SRP16 ISA Specification

v0.1.0

Vishnu Shankar B

Architecture Overview

Architecture Block Diagram

Fig 1 shows the overall architecture block diagram and internals of SRP16 processor.

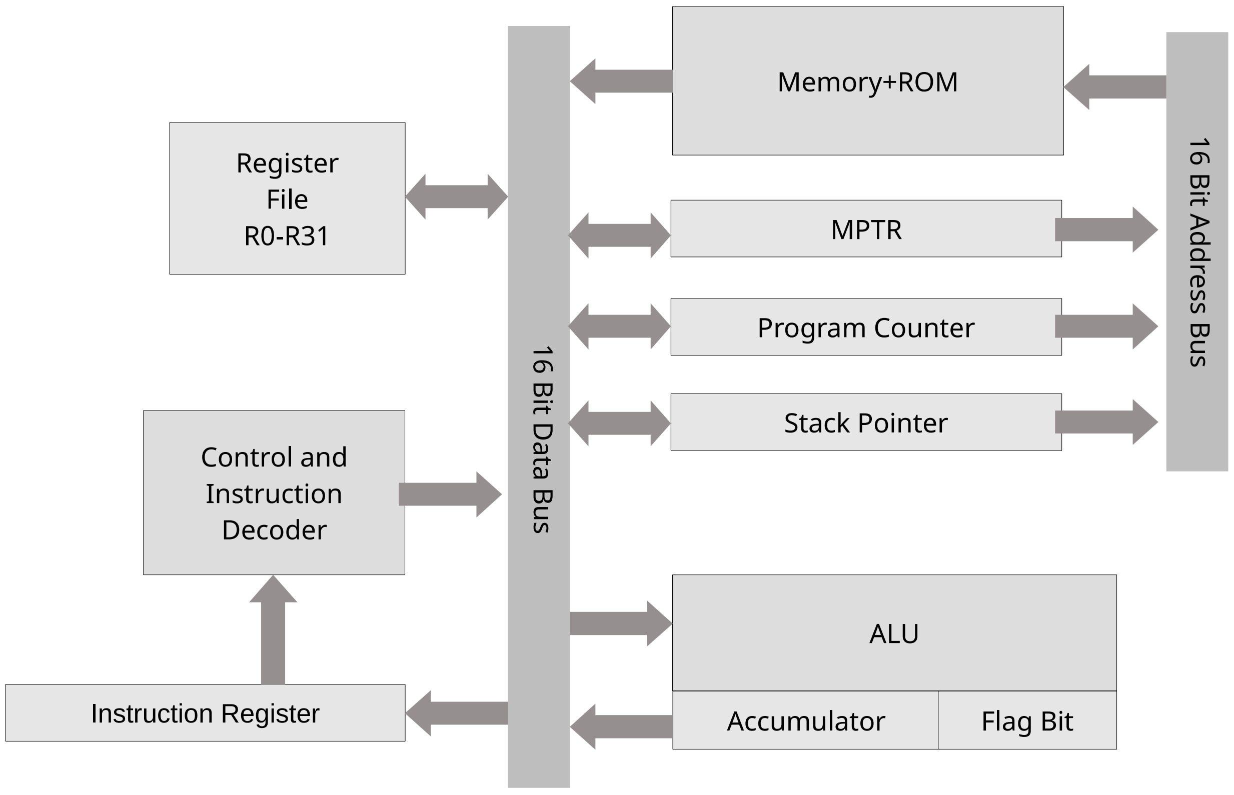

Fig 1: SRP16 Architecture Block Diagram

SRP16 Features

- 16-bit RISC processor
- 32 general purpose registers
- Up to 64 KiB of memory
- Fixed length instruction width (2-bytes)
- Little endian

General Purpose Registers

There are 32 general purpose registers, R0-R31. Only R0-R15 can be accessed by Load instructions.

Accumulator Register

The Accumulator register is used to accumulate/store results of arithmetic and logic instructions.

Flag Bit

The Flag Bit is used by compare instructions. After execution of a compare instruction, the flag bit will be set if the comparison is true or cleared otherwise.

Memory Pointer Register or MPTR

SRP16 is a load-store architecture, any instruction that reads from or writes to memory (except for instructions that deal with the stack) will use this register as pointer to a location in memory.

Stack Pointer

Holds the address of top of the stack. It decrements when you push an item onto the stack and increments when you pop an item off the stack.

Program Counter

The Program Counter holds the address of the current instruction being fetched/executed.

Memory

SRP16 can access up to 64 KiB of memory. This includes program memory, code memory and the stack.

Arithmetic and Logic Unit or ALU

Performs 16-bit arithmetic, logical/bitwise operations and comparisons. It store the result in the Accumulator Register and sets/clears the Flag Bit.

Instruction Register

The Instruction Register is used to store the instruction after fetching it from memory.

Control and Instruction Decoder

As each instruction is being fetched from memory, it will be first stored in the Instruction Register. The instruction stored in the Instruction Register will be used by the Instruction Decoder to generate control signals for different parts of the processor to execute the instruction. This block has internal counter registers to keep track of the current set of microinstructions the processor is executing.

Instruction Set

Instruction Encoding Types

SRP16 has three Instruction Encoding types as shown in Fig 2. All are 2 bytes in length.

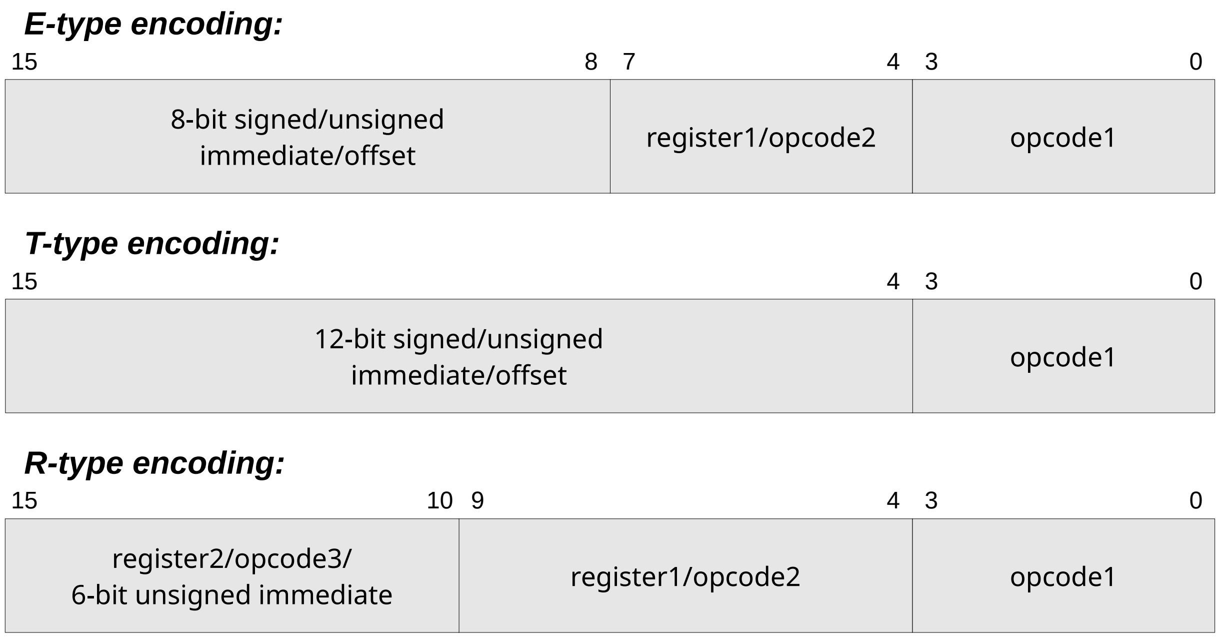

Fig 2: SRP16 Instruction Encoding Types

Register IDs

Each register has been given a unique 6-bit ID that will be used in the instruction encoding to specify which register to use as the operand.

• R0-R15:	000000 to 001111
• R16-R31:	010000 to 011111
• Accumulator (R60):	111100
• MPTR (R61):	111101
• SP (R62):	111110
• PC (R63):	111111

E-type instructions can only access R0-R15 since it has only four bits for the register field.

Immediate Values

All 8-bit or 12-bit signed immediate values are sign extended to 16-bits. All 8-bit or 12-bit unsigned immediate values are zero extended to 16-bits.

E-type Load Instructions

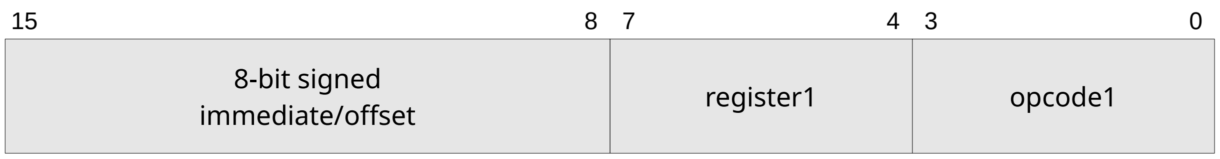

Fig 3: E-type Load Instructions' Encoding

LDR Instruction

- Description: Loads 8-bit signed immediate value to register
- Syntax: `LDR Rx, 8-bit-signed-immediate`
- Operation: `Rx ← immediate`
- Opcode1: 0000

LDRU Instruction

- Description: Loads upper 8-bits of register
- Syntax: `LDRU Rx, 8-bit-unsigned-immediate`
- Operation: `Rx[15:8] ← immediate`
- Opcode1: 0001

LD@MPTR Instruction

- Description: Loads 16-bit value from memory to register
- Syntax: `LD@MPTR Rx, 8-bit-signed-offset`
- Operation: `Rx ← memory[MPTR], MPTR ← MPTR+offset`
- Opcode1: 0010

ST@MPTR Instruction

- Description: Stores contents of register into memory
- Syntax: `ST@MPTR Rx, 8-bit-signed-offset`
- Operation: `memory[MPTR] ← Rx, MPTR ← MPTR+offset`
- Opcode1: 0011

LDB@MPTR Instruction

- Description: Loads a byte from memory to lower 8-bits of register
- Syntax: `LDB@MPTR Rx, 8-bit-signed-offset`
- Operation: `Rx[7:0] ← memory[MPTR], MPTR ← MPTR+offset`
- Opcode1: 0100

STB@MPTR Instruction

- Description: Stores lower 8-bits of register into memory
- Syntax: `STB@MPTR Rx, 8-bit-signed-offset`
- Operation: `memory[MPTR] ← Rx[7:0], MPTR ← MPTR+offset`
- Opcode1: 0101

T-type Load Instructions

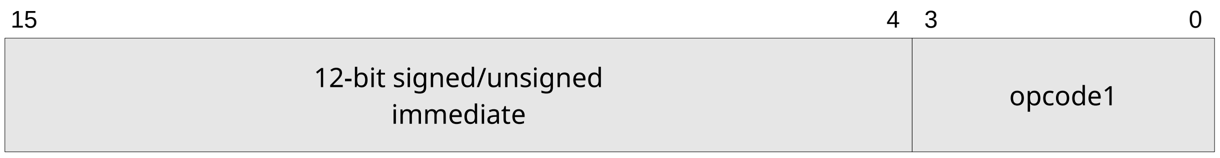

Fig 4: T-type Load Instructions' Encoding

LDA Instruction

- Description: Loads 12-bit signed immediate value to accumulator
- Syntax: `LDA 12-bit-signed-immediate`
- Operation: `A ← immediate`
- Opcode1: 0110

LDMPTR Instruction

- Description: Loads 12-bit unsigned immediate to MPTR
- Syntax: `LDMPTR 12-bit-unsigned-immediate`
- Operation: `MPTR ← immediate`
- Opcode1: 0111

LDMPTRU Instruction

- Description: Loads upper 4-bits of MPTR register
- Syntax: `LDMPTRU 12-bit-unsigned-immediate`
- Operation: `MPTR[15:12] ← immediate[3:0]`
- Opcode1: 1000

R-type Load Instruction

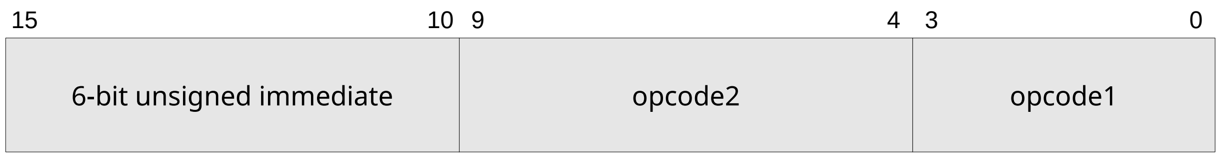

Fig 5: R-type Load Instruction's Encoding

LDAU Instruction

- Description: Loads upper 6-bits of accumulator
- Syntax: `LDAU 6-bit-unsigned-immediate`
- Operation: `A[15:12] ← immediate[3:0]`
- Opcode1: 1100
- Opcode2: 111011

R-type Move Instructions

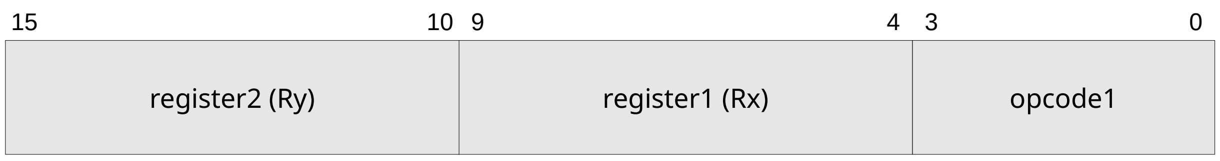

Fig 6: R-type Move Instructions' Encoding

MOV Instruction

- Description: Copies contents of source register to destination register
- Syntax: `MOV Rx, Ry`
- Operation: Rx ← Ry
- Opcode1: 1001

MOV Instruction (from PC)

- Description: Copies PC+4 to destination register
- Syntax: `MOV Rx, PC`
- `Operation:` `Rx ← PC+4`
- Opcode1: 1001
- register2: 111111 (PC register ID)

JMP Instruction

- Description: Branches to address specified by register
- Syntax: `JMP Ry or MOV PC, Ry`
- Operation: PC ← Ry
- Opcode1: 1001
- register1: 111111 (PC register ID)

T-type Jump Instructions

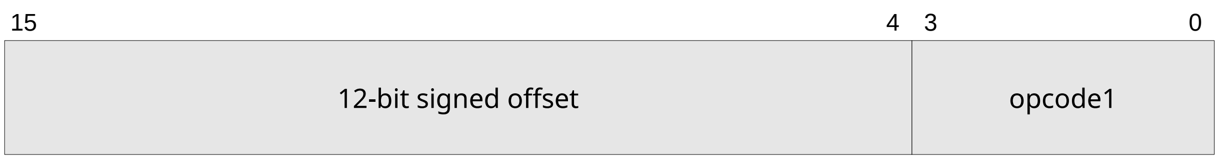

Fig 7: T-type Jump Instructions' Encoding

SJMP Instruction

- Description: Branches relative to PC
- Syntax: `SJMP 12-bit-signed-offset`
- Operation: PC ← `PC+offset`
- Opcode1: 1010

SJMPF Instruction

- Description: Branches relative to PC if flag is set
- Syntax: `SJMPF 12-bit-signed-offset`
- Operation: `if(flag): PC ← PC+offset`
- Opcode1: 1011

R-type Stack Instructions

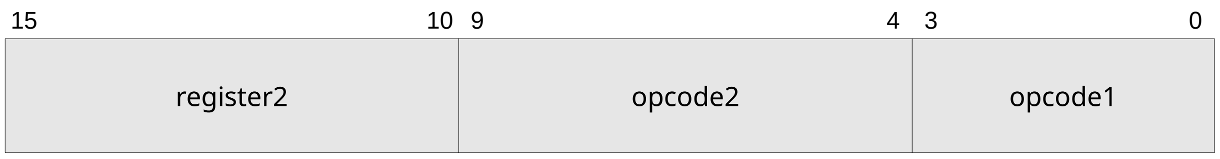

Fig 8: R-type Stack Instructions' Encoding

POP Instruction

- Description: Pops an item off the stack and copies it to register.
- Constraints: Can only access general purpose registers R0-R31.
- Syntax: `POP Rx`
- Operation: `Rx ← memory[SP], SP ← SP+1`
- Opcode1: 1100
- Opcode2: 111100

PUSH Instruction

- Description: Pushes contents of register to stack
- Constraints: Can only access general purpose registers R0-R31.
- Syntax: `PUSH Rx`
- Operation: `SP ← SP-1, memory[SP] ← Rx`
- Opcode1: 1100
- Opcode2: 111101

R-type Increment/Decrement Instructions

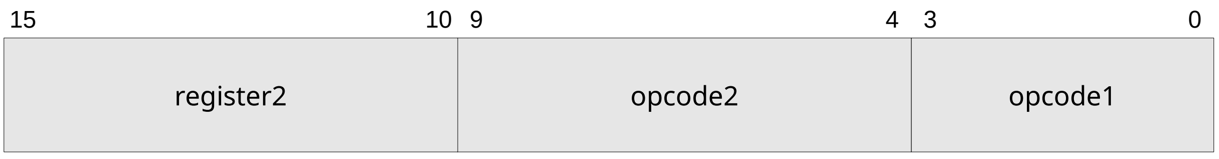

Fig 9: R-type Increment/Decrement Instructions' Encoding

INC Instruction

Description:	Increments contents of register
Constraints:	Can only access general purpose registers R0-R31.
Syntax:	`INC Rx`
Operation:	`Rx ← Rx+1`
Opcode1:	1100
Opcode2:	111110

DEC Instruction

Description:	Decrements contents of register
Constraints:	Can only access general purpose registers R0-R31.
Syntax:	`DEC Rx`
Operation:	`Rx ← Rx-1`
Opcode1:	1100
Opcode2:	111111

E-type Arithmetic Immediate Instructions

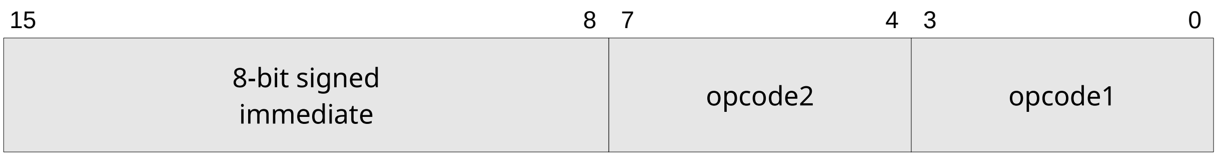

Fig 10: E-type Arithmetic Immediate Instructions' Encoding

ADDI Instruction

Description:	Adds 8-bit signed immediate value to accumulator
Syntax:	`ADDI 8-bit-signed-immediate`
Operation:	`A ← A+immediate`
Opcode1:	1101
Opcode2:	0001

ADCI Instruction

- Description: Adds 8-bit signed immediate value to accumulator with carry
- Syntax: `ADCI 8-bit-signed-immediate`
- Operation: `A ← A+immediate+carry`
- `Opcode1 :` 1101
- Opcode2: 1101

SBBI Instruction

- Description: Subtracts 8-bit signed immediate value from accumulator with borrow
- Syntax: `SBBI 8-bit-signed-immediate`
- Operation: `A ← A-immediate-carry`
- `Opcode1 :` 1101
- Opcode2: 1110

ANDI Instruction

- Description: Performs bitwise AND on accumulator and 8-bit signed immediate value
- Syntax: `ANDI 8-bit-signed-immediate`
- Operation: `A ← A&immediate`
- Opcode1: 1101
- Opcode 2: 0111

ORI Instruction

- Description: Performs bitwise OR on accumulator and 8-bit signed immediate value
- Syntax: `ORI 8-bit-signed-immediate`
- Operation: `A ← A|immediate`
- Opcode1: 1101
- Opcode 2: 1000

XORI Instruction

- Description: Performs bitwise XOR on accumulator and 8-bit signed immediate value
- Syntax: `XORI 8-bit-signed-immediate`
- Operation: `A ← A^immediate`
- Opcode1: 1101
- Opcode 2: 1001

R-type Immediate Arithmetic Instructions

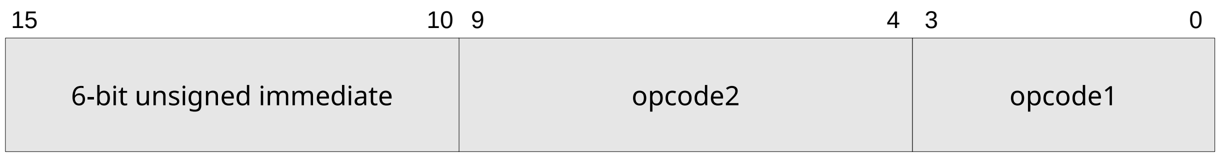

Fig 11: R-type Immediate Arithmetic Instructions' Encoding

SLAI Instruction

- Description: Performs arithmetic left shift on accumulator
- Syntax: `SLAI 6-bit-unsigned-immediate`
- Operation: `A← A<<<immediate`
- Opcode1: 1100
- Opcode2: 100011

SRAI Instruction

- Description: Performs arithmetic right shift on accumulator
- Syntax: `SRAI 6-bit-unsigned-immediate`
- Operation: `A← A>>>immediate`
- Opcode1: 1100
- Opcode2: 100100

SLLI Instruction

- Description: Performs logical left shift on accumulator
- Syntax: `SLLI 6-bit-unsigned-immediate`
- Operation: `A← A<<immediate`
- Opcode1: 1100
- Opcode2: 100101

SRLI Instruction

- Description: Performs logical right shift on accumulator
- Syntax: `SRLI 6-bit-unsigned-immediate`
- Operation: `A← A>>immediate`
- Opcode1: 1100
- Opcode2: 100110

R-type Arithmetic Instructions

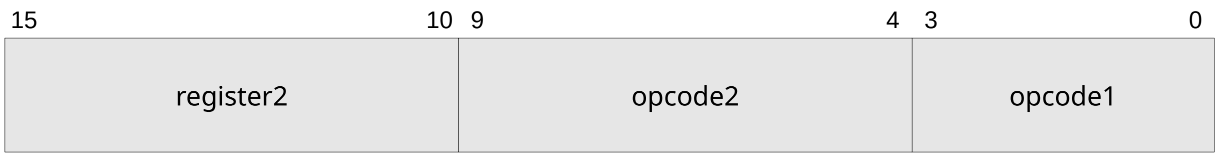

Fig 12: R-type Arithmetic Instructions' Encoding

ADD Instruction

- Description: Adds contents of register to accumulator
- Syntax: `ADD Rx`
- Operation: `A← A+Rx`
- Opcode1: 1100
- Opcode2: 000001

SUB Instruction

- Description: Subtracts contents of register from accumulator
- Syntax: `SUB Rx`
- Operation: `A← A-Rx`
- Opcode1: 1100
- Opcode2: 000010

ADC Instruction

- Description: Adds contents of register to accumulator with carry
- Syntax: `ADC Rx`
- Operation: `A← A+Rx+carry`
- Opcode1: 1100
- Opcode2: 001101

SBB Instruction

- Description: Subtracts contents of register from accumulator with borrow
- Syntax: `SBB Rx`
- Operation: `A← A-Rx-carry`
- Opcode1: 1100
- Opcode2: 001110

AND Instruction

- Description: Performs bitwise AND on accumulator and register
- Syntax: AND Rx
- Operation: A ← A&Rx
- Opcode1: 1100
- Opcode 2: 000111

OR Instruction

- Description: Performs bitwise OR on accumulator and register
- Syntax: OR Rx
- Operation: A ← A|Rx
- Opcode1: 1100
- Opcode 2: 001000

XOR Instruction

- Description: Performs bitwise XOR on accumulator and register
- Syntax: XOR Rx
- Operation: A ← A^Rx
- Opcode1: 1100
- Opcode 2: 001001

SLA Instruction

- Description: Performs arithmetic left shift by shift amount held in register
- Syntax: SLA Rx
- Operation: A← A<<<Rx
- Opcode1: 1100
- Opcode2: 000011

SRA Instruction

- Description: Performs arithmetic right shift by shift amount held in register
- Syntax: SRA Rx
- Operation: A← A>>>Rx
- Opcode1: 1100
- Opcode2: 000100

SLL Instruction

- Description: Performs logical left shift by shift amount held in register
- Syntax: SLL Rx
- Operation: A← A<<Rx
- Opcode1: 1100
- Opcode2: 000101

SRL Instruction

- Description: Performs logical right shift by shift amount held in register
- Syntax: SRL Rx
- Operation: A← A>>Rx
- Opcode1: 1100
- Opcode2: 000110

E-type Compare Immediate Instructions

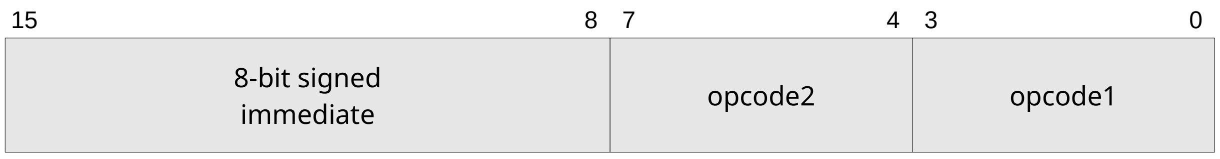

Fig 13: E-type Compare Immediate Instructions' Encoding

CLI Instruction

Description:	Sets flag bit if accumulator is less than 8-bit signed immediate
Syntax:	`CLI 8-bit-signed-immediate`
Operation:	`if(A<immediate): flag ← 1 else: flag ← 0`
Opcode1:	1101
Opcode2:	1010

CGI Instruction

Description:	Sets flag bit if accumulator is greater than 8-bit signed immediate
Syntax:	`CGI 8-bit-signed-immediate`
Operation:	`if(A>immediate): flag ← 1 else: flag ← 0`
Opcode1:	1101
Opcode2:	1011

CEI Instruction

Description:	Sets flag bit if accumulator is equal to 8-bit signed immediate
Syntax:	`CEI 8-bit-signed-immediate`
Operation:	`if(A==immediate): flag ← 1 else: flag ← 0`
Opcode1:	1101
Opcode2:	1100

R-type Compare Instructions

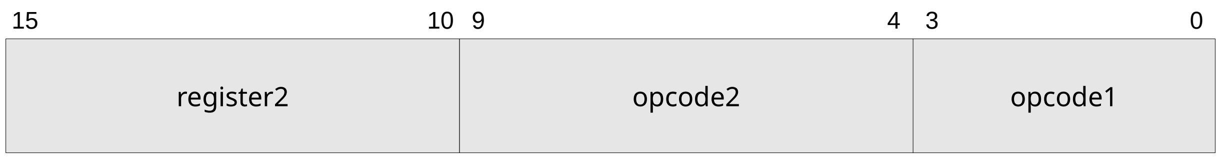

Fig 14: R-type Compare Instructions' Encoding

CL Instruction

- Description: Sets flag bit if accumulator is less than register
- Syntax: `CL Rx`
- Operation: `if(A<Rx): flag ← 1 else: flag ← 0`
- Opcode1: 1100
- Opcode2: 001010

CG Instruction

- Description: Sets flag bit if accumulator is greater than register
- Syntax: `CG Rx`
- Operation: `if(A>Rx): flag ← 1 else: flag ← 0`
- Opcode1: 1100
- Opcode2: 001011

CE Instruction

- Description: Sets flag bit if accumulator is equal to register
- Syntax: `CE Rx`
- Operation: `if(A==Rx): flag ← 1 else: flag ← 0`
- Opcode1: 1100
- Opcode2: 001100

R-type Flag Instruction

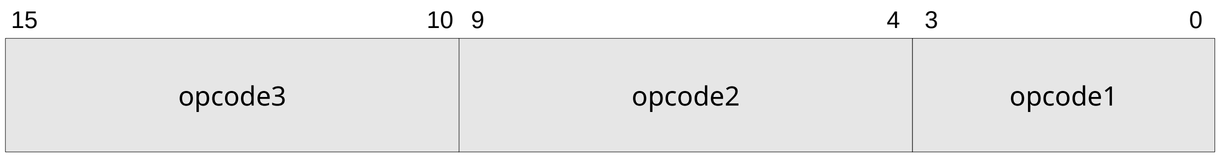

Fig 15: R-type Flag Instruction's Encoding

NOTF Instruction

- Description: Inverts flag bit
- Syntax: `NOTF`
- Operation: `flag ← !flag`
- `Opcode1:` `1100`
- `Opcode2:` `010000`
- `Opcode3:` `000000 or xxxxxx`

SRP16 Opcode Table

Instruction	Encoding Type	Opcode2	Opcode1
LDR	E-type	-	0000
LDRU	E-type	-	0001
LD@MPTR	E-type	-	0010
ST@MPTR	E-type	-	0011
LDB@MPTR	E-type	-	0100
STB@MPTR	E-type	-	0101
LDA	T-type	-	0110
LDMPTR	T-type	-	0111
LDMPTRU	T-type	-	1000
MOV	R-type	-	1001
SJMP	T-type	-	1010
SJMPF	T-type	-	1011
ADD	R-type	000001	1100
SUB	R-type	000010	1100
SLA	R-type	000011	1100
SRA	R-type	000100	1100
SLL	R-type	000101	1100
SRL	R-type	000110	1100
AND	R-type	000111	1100
OR	R-type	001000	1100
XOR	R-type	001001	1100
CL	R-type	001010	1100
CG	R-type	001011	1100
CE	R-type	001100	1100
ADC	R-type	001101	1100
SBB	R-type	001110	1100
NOTF	R-type	010000	1100
SLAI	R-type	100011	1100
SRAI	R-type	100100	1100
SLLI	R-type	100101	1100
SRLI	R-type	100110	1100
LDAU	R-type	111011	1100
POP	R-type	111100	1100
PUSH	R-type	111101	1100
INC	R-type	111110	1100

Instruction	Encoding Type	Opcode2	Opcode1
DEC	R-type	111111	1100
ADDI	E-type	0001	1101
ANDI	E-type	0111	1101
ORI	E-type	1000	1101
XORI	E-type	1001	1101
CLI	E-type	1010	1101
CGI	E-type	1011	1101
CEI	E-type	1100	1101
ADCI	E-type	1101	1101
SBBI	E-type	1110	1101

SRP16 Instruction Set Quick Reference

Instruction	Operation
`LDR Rx, 8-bit-signed-immediate`	`Rx ← immediate`
`LDRU Rx, 8-bit-unsigned-immediate`	`Rx[15:8] ← immediate`

`LD@MPTR Rx, 8-bit-signed-offset`	`Rx ← memory[MPTR]`
	`MPTR ← MPTR+offset`

`ST@MPTR Rx, 8-bit-signed-offset`	`memory[MPTR] ← Rx`
	`MPTR ← MPTR+offset`

`LDB@MPTR Rx, 8-bit-signed-offset`	`Rx[7:0] ← memory[MPTR]`
	`MPTR ← MPTR+offset`

`STB@MPTR Rx, 8-bit-signed-offset`	`memory[MPTR] ← Rx[7:0]`
	`MPTR ← MPTR+offset`

`LDA 12-bit-signed-immediate`	`A ← immediate`
`LDAU 6-bit-unsigned-immediate`	`A[15:12] ← immediate[3:0]`
`LDMPTR 12-bit-unsigned-immediate`	`MPTR ← immediate`
`LDMPTRU 12-bit-signed-immediate`	`MPTR[15:12] ← immediate[3:0]`
`MOV Rx, Ry`	`Rx ← Ry`
`MOV Rx, PC`	`Rx ← PC+4`
`JMP Ry or MOV PC, Ry`	`PC ← Ry`
`SJMP 12-bit-signed-offset`	`PC ← PC+offset`
`SJMPF 12-bit-signed-offset`	`if(flag): PC ← PC+offset`
`NOTF`	`flag ← !flag`

`POP Rx`	`Rx ← memory[SP]`
	`SP ← SP+1`

`PUSH Rx`	`SP ← SP-1`
	`memory[SP] ← Rx`

`INC Rx`	`Rx ← Rx+1`
`DEC Rx`	`Rx ← Rx-1`
`ADDI 8-bit-signed-immediate`	`A ← A+immediate`
`ADCI 8-bit-signed-immediate`	`A ← A+immediate+carry`
`SBBI 8-bit-signed-immediate`	`A ← A-immediate-carry`
`ANDI 8-bit-signed-immediate`	`A ← A&immediate`
`ORI 8-bit-signed-immediate`	`A ← A|immediate`
`XORI 8-bit-signed-immediate`	`A ← A^immediate`
`SLAI 6-bit-unsigned-immediate`	`A ← A<<<immediate`
`SRAI 6-bit-unsigned-immediate`	`A ← A>>>immediate`
`SLLI 6-bit-unsigned-immediate`	`A ← A<<immediate`
`SRLI 6-bit-unsigned-immediate`	`A ← A>>immediate`

Instruction	**Operation**
ADD Rx	A ← A+Rx
SUB Rx	A ← A-Rx
ADC Rx	A ← A+Rx+carry
SBB Rx	A ← A-Rx-carry
AND Rx	A ← A&Rx
OR Rx	A ← A|Rx
XOR Rx	A ← A^Rx
SLA Rx	A ← A<<<Rx
SRA Rx	A ← A>>>Rx
SLL Rx	A ← A<<Rx
SRL Rx	A ← A>>Rx

CLI 8-bit-signed-immediate	if(A<immediate): flag ← 1
	else: flag ← 0

CGI 8-bit-signed-immediate	if(A>immediate): flag ← 1
	else: flag ← 0

CEI 8-bit-signed-immediate	if(A==immediate): flag ← 1
	else: flag ← 0

CL Rx	if(A<Rx): flag ← 1
	else: flag ← 0

CG Rx	if(A>Rx): flag ← 1
	else: flag ← 0

CE Rx	if(A==Rx): flag ← 1
	else: flag ← 0