I **Volume**

1

UNIVERSITY OF TEHRAN

SAYAC

Simple **A**rchitecture **Y**et **A**mple **C**ircuitry

UNIVERSITY OF TEHRAN

SAYAC Reference Manual

Developers:
Professor Zain Navabi[1], Katayoon Basharkhah[2], Hanieh Tootoonchi Asl[3]
[1] Worcester Polytechnic Institute, [2]University of Tehran, [3]University of Tehran
navabi@wpi.edu, basharkhah.kt96@ut.ac.ir, h.totonchi@ut.ac.ir

November, 2021

Table of Contents

1. Chapter

1

An Overview

The SAYAC processor is a **S**imple **A**rchitecture **Y**et **A**mple **C**ircuitry. This is a RISCV-like open-source academic processor that was originally designed to support educational processor hardware architecture and implementation. However, the simple and ample architecture of this processor provides increased value enabling the researchers to devise new research directions in design, test, reliability, high-level modeling, and security. SAYAC is a 16-bit processor with a 16-bit address bus and a 16-bit bi-directional data bus. The processor has standard memory accessing handshaking signals, and separate IO read and write lines. The latter arrangement simplifies IO processing in small applications that do not require a complicated bussing structure for separation of memory from IO. Despite the small range of instruction width, all necessary instructions for executing different applications are covered. This is done by using the concept of shadow instructions.

Features:

- RISC architecture.
- Minimum instruction execution time: One instruction execution per one clock cycle
- General and Special purpose registers: 16 16-bit registers in a register file
- Number and Types of instructions: 28 instructions, 8 shadow instructions

To show the impact of using this architecture in academic and research, some of the works extended around this processor are mentioned. We have provided a model for the processor instruction set simulator. This is a SystemC model that can be used in a SystemC virtual platform for the purpose of hardware-software evaluation of a complete system. This SystemC-based model of ISS is equipped with debugging features that can help both software programmers and hardware designer to test the complete system. Such a platform can be used for fault injection and analysis in future.

We developed a virtual tester for SAYAC that is equipped with JTAG hardware. The virtual tester was used for external testing of SAYAC busses connecting to an accelerator bus structure. The virtual tester showed how serial data could be used for loading JTAG instruction, application of serial test data for EXTEST instruction and shifting response out for examination.

Registers

As shown in FIGURE 1.1, SAYAC has a register-file of sixteen 16-bit registers. This register-file is surrounded by arithmetic and logical units for add, subtract, logical, multiply, and divide operations. For memory access instructions, registers of this structure serve as source and destination. For one or two operand instructions, e.g., ADD or NOT, these registers serve as the operands and result destinations.
Address of addressed instructions or indirect addresses are also taken from the register-file. SAYAC register-file is a multi-port structure with two 4-bit read addresses and their 16-bit output ports. The two reads can be done simultaneously. In addition, the register-file has a 16-bit write port and its corresponding 4-bit write address port. Two reads and a write can be active simultaneously. However, the write operation will only take place on the active edge of the clock during which write is enabled.

1) *Special registers*

R15:

All 16 registers of the register-file are addressed through the three address ports of the register-file. Register 0 is always zero that cannot be written into. Register 15 consists of eight shadow bits, four flags and four status bits. FIGURE 1.2 shows the arrangement of the above mentioned bits in location 15 of the register-file. Other registers in the SAYAC architecture are for internal purposes and are not directly user accessible.

.

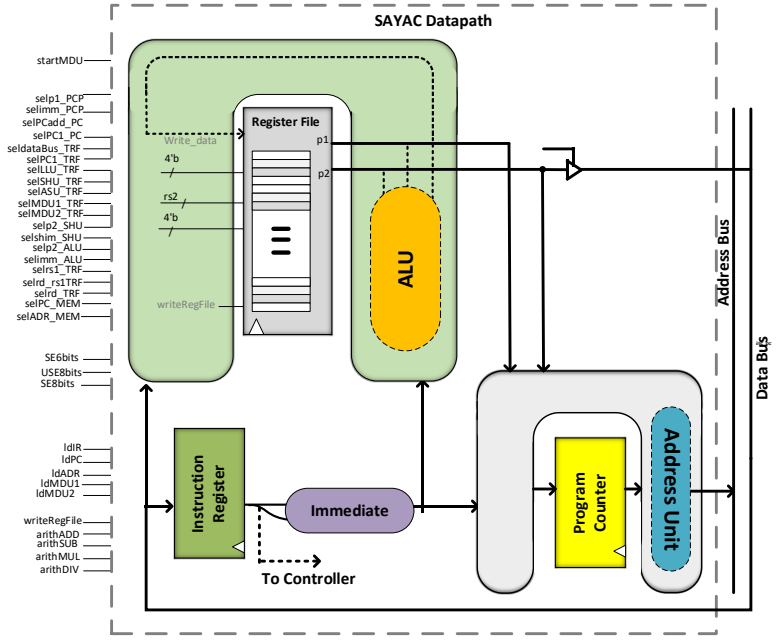


FIGURE 1.1 SAYAC Architecture

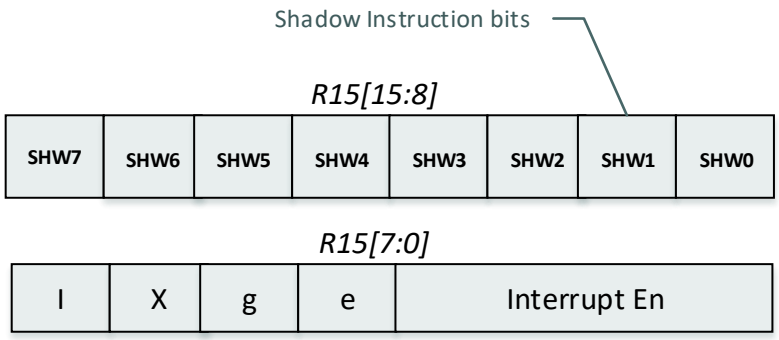


FIGURE 1.2 Special register R15

Addressing

In this section different addressing modes of SAYAC processor will be explained. FIGURE 1.3 shows these modes. In the SAYAC processor, operands will be stored in the register-file, therefore many of operands are accessed by register addressing mode. In this mode, the instruction has the address of the register of register-file where the operand is stored. As an example, instruction "*ANR* rs_2 rs_1 *rd*" performs AND operation on two operands in locations rs_2 and rs_1 and stores the result in location *rd*. Another mode is register indirect addressing. In this mode, register in the register-file is a pointer to an operand in the memory. In the case of PC relative addressing, two kinds of immediate and register relative addressing will be used as shown in FIGURE 1.4.

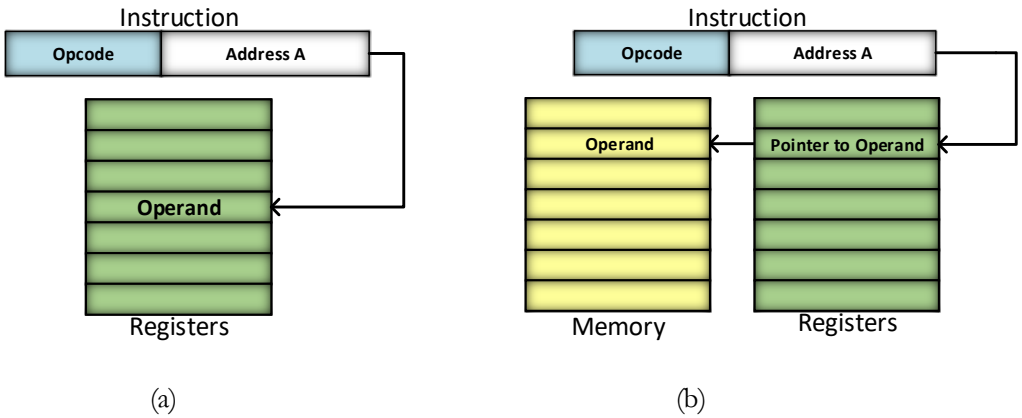


FIGURE 1.3 Addressing modes: (a) Register addressing (b) Register indirect addressing

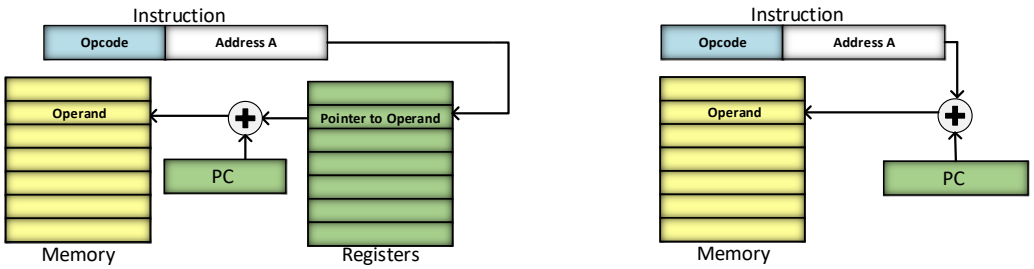


FIGURE 1.4 PC relative addressing modes

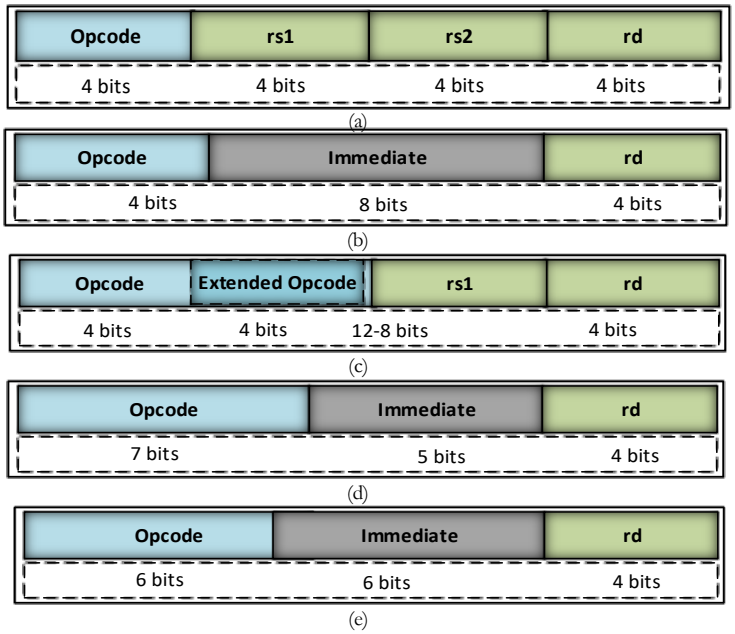


FIGURE 1.5 SAYAC instruction formats

Instruction Format

SAYAC instructions operate on registers of the register-file. As mentioned, the register-file has a destination address (*rd*) and two source addresses (*rs1* and *rs2*).

In general, instructions have a destination register and one of two sources. Destination and source registers refer to the locations of the register-file. For instructions that we refer to as immediate, one of their sources is taken from certain bits of the Instruction Register (IR).

FIGURE 1.5 shows various formats used in SAYAC instruction set. Format shown in FIGURE 1.5 (a) uses two 4-bit source register addresses and has a 4-bit destination

address. Logical and arithmetic instructions use this format. As an example, the add instruction (ADR) performs an add on the contents of the register in the register-file structure addressed by *rs1* with contents of another register in this structure -addressed by *rs2* and puts the result in register-file register addressed by *rd*. The shorthand notation for this instruction is shown below:

.rd. <= .rs1. + .rs2.

The dots shown are notations indicating register-file contents addressed by the variable enclosed in the pair of dots.

A variation of two operand instructions uses the format shown in FIGURE 1.5 (b). Here, *rd* is used for an operand and the destination. The other operand of the instruction is the 8-bit immediate that is sign extended (SE) to fill 16 bits. The shorthand notation of ADI (Add Immediate) is shown below:

Instruction Set

Instruction Set Summary

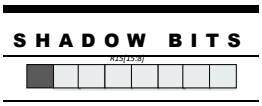
[15:12]	[11]	[10]	[9]	[8]	[7]	[6]	[5]	[4]	[3:0]	Instruction	Notation
0000	Reserved										
0001	Reserved										
0010	00		0		rs1				.rd.	LDR	.rd. <= [.rs1.]
			1								.rd. <= { .rs1. }
	01		0		rs1				.rd.	STR	[.rd.] <= .rs1.
			1								{ .rd. } <= .rs1.
	10		s		rs1				.rd.	JMR	PC <= PC + .rs1. .rd. <= PC + 1 if s = 1
	11		“imm”						.rd.	JMI	PC <= PC + “imm” .rd. <= PC + 1
0011	rs2				rs1				.rd.	ANR	.rd. <= .rs1. AND .rs2.
0100	“imm”								.rd.	ANI	.rd. <= .rd. AND USE“imm”
0101	“imm”								.rd.	MSI	.rd. <= SE“imm”
0110	“imm”								.rd.	MHL	.rd. ‘MSB <= “imm”
0111	rs2				rs1				.rd.	SIR	.rd. <= .rs1. LS± .rs2.
1000	rs2				rs1				.rd.	SAR	.rd. <= .rs1. AS± .rs2.
1001	rs2				rs1				.rd.	ADR	.rd. <= .rs1. + .rs2.
1010	rs2				rs1				.rd.	SUR	.rd. <= .rs1. – .rs2.
1011	“imm”								.rd.	ADI	.rd. <= .rd. + SE“imm”
1100	“imm”								.rd.	SUI	.rd. <= .rd. - SE“imm”
1101	rs2				rs1				.rd.	MUL	.rd. <= .rs1. × .rs2. ‘LSB .rd+1. <= .rs1. × .rs2. ‘MSB
1110	rs2				rs1				.rd.	DIV	.rd. <= .rs1. ÷ .rs2. ‘Quo .rd+1. <= .rs1. ÷ .rs2. ‘Rem
1111	00		0		rs1				.rd.	CMR	flags <= Cmp(.rs1. , .rd.)
			1	“imm”					.rd.	CMI	flags <= Cmp(.rd. , SE“imm”)
	01		0	flag interpretation bits					.rd.	BRC	PC <= .rd. if flag
			1	flag interpretation bits					.rd.	BRR	PC <= PC + .rd. if flag
	10		0	“shim”					.rd.	SHI	.rd. <= .rd. LS± “shim”
			1	“shim”					.rd.		.rd. <= .rd. AS± “shim”
	11		0	0	rs1				.rd.	NTR	.rd. <= 1sComp(.rs1.)
				1							.rd. <= 2sComp(.rs1.)
			1	0					.rd.	NTD	.rd. <= 1sComp(.rd.)
				1							.rd. <= 2sComp(.rd.)

Nomenclature:

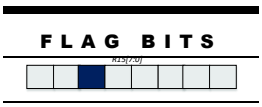
.rsd. → R_f content pointed by *rsd*
[*adr*] → Memory addressed by *adr*
{ *loc* } → IO device addressed by *loc*
“imm” → Immediate operand
SE”imm, USE”imm” → Signed, Unsigned Extension of “imm”
‘LSB, `MSB → Least, Most Significant Byte of multiplication
‘Quo, `Rem → Quotient, Remainder of division
flags → Greater and Less than flags
“shim” → Sign-and-magnitude 5-bit immediate shift amount
AS±, LS± → Arithmetic, Logical Shift by + or – shift amount
Cmp(,) → Compare
1sComp( ), 2sComp( ) → One’s, Two’s complement

Explanation of Instructions

This chapter lists the instructions in SAYAC processor. Different instruction types are explained.



Shadow Instruction: Some instructions have one or more shadows enabling more variations for performing an instruction. Through this chapter, shadow instructions are specified in this format. The figure on the left shows the configuration for R15 [15:8] that includes shadow flags for instructions. As an example, here R15[15] is the shadow bit for the corresponding shadow instruction.



Flag Bits: Figure on the left shows R15[7:0]. If an instruction influences control flags, the corresponding flag is highlighted in the figure. R15[7:4] includes these control flags.

SAYAC instructions can be classified in three types: Data transfer instructions, control flow instructions and arithmetic/logical operations. Instruction formats can be registered, immediate or a combination of register and immediate instructions. Shadow instructions usually include register-immediate operations. Details of instruction types are explained below.

Data transfer instructions

There are two data transfer instructions in SAYAC. Load and store instructions are used to transfer data from/to memory or I/O devices.

LDR/LDRio

Opcode				Opcode-Extended		Memory /IO		r_{s1}				r_d			
0	0	1	0	0	0	0/1									

LdR / LdRio	Load Registered
Instruction	load from memory or I/O peripheral
Operation	$r_d \leftarrow (r_{s1})$
Assembler Syntax	LdR r_d r_{s1}
Example	LdR r3 r2
Description	Loads register r_d with the desired memory byte at the address specified with r_{s1}. If Memory/IO bit is one, then it bypasses the memory transfer
Instruction Type	R
Instruction Fields	r_{s1}= Index of source register r_d= Index of destination register Memory/IO= selection bit for memory or I/O peripheral

STR/STRio

Opcode				Opcode-Extended		Memory /IO		r_{s1}				r_d			
0	0	1	0	0	1	0/1									

STR / STRio	Store Registered
Instruction	Store to memory or I/O peripheral
Operation	$(r_d) \leftarrow r_{s1}$
Assembler Syntax	STR r_d r_{s1}
Example	STR r3 r2
Description	Stores register r_{s1} to the memory location specified with r_d. If Memory/IO bit is one, then it bypasses the memory transfer
Instruction Type	R
Instruction Fields	r_{s1}= Index of source register r_d= Index of destination register Memory/IO= selection bit for memory or I/O peripheral

Control Flow Instructions

Control flow instructions include jump and branch instructions. In this type of instructions, program counter, PC, is set based on the PC relative addressing mode described in chapter one.

JMR

Opcode				Opcode-Extended		Save PC		r_{s1}				r_d			
0	0	1	0	1	0	s									

JMR	Jump Registered (Unconditional)
Instruction	Jump to address
Operation	$PC \leftarrow PC + r_{s1}$ $r_d \leftarrow PC+1$
Assembler Syntax	JMR r_d r_{s1}
Example	JMR r3 r2
Description	Transfers execution to the address contained in register r_{s1} relative to the current instruction pointer. Saves the address of the next instruction in register r_d if the Save PC bit (s) equals to 1. This option is used when returning from interrupts and exceptions.
Instruction Type	R
Instruction Fields	r_{s1}= Index of source register r_d= Index of destination register s= Option for saving PC contents

JMI

Opcode				Opcode-Extended		Imm [5:0]						r_d			
0	0	1	0	1	1										

JMI	Jump Immediate (Unconditional)
Instruction	Jump to immediate address
Operation	$PC \leftarrow PC + Imm$ $r_d \leftarrow PC+1$
Assembler Syntax	JMI r_d Imm
Example	JMI r3 32

Description	Transfers execution to the address contained in immediate value relative to the current instruction pointer and saves the address of the next instruction in register r_d.
Instruction Type	I
Instruction Fields	r_d= Index of destination register Imm = 6-bit signed immediate value

For branch instructions, PC is determined based on satisfying a condition. The condition is presented in 5 bits locating in Instruction [8:4] named as **F**lag **I**nterpretation **B**its, FIB[4:0]. Table 1 shows different conditions and FIB configuration.

	FIB[4:0]				
eq	X	X	0	0	0
lt	X	X	0	0	1
gt	X	X	0	1	0
gt/eq	X	X	0	1	1
lt/eq	X	X	1	0	0
neq	X	X	1	0	1

BRC

Opcode				Opcode-Extended			FIB					r_d			
1	1	1	1	0	1	0									

BRC	**Branch Conditional**
Instruction	Branch Registered with Condition
Operation	If (FIB) then PC← r_d
Assembler Syntax	BRC FIB r_d
Example	BRC 0x01 r_2
Description	If Flag Interpretation Bits (FIB) are true, then transfers program control to the instruction at the address specified by register r_d
Instruction Type	R
Instruction Fields	r_d= Index of destination register FIB = 5-bit compare flag interpretation bits

BRR

Opcode				Opcode-Extended			FIB					r_d			
1	1	1	1	0	1	1									

BRR	**Branch Conditional Relative**
Instruction	Branch Registered Relative with Condition
Operation	If (FIB) then PC←PC + r_d
Assembler Syntax	BRR FIB r_d
Example	BRR 0x01 r_2
Description	If Flag Interpretation Bits (FIB) are true, then transfers program control to the instruction to the address contained in register r_d relative to the current instruction pointer.
Instruction Type	R
Instruction Fields	r_d= Index of destination register FIB = 5-bit compare flag interpretation bits

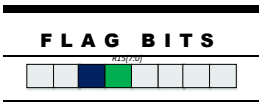
CMR

Opcode				Opcode-Extended				r_{s1}				r_d			
1	1	1	1	0	0	0									

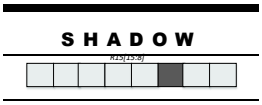
CMR	Compare Registered
Instruction	Comparing two registers

Operation	If ($r_{s1} > r_d$) then G←1
	If ($r_{s1} < r_d$) then L←1
	If ($r_{s1} = r_d$) then E←1

Assembler Syntax	CMR r_{s1} r_d
Example	CMR r_3 r_2
Description	Compares r_{s1} and r_{s2} and stores the comparison result in the corresponding flags. This is an unsigned comparison.
Instruction Type	R
Instruction Fields	r_{s1}= Index of first register r_d= Index of second register



Flag Bits: Based on the compare result, two bits of R15[5] or R[15][4] are set to 1. If it is greater, then the G flag, R15[5], will be set to 1 and if it equals E flag, R15[4] will be set to 1. Otherwise, it is considered less than.



CMR Shadow Instruction: Shadow bit for CMRS instruction is R15[10]. If this bit is 1, then the shadow instruction of CMR will be executed as below.

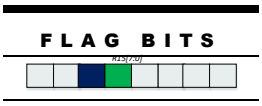
CMRS

Opcode				Opcode-Extended				r_{s2}				r_{s1}			
1	1	1	1	0	0	0									

CMRS	Compare Registered Signed
Instruction	Comparing two registers

Operation	If ($r_{s1} > r_d$) then G←1
	If ($r_{s1} < r_d$) then L←1
	If ($r_{s1} = r_d$) then E←1

Assembler Syntax	CMRS r_{s1} r_d
Example	CMRS r_3 r_2
Description	Compares r_{s1} and r_{s2} and stores the comparison result in the corresponding flags. This shadow instruction performs signed comparison.
Instruction Type	R
Instruction Fields	r_{s1}= Index of first register r_d= Index of second register



Flag Bits: Based on the compare result, two bits of R15[5] or R[15][4] are set to 1. If it is greater, then the G flag, R15[5], will be set to 1 and if it equals E flag, R15[4] will be set to 1. Otherwise, it is considered less than.

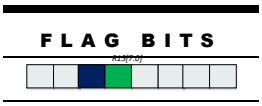
CMI

Opcode				Opcode-Extended			Imm[4:0]					r_d			
1	1	1	1	0	0	1									

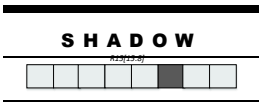
CMI	Compare Immediate
Instruction	Comparing register and immediate

Operation	If (SE(Imm) > r_d) then G←1
	If (SE(Imm) < r_d) then L←1
	If (SE(Imm) = r_d) then E←1

Assembler Syntax	CMI Imm r_d
Example	CMI 30 r_2
Description	Sign-extends the 5-bit immediate value to 16 bits and compares it to the value of r_d and stores the comparison result in the corresponding flags.
Instruction Type	I
Instruction Fields	r_d= Index of first register Imm = 5-bit unsigned immediate value



Flag Bits: Based on the compare result, two bits of R15[5] or R[15][4] are set to 1. If it is greater, then the G flag, R15[5], will be set to 1 and if it equals E flag, R15[4] will be set to 1. Otherwise, it is considered less than.



CMI Shadow Instruction

Shadow bit for CMIS instruction is R15[10]. If this bit is 1, then the shadow instruction of CMI will be executed as below.

CMIS

Opcode				Opcode-Extended			Imm[4:0]					r_d			
1	1	1	1	0	0	1									

CMIS	Compare Immediate Signed
Instruction	Comparing register and immediate

Operation	If (SE(Imm) > r_d) then G←1
	If (SE(Imm) < r_d) then L←1
	If (SE(Imm) = r_d) then E←1

Assembler Syntax	CMI Imm r_d
Example	CMI 30 r_2
Description	Sign-extends the 5-bit immediate value to 16 bits and compares it to the value of r_d and stores the comparison result in the corresponding flags. This instruction performs signed comparison.
Instruction Type	I
Instruction Fields	r_d= Index of first register Imm = 5-bit unsigned immediate value

Arithmetic / Logical Instructions

The last type of instruction is arithmetic/logical instructions. Registered instructions perform an operation on two source registers and stores the result in a destination register. Immediate instructions perform the operation on an immediate value and destination register and stores the result in the destination register. Shadow arithmetic/logical instructions enable performing operation on a source register and immediate value and storing the result in the destination register. Signed and unsigned version of multiplication and division is also supported in this processor.

ANR

Opcode				r_{s1}				r_{s2}				r_d			
0	0	1	1												

ANR	AND Registered
Instruction	Logical AND operation
Operation	$r_d \leftarrow r_{s1}$ AND r_{s2}
Assembler Syntax	ANR r_d r_{s1} r_{s2}
Example	ANR r5 r3 r2
Description	Calculates the bitwise logical AND of r_{s1} and rs2 and stores the result in r_d .
Instruction Type	R
Instruction Fields	r_{s1}= Index of source1 register r_{s2}= Index of source2 register r_d= Index of destination register

ANI

Opcode				Imm[7:0]								r_d			
0	1	0	0												

ANI	AND Immediate
Instruction	Logical AND operation
Operation	$r_d \leftarrow r_d$ AND ((0x00) & Imm)
Assembler Syntax	ANI r_d Imm
Example	ANI r5 250
Description	Calculates the bitwise logical AND of r_d and 16-bit concatenated Immediate value, ((0x00) & Imm), and stores the result in r_d.
Instruction Type	R
Instruction Fields	r_d= Index of destination register Imm = 8-bit unsigned immediate value

SHADOW

R15[15:8]

ANI Shadow Instruction

Shadow bit for ANI instruction is R15[15]. If this bit is 1, then the shadow instruction of ANI will be executed

ANIR

Opcode				Imm[3:0]				rs1				rd			
0	1	0	0												

ANI-SHADOW	AND Immediate-Register
Instruction	Logical AND operation
Operation	$r_d \leftarrow r_{s1}$ AND ((0x00) & Imm)
Assembler Syntax	ANIR r_d r_{s1} Imm
Example	ANIR r5 r1 2
Description	Calculates the bitwise logical AND of r_{s1} and 16-bit concatenated Immediate value, ((0x00) & Imm), and stores the result in r_d.
Instruction Type	R-I
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register Imm = 8-bit unsigned immediate value

MSI

Opcode				Imm[7:0]								r_d			
0	1	0	1												

MSI	Move Signed Immediate
Instruction	Move low sign extended immediate to register
Operation	$r_d \leftarrow r_d$ SE (Imm)
Assembler Syntax	MSI r_d Imm

Example	MSI r5 100
Description	Writes the immediate value, Imm, into the low halfword of r_d, and sign extends the higher halfword of r_d.
Instruction Type	I
Instruction Fields	r_d= Index of destination register Imm = 8-bit unsigned immediate value

SHADOW

R15[15:8]

MSI Shadow Instruction

Shadow bit for MSI instruction is R15[14]. If this bit is 1, then the shadow instruction of MSI will be executed as below.

MSI-SHADOW

Opcode				Imm[7:0]								r_d			
0	1	0	1												

MSI	**Move Signed Immediate**
Instruction	Move low sign extended immediate to register
Operation	$r_d \leftarrow r_d$ SE (Imm)
Assembler Syntax	MSI r_d Imm
Example	MSI r5 100
Description	Writes the immediate value, Imm, into the low halfword of r_d, and sign extends the higher halfword of r_d.
Instruction Type	I
Instruction Fields	r_d= Index of destination register Imm = 8-bit unsigned immediate value

MHI

Opcode				Imm[7:0]								r_d			
0	1	1	0												

MHI	**Move High Immediate**
Instruction	Move high immediate to register
Operation	$r_d[15:8] \leftarrow$ Imm
Assembler Syntax	MHI r_d Imm
Example	MHI r5 100
Description	Writes the immediate value, Imm, into the high halfword of r_d.
Instruction Type	I
Instruction Fields	r_d= Index of destination register Imm = 8-bit unsigned immediate value

SLR

Opcode				r_{s1}				r_{s2}				r_d			
0	1	1	1												

SLR	**Shift Logical Registered**
Instruction	Logical Left/Right shift
Operation	$r_d \leftarrow r_{s1} << (\pm r_{s2}[4:0])$
Assembler Syntax	SLR r_d r_{s1} r_{s2}
Example	SLR r_5 r_2 r_3
Description	Shifts r_{s1} by the number of bits specified in $r_{s2}[4:0]$ and then stores the result in r_d. Based on the sign of r_{s2} [4:0] left (-) or right (+) will be performed.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register

SAR

Opcode				r_{s1}				r_{s2}				r_d			
1	0	0	0												

SAR	**Shift Arithmetic Registered**
Instruction	Arithmetic Left/Right shift
Operation	$r_d \leftarrow r_{s1} <<< (\pm r_{s2}[4:0])$
Assembler Syntax	SAR r_d r_{s1} r_{s2}
Example	SAR r_5 r_2 r_3
Description	Shifts r_{s1} by the number of bits specified in $r_{s2}[4:0]$ and then stores the result in r_d. Based on the sign of r_{s2} [4:0] left (-) or right (+) will be performed.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register

ADD

Opcode				r_{s1}				r_{s2}				r_d			
1	0	0	1												

ADD	**Add Registered**
Instruction	Adding two registers
Operation	$r_d \leftarrow r_{s1} + r_{s2}$
Assembler Syntax	ADD r_d r_{s1} r_{s2}
Example	ADD r_5 r_2 r_3
Description	Calculates the sum of r_{s1} and r_{s2}. Stores the result in r_d. Used for unsigned addition.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register

SUB

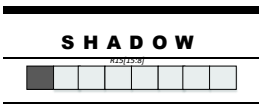
Opcode				r_{s1}				r_{s2}				r_d			
1	0	1	0												

SUB	**SUB Registered**
Instruction	Subtracting two registers
Operation	$r_d \leftarrow r_{s1} - r_{s2}$
Assembler Syntax	SUB r_d r_{s1} r_{s2}
Example	SUB r_5 r_2 r_3
Description	Subtract r_{s2} from r_{s1} and store the result in r_d.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register

ADI

Opcode				Imm[7:0]								r_d			
1	0	1	1												

ADI	**ADD Immediate**
Instruction	Adding Immediate to destination register
Operation	$r_d \leftarrow r_d$ + SE (Imm)
Assembler Syntax	ADI r_d Imm
Example	ADI r_5 150
Description	Sign-extends the 8-bit immediate value and adds it to the value of r_d. Stores the sum in r_d.
Instruction Type	I
Instruction Fields	r_d= Index of destination register Imm = 8-bit unsigned immediate value



ADI Shadow Instruction

Shadow bit for ADIR instruction is R15[15]. If this bit is 1, then the shadow instruction of ADI will be executed as below

ADIR

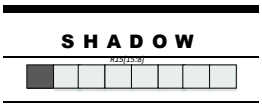
Opcode				Imm[3:0]				rs1				rd			
1	0	1	1												

ADI-SHADOW	ADD Immediate-Register
Instruction	Adding Immediate to source register
Operation	$r_d \leftarrow r_{s1} + SE(Imm)$
Assembler Syntax	ADIR r_d r_{s1} Imm
Example	ADIR r5 r1 2
Description	Sign-extends the 4-bit immediate value and adds it to the value of r_{s1}. Stores the sum in r_d.
Instruction Type	R-I
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register Imm = 4-bit unsigned immediate value

SUI

Opcode				Imm[7:0]								r_d			
1	1	0	0												

SUI	SUB Immediate
Instruction	Subtracting Immediate from destination register
Operation	$r_d \leftarrow r_d - SE(Imm)$
Assembler Syntax	SUI r_d Imm
Example	SUI r_5 150
Description	Sign-extends the 8-bit immediate value and subtracts it from the value of r_d. Stores the result in r_d.
Instruction Type	I
Instruction Fields	r_d= Index of destination register Imm = 8-bit unsigned immediate value



SUI Shadow Instruction

Shadow bit for SUIR instruction is R15[15]. If this bit is 1, then the shadow instruction of SUI will be executed as below.

SUIR

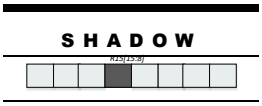
Opcode				Imm[3:0]				rs1				rd			
1	0	1	1												

SUI-SHADOW	SUB Immediate-Register
Instruction	Subtracting Immediate from source register
Operation	$r_d \leftarrow r_{s1} - SE(Imm)$
Assembler Syntax	SUIR r_d r_{s1} Imm
Example	SUIR r5 r1 2
Description	Sign-extends the 4-bit immediate value and subtracts it from the value of r_{s1}. Stores the sum in r_d.
Instruction Type	R-I
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register Imm = 4-bit unsigned immediate value

MUL

Opcode				r_{s1}				r_{s2}				r_d			
1	1	0	1												

MUL	**Multiply Registered**
Instruction	Multiplying two registers
Operation	$r_d \leftarrow r_{s1} * r_{s2}$
Assembler Syntax	MUL r_d r_{s1} r_{s2}
Example	MUL r_5 r_3 r_2
Description	Multiplies r_{s1} times r_{s2} and stores the 16 high-order bits of the product to r_d. This instruction performs unsigned multiplication. 16-bit low order bits will be stored in r_d+1.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register



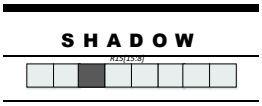
MUL Shadow Instruction

Shadow bit for MULS instruction is R15[12]. If this bit is 1, then the shadow instruction of MUL will be executed as below.

MULS

Opcode				r_{s1}				r_{s2}				r_d			
1	1	0	1												

MUL-SIGNED	**Multiply Registered Signed**
Instruction	Multiplying two registers
Operation	$r_d \leftarrow r_{s1} * r_{s2}$
Assembler Syntax	MULS r_d r_{s1} r_{s2}
Example	MULS r_5 r_3 r_2
Description	Multiplies r_{s1} times r_{s2} and stores the 16 high-order bits of the product to r_d. In this shadow instruction, signed multiplication will be executed. 16-bit low order bits will be stored in r_d+1.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register



MUL Shadow Instruction

Shadow bit for MULL instruction is R15[13]. If this bit is 1, then the shadow instruction of MUL will be executed as below.

MULL

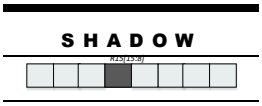
Opcode				r_{s1}				r_{s2}				r_d			
1	1	0	1												

MUL-LSB	**Multiply Registered**
Instruction	Multiplying two registers
Operation	$r_d \leftarrow r_{s1} * r_{s2}$
Assembler Syntax	MULSG r_d r_{s1} r_{s2}
Example	MULSG r_5 r_3 r_2
Description	Multiplies r_{s1} times r_{s2} .In this shadow instruction, only the right most bits of the result will be stored in the destination register.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register

DIV

Opcode				r_{s1}				r_{s2}				r_d			
1	1	1	0												

DIV	Divide Registered
Instruction	Dividing two registers
Operation	$r_d \leftarrow r_{s1} \div r_{s2}$
Assembler Syntax	DIV r_d r_{s1} r_{s2}
Example	DIV r_5 r_3 r_2
Description	Divides r_{s1} by r_{s2} and then stores the integer portion of the resulting quotient to r_d. This is an unsigned division and remainder will be stored in r_d+1.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register



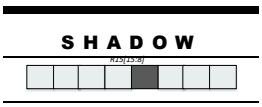
DIV Shadow Instruction

Shadow bit for DIVS instruction is R15[12]. If this bit is 1, then the shadow instruction of DIV will be executed as below.

DIVS

Opcode				r_{s1}				r_{s2}				r_d			
1	1	1	0												

DIV-SIGNED	Divide Registered
Instruction	Dividing two registers
Operation	$r_d \leftarrow r_{s1} \div r_{s2}$
Assembler Syntax	DIVS r_d r_{s1} r_{s2}
Example	DIVS r_5 r_3 r_2
Description	Divides r_{s1} by r_{s2} and then stores the integer portion of the resulting quotient to r_d. This shadow instruction performs signed division.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register



DIV Shadow Instruction

Shadow bit for DIVQ instruction is R15[11]. If this bit is 1, then the shadow instruction of DIV will be executed as below.

DIVQ

Opcode				r_{s1}				r_{s2}				r_d			
1	1	1	0												

DIV-SIGNED	Divide Registered
Instruction	Dividing two registers
Operation	$r_d \leftarrow r_{s1} \div r_{s2}$
Assembler Syntax	DIVQ r_d r_{s1} r_{s2}
Example	DIVQ r_5 r_3 r_2
Description	Divides r_{s1} by r_{s2} and stores only the quotient to r_d.
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1}= Index of source1 register r_{s2}= Index of source2 register

SHI

Opcode				Opcode-Extended			LA	r_{s1}				r_d			
1	1	1	1	1	1	0	0/1								

SHI	Shift Arithmetic/Logical Immediate
Instruction	Arithmetic Logical shift with immediate
Operation	$r_d \leftarrow r_d$ <<(± shimm) if (LA=0) $r_d \leftarrow r_d$ <<<(± shimm) if (LA=1)
Assembler Syntax	SAR r_d r_{s1} r_{s2}
Example	SAR r_5 r_2 r_3
Description	Shifts r_d by the number of bits specified in shimm and then stores the result in r_d. Based on the sign of shimm left (-) or right (+) will be performed. If the value of LA equals to zero and one, logical and arithmetic shift is done respectively.
Instruction Type	I
Instruction Fields	r_d= Index of destination register shimm=5-bit immediate value for shift

NTR

Opcode				Opcode-Extended			1/2C	r_{s1}				r_d			
1	1	1	1	1	1	0	0/1								

NTR	Not Registered
Instruction	Logical NOT
Operation	$r_d \leftarrow$ 1's complement (r_{s1}) if (1/2C=0) $r_d \leftarrow$ 2's complement (r_{s1}) if (1/2C=1)
Assembler Syntax	NTR r_d r_{s1}
Example	NTR r_5 r_2
Description	Based on the value of 1/2C being 0 and 1, calculates 1's and 2's complement of register r_{s1} respectively and stores the value in register r_d
Instruction Type	R
Instruction Fields	r_d= Index of destination register r_{s1} = Index of source register 1/2C=Selection between 1's and 2's complement

NTD

Opcode				Opcode-Extended			1/2C					r_d			
1	1	1	1	1	1	1	0/1								

NTD	Not Registered
Instruction	Logical NOT
Operation	$r_d \leftarrow$ 1's complement (r_d) if (1/2C=0) $r_d \leftarrow$ 2's complement (r_d) if (1/2C=1)
Assembler Syntax	NTD r_d
Example	NTD r_5
Description	Based on the value of 1/2C being 0 and 1, calculates 1's and 2's complement of register r_d respectively and stores the value in register r_d
Instruction Type	R
Instruction Fields	r_d= Index of destination register 1/2C=Selection between 1's and 2's complement

Index

No index entries found.