

Assignment A P2 Lab 4 do-while loop

Lab 4: do -While loop

```
int num[3] = {-1, -2, -3};
int total = 0;
int i = 0;
int N = 3
do{
    total += num[i]
    i += 1
} while ( i != N);
```

\$t0	num[]
\$t1	total
\$t2	N
\$t3	i
\$t4	
\$t5	offset or num[i]

show your result register and memory value after program execution (Screenshot)

Submit Items:

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

<https://github.com/AVC-CS/p2-mtvonbargen/blob/6eba2246bc44ef53a30596bf7d477e71d8b4259a/P2A4.asm>

Elaborate on your program

```
##### Data segment #####
.data
    num:        .word -1, -2, -3    # Array of integers to be summed
    total:      .word 0             # Variable to store the sum of the array

##### Code segment #####
.text
main:
    # do
    # total += num[i]    total is equal to the sum of total and num[i]
    # i += 1            increment i (array index) by 1.
    # while ( i != N)   check if i is not equal to N (number of indexes)

    la $t0, num         # Load the address of the array num into $t0
    lw $t1, total       # Load total into $t1(0)
    li $t2, 3           # N = 3
    li $t3, 0           # i = 0
    li $t5, 0           # initialize offset at 0

loop:
    sll $t5, $t3, 2      # multiply $t3(i(0)) by 2^2(4) to calculate byte offset $t5(0)
    add $t4, $t5, $t0     # add offset ($t5(0)) with num address ($t0(10010000)) in order to compute memory address. $t4(10010000)
    lw $t4, 0($t4)       # overwrite $t4 with the value located at $t4's current calculated index.
    add $t1, $t1, $t4     # total += num[i], adds ($t1(0)) and ($t4(-1))
    addi $t3, $t3, 1     # i += 1 (i = 1)
    bne $t3, $t2, loop   # branch off to loop, if $t3(1) does not equal $t2(3)
    j exit

exit:
    li $v0, 10
    syscall
```

IT WORKS IT WORKKSS AHH

Program Results

Special Registers

- PC = 00400044
- EPC = 00000000
- Cause = 00000000
- BadAddr = 00000000
- Status = 3000fff10
- HI = 00000000
- LO = 00000000

General Registers

- R0 (r0) = 00000000
- R1 (at) = 10010000
- R2 (v0) = 00000000
- R3 (v1) = 00000000
- R4 (a0) = 00000000
- R5 (a1) = 7fffffff78
- R6 (a2) = 7fffffff7c
- R7 (a3) = 00000000
- R8 (t0) = 10010000
- R9 (t1) = 00000000
- R10 (t2) = 00000003
- R11 (t3) = 00000000
- R12 (t4) = 10010000
- R13 (t5) = 00000000
- R14 (t6) = 00000000
- R15 (t7) = 00000000

User Text Segment

```

[00400000] lw $4, 0($29)          ; 183: lw $a0 0($sp)    # argc
[00400004] addiu $5, $29, 4        ; 184: addiu $a1 $sp 4    # argv
[00400008] addiu $6, $5, 4        ; 185: addiu $a2 $a1 4    # envp
[0040000c] sll $2, $4, 2          ; 186: sll $v0 $a0 2
[00400010] addu $6, $6, $2        ; 187: addu $a2 $a2 $v0
[00400014] jal 0x00400024 [main]  ; 188: jal main
[00400018] nop                   ; 189: nop
[0040001c] ori $2, $0, 10         ; 191: li $v0 10
[00400020] syscall               ; 192: syscall          # syscall 10 (exit)
[00400024] lui $8, 4097 [num]     ; 19: la $t0, num       # Load the address of the arr
[00400028] lui $1, 4097          ; 20: lw $t1, total     # Load total into $t1(0)
[0040002c] lw $9, 12($1)         ; 21: li $t2, 3         # N = 3
[00400030] ori $10, $0, 3        ; 22: li $t3, 0         # i = 0
[00400034] ori $11, $0, 0        ; 23: li $t5, 0         # initialize offset at 0
[00400038] sll $13, $11, 2       ; 26: sll $t5, $t3, 2    # multiply $t3(i(0)) by 2^2(4)
[0040003c] add $12, $13, $8      ; 27: add $t4, $t5, $t0 # add offset ($t5(0)) with nt
[00400040] lw $12, 0($12)        ; 28: lw $t4, 0($t4)    # overwrite $t4 with the val
[00400044] add $9, $9, $12       ; 29: add $t1, $t1, $t4 # total += num[i], adds ($t1(
[00400048] addi $11, $11, 1      ; 30: addi $t3, $t3, 1   # i += 1 (i = 1)
[00400050] bne $11, $10, -20 [loop-0x00400050]; 31: bne $t3, $t2, loop # branch off to loop, if $
[00400054] j 0x00400058 [exit]  ; 32: j exit
  
```

User Data Segment

```

[10010000] ffffffff ffffffff ffffffff 00000000
  
```

Kernel Data Segment

```

[90000000] 78452020 74706563 206e6f69 636f2000
[90000010] 72727563 61206465 6920646e 726f6e67
[90000020] 000a6465 495b2020 7265746e 74707572
[90000030] 2000205d 4c545b20 20005d42 4c545b20
[90000040] 20005d42 4c545b20 20005d42 64415b20
[90000050] 73657264 72652073 20726f72 69206e69
[90000060] 2f74736e 61746164 74656620 205d6863
[90000070] 5b202000 72646441 20737365 6f727265
[90000080] 6e692072 6f747320 205d6572 5b202000
[90000090] 20646142 74736e69 74637572 206e6f69
[900000a0] 72646461 5d737365 20200020 6461425b
  
```

User Stack

```

[7fffffff] 00000000 00000000 7fffffff
  
```

10010000 = ffffffff = -1

10010004 = fffffffe = -2

10010008 = ffffffff = -3

R8 (t0) = 10010000

R9 (t1) = 00000000

R10 (t2) = 00000003

R11 (t3) = 00000000

R12 (t4) = 10010000

R13 (t5) = 00000000

BEFORE Overwrite. We can see that from t0 to t3 has been

initialized. T4 before overwrite, contains the first memory address of num[]. T5 is 0 because we are still at the first index.

R8 (t0) = 10010000

R9 (t1) = 00000000

R10 (t2) = 00000003

R11 (t3) = 00000000

R12 (t4) = ffffffff

R13 (t5) = 00000000

AFTER Overwrite. We have now accessed what's inside 10010000, it

is -1, from num[]. Now we can proceed and calculate the rest. iteration.

1st Iteration

R8 (t0) = 10010000

R9 (t1) = ffffffff

R10 (t2) = 00000003

R11 (t3) = 00000001

R12 (t4) = ffffffff

R13 (t5) = 00000000

The first iteration, we calculated total as -1, because total was 0 and the first value in the array is -1. So 0 + -1 = -1. Represented as ffffffff.

2nd Iteration

```
R8 (t0) = 10010000
R9 (t1) = ffffffff
R10 (t2) = 00000003
R11 (t3) = 00000002
R12 (t4) = ffffffff
R13 (t5) = 00000004
```

Total is now -3, because we calculated total, which was -1, with the 2nd value in the array, which was -2. So $-1 + -2 = -3$. Represented as fffffffd. Also notice how t3 successfully incremented by 1, and t5 is incremented by 4 which is the offset.

3rd Iteration (Final Iteration)

```
R8 (t0) = 10010000
R9 (t1) = fffffffa
R10 (t2) = 00000003
R11 (t3) = 00000003
R12 (t4) = fffffffd
R13 (t5) = 00000008
```

Total is now -6. Total previously was -3, and now we use the 3rd value in the array, -3, to add total with. So $-3 + -3 = -6$. Represented as fffffffa. Because t2 now matches with t3, which is 3, we move on and execute the exit branch to exit the program.

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

\$t0 - holds the entire num array

\$t1 - holds total

\$t2 - holds N (number of indexes)

\$t3 - hold i (current index)

\$t4 - holds memory address of current index, then allows us to access value in that memory address's index

\$t5 - holds calculated offset value

The interaction between register and memory space

The la (load address) instruction loads the address of the num array into register \$t0. Then, using the index i stored in \$t3, the program calculates the correct byte offset by shifting i left by two bits (sll \$t5, \$t3, 2), which effectively multiplies i by 4 to account for the size of each integer in the array. This offset is added to the base address (\$t0) of the array to compute the memory address of the current element (add \$t4, \$t5, \$t0). The lw (load word) instruction retrieves the value at that address into \$t4. The sum of these values is then added to the total variable, which is stored in \$t1. After each iteration, \$t3 is incremented to move to the next index, and the program continues until the loop condition is met which is $i \neq N$.

Program logic

This program implements a loop that calculates the sum of an array of integers. It initializes the array num with values -1, -2, and -3 and the total variable to 0. Using a do-while loop structure, the program continuously adds the value at num[i] to total, where i starts at 0 and increments by

1 in each iteration. The loop terminates when i equals the total number of elements in the array ($N = 3$). Each iteration calculates the correct memory address for the current element using the index i , and the program adds the element's value to the cumulative sum stored in `total`. The program ends with a system call to exit.

EXTRA

I realized I didn't save `total`'s value anywhere for convenience of seeing what the previous sums were. Maybe it was required maybe it's not. But I implemented an additional array called "save" which does exactly this. So now the Data Segment showcases the previous total calculations.

User Data Segment

```
[10010000] ffffffff fffffffe fffffffd 00000000 .....
[10010010] ffffffff fffffffd fffffffa 00000000 .....
```

Here's how the code looks now with this additional change.

```
##### Data segment #####
.data
    num:        .word -1, -2, -3    # Array of integers to be summed
    total:      .word 0             # Variable to store the sum of the array
    save:       .space 12           # Allocated space to showcase previous total calculations
##### Code segment #####
.text
main:
    # do
    # total += num[i]    total is equal to the sum of total and num[i]
    # i += 1            increment i (array index) by 1.
    # while ( i != N)   check if i is not equal to N (number of indexes)

    la $t0, num         # Load the address of the array num into $t0
    lw $t1, total        # Load total into $t1(0)
    li $t2, 3           # N = 3
    li $t3, 0           # i = 0
    li $t5, 0           # initialize offset at 0
    la $t6, save         # loads address of empty save array

loop:
    sll $t5, $t3, 2      # multiply $t3(i(0)) by 2^2(4) to calculate byte offset $t5(0)
    add $t4, $t5, $t0    # add offset ($t5(0)) with num address ($t0(10010000)) in order to compute memory address. $t4(10010000)
    lw $t4, 0($t4)       # overwrite $t4 with the value located at $t4's current calculated index.
    add $t1, $t1, $t4     # total += num[i], adds ($t1(0)) and ($t4(-1))
    sw $t1, 0($t6)       # save total into first save array
    addi $t3, $t3, 1     # i += 1 (i = 1)
    addi $t6, $t6, 4     # increment memory address of save (t6).
    bne $t3, $t2, loop   # branch off to loop, if $t3(1) does not equal $t2(3)
    j exit
```