１９５０年頃から６０年代に手がけた ハードウェアおよびソフトウェア

安井　裕
大阪産業大学工学部情報システム工学科

筆者が１９４８年より大阪大学工学部城研究室で携わった計算機に関する試作研究経過の中から初期の研究開発経過とその内容について述べる．トリガー回路などの基本回路から，１０進方式，２進方式の真空管式の電子計算機の製作，さらに磁気ドラム式計算機のソフトウェア，言語プロセサー，計算機自身にプログラムを作成させる試み，ＴＳＳのシステム開発とソフトウェアの検査の計算機による自動化の話題などについて紹介する．

Our early works on the computer hardware and software in the 1950s and 1960s.

Hiroshi　YASUI

This paper presents auther's early works on the computer hardware and software from 1948. It was auther's studies and developments which include the following, the studies and experiments of the basic digital circuits of vacuum tube computers, the design and constructions of the decimal and binary computers by vacuum tube, the programmings of the magnetic drum memory computer with quick access bands, the implementations of the languge processors (SAL1, MALT assemblers and LISP1.5 interpreter), the self-programming system, the developments of TSS and the implementations of the automatic inspection systems of softwares.

１．はじめに　　まずハードウェアとして回路素子からの出発

「我国のコンピューティングの歴史」と題したシンポジウムであって，１９５０年代から６０年代のコンピュータ（ハードウェア，ソフトウェア）の研究・開発に携わった人達が存命中に集まる企画なのだということで参加しました．筆者は１９４８年（昭和２３年）７月から大阪大学の城　憲三教授のもとで真空管式に始まる計算機に関した試作研究に今日まで携わってきました（表１）．城教授は単葉関数などのご研究をなさる数学者でありましたが，工学部の精密工学教室に研究室をもたれてから応用数学解析などに加えて，

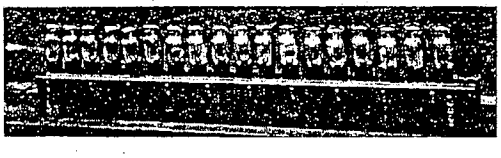

図１　１０進カウンターの実験装置

大阪産業大学工学部情報システム工学科
Department of Informatin Systems Engineering,
Faculty of Engineering, Osaka Sangyo University

第二次世界大戦より以前の１９４０年頃よりすでに計算機械の研究をしておられました．そして戦後のNewsweek誌でＥＮＩＡＣの記事を見て，すぐさま電子計算機の研究に着手されました．したがって，研究室には手動の純機械的な計算機（ハンドル一回転で乗算が出来るミリオネアなど）や電気機械式の統計機械，リレー計算機などのディジタル方式に加えて，さらに積分器や調和解析機，微分解析機などの純機械的，電気機械式のアナログ方式の機器の実物（後述の手作り計算機等と共に工学部電子計算機資料室に常設展示）や文献など多くの情報資産がありました．しかし電子計算機の文献は少なく，我々はまずＥＮＩＡＣの文献にあったフリップフロップ，パルス技術から勉強することになる．電子工学と言う言葉も無かった頃でパルス回路についても最初はテレビジョンの文献などが頼りであった．

表１

いままで やってきたことの あらまし

1948 ・わが国最初の真空管式電子計算機の研究と試作
1959 ・国産初期の汎用機のシステムプログラムの製作
1965 ・LISPインタープリータを開発 (HEAC-2206)
1966 ・計算機自身にプログラムを作成させる試み (Self programming system)
1967 ・実用TSSの実現 俗称：阪大 MAC (Machine Aided Cognition) システム
1968 ・コンピュータによるソフトウェアの検査の機械化
1972 ・超大型機用LISPシステム OLISP (HEAC-2200モデル500,700)
1973 ・わが国最初の長距離コンピュータネットワークの実験（東北大－阪大間）
1974 ・誤差評価の可能な多重精度演算，区間演算プログラムの開発
1975 ・超大型機用LISPシステム OLISP (ACOS-700)
1977 ・経営における意思決定のためのシステム CAPSS
(Computer Assisted Problem Solving System)
1979 ・世界最初のLISP並列処理マシン －EVLISマシン－ の発表
1982 ・EVLISマシンの並列LISPインタープリータの稼働
1984 ・PrologマシンとしてEVLISマシンが稼働 (Prologのマイクロプログラム化)
1985 ・ギガセル級LISP －OLISP(V)－ (ACOS-1000)
1986 ・EVLISマシンの並列LISPコンパイラの稼働
1987 ・EVLISマシンでの並列LISPプログラム開発支援環境，マルチポートメモリ
1988 ・Neuro-Lisp，Vecto-Lisp スーパーコンピュータSX-2H上で試作
1989 ・EVLISマシンとニューロエンジンの結合システム
1990 ・スーパーコンピュータのためのベクトル化 Lisp コンパイラ (SX-2H)
（ベクトル化率９９．８％）

１９１９年のRadio Rev.に載っていたEccles-Jordanの回路としてトリガーリレーと云う名前で出ていたフリップフロップの回路があり，その静的な特性には負性抵抗が存在することがわかる．そもそも記憶させるためには，どのような性質を有しておればよいか，高速に動作させるためには何をなすべきかなどを考察して，ヒステリシス特性に着目してネオン管でフリップフロップを作ったり，また真空管も回路の浮遊容量が小さくゲインの大きい球（タマ）を探すことなどを試みる．それらの試みた中には，例えば戦時中に馴染みのあったエーコン管（acon tube）と呼ばれた蛸の様な形の超短波用真空管でカウンターをつくったり，シフトレジスタをつくったが，形はどんぐり（椎の実）のように（図２）小さい球（直径１０ミリ位）だが，竹刀の鍔のような大きな陶器製のソケットなので実装上問題があるので断念した．その頃（１９４８年）何処にでもあったラジオ用真空管ＵＹ７６で１０進カウンターを作成し，翌年の関西統計機研究会（IBMの統計機械のユーザの集まり）で実演したのがおそ

図２　エーコン管

らく我国で最初のデモストレーションであったと思う（図１）．写真の真空管の数がカウンターの桁数と合わないのに気がつく方もおられると思いますが，そのシャシーにはパルス発生器やその制御回路も組込んである．

２．ＥＮＩＡＣ型真空管式電子計算機の試作研究

カウンターの実験に次いで，ＥＮＩＡＣの文献から１０進４桁のアキュムレータを作った．今度はカウンターのフリップフロップの真空管はＥＮＩＡＣにならって６ＳＮ７である．この球は，大阪の日本橋（その頃の街並みはまだ古本屋が多かったと記憶している）の電気機器専門のジャンク屋で進駐軍の払い下げの中古品を買ってくる．これは双三極管（一本の球のなかに特性の揃った三極管が左，右と二つ封入されている）であるが，既に使用済みの真空管であるので，その使われ方によれば特性に酷くアンバランスなものもある．店の社長の好意で「在庫品を全部持って帰りなはれ，そのうちのええ（良い）のだけ採らはったらよろしい」と云って貰って何度も大学と日本橋を往復する．その頃の検査の手法では真空管一つ毎に，陽極に電流計をつっこんで，格子電圧を変えながら測定値を記録して，それから手で特性曲線を描くのが常套手段であった．しかしその様な事ではとても間に合わないので，ブラウン管オッシログラフのＸ，Ｙ軸に電圧値と電流値をダイナミックに入力して真空管の特性曲線をブラウン管上に描かせ，左右をスナップスイッチで切り換えて比較出来るようにする測定器を帰りの市電の中で考えて，その日の内に手作りして選別作業を省力化した．図３の黒いパネルの１０本ずつ並んでいる光った球はその６ＳＮ７である．この計算機は，配線を換えることにより置数した２桁の数を別の２桁の数に加減したり，３桁あるいは４桁のアキュムレータなどの実演ができた．手前の三つの真空管群は左から制御回路，数値パルス発生のための制御回路，同期パルス発生回路である．左奥は電源回路の一部である．

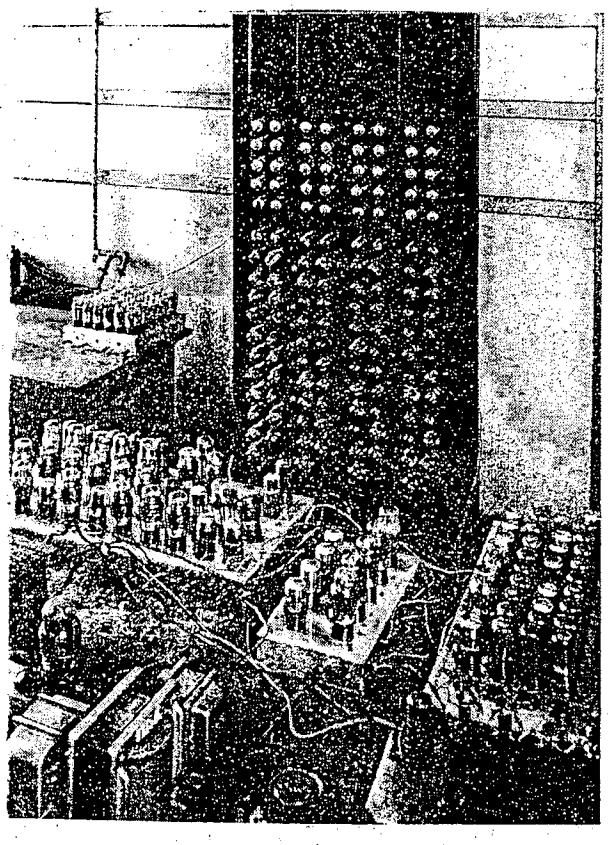

図３　１０進２桁の加減算を行う演算装置

３．２進方式の計算機の試作．

研究室では，H.H.GoldstineとJ.von Neumannらによる研究資料（１９４７）やＥＤＳＡＣのM.V.Wilkesらのプログラミングの本（１９５１）の勉強の傍ら，今度作るのなら２進方式だ！となり，科学研究費を戴い

て着手されたのが，図４の試作機である．しかし文部省から予算を戴くときに付属して届いた試験研究審査員からのコメントには，実用機製作の基礎となる最小限度の模型セットを作製するために使用することを希望すると希望事項がついていた．研究費は８０万円である．直列方式でクロックは１メガサイクル，１アドレス方式，ショートワード２０ビット，ロングワード４０ビット，命令語２０ビット，数値語４０ビット，メモリーは当初は水銀タンクによる超音波遅延回路方式であったが，途中で音速の温度変化の少ない媒体としてＥＴＬ－ＫＡＲＫIIIで採用された光学ガラスによる超音波固体デレイラインに変更しショートワードで３２０語，ダイオード約４０００本，真空管１５００本の規模で，加算０．０４ms，乗算１．６msの設計であった．当時研究室のまわりでは，”真空管が途中で波状的にあちこちボケてきて，次々と取り替えることになり全部の球が揃って働かなくなるのでは”などの声がきこえてきたが，真空管のヒーター電圧は定格の約９０％で加熱するなどの設計がしてあった．後日（１９６４年）ＥＮＩＡＣを作ったJ.P.Eckert氏が来日の時，やはりＥＮＩＡＣもヒーター電圧を下げていたことを語っていた．我々はＥＮＩＡＣのモデルを作るときから既に部品数ｎがたいへん大きい装置であることに充分配慮していた．アーキテクチャはＥＤＳＡＣに準じていたが，ハードの回路設計は，ＢＩＮＡＣ[1]の文献を参考にした．まだシンクロスコープが無い頃なので，１Ｍサイクルのクロックで働く回路のパルスを観測するには，既製品のオッシロのアンプでは周波数帯域が狭く使えないので，ブラウン管を裸にしたオッシロ装置を自作しその電極ソケットから直接ビニール線を単線で延ばして，増幅器を介さずその線の先を引っかける様にしたプローブとして観測していた．偏向感度を良くするために加速電圧を低くしてある．したがってブラウン管上に出るビームの映す波形が１ミリ位の太さになる．また，計算機内部の各回路間の配線も予備実験では同軸ケーブルを使ったが，反射波形が発生し実装上実用的でないので，普通のビニール被覆の線を採用した．これも外国機が入ってきたとき見学会でその実装を見せて貰ったら同じように同軸を使っていなかったので我が意を得たりと嬉しかった．製作に携わるのは専任職員として牧之内助教授と助手の私と毎年研究室に新しく入ってくる学部４年生２，３人である．しかし，製作，調整期間が長びいている間に（ゲルマニュームダイオードの形状も当初はセラミックの容器からガラスさらに細いチューブ状へと変遷する．），トランジスタによる国産の汎用機が出るようになり実験は中止となった．そして大阪大学の学内共同利用の計算機として，日本電気の１０進方式の計算機ＮＥＡＣ２２０３を城研究室に預かりお世話することになる．

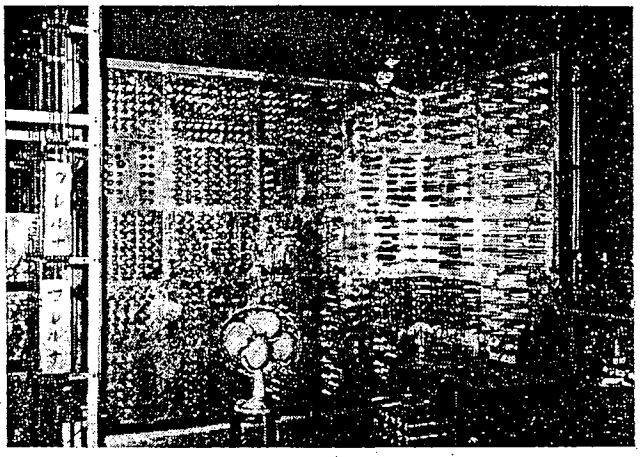

図４　直列２進方式の試作計算機

4．Quick Access Band をもつ磁気ドラム記憶方式電子計算機のソフトウェアの製作

日本電気の初期の電子計算機ＮＥＡＣ－２２０３（１０進方式，１２桁／語，命令は左右に２命令／語）の主記憶は１トラック１００ワードのドラムメモリーで２Ｋワードであった．しかし実行速度を上げるために技術的にはquick access bandとも，または一般にはテンポラリメモリーと称した１トラックを５分割し各々に同一の内容を読み書きする１トラックで２０ワードのメモリーが二つ準備されていた．すなわちこのトラックを使用するとアクセスタイムは１／５となる．番地は８０００から８０１９までと，８１００から８１１９番地までである．プログラミングに際して，片方にプログラムをもう一方をデータ領域として配置するなどの工夫をして，命令一つ毎の実行時間とドラムの回転位置を考えてオペランドとなるデータの格納場所をばらまく．この様にすることによりドラム上の情報への待ち時間を少なくする．テンポラリメモリー上で次に続くプログラムの一群をテンポラリメモリーへロードするのも，終わりの番地に行ってからするのではなく少し手前でロード命令を書いておく．そのため出来上がったプログラムはそのアドレス部をみると殆どがテンポラリメモリーの８０００番地と８１００番地台のアドレスばかりが書いてあるということになる．ライブラリの作成などに結構処理速度が速くなるので楽しく製作した．後年，バファやキャッシュメモリーの利用状況の話題が賑やかになったが面白く聞くことができた．

5．ＳＡＬＩ（Symbolic Address Language Input routine）システム

まだアセンブラーそのものが珍しい頃で，また商用の外国機も含めてアセンブラーはマルチパス方式が普通であった．しかし紙テープ（カードの場合もある）の入力で，さらに一般にはメモリー容量が小さいので，紙テープへ中間出力を出すシステムとなる．入出力機器の速度が遅く，また信頼性も低いので，ユーザは割り当てられた限られた計算機利用時間中に効率的な計算（ＣＰＵを数値計算に使う事に専念する）を望んで，アセンブラーなどの利用は避けていた．中間コードなどの情報の出力に機械のパンチ（鑽孔）ミスが起こったりすると何をしているのか判らなくなる．またマルチパス方式をとると媒体が磁気テープになっても結構時間がかかる．そのためユーザーは機械語のローダの様な絶対番地と相対アドレスが許される程度のイニイシアルインプットルーチンを使うことになる．

筆者はライブラリなどの自分が作るソフトウェアの開発環境のためと，ユーザーの人達に記号番地の便利さを知って貰うためにワンパスのアセンブラを作り大学のユーザーに提供した．ワンパスのアセンブラでは，記号番地の処理系を作る側にとって興味のおこる所はアセンブルの途中で出くわす未定義の記号アドレスの処理である．当時ではＥＤＳＡＣ流にジャンプテーブルにするか，未定義アドレスをもつ命令のアドレス部を利用して，未定義アドレスをもつ命令を芋ずる式に手繰れる様にするのが普通であった．前者は余分にメモリーが必要であり，後者では未定義アドレスをもつ命令のアドレス部を使って次に手繰る命令のための格納場所を記入しておくことになる．したがって，もしアドレス部に記号番地をオリジンとする相対アドレスを書くことを許すとするならば，ちょっと工夫が必要となる．そこでその解決には命令の格納場所の全ての位置を印しできる領域をつくりそこに未定義アドレスをもつ命令の格納場所を１ビットでマークすることにした．しかしこのマシンは１０進方式なので１ワード１２桁を１０進化した２進数で記録することにした．

したがって，命令語の４桁のアドレス部では，未定義のアドレスをもつ命令の４桁のアドレス部に，例えば１桁の相対アドレスとシンボルの個数が３桁分まで盛り込める様にになる．後日，J.MacCarthyのＬＩＳＰ１．５のマニュアルのＧＣのところでビットテーブルでやると書いたくだりがでてきたがすぐ理解できた．

このときに作成したアセンブラはＳＡＬＩ[1]となずけており，ＯＰコード部は１０進２桁を機械語そのまま書くことにしている．加算が２０，引き算が２１，アキュムレータをクリヤーして加算が３０，同じく引き算が３１と，ハードウェアのデコーダの設計をプログラミング講習会でユーザに説明して理解して貰うとかえって喜ばれ，プログラミングに支障はなかった．その頃のシャノンの情報理論の影響で，記号アドレスは文字列が長くてもよく最後の４字が有効にしてあるので，アドレスとしての冗長度をもたせてある．このようにシンボリックアドレシングの便利さを提供する機能に徹したので，ほとんどのユーザに利用して貰い，ユーザの中にはアセンブラを密かに解読する人も出てきた．気楽に日常茶飯事のことの様に欠かすことなく役にたつことをねがってＳＡＬＩには「塩」という気持ちがこめてある．実アドレスを二つあるテンポラリメモリー上に記号アドレスで設定したいときは（ＣＮＳＴ＃１）のように＃１または＃２を書くことで指定できる．処理速度はソーステープを光電式紙テープ読み取り機からパリッパリと音をたてて読み込んで，むにゃむにゃとメモリーが働いて実行可能状態となる．処理速度は記号アドレスの書かれた命令に対して１秒間に約４命令の速度であった．また，サブルーチンのボディとしてリンクできるオブジェクトテープの出力や，シンボリックなアドレスのまま使用できるダイナミックデバッギングルーチンなどのユーティリティルーチンもシステムの一環として整備した．ループの所は何回目からどのステップで実行中の各種レジスタの値も含めて何処の情報をどれだけ出すかの制御もできる．　これはシンボリックアドレスによるコントロールワードで指定できる．　一回のランでプログラムの働く様子がわかりデバグが捗った．その使用方法とコントロールワードの例を次に述べるＭＡＬＴシステムの例を兼ねて当時のユーザ会の資料から図５に示す．

６．ＭＡＬＴ（Mnemonic Assembly Language Translator）システム

しかしながら当時の大学のユーザーは，あちらこちらの会社企業のマシンを利用されている方達も多く，機種も数機種に及んでいた．内外有名各社製の他かに，現在では忘れ去さられ名前も聞かれなくなった外国機も何種類かあった．このような既に他のマシンの経験をもつユーザに役に立つのではないかと考え，また大学のマシンももう少し大きい規模でコアメモリで機械語も異なる機種が追加されることが近ずいていたので，新しくアセンブラーを作成した．今流に云えばユーザーフレンドリーなシステムとでも云うべきか，記号アドレスに加えてＯＰコードもユーザーが勝手に名前を決めて宣言してもらえるシステムである．これには筆者の夢があり，当時では，その全ての実現は大変なことではあったが，文献４の一節を今ここにそのまま取り出すと”低いレベルではハードウェアのコントロール回路のデコーダ，エンコーダのファンクションテーブルを，計算機各部のゲイト群に対応するディストリビュータに接続した記憶領域に記憶した情報として置き換えた方式をとることにより計算機の機能はソフトウェア的なロジックパッケージとして，ユーザの目的に応じて変更定義することができ計算機の機能に多様性をもたせようとするものであ

2.2 Check point type の操作法

Control word を読込んだ後にこのroutine の格納開始番地から実行する。routine は第1段階で check point との linkage を完了してK.B.をcall する。K.B.より対象となるprogram の実行開始番地を4桁で与えるとdebugging を開始する

3.1 MALT形式の program への応用例。

Debugging routine の動作の一例として check point type で次の仕様に対しての結果を示す。

仕様。別稿のMALTの説明に用いた$\sqrt{X}$の製表routineについて

Check point名	skip 回数	印刷回数	Check point	照会内容	連続語数	印刷形式
1		1	MAIN	MAIN のつぎから	3	INST
2		5	GOSQ	Register のみ		DATA
3		6	ENT	X#2	2	DATA
	1	1		LOOP のつぎから	4	INST

のようにdebugging を行うための control word を含むprogram tape は次のごとく準備される。

－MALT LANGUAGE－

Debugging を行うために編成した source program。

```
4304ZE

LT2#04 UJ#43 SETX#72 FOF#58 TYPSP#61 CHB#19 PRNTC#65
CLA#30 A#20 T#11 PRNTN#64 SETXJ#77 JXNZ#76 LOWX#71
HALT#44 NEFOT#10 RAISX#70 －#21 ＋#20 0＋#30 －MLT#23
＋MLT#22 0＋MLT#32 バグス#30 タス#20 ヒク#21 ワル#17 ワメモ#12
バカ#32 カカル#22 オウル#11
```

－224－

```
0ZA

(C.W.)
010Z(MAIN) 5301Z(MAIN)
050Z(GOSQ) 010Z0
060Z(ENT) 020Z(X#2)
110Y 5401Z(LOOP)
－0Z1
```

Control word の部分。

```
60ZA
(MAIN)LT201XUJ01Z2 LT2010X SETX1Z(N#2) FOFOY UJ 0Z(ENT#2)
(LOOP)TYPSP0154Y CHB0Z(SPA0#2) PRNTC0106Y CLA0Z(X#2)＋0Z(.01#2)
オウル0Z(X#2) PRNTN0102Y CHB0Z(SPA0#2) PRNTC0106Y 0＋0Z(X#2)
(GOSQ)SETXJ 3Z(SQRT)NEFOT0Y PRNTN011lY LOWX1Y (ENT)JXNZ1Z(LOOP#2)
HALT0Y(X)0ZI (.01)01ZD(SPA0)6505555Y5505523Y(N)99ZI

/(LOOP) /(GOSQ) /(ENT)

(SQRT) ZA

@
0301X 430121 400221 44.3X 42.2X 731Z1 721521 140421 070721 5002X 701X 430421
01ZD 110121 030101 140121 3201421 170151 800220 8201421 110185Z 070121
430821 3001521 521Y 72121 433X 6ZD 999999999992 D
Z@
440Z(MAIN)Z
```

－225－

図5　MALTのソースプログラム例

り，高いレベルではコンパイラ言語の仕様もユーザーに定義させる”ことを考えていた．（のちになってから，ダイナミックマイクロプログラミングなる用語が登場した．当時のダイナミック・マイクロプログラミングを想い着いたときの，眺めていたＢＩＮＡＣの回路図を示す（図６）．また暫くしてSyntax Directed Compilerも文献に見かける様になった．）

そして当時としてそれをまずアセンブラで試みた．それはＭＡＬＴ（Mnemonic Assembly Language Translator）[4]と名付けたシステムである．ユーザはシングルアドレス方式のプログラミングを自分の慣れた好きなＯＰコードを定義してその上でプログラムできる．ＭＡＬＴは「塩」に対する「麦」と云う意味も意識しておりモルトウィスキーの「モルト」でもある．

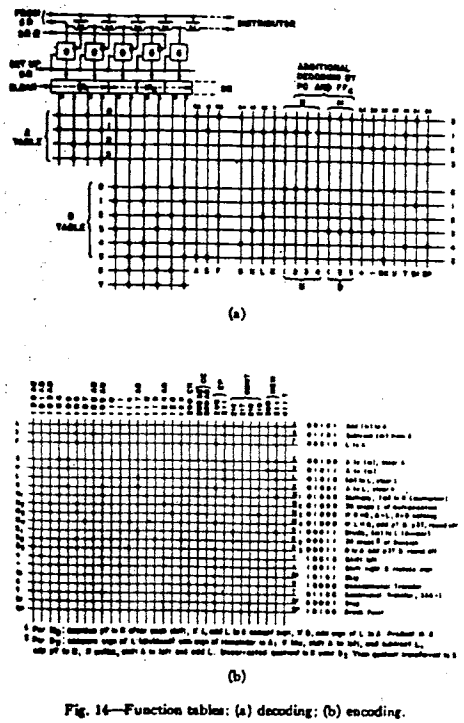

Fig. 14—Function tables: (a) decoding; (b) encoding.

図６　Binacの回路図：文献１p.23より転載

７．計算途中で行方不明にならないロバストなサブルーチン

大学に設置される当時の計算機は，ほとんどハードだけの裸の計算機である．ゼロ号機と呼ばれるような出来たてホヤホヤのマシンのためのソフト作りが研究室で毎日続く．数値計算のサブルーチンや入出力などのユーティリティルーチンなど，ライブラリプログラム作成が主な仕事である．時には我々の作ったサブルーチンが変な答を出すとクレームがくることがある．それもメーカーを経由して学外のユーザーのときもあった．しかしその頃すでに研究室の我々の間では，与えている問題に，原因があることに気がついていた．それは，計算させようとする問題自身のもつイルコンディションである．何でも計算機にかけると正しい答が得られるものと思っている人達には意外であったようである．また実際のアプリケーションで大規模な計算システムを組んで走らせたときに，途中で計算がループを廻ることになって何時までも停止しないことも起きる．そこで有限回の演算回数で必ず結果をだすロバストなプログラムを作ろうとした．具体的な大規模なシステムとして，機械系のアプリケーションであったので，その中で使う実係数代数方程式の実根を求めるルーチンが標的になった．このプログラム[5]は，二つのアイディアから成り立っており，まず原式を一次式まで数式微分して，一次式から根の存在区間を決定する．そして区間内の根を探索し，その根を一つ高次の根の区間の境界とし，つぎにその区間内の根の探索は，バイナリサーチである．そしてバイナリサーチの計算は浮動小数点の演算を指数部と仮数部それぞれを整数とみなして二分する．したがって高々１語分のビット数の回数だけの演算で探索は終了する．勿論，根の存在しない区間は飛ばしなが

ら，得られた根の値が一つ上位の次数の式の根の探索区間の境界にする．このプログラムは，多重根の場合でもさまようことはない．

8．計算機自身にプログラムを作成させる試み：Self-Programming Systemの実験

コンパイラが自動プログラミングと言われて大変もてはやされていた頃，何が自動かと反発して，もう少し自動らしきプログラミングシステムを作ろうとして試みたのがSelf-programming sysytem [6,8]である．丁度その頃，筆者の長男が2，3才で，日々いろいろなことを私らとやりとりしながら彼なりに習得していく過程を看ていたので，こいつをプログラミングに実現させてみようと想ったのが動機である．実現したシステムにはターゲットマシンの命令セットが知らしてあり，外部から与える情報としてはＬＩＳＰのＳ式（ＬＩＳＰのインタプリタもこのプロジェクトと併行して作成中であった）の様なスタイルで，ユーナリとバイナリオペレータ形式で与える入力形式が決められているだけである．ユーザが期待する機能に勝手な名前をつけてオペレータとし，そのオペランドとして適当な値を与えて，その時期待されるゴールとなる値にたいして，満足する結果を求め得る欲しいプログラムをジェネレートさせる（図7）．このシステムは当初の計画ではマイクロプログラムのレベルで遣りたかったのであるが，その時の計算機の環境では機械語命令レベルでやるより仕方がなかった．

9．ＬＩＳＰインタプリタの作成

数値計算ライブラリを作ったり，大学のユーザの人達と計算をしていると，浮動小数点の計算は丸め誤差との闘争であることに問題を感じていたので，記号的な処理の世界でも何とか計算機で扱うことが出来ないかと思っていた．たまたまShort tables of Integralと題のついた洋書の公式集が手元にあったので，式を入力すると，適当な公式を引っ張ってくるシステムを作ろうとしたが，当時では，数値計算や数値解析の話題が盛んで文字・キャラクタの操作などは，事務処理の仕事の部類にみられ，その上ファイルの実現媒体は磁気テープとなり，とても探索の目的には無理な状況であった．その頃の外部記憶としてのオープンリールのテープはとろとろよたよたとまわる有り様であった．また，フリップフロップ回路，論理回路など初期の勉強時代にはブール代数や論理学の教科書を読み，記号論理学の機械化なども興味を持っていたので，ＬＩＳＰ１．５の文献をみたときは，やっぱりやっている人がおったかと思い早速勉強して作ることにした．城教授には，当時の工学部の講義の中では見られないチャーチのラムダ計算など彼らのもっている様なバックグラウンドの必要性を訴えたことをおもいだす．学部４年生の卒業研究テーマとして１０進方式のＮＥＡＣ２２０６上でＬＩＳＰ１．５のインタプリター[7]を作成した．

１０．ＴＳＳの開発とコンピュータによるソフトウェアの検査の機械化

ＩＢＭ７０９０無償貸与の話題が大阪大学に持ち込まれ大論争の後，導入をお断りした事で，その後始末として大型国産機メーカー各社のご好意でマシンの提供のご提案をいただくことになった．M.I.T.のＭＡＣシステムをモデルとしたＴＳＳ構築の共同研究開発の提案を示されたＮＥＣのシステムに決まることになりました．電電公社も加わった阪大ＭＡＣと呼ばれた共同研究がはじまった．筆者は大学側のメンバーとして研究開発に携わる

Vol. 8 No. 3　　情　報　処　理　　May 1967

計算機自身にプログラムを作成させる試み*

石田晴也**　安井　裕***　杉山　博***　城　恵三***

まえがき

人工智能実現への努力は，種々の立場からなされてきた．第1の立場は，最も興味深いもので，非常に原始的な情報処理要素の集合を仮定し，それが学習により自己組織化することをねらっているが，その成果は，人工智能実現の目的から見ればまだ数々たるものであるといえる．これに対して，できる限り複雑な情報処理能力を機械に賦与し，人間が問題を解く過程の，Protocol に基づいて人工智能的経路をさせることを試みる第2の立場がある．この方針に基づく Newell, Simon などの研究は，リスト処理言語の開発により，かなり顕著な成果を報告している[1]．第3の立場は，機械はある程度の情報処理能力を最初から持っていると仮定し，その能力により学習を有効におこない，機械が自らの能力を高めていくことを目的とするものである（たとえば Samuel の Checker Playing Program[2]）．われわれの論文も一つの人工智能の問題を扱っているが，われわれは第3の立場をとっている．すでに報告されている計算機にプログラムを作成させる実験は，第2の立場でたされて学習の余地が残されていないもの[3]や，単に学習の一対象としてとらえたにすぎないもの[4,5]である．この論文では，次のようなひとつのシステム，すなわち計算機が外部からの助けを得つつ学習し，自らにその能力を高め，遂には，ある程度未知の問題に対しても，計算機自らがそれを解決するプログラムを正しく作成し得るようになるシステムを作成することを目的としている．したがって，われわれのシステムでは，人工智能そのものとなる計算機（B-計算機と名付ける）とある仮想の計算機（H-計算機と名付ける．B-計算機と一致することもあり得る．）を用意し，与えられた問題を解決するプログラムを H-計算機の言語で B-計算機が作成できるようになることを目指している．ここに H-計算機の言語は任意であり，学習の最初において B-計算機は H-計算機の言語の持つ意味を知らないという，前提をおく．

実際に作成したシステムでは，B-計算機として，NEAC-2206 を選んでいる．このシステム全体のプログラムは約3,000ステップである．このシステムでは H-計算機が1アドレス方式であることを，仮定しているが，特に，H-計算機として第1表のような命令を持つ仮想計算機を採用し，加減乗除からなる数式の問題について，実験がおこなわれた．

第1表　実験に使用した H-計算機の命令

命　令	意　味
CAD α	(α) → Acc
CSB α	−(α) → Acc
ADD α	(Acc)+(α) → Acc
SUB α	(Acc)−(α) → Acc
MUL α	(Acc)×(α) → Acc
DIV α	(Acc)÷(α) → Acc
STR α	(Acc) → α
（特殊命令）	
LINK α	α番地から始まるプログラムへのリンク命令
MACR α	マクロ命令αを実行する
HALT	停止命令

ただし，Acc: Accumulator を示す．
(Acc), (α): それぞれ Acc, α 番地の内容を示す．

このようなシステムの概略はすでに報告した[6]が，さらに改良を加えたものについて，次節以下に詳細に説明し，最後に実験例を示す．

1. システムの構成

Man（人間）がある Problem（問題）を B-計算機に提示すると，まず Interpreter が Memory（過去に学習してある事例）を参照しつつ，Problem の解決，すなわち答となるプログラムの作成を試みる．そこで解決がなされたときは答のプログラムを印字する．解決できなかった場合は，その時点での既知事項を Trainer と Analyser と Learning Body に報告したのち，Man にその Problem を解くための Advice を要求する．Man より Trainer に Advice が与えられると，Generator が Learning Body のパラメータを適当に選ぶことにより，学習速度の現性において，その確率を，十分1に近づけることができる．

3. 実験例

binary operator, MOVE, ADD, SUB と，unary operator, MINUS, PLUS を学習させる実験の一例では，操作時間も含めて約65分で，これらの operator すべてを学習させることができた．その後 B-計算機はこれら五つの operator に関しては，自立状態にあり，たとえば次のような Problem (P 1)

MOVE (SUB(ADD(SUB(MINUS(ADD(ABCD, PLUS(EFG))), SUB(MINUS(H 1), H 2)), IJ), ADD(KLM, N)), ANSWER)

を提示すると，第1図のような (P 1) のプログラムを，約2.5秒（読み込み時間も含める）で，印字する（解決する）．ところが未知の operator DIV を含む

```
MPROG
CAD     N
ADD     KLM
STR     WORK 1
CSB     H 1
SUB     H 2
STR     WORK 2
CAD     EFG
ADD     ABCD
STR     WORK 3
CSB     WORK 3
SUB     WORK 2
ADD     U
SUB     WORK 1
STR     ANSWER
HALT
```

(R) WORK (U−L ≦ 2) 作作領域を示す

第1図　Problem (P 1) を提示したときの印字

Problem (P 2)

MOVE(SUB(ADD(SUB(MINUS(DIV(ABCD, EFG)), SUB(MINUS(H 1), (H 2)), IJ), ADD (KLM, N)), ANSWER)

を提示すると，第2図のような印字をし，DIV に対する Advice を要求して一旦止る．つぎに未知の operator DIV を学習するための Problem (P 3) と Advice として

MOVE (DIV(A, B), C)

SET (A ∦ 512449048, B ∦ 511414)

GOAL (C ∦ 511732. A ∦ 512449048. B ∦ 511414)

等

を第2図のように与えると約24秒で学習を完了する．この状態において改めて Problem (P 2) を提示すると約2.5秒で解となるプログラムが得られる．

むすび

ここに報告した，実際にわれわれの作成したシステムは，コンパイラーを作り出すシステムのひとつであるとみなすこともできる．このシステムにより，B-計算機は Problem の syntax に対応する H-計算機の言語で実現される semantics を学習することになり，一連りの事例を学習した後には，一種の syntax directed compiler[7] になっている．

この論文での実験をより一般的な形に拡張するためには，まず学習過程に対して，1.5 に触れたように，time dependent な学習が可能になるように考慮が払われなければならない．なぜなら一般の学習では，ある state S_i から，推移する先の state は，time（その state S_i が推移系列中のいくつかの state S_i のうちの何番目に現われたものか）を考慮して始めて一意的に決定され得るからである．その他，Problem の拡張，それに伴なう Advice の高度化，また Analyser の改良など，システムを一般的なものに拡張するにはまだまだ多くの問題が残されているようである．さらに与えられた任意の問題を解決する人工智能を実現したい目的からは，H-計算機が持つべき言語の検討が必要である．

参考文献

1) Feigenbaum, E.A. and Feldman, J. (eds): Computers and thought. McGraw-Hill, 1963.
2) Simon, H.A.: Experiments with a Heuristic Compiler. J. ACM 10, 4 (Oct. 1963).
3) Friedberg, R.M.: A learning machine, Part I. IBM J. Res.

* An Experiment with a Self-Programming System, by Tokuya Ishida, Hiroshi Yasui, Hiroshi Sugiyama and Keizo Joh (Faculty of Engineering, University of Osaka)
** 大阪大学大学院工学研究科　*** 大阪大学工学部


```
MPROG
CAD     N
ADD     KLM
STR     WORK 1
CSB     H 1
SUB     H 2
STR     WORK 2
LINK    GPROG
STR     WORK 3
CSB     WORK 3
SUB     WORK 2
ADD     IJ
SUB     WORK 1
STR     ANSWER
HALT
   UNKNOWN
DIV (ABCD, EFG)
DIV(ATOM, ATOM)
   ADVICE
```

未知の operator DIV を含む Problem (P 2) を与えて得られた計算機の応答．

```
MOVE (DIV (A, B), C)
```

未知の operator DIV (ATOM, ATOM) を学習するための Problem (P 3) を与える．

```
MPROG
LINK    GPROG
STR     C
HALT
   UNKNOWN
DIV(A, B)
DIV(ATOM, ATOM)
   ADVICE
```

(P 3) に対する応答．

```
SET (A ∦ 512449048, B ∦ 511414)
GOAL (C ∦ 511732. A ∦ 512449048.
B ∦ 511414)
等
```

(P 3) のための Advice を与える．

```
   LEARNED
```

約24秒で学習を完了する．

```
MOVE(SUB(ADD(SUB(MINUS(DIV(ABCD, EFG)), SUB
(MINUS(H 1), H 2)), IJ), ADD(KLM, N)), ANSWER)
```

学習後 (P 2) を計算機に提示する．

```
MPROG
CAD     N
ADD     KLM
STR     WORK 1
CSB     H 1
SUB     H 2
STR     WORK 2
CAD     ABCD
DIV     EFG
STR     WORK 3
CSB     WORK 3
SUB     WORK 2
ADD     IJ
SUB     WORK 1
STR     ANSWER
HALT
```

約2.5秒で (P 2) の解となるプログラムを印刷する．

第2図　Problem (P 2) に含まれる未知の operator を学習させて (P 2) のプログラムを得るまでの過程とその印字

図7　Self-programming system　の実行例：文献8より（中略）

ことになって，やがてめでたくシステム[9]は１９６７年に大学内のユーザーを対象にサービスを開始しました．しかしながら，システムが非公式に少しずつ働きだした頃，そのテストをするのは，筆者等が大学側として遣らねばならなくなりました．そこで，ユーザーコマンドのシンタックスから自動的に一連のコマンドの続きを生成させそれを紙テープに出力して，数カ所あったＴＳＳ端末に持って行き，それぞれ一斉にテープで入力させてシステムのレスポンスを観測しました．テスト用のコマンドの生成のためのストラテジーにはシンタックスツリーの辿り方が選択出来るようにしてあったが，ランダムに選ぶ作戦が人間では思いつかないコマンド列を生成し，システムの虫とりに役だち思わぬ効果があった．このテストシステム[10]のアイディアは後になって，まだテストしていないモジュールのパスを選択して行く学習機能を伴った生成方法をテストプログラムの生成に適用し，検査の対象とするソフトウェアをグラフ化してソフトウェアモジュールの効率的なテストを行うシステム[11]を提案した．これは第１４回のプログラムシンポジウムで報告してある．

１１．むすび

かって，ＥＶＬＩＳマシンを作って働かしていた頃，記号処理の研究会の懇親会での雑談中に東大の後藤英一先生が私に言われたことがあります．「お互いに”雀 百まで踊り忘れず”ですね 」と．たしかに，いつまでたっても，ついついハードウェアを作ろうとし，ハードウェアとソフトウェアの間をさまようことになりそうである．巷間のユーザーのなかにはソフト，ソフトとソフトウェアが全ての様に口にして，ハードを軽視してよいような誤解を与えていそうな人達がいますが，やはり人に使って貰えるすぐれたプログラムを作るためにはハードウェアをちゃんと勉強してそのアーキテクチャもよく理解しておらないといけないと思っています．並列処理や実時間処理が盛んになってきた昨今，いまさらにこの思いをつよくするところです．

話題も表１にあるように，次の１０年の１９７０年代に入って来ました．ここに，ずうっとこれらの研究に共にご一緒できた先生や多くの諸兄のご尽力とご支援に対してこころより深く感謝の意を表しまして，この辺りで本稿を終わります．

参考文献

１）A.A.Auerbach, J.P.Eckert,Jr., R.F.Shaw, J.R.Weiner,and L.D.Wilson : The Binac, Proc.I.R.E.,pp.12-29,1952.

２）城憲三，牧之内三郎，安井裕：大阪大学の電子計算機について，電気通信学会誌,Vol.40,No.6,pp.730-732,1957.

３）安井裕：Basic NEAC-2203のための Symbolic Address Language Input Routine，情報処理学会第２回全国大会予稿集,pp.19-20,1961.

４）安井裕：Mnemonic Assembly Language Translator MALT System，情報処理学会第４回全国大会予稿集,pp.29-30,1963.

５）安井裕，若林堯雄：実係数代数方程式の実根を求める一方法について，情報処理学会第５５回全国大会予稿集，pp.29-30,1964.

６）石田喬也，安井裕，杉山博，城憲三：計算機自身にプログラムを作成させる試み，情報処理学会第６回全国大会予稿集，pp.3-4,1967.

7）安井裕，立花道明，山本昌弘：NEAC-2206のＬＩＳＰについて，情報処理学会第７回全国大会予稿集，pp.36-37,1966.
8）石田喬也，安井裕，杉山博，城憲三：計算機自身にプログラムを作成させる試み，情報処理，Vol.8,NO.3,pp.121-130,1967.
9）安井裕：大阪大学のタイムシェアリングシステム，情報処理学会第１回月例会資料，1969.
１０）安井裕，藤田安臣，泉正晴：テストプログラム・ジェネレータによるシステムプログラムの検査の機械化，情報処理学会第１０回全国大会予稿集，pp.55-56,1969.
１１）安井裕，治田倫男，永田恒一，藤田安臣：テストプログラム・ジェネレータによるシステムプログラムの検査の機械化，情報処理学会第１４回プログラミングーシンポジウム報告集，pp.93-105,1973.