

LECTURE 15: MIPS (FOR LAB 10)

Computer Systems and Networks

Dr. Pallipuram

(vpallipuramkrishnamani@pacific.edu)

Class Today

MIPS Basics: Arithmetic and Branching Instructions

First, help me grab this cup and sip coffee

All that I understand are move, go, bc, sip, pb, ro

move <direction (left or right)>, <# of steps>

go <object> #go is **g**rab **o**bject

bc <distance> #bc is **b**ring **c**lose distance amount

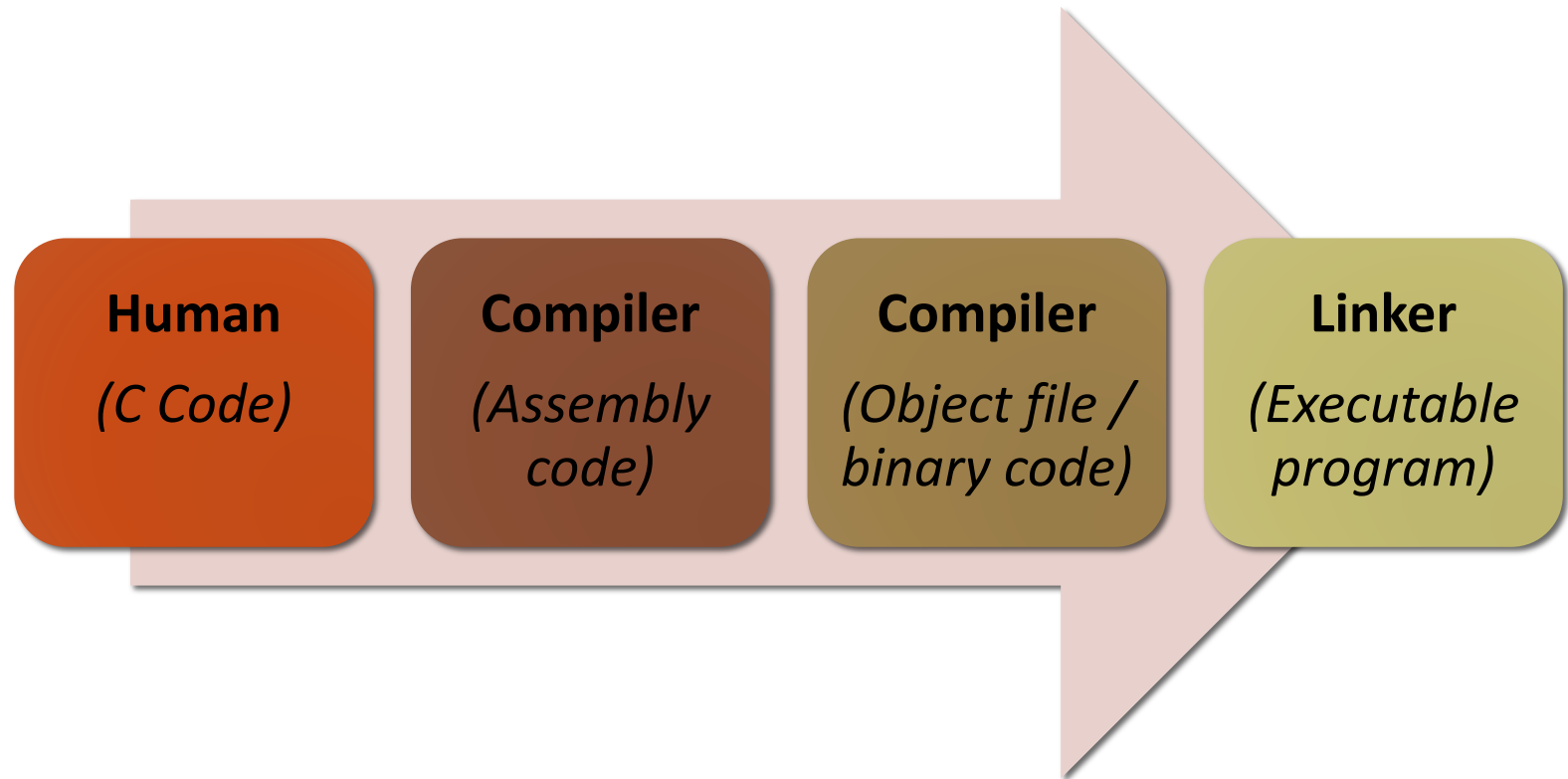
For ref., distance=3 from arm to mouth

sip #take a sip

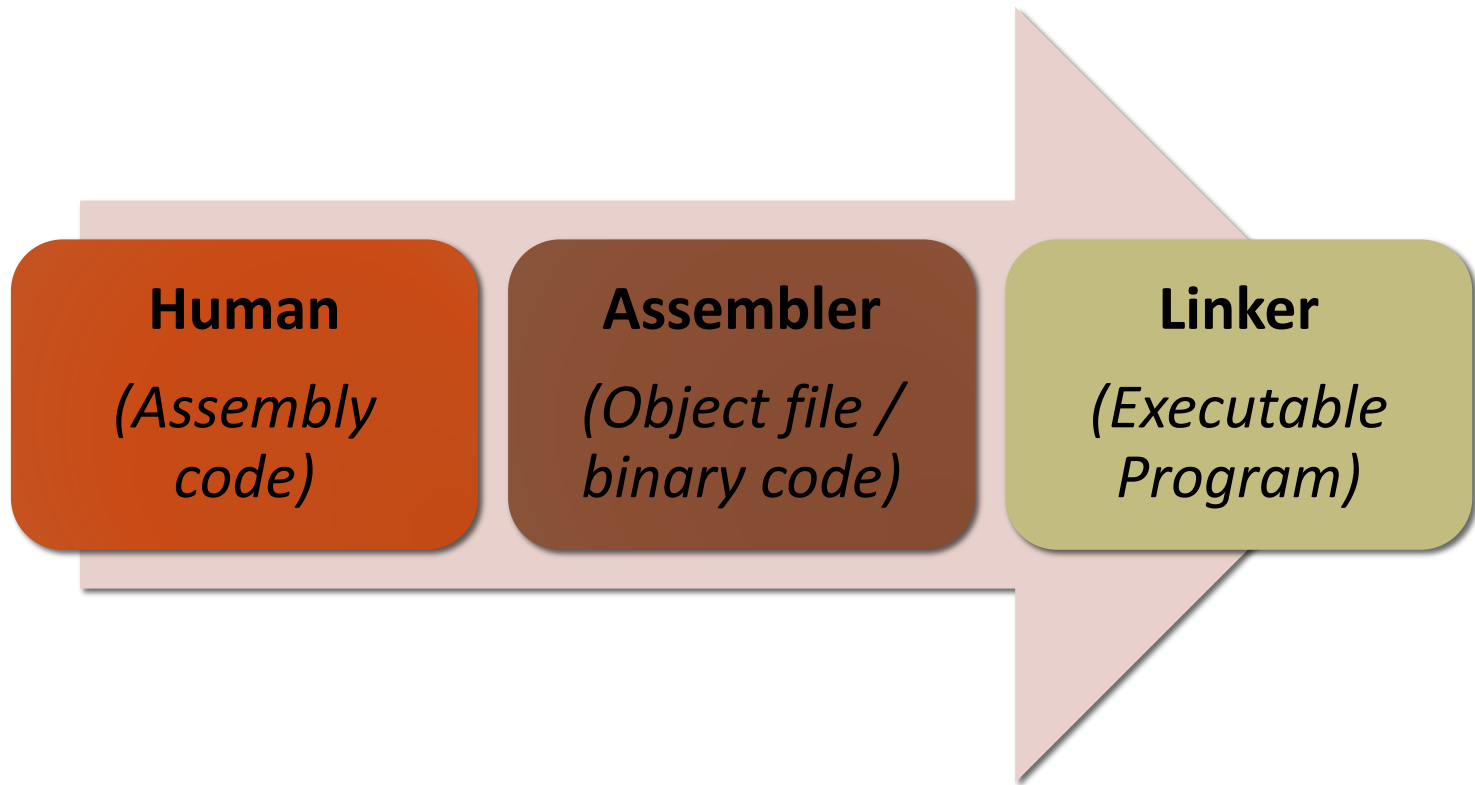
pb <distance> #pb is **p**ut **b**ack at distance (a number)

ro <object> #ro is **r**elease **o**bject

Class to Date



Class Now



MIPS Design

RISC – **What does this mean?**

- **Reduced Instruction Set Computing**
- Simplified design for instructions
- Use more instructions to accomplish same task
 - But each instruction runs much faster!

32 bits (originally) – **What does this mean?**

- 1 “word” = 32 bits
- Size of data processed by an integer add instruction
- New(er) MIPS64 design is 64 bits, but we won’t focus on that

Why should I learn MIPS?

Computer Science majors -- Compilers optimize assembly to improve performance. One day, you'll develop such efficient compilers

ECPE majors -- Assembly language will help you design Microcontroller applications. One day, you may be the coder for a microcontroller that goes on a space mission

CS and ECPE majors -- You will collaborate and develop the next-generation processors

ISA Definition

Instruction Set Architecture is an interface between hardware and software

Instruction Set Architecture defines format for arithmetic/logic instructions, addressing instructions, and branching instructions. We will broadly divide ISA into these classes:

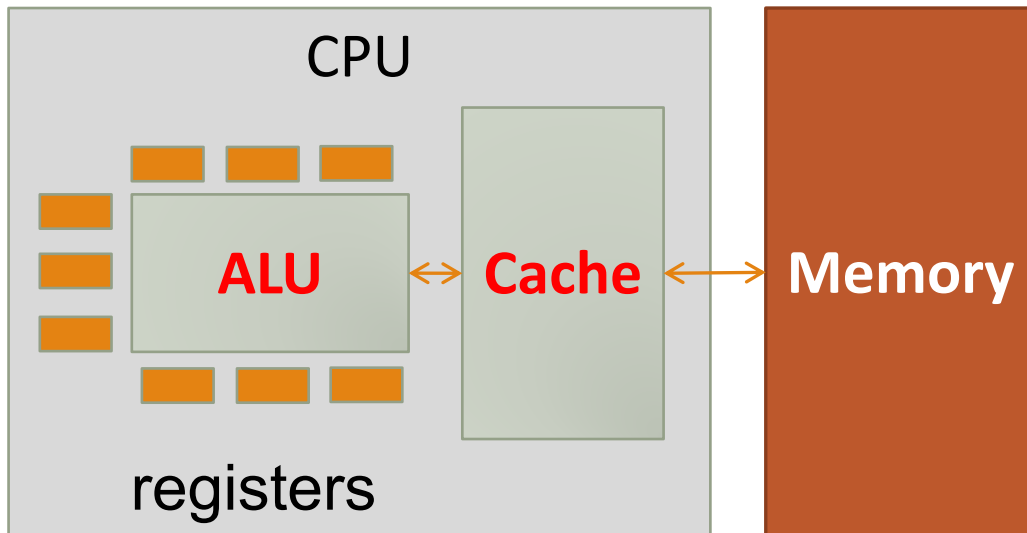
- Arithmetic** instructions

- Branching** instructions

- Addressing** instructions

Recap of a typical Processor

We will actively use registers to fetch data and perform ALL arithmetic and logic computations



Load values into registers and try performing everything with them

1. Registers

- On the CPU itself
- Very close to ALU
- Tiny
- Access time: 1 cycle

2. Memory

- Off-chip
- Large
- Access time: 100+ cycles

Registers in MIPS

MIPS design: **32 integer registers**, each holding **32 bits**

- “Word size” = 32 bits

Name	Use
\$zero	Constant value: ZERO
\$s0-\$s7	Local variables
\$t0-\$t9	Temporary registers

This is only 19 – where are the rest of the 32?

- Reserved *by convention* for other uses
- We’ll learn a few more later...

Arithmetic Instructions

Add Instructions

Add (perform addition using registers)

`add <destination>, <source 1>, <source 2>`

E.g.:

`add $t0, $s1, $s2 #t0 = s1 + s2`

`add $s1, $s1, $s2 #s1 = s1 + s2`

Add Immediate

`addi <destination>, <source 1>, <signed value>`

E.g.:

`addi $s0, $s1, -2 #s0 = s1 - 2`

`addi $s1, $s1, 1 #s1++`

`addi $t1, $zero, 6 # t1 = 6`

Sub Instructions

Sub (perform addition using registers)

`sub <destination>, <source 1>, <source 2>`

E.g.:

`sub $t0, $s1, $s2 #t0 = s1 - s2`

`sub $s1, $s1, $s2 #s1 = s1 - s2`

Problem 1: Convert this snippet to assembly

```
g = 15; //view it as g=15+0
h = 20;
i = 5;
j= 18;
f = (g + h) - (i+j);
```

Assume Map:

```
$s0 = g
$s1 = h
$s2 = i
$s3 = j
$s4 = f
```

Currently, your computer only knows add and sub instructions!

Branching Instructions

ASSEMBLY FOR IF-ELSE, FOR, WHILE, AND SUCH..

A solid orange horizontal bar at the bottom of the slide.

How would you explain this to a non-major? Draw flowchart

if-else statements and loops result in branching of control

```
if ( (A>=B) || A>6)
    C=A;
else
    C=B-A;
//outside if-else
```


Branching (“Go to”) Instructions

Branch on equal:

`beq <register 1>, <register 2>, label` #if
register 1 is equal to register 2, then branch to label.

NOTE: If the condition is true, the branching will happen. If the condition is not true, then the next instruction will be executed

Branching (“Go to”) Instructions

Branch on not equal:

`bne <register 1>, <register 2>, label #if
register 1 is not equal to register 2, then branch to
label.`

Branch on greater than:

`bgt <register 1>, <register 2>, label #if
register 1 is greater than register 2, then branch to
label.`

Branching Instructions

Branch on greater than equal to:

`bge <register 1>, <register 2>, label #if`
register 1 is greater than or equal to register 2, then
branch to label.

Branch on less than:

`blt <register 1>, <register 2>, label #if`
register 1 is less than register 2, then branch to label.

Branch on less than or equal to:

`ble <register 1>, <register 2>, label #if`
register 1 is less than or equal to register 2, then
branch to label.

Branching Instructions

Unconditional jump to a label:

`j label` #just jump to this label and proceed there onwards

Problem 2: Convert the following to Assembly

```
if ( (A>=B) || A>6 )  
    C=A;  
else  
    C=B-A;
```

Currently, your computer only understands add, sub, and some branching.

Assume Map:

\$s0 = C

\$s1 = A

\$s2 = B

Problem 3: Convert the following to Assembly

```
sum=0;  
for (i=0; i<10; i++)  
{  
    sum+=i;  
}
```

Currently, your computer only understands add, sub, and some branching. Draw a flowchart first.

Assume Map:

\$s0 = sum

\$t0 = i

Problem 4: Convert the following to Assembly

```
sum=0;
for (i=0; i<10; i++)
{
    j=i;
    while (j<2*i)
    {
        sum=sum+j;
        j++;
    }
}
```

Currently, your computer only understands add, sub, and some branching.

Assume Map:

\$s0 = sum

\$s1 = i

\$s2 = j

You are prepared
for Lab 10 Parts 1,
2, and 3

MAKE PROGRESS ON THESE UNTIL NEXT CLASS

USE THIS CODE AS A STUB. Also on Lab 10 Page (a link)

```
# Declare main as a global function
# Pound is for comments
.globl main
# All program code is placed after the
# .text assembler directive
.text

# The label 'main' represents the starting point
main:

    #fill out main here

# Exit program by syscall
    li $v0, 10 # select exit syscall
    syscall # Exit the program
Assembler directive .data
    .data

# Reserves space in memory for word with initial value 0
# used to store Z in memory
value: .word 0
```

Storing a value in Memory

In the end of the stub, note a `.data` section

E.g.:

```
.data
value: .word 0 # a word set to zero
msg: .asciiz "Hello World!\n" #a string
pow2: .word 1, 2, 4, 8, 16, 32, 64, 128
#an initialized array of integers
```

Next Class – MIPS for Labs 10 and 11

Arrays and memory accesses in Assembly

Writing Functions in assembly

For Next Class

Carefully go through MIPS example programs (see Lab 10 page). Teach yourself:

- `printf` in MIPS
- `scanf` in MIPS

Finish SPIM tutorial on Lab 10 page. In the next class, I will assume you know your way around SPIM