CENG311 Computer Architecture

Representing Instructions

IZTECH, Fall 2023

02 November 2023

From a High-Level Language to the Language of Hardware

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

Assembly language program (for MIPS)

```
swap:
      multi $2, $5,4
      add   $2, $4,$2
      lw    $15, 0($2)
      lw    $16, 4($2)
      sw    $16, 0($2)
      sw    $15, 4($2)
      jr    $31
```

Assembler

Binary machine language program (for MIPS)

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

Representing Instructions

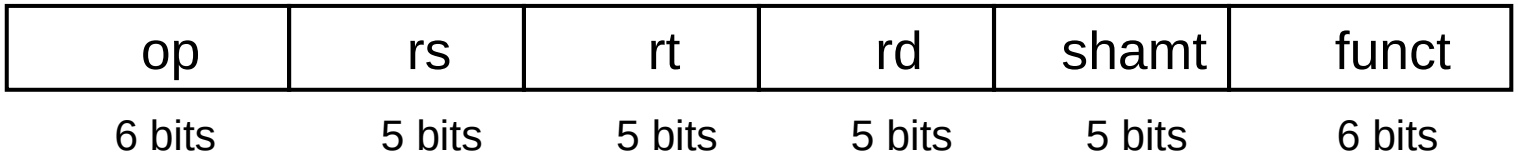
Kept as a series of high and low electronic signals (binary) and represented as numbers

MIPS instructions are 32 bits long

```
add $t0, $s1, $s2
```

000000	10001	10010	01000	00000	100000

MIPS R-Type Instructions



op: operation code (opcode)

rs: first source register number

rt: second source register number

rd: destination register number

shamt: shift amount (00000 for now)

funct: function code (extends opcode)

Register Numbers

$t0 – $t7 are reg’s 8 – 15

$t8 – $t9 are reg’s 24 – 25

$s0 – $s7 are reg’s 16 – 23

Add/sub Instruction Formats

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.

R-Type Example

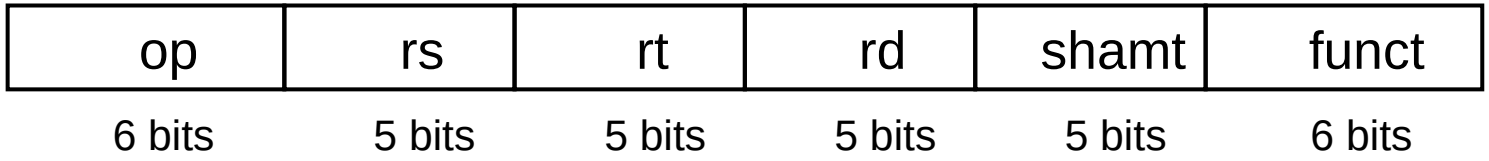
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$00000010001100100100000000100000_2$

Shift Operations



shamt: how many positions to shift

Shift left logical

Shift left and fill with 0 bits

sll by i bits multiplies by 2^i

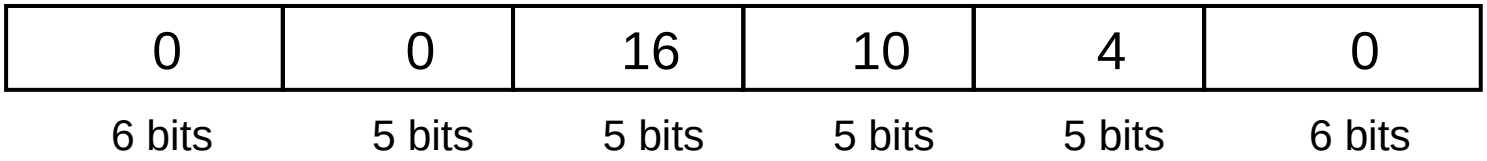
Shift right logical

Shift right and fill with 0 bits

srl by i bits divides by 2^i (unsigned only)

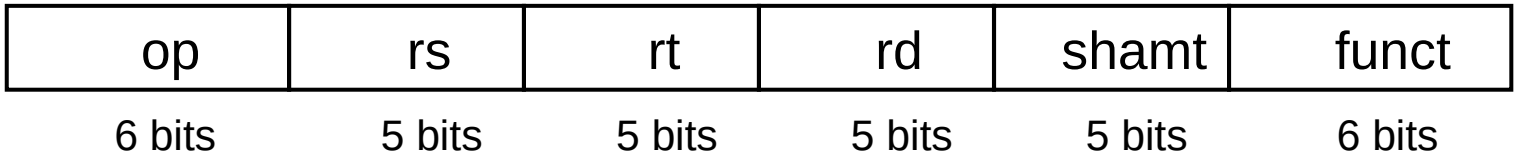
Shift Example

```
sll $t2, $s0, 4
```



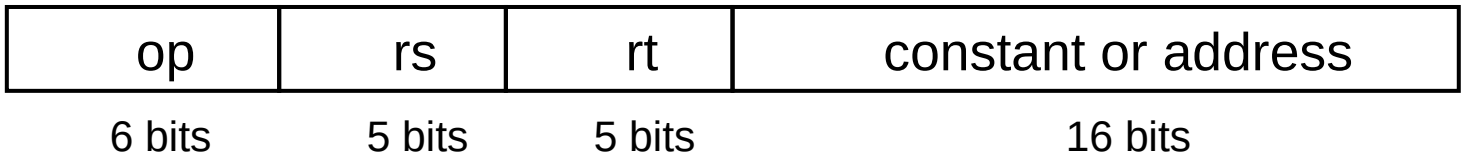
AND Example

```
and $t0, $t1, $t2
```



special	$t1	$t2	$t0	0	and
0	9	10	8	0	36
000000	01001	01010	01000	00000	100100

MIPS I-Type Instructions



Immediate arithmetic and load/store instructions, branch instructions

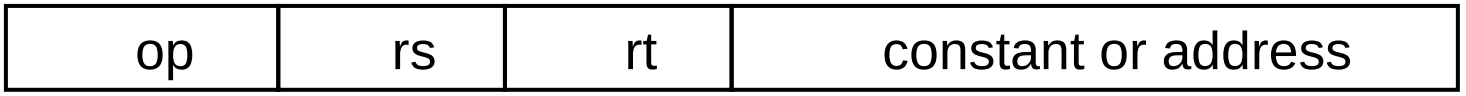
rt: destination or source register number

Constant: -2^{15} to $+2^{15} - 1$

Address: offset added to base address in rs

Instruction	Format	op	rs	rt	rd	shamt	funct	address
`add`	R	0	reg	reg	reg	0	32_{ten}	n.a.
`sub` (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
`add immediate`	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
`lw` (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
`sw` (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

I-Type Example




```
lw $t0, 32($s3)
```

op	rs	rt	constant or address
35	19	8	32
100011	10011	01000	0000000000100000

$$10001110011010000000000000100000_2$$

I-Type Example

A[300] = h + A[300]

```
lw    $t0, 1200($t1)
add   $t0, $s2, $t0
sw    $t0, 1200($t1)
```

Op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

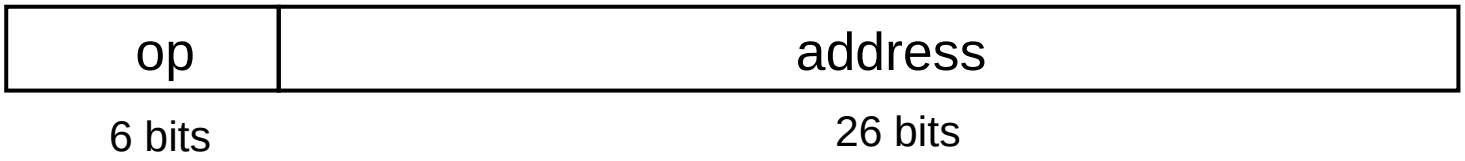
I-Type Example

Op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

$$1200_{ten} = 0000\ 0100\ 1011\ 0000_{two}$$

MIPS J-Type Instructions



jump (j and jal) instructions

jump instruction contains a word address, not an offset

A 26-bit address field lets you jump to any address from 0 to 2^{28}

Addressing Modes

Immediate addressing

Register addressing

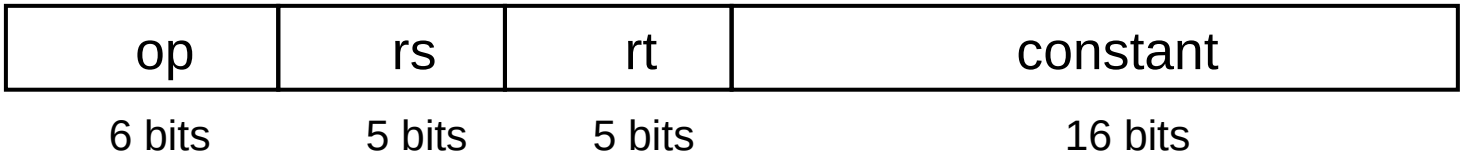
Base (displacement) addressing

PC-relative addressing

Pseudodirect addressing

Immediate Addressing

The operand is a constant within the instruction itself



addi $s3, $s3, 4

$s3 = $s3 + 4

32-Bit Immediate Operands

Most constants are small

16-bit immediate is sufficient

For the occasional 32-bit constant

lui rt, constant

Copies 16-bit constant to left 16 bits of rt

Clears right 16 bits of rt to 0

Load 4000000 (11 1101 0000 1001 0000 0000) into $s0

```
lui $s0, 61
```

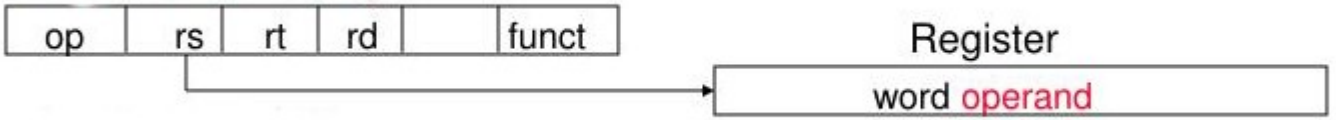
0000 0000 0011 1101 | 0000 0000 0000 0000

```
ori $s0, $s0, 2304
```

0000 0000 0011 1101 | 0000 1001 0000 0000

Register Addressing

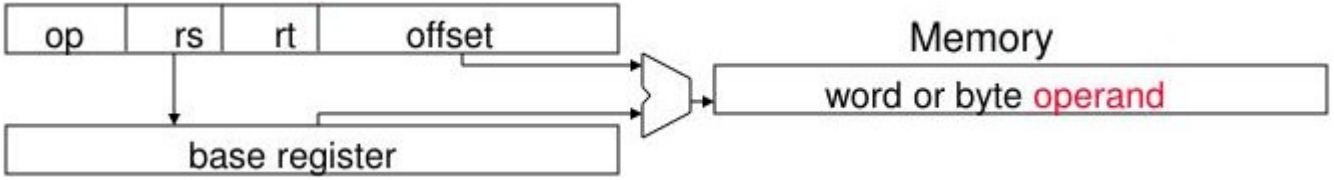
The operand is register



add $t0, $s1, $s2

Base Addressing

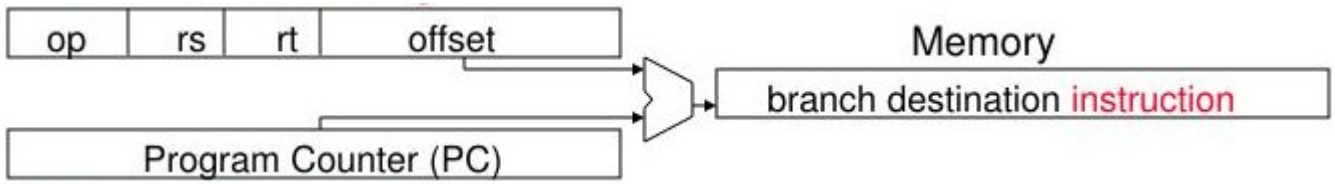
The operand is at the memory location whose address is the sum of a register and a constant in the instruction



lw $t0, 1200($t1)

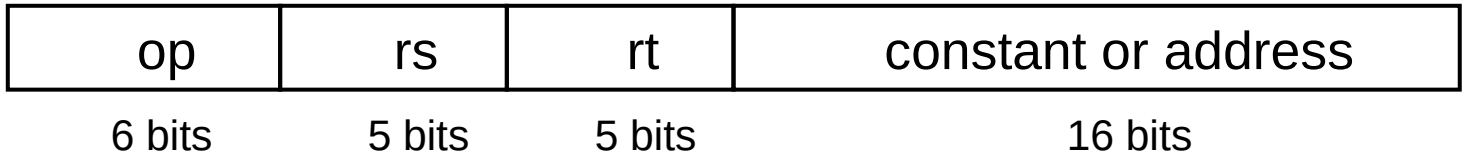
PC-Relative Addressing

The branch address is the sum of the PC and a constant in the instruction



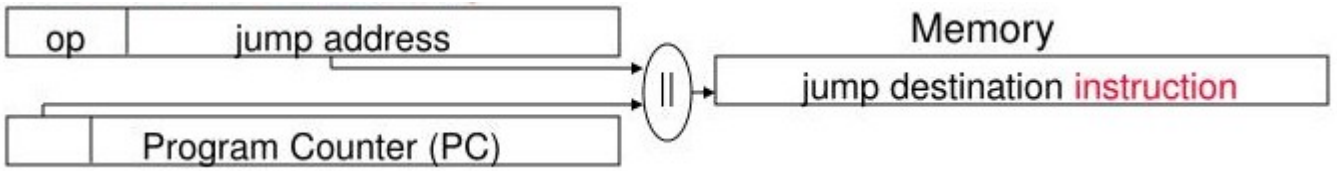
Target address = PC + offset × 4

beq rs, rt, L1



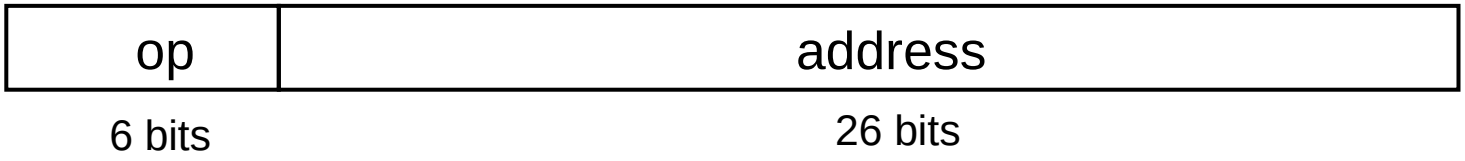
Pseudo-direct Addressing

The jump address is the 26 bits of the instruction concatenated with the upper bits of the PC



Target address = $PC_{31...28}$: (address × 4)

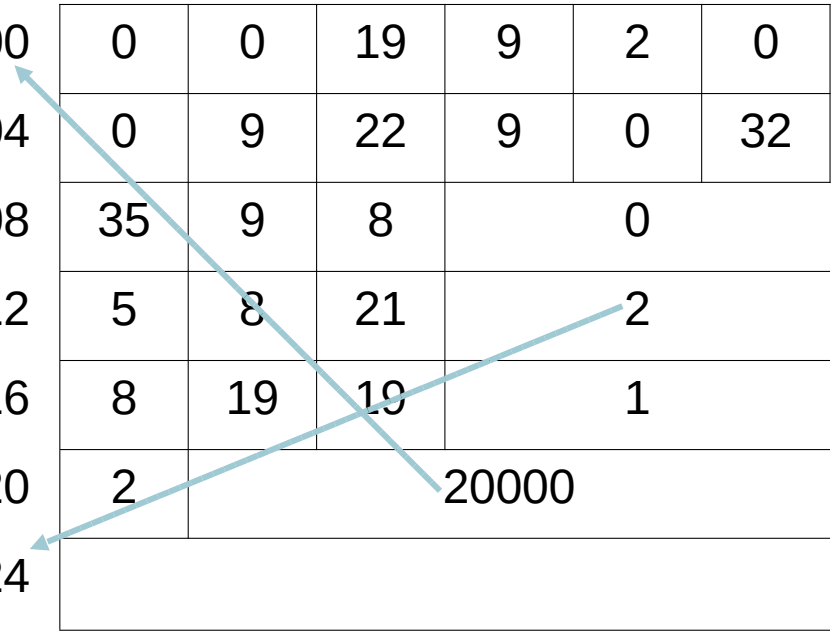
j L1



Target Addressing Example

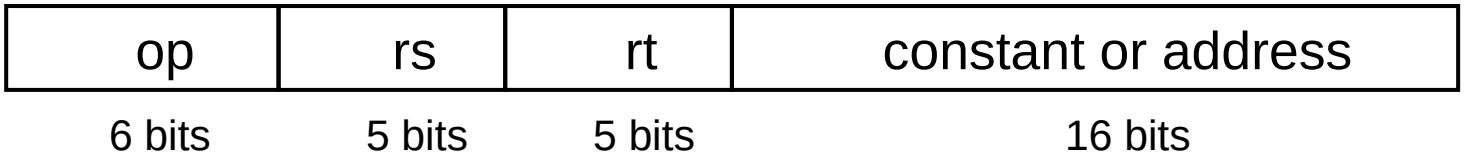
Assume Loop at location 80000

```
Loop: sll  $t1, $s3, 2      80000
      add  $t1, $t1, $s6    80004
      lw   $t0, 0($t1)      80008
      bne  $t0, $s5, Exit   80012
      addi $s3, $s3, 1      80016
      j    Loop             80020
Exit: …                     80024
```



Branching Far Away

If branch target is too far to encode with 16-bit offset, assembler rewrites the code




```
beq $s0,$s1, L1
        ↓
bne $s0,$s1, L2
j L1
L2: …
```

Decoding Machine Code

What is the assembly language statement corresponding to this machine instruction?

00af8020hex

0000 0000 1010 1111 1000 0000 0010 0000

R-format

00 101/0 1111/ 1000 0/000 00/10 0000

rs /rt /rd /shamt /funct

$a1 /$t7 /$s0 add

add $s0, $a1, $t7

References

Chapter 2.5

Chapter 2.10

(Computer Organization and Design: The Hardware/Software Interface by Hennessy/Patterson, 5th edition)