Appendix A - Instruction Set Manual

Notes about the syntax in this document:

All optional parameters in the assembly are denoted in square brackets. For example, the absolute value (`abs [rd] rs`) using the same register as the destination and source (i.e. `abs rs rs`) can be shortened without specifying the destination (`abs rs`).

For all the immediate instructions, since the difference between a register and an immediate value can easily be made, the mnemonics with immediate values (I type) can be written like their register (R type) counterparts. Example: `addi ra 5` can be written `add ra 5`.

In the machine code syntax, question marks (`?`) mark unused bits reserved for future usage. These bits should `0` by default to ensure forward compatibility.

Table of Contents:

ABS - Absolute value

Description

Calculates the absolute value of the source register and stores the result in the destination register.

Syntax

```
abs [rd] rs
```

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: Source register

Operation

```
if (rs < 0)
  rd <- -rs
else
  rd <- rs
```

Machine code

```
0000111? ???????? ??????SS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
set ra -5
abs rb ra
```

ADD - Addition

Description

Adds the content of two registers without carry and stores the result in the destination register.

Syntax

`add [rd] rs rt`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: First source register
- `rt`: Second source register

Operation

```
rd <- rs + rt
```

Machine code

`0000000? ???????? ?TTTTTSS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
add rc ra rb // rc = ra + rb
add rz sp // rz += sp
```

ADDI - Addition with immediate

Description

Adds the content of two registers without carry and stores the result in the destination register.

Syntax

`add[i] [rd] rs imm`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: Source register
- `imm`: Immediate value of the subtrahend

Operation

```
rd <- rs + imm
```

Machine code

`010000II IIIIIIII IIIIIISS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
addi rb ra 5 // rb = ra + 5
add rz -26 // rz += -26
```

AND - Logical AND

Description

Performs a bitwise logical AND operation between two registers and stores the result in the destination register.

Syntax

```
and [rd] rs rt
```

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: First source register
- `rt`: Second source register

Operation

```
rd <- rs & rt
```

Machine code

```
0000101? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
and rc ra rb // rc = ra & rb
and rz sp // rz &= sp
```

ANDI - Logical AND with immediate

Description

Performs a bitwise logical AND operation between the source register and the immediate value and stores the result in the destination register.

Syntax

`and[i] [rd] rs imm`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: First source register
- `imm`: Immediate value

Operation

```
rd <- rs & imm
```

Machine code

`010101II IIIIIIII IIIIIISS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
andi rb ra 5 // rb = ra & 5
and rz 26 // rz &= 26
```

B - Relative indirect jump

Description

Branches from the current address by the base in the specified register and a specific offset.

Syntax

```
b rb [offset]
```

Operands

- `rb`: Base address register
- `offset`: Signed offset, defaults to `0`

Operation

```
ip <- rb + offset
```

Machine code

```
10000000 OOOOOOOO OOOOOOOO OOORRRRR
```

Restrictions

None

Example

```
// switch case
tgei rb ra 3
jmp rb default
b ra 1
jmp case_0
jmp case_1
jmp case_2
switch_return:
```

BI - Relative jump

Description

Branches from the current address by the specified amount.

Syntax

```
bi offset
```

Operands

- `offset`: Signed offset for the address

Operation

```
ip <- ip + offset
```

Machine code

```
1001AAAA AAAAAAAA AAAAAAAA AAARRRRR
```

Restrictions

None

Example

```

```

BNZ - Relative jump if not zero

Description

Branches to the address if the value in the specified register is different from zero.

Syntax

`bnz reg offset`

Operands

- `reg`: Comparison register
- `offset`: Signed offset for the address

Operation

```
if (reg != 0)
  ip <- ip + offset
```

Machine code

`1011AAAA AAAAAAAA AAAAAAAA AAARRRRR`

Restrictions

None

Example

```
bnz ra afterInitialization
// Initialize the value
afterInitialization:
```

BZ - Relative jump if zero

Description

Branches to the address if the value in the specified register is equal to zero.

Syntax

`bz reg offset`

Operands

- `reg`: Comparison register
- `offset`: Signed offset for the address

Operation

```
if (reg = 0)
  ip <- ip + offset
```

Machine code

`1010AAAA AAAAAAAA AAAAAAAA AAARRRRR`

Restrictions

None

Example

```
bz ra dealWithDivisionByZero
div rb ra
```

CALL - Call indirect subroutine (Jump and link)

Description

Branches from the current address by the base in the specified register and a specific offset. Stores the address of the next current instruction on the top of the stack.

Syntax

```
call rb [offset]
```

Operands

- `rb`: Base address register
- `offset`: Optional offset, defaults to `0`

Operation

```
[sp] <- ip + 1
sp <- sp - 1
ip <- rb + offset
```

Machine code

```
11000000 00000000 00000000 000RRRRR
```

Restrictions

None

Example

```

```

CALLI - Call subroutine (Jump and link)

Description

Branches from the current address by the specified amount. Stores the address of the next current instruction on the top of the stack.

Syntax

`calli offset`

Operands

- `offset`: Signed offset for the address

Operation

```
[sp] <- ip + 1
sp <- sp - 1
ip <- offset
```

Machine code

`1101AAAA AAAAAAAA AAAAAAAA AAARRRRR`

Restrictions

None

Example

```
recursion:
teqi rt ra 10
jnz rt endRecursion
addi ra ra 1
call recursion
endRecursion
ret
```

DIV - Division

Description

Performs an integer division of the content of the first source register by the content of the second one and stores the result in the destination register.

Syntax

```
div [rd] rs rt
```

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: Source register for the dividend
- `rt`: Source register for the divisor

Operation

```
rd <- rs / rt
```

Machine code

```
0000011? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

The value of `rt` must be different than 0. Otherwise, an exception must be raised both at compile and runtime.

Example

```
[sp] <- ip + 1
sp <- sp + 1
ip <- ip + imm
```

HALT - Stop the program

Description

Stops the execution of the program.

This instruction is automatically added at the end of a program by the assembler.

Without it, the program keeps going, executing through the rest of the memory, leading to undefined behaviors.

Syntax

`halt`

Operands

None

Operation

None

Machine code

`1110???? ???????? ???????? ???????1`

Note: This is the same opcode as the `ret` instruction.

Restrictions

None

Example

```
// Some code
earlyExit:
halt
```

JMP - Absolute jump

Description

Jumps to the specified absolute address. It has 28 bits of address but is rarely used.

Syntax

```
jmp addr
```

Operands

- `addr`: Absolute address to jump to

Operation

```
ip <- addr
```

Machine code

```
1111AAAA AAAAAAAA AAAAAAAA AAAAAAAA
```

Restrictions

None

Example

```
infiniteLoop:
jmp infiniteLoop
```

LD - Load with direct addressing

Description

Reads the address stored in the source register and loads the value at that address in the destination register.

Syntax

```
ld rd rs
```

Operands

- `rd`: Destination register
- `rs`: Source register holding the address

Operation

```
rd <- [rs]
```

Machine code

```
0010001? ???????? ??????SS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

```
// rp: pointer to a 0-ended array of integers
set ra 0
sum:
ld rt rp
add ra ra rt
bnz rt sum
// ra: holds the sum
```

LDI - Load with direct immediate addressing

Description

Loads the value stored at the immediate address and stores it in the destination register.

Syntax

```
ldi rd imm
```

Operands

- `rd`: Destination register
- `imm`: Immediate address of the value

Operation

```
rd <- [imm]
```

Machine code

```
010011II IIIIIIII IIIIIIII IIIDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

```

```

LDP - Load with indirect addressing

Description

Reads the address of the pointer stored in the source register and loads the value at the address pointed by the register into the destination register.

Syntax

`ld rd rs`

Operands

- `rd`: Destination register
- `rs`: Source register holding the address

Operation

```
temp <- [rs]
rd <- [temp]
```

or

```
temp <- [rs]
rd <- [temp]
```

Machine code

`0010011? ???????? ??????SS SSSDDDDD`

Restrictions

Neither `rd` nor `rs` can be `sp` or `ip`

Example

```

```

MUL - Multiplication

Description

Multiplies the content of two registers without carry and stores the result in the destination register.

In case of overflow, the extra bits are discarded.

Syntax

`mul [rd] rs rt`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: First source register
- `rt`: Second source register

Operation

```
rd <- rs * rt
```

Machine code

`0000010? ???????? ?TTTTTSS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
mul rc ra rb // rc = ra * rb
mul rz sp // rz * sp
```

OR - Logical OR

Description

Performs a bitwise logical OR operation between two registers and stores the result in the destination register.

Syntax

`or [rd] rs rt`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: First source register
- `rt`: Second source register

Operation

```
rd <- rs | rt
```

Machine code

`0000100? ???????? ?TTTTTSS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
or rc ra rb // rc = ra | rb
or rz sp // rz |= sp
```

ORI - Logical OR with immediate

Description

Performs a bitwise logical OR operation between the source register and the immediate value and stores the result in the destination register.

Syntax

`or[i] [rd] rs imm`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: First source register
- `imm`: Immediate value

Operation

```
rd <- rs | imm
```

Machine code

`010100II IIIIIIII IIIIIISS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
ori rb ra 5 // rb = ra | 5
or rz 26 // rz |= 26
```

POP - Pop register from stack

Description

Loads the value stored at the address in the `sp` register into the destination register and increments the value of `sp` (aligned on 4 bytes).

Syntax

```
pop rd
```

Operands

- `rd`: Destination register

Operation

```
sp <- sp + 4
rd <- [sp]
```

Machine code

```
0010101? ???????? ???????? ???DDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

If an overflow occurs, the behavior is undefined and depends on hardware implementation.

Example

```
set ra 8
push ra // Keep for later
call overwrite
pop ra // Retrieve the value
exit

// A subroutine that would overwrite ra
overwrite:
set ra 5
ret
```

PUSH - Push register on stack

Description

Loads the value stored in the source register at the address in the `sp` register and decrements the value of `sp` (aligned on 4 bytes).

Syntax

```
push rs
```

Operands

- `rs`: Source register

Operation

```
[sp] <- rs
sp <- sp - 4
```

Machine code

```
0010100? ???????? ??????SS SSS?????
```

Restrictions

If the stack pointer goes past the stack bound (stack overflow), the behavior is undefined and depends on hardware implementation.

Example

```
set ra 8
push ra // Keep for later
call overwrite
pop ra // Retrieve the value
exit

// A subroutine that would overwrite ra
overwrite:
set ra 5
ret
```

RET - Return from subroutine

Description

Pops the absolute address stored on top of the stack and jumps to this address.

Syntax

`ret`

Operands

None

Operation

```
sp <- sp + 1
ip <- [sp]
```

Machine code

`1110???? ???????? ???????? ???????0`

Restrictions

`rd` cannot be `sp` or `ip`

If an overflow occurs, the behavior is undefined and depends on hardware implementation.

Example

SET - Set register to immediate

Description

Sets a register to an immediate value.

Syntax

```
set rd imm
```

Operands

- `rd`: Destination register
- `imm`: Immediate value

Operation

```
rd <- imm
```

Machine code

```
010111II IIIIIIII IIIIIIII IIIDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

```

```

STR - Store with direct addressing

Description

Reads the address stored in the destination register and writes the value in the source register at that address.

Syntax

```
str rd rs
```

Operands

- `rd`: Destination register holding the address
- `rs`: Source register

Operation

```
[rd] <- rs
```

Machine code

```
0010000? ???????? ??????SS SSSDDDDD
```

Restrictions

None

Example

```
// Writes the values from 0 to n-1 in memory
// rp: pointer to an array of integers
// rl: length of the array (>= 1)
set ra 0
loop:
str rp ra
addi rp rp 1
addi ra ra 1
tlt rt ra rl
jz loop
```

STRI - Store with direct immediate addressing

Description

Writes the value in the source register at the immediate address.

Syntax

```
stri rs imm
```

Operands

- `rs`: Source register
- `imm`: Immediate address of the value

Operation

```
[imm] <- rs
```

Machine code

```
010010II IIIIIIII IIIIIIII IIISSSSS
```

Restrictions

None

Example

```

```

STRP - Store with indirect addressing

Description

Reads the address of the pointer stored in the destination register and writes the value of the source register at the address pointed by the pointer.

Syntax

`strp rd rs`

Operands

- `rd`: Destination register holding the address
- `rs`: Source register

Operation

```
temp <- [rd]
[temp] <- rs
```

Machine code

`0010010? ???????? ??????SS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`

Example

SUB - Subtraction

Description

Subtracts the content of the second source register from the content of the first one without carry and stores the result in the destination register.

Syntax

`sub [rd] rs rt`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: Source register for the minuhend
- `rt`: Source register for the subtrahend

Operation

```
rd <- rs - rt
```

Machine code

`0000001? ???????? ?TTTTTSS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
sub rc ra rb // rc = ra - rb
sub rz sp // rz -= sp
```

SUBI - Subtraction with immediate

Subtracts the immediate value from the content of the source register without carry and stores the result in the destination register.

Syntax

`sub[i] [rd] rs imm`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: Source register for the minuhend
- `imm`: Subtrahend immediate value

Operation

```
rd <- rs - imm
```

Machine code

`010001II IIIIIIII IIIIIISS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
subi rb ra 5 // rb = ra - 5
sub rz -26 // rz -= -26
```

TEQ - Test if equal

Description

Performs a comparison between the values in both source registers and sets the destination register to 1 if the values are equal and 0 if not.

Syntax

```
teq rd rs rt
```

Operands

- `rd`: Destination register
- `rs`: First source register
- `rt`: Second source register

Operation

```
if rs == rt
  rd = 1
else
  rd = 0
```

Machine code

```
0001100? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

```

```

TEQI - Test if equal with immediate

Description

Performs a comparison between the value in the source register and the immediate value and sets the destination register to 1 if the values are equal and 0 if not.

Syntax

`teqi rd rs imm`

Operands

- `rd`: Destination register
- `rs`: Source register
- `imm`: Immediate value

Operation

```
if rs == imm
  rd = 1
else
  rd = 0
```

Machine code

`011100II IIIIIIII IIIIIISS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`

Example

TGE - Test if greater or equal

Description

Performs a comparison between the value in the source register and the immediate value and sets the destination register to 1 if the first value is greater or equal to the second one and 0 if not.

Syntax

```
tge rd rs rt
```

Operands

- `rd`: Destination register
- `rs`: First source register
- `rt`: Second source register

Operation

```
if rs >= rt
  rd = 1
else
  rd = 0
```

Machine code

```
0001011? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

TGEI - Test if greater or equal with immediate

Description

Performs a comparison between the values in both source registers and sets the destination register to 1 if the first value is greater or equal to the second one and 0 if not.

Syntax

`tgei rd rs imm`

Operands

- `rd`: Destination register
- `rs`: Source register
- `imm`: Immediate value

Operation

```
if rs == imm
  rd = 1
else
  rd = 0
```

Machine code

`011011II IIIIIIII IIIIIISS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`

Example

TGT - Test if strictly greater

Description

Performs a comparison between the value in the source register and the immediate value and sets the destination register to 1 if the first value is strictly greater than the second one and 0 if not.

Syntax

```
tgt rd rs rt
```

Operands

- `rd`: Destination register
- `rs`: First source register
- `rt`: Second source register

Operation

```
if rs > rt
  rd = 1
else
  rd = 0
```

Machine code

```
0001010? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

TGTI - Test if strictly greater with immediate

Description

Performs a comparison between the values in both source registers and sets the destination register to 1 if the first value is strictly greater than the second one and 0 if not.

Syntax

```
tgti rd rs imm
```

Operands

- `rd`: Destination register
- `rs`: Source register
- `imm`: Immediate value

Operation

```
if rs > imm
  rd = 1
else
  rd = 0
```

Machine code

```
011010II IIIIIIII IIIIIISS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

TLE - Test if lower or equal

Description

Performs a comparison between the value in the source register and the immediate value and sets the destination register to 1 if the first value is lower or equal to the second one and 0 if not.

Syntax

```
tle rd rs rt
```

Operands

- `rd`: Destination register
- `rs`: First source register
- `rt`: Second source register

Operation

```
if rs <= rt
  rd = 1
else
  rd = 0
```

Machine code

```
0001001? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

```

```

TLEI - Test if lower or equal with immediate

Description

Performs a comparison between the values in both source registers and sets the destination register to 1 if the first value is lower or equal to the second one and 0 if not.

Syntax

```
tlei rd rs imm
```

Operands

- `rd`: Destination register
- `rs`: Source register
- `imm`: Immediate value

Operation

```
if rs <= imm
  rd = 1
else
  rd = 0
```

Machine code

```
011001II IIIIIIII IIIIIISS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

TLT - Test if strictly lower

Description

Performs a comparison between the value in the source register and the immediate value and sets the destination register to 1 if the first value is strictly lower than the second one and 0 if not.

Syntax

```
tgt rd rs rt
```

Operands

- `rd`: Destination register
- `rs`: First source register
- `rt`: Second source register

Operation

```
if rs < rt
  rd = 1
else
  rd = 0
```

Machine code

```
0001000? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

```

```

TLTI - Test if strictly lower with immediate

Description

Performs a comparison between the values in both source registers and sets the destination register to 1 if the first value is strictly lower than the second one and 0 if not.

Syntax

```
tlti rd rs imm
```

Operands

- `rd`: Destination register
- `rs`: Source register
- `imm`: Immediate value

Operation

```
if rs < imm
  rd = 1
else
  rd = 0
```

Machine code

```
011000II IIIIIIII IIIIIISS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

TNE - Test if not equal

Description

Performs a comparison between the values in both source registers and sets the destination register to 1 if the values are different and 0 if not.

Syntax

```
tne rd rs rt
```

Operands

- `rd`: Destination register
- `rs`: First source register
- `rt`: Second source register

Operation

```
if rs != rt
  rd = 1
else
  rd = 0
```

Machine code

```
0001101? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

```

```

TNEI - Test if not equal with immediate

Description

Performs a comparison between the value in the source register and the immediate value and sets the destination register to 1 if the values are different and 0 if not.

Syntax

```
tnei rd rs imm
```

Operands

- `rd`: Destination register
- `rs`: Source register
- `imm`: Immediate value

Operation

```
if rs != imm
  rd = 1
else
  rd = 0
```

Machine code

```
011101II IIIIIIII IIIIIISS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`

Example

```

```

XCHG - Exchange registers

Description

Swaps the content of two registers.

Syntax

`xchg ra rb`

Operands

- `ra`: First register
- `rb`: Second register

Operation

```
interal <- ra
ra <- rb
rb <- internal
```

Machine code

`0010110? ???????? ??????BB BBBAAAAA`

Restrictions

`ra` and `rb` cannot be `sp` or `ip`.

Example

```
loopFibonacci:
add ra ra rb
xchg ra rb
jmp loopFibonacci
```

XOR - Logical XOR

Description

Performs a bitwise logical XOR operation between two registers and stores the result in the destination register.

Syntax

```
xor [rd] rs rt
```

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: First source register
- `rt`: Second source register

Operation

```
rd <- rs ^ rt
```

Machine code

```
0000110? ???????? ?TTTTTSS SSSDDDDD
```

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
xor rc ra rb // rc = ra ^ rb
xor rz sp // rz ^= sp
```

XORI - Logical XOR with immediate

Description

Performs a bitwise logical XOR operation between the source register and the immediate value and stores the result in the destination register.

Syntax

`xor[i] [rd] rs imm`

Operands

- `rd`: Optional destination register, defaults to `rs`
- `rs`: First source register
- `imm`: Immediate value

Operation

```
rd <- rs ^ imm
```

Machine code

`010110II IIIIIIII IIIIIISS SSSDDDDD`

Restrictions

`rd` cannot be `sp` or `ip`. Same condition applies to `rs` if no destination register is specified.

Example

```
xori rb ra 5 // rb = ra ^ 5
xor rz 26 // rz ^= 26
```