RISC-Toy Manual

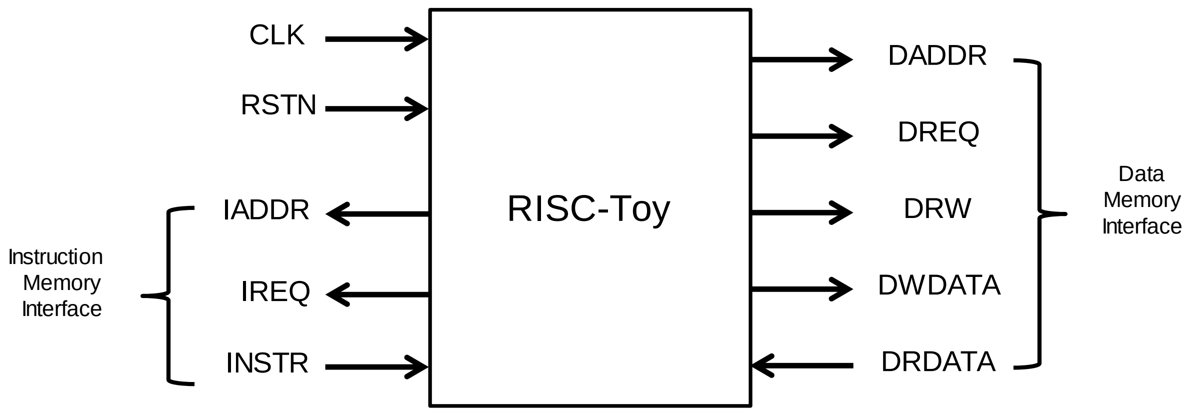
EE361 Spring 2025

Digital System Design Lab

Electronic Engineering, Kyung Hee University

1. Interface of RISC-Toy

Top module



Components

- **Special Purpose Registers**

  - PC<31..0> (Program Counter)

  PC holds the address to fetch the next instruction. It increases by 4 after an instruction is fetched. Instructions that are allowed to access or modify the PC are J, JL, BR, BRL.

  - IR<31..0> (Instruction Register)

- **General-Purpose Registers**

  The processor has 32 general-purpose registers, all 32-bit long.

- **Memory**

  The processor has separate address spaces for instruction program and data memory. The instruction memory can only be read (written only once at the initial step) while the data memory can be read and written. The size of the word bit is 32 bits. We assume that they are byte-addressable, but there does not exist any instruction that can be accessed in byte-wise manner (always 32 bits with difference of 4 of the address).

2. Instruction Sets

Instruction	**OPCODE**	**Instruction**	**OPCODE**
ADD	0	**JL**	16
ADDI	1	**BR**	17
SUB	2	**BRL**	18
NEG	3	**ST**	19
NOT	4	**STR**	20
AND	5	**LD**	21
ANDI	6	**LDR**	22
OR	7		
ORI	8		
XOR	9		
LSR	10		
ASR	11		
SHL	12		
ROR	13		
MOVI	14		
J	15		

Operator Meaning

+	Add	**<**	Smaller than
-	Subtract or 2's complement	**>=**	Greater than or equal
&	Logical AND	**<=**	Smaller than or equal
|	Logical OR	**!=**	Not equal
^	Logical exclusive-OR	**<<**	Shift left
~	Logical inversion	**>>**	Shift right
=	Substitution	**signExt()**	Sign-extension to 32 bits
==	Equal to	**zeroExt()**	Zero-extension to 32 bits
>	Greater than		

Bit Notation in the Operation Description

E.g. R[4]<8> : The bit 8 of register 4.

M[1020]<4:0> : From bit 4 to bit 0 of the memory word at address 1020

{n{R[3]}} : The bit 29 of register 3 is repeated n times.

{a<3..0>, b<7..4>} : Concatenation of lower nibble of a and higher nibble of b.

Reset Procedure - When RSTN goes low (active low),

General-purpose registers are set to zero. (R[0..31] ← 00000000h)

PC is set to zero. (PC ← 00000000h)

ADD

ASSEMBLY

ADD ra, rb, rc

DESCRIPTION

The ADD instruction performs a 2's complement signed addition where both operands, rb and rc, are general-purpose register values. The result of the addition is stored in a general-purpose register, ra.

FORMAT

31 27	26 22	21 17	16 12	11 0
0 0 0 0 0	ra	rb	rc	unused

OPERATION

```
R[ra] = R[rb] + R[rc];
```

EXAMPLE

ADD r1, r2, r3

ADDI

ASSEMBLY

ADDI ra, rb, imm17

DESCRIPTION

The ADDI (add immediate) instruction performs a 2’s complement signed addition where one operand is a general-purpose register value and the other is a 32-bit immediate value. Since the addition is signed, the shorter operand is sign-extended to match the bit-length of longer one. The result of the addition is stored in a general-purpose register, ra.

FORMAT

31 27	26 22	21 17	16 0
0 0 0 0 1	ra	rb	Imm17

OPERATION

```
R[ra] = R[rb] + signExt(imm17);
```

EXAMPLE

ADDI r1, r2, #2

SUB

ASSEMBLY

SUB ra, rb, rc

DESCRIPTION

The SUB (subtract) instruction performs a 2's complement signed subtraction where both operands, rb and rc, are general-purpose register values. The result of the subtraction is stored in a general-purpose register, ra. rc is subtracted from rb.

FORMAT

31 27	26 22	21 17	16 12	11 0
0 0 0 1 0	ra	rb	rc	unused

OPERATION

```
R[ra] = R[rb] - R[rc];
```

EXAMPLE

SUB r1, r2, r3

NEG

ASSEMBLY

NEG ra, rc

DESCRIPTION

The NEG (negate) instruction performs a 2's complement negation where the operand rc is a general-purpose register value. The result of the negation is stored in a general-purpose register, ra.

FORMAT

31 27	26 22	21 17	16 12	11 0
0 0 0 1 1	ra	unused	rc	unused

OPERATION

```
R[ra] = -R[rc];
```

EXAMPLE

NEG r5, r6

NOT

ASSEMBLY

NOT ra, rc

DESCRIPTION

The NOT instruction performs a 1’s complement inversion where the operand rc is a general-purpose register value. The result of the operation is stored in a general-purpose register, ra.

FORMAT

31 27	26 22	21 17	16 12	11 0
0 0 1 0 0	ra	unused	rc	unused

OPERATION

```
R[ra] = ~R[rc];
```

EXAMPLE

NOT r15, r8

AND

ASSEMBLY

AND ra, rb, rc

DESCRIPTION

The AND instruction performs a logical bit-wise and operation where both operands, rb and rc, are general-purpose register values. The result of the logical operation is stored in a general-purpose register, ra.

FORMAT

31 27	26 22	21 17	16 12	11 0
0 0 1 0 1	ra	rb	rc	unused

OPERATION

```
R[ra] = R[rb] & R[rc];
```

EXAMPLE

AND r1, r2, r3

ANDI

ASSEMBLY

ANDI ra, rb, imm17

DESCRIPTION

The ANDI (and immediate) instruction performs a logical bit-wise AND operation where one operand is a general-purpose register value and the other is a 32-bit immediate value. The immediate operand is sign-extended to match the bit-length of the GPR operand. The result is stored in a general-purpose register, ra.

FORMAT

31 27	26 22	21 17	16 0
0 0 1 1 0	ra	rb	Imm17

OPERATION

```
R[ra] = R[rb] & signExt(imm17);
```

EXAMPLE

ANDI r1, r2, #3

OR

ASSEMBLY

OR ra, rb, rc

DESCRIPTION

The OR instruction performs a logical bit-wise or operation where both operands, rb and rc, are general-purpose register values. The result of the logical operation is stored in a general-purpose register, ra.

FORMAT

31 27	26 22	21 17	16 12	11 0
0 0 1 1 1	ra	rb	rc	unused

OPERATION

```
R[ra] = R[rb] | R[rc];
```

EXAMPLE

OR r1, r2, r3

ORI

ASSEMBLY

ORI ra, rb, imm17

DESCRIPTION

The ORI (or immediate) instruction performs a logical bit-wise OR operation where one operand is a general-purpose register value and the other is a 32-bit immediate value. The immediate operand is sign-extended to match the bit-length of the GPR operand. The result is stored in a general-purpose register, ra.

FORMAT

31 — 27	26 — 22	21 — 17	16 — 0
0 1 0 0 0	ra	rb	Imm17

OPERATION

```
R[ra] = R[rb] | signExt(imm17);
```

EXAMPLE

ORI r10, r11, #0x10

XOR

ASSEMBLY

XOR ra, rb, rc

DESCRIPTION

The XOR instruction performs a logical bit-wise exclusive-or operation where both operands, rb and rc, are general-purpose register values. The result of the logical operation is stored in a general-purpose register, ra.

FORMAT

31 27	26 22	21 17	16 12	11 0
0 1 0 0 1	ra	rb	rc	unused

OPERATION

```
R[ra] = R[rb] ^ R[rc];
```

EXAMPLE

XOR r1, r2, r3

LSR

ASSEMBLY

LSR ra, rb, shamt

LSR ra, rb, rc

DESCRIPTION

The LSR (logical shift right) shifts the content of a general-purpose register, rb, to the right by a given amount and store the result in a general-purpose register, ra. If the ‘i' bit is 0, the shift amount is an immediate value, specified in shamt field, ranging from 0 to 31. Otherwise the shift amount is the value in a general-purpose register, rc.

Note that this shift operation is not arithmetic, thus no sign-extension is performed.

FORMAT

31 27	26 22	21 17	16 12	11 6	5	4 0
0 1 0 1 0	ra	rb	rc	unused	i	shamt

OPERATION

```
if(i == 0)
      R[ra] = {{(shamt){0}}, R[rb]<31..shamt>};
else
      R[ra] = {{(R[rc]<4:0>){0}}, R[rb]<31..R[rc]<4:0>>};
```

EXAMPLE

LSR r1, r2, #5

ASR

ASSEMBLY

ASR ra, rb, shamt

ASR ra, rb, rc

DESCRIPTION

The ASR (arithmetic shift right) performs an arithmetic shift of the content of a general-purpose register, rb, to the right by a given amount and store the result in a general-purpose register, ra. If the I bit is 0, the shift amount is an immediate value, specified in shamt field, ranging from 0 to 31. Otherwise the shift amount is the value in a general-purpose register, rc.

Note that this shift operation is arithmetic, thus sign-extension is performed.

FORMAT

31 27	26 22	21 17	16 12	11 6	5	4 0
0 1 0 1 1	ra	rb	rc	unused	i	shamt

OPERATION

```
if(i == 0)
     R[ra] = {{(shamt){R[rb]<31>}}, R[rb]<31..shamt>};
else
     R[ra] = {{(R[rc]<4..0>){R[rb]<31>}},
                  R[rb]<31..R[rc]<4..0>>};
```

EXAMPLE

ASR r1, r2, #5

SHL

ASSEMBLY

SHL ra, rb, shamt

SHL ra, rb, rc

DESCRIPTION

The SHL (shift left) shifts the content of a general-purpose register, rb, to the left by a given amount and store the result in a general-purpose register, ra. If the I bit is 0, the shift amount is an immediate value, specified in shamt field, ranging from 0 to 31. Otherwise the shift amount is the value in a general-purpose register, rc.

FORMAT

31 27	26 22	21 17	16 12	11 6	5	4 0
0 1 1 0 0	ra	rb	rc	unused	i	shamt

OPERATION

```
if(I == 0)
    R[ra] = {R[rb]<31-shamt..0>, {(shamt){0}}};
else
    R[ra] = {R[rb]<31-R[rc]<4..0>..0>, {(R[rc]<4..0>){0}}};
```

EXAMPLE

SHL r1, r2, #5

ROR

ASSEMBLY

ROR ra, rb, shamt

ROR ra, rb, rc

DESCRIPTION

The ROR (rotate right) rotates the content of a general-purpose register, rb, to the right by a given amount and store the result in a general-purpose register, ra. If the I bit is 0, the rotating amount is an immediate value, specified in shamt field, ranging from 0 to 31. Otherwise the rotating amount is the value in a general-purpose register, rc.

FORMAT

31 27	26 22	21 17	16 12	11 6	5	4 0
0 1 1 0 1	ra	rb	rc	unused	i	shamt

OPERATION

```
if(I == 0)
     R[ra] = {R[rb]<shamt..0>, R[rb]<31..shamt+1>};
else
     R[ra] = {R[rb]<R[rc]<4..0>..0>,
                R[rb]<31..R[rc]<4..0>+1>};
```

EXAMPLE

ROR r1, r2, #5

MOVI

ASSEMBLY

MOVI ra, imm17

DESCRIPTION

The MOVI (move immediate) instruction moves the 32-bit immediate value into a general-purpose register, ra. The immediate operand is sign-extended to match the bit-length of the destination GPR.

FORMAT

31 27	26 22	21 17	16 0
0 1 1 1 0	ra	unused	Imm17

OPERATION

```
R[ra] = signExt(imm17);
```

EXAMPLE

MOVI ra, #0x11

J

ASSEMBLY

J imm22

DESCRIPTION

The J (jump) instruction provides changes to program flow unconditionally. The 22 bit immediate (imm22) is sign-extended and the target address is generated by adding this extended value to the currentPC (currentPC = Address of J instruction + 4). One more instruction after J is executed before actual jump occurs, i.e. the J instruction has 1 delay slot.

FORMAT

31 … 27	26 … 22	21 … 0
0 1 1 1 1	unused	Imm22

OPERATION

```
PC = currentPC + signExt(imm22);
```

EXAMPLE

J #11

JL

ASSEMBLY

JL ra, imm22

DESCRIPTION

The JL (jump and link) instruction provides changes to program flow unconditionally while storing the currentPC (currentPC = Address of J instruction + 4) into a link register(R[ra]). The link address is mainly used to return from a subroutine or a procedure. The 22-bit immediate (imm22) is sign-extended and the target address is generated by adding this extended value to the currentPC. One more instruction after JL is executed before actual jump occurs, i.e. the JL instruction has 1 delay slot.

FORMAT

31 27	26 22	21 0
1 0 0 0 0	ra	Imm22

OPERATION

```
R[ra] = currentPC;
PC = currentPC + signExt(imm22);
```

EXAMPLE

JL r1, #11

BR

ASSEMBLY

BR{cond} rb{, rc}

DESCRIPTION

The BR (branch) instruction provides changes to program flow based on the specified condition. The target address of the branch is the value in the general-purpose register rb and the value to be tested for condition evaluation is in the general-purpose register rc. There are six conditions, never, always, zero, nonzero, plus and minus, each encoded as 3’b000, 3’b001, 3’b010, 3’b011, 3’b100 and 3’b101, respectively. These are specified in the 3-bit condition field in the instruction format. With the condition satisfied, one more instruction after BR is executed before actual branch occurs, i.e. the BR instruction has 1 delay slot.

FORMAT

31 27	26 22	21 17	16 12	11 3	2 0
1 0 0 0 1	unused	rb	rc	unused	cond

OPERATION

```
if(cond == 0) { /* Never */
   do_nothing;}
else if(cond == 1) { /* Always */
   PC = R[rb];}
else if(cond == 2) { /* Zero */
   if(R[rc] == 0)
         PC = R[rb];}
else if(cond == 3) { /* Nonzero */
   if(R[rc] != 0)
         PC = R[rb];}
else if(cond == 4) { /* Plus */
   if(R[rc] >= 0)
         PC = R[rb];}
else if(cond == 5) { /* Minus */
   if(R[rc] < 0)
         PC = R[rb];}
```

EXAMPLE

BR r1

BRL

ASSEMBLY

BRL{cond} ra, rb{, rc}

DESCRIPTION

The BRL (branch and link) instruction provides changes to program flow based on the specified condition while storing the current value of PC into a link register. The link address is mainly used to return from a subroutine or a procedure. The target address of the branch is the value in the general-purpose register rb, the value to be tested for condition evaluation is in the general-purpose register rc and the link register is ra. Branch conditions are the same as those of BR. With the condition satisfied, one more instruction after BRL is executed before actual branch occurs, i.e. the BRL instruction has 1 delay slot. Note that the link register is updated regardless of whether the condition is satisfied or not.

FORMAT

31 27	26 22	21 17	16 12	11 3	2 0
1 0 0 1 0	ra	rb	rc	unused	cond

OPERATION

```
R[ra] = currentPC;
if(cond == 0) { /* Never */
   do_nothing;}
else if(cond == 1) { /* Always */
   PC = R[rb];}
else if(cond == 2) { /* Zero */
   if(R[rc] == 0)
         PC = R[rb];}
else if(cond == 3) { /* Nonzero */
   if(R[rc] != 0)
         PC = R[rb];}
else if(cond == 4) { /* Plus */
   if(R[rc] >= 0)
         PC = R[rb];}
else if(cond == 5) { /* Minus */
   if(R[rc] < 0)
         PC = R[rb];}
```

EXAMPLE

BRL r1, r2

ST

ASSEMBLY

ST ra, imm17

ST ra, imm17(rb)

DESCRIPTION

The ST (store) instruction stores the content in a general-purpose register, ra, to the given address of memory. The address may be either an absolute address or a displacement address. An absolute address is given by a 17-bit immediate value (imm17). Thus, the memory area that the absolute address can handle ranges from 00000000h to 00020000h. When a displacement address is used, it is formed by adding imm17 and the value stored in the register, rb. The field rb being 31 indicates that the address is absolute.

FORMAT

31 27	26 22	21 17	16 0
1 0 0 1 1	ra	rb	Imm17

OPERATION

```
if(rb == 5'b11111)
     M[zeroExt(imm17)] = R[ra];
else
     M[R[rb] + signExt(imm17)] = R[ra];
```

EXAMPLE

ST r1, #0x1100

STR

ASSEMBLY

STR ra, imm22

DESCRIPTION

The STR (store PC relative) instruction stores the content of a general-purpose register ra to the given relative address of memory. The relative address is calculated by adding a 22-bit immediate value (imm22) to the current value of PC. Note that, since this addition is signed, imm22 is sign-extended to be used as an operand.

FORMAT

31 27	26 22	21 0
1 0 1 0 0	ra	Imm22

OPERATION

```
M[currentPC + signExt(imm22)] = R[ra];
```

EXAMPLE

STR r1, #0x1100

LD

ASSEMBLY

LD ra, imm17

LD ra, imm17(rb)

DESCRIPTION

The LD (load) instruction loads a general-purpose register, ra, with a word (4 bytes), of data from the given address of memory. The address may be either an absolute address or a displacement address. An absolute address is given by a 17-bit immediate value (imm17). Thus, the memory area that the absolute address can handle ranges from 00000000h to 00020000h. When a displacement address is used, it is formed by adding a 17-bit immediate value and the value stored in the register, rb. The field rb being 31 indicates that the address is absolute.

FORMAT

31 27	26 22	21 17	16 0
1 0 1 0 1	ra	rb	Imm17

OPERATION

```
if(rb == 5’b11111)
     R[ra] = M[zeroExt(imm17)];
else
     R[ra] = M[signExt(imm17) + R[rb]];
```

EXAMPLE

LD r1, #0x1100

LDR

ASSEMBLY

LDR ra, imm22

DESCRIPTION

The LDR (load PC relative) instruction loads a general-purpose register ra with a word of data from the given relative address of memory. The relative address is calculated by adding a 22-bit immediate value (imm22) to the current value of PC. Note that, since this addition is signed, imm22 is sign-extended to be used as an operand.

FORMAT

31 27	26 22	21 0
1 0 1 1 0	ra	Imm22

OPERATION

```
R[ra] = M[currentPC + signExt(imm22)];
```

EXAMPLE

LDR r1, #0x1100

Acknowledgement

We thank to the KAIST ICSL Lab, as this exercise has been significantly influenced by their creation of the KRP-VIII ISA.