

Tutorial4.3

Assignment 3

Lei Wang

lei.wang2@ucalgary.ca

Assignment3

- Sorting One-Dimensional Arrays
- create an ARMv8 assembly language program that implement this algorithm(insertion sort)

```
#define SIZE 50

int main()
{
    int v[size], i, j, temp;

    //initialize array to random positive integers, mod 256
    for(i = 0; i < SIZE; i++){
        v[i] = rand() & 0xFF;
        printf("v[%d]: %d\n", i, v[i]);
    }

    //sort the array using an insertion sort
    for (i = 1; i < SIZE; i++)
    {
        temp = v[i];
        for (j = i; j > 0 && temp < v[j-1]; j--)
        {
            v[j] = v[j-1];
        }
        v[j] = temp;
    }

    //print out the sorted array
    printf("\nSorted array:\n");
    for (i = 0; i < SIZE; i++)
        printf("v[%d]: %d\n", i, v[i]);

    return 0;
}
```

requirements

- Create space on the stack to store all local variables
- Use m4 or assembler equate for all stack variables offsets
- Optimization(loop, addressing mode)
- always read or write memory when using or assigning to the local variable
- name the program *assign3.asm*
- run the program in *gdb*, first display the contents of the array before sorting, and then displaying it again once the sort is complete. use script to capture the *gdb* session

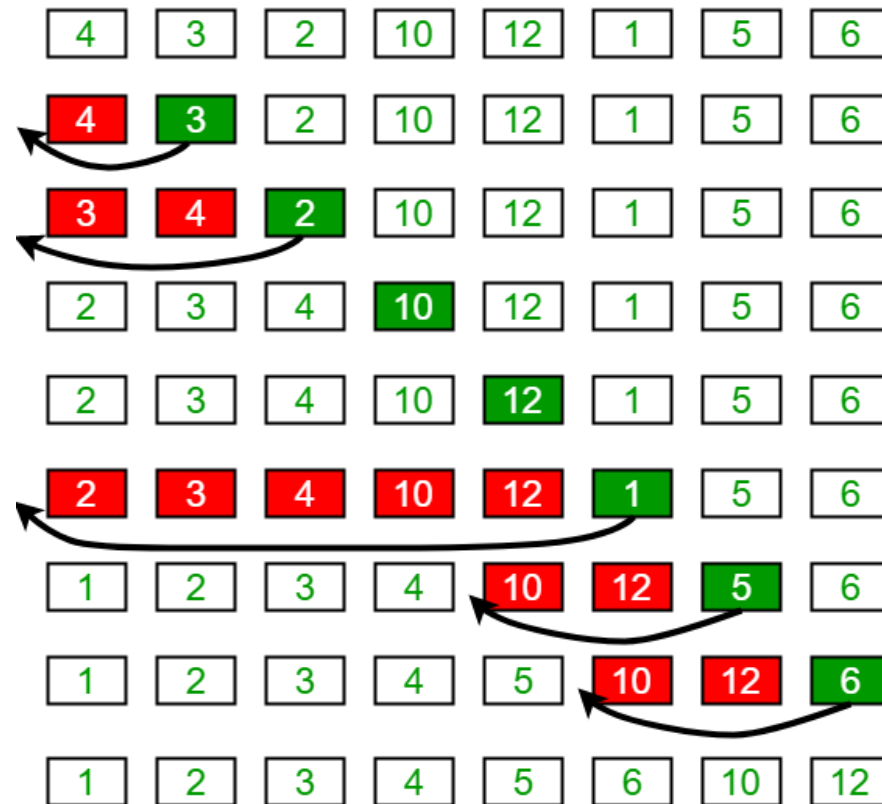
insertion sort

- Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

- Algorithm

- insertionSort(arr, n)
- Loop from $i = 1$ to $n-1$
- Pick element $\text{arr}[i]$
- insert it into sorted sequence $\text{arr}[0 \dots i-1]$

Insertion Sort Execution Example



Pseudo-random

- the program always generates the same sequence of numbers using *rand()*, and such they are not truly random
- because *rand()* always use the algorithm, there are a lot of ways to improve this function(STS)
- this is not a problem for your assignment

macros, equates and register aliases

macros

```
//.asm

//define macros for stack variables and
registers
define(size, 50)
define(array_size, 200)
define(alloc, -224)
define(delloca, 224)
define(fp, x29)
define(lr, x30)
define(offset, x19)
```

equates

```
//.s

//define equates for stack variables
size = 50
array_size = size * 4
alloc = -(16 + array_size) & -16
dealloc = -alloc

//define register aliases
fp      .req    x29
lr      .req    x30
```

show array in gdb

- `x/nd $fp+offset`
- **n** can be the SIZE of the array
- **offset** is the base address of the array

review

- allocate/deallocate memory for variables

```
stp    fp, lr, [sp, alloc]!  
mov    fp, sp  
...  
ldr    fp, lr, [sp], dealloc  
ret
```

- store value. read value, change value

```
mov    Wn, #imm  
str    Wn, [fp, offset]  
ldr    Wn, [fp, offset]  
//change value of Wn  
str    Wn, [fp, offset]
```

- access i^{th} element

```
mov    Wn, #i  
ldr    Ws, [base_address, Wn, SXTW, 2]  
//change value of Ws  
str    Ws, [base_address, Wn, SXTW, 2]
```


loop with two conditions, nested loop

- $j > 0$ AND $\text{temp} < v[j-1] \rightarrow j \leq 0$ OR $\text{temp} \geq v[j-1]$ (then jump out)
- write loop2 inside loop1

```
//sort the array using an insertion sort
```

```
for (i = 0; i < SIZE; i++)  
{
```

```
    temp = v[i];
```

```
    for (j = i; j > 0 && temp < v[j-1]; j--)
```

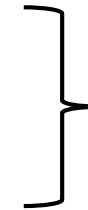
```
    {
```

```
        v[j] = v[j-1];
```

```
    }
```

```
    v[j] = temp;
```

```
}
```



loop2

loop1

endloop2

in assembly

```
loop1:      //set i = 0 before loop1
            //...
            //loop1 body
            //...

            //set j = i before loop2
loop2:      //test in the beginning
            //if j <= 0 branch to endloop2
            //if temp >= v[j-1] also branch to endloop2
            //...
            //loop body2
            //...
            //branch to loop2

endloop2:   //v[j] = v[j-1]

test1:      // i < SIZE?
            // branch to loop1 if the condition meets
```

work period

• Functionality		
Loop to initialize array	2	_____
Display of unsorted array	2	_____
Outer loop of sort	2	_____
Inner loop of sort	2	_____
Comparison of array elements	2	_____
Exchange of array elements	2	_____
Display of sorted array	2	_____
• Use of macros/equates for stack variable offsets	4	_____
• Optimization	4	_____
• Script showing <i>gdb</i> session	2	_____
• Complete documentation and commenting	4	_____
• Design quality	2	_____