Arrays

Bello Melido

10/15/2023

CSC 210

C code

```
clear1(int array[], int size) {
  int i;
  for (i = 0; i < size; i += 1)
    array[i] = 0;
}
clear2(int *array, int size) {
  int *p;
  for (p = &array[0]; p < &array[size];
       p = p + 1)
    *p = 0;
}
```

clear1 and clear2 are two functions in C programming that do the same job: they take an array of numbers and change all the numbers in that array to zero. However, they use different methods to do this.

To translate the provided C code into MIPS assembly code for use in MARS:

- Create a .text section and define your functions using labels, such as clear1 and clear2.
- Implement loops using branch instructions (beq, bne) for the C for loops, and use load (lw) and store (sw) instructions to access memory for array elements.
- Make sure to set up the stack frame properly and use the $ra register for function calls and returns, and use system calls (syscall) to print the resulting arrays in the main function.

Difference between two codes

Both code snippets aim to initialize an array with values [1, 2, 3], clear the array, and then print the cleared array. However, they differ in the approach they take to clear the array:

1. The first snippet uses a function named `clear2`. This function clears the array by directly manipulating memory addresses. It calculates the end address of the array and iteratively zeroes out each element using address arithmetic until the entire array is cleared.

2. The second snippet uses a function named `clear1`. This function clears the array using a more traditional loop-based approach. It iteratively zeroes out each element of the array by its index, using the index to calculate the appropriate memory address for each element.

3. Other than the method of clearing the array, the rest of the logic in both snippets (initializing the array and printing it) is identical.

Result for both of them

This MIPS assembly code initializes an array containing the values 1, 2, and 3. It then calls the clear1 function to set all elements of the array to 0 and uses the print_array function to print the modified array, separated by spaces and followed by a newline character. The program concludes by exiting gracefully.

At the beginning you can see the values 1,2,3 in the data segment Address 0x1001000,0x1001004,0x1001008

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1	2	3	3	655392	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0

Than, after calling the clear1 function it makes all the values 0 and prints them with the print function which I called print_loop.

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	3	655392	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Mars Messages Run I/O

```
0 0 0
```

Clear 1

```
.data
    # Define an array and its size
    array:  .space 12    # 3 integers, 3 * 4 bytes
    size:   .word 3      # Array size
    space:  .asciiz " "
    newline: .asciiz "\n"

.text
main:
    # Initialize an array: [1, 2, 3]
    li $t0, 1
    sw $t0, array
    li $t0, 2
    sw $t0, array + 4
    li $t0, 3
    sw $t0, array + 8



    # Call clear1
    la $a0, array   # Load array address into $a0
    lw $a1, size    # Load size into $a1
    jal clear1      # Call clear1

    # Print the resulting array
    la $a0, array
    lw $a1, size
    jal print_array

    # Exit the program
    li $v0, 10
    syscall

clear1:
    li $t0, 0  # i = 0

loop_clear1:
    bge $t0, $a1, end_clear1
```

Line: 14 Column: 19 ☑ Show Line Numbers

```
    sll $t1, $t0, 2
    add $t2, $a0, $t1
    sw $zero, 0($t2)
    addi $t0, $t0, 1
    j loop_clear1

end_clear1:
    jr $ra

print_array:
    li $t0, 0  # i = 0

print_loop:
    bge $t0, $a1, end_print
    sll $t1, $t0, 2
    add $t2, $a0, $t1
    lw $t3, 0($t2)

    # Print integer
    li $v0, 1
    move $a0, $t3
    syscall

    # Print a space for separation
    li $v0, 4
    la $a0, space
    syscall

    addi $t0, $t0, 1
    j print_loop

end_print:
    # Print newline
    li $v0, 4
    la $a0, newline
    syscall

    jr $ra
```

Clear 2

```
.data
    # Define an array and its size
    array:  .space 12    # 3 integers, 3 * 4 bytes
    size:   .word 3      # Array size
    space:  .asciiz " "
    newline: .asciiz "\n"

.text
main:
    # Initialize an array: [1, 2, 3]
    li $t0, 1
    sw $t0, array
    li $t0, 2
    sw $t0, array + 4
    li $t0, 3
    sw $t0, array + 8

    # Call clear2
    la $a0, array   # Load array address into $a0
    lw $a1, size    # Load size into $a1
    jal clear2      # Call clear2

    # Print the resulting array
    la $a0, array
    lw $a1, size
    jal print_array

    # Exit the program
    li $v0, 10
    syscall

clear2:
    li $t0, 0  # p = 0
    add $t1, $a0, $zero

    # Calculate the end address of the array
    sll $t4, $a1, 2  # Multiply size by 4 to get byte offset
    add $t5, $a0, $t4  # Add offset to base address to get end address
```

Line: 78 Column: 1 ☑ Show Line Numbers

```
loop_clear2:
    bge $t1, $t5, end_clear2  # Compare with the end address
    sw $zero, 0($t1)
    addi $t1, $t1, 4
    j loop_clear2

end_clear2:
    jr $ra

print_array:
    li $t0, 0  # i = 0

print_loop:
    bge $t0, $a1, end_print
    sll $t1, $t0, 2
    add $t2, $a0, $t1
    lw $t3, 0($t2)

    # Print integer
    li $v0, 1
    move $a0, $t3
    syscall

    # Print a space for separation
    li $v0, 4
    la $a0, space
    syscall

    addi $t0, $t0, 1
    j print_loop

end_print:
    # Print newline
    li $v0, 4
    la $a0, newline
    syscall

    jr $ra
```

Line: 78 Column: 1 ☑ Show Line Numbers