Benchmark: “Scalar Multiplication Using Addition”

Author: Abdalrahman Alshannaq *Reviewed by: “Abdullah Matar”*

Description & Notes

- Refer to Benchmark “Multiplication Using Addition” to understand the concept of this benchmark
- The benchmark applying Scalar Multiplication (C * Arr[]), while C is int and Arr is array of integers
- Array must be stored in memory (suggested to store on first location and change offset by 0)
- Benchmark Scalability:
  - Changing constant C: change results and time
  - Changing Array content: by changing values in memory
  - Changing Array size: by changing corresponding register and content
- Results are stored on place (same as original memory locations)

Algorithm (Pseudo or C)

```
C ← 2                    // Constant multiplier
size ← 10                // Size of the array
Arr ← [5, 3, 2, 1, 2, 16, 48, 1, 1, 5]  // Array elements

// Outer loop to iterate through the array
for count ← 0 to size - 1 do
    index ← count
    num1 ← Arr[index]  // Load current array element
    Arr[index] ← Mul_Fun(C, num1)  // Call Mul_Fun and store result
end for

// Function to multiply two numbers
function Mul_Fun(num1, num2)
    mulOut ← 0
    for i ← num2 - 1 down to 0 do
        mulOut ← mulOut + num1
    end for
    return mulOut
end function
```

Registers and memory used in implementation

s1($17): array size

s2($18): counter to the array

s3($19): memory index (helping counter to support memory addressing modes)

s4($20): temp register for condition

t8($24): holds num1 value (constant C)

t9($25): holds num2 value (Arr[count])

s7($23) are used as output from Mul_Fun

s6($22) are used as counter in Mul_Fun

Memory: to store array content

Code (.data and .text)

```
.data:
        Arr: .word 0x5, 0x3, 0x2, 0x1, 0x2, 0x10, 0x30, 0x1, 0x1, 0x5

.text:

ADDI $17, $0, 10                # size = 10 (initilize)
ANDI $18, $0, 0                 # count = 0 (initilize)
XORI $24, $0, 2                 # load the constant C = 2

Outer_Loop:
        ADD $19, $18, $0        # index = count
        # this line is for byte addressing memories only
        # SLL $19, $19, 2       # index = index * 4
        LW $25, Arr($19)        # num1 = Arr[count]
        JAL Mul_Fun             # call Mul_Fun (C, Arr[count])
        SW $23, Arr($19)        # Arr[count] = mulOut

        ADDI $18, $18, 1        # count++
        SUB  $20, $18, $17      # temp = count - size (check loop condition)
        BLTZ $20, Outer_Loop
        J Finish

############
Mul_Fun :
        ANDI $23, $0, 0         # mulOut = 0 (initilize)
        ADDI $22, $25, -1       # i = num2-1 (initilize by value of num2-1)

Mul_Loop :
        ADD  $23, $23, $24      # mulOut += num1
        ADDI $22, $22, -1       # i--
        BGEZ $22, Mul_Loop
        JR $31
############

Finish : NOP
```

Expected Output

Memory: contains resulted array on same location

Additional Notes

- Try changing value of C, note results and execution time
- Try changing array content, note results and execution time
- Try scaling size of array and its content, note results and execution time