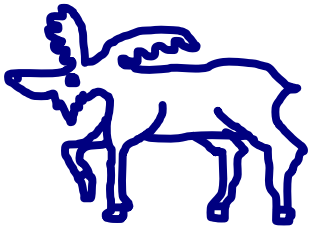
Lecture 3
Machine Instructions

Byung-gi Kim

School of Computer Science and Engineering

Soongsil University



2. Instruction: Language of the Computer (3rd edition)

MIPS Fields

- **R-type (Register-type)**

  - op: Basic operation of the instruction (*opcode*)
  - rs: The first register source operand
  - rt: The second register source operand
  - rd: The register destination operand
    It gets the result of the operation.
  - shamt: Shift amount (Explained in §2.5)
  - funct(function): This field selects the specific variant of the operation in the op field, and is sometimes called the *function code*. (cf) opcode extension

2.4 Representing Instructions in the Computer

- **Example : machine instruction**

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

[Answer]

0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

- First and last fields: addition operation
- Second field: register number of the first source operand ($s1)
- Third field: register number of the other source operand ($s2)
- Fourth field: number of the destination register ($t0)
- Fifth field: unused in this instruction (set to 0)

I-type Instruction Format

Design Principle 4 :

Good design demands good compromises.

[Example] MIPS instructions

- Same length but different formats

- **I-type (Immediate-type)**

op	rs	rt	constant/address
6 bits	5 bits	5 bits	16 bits

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

Example: A[300]=h+A[300];

```
lw  $t0,1200($t1) # Temporary reg. $t0 gets A[300]
add $t0,$s2,$t0   # Temporary reg. $t0 gets h+A[300]
sw  $t0,1200($t1) # Stores h+A[300] back into A[300]
```

[Answer]

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

The BIG Picture

- Instructions are represented as numbers.
- Programs can be stored in memory to be read or written, just like numbers.

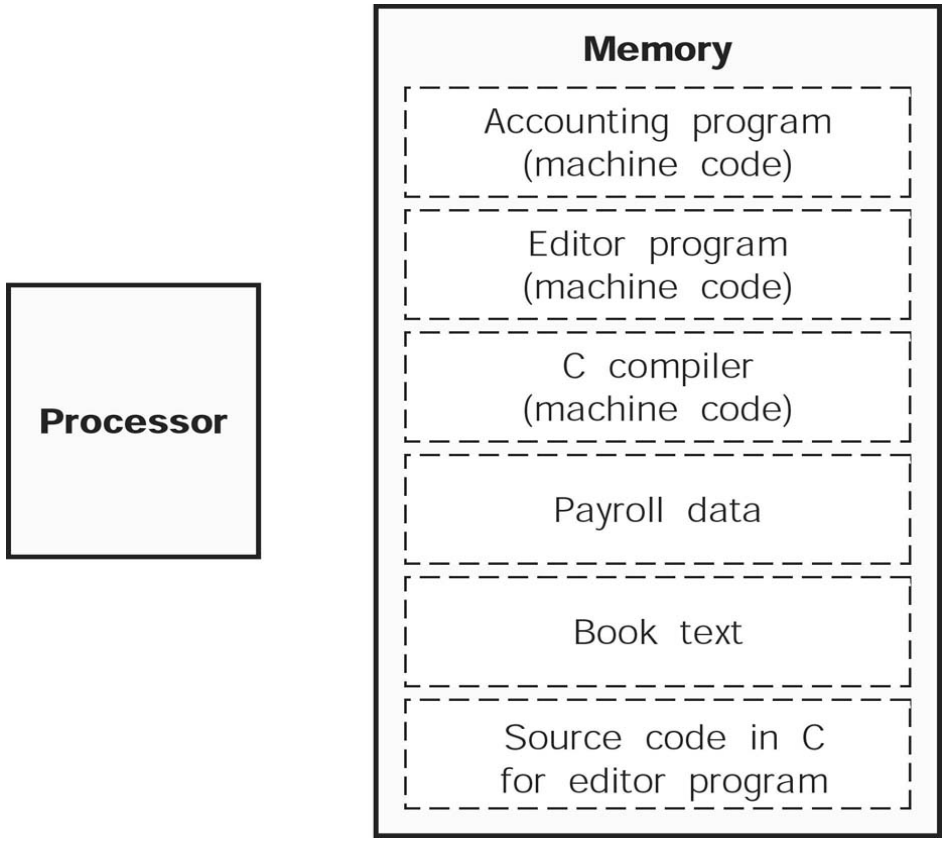


Figure 2.8 Stored-program concept

2.5 Logical Operations

Operations	C	Java	MIPS
Shift left	`<<`	`<<`	`sll`
Shift right	`>>`	`>>`	`srl, sra`
Bit-by-bit AND	`&`	`&`	`and, andi`
Bit-by-bit OR	`|`	`|`	`or, ori`
Bit-by-bit NOT	`~`	`~`	`nor`

Fig. 2.9 Logical operators

Shift Operations

$$X = x_{31}x_{30} \cdots x_0$$

1. Logical shift

- Logical shift right (X) = $0x_{31}x_{30} \cdots x_1$
- Logical shift left (X) = $x_{30} \cdots x_0 0$

2. Arithmetic shift (for 2's complement)

- Arithmetic shift right (X) = $x_{31}x_{31}x_{30} \cdots x_1$ (cf) ÷2
- Arithmetic shift left (X) = $x_{30} \cdots x_0 0$ (cf) *2

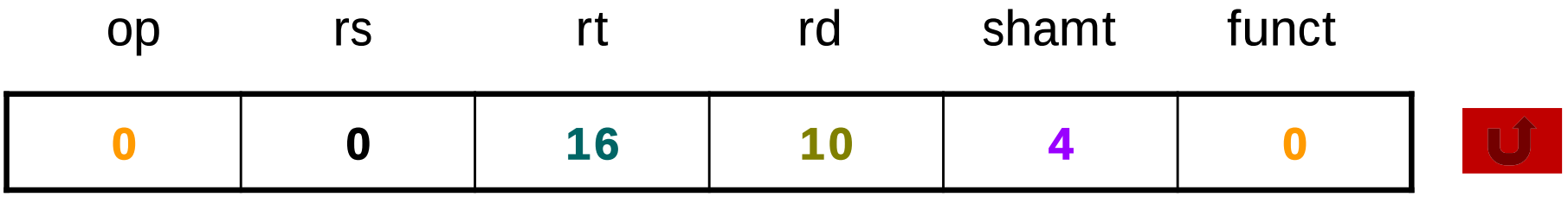
3. Circular shift (= rotate)

- Circular shift right (X) = $x_0x_{31}x_{30} \cdots x_1$
- Circular shift left (X) = $x_{30} \cdots x_0x_{31}$

MIPS Shift Instructions

- **sll (shift left logical); funct = 000 000**
- **srl (shift right logical); funct = 000 010**
- **sra (shift right arithmetic); funct = 000 011**

- **Instruction format**
  - `sll $t2, $s0, 4  # reg $t2 = reg $s0 << 4 bits`



- **sllv (shift left logical variable), srlv, srav**
  - `sllv $t2, $s0, $s1`

MIPS Rotate Instructions

- `rotr (rotate word right); funct = 000 010 (SRL)`

op	rs	rt	rd	shamt	funct
000000	00001				000010

- `rotrv (rotate word right variable)`
  `; funct = 000 110 (SRLV)`

op	rs	rt	rd	shamt	funct
000000				00001	000110

Logical Instructions

- `and  $t0,$t1,$t2`
- `or  $t0,$t1,$t2`
- `xor  $t0,$t1,$t2`
- `nor  $t0,$t1,$t2`

- `andi $s1,$s1,100`
- `ori  $s1,$s1,100`
- `xori $s1,$s1,100`
- `lui  $s1,100`

2.6 Instructions for Making Decisions

- **Similar to an *if* statement with a *go to* statement**
  - `beq register1, register2, L1`
  - `bne register1, register2, L1`
- **Example**

```
if (i==j) f=g+h; else f=g-h;
```

[Answer]

```
     bne  $s3,$s4,Else # go to Else if i≠j
     add  $s0,$s1,$s2  # f=g+h (skipped if i≠j)
     j    Exit         # go to Exit
Else: sub  $s0,$s1,$s2  # f=g-h (skipped if i=j)
Exit:
```

Loops

- **Example**

```
while (save[i]==k)  i += 1;
```

[Answer]

```
Loop: sll  $t1,$s3,2     # Temp reg $t1 = 4*i
      add  $t1,$t1,$s6   # $t1 = address of save[i]
      lw   $t0,0($t1)    # Temp reg $t0 = save[i]
      bne  $t0,$s5,Exit  # go to Exit if save[i]≠k
      addi $s3,$s3,1     # i = i + 1 (원서 error)
      j    Loop          # go to Loop
Exit:
```

`slt(set on less than)` Instruction

- `slt  $t0,$s3,$s4   # if($s3<$s4) then $t0=1`
  `                   #             else $t0=0`
- `slti $t0,$s2,10    # if $s2<10 $t0=1; else $t0=0`
- **Pseudo instruction**

  `blt  $s0,$s1,Less   # branch on less than`
- **Why no `blt` instruction in MIPS architecture?**
  - It would stretch the clock cycle time,
    or it would take extra clock cycles per instruction.

Hardware/Software Interface

- All comparisons are possible with slt, beq, bne with $zero.

Other Jump Instructions

- `jr` (jump register) instruction

```
jr  $t0      # jump to the address specified
             # in a register($t0)
             # i.e. PC ← $t0
```

- `jal` (jump-and-link) instruction

```
jal  ProcedureAddress   # $ra ← PC + 4,
                        # PC ← ProcedureAddress
```

- Return from procedure

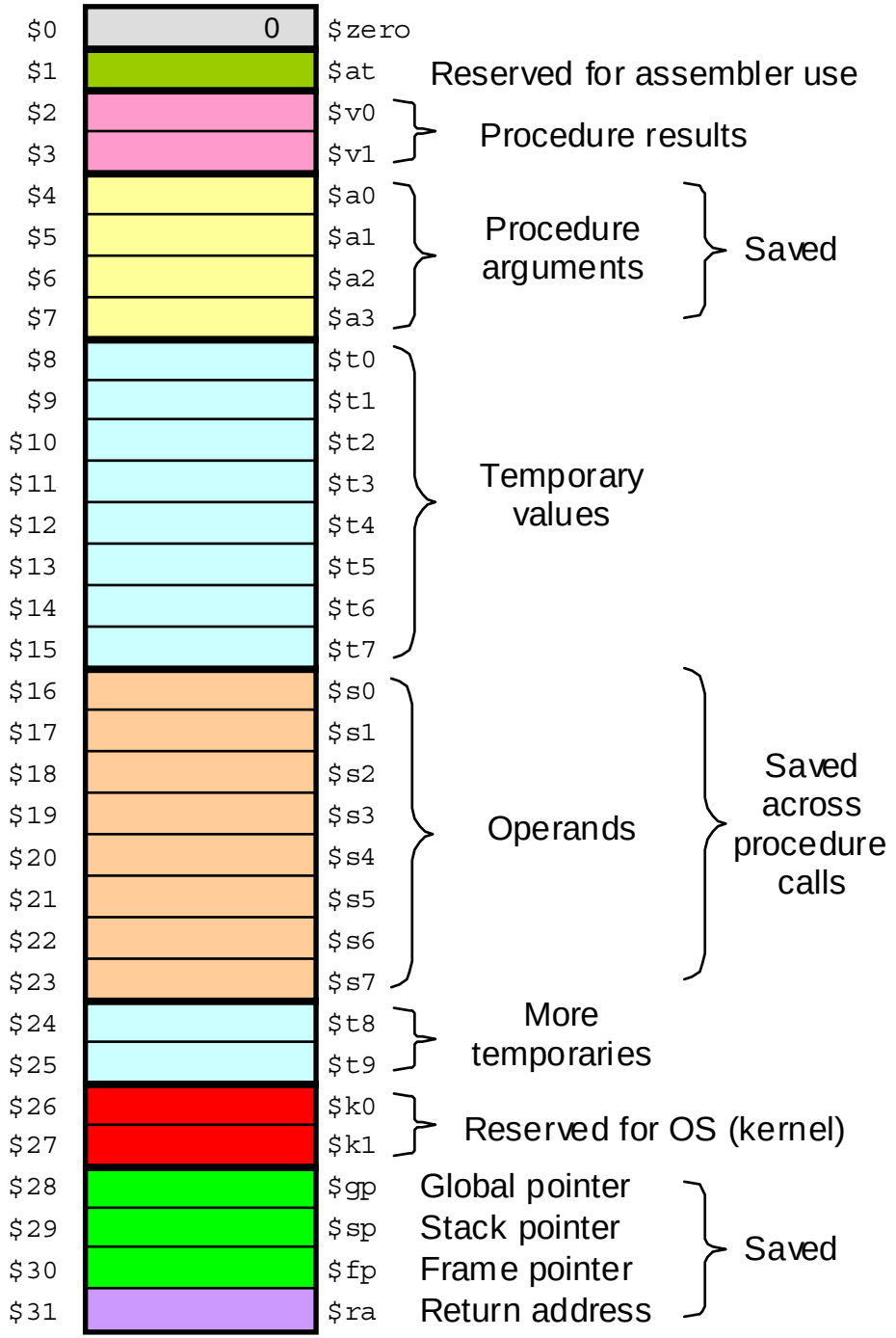
```
jr   $ra
```

MIPS Register Convention

Name	Register Number	Usage
`$zero`	0	the constant value 0
`$at`	1	reserved for assembler
`$v0-$v1`	2-3	values for results and expression evaluation
`$a0-$a3`	4-7	arguments
`$t0-$t9`	8-15, 24-25	temporaries
`$s0-$s7`	16-23	saved
`$k0-$k1`	26-27	reserved for operating system
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address

Figure 2.18

MIPS Registers



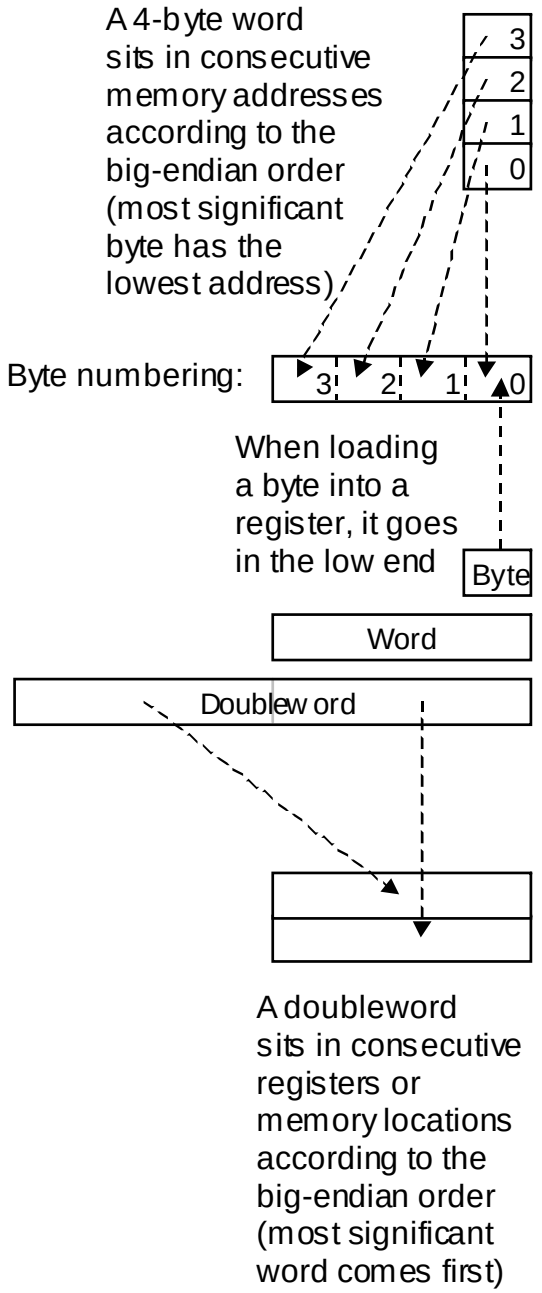


Figure 5.1 of Parhami

2.8 Communicating with People

- **Load byte instructions**
  - `lb $t0,0($s1)`
  - Loading a byte from memory and placing it in the rightmost 8 bits of a register
- **Store byte instructions**
  - `sb $t0,0($s0)`
  - Taking a byte from the rightmost 8 bits of a register and writing it to memory
- **Halfword transfer instructions**

```
lh $t0,0($s1) # Read 16 bits from source
sh $t0,0($s0) # Write 16 bits to destination
```

Summary

- **Instruction formats**
  - R-type and I-type
- **Shift instructions**
  - `sll, srl, sra, sllv, srlv, srav, rotr, rotrv`
- **Logical instructions**
  - `and, or, xor, nor, andi, ori, xori, lui`
- **Branch and jump instructions**
  - `beq, bne, j, jal, jr`
  - `slt, slti, sltu, sltiu`
- **Data transfer instructions**
  - `lw` and `sw`
  - `lh` and `sh`
  - `lb` and `sb`

Supplement

Using More Registers

- **Spilling registers**
  - More than 4 arguments
  - More than 2 return values
- **Stack**
  - Ideal data structure for spilling registers
  - Stack Pointer

    `$sp` (=$29)

Register Preservation

- `$t0-$t9` : not preserved
- `$s0-$s7` : preserved
- What is and what is not preserved across a procedure call: Fig 2.15

Preserved	Not preserved
Saved register: `$s0-$s7`	Temporary registers: `$t0-$t9`
Stack pointer register: `$sp`	Argument register: `$a0-$a3`
Return address register: `$ra`	Return value registers: `$v0-$v1`
Stack above the stack pointer	Stack below the stack pointer

Characters and Strings in Java

- **Unicode**
  - Used by Java
  - 16 bits for a character
  - UTF-8, UTF-16 and UTF-32
- **Halfword transfer instructions**

```
lh $t0,0($sp)  # Read 16 bits from source
sh $t0,0($gp)  # Write 16 bits to destination
```

- **Strings in Java**
  - Standard class
  - Special built-in support and predefined methods for concatenation, comparison and conversion.
  - Including a word that gives the length of the string.