Assignment A P2 Lab 6 for loop

Lab 6: for loop

```
int num[5] = {1,2,3,4,5};
int N = 5;
int sum = 0;
for(i=0;i<N;i+=1)
    sum += num[i]
```

$t0	num[ ]
$t1	i
$t2	N
$t3	sum
$t4	test result
$t5	num[i], offset

Submit Items:

GitHub Link to your file directly. (Please make it a link, not a text)

https://github.com/AVC-CS/p2-mtvonbargen/blob/63eb552f6da606090c44ff6a6b33f18f3b7d8ae9/P2A6.asm

Elaborate on your program

```
################ Data segment ####################
.data
    num:    .word 1, 2, 3, 4, 5
    sum:    .word 0
    save:   .space 20

################ Code segment ####################
.text
    #for (i=0, i<N, i+=1)   basic loop conditions
    #    sum += num[i]      sum + array value

    main:
        la $t0, num                 # Num[] address
        lw $t3, sum                 # Sum = 0
        li $t1, 0                   # i = 0
        li $t2, 4                   # N = 5
        la $t7, save                # allocated space for previous sums

    loop:
        sll $t5, $t1, 2             # multiply $t1(i = 0) by 2(^2 = 4) to calculate byte offset, store in $t5(0)
        add $t6, $t5, $t0           # add $t5(0) to the memory address of $t0(10010000) and store it in $t6
        lw $t6, 0($t6)              # saves the value located in the memory address that's stored in $t6(1) previously.

        add $t3, $t3, $t6           # adds $t3(Sum = 0) and $t6(1), stores result back into $t3
        sw $t3, 0($t7)              # saves sum into save array

        slt $t4, $t1, $t2           # (i < N) stores test result into $t4

        addi $t1, 1                 # (i += 1)
        addi $t7, $t7, 4            # increment memory address of save

        bne $t4, $zero, loop        # (if test result is not 0, keep looping)
        j exit


    exit:
        sw $t3, sum                 # saves sum into sum memory
        li $v0, 10
        syscall
```

Register and Variables mapping table (Examples using first iteration)

$t0	Holds the memory address of num[]
$t1	Holds i (index place)
$t2	Holds N (condition comparison)
$t3	Sum
$t4	Test result
$t5	Num[i], offset calculation
$t6	Memory Address Calculation of Num[i]
$t7	Saves previous sum results

Final Results(Screenshot) = Register and Memory snapshot (at the very end)

```
R8 (t0)  = 10010000
R9 (t1)  = 00000005
R10 (t2) = 00000004
R11 (t3) = 0000000f
R12 (t4) = 00000000
R13 (t5) = 00000010
R14 (t6) = 00000005
R15 (t7) = 1001002c
```

```
                    User Data Segment

[10010000] 00000001 00000002 00000003 00000004 ................
[10010010] 00000005 0000000f 00000001 00000003 ................
[10010020] 00000006 0000000a 0000000f 00000000 ................
```

Pseudocode (abstractional sketch)

```
# Data Initialization
array num[] = {1, 2, 3, 4, 5}
integer sum = 0
space save[] = allocated (for storing intermediate sums)

# Main Program
Set i = 0              # Loop counter
Set N = 5              # Number of elements in the array
Point to num array     # Base address of the array
Point to save array    # Base address of save memory space
```

```
# Loop through the array
while (i < N):
    Calculate the address of num[i]     # Find the memory location of the current array element
    Load num[i] into a temporary value  # Retrieve the value at that address
    Add num[i] to sum                   # Update the running total
    Store sum in the save array         # Save the intermediate sum to save[]

    Increment i                         # Move to the next index
    Move to the next save memory slot   # Update save pointer

# Exit
Store the final sum in the sum variable
Terminate the program
```

Register Value. What is the purpose of this register? How is it updated throughout your program?

$t0	Holds the memory address of num[]	10010000 (Stays)
$t1	Holds i (index place)	0 > 1 > 2 > 3 > 4 (Increments by 1)
$t2	Holds N (condition comparison)	0000004 = 4 (Stays)
$t3	Sum	1 > 3 > 6 > 10 > 15
$t4	Test result	1 > 1 > 1 > 1 > 0 (1 true, 0 false, ends loop once loop condition is false) (Updates)
$t5	Num[i], offset calculation	0 > 4 > 8 > c > 10 (got rid of leading 0s) (Increments by 4)
$t6	Calculated Memory Address. Stores memory address of i first, then overwrites with the value held by i in the array.	Before: 10010000 After: 00000000 = 0 Before: 10010004 After: 00000001 = 1 Before: 10010008 After: 00000002 = 2 Before: 1001000c After: 00000003 = 3 Before: 10010010 After: 00000004 = 4 (Memory address Increments by 4)
$t7	Saves previous Test results	1 > 1 > 1 > 1 > 0 (just allocates result values into the save array to record result history) (Updates)

Show all the updated history of each register value (Trace the program execution like a debugging tool)

1st Iteration

```
R8 (t0)  = 10010000
R9 (t1)  = 00000001
R10 (t2) = 00000004
R11 (t3) = 00000001
R12 (t4) = 00000001
R13 (t5) = 00000000
R14 (t6) = 00000001
R15 (t7) = 1001001c
```

$t0, $t2 will stay the same at all iterations. $t1 is i, and it'll increment by 1 after every iteration, before this it was 0 so the offset calculation will start at the very first array index. This is the value used for the next iteration. $t6 holds the memory address of the current index and pulls out the integer from it, which is 00000001 = 1. $t4 is the result of the loop condition, which dictates if the value in the index is less than N = 5. Therefore, it's true, so the result is saved as 1 to indicate true. $t7 just saves the memory address that it will allocate the sum into.

2nd Iteration

```
R8 (t0)  = 10010000
R9 (t1)  = 00000002
R10 (t2) = 00000004
R11 (t3) = 00000003
R12 (t4) = 00000001
R13 (t5) = 00000004
R14 (t6) = 00000002
R15 (t7) = 10010020
```

$t1 is i, and it'll increment by 1 after every iteration, before this it was 1 so the offset calculation will start at the second array index. This is the value used for the next iteration. $t6 holds the memory address of the current index and pulls out the integer from it, which is 00000002 = 2. $t4 is the result of the loop condition, which dictates if the value in the index is less than N = 5. Therefore, it's true, so the result is saved as 1 to indicate true. $t7 just saves the memory address that it will allocate the sum into.

3rd Iteration (Final Iteration)

```
R8 (t0)  = 10010000
R9 (t1)  = 00000003
R10 (t2) = 00000004
R11 (t3) = 00000006
R12 (t4) = 00000001
R13 (t5) = 00000008
R14 (t6) = 00000003
R15 (t7) = 10010024
```

$t1 is i, and it'll increment by 1 after every iteration, before this it was 2 so the offset calculation will start at the third array index. This is the value used for the next iteration. $t6 holds the memory address of the current index and pulls out the integer from it, which is 00000003 = 3. $t4 is the result of the loop condition, which dictates if the value in the index is less than N = 5. Therefore, it's true, so the result is saved as 1 to indicate true. $t7 just saves the memory address that it will allocate the sum into.

4th Iteration

```
R8 (t0)  = 10010000
R9 (t1)  = 00000004
R10 (t2) = 00000004
R11 (t3) = 0000000a
R12 (t4) = 00000001
R13 (t5) = 0000000c
R14 (t6) = 00000004
R15 (t7) = 10010028
```

$t1 is i, and it'll increment by 1 after every iteration, before this it was 3 so the offset calculation will start at the fourth array index. This is the value used for the next iteration. $t6 holds the memory address of the current index and pulls out the integer from it, which is 00000004 = 4. $t4 is the result of the loop condition, which dictates if the value in the index is less than N = 5. Therefore, it's true, so the result is saved as 1 to indicate true. $t7 just saves the memory address that it will allocate the sum into.

5rd Iteration (Final Iteration)

```
R8 (t0)  = 10010000
R9 (t1)  = 00000005
R10 (t2) = 00000004
R11 (t3) = 0000000f
R12 (t4) = 00000000
R13 (t5) = 00000010
R14 (t6) = 00000005
R15 (t7) = 1001002c
```

$t1 is i, and it'll increment by 1 after every iteration, before this it was 4 so the offset calculation will start at the fifth array index. This is the value used for the next iteration. $t6 holds the memory address of the current index and pulls out the integer from it, which is 00000004 = 4 (this is the 5th index in the array don't worry). $t6 is the result of the loop condition, which dictates if the value in the index is less than N = 5. Therefore, it's false, so the

result is saved as 0 to indicate false, so it will exit the loop and program. $t7 just saves the memory address that it will allocate the sum into.

The interaction between register and memory space

The memory space holds the array of numbers (num), the cumulative sum (sum), and an allocated area (save) to store intermediate results. Registers like $t0 store the base address of the array, $t6 holds individual array values loaded from memory, and $t3 accumulates the running total. The program calculates memory addresses dynamically using offsets and stores intermediate and final results back into memory for persistence.

Program logic

The program iterates through an array (num), adding each element to a cumulative total (sum) while storing intermediate sums in an allocated space (save). It uses a loop controlled by a counter (i) and a comparison to ensure all elements up to a specified size (N) are processed. The loop calculates each element's memory address, loads its value, adds it to the sum, and checks the loop condition to continue until all elements are processed. Finally, the total sum is saved to memory and the program exits.