گزارش‌کار کد سوم معماری

نگار هنرورصدیقیان-99243076

در این کد قصد داریم تا الگوریتم merge sort را به کمک زبان اسمبلی MIPS پیاده سازی کنیم. برای این کار به شیوه ذیل عمل میکنیم:

```
main: # declaration of function main
    la $a0, enter_length # load address of the text you want in $a0
    li $v0, 4 # define what system call you want
    syscall # call the system call
    li $v0, 5 # at first we need to read the length of our array
    move $s0, $v0 # now we move the value from $v0 to $s0 cause we want to use it later
    sll $s1,$s0,2 # we need to left shift our $s0 2 times to $s1 cause each array element occuipies
    # 4 bits
    move $a0,$s1 # we move $s1 value to $a0
    li $v0, 9   # now we allocate heap memory which its size is equal to value stored in $a0
    syscall

    move $s2,$v0 # $s2 points to the array

    li $v0, 9 # heap memory allocation
    syscall

    move $t0,$s2 # now $t0 points to the beginning of our array
    addu $t1,$s2,$s1 # now $t1 points to the end of our array
```

در ابتدا در بخش main به عنوان اولین ورودی تعداد اعضای آرایه را از کاربر میگیریم. سپس مقدار آن را در $s0 ذخیره کرده و سپس 2 بار به چپ شیفت می‌دهیم و مقدار حاصل را در$s2 ذخیره می‌کنیم . این کار به این دلیل است که هر عنصر آرایه 4 خانه از حافظه را اشغال میکند. سپس به کمک دستور شماره 9 پوینتری به این آرایه ساخته می‌شود.در حال حاضر پس از انتقال مقدار $v0 به $s2 ، $s2 به آرایه اشاره می‌کند.مجددا به کمک دستور 9 این بار تعداد خانه حافظه مورد نیاز تخصیص می‌یابد. سپس این پوینتر را به $t0 منتقل کرده و در بخش بعد با انجام عمل محاسبه اشاره‌گری به انتهای آرایه به نام $t1 میسازیم و وارد بخش بعد میشویم.

```
input_reading:

    bgeu $t0, $t1, end_of_input_reading # if we reach the last index of our array we will move to the mentioned function
    li $v0, 5 # we read an integer which will be an element of our array
    syscall
    sw $v0, 0($t0) # we move our element of array to $t0 and store it there
    addi $t0,$t0,4 # now we move $t0 4bits so that at the next round of our loop , the new elemnt would be stored
    # in correct position and to finally reach the end loop condition
    b input_reading # the new element is stored in its right place and we are ready to assign the next array element to
    # input value
end_of_input_reading:
    move $a0,$s5 #$a0 points to the array
    li $a1, 0 #$a1 stores first index of our array
    addi $a2,$s0,-1 # now $a2 stores last inex of our array
    jal quick_sort # now we jump to quick_sort function and link our array to it
    sll $t7,$s0,2 # we left shift $s0 4 bits due to reason mentioned above and store it
    # in $t7,now $t7 stores the length of our array
    add $t7,$t7,$a0 #$a0 was already pointing to our array,by adding that to $t7 and storeing
    # the result in $t7 we made $t7 a pointer to last index of our array
    move $t0,$a0 # we store the $a0 value in $t0 and we make it a pointer to the beginning of
    # our array
```

هدف این تابع خواندن اعضای آرایه و ذخیره‌سازی مقادیر آن‌ها در آرایه است. برای این کار یک حلقه در نظر میگیریم که تا زمانی که اشاره‌گر به ابتدای آرایه مقداری کوچکتر از اشاره‌گر به انتهای آن دارد ورودی از کاربر گرفته و مقادیر را ذخیره میکنیم و سپس مقدار $t0 که اکنون اشاره گر به اولین خانه خالی آرایه می‌باشد را 4 بیت جلو برده و حلقه را تا زمان برقراری شرط پایان ادامه می‌دهیم. با پایان حلقه وارد تابع دوم میشویم . در اینجا با نظر به اینکه می‌دانیم آرایه تکمیل شده آرایه را به تابع quick_sort پاس میدهیم. میدانیم که سازوکار این تابع به گونه‌ایست که با آرایه و دو آرگومان حد پایین (low) و حد بالا (high) کار میکند.$a1 حد پایین و $a2 حد بالا آرایه را در خود نگه میدارند. پس از پاس دادن آرایه و ذخیره‌سازی آرگومان‌های مربوطه در تابع quick_sort داریم:

```
quick_sort:

    addi $sp,$sp, -16 # a stack pointer with 4 bytes space
    sw $a0, 0($sp) # we set $sp to point at our array , $a0
    sw $a1,4($sp) #low
    sw $a2,8($sp) #high
    sw $ra,12($sp) # return address

    move $t0 , $a2 # we set $a2=high to $t0

    slt $t1 ,$a1,$t0 # since $a1=low, if low < high then $t1=1 else its 0 , if low>high means our array is sort
    beq $t1,$zero,end_of_quick_sort # if our array is sorted then we jump to end_Of_quick_sort function

    jal random_partition #jump and link array to random_partition function
    move $s0,$v0 # pivot(pi)

    lw $a1, 4($sp) #a1=low
    addi $a2 ,$s0,-1 #$a2=pi-1
    jal quick_sort

    addi $a1,$s0, 1 #$a1=pi+1
    lw $a2, 8($sp) #$a2=high
    jal quick_sort

end_of_quick_sort:
    lw $a0 ,0($sp) #load word $a0
    lw $a1,4($sp) #load word $a1
    lw $a2,8($sp) #load word $a2
    lw $ra,12($sp) #return address
    addi $sp,$sp,16 #stack
    jr $ra #now the array is sorted
```

در اینجا ما قصد داریم تا در پوینترsp$ مربوط به تابع 4 مقدار 4 بیتی را ذخیره کنیم ، باتوجه به اینکه اشارهگر به استک به بالای آن اشاره دارد لازم است تا آن را 4 بایت به پایین انتقال دهیم و به همین دلیل آن را -16 بیت جابه‌جا کردیم.سپس به وسیله $sp جابها شده 4 مقدار اشاره‌گر به آرایه($a0) ،اشاره‌گر به پایینترین خانه آرایه ($a1)، اشاره گر به بالاترین خانه آرایه ($a2) و در آخرین خانه آرگومان بازگشتی را قرار می‌دهیم.سپس لازم است تا شرطی برای پایان quick sort قراردهیم. اگر مقدار low بزرگتر مساوی high

شود آنگاه $t1=0 شده و در این صورت تابع end_of_quick_sort برای پایان مرتب سازی فراخوانی میشود. در این تابع چهار مقدار ذخیره شده در استک لود شده،$sp 16 بایت به بالا آمده و به محل فراخوانی تابع بازمیگردیم. سپس در ادامه تابع quick_sort تابعrandom_partition فراخوانی میشود. در این تابع به دنبال تولید یک عدد تصادفی بین حدبالا و حد پایین آرایه داده شده هستیم.

```
random_partition:
    addi $sp, $sp, -16 # we consider 4 bytes for $sp(pointer)
    sw $a0, 0($sp) # store word $a0
    sw $a1, 4($sp) # store word $a1 = low
    sw $a2, 8($sp)  # store word $a2=high
    sw $ra, 12($sp) # return address

    li $a0 , 1 # low  a0=1
    li $a1 , 4 # high a1=4
    move $t0 , $a0 # save a0 in t0
    sub $a1,$a1,$a0 # high - low
    li $v0, 42 # this order gives us a random number
    syscall # a random number in range ( 0 , high - low )
    add $a0 , $a0 , $t0 #  a random number in range ( low  , high )


    move $s0, $a1 #store s0 = low
    move $s1 , $ra      # s1 = return address as random number

    move $a0, $a1 # store $a0 = low
    addi $a1, $a2, 0 # store $a1 = high
    sub $a1, $a1, $a0 # store $a1 = high  - low

    div $s1, $a1 # $t0 = random number / (high -low)
    mfhi $a2 # $a2 = random number % (high -low)
    add $a1, $a0 , $a2 # $a1 = right + low

    lw $a0, 0($sp) #load word $a0
    lw $a2, 8($sp)      #load word $a2
```

دستور شماره 42 عددی تصادفی در رنج مورد نظر تولید کرده و در اختیار ما قرار می‌دهد.در الگوریتم کوییک سورت هدف مرتب سازی آرایه بر حسب pivot به کمک مقادیر high وlow به گونه ای است که آرایه پیرامون pivot به دو زیر آرایه به گونه ای تقسیم میشود که زیرآرایه قبل آن همگی کوچکتر از pivotبوده و مقادیر پس از آن همگی بزرگتر از pivot هستند ؛ به این ترتیب و با مرتب سازی زیرآرایه ها در هر بخش به روش بازگشتی در نهایت آرایه اصلی به صورت صعودی مرتب میشود. با توجه به نحوه عملکرد الگوریتم کوییک سورت یکی از مهمترین بخش های الگوریتم فرآیند جابه‌جایی

اعداد یا swap است.

```
swap: #swap method

    addi $sp, $sp, -12 # we consideer 12 bits(3 bytes) for following elemnts
    sw $a0, 0($sp) # Store word $a0
    sw $a1, 4($sp) # Store word $a1
    sw $a2, 8($sp) # store word $a2

    sll $t1, $a1, 2  #$t1 = 4a($a1*4 => because we had 2 left shifts)
    add $t1, $a0, $t1 #$t1 = arr + 4a
    lw $s3, 0($t1) #$t1 = array[a]

    sll $t2, $a2, 2  #$t2 = 4b
    add $t2, $a0, $t2 #$t2 = arr + 4b
    lw $s4, 0($t2) #$s4 = arr[b]

    sw $s4, 0($t1)#arr[a] = arr[b]
    sw $s3, 0($t2) #arr[b] = $t3


    addi $sp, $sp, 12#Restoring the stack size
    jr $ra #jump back (return) to random partition
```

در ابتدا اشارهگر به آرایه،حد بالا و حد پایین(تنها این سه مقدار) را ذخیره سازی میکنیم.

مقدار pivot را در 4 ضرب کرده و با اشارهگر به ابتدای آرایه جمع میکنیم تا pivot مشخص شود.پس از انجام فرآیند swap، مجددا اشارهگر $sp را به حالت اولیه برگرانده و به تابع اصلی برمیگردیم و به سراغ اجرا تابع partition میرویم که سه مقدار $sp، حد بالا و حد پایین به آن پاس داده میشوند.

```
partition:
# we do the partitioning using pi,i and j , their function is the same as what they do in cpp mergesort code

    addi $sp, $sp, -16 # we consider 16 bits(4 bytes) for pointer

    sw $a0, 0($sp) #word store $a0
    sw $a1, 4($sp) #store $a1
    sw $a2, 8($sp) #store $a2
    sw $ra, 12($sp) #store return address

    move $s1, $a1#$s1 = low
    move $s2, $a2#$s2 = high

    sll $t1, $s2, 2 # $t1 = 4*high, because we shifted $s2 to left 2 times
    add $t1, $a0, $t1# $t1 = array + 4*high
    lw $t9, 0($t1)# $t9 = array[high] (pi)
# the i and j,each one,point at one number in algorithm.after reaching the end of array they go back to beginning of our array
    addi $t3, $s1 , -1 #$t3, i=low -1
    move $t4, $s1#$t4, j=low
    addi $t5, $s2, -1  #$t5 = high - 1
```

در این تابع نیز مقادیر را ابتدا ذخیره میکنیم سپس مقادیر I وj را مشابه آن چه در الگوریتم گفته شده محاسبه میکنیم و وارد حلقه پارتیشن بندی کد میشویم. پس از انجام محاسبات در هر مرحله طبق الگوریتم و اجزای حلقه وارد مرحله بعد میشویم:

```
for_Partition:
     slt $t6, $t5, $t4 #$t6=1 if j>high-1, $t6=0 if j<=high-1
     bne $t6, $zero, end_for #if $t6=1 then branch to endfor

     sll $t1, $t4, 2 #$t1 = j*4
     add $t1, $t1, $a0 #$t1 = array + j*4
     lw $t7, 0($t1) #$t7 = array[j]

     slt $t8, $t9, $t7 #pi < array[j] =>$t8=1,else $t8=0
     bne $t8, $zero, If_For #if $t8=1 then branch to If_For
     addi $t3, $t3, 1 #moving i forward(i++)
     move $a1, $t3 #$a1 = i++
     move $a2, $t4 #$a2 = j
     jal swap #swap(array, i, j)
     addi $t4, $t4, 1 #j++

     j for_Partition # jump to for_partition function

If_For:
     addi $t4, $t4, 1 #moving j forward(j++)
     j for_Partition #jump back to for_Partition function

end_for:
     addi $a1, $t3, 1 #$a1 = i + 1
     move $a2, $s2 #$a2 = high
     add $v0, $zero, $a1 #$ v0 = i+1 => return (i + 1);
     jal swap #jump to swap(array, i + 1, high);
     lw $ra, 12($sp) #return address
     addi $sp, $sp, 16  #restore the stack
     jr $ra  #jump back to for_partition function
```

در این مرحله جابه‌جایی های لازم در سه تابع برای مرتب سازی آرایه انجام میشود.زمانی که از مرتب بودن آرایه اطمینان حاصل کردیم توابع فوق پایان یافته ومجددا وارد ادامه تابع random_partition و سپس quick_sort میشود که در این تابع به صورت بازگشتی quick_sort تعریف شده و هربار برای مرتب سازی یکی از دو زیرآرایه صدا میشود. با پایان مرتب سازی در نهایت آرایه مرتب شده مطابق آنچه انتظار میرود در تابع main قرار داشته و آماده چاپ شدن طبق دستور زیر است:

```
print_array:
    bgeu $t0,$t7,end_of_print_array # similar to previous part , we first declare the loop end condition which functions just
    # like the previous part
    lw $a0, 0($t0) #load word:we set contents of $a0 to $t0
    li $v0, 1 # we print the integer which is our first array element
    syscall
    li $v0, 4 # load immediate
    la $a0, next_line # we load next_line address to $a0 so when we issue a system call it will be
    # executed
    syscall
    addi $t0,$t0, 4 # we move to next array element by moving $t0 4 bits forward
    b print_array # we go back to beginning of our loop
end_of_print_array:
# if our loop is finished it means that we have printed all of our sorted array elements and we are done!
    li $v0, 10  # exit the program
    syscall     # make the systemcall
```