Instructions: Language of the Computer

Computer Architecture

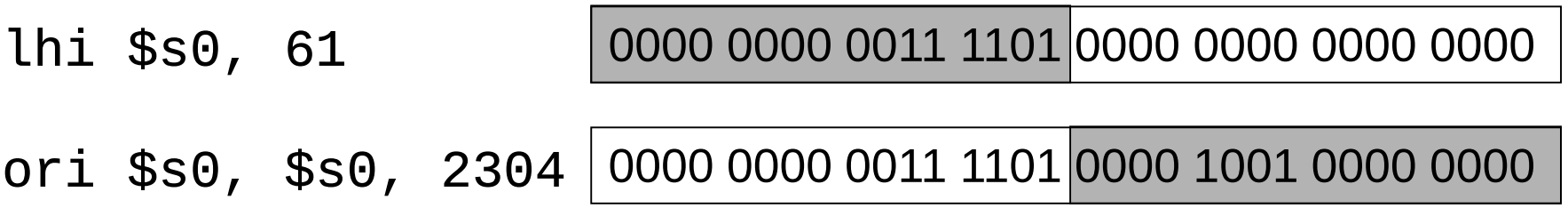
2019 1학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

32-bit Constants

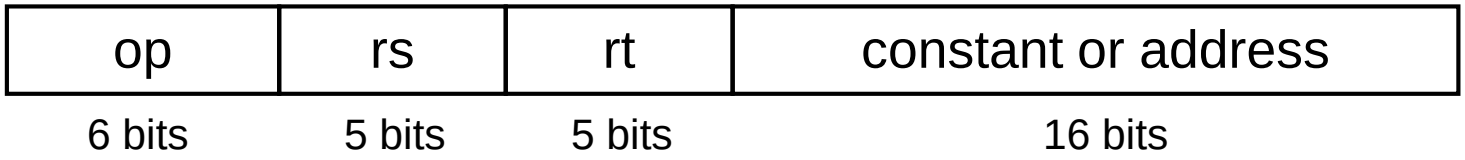
- Most constants are small
  - 16-bit immediate is sufficient
- For the occasional 32-bit constant

  `lui rt, constant`
  - Copies 16-bit constant to left 16 bits of rt
  - Clears right 16 bits of rt to 0



Branch Addressing

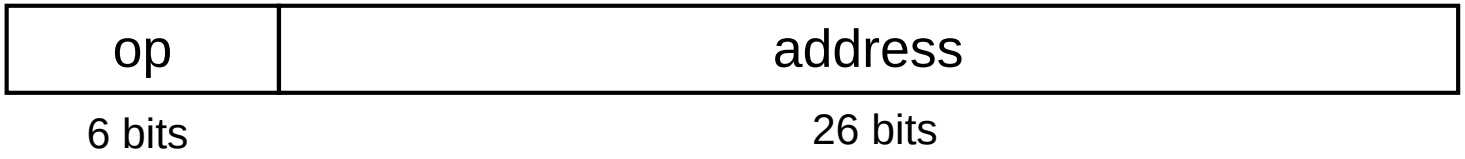
- Branch instructions specify
  - Opcode, two registers, target address
- Most branch targets are near branch
  - Forward or backward



- PC-relative addressing
  - Target address = PC + offset × 4
  - PC already incremented by 4 by this time

Jump Addressing

- Jump (`j` and `jal`) targets could be anywhere in text segment
  - Encode full address in instruction

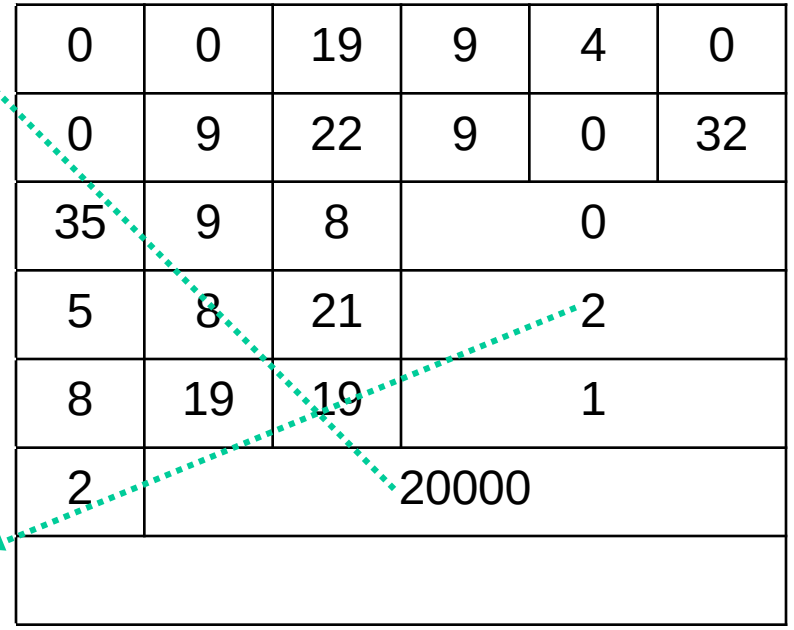


- (Pseudo)Direct jump addressing
  - Target address = $PC_{31\ldots28}$: (address × 4)

Target Addressing Example

- Loop code from earlier 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
```

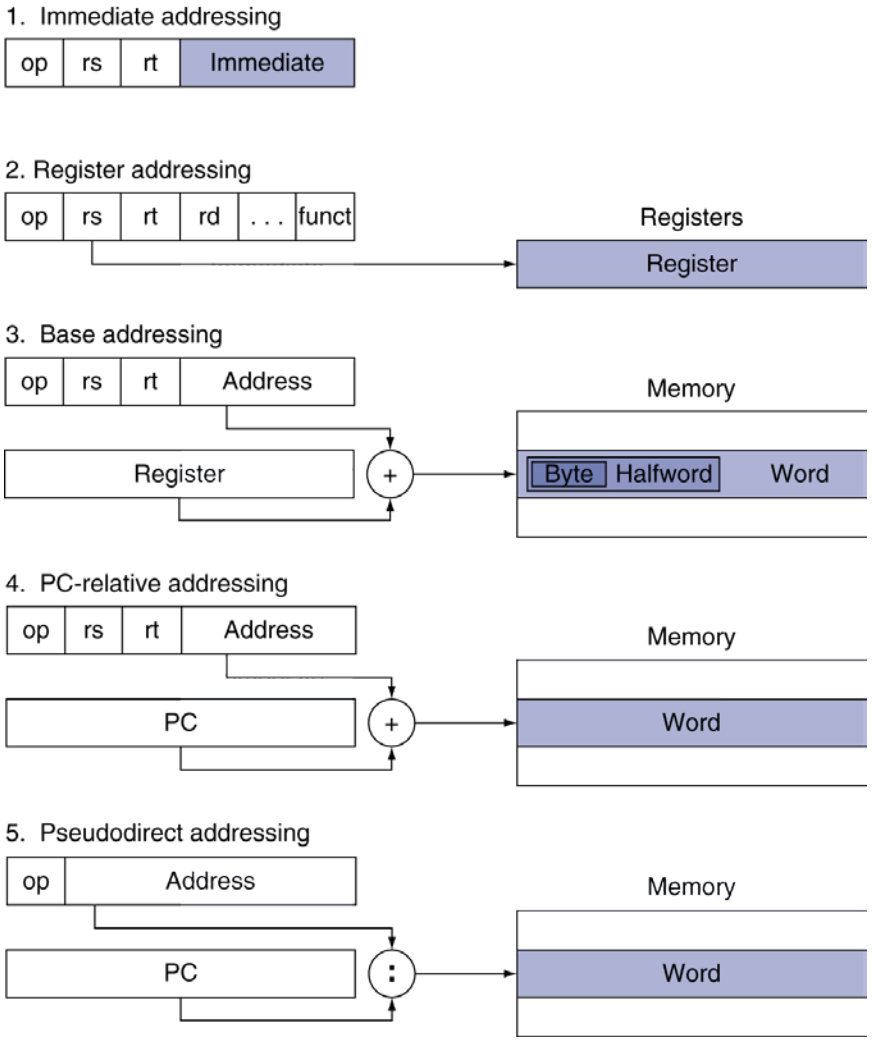


Branching Far Away

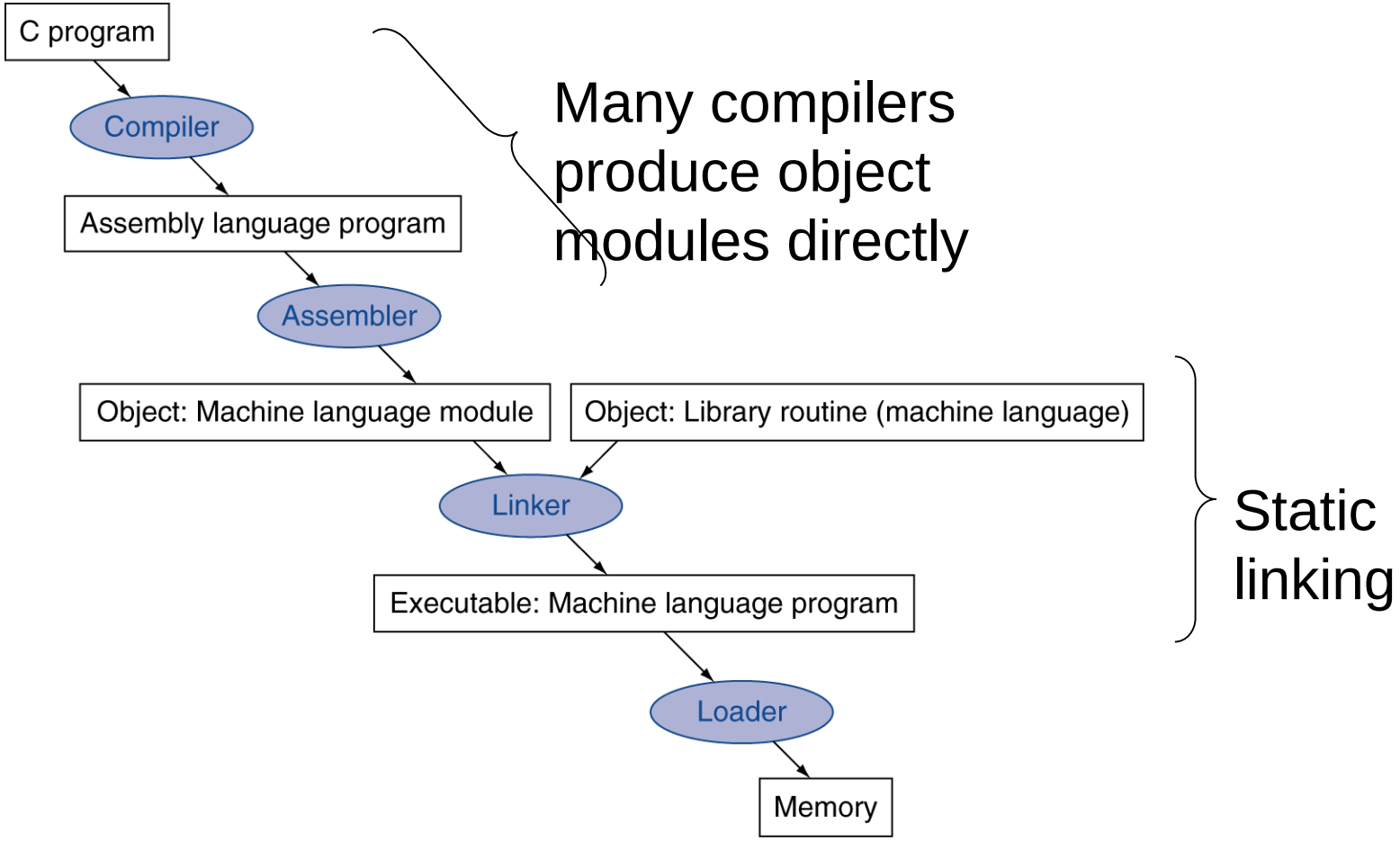
- If branch target is too far to encode with 16-bit offset, assembler rewrites the code
- Example

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

Addressing Mode Summary



Translation and Startup



Assembler Pseudoinstructions

- Most assembler instructions represent machine instructions one-to-one
- Pseudoinstructions: figments of the assembler's imagination

```
move $t0, $t1     →  add $t0, $zero, $t1
blt $t0, $t1, L   →  slt $at, $t0, $t1
                     bne $at, $zero, L
```

  - $at (register 1): assembler temporary

Producing an Object Module

- Assembler (or compiler) translates program into machine instructions
- Provides information for building a complete program from the pieces
  - Header: described contents of object module
  - Text segment: translated instructions
  - Static data segment: data allocated for the life of the program
  - Relocation info: for contents that depend on absolute location of loaded program
  - Symbol table: global definitions and external refs
  - Debug info: for associating with source code

Linking Object Modules

- Produces an executable image
  1. Merges segments
  2. Resolve labels (determine their addresses)
  3. Patch location-dependent and external refs
- Could leave location dependencies for fixing by a relocating loader
  - But with virtual memory, no need to do this
  - Program can be loaded into absolute location in virtual memory space

Loading a Program

- Load from image file on disk into memory
  1. Read header to determine segment sizes
  2. Create virtual address space
  3. Copy text and initialized data into memory
     - *Or set page table entries so they can be faulted in*
  4. Set up arguments on stack
  5. Initialize registers (including $sp, $fp, $gp)
  6. Jump to startup routine
     - *Copies arguments to $a0, … and calls main*
     - *When main returns, do exit syscall*

Dynamic Linking

- Only link/load library procedure when it is called
  - Requires procedure code to be relocatable
  - Avoids image bloat caused by static linking of all (transitively) referenced libraries
  - Automatically picks up new library versions

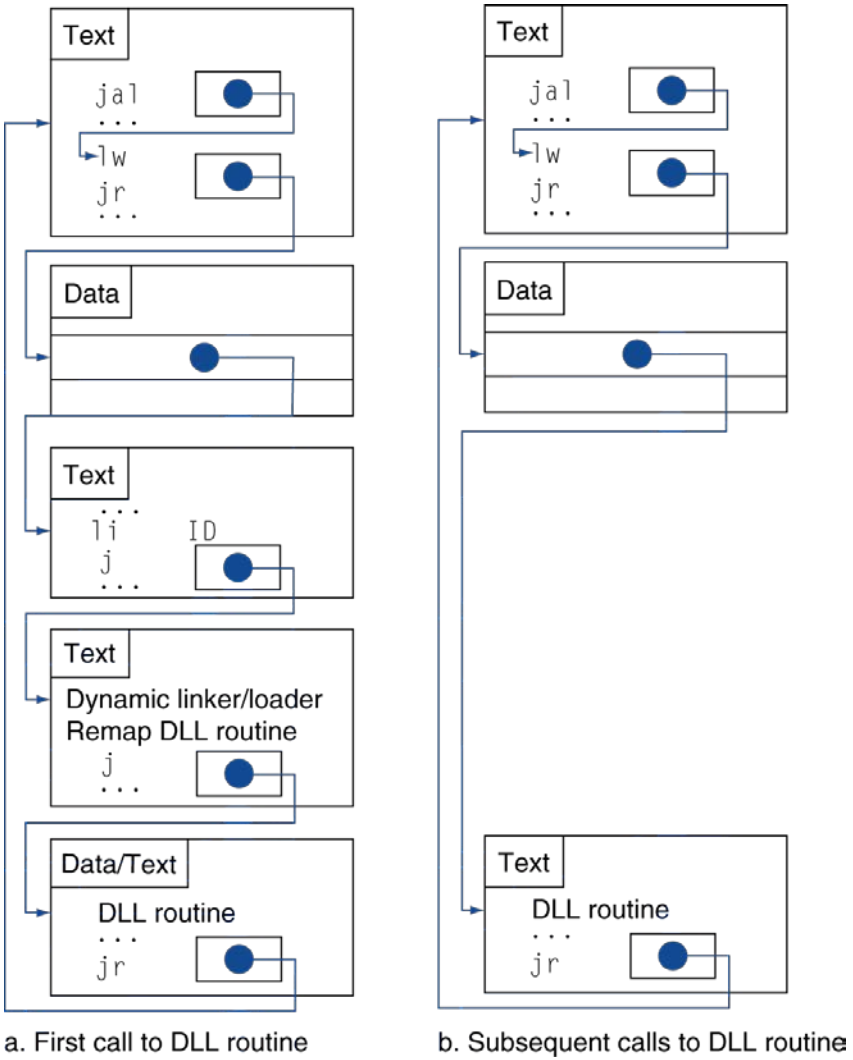
Lazy Linkage

Indirection table

Stub: Loads routine ID, Jump to linker/loader

Linker/loader code

Dynamically mapped code



a. First call to DLL routine

b. Subsequent calls to DLL routine

Starting Java Applications