電子計算機의 프로그래밍과 그應用 (2)

連載講座
2

宋 吉 永*
(Kil-yeong, Song)

1. Machine Language Programming

1-1 基本的인 Machine language 解說

電子計算機는 元來가 高速 精確하게 計算勞力의 代行을 하기 爲하여 發明된 것이지만 所望의 DATA 處理를 實行하자면 計算順序를 미리 一定한 規則에 따라 計算機에 가르켜 주지않으면 안된다. 이러한 目的으로 計算機 使用者는 Program 製作이라는 過程을 避할 수 없는 것이다.

Program 이라는것은 計算開始로부터 終了까지의 順序를 詳細하게 記述한 것으로 計算機는 이 Program의 內容에 따라 忠實하게 計算을 實行하는 것이다. 또 이 Program을 作成하는 것을 Programming라 하며 이것을 하는 사람을 Programmer라고 부른다.

지금 여기서 問題로 삼고저 하는 計算機는 Program 內藏方法 (Stored Program)임으로 計算開始에 앞서 Program을 記憶裝置에 집어 넣는 手續이 必要하다. 前述한바와 같이 電子計算機에서는 몇가지의 計算順序가 符號化되어 基本的인 演算을 實施하고 있는데 이것을 命令(Instruction 또는 Order)이라고 하고있다. 따라서 Program을 作成하기 爲하여서는 이들의 命令語를 適當한 順序로 配列하여야 하는것이다.

다음 命令語는 計算機의 種類에 따라 若干 틀리지만 大略 아래와 같은 몇가지 內容으로 分類할 수가 있다.

(i) 四則演算: 加減乘除 桁移動 四捨五入
(ii) 情報移動: 入力裝置로 부터의 Input
記憶裝置에의 Store
出力裝置에의 Output
(iii) 論理演算: 條件附 jump 無條件 jump
論理積 和等
(iv) 停止 : Program에 依한 停止
Over-flow 〃 〃

本節에서는 初步的인 Programming을 說明하기 爲하여 便宜上 다음과 같은 몇가지의 具體的인 命令語를 準備해 주기로 한다.

곧 計算機에 따라 命令의 種類 機能이 若干씩 틀리겠지만 現在入手可能한 中形計算機를 念頭에두고 27種의 命令語를 選出하여 우선 이것을 標準命令으로써 **表 1-1**에 실었다.

表 1-1 基 本 的 인 命 令 語

命令	Code			說明 (1)
	命令記號	Index	Adress	
加算	ADD(5)	1	n	〔n+〔I〕+UA〕→〔UA〕(2)
減算	SUB(5)	1	n	−〔n+〔I〕+UA〕→〔UA〕
積 : 加算	MAD(5)	1	n	〔n+〔I〕×〔MDR〕+〔ACC〕→〔ACC〕(3)
積 : 減算	MSB(5)	1	n	−〔n+〔I〕〕×〔MDR〕+〔ACC〕→〔ACC〕
除算	DIV	0	o	〔ACC〕÷〔MDR〕(7)→〔MQR〕(8)
四捨五入	RND	0	o	
桁右移動	SRT	1	m	m+〔I〕桁 〔ACC〕를 右便으로Shift
桁左移動	SLT	1	m	m+〔I〕桁 〔ACC〕를 左로移動
格納	STU(5)	1	n	〔UA〕→〔n+〔I〕〕
MQR 를 格納	STQ	1	n	〔MQR〕→〔n+〔I〕〕
乘算 Register 格納	LMD	1	n	〔n+〔I〕〕→〔MDR〕
無條件 Jump	JUN	1	n	無條件 n+〔I〕에 Jump
無條件 Jump 停止	HLT	1	n	無條件 n+〔I〕에 Jump해서 停止

* 韓國電力株式會社技術部 正會員

累算器內正 Jump	JAP	1	n	〔UA〕>0이면 n+〔I〕에 Jump
累算器內負 Jump	JAN	1	n	〔UA〕<0이면 〃 〃
累算器內零 〃	JAZ	1	n	〔UA〕=0이면 〃 〃
Index 內正 Jump	JIP	1	n	〔I〕>0이면 n에 Jump
〃 內負 〃	JIN	1	n	〔I〕<0이면 〃 〃
〃 內零 〃	JIZ	1	n	〔I〕=0 〃 〃
Index 를하나올려 Jump	RIJ	1	n	〔I〕+1→〔I〕한後 Jump
〃 하나나춘後 Jump	LII	1	n	〔I〕−1→〔I〕한後 Jump
Index Set	STI	1	m	m→〔Ind I〕
論理積	EXT	1	n	〔ACC〕※〔n+〔IndI〕〕→〔ACC〕(6)
Read	RED(5)	1	n	n+〔I〕桁 UA에 READ
Type out	TYP	1	n	〔ACC〕左로성터 n+〔I〕桁 印字
特殊文字 Type	TYS	1	n	n+〔I〕에서 指定된 文字 印字 (4)
効力없음	NEF	0	o	아무것도 아니함

(註) (1) m : 數値自體
n : n番地
〔n〕: n番地의 內容
I : Index Register의 番號
〔I〕: Index Register I의 內容
(2) UA : Upper AccumulatoIr의 略記號
(3) ACC : Accumulator의 略號
Upper Accumulator와 Lower Accumulator가 連結되어 하나의 Accumulator를 構成함
(4) 特殊文字Type는 미리數字로 文字를 指定해둠
(5) 命令 머리에 C (Clear)를 追加하면 그命令의 實行에 앞서 〔ACC〕→0으로함
(6) ※—論理積의 記號
(7) MDR : 乘算 Register 被乘數 또는 除數가 格納됨
(8) MQR : 除算 Register 乘數 또는 商이 格納됨

加 算(ADD)

「ADD 0 1234」라는 命令을 實行하면 記憶裝置의 1234番地의 內容과 Upper Accumulator의 內容을 보탠 和가 새로히 Upper Accumulatore에 格納된다. 곧 記號로 表現하면

〔1234〕 + 〔UA〕 ⟶ 〔UA〕

1234番地의 內容 / 먼저UA에 들어있던 內容 / UA의 새로운內容

다음 〔ADD 1 1234」라는 命令을 생각하여보자.

이번에는 ADD의 다음이 零이 아니고 "1"이되어 있으므로 Index Register No. 1이 命令의 實行에 關係하게 된다. 萬一 Ind. 1의 內容이 0085라고 하면 이때에는 1234番地의 內容이 아니고 1234+005=1319番地의 內容이 〔UA〕에 加算되게된다.

곧 〔1234+85〕+〔UA〕→〔UA〕

減 算(SUB)

減算은 加算과 마찬가지이다.

但여기서 注意해야 할것은 負數의 表現法이다. 主로 이것은 補數를 使用하고 있는데 지금 한例를 들어보겠다.

74−56의 計算은 다음에 보인바와 같이 100−56=44 (이것을 100에 對한 補數라고함)를 加算하여 74+44=118로 計算하고 마지막에 100位의 1을 버려 18로 하면 되는것이다.

$$74-56=74-\underbrace{56}_{44}+100-100=74+44-100$$

$$=118-100=18$$

實際는 減算을 하지않고 첫머리 1만 빼버린다.

一般으로 取扱하는 數의 桁數가 미리定해져 있을때에는 그 桁數보다 1桁은 10, 100, 과같이 나중에 첫머리에 벗어난 數字를 버리기에 便利한 數에 對한 補數를 使用할 境遇가 많다.

2進法計算機에서는 普通 2의補數를 使用하고 있다.

乘 算(MAD)

「MAD 1 1234」라고하는 命令은 番地部의 數値 1234에 Index×1의 內容 0085를 보탠 數値 곧 1319番地의 內容과 乘算 Register (MDR)의 內容의 積에 累算器 (ACC)의 內容을 加하여 그 結果를 새로히 累算器에 貯藏한다.

〔1234+0085〕×〔 MDR〕+〔ACC〕→〔ACC〕

그림 1-1 과 **그림 1-2**에이 乘算內容을 說明한다.
前者는 正數의 積이며 後者는 負數의 乘算例이다.

⊕. 2358314 (1319番地의 內容)——乘數
×) ⊕. 6241073 (MDR)——被乘數
⊕. 1471840 9830922 積
Upper ACC Lower ACC

그림 1-1 乘算의 說明

(註) (i) 概念圖이기 때문에 10進法으로 表現하였다.

(ii) ⊕符號直後의 은 計算機 回路에서 본 小數點의 位置이다. 이것은 一般으로 Programmer가 生覺하는 計算上의 小數點 位置와 一致하지 않는다.

(iii) 이 그림 例에서는 乘算開始前의 〔ACC〕=0이라 假定하고있다. 萬一이때 〔ACC〕=1234567 6543210 이라면 上記의 結果는

1471840 9830922
+1234567 6543210
2706408 6374132 로 될것이다.

⊕. 2358314 (1319番地의 內容)——乘數
×) ⊖. 6241073 (MDR)——被乘數
⊖. 1471840 9830922 符號+絕對値
⊖. 8528159 0169077 符號+9의 補數
⊖. 8528159 0169078 符號+10의 補數

그림 1-2, 負數의 乘算例

除 算(DIV)

「DIV 0 0000」의 命令을 實行하면 累算器의 內容÷乘算 Register(MDR)의 內容이 商 Register(MQR)에 格納된다.

이때 剩餘는 Lower ACC에 간다.

除算命令의 Index 및 番地部分은 除數 被除數의 番地를 指定하지 않는다.

그림 1-3에 除算의 1例를 보인다.

⊕. 2743528 371429 (ACC) 被除數
÷) ⊖. 3000000 (MDR) 除數
⊖. 9145094 (MQR) 商
⊖. 0000000 1714259 (ACC) 剩餘
(Upper ACC) (Lower ACC)

그림 1-3, 除算의 說明

格 納(Store)

「STU 1 1234」를 實行하면 〔UA〕 內容을 1319番地(1234+0085)에 格納한다. 이때 〔UA〕는 不變이다.

「STQ 1 1234」에 依하여서는 MQR의 內容이 1319番地에 格納된다.

桁移動(Shift Right or Left)

「SRT 2 0002」(Index 2의 內容을 3이라 假定함)을 實行하면 〔ACC〕의 內容이 오른쪽으로 5桁 移動한다. 이때 〔LA〕의 右쪽 5桁의 數字는 없어진다.

實 行 前 (SRT) 實 行 後
1234567 7654321 —5桁 Shift→ 0000012 3456776

「SLT 2 0002」를 實行하면 (ACC)의 內容이 왼쪽으로 5桁 Shift하고 이때 (UA)의 왼쪽 5桁는 없어진다.

實 行 前 (SLT) 實 行 後
1234567 7654321 ←5桁 Shift— 6776543 2100000

無條件 Jump (Jump Unconditionally, Halt)

「JUN 2 1234」의 命令으로 다음命令은 1234+3=1237番地 (Ind 2의 內容=3)에 뛰어 거기서부터 시작한다.

「HLT 2 1234」의 命令이면 1237番地에 Jump한後 演算을 停止한다. 計算開始 Switch를 누르면 1237番地부터 命令을 實行하기 시작한다.

條件付 Jump(JAP. JAN. JAZ. JIN. JIZ 等)

「JAP 1 1234」(Index 1의 內容=0085)를 實行하면 (UA)의 內容이 零 또는 負이면 그냥 그대로 다음 番地의 命令을 實行하고, (UA)가 正 (Positive)이면 1234+0085=1319番地에 Jump하여 演算을 여기서 부터 곧 1319番地부터 始作한다.

「JIZ 2 1234」 (Index 2의 內容=3≠0)을 實行하면 Index 2의 內容이 零이 아님으로 그대로 通過하여 다음 番地의 命令을 實行한다. 萬一 이때 「JIP 2 1234」를 實行 한다면 〔Ind 2〕=3>0으로 條件에 合致됨으로 다음 命令은 1234番地부터 實行한다.

Index 增減 Jump(RIJ. LIJ)

「LIJ: 1234」를 實行하면 (Index 1)의 內容을 0085−0001=00084로 하나 줄이고 1234番地에 Jump한다.

Set Index (STI)

「STI 1 1234」의 命令으로 Index 1의 內容이 1234로 된다. 따라서 命令實行以前의 Index 1의 內容(이제까지 0085로 假定했음)은 Clear 되고 새로운 1234로 바뀐다.

Read (RED)

「RED 2 0002」(但 〔Ind 0. 2〕=3)의 命令으로 (UA)의 內容이 5桁(=0002+0003) Shift하고 Tape Input裝置로 부터 새로운 數字가 5개 아래 桁으로 부터 Inptu된다.

이때 (UA)의 위 5桁은 사라진다.

印 字(Type, Type Special)

「Type 2 0002」에 따라 (UA)의 內容을 위로부터 順次 5桁 Type 한다.

2進法의 計算機에서는 印字하기 前에 2進→10進의 變換을 하고 印字는 普通 Sub-Routine에 의거하고 있다. 「TYS 2 0022」(〔Ind 2〕=3)의 命令으로 3+22=25에 相當하는 文字 例를들면 "+"라는 字를 Type한다.

無効力(No effect)

「NEF 0 0000」의 命令으로 아무것도 實行하지않고 다음番地의 命令을 實行한다.

이것은 部分的으로 나누어진 Program 사이를 메꾸거나 뒤에 Program을 追加 시킬때 使用한다.

Clear에 關係되는 命令(ADD…RED)

表1-1 右肩에 (5)라고 註를 부친 命令앞에 "C" (=Clear)를 加한것은 그 命令의 實行에 앞서 〔ACC〕=〔UA+LA〕를 0으로 Clear한다. 지금 「CSTU 0 1234」를 實行하면 1234番地의 內容을 0으로 하게된다.

다음에 몇가지 알아두어야 할 事項을 적어보겠다.

앞서 『命令의 機能』에서 說明 한바와 같이 Program 內藏方式의 計算機는 命令語와 數値語를 同一한 記憶裝置內에 같은 形式으로 格納하는 것이다.

곧 表에 쓴 命令은 加算이면 "ADD 1 1234"와 같이 表現하하지만 이것이 記憶裝置에 格納될때에는 實은 "⊕20 1 1234"라하는 數値의 形式을 取하게 되는 것이다.

ADD라고하는 表記法은 어디까지나 우리늘 Programmer가 記憶하기 쉽게끔 만들어진 記號(Mnemonic code)인 것이며 格納되기 前에 入力 Program이라고 불려지는 特殊 Program 또는 回路의 힘을 빌려 ADD→20이라는 變換을 거치고 있는 것이다.

따라서 計算機自身은 例를들면 1234番地에 들어있는 情報가 命令인지 數値인지는 全然 모른다. 그러나 數値라고 생각하면 그것으로서의 意味를 가지며, 命令語라면 命令語대로의 意味를 가지는 것이다. 다만 이 區別을 明確히 하는것은 Program 自身인 것이다. 따라서 Program에 Error가 있어서 올바르게 Jump하여야 할데를 못찾아가고 元來 數値로 取扱 하여야 할 番地에 Jump 한다면 이것을 命令語로 解釋하여 Programmer가 豫想치도 않은 그릇된 計算을 하거나 停止하게 되는 것이다.

1-2 數値計算法解說

다음計算機의 Programming의 槪念을 明確하게 하기 위하여 具體的인 數値計算法 例를 하나 說明하여 보겠다.

지금 代數方程式 $f(x)=0$의 實根을 求할려면 어떻게 하여야 하는가? 라고하는 問題를 例題로 삼아 筆算(手計算)과 自動計算의 相違點. Flow Chart 및 Program 作成을 說明하겠다.

$f(x)=0$ 이 가지는 實根中 特定한 한根에 着目하여 그 部分의 横軸를 擴大한 것이 **그림 1-4**의 (a) (b)이다.

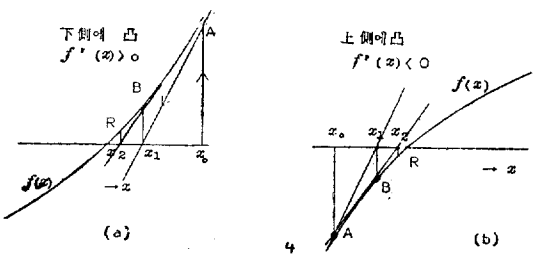


그림 1-4

同그림 (a)와 같이 曲線 $f(x)$가 下側에凸 $f''(x)>0$의 遇遇에는 實根R의 右側에 推定値 x_0를 選擇하여 x_0에서의 垂直線과 $f(x)$와의 交點A에서의 切線 Ax_1을 그어 第1近似値 x_1을 求하면 x_1은 x_0와 比較해서 훨씬 實根R에 가까워지고 있다. 다시 x_1에서 세운 垂線과 $f(x)$와의 交點B에서의 切線 Bx_2를 그어 第2近似値 x_2를 求하면 한거름 더 實根R에 接近하게 된다. 이와같은 手續을 되푸리 해나가면 第n近似値 x_n와 第$(n+1)$ 近似値 x_{n+1}과의 差가 漸漸 적어져 豫定된 許容 誤差의 範圍內에서 實根R가 求해질 것이다. 同그림 (b)와 같이 曲線 $f(x)$가 上側으로 凸 $(f''(x)<0)$인 경우 에는 實根R의 左側에 推定値 x_0를 選擇하여 上述한 手續을 되푸리 하면 될것이다.

위에서 說明한 方法은 Newton의 逐次近似法이라고 불려지는 것으로 代數式의 實根을 求하는데 極히 有効한 方法이다. 그림 1-5에 의거하여 이 Newton의 逐次近似法이 公式을 誘導하여 보겠다.

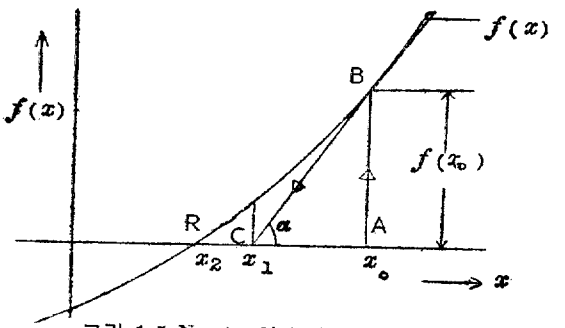


그림 1-5 Newton의 逐次近似法 說明圖

推定値 x_0에 세운 垂線 AB의 길이는 $f(x_0)$이다.

垂線 AB와 $f(x)$의 交點B에 있어서의 切線 BC는 横軸과 C點 $(x=x_1)$에서 마주치게 된다. 따라서 三角形 ABC에서 (1)式이 成立한다.

$$AB=AC\times\tan\alpha \cdots\cdots\cdots\cdots\cdots\cdots(1)$$

(1) 式부터

$$f(x_0)=(x_0-x_1)f'(x_0) \cdots\cdots\cdots(2)$$

$$x_1=x_0-\frac{f(x_0)}{f'(x_0)} \cdots\cdots\cdots\cdots\cdots(3)$$

곧 (3)式의 右邊은 x_0가 指定되면 計算되고 이로부터 左邊 x_1이 求해진다.

여기서 다시 새로운 x_1을 x_0이라 생각하여 (3)式의 右邊에 代入하면 第2近似値 x_2가 求해질 것이다.

一般으로 第n近似値 x_n부터 第$(n+1)$近似値를 求할려면는 아래의 (4)式을 利用하면 될것이다.

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)} \cdots\cdots\cdots\cdots\cdots(4)$$

이 (4)式이 Newton의 逐次近似法의 基本式이된다.

여기서 $f(x)=x^2-a$ $(a>0)$이라하여 平方根을 求하

여 보자 $f(x)=x^2-a=0$을 풀면 $x=\pm\sqrt{a}$가 됨으로 그림 2-6 (a)에서 原點와 交點의 距離 OR가 $+\sqrt{a}$로 된다는 것을 알 수 있다. 따라서 $\sqrt{a}$의 計算은 二次曲線 $f(x)=x^2-a$의 實根, 곧 $f(x)$와 x軸과의 交點의 座標를 求한다는 問題에 歸着될 것이다.

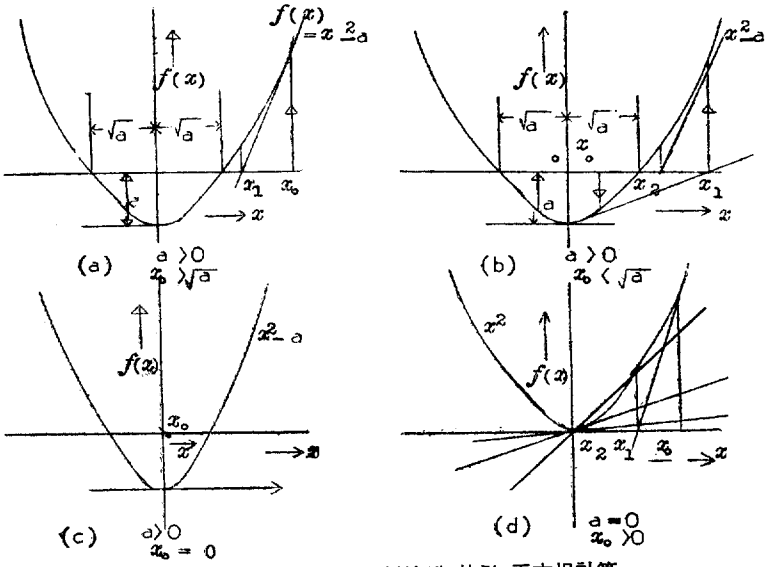


그림 1-6 Newton의 遂次近似法에 依한 平方根計算

$$f(x)=x^2-a=0$$
$$f'(x)=2x \cdots\cdots\cdots\cdots (5)$$
$$f''(x)=2>0$$

(5)式에서 보는 바와같이 二次曲線 x^2-a는 x의 全域에 걸쳐 下側에 凸 ($f''(x)=2\times0$)임으로 **그림** (a)의 說明에 따라 點R의 右側에 第一推定値x_0를 選擇하는 것이 좋을 것이다. 그러나 만일 **그림** (b)에서 보는바와 같이 이때 가령 x_0를 R의 左側에 推定하였다 하더라도 第1近似値 x_1부터는 R의 右側에 오므로 推定値 x_0는 R의 左右 어느쪽에 골라도 괜찮을 것이다)

但 여기서 注意하여야 할것은 同 **그림** (c)에서 알 수 있는바와 같이 推定値 x_0를 零으로 해서는 안된다는 것이다. 그 理由는 切線이 x軸과 平行이 되어 第1近似値 x_1이 無限遠點으로 되어 計算機가 取扱할 수 있는 數値의 範圍를 벗어나기 때문이다.

한편 **그림** (d)에서 보는바와같이 $a=0$일때는 推定値 x_0가 零이 아니더라도 計算이 進行됨에 따라 切線이 x軸과 平行되기 때문에 零의 平方根이 나오지 않게 될때도 있을것이다. 따라서 이러한 狀態를 避하기 爲하여서는 平方根의 計算에 앞서 a가 零인가 아닌가를 조사하면 될것이다. 곧 a가 零이면 計算을 빼고 答을 零으로 하고, $a>0$이면 Newton의 方法으로 $\sqrt{a}$를 計算하고, $a<0$이면 $\sqrt{|a|}$를 計算하여 $j\sqrt{|a|}$라고 印刷 하도록 하면 될것이다.

(4) 式에 $f(x)=x^2-a$ 및 $f'(x)=2x$를 代入하면

$$x_{n+1}=x_n-\frac{x_n^2-a}{2x_n}=\frac{1}{2}\left(x_n+\frac{y}{a_n}\right) \cdots\cdots (6)$$

(6) 式이 Newton의 遂次近似法에 依한 平方根의 計算式이다.

다음에 $\sqrt{3}$을 具體的으로 計算하여보자.

表 1-2 는 우리들이 日常흔히 計算하는 方法인데 上述한 Newton의 方法과 比較하기 爲하여 실어본것이다.

1	1.7320508 $\sqrt{3}$
1	1
27	200
7	189
343	1100
3	1029
3462	7100
2	6924
34640	17600
0	0
346405	1760000
5	1732025
3464100	2797500
0	0
34641008	279750000
8	277128064
34641016	2621936

表 1-2 $\sqrt{3}$ 의 計算

表 1-3 (a) (b) 및 (c)는 推定値 x_0를 여러가지로 變化시켜 Newton의 方法으로 $\sqrt{3}$에 收束해가는 모양을 본것

이다.

여기서 볼 수 있는 바와같이 Newton의 方法뿐만 아니라 一般으로 逐次近似法은 安定한 性格을 가지는 것이며 使用條件만을 알아두면 極히 有用하고 便利한 方法이라는 것은 알수 있을 것이다.

表 1-3 Newton 의 根似法에 依한 $\sqrt{3}$ 計算結果

(a) $x_0 > \sqrt{3}$			(b) $x_0 < \sqrt{3}$			(c) $x^0 \gg \sqrt{3}$		
n	x_n	$x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right)$	n	x_n	$x_{n+1}=\left(x_n+\frac{a}{x_n}\right)\times\frac{1}{2}$	n	x_n	$x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right)$
0	3.0(推定定)→	2.0	0	1.0(値推定)→	2.0	0	10(値推定)←	→5.15
1	2.0 →	1.75	1	2.0 →	1.75	1	5.15 →	2.86626
2	1.75	1.73214	2	1.75	1.73214	2	2.86626	1.95646
3	1.73214	1.7320508	3	1.73214	1.7320508	3	1.95646	1.74492
4	1.7320508	1.732050805	4		1.732050805	4	1.74492	1.732098
5	1.732050805	1.7320508075	5	1.732050805	1.732050805	5	1.732098	1.732050805
6	1.7320508075		6	1.7320508075		6	1.7320508085	1.7320508075
						7	1.7320508079	
	$\sqrt{3}=1.7320508075$ (6)			$\sqrt{3}=1.7320508075$ (6)			$\sqrt{3}=1.7320508075$ (7)	

다음 이것을 實際로 計算機에 Programming 시키자면 먼저 그 準備로서 計算內容과 順序를 明確하게 보이는 Flow chart를 만들어야 할것이다.

그림 1-7과 **그림 1-8**은 (6)式 $x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right)$에 依한 平方根計算用의 Flow chart이다.

이와같은 Flow chart는 반드시 特定한 計算機를 念頭에 두고 쓸 必要는 없다. 事實 여기에 쓴 Flow chart는 어느 計算機에 對하여서도 使用할 수 있는 것이다.

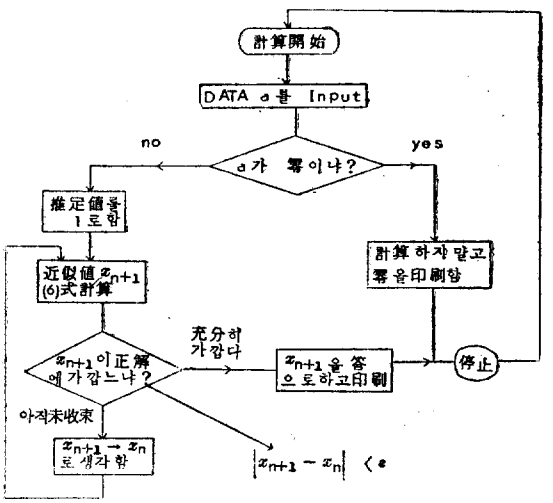


그림 1-7 平方根 Program의 Flow chart(A1)

*(註) Sub-Routine

Sub-Routine은 文字그대로 主Program의 副次的인 役割을 하는것으로 平方根 Sine-Cosine 函數·指數·對數 Simpson의 積分公式, Runge-Kutta의 數値積分法等 使用頻度가 높은 計算內容을 擔當하여 必要에 따라 主Program에 連結시켜 使用하는 것이다.

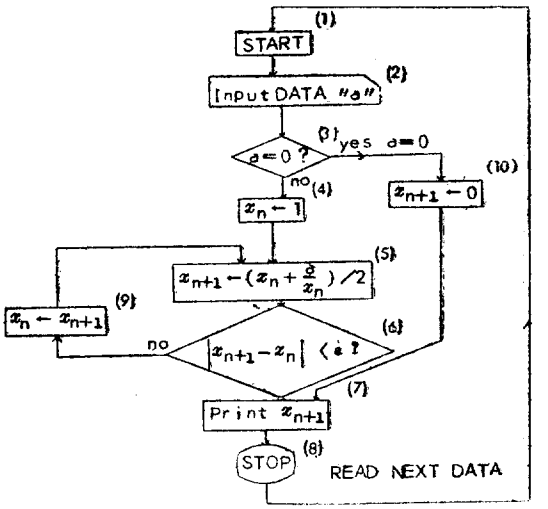


그림 1-8 平方根 Program의 Flow chart(其 2)

以上의 Flow chart에 이어 平方根의 計算을 範例로 하여 具體的으로 Programming을 說明하겠다.

表 1-4 는 **그림 1-8**의 Flow chart에 依據하여 **表1-1**의 標準命令을 參照하여 만들어진 平方根 計算 Program이다. 但 여기서는 Sub-Routine (註) 形式을 採用하고 있다.

即 主 Program으로부터 300番地에 Jump해와서 平方根을 計算한 後 321番地로부터 主 Program에로 되돌아가게 하고 있다.

表 1-4 中 左側欄은 命令의 格納番地, 가운데欄은 命令語 또는 數値語, 右側欄에 各命令의 說明과 實際計算內容을 說明하고 있다.

또 ③ ④……⑩은 **그림 1-8**의 Flow chart에 記載한 block 番號와 對應하는 것으로서 上記그림과 本表와를 比較하면 Program의 構成이 容易하게 把握될 것이다.

表 1-4 平方根 Sub-Routine의 Program

番 地	命令또는數値	說 明	
主Program-300 ram부터	STU0 0330	〔UA〕→a	③
301	JAZ0 0325	a≠0?	
302	CADD0 0327	初 値(=1.0) 設定	④
303	STU0 0331	x_n←0.999……99	
→ 304	CADD0 0330	a→〔UA〕	
305	LMD0 0331	x_n→〔MDR〕	
306	DIV0 0000	a/x_n→〔MQR〕	⑤
307	STQ0. 0332	〔MQR〕=a/x_n→x_n+1	
308	LMD0 0328	0.5→〔MDR〕	
309	CMAD0 0331	0.5x_n→〔ACC〕	
310	MAD0 0332	0.5x_n+0.5×$\frac{a}{x_n}$=(x_n+a/x_n)/2→〔ACC〕	
311	STU0 0332	($x_n+\frac{a}{x_n}$)/2→x_{n+1}	
312	CADD0 0331	x_n→〔UA〕	
313	SUB0 0332	x_n-x_{n+1}→〔UA〕	⑥
314	JAZ0 0320	x_n-x_{n+1}=0? 0이면 320番地에	
315	JAP0 0318	x_nx_{n+1}>0? Positive면 318番地에	
316	CADD0 0332	x_{n+1}→〔UA〕	
317	SUB0 0331	$x_{n+1}-x_n$→〔UA〕	
318	SUB0 0329	$x_{n+1}-x_n-\varepsilon$	
319	JAP0 0322	未完了	
320	CADD0 03E2	x_{n+1}→〔UA〕	
321	JUN2 0000←		
→ 322	CADD0 0332		
323	STU0 0331	x_{n+1}→x_n	⑨
324	JUN0 0304		
325	CSTU0 0332	0→x_{n+1}	⑩
326	JUN0 0321—		
327	9999999999	初期値≒1.0	
328	5000000000	(0.5=$\frac{1}{2}$)	
329	0000000001	ε(收束範圍)	
330	a		
331	x_n	一時格納場所	
332	x_{n+1}		

(註) ③ ④…前出 **그림 1-8** Flow chart Box의 番號와 對照

表 1-5 는 以上의 內容을 土台로 하여 『正數를 읽고 (Read in) 平方根을 計算하여 이것을 印字하고 停止한다』는 Program 全體를 보인것이다.

番 地	命 令 語	
200	CRED0 0010	往 平方根Sub-Routine
201	STI2 0203	→ 300番地
202	JUN0 0300	
203	TYS(CR)	表
204	TYS (+)	← 復 2—5
205	TYS (.)	
206	TYP0 0010	321番地로부터
207	HLT0 0200	

表 2-6 平方根의 計算

이 Program에서는 主Program과 Sub-Routine의 두部分으로 構成된다.

곧 8個의 命令語로부터 이루어지는 主Program은 DATA(a)의 Read, Sub-Routine에의 往復 및 答의 印刷, 停止를 받고 33個의 命令語와 數値語로 이루어지는 平方根 Sub-Routine은 上記 主Program와 結付되어 平方根 計算을 分擔하고 있다.

다음에 簡單한 Program 例題를 하나 들어보겠다.

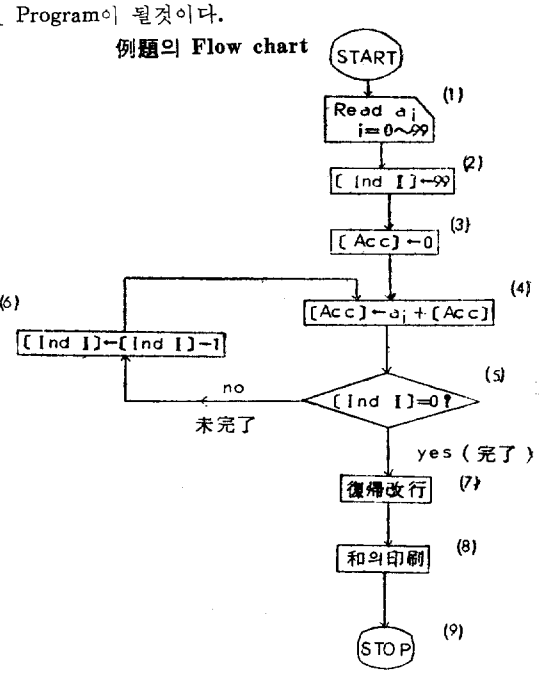
例題

『任意의 數値가 100個 있다. 이것의 總和$S=\sum_{i=0}^{99}a_i$를 求하라』

多數의 數値의 和를 求하는데 ADD, ADD…의 連續으로 Program을 쓴다면 記憶裝置가 얼마있어도 不足할 것이며 지루할것이다.

이러한 境遇에는 計算을 Loop 化하여 數値를 適切한 適切한 番地로부터 끄집어 내도록 한다면(番地變更)經濟的인 Program이 될것이다.

例題의 Flow chart



例題의 Program

番 地	命 令 語	說 明
0	S T I1 0990	① a_i(10桁) Read
1	C R E D0 0010	i=0～99
2	S T U1 0300※	※ DATA는 300～399 番地에 Store한다고 함
3	J I Z1 0005	
4	L I Z1 0001	
5	S T I1 0099	② 〔Ind 1〕←99
6	C S T U0 0013	③ 〔ACC〕←0
7	A D D1 0300	④ 〔ACC〔←a_i+〔ACC〕
8	J I Z1 0010	⑤ 〔Ind 1〕=0？
9	L I Z1 0007	⑥ 〔Ind 1〕 하나씩 減함
10	T Y S0 (C.R)	⑦ Carrige Return
11	T Y D0 0010	⑧ $S=\Sigma a_i$ Type (10桁)
12	H L T0 0000	⑨ 停止

→前出 Flow chart block 番號에 對應

<次 號 繼 續>

支 部 消 息

(釜山支部 편)

지난 11月 當學會 釜山支部에서 傳해온 消息에 依하면 30餘名의 會員이 大擧 入會하였다 한다. 會正員이 23名, 准會員 3名과 함께 埠頭發電所에서도 4名이 入會하고 있다.

또한 67年 9月24日 同支部에서는 馬山地區에 있는 韓一合纖, 中央紡織, 韓國鐵鑛, 鎭海化學, 馬山發電所를 見學하여 豊盛한 成果를 올렸다고 한다.