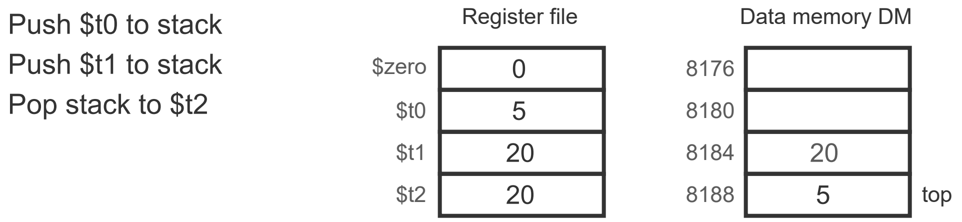
5.20 Subroutines and the program stack

Stack

A ***stack*** is a data structure in which items are inserted on or removed from the top of the stack. A stack ***push*** operation inse the top of the stack. A stack ***pop*** operation removes and returns the item at the top of the stack.

PARTICIPATION ACTIVITY 5.20.1: Stack push and pop operations.

Start □ 2x speed



Push $t0 to stack
Push $t1 to stack
Pop stack to $t2

Register file

$zero	0
$t0	5
$t1	20
$t2	20

Data memory DM

8176		
8180		
8184	20	
8188	5	top

PARTICIPATION ACTIVITY 5.20.2: Stack push and pop operations.

Type the stack as: 1, 2, 3

1) Given stack: 7, 5 (top is 7).

Type the stack after the following push operation:
Push 8 to stack

Check Show answer

2) Given stack: 34, 20 (top is 34)

Type the stack after the following two push operations:
Push 11 to stack
Push 4 to stack

Check Show answer

3) Given stack: 5, 9, 1 (top is 5)

What is $t1 after the following pop operation?
Pop stack to $t1

Check Show answer

4) Given stack: 5, 9, 1 (top is 5)

Type the stack after the following pop operation:
Pop stack to $t1

Check Show answer

5) Given stack: 2, 9, 5, 8, 1, 3 (top is 2).

What is $t2 after the following pop operations?
Pop stack to $t1
Pop stack to $t2

Check Show answer

6) Given stack: 41, 8 (top is 41)

Type the stack after the following pop operations:
Pop stack to $t0
Push 2 to stack
Push 15 to stack
Pop stack to $t5

Check Show answer

Program stack and stack pointer

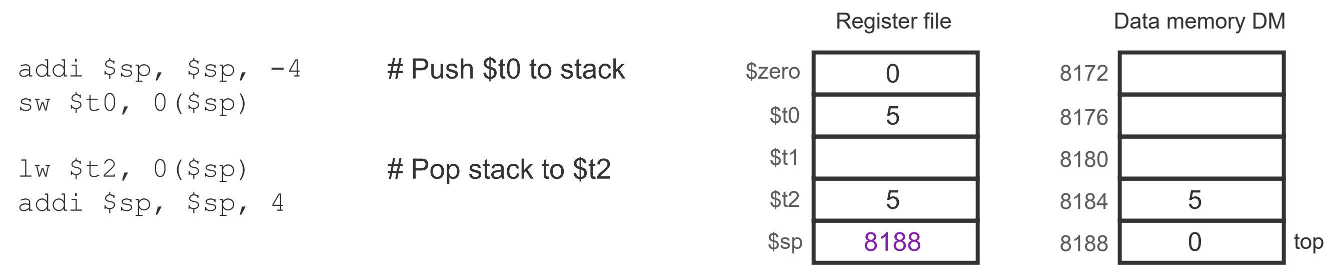
The ***program stack*** is a stack used by a program to store data for subroutines. The ***stack pointer*** (***$sp***) register is used to h of the top of the program stack. In MIPSzy, the $sp register is automatically initialized to the last data memory location, wh 8188. The MIPszy program stack is limited in size to 1KB, or 256 words. The stack grows toward decreasing memory addr value to the stack first decrements $sp by 4 and then copies the value held in a register to data memory at address $sp. Pc from the stack first copies the top of the stack to a register and then increments $sp by 4.

Stack overflow

*A **stack overflow** occurs when the number of values pushed to the stack exceeds the size allocated for the stack. Ex: Pushing 1005 values to the MIPSzy results in a stack overflow, as the stack size is limited to 1000 entries. A processor may have special circuitry to detect a stack overflow, allowing the system to execute special operation to handle the overflow, such as terminating the program.*

PARTICIPATION ACTIVITY 5.20.3: Instructions for stack push and pop operations.

Start □ 2x speed



```
addi $sp, $sp, -4      # Push $t0 to stack
sw $t0, 0($sp)

lw $t2, 0($sp)         # Pop stack to $t2
addi $sp, $sp, 4
```

Register file

$zero	0
$t0	5
$t1	
$t2	5
$sp	8188

Data memory DM

8172		
8176		
8180		
8184	5	
8188	0	top

PARTICIPATION ACTIVITY 5.20.4: MIPS program stack.

1) Complete the assembly to push the

value held in $t3 to the stack.

```
sw $t3, 0($sp)
```

Check Show answer

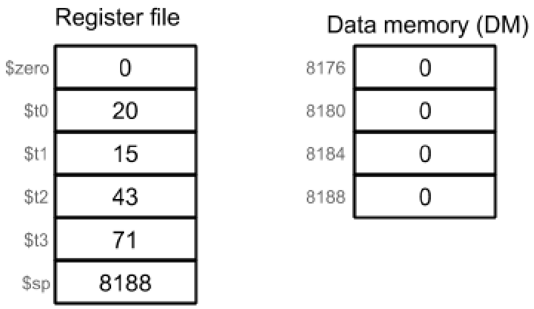
2) Complete the assembly to pop a value from the top of the stack to $t4.

```
addi $sp, $sp, 4
```

Check Show answer

3) Given the registers and stack content below, what is DM[8184] after the following instructions?

```
addi $sp, $sp, -4
sw $t0, 0($sp)
```



Register file

$zero	0
$t0	20
$t1	15
$t2	43
$t3	71
$sp	8188

Data memory (DM)

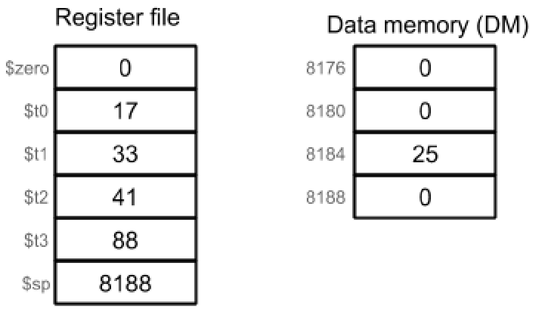
8176	0
8180	0
8184	0
8188	0

Check Show answer

4) Given registers and stack content below,

what is $sp after the following instructions?

```
addi $sp, $sp, -4
sw $t2, 0($sp)
addi $sp, $sp, -4
sw $t3, 0($sp)
```



Register file

$zero	0
$t0	17
$t1	33
$t2	41
$t3	88
$sp	8188

Data memory (DM)

8176	0
8180	0
8184	25
8188	0

Check Show answer

5) Given registers and stack content below, what is $t3 after the following instructions?

```
lw $t2, 0($sp)
addi $sp, $sp, 4
lw $t3, 0($sp)
addi $sp, $sp, 4
```

Register file

$zero	0
$t0	17
$t1	33
$t2	41
$t3	88
$sp	8176

Data memory (DM)

8176	105
8180	47
8184	25
8188	0

Check Show answer

6) Given registers and stack content below, what is DM[8176] after the following instructions?

```
lw $t2, 0($sp)
addi $sp, $sp, 4
```

Register file

$zero	0
$t0	0
$t1	0
$t2	0
$t3	0
$sp	8176

Data memory (DM)

8176	105
8180	47
8184	25
8188	0

Check Show answer

Using the program stack for subroutines

Because registers are limited, a subroutine call can use the program stack for arguments and return values rather than dire registers. The values stored in the program stack for a subroutine is called a ***stack frame***. A subroutine call using the progr performs the following steps.

1. Push subroutine arguments to program stack
2. Reserve space on program stack for the return value.
3. Jump to the subroutine.
4. Subroutine performs task storing return value to the reserved program stack location.
5. Subroutine returns.
6. Pop return value from program stack.
7. Pop arguments from program stack.

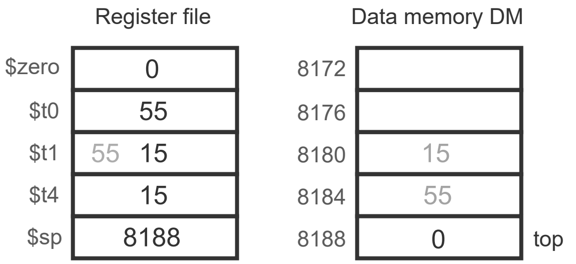
PARTICIPATION ACTIVITY 5.20.5: Calling subroutine using program stack: CalcOvertimeHours.

Start □ 2x speed

```
# Call CalcOvertimeHours subroutine
addi $sp, $sp, -4   # Push $t0 to stack
sw $t0, 0($sp)
addi $sp, $sp, -4   # Make space for return value
jal CalcOvertimeHours
lw $t1, 0($sp)      # Pop return value to $t1
addi $sp, $sp, 4
addi $sp, $sp, 4    # Pop argument from stack

...

CalcOvertimeHours:
  lw $t1, 4($sp)    # Load argument from stack
  addi $t2, $zero, 40
  slt $t3, $t1, $t2
  bne $t3, $zero, NoOvertime
  sub $t4, $t1, $t2
  j ReturnOvertime
NoOvertime:
  addi $t4, $zero, 0
ReturnOvertime:
```



Register file

$zero	0
$t0	55
$t1	55 15
$t4	15
$sp	8188

Data memory DM

8172		
8176		
8180	15	
8184	55	
8188	0	top

```
    sw $t4, 0($sp)    # Copy return value to stack
    jr $ra
```

In the subroutine, an offset is used in lw or sw instructions to load arguments or store the return value. Ex: For a subroutine argument and 1 return value, `0($sp)` is the address for the return value, and `4($sp)` is the address for the argument.

PARTICIPATION ACTIVITY 5.20.6: Calling subroutine using program stack.

Assume the program stack is used for all subroutine arguments and return values.

1) What is the stack frame size for a subroutine with one argument and one return value?

○ 1

○ 2

2) What is the stack frame size for a subroutine with one argument and no return values?

○ 1

○ 2

3) For a subroutine with 2 arguments and a return value, which instruction loads the first argument to $t1.

○ lw $t1, 0($sp)

○ lw $t1, 4($sp)

○

○ lw $t1, 8($sp)

4) For a subroutine with 1 argument and a return value, which instruction stores $t4 to the stack entry allocated for the return value.

○ sw $t4, 8($sp)

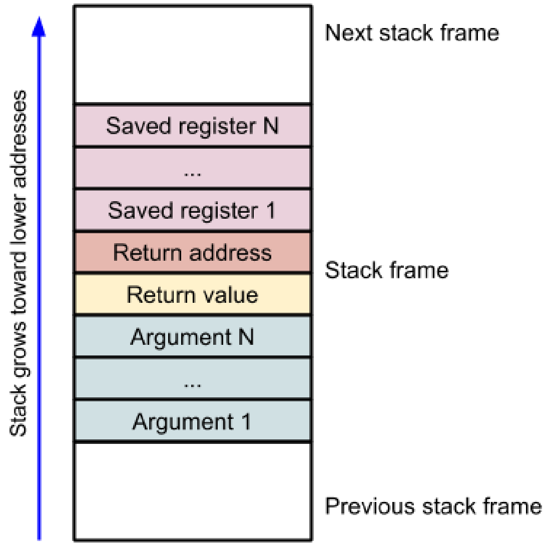
○ sw $t4, 4($sp)

○ sw $t4, 0($sp)

Saving return address and registers to the program stack

If a subroutine calls another subroutine, the value held in $ra must be saved before the second subroutine is called, becaus second subroutine writes a new value to $ra. So, a subroutine that calls another subroutine will also push $ra to the progra the value held in registers used by a subroutine may still be needed by code that called the subroutine. To avoid overwriting subroutine can save the values held in any registers used by the subroutine to the program stack, and restore them before the subroutine. The following shows the organization for a complete MIPSzy stack frame.

Figure 5.20.1: MIPSzy stack frame.



PARTICIPATION ACTIVITY 5.20.7: Saving and restoring register using the program stack.

Complete the CalcQuadFunc subroutine to save and restore any registers used by the subroutine using the program stack by determine the missing instructions for the highlighted lines.

```
# Computes x^2 + 2x
CalcQuadFunc:
   # Save $t0, $t1, and $t2 to stack
   addi $sp, $sp, -4
   sw $t0, 0($sp)
   addi $sp, $sp, -4
   sw $t1, 0($sp)
   (a)
   (b)
   lw $t0, 16($sp)    # Load argument from stack
   addi $t1, $zero, 2
   mul $t2, $t0, $t1  # Calculate 2*x
   mul $t1, $t0, $t0  # Calculate x*x
   add $t2, $t2, $t1  # Calculate x*x + 2x
   sw $t2, 12($sp)    # Copy return value to stack

   # Restore $t0, $t1, and $t2 from stack
   (c)
   addi $sp, $sp, 4
   lw $t1, 0($sp)
   addi $sp, $sp, 4
   (d)
   addi $sp, $sp, 4
   jr $ra
```

sw $t2, 0($sp)   lw $t2, 0($sp)   addi $sp, $sp, -4   lw $t0, 0($sp)

(a)

(b)

(c)

(d)

Reset

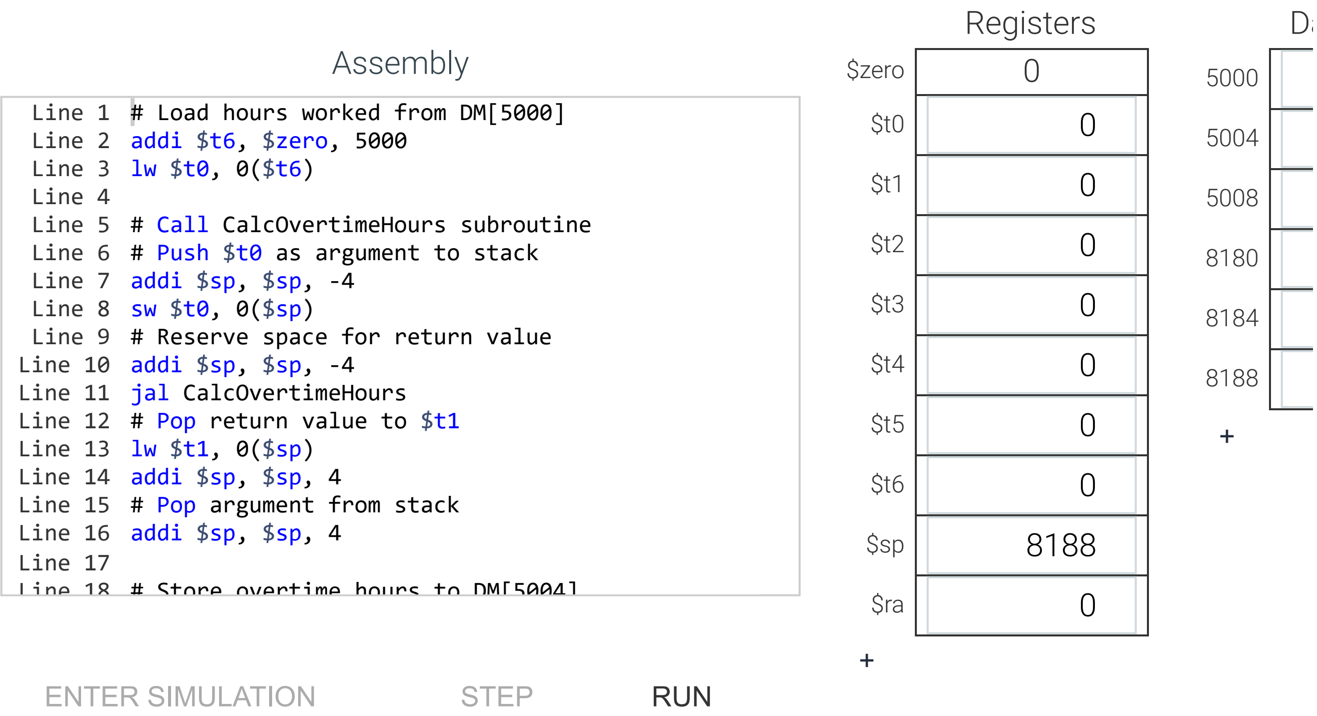
PARTICIPATION

ACTIVITY

5.20.8: Saving and restoring registers using the program stack.

The CalcOvertimeHours subroutine uses registers $t1, $t2, $t3, and $t4. Modify the subroutine to save and restore those registers.

1. At the beginning of the subroutine, push the values held in $t1, $t2, $t3, and $t4 to the stack in that order.
2. Because more values have been pushed on the stack, the offsets for the location of the subroutine's argument and return value in the stack have changed. Modify the `lw $t1, 4($sp)` and `sw $t4, 0($sp)` instructions to use the correct offsets.
3. Just before the `jr $ra` instruction, restore the saved registers' values by popping 4 values from the stack into $t4, $t3, $t2, and $t1, in that order. Note the order is reversed from when the values were pushed on the stack.



```
# Load hours worked from DM[5000]
addi $t6, $zero, 5000
lw $t0, 0($t6)

# Call CalcOvertimeHours subroutine
# Push $t0 as argument to stack
addi $sp, $sp, -4
sw $t0, 0($sp)
# Reserve space for return value
addi $sp, $sp, -4
jal CalcOvertimeHours
# Pop return value to $t1
lw $t1, 0($sp)
addi $sp, $sp, 4
# Pop argument from stack
addi $sp, $sp, 4

# Store overtime hours to DM[5004]
```

$zero	0
$t0	0
$t1	0
$t2	0
$t3	0
$t4	0
$t5	0
$t6	0
$sp	8188
$ra	0

5000	
5004	
5008	
8180	
8184	
8188	

More options

CHALLENGE ACTIVITY 5.20.1: Push and pop from stack.

Start

Push $t5 to the stack. Update $sp appropriately.

addi ▼ $t5 ▼, $t5 ▼, 0

addi ▼ $t5 ▼, $t5 ▼, 0

Registers	
$t5	2
$sp	8188

Data memory	
8184	0
8188	0

1 2 3 4 5

Check Next

Provide feedback on this section