組譯器之實作（Assembler）

規定一

- CPU Instruction Set

  SIC 及SIC/XE , 808X , Z-80 ,....任選一種
- 處理的指令

  1. Executable Instructions

  2. Pseudo Instruction

  *START/END

  *EQU/ORG

  *Define constant/ storage(BYTE,WORD)

  *LTORG

  *USE BASE register

規定二

- Literal(常數值)

  包括string , character , decimal , hexadecimal
- 算數運算

  +,-,*,/ ， 不必處理括號
- Error Diagnostic，並report Unsolved reference
- 不用處理項目

  Macro , Multiple Segments(僅一個Control section，不用分開data segment , code segment)

繳交期限

- Lexical Analysis （Token Report）

  10月底，不用親自驗收

- System Design Document

  11月中旬

- 程式驗收(上機)

  期中考後一週

System Design Document

- 選用那一個CPU？使用何種程式語言撰寫？使用何種電腦執行？
- 可處理那幾個pseudo Instructions，該pseudo Instruction 做什麼工作？
- Data structure 之設計(重點)

  Instruction Format，Instruction type,…
- Output Format

Example 1 - Hello World!

```
Program: ; Program: Hello World !
         MOV AH, 9
         MOV DX, OFFSET(MESSAGE)
         INT 21H                 ;call DOS
         INT 20H            ;return to DOS
MESSAGE DB 'Hello, World!$'
```

Sample Output

```
LOC  OBJ                LINE         SOURCE
0100                     1      ; Program: Hello World !
0100                     2
0100  B409               3            MOV AH, 9
0102  BA0901             4            MOV DX, OFFSET(MESSAGE)
0105  CD21               5            INT 21H ;call DOS
0107                     6
0107  CD20               7            INT 20H ;return to DOS
0109                     8
0109  48656C6C6F2C       9      MESSAGE  DB 'Hello, World!$'
      20576F726C64
      2124
```

Assembler

- org 100h
- MOV AH, 9
  - p.94, Fig.4.5 #3(Machine Language Coding…)
  - Byte 1 = OpCode, 1011.w.reg = 1011.0.100 = B4h
  - Byte 2 = 09h
- 0100 B4
- 0101 09

Assembler

- MOV DX, OFFSET(MESSAGE)
  - Byte 1 = 1011.w.reg = 1011.1.010 = BAh
  - Byte 2, 3 = Offset(Message)
  - will be found in 2nd pass
- 0102 BA
- 0103 Message(Lo)
- 0104 Message(Hi)

Assembler

- INT 21H
- INT 20H
  - p.99, Fig.4.5 #1
  - Byte 1 = OpCode = 11001101 = CDh
  - Byte 2 = 21h/20h
- 0105 CD
- 0106 21
- 0107 CD
- 0108 20

Assembler

- MESSAGE DB 'Hello, World!$'
- Start at 0109h
- 0109h~0116h = 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21 24
- Fill 0103/0104h (2nd pass)
  - 0103 Message(Lo) = 09h
  - 0104 Message(Hi) = 01h

Example 2 - CLS

```
      mov ah,15
      int 10h
      mov bl,bh
      xor cx,cx
      mov dl,ah
      dec dl
      mov dh,24
      mov bh,7
      cmp al,4
      jb point
      cmp al, 7
      je point
      mov bh,cl
point:mov al,cl
      mov ah,6
      int 10h
      mov ah,2
      mov bh,bl
      mov dx,cx
      int 10h
      int 20h
```

Assembler

- mov ah,15 (1000h)
  - Byte 1 = 1011.0.100 = B4h
  - Byte 2 = 15 = 0fh
- int 10h (1002h)
  - Byte 1 = 11001101 = CDh
  - Byte 2 = 10h
- mov bl,bh (1004h)
  - p.94, %1 #1
  - Byte 1 = 100010.d.w = 100010.1.0 = 8Ah
  - Byte 2 = mod.reg.r/m = 11.011.111 = DFh

Assembler

- xor cx, cx (1006h)
  - p.97 %4 #1
  - Byte 1 = 001100.d.w = 001100.0.1 = 31h
  - Byte 2 = mod.reg.r/m = 11.001.001 = C9h
- mov dl, ah (1008h)
  - Byte 1 = 10001010 = 8Ah
  - Byte 2 = 11.010.100 = D4h

Assembler

- dec dl (100Ah)
  - p.96 %2
  - Byte 1 = 01001.reg = 01001010 (X) (Why?)
  - Byte 1 = 1111111.w = 11111110 = FEh
  - Byte 2 = mod.001.r/m = 11.001.010 = CAh
- mov dh, 24 (100Ch)
  - Byte 1 = 1011.0.110 = B6h
  - Byte 2 = 24 = 18h

Assembler

- mov bh, 7 (100Eh)
  - Byte 1 = 1011.0.111 = B7h
  - Byte 2 = 7 = 07h
- cmp al, 4 (1010h)
  - p.96 %3 #3
  - Byte 1 = 0011110.w = 0011110.0 = 3Ch
  - Byte 2 = 4 = 04h

Assembler

- jb point (1012h)
  - p.98 %2 #8
  - Byte 1 = 01110010 = 72h
  - Byte 2 = shift(point)
- cmp al, 7 (1014h)
  - Byte 1 = 0011110.w = 0011110.0 = 3Ch
  - Byte 2 = 7 = 07h

Assembler

- je point (1016h)
  - p.98 %2 #5
  - Byte 1 = 01110100 = 74h
  - Byte 2 = shift(point)
- mov bh, cl (1018h)
  - Byte 1 = 10001010 = 8Ah
  - Byte 2 = 11.111.001 = F9h

Assembler

- point: mov al, cl (101Ah)
  - remember point address = 101Ah
  - Byte 1 = 10001010 = 8Ah
  - Byte 2 = 11.000.001 = C1h
- mov ah, 6 (101Ch)
  - Byte 1 = 1011.0.100 = B4h
  - Byte 2 = 6 = 06h

Assembler

- int 10h (101Eh)
  - Byte 1 = 11001101 = CDh
  - Byte 2 = 10h
- mov ah, 2 (1020h)
  - Byte 1 = 1011.0.100 = B4h
  - Byte 2 = 2 = 02h
- mov bh, al (1022h)
  - Byte 1 = 10001010 = 8Ah
  - Byte 2 = 11.111.000 = F8h

Assembler

- mov dx, cx (1024h)
  - Byte 1 = 10001011 = 8Bh
  - Byte 2 = 11.010.001 = D1h
- int 10h (1026h)
  - Byte 1 = 11001101 = CDh
  - Byte 2 = 10h
- int 20h (1028h)
  - Byte 1 = 11001101 = CDh
  - Byte 2 = 20h (102ah)

Assembler (2nd pass)

- point mov al, cl (101Ah)
  - remember point address = 101Ah
- jb point (1012h)
  - Byte 2 = shift(point) = 101A - 1014 = 06h
- je point (1016h)
  - Byte 2 = shift(point) = 101A - 1018 = 02h

How to Prove

- Assembler
- Dis-Assembler
- Binary Test using Debug

Assembler之實作步驟

- Lexical Analysis
- Syntax Analysis
- Convert Assembly Instruction to Machine Code
  - Pass1
  - Pass2

Example - Hello World!

```
;_Program:_Hello_World_!←
←
__________MOV_AH,_9←
__________MOV_DX,_OFFSET(MESSAGE)←
__________INT_21H___________________;call DOS←
←
__________INT_20H_____________;return to DOS←
←
MESSAGE_DB__'Hello,_World!$'●
```

ASM Grammar

label:	*opcode*	*operand*	*;comment*
label:	*opcode*	*operand*	
label:	*opcode*		*;comment*
label:	*opcode*		
label:			*;comment*
	opcode	*operand*	*;comment*
	opcode	*operand*	
	opcode		*;comment*
	opcode		
			;comment

Lexical Analysis

- 將輸入的原始程式轉換成Token

MOV	AH	,	9
(1,109)	(3,1)	(4,3)	(6,10)

Syntax Analysis

- 將Token分辨成Token Group (label,opcode,operand),並判斷指令是否合乎文法
- Example 1

MESSAGE	DB	‘Hello,World!$’
Label	opcode	operand(literal)

- Example 2

MOV	DX ,	
Opcode	operand	error

Instruction Table 1

47	INT
48	IN
109	MOV

Pseudo and Extra Table 2

1	CODE
2	SEGMENT
3	PROC
4	NEAR
5	ASSUME
6	ORG
7	DB
8	DW
9	EQU
10	ENDP
11	ENDS
12	END
13	WORD
14	BYTE
15	PTR
16	DUP
17	OFFSET

Register table 3

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	

Delimiter Table 4

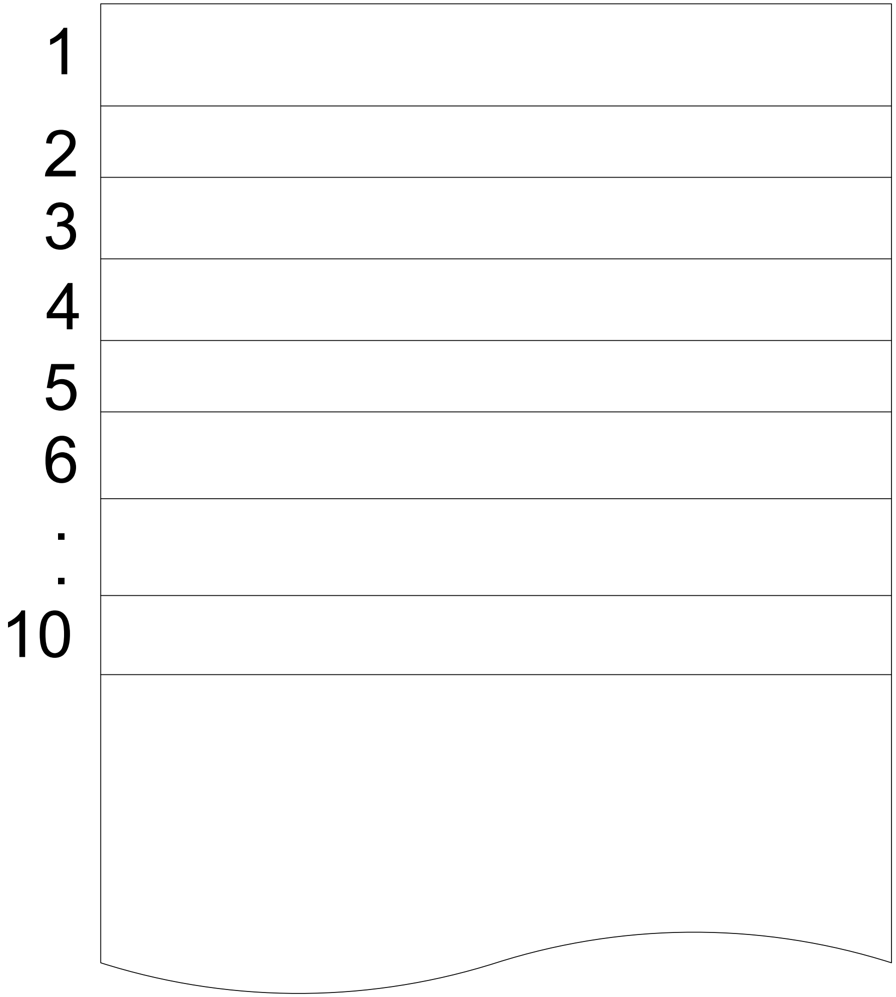
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	

Symbol Table 5

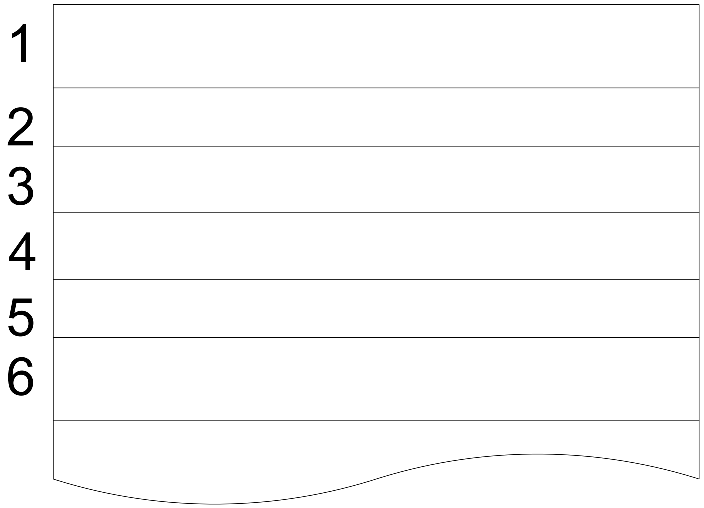
1	
2	
3	
4	
5	
6	

ME

Integer/Real Table 6



String Table 7



Hello,

Useless information for assembler

- *Space/Tab*
- *Enter*
  - *only used for determining the end of line*
- *Comment*
  - *begin with semicolon (;)*
- *Comma*
  - *used for dividing operand/literal*
- *Colon*
  - *end of label (as language definition)*

Example 1 - Hello World!

```
; Program: Hello World !
         MOV AH, 9
         MOV DX, OFFSET(MESSAGE)
         INT 21H                ;call DOS
         INT 20H             ;return to DOS
MESSAGE DB 'Hello, World!$'
```

Lexical Analyzer

MOV	AH	,	9			
(1,109)	(3,1)	(4,3)	(6,10)			
MOV	DX	,	OFFSET	(	MESSAGE	)
(1,109)	(3,12)	(4,3)	(2,17)	(4,11)	(5,1)	(4,12)
INT	21H					
(1,47)	(6,5)					
INT	20H					
(1,47)	(6,4)					
MESSAGE	DB	`	Hello,World!$	`		
(5,1)	(2,7)	(4,13)	(7,1)	(4,13)		

如何切Token

- Space/Tab/Enter (white space)
- Delimiter

[，] ，, ，+ ，- ，…

範例:

MOV WORD PTR [BP] [DI] + 1234H

Lexical Analysis 方法

- 找到white space 或 Delimiter
- 當遇到white space,到各table內查是否爲預先設定之指令,符號,…等,如果是則建立token
- 當遇到Delimiter,則到各table內查並建立token(可能有一個或兩個token)
- 若查表沒有此token,表示它爲symbol或integer/Real或String,以Hashing function將其放入table內

Hash function

- 將identifier 中的每個字元的ASCII 碼相加之後取 100 的餘數
- 有碰撞產生,就向後遞增至空的地方

作業繳交

- Lexical Analysis (Token Report)
- 10月下旬

Syntax Analysis實作

- 分辨為token,並依文法需求保留下label,opcode,opeand資訊

Useful information for assembler

- *opcode*
  - – *MOV, ADD, JP, …*
  - – *can be stored in a table to access easily*
- *operand*
  - – *“AX,09h” “AL, Label+2” “dx, offset(A)” …*
  - – *have to be divided into several parts*
- *label*
  - – *“A DB ‘1234$’” “A: MOV AX, BX”… .*
  - – *recognize and store in a different table*

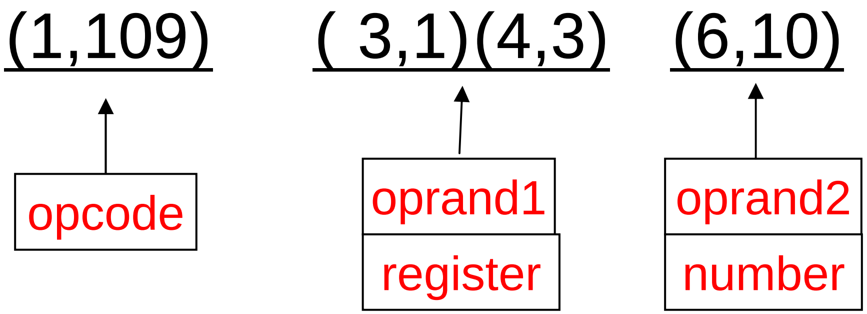
Operand grammar

- *2-parameter operand*
  - *REG, REG*
  - *REG, address*
  - *REG, number*
  - *…*
- *1-parameter operand*
  - *label*
  - *offset(something)*
  - *…*

Lexical Analysis及Syntax Analysis後之結果

- Example 1

MOV AH , 9



(1,109) (3,1)(4,3) (6,10) ←token

←token Group

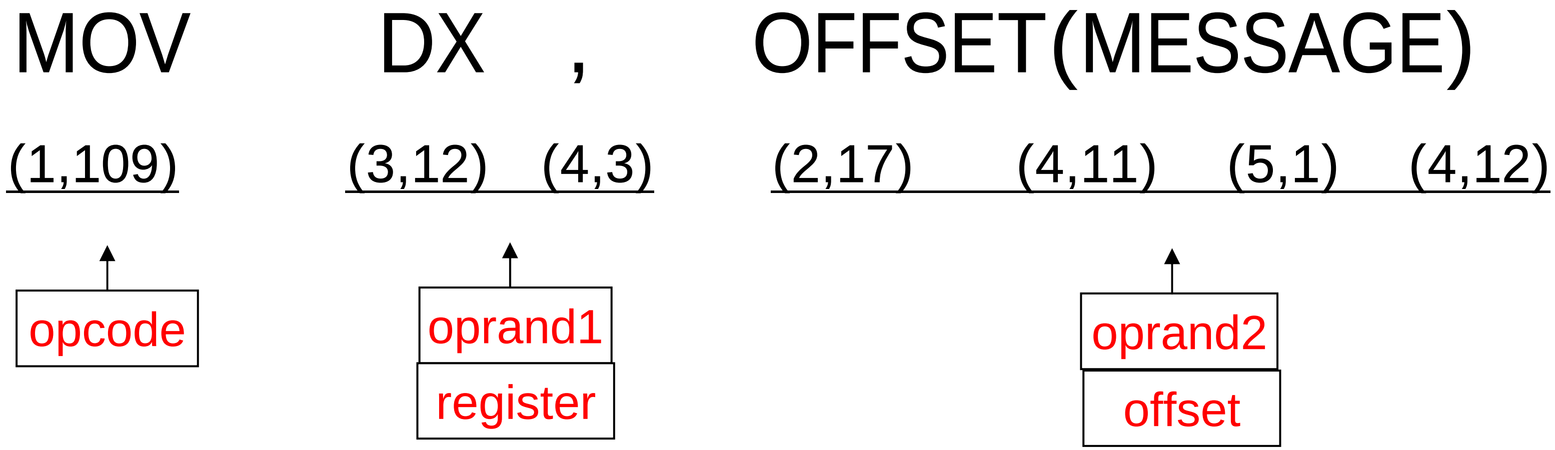
opcode

oprand1
register

oprand2
number

Lexical Analysis及Syntax Analysis後之結果

- Example 2



MOV DX , OFFSET(MESSAGE)

(1,109) (3,12) (4,3) (2,17) (4,11) (5,1) (4,12)

opcode

oprand1

register

oprand2

offset

如何翻Machine code

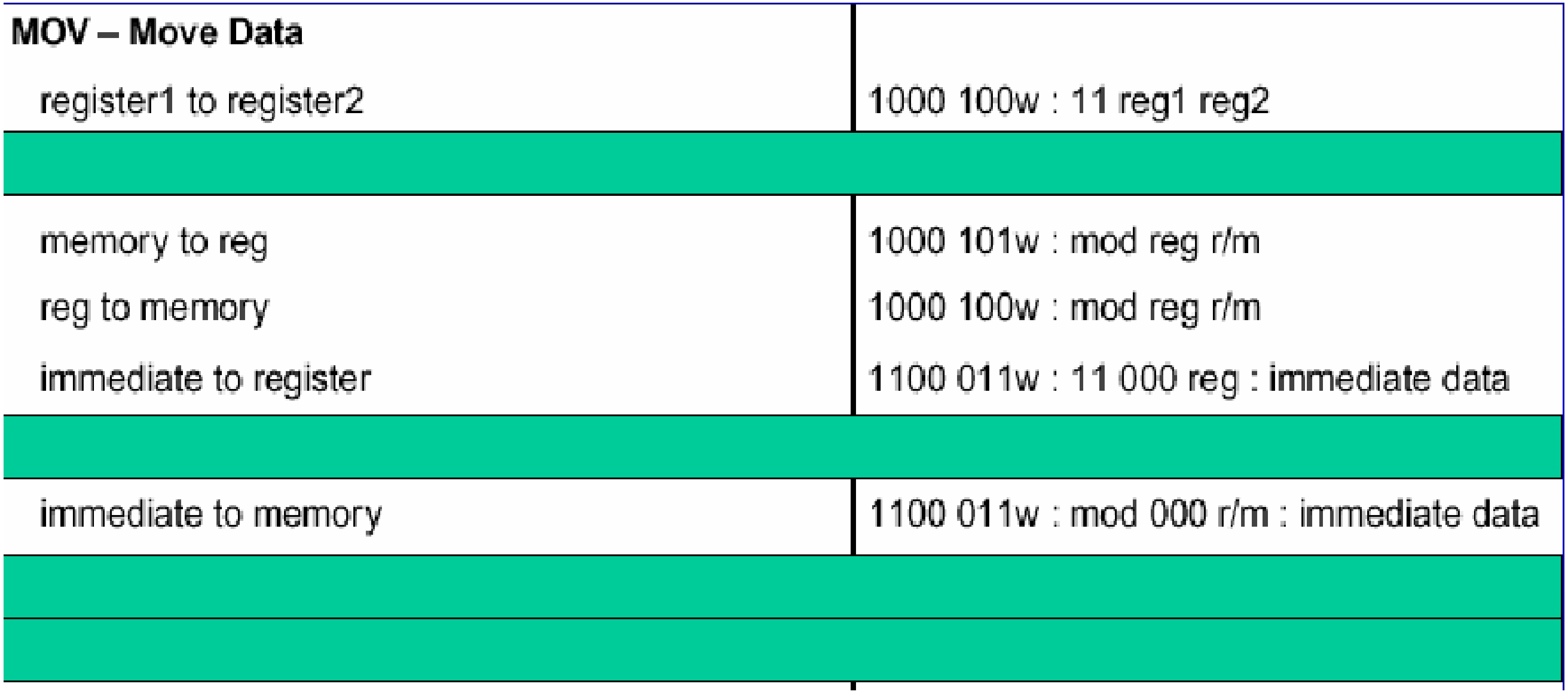
- 指令分類
- 各指令與機器碼的對照表
- Symbol Table之進一步考慮
- 其他必須之tables

ASMer Writing Techniques

指令行	*OP Code*	同類項
AAD	D5	0001
AAM	D4	0001
ADD	00	0020
AND	20	0020
CMP	38	0020
OR	08	0020
SBB	18	0020
SUB	28	0020
XOR	30	0020

Encode MOV Instruction

- *MOV instruction format (Partial)*

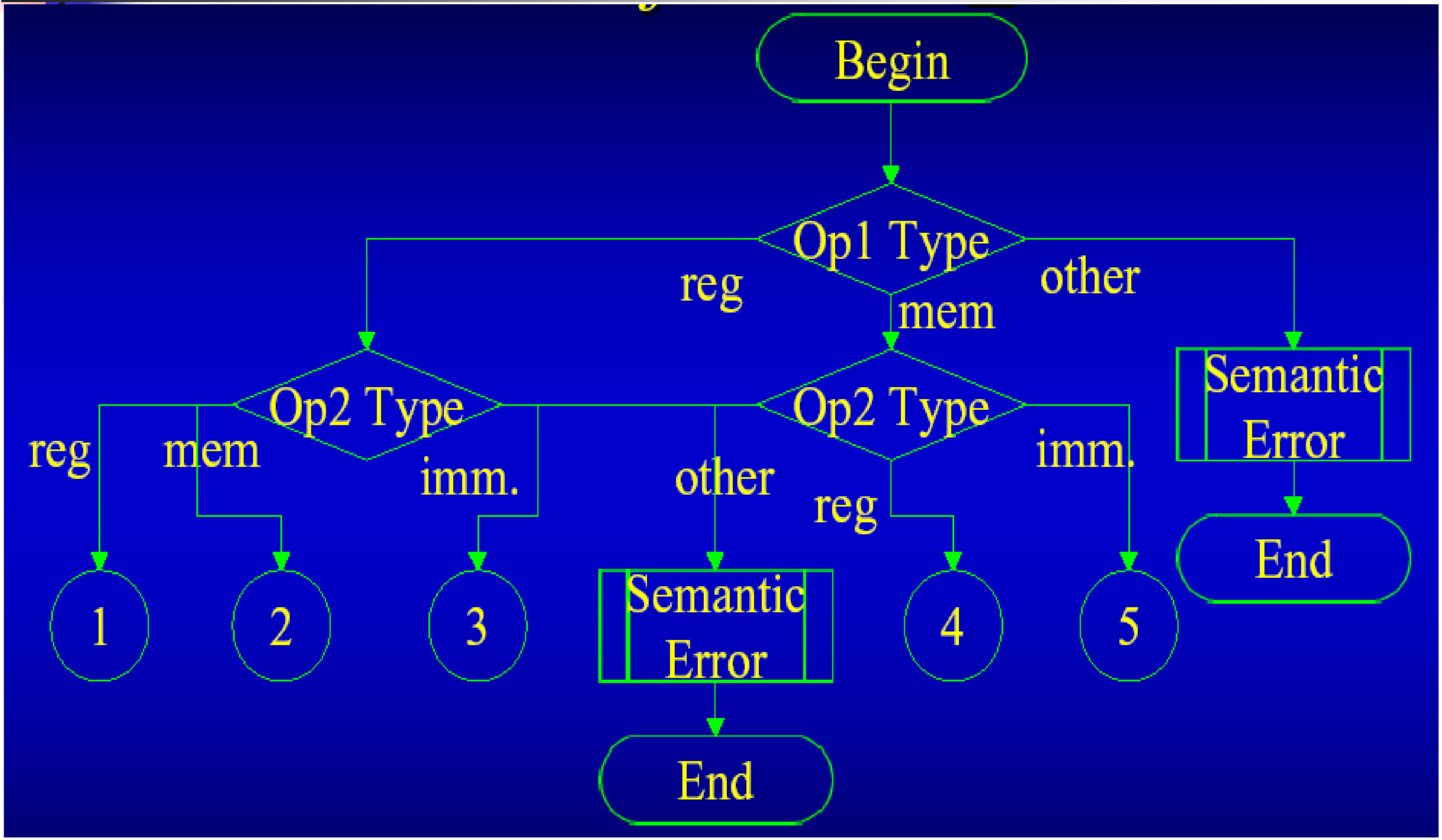


MOV – Move Data	
register1 to register2	1000 100w : 11 reg1 reg2
memory to reg	1000 101w : mod reg r/m
reg to memory	1000 100w : mod reg r/m
immediate to register	1100 011w : 11 000 reg : immediate data
immediate to memory	1100 011w : mod 000 r/m : immediate data

Encode Data

Op2	*Op1*	*OPCode*	*D*	*W*	*Mod*
REG	REG	100010	0	?	11
MEM	REG	100010	0	?	??
IMM	REG	110001	1	?	11 no r/m
REG	MEM	100010	1	?	??
IMM	MEM	110001	1	?	?? no reg

Flowchart of Handle_MOV



Kinds of Handle_MOV

OpCode	*D*	*W*	*MOD*	*REG*	*R/M*
1. 100010	0	Test OP1	11	OP1	OP2
2. 100010	0	Test OP1	??	OP1	OP2
3. 110001	1	Test OP1	11	000	OP1
4. 100010	1	Test OP1	??	OP1	OP2
5. 110001	1	Test OP1	??	000	OP1

Something about MOV

- *Check for Semantic Error*
  - *Think about “data type” of operands.*
  - *Check for type matching*
    - *Byte*
    - *Word*
    - *DWord*
  - *Check for Destination operand*
    - *no literal*

Instruction Table Lookup

Name	*Operand#*	*Length#*	*OpCode*
Add	2	x	…
Mov	2	x	…
Jmp	1	x	…
Nop	0	1	…
Start	0	0	…
End	0	0	…

Label/Symbol Table Lookup

Name	*Start*	*End*	*Type*
Msg	*0000*	*0010*	*string*
Num1	*0011*	*0012*	*word*
Num2	*0013*	*0013*	*byte*
…	*…*	*…*	*…*

程式驗收

- 期中考後一周

輸出格式

```
Microsoft (R) Macro Assembler Version 6.1a              07/30/99 09:20:50
test13.asm                                                    Page 1 - 1

 0000                        CODE       SEGMENT
 0000                        Mycode     PROC        NEAR
                                        ASSUME      CS:CODE
                                        ORG         0
 0000  47 72 65 65 6E 20     Msg        BYTE        'Green '
 0006  47 72 65 65 6E 20                BYTE        'Green '
 000C  47 72 61 73 73 20                BYTE        'Grass '
 0012  48 6F 6D 65                      BYTE        'Home'
 0016  0A 0D 24              LF         BYTE        0AH, 0DH, '$'
 0019  2E: A1 0000 R                    MOV         AX,WORD PTR Msg
 001D  8E D8                            MOV         DS,AX
 001F  2E: 8B 16 0000 R                 MOV         DX,WORD PTR Msg
 0024  E8 0004                          CALL        DispMsg
 0027  B4 4C                            MOV         AH,4CH
 0029  CD 21                            INT         21H
 002B                        Mycode     ENDP
 002B                        DispMsg    PROC        NEAR
 002B  B4 09                            MOV         AH,09H
 002D  CD 21                            INT         21H
 002F  C3                               RET
 0030                        DispMsg    ENDP
 0030                        CODE       ENDS
                                        END         Mycode
```

輸出格式

```
Microsoft (R) Macro Assembler Version 6.1a              07/30/99 09:20:50
test13.asm                                         Symbols 2 - 1

Segments and Groups:
Name                        Size      Length    Align     Combine Class
CODE . . . . . . . . . . . . . . .    16 Bit    0030      Para        Private

Procedures,  parameters and locals:
Name                        Type      Value     Attr
DispMsg   . . . . . . . . . . . . .   P Near    002B       CODE Length= 0005 Private
Mycode . . . . . . . . . . . . . .    P Near    0000       CODE Length= 002B Private

Symbols:
Name                        Type      Value     Attr
LF . . . . . . . . . . . . . . . .    Byte      0016       CODE
Msg   . . . . . . . . . . . . . . .   Byte      0000       CODE

          0 Warnings
          0 Errors
```

組譯器輸出之報表