PROBLEM SESSION 2

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

Problem 1

ชุดคำสั่ง (Instruction Set)

คำสั่ง (Instruction)	คำอธิบาย (Description)
LOAD R, [A]	โหลดค่าจากตำแหน่งหน่วยความจำ A เข้าสู่รีจิสเตอร์ R
STORE R, [A]	เก็บค่าในรีจิสเตอร์ R ไปยังตำแหน่งหน่วยความจำ A
ADD R1, R2, R3	R1 = R2 + R3
SUB R1, R2, R3	R1 = R2 - R3
MOV R1, R2	คัดลอกค่าจาก R2 ไปยัง R1

ข้อสมมติ (Assume):

- คำสั่งทั้งหมดใช้เวลา 1 รอบนาฬิกา (cycle) ยกเว้นมีการระบุเป็นอย่างอื่น
- รีจิสเตอร์เริ่มต้นมีค่าเป็น 0 ยกเว้นมีการระบุ

กำหนดค่าเริ่มต้นดังนี้: R1=5, R2=10, R3=0, และ MEM[100] = 20 พร้อมด้วยลำดับคำสั่งต่อไปนี้:

```
1. LOAD R3, [100]
2. ADD R1, R3, R2
3. SUB R2, R1, R3
4. STORE R2, [104]
```

เติมค่าของรีจิสเตอร์และตำแหน่งหน่วยความจำแต่ละช่อง:

Ri/LOC	R1	R2	R3	[100]	[104]
ค่าเริ่มต้น (Initial)	5	10	0	20	X
คำสั่งที่ 1 (Inst 1.)	5	10	20	20	X
คำสั่งที่ 2 (Inst 2.)	30	10	20	20	X
คำสั่งที่ 3 (Inst 3.)	30	10	20	20	X
คำสั่งที่ 4 (Inst 4.)	30	10	20	20	10

Problem 2

1. Write a program to calculate the 44th Fibonacci number without using any special library in C language

Source code

Program to calculate the 44th Fibonacci number without using any special library
(Code : https://github.com/MadMax168/CPE223-ComArk-Labs/tree/main/lab01)

Algorithm explanation

44th Fibonacci number สำหรับของกลุ่มเรามีการเขียนในรูปแบบ Iterative Bottom-Up โดยจะมีหลักการการทำงานดังต่อไปนี้

1. Base Case:
    - ถ้า n = 0 จะคืนค่า 0
    - ถ้า n = 1 จะคืนค่า 1
2. Iterative Calculation:
    - กำหนดตัวแปร
        - a = 0 //แทน F(n-2)
        - b = 1 //แทน F(n-1)
        - curr = 0 //ไว้สำหรับเก็บค่า F(n)
    - Loop ตั้งแต่ 2 ไปถึง n
        - คำนวณหา curr = a + b ตามสูตร F(n) = F(n-1) + F(n-2)
        - เปลี่ยนค่า a = b
        - เปลี่ยนค่า b = curr
        - Example : F(2) = F(0) + F(1)
        - Example : F(3) = F(1) + (2)
    - Return ค่า curr [F(n)]

C code

```c
#include<stdio.h>
long fibonacci(int n) {
    if (n <= 0) return 0;
    if (n == 1) return 1;
    long a = 0 , b = 1;
    long curr = 0;
    for ( int i = 2; i <= n; i++ ) {
        curr = a + b;
        a = b;
        b = curr;
    }
    return curr;
}
int main() {
    int n = 44;
    printf("The %dth Fibonacci number is: %lu\n", n, fibonacci(n));
    return 0;
}
```

C to ARM Assembly Mapping

Key ARM Assembly Instructions Used:

ARM Instruction	Description
sub sp, sp, #N	Allocate stack space
str x0, [sp, #N]	Store register to stack
ldr x8, [sp, #N]	Load from stack to register
cmp x8, #N	Compare register with immediate value
b.gt LABEL	Branch if greater than
add x8, x0, x8	Add two registers
bl _function	Branch with link (function call)
ret	Return from function

1. Function prologue (setup):

C	`long fibonacci(int n) {`

Assembly	`fibonacci:`
	`push rbp`
	`mov rbp, rsp`
	`mov DWORD PTR [rbp-36], edi`

2. Base case checker (n == 0):

C	`if (n == 0) return 0;`

Assembly	`cmp DWORD PTR [rbp-36], 0`
	`jg .L2`
	`mov eax, 0`
	`jmp .L3`

3. Base case checker (n == 1):

C	`if (n == 1) return 1;`

Assembly	`.L2:`
	`cmp DWORD PTR [rbp-36], 1`
	`jne .L4`
	`mov eax, 1`
	`jmp .L3`

4. Variable setting:

C	`long a = 0 , b = 1;` `long curr = 0;`

Assembly	`.L4:`
	`mov QWORD PTR [rbp-8], 0`
	`mov QWORD PTR [rbp-16], 1`
	`mov QWORD PTR [rbp-24], 0`

5. Loop initial:

C	for (int i = 2; i <= n; i++) {

Assembly	`mov DWORD PTR [rbp-28], 2`
	`jmp .L5`
	`.L6:`
	`...`
	`add DWORD PTR [rbp-28], 1`
	`.L5:`
	`mov eax, DWORD PTR [rbp-28]`
	`cmp eax, DWORD PTR [rbp-36]`
	`jle .L6`

6. Fibonacci concept [F(n) = F(n-1) + f(n-2)]: //Iterative version

C	curr = a + b a = b b = curr

Assembly	`.L6:`
	`mov rdx, QWORD PTR [rbp-8]`
	`mov rax, QWORD PTR [rbp-16]`
	`add rax, rdx`
	`mov QWORD PTR [rbp-24], rax`
	`mov rax, QWORD PTR [rbp-16]`
	`mov QWORD PTR [rbp-8], rax`
	`mov rax, QWORD PTR [rbp-24]`
	`mov QWORD PTR [rbp-16], rax`

7. Return value (curr) + Ending function:

C	return curr; }

Assembly	`mov rax, QWORD PTR [rbp-24]`
	`.L3:`
	`pop rbp`
	`ret`

8. Main function:

C	int main() {
	int n = 44;
	printf(“The %dth Fibonacci number is: %lu\n”, n, fibonacci(n));
	return 0;
	}

Assembly	.LC0:
	.string "The %dth Fibonacci number is: %lu\n"
	main:
	push rbp
	mov rbp, rsp
	sub rsp, 16
	mov DWORD PTR [rbp-4], 44
	mov eax, DWORD PTR [rbp-4]
	mov edi, eax
	call fibonacci
	mov rdx, rax
	mov eax, DWORD PTR [rbp-4]
	mov esi, eax
	mov edi, OFFSET FLAT:.LC0
	mov eax, 0
	call printf
	mov eax, 0
	leave
	ret

Command to convert C into ARM Assembly

Method 1 : Using GCC/Clang on Mac

```
# Generate ARM assembly from C code
gcc -S -O0 fibonacci.c -o fibonacci_arm.s
# Or using Clang
clang -S -O0 fibonacci.c -o fibonacci_arm.s
```

Command Flags Explanation:

- -S : Generate assembly code (stop before assembling)
- -O0 : No optimization (makes assembly easier to read and map to C)
- fibonacci.c : Input C source file
- -o fibonacci_arm.s : Output assembly file

Method 2: Using Godbolt Compiler Explorer (Cross-platform)

1. Go to https://godbolt.org/
2. Paste the C code in the left panel
3. Select compiler: ARM64 gcc or ARM gcc
4. Add compiler flag: -O0
5. The assembly appears automatically in the right panel
6. Copy the assembly code

Output

```
natthawat@Natthawats-MacBook-Pro lab1-q2 % cd "/Users/natthawat/Desktop/
code/cpe223/lab1-q2/" && gcc iterative.c -o iterative && "/Us
ers/natthawat/Desktop/code/cpe223/lab1-q2/"iterative
701408733
natthawat@Natthawats-MacBook-Pro lab1-q2
```