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 ← A[r]
5:  i ← l – 1
6:
7:  repeat from  j ← l  to r – 1
8:          if A[j] < p then
9:                  i ← i + 1
10:                 swap(A[i], A[j])
11:
12: swap(A[i + 1 ], A[r])
13: return i + 1                                  ▷ Called s in QuickSort
```

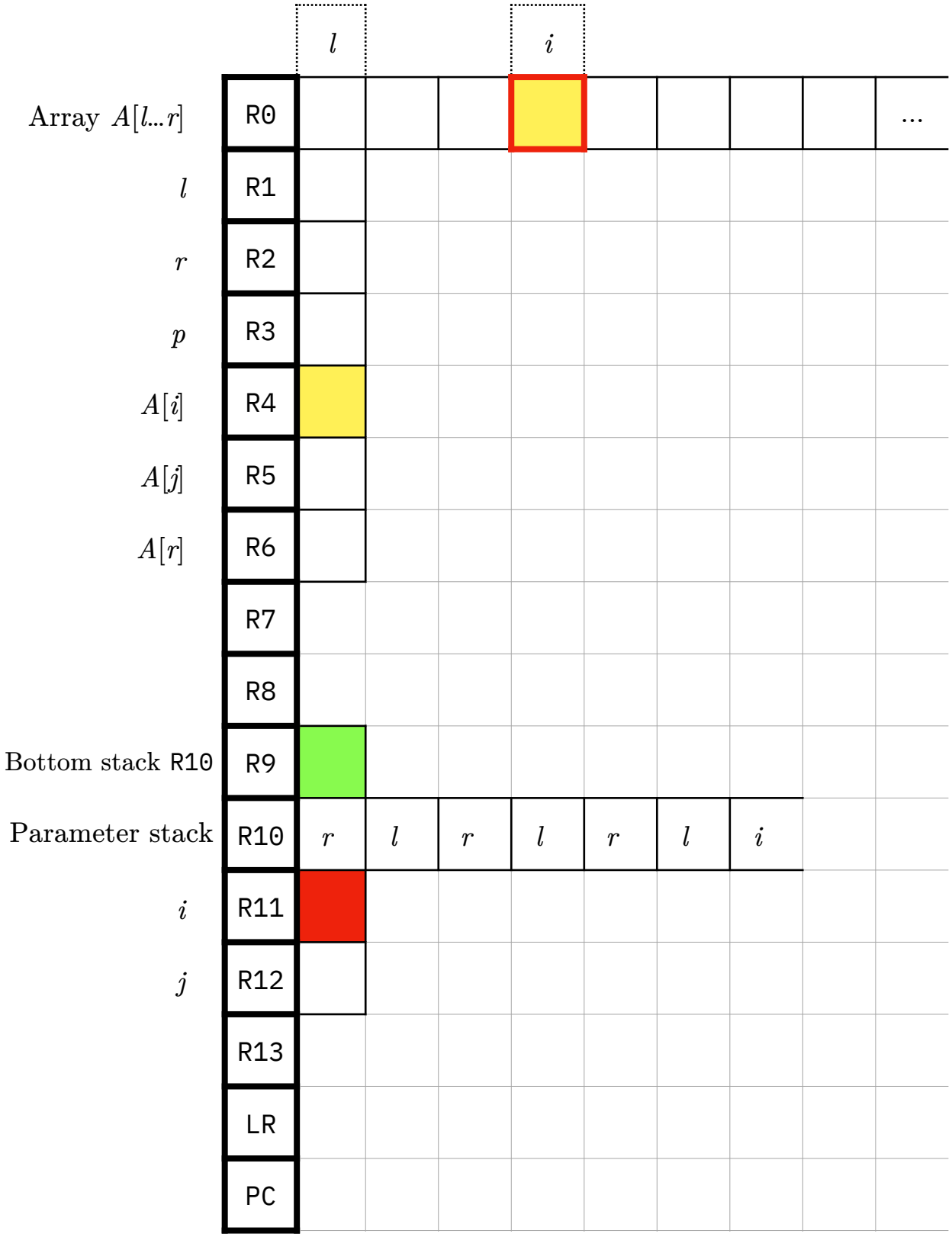
1.2.QuickSort

Algorithm QuickSort

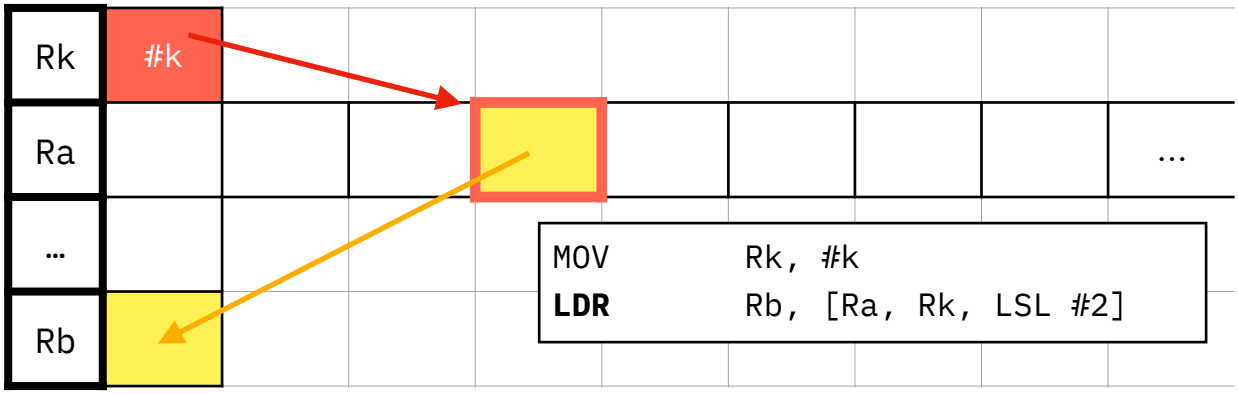
```
1:  Input: Array A[l...r] of orderable elements
2:  Output:  A partition of A[l...r] in non-decreasing order
3:
4:  if l < r then
5:          s ← Partition(A[l...r])
6:          Quicksort(A[l...s – 1])
7:          Quicksort(A[s +1...r])
```

2. ARM Assembly

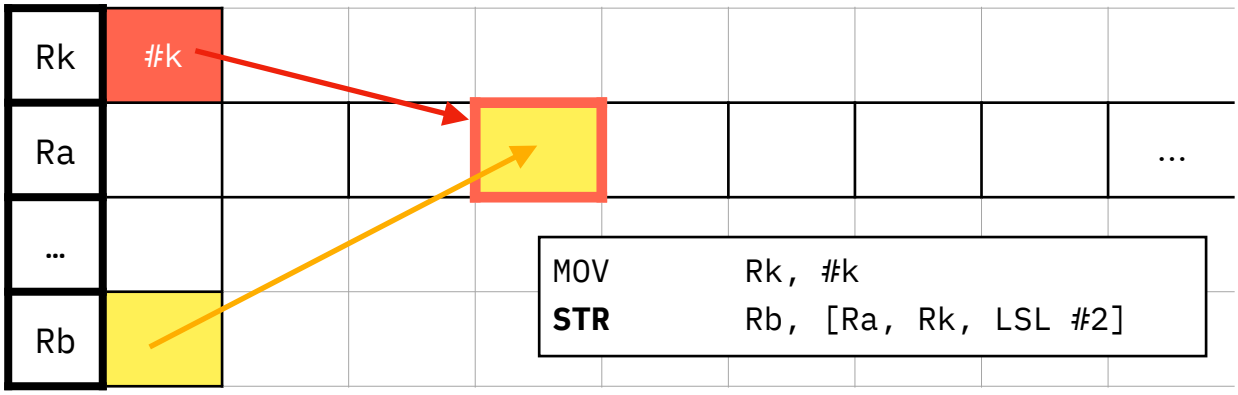
2.1.Register, Use all name from 1.)



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

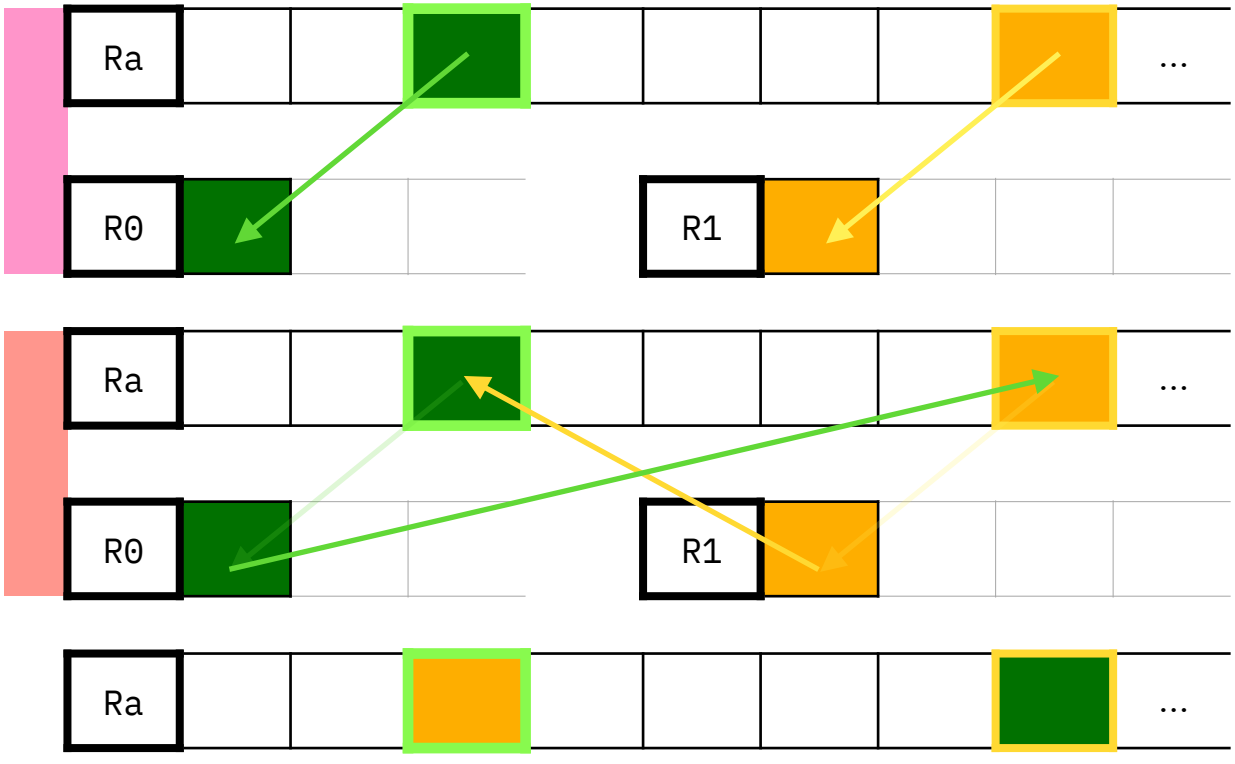


2.3.Store address in Rb to k^{th} index in Array of register Ra



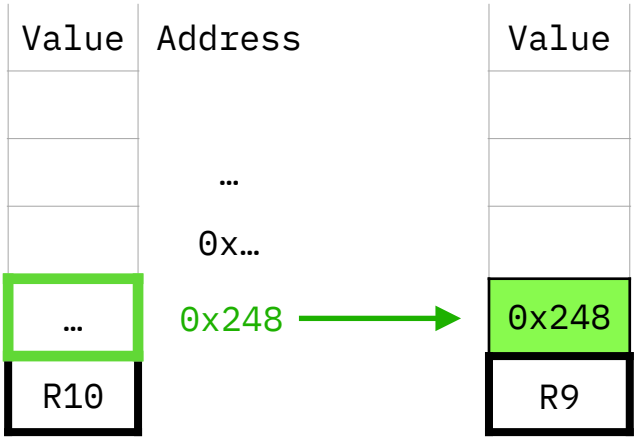
2.4.Swap index in Ra

MOV > LDR > STR

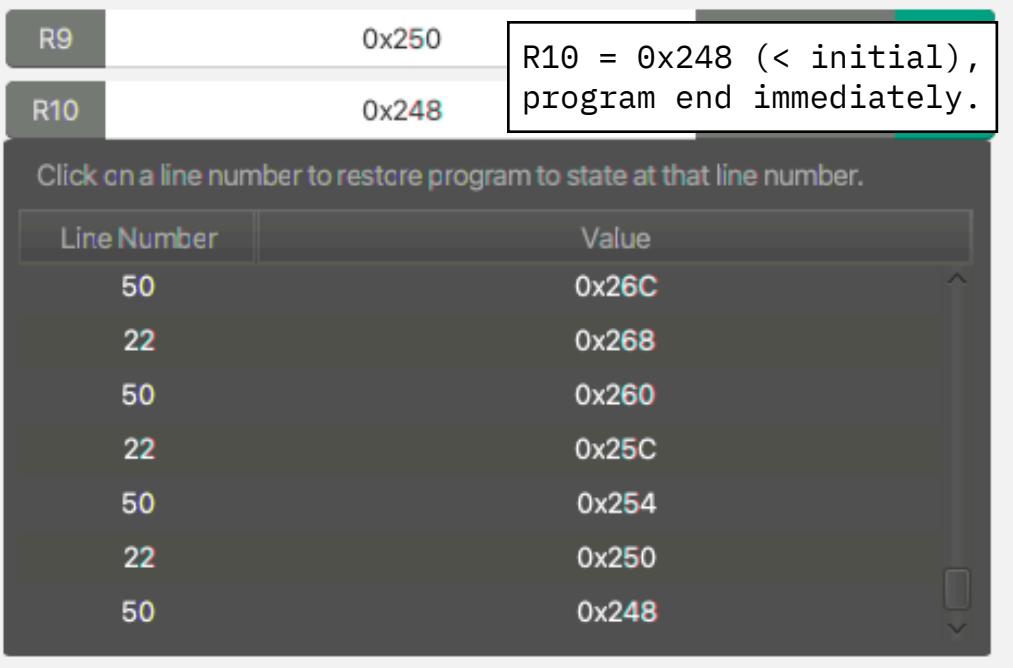


2.5.Termination

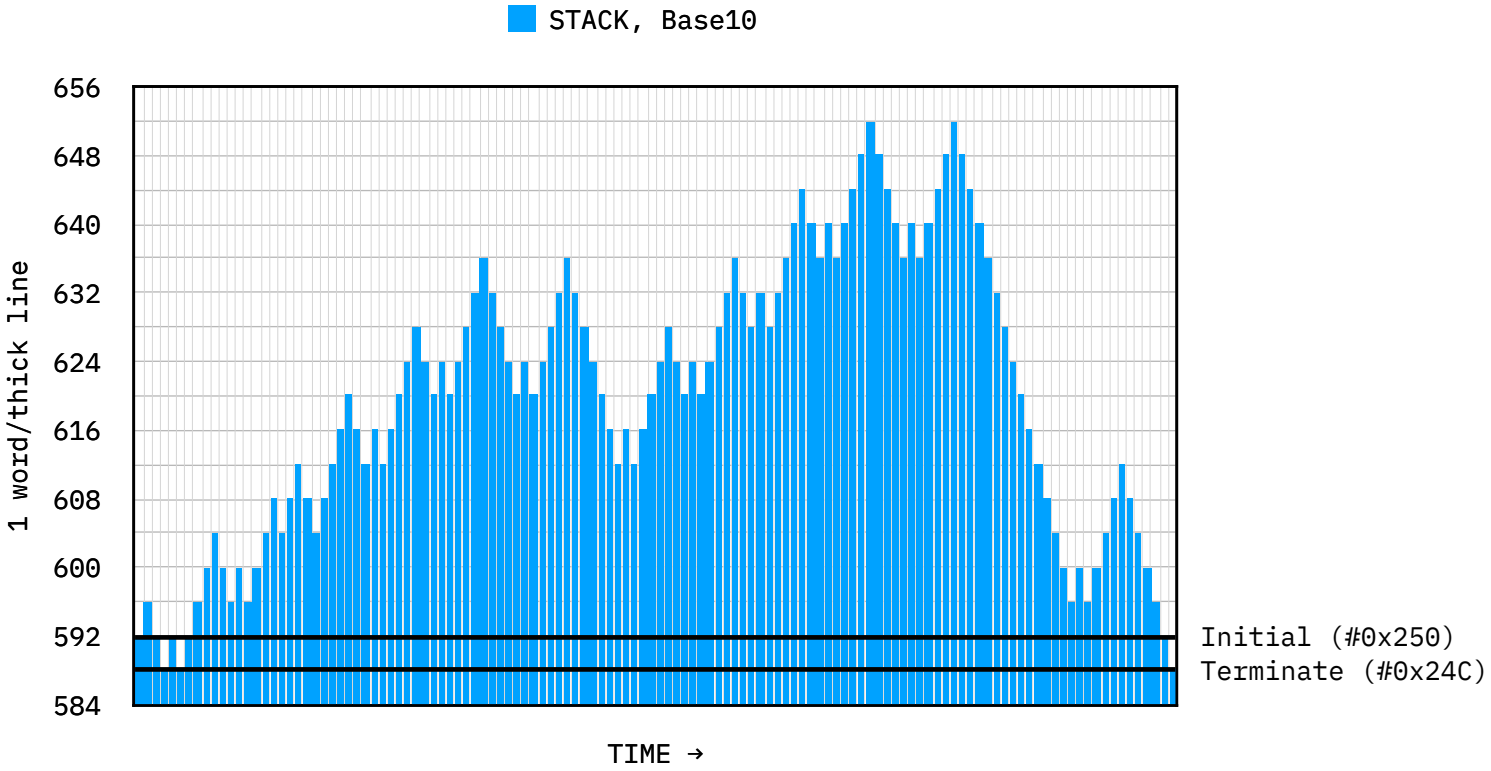
```
STACK       DCD         …
            ADR         R10,
STACK
            MOV         R9, R10

…

            CMP         R10, R9
            BGE         …
            END
```

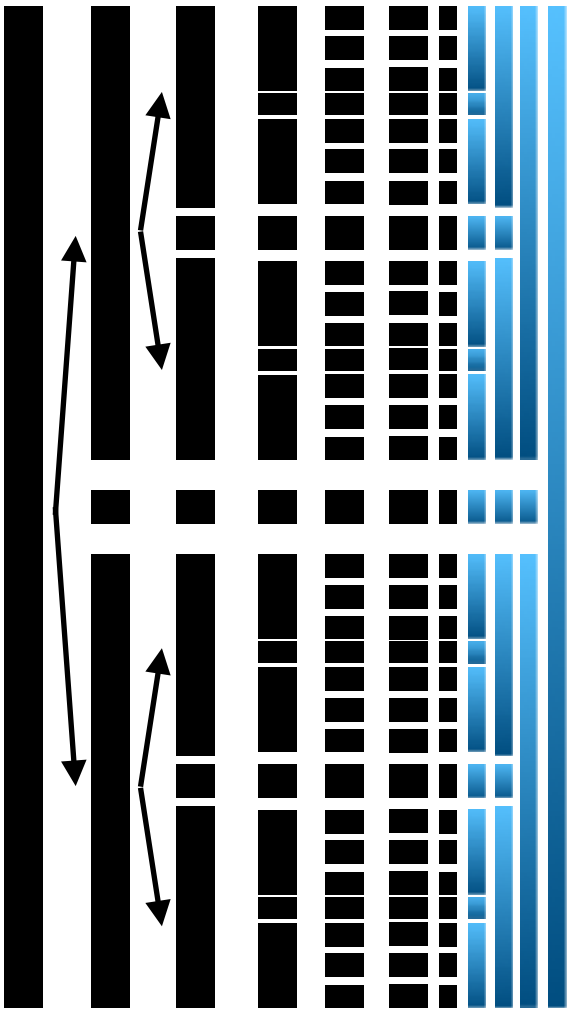


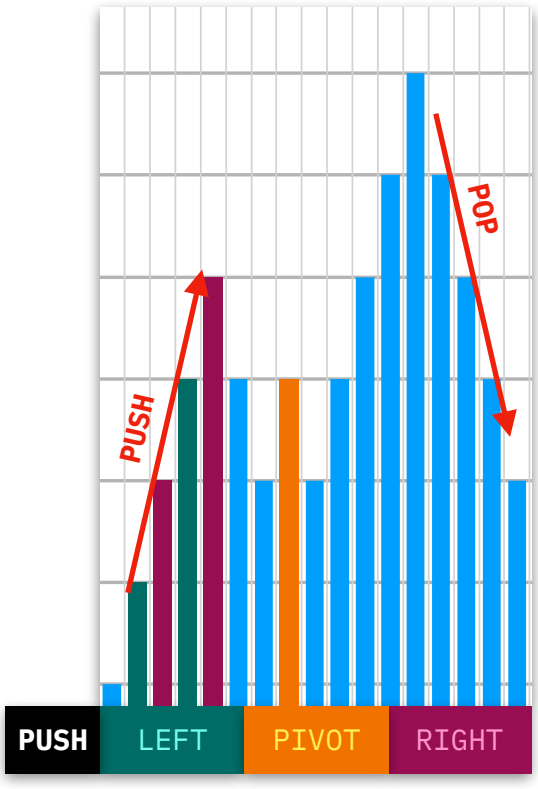
The address of register is changed due to decreasing of its parameters; however, it needs to be over the initial address. By means of comparing `R9` with `R10` to check the address, if the `BGE` is true, it continues performing quick sort with the remaining parameters in `R10`. Otherwise it ends because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



- Example from 3.) Test set

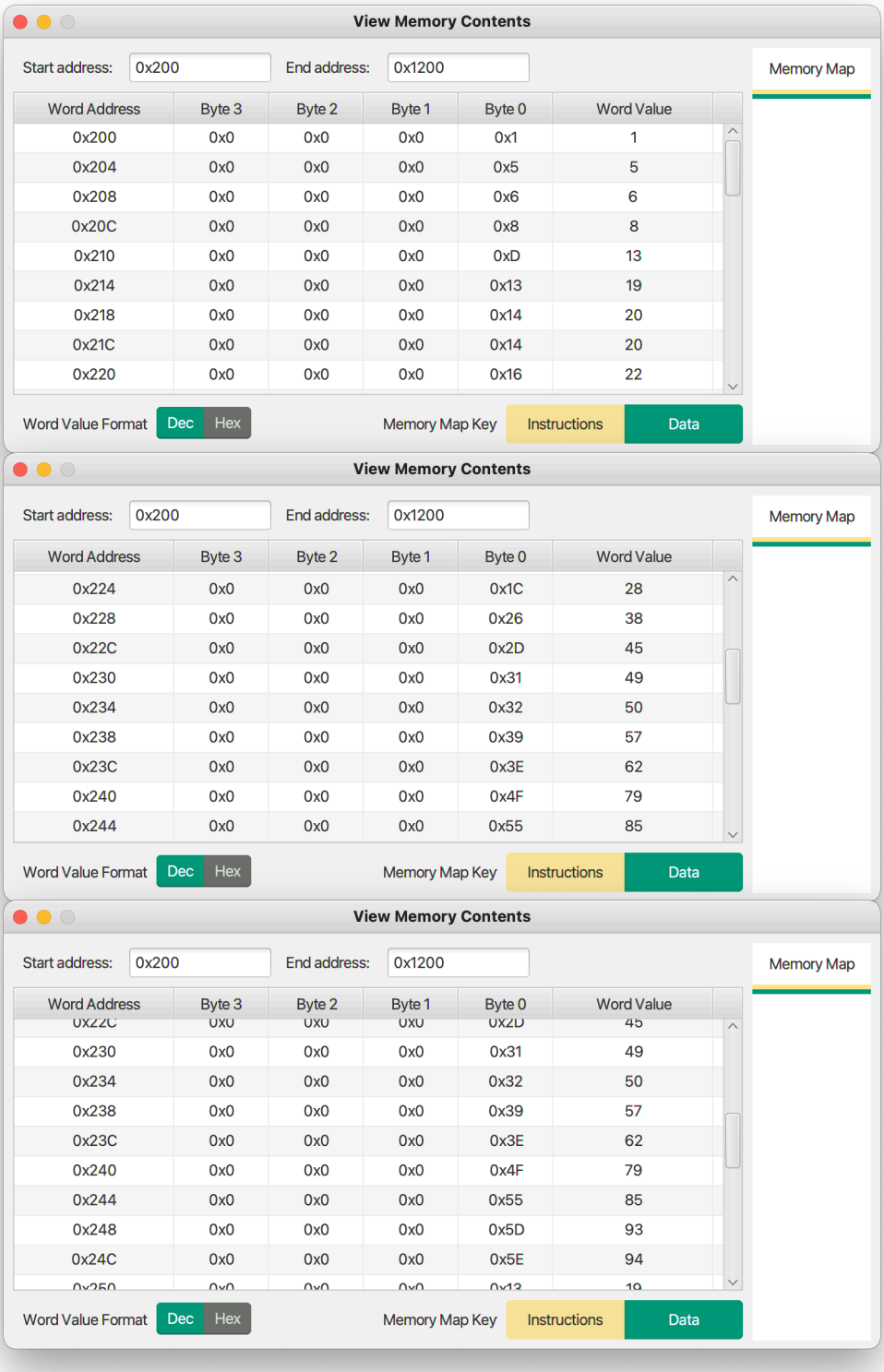






3. Result

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



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