

## 1. Quick Sort procedure

### 1.1.Partition

---

**Algorithm Partition**

---

```
1:  Input: Array  $A[l...r]$  of orderable elements
2:  Output: A pivot index
3:
4:   $p \leftarrow A[r]$ 
5:   $i \leftarrow l - 1$ 
6:
7:  repeat from  $j \leftarrow l$  to  $r - 1$ 
8:      if  $A[j] < p$  then
9:           $i \leftarrow i + 1$ 
10:         swap( $A[i]$ ,  $A[j]$ )
11:
12: swap( $A[i + 1]$ ,  $A[r]$ )
13: return  $i + 1$  ▷ Called  $s$  in QuickSort
```

---

### 1.2.QuickSort

---

**Algorithm QuickSort**

---

```
1:  Input: Array  $A[l...r]$  of orderable elements
2:  Output: A partition of  $A[l...r]$  in non-decreasing order
3:
4:  if  $l < r$  then
5:       $s \leftarrow \text{Partition}(A[l...r])$ 
6:      Quicksort( $A[l...s - 1]$ )
7:      Quicksort( $A[s + 1...r]$ )
```

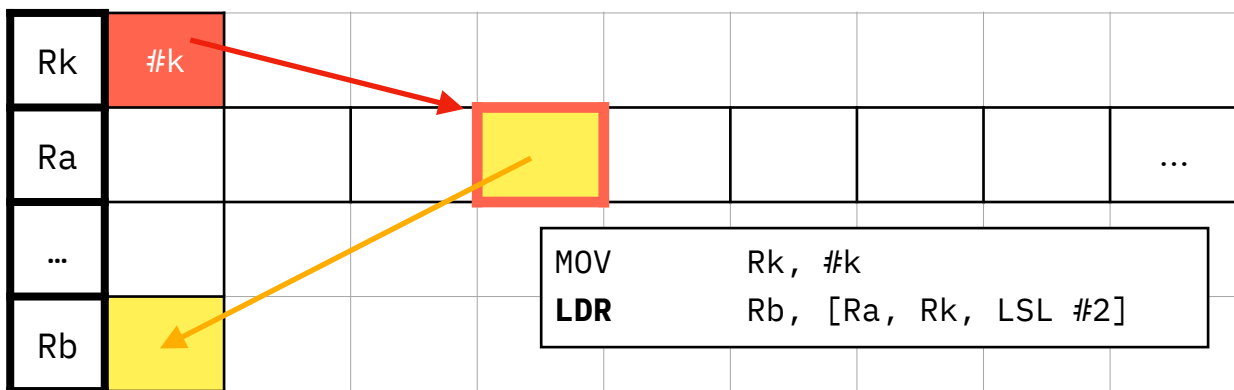
---

## 2. ARM Assembly

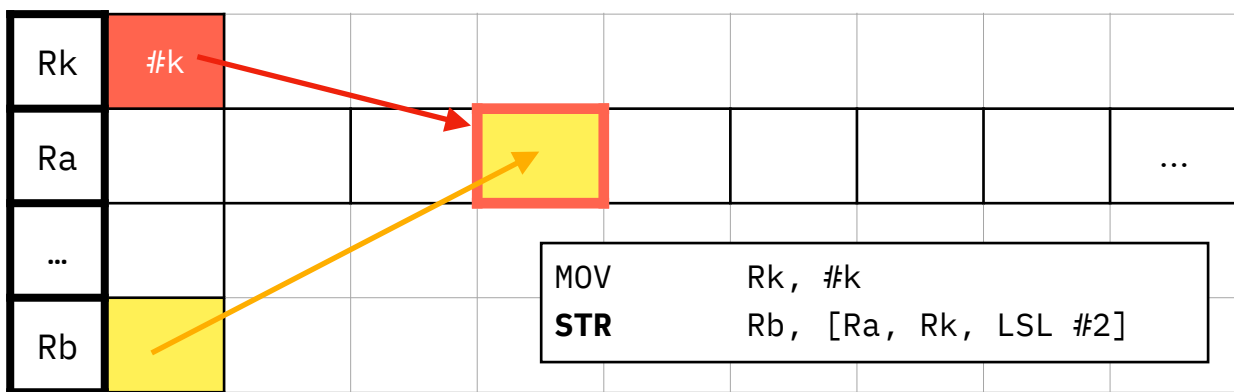
### 2.1.Register, Use all name from 1.)

		$l$		$i$					
Array $A[l...r]$	R0								...
$l$	R1								
$r$	R2								
$p$	R3								
$A[i]$	R4								
$A[j]$	R5								
$A[r]$	R6								
	R7								
	R8								
Bottom stack R10	R9								
Parameter stack	R10	$r$	$l$	$r$	$l$	$r$	$l$	$i$	
$i$	R11								
$j$	R12								
	R13								
	LR								
	PC								

## 2.2. Load $k^{\text{th}}$ index in Array of register Ra, store in Rb

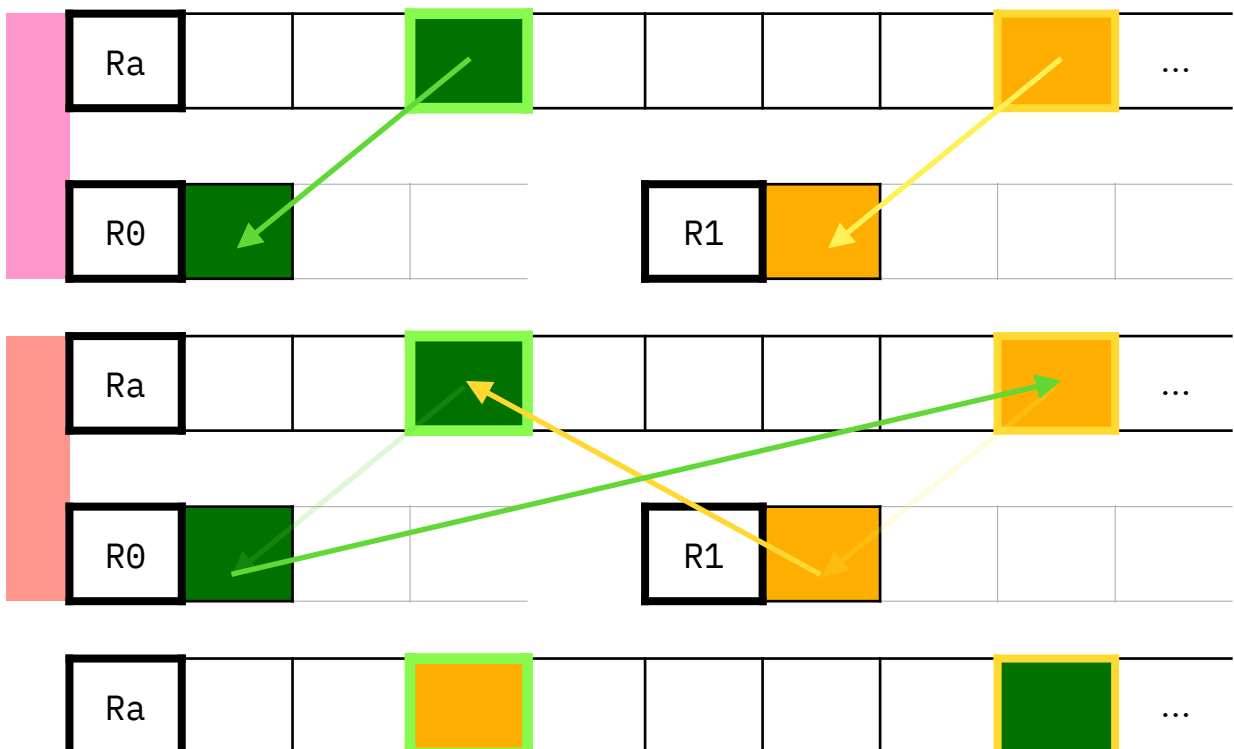


## 2.3. Store address in Rb to $k^{\text{th}}$ index in Array of register Ra

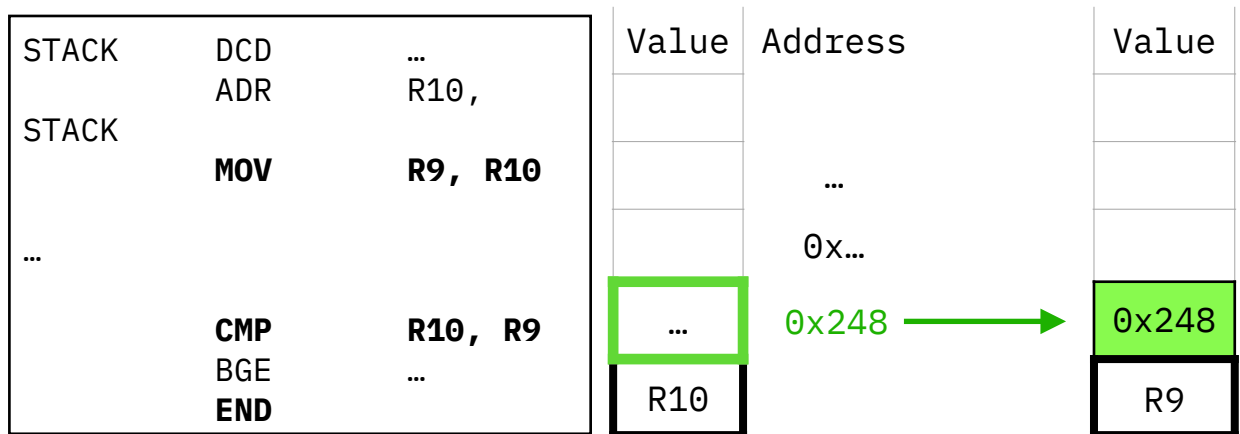


## 2.4. Swap index in Ra

MOV > LDR > STR



## 2.5.Termination



The address of register is changed due to decreasing of its parameters; however, it needs to be over the initial address. By means of comparing R9 with R10 to check the address, if the BGE is true, it continues performing quick sort with the remaining parameters in R10. Otherwise it ends because the stack is empty.

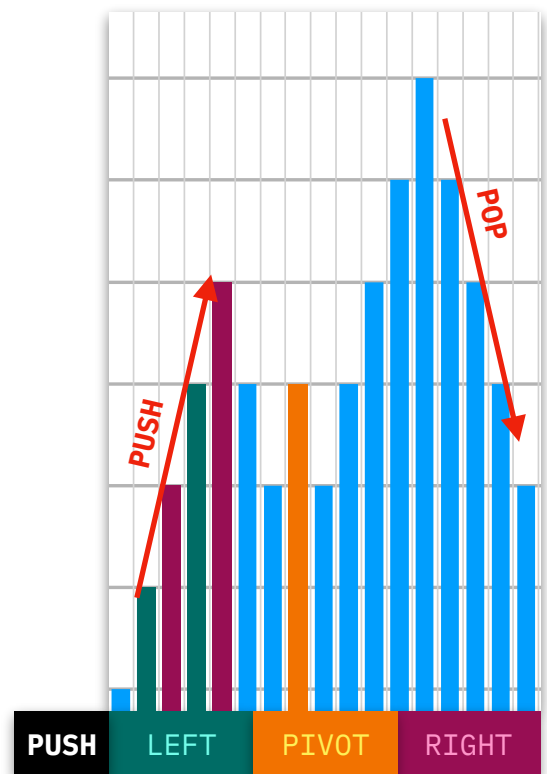
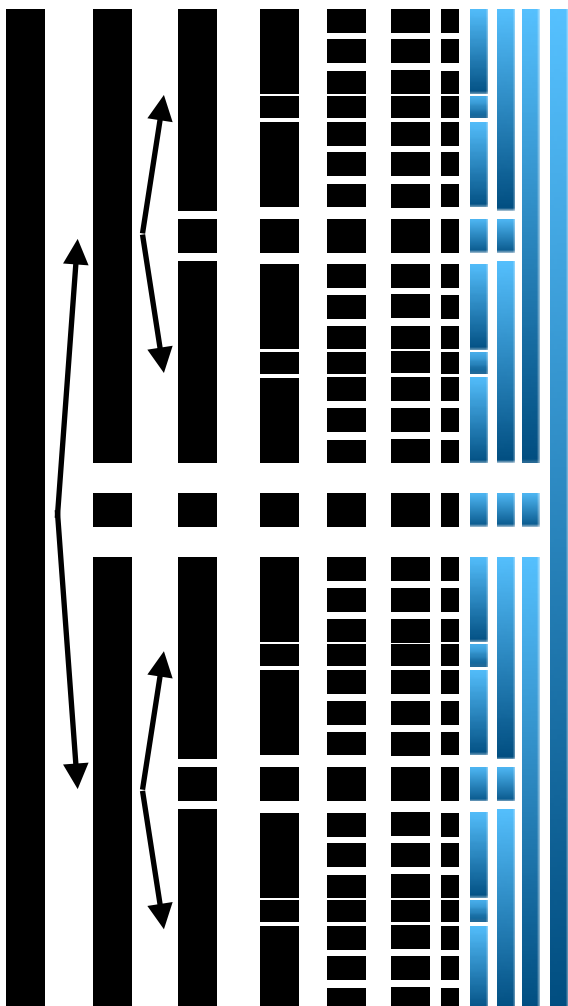
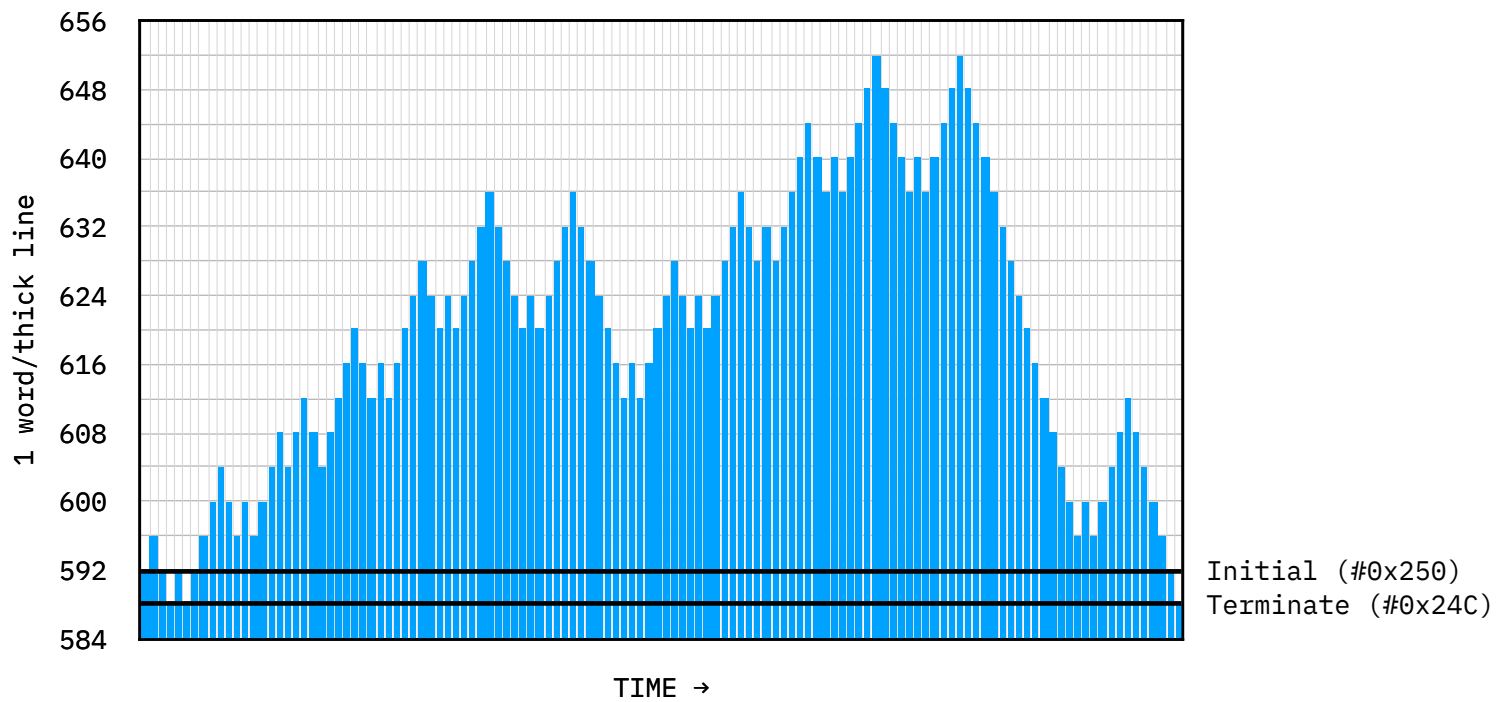
R9	0x250	R10 = 0x248 (< initial), program end immediately.
R10	0x248	

Click on a line number to restore program to state at that line number.

Line Number	Value
50	0x26C
22	0x268
50	0x260
22	0x25C
50	0x254
22	0x250
50	0x248

- Example from 3.) Test set

■ STACK, Base10



### 3. Result

ARRAY: 85, 22, 79, 20, 1, 28, 45, 20, 6, 19, 94, 62, 38, 8, 50, 5, 13, 49, 57, 93 (N = 20)

The image displays three sequential screenshots of a software window titled "View Memory Contents". Each window shows a table of memory data with columns for Word Address, Byte 3, Byte 2, Byte 1, Byte 0, and Word Value. The Start address is 0x200 and the End address is 0x1200. The Word Value Format is set to Dec, and the Memory Map Key is set to Instructions.

**Window 1 (Top):** Shows memory data from address 0x200 to 0x220. The Word Values are 1, 5, 6, 8, 13, 19, 20, 20, and 22.

Word Address	Byte 3	Byte 2	Byte 1	Byte 0	Word Value
0x200	0x0	0x0	0x0	0x1	1
0x204	0x0	0x0	0x0	0x5	5
0x208	0x0	0x0	0x0	0x6	6
0x20C	0x0	0x0	0x0	0x8	8
0x210	0x0	0x0	0x0	0xD	13
0x214	0x0	0x0	0x0	0x13	19
0x218	0x0	0x0	0x0	0x14	20
0x21C	0x0	0x0	0x0	0x14	20
0x220	0x0	0x0	0x0	0x16	22

**Window 2 (Middle):** Shows memory data from address 0x224 to 0x244. The Word Values are 28, 38, 45, 49, 50, 57, 62, 79, and 85.

Word Address	Byte 3	Byte 2	Byte 1	Byte 0	Word Value
0x224	0x0	0x0	0x0	0x1C	28
0x228	0x0	0x0	0x0	0x26	38
0x22C	0x0	0x0	0x0	0x2D	45
0x230	0x0	0x0	0x0	0x31	49
0x234	0x0	0x0	0x0	0x32	50
0x238	0x0	0x0	0x0	0x39	57
0x23C	0x0	0x0	0x0	0x3E	62
0x240	0x0	0x0	0x0	0x4F	79
0x244	0x0	0x0	0x0	0x55	85

**Window 3 (Bottom):** Shows memory data from address 0x22C to 0x250. The Word Values are 45, 49, 50, 57, 62, 79, 85, 93, 94, and 10.

Word Address	Byte 3	Byte 2	Byte 1	Byte 0	Word Value
0x22C	0x0	0x0	0x0	0x2D	45
0x230	0x0	0x0	0x0	0x31	49
0x234	0x0	0x0	0x0	0x32	50
0x238	0x0	0x0	0x0	0x39	57
0x23C	0x0	0x0	0x0	0x3E	62
0x240	0x0	0x0	0x0	0x4F	79
0x244	0x0	0x0	0x0	0x55	85
0x248	0x0	0x0	0x0	0x5D	93
0x24C	0x0	0x0	0x0	0x5E	94
0x250	0x0	0x0	0x0	0x10	10

## Appendix; All instructions used

1. DCD: Define Constant Data เป็น instruction ประเภทหนึ่งของ Data Reservation Directives ที่ใช้สำหรับป้อนค่าคงที่ ใช้เมื่อต้องการป้อนข้อมูลแบบ fixed data เข้าไปใน memory โดยคำสั่งนี้สามารถจัดสรร memory ที่มี 1 word ขึ้นไป โดยมีไม่เกิน 4 bytes
2. ADR: ใช้สำหรับโหลด program-relative หรือ register-relative address เข้าที่ register โดยมี syntax คือ *ADR register, expression*
3. ADD: ใช้ในการบวกค่าของ 2 operand หลัง โดยจะเก็บผลลัพธ์ไว้ที่ operand ตัวแรก, SUB: ใช้ลบในทำนองเดียวกัน
4. CMP: ใช้ในการเปรียบเทียบระหว่างสอง operand โดยการนำค่าใน register มาลบกัน
5. B: ย่อมาจาก branch ซึ่งเป็น unconditional branch กล่าวคือจะไปทำงานที่ label นั้น ๆ ต่อโดยไม่ต้องมีเงื่อนไขใด ๆ
6. BGE: Branch Greater Than Equal, BLT: Branch Less Than
7. MOV: ใช้ copy ข้อมูลจาก operand ที่สอง ซึ่งสามารถเป็นได้ทั้ง register, memory และ constant value ไปที่ operand ตัวแรก ซึ่งอาจเป็น register หรือ memory
8. STR: ใช้เก็บค่าของ register เข้าไปใน memory
9. LDR: ใช้ในการโหลด address หรือ 32-bit immediate data จาก memory เข้าไปที่ register
10. LSL: ย่อมาจาก Logical Shift Left จะนำค่าของ register ไปคูณกับเลขยกกำลังฐานสอง เช่น STR R3, [R0, R12, LSL #2] หมายความว่าเก็บค่า R3 ที่ address ที่มีค่าเท่ากับผลรวมของ R0 กับ 4 เท่าของ R12 ( $LSL \#2 = 2^2$ )

### ความแตกต่างระหว่าง LDR และ STR

*LDR Ra, [Rb]* ค่าใน [address] ของ Rb จะถูกโหลดไปที่ register Ra  
*STR Ra, [Rb]* ค่าที่อยู่ใน register Ra จะถูกเก็บที่ [address] ของ Rb