs1,rs2`
		`rd,rs1,eim`
Shift right logical	`srl`	`rd,rs1,rs2`
		`rd,rs1,eim`

3.4 Bit field operation

Instruction	Mnemonic	Operands
Extract	`ext`	`rd,rs1,rs2`
		`rd,rs1,b6,b6`
Insert	`ins`	`rd,rs1,rs2,b1,b6,b3`
Find first bit set	`ff1`	`rd,rs1`

3.5 Floting Point

Instruction	Mnemonic	Operands
Floating point add	`fadd`	`rd,rs1,rs2`
Floating point subtract	`fsub`	`rd,rs1,rs2`
Floating point multiply	`fmul`	`rd,rs1,rs2`

3.6 Flow Contorol

Instruction	Mnemonic	Operands
Conditional branch	bcnd	bcc,rs1,bra
Conditional branch with hint	bcnd.h	bcc,rs1,bra
Branch on bit set	bbs	b4,rs1,bra
Branch on bit set with hint	bbs.h	b4,rs1,bra
Unconditional branch	bra	bra
Unconditional branch and link register	bra.l	bra
Unconditional jump	jmp	jpa
Unconditional jump and link register	jmp.l	jpa
Jump register	jmpr	rs1
Jump register and link register	jmpr.l	rs1
Break current thread	break	b14
Halt	halt	code
Unconditional software trap	trap	code
Return from exception	rte	

3.7 Packet generation

Instruction	Mnemonic	Operands
Make packet	mkpkt	rs1,rs2,b3,b1,b1,b1,b8
Make packet with idex	mkpkt	rs1,rs2,lvia,lvoa,b3,b1,b1,b1,b8
Make continuation	mkcnt	rd,rs1,lvia,lvoa,b3,b1,b1,b1,b1
Set instruction address onto continuation	sia	rd,rs1,rs2,b1,b1

3.8 Input/Output

Instruction	Mnemonic	Operands
Begin io transfer	bgnio	
Stop io transfer	stpio	
Make io massage	mkmsg	

3.9 Others

Instruction	Mnemonic	Operands
Purge memory access pipeline	prgm	
Purge make packet genaration pipline	prgp	
Move GPR to register	mgr	[rsn,rn],rs1
Move register to GPR	mrg	rd,[rsn,rn]
Swap GPR and register	sgr	rd,[rsn,rn],rs2
Write back cache line	cwb	[rsn,rn]

3.10 Builtin Macro

macro		generate	code
nop		xor	$r31,$r31,$r31
mov	rd,rs1	or	rd,$r31,rs1
seti	rd,xim	seth	rd,him
		or	rd,rd,lim

Chapter 4

How to Use the Assembler

4.1 Invoke

`ras` is invoked:

> `ras` [*option*] *inputfile*

All undefined symbols in the assembly are treated as being global.

4.2 Options

`-o` *objfile* Names the object file *objfile*. If the `-o` flag is not set, the object file is named `a.out`.

Contents