연구논문집 제10권 제2호 p.7~15 1979.

Facom OS Ⅳ Fast 開發過程 및 內部論理에 대해서

高 在 鎭
電子計算學科

〈要 約〉

學生들의 Fortran교육을 위한 Fortran Compiler인 Fast의 開發배경, 開發過程, 開發소요비용, 개발에 필요한 Resources, 그리고 Fast의 實行論理, implementation方法등을 서술하였다.

The Development of the Facom OS Ⅳ Fast and It's Internal Logics

Ko, Jae-Jin
Department of Computer Science

〈**Abstract**〉

This Paper Describes the Backgrounds, Processes, Costs, Resources of the Fast Development and the Execution Logic, Implemetation Methods, etc of the Fast Compiler.

Ⅰ. 序 文

Facom OS Ⅳ Fast는 1976年 4月부터 日本의 富士通(株)의 Software開發業務의 受注를 받아서 화콤코리아株式會社의 技術開發部에서 開發에 着手했다. Fast라는 이름은 Fortran to Assist Students' Training라는 말의 略字이다. 즉, 學生들의 敎育을 돕기 위한 Fortran Compiler라는 뜻이다. 學校에서 學生들에게 EDPS敎育을 實施할때 實習을 Fortran言語로 하는게 普通이다. 그래서, 一般的으로 學校에서 Fortran의 使用頻度가 높아서 Computer의 CPU占有率을 많이 차지하게 된다. 一般汎用 Fortran Compiler을 使用하면 Compile時間, 實行時間이 증가해서 다른 一般硏究用 Job이라든지, 事務處理用 Job의 實行에 支障을 받아서 Computer의 利用効率을 떨어지게 한다.

이런 觀點에서, 比較的論理가 簡單하고, 文數가 적고, 다량의 Job을 一括的으로 빠르게 처리할 수 있는 Fortran Compiler의 開發을 構想하게 된 것이다. 그래서, 敎育用 Fortran Compiler인 Fast의 開發에 着手하게 된 것이다. EDPS敎育의 入門者인 學生들은 비교적 처음投入하는 Job에 Error를 범하는게 보통이고, 단순한 入出力機能을 使用한다. 그래서, 敎育用 Fortran인 Fast에서는 理解하기 쉽고 친절한 Error Message를 내도록 했으며, 단순한 形態의 入出力文을 도입했다. 이러한 用途의 Fast는 다른 汎用 Fortran Job에도 有用한 面이 있다. 一般的으로 論理가 복잡하고, Program Size가 큰 Fortran Program을 作成初期段階에서는 Fast을 利用해서 Compile & Run을 해보고, 모든 Error가 해결되면 汎用 Fortran Compiler을 사용해서 Compile & Run하면 比較的 빠른 시간에 Progran을 完成할 수 있다.

Facom OS Ⅳ Fast는 그 文法仕樣이 Fortran Ⅳ以上의 水準이고 그 處理能力이 最大 1000枚 以上의 文을 處理할 수 있다. 이 다음 章부터 Facom OS Ⅳ Fast의 開發過程및 內部論理에 대해서 記述하겠다.

Ⅱ. Fast의 開發過程

Fast의 開發은 1976年 4月부터 시작하여 1978年 10月에 完了하였다. OS Ⅳ/F4 Fast는 1976年 4月~1978年 3月에 開發하였고, OS Ⅳ/×8 Fast, OS Ⅳ/F2 Fast는 1978年 4月~1978年 10月에 OS Ⅳ/F4 Fast를 改造하여 開發하였다.

合하여 3種의 Compiler를 開發하는데 2年 7個月의 期間이 걸렸다. 工數는 332人月이 걸렸다. 즉, 332人月을 32月로 나누면 平均 11人이 2年 7個月동안 이 Project를 開發하는데 從事하였다는 뜻이다. 計算機時間은 1811時間이 걸렸다. 月平均 60時間의 計算機時間이 소모되었다. 이러한 막대한 人力과 費用이 投入된 Fast의 開發過程을 記述해나가겠다.

1. 開發背景

大學이나 硏究機關에서 敎育用 Fortran Compiler의 要求가 증가하였다. 보통 汎用 Fortran Compiler는 Compile 過程이 多 Pass로서 Program이 Overlay 構造로 되어있다. 우리가 開發한 Fast는 Compile 過程이 One Pass로서 Program이 單純構造로 되어있다. 그래서, Fast는 構造上 汎用 Fortran Compiler보다 Compile速度가 빠르게 되어 있다.

그리고, 汎用 Fortran Compiler는 Compile이 끝나서 Object Code를 出力한다음, i/o Module과 Function Module을 出力한 Object Code와 Linkage Edit하여야 하고, 그 結果인 Load Module을 Load하여 實行하여야 한다.

그러나, Fast는 Compiler 自體內에 *i/o* Module과 Function Module이 붙어 있어서 Linkage Edit할 필요가 없고 Object Code를 主記憶上의 作業領域에 出力하여서 Compile이 끝난직후 바로 실행에 들어갈 수 있다.

敎育用 Fortran Compiler는 大學이나 硏究機關에서 自體開發한것이 이러번 있다. 그 言語仕樣이나 Program size가 小規模的인 것이 大部分이었다. FAST는 言語仕樣이 汎用 Fortran Compiler와 互換性이 있고, 그 處理能力이나 Speed 및 各種機能이 大規模的인 것이 特徵이다.

이러한 要求에 의하여 Fast는 開發되었다.

2. Fast의 特徵

(1) One-Pass 主記憶常駐形 敎育用 Fortran Compiler이다.

(2) 再入可能한(Reentrant) 單純 Program構造를 하고 있다.

(3) Compile速度가 汎用 Fortran Compiler에 비해 빠르다.

(4) 飜譯·實行을 通하여 親切하고 理解하기 쉬운 診斷 Message를 내고 있다.

(5) Compile이 끝나면 Linkage Edit절차없이 바로 實行에 들어간다.

(6) 여러개의 Fortran Program을 連續的으로 處理한다.

(7) 簡單한 Job制御文으로 處理할 수 있다.

(8) 汎用 Fortran Compiler와 互換性이 있다.

(9) Logging機能이 있어서 各 Program마다 各種情報를 收集할 수 있다.

3. 外部仕樣

(1) 文 法

OS Ⅳ Fast의 言語仕樣은 「Facom OS Ⅳ Fortran 文法書」에 의하고 있으나, Compiler의 設計上, Compiler Size, User가 본 必要度에 의하여 OS Ⅳ/F2 Fortran의 言語仕樣에 準하고 있다.

단, 다음과 같은 制限項目과 擴張項目이 있다.

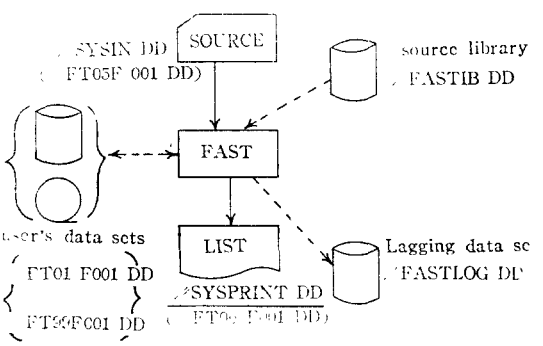
① 制限되는 項目

a. 他言語와의 結合은 할 수 없다.

b. Generic文은 使用할 수 없다.

c. Do의 擴張範圍는 使用할 수 없다.

System構成



※…線은 Option을 意味한다.

d. 다음의 基本 Subroutine은 使用할 수 없다.
Dump, Pdump

e. Debug文, Debug Packet은 使用할 수 없다.

f. "Format"을 豫約語로 한다.

② 擴張되는 項目

a. Data文의 Do型 list의 使用을 許容한다.

b. 文關數의 式에 配列要素을 許容한다.

(3) Install Parameter

Fast를 User의 Environment(環境)에 맞게 하도록 하기 위해서, Install Fast Compiler의 一部分으로서 一連의 Assembly Parameter Set가 提供된다.

以下, 各 Parameter을 列擧하겠다.

단 一친것은 표준치를 의미 한다.

① 繼續行의 許容枚數 $=\left\{\frac{5}{n}\right\}$

이 Option은 Source Statement을 Scan하기 위한 Text Area의 크기를 決定하기 위한 것이다. 繼續行 1行當 280 byte의 Text Area가 確保된다.

② 1 page當 行數 $=\left\{\frac{60}{n}\right\}$

③ 實行時出力 List의 Page打切枚數 $=\left\{\frac{30}{n}\right\}$

④ Program의 實行打切時間(CPU) $=\left\{\frac{20}{n}\right\}$

但, 單位는 1/10秒로 한다.

⑤ Logging情報의 取捨 $=\left\{\frac{\text{NO}}{\text{YES}}\right\}$

⑥ Program의 Error 打切回數 $=\left\{\frac{30}{n}\right\}$

⑦ Source入力媒體의 選擇 $=\left\{\frac{\text{CARD}}{\text{OMR}}\right\}$

⑧ Run Mode $=\left\{\begin{array}{c}\underline{\text{CHECK}}\\ \text{NOCHECK}\\ \text{FREE}\end{array}\right\}$

Check: Serious Error가 없으면 實行하고 實行時의 Error를 Check한다. Undefined Variable의 Check를 한다.

Nocheck: Check Mode와 같으나, Undefined Variable의 Check를 하지 않는다.

Free: Check Mode와 같으나, Serious Error가 있어도 實行을 繼續한다.

⑨ Keypunch Code $=\left\{\frac{\text{EBCDIC}}{\text{BCD}}\right\}$

⑩ Source Library의 使用 $=\left\{\frac{\text{NO}}{\text{YES}}\right\}$

⑪ Error Message의 選擇 $=\left\{\frac{\text{SHORT}}{\text{CODE}}\right\}$

⑫ Function Level의 選擇 $=\left\{\begin{array}{c}0\\ 1\\ 2\end{array}\right\}$

⑬ Direct機能의 選擇 $=\left\{\frac{\text{YES}}{\text{NO}}\right\}$

(4) Compile · 實行方法

Fast의 Compile · 實行의 單位는 Fast Batch와 FAST JOB의 두 段階로 되어 있다. 各 Fast Job은 1個의 Fast Batch制御밑에서 Compile · 實行된다.

Fast Batch는

```
// EXEC PGM=JZCFAST, REGION=□,
  PARM='_______'부터 시작해서·
/ *
```

로 끝난다. 그 사이에 各 Fast Job이 있게 된다.

例
```
// EXEC PGM=JZCFAST, REGION=□,
  PARM='_______'
//FT06F001 DD Sysout = A
//FT05F001 DD *
        Fast Job
/ *
```

"Region=□은 Fast가 動作하는 Region의 크기를 指定한다.

"Parm='_____'"은 이 Fast Batch에서 適用되는 Parameter이다.

Parm Parameter은 다음과 같다.

① Linecount(n)

Compiler가 出力하는 List의 1page當 行數를 1~99의 10進數로서 指定한다.

② Homeskip

Source Program List을 出力할때, 各 出力의 單位(Main, Sub-Program)마다 Page의 先頭에 맞추어서 印刷하고 싶을때 指定한다.

③ $\left\{\begin{array}{l}\text{Log}\\ \text{Nolog}\end{array}\right\}$

Fast에는 Logging機能이 準備되어 있어서 各 Fast Job마다의 Logging情報의 取捨를 定한다.

Log:Logging情報를 낸다.

Nolog:Logging情報를 내지 않는다.

④ { Card / OMR }

Source Program의 入力形式을 指定한다.

Card: Card image로서 入力한다.

OMR: OMR image로서 入力한다.

⑤ { Lib / Nolib }

Source Library를 使用할 것인가 안할 것인가를 指定한다.

Lib: Source Library를 使用한다.

Nolib: 〃 使用안한다.

Fast Job은

/〔Fast-Job-Name〕 Fast 〔Fast-Parameter〕

로 부터 시작해서

다음의

/〔 〕 Fast 〔 〕

카드 또는 .

/*

카드 까지이다.

만약 data가 있으면 Source Program과 data사이에

/〔data-name〕 Data

카드를 넣는다.

例
```
/〔Fast-job-name〕 Fast 〔Fast-paremeter〕
        ⌇
  Source Statements
        ⌇
/〔data-name〕 Data
        ⌇
      data
        ⌇
/〔Fast-job-name〕 Fast 〔Fast-parameter〕
또는
/*
```

다음에

/〔Fast-job-name〕 Fast 〔Fast-parameter〕

카드의 各 PARAMETER에 대해서 說明한다.

① 〔Fast-job-name〕

한개의 實行可能한 Program 名 또는 다른 目的에 自由스럽게 사용 할 수 있는 이름으로 쓸 수 있다. 記述할 때는 /의 바로 다음 2 Column부터 시작해야 한다. 1～6字의 英字名으로서 Logging要素로서 使用된다.

② 〔Fast-parameter〕에 대해서

〔Fast-parameter〕는 다음과 같은 種類가 있다.

a. "Code=_____"

課題番號等에 使用하면 좋다. (다른 目的에도 使用할 수 있다)

1～8行의 英數字를 指定한다. 省略한 경우의 Code는 Blank Logging이 된다. 이것은 File에 出力된다.

b. "Nogo"

Compile만을 할 경우 記述한다.

省略한 경우는 Compile에서 實行까지 한다.

c. { Check / Nocheck / Free }

Check: Serious Error가 없으면 實行하고 實行時 Error를 Check한다. Undefined Variable Check를 한다.

Nocheck: Check Mode와 같으나. Undefined Variable Check를 하지 않는다.

Free: Check Mode와 같으나 Serious Error가 있어도 實行을 繼續한다.

d. "Timelimit(*n*)"

한개의 實行可能한 Program의 實行打切 時間 (CPU) 但, 1/10秒單位로서 指定한다.

e. "pagelimit(*n*)"

實行時 出力 List의 Page打切枚數를 指定한다.

f. "Errlimit(*n*)"

한개의 實行可能한 Program Error의 打切回數를 指定한다.

g. "Pause"

이 指定에 의해 Pause文이 有効가 된다.

省略한 경우는 Pause文에 대해서 warning message를 내고 Nop處理한다.

h. "{EBCDIC / BCD}"

Source Program이 EBCDIC Code로 되어 있는지 BCD Code로 되어 있는지를 指定한다.

4. 工程에 대해서

區 分	'76 4	5	6	7	8	9	10	11	12	'77 1	2	3	4	5	6	7	計
工 程		F4 R	S	▲		F4 B	D	▲		F4 FD	▲		F4 DD	▲	F4 MK	1 ▲	
工 數(人)	9	8	8	8	8	8	8	8	10	10	10	11	17	17	17	17	174人月
計算機時間(H)	*i*135 0	0	0	0	0	0	0	0	0	0	0.5	0.5	2	6	23	53	(*i*135) 85*H*

區 分	'77 8	9	10	11	12	'78 1	2	3	4	5	6	7	8	9	10	11	計
工 程	F4 MK_2	MK_3	▲			F4 CT		▲			F4 保守				▲		
									F2 FD ▲	F2 DD ▲	F2 MK 1/2 ▲	F2 MK 3 ▲	F2 C	T ▲	F2 保守 ▲		
									X8 FD/	DD ▲	X8 MK ▲	X8 C	T ▲	X8 保	守▲		
工 數(人)	15	15	15	13	13	13	13	13	8	8	8	8	8	8			158人月
計算機時間(H)	*i*135 97	114	91	59	34	M 160 F4 74	130	190	125	82	119	82	87	28			(*i*135) 395H (M160) F4=917H
									F2 0	0	25.5	56.5	59	107			F2=248H
									X8 9	15	13.5	31	56.5	41			X8=166H

Ⅲ. Fast의 內部論理

Fast는 主記憶常駐型 Compiler로서 Re-entrant 再入可能構造로 되어있다. Compile形式은 One Pass로서 文單位로 Compile한다. 各文마다 中間 Text를 만들어서 그 文의 種類와 屬性을 Identify 해서 該當하는 Processor에 分岐한다. 各 Processor 에서는 生成된 中間 Text를 基本으로해서 作業領域(Work Area)에 Symbol Table을 作成하고, Object Code를 出力한다. 이 Object Code는 Symbol Table에의 Pointer를 갖고 있다.

各文마다 上記와 같이 Symbol Table과 Object Code를 生成해 나간다.

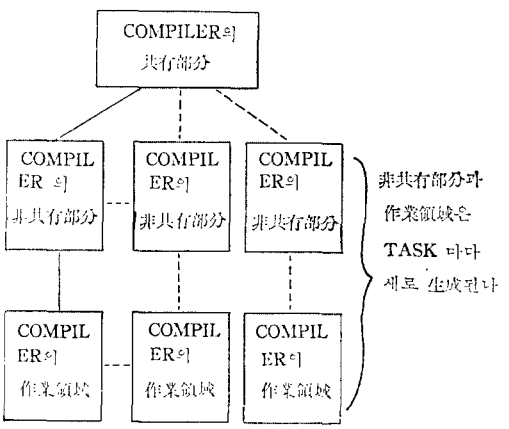
Program單位의 끝(end文)을 만나면 Relocation Phase에 分岐한다. Relocation Phase에서는 Program 單位마다 生成되어 있는 Symbol Table에 대해서 領域을 割當한다. 또 Object Code中의 Symbol Table에의 Pointer을 割當된 領域의 番地로 置換한다.

한개의 實行可能한 Program의 끝(/Data 또는 /Fast)을 만나면 Linkage Phase에 分岐한다.

Linkage Phase에서는 Subprogram 名, Entry 名, Common Block名에 대해서 外部 Symbol參照의 解決과 Common 領域을 割當한다. Linkage가 끝나면 實行에 들어가게 된다.

1. 實行論理

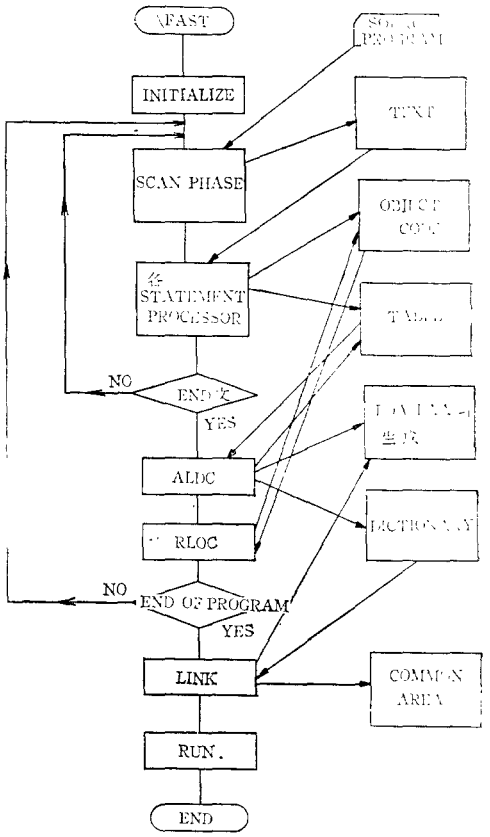
(1) Memory Layout



(2) 作業領域(Work Area)의 Memory Layout

Compile Time	Execution Time
標準入力, 標準出力에 대한 DCB, DECB, Buffer	標準入力, 標準出力에 대한 DCB, DECB, Buffer
Object Code for Main Prog.	Object Code for Main Prog.
LDA for Main	LDA for Main
Object Code for Sub-Program A	Object code for Sub-Program A
LDA for sub A	LDA for Sub A
⋮	⋮
	Object Code for Last Sub-Prog
現在 Compile中의 Sub-Prog의 各種 Symbol Table	LDA for Last Sub-Prog
	實行時 I/O Unit에 대한 DCB, DECB Buffer
	Common Area
⋮	LAA for Last Sub-Prog
Sub-Prog A의 Dictionary	⋮
	LAA for Sub-Prog A
Main Prog의 Dictionary	LAA for Main Prog

(3) Compiler의 實行論理



(4) Compiler의 實行例

① Source Program

isn	/Fast
1	A=1. 0
2	B=2.0
3	C=A+B
4	Stop
5	End

② 各文마다 作成되는 Text의 內容

a. A=1.0

Link	Operator	Code	Operand
8	ㅓ	01	A
8	=	81	$(1)_2$
8	.	81	$(0)_2$
0	ㅏ	00	

b. B=2.0

Link	Operator	Code	Operand
8	ㅓ	0 1	B
8	=	8 1	$(2)_2$
8	.	8 1	$(0)_2$
0	ㅏ	00	

c. C=A+B

Link	Operator	Code	Operand
8	ㅓ	01	C
8	=	01	A
8	ㅗ	01	B
0	ㅏ	00	

d. Stop

Link	Operator	Code	Operand
8	ㅓ	01	Stop
0	ㅏ	00	

e. End

Link	Operator	Code	Operand
8	ㅏ	01	End
0	ㅓ	00	

③ Source Program에 의해서 生成된 Symbol Table Object Code

```
BAL R11, ISNRTN
DC    H▼ 1▼
LE    Fϕ, 9FEF
STE   Fϕ, 9FEF

BAL   R11, ISNRTN
DC    H▼ 2▼
LE    Fϕ, 9FCF
STE   Fϕ, EDF

BAL   R11, ISNRTN
DC    H▼ 3▼
LE    Fϕ, 9FFF
AE    Fϕ, 9FDF
STE   Fϕ, 9FBF

BAL   R11, ISNRTN
DC    H▼ 4▼
```

番地	LINK	屬性	SYMBOL
9FBF	ϕ	單變純數	C UUUUU
9FCF	ϕ	實定數	2 ϕ
9FDF	32	單變純數	B UUUUU
9FEF	32	實定數	1. ϕ
9FFF	32	單變純數	A UUUUU

2. 設計의 特徵

(1) Compiler의 作業領域을 Getmain Macro命令에 의해서 動的으로 確保한다.

(2) 共通制御情報를 한개의 Phase에 集中하였다

(3) Text는 Operator, Operand Pair의 List構造로 되어 있고 Symbol Table作成, Object Code 出力과 Syntax Check의 母體가 된다.

(4) Compile時, 實行時의 共通 Routine을 各各 한개의 Phase로 集中하였다.

(5) Symbol Table은 그 屬性에 따라, 連續的으로 Link된 List構造로 되어 있다.

(6) Symbol Table에는 變數名 Table, Sub-Program名 Table, Namelist名 Table, Common Block名 Table, 文番號 Table, Hollerith Constant Table, 常數 Table, 汎用英字名 Table, 等 8個의 種類가 있다.

(7) Symbol Table Look-up에 Hash Technique을 使用했다.

(8) Symbol Table Relocation이 끝난 다음 汎用英字名 Table을 移動시키고, 다른 Table을 전부 Clear시켰다.

(9) 各文의 Object Code의 先頭에 ISN Coding을 出力시켰다.

(10) Object Code의 Relocation을 위해서 Relocation Code를 使用하였다.

(11) Program interrupt處理出口 Routine을 作成했다.

3. 共通情報

共通情報는 다음과 같은 것으로 構成되어있다.

(1) Install Parameter情報

Install時 決定되는 값이 이 속에 포함된다. 이 內容은 Parm Parameter, Fast Parameter의 解析에 의해서 變更될 수가 있다.

(2) Fast Job制御情報

한개의 Fast Job을 處理하기 위해서 必要한 制御情報이다.

이 內容은 한개의 Program 實行可能한 單位마다 初期化된다.

(3) 飜譯單位制御情報

한개의 Program單位를 處理하기 위해서 必要한 制御情報이다.

이 內容은 한개의 Program單位마다 初期化된다

4. 制御 Phase

制御 Phase는 다음과 같은 機能을 遂行한다.

① Compiler의 初期化
② Fast Job의 初期化
③ 作業領域(Work Area)의 確保
④ Logging情報의 管理
⑤ Source Library 管理
⑥ 各種 Parameter의 解析

5. Compile Phase

Compile Phase에는 다음과 같은 Phase가 있다. 各 Phase別 機能을 記述하겠다.

(1) Compile時의 共通 Routine

① Symbol Table Lookup
② Object Code出力
③ Constant Collecting
④ Text作成
⑤ 各種 Switch와 Constant의 Setting
⑥ Compile時의 入出力

(2) Source-Scan Processor

① Source Statement을 中間 Text Format으로 變換
② 文 Type을 分析해서 各 該當 Processor에 制御를 넘겨줌.
③ 文番號를 處理해서 文番號 Table을 作成

(3) 算術式處理 Processor

① 各文에 있는 術算表現을 Compile한다.
② Call文, 算術 if文, 論理 if文을 Compile한다.

(4) Do文과 分岐分處理 Processor

① Do Loop의 Compile (implied do포함)
② 모든 分岐文 (Go To文)의 Compile
③ Assign文의 Compile
④ 算術 if文의分岐部分의 Compile

(5) 入口文處理 Processor

① subroutine文, Function文, Entry文, Return文, End文을 Compile한다.
② 새로운 Program單位의 Compile을 위한 初期化

(6) 宣言文處理 Processor

① 모든 型宣言文의 Compile
② Equivalence文, Common文, Implicit文, External文, Namelist文, Define File의文 Compile

(7) 入出力文處理 Processor

① 順次入出力文의 Compile
② 直接入出力文 Comqile
③ Namelist 入出力文의 Compile
④ List指示 入出力文 〃
⑤ Data文의 Compile
⑥ Encode/Decode文의 Compile
⑦ Continue文, Stop文, Pause文, Backspace文, Endfile文, Rewind文, Find文의 Compile

(8) Format處理 Processor

① Format文과 Variable Format의 Compile

6. Relocation Phase

Reloction Phase에는 다음과 같은Phase가 있다. 各 Phase別 機能을 記述하겠다.

(1) Aloc Phase

① 生成된 Symbol Table에 對해서 LDA와 LAA에 領域을 確保한다.
② Subprogram名과 Common Block名에 대해서 Dictionary를 作成한다.

(2) Rloc Phase

① 生成된 Object Code에 對해서 Symbol Table에의 Pointer를 LDA의 B/D Address로 변환한다.
② Pseudo Operation Code에 대해서 適切한 處理를 한다.

(3) Link Phase

① 外部 Symbol參照를 解決한다.
② Common領域을 確保한다.

7. Execution Phase

實行時 Phase에는 다음과 같은 Phase가 있다. 各 Phase別 機能을 記述해 나가겠다.

(1) 實行時 共通 Routine

① Interrupt處理 Routine
② 實引數, 假引數 結合 Routine
③ Error Message 編集 Routine
④ Data文의 Pre-Execution Routine
⑤ 配列의 Subscripting Routine
⑥ i/0 Phase와의 Interface調整

(2) Format 變換 Phase

① 入出力 Buffer에의 data의 轉送關係處理
② 書式付 Read/Write文의 Format變換
③ List指示入出力文의 Format變換
④ Namelist 入出力文의 Format 〃
⑤ Encode/Decode 文의 Format 〃

(3) 入力 Data Scan Phase

① Data를 Scan해서 Format變換을 위한 Constant를 Set-up한다.

② Namelist名에 屬하는 變數名의 Check

③ Data文의 16進定數 Collecting

(4) 入出力遂行 Phase

① Operating System의 Data管理 Program을 利用해서 實際의 i/0動作을 實行시킨다.

(5) Function Phase

① 基本外部關數의 計算을 遂行한다.

② 基本外部 Subroutine의 機能을 遂行한다.

8. Installation方法

Fast Compiler의 System編集은 準備·實行·確認의 세段階로 행하여진다.

(1) 提供 Tape의 內容과 形式

順 番	data set의 內容	data set의 屬性		
		Block長	Record長	Record形式
1	System編集의 Job制御文	3200	80	FB
2	Install Macro 定義體	3200	80	FB
3	Sample Program	3200	80	FB
4	Fast Compiler Load Module	Facom OS Ⅳ/F4 data set Utility Program Jsecopy의 仕樣에 의한다.		
5	Catalogue Procedure	3200	80	FB

(data set은 全部 No Label形式이다)

(2) 準 備

① 登錄用data set의 決定

a. Fast Compiler用

b. Catalogue Procedure用

c. Install Macro用

(3) 實 行

提供 Tape를 磁氣 Tape에 걸고서 Console로 부터 Start Command를 入力시키면 System編集의 Job이 實行되어서 Fast Compiler와 Catalogue가 Procedur System에 登錄된다.

(4) 確 認

Fast Compiler의 System編集이 끝나면 提供 Tape의 3번째 File에 있는 Sample Program을 Run 시켜서 System編集 結果를 確認한다.

Ⅳ. 結 論

지금까지 Fast의 開發過程 및 內部論理에 대해서 記述해 왔는데 槪略的인 內容에 그쳤다고 생각한다.

Compiler System은 원래 규모가 방대하고 기능이 복잡하고 세밀한 것이기 때문에 詳細하게 알고 싶은 분은 構成仕樣書나 機能設計書를 보도록 勸한다. 그리고 實際使用方法에 대해서는 System編集手引書나 使用手引書를 보도록 勸한다.

Compiler를 開發하는데 있어서 어떤 Operating System에서 動作할것인가도 問題지만, 昨今의 추세가 Software仕樣의 規格化가 이루어지고 있기 때문에, 어떠한 Operating System에서도 O.S와의 Interface만 調整하면 動作이 可能한 Compiler의 製作도 可能하다. 지금, 우리나라에서 Computer의 國産化를 위해서 準備를 하고 있는 줄 아는데, Compiler쪽의 開發에도關心을 가져줄 것을 要望한다.

참 고 문 헌

1. FACOM OS IV FAST 基本設計書, 富士通(株), 1977.

2. WATFOR LOGIG MANUAL, Univ. of WATERLOO, Canada, 1968.