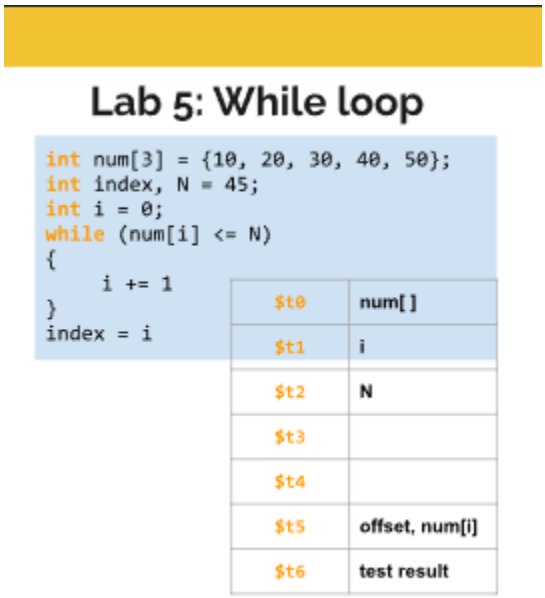
Assignment A P2 Lab 5 while loop



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/P2A5.asm

Elaborate on your program

```
################ Data segment ####################
.data
    num:        .word   10, 20, 30, 40, 50
    result:     .word   0
    save:       .space  20
    index:      .word   0

################ Code segment ####################
.text
.globl main
main:
    la $t0, num             # load address of num array
    li $t1, 0               # load 0 into $t1 (i = 0)
    li $t2, 45              # load 45 into $t2 (N = 45)
    la $t7, save            # allocated space for previous results

loop:

    sll $t5, $t1, 2         # multiply $t1(i = 0) by 2(^2 = 4) to calculate byte offset, store in $t5(0)
    add $t4, $t5, $t0       # add $t5(0) to the memory address of $t0(10010000) and store it in $t4
    lw $t4, 0($t4)          # saves the value located in the memory address that's stored in $t4 previously.
    slt $t6, $t4, $t2       # set on less than. Compares $t4 if it's less than or equal to $t2. Saves the result into $t6. (num[i] <- N)
    sw $t6, 0($t7)          # saves result into save array.
    sw $t6, result          # also saves result into result memory
    addi $t1, 1             # (i += 1) add 1 to increment i.
    addi $t7, $t7, 4        # increment memory address of save.
    bne $t6, $zero, loop    # if $t6(result) does not equal to 0, then execute loop.
    j exit


exit:
    sw $t1, index
    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	Not used
$t4	Holds the calculated memory address of num[i]. Then loads the value residing in that same memory address.
$t5	Offset calculation
$t6	Test result
$t7	Saves previous Test results

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

```
R8 (t0)  = 10010000
R9 (t1)  = 00000005
R10 (t2) = 0000002d
R11 (t3) = 00000000
R12 (t4) = 00000032
R13 (t5) = 00000010
R14 (t6) = 00000000
R15 (t7) = 1001002c
```

```
                    User Data Segment

[10010000] 0000000a 00000014 0000001e 00000028 ............(...
[10010010] 00000032 00000000 00000001 00000001 2...............
[10010020] 00000001 00000001 00000000 00000005 ................
```

Pseudocode (abstractional sketch)

```
# Data Initialization
num = [10, 20, 30, 40, 50]  # Array of numbers
result = 0                  # Variable to store comparison result
save = []                   # Array to store comparison results
index = 0                   # To store final index

# Main program
i = 0                     # Start with index 0
N = 45                    # Threshold value
```

```
while num[i] <= N:            # Loop while num[i] is less than or equal to N
   result = (num[i] <= N)    # Set result to 1 if num[i] <= N, else 0
   save.append(result)       # Store result in save array
   i += 1                    # Increment index i

index = i                    # Store final index
exit                         # End of 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 > 5 (Increments by 1)
$t2	Holds N (condition comparison)	0000002d = 45 (Stays)
$t3	Not used	
$t4	Holds the calculated memory address of num[i]. Then loads the value residing in that same memory address.	Before: 10010000 After: 0000000a = 10 Before: 10010004 After: 00000014 = 20 Before: 10010008 After: 0000001e = 30 Before: 1001000c After: 00000028 = 40 Before: 10010010 After: 00000032 = 50 (Increments)
$t5	Offset calculation	0 > 4 > 8 > c > 10 (got rid of leading 0s) (Increments by 4)
$t6	Test result	1 > 1 > 1 > 1 > 0 (1 true, 0 false, ends loop once loop condition is false) (Updates)
$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) = 0000002d
R11 (t3) = 00000000
R12 (t4) = 0000000a
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. $t4 holds the memory address of the current index and pulls out the integer from it, which is a = 10. $t6 is the result of the loop condition, which dictates if the value in the index is less than N = 45. 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 result into.

2nd Iteration

```
R8 (t0)  = 10010000
R9 (t1)  = 00000002
R10 (t2) = 0000002d
R11 (t3) = 00000000
R12 (t4) = 00000014
R13 (t5) = 00000004
R14 (t6) = 00000001
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. $t4 holds the memory address of the current index and pulls out the integer from it, which is 14 = 20. $t6 is the result of the loop condition, which dictates if the value in the index is less than N = 45. Therefore, it's true, so the result is saved as 1 to indicate true.

3rd Iteration (Final Iteration)

```
R8 (t0)  = 10010000
R9 (t1)  = 00000003
R10 (t2) = 0000002d
R11 (t3) = 00000000
R12 (t4) = 0000001e
R13 (t5) = 00000008
R14 (t6) = 00000001
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. $t4 holds the memory address of the current index and pulls out the integer from it,

which is 12 = 30. $t6 is the result of the loop condition, which dictates if the value in the index is less than N = 45. Therefore, it's true, so the result is saved as 1 to indicate true.

4th Iteration

```
R8 (t0)  = 10010000
R9 (t1)  = 00000004
R10 (t2) = 0000002d
R11 (t3) = 00000000
R12 (t4) = 00000028
R13 (t5) = 0000000c
R14 (t6) = 00000001
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. $t4 holds the memory address of the current index and pulls out the integer from it, which is 28 = 40. $t6 is the result of the loop condition, which dictates if the value in the index is less than N = 45. Therefore, it's true, so the result is saved as 1 to indicate true.

5rd Iteration (Final Iteration)

```
R8 (t0)  = 10010000
R9 (t1)  = 00000005
R10 (t2) = 0000002d
R11 (t3) = 00000000
R12 (t4) = 00000032
R13 (t5) = 00000010
R14 (t6) = 00000000
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 third array index. This is the value used for the next iteration. $t4 holds the memory address of the current index and pulls out the integer from it, which is 32 = 50. $t6 is the result of the loop condition, which dictates if the value in the index is less than N = 45. Therefore, it's false, so the result is saved as 0 to indicate false and exits the loop.

The interaction between register and memory space

The program uses registers like $t0 to hold the base address of the num array, $t1 for the index i, and $t2 for the threshold value N. The offset to access each element in the array is calculated using the index in $t1, and the result of each comparison is stored in memory (save[]) via the sw instruction. The registers facilitate quick data manipulation, while memory stores both the array and results over the course of the program.

Program logic

The program iterates through the num[] array, comparing each element to a threshold value N. For each comparison, it stores the result (1 if the value is less than or equal to N, 0 otherwise) in the save[] array. The loop continues until an element greater than N is encountered, and the index of the last valid element is stored in the index. The program uses the slt instruction to compare the values and manages control flow with conditional branching to continue the loop until the condition is no longer met.