Lab 3

CMPUT 229

University of Alberta

Outline

The Assignment: `CubeStats`

Input:

- `corner` in `$a0` - the address of the top left corner of an n-dimensional cube residing in an n-dimensional array.
- `edge` in `$a1` - the edge size of the cube.
- n in `$a2` - the number of dimensions.
- `size` in `$a3` - the total edge size of the array.

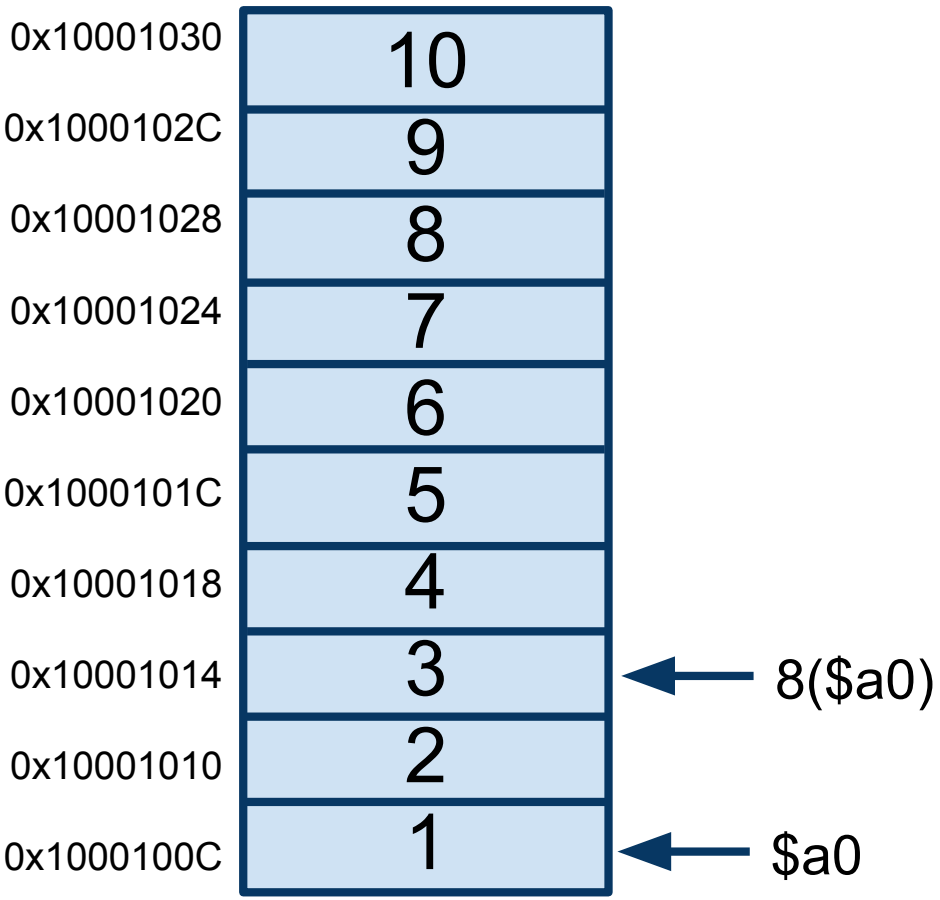
Output:

- In `$v0`, a signed integer containing the range of the elements in the specified cube.
- In `$v1`, a signed integer containing the floor of the average of the elements in the specified cube.

You may assume that the parameters are correct, and the cube is contained in the base array.

A 1-Dimensional Example

A 1-D cube is just an array:



If your parameters were:

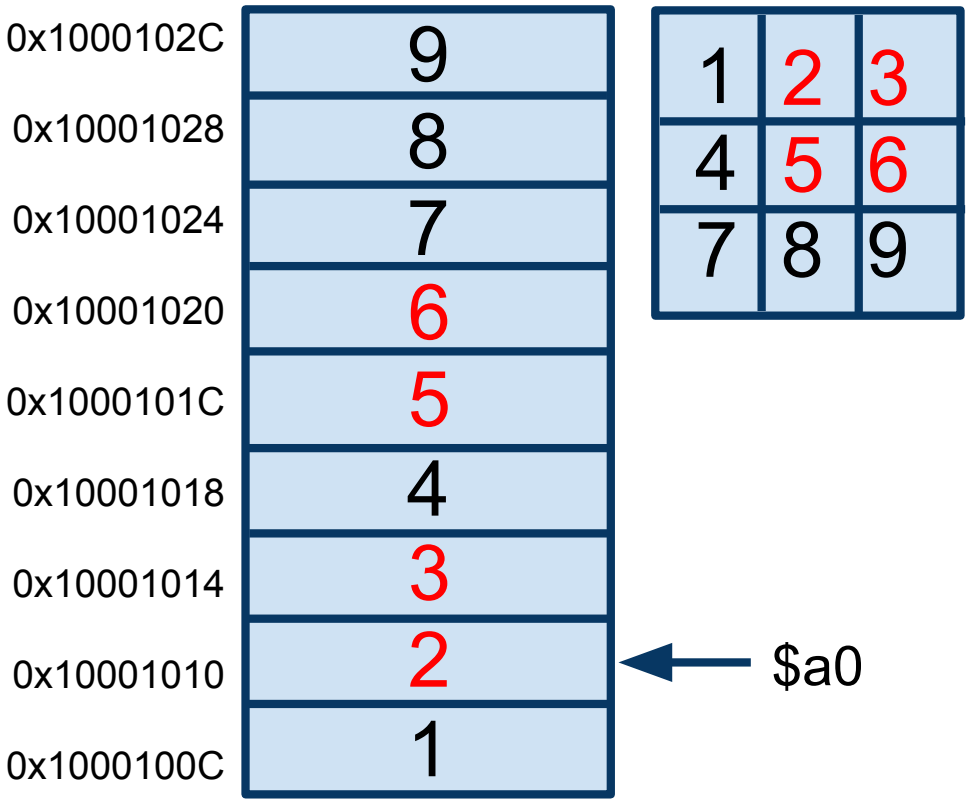
- $a0 = 0x1000 100C
- $a1 = 5
- $a2 = 1
- $a3 = 10

Then you would return:

- $v0 = 4
- $v1 = 3

A 2-Dimensional Example

A 2-D cube (with row-major storage) looks like:



If your parameters were:

- $a0 = 0x1000 1010
- $a1 = 2
- $a2 = 2
- $a3 = 3

Then you would return:

- $v0 = 4
- $v1 = 4

The Stack

- To preserve data across subroutine calls, we use the stack.
- The extent of the stack is defined by the **stack pointer**, `$sp`.
- Each executing subroutine gets its own **stack frame**, which is delimited by the **frame pointer**, `$fp`, and the stack pointer.
- To grow the stack, you subtract from `$sp`.
- To access data on the stack, you use `$fp`, since the offsets from `$sp` might change if you grow the stack during your subroutine.

Stack Example

- We are inside a subroutine, and need to call another subroutine.
- Assume we have not used the stack yet.
- We have a value we want to keep in `$t0`.
- We also need to keep `$ra` so that we know how to return.
- We must also keep `$fp` in case the caller was using it.

Stack Example

- We are inside a subroutine, and need to call another subroutine.
- Assume we have not used the stack yet.
- We have a value we want to keep in `$t0`.
- We also need to keep `$ra` so that we know how to return.
- We must also keep `$fp` in case the caller was using it.

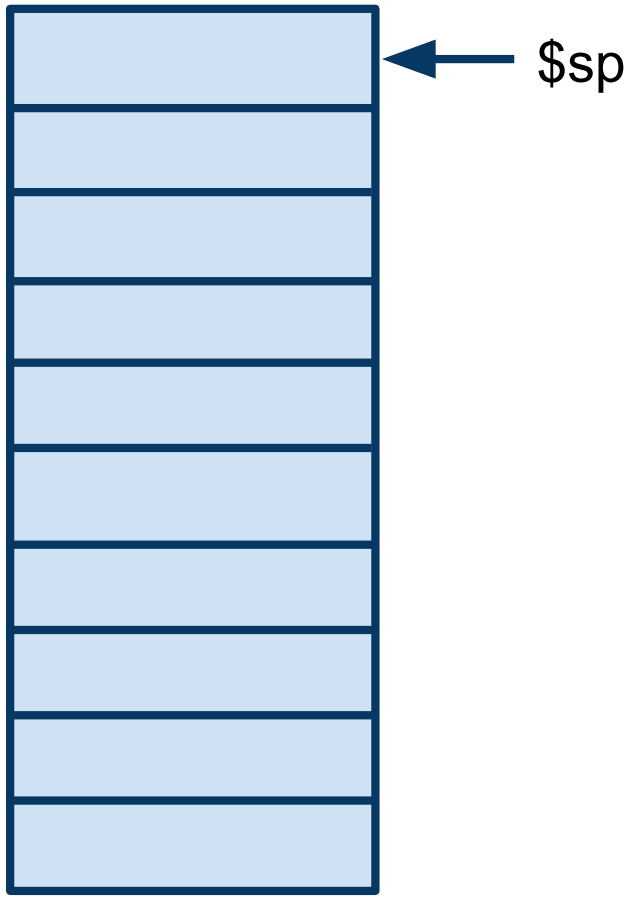
Before the call:

```
# Set up our frame
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
# Grow the stack
subu $sp, $sp, 8
# Store our values
sw $ra, -4($fp)
sw $t0, -8($fp)
# Now we can safely jal
jal some_sub
```

After the call:

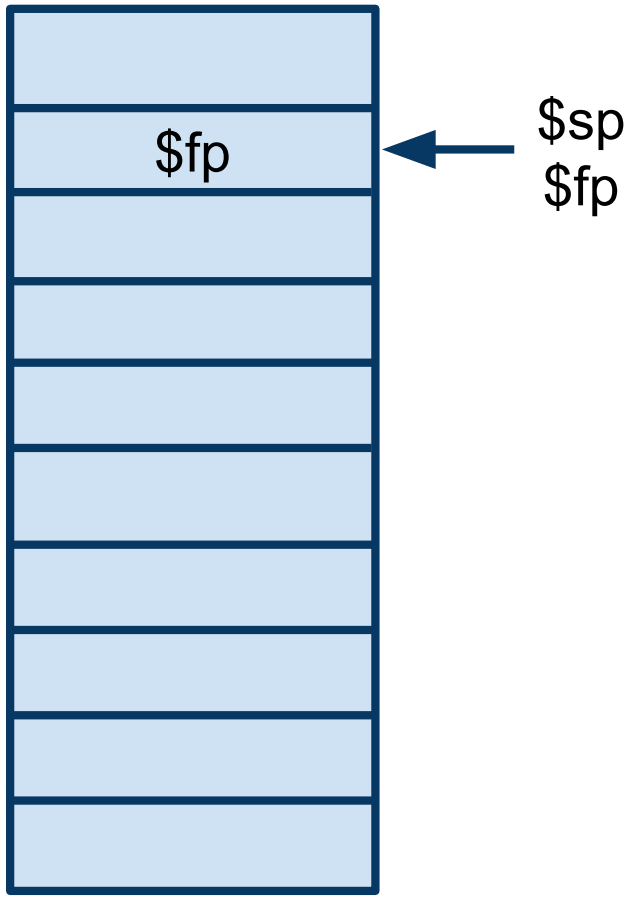
```
# Restore our values
lw $ra, -4($fp)
lw $t0, -8($fp)
# Unwind
addu $sp, $sp, 12
lw $fp, -4($sp)
jr $ra
```

Stack Example – Illustrated



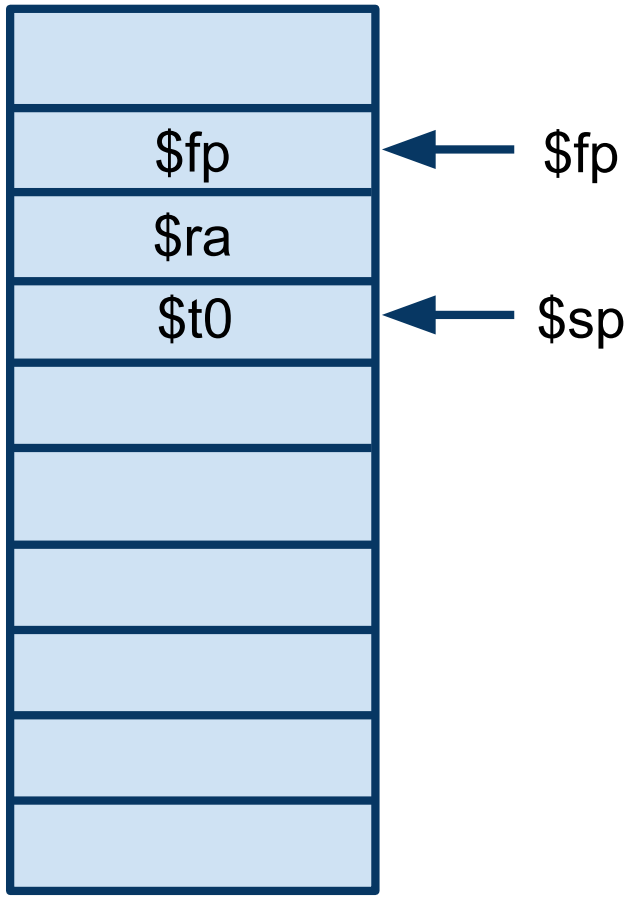
Stack Example – Illustrated

```
# Set up our frame
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
```



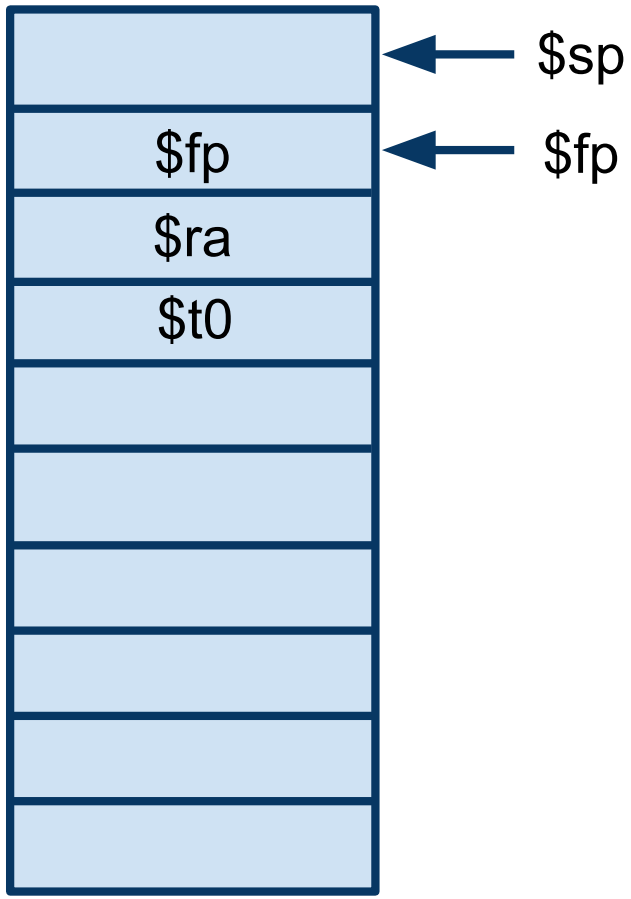
Stack Example – Illustrated

```
# Set up our frame
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
# Grow the stack
subu $sp, $sp, 8
# Store our values
sw $ra, -4($fp)
sw $t0, -8($fp)
```



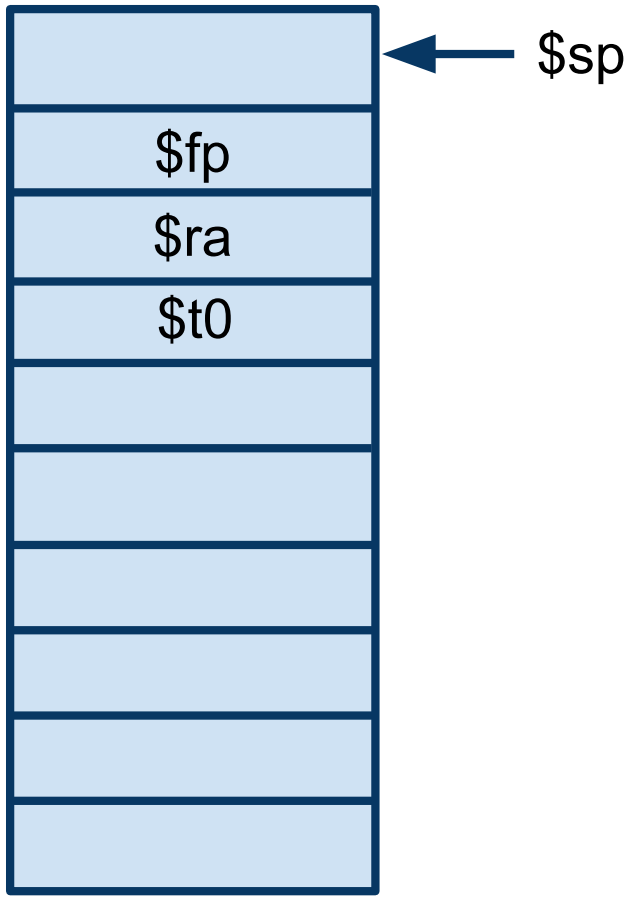
Stack Example – Illustrated

```
# Set up our frame
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
# Grow the stack
subu $sp, $sp, 8
# Store our values
sw $ra, -4($fp)
sw $t0, -8($fp)
# Now we can safely jal
jal some_sub
# Restore our values
lw $ra, -4($fp)
lw $t0, -8($fp)
# Unwind
addu $sp, $sp, 12
```



Stack Example – Illustrated

```
# Set up our frame
subu $sp, $sp, 4
sw $fp, 0($sp)
move $fp, $sp
# Grow the stack
subu $sp, $sp, 8
# Store our values
sw $ra, -4($fp)
sw $t0, -8($fp)
# Now we can safely jal
jal some_sub
# Restore our values
lw $ra, -4($fp)
lw $t0, -8($fp)
# Unwind
addu $sp, $sp, 12
lw $fp, -4($sp)
jr $ra
```



Caller/Callee Conventions

In MIPS the following convention for calling subroutines applies:

- When calling another routine, the caller or callee is responsible for saving some registers.
- MIPS does not save automatically these registers, it is the coder's responsability.
- There are two types of registers: callee-saved and caller-saved.
- Callee-saved ($s registers) must be restored to the same value they had when the callee obtained control. This should always be done, as the caller expects all the $s values untouched.
- Caller-saved registers are the temporary registers $t. Programers should use them freely as the caller is obliged to save them before calling another routine.

Assignment Tips

- Read specifications very carefully. Pay special attention to what you have to include - we don't want a `main` method.
- Read and understand the main-row-first .c and .s files.
- You are free to use the "power" subroutine from main-row-first.s
- Adhere to the calling convention to avoid losing marks.
- Test your assignments on the lab machines before you submit.
- Look at the marksheet to get an idea of how the grading will be done.
- Style marks are easy marks. Format your code like the `example.s` file we provided, and write good comments.
- Be sure to submit code that runs and loads. Otherwise you will lose many marks.

Lab 3 Questions?