Mohammad Iqbal

CSC 210

11/06/2024

**MIPS assembly program that initializes an array of 1000 elements in the data segment and clears all elements using index arithmetic in the main function:**

.data

array: .space 4000 # Reserve 4000 bytes (1000 words, 4 bytes each)

.text

.globl main

main:

# Syscall to get the start time in milliseconds

li $v0, 30 # 30 is the syscall code for time in milliseconds

syscall

move $t0, $v0 # Store start time in $t0

# Clear the array

li $t1, 0 # $t1 = index

li $t2, 0 # $t2 = value to store (0)

clear\_loop:

bge $t1, 1000, end\_clear # if index >= 1000, exit the loop

sll $t3, $t1, 2 # $t3 = index \* 4 (for byte offset)

add $t4, $t3, $t4 # Calculate array address: array + (index \* 4)

sw $t2, array($t3) # Set array[index] = 0

addi $t1, $t1, 1 # index++

j clear\_loop # Jump back to the beginning of the loop

end\_clear:

# Syscall to get the end time in milliseconds

li $v0, 30

syscall

move $t5, $v0 # Store end time in $t5

# Calculate runtime

sub $t6, $t5, $t0 # runtime = end\_time - start\_time

# Print runtime

li $v0, 1 # 1 is the syscall code to print integer

move $a0, $t6 # Move runtime to $a0 for printing

syscall

# Exit

li $v0, 10 # 10 is the syscall code to exit

syscall

**Explanation**

* **Array Initialization:** The .space directive reserves 4000 bytes for an array of 1000 words.
* **Runtime Measurement:** Using syscall 30, we capture the time in milliseconds before and after clearing the array.
* **Clearing the Array:** The loop iterates over each index, calculates the byte offset, and sets each element to zero using the sw instruction.
* **Runtime Output:** The difference between the end and start times is printed, giving the total time taken for clearing the array in milliseconds.

**MIPS assembly using pointer arithmetic instead of index arithmetic:**

.data

array: .space 4000 # Reserve 4000 bytes (1000 words, 4 bytes each)

.text

.globl main

main:

# Syscall to get the start time in milliseconds

li $v0, 30 # 30 is the syscall code for time in milliseconds

syscall

move $t0, $v0 # Store start time in $t0

# Initialize pointers

la $t1, array # $t1 points to the start of the array

li $t2, 0 # $t2 = value to store (0)

la $t3, array + 4000 # $t3 points to the end of the array (array + 4000 bytes)

clear\_loop:

bge $t1, $t3, end\_clear # if pointer >= end of array, exit the loop

sw $t2, 0($t1) # Set \*($t1) = 0

addi $t1, $t1, 4 # Move pointer to the next word (4 bytes)

j clear\_loop # Jump back to the beginning of the loop

end\_clear:

# Syscall to get the end time in milliseconds

li $v0, 30

syscall

move $t5, $v0 # Store end time in $t5

# Calculate runtime

sub $t6, $t5, $t0 # runtime = end\_time - start\_time

# Print runtime

li $v0, 1 # 1 is the syscall code to print integer

move $a0, $t6 # Move runtime to $a0 for printing

syscall

# Exit

li $v0, 10 # 10 is the syscall code to exit

syscall

**Explanation**

* **Pointer Initialization:** $t1 is initialized to the start of array, and $t3 is set to the end of the array by adding 4000 bytes to the starting address.
* **Pointer Arithmetic for Clearing:** The loop continues until $t1 reaches $t3, clearing each word by setting it to zero. After each sw operation, $t1 is incremented by 4 to point to the next word.
* **Runtime Measurement and Output:** The runtime is calculated the same way as before, using the difference between the start and end times, and it’s printed with syscall 1.

**Difference in Runtimes:**

|  |  |  |
| --- | --- | --- |
| Size | Index | Pointer |
| 0 | 0 | 0 |
| 2000 | 8 | 11 |
| 4000 | 30 | 12 |
| 6000 | 22 | 25 |
| 8000 | 30 | 15 |
| 10000 | 45 | 29 |
| 12000 | 38 | 39 |
| 14000 | 53 | 38 |
| 16000 | 60 | 47 |
| 18000 | 73 | 47 |
| 20000 | 64 | 52 |