

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:

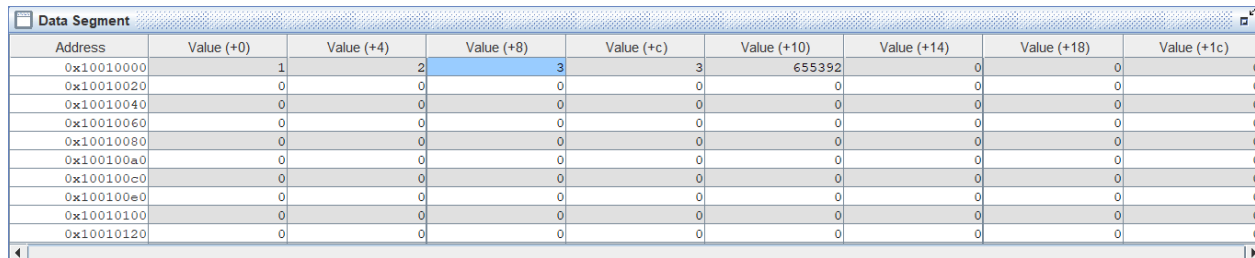
1. The first snippet uses a function named `clear2`. This function clears the array by directly manipulating memory addresses. It calculates the end address of the array and iteratively zeroes out each element using address arithmetic until the entire array is cleared.
2. The second snippet uses a function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

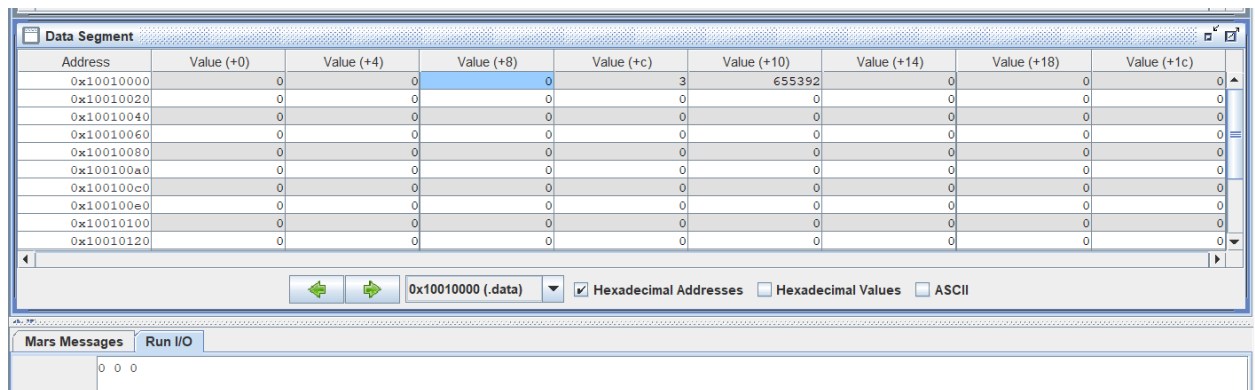
This MIPS assembly code initializes an array containing the values 1, 2, and 3. It then calls the `clear1` function to set all elements of the array to 0 and uses the `print_array` function to print the modified array, separated by spaces and followed by a newline character. The program concludes by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

At the beginning you can see the values 1,2,3 in the data segment Address 0x10010000,0x10010004,0x10010008



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

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



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) [X] Hexadecimal Addresses [ ] Hexadecimal Values [ ] ASCII

Mars Messages Run I/O

0 0 0

# Clear 1

```
1 .data
2     # Define an array and its size
3     array: .space 12    # 3 integers, 3 * 4 bytes
4     size:  .word 3      # Array size
5     space:  .asciiz " "
6     newline: .asciiz "\n"
7
8 .text
9 main:
10     # Initialize an array: [1, 2, 3]
11     li $t0, 1
12     sw $t0, array
13     li $t0, 2
14     sw $t0, array + 4
15     li $t0, 3
16     sw $t0, array + 8
17
18
19
20     # Call clearl
21     la $a0, array    # Load array address into $a0
22     lw $a1, size     # Load size into $a1
23     jal clearl       # Call clearl
24
25     # Print the resulting array
26     la $a0, array
27     lw $a1, size
28     jal print_array
29
30     # Exit the program
31     li $v0, 10
32     syscall
33
34 clearl:
35     li $t0, 0    # i = 0
36
37 loop_clearl:
38     bge $t0, $a1, end_clearl
39     li $t1, $a0
40     sw $t0, ($t1)
41     addi $t0, $t0, 1
42     j loop_clearl
43
44 end_clearl:
45     jr $ra
```

Line: 14 Column: 19 ☒ Show Line Numbers

```

39     sll $t1, $t0, 2
40     add $t2, $a0, $t1
41     sw $zero, 0($t2)
42     addi $t0, $t0, 1
43     j loop_clear1
44
45 end_clear1:
46     jr $ra
47
48 print_array:
49     li $t0, 0 # i = 0
50
51 print_loop:
52     bge $t0, $a1, end_print
53     sll $t1, $t0, 2
54     add $t2, $a0, $t1
55     lw $t3, 0($t2)
56
57     # Print integer
58     li $v0, 1
59     move $a0, $t3
60     syscall
61
62     # Print a space for separation
63     li $v0, 4
64     la $a0, space
65     syscall
66
67     addi $t0, $t0, 1
68     j print_loop
69
70 end_print:
71     # Print newline
72     li $v0, 4
73     la $a0, newline
74     syscall
75
76     jr $ra

```

# Clear 2

```
1 .data
2     # Define an array and its size
3     array: .space 12    # 3 integers, 3 * 4 bytes
4     size:  .word 3      # Array size
5     space:  .asciiz " "
6     newline: .asciiz "\n"
7
8 .text
9 main:
10     # Initialize an array: [1, 2, 3]
11     li $t0, 1
12     sw $t0, array
13     li $t0, 2
14     sw $t0, array + 4
15     li $t0, 3
16     sw $t0, array + 8
17
18     # Call clear2
19     la $a0, array    # Load array address into $a0
20     lw $a1, size      # Load size into $a1
21     jal clear2        # Call clear2
22
23     # Print the resulting array
24     la $a0, array
25     lw $a1, size
26     jal print_array
27
28     # Exit the program
29     li $v0, 10
30     syscall
31
32 clear2:
33     li $t0, 0    # p = 0
34     add $t1, $a0, $zero
35
36     # Calculate the end address of the array
37     sll $t4, $a1, 2    # Multiply size by 4 to get byte offset
38     add $t5, $a0, $t4  # Add offset to base address to get end address
```

Line: 78 Column: 1 ☒ Show Line Numbers



```

40 loop_clear2:
41     bge $t1, $t5, end_clear2 # Compare with the end address
42     sw $zero, 0($t1)
43     addi $t1, $t1, 4
44     j loop_clear2
45
46 end_clear2:
47     jr $ra
48
49 print_array:
50     li $t0, 0 # i = 0
51
52 print_loop:
53     bge $t0, $a1, end_print
54     sll $t1, $t0, 2
55     add $t2, $a0, $t1
56     lw $t3, 0($t2)
57
58     # Print integer
59     li $v0, 1
60     move $a0, $t3
61     syscall
62
63     # Print a space for separation
64     li $v0, 4
65     la $a0, space
66     syscall
67
68     addi $t0, $t0, 1
69     j print_loop
70
71 end_print:
72     # Print newline
73     li $v0, 4
74     la $a0, newline
75     syscall
76
77     jr $ra

```

Line: 78 Column: 1 ☒ Show Line Numbers