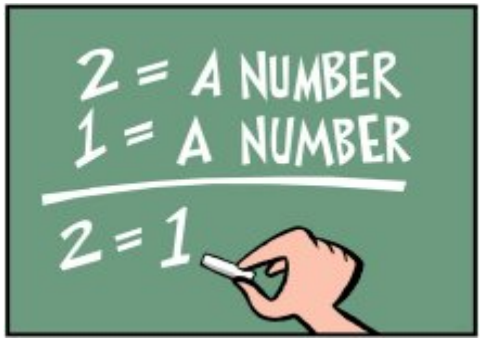
Check yourself

- What is the MIPS assembly language instruction corresponding to the machine instruction with the value $0000\ 0000_{hex}$?
  - A. add $t0, $t0, $t0
  - B. lw $t0, 0($t0)
  - C. sll $0, $0, 0
  - D. srl $0, $0, $0

Quick exercise

- Which of the following appears to be correct?
  - A. More powerful instructions mean higher programmer productivity.
  - B. Write in assembly language to obtain the fastest speeds.
  - C. The importance of commercial binary compatibility means successful instruction sets never change.
  - D. None of the above.



Instruction category

- Each category of MIPS instructions is associated with constructs that appear in programming languages.

Arithmetic and logic instructions

Data transfer instructions

Conditional branch

Unconditional jump

Dealing with data structures like arrays

if statement and loops

Function call and returns

Assignment statements

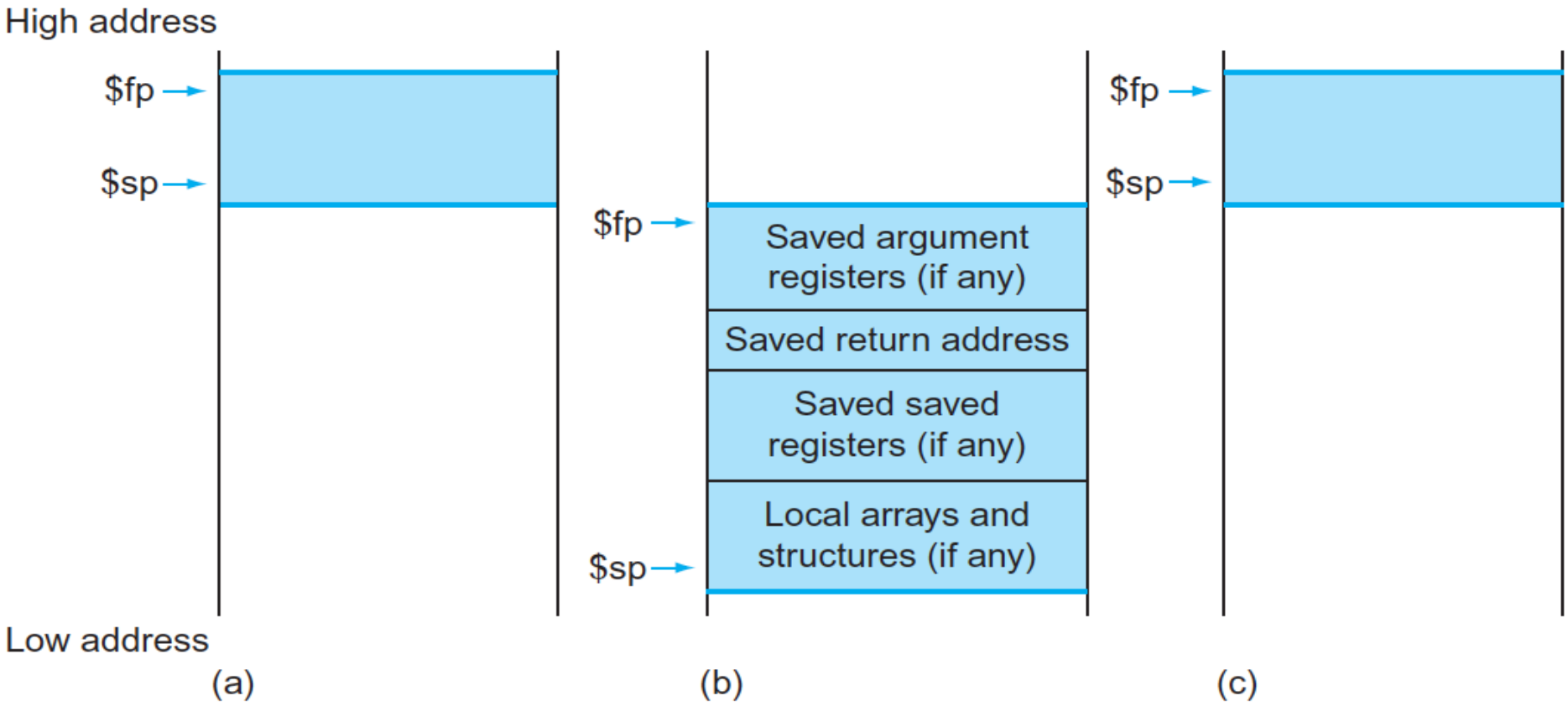
Summary of design principles

- Simplicity favors regularity
  - Keep all instructions a single size, always requiring three register operands in arithmetic instructions, and keeping the register fields in the same place in each instruction format.
- Smaller is faster
  - 32 registers rather than many more
- Make the common case fast
  - PC-relative addressing for conditional branches and immediate addressing for constant operands
- Good design demands good compromises
  - Compromise between providing for larger addresses and constants in instructions and keeping all instructions the same length

Allocating space for new data on the stack

- Stack is used to store local variables
  - Data that do not fit in registers
    - (Local arrays or structures)
- Procedure frame:
  - The segment of the stack containing a procedure's saved registers and local variables.
- Frame pointer (`$fp`)
  - A value denoting the location of the head of a procedure frame.

Stack illustrated

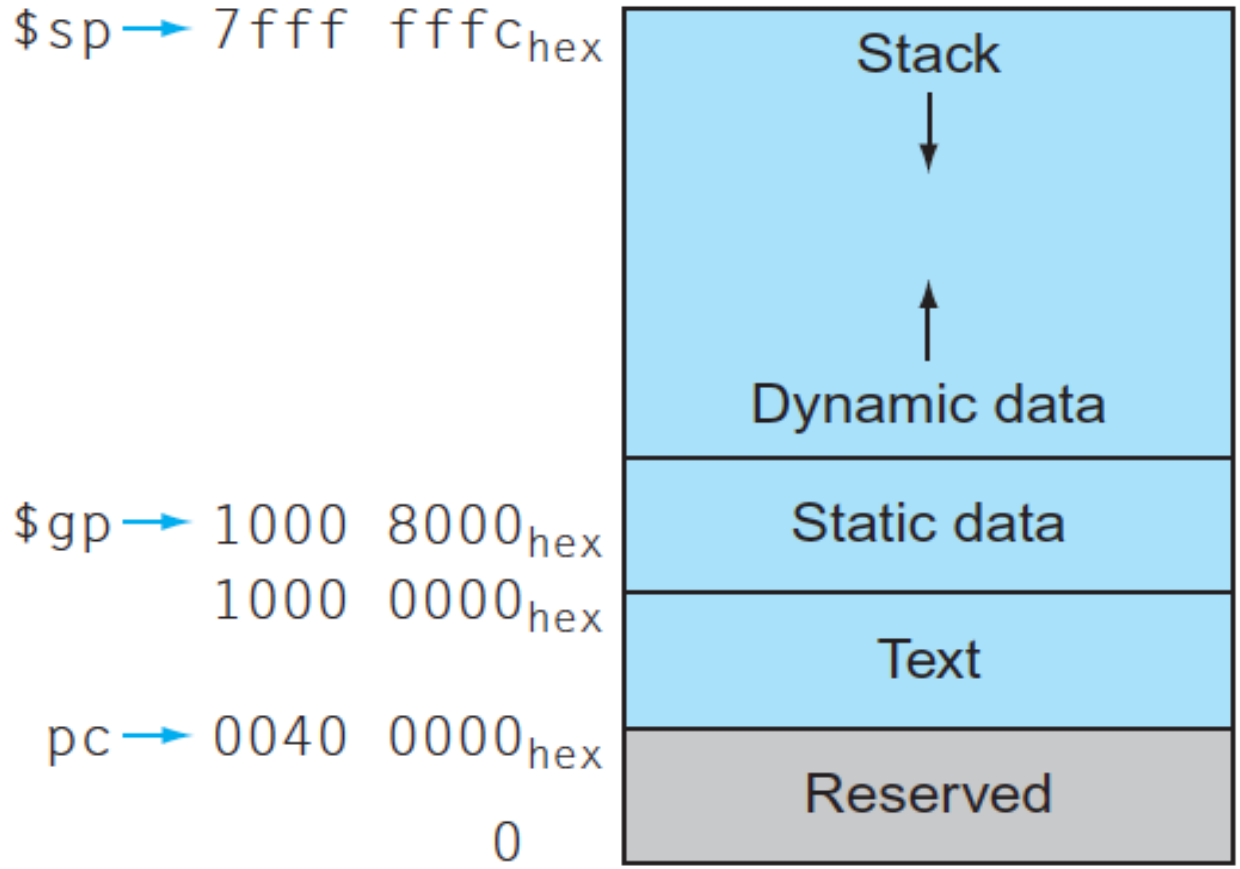


How about more than four arguments?

- The MIPS convention is to place all arguments on the stack, before calling the procedure.
- The use of frame pointer is convenient but not compulsory.
- Compilers are free to choose whether to use frame pointer to handle procedure calls or not.

Allocating space for new data on the heap

- Heap: an area of memory used for dynamic memory allocation



Addressing in branches

- PC-relative addressing

```
bne  $s0,$s1,Exit   # go to Exit if $s0 ≠ $s1
```

- Address in the branch instructions is measured in words (32-bit unit)
  - Branch within -2^{15} to $+2^{15}$ words of the current instruction

- Note: MIPS address is relative to the address of the following instruction (PC+4) as opposed to the current instruction (PC)

Threads and Race Conditions

- A race condition is when the output and/or result of the process/thread is unexpectedly and critically dependent on the sequence or timing of other events.
- The term originates with the idea of two signals racing each other to influence the output first.
- Example: two threads T1 and T2 each want to increment the value of a global integer by one.

Example

1. Integer i = 0; (memory)
2. T1 reads the value of i from memory into register1 : 0
3. T2 reads the value of i from memory into register2 : 0
4. T1 increments the value of i in register1: (register1 contents) + 2 = 2
5. T2 increments the value of i in register2: (register2 contents) + 1 = 1
6. T1 stores the value of register1 in memory : 2
7. T2 stores the value of register2 in memory : 1
8. Integer i = 1; (memory)

Can you provide another operation execution sequence that results in a different value for integer i?

Mutual exclusion

- Mutual exclusion (often abbreviated to mutex) algorithms
  - used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable
  - critical sections
- Dekker's algorithm:

```
f0 := false
f1 := false
turn := 0   // or 1

p0:                                    p1:
    f0 := true                             f1 := true
    while f1 {                             while f0 {
        if turn ≠ 0 {                          if turn ≠ 1 {
            f0 := false                            f1 := false
            while turn ≠ 0 {                       while turn ≠ 1 {
            }                                      }
            f0 := true                             f1 := true
        }                                      }
    }                                      }

   // critical section                    // critical section
   ...                                    ...
   // remainder section                   // remainder section
   turn := 1                              turn := 0
   f0 := false                            f1 := false
```

Problems with Dekker's algorithm

- Dekker's algorithm guarantees mutual exclusion and is highly portable between languages and machine architectures.
- One disadvantage is that it is limited to two processes and makes use of busy waiting instead of process suspension.
  - The use of busy waiting suggests that processes should spend a minimum of time inside the critical section

MIPS support for synchronization

- A pair of instructions
  - Load linked (ll)
  - Store conditional (sc)

- The two instructions are used in sequence. If the contents of memory location specified by ll are changed before the sc to the same address occurs, then the sc fails.

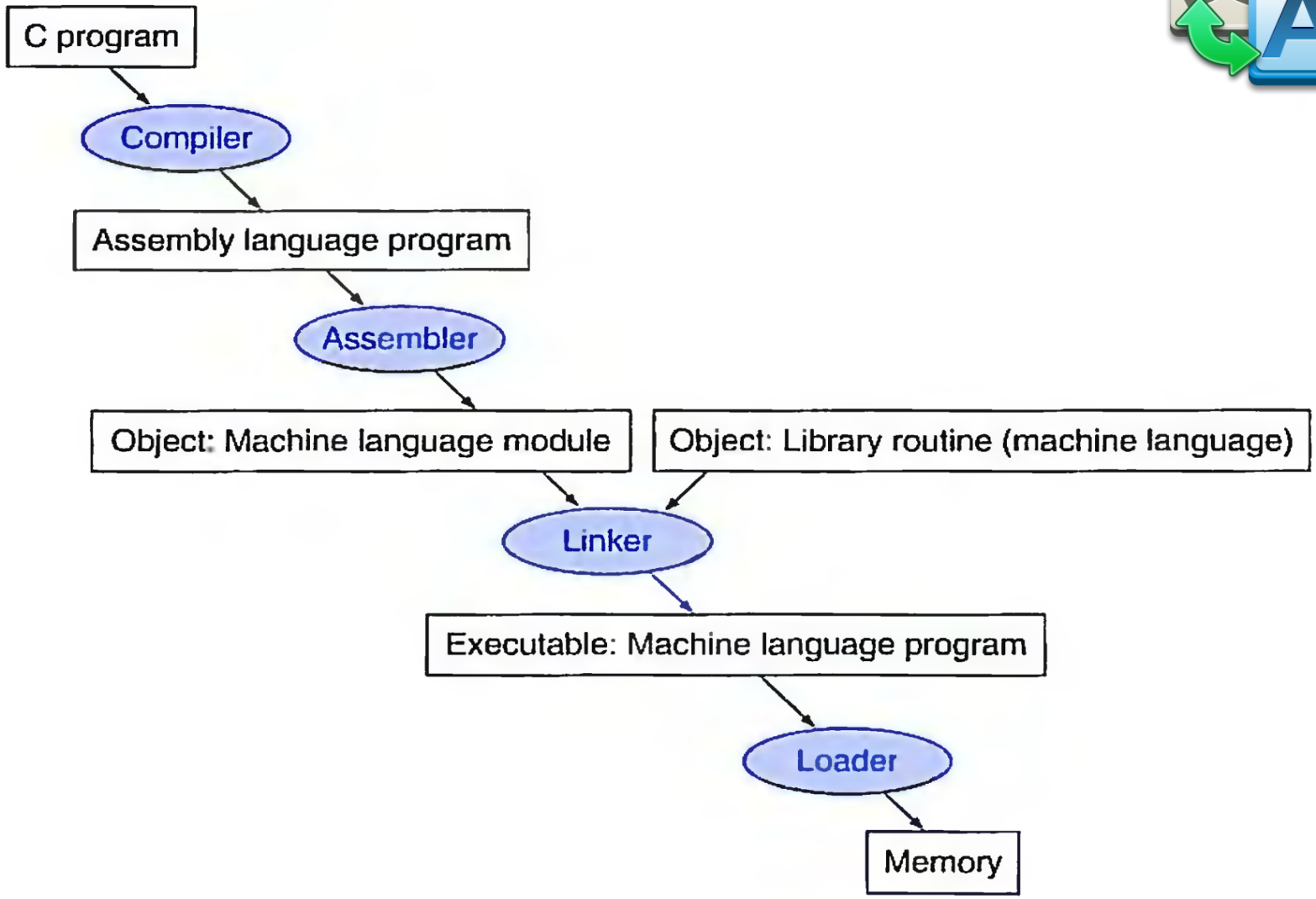
- Example:

```
try: add $t0,$zero,$s4     ;copy exchange value
     ll    $t1,0($s1)      ;load linked
     sc    $t0,0($s1)      ;store conditional
     beq   $t0,$zero,try   ;branch store fails
     add   $s4,$zero,$t1   ;put load value in $s4
```

More code here, using $t1

  - $t0=1 if sc is successful, otherwise $t0=0

Translation hierarchy for c



Compiler

- A program that transforms the c program into an assembly language program.
- High-level language programs take many fewer lines of code than assembly language, so programmer productivity is much higher.
- In 1975 many operating systems and assemblers were written in assembly language because memories were small and compilers were inefficient.
- Now, vast memory capacity has reduced program size concerns. Optimized compilers can produce assembly programs nearly as good as assembly language expert (sometimes even better for large programs)

Assembler

- Translate assembly language programs into machine code (object).
- Pseudoinstructions
  - A common variation of assembly language instructions often treated as if it were an instruction in its own right.
  - Hardware does not implement pseudoinstructions.
  - Pseudoinstructions simplify translation and programming.
- Example
  - move is a pseudoinstruction that copies the contents of one register to another.

```
move $t0,$t1          # register $t0 gets register $t1
```

```
add  $t0,$zero,$t1    # register $t0 gets 0 + register $t1
```

- Other pseudoinstruction examples:
  - Branch on less than (blt)
    - bgt, bge, ble
  - Load immediate (li)

- One register $at is preserved by the assembler in order to implement pseudoinstructions.

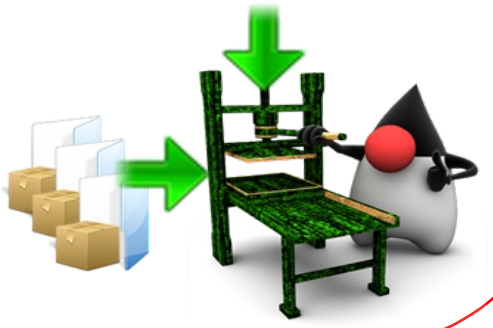
- Which code below is suitable for implementing "li $a0, 5"
  - A. ori $a0, $zero, 5
  - B. andi $a0, $zero, 5
  - C. addi $a0, $zero, 5
  - D. addiu $a0, $zero, 5

Linker

- A systems program that combines independently assembled machine language programs and resolves all undefined labels into an execute file.

- Execute file: a program that can be directly executed on a computer.

- Steps of the linker:
  - Place code and data modules in memory.
  - Determine the address of data and instruction labels.
  - Patch both the internal and external references.

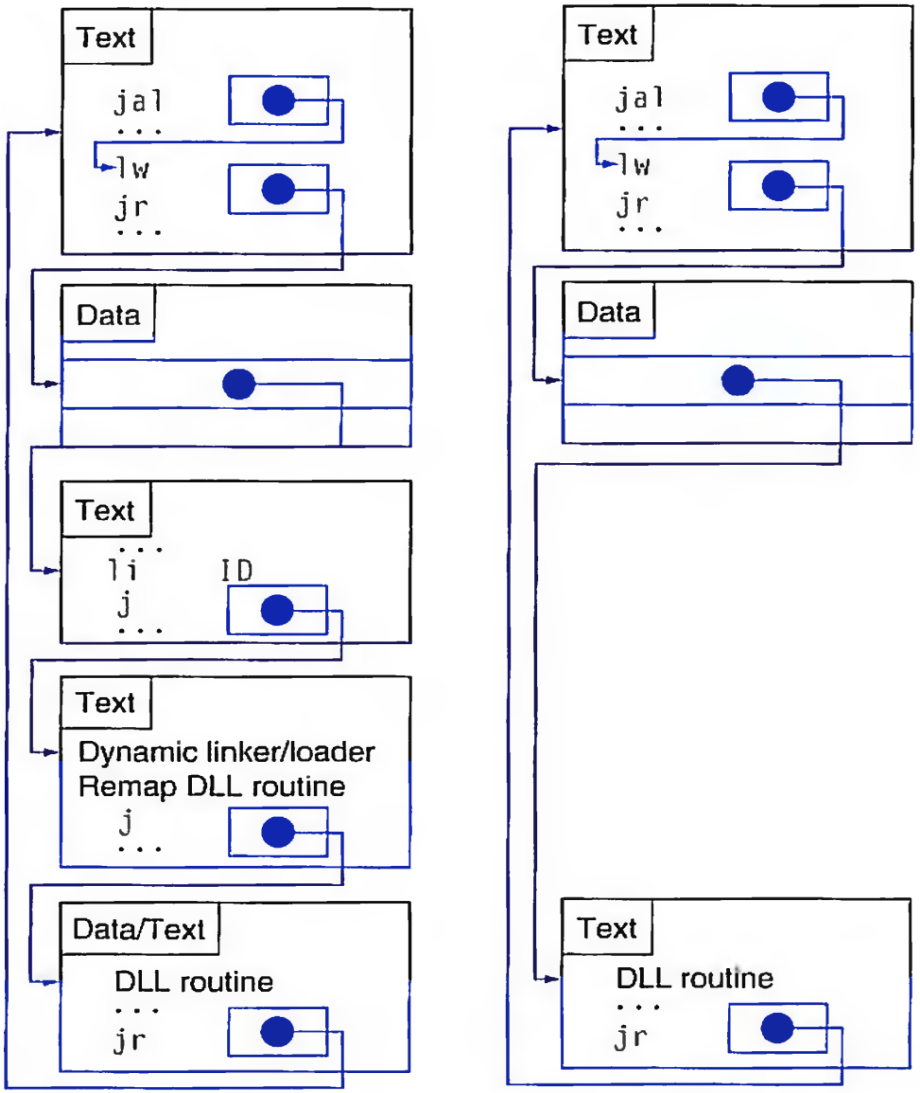


Loader

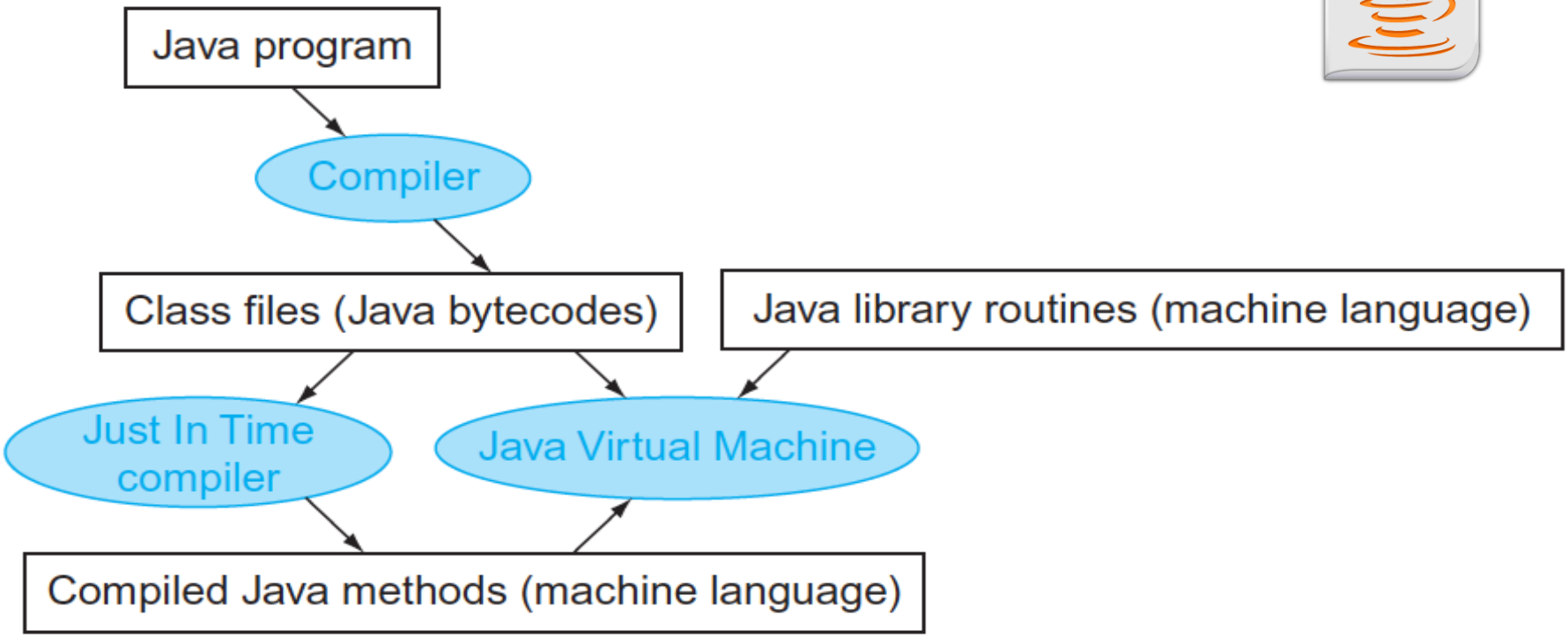
- A systems program that places an executable program in main memory so that it is ready to execute
- Steps of the loader:
  - Read the executable file header to determine size of text and data segments.
  - Create an address space large enough for the text and data.
  - Copy the instructions and data into memory.
  - Initialize the machine registers and set the stack pointer to the first free location.
  - Jump to the start-up routine that copies the parameters into the argument registers and calls the main routine of the program.

Dynamically linked libraries

- Library routines that are linked to a program during execution.



Translation hierarchy for Java



- Just in time compiler (JIT): a compiler that operates at runtime, translating the interpreted code segments into the native code of the computer.

Array versus pointers

- Example: clear a sequence of words in memory.

- Approach 1:

```
clear1(int array[], int size)
{
  int i;
  for (i = 0; i < size; i += 1)
       array[i] = 0;
}
```

- Approach 2:

```
clear2(int *array, int size)
{
  int *p;
  for (p = &array[0]; p <
&array[size]; p = p + 1)
       *p = 0;
}
```

MIPS code for approach 1

- Assume that
  - array -> $a0
  - size -> $a1
  - i -> $t0

```
clear1(int array[], int size)
{
  int i;
  for (i = 0; i < size; i += 1)
       array[i] = 0;
}
```

```
      move  $t0,$zero       # i = 0
loop1: sll  $t1,$t0,2       # $t1 = i * 4
      add   $t2,$a0,$t1     # $t2 = address of array[i]
      sw    $zero, 0($t2)   # array[i] = 0
      addi  $t0,$t0,1       # i = i + 1
      slt   $t3,$t0,$a1     # $t3 = (i < size)
      bne   $t3,$zero,loop1 # if (i < size) go to loop1
```

MIPS code for approach 2

- Assume that
  - array -> $a0
  - size -> $a1
  - p -> $t0

```
clear2(int *array, int size)
{
  int *p;
  for (p = &array[0]; p <
&array[size]; p = p + 1)
        *p = 0;
}
```

```
   move   $t0,$a0          # p = address of array[0]
   sll    $t1,$a1,2        # $t1 = size * 4
   add    $t2,$a0,$t1      # $t2 = address of array[size]
loop2:sw$zero,0($t0)       # Memory[p] = 0
   addi   $t0,$t0,4        # p = p + 4
   slt    $t3,$t0,$t2      # $t3 = (p<&array[size])
   bne    $t3,$zero,loop2  # if (p<&array[size]) go to loop2
```

Comparing the two approaches

```
      move  $t0,$zero      # i = 0
loop1:sll   $t1,$t0,2      # $t1 = i * 4
      add   $t2,$a0,$t1    # $t2 = &array[i]
      sw    $zero, 0($t2)  # array[i] = 0
      addi  $t0,$t0,1      # i = i + 1
      slt   $t3,$t0,$a1    # $t3 = (i < size)
      bne   $t3,$zero,loop1# if () go to loop1
```

```
      move  $t0,$a0        # p = & array[0]
      sll   $t1,$a1,2      # $t1 = size * 4
      add   $t2,$a0,$t1    # $t2 = &array[size]
loop2:sw    $zero,0($t0)   # Memory[p] = 0
      addi  $t0,$t0,4      # p = p + 4
      slt   $t3,$t0,$t2    # $t3=(p<&array[size])
      bne   $t3,$zero,loop2# if () go to loop2
```

- The pointer version reduces the instructions executed per iteration from 6 to 4.

Modern optimizing compilers can produce code for the array version that is just as good as the pointer version.

Popularity of different MIPS instructions

Instruction class	MIPS examples	HLL correspondence	Frequency	
			Integer	Ft. pt.
Arithmetic	`add, sub, addi`	Operations in assignment statement s	16%	48%
Data transfer	`lw, sw, lb, lbu, lh, lhu, sb, lui`	References to data structures, such as arrays	35%	36%
Logical	`and, or, nor, andi, ori, sll, srl`	Operations in assignment statement s	12%	4%
Conditional branch	`beq, bne, slt, slti, sltiu`	*If* statements and loops	34%	8%
Jump	`j, jr, jal`	Procedure calls, returns, and *case/switch* statements	2%	0%

Summary of design principles

- Simplicity favors regularity
  - Keep all instructions a single size, always requiring three register operands in arithmetic instructions, and keeping the register fields in the same place in each instruction format.
- Smaller is faster
  - 32 registers rather than many more
- Make the common case fast
  - PC-relative addressing for conditional branches and immediate addressing for constant operands
- Good design demands good compromises
  - Compromise between providing for larger addresses and constants in instructions and keeping all instructions the same length