EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 7_1: MIPS ISA

Xiaolin Xu
Department of ECE
Northeastern University

Big-Endian & Little-Endian Memory

- How to number bytes within a word?
- **Little-endian:** byte numbers start at the little (least significant) end
- **Big-endian:** byte numbers start at the big (most significant) end
- Difference:
  - Byte address start from where?

Big-Endian & Little-Endian Memory

- It doesn't really matter which addressing type used – except when the two systems need to share data!
- ***Computer itself not get confused at all!***

Big-Endian & Little-Endian Example

- Suppose `$t0` initially contains 0x23456789
- After following code runs on big-endian system, what value is `$s0`?

- In a little-endian system?

```
sw $t0, 0($0)
lb $s0, 1($0)
```

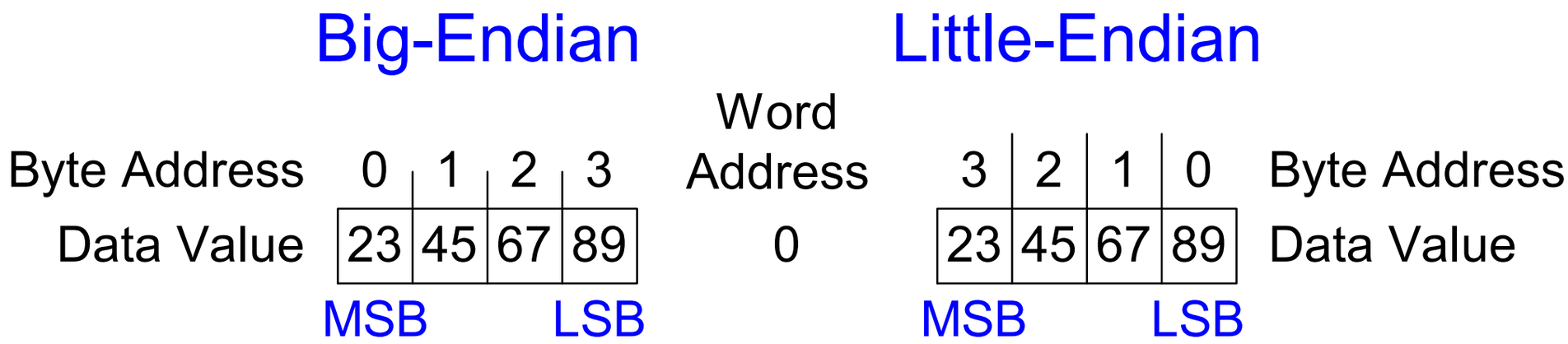
- **lb $s0, 1($0)** loads the data at byte address (1 + $0) = 1 into the least significant byte of $s0

Big-Endian & Little-Endian Example

- Suppose `$t0` initially contains 0x23456789
- After following code runs on big-endian system, what value is `$s0`?
- In a little-endian system?

```
sw $t0, 0($0)
lb $s0, 1($0)
```

- Big-endian: 0x00000045
- Little-endian: 0x00000067



Design Principle 4

Good design demands good compromises

- Multiple instruction formats allow flexibility
  - `add`, `sub`: use 3 register operands
  - `lw`, `sw`: use 2 register operands and a constant
- Number of instruction formats kept small
  - to adhere to design principles 1 and 3 (simplicity favors regularity and smaller is faster).

Operands: Constants / Immediates

- `lw` and `sw` use constants or *immediates*
- *immediate*ly available from instruction
- 16-bit two's complement number
- `addi`: add immediate
- Do we need immediate (`subi`)?

C Code

```
a = a + 4;
b = a - 12;
```

MIPS assembly code

```
# $s0 = a, $s1 = b
addi $s0, $s0, 4
addi $s1, $s0, -12
```

Machine Language

- Binary representation of instructions
- Computers only understand 1's and 0's
- 32-bit instructions
  - Simplicity favors regularity: 32-bit data & instructions
- 3 instruction formats:
  - **R-Type:** register operands
  - **I-Type:** immediate operand
  - **J-Type:** for jumping (discuss later)

R-Type

- *Register-type*
- 3 register operands:
  - `rs`, `rt`: source registers
  - `rd`: destination register
- Other fields:
  - `op`: the *operation code* or *opcode* (0 for R-type instructions)
  - `funct`: the *function*
    with opcode, tells computer what operation to perform
  - `shamt`: the *shift amount* for shift instructions, otherwise it's 0

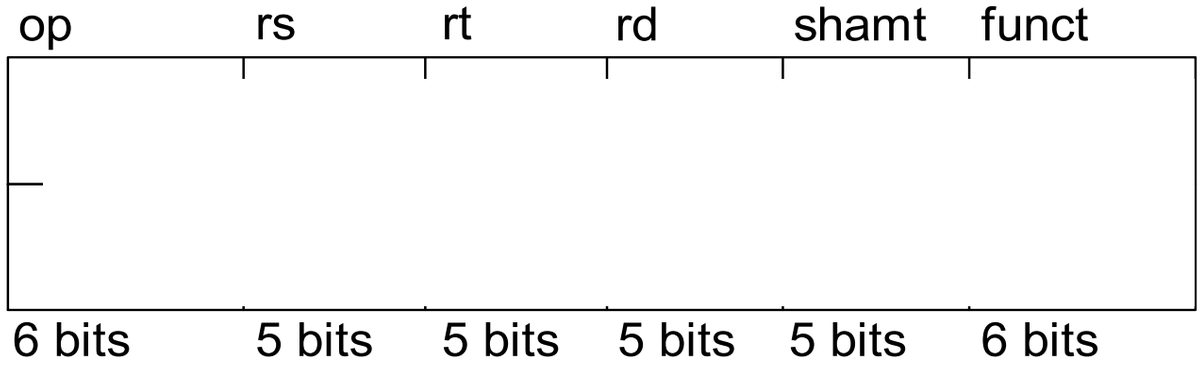
R-Type

R-Type Examples

Assembly Code

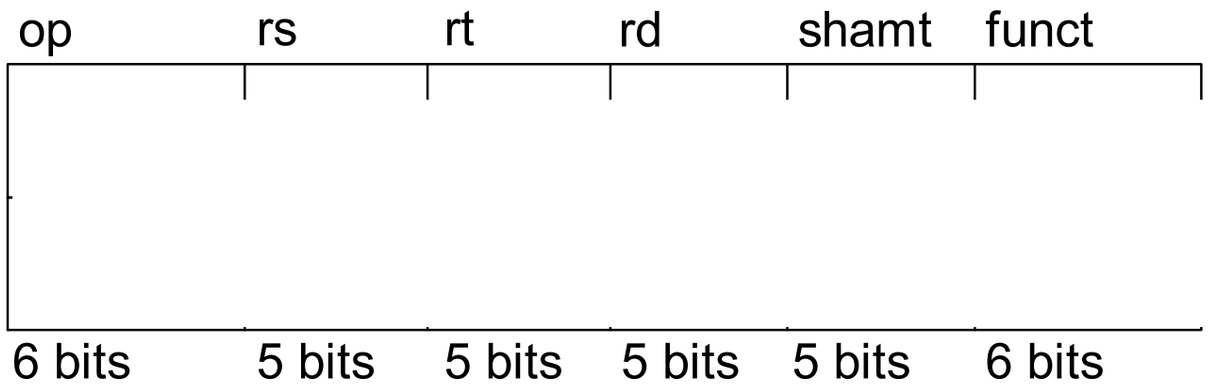
```
add $s0, $s1, $s2

sub $t0, $t3, $t5
```

Field Values



add has op code of 0 and funct code of 32

Machine Code



Note the order of registers in the assembly code:

```
add rd, rs, rt
```

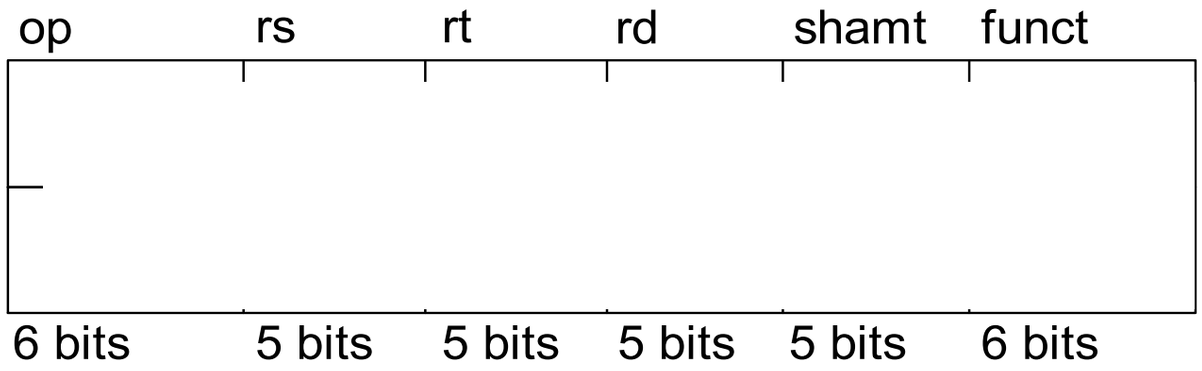
R-Type Examples

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

Assembly Code

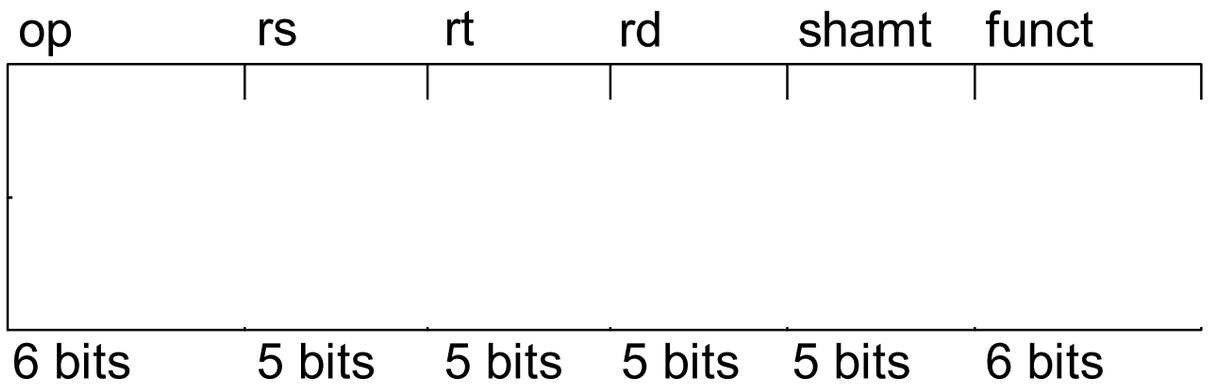
```
add $s0, $s1, $s2

sub $t0, $t3, $t5
```

Field Values



add has op code of 0 and funct code of 32

Machine Code



Note the order of registers in the assembly code:

```
add rd, rs, rt
```

R-Type Examples

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

Assembly Code

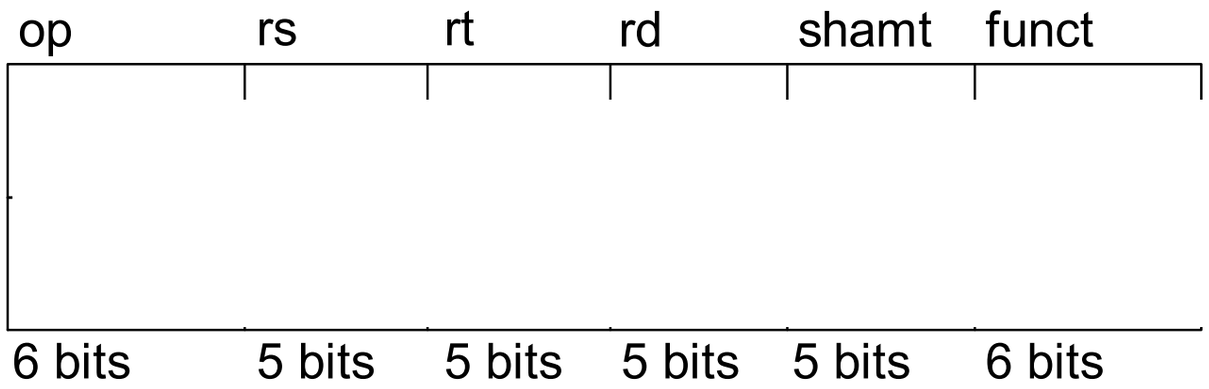
```
add $s0, $s1, $s2
sub $t0, $t3, $t5
```

Field Values

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

add has op code of 0 and funct code of 32

Machine Code



Note the order of registers in the assembly code:

```
add rd, rs, rt
```

R-Type Examples

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

Assembly Code

```
add $s0, $s1, $s2
sub $t0, $t3, $t5
```

Field Values

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

add has op code of 0 and funct code of 32

Machine Code

op	rs	rt	rd	shamt	funct	
000000	10001	10010	10000	00000	100000	(0x02328020)
000000	01011	01101	01000	00000	100010	(0x016D4022)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Note the order of registers in the assembly code:

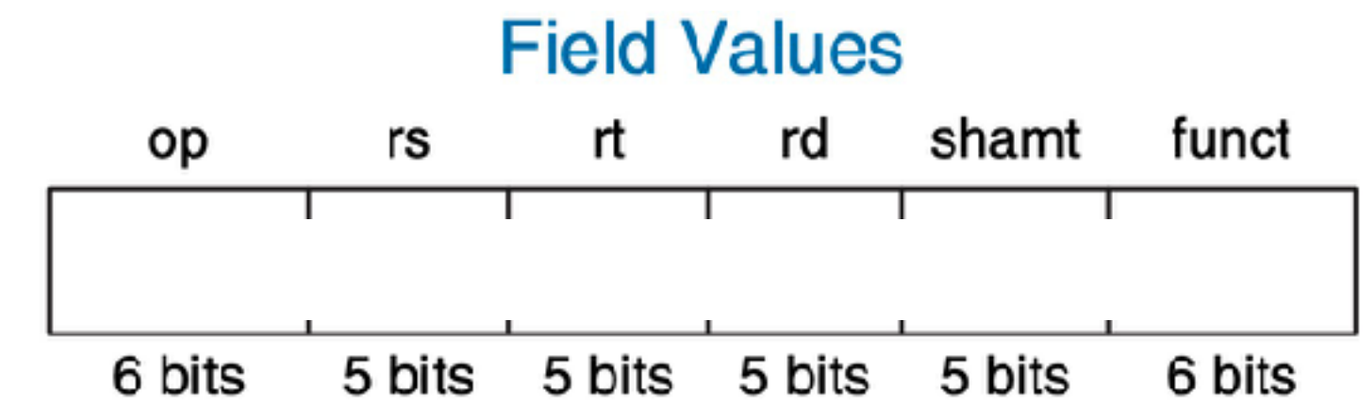
```
add rd, rs, rt
```

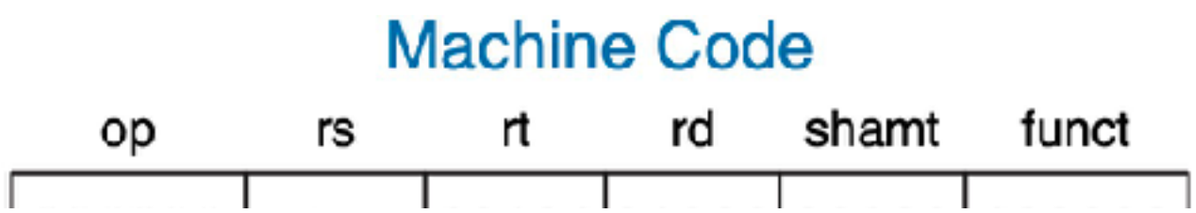
Practice of R-type

```
add $t4, $s4, $s5
```

add has op code of 0 and funct code of 32

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31





Practice of R-type

```
add $t4, $s4, $s5
```

add has op code of 0 and funct code of 32

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

Field Values

op	rs	rt	rd	shamt	funct
0	20	21	8	0	32
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

Practice of R-type

```
add $t4, $s4, $s5
```

add has op code of 0 and funct code of 32

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

Field Values

op	rs	rt	rd	shamt	funct
0	20	21	8	0	32
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

op	rs	rt	rd	shamt	funct	
000000	10100	10101	01000	00000	100000	(0x02954020)
0	2	9	5	4	0	2 0

I-Type

- *Immediate-type*
- 3 operands:
  - `rs`, `rt`: register operands
  - `imm`: 16-bit two's complement immediate
- Other fields:
  - `op`: the opcode
  - Simplicity favors regularity: all instructions have opcode
  - Operation is completely determined by opcode

I-Type

Immediate Data Representation

- As a 16-bit two's complement number, ***imm*** can denotes values in range [-32,768, 32,767]
  - If ***imm*** is a positive value, then just use its binary representation
  - If ***imm*** is a negative value, then just use its two's complement representation
- MIPS operates with 32-bit data, how to proceed?

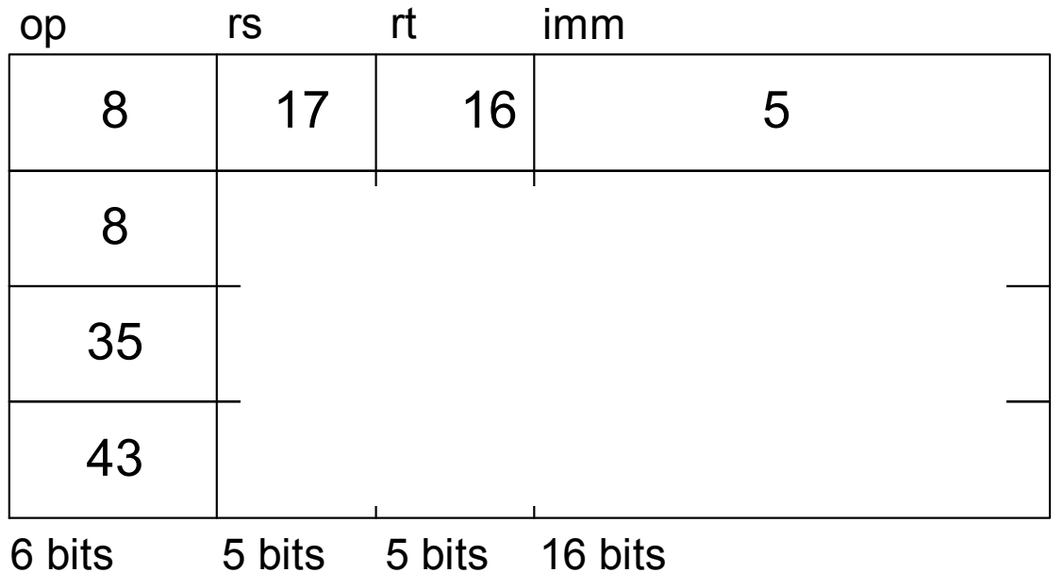
Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

I-Type Examples

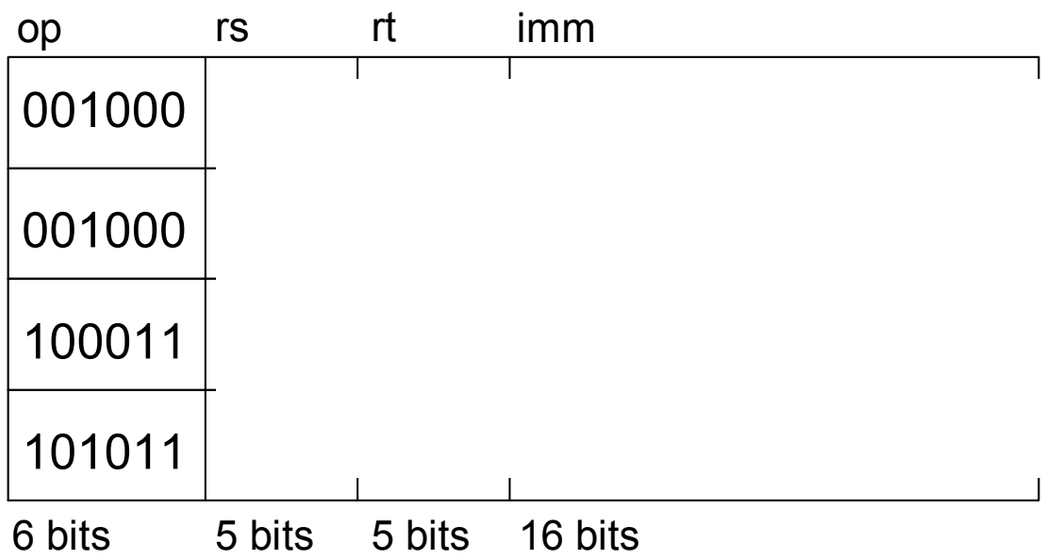
Assembly Code

```
addi $s0, $s1, 5
addi $t0, $s3, -12
lw   $t2, 32($0)
sw   $s1,  4($t1)
```

Field Values



Machine Code



Note the differing order of registers in assembly and machine codes:

```
addi rt, rs, imm
lw   rt, imm(rs)
sw   rt, imm(rs)
```

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

I-Type Examples

Assembly Code

```
addi $s0, $s1, 5
addi $t0, $s3, -12
lw   $t2, 32($0)
sw   $s1,  4($t1)
```

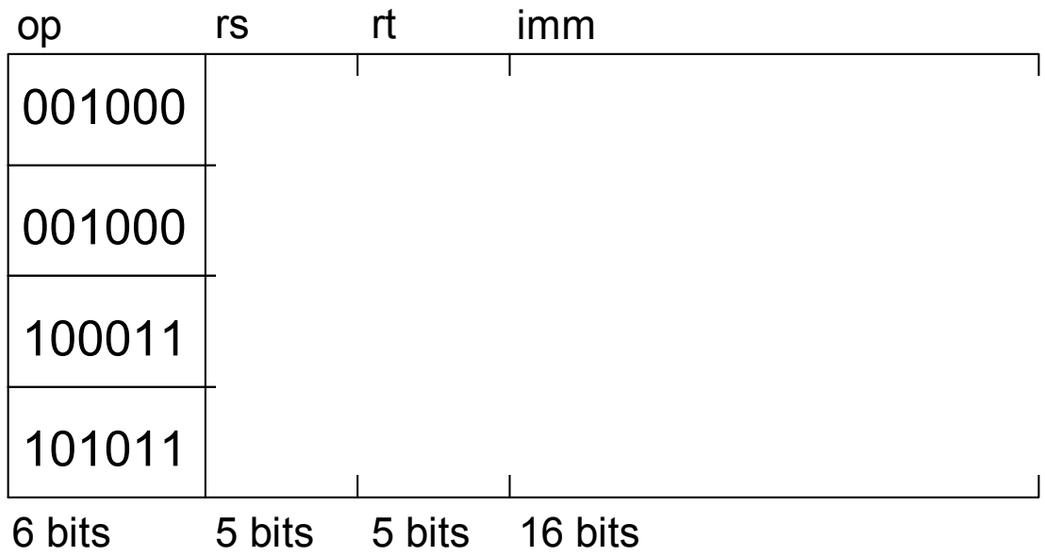
Field Values

op	rs	rt	imm
8	17	16	5
8	19	8	-12
35	0	10	32
43	9	17	4
6 bits	5 bits	5 bits	16 bits

Note the differing order of registers in assembly and machine codes:

```
addi rt, rs, imm
lw   rt, imm(rs)
sw   rt, imm(rs)
```

Machine Code



Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

I-Type Examples

Assembly Code

```
addi $s0, $s1, 5
addi $t0, $s3, -12
lw   $t2, 32($0)
sw   $s1,  4($t1)
```

Field Values

op	rs	rt	imm
8	17	16	5
8	19	8	-12
35	0	10	32
43	9	17	4
6 bits	5 bits	5 bits	16 bits

Note the differing order of registers in assembly and machine codes:

```
addi rt, rs, imm
lw   rt, imm(rs)
sw   rt, imm(rs)
```

Machine Code

op	rs	rt	imm	
001000	10001	10000	0000 0000 0000 0101	(0x22300005)
001000	10011	01000	1111 1111 1111 0100	(0x2268FFF4)
100011	00000	01010	0000 0000 0010 0000	(0x8C0A0020)
101011	01001	10001	0000 0000 0000 0100	(0xAD310004)
6 bits	5 bits	5 bits	16 bits	

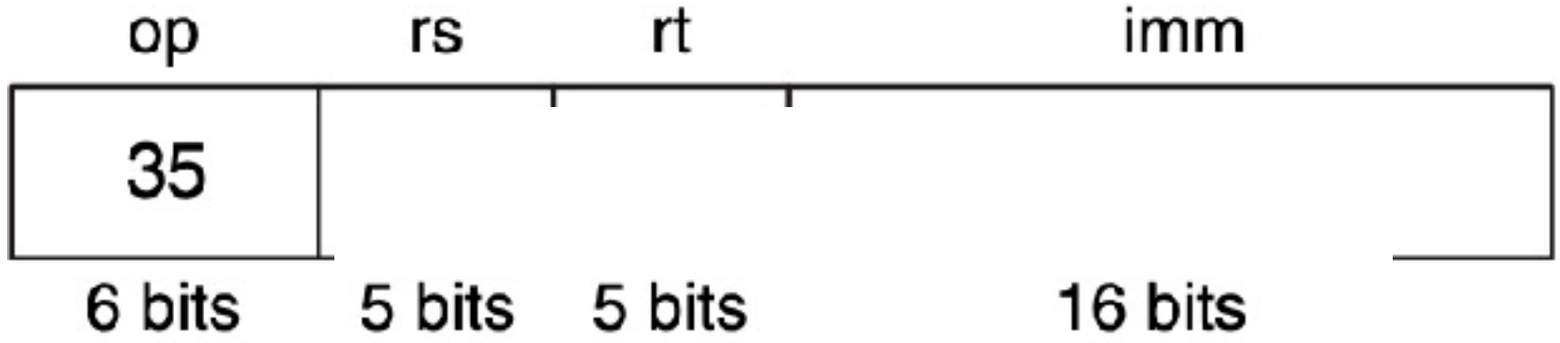
Practice of I-type

```
lw $s3, -24($s4)
```

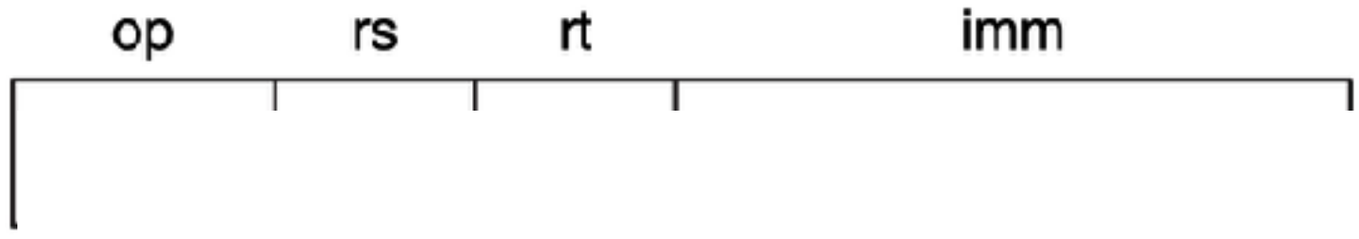
lw has opcode of 35

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

Field Values



Machine Code



Practice of I-type

`lw $s3, -24($s4)`

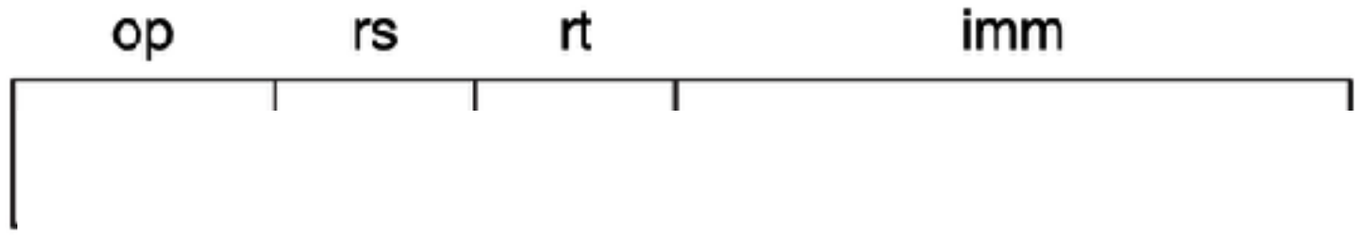
lw has opcode of 35

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

Field Values

op	rs	rt	imm
35	20	19	–24
6 bits	5 bits	5 bits	16 bits

Machine Code



Practice of I-type

`lw $s3, -24($s4)`

lw has opcode of 35

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31

Field Values

op	rs	rt	imm
35	20	19	–24
6 bits	5 bits	5 bits	16 bits

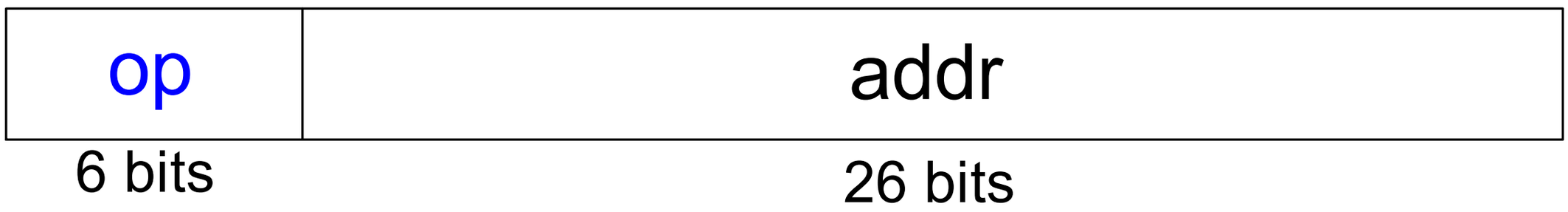
Machine Code

op	rs	rt	imm	
100011	10100	10011	1111 1111 1110 1000	(0x8E93FFE8)
8	E	9	3	F F E 8

Machine Language: J-Type

- *Jump-type*
- 26-bit address operand (`addr`)
- Used for jump instructions (`j`)

J-Type



Review: Instruction Formats

R-Type

I-Type

J-Type