Computer Organization

605.204

Module Four

Part One

The Assembler

Module Four

- Part One

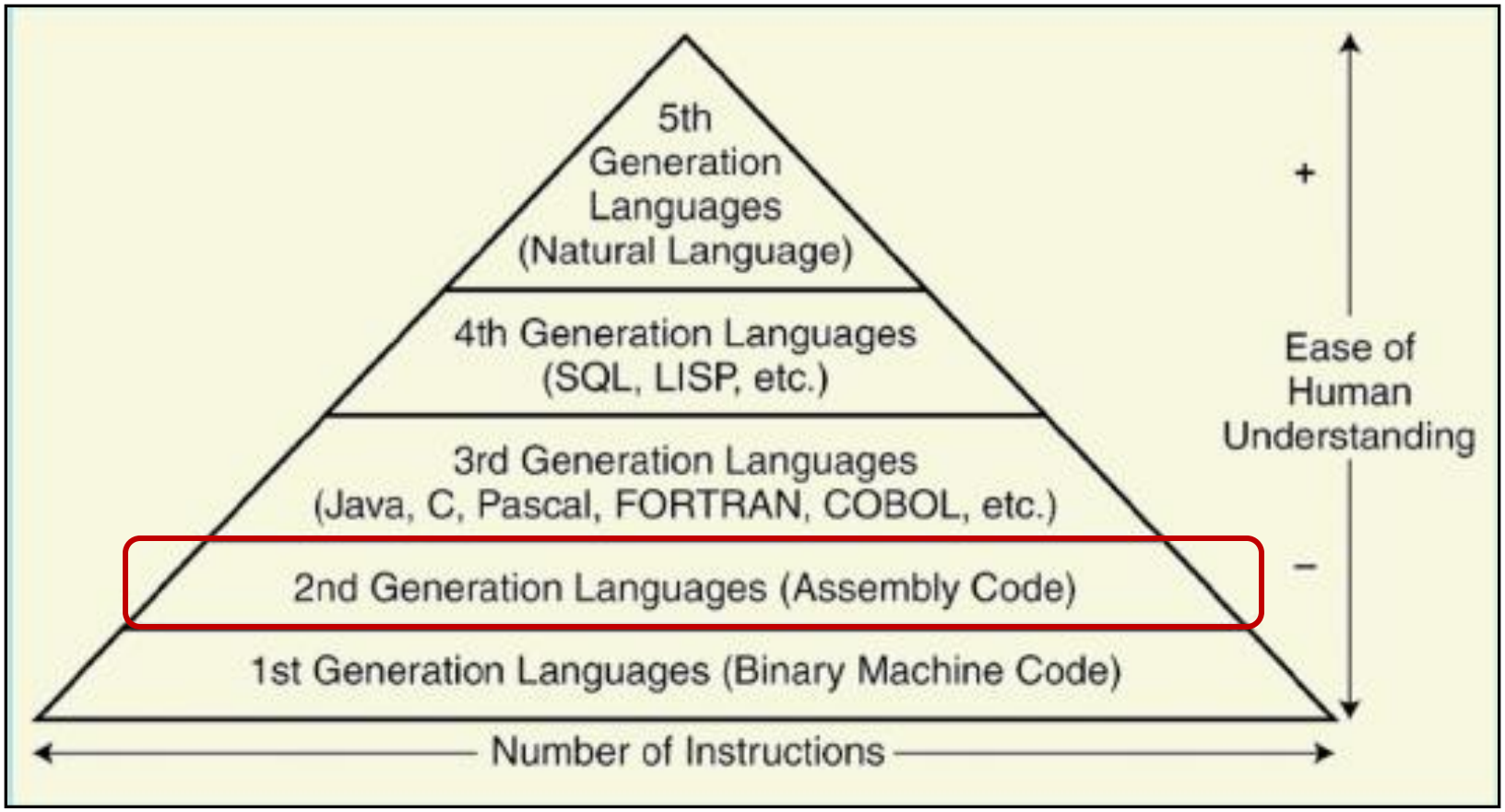
- This week:
- Assembly Language review
- Assembler
  - Basic Functions of every Assembler
  - The Process
  - The Organization
- Object file
- Additional Assembler features

MIPS Programming Language

The Big Picture

Assembly language is a programming language. Its principal difference from high-level languages such as BASIC, Java, and C is that assembly language provides only a few, simple types of data and control flow. Assembly language programs do not specify the type of value held in a variable. Instead, a programmer must apply the appropriate operations (e.g., integer or floating-point addition) to a value. In addition, in assembly language, programs must implement all control flow with *go tos*. Both factors make assembly language programming for any machine—MIPS or 80x86—more difficult and error-prone than writing in a high-level language.

Language Generations



Assembly Language review

- From last week:

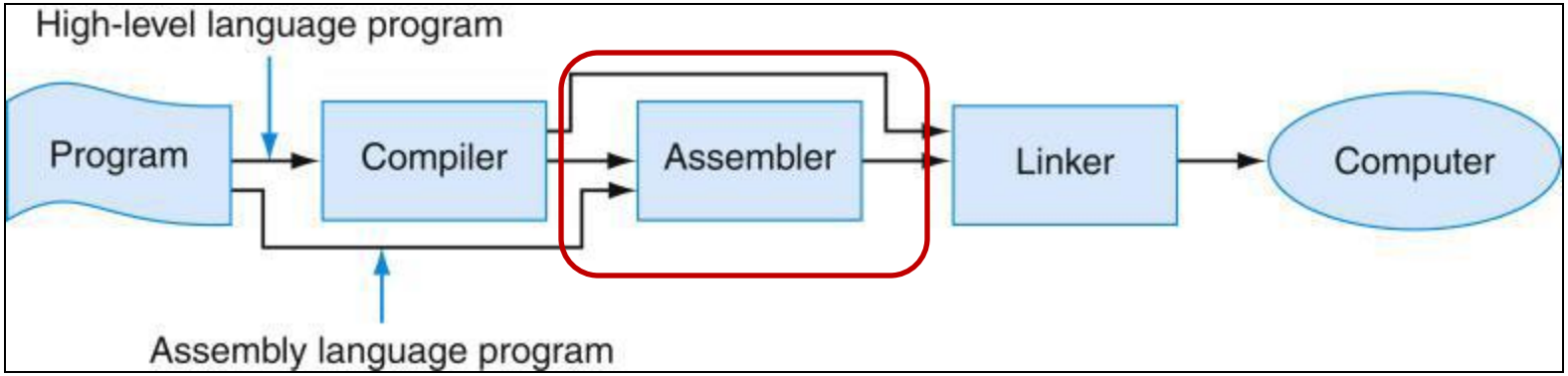
Category	Instruction	Example
Arithmetic	add	`add $s1, $s2, $s3`
	subtract	`sub $s1, $s2, $s3`
	add immediate	`addi $s1, $s2, 100`
Data transfer	load word	`lw $s1, 100($s2)`
	store word	`sw $s1, 100($s2)`
	load byte	`lb $s1, 100($s2)`
	store byte	`sb $s1, 100($s2)`
	load upper immediate	`lui $s1, 100`

Category	Instruction	Example
Conditional branch	branch on equal	`beq $s1, $s2, 25`
	branch on not equal	`bne $s1, $s2, 25`
	set on less than	`slt $s1, $s2, $s3`
	set less than immediate	`slti $s1, $s2, 100`
Unconditional jump	jump	`j 2500`
	jump register	`jr $ra`
	jump and link	`jal 2500`

- MIPS Assembly Summary

Translating

- People language to machine

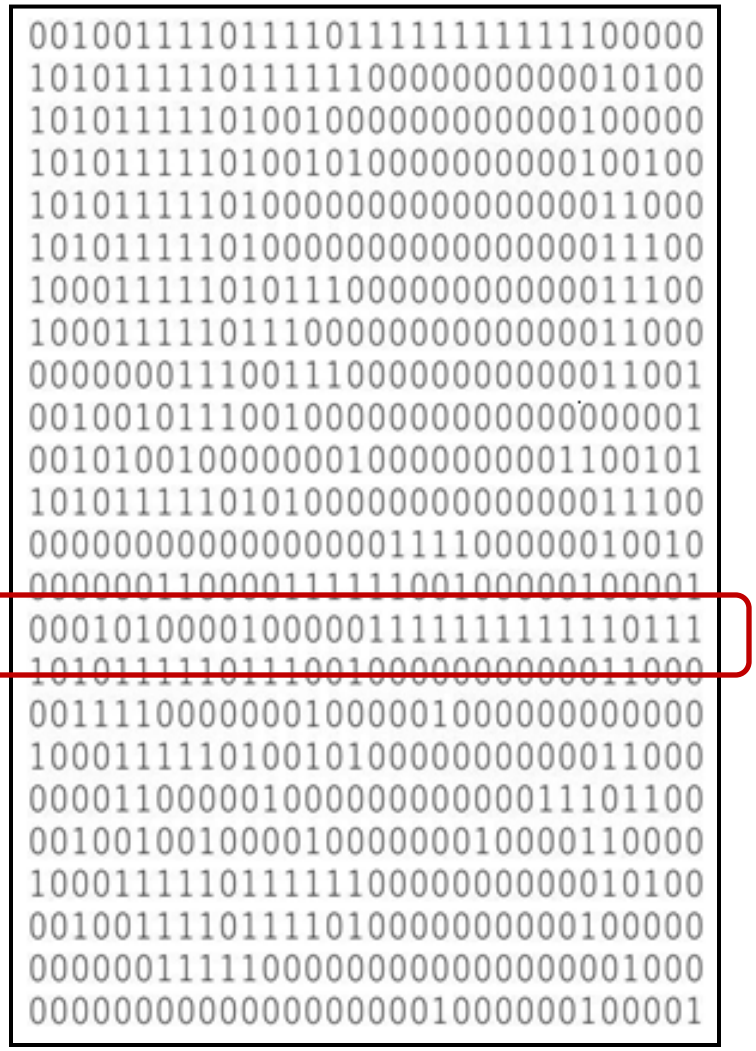


MIPS Program

```
            .globl    main        # sum of the integers from 1 to 100
            .text
main:       addi $t0, $zero, 0    # I is zero
            addi $s0, $zero, 0    # sum is zero
            addi $t1, $zero, 100  # set the limit value (100)
loop:       addi $t0, $t0, 1      # I = I + 1
            add  $s0, $s0, $t0    # sum = sum + I
            blt  $t0, $t1, loop   # I < 100 loop to do again
            addi $v0, $zero, 4    # print string
            la   $a0, str         # the text for output
            syscall               # call opsys
            addi $v0, $zero, 1    # print integer
            add  $a0,$zero, $s0   # the integer is sum
            syscall               # call opsys
            jr $ra                # finished .. return
            .data
str:        .asciiz "The sum of 1 .. 100 is "
```

Machine readable program

- MIPS
- Calculates SUM
- Prints result



Program as Assembly

- MIPS
- Calculates SUM
- Prints result
- No labels
- No comments

```
addiu     $29, $29, -32
sw        $31, 20($29)
sw        $4, 32($29)
sw        $5, 36($29)
sw        $0, 24($29)
sw        $0, 28($29)
lw        $14, 28($29)
lw        $24, 24($29)
multu     $14, $14
addiu     $8, $14, 1
slti      $1, $8, 101
sw        $8, 28($29)
mflo      $15
addu      $25, $24, $15
bne       $1, $0, -9
sw        $25, 24($29)
lui       $4, 4096
lw        $5, 24($29)
jal       1048812
addiu     $4, $4, 1072
lw        $31, 20($29)
addiu     $29, $29, 32
jr        $31
move      $2, $0
```

Program as Assembly

- MIPS
- Calculates SUM
- Prints result
- With labels
- With Directives
- No comments

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra

        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Same program C language

- Let us explain

```
    #include <stdio.h>
int
main (int argc, char *argv[])
{
    int i;
    int sum = 0;
    for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
    printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Some Directives

- **.text** indicates that succeeding lines contain instructions.
- **.data** indicates that succeeding lines contain data.
- **.align** n indicates that the items on the succeeding lines should be aligned on a 2*n* byte boundary.
- **.align 2** means the next item should be on a word boundary.
- **.globl** main declares that main is a global symbol that should be visible to code stored in other files.
- **.asciiz** stores a null-terminated string in memory.

MIPS Assembly Language

- Directives to the SPIM Assembler

  .data, .text, .globl, .word, .ascii, .space

- Instruction set

Data transfer	LW / SW
Arithmetic and logic	ADD / AND
Program sequencing	BEQ / J

Summary

- Assembly Language review

- Next:
  - Assembler

    Basic Functions of every Assembler

    The Process

    The Organization