**รายงาน**

**Project 1 Computer Architecture (CPE 261304)**

**จัดทำโดย**

นางสาวณัฐรดา หนูจิตร            660610757

นางสาวณัฐวรา ไชยสิทธิ์          660610758

นางสาวทิพวรินทร์ สีห์วรางกูร 660610760

นางสาวธนพร ตั้งผดุงสุข 660610762

นายศตคุณ นนทา 660612158

**เสนอ**

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของรายวิชา 261304 Computer Architecture

มหาวิทยาลัยเชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2568

**สารบัญ**

[**ส่วนที่ 1 Assembler** 3](#_Toc211775852)

[**Flowchart** 4](#_Toc211775853)

[**Pseudo Code** 5](#_Toc211775854)

[**การทดลอง** 17](#_Toc211775855)

[**ส่วนที่ 2** 19](#_Toc211775856)

[**ส่วนที่ 3** 20](#_Toc211775857)

# **ส่วนที่ 1 Assembler**

ในส่วนนี้เป็นการพัฒนาโปรแกรม Assembler ซึ่งทำหน้าที่แปลงชุดคำสั่งจากภาษา Assembly ให้อยู่ในรูปแบบของ Machine Code ที่คอมพิวเตอร์สามารถเข้าใจและนำไปประมวลผลได้โดยตรง โปรแกรมนี้ถูกพัฒนาด้วยภาษา C++ โดยออกแบบให้ทำงานตามหลักการของ Two-pass Assembler เพื่อให้สามารถรองรับการอ้างอิง Label ได้ในโปรแกรม Assembly

การทำงานของโปรแกรมสามารถแบ่งออกได้เป็น 2 ขั้นตอนหลัก (2 Pass) ดังนี้

1. Pass 1 – การสร้างตาราง Label (Label Table Construction)  
    โปรแกรมจะทำการอ่านไฟล์คำสั่งภาษา Assembly ทีละบรรทัด และตรวจสอบคำแรกของแต่ละบรรทัดว่าเป็นชื่อ Label หรือไม่ หากพบว่าเป็น Label โปรแกรมจะทำการตรวจสอบความถูกต้องของชื่อ (ต้องไม่เกิน 6 ตัวอักษร, ต้องขึ้นต้นด้วยตัวอักษร, และห้ามซ้ำกับ Label อื่น) จากนั้นจะเก็บชื่อ Label พร้อมตำแหน่งบรรทัด (Address) ไว้ในตาราง Label เพื่อใช้ในการแปลงคำสั่งในขั้นตอนถัดไป
2. Pass 2 – การแปลงคำสั่งเป็น Machine Code (Instruction Translation)  
    โปรแกรมจะอ่านไฟล์อีกครั้งเพื่อทำการวิเคราะห์คำสั่งแต่ละบรรทัด แยกส่วนประกอบของคำสั่ง เช่น Opcode, Register (regA, regB, destReg) และ Offset แล้วทำการเข้ารหัส (Encoding) ให้เป็นเลขฐานสิบ (Decimal) ตามรูปแบบของคำสั่งแต่ละประเภท
3. จากนั้นโปรแกรมจะทำการรวมค่าแต่ละส่วนด้วยการ เลื่อนบิต (Bit shifting) และ เชื่อมด้วย OR operation (|) เพื่อให้ได้รหัสเครื่องสมบูรณ์ในรูปแบบ 32 บิต ก่อนจะเขียนผลลัพธ์ลงในไฟล์ output

## **Flowchart**

## **Pseudo Code**

STRUCT Label

    string name

    int address

END STRUCT

หน้าที่ : ใช้สำหรับเก็บข้อมูลของ label แต่ละตัวในโปรแกรม assembly

* name คือชื่อของ label (เช่น “start”, “loop”)
* address คือหมายเลขบรรทัดหรือ address ในหน่วยความจำของคำสั่งนั้น ๆ

ตัวนี้จะถูกสร้างขึ้นระหว่าง Pass 1 เพื่อใช้ในภายหลังตอนแปลง label เป็นค่าจำนวนจริง (address)

FUNCTION printError(fout, message)

    PRINT message TO standard error (cerr)

    WRITE message INTO fout (output file)

    CLOSE fout

    TERMINATE program with exit code 1

END FUNCTION

หน้าที่ : ใช้แสดงข้อความ error ทั้งบนหน้าจอ (cerr) และบันทึกลงในไฟล์ output (fout) จากนั้นปิดไฟล์และหยุดการทำงานของโปรแกรมทันที

ตัวอย่าง error (จะแสดงในส่วนต่อไป)

* label ซ้ำ
* label ไม่ถูกต้อง
* register เกินช่วง
* opcode ผิด

FUNCTION isNumber(string s) RETURNS boolean

    IF s is empty THEN

        RETURN false

    ENDIF

    TRY to convert s to integer using strtol

    IF conversion successful (no invalid characters)

        RETURN true

    ELSE

        RETURN false

END FUNCTION

หน้าที่ : ตรวจสอบว่า string ที่รับเข้ามาเป็นตัวเลขหรือไม่

* ใช้ strtol() เพื่อพยายามแปลง string เป็น integer
* ถ้ามีตัวอักษรที่ไม่ใช่ตัวเลข จะถือว่าไม่ใช่ตัวเลข

ตัวอย่าง :

|  |  |
| --- | --- |
| input | ผลลัพธ์ |
| 5 | true |
| -12 | true |
| five | false |

FUNCTION isValidLabelName(string s) RETURNS boolean

    IF s is empty OR s.length > 6 THEN RETURN false

    IF first character of s is not a letter THEN RETURN false

    FOR each character c in s:

        IF c is not alphanumeric THEN RETURN false

    RETURN true

END FUNCTION

หน้าที่ : ตรวจสอบความถูกต้องของชื่อ label ตามที่โจทย์กำหนด

* ต้องไม่เกิน 6 ตัวอักษร
* ต้องขึ้นต้นด้วยตัวอักษร (A–Z, a–z)
* ตัวถัด ๆ ไปต้องเป็นตัวอักษรหรือเลขเท่านั้น

เช่น

|  |  |
| --- | --- |
| input | ผลลัพธ์ |
| start | true |
| 3loop | false (ขึ้นต้นด้วยตัวเลข) |
| abc-1 | false(มีตัวอักษรพิเศษ) |

FUNCTION findLabelAddress(fout, labels, name) RETURNS int

    FOR each label l in labels:

        IF l.name == name THEN

            RETURN l.address

    END FOR

    CALL printError(fout, "undefined label: " + name)

END FUNCTION

หน้าที่ : ค้นหาว่า label ที่ให้มามีอยู่ในตาราง labels หรือไม่

* ถ้ามี → คืนค่าหมายเลข address ของ label นั้น
* ถ้าไม่มี → แสดง error และหยุดโปรแกรม

ใช้ตอนแปลงคำสั่ง เช่น

lw 0 1 five → ต้องดูว่า “five” อยู่บรรทัดไหน แล้วเอาเลข address มาคำนวณ offset

FUNCTION checkRegRange(fout, r, which, line)

    IF r < 0 OR r > 7 THEN

        CALL printError(fout, "register out of range at line " + line)

    ENDIF

END FUNCTION

หน้าที่ : ตรวจสอบว่าค่า register อยู่ในช่วงที่ถูกต้องหรือไม่ (0–7 เท่านั้น) หากอยู่นอกช่วง จะ error ทันที โดยใช้ในทุกคำสั่งที่มี register เช่น add, lw, sw, jalr

**MAIN PROGRAM**

**1️. Check Input Arguments**

IF argc != 3 THEN

    PRINT "usage: assembler <inputfile> <outputfile>" TO error

    TERMINATE program

ENDIF

หน้าที่ : ตรวจสอบว่ารับ argc มาครบหรือไม่ ถ้าไม่ครบ แสดง error

**2️. Open Files**

OPEN input file (argv[1]) FOR reading AS fin

OPEN output file (argv[2]) FOR writing AS fout

IF fin OR fout cannot be opened THEN

    CALL printError(fout, "cannot open file")

ENDIF

หน้าที่ : เปิดไฟล์ assembly สำหรับอ่าน เปิดไฟล์ output สำหรับเขียน หากเปิดไม่ได้ จะแจ้ง error ทันที

**3️. Read All Lines into Memory**

CREATE empty list lines

FOR each line from fin:

    TRIM leading whitespace

    IF line not empty THEN

        APPEND line to lines

ENDFOR

หน้าที่ : อ่านทุกบรรทัดจากไฟล์เก็บใน lines[] พร้อมลบช่องว่างหน้า/ท้ายบรรทัด

**4.  PASS 1 — Build Label Table**

CREATE empty list labels

FOR i = 0 TO lines.size() - 1:

    TOKENIZE lines[i] by spaces INTO tokens

    IF tokens is empty THEN CONTINUE

    first = tokens[0]

    IF first is not one of {add, nand, lw, sw, beq, jalr, halt, noop, .fill} THEN

        IF not isValidLabelName(first) THEN

            CALL printError(fout, "invalid label name")

        ENDIF

        FOR each label l in labels:

            IF l.name == first THEN

                CALL printError(fout, "duplicate label")

        ENDFOR

        ADD (first, i) TO labels

    ENDIF

ENDFOR

หน้าที่ : วนอ่านทุกบรรทัด

1. แยก token (คำในบรรทัด เช่น label, opcode, arg)
2. ถ้าคำแรกไม่ใช่ opcode  ถือว่าเป็น label
3. ตรวจสอบความถูกต้องของชื่อ label
4. ตรวจซ้ำ
5. เก็บชื่อกับตำแหน่งบรรทัดไว้ใน labels

 ตัวอย่าง :

start add 0 1 2

loop beq 0 1 start

ได้ตาราง labels:

|  |  |
| --- | --- |
| name | address |
| start | 0 |
| loop | 1 |

**5️. PASS 2 — Assemble Instructions**

FOR i = 0 TO lines.size() - 1:

TOKENIZE lines[i] INTO tokens

IF tokens empty THEN CONTINUE

IF tokens[1] is opcode THEN

opcode = tokens[1]

arg0 = tokens[2] (if any)

arg1 = tokens[3] (if any)

arg2 = tokens[4] (if any)

ELSE

opcode = tokens[0]

arg0 = tokens[1] (if any)

arg1 = tokens[2] (if any)

arg2 = tokens[3] (if any)

ENDIF

หน้าที่ : วนอ่านแต่ละบรรทัดใหม่ แยก opcode และ arguments

**5.1 Determine Opcode Value**

SET op = -1

SWITCH (opcode)

    CASE "add": op = 0

    CASE "nand": op = 1

    CASE "lw": op = 2

    CASE "sw": op = 3

    CASE "beq": op = 4

    CASE "jalr": op = 5

    CASE "halt": op = 6

    CASE "noop": op = 7

    CASE ".fill": op = -2

    DEFAULT:

        CALL printError(fout, "invalid opcode at line i")

ENDSWITCH

หน้าที่ : แปลงชื่อ opcode ให้เป็นตัวเลข op

**5.2 fill Directive**

IF op == -2 THEN

    IF arg0 is numeric THEN

        value = int(arg0)

    ELSE

        value = findLabelAddress(fout, labels, arg0)

    ENDIF

    PRINT "(address i): value (hex 0x...)" TO console

    WRITE value TO fout

    CONTINUE to next instruction

ENDIF

หน้าที่ : เข้ารหัส (encode) ตามรูปแบบ .fill

**5.3 Instruction Encoding**

▪️ R-Type Instructions (add, nand)

IF op <= 1 THEN

    regA = int(arg0)

    regB = int(arg1)

    dest = int(arg2)

    checkRegRange(fout, regA, "regA", i)

    checkRegRange(fout, regB, "regB", i)

    checkRegRange(fout, dest, "destReg", i)

    machineCode = (op << 22) | (regA << 19) | (regB << 16) | dest

ENDIF

▪️ I-Type Instructions (lw, sw, beq)

IF op >= 2 AND op <= 4 THEN

    regA = int(arg0)

    regB = int(arg1)

    IF arg2 is number THEN

        offset = int(arg2)

    ELSE

        labelAddr = findLabelAddress(fout, labels, arg2)

        IF opcode == "beq" THEN

            offset = labelAddr - (i + 1)

        ELSE

            offset = labelAddr

        ENDIF

    ENDIF

    IF offset < -32768 OR offset > 32767 THEN

        CALL printError(fout, "offset out of range")

    ENDIF

    offset = offset & 0xFFFF

    machineCode = (op << 22) | (regA << 19) | (regB << 16) | offset

ENDIF

▪️ J-Type (jalr)

IF op == 5 THEN

    regA = int(arg0)

    regB = int(arg1)

    checkRegRange(fout, regA, "regA", i)

    checkRegRange(fout, regB, "regB", i)

    machineCode = (op << 22) | (regA << 19) | (regB << 16)

ENDIF

▪️ O-Type (halt, noop)

IF op >= 6 THEN

    machineCode = (op << 22)

ENDIF

หน้าที่ : เข้ารหัส (encode) ตามรูปแบบ instruction type

**6. Write Output**

PRINT "(address i): machineCode (hex 0x...)" TO console

WRITE machineCode TO fout

หน้าที่ : ปริ้น output ออกมา

**7. Finish**

CLOSE output file

EXIT(0)

หน้าที่ : ปิดไฟล์ ปิดโปรแกรม และ exit(0) ถ้า program สามารถทำงานได้โดยที่ไม่เจอ error

|  |  |
| --- | --- |
| ขั้นตอน | รายละเอียด |
| 1 | ตรวจ arguments |
| 2 | เปิดไฟล์ input / output |
| 3 | อ่านบรรทัดทั้งหมดจากไฟล์ assembly |
| 4 | Pass 1: เก็บ label และตรวจชื่อ |
| 5 | Pass 2: แปลงคำสั่งเป็น machine code และ เข้ารหัสคำสั่งแต่ละแบบ (R, I, J, O, .fill |
| 6 | เขียน machine code ลงไฟล์ output |
| 7 | ปิดไฟล์และจบการทำงาน |

## **การทดลอง**

**Case 1 : assembly code ที่ใช้ทดลองจากโจทย์ที่ให้มา**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

จากการทดลองโปรแกรมพบว่าสามารถทำงานได้ตามโจทย์ที่ได้รับมา โปรแกรมสามารถรันคำสั่งทุกประเภทได้ครบถ้วน ไม่ว่าจะเป็นคำสั่งคำนวณหรือคำสั่งควบคุมลำดับการทำงาน อีกทั้งยังสามารถจัดการกับ label และคำสั่ง .fill ได้ตามที่ต้องการ ทำให้ผลลัพธ์ที่ออกมาถูกต้องตรงกับที่คาดไว้

**Case 2 : Multiplication assembly code**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

จากการทดลองโปรแกรมพบว่าสามารถทำงานได้ตามโจทย์ที่ได้รับมา โปรแกรมสามารถรันคำสั่งทุกประเภทได้ครบถ้วน ไม่ว่าจะเป็นคำสั่งคำนวณหรือคำสั่งควบคุมลำดับการทำงาน อีกทั้งยังสามารถจัดการกับ label และคำสั่ง .fill ได้ตามที่ต้องการ ทำให้ผลลัพธ์ที่ออกมาถูกต้องตรงกับที่คาดไว้

**Case 3 : Combination assembly code**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

จากการทดลองโปรแกรมพบว่าสามารถทำงานได้ตามโจทย์ที่ได้รับมา โปรแกรมสามารถรันคำสั่งทุกประเภทได้ครบถ้วน ไม่ว่าจะเป็นคำสั่งคำนวณหรือคำสั่งควบคุมลำดับการทำงาน อีกทั้งยังสามารถจัดการกับ label และคำสั่ง .fill ได้ตามที่ต้องการ ทำให้ผลลัพธ์ที่ออกมาถูกต้องตรงกับที่คาดไว้

**Case 4 : Divide assembly code**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

จากการทดลองโปรแกรมพบว่าสามารถทำงานได้ตามโจทย์ที่ได้รับมา โปรแกรมสามารถรันคำสั่งทุกประเภทได้ครบถ้วน ไม่ว่าจะเป็นคำสั่งคำนวณหรือคำสั่งควบคุมลำดับการทำงาน อีกทั้งยังสามารถจัดการกับ label และคำสั่ง .fill ได้ตามที่ต้องการ ทำให้ผลลัพธ์ที่ออกมาถูกต้องตรงกับที่คาดไว้

**Case 5 : Fibonacci assembly code**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

จากการทดลองโปรแกรมพบว่าสามารถทำงานได้ตามโจทย์ที่ได้รับมา โปรแกรมสามารถรันคำสั่งทุกประเภทได้ครบถ้วน ไม่ว่าจะเป็นคำสั่งคำนวณหรือคำสั่งควบคุมลำดับการทำงาน อีกทั้งยังสามารถจัดการกับ label และคำสั่ง .fill ได้ตามที่ต้องการ ทำให้ผลลัพธ์ที่ออกมาถูกต้องตรงกับที่คาดไว้

**Case 6 : Exponential assembly code**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

จากการทดลองโปรแกรมพบว่าสามารถทำงานได้ตามโจทย์ที่ได้รับมา โปรแกรมสามารถรันคำสั่งทุกประเภทได้ครบถ้วน ไม่ว่าจะเป็นคำสั่งคำนวณหรือคำสั่งควบคุมลำดับการทำงาน อีกทั้งยังสามารถจัดการกับ label และคำสั่ง .fill ได้ตามที่ต้องการ ทำให้ผลลัพธ์ที่ออกมาถูกต้องตรงกับที่คาดไว้

**Case 7 : assembly code เมื่อ label ไม่ถูกประกาศ**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

**Case 8 : assembly code เมื่อมีการใช้ label ซ้ำ**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

**Case 9 : assembly code ที่ offset เกินขอบเขต**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

**Case 10 : assembly code  ที่ opcode ไม่ถูกต้อง**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

**Case 11 : assembly code เมื่อ argument ไม่ครบ**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

**Case 12 : assembly code ที่ใช้ register เกินขอบเขต**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

**Case 13 : assembly code เมื่อ label มีความยาวมากกว่า 6 ตัวอักษร**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

**Case 14 : assembly code เมื่อ label มีตัวเลขนำหน้า**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

**Case 15 : assembly code เมื่อ label มีอักษรพิเศษ**

**ผลการทดลอง**

**วิเคราะห์ผลการทดลอง**

# **ส่วนที่ 2**

# **ส่วนที่ 3**