



Assignment Project Exam Help

CPU Instructions  
<https://tutorcs.com>

CS 154: Computer Architecture  
WeChat: cstutorcs

Lecture 4  
Winter 2020

Ziad Matni, Ph.D.  
Dept. of Computer Science, UCSB

# Administrative

---

- Lab 01 – due today!

Assignment Project Exam Help

- Lab 02 – description will be out soon!

<https://tutorcs.com>

WeChat: cstutorcs

# Lecture Outline

---

- Instruction Set Architectures (ISA)

Assignment Project Exam Help

- MIPS instruction formats  
<https://tutorcs.com>
- Refresher on some other MIPS instructions

WeChat: cstutorcs

*Reference material from CS64 – I'll be going over this a little fast...*

# Other Factors to CPU Performance: Power Consumption

---

Market trends DEMAND that power consumption of CPUs keep decreasing.

**BUT** Power and Performance DON'T always go together...

<https://tutorcs.com>

- Power = Capacitive Load x Voltage<sup>2</sup> x Clock Frequency
- So:
  - Decreasing Voltage helps to get lower power, but it can make individual logic go slower!
  - Increasing clock frequency helps performance, but increases power!
- It's a dilemma that has contributed to Moore's Law "plateau"

# Other Factors to CPU Performance: Multiple Processors

---

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
  - Hardware executes multiple instructions at once
  - Hidden from the programmer
- Hard to do
  - Programming for performance
  - Load balancing
  - Optimizing communication and synchronization

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

# Pitfalls: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

**Assignment Project Exam Help**

- Your benchmark time is 100, 80 of which comes from a part of the CPU that you want to improve by a factor of  $n$ , so:

$$T_{\text{improved}} = (80 / n) + 20$$

**WeChat: cstutorcs**

- If you wanted to improve your overall  $T$  by a factor of 2 (i.e. drop total from 100 to 50), then you'd need to make  $n = 2.7$

because  $50 = (80 / 2.7) + 20$  ...ok...

- Keep that up! Let's go for a factor of  $n = 5$ , so drop total from 100 to 20:

i.e.  $20 ?= (80 / 5) + 20$  ...uh... can't do that... 😞

# Pitfalls: Idle Power

---

- Simply put:  
CPUs will still draw *disproportionate* power when idling.

**Assignment Project Exam Help**

- Example, even when operating at 10% load,  
~~<https://cstutorcs.com>~~ the i7 will draw 47% of the power

**WeChat: cstutorcs**

- Becomes a problem when dealing with large scale implementations, like data centers (Google, Facebook, Amazon, etc...)
- Design challenge: design processors to draw power more proportional to load (requires Physics-level approach, tho...)

# Pitfall: MIPS as a Performance Metric

---

- Note: We're NOT talking about **MIPS the processor type!!!!**

Assignment Project Exam Help

- **MIPS** (millions of instructions per second) is a popular performance metric, <https://tutorcs.com>

WeChat: cstutorcs

- Doesn't account for
  - Differences in ISAs between computers  
(some ISAs may be more efficient than others)
  - Differences in complexity between instructions (weighted CPIs)

---

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

# Instruction Set Architecture (ISA)

---

- The “contract” between software and hardware  
(hence, it’s an abstract model of a computer!)  
**Assignment Project Exam Help**
- Typically described with:
  - programmer-visible states (i.e. registers + memory)
  - the semantics/syntax of the instructions
  - Examples abound in your MIPS Reference Card!

# Instruction Set Architecture (ISA)

---

***Many implementations possible for a given ISA***

- Most microprocessor families have their own ISA  
**Assignment Project Exam Help**
- Some can be shared across families (b/c they're popular)
  - Example: AMD and Intel processors both run the x86-64 ISA (orig. Intel).
- Some of the same ISAs can be *customized*
  - Many cellphones use the ARM ISA with specific implementations from many different companies including
    - Apple, Qualcomm, Samsung, Huawei, etc.
- We'll be using the MIPS ISA in this class.

# Classification of ISAs

Intel, AMD (x86)	CISC
ARM, MIPS	RISC
GPUs (AMD, Nvidia)	RISC
Intel/HP (IA-86)	EPIC

- By architectural complexity\*

- **CISC** (complex instruction set computer) and  
**RISC** (reduced instruction set computer)

\* *Most popular distinction in commercial CPUs*

<https://tutorcs.com>

- By instruction-level parallelism

- **VLIW** (very long instruction word) and  
**EPIC** (explicitly parallel instruction computing)

- By extreme simplification of instructions

- **MISC** (minimal instruction set computer) and  
**OISC** (one instruction set computer)

***CISC vs RISC:***

- *Higher instruction complexity (and CPI)*
- *More transistors*
- *Higher power*
- *Commercial computers vs. embedded computers*

***EPIC/VLIW:***

- *Less commercial than CISC/RISC*
- *Server/supercomputer use mostly*

***MISC/OISC:***

- *Little to no parallelism*
- *Mostly in research*

# The MIPS ISA

---

- Developed at Stanford then commercialized by MIPS Technologies, created/led by John Hennessy
  - Stanford CS prof, President (2000-16), author of our textbook...
  - Started multiple important SV companies,  
**Assignment Project Exam Help**  
current Chair of Alphabet, Inc.  
<https://tutorcs.com>
- Hennessy and Patterson ~~won the~~ 2017 Turing Award for their work in developing RISC architecture
- MIPS still has a large share of embedded core market
  - Consumer electronics, storage peripherals, cameras, printers, ...

# Code on MIPS

**Original**

```
x = 5;  
y = 7;  
z = x + y;
```

**MIPS**

```
li $t0, 5  
li $t1, 7  
add $t3, $t0, $t1
```

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

# Available Registers in MIPS

## 32 registers in all

- Refer to your MIPS Reference Card
- Bring it to class from now on...
- Copy on main webpage

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

User for Q

NAME	NUMBER	USE
\$zero	0	The Constant Value 0
\$at	1	Assembler Temporary
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation
\$a0-\$a3	4-7	Arguments
\$t0-\$t7	8-15	Temporaries
\$s0-\$s7	16-23	Saved Temporaries
\$t8-\$t9	24-25	Temporaries
\$k0-\$k1	26-27	Reserved for OS Kernel
\$gp	28	Global Pointer
\$sp	29	Stack Pointer
\$fp	30	Frame Pointer
\$ra	31	Return Address

# MIPS Instruction Formats

---

- Each instruction is represented with **32 bits**

Assignment Project Exam Help

- There are **three** different *instruction formats*: **R, I, J**
  - These allow for instructions to take on different roles
  - R-Format is used when it's all about registers
  - I-Format is used when you involve (**immediate**) numbers
  - J-Format is used when you do code “jumping”  
(i.e. branching)

## Instruction Register

?

## Registers

\$t0: ?

\$t1: ?

\$t2: ?

Since all instructions are 32 bits then they each occupy 4 Bytes of memory.

Remember: Memory is addressed in Bytes.  
<https://tutorcs.com>

WeChat: cstutorcs

## Memory

?

## Program Counter

?

## Arithmetic Logic Unit

?

## Instruction Register

?

## Registers

\$t0: ?

\$t1: ?

\$t2: ?

## Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

```
0: li $t0, 5
4: li $t1, 7
8: add $t3, $t0, $t1
```

## Program Counter

0

## Arithmetic Logic Unit

?

## Instruction Register

```
li $t0, 5
```

## Registers

\$t0: ?

\$t1: ?

\$t2: ?

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

```
0: li $t0, 5
```

```
4: li $t1, 7
```

```
8: add $t3, $t0, $t1
```

## Program Counter

0

## Arithmetic Logic Unit

?

## Instruction Register

```
li $t0, 5
```

## Registers

\$t0: 5

\$t1: ?

\$t2: ?

Assignment Project Exam Help

<https://tutorcs.com>

## Memory

0: li \$t0, 5

4: li \$t1, 7

8: add \$t3, \$t0, \$t1

## Program Counter

0

## Arithmetic Logic Unit

?

## Instruction Register

li \$t0, 5

## Registers

\$t0: 5

\$t1: ?

\$t2: ?

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

0: li \$t0, 5  
4: li \$t1, 7  
8: add \$t3, \$t0, \$t1

## Program Counter

4

## Arithmetic Logic Unit

0 + 4 = 4

## Instruction Register

```
li $t1, 7
```

## Registers

\$t0: 5

\$t1: ?

\$t2: ?

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

```
0: li $t0, 5
```

```
4: li $t1, 7
```

```
8: add $t3, $t0, $t1
```

## Program Counter

4

## Arithmetic Logic Unit

?

## Instruction Register

li \$t1, 7

## Registers

\$t0: 5  
\$t1: 7  
\$t2: ?

## Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

0: li \$t0, 5  
4: li \$t1, 7  
8: add \$t3, \$t0, \$t1

## Program Counter

4

## Arithmetic Logic Unit

?

## Instruction Register

```
li $t1, 7
```

## Registers

\$t0: 5  
\$t1: 7  
\$t2: ?

## Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

```
0: li $t0, 5
4: li $t1, 7
8: add $t3, $t0, $t1
```

## Program Counter

8

## Arithmetic Logic Unit

4 + 4 = 8

## Instruction Register

add \$t3, \$t0, \$t1

## Registers

\$t0: 5

\$t1: 7

\$t2: ?

## Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

0: li \$t0, 5

4: li \$t1, 7

8: add \$t3, \$t0, \$t1

## Program Counter

8

## Arithmetic Logic Unit

?

## Instruction Register

add \$t3, \$t0, \$t1

## Registers

\$t0: 5  
\$t1: 7  
\$t2: ?

## Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

0: li \$t0, 5  
4: li \$t1, 7  
8: add \$t3, \$t0, \$t1

## Program Counter

8

## Arithmetic Logic Unit

5 + 7 = 12

## Instruction Register

add \$t3, \$t0, \$t1

## Registers

\$t0: 5

\$t1: 7

\$t2: 12

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## Memory

0: li \$t0, 5

4: li \$t1, 7

8: add \$t3, \$t0, \$t1

## Program Counter

8

## Arithmetic Logic Unit

5 + 7 = 12

# Talking to the OS

---

- We are going to be running on MIPS *emulator* called **SPIM**
  - Optionally, through a program called **QtSPIM** (GUI based)
  - *What is an emulator?*
- MIPS features a ~~syscall~~ <https://tutorcs.com> instruction, which triggers a *software interrupt*, or *exception*  
**WeChat: cstutorcs**
- Outside of an emulator (i.e. in the real world), these instructions **pause the program** and tell the OS to go do something with I/O
- Inside the emulator, it tells the emulator to go *emulate* something with I/O

# syscall (for spim use)

---

- The OS/emulator has access to the CPU registers
- So we have the OS/emulator's attention, but how does it know what we want?  
<https://tutorcs.com>
- We put special values (codes) in the registers to indicate what we want
  - These are codes that can't be used for anything else, so they're understood to be just for syscall
  - So... is there a “code book”????

Yes! All CPUs come with manuals.  
For us, we have the **MIPS Ref. Card**

# MIPS System Services

System call code:  
Argument:

placed in \$v0  
placed in \$a0

Service	System Call Code	Arguments	Result
print_int	1	\$a0 = integer	stdout
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
read_int	5		integer (in \$v0)
read_float	6		float (in \$f0)
read_double	7		double (in \$f0)
read_string	8	\$a0 = buffer, \$a1 = length	stdin
sbrk	9	\$a0 = amount	
exit	10		
print_character	11	\$a0 = character	
read_character	12		character (in \$v0)
open	13	\$a0 = filename,	File I/O
		\$a1 = flags, \$a2 = mode	
read	14	\$a0 = file descriptor,	
		\$a1 = buffer, \$a2 = count	
write	15	\$a0 = file descriptor,	bytes written (in \$v0)
		\$a1 = buffer, \$a2 = count	
close	16	\$a0 = file descriptor	0 (in \$v0)
exit2	17	\$a0 = value	

Assignment Project Exam Help  
Bring out your MIPS Reference Cards!

<https://tutorcs.com>

WeChat: cstutorcs

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs]] +SignExtImm}(7:0)	(2) 14 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs]] +SignExtImm}(15:0)	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt] = M[R[rs]]+SignExtImm	(2) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt] = M[R[rs]]+SignExtImm	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 0d <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>
Set Less Than Imm.	siti	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2) a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6) b <sub>hex</sub>
Set Less Than Unsigned	situ	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll	R R[rd] = R[rt] << sham	0 / 00 <sub>hex</sub>
Shift Right Logical	srl	R R[rd] = R[rt] >> sham	0 / 02 <sub>hex</sub>
Store Byte	sb	I M[R[rs]]+SignExtImm](7:0) = R[rt](7:0)	(2) 28 <sub>hex</sub>
Store Conditional	sc	I M[R[rs]]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7) 38 <sub>hex</sub>
Store Halfword	sh	I M[R[rs]]+SignExtImm](15:0) = R[rt](15:0)	(2) 29 <sub>hex</sub>
Store Word	sw	I M[R[rs]]+SignExtImm] = R[rt]	(2) 2b <sub>hex</sub>
Subtract	sub	R R[rd] = R[rs] - R[rt]	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu	R R[rd] = R[rs] - R[rt]	0 / 23 <sub>hex</sub>

## NOTE THE FOLLOWING:

### 1. Instruction Format Types: R vs I vs J

### 2. OPCODE/FUNCT (Hex)

<https://tutorcs.com>

WeChat: cstutorcs  
Where the actual bits go

#### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	
	31 26 25	21 20	16 15	11 10	6 5	0	
I	opcode	rs	rt			immediate	
	31 26 25	21 20	16 15				0
J	opcode				address		
	31 26 25						0

## PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

## REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	No

## Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## NOTE THE FOLLOWING:

### 1. Pseudo-Instructions

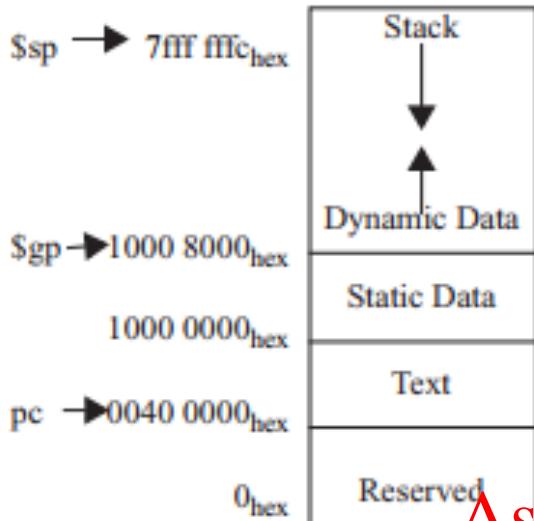
- There are more of these, but in this class, you are ONLY allowed to use these + la

### 2. Registers and their numbers

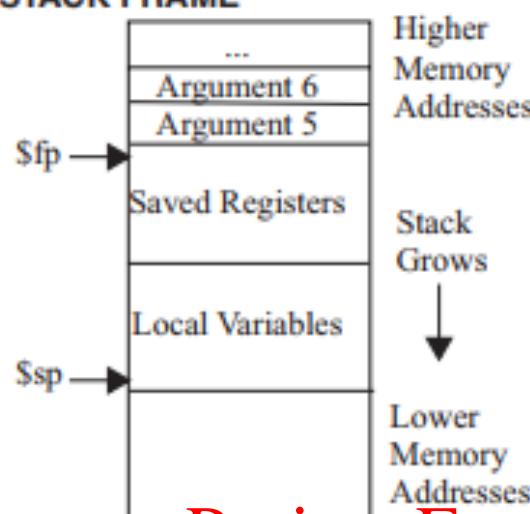
### 3. Registers and their uses

### 4. Registers and their calling convention

## MEMORY ALLOCATION



## STACK FRAME

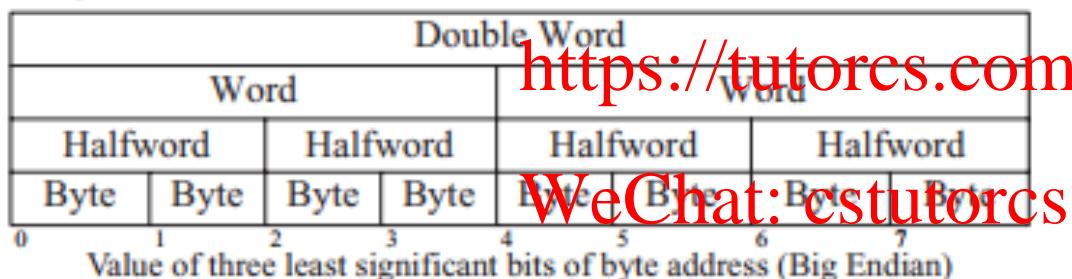


## NOTE THE FOLLOWING:

1. This is only part of the 2<sup>nd</sup> page that you need to know

## Assignment Project Exam Help

### DATA ALIGNMENT



<https://tutorcs.com>  
WeChat: cstutorcs

### SIZE PREFIXES (10<sup>x</sup> for Disk, Communication; 2<sup>x</sup> for Memory)

PRE-SIZE	FIX	PRE-SIZE	FIX	PRE-SIZE	FIX	PRE-SIZE	FIX
10 <sup>3</sup> , 2 <sup>10</sup>	Kilo-	10 <sup>15</sup> , 2 <sup>50</sup>	Peta-	10 <sup>-3</sup>	milli-	10 <sup>-15</sup>	femto-
10 <sup>6</sup> , 2 <sup>20</sup>	Mega-	10 <sup>18</sup> , 2 <sup>60</sup>	Exa-	10 <sup>-6</sup>	micro-	10 <sup>-18</sup>	atto-
10 <sup>9</sup> , 2 <sup>30</sup>	Giga-	10 <sup>21</sup> , 2 <sup>70</sup>	Zetta-	10 <sup>-9</sup>	nano-	10 <sup>-21</sup>	zepto-
10 <sup>12</sup> , 2 <sup>40</sup>	Tera-	10 <sup>24</sup> , 2 <sup>80</sup>	Yotta-	10 <sup>-12</sup>	pico-	10 <sup>-24</sup>	yocto-

The symbol for each prefix is just its first letter, except  $\mu$  is used for micro.

# Bring Out Your MIPS Reference Cards!

---

**Look for the following instructions:**

- nor                      Assignment Project Exam Help
- addi                     <https://tutorcs.com>
- beq                      WeChat: cstutorcs
- move

*Tell me everything you can about them, based on what you see on the Ref Card!*

## Example 1

Syntax for “add”

**add rd, rs, rt**

destination, source1, source2

$$f = (g + h) - (i + j)$$

i.e.  $\$s0 = (\$s1 + \$s2) - (\$s3 + \$s4)$

Assignment Project Exam Help

<https://tutorcs.com>

add \$t0, \$s1, \$s2

add \$t1, \$s3, \$s4

sub \$s0, \$t0, \$t1

## Example 2

---

$$f = g * h - i$$

i.e.  $\$s0 = (\$s1 * \$s2) - \$s3$

Assignment Project Exam Help

mult \$s1, \$s2  
<https://tutorcs.com>

mflo \$t0  
WeChat: cstutorcs

# mflo directs where the answer of the  
mult should go

sub \$s0, \$t0, \$s3

# Recap: The mult instruction

---

- To multiply 2 integers together:

```
li $t0, 5          # t0 = 5
li $t1, 6          # t1 = 6
mult $t1, $t0      # multiply t0 * t1
mflo $t2          # t2 = t0 * t1
```

- **mult** cannot be used with an ‘immediate’ value
- Then we multiply our multiplier (\$t0) with our multiplicand (\$t1)
- And we put the result in the destination reg (\$t2) using the **mflo** instruction

# Memory Operations

---

- Main memory used for composite data
  - e.g.: Arrays, structures, dynamic data
  - In MIPS, use the **.data** declaration to initialize memory values  
(must be above **.text** declaration)

<https://tutorcs.com>

- Example:

WeChat: cstutorcs

**.data**

**var1: .word 42**

**.text**

**la \$t0, var1 # t0 = &var1**

**lw \$t1, 0(\$t0) # t1 = \*(&var1) = 42**

```
.data  
name: .asciiz "Lisa speaks "  
rtn: .asciiz " languages!\n"  
age: .word 7
```

```
.text
```

```
main:  
    li $v0, 4  
    la $a0, name  
    syscall  
  
    la $t2, age  
    lw $a0, 0($t2)  
    li $v0, 1  
    syscall  
  
    li $v0, 4  
    la $a0, rtn  
    syscall  
  
    li $v0, 10  
    syscall
```

Example  
*What does this do?*



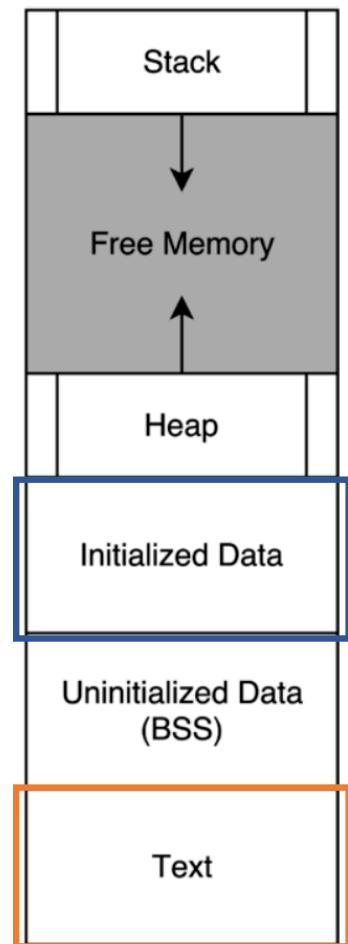
Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

*What goes in here? →*

*What goes in here? →*



# .data Declaration Types w/ Examples

---

```
var1: .byte 9           # declare a single byte with value 9
var2: .half 63          # declare a 16-bit half-word w/ val. 63
var3: .word 9433         # declare a 32-bit word w/ val. 9433
num1: .float 3.14        # declare 32-bit floating point number
num2: .double 6.28       # declare 64-bit floating pointer number
str1: .ascii "Text"      # declare a string of chars
str3: .asciiz "Text"     # declare a null-terminated string
str2: .space 5           # reserve 5 bytes of space (useful for arrays)
```

*These are now reserved in memory and we can call them up by loading their memory address into the appropriate registers.*

# YOUR TO-DOs for the Week

---

- Do your reading for next class (see syllabus)
- Work on Assignment #1 for lab (*lab01*)
  - Meet up in the lab this Friday <https://tutorcs.com>
  - Do the lab assignment
  - You have to submit it as a PDF using *Gradescope*
  - Due on **Wednesday, 1/15, by 11:59:59 PM**

Assignment Project Exam Help

