Computer Systems and Networks

ECPE 170 – University of the Pacific

MIPS Assembly (Functions)

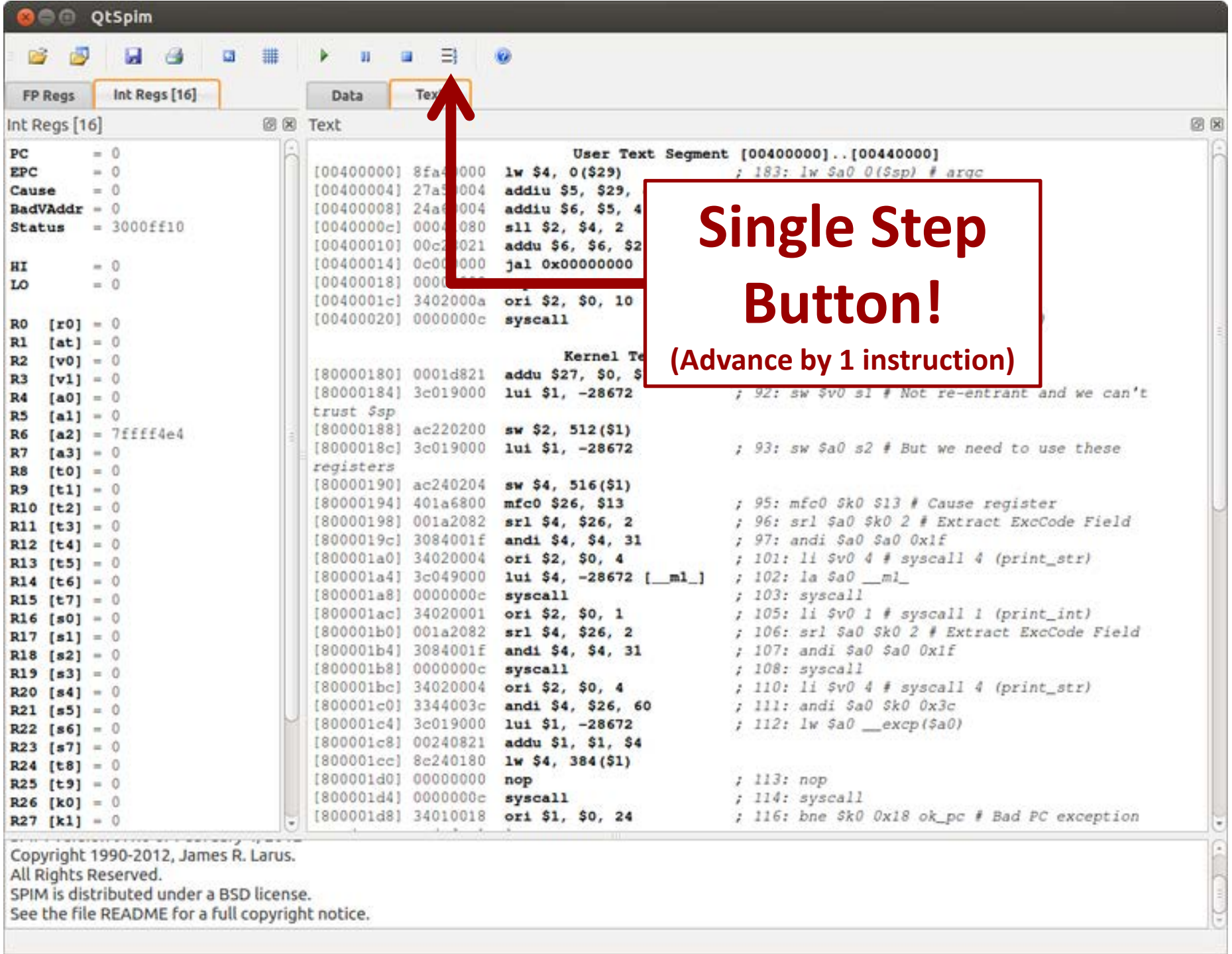
Lab Schedule

Activities

- **Today**
  - Discuss: MIPS Functions
  - Lab 10
- **Next week**
  - Lab 10

Assignments Due

- **Tues Nov 12th**
  - **Lab 9 due by 11:59pm**
- **Tues Nov 19th**
  - **Lab 10 due by 11:59pm**


QtSpim
FP Regs
Int Regs [16]
Data
Text
Int Regs [16]
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 0
R5 [a1] = 0
R6 [a2] = 7ffff4e4
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
Text
User Text Segment [00400000]..[00440000]
[00400000] 8fa4 0000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a5 0004 addiu $5, $29,
[00400008] 24a6 0004 addiu $6, $5, 4
[0040000c] 0004 4080 sll $2, $4, 2
[00400010] 00c2 3021 addu $6, $6, $2
[00400014] 0c00 0000 jal 0x00000000
[00400018] 0000
[0040001c] 3402000a ori $2, $0, 10
[00400020] 0000000c syscall
Kernel Te
[80000180] 0001d821 addu $27, $0, $
[80000184] 3c019000 lui $1, -28672 ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672 ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204 sw $4, 516($1)
[80000194] 401a6800 mfc0 $26, $13 ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082 srl $4, $26, 2 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f andi $4, $4, 31 ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004 ori $2, $0, 4 ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000 lui $4, -28672 [__m1_] ; 102: la $a0 __m1_
[800001a8] 0000000c syscall ; 103: syscall
[800001ac] 34020001 ori $2, $0, 1 ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082 srl $4, $26, 2 ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f andi $4, $4, 31 ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c syscall ; 108: syscall
[800001bc] 34020004 ori $2, $0, 4 ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c andi $4, $26, 60 ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000 lui $1, -28672 ; 112: lw $a0 __excp($a0)
[800001c8] 00240821 addu $1, $1, $4
[800001cc] 8c240180 lw $4, 384($1)
[800001d0] 00000000 nop ; 113: nop
[800001d4] 0000000c syscall ; 114: syscall
[800001d8] 34010018 ori $1, $0, 24 ; 116: bne $k0 0x18 ok_pc # Bad PC exception
Single Step Button!
(Advance by 1 instruction)
Copyright 1990-2012, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.

MIPS Functions

Function Requirements?

- **What happens when we call a function?**
  1. Place function arguments in standard location where function can find them
  2. Save current program location to return to later (the "Program Counter" register)
  3. Jump to the function location
  4. Function runs using provided arguments
  5. Function produces output (return value) and saves it in standard location
  6. Jump to original program location (return)
     1. *Technically, +1 instruction*

Function Requirements

- **Can a function change local variables of its calling function?**
- No! The function operates in its own “bubble”
- **What happens if the function changes $s0 which was also used by the calling function?**
- Problem! Your function has corrupted the calling function

Functions in Assembly

In assembly, **you** must do all the background work for functions that the compiler did automatically in a higher level language

Functions still allow for **code re-use** (good!), but they're more complicated than in C or C++

Registers

Name	Use
$zero	Constant value: ZERO
$s0-$s7	Local variables (Convention: These are saved by *the function* if they are needed)
$t0-$t9	Temporary results (Convention: These are saved by *the caller* if they are needed)
$a0-$a3	Arguments to pass to function (max of 4)
$v0-$v1	Return value to obtain from function (max of 2)
$ra	Return address of function
$sp	Stack pointer (current top of stack)

New!

More Jumps

- Jump and Link
  (side effect: `$ra` stores address of next instruction)

```
jal <destination>
```

Use this to *call* a function!

- Jump Register
  (destination address is stored in <reg1>

```
jr <reg1>
```

Use this to *return from* a function!

Task : Write Code

```
#include <stdio.h>

int function(int a);

int main()
{
  int x=5;
  int y;

  y = function(x);

  printf("y=%i\n", y);

  return 0;
}

int function(int a)
{
  return 3*a+5;
}
```

- Place arguments in `$a0-$a3`
- Place return values in `$v0-$v1`
- Return address saved automatically in `$ra`
- Ignore the stack for this example. (Thus, the function will destroy registers used by calling function)

```
# Simple routine to demo functions
# NOT using a stack in this example.
# Thus, the function does not preserve values
# of calling function!

# ------------------------------------------------------------------

        .text

        .globl  main
main:
        # Register assignments
        # $s0 = x
        # $s1 = y

        # Initialize registers
        lw      $s0, x          # Reg $s0 = x
        lw      $s1, y          # Reg $s1 = y

        # Call function
        move    $a0, $s0        # Argument 1: x ($s0)
        jal     fun             # Save current PC in $ra, and jump to fun
        move    $s1,$v0         # Return value saved in $v0. This is y ($s1)

        # Print msg1
        li      $v0, 4          # print_string syscall code = 4
        la      $a0, msg1
        syscall

        # Print result (y)
        li      $v0,1           # print_int syscall code = 1
        move    $a0, $s1        # Load integer to print in $a0
        syscall

        # Print newline
        li      $v0,4           # print_string syscall code = 4
        la      $a0, lf
        syscall

        # Exit
        li      $v0,10          # exit
        syscall
```

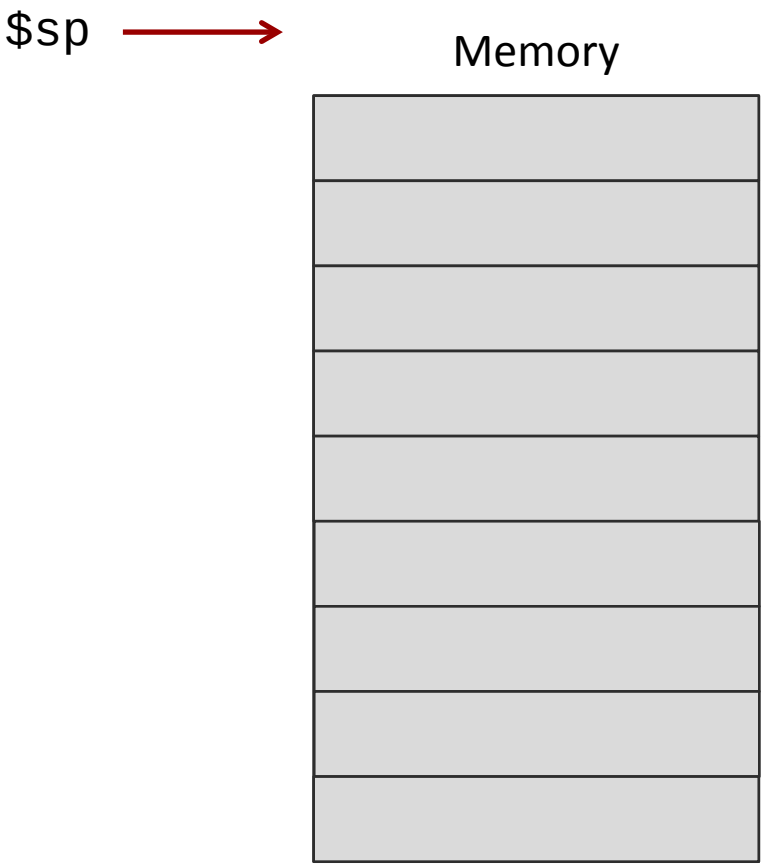
```
# ------------------------------------------------------------------

        # FUNCTION: int fun(int a)
        # Arguments are stored in $a0
        # Return value is stored in $v0
        # Return address is stored in $ra (put there by jal instruction)
        # Typical function operation is:

fun:    # Do the function math
        li $s0, 3
        mul $s1,$s0,$a0# s1 = 3*$a0  (i.e. 3*a)
        addi $s1,$s1,5          # 3*a+5

        # Save the return value in $v0
        move $v0,$s1

        # Return from function
        jr $ra                  # Jump to addr stored in $ra

# ------------------------------------------------------------------

        # Start .data segment (data!)
        .data
x:      .word 5
y:      .word 0
msg1:   .asciiz "y="
lf:      .asciiz"\n"
```

Preserving Registers

- **What if we don't want to destroy registers used by the calling function?**
- Need to save those registers somewhere while our function runs (like memory!)
- A **stack** is a good structure for this

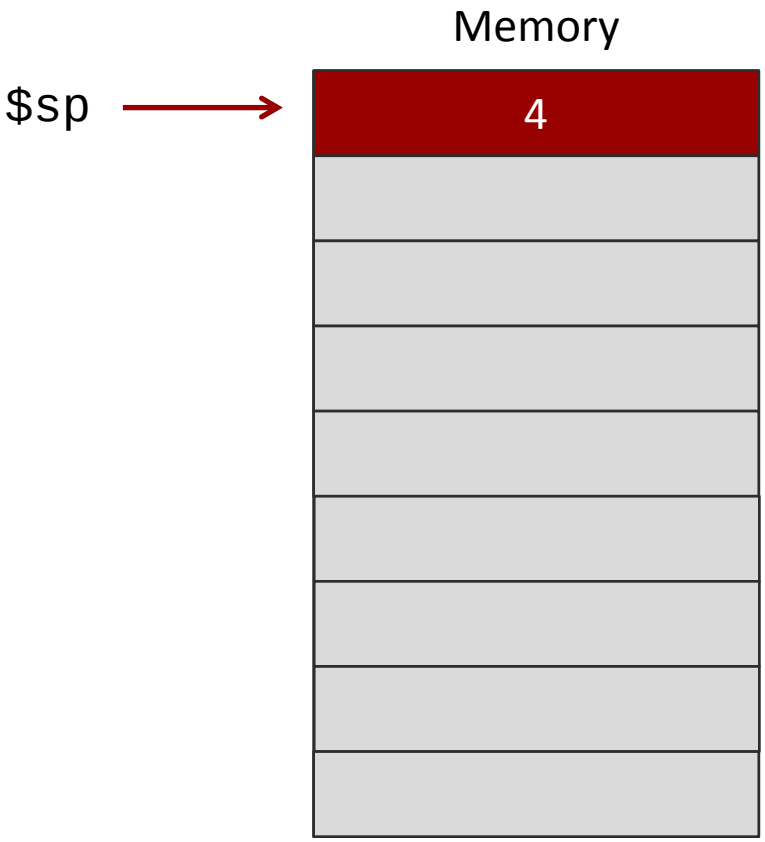
The Stack

- Stack is a data structure stored in memory
- $sp ("Stack Pointer") points to top of stack
  - But stack grows **down** in memory!
- Example
  - Push 4 to stack
  - Push 5 to stack
  - Pop (5 from stack)
  - Pop (4 from stack)



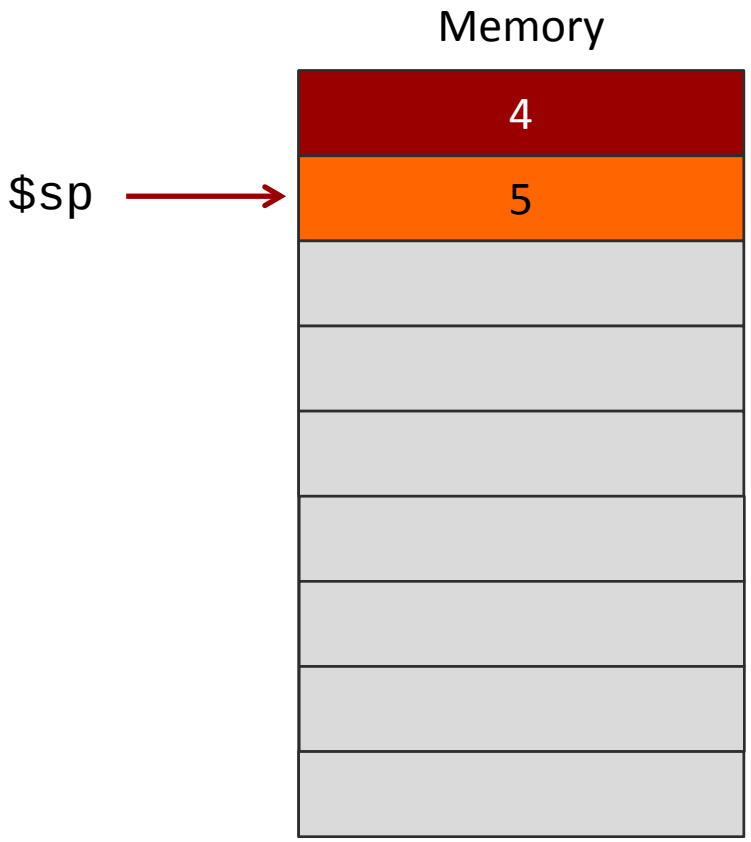
The Stack

- Stack is a data structure stored in memory
- $sp ("Stack Pointer") points to top of stack
  - But stack grows **down** in memory!
- Example
  - Push 4 to stack
  - Push 5 to stack
  - Pop (5 from stack)
  - Pop (4 from stack)



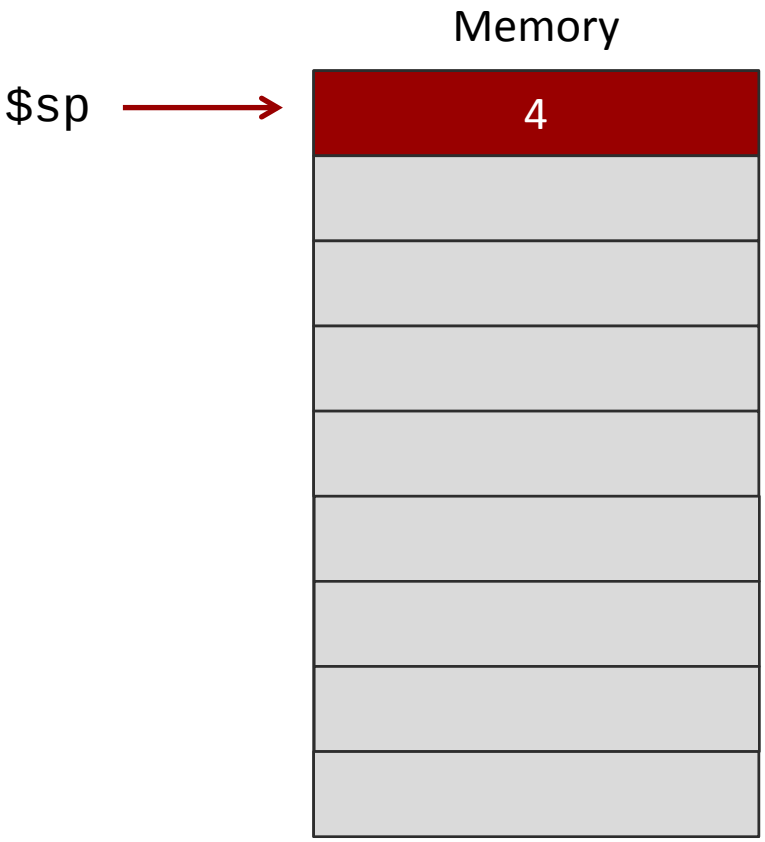
The Stack

- Stack is a data structure stored in memory
- `$sp` ("Stack Pointer") points to top of stack
  - But stack grows **down** in memory!
- Example
  - Push 4 to stack
  - Push 5 to stack
  - Pop (5 from stack)
  - Pop (4 from stack)



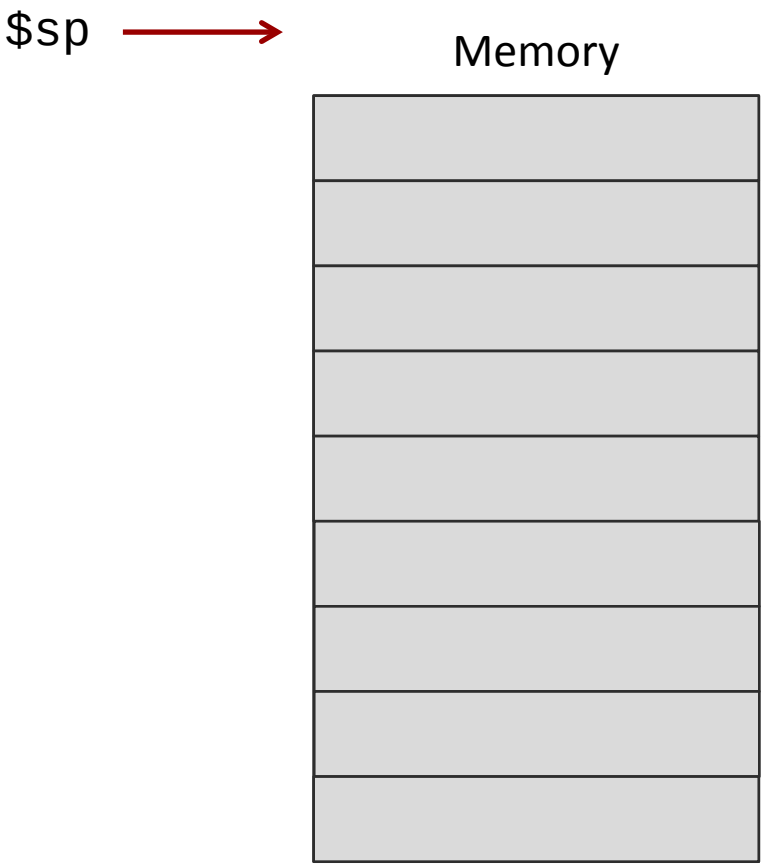
The Stack

- Stack is a data structure stored in memory
- $sp ("Stack Pointer") points to top of stack
  - But stack grows **down** in memory!
- Example
  - Push 4 to stack
  - Push 5 to stack
  - Pop (5 from stack)
  - Pop (4 from stack)



The Stack

- Stack is a data structure stored in memory
- $sp ("Stack Pointer") points to top of stack
  - But stack grows **down** in memory!
- Example
  - Add 4 to stack
  - Add 5 to stack
  - Pop
  - Pop



The Stack

- **How would we modify previous solution to use a stack?**

```
# Simple routine to demo functions
# NOT using a stack in this example.
# Thus, the function does not preserve values
# of calling function!

# ------------------------------------------------------------------

        .text

        .globl  main
main:
        # Register assignments
        # $s0 = x
        # $s1 = y

        # Initialize registers
        lw      $s0, x          # Reg $s0 = x
        lw      $s1, y          # Reg $s1 = y

        # Call function
        move    $a0, $s0        # Argument 1: x ($s0)
        jal     fun             # Save current PC in $ra, and jump to fun
        move    $s1,$v0         # Return value saved in $v0. This is y ($s1)

        # Print msg1
        li      $v0, 4          # print_string syscall code = 4
        la      $a0, msg1
        syscall

        # Print result (y)
        li      $v0,1           # print_int syscall code = 1
        move    $a0, $s1        # Load integer to print in $a0
        syscall

        # Print newline
        li      $v0,4           # print_string syscall code = 4
        la      $a0, lf
        syscall

        # Exit
        li      $v0,10          # exit
        syscall
```

```
# ------------------------------------------------------------------

        # FUNCTION: int fun(int a)
        # Arguments are stored in $a0
        # Return value is stored in $v0
        # Return address is stored in $ra (put there by jal instruction)
        # Typical function operation is:

fun:    # This function overwrites $s0 and $s1
        # We should save those on the stack
        # This is PUSH'ing onto the stack
        addi $sp,$sp,-4# Adjust stack pointer
        sw $s0,0($sp)           # Save $s0
        addi $sp,$sp,-4# Adjust stack pointer
        sw $s1,0($sp)           # Save $s1

        # Do the function math
        li $s0, 3
        mul $s1,$s0,$a0# s1 = 3*$a0  (i.e. 3*a)
        addi $s1,$s1,5          # 3*a+5

        # Save the return value in $v0
        move $v0,$s1

        # Restore saved register values from stack in opposite order
        # This is POP'ing from stack
        lw $s1,0($sp)           # Restore $s1
        addi $sp,$sp,4          # Adjust stack pointer
        lw $s0,0($sp)           # Restore $s0
        addi $sp,$sp,4          # Adjust stack pointer

        # Return from function
        jr $ra                  # Jump to addr stored in $ra

# ------------------------------------------------------------------

        # Start .data segment (data!)
        .data
x:      .word 5
y:      .word 0
msg1:   .asciiz "y="
lf:     .asciiz"\n"
```

Lab 10

Lab 10 – MIPS Assembly Programming (Basic)

- **Consists of five small programs which demonstrate basic assembly concepts**
  - **Arithmetic**
  - **Branches**
  - **Loops**
  - **Arrays**
  - **I/O, Loops and Arrays**
- **Use QtSpim to test**