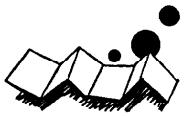
解　説



アセンブラ†

一　松　　信††

はじめに

日本の創草期のソフトウェアとして，アセンブラについては，和田英一による PC-1 用の R 0 (1957年頃) や[1]，島内剛一による HIPAC-101 用の S コード[2] (1959年頃) などが，EDSAC の入力プログラム[3] (1948年) に並ぶ芸術品として高い価値をもっている．しかしその両者とも執筆していただき難い事情がある．筆者はその二番煎じで1965年頃 TOSBAC-3300 用のアセンブラを作ったことがあるだけで，この主題には不適任である上に，個人的な事情だが外国出張の日時が切迫したため，はなはだ不完全な記述にならざるをえないことをお詫びする．

1. 1950年代のソフトウェアの特長

他の稿にも論ぜられると思うが，当時の計算機を今と比較すると隔世の感がある．例えば：

1°　素子の信頼性が低く，長時間にわたる計算を確実に実行すること自体が困難であった．

2°　記憶が極度に小さく (M-1 で 256 語，PC-1 で 512 語，TAC で 1024 語)，プログラムを主記憶に収めるのが容易でなかった．

3°　ハードウェアが直接使用者に接しており，記憶内容自体を，命令，データの区別なくビットパターンとして操作できたし，ランプや音で結果がわかることもあった．

4°　ソフトウェアは大体において自給自足であり，互換性は初めから問題にならなかった．

もっとも1950年代の後半には，各自が自家用の入力プログラムを用意する時代は終り，優れたアセンブラが共有財産ないしは機械の一部として広く使われるようになった．

当時のプログラムの至上要請は短く，実行が高能率になるように作ることだった．一命令でも短くしなければ，計算機におさまり難かった．EDSAC の入力プログラムが40語，R 0 が68語 (小数入力を加えると116語) というのは，いまでは信じ難い値である．必然的にアセンブラの類は“芸術品”であり，内容は高度の“暗号文”であって，それを解読するには，新たに作る以上の努力が必要だった．

この“古き悪しき時代”の体験談は，現在のプログラマには“有害”な手法も多いようだが，記録の意味で以下にいくつかを述べる．

当時のソフトウェアの最大の欠陥は，仕様記述が明確でないことだったといってよかろう．作っているうちに，仕様も次第に固まってくる場合が普通だった．アセンブラについていえば，“入力エラー”がほとんどなく，どんな記号列でも受けつけ，適当に解釈して，何らかのコードに変換していた．じっさいその種の“正規でない”用法を発見して，利用した経験も少なくない．もちろん仕様・設計を正しく定め，仕様以外の入力はエラーとすべきだと指導した先駆者もおられたが，そういうことに余分の手間をかけるのは意味が薄いという風潮が強かったように思う．

そのような時代は過去のものになったと思っていたが，1970年代初期に，プログラマブルなポケット電卓が現れたとき，ふたたび似た傾向が現れた．もっともこれも，記憶装置が急激に安価になるにつれて，いまでは昔の名残りになりつつある．

2. 当時のプログラミング法

ソフトウェアの複雑さを，それを構成する命令数で測るのには問題があるが，もし命令を部品と数えれば，現在の金融システムのような巨大ソフトウェアは，人類がこれまでに作った“最も複雑な製品”といわれている[4]．その測度によれば，EDSAC の入力プログラムや R 0 などは，自転車を部品から自作したのに匹敵する作業といえる．R 0 では後述のような“隠字詩”の部分があるので，これは既製の別の装置

† Assembler by Sin HITOTUMATU (Research Institute for Mathematical Sciences, Kyoto University).

†† 京都大学数理解析研究所

を改造して転用したのに相当するかもしれない.

この測度でいえば,コンパイラや小さいOSは,自動車や軽飛行機に相当する.もはや名人が自宅で作るものではなく,設備のととのった工場で,専門の工員が作るべきものである.

こういうことが明確になってきたのは,ソフトウェア工学が形をなしてきた1970年代後半以降である.1950年代にはそういった意識さえなかった.プログラムは名人が机の上で書き,必要なら計算機にかけて手直しするものだった.数十ステップの小プログラムは,全体像を頭の中に入れて,一気に書き下すほうが,たいていよいものができた.最も注意すべき点はコードの記憶誤り,記憶内容のわりつけの誤り,飛躍先の番地の誤りなどだった.流れ図はプログラムができてから他人に説明する文書を作るときに初めて書いたことが多い.私自身の経験を回想すると,まず一種の擬似言語で算法を表現し,それを人間コンパイラが訳すか,あるいは型にはまったプログラムの定跡集を頭に入れて,それをひきだしながらプログラムをまとめた感がある.数時間の列車の内が,最も妨害が少なく,落ちついてまとまった仕事のできる環境だった.ただそこでは大きな図を拡げるスペースがなく,部分的なブロックしか作れなかったのも事実である.

こういう"古き悪しき時代"の作業を真似すべきではないが,プログラムの定跡データベースなどは,いまでもやってみる価値があるように思う.

3. 当時のアセンブラでの技法例

初期の計算機には,インデックス・レジスタがなく,ループでは命令自体に演算を施して,変更しつつ使うのが,プログラム内蔵方式の利点として常識だった.この変更は,たいてい命令の数値部(番地指示部)に定数を加減するだけであったから,そのための専用のB-レジスタができ,それがインデックス・レジスタに進化したのは,周知のとおりである.しかし形の上では命令も数値データも区別がなかったから,命令の機能部をも変更して,別の命令にすりかえる場合もあった.最も極端な場合には,命令を表わす数値に乗算を施して,別の命令を作りだした例もあった.このようなプログラムは,計算機の内部コードに熟達していなければ,"読む"ことさえできなかった.

PC-1のR0では,テレタイプのコードをそのまま使用したため,数字コードの変換が一苦労だった.いまならば変換表をおくわけだが,そのためのスペースさえも切りつめる必要があった.R0では,数字の外部コードに対応する番地に,その数字の値を数値部にもつ命令をおき,プログラムと変換表とを兼用するという"曲芸"をやっていた.こういうプログラムは,いわば指定された位置に指定された文字をあらかじめ設定した"隠字詩"である.それを作るには,最高度の知的遊戯ともいうべき"言葉遊び"の精神がいる.R0のプログラムがやたらにあちこち飛び廻ったり,また十進二進変換中,2桁左へシフトするのにわざわざ一度右へ7桁シフトしてから左へ9桁シフトするようにしたりしたのも,すべてこの"隠字詩"実現のための苦心であった.

Sコードでは外部コードが整然としていたから,その種の苦心は不要だったが,命令の機能部を示す英字を区切り符号と兼用し,命令をテープ上に空白なしにベタ書きしてもよいようになっていた.

概してこの時代のアセンブラの文法では,各文字や記号に特有の意味をもたせ,あるいは位置によって意味を変えるという方式であった.全般的になるべく例外を作らないのが,簡潔なアセンブラを作るためにもよい方策でもあった.

4. 最適配置プログラム

筆者はTOSBAC-3300の設計の前に,一時期TOSBAC-3100を手掛けたことがあった.これは磁気ドラムを主記憶とし,各命令に次の実行命令番地を指定する二番地方式の機械であった.

原理的には当時の広告文にあったとおり,"能率のよいプログラム"が作れるはずだった.しかし実際にやってみて,"原理的に可能"と"実際に製作可能"とはまったく別の話であるという工学の基礎的常識をいやというほど体験させられた.それでも機械的に次の命令を次の番地に並べたプログラムと,次々の命令の配置を考慮したプログラムとでは,実行速度が数倍違ったから,配置を工夫する必要があった.

そのための"最適配置プログラム"も考えたが,これは結局実現しなかった.その原因は,そのための算法が不明であったこと,そのプログラム自身をごく短かく作らなければならなかったこと,さらに乗算除算以外は,大体一命令の所要時間が同一なので,乗除算の少ない計算プログラムは,命令を次々に等間隔に並べれば大体まにあったことなどである.磁気ドラムが主記憶という計算機それ自体が,いまではもはや昔話になったが,二番地方式でなく,論理的な番地を物理

的に一定の間隔につけて，一番地方式にした計算機（筆者の使ったものでは HIPAC-101）が実用的であったように思う．

配置の最適化も，プログラムのしめる容量が全記憶の 1/4 以下なら，比較的容易に，ほぼ理論上の最適化ができたが，それが全記憶の半分以上を占めるようになると，急激に困難になった．計算機の番地は，カードと違って1枚挿入して一つずつずらすことが直接には困難なのが，致命的だった．またプログラムの最初から機械的に並べるよりも，最も多く廻るループをまず最適の位置におき，あとでその前後を埋めこむとよいことが，次第にわかってきた．しかしこの操作は，完成したプログラムを人間が見渡して初めて可能であり，計算機に自動的にやらせるのは困難だったし，何よりも絶えず手直しして修正してゆくプログラムには不可能だった．

ただこの作業中に米田信夫の発見した次の注意は重要と思う．これは十進法の計算機であり，乗算は乗数によって所要時間が大きく変動した．当然次命令をどこに置けば，平均待ち時間が最短かという問題が生ずる．もし乗数の各桁の数字がそれぞれ一様分布をしていると仮定すれば，その演算時間の分布は，ほぼ正規分布になる．このとき，間隔をその平均所要時間（最大と最小の相加平均）にとると，平均待ち時間が最大（!）に近くなる．これを最小にする点は，ほぼ最大と最小を1：2に内分した点（2/3 の地点）であった．そうすると大部分の場合はそれまでに完了し，1回転待つべき場合は比較的稀になるからである．

除算の場合も，演算速度の変動が大きいが，このときは商の数値によって定まるので，事前に判定し難い．しかしやはり平均所要時間をとるのは損で，2/3～3/4 の地点にとるのが最適であった．

5. 初期の珍談

アセンブラからは逸脱して，数値計算の話になるが，筆者にとって忘れられない例を記しておく．

（i） $e^{-8}<0$?[5] e^x の Taylor 展開に，機械的に $x=-8$ を代入したら，負の答がでたという話である．途中に絶対値の大きな項があって，正負加えて桁落ちを生じ，誤差ばかりが残ったためである．現在では不適切な算法の典型例だが，当時は計算機の金物，プログラムその他の誤りか，と大騒ぎした一例である．

（ii） $\sqrt{0}\neq 0$?[6] これも何度もくりかえし報告されている．$\sqrt{a}$ を Newton 法で $f(x)=x^2-a=0$ の解として求めるとき，$a=0$ のときは，$f'(0)=0$ であるのに，機械的に適用すると，0にならない値で止ることがある．それは $\alpha\neq 0$ でも，$\alpha^2=0$（下へのあふれで）と判定される数 α があるから，といってもよい．いまでは0を判定して，$\sqrt{0}=0$ として返すのが常識だが，$\sqrt{0}\neq 0$ をめぐって1時間にわたる大議論があった[6]．

（iii） $J_3(0)=1$？ これは Bessel 関数 $J_n(x)$ を積分

$$J_n(x)=\frac{1}{\pi}\int_0^{\pi}\cos(n\theta-x\sin\theta)d\theta$$

で求めようとしたときに生じた．原因は Romberg 式の算法をとるので，$x=0, n=8$ では最初3回続けて1になるために，収束したと誤って判定したせいである．最小反復回数の指定が不可欠な例である．

以上はいずれも現在では笑い話に近いし，その理由をきけば当然でさえあるが，当時のわれわれは，こうした思いもかけない現象に首をひねったものである．

個人的な思い出が主になり，本特集にそぐわないものになったかもしれないが，当時の雰囲気の若干を伝えて，責をふさいだ次第である．

参 考 文 献

1) PC-1 マニュアル，東大物理教室（1958）．
2) S-コードマニュアル（1962）．
3) Wilkes, M. V. et al.: The Preparation of Programs for An Electronic Digital Computer, Addison-Wesley-Press, Reading, MASS (1951).
4) 赤木昭夫：ソフトウェアのイドラ，自然，1982年10月号．
5) 数理科学総合研究第4班報告（1959）．
6) 第2回プログラミングシンポジウム報告（1961）．

（昭和57年10月13日受付）