Computer Sciences Department

University of Wisconsin-Madison

CS/ECE 552 – Introduction to Computer Architecture

WISC-SP19 ISA Specification

1. Instruction Summary

(**KEY**: sss = rs, ddd = rd, ttt = rt, iii* = immediate)

Instruction Format	**Syntax**	**Semantics**
00000 xxxxxxxxxx	HALT	Cease instruction issue, dump memory state to file
00001 xxxxxxxxxx	NOP	None
01000 sss ddd iiiii	SUBI Rd, Rs, immediate	Rd <- I(sign ext.) – Rs
01001 sss ddd iiiii	ADDI Rd, Rs, immediate	Rd <- Rs + I(sign ext.)
01010 sss ddd iiiii	ANDNI Rd, Rs, immediate	Rd <- Rs AND ~I(zero ext.)
01011 sss ddd iiiii	XORI Rd, Rs, immediate	Rd <- Rs XOR I(zero ext.)
10100 sss ddd iiiii	ROLI Rd, Rs, immediate	Rd <- Rs <<(rotate) I(lowest 4 bits)
10101 sss ddd iiiii	SLLI Rd, Rs, immediate	Rd <- Rs << I(lowest 4 bits)
10110 sss ddd iiiii	RORI Rd, Rs, immediate	Rd <- Rs >>(rotate) I(lowest 4 bits)
10111 sss ddd iiiii	SRLI Rd, Rs, immediate	Rd <- Rs >> I(lowest 4 bits)
10000 sss ddd iiiii	ST Rd, Rs, immediate	Mem[Rs + I(sign ext.)] <- Rd

10001 sss ddd iiiii	LD Rd, Rs, immediate	Rd <- Mem[Rs + I(sign ext.)]
10011 sss ddd iiiii	STU Rd, Rs, immediate	Mem[Rs + I(sign ext.)] <- Rd Rs <- Rs + I(sign ext.)
11001 sss xxx ddd xx	BTR Rd, Rs	Rd[bit i] <- Rs[bit 15-i] for i=0..15
11011 sss ttt ddd 00	ADD Rd, Rs, Rt	Rd <- Rs + Rt
11011 sss ttt ddd 01	SUB Rd, Rs, Rt	Rd <- Rt - Rs
11011 sss ttt ddd 10	XOR Rd, Rs, Rt	Rd <- Rs XOR Rt
11011 sss ttt ddd 11	ANDN Rd, Rs, Rt	Rd <- Rs AND ~Rt
11010 sss ttt ddd 00	ROL Rd, Rs, Rt	Rd <- Rs << (rotate) Rt (lowest 4 bits)
11010 sss ttt ddd 01	SLL Rd, Rs, Rt	Rd <- Rs << Rt (lowest 4 bits)
11010 sss ttt ddd 10	ROR Rd, Rs, Rt	Rd <- Rs >> (rotate) Rt (lowest 4 bits)
11010 sss ttt ddd 11	SRL Rd, Rs, Rt	Rd <- Rs >> Rt (lowest 4 bits)
11100 sss ttt ddd xx	SEQ Rd, Rs, Rt	if (Rs == Rt) then Rd <- 1 else Rd <- 0
11101 sss ttt ddd xx	SLT Rd, Rs, Rt	if (Rs < Rt) then Rd <- 1 else Rd <- 0
11110 sss ttt ddd xx	SLE Rd, Rs, Rt	if (Rs <= Rt) then Rd <- 1 else Rd <- 0
11111 sss ttt ddd xx	SCO Rd, Rs, Rt	if (Rs + Rt) generates carry out then Rd <- 1 else Rd <- 0

01100 sss iiiiiiii	BNEZ Rs, immediate	if (Rs != 0) then PC <- PC + 2 + I(sign ext.)
01101 sss iiiiiiii	BEQZ Rs, immediate	if (Rs == 0) then PC <- PC + 2 + I(sign ext.)
01110 sss iiiiiiii	BLTZ Rs, immediate	if (Rs < 0) then PC <- PC + 2 + I(sign ext.)
01111 sss iiiiiiii	BGEZ Rs, immediate	if (Rs >= 0) then PC <- PC + 2 + I(sign ext.)
11000 sss iiiiiiii	LBI Rs, immediate	Rs <- I(sign ext.)
10010 sss iiiiiiii	SLBI Rs, immediate	Rs <- (Rs << 8) | I(zero ext.)
00100 dddddddddddd	J displacement	PC <- PC + 2 + D(sign ext.)
00101 sss iiiiiiii	JR Rs, immediate	PC <- Rs + I(sign ext.)
00110 dddddddddddd	JAL displacement	R7 <- PC + 2 PC <- PC + 2 + D(sign ext.)
00111 sss iiiiiiii	JALR Rs, immediate	R7 <- PC + 2 PC <- Rs + I(sign ext.)
00010	siic Rs	produce IllegalOp exception. Must provide one source register.
00011 xxxxxxxxxxx	NOP / RTI	PC <- EPC

2. Formats

WISC-SP19 supports instructions in four different formats: J-format, 2 I-formats, and the R-format. These are described below.

2.1 J-format

The J-format is used for jump instructions that need a large displacement.

J-Format

5 bits	11 bits
Op Code	Displacement

Jump Instructions

The Jump instruction loads the PC with the value found by adding the PC of the next instruction (PC+2, not PC+4 as in MIPS) to the **sign-extended** displacement.

The Jump-And-Link instruction loads the PC with the same value and also saves the address of the next sequential instruction (i.e., PC+2) in the link register R_7.

The syntax of the jump instructions is:

- `J displacement`
- `JAL displacement`

2.2 I-format

I-format instructions use either a destination register, a source register, and a 5-bit immediate value; or a destination register and an 8-bit immediate value. The two types of I-format instructions are described below.

I-format 1 Instructions

I-format 1

5 bits	3 bits	3 bits	5 bits
Op Code	R_s	R_d	Immediate

The I-format 1 instructions include XOR-Immediate, ANDN-Immediate, Add-Immediate, Subtract-Immediate, Rotate-Left-Immediate, Shift-Left-Logical-Immediate, Rotate-Right-Immediate, Shift-Right-Logical-Immediate, Load, Store, and Store with Update.

The **ANDNI** instruction loads register R_d with the value of the register R_s AND-ed with the **one's complement** of the zero-extended immediate value. (It may be thought of as a bit-clear instruction.) **ADDI** loads register R_d with the sum of the value of the register R_s plus the **sign-extended** immediate value. **SUBI** loads register R_d with the result of subtracting register R_s from the **sign-extended** immediate value. (That is, immed - R_s, **not** R_s - immed.) Similar instructions have similar semantics, i.e. the logical instructions have zero-extended values and the arithmetic instructions have sign-extended values.

For Load and Store instructions, the effective address of the operand to be read or written is calculated by adding the value in register R_s with the **sign-extended** immediate value. The value is loaded to or stored from register R_d. The **STU** instruction, Store with Update, acts like Store but also writes R_s with the effective address.

The syntax of the I-format 1 instructions is:

- `ADDI` R_d, R_s, `immediate`
- `SUBI` R_d, R_s, `immediate`
- `XORI` R_d, R_s, `immediate`
- `ANDNI` R_d, R_s, `immediate`
- `ROLI` R_d, R_s, `immediate`
- `SLLI` R_d, R_s, `immediate`
- `RORI` R_d, R_s, `immediate`
- `SRLI` R_d, R_s, `immediate`
- `ST` R_d, R_s, `immediate`
- `LD` R_d, R_s, `immediate`

- STU R_d, R_s, immediate

I-format 2 Instructions

I-format 2

5 bits	3 bits	8 bits
Op Code	R_s	Immediate

The Load Byte Immediate instruction loads R_s with a sign-extended 8-bit immediate value.

The Shift-and-Load-Byte-Immediate instruction shifts R_s 8 bits to the left and replaces the lower 8 bits with the immediate value.

The format of these instructions is:

- LBI R_s, signed immediate
- SLBI R_s, unsigned immediate

The Jump-Register instruction loads the PC with the value of register R_s + signed immediate. The Jump-And-Link-Register instruction does the same and also saves the return address (i.e., the address of the JALR instruction plus one) in the link register R_7. The format of these instructions is

- JR Rs, immediate
- JALR Rs, immediate

The branch instructions test a general-purpose register for some condition. The available conditions are: equal to zero, not equal to zero, less than zero, and greater than or equal to zero. If the condition holds, the signed immediate is added to the address of the next sequential instruction and loaded into the PC. The format of the branch instructions is

- BEQZ Rs, signed immediate
- BNEZ Rs, signed immediate
- BLTZ Rs, signed immediate

- `BGEZ Rs, signed immediate`

2.3 R-format

R-format instructions use only registers for operands.
R-format

5 bits	3 bits	3 bits	3 bits	2 bits
Op Code	Rs	Rt	Rd	Op Code Extension

ALU and Shift Instructions

The ALU and shift R-format instructions are similar to I-format 1 instructions, but do not require an immediate value. In each case, the value of R_t is used in place of the immediate. No extension of its value is required. **In the case of shift instructions, all but the 4 least-significant bits of R_t are ignored**.

The ADD instruction performs signed addition. The SUB instruction subtracts R_s from R_t. (*Not* R_s - R_t.) The set instructions SEQ, SLT, SLE instructions compare the values in R_s and R_t and set the destination register R_d to 0x1 if the comparison is true, and 0x0 if the comparison is false. SLT checks for R_s less than R_t, and SLE checks for R_s less than or equal to R_t. (R_s and R_t are two's complement numbers.) The set instruction SCO will set R_d to 0x1 if R_s plus R_t would generate a carry-out from the most significant bit; otherwise it sets R_d to 0x0. The Bit-Reverse instruction, BTR, takes a single operand R_s and copies it to R_d, but with a left-right reversal of each bit; i.e. bit 0 goes to bit 15, bit 1 goes to bit 14, etc.

The syntax of the R-format ALU and shift instructions is:

- `ADD` R_d, R_s, R_t
- `SUB` R_d, R_s, R_t
- `ANDN` R_d, R_s, R_t
- `ROL` R_d, R_s, R_t
- `SLL` R_d, R_s, R_t
- `ROR` R_d, R_s, R_t
- `SRL` R_d, R_s, R_t
- `SEQ` R_d, R_s, R_t

- SLT R_d, R_s, R_t
- SLE R_d, R_s, R_t
- SCO R_d, R_s, R_t
- BTR R_d, R_s

3. Special Instructions

Special instructions use the R-format. The HALT instruction halts the processor. The HALT instruction and all older instructions execute normally, but the instruction after the halt will never execute. The PC is left pointing to the instruction directly after the halt.

The No-operation instruction occupies a position in the pipeline but does nothing.

The syntax of these instructions is:

- HALT
- NOP

The SIIC and RTI instructions are extra credit and can be deferred for later. They will be not tested until the final demo.

The SIIC instruction is an illegal instruction and should trigger the exception handler. EPC should be set to PC + 2, and control should be transferred to the exception handler which is at PC 0x02.

The syntax of this instruction is:

- SIIC Rs

The source register name must be ignored. The syntax is specified this way with a dummy source register, to reuse some components from our existing assembler. The RTI instruction should remain equivalent to NOP until the rest of the design has been completed and thoroughly tested.

RTI returns from an exception by loading the PC from the value in the EPC register.

The syntax of this instruction is:

- `RTI`

See the Part 4 in the Microarchitecture description for more information on optimizations.