PROBLEM SESSION 3

Instructions Sequencing

คณะผู้จัดทำ

นัธทวัฒน์	ปริมสิริคุณาวุฒิ	67070501027
วิศิษฐ์	สุวรรณเนาว์	67070501042
พลวริษฐ์	วัฒนเหมรัตน์	67070501067

เสนอ

ผศ. ราชวิชช์ สโรชวิกสิต

รายงานนี้เป็นส่วนหนึ่งของรายวิชา

CPE223 สถาปัตยกรรมคอมพิวเตอร์ (Computer Architectures)

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

ภาคเรียนที่ 2 ปีการศึกษา 2568

Instruction

1. Download “visUAL2” emulator software to compile and test your ARM assembly source code from this link: https://scc416.github.io/Visual2-doc/download
2. Write a program from Assembly programming language in the “visUAL2” emulator software, you can choose only one between
    a. Insertion sort (get 80% score.)
    b. Selection sort (get 80% score.)
    c. **Quick sort (get 100% score.)** **-> เลือกโจทย์ sorting program ข้อนี้**
3. Test and compile your program on “visUAL2” emulator software only, there is no need to compile with your native compiler on your workstation.

1. ซอร์สโค้ด (Source Code): Quick sort

C Programming language: Quick sort

```c
// QuickSort Algorithm (Lomuto Partition)
// Mapped to ARM Assembly Registers

void quickSort(int arr[], int low, int high) {
    // R0=arr, R1=low, R2=high

    // Check Base Case
    if (low < high) {

        int pivot = arr[high];  // R5 = Pivot
        int i = (low - 1);      // R3 = i

        // Loop j from low to high-1 (j = R4)
        for (int j = low; j < high; j++) {

            // Compare arr[j] vs Pivot (CMP R8, R5)
            if (arr[j] < pivot) {
                i++;            // i++ (ADD R3, #1)

                // Swap arr[i] and arr[j] (Use R12 as temp)
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // Place Pivot in correct position (Swap arr[i+1], arr[high])
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        int pi = i + 1;         // R3 = pi

        // Recursive Calls (Save/Restore Context with STMFD/LDMFD)
        quickSort(arr, low, pi - 1);  // Sort Left
        quickSort(arr, pi + 1, high); // Sort Right
    }
}
```

Assembly Programming language: Quick sort

```
          ;  Program: Quick Sort (Recursive Lomuto Partition)
          ;  ================================================
Data      DCD     10, 12, 8, 1, 5, 7, 11, 6, 8 ; Initial Data (9 Elements)

          ;  --- Main Program Initialization ---
          LDR     SP, =0x1000 ; Init Stack Pointer (Safe Memory Area)
          LDR     R0, =Data ; R0 = Base Address of Array
          MOV     R1, #0 ; R1 = Low Index (Start)
          MOV     R2, #8 ; R2 = High Index (End)

          BL      QuickSort ; Call Main Recursive Function

Stop       B       Stop ; End Program (Infinite Loop)

          ;  Subroutine: QuickSort
          ;  Args: R0=Base, R1=Low, R2=High
          ;  ================================================
QuickSort
          STMFD   SP!, {R4-R8, LR} ; Save Context & Return Address

          CMP     R1, R2 ; Check Base Case: Low >= High?
          BGE     QSReturn ; If true, single element left -> Return

          ;       --- Partition Setup (Pivot = Data[High]) ---
          SUB     R3, R1, #1 ; R3 (i) = Low - 1 (Partition Boundary)
          MOV     R4, R1 ; R4 (j) = Low (Loop Counter)

          LSL     R6, R2, #2 ; Calculate Offset for High
          LDR     R5, [R0, R6] ; R5 = Pivot Value (Data[High])

          ;       --- Partition Loop (Scan j from Low to High-1) ---
P_Loop    CMP     R4, R2 ; Check if j >= High
          BGE     P_End ; If true, exit loop
```

Assembly Programming language: Quick sort

```
        LSL     R7, R4, #2 ; Calculate Offset for j
        LDR     R8, [R0, R7] ; R8 = Data[j]

        CMP     R8, R5 ; Compare Data[j] vs Pivot
        BGE     P_Next ; If Data[j] >= Pivot, Skip Swap

        ;  --- Swap Operation: Data[i] <-> Data[j] ---
        ADD     R3, R3, #1 ; Increment i (Move Boundary)
        LSL     R6, R3, #2 ; Calculate Offset for i
        LDR     R12, [R0, R6] ; R12 = Data[i] (Temp)
        STR     R8, [R0, R6] ; Data[i] = Data[j]
        STR     R12, [R0, R7] ; Data[j] = Temp (Old Data[i])

P_Next  ADD     R4, R4, #1 ; Increment j
        B       P_Loop ; Repeat Loop

P_End    ;  --- Place Pivot Correctly: Swap Data[i+1] <-> Data[High] ---
        ADD     R3, R3, #1 ; i = i + 1 (Correct Pivot Position)
        LSL     R6, R3, #2 ; Offset for i_new
        LSL     R7, R2, #2 ; Offset for High
        LDR     R8, [R0, R6] ; R8 = Data[i_new]
        LDR     R12, [R0, R7] ; R12 = Data[High] (Pivot)
        STR     R12, [R0, R6] ; Data[i_new] = Pivot
        STR     R8, [R0, R7] ; Data[High] = Old Data[i_new]

        ;  --- Recursive Calls ---
        ;  Note: R3 now holds 'pi' (Partition Index)

        ;       1. Sort Left Side: QuickSort(Low, pi - 1)
        STMFD   SP!, {R1, R2, R3} ; Save current context
        SUB     R2, R3, #1 ; Set New High = pi - 1
        BL      QuickSort ; Recursive Call
        LDMFD   SP!, {R1, R2, R3} ; Restore context
```

Assembly Programming language: Quick sort

```
        ;       2. Sort Right Side: QuickSort(pi + 1, High)
        STMFD   SP!, {R1, R2, R3} ; Save current context
        ADD     R1, R3, #1 ; Set New Low = pi + 1
        BL      QuickSort ; Recursive Call
        LDMFD   SP!, {R1, R2, R3} ; Restore context

QSReturn
        LDMFD   SP!, {R4-R8, PC} ; Restore Registers & Return
        END
```

2. หลักการทำงานของโปรแกรม (Program Operation Description)

โปรแกรมนี้ใช้อัลกอริทึม **Quick sort** แบบ **Recursive** และใช้เทคนิคการแบ่งข้อมูลแบบ **Lomuto Partition Scheme** โดยแบ่งการทำงานออกเป็น 7 ส่วนหลัก ดังนี้

1) **MAIN PROGRAM (Initialization)**
   a) กำหนดค่า **Stack Pointer (SP)** ที่ 0x1000 สำหรับการจัดการหน่วยความจำชั่วคราวในการเรียกฟังก์ชันซ้ำ (Recursion)
   b) กำหนดให้ **R0** เก็บค่าตำแหน่งเริ่มต้นของชุดข้อมูล (Base Address)
   c) กำหนดให้ **R1** เป็นดัชนีเริ่มต้น (**Low**) เท่ากับ 0
   d) กำหนดให้ **R2** เป็นดัชนีสุดท้าย (**High**) เท่ากับ 8 (สำหรับข้อมูล 9 ตัว)
   e) ทำการเรียก (Branch) ไปยังฟังก์ชันหลัก QuickSort เพื่อเริ่มกระบวนการเรียงลำดับ
2) **QUICKSORT (Entry & Base Case)**
   a) **Context Saving:** ใช้คำสั่ง STMFD เพื่อบันทึกค่า Register ที่สำคัญ (R4-R8) และที่อยู่สำหรับกลับ (LR) ลงใน Stack ก่อนเริ่มฟังก์ชัน
   b) **Base Case:** ตรวจสอบเงื่อนไขหยุดการเรียกซ้ำ โดยเปรียบเทียบ **R1 (Low)** และ **R2 (High)**
   c) หาก **R1 ≥ R2** (เหลือข้อมูลเพียง 1 ตัวหรือไม่มี) จะกระโดดไปยัง QSReturn เพื่อจบ

3) **PARTITION SETUP (Pivot Selection)**
   a) กำหนดให้ **R3** (ตัวแปร i) เป็น Partition Boundary โดยมีค่าเริ่มต้นที่ **R1 - 1** (Low - 1)
   b) กำหนดให้ **R4** (ตัวแปร j) เป็น Loop Counter โดยมีค่าเริ่มต้นเท่ากับ **R1** (Low)
   c) คำนวณตำแหน่งของข้อมูลตัวสุดท้าย และโหลดค่า **Pivot** (Data[High]) มาเก็บไว้ใน **R5**
4) **P_LOOP (Scanning Loop)**
   a) ลูปนี้จะวนตรวจสอบข้อมูลที่ตำแหน่ง **j** ตั้งแต่ Low จนถึง High-1
   b) โหลดค่า **Data[j]** มาเก็บไว้ใน **R8**
   c) ทำการเปรียบเทียบ **R8 (Data[j])** กับ **R5 (Pivot)**
   d) หาก **Data[j] ≥ Pivot** โปรแกรมจะข้ามไปที่ P_Next เพื่อตรวจสอบข้อมูลตัวถัดไป (ไม่ต้องทำการสลับค่า)
5) **SWAP OPERATION (Inside Loop)**
   a) **Increment i :** ทำการเพิ่มค่า **R3 (i)** ขึ้น 1 เพื่อขยายขอบเขตของกลุ่มข้อมูลที่มีค่าน้อยกว่า Pivot
   b) **Swap:** ทำการสลับค่าระหว่างข้อมูลที่ตำแหน่ง **i** และตำแหน่ง **j** โดยใช้ **R12** เป็น Register สำหรับพักค่าชั่วคราว (Temp) เพื่อนำค่าน้อยไปไว้ทางซ้าย
6) **P_END (Final Pivot Placement)**
   a) ส่วนนี้ทำงานเมื่อจบลูปการตรวจสอบ (j ≥ High)
   b) เพิ่มค่า **R3 (i)** ขึ้น 1 เพื่อชี้ไปยังตำแหน่งที่ถูกต้องของ Pivot **(pi = i+1)**
   c) ทำการ **สลับค่า** ระหว่างข้อมูลที่ตำแหน่ง **i** (ซึ่งคือตำแหน่งแรกของกลุ่มค่ามาก) กับข้อมูลที่ตำแหน่ง **High** (ค่า Pivot)
   d) ผลลัพธ์คือ Pivot ถูกจัดวางในตำแหน่งที่ถูกต้อง และ **R3** จะเก็บค่าดัชนีของ Pivot **(pi)** ไว้
7) **RECURSIVE CALLS (Divide and Conquer)**
   a) **Sort Left Side:** เรียก QuickSort(arr, Low, pi - 1) เพื่อเรียงลำดับข้อมูลฝั่งซ้ายของ Pivot โดยมีการ STMFD (Push) และ LDMFD (Pop) ค่า Register R1, R2, R3 กลับมาเมื่อจบการทำงาน
   b) **Sort Right Side:** เรียก QuickSort(arr, pi + 1, High) เพื่อเรียงลำดับข้อมูลฝั่งขวาของ Pivot โดยมีการ STMFD (Push) และ LDMFD (Pop) ค่า Register R1, R2, R3 กลับมาเมื่อจบการทำงาน
   c) **Function Return:** จบการทำงานที่ QSReturn ด้วยคำสั่ง LDMFD {R4-R8, PC} เพื่อคืนค่า Register เดิมและกระโดดกลับไปยังจุดที่เรียกฟังก์ชันมา

3. ตัวอย่างการทำงานของโปรแกรม (Example of Program Execution)

กำหนดให้ชุดข้อมูลเริ่มต้นเป็น **10, 12, 8, 1, 5, 7, 11, 6, 8** โดยโปรแกรมจะเริ่มทำการ Partition ในรอบแรก (First Partition) เพื่อแบ่งแยกข้อมูลด้วย **Lomuto Partition Scheme** ดังนี้:

- **Pivot:** เลือกตัวสุดท้ายคือ **8** (Index 8)
- **i (Partition Index):** เริ่มต้นที่ -1 (ยังไม่มีข้อมูลที่น้อยกว่า Pivot)
- **j (Loop Counter):** เริ่มต้นที่ 0 วนไปจนถึง 7 (High - 1)

ตารางแสดงขั้นตอนการทำงานในรอบแรก (Step-by-Step Trace):

รอบที่ (j)	ค่า Data [j]	เปรียบเทียบกับ Pivot (8)	การดำเนินการ (Operation)	ค่า i (ล่าสุด)	สถานะ Array หลังจบขั้นตอนนี้
0	10	10 ≥ 8	Pass (มากกว่า Pivot ปล่อยผ่าน)	-1	`[10, 12, 8, 1, 5, 7, 11, 6, 8]`
1	12	12 ≥ 8	Pass	-1	`[10, 12, 8, 1, 5, 7, 11, 6, 8]`
2	8	8 ≥ 8	Pass	-1	`[10, 12, 8, 1, 5, 7, 11, 6, 8]`
3	1	1 < 8	**Swap** (Data[0] กับ Data[3])	0	`[1, 12, 8, 10, 5, 7, 11, 6, 8]`
4	5	5 < 8	**Swap** (Data[1] กับ Data[4])	1	`[1, 5, 8, 10, 12, 7, 11, 6, 8]`
5	7	7 < 8	**Swap** (Data[2] กับData[5])	2	`[1, 5, 7, 10, 12, 8, 11, 6, 8]`
6	11	11 ≥ 8	Pass	2	`[1, 5, 7, 10, 12, 8, 11, 6, 8]`
7	6	6 < 8	**Swap** (Data[3] กับ Data[7])	3	`[1, 5, 7, 6, 12, 8, 11, 10, 8]`

4. ผลลัพธ์การแสดงผล (Output Results)

1. ตัวอย่างผลลัพธ์ เมื่อมีชุดข้อมูลเป็น 10, 12, 8, 1, 5, 7, 11, 6, 8

Hex Bin Dec UDec

Registers | Memory | Symbols

Enable Byte View

Enable Reverse Direction

Symbol	Address	Value
Data	0×200	1
	0×204	5
	0×208	6
	0×20C	7
	0×210	8
	0×214	8
	0×218	10
	0×21C	11
	0×220	12

2. ตัวอย่างผลลัพธ์ เมื่อมีชุดข้อมูลเป็น 84, 15, 62, 29, 91, 37, 50, 48, 73

Hex Bin Dec UDec

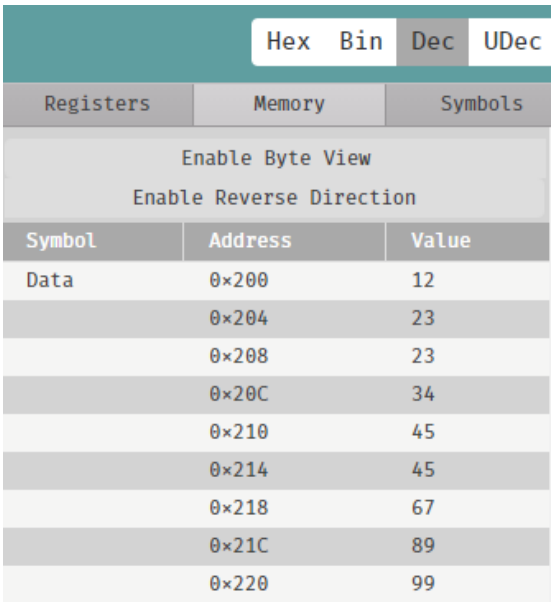
Registers | Memory | Symbols

Enable Byte View

Enable Reverse Direction

Symbol	Address	Value
Data	0×200	15
	0×204	29
	0×208	37
	0×20C	48
	0×210	50
	0×214	62
	0×218	73
	0×21C	84
	0×220	91

3. ตัวอย่างผลลัพธ์ เมื่อมีชุดข้อมูลเป็น 45, 23, 89, 45, 12, 67, 99, 34, 23



Hex Bin Dec UDec

Registers | Memory | Symbols

Enable Byte View

Enable Reverse Direction

Symbol	Address	Value
Data	0×200	12
	0×204	23
	0×208	23
	0×20C	34
	0×210	45
	0×214	45
	0×218	67
	0×21C	89
	0×220	99