系統程式書面報告

資工系 41071102H 徐敏皓

目錄

頁次

The architecture of the implemented assembler 1
What you have learned and experienced during the implementation 7
In case you implement more than the required specification 7
Copyright Claim 8
Any thing you would like to let G.H.Hwang know 8
How to run the program 9

The architecture of the implemented assembler

Pass one:

1. 建立 SYMTAB：

- 先建立 DIRECTIVES 和 OPCODE_TABLE（如圖一）。後續在確認 label 時，如果 label 的命名有在 DIRECTIVES 或 OPCODE_TABLE 中，會回報錯誤，否則記錄下 label 和 address of label（如圖二）。
- 計算 LOCCTR（Location Counter），並追蹤 address of instruction, label, and reversed space（如圖三）。

```
# Define global opcode table and directives based on the provided code
OPCODE_TABLE = {
    "STL": "14", "LDB": "68", "BASE": None, "COMP": "28", "JEQ": "30",
    "J": "3C", "JSUB": "48", "LDA": "00", "STA": "0C", "CLEAR": "B4",
    "LDT": "74", "TD": "E0", "RD": "D8", "COMPR": "A0", "STCH": "54",
    "TIXR": "B8", "LDCH": "50", "WD": "DC", "RSUB": "4C", "JLT": "38",
    "STX": "10", "LDT": "74"
}

DIRECTIVES = ["START",  "+", "BASE", "END", "BYTE", "RESW", "RESB"]
```

圖一

```
# Record symbol in symbol table if it has a label
if label:
    if label in OPCODE_TABLE or label in DIRECTIVES:
        raise ValueError(f"Label '{label}' cannot be an opcode or directive.")
    print(f"{label}: {locctr}")
    symtab[label] = locctr
```

圖二

```
# Update LOCCTR based on instruction size
if opcode[0] == "+":
    locctr = hex_add(locctr, "4")
elif opcode in OPCODE_TABLE:
    if opcode == "CLEAR" or opcode == "COMPR" or opcode == "TIXR":
        locctr = hex_add(locctr, "2")
    else:
        locctr = hex_add(locctr, "3")
elif opcode == "RESW":
    locctr = hex_add(locctr, f"{int(operand) * 3}")
elif opcode == "RESB":
    locctr = hex_add(locctr, str(hex(int(operand))))
elif opcode == "BYTE":
    if operand[0] == "C":
        locctr = hex_add(locctr, str(len(operand) - 3))  # Account for quotes
    else:
        locctr = hex_add(locctr, "1")
else:
    raise ValueError(f"Unknown opcode: {opcode}")
```

圖三

2. 產生 Intermediate Representation：

- 將每行指令的 memory address 和原始程式碼紀錄下來給 Pass two 使用，Pass two 就不需要再重讀檔案一次（如圖四）。

```
# Add current location and line to intermediate representation
intermediate.append({"loc": locctr, "line": line})
```

圖四

3. 處理 Base Addressing：

- 檢查是否有 BASE 這個 directive，後續在有必要時就能使用 base-relative addressing（如圖五）。

```
if opcode == "BASE":
    base_address = operand  # 取得 BASE 的位址
    continue
```

圖五

4. Output：

- **Symbol table（SYMTAB）**：紀錄有哪些 label 和每個 label 的 memory address（如圖六）。
- **Intermediate Representation**：每行指令的 memory address 和原始程式碼（如圖七）。

```
SYMTAB:
     Label Address
0     COPY       0
1    FIRST       0
2    CLOOP       6
3   ENDFIL      1A
4      EOF      2D
5   RETADR      30
6   LENGTH      33
7   BUFFER      36
8    RDREC    1036
9    RLOOP    1040
10    EXIT    1056
11   INPUT    105C
12   WRREC    105D
13   WLOOP    1062
14  OUTPUT    1076
```

圖六

```
{'loc': '0', 'line': 'COPY\tSTART\t0'}
{'loc': '0', 'line': 'FIRST\tSTL\tRETADR'}
{'loc': '3', 'line': 'LDB\t#LENGTH'}
{'loc': '6', 'line': 'BASE\tLENGTH'}
{'loc': '6', 'line': 'CLOOP\t+JSUB\tRDREC'}
{'loc': 'A', 'line': 'LDA\tLENGTH'}
{'loc': 'D', 'line': 'COMP\t#0'}
{'loc': '10', 'line': 'JEQ\tENDFIL'}
{'loc': '13', 'line': '+JSUB\tWRREC'}
{'loc': '17', 'line': 'J\tCLOOP'}
{'loc': '1A', 'line': 'ENDFIL\tLDA\tEOF'}
{'loc': '1D', 'line': 'STA\tBUFFER'}
{'loc': '20', 'line': 'LDA\t#3'}
{'loc': '23', 'line': 'STA\tLENGTH'}
{'loc': '26', 'line': '+JSUB\tWRREC'}
{'loc': '2A', 'line': 'J\t@RETADR'}
{'loc': '2D', 'line': "EOF\tBYTE\tC'EOF'"}
{'loc': '30', 'line': 'RETADR\tRESW\t1'}
{'loc': '33', 'line': 'LENGTH\tRESW\t1'}
{'loc': '36', 'line': 'BUFFER\tRESB\t4096'}
```

圖七

Pass two:

1. 產生 Object Code：

- 處理不同的 instruction format（如圖八）：
  - Format 4 獨立寫成一個 if 判斷是為了要處理 M record 的部分。
  - Format 2 和 Format 3 直接處理。
  - Directives 中 BYTE 也會產生 object code 所以也會單獨處理。

```
# Generate object code
if opcode.startswith("+"): #format 4
    obj_code = format_object_code(opcode, operand, loc, symtab, base_address)
    print(obj_code)
    if operand[0] != "#":
        modification_loc = f"{int(loc)+1}"
        modification_records.append(f"M^{modification_loc.zfill(6)}^05")
elif opcode in OPCODE_TABLE: #format 3 or format 2
    obj_code = format_object_code(opcode, operand, loc, symtab, base_address)
    print(obj_code)
elif opcode == "BYTE":  # Handle BYTE
    if operand.startswith("C'"):
        obj_code = ''.join(f"{ord(c):X}" for c in operand[2:-1])
        print(obj_code)
    elif operand.startswith("X'"):
        obj_code = operand[2:-1]
        print(obj_code)
else:
    raise ValueError(f"Unhandled opcode: {opcode}")
```

圖八

- 處理不同的 instruction format（actual function）（如圖九）：
  - RSUB 較為特別，通常不會在其前後加上 label，因此將其單獨拉出來寫。
  - 遇到 Format 2 的部分也將其先做處理。
  - 後面為 Format 3 和 Format 4 的處理，要決定 n, i, x, b, p, e 和算出 disp。

```
def format_object_code(opcode, operand, locctr, symtab, base):
    #deal with RSUB
    if opcode == "RSUB" or opcode == "+RSUB":
        op_bin = f"{int(OPCODE_TABLE[opcode], 16):06b}"
        ni_bin = "11"
        e = 0
        if opcode[0] == "+":
            e = 1
            xbpe_bin = f"{000}{e}"
            return f"{(int(op_bin,2) | int(ni_bin,2)):02X}{int(xbpe_bin, 2):01X}00000"
        xbpe_bin = f"{000}{e}"
        return f"{(int(op_bin,2) | int(ni_bin,2)):02X}{int(xbpe_bin, 2):01X}000"

    # deal with format 2
    if opcode == "CLEAR" or opcode == "TIXR":
        return f"{OPCODE_TABLE[opcode]}{REGISTER[operand]}0"
    if opcode == "COMPR":
        reg1, reg2 = operand.split(',')
        return f"{OPCODE_TABLE[opcode]}{REGISTER[reg1]}{REGISTER[reg2]}"

    # Initialize flags
    n = i = x = b = p = e = 0
    disp = 0
```

圖九

- 處理不同的 instruction format（actual function）：
  - 利用不同判斷條件決定 n, i, x, e（如圖十）。
  - 取得 target address，如果發現是 immediate address，b = p = 0，可以直接回傳不用算 disp（如圖十一）。

```
# Handle Format 4 (extended format)
if opcode.startswith("+"):
    e = 1
    opcode = opcode[1:]  # Remove '+' for opcode lookup

# Determine addressing mode (n, i)
if operand:
    if operand.startswith("#"):  # Immediate addressing
        i = 1
        n = 0
        operand = operand[1:]
    elif operand.startswith("@"):  # Indirect addressing
        n = 1
        i = 0
        operand = operand[1:]
    else:  # Simple addressing (direct mode)
        n = i = 1

# Check for indexed addressing (x)
if operand and ",X" in operand:
    x = 1
    operand = operand.replace(",X", "")
```

圖十

```
# Get target address
if operand in symtab:
    target_address = int(symtab[operand], 16) #轉16進位
    # print(target_address)
elif operand.isdigit():  # Immediate value
    target_address = int(operand)
    op_bin = f"{int(OPCODE_TABLE[opcode], 16):06b}"  # Get opcode binary
    ni_bin = f"{n}{i}"  # Combine n and i
    xbpe_bin = f"{x}{b}{p}{e}"
    if e == 1:
        return f"{(int(op_bin,2) | int(ni_bin,2)):02X}{int(xbpe_bin, 2):01X}{target_address:05X}"
    else:
        return f"{(int(op_bin,2) | int(ni_bin,2)):02X}{int(xbpe_bin, 2):01X}{target_address & 0xFFF:03X}"
else:
    target_address = 0
```

圖十一

- 處理不同的 instruction format（actual function）：
  - 利用不同判斷條件決定 b, p（如圖十二）。
  - 將 opcode, n, I, x, b, p, e, disp 整合成一個完整的 object code（如圖十三）。

```
# Determine displacement (b, p)
if e:  # Format 4 (no displacement, just target address)
    disp = target_address
else:  # Format 3
    pc = int(locctr, 16) + 3  # PC is the next instruction
    disp = target_address - pc

    if -2048 <= disp <= 2047:  # PC-relative
        p = 1
    else:  # Try base-relative
        disp = target_address - int(base, 16)
        if 0 <= disp <= 4095:
            b = 1
        else:
            raise ValueError(f"Address out of range for PC or Base relative: {operand}")
```

圖十二

```
# Construct the object code
op_bin = f"{int(OPCODE_TABLE[opcode], 16):06b}"  # Get opcode binary
ni_bin = f"{n}{i}"  # Combine n and i
xbpe_bin = f"{x}{b}{p}{e}"  # Combine x, b, p, e

# Combine into final object code
if e:  # Format 4
    return f"{(int(op_bin,2) | int(ni_bin,2)):02X}{int(xbpe_bin, 2):01X}{disp:05X}"
else:  # Format 3
    return f"{(int(op_bin,2) | int(ni_bin,2)):02X}{int(xbpe_bin, 2):01X}{disp & 0xFFF:03X}"
```

圖十三

2. 產生 Object program：

- Head record: 因為 Head record 只有一行就直接單獨處理（如圖十四）。
- Text record: 有兩種情形會將 Text record 段開:
  - 第一個是遇到圖中列出的 Directives 時（如圖十五）。
  - 另一個則是因為 Text record 本身 format 的關係，Col. 10-69 會填入 object code（課本 p. 49），當 object code 的總長度超過 60 的時候，就要斷開（如圖十六）。

- Modification record: 處理含有 Format 4 的指令時就會紀錄下來（如圖十七）。
- End record:因為 End record 只有一行就直接單獨處理（如圖十八）。

```
# Initialize header record
start_address = intermediate_rep[0]['loc']
program_name = intermediate_rep[0]['line'].split()[0]
program_length = hex(int(intermediate_rep[-1]['loc'], 16) - int(start_address, 16))[2:].zfill(6).upper()
object_program.append(f"H^{program_name:<6}^{start_address.zfill(6)}^{program_length}")
```

圖十四

```
# Skip directives not generating object code
if opcode in ["START", "END", "BASE", "RESW", "RESB"]:
    if current_text:
        object_program.append(
            f"T^{current_text_start.zfill(6)}^{current_text_length:02X}^{current_text}"
        )
        current_text, current_text_start, current_text_length = "", None, 0
    continue
```

圖十五

```
if current_text_length + len(obj_code) // 2 > 30:
    object_program.append(
        f"T^{current_text_start.zfill(6)}^{current_text_length:02X}^{current_text}"
    )
    current_text, current_text_start, current_text_length = "", loc, 0
```

圖十六

```
# Generate object code
if opcode.startswith("+"): #format 4
    obj_code = format_object_code(opcode, operand, loc, symtab, base_address)
    print(obj_code)
    if operand[0] != "#":
        modification_loc = f"{int(loc)+1}"
        modification_records.append(f"M^{modification_loc.zfill(6)}^05")
```

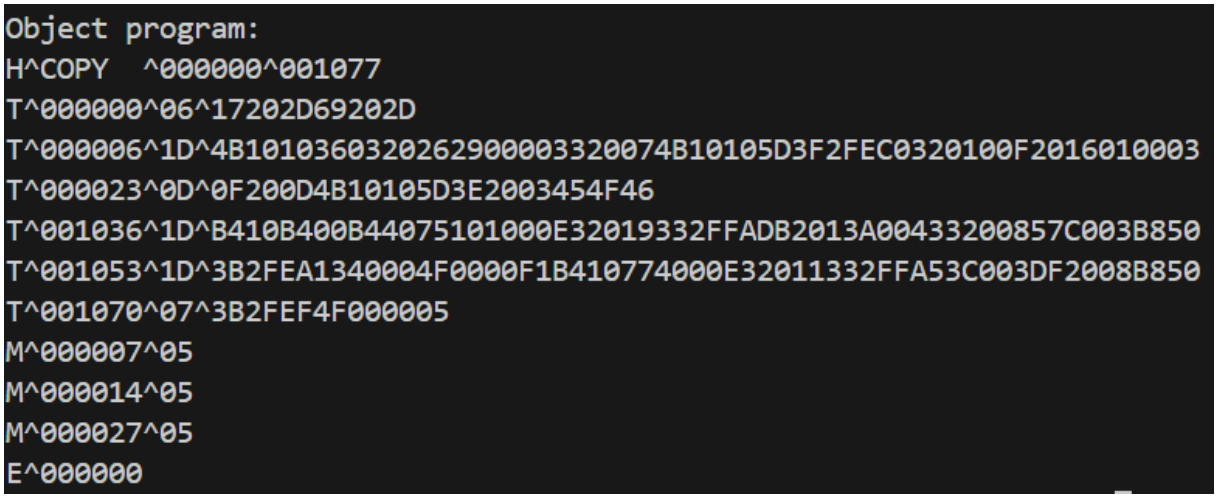
圖十七

```
# End Record
end_label = intermediate_rep[-1]['line'].split()[1]
end_address = symbol_table[end_label]
end = f"E^{end_address.zfill(6)}"
object_program.append(end)
```

圖十八

3. Output：

- **Object program**: 包含 Head record, Text record, Modification record, End record（圖十九）。



圖十九

What you have learned and experienced during the implementation

要把上課學到的理論轉換成實作，這個過程並沒有我想像中的簡單。在實作的時候，我常常因為思慮不周而要反覆檢視整個程式的正確性，特別是在處理 Object code 的部分。

由於每個 opcode 的使用方式和指令的 Format 都不盡相同，導致我在實作這部分的程式時想了很久。後來我決定以最簡單的方式將一些特殊用法的 opcode 和不同 Format 的指令單獨拉出來處理，但這樣做同時也會讓整個程式看起來「寫得很死」，缺乏靈活性。在這個問題上我很糾結：是選擇設計更靈活但可能更複雜的架構，還是簡化邏輯但降低程式的通用性？

考慮到 opcode 和 directives 的使用有被限制在 Figure 2.5 中定義的範圍內，最後決定將程式寫得稍微固定一些，降低了實作的複雜度並確保正確性。

In case you implement more than the required specification

有實作出忽略掉註解的功能。實作額外功能的程式檔案名稱和基本功能的程式檔案名稱相同，為 project.py。而 Test case 的檔案名稱: test.txt。

```
COPY     START   0               . test1
FIRST    STL     RETADR
         LDB     #LENGTH
         BASE    LENGTH
CLOOP    +JSUB   RDREC
         LDA     LENGTH
         COMP    #0              . test2
         JEQ     ENDFIL
         +JSUB   WRREC
         J       CLOOP
. test3
ENDFIL   LDA     EOF
         STA     BUFFER
         LDA     #3
         STA     LENGTH
         +JSUB   WRREC
         J       @RETADR
```

圖二十

片段程式碼（課本中的 Figure 2.5 增加了一些註解）

```
Object program:
H^COPY   ^000000^001077
T^000000^06^17202D69202D
T^000006^1D^4B10103603202629000033200074B10105D3F2FEC0320100F2016010003
T^000023^0D^0F200D4B10105D3E2003454F46
T^001036^1D^B410B400B44075101000E32019332FFADB2013A00433200857C003B850
T^001053^1D^3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T^001070^07^3B2FEF4F000005
M^000007^05
M^000014^05
M^000027^05
E^000000
```

圖二十一

Object program

Copyright Claim

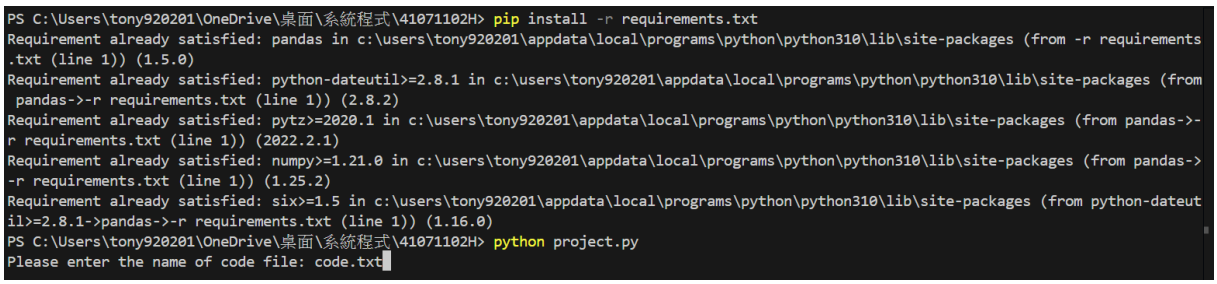
程式碼確實是我徐敏皓撰寫的。

Any thing you would like to let G.H.Hwang know

可能是我自己的問題，我覺得最後面”Advanced interprocess communication”的地方講得有點偏快，回家複習的時候沒什麼印象。

How to run the program

1. 因為在程式中有使用到 pandas 這個 package，所以要先安裝。我有將需要安裝的 package 寫入 requirements.txt，因此只需在 terminal 中輸入 **pip install -r requirements.txt** 就會安裝所需的 package。
2. 安裝完成後把含有 SIC/XE 程式碼的檔案放入資料夾中即可輸入 **python project.py** 執行。
3. 執行後會要求使用者輸入檔案名稱，輸入完後按下 Enter 就會跑出結果。



圖二十二

示範在 terminal 中執行