Digital Design & Computer Arch.

Lab 7 Supplement: Writing Assembly Code

(Presentation by Aaron Zeller)
Frank K. Gürkaynak
Seyyedmohammad Sadrosadati
ETH Zurich

Spring 2024

[30. April 2024]

Writing Assembly Code

- In Lab 7, you will write MIPS Assembly code
- You will use the MARS simulator to run your code
- References
  - H&H Chapter 6
  - Lectures 12 to 15
    - https://safari.ethz.ch/ddca/doku.php?id=schedule
  - MIPS Cheat Sheet
    - https://safari.ethz.ch/ddca/spring2024/lib/exe/fetch.php?media=mips_reference_data.pdf

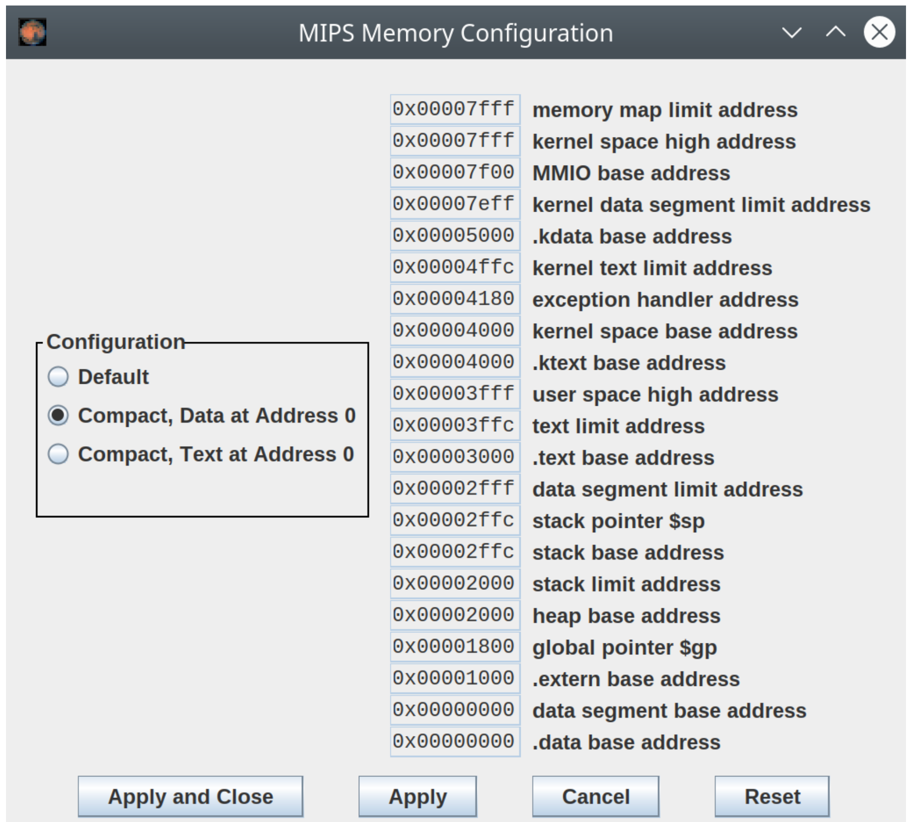
Writing Assembly Code: Mars Simulator

- You will use the MARS simulator to run your code
  - Provided in the course website OR
  - Can be downloaded from this website.

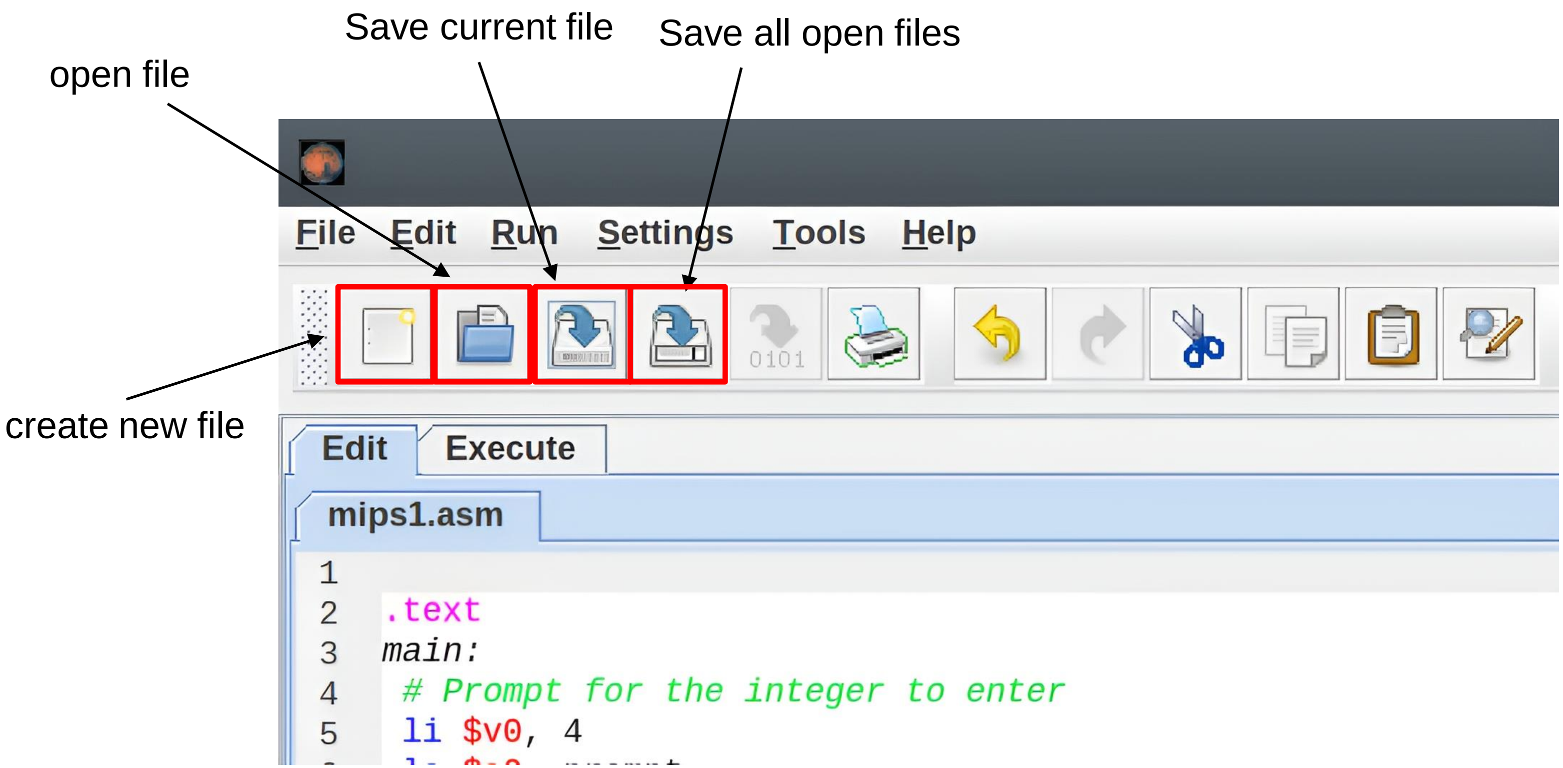
- Once downloaded navigate to the Downloads folder in the Terminal / Console / Cmd and execute it.
  - `java -jar Mars4_5.jar` (Version may vary)

Writing Assembly Code: Mars Simulator

- Under Settings -> Memory Configuration make sure to select the "Compact" memory configuration.



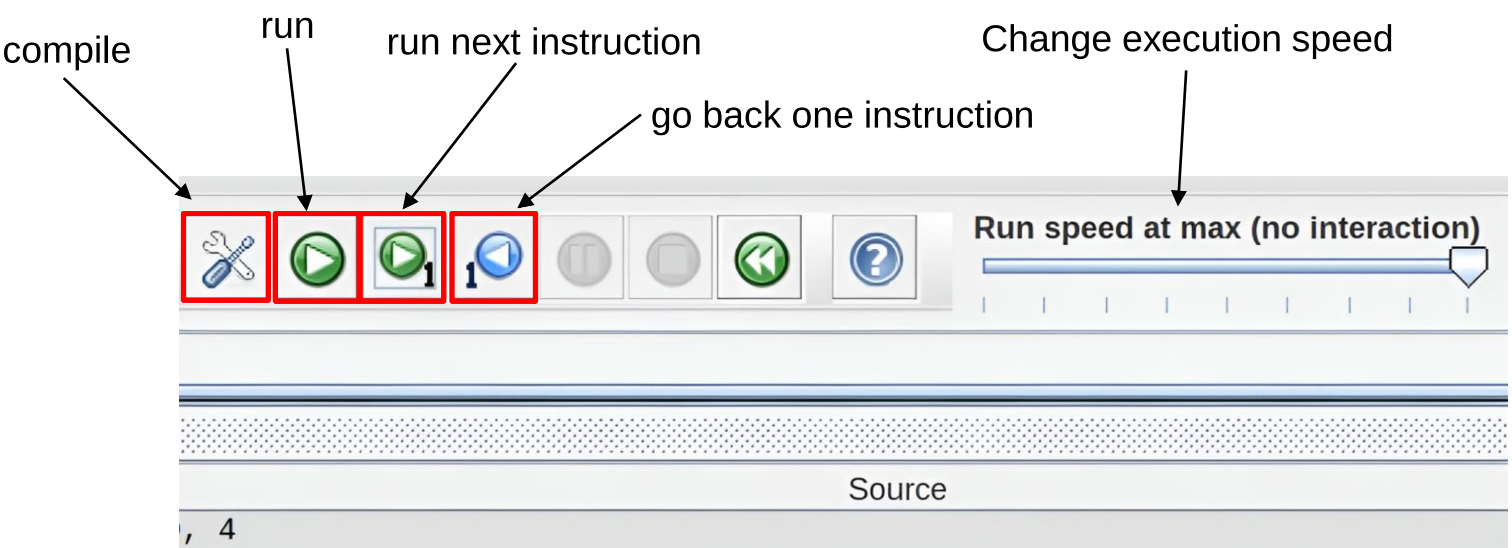
Writing Assembly Code: Mars Simulator



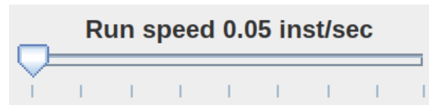
Writing Assembly Code: Mars Simulator

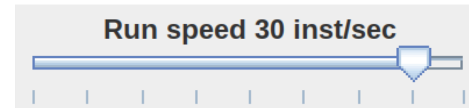
- Before you can compile your MIPS assembly code you must first save the file.
  - The option to compile will not be available until you save your code.
  - The option to run your code will not be available until you compile your code.

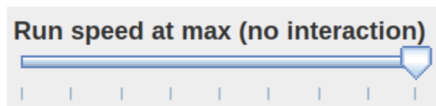
Writing Assembly Code: Mars Simulator



Writing Assembly Code: Mars Simulator

- You can change the speed of execution using the slider.
- The speed ranges from 

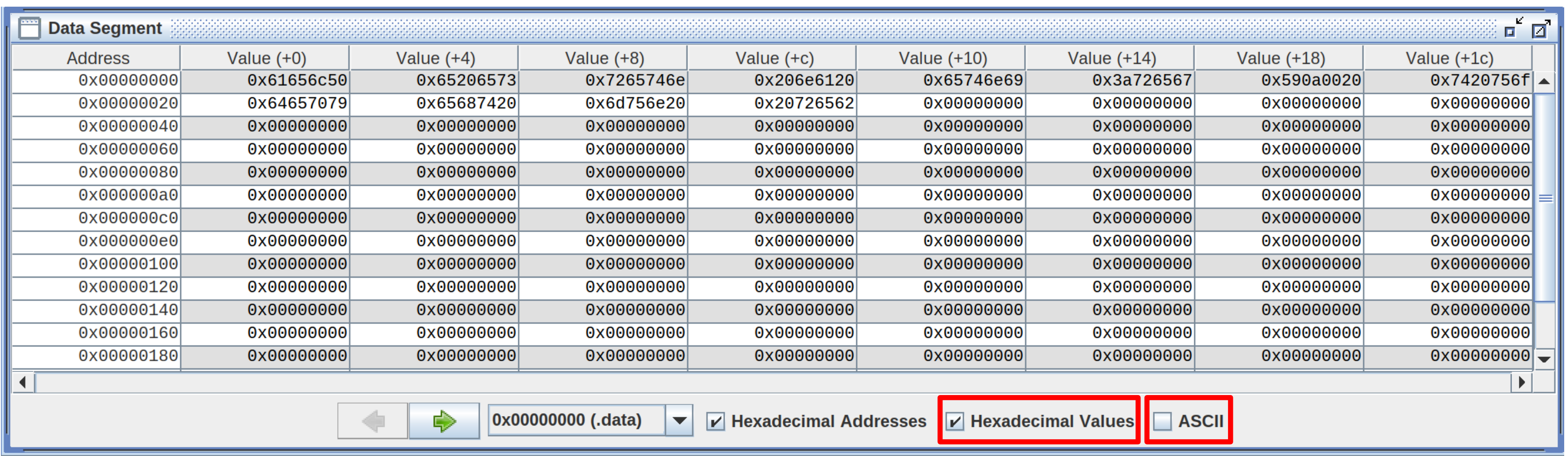
to 

- Default speed is 

(all instructions are executed immediately)

Writing Assembly Code: Mars Simulator

- In the memory view you want to see decimal or ASCII values.
  - Select the option to do so depending on the task, i.e. whether you want to see number or characters.



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x61656c50	0x65206573	0x7265746e	0x206e6120	0x65746e69	0x3a726567	0x590a0020	0x7420756f
0x00000020	0x64657079	0x65687420	0x6d756e20	0x20726562	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

unselect this for decimal

Select this for ASCII

An Example of MIPS Assembly Code

- Add all the even numbers from 0 to 10
  - $0 + 2 + 4 + 6 + 8 + 10 = 30$

High-level code

```
int sum = 0;

for(int i = 0;i <= 10;i += 2)
{

  sum += i;

}
```

MIPS assembly

```
# i=$s0; sum=$s1

        addi $s0, $0, 0
        addi $s1, $0, 0
        addi $t0, $0, 12
loop:   beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 2
        j    loop
done:
```

An Example of MIPS Assembly Code

- Add all the even numbers from 0 to 10
  - $0 + 2 + 4 + 6 + 8 + 10 = 30$

High-level code

```
int sum = 0;

for(int i = 0;i <= 10;i += 2)
{

  sum += i;

}
```

MIPS assembly

```
# i=$s0; sum=$s1

        addi $s0, $0, 0
        addi $s1, $0, 0
        addi $t0, $0, 12
loop:   beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 2
        j    loop
done:
```

An Example of MIPS Assembly Code

- Add all the even numbers from 0 to 10
  - $0 + 2 + 4 + 6 + 8 + 10 = 30$

High-level code

```
int sum = 0;

for(int i = 0;i <= 10;i += 2)
{

  sum += i;

}
```

MIPS assembly

```
# i=$s0; sum=$s1

        addi $s0, $0, 0
        addi $s1, $0, 0
        addi $t0, $0, 12
loop:   beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 2
        j    loop
done:
```

An Example of MIPS Assembly Code

- Add all the even numbers from 0 to 10
  - $0 + 2 + 4 + 6 + 8 + 10 = 30$

High-level code

```
int sum = 0;

for(int i = 0;i <= 10;i += 2)
{

  sum += i;

}
```

MIPS assembly

```
# i=$s0; sum=$s1

        addi $s0, $0, 0
        addi $s1, $0, 0
        addi $t0, $0, 12
loop:   beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 2
        j    loop
done:
```

An Example of MIPS Assembly Code

- Add all the even numbers from 0 to 10
  - $0 + 2 + 4 + 6 + 8 + 10 = 30$

High-level code

```
int sum = 0;

for(int i = 0;i <= 10;i += 2)
{

  sum += i;

}
```

MIPS assembly

```
# i=$s0; sum=$s1

        addi $s0, $0, 0
        addi $s1, $0, 0
        addi $t0, $0, 12
loop:   beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 2
        j    loop
done:
```

An Example of MIPS Assembly Code

- Add all the even numbers from 0 to 10
  - $0 + 2 + 4 + 6 + 8 + 10 = 30$

High-level code

```
int sum = 0;

for(int i = 0;i <= 10;i += 2)
{

  sum += i;

}
```

MIPS assembly

```
# i=$s0; sum=$s1

        addi $s0, $0, 0
        addi $s1, $0, 0
        addi $t0, $0, 12
loop:   beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 2
        j    loop
done:
```

Recall: Arrays (Code Example)

- We first load the base address of the array into a register (e.g., $s0) using lui and ori

High-level code

```
int array[5];

array[0] = array[0] * 2;

array[1] = array[1] * 2;
```

MIPS assembly

```
# array base address = $s0
# Initialize $s0 to 0x12348000
lui   $s0, 0x1234
ori   $s0, $s0, 0x8000

lw    $t1, 0($s0)
sll   $t1, $t1, 1
sw    $t1, 0($s0)
lw    $t1, 4($s0)
sll   $t1, $t1, 1
sw    $t1, 4($s0)
```

Recall: MIPS R-Type Instructions

Description:	Add two registers and store the result in a register $d.	
Operation:	$d = $s + $t; advance_pc (4);	
Syntax:	add $d, $s, $t	ADD

Description:	Subtract $t from $s and store the result in $d.	
Operation:	$d = $s - $t; advance_pc (4);	
Syntax:	sub $d, $s, $t	SUB

Description:	If $s is less than $t, $d is set to one. $d gets zero otherwise.	
Operation:	if $s < $t: $d = 1; advance_pc (4); else: $d = 0; advance_pc (4);	
Syntax:	slt $d, $s, $t	SLT

Description:	Exclusive or of $s and $t and store the result in $d.	
Operation:	$d = $s ^ $t; advance_pc (4);	
Syntax:	xor $d, $s, $t	XOR

Description:	Bitwise and of $s and $t and store the result in the register $d.	
Operation:	$d = $s & $t; advance_pc (4);	
Syntax:	and $d, $s, $t	AND

Description:	Bitwise logic or of $s and $t and store the result in $d.	
Operation:	$d = $s | $t; advance_pc (4);	
Syntax:	or $d, $s, $t	OR

Recall: MIPS I-Type Instructions

Description:	Add sign-extended immediate to register $s and store the result in $t.	
Semantics:	$t = $s + imm; PC=PC+4;	
Syntax:	addi $t, $s, imm	ADDI

Description:	Branch if the contents of $s and $t are equal.	
Semantics:	if $s == $t: advance_pc (offset << 2)); else: PC=PC+4;	
Syntax:	beq $s, $t, offset	BEQ

Recall: MIPS J-Type Instructions

Description:	Jump to the address.	
Semantics:	PC = nPC; nPC = (PC & 0xf0000000) | (target << 2);	
Syntax:	j target	J

Writing Assembly Code: Extra Resources

- The lecture contains all information needed.

- For students that would like to see more about MIPS the following resource is a good starting point.
  - Additional Resource (not needed for exam / lecture)

Lab 7: Exercise 1

- Write MIPS assembly code to compute the sum $A + (A + 1) + \cdots (B - 1) + B$, given two inputs A and B.

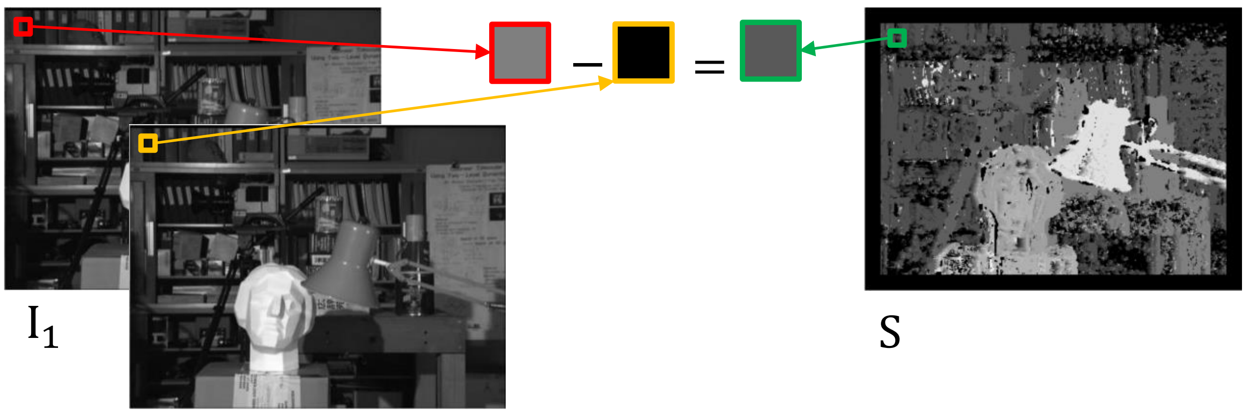
- Example
  - $A = 5, B = 10$ ➔ $S = 5 + 6 + 7 + 8 + 9 + 10 = 45$

- For this exercise, you can use a subset of MIPS instructions: ADD, SUB, SLT, XOR, AND, OR and NOR, which are the instructions supported by the ALU you designed in the previous labs

- Additionally, you are allowed to use J, ADDI and BEQ

Lab 7: Exercise 2

- Write MIPS assembly code to compute the Sum of Absolute Differences of two images



$$S(x, y) = |I_1(x, y) - I_2(x, y)|$$

- Hints
  - Recall the function calls and the use of the stack in Lecture 10
  - Read how to implement recursive function calls in H&H 6.4

Lab 7: Assembly Basics

- Respect calling conventions – they are your friend.

Number	**Name**	**Purpose**
$0	$0	Always 0
$1	$at	The *Assembler Temporary* used by the assembler in expanding pseudo-ops.
$2-$3	$v0-$v1	These registers contain the *Returned Value* of a subroutine; if the value is 1 word only $v0 is significant.
$4-$7	$a0-$a3	The *Argument* registers, these registers contain the first 4 argument values for a subroutine call.
$8-$15,$24,$25	$t0-$t9	The *Temporary Registers*.
$16-$23	$s0-$s7	The *Saved Registers*.
$26-$27	$k0-$k1	The *Kernel Reserved registers*. DO NOT USE.
$28	$gp	The *Globals Pointer* used for addressing static global variables. For now, ignore this.
$29	$sp	The *Stack Pointer*.
$30	$fp (or $s8)	The *Frame Pointer*, if needed (this was discussed briefly in lecture). Programs that do not use an explicit frame pointer (e.g., everything assigned in ECE314) can use register $30 as another saved register. Not recommended however.
$31	$ra	The *Return Address* in a subroutine call.

Image taken from: https://courses.cs.washington.edu/courses/cse410/09sp/examples/MIPSCallingConventionsSummary.pdf

Lab 7: Assembly Basics

- Using the stack may seem a bit counterintuitive at first.

- Any function call shares the same registers hence you must:

  - **Store the state** of the current function call before any other function call.

  - **Restore the state** from stack when returning from a function call.

Last Words

- In this lab, you will do what a compiler does: transforming high level code to MIPS assembly
- Exercise 1: Write simple code and get familiar with the MARS simulator
- Exercise 2: Sum of Absolute Differences of two images
- Find Exercise 3 in the lab report

Report Deadline

[24. Mai 2024 23:59]

Digital Design & Computer Arch.
Lab 7 Supplement: Writing Assembly Code

Frank K. Gürkaynak
Seyyedmohammad Sadrosadati
ETH Zurich

Spring 2024

[30. April 2024]