MIN16 Instruction Set

Masakazu Tanami (tanami@g.harvard.edu)

November 15, 2017

Preliminary Final Project Problem Set

EXECUTIVE SUMMARY

Introduction

This document defines the design for the MIN16 Instruction Set.

Overview

The MIN16 Instruction Set is a part of 16-bit RISC CPU architecture to define components of the instructions. The components include data types, resisters, addressing modes and functions. The MIN16 Instruction Set is responsible for providing information for assembler language programmers.

PROGRAMMER'S MODEL

Data Types

The MIN16 Instruction Set uses the following data types. In this architecture, a full word is defined as 16 bits.

- Byte - 8 bits
- Word - 16 bits

Registers

The following 16 registers are used to store data in the register file including a program counter.

- General Purpose Register - Accessible as an argument of functions.

```
        r0, at, sp, fp, ra, rb, rc, rd, s0, s1, t0, t1, hi, lo,
```

- Special Purpose Register - Access is limited. It is only accessible via instructions.

```
        pc, fl
```

MIN16 INSTRUCTION SET

Register List

This list defines register's properties and its Usage. Saved register's value is preserved across function calls. Purpose is either G (general) or S (special). The general purpose register consists an array.

Number	Name	Use	Saved	Purpose
0	`r0`	The constant value 0	N/A	G
1	`at`	Assembler Temporary	No	G
2	`sp`	Stack Pointer	Yes	G
3	`fp`	Frame Pointer	Yes	G
4	`ra`	Return Address	Yes	G
5	`rb`	General Use	No	G
6	`rc`	General Use	No	G
7	`rd`	General Use	No	G
8	`s0`	Saved Temporary	Yes	G
9	`s1`	Saved Temporary	Yes	G
10	`t0`	Temporary	No	G
11	`t1`	Temporary	No	G
12	`hi`	High Part	No	G
13	`lo`	Low Part	No	G
14	`pc`	Program Counter	N/A	S
15	`fl`	Status Flag	N/A	S

Status Flag Special Register

This register is used to keep status information in the circuit of the processor. Those are used to detect errors such as Overflow, Underflow, Carry(over), Carry(borrow).

Bit Number	Abbreviation	Description	Category
0	ZF	Zero Flag	Status
1	SF	Sign Flag	Status
2	CF	Carry Flag	Status
3	OF	Overflow Flag	Status
4	EQ	Equal Flag	Status

MIN16 INSTRUCTION SET

Operation Group

Instruction set is divided into four operation groups. An operation group consists of four kinds of instructions represented by operation codes. Therefore it is identified by the top two bits of the operation code. The following describes top two bits, kind of instructions, and type of addressing mode.

- 0b00: Arithmetic, Logical, Shift (R-Type)
- 0b01: Immediate Arithmetic, Logical, Shift (I-Type)
- 0b10: Jump/Branch Flow Control (O, R, J - Type)
- 0b11: Register/Memory Data Move (O, R - Type)

Operation Code

This list describes operation codes (binary & hex) and instructions. The hex opcode distinguishes those instructions. $\text{Opcode}_{\text{HEX}}$ (0x8, 0x9, 0xE, 0xF) are reserved for extension.

opcode	$\text{opcode}_{\text{HEX}}$	Instructions
0000	0	Signed Arithmetic
0001	0x1	Unsigned Arithmetic
0010	0x2	Logical
0011	0x3	Shift
0100	0x4	Immediate Signed Arithmetic
0101	0x5	Immediate Unsigned Arithmetic
0110	0x6	Immediate Logical
0111	0x7	Immediate Shift
1000	0x8	Reserved
1001	0x9	Reserved
1010	0xA	Jump
1011	0xB	Branch
1100	0xC	Memory Load and Store
1101	0xD	Move HI/LO Data
1110	0xE	Reserved
1111	0xF	Reserved

MIN16 INSTRUCTION SET

Addressing Modes

Instructions takes arguments differently, and use four different addressing modes. The following addressing modes are defined by Type, Structure, and Format.

R-type (Two Registers)

[op] [func] [rd] [rs] [sbz] *

```
0000 00 0000 0000 00
```

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---------------+-------+---------------+---------------+-------+
|      OP       | FUNC  |      RD       |      RS       |  SBZ  |
|     4bits     | 2bits |     4bits     |     4bits     | 2bits |
+---------------+-------+---------------+---------------+-------+
```

I-type (One Register, One Immediate)

[op] [func] [rd] [imm] **

```
0100 00 0000 000000
```

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---------------+-------+---------------+-----------------------+
|      OP       | FUNC  |      RD       |       IMMEDIATE       |
|     4bits     | 2bits |     4bits     |        6 bits         |
+---------------+-------+---------------+-----------------------+
```

O-type (Two Registers, One Offset)

[op] [func] [rd] [rs] [offset]

```
1000 00 000 000 0000
```

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---------------+-------+-----------+-----------+---------------+
|      OP       | FUNC  |    RD     |    RS     |    OFFSET     |
|     4bits     | 2bits |   3bits   |   3bits   |    4bits      |
+---------------+-------+-----------+-----------+---------------+
```

J-type (Jump to Target Address)

[op] [func] [target]

```
1000 00 0000000000
```

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---------------+-------+---------------------------------------+
|      OP       | FUNC  |                TARGET                 |
|     4bits     | 2bits |                10 bits                |
+---------------+-------+---------------------------------------+
```

* [op] operation code, [func] function code, [rd] destination register, [rs] source register, [sbz] should be zero, 0000 represents a bit group.

** [imm] immediate signed numeric value, [offset] offset numeric signed value, [target] target unsigned numeric value.

MIN16 INSTRUCTION SET

Instruction Mnemonics

In combination with operation code and function code, 46 instructions are identified. Mnemonic represents the operation of the instruction.

ALU R-Type Instructions (10)

```
ADD  : Add
ADDU : Add Unsigned
SUB  : Subtract
SUBU : Subtract Unsigned
SLT  : Set on Less Than
SLTU : Set on Less Than Unsigned
AND  : Bitwise Logical AND
OR   : Bitwise Logical OR
XOR  : Bitwise Logical XOR
NOR  : Bitwise Logical NOR
```

Multiply R-Type Instructions (2)

```
MUL  : Multiply
MULU : Multiply Unsigned
```

Shift R-Type Instructions (4)

```
SLL  : Shift Left Logical
SRL  : Shift Right Logical
SRA  : Shift Right Arithmetic
ROTL : Rotate Left
```

ALU I-Type Instructions (10)

```
ADDI : Add Immediate
ADDIU: Add Immediate Unsigned
SUBI : Subtract Immediate
SUBIU: Subtract Immediate Unsigned
SLTI : Set on Less Than Immediate
SLTIU: Set on Less Than Immediate Unsigned
ANDI : Bitwise Logical AND Immediate
ORI  : Bitwise Logical OR Immediate
XORI : Bitwise Logical XOR Immediate
NORI : Bitwise Logical NOR Immediate
```

Multiply I-Type Instructions (2)

```
MULI : Multiply Immediate
MULIU: Multiply Immediate Unsigned
```

Shift I-Type Instructions (4)

```
SLLI : Shift Left Logical Immediate
SRLI : Shift Right Logical Immediate
SRAI : Shift Right Arithmetic Immediate
ROTLI: Rotate Left Immediate
```

MIN16 INSTRUCTION SET

Jump J-Type Instructions (2)

```
J    : Jump
JAL  : Jump And Link
```

Jump R-Type Instructions (2)

```
JR   : Jump Register
JALR : Jump And Link Register
```

Branch O-Type Instructions (2)

```
BEQ  : Branch on Equal
BNE  : Branch on Not Equal
```

Load and Store O-Type Instructions (4)

```
LW   : Load Word
LB   : Load Byte
SW   : Store Word
SB   : Store Byte
```

Move R-type Instructions (4)

```
MFHI : Move from HI
MFLO : Move from LO
MTHI : Move to HI
MTLO : Move to LO
```

Instruction Mnemonic map

Mode	op \ func	0b00	0b01	0b10	0b11	
R-type	0x0	ADD	SUB	MUL	SLT	Signed Arithmetic
	0x1	ADDU	SUBU	MULU	SLTU	Unsigned Arithmetic
	0x2	AND	OR	XOR	NOR	Logical
	0x3	SLL	SRL	SRA	ROTL	Shift
I-type	0x4	ADDI	SUBI	MULI	SLTI	Immediate Signed Arithmetic
	0x5	ADDIU	SUBIU	MULIU	SLTIU	Immediate Unsigned Arithmetic
	0x6	ANDI	ORI	XORI	NORI	Immediate Logical
	0x7	SLLI	SRLI	SLAI	ROTLI	Immediate Shift
	0x8					Reserved
	0x9					Reserved
J-type	0xA	J	JAL	JR	JALR	Jump
O-type	0xB	BEQ	BNE			Branch
	0xC	LW	LB	SW	SB	Memory Load and Store
	0xD	MFHI	MFLO	MTHI	MTLO	Move HI/LO Data
	0xE					Reserved
	0xF					Reserved

Mnemonics are mapped by operation codes (4 bits) and function codes (2 bits), then colored by addressing modes.

MIN16 INSTRUCTION SET

ALU Function Code

Arithmetic, Logical, Shift operation is performed at ALU, Multiplier, and Shifter. Those unit uses function codes to specify which operation should be performed. The code is combination of the latter two bits of the operation code and the function code.

ALU Function Code	Instruction
0000	Add
0001	Subtract
0010	Multiply
0011	Set on Less Than
0100	Add Unsigned
0101	Subtract Unsigned
0110	Multiply Unsigned
0111	Set on Less Than Unsigned
1000	Bitwise Logical AND
1001	Bitwise Logical OR
1010	Bitwise Logical Exclusive-OR
1011	Bitwise Logical NOR
1100	Shift Left Logical
1101	Shift Right Logical
1110	Shift Right Arithmetic
1111	Rotate Left

THE MIN16 INSTRUCTION SET

This section specifies the detailed information of instructions. Each subgroup contains instructions ordered by function code. Operation is explained by C language like expression. In the example, `ra(0100)` is used for `rd`, `r0(0000)` is used for `rs`. The numeric value of 0 is used for immediate, offset, and target.

Operation Group 1

0b00: Arithmetic, Logical, Shift (R-type)

Add (R-type)

Take the contents of two general registers and apply '+' operand. The result is placed in general register rd.

```
Format:      ADD $rd, $rs
Operation:   $rd = $rd + $rs
Example:     0000 00 0100 0000 00 (0100: ADD $ra, $r0)
```

Subtract (R-type)

Take the contents of two general registers and apply '-' operand. The result is placed in general register rd.

```
Format:      SUB $rd, $rs
Operation:   $rd = $rd - $rs
Example:     0000 01 0100 0000 00 (0500: SUB $ra, $r0)
```

Multiply (R-type)

Take the contents of two general registers and apply '*' operand. The result is placed in general register rd.

```
Format:      MUL $rd, $rs
Operation:   $rd = $rd * $rs
Example:     0000 10 0100 0000 00 (0900: MUL $ra, $r0)
```

Set on Less Than (R-type)

Take the contents of two general registers and apply '<' operand. The result either 1 or 0 is placed in general register rd.

```
Format:      SLT $rd, $rs
Operation:   $rd = $rd < $rs ? 1 : 0
Example:     0000 11 0100 0000 00 (0D00: SLT $ra, $r0)
```

Add Unsigned (R-type)

Take the contents of two general registers and apply '+' operand. The result is placed in general register rd.

```
Format:       ADDU $rd, $rs
Operation:    $rd = $rd + $rs
Example:      0001 00 0100 0000 00 (0x1100: ADDU $ra, $r0)
```

Subtract Unsigned (R-type)

Take the contents of two general registers and apply '-' operand. The result is placed in general register rd.

```
Format:       SUBU $rd, $rs
Operation:    $rd = $rd - $rs
Example:      0001 01 0100 0000 00 (0x1500: SUBU $ra, $r0)
```

Multiply Unsigned (R-type)

Take the contents of two general registers and apply '*' operand. The result is placed in general register rd.

```
Format:       MULU $rd, $rs
Operation:    $rd = $rd * $rs
Example:      0001 10 0100 0000 00 (0x1900: MULU $ra, $r0)
```

Set on Less Than Unsigned (R-type)

Take the contents of two general registers and apply '<' operand. The result either 1 or 0 is placed in general register rd.

```
Format:       SLTU $rd, $rs
Operation:    $rd = $rd < $rs ? 1 : 0
Example:      0001 11 0100 0000 00 (0x1D00: SLTU $ra, $r0)
```

Bitwise Logical AND (R-type)

Take the contents of two general registers and apply '&' operand. The result is placed in general register rd.

```
Format:      AND $rd, $rs
Operation:   $rd = $rd & $rs
Example:     0010 00 0100 0000 00 (0x2100: AND $ra, $r0)
```

Bitwise Logical OR (R-type)

Take the contents of two general registers and apply '|' operand. The result is placed in general register rd.

```
Format:      OR $rd, $rs
Operation:   $rd = $rd | $rs
Example:     0010 01 0100 0000 00 (0x2500: OR $ra, $r0)
```

Bitwise Logical XOR (R-type)

Take the contents of two general registers and apply '^' operand. The result is placed in general register rd.

```
Format:      XOR $rd, $rs
Operation:   $rd = $rd ^ $rs
Example:     0010 10 0100 0000 00 (0x2900: XOR $ra, $r0)
```

Bitwise Logical NOR (R-type)

Take the contents of two general registers and apply '|' operand and flip all bits. The result is placed in general register rd.

```
Format:      NOR $rd, $rs
Operation:   $rd = ~($rd | $rs)
Example:     0010 11 0100 0000 00 (0x2D00: NOR $ra, $r0)
```

MIN16 INSTRUCTION SET

Shift Left Logical (R-type)

Take the contents of two general registers and apply ‘<<’ operand. The result is placed in general register rd.

```
Format:      SLL $rd, $rs
Operation:   $rd = $rd << $rs
Example:     0011 00 0100 0000 00 (0x3100: SLL $ra, $r0)
```

Shift Right Logical (R-type)

Take the contents of two general registers and apply ‘>>’ operand, inserting zeros into the high order bits. The result is placed in general register rd.

```
Format:      SRL $rd, $rs
Operation:   $rd = $rd >> $rs
Example:     0011 01 0100 0000 00 (0x3500: SRL $ra, $r0)
```

Shift Right Arithmetic (R-type)

Take the contents of two general registers and apply ‘>>’ operand, sign-extending the high order bit. The result is placed in general register rd.

```
Format:      SRA $rd, $rs
Operation:   $rd = ($rd >> $rs) || 0x8000
Example:     0011 10 0100 0000 00 (0x3900: SRA $ra, $r0)
```

Rotate Left (R-type)

Take the contents of two general registers and apply ‘<<’ operand, then OR with bits that is overflowed. The result is placed in general register rd. This is a novel feature, therefore discussed in the later section.

```
Format:      ROTL $rd, $rs
Operation:   $rd = ($rd << $rs) || ($rd >> (16 - $rs))
Example:     0011 11 0100 0000 00 (0x3D00: ROTL $ra, $r0)
```

Operation Group 2

0b01: Immediate (I-type)

Add Immediate (I-type)

Take the contents of the general register and apply '+' operand to immediate. The result is placed in general register rd. Immediate signed field has 6 bits and will be sign extended.

```
Format:       ADDI $rd, imm
Operation:    $rd = $rd + imm
Example:      0100 00 0100 000000 (0x4100: ADDI $ra, 0)
```

Subtract Immediate (I-type)

Take the contents of the general register and apply '-' operand to immediate. The result is placed in general register rd. Immediate signed field has 6 bits and will be sign extended.

```
Format:       SUBI $rd, imm
Operation:    $rd = $rd - imm
Example:      0100 01 0100 000000 (0x4500: SUBI $ra, 0)
```

Multiply Immediate (I-type)

Take the contents of the general register and apply '*' operand to immediate. The result is placed in general register rd. Immediate signed field has 6 bits and will be sign extended.

```
Format:       MULI $rd, imm
Operation:    $rd = $rd * imm
Example:      0100 10 0100 000000 (0x4900: MULI $ra, 0)
```

Set on Less Than Immediate (I-type)

Take the contents of the general register and apply '<' operand to immediate. The result either 1 or 0 is placed in general register rd. Immediate signed field has 6 bits and will be sign extended.

```
Format:       SLTI $rd, imm
Operation:    $rd = $rd < imm ? 1 : 0
Example:      0100 11 0100 000000 (0x4D00: SLTI $ra, 0)
```

Add Immediate Unsigned (I-type)

Take the contents of the general register and apply '+' operand to immediate. The result is placed in general register rd. Immediate unsigned field has 6 bits and will be zero extended.

```
Format:       ADDIU $rd, imm
Operation:    $rd = $rd + imm
Example:      0101 00 0100 000000 (0x5100: ADDIU $ra, 0)
```

Subtract Immediate Unsigned (I-type)

Take the contents of the general register and apply '-' operand to immediate. The result is placed in general register rd. Immediate unsigned field has 6 bits and will be zero extended.

```
Format:       SUBIU $rd, imm
Operation:    $rd = $rd - imm
Example:      0101 01 0100 000000 (0x5500: SUBIU $ra, 0)
```

Multiply Immediate Unsigned (I-type)

Take the contents of the general register and apply '*' operand to immediate. The result is placed in general register rd. Immediate unsigned field has 6 bits and will be zero extended.

```
Format:       MULIU $rd, imm
Operation:    $rd = $rd * imm
Example:      0101 10 0100 000000 (0x5900: MULIU $ra, 0)
```

Set on Less Than Immediate Unsigned (I-type)

Take the contents of the general register and apply '<' operand to immediate. The result either 1 or 0 is placed in general register rd. Immediate unsigned field has 6 bits and will be zero extended.

```
Format:       SLTIU $rd, imm
Operation:    $rd = $rd < imm ? 1 : 0
Example:      0101 11 0100 000000 (0x5D00: SLTIU $ra, 0)
```

Bitwise Logical AND Immediate (I-type)

Take the contents of the general register and apply ‘&’ operand to immediate. The result is placed in general register rd. Immediate logical field has 6 bits and will be zero extended.

```
Format:       ANDI $rd, imm
Operation:    $rd = $rd & imm
Example:      0110 00 0100 000000 (0x6100: ANDI $ra, 0)
```

Bitwise Logical OR Immediate (I-type)

Take the contents of the general register and apply ‘|’ operand to immediate. The result is placed in general register rd. Immediate logical field has 6 bits and will be zero extended.

```
Format:       ORI $rd, imm
Operation:    $rd = $rd | imm
Example:      0110 01 0100 000000 (0x6500: ORI $ra, 0)
```

Bitwise Logical XOR Immediate (I-type)

Take the contents of the general register and apply ‘^’ operand to immediate. The result is placed in general register rd. Immediate logical field has 6 bits and will be zero extended.

```
Format:       MULI $rd, imm
Operation:    $rd = $rd ^ imm
Example:      0110 10 0100 000000 (0x6900: XOR $ra, 0)
```

Bitwise Logical NOR Immediate (I-type)

Take the contents of the general register and apply ‘|’ operand to immediate, then flip all bits. The result is placed in general register rd. Immediate logical field has 6 bits and will be zero extended.

```
Format:       SLTI $rd, imm
Operation:    $rd = ~($rd | imm)
Example:      0110 11 0100 000000 (0x6D00: SLTI $ra, 0)
```

Shift Left Logical Immediate (I-type)

Take the contents of the general register and apply '<<' operand to immediate. The result is placed in general register rd. Immediate logical field has 6 bits and will be zero extended.

```
Format:       SLLI $rd, imm
Operation:    $rd = $rd << imm
Example:      0111 00 0100 000000 (0x7100: SLLI $ra, 0)
```

Shift Right Logical Immediate (I-type)

Take the contents of the general register and apply '>>' operand to immediate, inserting zeros into the high order bits. The result is placed in general register rd. Immediate logical field has 6 bits and will be zero extended.

```
Format:       SRLI $rd, imm
Operation:    $rd = $rd >> imm
Example:      0111 01 0100 000000 (0x7500: SRLI $ra, 0)
```

Shift Right Arithmetic Immediate (I-type)

Take the contents of the general register and apply '>>' operand to immediate, sign-extending the high order bit. The result is placed in general register rd. Immediate signed field has 6 bits and will be sign extended.

```
Format:       SRAI $rd, imm
Operation:    $rd = ($rd >> imm) || 0x8000
Example:      0111 10 0100 000000 (0x7900: SRAI $ra, 0)
```

Rotate Left Immediate (I-type)

Take the contents of the general register and apply '<<' operand to immediate, then OR with bits that is overflowed. The result is placed in general register rd. This is a novel feature, therefore discussed in the later section. Immediate logical field has 6 bits and will be zero extended.

```
Format:       ROTLI $rd, imm
Operation:    $rd = ($rd << imm) || ($rd >> (16 - imm))
Example:      0111 11 0100 000000 (0x7D00: ROTLI $ra, 0)
```

MIN16 INSTRUCTION SET

Operation Group 3

0b11: Jump/Branch Flow Control (O, R, J - type)

Jump (J-type)

Jump to address within 10 bit range. Target is an unsigned number of bytes (not shifted). If target is greater than 0x3ff, the assembler should store the number to a register and use JR instruction.

```
Format:       J target
Operation:    $pc = target
Example:      0110 00 01 0000 0000 (0xA100: J 0x100)
```

Jump And Link (J-type)

Jump to address with the same way as Jump instruction and store the next instruction address of program counter ($pc + 2) to return address.

```
Format:       JAL target
Operation:    $ra = $pc + 2, $pc = target
Example:      0110 01 01 0000 0000 (0xA500: JAL 0x100)
```

Jump Register (R-type)

Jump to the address in the general register.

```
Format:       JR $rd
Operation:    $pc = $rd
Example:      0110 10 0100 000000 (0xA900: JR 0x4)
```

Jump And Link Register (R-type)

Jump to the address in the general register and store the program counter to the general register.

```
Format:       JALR $rd, $rs
Operation:    $rs = $pc, $pc = $rd
Example:      0110 11 0100 0000 00 (0xAD00: JALR $ra, $r0)
```

Branch on Equal (O-type)

If two values in the general registers are equal, go to the target address computed from the program counter and offset. Offset is a signed number of words, hence arithmetic shifted by 1.

```
Format:      BEQ $rd, $rs, offset
Operation:   $rd == $rs ? $pc = $pc + (offset << 1) : (Do nothing)
Example:     1011 00 100 000 0000 (0xB200: BEQ $ra, $r0, 0)
```

Branch on Not Equal (O-type)

If two values in the general registers are not equal, go to the target address computed from the program counter and offset. Offset is a signed number of words, hence arithmetic shifted by 1.

```
Format:      BNE $rd, $rs, offset
Operation:   $rd != $rs ? $pc = $pc + (offset << 1) : (Do nothing)
Example:     1011 01 100 000 0000 (0xB600: BNE $ra, $r0, 0)
```

MIN16 INSTRUCTION SET

Operation Group 4

0b11: Register / Memory Data Move (O, R - type)

Load Word (O-type)

Load a word from the address computed by the source general register and offset. Offset is a signed number of words, hence arithmetic shifted by 1.

```
Format:      LW $rd, $rs, offset
Operation:   $rd = $($rs + (offset << 1))
Example:     1100 00 100 000 0000 (0xC200: LW $ra, $r0, 0)
```

Load Byte (O-type)

Load a byte from the address computed by the source general register and offset. Offset is a signed number of words, hence arithmetic shifted by 1.

```
Format:      LB $rd, $rs, offset
Operation:   $rd = $($rs + (offset << 1)) & 00FF
Example:     1100 01 100 000 0000 (0xC600: LB $ra, $r0, 0)
```

Store Word (O-type)

Store a word to the address computed by the destination general register and offset. Offset is a signed number of words, hence arithmetic shifted by 1.

```
Format:      SW $rd, $rs, offset
Operation:   $($rd + (offset << 1)) = $rs
Example:     1100 10 100 000 0000 (0xCA00: SW $ra, $r0, 0)
```

Store Byte (O-type)

Store a byte to the address computed by the destination general register and offset. Offset is a signed number of words, hence arithmetic shifted by 1. Byte write mode on the memory should be prepared to write only the 8 bits.

```
Format:      SB $rd, $rs, offset
Operation:   $($rd + (offset << 1)) & 00FF = $rs & 00FF
Example:     1100 11 100 000 0000 (0xCE00: SB $ra, $r0, 0)
```

(Note: The instructions on this page are not useful until 32bit multiplier is implemented.)

Move From HI (R-type, Pseudo)

Move data in HI special register to the destination general purpose register. This pseudo instruction performs multiple instructions.

```
Format:      MFHI $rd
Operation:   $rd = $HI
Instruction: AND $rd, $r0
             ADD $rd, 12
Example:     1101 00 0100 0000 00 (0xD100: MFHI $ra)
```

Move From LO (R-type, Pseudo)

Move data in LO special register to the destination general purpose register. This pseudo instruction performs multiple instructions.

```
Format:      MFLO $rd
Operation:   $rd = $LO
Instruction: AND $rd, $r0
             ADD $rd, 13
Example:     1101 01 0100 0000 00 (0xD500: MFLO $ra)
```

Move To HI (R-type, Pseudo)

Move data to HI special register from the destination general purpose register. This pseudo instruction performs multiple instructions.

```
Format:      MTHI $rd
Operation:   $HI = $rd
Instruction: AND 12, $r0
             ADD 12, $rd
Example:     1101 10 0100 0000 00 (0xD900: MTHI $ra)
```

Move To LO (R-type, Pseudo)

Move data to LO special register from the destination general purpose register. This pseudo instruction performs multiple instructions.

```
Format:      MTLO $rd
Operation:   $LO = $rd
Instruction: AND 13, $r0
             ADD 13, $rd
Example:     1101 11 0100 0000 00 (0xDD00: MTLO $ra)
```

MIN16 INSTRUCTION SET

Instruction Set Summary

This summary is a list of 46 instructions to provide overview of the assembly language syntax.

```
[opcode] [function code] [directive] [Hex value of function code] [format]

00: R-Type (Arithmetic, Logical, Shift)
0000 00 ADD   0 0        ADD $rd, $rs
0000 01 SUB   0 1        SUB $rd, $rs
0000 10 MUL   0 2        MUL $rd, $rs
0000 11 SLT   0 3        SLT $rd, $rs

0001 00 ADDU  0x1 0        ADDU $rd, $rs
0001 01 SUBU  0x1 1        SUBU $rd, $rs
0001 10 MULU  0x1 2        MULU $rd, $rs
0001 11 SLTU  0x1 3        SLTU $rd, $rs

0010 00 AND   0x2 0        AND $rd, $rs
0010 01 OR    0x2 1        OR  $rd, $rs
0010 10 XOR   0x2 2        XOR $rd, $rs
0010 11 NOR   0x2 3        NOR $rd, $rs

0011 00 SLL   0x3 0        SLL $rd, $rs
0011 01 SRL   0x3 1        SRL $rd, $rs
0011 10 SLA   0x3 2        SLA $rd, $rs
0011 11 ROTL  0x3 3        ROT $rd, $rs

01: I-Type (Immediate)
0100 00 ADDI  0x4 0        ADDI $rd, imm
0100 01 SUBI  0x4 1        SUBI $rd, imm
0100 10 MULI  0x4 2        MULI $rd, imm
0100 11 SLTI  0x4 3        SLTI $rd, imm

0101 00 ADDIU 0x5 0        ADDIU $rd, imm
0101 01 SUBIU 0x5 1        SUBIU $rd, imm
0101 10 MULIU 0x5 2        MULIU $rd, imm
0101 11 SLTIU 0x5 3        SLTIU $rd, imm

0110 00 ANDI  0x6 0        ANDI $rd, imm
0110 01 ORI   0x6 1        ORI  $rd, imm
0110 10 XORI  0x6 2        XORI $rd, imm
0110 11 NORI  0x6 3        NORI $rd, imm

0111 00 SLLI  0x7 0        SLLI $rd, imm
0111 01 SRLI  0x7 1        SRLI $rd, imm
0111 10 SRAI  0x7 2        SRAI $rd, imm
0111 11 ROTLI 0x7 3        ROTI $rd, imm

10 : ORJ-Type (Jump/Branch Flow Control)
1010 00 J     0xA 0        J    target   (J-type)
1010 01 JAL   0xA 1        JAL  target   (J-type)
1010 10 JR    0xA 2        JR   $rd      (R-type)
1010 11 JALR  0xA 3        JALR $rd, $rs (R-type)

1011 00 BEQ   0xB 0        BEQ  $rd, $rs, offset (O-type)
1011 01 BNE   0xB 1        BNE  $rd, $rs, offset (O-type)


11 : OR-TYPE (Register/Memory Data Move)
1100 00 LW    0xC 0        LW $rd, $rs, offset (O-type)
1100 01 LB    0xC 1        LB $rd, $rs, offset (O-type)
1100 10 SW    0xC 2        SW $rd, $rs, offset (O-type)
1100 11 SB    0xC 3        SB $rd, $rs, offset (O-type)

1101 00 MFHI  0xD 0        MFHI $rd (R-type)
1101 01 MFLO  0xD 1        MFLO $rd (R-type)
1101 10 MTHI  0xD 2        MTHI $rd (R-type)
1101 11 MTLO  0xD 3        MTLO $rd (R-type)
```

NOVEL FEATURE

This section describes the novel feature and its algorithm of implementation.

Barrel Shifter

The Rotate Left (R-type) and Rotate Left Immediate (I-type) instruction allows multiple bit shifts specified by either a general purpose register or a fixed immediate value.

The problem of multiple bit shift is the cost of execution. If a simple sifter is used, n-bit shift require n clock cycles. The barrel shifter aims to perform this operation within a single clock cycle.

Algorithm

In order to operate quickly, divide and conquer algorithm is used.

Consider an array:

```
abcdefgh
```

Apply Rotate Left by 3 letters, then the array will be as follows:

```
defghabc
```

First, split the original array as follows:

```
abc, defgh
```

Second, we apply merge sort algorithm.

```
cba, hgfed
```

Third, combine them and apply merge sort algorithm again.

```
defghabc
```

Cost Analysis

In a merge sort of n bits, the number of bit-by-bit sort is n/2 times, and the number of division is log(n). Therefore the cost per merge sort is n/2*log(n). For Barrel Shifter using two merge sorts, the cost is n*log(n).

Hardware Merge Sort

On each base case, bit-by-bit sort is performed. In hardware, this is done by using multiplexer. Therefore the Barrel Shifter requires n*log(n) numbers of multiplexer. In the MIN16 architecture, 64 multiplexers are needed to compose the Barrel Shifter. By combining multiplexers, we can achieve n-bit shift within a single clock cycle.