

รายงาน

หัวข้อ Simple Machine Computer (SMC) – Assembly and Simulator

261304 Computer Architecture

จัดทำโดย

นายณฐพร ไพรินทร์	660610749
นางสาวโยษิตา สติหมื่น	660610788
นางสาวรัชชาพร บัวนุช	660610790
นางสาววรรณคณา จิตวรรณคณา	660601792
นายธีรชัย ยอดบุญ	660612146

เสนอ

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยพกุล

รายงานเล่มนี้เป็นส่วนหนึ่งของรายวิชาสถาปัตยกรรมคอมพิวเตอร์

รหัสวิชา 261304

ภาคเรียนที่ 1 ปีการศึกษา 2568

คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเชียงใหม่

สารบัญ

	หน้า
หน้าปก	1
สารบัญ	2
บทนำ	3
Design ของ Assembler	4
Design ของ Simulator	8
ภาคผนวก	11
ตารางวางแผนการทำงานกลุ่มและรายละเอียด	13
โค้ดในส่วนต่างๆ	19
1. Assembler	19
1.1 Parser	19
1.2 Assembler	27
2. Simulator	36
3. Assembly Programs	40
3.1 multiply.asm	40
3.2 multiply.mc	40
3.3 factorial.asm	41
3.4 factorial.mc	41
4. Test	42
4.1 multiply test	42
4.2 factorial test	43

บทนำ

โครงการนี้ เป็นส่วนหนึ่งของรายวิชา Computer Architecture รหัสวิชา 261304 มีวัตถุประสงค์เพื่อให้นักศึกษาเข้าใจหลักการทำงานของเครื่องคอมพิวเตอร์ในระดับสถาปัตยกรรม ผ่านการออกแบบและจำลองการทำงานของเครื่องจำลองแบบง่ายที่เรียกว่า Simple Machine Computer (SMC) ซึ่งสามารถแปลคำสั่งและประมวลผลคำสั่งในรูปแบบภาษา Assembly ได้

โดยในโครงการนี้ได้แบ่งการทำงานออกเป็นสามส่วนหลัก ได้แก่ ส่วนของ Assembler สำหรับแปลงโปรแกรมภาษา Assembly ให้เป็นรหัสเครื่อง (Machine Code) ซึ่งพัฒนาโดยใช้ภาษา C++, ส่วนของ Simulator สำหรับจำลองการทำงานของหน่วยประมวลผล (CPU) และหน่วยความจำ โดยอ่านคำสั่งจากไฟล์รหัสเครื่องและดำเนินการตามวัจกรรม Fetch–Decode–Execute พัฒนาโดยใช้ภาษา Java, และส่วนของ Assembly Programs & Test Cases ซึ่งเป็นส่วนของการเขียนโปรแกรม ตัวอย่างในภาษา Assembly (คือ โปรแกรมคุณตัวเลข และ โปรแกรมหาค่าแฟกทอเรียล) พร้อมออกแบบกรณีทดสอบเพื่อใช้ตรวจสอบความถูกต้องของ Assembler และ Simulator

การทำงานร่วมกันของทั้งสามส่วนนี้ช่วยให้นักศึกษาได้เข้าใจตั้งแต่กระบวนการแปลคำสั่ง ในระดับคำสั่ง ไปจนถึงการจำลองการทำงานของคำสั่งแต่ละประเภทในระดับฮาร์ดแวร์ เช่น ซึ่งเป็นพื้นฐานสำคัญของการศึกษาสถาปัตยกรรมคอมพิวเตอร์และระบบประมวลผล

Design ของ Assembler

1. ภาพรวม

Assembler เป็นส่วนประกอบสำคัญของระบบ SMC (Simple Machine Computer) ซึ่งมีหน้าที่หลักในการแปลงโปรแกรมภาษา Assembly (.asm) ให้กลายเป็น Machine Code (.mc) ที่สามารถนำไปประมวลผลได้โดย Simulator ซึ่งเขียนด้วยภาษา Java โปรแกรมนี้ถูกพัฒนาโดยใช้ภาษา C++ เพื่อให้มีประสิทธิภาพสูงในการจัดการข้อมูลและไฟล์ขนาดใหญ่ โดยใช้คุณสมบัติของ STL เช่น vector, unordered_map และ fstream เพื่อให้การทำงานสะดวกและรวดเร็ว

2. โครงสร้างการทำงานแบบ Two-Pass Assembler

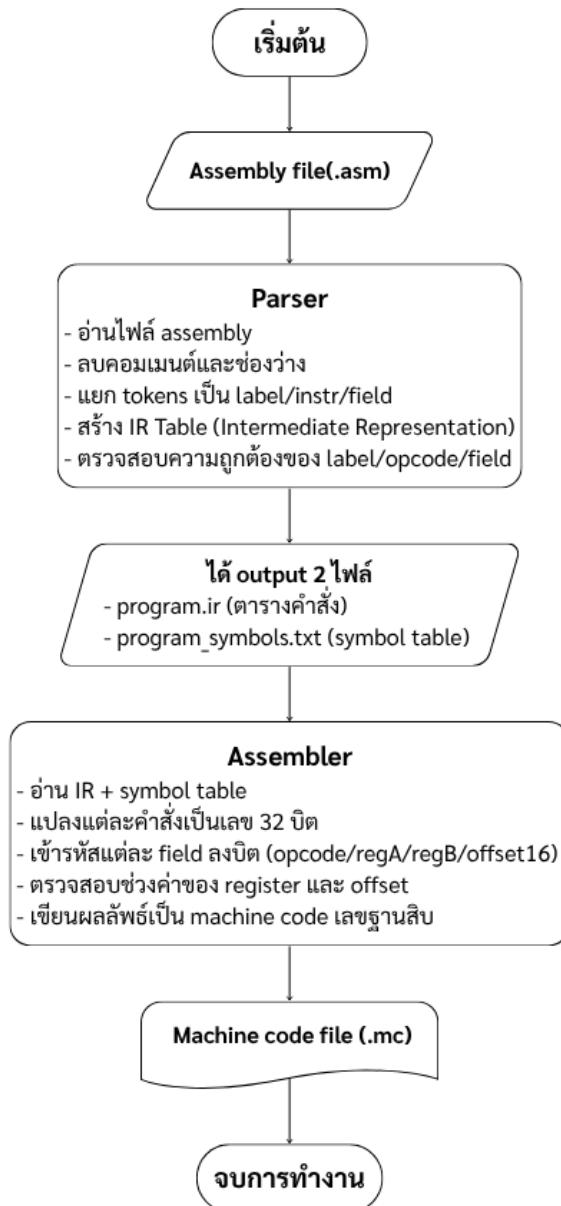
Assembler ถูกออกแบบให้ทำงานสองรอบ (Two-Pass) เพื่อให้สามารถจัดการกับ label ที่ยังไม่ได้ประกาศได้อย่างถูกต้อง โดยแบ่งการทำงานออกเป็นสองขั้นตอนหลักคือ Pass 1 และ Pass 2 ดังนี้:

Pass 1 – Parsing & Symbol Resolution:

- อ่านไฟล์ .asm ทีละบรรทัด
- แยก label, mnemonic, operand
- ตรวจสอบชื่อ label และบันทึกใน Symbol Table
- แยกคำสั่ง .file เพื่อเก็บข้อมูลประเภท Data

Pass 2 – Encoding:

- ใช้ข้อมูลจาก Pass 1 เพื่อแทนที่ label ด้วย address ที่แท้จริง
- ตรวจสอบความถูกต้องของ opcode และ operand
- แปลงเป็น Machine Code และเขียนผลลัพธ์เป็นไฟล์ .mc



3. โครงสร้างข้อมูล (Data Structures)

1. Symbol Table ใช้ `unordered_map<string,int>` เก็บชื่อ label และ address
2. Intermediate Representation (IR) ใช้ struct `IRLine` เก็บข้อมูลคำสั่งหลังจาก Parser แยก field แล้ว

4. รูปแบบผลลัพธ์ (Output Format)

Assembler จะสร้างไฟล์ผลลัพธ์ .mc โดยแต่ละบรรทัดคือค่าจำนวนเต็ม 32 บิต (ฐาน 10) ซึ่งเป็นผลจากการรวมบิตตามรูปแบบของคำสั่ง R/I/J/O-type ของ SMC เช่น add, lw, sw, beq, halt, .fill เป็นต้น

5. ตัวอย่าง Input/Output

ตัวอย่าง Assembly:

```
assembler > test(assembly-language.asm
RatchapornB, 3 hours ago | 1 author (RatchapornB)
1 # test ตามรายละเอียด project ของอาจารย์
2      lw 0 1 five      load reg1 with 5 (uses symbolic address)
3      lw 1 2 3         load reg2 with -1 (uses numeric address)
4 start  add 1 2 1    decrement reg1
5      beq 0 1 2       goto end of program when reg1==0
6      beq 0 0 start   go back to the beginning of the loop
7      noop
8 done   halt          end of program
9
10     five   .fill 5
11     neg1   .fill -1
12     stAddr .fill start   will contain the address of start
13
```

ผลลัพธ์ Machine Code (.mc):

```
assembler > machineCode.mc
RatchapornB, 4 days ago | 1 author (RatchapornB)
1 8454151
2 9043971
3 655361
4 16842754
5 16842749
6 29360128
7 25165824
8 5
9 -1
10 2 Generate code: Alt+Shift+Enter RatchapornB, 5 d
11
```

6. การตรวจสอบความผิดพลาด (Error Handling)

Assembler มีระบบตรวจสอบข้อผิดพลาด เช่น:

- ตรวจสอบ label ซ้ำกัน
- ตรวจสอบ opcode ไม่ถูกต้อง
- ตรวจสอบจำนวน operand
- ตรวจสอบ immediate overflow (เกิน 16-bit signed)
- ตรวจสอบ register ผิดช่วง (0-7)
- ตรวจสอบ undefined label

7. ภาษาและเทคนิคที่ใช้ (Implementation Language)

Assembler พัฒนาโดยใช้ภาษา C++17 เพื่อให้มีประสิทธิภาพและความยืดหยุ่นสูง ใช้ STL เช่น unordered_map, vector, และ fstream สำหรับการจัดการข้อมูล ใช้ฟังก์ชันช่วยเหลือเช่น tryParseInt, packR/I/J/O และ rtrim เพื่อให้การเข้ารหัสคำสั่งทำได้ง่ายและมีความถูกต้อง

8. สรุป

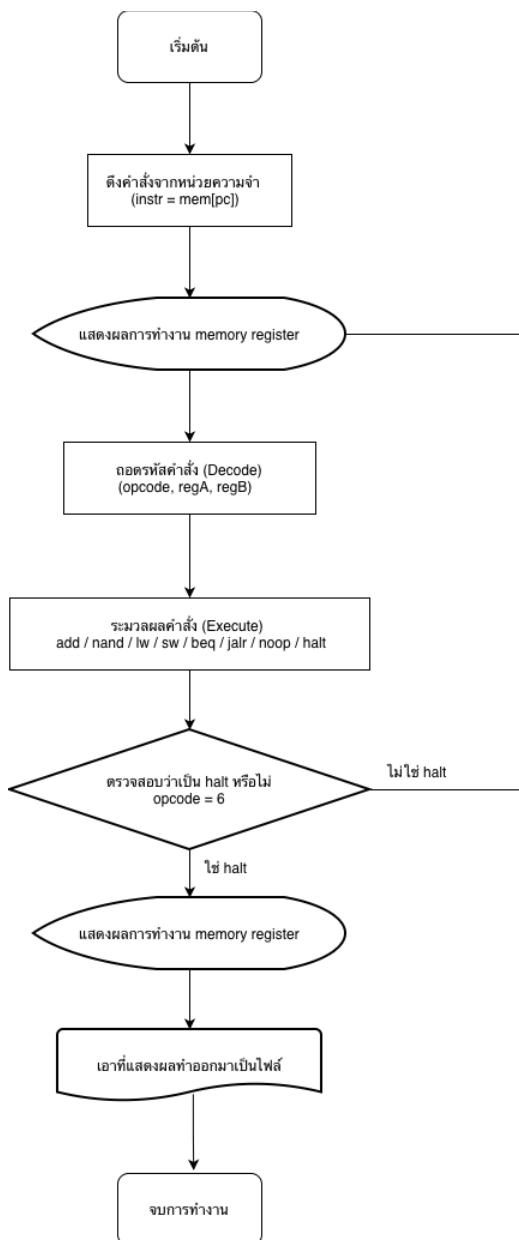
Assembler ที่พัฒนานี้สามารถอ่านไฟล์ Assembly ได้อย่างถูกต้อง ตรวจสอบ label และ opcode ได้ครบถ้วน และแปลงเป็น Machine Code ที่สามารถใช้งานร่วมกับ SMC Simulator ได้อย่างสมบูรณ์ โดยมีโครงสร้างชัดเจน และกำหนดที่ระหว่าง Parser (Pass 1) และ Assembler (Pass 2) อย่างเป็นระบบ

Design ของ Simulator

1. ภาพรวม

ส่วนของ simulator นี้ ก็คือการเขียน program ที่สามารถ simulate machine code program input ของส่วนนี้ ก็คือ machine code ที่ถูกสร้างจาก assembler นี้ ควรเริ่มจากการ initialize registers ทุกตัวและ set program counter เป็น 0 และจะ simulate จนกระทั่ง halt simulator ควรจะเรียก printState ก่อนที่จะทำแต่ละ instruction และก่อนที่จะ exit จาก program function นี้ จะ print state ปัจจุบันของ machine

2. โครงสร้างการทำงาน



3. โครงสร้างข้อมูล (Data Structures)

ใช้ int array เก็บข้อมูล Register และ Memory

ใช้ boolean ในการเช็ค halt

ใช้ int เก็บ pc , numMemory

ใช้ ArrayList เก็บค่าคำสั่ง (machine code) ตอนอ่านจากไฟล์

4. รูปแบบผลลัพธ์ (Output Format)

จะสามารถดูใน terminal หรือ สร้างไฟล์ใหม่มาเพื่อดูผลลัพธ์โดยจะแสดงแต่ละ state ที่แสดง mem กับ reg ที่ได้ที่ละขั้น

5. ตัวอย่าง Input/Output

```

☰ machine_code.mc
 1  8454151
 2  9043971
 3  655361
 4  16842754
 5  16842749
 6  29360128
 7  25165824
 8  5
 9  -1
10  2
11

1  memory[0]=8454151
2  memory[1]=9043971
3  memory[2]=655361
4  memory[3]=16842754
5  memory[4]=16842749
6  memory[5]=29360128
7  memory[6]=25165824
8  memory[7]=5
9  memory[8]=-1
10 memory[9]=2
11 @@@
12 ✓ state:
13   | pc 0
14   |   memory:
15   |     mem[ 0 ] 8454151
16   |     mem[ 1 ] 9043971
17   |     mem[ 2 ] 655361
18   |     mem[ 3 ] 16842754
19   |     mem[ 4 ] 16842749
20   |     mem[ 5 ] 29360128
21   |     mem[ 6 ] 25165824
22   |     mem[ 7 ] 5
23   |     mem[ 8 ] -1
24   |     mem[ 9 ] 2
25   | registers:
26   |     reg[ 0 ] 0
27   |     reg[ 1 ] 0
28   |     reg[ 2 ] 0
29   |     reg[ 3 ] 0
30   |     reg[ 4 ] 0
31   |     reg[ 5 ] 0
32   |     reg[ 6 ] 0
33   |     reg[ 7 ] 0
34 end state
428  |   reg[ 2 ] -1
429  |   reg[ 3 ] 0
430  |   reg[ 4 ] 0
431  |   reg[ 5 ] 0
432  |   reg[ 6 ] 0
433  |   reg[ 7 ] 0
434 end state
435
436 @@@
437 ✓ state:
438   |   pc 7
439   |   memory:
440   |     mem[ 0 ] 8454151
441   |     mem[ 1 ] 9043971
442   |     mem[ 2 ] 655361
443   |     mem[ 3 ] 16842754
444   |     mem[ 4 ] 16842749
445   |     mem[ 5 ] 29360128
446   |     mem[ 6 ] 25165824
447   |     mem[ 7 ] 5
448   |     mem[ 8 ] -1
449   |     mem[ 9 ] 2
450   | registers:
451   |     reg[ 0 ] 0
452   |     reg[ 1 ] 0
453   |     reg[ 2 ] -1
454   |     reg[ 3 ] 0
455   |     reg[ 4 ] 0
456   |     reg[ 5 ] 0
457   |     reg[ 6 ] 0
458   |     reg[ 7 ] 0
459 end state
460
461

```

6. การตรวจสอบความผิดพลาด (Error Handling)

- ตรวจสอบ infinity จากการหา halt ไม่เจอใส่เพดานสเต็ป
- ถ้า pc หลุดช่วงหน่วยความจำ จะอ่าน mem[pc] ไม่ได้ เช็ค pc ก่อน fetch
- ที่อยู่ addr = R[rs] + imm32 หลุดนอกช่วง 0..65535
- offset 16 บิตของ I-type (lw, sw, beq) อาจติดลบ ต้องแปลงเป็น 32 บิตก่อนใช้งาน
convertNum

7. ภาษาและเทคนิคที่ใช้ (Implementation Language)

Simulator พัฒนาโดยใช้ภาษา Java (JDK 17+) เพื่อให้ดูผลงานจำลองได้เสถียรและพกพาจ่ายใช้โครงสร้างข้อมูลมาตราฐานของ Java เช่น int[] สำหรับหน่วยความจำ/รีจิสเตอร์ และ ArrayList สำหรับอ่านไฟล์อินพุต ร่วมกับ BufferedReader/FileReader ในการโหลด machine code

8. สรุป

Simulator นี้สามารถทำงาน Fetch → Decode → Execute → Update PC จนกว่าจะเจอ halt และแต่ละรอบก่อนการทำงานจะมีการ printState เพื่อแสดงสถานะในขณะนั้น

มีการใช้ int array ในการเก็บ register, memory และยังมี ArrayList เก็บคำคำสั่ง (machine code) ตอนอ่านจากไฟล์

ภาคผนวก

รายชื่อสมาชิกกลุ่ม

นายณธพร ไพรินทร์	660610749
นางสาวโยษิตา สติหมื่น	660610788
นางสาวรัชชาพร บัวนุช	660610790
นางสาววรรณคณา จิตวรรณคณา	660601792
นายธีรัช ยอดบุญ	660612146

รายงานหน้าที่สมาชิก

1. Assembler (รับผิดชอบโดย นางสาวรัชชาพร บัวนุช และ นางสาวโยษิตา สติหมื่น)

นางสาวรัชชาพร บัวนุช :

- Parsing assembly (อ่านไฟล์ .asm, แยกบรรทัด, tokenize)
- สร้าง symbol table (สำหรับ labels กับ addresses)

นางสาวโยษิตา สติหมื่น :

- Encode instructions เป็น machine code ตาม ISA
- เขียน output file (.mc หรือ .bin)

2. Simulator (รับผิดชอบโดย นายณธพร ไพรินทร์ และ นายธีรัช ยอดบุญ)

นายณธพร ไพรินทร์ :

- ออกแบบโครงสร้าง simulator (register file, memory, PC)
- Implement fetch-decode-execute loop

นายธีรัช ยอดบุญ :

- Implement execution ของ instructions ตามประเภท (R/I/J/O-type)
- Handle special cases เช่น halt, beq (branch), overflow check

3. Assembly Programs (รับผิดชอบโดย นางสาววรรณคณา จิตารางคณา)

เขียน assembly 2 โปรแกรม (ตามโจทย์):

- โปรแกรม คูณเลข 2 จำนวน (เช่น repeated addition หรือ shift-and-add)
- โปรแกรม recursive function (factorial)
- ช่วยทุกคนตรวจสอบ/ทดสอบด้วย assembler + simulator ว่าได้ผลลัพธ์ถูกต้อง
- รวมกับทุกคนทำ test case สำหรับตรวจสอบ assembler/simulator

4. ส่วนอื่นๆ

Testing: ทุกคนช่วยกันออกแบบ test case เพิ่ม (ทั้งถูกและผิด)

Report:

- บทนำ (นางสาววรรณคณา)
- Design ของ assembler (นางสาวรัชชาพร และ นางสาวโยษิตา)
- Design ของ simulator (นายณัฐพร และ นายธีรัชช)
- ภาคผนวก รายงานหน้าที่สมาชิก (นางสาววรรณคณา)
- ภาคผนวก Example program (multiply + recursive) (ทุกคน)
- ภาคผนวก Result & Conclusion (ทุกคน)

ตารางวางแผนการทำงานกลุ่ม

	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
สรุป discord																																
เขียนสมาร์ทิก																																
ประชุม1 วางแผน																																
วางแผนบุคคล																																
ส่งความคืบหน้า1																																
ประชุม2 รายงาน																																
ดำเนินงาน																																
ส่งความคืบหน้า2																																
ประชุม3 รายงาน																																
ดำเนินงาน																																
ส่งความคืบหน้า3																																
ประชุม4 รายงาน																																
ดำเนินงาน																																
เรื่องงาน																																
ส่งงาน																																
ผ่านเสนอ																																

รายละเอียด

สัปดาห์ที่ 1 : วางแผนการทำงานกลุ่ม และวางแผนการทำงานในส่วนที่ได้รับมอบหมาย

ประชุมครั้งที่ 1 วันที่ 21.09.2568

- อธิบายรายละเอียดแผนงาน + หน้าที่ อธิบายแผนการทำงานกลุ่ม
- แบ่งงานเป็นทีมย่อย นัดหมายส่งความคืบหน้าทุกวันศุกร์ ประชุมวันเสาร์
- วางแผนการทำงานในส่วนของตัวเอง แล้วส่งเป็นความคืบหน้าในครั้งถัดไป

สัปดาห์ที่ 2 : เริ่มต้นการทำงานตามแผนที่ได้วางไว้

สัปดาห์ที่ 3 : ดำเนินงานตามแผนและเริ่มพูดคุยกับสมาชิกในส่วนของงานที่ต้องทำร่วมกัน

ประชุมครั้งที่ 2 วันที่ 04.10.2025

- อธิบายความคืบหน้าจากการดำเนินงานในขั้นต้น
- แก้ไขปัญหาของ Github
- ปรับแผนของสมาชิกบางคนให้ช่วยกันทำงานอย่างพร้อมเพียงกัน เพื่อความรวดเร็วและถูกต้อง
- นัดหมายส่งความคืบหน้าครั้งถัดไปในวันศุกร์ที่ 10.10 และนัดหมายการประชุมครั้งที่ 3 เสาร์ 11.10
- สามารถเริ่มทำรายงานได้แล้ว

สัปดาห์ที่ 4 : ดำเนินงานให้เสร็จตามเป้าหมาย เตรียมสรุปการทำงานทั้งหมด จัดทำรายงาน

ประชุมครั้งที่ 3 วันที่ 11.10.2025

- มีการปรับเปลี่ยนโครงสร้างภาษาจาก py เป็น cpp กับ java ทีม assembler เลือกที่จะทำ c++ และ ทีม simulator ใช้โครงสร้างเป็น java
- อัปเดตโค้ดลงใน GitHub ให้เรียบร้อยพร้อมทั้งเขียนคอมเม้นให้ละเอียดและเข้าใจง่ายลงในโค้ดด้วย ในส่วนที่จำเป็น
- เข้าไปเขียนรายงานในส่วนของตัวเอง ตามที่ได้ดำเนินการ
- หากโปรแกรมเสร็จแล้วให้ทดลองรันแล้วนำโค้ดพร้อมผลลัพธ์ส่ง และเขียนอธิบายตามสมควร
- มีสมาชิกบางคนทำเสร็จแล้ว สมาชิกคนอื่นๆ ที่ต้องการความช่วยเหลือสามารถติดต่อได้ตามต้องการ

ตารางเวลาทำงานของนักศึกษาแต่ละคนในกลุ่ม

วันที่	ผู้รับผิดชอบ	การทำงาน
สัปดาห์ที่ 1 19-26/09	นายณัฐพร	<p>ศึกษาหน้าที่หลัก</p> <ul style="list-style-type: none"> - ศึกษาการทำงานของ Simulator และหาทางออกแบบ simulator จากไฟล์ที่นำมาในตัวอย่าง วางแผนการทำงาน - สร้างออกแบบการทำงานของ Simulator
	นางสาวโยษิตา	<p>ศึกษาหน้าที่หลัก</p> <ul style="list-style-type: none"> - รับค่า parser(ค่า IR) + symbol table และนำมาแปลงเป็น 32-bit machine code - encode ทุก instruction ให้เป็น binary ตาม format ที่กำหนด (R/I/J/O type) <p>ศึกษา instruction format (R/I/J/O)</p> <p>วางแผนการทำงาน</p> <ul style="list-style-type: none"> - Opcode Mapping - Field Resolution - Instruction Encoding - Error Handling - Output <p>ต้องการ</p> <ul style="list-style-type: none"> - ไฟล์ผลลัพธ์ machine code (ฐาน 10, 1 บรรทัด/คำสั่ง)

		<ul style="list-style-type: none"> - Error Status
นางสาวรัชชาพร		<p>ศึกษาหน้าที่หลัก</p> <ul style="list-style-type: none"> - รับค่า อ่านและแปล assembly code ที่รับเข้ามา (Parsing) เป็นการเตรียมข้อมูลสำหรับเอาไปสร้าง machine code ต่อ ศึกษา format assembly + machine code <p>วางแผนการทำงาน</p> <ul style="list-style-type: none"> - Parser & Tokenizer - Symbol Table Management - Preprocessing <p>ต้องการ</p> <ul style="list-style-type: none"> - ค่า IR (Intermediate Representation) ที่มีข้อมูล address, label, instr, field0, field1, field2 - Symbol Table
นางสาววรรณคณา		<p>วางแผนการทำงานของกลุ่ม/แจกแจงหน้าที่ให้กับสมาชิก เตรียมทรัพย์กรณีที่จำเป็นต้องใช้ เช่น ช่องทางการทำงานร่วมกัน, GitHub repository, Document file, นัดหมายประชุม</p> <p>วางแผนการทำงาน</p> <ul style="list-style-type: none"> - วิเคราะห์โจทย์ + Instruction Format - ออกรอบแบบโปรแกรมที่ 1: คูณเลข 2 จำนวน - ออกรอบแบบโปรแกรมที่ 2: Recursive function <ul style="list-style-type: none"> • เลือกโปรแกรม factorial(n) <p>ต้องการ</p> <ul style="list-style-type: none"> - assembly/machine code - ตารางแปลงเป็น machine code (field และ opcode, reg, imm)
นายธีรช		<ul style="list-style-type: none"> - ศึกษาเนื้อหาเรื่องที่ต้องใช้ ทำความเข้าใจ Project ศึกษา file ตัวอย่าง simulator ที่อาจารย์ให้มา
	นายณธพร	<ul style="list-style-type: none"> - ทำ function อ่านไฟล์ กับ printstate - หาวิธีทำ execute switch จาก opcode ที่ให้มา
	นางสาวโยษิตา	<ul style="list-style-type: none"> - Opcode Mapping

สัปดาห์ที่ 2 27/09-3/10		<ul style="list-style-type: none"> - สัปดาห์ที่ 2 → ศึกษา instruction format (R/I/J/O) - กำหนด mapping: add=0, nand=1, lw=2, sw=3, beq=4, jalr=5, halt=6, noop=7, .fill=พิเศษ - Field Resolution <ul style="list-style-type: none"> - สัปดาห์ที่ 2 → เขียน findLabel (lookup address จาก symbol table) - เขียน getFieldValue (แปลง field เป็นตัวเลขหรือ offset ที่คำนวณแล้ว) - ทำงานได้ตามแผน
นางสาวรัชชาพร		<ul style="list-style-type: none"> - เตรียมไฟล์ Assembly ตัวอย่างสำหรับทดสอบ - เขียนฟังก์ชันเช็คค่าที่รับเข้ามาเบื้องต้น <ul style="list-style-type: none"> - ตรวจสอบว่าข้อมูลที่รับมาเป็นตัวเลข - ตรวจสอบแบบ label ให้ตรงตามเงื่อนไข - แยกคำในบรรทัดด้วยช่องว่าง - อ่านทุกบรรทัดในไฟล์, ตัด comment - เช็ค error ว่ามี label ซ้ำ, label ไม่ถูกต้อง (เกิน 6 ตัวอักษร, ไม่มีขั้นต้นด้วยตัวอักษร) รีบปรับ
นางสาววรรณคณา		<ul style="list-style-type: none"> - ศึกษาและทดลองทำ Assembly programs <ul style="list-style-type: none"> - โปรแกรมคูณเลขสองจำนวน - โปรแกรม factorial - ลองแปลง machine code แบบไม่ใช้ Assembler - ทดลองทำ opcode table เพื่อนำมาใช้กับ Assembly - เตรียมการทำงานร่วมกับคนอื่น
นายธีรช	นายธีรช	<ul style="list-style-type: none"> - แปลง file ตัวอย่างของอาจารย์เป็น Java - ทำ Function อ่านไฟล์, ConvertNum , และ Decode
	นางวนิชพร 4-10/10	<ul style="list-style-type: none"> - เปลี่ยนงานเป็นภาษา java และให้เพื่อนมาช่วยทำ flowchart - Instruction Encoding - สัปดาห์ที่ 3 → เขียน assemble

	<ul style="list-style-type: none"> ● R-type: $(opcode << 22) (regA << 19) (regB << 16) destReg$ ● I-type: $(opcode << 22) (regA << 19) (regB << 16) offsetField$ ● J-type: $(opcode << 22) (regA << 19) (regB << 16)$ ● O-type: $(opcode << 22)$ ● .fill → ตัวเลขหรือ address ตรง ๆ <ul style="list-style-type: none"> - Error Handling <ul style="list-style-type: none"> - ตรวจ error: <ul style="list-style-type: none"> ● opcode ไม่อยู่ในชุดที่กำหนด ● offsetField เกินช่วง -32768 ถึง 32767 ● ใช้ label ที่ไม่มีการ define <ul style="list-style-type: none"> - ทำงานได้ตามแผน
นางสาวรัชชาพร	<ul style="list-style-type: none"> - เขียนคลาส Parser <ul style="list-style-type: none"> - สร้าง symbols table ที่เก็บ label กับ address - ทดสอบ symbols table ที่ได้ - เขียนฟังก์ชันแยกการทำงานตามประเภทคำสั่ง ว่าแต่ละ type คำสั่ง ต้องใช้ข้อมูลอะไรบ้าง - ทดสอบว่าคำสั่งที่ได้และข้อมูลต่างๆ แยกตาม type ได้ถูกต้องตาม format - เขียนฟังก์ชันเขียนผลลัพธ์ลงในไฟล์ <ul style="list-style-type: none"> - Intermediate Representation ที่ได้ - Symbol Table - ส่งไฟล์ผลลัพธ์ให้ assembler นำไปแปลงเป็น machine code
นางสาววรรณคณา	<ul style="list-style-type: none"> - เริ่มจัดทำรายงาน - โค้ด asm รอนำไปทดสอบกับโปรแกรมของเพื่อนๆ - ติดตามงานและความคืบหน้า รวบรวมโค้ด

		<ul style="list-style-type: none"> - ช่วยเหลือสมาชิกกลุ่มคนอื่นๆ พูดคุยระหว่างโปรแกรมและ simulator
	นายธีรัช	<ul style="list-style-type: none"> — ช่วยเพื่อนทำ Flowchart และทำ executeFuntion ให้เสร็จ
สัปดาห์ที่ 4 11-19/10	นายณัฐพร	<ul style="list-style-type: none"> - ทำ Simulator ใน java พร้อมทั้ง test กับ Machine code - ทดสอบกับ Machine code ที่ได้จาก Assembly
	นางสาวโยษิตา	<ul style="list-style-type: none"> - Output <ul style="list-style-type: none"> ● สัปดาห์ที่ 4 → เขียน assembleProgram ● วนลูปทุก instruction → แปลงเป็น machine code (ฐาน 10) ● แสดงผลไฟล์ output (เลขฐาน 10 บรรทัดละ 1 ค่า) ● ถ้าเจอ error → exit(1) ● ถ้าสำเร็จ → exit(0) - รวมโค้ด (main + parser.cpp + assembler.cpp) → ทดสอบกับ test cases - นำข้อมูลที่ได้จาก parser ไปสร้าง machine code - ทำงานได้ตามแผน - ทำการรายงาน
	นางสาวรัชชพร	<ul style="list-style-type: none"> - ทดสอบกับ assembly หลายรูปแบบ - ทำ test case เช็ค error ว่าโค้ดที่เขียนไปครอบคลุมทุกรณีที่คาดว่าจะเกิด error รึยัง - ทำการรายงาน
	นางสาววรรณคณา	<ul style="list-style-type: none"> - รวบรวมโค้ดและทดสอบโปรแกรม Assembly และ Simulator - ตรวจสอบโปรแกรมว่ามีข้อผิดพลาดหรือจุดที่ควรแก้ไข - เก็บรายละเอียดรายงาน - เตรียมแบ่งการนำเสนอ
	นายธีรัช	<ul style="list-style-type: none"> — ทำ Simulator ให้เสร็จ — Test จาก Code ของอาจารย์ และ Machine Code จาก Assembly เพื่อน

โค้ดในส่วนต่างๆ

1. Assembler

1.1 Parser

parser.h

```

1 #ifndef PARSER_H
2 #define PARSE_H
3
4 #include <string>
5 #include <vector>
6
7 using namespace std;
8
9 struct Label {
10     string name;
11     int address;
12 };
13
14 struct IRLine {
15     int address;
16     string rawLabel;
17     string instr;
18     string f0, f1, f2;
19
20     bool isFill = false;
21     bool hasError = false;
22     string errorMsg;
23
24     int regA = 0;
25     int regB = 0;
26     int dest = 0;
27     int offset16 = 0;
28     int fillValue = 0;
29 };
30
31 class Parser {
32 public:
33     Parser();
34
35     void parseFile(const string &filename, bool countBlankLines = false, const string &commentChars = "#;");
36
37     const vector<IRLine>& getIR() const;
38     const vector<Label>& getSymbols() const;
39
40     void writeIRFile(const string &outname = "program.ir") const;
41     void writeSymbolsFile(const string &outname = "program_symbols.txt") const;
42
43 private:
44     vector<string> rawLines;
45     vector<IRLine> ir;
46     vector<Label> symbols;
47
48     void readAllLines(const std::string &filename, const std::string &commentChars);
49     void pass1_buildSymbolTable(bool countBlankLines);
50     void pass2_resolve(bool countBlankLines);
51 };
52
53
54 #endif

```

parser.cpp บรรทัดที่ 1-58

```

1 // Compile : g++ -std=c++17 parser.cpp -o parser
2 // Run : .\parser
3
4 #include "parser.h"
5 #include <fstream>
6 #include <sstream>
7 #include <unordered_set>
8 #include <unordered_map>
9 #include <cctype>
10 #include <stdexcept>
11 #include <iostream>
12 #include <limits>
13 #include <iomanip>
14
15 using namespace std;
16
17 // set ของ mnemonic ที่เก็บ opcode(สื่อค่าสั่ง) ที่ valid
18 static const unordered_set<string> MNEMONICS = {
19     "add", "nand", "lw", "sw", "beq", "jalr", "halt", "noop", ".fill"
20 };
21
22 // เช็คว่าค่าที่รับเข้ามาเป็นตัวเลขนัย
23 bool isNumber(const string &s) {
24     if (s.empty()) return false;
25     size_t i = 0;
26     if (s[0] == '+' || s[0] == '-') i = 1; // รองรับทั้ง + และ -
27     if (i == s.size()) return false;
28     for (; i < s.size(); ++i)
29         if (!isdigit((unsigned char)s[i])) return false;
30     return true;
31 }
32
33 // เช็คว่ารูปแบบของ label ถูกต้องตามเงื่อนไขนัย
34 static bool validLabelName(const string &s) {
35     if (s.empty()) return false;
36     if (!isalpha((unsigned char)s[0])) return false; // ต้องชื่นต้นด้วยตัวอักษร
37     if (s.size() > 6) return false; // ความยาวไม่เกิน 6 ตัวอักษร
38     for (char c: s)
39         if (!isalnum((unsigned char)c)) return false; // ตัวอักษรที่เหลือเป็นตัวอักษรหรือตัวเลขก็ได้
40     return true;
41 }
42
43 // สำหรับ error ให้แสดงใน terminal และ exit
44 void dieError(const string &msg) {
45     cerr << "Error: " << msg << "\n";
46     exit(1);
47 }
48
49 // แยก string เป็น tokens ตาม whitespace
50 static vector<string> tokenize_ws(const string &s) {
51     vector<string> toks;
52     istringstream iss(s);
53     string t;
54     while (iss >> t) toks.push_back(t);
55     return toks;
56 }
57
58

```

parser.cpp บรรทัดที่ 59-87

```
1
2 Parser::Parser() {}
3
4 // อ่านไฟล์ทั้งหมดและตัด comment ออก
5 void Parser::readAllLines(const string &filename, const string &commentChars) {
6     rawLines.clear();
7     ifstream ifs(filename);
8     if (!ifs.is_open()) throw runtime_error("cannot open input file: " + filename);
9     string line;
10    while (getline(ifs, line)) {
11
12        // ตัด comment ออกสำหรับ commentChars อยู่
13        if (!commentChars.empty()) {
14            size_t pos = string::npos;
15            for (char cc : commentChars) {
16                size_t p = line.find(cc);
17                if (p != string::npos) {
18                    if (pos == string::npos) pos = p;
19                    else pos = min(pos, p);
20                }
21            }
22            if (pos != string::npos) line = line.substr(0, pos);
23        }
24
25        // เก็บ raw line (อาจเป็น blank line หรือ whitespace)
26        rawLines.push_back(line);
27    }
28    ifs.close();
29 }
30 }
```

parser.cpp บรรทัดที่ 88-156

```

1 // pass2: resolve operand, registers, offsets
2 void Parser::pass2_resolve(bool countBlankLines) {
3
4     // ส่ง map ของ label กับ address เพื่อ lookup เข้าชื่น
5     unordered_map<string,int> labelToAddr;
6     for (const auto &lab : symbols) labelToAddr[lab.name] = lab.address;
7
8     for (size_t i = 0; i < ir.size(); ++i) {
9         IRLine &L = ir[i];
10        string m = L.instr;
11
12        // ต้องค่าสั่งมีแต่ label และมีคำสั่ง (ไม่ควรเกิดชื่น)
13        if (m.empty()) {
14            throw runtime_error("missing instruction at address " + to_string(L.address));
15        }
16
17        // เช็คว่า opcode(คำสั่ง) ถูกต้องนี้ย
18        if (MNEMONICS.find(m) == MNEMONICS.end()) {
19            throw runtime_error("invalid opcode '" + m + "' at address " + to_string(L.address));
20        }
21
22        // เมกค่าใน memory ให้คำตัวเดียวหรือตัวเดียว label ลงใน memory
23        // .fill ให้ได้ค่าเดียวหรือตัวเดียว label ลงใน memory
24        if (m == ".fill") {
25            L.isFill = true;
26            // ให้ field1 (f0) ตามหลัง
27            if (L.f0.empty()) {
28                throw runtime_error(".fill without operand at address " + to_string(L.address));
29            // ถ้าเป็นตัวเลข เก็บลงใน mem
30            if (isNumber(L.f0)) {
31                long long v = stoll(L.f0);
32                L.fillValue = static_cast<int>(v);
33            // ถ้าเป็นชื่อ label ให้หาน address ของ label นั้นๆ
34            } else {
35                // ถ้าไม่ใน list ของ label ที่เคยเห็นแสดงว่า error
36                if (labelToAddr.find(L.f0) == labelToAddr.end())
37                    throw runtime_error("undefined label '" + L.f0 + "' used in .fill at address " + to_string(L.address));
38                L.fillValue = labelToAddr[L.f0];
39            }
40            continue;
41        }
42
43        // R-type: add, nand
44        if (m == "add" || m == "nand") {
45            // ให้เช็คว่ามีค่า field 3 อัน
46            if (L.f0.empty() || L.f1.empty() || L.f2.empty())
47                throw runtime_error("R-type instruction missing field at address " + to_string(L.address));
48            // ต้องทุก field เป็นแบบนี้
49            if (!isNumber(L.f0) || !isNumber(L.f1) || !isNumber(L.f2))
50                throw runtime_error("R-type registers must be numeric at address " + to_string(L.address));
51            // แปลงเป็น int และซัก reg ต้องในช่วง 0-7 นี้ย
52            L.regA = stoi(L.f0);
53            L.regB = stoi(L.f1);
54            L.dest = stoi(L.f2);
55            if (L.regA < 0 || L.regA > 7 || L.regB < 0 || L.regB > 7 || L.dest < 0 || L.dest > 7)
56                throw runtime_error("register out of range (0..7) at address " + to_string(L.address));
57            continue;
58        }
59

```

parser.cpp บรรทัดที่ 157-215

parser.cpp บรรทัดที่ 216-298

```

1 // I-type: lw, sw
2 if (m == "lw" || m == "sw") {
3     // ໄສເຊື່ອຕາມຫຼັກ field ມີ
4     if (L.f0.empty() || L.f1.empty() || L.f2.empty())
5         throw runtime_error("lw/sw missing field at address " + to_string(L.address));
6     // ເຊື່ວ່າ regA, regB ເປັນລວມເລີດ
7     if (!isNumber(L.f0) || !isNumber(L.f1))
8         throw runtime_error("lw/sw regA/regB must be numeric at address " + to_string(L.address));
9     // ນຳລົມເປັນ int ແລະເຊື່ວ່າ reg ວ່າຍໃຫ້ຈຳກູດ 0-7 ມີ
10    L.regA = stoi(L.f0);
11    L.regB = stoi(L.f1);
12    if (L.regA < 0 || L.regA > 7 || L.regB < 0 || L.regB > 7)
13        throw runtime_error("register out of range (0..7) at address " + to_string(L.address));
14    // ດໍາ offset ພຶບເຊື່ອລົມຄະແລ້ວເຂົ້າກຸຍໃຫ້ຈຳກູດ 16 ມີ
15    if (isNumber(L.f2)) {
16        long long v = stoll(L.f2);
17        if (v < -32768 || v > 32767)
18            throw runtime_error("offset out of 16-bit range for lw/sw at address " + to_string(L.address));
19        L.offset16 = static_cast<int>(v);
20    // ດໍາ offset ພຶບ label ລົມທຳນົມ label ຕ່າງ addr ແລະເຊື່ວ່າ addr ວ່າຍໃຫ້ຈຳກູດ 16 ມີ ມີ
21    } else {
22        if (labelToAddr.find(L.f2) == labelToAddr.end())
23            throw runtime_error("undefined label '" + L.f2 + "' used in lw/sw at address " + to_string(L.address));
24        int addrLabel = labelToAddr[L.f2];
25        if (addrLabel < -32768 || addrLabel > 32767)
26            throw runtime_error("label address out of 16-bit range for lw/sw at address " + to_string(L.address));
27        L.offset16 = addrLabel;
28    }
29    continue;
30 }

31 // Branch: beq
32 if (m == "beq") {
33     // ໄສເຊື່ອຕາມຫຼັກດອງເຫຼືອນ I-type
34     if (L.f0.empty() || L.f1.empty() || L.f2.empty())
35         throw runtime_error("beq missing field at address " + to_string(L.address));
36     if (!isNumber(L.f0) || !isNumber(L.f1)) throw runtime_error("beq regA/regB must be numeric at address " + to_string(L.address));
37     L.regA = stoi(L.f0);
38     L.regB = stoi(L.f1);
39     if (L.regA < 0 || L.regA > 7 || L.regB < 0 || L.regB > 7)
40         throw runtime_error("register out of range (0..7) at address " + to_string(L.address));
41     if (isNumber(L.f2)) {
42         long long v = stoll(L.f2);
43         if (v < -32768 || v > 32767)
44             throw runtime_error("beq numeric offset out of 16-bit range at address " + to_string(L.address));
45         L.offset16 = static_cast<int>(v);
46     // ດໍາ offset ພຶບ label ລົມທຳນົມ offset ພຶບ relative ຕ່າງ address(label) - (address(ສໍາງໆ) + 1)
47    } else {
48        if (labelToAddr.find(L.f2) == labelToAddr.end())
49            throw runtime_error("undefined label '" + L.f2 + "' used in beq at address " + to_string(L.address));
50        int addrLabel = labelToAddr[L.f2];
51        long long offset = static_cast<long long>(addrLabel) - (static_cast<long long>(L.address) + 1LL);
52        if (offset < -32768 || offset > 32767)
53            throw runtime_error("beq offset out of range for label '" + L.f2 + "' at address " + to_string(L.address));
54        L.offset16 = static_cast<int>(offset);
55    }
56    continue;
57 }

58 // J-type: jalr
59 if (m == "jalr") {
60     // ເຊື່ວ່າ field regA regB ຫຼືມີ I-type
61     if (L.f0.empty() || L.f1.empty())
62         throw runtime_error("jalr missing field at address " + to_string(L.address));
63     if (!isNumber(L.f0) || !isNumber(L.f1))
64         throw runtime_error("jalr registers must be numeric at address " + to_string(L.address));
65     L.regA = stoi(L.f0);
66     L.regB = stoi(L.f1);
67     if (L.regA < 0 || L.regA > 7 || L.regB < 0 || L.regB > 7)
68         throw runtime_error("register out of range (0..7) at address " + to_string(L.address));
69     continue;
70 }

71 // O-type: halt, noop ຫຼືມີ field
72 if (m == "halt" || m == "noop") {
73     continue;
74 }

75 // instruction ທີ່ບໍ່ໄດ້ຮັບຮັດ
76 throw runtime_error("unhandled instruction '" + m + "' at address " + to_string(L.address));
77 }

78 
```

parser.cpp บรรทัดที่ 299-360

```

1 void Parser::parseFile(const string &filename, bool countBlankLines, const string &commentChars) {
2     readAllLines(filename, commentChars);
3     pass1_buildSymbolTable(countBlankLines);
4     pass2_resolve(countBlankLines);
5 }
6
7 const vector<IRLine>& Parser::getIR() const { return ir; }
8 const vector<Label>& Parser::getSymbols() const { return symbols; }
9
10 void Parser::writeIRFile(const string &outname) const {
11     ofstream ofs(outname);
12     if (!ofs.is_open()) throw runtime_error("cannot write IR file: " + outname);
13
14     ofs << left
15         << setw(8) << "addr"
16         << setw(8) << "label"
17         << setw(8) << "instr"
18         << setw(8) << "field0"
19         << setw(8) << "field1"
20         << setw(10) << "field2"
21         << setw(8) << "regA"
22         << setw(8) << "regB"
23         << setw(8) << "dest"
24         << setw(10) << "offset16"
25         << setw(10) << "fillValue"
26         << "\n";
27
28     for (const auto &L : ir) {
29         ofs << setw(8) << L.address
30             << setw(8) << L.rawLabel
31             << setw(8) << L.instr
32             << setw(8) << L.f0
33             << setw(8) << L.f1
34             << setw(10) << L.f2
35             << setw(8) << L.regA
36             << setw(8) << L.regB
37             << setw(8) << L.dest
38             << setw(10) << L.offset16
39             << setw(10) << L.fillValue
40             << "\n";
41     }
42     ofs.close();
43 }
44
45
46 void Parser::writeSymbolsFile(const string &outname) const {
47     ofstream ofs(outname);
48     if (!ofs.is_open()) throw runtime_error("cannot write symbols file: " + outname);
49
50     ofs << left
51         << setw(10) << "LabelName"
52         << setw(10) << "Address"
53         << "\n";
54
55     for (const auto &p : symbols) {
56         ofs << setw(10) << p.name
57             << setw(10) << p.address
58             << "\n";
59     }
60     ofs.close();
61 }
62

```

parser.cpp บรรทัดที่ 361-398

```
1 int main() {
2     string inputFile = "test(assembly-language).asm"; // ไฟล์ assembly สำหรับทดสอบ
3
4     Parser parser;
5     parser.parseFile(inputFile); // เรียกฟังก์ชันหลักเพื่ออ่านและแยกข้อมูล
6
7     cout << "\nParsing...";
8     cout << "\n-----\n";
9
10    cout << "Symbol Table:\n";
11    for (auto &sym : parser.getSymbols()) {
12        cout << " " << setw(10) << left << sym.name
13            << "-> Address: " << sym.address << endl;
14    }
15
16    cout << "\nParsed Instructions:\n";
17    auto insts = parser.getIR();
18    for (auto &inst : insts) {
19        cout << " " << "Address " << setw(3) << inst.address << " | "
20            << setw(6) << left << inst.rawLabel << " | "
21            << setw(6) << left << inst.instr << " | "
22            << setw(6) << left << inst.f0 << " | "
23            << setw(6) << left << inst.f1 << " | "
24            << setw(6) << left << inst.f2
25            << endl;
26    }
27
28    string baseName = "program";
29
30    parser.writeIRFile(baseName + ".ir"); // สร้างไฟล์ IR สำหรับ assembler
31    parser.writeSymbolsFile(baseName + "_symbols.txt"); // สร้างไฟล์ symbol table
32
33    cout << "\n-----\n";
34    cout << "Parse success!\n";
35    cout << "Output written to: " << baseName << ".ir and " << baseName << "_symbols.txt\n";
36
37    return 0;
38 }
39
```

1.2 Assembler

assembler.cpp บรรทัดที่ 1-55

```

1 // assembler.cpp
2
3 #include <iostream>      // หิมพ์ชื่อความ/ผลลัพธ์ที่นุกราน
4 #include <string>        // std::string
5 #include <vector>         // std::vector
6 #include <unordered_map> // symbol table: ชื่อ label -> address
7 #include <iomanip>        // setw สำหรับจัด format และผล
8 #include <cctype>         // isdigit/isxdigit ใช้ช่วยตรวจสอบ
9 #include <stdexcept>     // รองรับ exception ตอนแปลงเลข
10 #include <cstdint>        // int32_t/uint32_t ให้กับมิติ 32 มิติ
11 #include <fstream>         // อ่าน/เขียนไฟล์
12 #include <sstream>         // istreamstream ใช้ parse บรรทัด
13 #include <algorithm>       // find_if ใช้ trim ด้านขวา
14
15 using namespace std;
16
17 // ----- ยุทธีลพันธุ์ฐานเล็ก ๆ (คุณภาพชีวิต) -----
18
19 // rtrim: ตัดช่องว่างขวาสุดของสตริง (กันปัญหาช่องว่างເຈືອປັນຈາກໄຟສຄອລັນນົດທີ່)
20 static inline string rtrim(string s) {
21     auto it = find_if(s.rbegin(), s.rend(), [](unsigned char ch){return !isspace(ch);});
22     s.erase(it.base(), s.end());
23     return s;
24 }
25
26 // safeSubstr: substr แบบปลอดภัย (ถ้า pos เกินความยาว ให้เดิน "" เพื่อกัน out_of_range)
27 static inline string safeSubstr(const string& s, size_t pos, size_t len) {
28     if (pos >= s.size()) return "";
29     len = min(len, s.size() - pos);
30     return s.substr(pos, len);
31 }
32
33 // tryParseInt: แปลงสตริงเป็น int (รองรับฐาน 10/16 แบบ auto) ถ้าผังให้ false
34 static bool tryParseInt(const string& s, int& out) {
35     try {
36         size_t p=0; long long v = stoll(rtrim(s), &p, 0); // base 0 => auto (0x.. เป็น hex)
37         if (p != rtrim(s).size()) return false;           // มีขยะตามท้าย → ໄມເຂົາ
38         if (v < INT32_MIN || v > INT32_MAX) return false; // เกินช่วง int 32
39         out = (int)v;
40         return true;
41     } catch (...) { return false; }
42 }
43
44 // ----- Opcodes และ mapping -----
45 // หมายเหตุ: .fill เป็น "directive" ไม่ใช่ instruction จึง set เป็น -1
46 enum class Op : int {
47     ADD=0, NAND=1, LW=2, SW=3, BEQ=4, JALR=5, HALT=6, NOOP=7, FILL=-1
48 };
49 static const unordered_map<string, Op> OPCODE_MAP = {
50     {"add", Op::ADD}, {"nand", Op::NAND},
51     {"lw", Op::LW}, {"sw", Op::SW},
52     {"beq", Op::BEQ}, {"jalr", Op::JALR},
53     {"halt", Op::HALT}, {"noop", Op::NOOP},
54     {".fill", Op::FILL}
55 };

```

assembler.cpp บรรทัดที่ 57-88

```

1 // ----- Bit layout ของคำสั่ง 32 บิต -----
2 // เราใช้การ shift มิตเข้าตำแหน่งที่สเปกกำหนด:
3 // [ opcode(3) | regA(3) | regB(3) | (ที่เหลือ 16 บิต/3 บิตขั้นกับชนิด) ]
4 constexpr int OPCODE_SHIFT = 22;
5 constexpr int REGA_SHIFT = 19;
6 constexpr int REGB_SHIFT = 16;
7
8 // พิ่งกซัน pack ปิตสำหรับแต่ละฟอร์แมต (ลด duplicate ໂຄດ)
9 inline uint32_t packR(int opcode, int rA, int rB, int dest) {
10    // R-type: ช่องท้ายสุดໃช้เพียง 3 บิตสำหรับ destReg
11    return (uint32_t(opcode) << OPCODE_SHIFT)
12        | (uint32_t(rA)      << REGA_SHIFT)
13        | (uint32_t(rB)      << REGB_SHIFT)
14        | (uint32_t(dest) & 0x7u);
15 }
16 inline uint32_t packI(int opcode, int rA, int rB, int offset16) {
17    // I-type: 16 บิตท้ายໃช้เก็บ offset แบบ two's complement
18    return (uint32_t(opcode) << OPCODE_SHIFT)
19        | (uint32_t(rA)      << REGA_SHIFT)
20        | (uint32_t(rB)      << REGB_SHIFT)
21        | (uint32_t(offset16) & 0xFFFFu); // mask 16 บิตล่างให้ชัดเจน
22 }
23 inline uint32_t packJ(int opcode, int rA, int rB) {
24    // J-type: ໃช้แค่ opcode + regA + regB, ที่เหลือ (16 บิต) เป็น 0
25    return (uint32_t(opcode) << OPCODE_SHIFT)
26        | (uint32_t(rA)      << REGA_SHIFT)
27        | (uint32_t(rB)      << REGB_SHIFT);
28 }
29 inline uint32_t packO(int opcode) {
30    // O-type: ໃช้เฉพาะ opcode, ที่เหลือทั้งหมดเป็น 0
31    return (uint32_t(opcode) << OPCODE_SHIFT);
32 }
33

```

assembler.cpp บรรทัดที่ 90-137

```

1 // ----- โครงสร้าง error ที่เราจะรายงาน -----
2 enum class AsmError {
3     NONE = 0,
4     UNKNOWN_OPCODE, // คำสั่งไม่อยู่ใน mapping
5     UNDEFINED_LABEL, // อ้าง label ที่ไม่มีใน symbol table
6     OFFSET_OUT_OF_RANGE, // offset 16-bit เกินกว่าง signed (-32768..32767)
7     BAD_IMMEDIATE, // immediate/number แปลงไม่ได้หรือรูปแบบผิด
8     BAD_REGISTER // เรจิสเตอร์หายไป หรืออยู่นอกช่วง 0..7 (3 มิต)
9 };
10 struct ErrInfo {
11     AsmError code{AsmError::NONE};
12     string msg;
13 };
14
15 inline bool okReg(int r){ return 0<=r && r<=7; } // reg 3 มิต: 0..7 เท่านั้น
16
17 // ----- Helper: mapping/parse/symbol/offset -----
18
19 // แปลง mnemonic → opcode (int) (.fill = -1)
20 ErrInfo toOpcode(const string& mnemonic, int& outOpcode) {
21     auto it = OPCODE_MAP.find(mnemonic);
22     if (it == OPCODE_MAP.end())
23         return {AsmError::UNKNOWN_OPCODE, "unknown opcode: " + mnemonic};
24     if (it->second == Op::FILL) { outOpcode = -1; return {AsmError::NONE,""}; }
25     outOpcode = static_cast<int>(it->second);
26     return {AsmError::NONE,""};
27 }
28
29 // helper เช็คว่า x อยู่ในช่วง signed 16-bit หรือไม่
30 inline bool inSigned16(long long x){ return -32768<=x && x<=32767; }
31
32 // ตรวจสอบ string ว่าเป็น “เลข” นัย (รองรับ +/-, 0x..)
33 bool looksNumber(const string& s){
34     if (s.empty()) return false;
35     size_t i=0; if (s[0]=='+'||s[0]=='-') i=1;
36     if (i>s.size()) return false;
37
38     // รูปแบบ 0x.. (hex)
39     if (i+1<s.size() && s[i]=='0' && (s[i+1]=='x'||s[i+1]=='X')){
40         i+=2; if (i>s.size()) return false;
41         for(; i<s.size(); ++i) if(!isxdigit((unsigned char)s[i])) return false;
42         return true;
43     }
44
45     // รูปแบบตัวเลขฐาน 10
46     for(; i<s.size(); ++i) if(!isdigit((unsigned char)s[i])) return false;
47     return true;
48 }

```

assembler.cpp บรรทัดที่ 139-192

```

1 // แปลงสติง → long long (ฐานอัตโนมัติ) พัฒนาระยะตามท้าย
2 ErrInfo parseNumber(const string& token, long long& outVal){
3     string t = rtrim(token);
4     if (!looksNumber(t))
5         return {AsmError::BAD_IMMEDIATE, "not a valid number: " + t};
6     try{
7         size_t pos=0;
8         outVal = stoll(t, &pos, 0); // base 0: 0x..=hex, อื่น ๆ=dec
9         if (pos!=t.size())
10            return {AsmError::BAD_IMMEDIATE, "trailing junk: " + t};
11         return {AsmError::NONE,""};
12     }catch(...){
13         return {AsmError::BAD_IMMEDIATE, "cannot parse: " + t};
14     }
15 }
16
17 // หา address ของ label จาก symbol table (ไม่เจอ → error)
18 ErrInfo findLabel(const unordered_map<string,int>& symtab,
19                     const string& label, int& outAddr){
20     auto it = symtab.find(label);
21     if (it==symtab.end()) return {AsmError::UNDEFINED_LABEL, "undefined label: " + label};
22     outAddr = it->second; return {AsmError::NONE,""};
23 }
24
25 // แปลงค่า field (ซึ่งอาจเป็นตัวเลขหรือ label) ให้อยู่ในรูป int พัฒนาระยะใน:
26 // - asOffset16=true → ต้องเป็น 16 บิต (ใช้กับ I-type)
27 // - isBranch=true → beq: offset = labelAddr - (PC+1) (relative)
28 // - isBranch=false → lw/sw/.fill: ใช้ “address ตรง ๆ” ถ้าเป็น label
29 ErrInfo getFieldValue(const unordered_map<string,int>& symtab,
30                       const string& token, int currentPC,
31                       bool asOffset16, bool isBranch, int& outVal){
32     long long val=0;
33     string t = rtrim(token);
34
35     if (looksNumber(t)){
36         // เป็นตัวเลขตรง ๆ → แปลงเป็น long long
37         ErrInfo e = parseNumber(t,val);
38         if(e.code!=AsmError::NONE) return e;
39     }else{
40         // เป็น label → หา address จาก symbol table
41         int addr=0; ErrInfo e = findLabel(symtab, t, addr);
42         if (e.code!=AsmError::NONE) return e;
43         // ถ้าเป็น beq: ใช้ offset แบบ relative เป้าหมาย - (PC+1)
44         if (isBranch) val = (long long)addr - (long long)(currentPC+1);
45         else           val = addr;
46     }
47
48     // ถ้าเป็น offset 16 บิต ต้องไม่เกินช่วง signed 16-bit
49     if (asOffset16 && !isSigned16(val))
50         return {AsmError::OFFSET_OUT_OF_RANGE, "offset out of 16-bit range: " + to_string(val)};
51
52     outVal = (int)val;
53     return {AsmError::NONE,""};
54 }

```

assembler.cpp บรรทัดที่ 194-221

```
1 // ----- โครงสร้าง IR (รับจาก Part A) -----
2 // หมายเหตุ: Part A จะ parse ข้อความ assembly และส่ง IR ให้เรา (Part B)
3 // - mnemonic: ชื่อคำสั่ง เช่น add/lw/beq/.../.fill
4 // - regA/regB/dest: ตัวแหนงเรจิสเตอร์ที่เกี่ยวข้อง (R/J type ใช้ dest)
5 // - fieldToken: ค่าฟิล์ต (offset/label) ส่าหรับ I-type และค่าใน .fill
6 // - pc: address ของคำสั่งบรรทัดนั้น (เริ่มนับจาก 0)
7 struct IRInstr {
8     string mnemonic;
9     int    regA{-1}, regB{-1}, dest{-1};
10    string fieldToken;
11    int    pc{-1};
12 };
13
14 // ----- helper: ตรวจสอบ register ให้ครบและอยู่ในช่วง -----
15 static ErrInfo needReg(int reg, const string& name){
16     if (reg < 0)
17         return {AsmError::BAD_REGISTER, "missing "+name}; // ไม่ได้ใส่มา
18     if (!okReg(reg))
19         return {AsmError::BAD_REGISTER, "bad "+name+": "+to_string(reg)}; // เกินช่วง 0..7
20     return {AsmError::NONE, ""};
21 }
22
23 // ----- เข้ารหัสคำสั่งเดียว (Week 3) -----
24 // ผลลัพธ์: word = machine code (32-bit) ในรูป int32_t ส่าหรับพิมพ์เป็นฐาน 10
25 struct EncodeResult {
26     ErrInfo  error;
27     int32_t  word{0};
28 };
```

assembler.cpp บรรทัดที่ 223-289

```

1 EncodeResult assembleOne(const unordered_map<string,int>& symtab, const IRInstr& ir){
2     EncodeResult r;
3     int opcode=-1;
4
5     // 1) เมื่อ mnemonic -> opcode (ตัว .fill จะให้ opcode=-1)
6     r.error = toOpcode(ir.mnemonic, opcode);
7     if (r.error.code!=AsmError::NONE) return r;
8
9     // 2) .fill: ไม่ได้ค่าสั่ง ให้ตั้งค่าด้วย/addr ตรง ๆ
10    if (opcode < 0){
11        int val=0;
12        ErrInfo e = getFieldValue(symtab, ir.fieldToken, ir.pc, false, false, val);
13        if (e.code!=AsmError::NONE){ r.error=e; return r; }
14        r.word = val;
15        return r;
16    }
17
18    // 3) เลือก pack ตามชนิดของ opcode (R/I/J/O)
19    switch (static_cast<Op>(opcode)){
20        case Op::ADD:
21        case Op::NAND: {
22            // R-type: add/nand regA regB dest
23            ErrInfo e = needReg(ir.regA, "regA"); if (e.code!=AsmError::NONE){ r.error=e; return r; }
24            e = needReg(ir.regB, "regB");           if (e.code!=AsmError::NONE){ r.error=e; return r; }
25            e = needReg(ir.dest, "destReg");       if (e.code!=AsmError::NONE){ r.error=e; return r; }
26            r.word = (int32_t)packR(opcode, ir.regA, ir.regB, ir.dest);
27            return r;
28        }
29        case Op::LW:
30        case Op::SW: {
31            // I-type: lw/sw regA regB offset(16 มีต signed)
32            ErrInfo e = needReg(ir.regA, "regA"); if (e.code!=AsmError::NONE){ r.error=e; return r; }
33            e = needReg(ir.regB, "regB");           if (e.code!=AsmError::NONE){ r.error=e; return r; }
34            int off=0;
35            e = getFieldValue(symtab, ir.fieldToken, ir.pc, true, false, off);
36            if (e.code!=AsmError::NONE){ r.error=e; return r; }
37            r.word = (int32_t)packI(opcode, ir.regA, ir.regB, off);
38            return r;
39        }
40        case Op::BEQ: {
41            // I-type: beq regA regB offset(label) → relative = labelAddr - (PC+1)
42            ErrInfo e = needReg(ir.regA, "regA"); if (e.code!=AsmError::NONE){ r.error=e; return r; }
43            e = needReg(ir.regB, "regB");           if (e.code!=AsmError::NONE){ r.error=e; return r; }
44            int off=0;
45            e = getFieldValue(symtab, ir.fieldToken, ir.pc, true, true, off);
46            if (e.code!=AsmError::NONE){ r.error=e; return r; }
47            r.word = (int32_t)packI(opcode, ir.regA, ir.regB, off);
48            return r;
49        }
50        case Op::JALR: {
51            // J-type: jalr regA regB (ไม่ใช้ก่อ 16 มิตท้าย)
52            ErrInfo e = needReg(ir.regA, "regA"); if (e.code!=AsmError::NONE){ r.error=e; return r; }
53            e = needReg(ir.regB, "regB");           if (e.code!=AsmError::NONE){ r.error=e; return r; }
54            r.word = (int32_t)packJ(opcode, ir.regA, ir.regB);
55            return r;
56        }
57        case Op::HALT:
58        case Op::NOOP: {
59            // O-type: เลข値 opcode ที่เก็บเป็น 0
60            r.word = (int32_t)packO(opcode);
61            return r;
62        }
63        default:
64            r.error = {AsmError::UNKNOWN_OPCODE, "unhandled opcode"};
65            return r;
66    }
67 }

```

assembler.cpp บรรทัดที่ 291-328

```

1 // ----- Week 4: assembleProgram (fail-fast + เพิ่มไฟล์) -----
2 // และคิด:
3 // - วนทุก IR → แปลงเป็น machine code
4 // - ถ้าเจอ error บรรทัดใด → รายงานและหยุดทันที (ไม่เขียนไฟล์ต่อ) → return 1
5 // - ถ้าสำเร็จครบ → เพิ่ม "เขียนฐาน 10" ลงไฟล์ บรรทัดละ 1 ค่า → return 0
6 int assembleProgram(const unordered_map<string,int>& symtab,
7                     const vector<IRInstr>& irs,
8                     const string& outPath)
9 {
10    ofstream out(outPath);
11    if (!out) {
12        cerr << "ERROR: cannot open output file: " << outPath << "\n";
13        return 1;
14    }
15
16    for (const auto& ir : irs) {
17        EncodeResult res = assembleOne(symtab, ir);
18        if (res.error.code != AsmError::NONE) {
19            // รายงาน PC และตัวสั้งเพื่อ debug ครอสโค้ดง่าย
20            cerr << "ERROR at PC=" << ir.pc
21            << " (" << ir.mnemonic << ")": " << res.error.msg << "\n";
22            return 1; // หยุดทันทีตามสเปก
23        }
24        // เพิ่มเป็นเขียนฐาน 10 หนึ่งค่า/บรรทัดตามข้อกำหนด
25        out << res.word << "\n";
26        if (!out) {
27            cerr << "ERROR: write failed at PC=" << ir.pc << "\n";
28            return 1;
29        }
30
31        cout << "(address " << ir.pc << "): "
32           << res.word << " (hex 0x"
33           << hex << uppercase << setw(0)
34           << ((uint32_t)res.word & 0xFFFFFFFF)
35           << dec << ")\n";
36    }
37    return 0; // สำเร็จครบทุกบรรทัด
38 }

```

assembler.cpp บรรทัดที่ 330-352

```

1 // ----- โหลด Symbols & IR (ทบทวนต่อช่องว่าง/คลอสัมบ์) -----
2 //ไฟล์ symbolsTable.txt คาดว่าแต่ละบรรทัดเป็น: "<label> <addr>"
3 // บรรทัดแรกอาจเป็น header จึงข้ามไปหนึ่งบรรทัด
4 unordered_map<string,int> loadSymbolTable(const string& filename){
5     unordered_map<string,int> symbols;
6     ifstream fin(filename);
7     if(!fin){ cerr<<"ERROR: cannot open symbols file: "<<filename<<"\n"; return symbols; }
8
9     string line;
10    getline(fin, line); // ข้าม header ตัวนี้
11
12    while (getline(fin, line)) {
13        if (line.empty()) continue;
14        istringstream iss(line);
15        string label; int addr;
16        if (iss >> label >> addr)
17            symbols[label] = addr; // เก็บ label -> address
18        // ต้า parse ไฟล์ (format เพิ่ม) เรายังไม่ได้พิ้งทั้งไฟล์
19    }
20
21    cerr << "Loaded " << symbols.size() << " symbol(s) from: " << filename << "\n";
22    return symbols;
23 }

```

assembler.cpp บรรทัดที่ 354-412

```

1 // ฟอร์แมตไฟล์ program.ir ตามคอลัมน์คงที่ (fixed width) ที่ใช้อยู่ในที่ม:
2 // [0..7]=addr, [8..15]=label, [16..23]=instr, [24..31]=f0, [32..39]=f1, [40..47]=f2,
3 // [48..55]=regA, [56..63]=regB, [64..71]=dest, [72..79]=off, [80..]=fill
4 // หมายเหตุ:
5 // - เราก่อนแบน "ตัดคอลัมน์" + rtrim ปลายทาง เพื่อเก็บ token ที่สะอาด
6 // - IR ที่ต้องการสำหรับ Part B ดีด:
7 //      mnemonic, regA, regB, dest, fieldToken (กรณี I-type/.fill), pc
8 vector<IRInstr> loadIR(const string& filename){
9     vector<IRInstr> IR;
10    ifstream fin(filename);
11    if(!fin){ cerr<<"ERROR: cannot open IR file: "<<filename<<"\n"; return IR; }
12
13    string line;
14    getline(fin, line); // header
15
16    while (getline(fin, line)) {
17        if (line.empty()) continue;
18
19        IRInstr ir;
20
21        // 1) ลอกชิ้นฐานจากคอลัมน์คงที่
22        string addrStr = safeSubstr(line, 0, 8); ir.pc = tryParseInt(addrStr, ir.pc) ? ir.pc : 0;
23        string label   = rtrim(safeSubstr(line, 8, 8));
24        string instr   = rtrim(safeSubstr(line,16, 8));
25        string f0      = rtrim(safeSubstr(line,24, 8));
26        string f1      = rtrim(safeSubstr(line,32, 8));
27        string f2      = rtrim(safeSubstr(line,40, 8));
28
29        string regAstr = safeSubstr(line,48, 8);
30        string regBstr = safeSubstr(line,56, 8);
31        string destr   = safeSubstr(line,64, 8);
32        string offstr  = safeSubstr(line,72, 8);
33        string fillstr = safeSubstr(line,80, 16); // กันอนาคตสำคัญๆ
34
35        // 2) แปลงค่าที่เป็นตัวเลข ถ้าแปลงไม่ได้ให้เป็น -1 (หมายถึง "ไม่กรอก")
36        int tmp;
37        if (tryParseInt(regAstr, tmp)) ir.regA = tmp; else ir.regA = -1;
38        if (tryParseInt(regBstr, tmp)) ir.regB = tmp; else ir.regB = -1;
39        if (tryParseInt(destr,  tmp)) ir.dest = tmp; else ir.dest = -1;
40
41        // 3) ตัดสินใจ fieldToken ที่ Part B ต้องใช้:
42        // - .fill → ใช้ f0 เป็นหลัก (ตามฟอร์แมตที่วาง), ถ้าไม่มีค่อยลองดู fillstr
43        // - lw/sw/beq → ใช้ f2 (offset/label)
44        // - อื่น ๆ → เก็บไว้
45        if (instr == ".fill") {
46            ir.fieldToken = !f0.empty() ? f0 : rtrim(fillstr);
47        } else if (instr == "lw" || instr == "sw" || instr == "beq") {
48            ir.fieldToken = f2;
49        } else {
50            ir.fieldToken.clear();
51        }
52
53        ir.mnemonic = instr;
54        IR.push_back(ir);
55    }
56
57    cerr << "Loaded " << IR.size() << " IR row(s) from: " << filename << "\n";
58    return IR;
59 }

```

assembler.cpp บรรทัดที่ 414-451

```

1 // ----- main: ผูกทุกอย่างเข้าด้วยกัน + exit code -----
2 // การทำงานหลัก:
3 // - รับ파าร์มาส์จาก argv (หรือใช้ดีฟอลต์)
4 // - โหลดสัญลักษณ์+IR
5 // - ถ้า IR ว่าง → error ทันที
6 // - เรียก assembleProgram เพื่อแปลงและเขียนไฟล์ผลลัพธ์
7 // - คืนค่า 0/1 ตามผล (สอดคล้องสเปก project)
8 int main(int argc, char** argv){
9     ios::sync_with_stdio(false);
10    cin.tie(nullptr);
11
12    // ดีฟอลต์ชื่อไฟล์ (สามารถส่งเองผ่าน argv)
13    string irPath = (argc >= 2 ? argv[1] : "program.ir");
14    string symPath = (argc >= 3 ? argv[2] : "program_symbols.txt");
15    string outPath = (argc >= 4 ? argv[3] : "machineCode.mc");
16
17    // โหลดข้อมูลที่จำเป็น
18    auto symtab = loadSymbolTable(symPath);
19    auto irs    = loadIR(irPath);
20
21    // ถ้าไม่มี IR → ไม่มีอะไรให้แปลง → นับเป็น error
22    if (irs.empty()) {
23        cerr << "ERROR: no IR to assemble (check " << irPath << ")\n";
24        return 1;
25    }
26
27    cerr << "--- Assembling " << irs.size() << " instruction(s) ---\n";
28
29    // วนประกอบ + เขียนไฟล์ (fail-fast)
30    int code = assembleProgram(symtab, irs, outPath);
31
32    if (code == 0) {
33        cout << "Assemble success. Wrote machine code to: " << outPath << "\n";
34    } else {
35        cerr << "Assemble failed. See errors above.\n";
36    }
37    return code; // 0=สำเร็จ, 1=ล้มเหลว (สอดคล้องกับข้อกำหนด)
38 }
```

2. Simulator

Simulator.java บรรทัดที่ 1-30

```
1 import java.io.*;
2 import java.util.*;
3
4 class Main {
5     Run|Debug
6     public static void main(String[] args) throws Exception {
7         Simulator.main(args); // pass through "machine_code.txt"
8     }
9 }
10 /**
11 * SMC Simulator (8 regs, 32-bit, word-addressed) – filename via main(args)
12 * Usage: java Simulator <machine-code.txt>
13 *
14 * - One signed 32-bit decimal per line in the input file.
15 * - numMemory = number of lines.
16 * - Registers init to 0; R0 is always 0 (writes ignored).
17 * - PC starts at 0.
18 * - printState is called before each instruction executes, and once more before exit.
19 * - I-type offsets are 16-bit two's complement; sign-extended at runtime.
20 * Opcodes (bits 24..22): add=000, nand=001, lw=010, sw=011, beq=100, jalr=101, halt=110, noop=111
21 */
22
23 // usage
24 //      - java Simulator <factorial.mc>
25 //      - java Simulator factorial.mc > new file name <fact_sim.txt>
26
27 public class Simulator {
28
29     private static final int NUM_REGS = 8;
```

Simulator.java บรรทัดที่ 31-58

```
// Machine state
31 private int pc = 0;
32 private int[] regs = new int[NUM_REGS];
33 private static final int NUMMEMORY = 65536;
34 private int[] mem = new int[NUMMEMORY]; // instead of sizing to numMemory
35 private int numMemory = 0; // still track how many lines you loaded
36 private boolean halted = false;
37
38 Run | Debug
39 public static void main(String[] args) throws Exception {
40     if (args.length != 1) {
41         System.out.println("error: usage: java Simulator <machine-code file>");
42         System.exit(status:1);
43     }
44     Simulator sim = new Simulator();
45     sim.loadFromFile(args[0]);
46     sim.run();
47 }
48
49 /** Load machine code (one integer per line) from a file path */
50 private void loadFromFile(String path) throws IOException {
51     try (BufferedReader br = new BufferedReader(new FileReader(path))) {
52         ArrayList<Integer> lines = new ArrayList<>();
53         String s;
54         int lineNo = 0;
55         while ((s = br.readLine()) != null) {
56             lineNo++;
57             s = s.trim();
58             if (s.isEmpty()) continue; // allow blank lines
```

Simulator.java บรรทัดที่ 58-85

```

58     if (s.isEmpty()) continue;                                // allow blank lines
59     String first = s.split(regex:"\\s+")[0];                // tolerate trailing comments/tokens
60     try {
61         int val = Integer.parseInt(first);
62         lines.add(val);
63     } catch (NumberFormatException nfe) {
64         System.err.println("error in reading address " + (lines.size()) + " (line " + lineno + ")");
65         System.exit(status:1);
66     }
67 }
68 numMemory = lines.size();
69 //mem = new int[numMemory];
70 for (int i = 0; i < numMemory; i++) {
71     mem[i] = lines.get(i);
72     System.out.println("memory[" + i + "]=" + mem[i]); // ← echo
73 }
74     | Arrays.fill(regs, val:0);
75     regs[0] = 0; // enforce R0=0
76 }
77
78
79 private static final int MAX_STEPS = 1_000_000;
80 private int steps = 0;
81
82 /** Main simulation loop */
83 private void run() {
84     while (true) {
85         printState(); // before executing each instruction

```

Simulator.java บรรทัดที่ 86-114

```

86
87 // Step-limit guard (before fetch)
88 if (++steps > MAX_STEPS) {
89     System.err.println("possible infinite loop (exceeded " + MAX_STEPS + " steps)");
90     // print one more state before exit, per project rules
91     printState();
92     break; // or return;
93 }
94
95 // Fetch
96 if (pc < 0 || pc >= numMemory) {
97     // Out-of-bounds fetch: stop (optional safety)
98     System.err.println("error: pc out of bounds: " + pc);
99     break;
100 }
101 int instr = mem[pc];
102
103 // Decode
104 int opcode = (instr >>> 22) & 0x7;
105 int rs    = (instr >>> 19) & 0x7;
106 int rt    = (instr >>> 16) & 0x7;
107 int rd    = instr        & 0x7;    // R-type dest
108 int imm16 = instr        & 0xFFFF; // I-type raw
109 int imm32 = convertNum(imm16);      // sign-extend 16->32
110
111 int nextPC = pc + 1; // default advance
112
113 switch (opcode) {
114     case 0: { // add

```

Simulator.java บรรทัดที่ 114-139

```

112
113     switch (opcode) {
114         case 0: { // add
115             regs[rd] = regs[rs] + regs[rt];
116             break;
117         }
118         case 1: { // nand
119             regs[rd] = ~(regs[rs] & regs[rt]);
120             break;
121         }
122         case 2: { // lw
123             int addr = regs[rs] + imm32;
124             if (addr < 0 || addr >= NUMMEMORY) {
125                 System.out.println("error: lw address out of bounds: " + addr);
126                 break;
127             }
128             regs[rt] = mem[addr];
129             break;
130         }
131         case 3: { // sw
132             int addr = regs[rs] + imm32;
133             if (addr < 0 || addr >= NUMMEMORY) {
134                 System.out.println("error: sw address out of bounds: " + addr);
135                 break;
136             }
137             mem[addr] = regs[rt];
138             break;
139         }

```

Simulator.java บรรทัดที่ 139-166

```

139
140     }
141     case 4: { // beq
142         if (regs[rs] == regs[rt]) {
143             nextPC = pc + 1 + imm32; // offset is words; no shifting
144         }
145         break;
146     }
147     case 5: { // jalr
148         int ret = pc + 1; // compute return address first
149         int target = regs[rs]; // jump target
150         regs[rt] = ret; // write link
151         // Special case: rs==rt → jump to PC+1 (per spec)
152         nextPC = (rs == rt) ? ret : target;
153         break;
154     }
155     case 6: { // halt
156         halted = true;
157         // nextPC remains pc+1 (spec)
158         break;
159     }
160     case 7: { // noop
161         // nothing
162         break;
163     }
164     default: {
165         // Unknown opcode
166         halted = true;
167         break;
168     }

```

Simulator.java บรรทัดที่ 167-195

```

167         }
168     }
169
170     // Enforce R0 is always 0
171     regs[0] = 0;
172     // Commit PC
173     pc = nextPC;
174
175     // On halt: print state once more and exit
176     if (halted) {
177         printState();
178         break;
179     }
180 }
181
182 /**
183  * Sign-extend 16-bit value to 32-bit */
184 private static int convertNum(int num) {
185     if ((num & (1 << 15)) != 0) {
186         num -= (1 << 16);
187     }
188     return num;
189 }
190
191 /**
192  * Print machine state in the project required format */
193 private void printState() {
194     System.out.println("@@@");
195     System.out.println("state:");
196     System.out.println("\tpc " + pc);

```

Simulator.java บรรทัดที่ 190-208

```

190
191 /**
192  * Print machine state in the project required format */
193 private void printState() {
194     System.out.println("@@@");
195     System.out.println("state:");
196     System.out.println("\tpc " + pc);
197     System.out.println("\tmemory:");
198     for (int i = 0; i < numMemory; i++) {
199         System.out.println("\t\tmem[ " + i + " ] " + mem[i]);
200     }
201     System.out.println("\tregisters:");
202     for (int i = 0; i < NUM_REGS; i++) {
203         System.out.println("\t\treg[ " + i + " ] " + regs[i]);
204     }
205     //end state
206     System.out.println("end state");
207     System.out.println(" ");
208 }
209

```

3. Assembly Programs

3.1 multiply.asm

```
programs > asm multiply.asm
1      lw    0    1    five ; โหลดค่า 5 จาก address 9 ไปยัง register 1
2      lw    0    2    three ; โหลดค่า 3 จาก address 10 ไปยัง register 2
3      lw    0    4    neg1 ; โหลดค่า -1 จาก address 11 ไปยัง register 4
4      add   0    0    5    ; ตั้งค่า register 5 = 0 (ผลลัพธ์)
5
6      loop  beq   2    0    done ; ถ้า register 2 == 0 กระโดดไปที่ done
7      add   5    1    5    ; ผลลัพธ์ = ผลลัพธ์ + 5
8      add   2    4    2    ; ตัวนับ = ตัวนับ - 1
9      beq   0    0    loop ; กระโดดกลับไปที่ loop
10
11     done   halt          ; จบโปรแกรม
12
13     five   .fill   5
14     three  .fill   3
15     neg1   .fill  -1
```

3.2 multiply.mc

```
programs > mc multiply.mc
1 8454153 ; lw 0 1 9 : 010(2)<<22 | 000(0)<<19 | 001(1)<<16 | 9 = 8454153
2 9043970 ; lw 0 2 10 : 010(2)<<22 | 000(0)<<19 | 010(2)<<16 | 10 = 9043970
3 9043972 ; lw 0 4 11 : 010(2)<<22 | 000(0)<<19 | 100(4)<<16 | 11 = 9043972
4 65541   ; add 0 0 5 : 000(0)<<22 | 000(0)<<19 | 000(0)<<16 | 5 = 65541
5 16842754 ; beq 2 0 4 : 100(4)<<22 | 010(2)<<19 | 000(0)<<16 | 4 = 16842754
6 7208965 ; add 5 1 5 : 000(0)<<22 | 101(5)<<19 | 001(1)<<16 | 5 = 7208965
7 7208970 ; add 2 4 2 : 000(0)<<22 | 010(2)<<19 | 100(4)<<16 | 2 = 7208970
8 16842749 ; beq 0 0 -3 : 100(4)<<22 | 000(0)<<19 | 000(0)<<16 | (-3 & 0xFFFF) = 16842749
9 29360128 ; halt      : 110(6)<<22 = 29360128
10 5        ; five
11 3        ; three
12 -1       ; neg1
```

3.3 factorial.asm

```

programs > ASM factorial.asm
1    lw     0      1      n      ; R1 = 5
2    add   0      0      2      ; R2 = 1 (result)
3    lw     0      4      neg1   ; R4 = -1
4
5 fact beq   1      0      done   ; ถ้า R1 == 0 จะ
6
7    ; คูณ R2 × R1 โดยวน R2 ส่วน R1 ครั้ง
8    add   0      0      5      ; R5 = 0 (temp)
9    add   0      1      3      ; R3 = R1 (ตัวนับ)
10
11 mult beq   3      0      mult_done
12    add   5      2      5      ; R5 += R2
13    add   3      4      3      ; R3--
14    beq   0      0      mult
15
16 mult_done add   0      5      2      ; R2 = ผลคูณใหม่
17    add   1      4      1      ; R1-- (n--)
18    beq   0      0      fact
19
20 done halt
21
22 n     .fill   5
23 one   .fill   1
24 neg1 .fill   -1

```

3.4 factorial.mc

```

programs > ≡ factorial.mc
1 8454153 ; lw 0 1 n      (address 16): 010<<22 | 000<<19 | 001<<16 | 16 = 8454153
2 65538  ; add 0 0 2      : 000<<22 | 000<<19 | 000<<16 | 2 = 65538
3 9043972 ; lw 0 4 neg1   (address 17): 010<<22 | 000<<19 | 100<<16 | 17 = 9043972
4 16842764 ; beq 1 0 done  (address 18): 100<<22 | 001<<19 | 000<<16 | 18 = 16842764
5 65541  ; add 0 0 5      : 000<<22 | 000<<19 | 000<<16 | 5 = 65541
6 65539  ; add 0 1 3      : 000<<22 | 000<<19 | 001<<16 | 3 = 65539
7 16842758 ; beq 3 0 mult_done(address 19):100<<22 | 011<<19 | 000<<16 | 19 = 16842758
8 7208965  ; add 5 2 5      : 000<<22 | 101<<19 | 010<<16 | 5 = 7208965
9 7208971  ; add 3 4 3      : 000<<22 | 011<<19 | 100<<16 | 3 = 7208971
10 16842752 ; beq 0 0 mult   (address 14): 100<<22 | 000<<19 | 000<<16 | -5 = 16842752
11 65538  ; add 0 5 2      : 000<<22 | 000<<19 | 101<<16 | 2 = 65538
12 7208969  ; add 1 4 1      : 000<<22 | 001<<19 | 100<<16 | 1 = 7208969
13 16842749 ; beq 0 0 fact   (address 3) : 100<<22 | 000<<19 | 000<<16 | -11 = 16842749
14 29360128 ; halt          : 110<<22 = 29360128
15 5      ; n
16 1      ; one
17 -1     ; neg1

```

4. Test

4.1 multiply test

```

programs asm > tests > multiply_test.txt
1 # Test Cases for Multiply Program ทดสอบโปรแกรมคูณเลข 2 จำนวน
2 # - Input: A (r1), B (r2) Output: ผลลัพธ์อยู่ใน r3 หรือ memory[OUT]
3
4 # Case 1: Normal case
5 # การคูณเลขบวกทั่วไป
6 # Expected: 3 * 4 = 12
7 A = 3
8 B = 4
9 Expected Output = 12
10
11 # Case 2: Multiplication with zero
12 # การคูณเลขด้วยจำนวนศูนย์ ต้องได้ผลลัพธ์เป็น 0
13 # Expected: 5 * 0 = 0
14 A = 5
15 B = 0
16 Expected Output = 0
17
18 # Case 3: Multiplication with one
19 # การคูณเลขด้วยจำนวน 1 ต้องได้ค่าเดิม
20 # Expected: 7 * 1 = 7
21 A = 7
22 B = 1
23 Expected Output = 7
24
25 # Case 4: Equal numbers
26 # การคูณเลขเดียวกันเข้าด้วยกัน ตรวจสอบการวน loop ซ้ำหลายรอบ
27 # Expected: 5 * 5 = 25
28 A = 5
29 B = 5
30 Expected Output = 25
31
32 # Case 5: Larger operands
33 # ตรวจสอบความถูกต้องเมื่อ loop หลายครั้ง
34 # Expected: 8 * 6 = 48
35 A = 8
36 B = 6
37 Expected Output = 48
38
39 # Case 6: Negative number (optional)
40 # ถ้า simulator รองรับค่าลบ ให้ทดสอบด้วยการลองคูณจำนวนลบ
41 # Expected: (-3) * 4 = -12
42 A = -3
43 B = 4
44 Expected Output = -12

```

4.2 factorial test

```

programs asm > tests > factorial_test.txt
1 # Test Cases for Factorial Program :
2 # - ทดสอบโปรแกรม factorial (n!)
3 # - Input: n (r1 หรือ memory[N])
4 # - Output: ผลลัพธ์อยู่ใน r3 หรือ memory[OUT]
5
6 # Case 1: Base case
7 # n = 0 → factorial(0) = 1
8 # ตรวจสอบว่าโปรแกรม handle base case ถูกต้อง
9 n = 0
10 Expected Output = 1
11
12 # Case 2: Small number
13 # n = 1 → factorial(1) = 1
14 # ตรวจสอบว่า recursion หรือ loop เริ่มทำงานถูกต้อง ด้วยลูปเล็กที่ง่าย
15 n = 1
16 Expected Output = 1
17
18 # Case 3: Normal case
19 # n = 3 → factorial(3) = 3 * 2 * 1 = 6
20 # ตรวจสอบการสร้าง recursion/loop หลาຍรอบ
21 n = 3
22 Expected Output = 6
23
24 # Case 4: Bigger number
25 # n = 5 → factorial(5) = 120
26 # ตรวจสอบการคำนวณหลายรอบ ด้วยลูปใหญ่ที่มีความซับซ้อนมาก
27 n = 5
28 Expected Output = 120
29
30 # Case 5: Large number stress test
31 # n = 8 → factorial(8) = 40320
32 # ทดสอบว่า simulator สามารถทำงานได้โดยไม่ overflow (ถ้าหากเป็น 16-bit signed)
33 n = 8
34 Expected Output = 40320
35
36 # Case 6: Invalid / negative input (optional)
37 # n = -1 → factorial(-1) undefined
38 # ตรวจสอบว่าระบบหยุดหรือให้ค่า 0/1 (ตาม design)
39 n = -1
40 Expected Output = undefined (should handle gracefully)
41

```

GitHub

<https://github.com/660610792WarangkanaJitwarangkana/261304Project-SMC>