計算機設計言語間の変換*

萩原　宏** 黒住祥祐***

Abstract

In order to design a computer by use of computer (CAD), we have proposed the design language of computer. This language has three levels which are T, B, and M-level respectively. In this paper, we deal with translation between this language.

First, various designs of logical control circuits are described. The three methods of design which are possible to computer-aided design are adopted.

Next, the state assignment type of sequential network is discussed. State minimization, state assignment, and reduction of Boolean expression are examined.

Finally, our translation algorithm and processors are described.

1. まえがき

さきに，筆者らが提案した3段階の計算機設計言語(T, B, M-言語)[5]間の変換について述べる．

T-言語は計算機のマクロな記述に適し，方式設計を目標としている．

B, T-言語は計算機のミクロな記述に適し，論理設計に相当する言語である．これらの言語間の変換とは設計作業そのものであり，従来からの各種の設計方法が利用できるかどうか検討する．

ところで計算機の設計機械化において，シミュレーション，または記述のための言語仕様の発表は多いが，設計言語とその変換法について詳しく述べた例は意外に少ない[3),4)]．この理由として，次の事情が考えられる．従来の最適設計の手法はきわめて小規模の設計には利用できるが，大規模になると処理時間が飛躍的に増加し，高速計算機といえども不可能な状態になる．また，適用できる範囲がかなり狭く，総合的な設計システムに採用しにくい．その結果，最適設計をあきらめるか，設計条件を狭ばめるかのいずれを選択しなければならない．

筆者らは，まず前者の方針をとり，一貫性のある設計システムを作ることを試みた．後者の強い条件下での最適設計の手順は，システム・ライブラリとして漸次追加すれば，人間-機械系によってその本領を発揮できるであろう．

本文では，最適設計を試みない理由，採用した設計手順，および実際に作成した変換プログラムについて説明する．

2. 論理回路の設計手順

一般の論理回路(順序回路および組合せ回路)の設計手順を **Fig. 1** に示す．各手順に対応した筆者らの設計言語の種類を同図右側に示した．方式仕様とは，入出力装置・記憶装置・演算装置などの大きさ・速度・種類を決めることである．この段階から T-言語により記述できるが，さらに詳細に記述された T-言語プログラムは方式図に相当する．つまり，レジスタの大きさ・種類，演算器の大きさ・速度・種類などを記述する．また，制御信号の発生順序および制御記号の送り先も完全に記述できる．

さて，一般の設計では，方式構成図に示された仕様から，組合せ回路と順序回路とを分離する。このときゲートの統合，組合せ回路の統合などのマクロな最適化が施される．この最適化された方式構成の記述も T-言語で可能である．分離された順序回路はタイム・チャート，または，状態図に浮きぼりされ，制御回路

---

* Translation between the disign language of computer. by Hiroshi Hagiwara (Faculty of Engineering, Kyoto University) and Yoshisuke Kurozumi (Faculty of Science, Kyoto Sangyo University)

** 京都大学工学部
*** 京都産業大学理学部

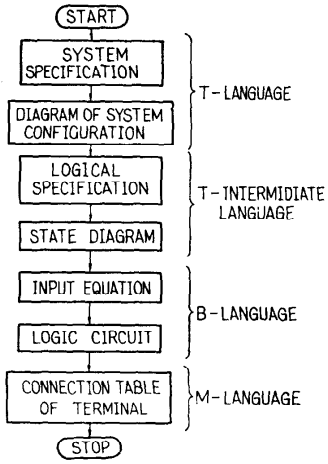


Fig. 1 Design steps of logical circuit.

の設計方式に従って状態割当て，入力方程式の決定がなされる．この制御信号発生回路は B-言語で記述される．順序回路設計が終了すればレジスタ，演算器などの素子とともに論理回路図に記述される．これらは設計言語では M-言語で表現する．この後，実装設計に移る．

3. 制御回路の設計方式

計算機の制御回路は，おもに順序回路から導びかれる．順序回路の設計は，一般には，状態図，遷移表，状態の簡単化，状態割当，励起表，応用方程式，入力方程式，論理式の簡単化などの手順を踏む．このような手順を定式化することは，設計言語間の変換において重要な問題のひとつである．そこで制御回路の設計方式について分類する．

3.1 集中割当型制御方式

主制御装置として制御レジスタを置き，制御信号はすべて制御レジスタに発生させる．制御レジスタの長さは全状態数を S とすれば $\langle \log_2 S \rangle$ 以下である．このとき，制御レジスタの状態を直ちに順序回路の状態に割り当てる方法〔Fig. 5 (a)〕と，制御レジスタの状態を時差パルス回路に通し状態数を多くしたうえ，順序回路の状態に割り当てる方法 (**Fig. 2**) とがある．時差パルス回路を使うと，さらに制御レジスタを小さくすることができる　これらの方法は制御レジスタのフリップフロップ (以下，フリップフロップを FFとする) の個数を減少させる点では有効であるが，状態数の多い順序回路を作ることには適さない．たとえば，計算機1台の状態数は 1,000～10,000 程度と考えられ，状態の簡単化，最適状態割当は実用上不可能である．また，実際に制御回路を1つの制御レジスタとその状態遷移回路のみで構成することは少なく，見通しをよくするためエンコーダやデコーダを挿入して制御回路を分散する方式がとられる．

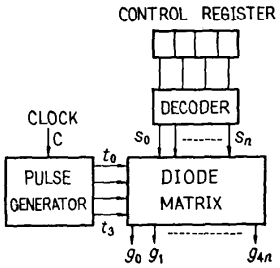


Fig. 2 Control signal generator by decoder and pulse generator.

3.2 分散割当型制御方式

制御回路を分散させ，命令取出し段階，命令実行段階で大きく分け，各命令の実行もおのおの独立した小制御回路 (サブ・コントロール・ブロック――以下，SCB とする) で制御する方式 (**Fig. 3**) である．この方式も時差パルスを使う方式と使わない方法とがある．

もっとも広く採用されている方式で，全体としての制御 FF の数は増加するが，見通しのよい設計ができる．また，製造者・保守者にとっても扱いやすい制御回路となる．しかし，SCB (小制御回路) の分割の方法が問題である．分割の良し悪しは回路の複雑さに影響を与えるが，その方法に標準的な手段はなく自動化は困難である．そこで，設計者が分割を設計言語で指定する．これは T-言語の制御部で指定する．なお，SCB に分割しないで1個の制御回路で構成する方式が **3.1** であり，分割を最高に細かくすると次の **3.3** 方式となる．

3.3 順序型制御方式

1 SCB を 1 FF で構成し，1状態を割り当てる方式である (**Fig. 4**)．左端の信号により F1 がオンとなり，その時刻での制御信号を発生する．次の時刻では，F1 はオフとなり，F2 がオンとなる．F2 はデイレーを持っているため数ユニット・タイム持続する信号を発生することができる．なお，状態遷移をクロックにより行なわないで，ゲートの終了信号によって行なうようにすると，非同期式制御回路となる．

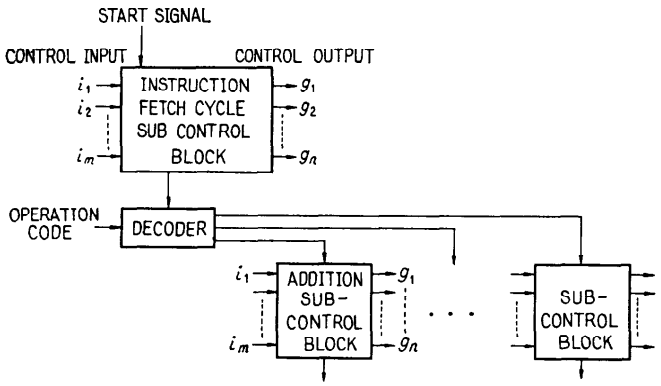


Fig. 3 Assignment type of control circuit by sub-control block.

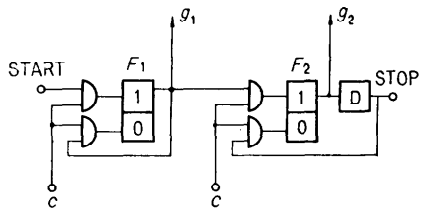


Fig. 4 Sequential type of control circuit.

順序型制御方式は順序回路の状態数に等しい制御FFを必要とするため，コストが高くなる．しかし，状態簡単化や最適状態割当，論理関数の簡単化などの操作がはいらないため，機械化はもっとも簡単である．

3.4 遅延型制御方式

3.1，**3.2** および **3.3** の方式はFFによる状態遷移を利用し，遷移のタイミングはクロック・パルスに同期させる．ところで，簡単な制御回路ではパルス・アンプまたは，ディレーにより回路の遅延を利用して制御信号を発生させる方法がある．

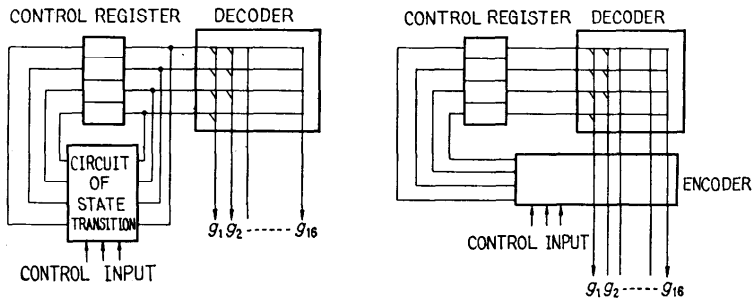
この方法を多段に使うことができるならば，フリップ・フロップ，クロック・パルスなどを使わないで制御回路が設計できるから，コストが安くなる．しかも，**3.3** と同様に機械化はもっとも簡単である．しかし実際には，数段以上の多段にすると不安定になること，および製造・保守が困難であることから，この方式のみで多段のものは実用化されていない．

3.5 混成型制御方式

この方法は，**3.2**，**3.3**，**3.4** の方式を適当に組み合わせてもっとも能率のよい制御回路を設計する方法である．たとえば，乗算回路などでフリップ・フロップとディレーによって，簡単で安価な制御回路を構成している例は多い．しかし，この方法は人手には可能であるが，機械化の不可能なほど困難である．

3.6 マイクロ・プログラムの方式[5]

これは**3.1**と同じく集中制御方式が多いが，**3.1** では制御レジスタの状態遷移回路を別個の組合せの回路とし，これを簡単化する〔**Fig. 5** (a)〕．制御信号は制御レジスタを直接取り出すか，デコードして取り出している．ところが，マイクロ・プログラム方式では制御信号を必ず制御レジスタのデコーダの出力として取り出し，制御レジスタの状態遷移回路は（組合せ回



(a) Assignment type of control circuit.
(b) Micro programed type of control circuit.

Fig. 5

路で作らず)，デコーダの出力をエンコードすることによって行なう方法〔**Fig. 5** (b)〕である．したがって，制御レジスタの状態割当，状態遷移回路の簡単化などの手順がはいらないため，機械化は簡単である．

以上の考察により，機械化可能な制御回路の設計方式は **3.2**, **3.3** および **3.6** の3方式である．

3.2 の方式を T-言語の制御部において assignment と指定する．これは次章でさらに検討する．**3.3** の方式は sequence と指定する．この方式の変換手順はもっとも簡単であり，一般のプログラミング言語の翻訳とほぼ同じ手順で行なえる．この場合は制御回路を B-言語に変換することなく，直ちに M-言語で記述することもできる．**3.6** の方式は microprogram で指定する．この方法も簡単である．

4. 割当形制御方式の変換

制御回路は T-言語の順序部で記述され，その SCB (サブ・コントロール・ブロック) の分割点と大きさは T-言語の制御部で記述される．T-言語の変換が進みゲートと制御回路とが分離された段階では，次のような中間表現となる．中間表現は状態図に相当する．状態遷移のみを抜き書きしたものである．**Fig. 6** (a) の Q_i は状態，I_i は制御入力，E, F は SCB の入口と出口をを表わす．Moore model 型で制御出力 g_i は各状態に存在する．一般の順序回路の設計では，この状態図から出発して最簡形式の入力方程式を求めることになる．この手順を機械化するために，開始信号は制御入力，終了信号は制御出力と考え，停止状態を表わす状態を1個追加する (**Fig. 7**)．開始信号が0である限り SCB は停止状態 Q_0 にあって，停止中

```
Q1(E/g1,Q2)
Q2(Q1/g2,Q3)
Q3(Q2,I1/g3,Q4,Q7)
Q4(Q3/g3,Q5)
Q5(Q4/g5,Q6)
Q6(Q5/g6,F)
Q7(Q3/g7,Q6);
```

↑(a) Expression of a sequential circuit.
→(b) state diagram for Fig. 7 by Moore mode.

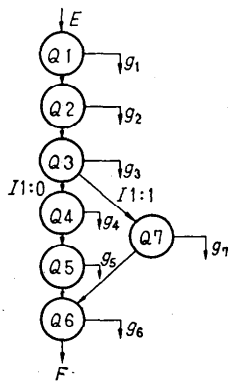


Fig. 6

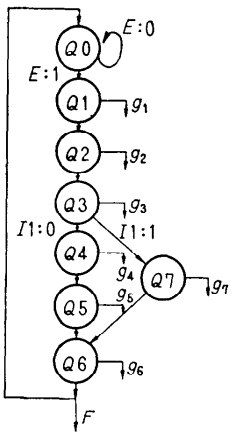


Fig. 7 Modified state diagram of Fig. 6.

(dormant) である．Q_0 ではすべての制御 FF がリセット状態にある．計算機のイニシャル・リセットの状態はすべての SCB が状態 Q_0 であるときである．開始信号のうち1個 (開始信号であるから，2つ以上同時にはオンになることはない) がオンになると SCB は動作中 (busy) となり，クロック・パルスおよび制御入力によって状態遷移を起こす．制御出力は各状態に依存するもの (Moore model) とする．SCB の終了状態に達すると終了信号を出して停止状態 Q_0 に移る．終了信号は終了状態の制御出力そのものか，ディレーにより少し遅らされた信号である．これは次の SCB の開始信号となる．現実の回路では開始信号は命令デコーダの出力から導びかれ，各命令の実行が SCB となる．SCB の終了信号は命令実行終了であり，命令取出しブロックか，割出し処理ブロックの開始信号となる．

以上のように定型化することにより，T-言語により記述された順序回路はいかなる箇所で分割されても SCB として表現できる．この SCB は開サブルーチンにあたるものであるが，閉サブルーチンにも適用できる．T-言語では開サブルーチンと閉サブルーチンの両方の記述ができる．開サブルーチンは構文分析時にメインルーチンに組み込まれるため，順序回路の作成時には影響を与えない．閉サブルーチンの取扱いは順序回路の作成時まで持ち越される，閉サブルーチンの処理は次のように定型化する．サブルーチンはコールされるごとにその入口用意し，OR で1本にまとめサブ・ブロックの開始信号に導く．サブルーチンの変換は SCB と同じである．SCB の終了信号はコールされ

た箇所へ導びかれる．一方，メインルーチンはコールする状態で，制御出力そのものか，適当に遅らせた信号をサブルーチンの呼出し信号として，サブルーチンの入口に導く．メインルーチンの次の状態は待機状態であり，サブルーチンの終了信号を制御入力として受け，終了信号がオンになるまで待機する．この方法により，コールした状態へリターンすることができる．

結局，順序回路の設計では Fig. 6 のような SCB から出発し，状態割当，入力方程式の簡単化などを行なえばよいにとになる．なお，順序型のサブルーチン・コントロール・ブロックの作成法とサブルーチン・コールの方法は，上記の割当型と同じである．マイクロ・プログラムではサブルーチンをコールするごとに制御レジスタの値を退避させる作業レジスタが必要となる．多重コールを許しているから，この作業レジスタは first in last out 型のレジスタでなければならない．この作業レジスタは変換プログラムにより生成される．たとえば，コールの多重度を n とすると〔$\log_2 n$〕ビットの多重ポインタ・レジスタと制御レジスタと同じビット数を持つ n 個の退避レジスタ，およびそのセレクタ・ブロックが生成される．人手による設計では，あいた一般の作業レジスタなどに退避させることもできるが，この手順を定式化することは困難である．

5. 順序回路の変換

SCB の中間表現は状態図に相当する．このことは一般の順序回路の設計に帰着できることになる．そこで，1つの SCB に注目し，従来からの設計方法が適用できるかどうか検討する．

5.1 状態の簡単化

状態の簡単化で計算機向きの方法は Paull-Unger の方法[2] である．この方法による処理時間の概算を行なったところ，大型計算機で，状態数が 100 のとき，数時間を必要とする．さらに，筆者らが目標とする SCB では一方通行的な状態遷移が多いから，簡単化が成功する機会はほとんどない．以上，2つの理由から，SCB の状態簡単化は行なっていない．

5.2 状態の割当

状態の割当については Hartmanis の方法[2]がある．この方法の所要時間の概算と成功の確率を評価したが，状態の簡単化に比べ，所要時間は少ないが，最適割当が成功する機会はきわめて少ないことが判明した．

次に考えられる方法は，隣接する状態に adjacency な論理変数を割り当てる方法[2] である．サブ・ブロックの状態数を N_q とする．分岐のない直進型のサブ・ブロックについては $2^n \leq N_q$ なる n 個の変数を使って1箇所を除いて他のあらゆる隣り合う2つの状態に adjacency な変数を割り当てることができる．1箇所とは状態数が 2^n でないときに必然的に起こる箇所であるが，残った組を組合せ禁止とするならば，adjacency でない箇所がどこになっても入力方程式の複雑さは同じである(注)．adjacency な変数の組の数は1つの初期状態から出発して多数あるが，これらのうちどれを採用しても入力方程式の複雑さは同じであるから，もっともシステマティクに発生できるものを選べばよい．SCB は Q_0 において停止中のループがあるため，Q_0 に変数の組 0～0 を割り当てなければならない．そこで，次のアルゴリズムによって，他の状態に順次変数の組を割り当てる．

変数を $F_n, F_{n-1}, \cdots\cdots F_2, F_1$, とする．

(1) 最初に，各 F_i に対し 2^{i-1} 回の0を与える．

(2) 次に，各 F_i に対し 2^i 回の1を与える．

(3) 次に，各 F_i に対し 2^i 回の0を与える．

(2), (3) を N_q 組の割当が終わるまで繰り返す．残った変数は組合せ禁止とする．上記のアルゴリズムを使うと隣り合う2つの状態をすべて adjacency にすることができる．しかし，分岐のない直進型の SCB はむしろ少数派であり，一般には多数の分岐を含んだ SCB となる．この分岐型の割当については，多くの問題点が未解決のまま残されている．これらの問題点が解決されたとしても，入力方程式の最簡形式を求めることは，一般的には非常に困難なように思われる．なぜならば，これまでの方法の最簡形式の条件は，AND-OR 2段の応用方程式に現われる変数の数を最小にすることであった．しかし，この条件は必ずしも実用的ではない．つまり，実装設計において，fan-in, fan-out の制限や多段論理，および NAND, NOR などの論理素子の使用により，AND-OR 2段という条件は有効に利用されないことがある．理想的には，実装設計での条件も考慮したうえで，最適状態割当を行なうべきである．この意味での研究はきわめて多い．しかし，条件が細分化されるため，この種の設計方法を一般的な設計機械化システムに組み入れる場合には，設計プログラム・ライブラリの1つとして参加さ

注：これは Fig. 6 のような場合にいえることである．われわれが目標とする直進型 SCB では Q_0 にループがあるから，事情が少し違うが，たかだか最小項が1つふえるだけである．

せる予定である．そして T-言語のコントロール記述部で指定された設計条件により，適当な設計プログラムを選択し実行することにより最適設計が可能となる．筆者らの当面の目標はサブ・ブロックの設計であり，最終的に使用される回路素子の種類は予想していない．B-言語表現は AND-OR の2段の論理回路であるが，これは実装設計へのデータとして使われるため最終的には未知である．そこで，筆者らが採用した状態割当の方法は T-言語の中間表現で記述された SCB に直進型・分岐型の区別をせず，上記のアルゴリズムを使って割り当てることにした．これは必ずしも最適割当ではない．割当が決まれば応用方程式の作成は機械的に可能であり，使用する制御 FF の特性方程式から入力が決定される．その後，SCB を B-言語で記述する．

5.3 論理式の簡単化

T-言語で記述された論理関数や B-言語で記述された SCB の入力方程式は多出力関数であり，論理式の簡単化が適用される可能性がある．そこで，B-言語から M-言語への変換の途中で，論理式の簡単化を行なうことができる．論理式の簡単化の方法としては，すでに多くの研究があり，計算機向きの方法としてクイン・マクラスキ[2]の方法，およびこれを改良した方法[1]などが知られている．しかし，変数の数が増すと所要時間が飛躍的に長くなり，B-言語から M-言語に変換する際に，画一的に簡単化を施すことは無理である．特に，T-言語の関数記述部や条件文の論理関数などの簡単化では，どの関数を組み合わせて多出力関数とすべきかの機械的判定が困難で，変換プログラムには組み入れていない．現実に，しらみつぶし的に関数と組み合わせて簡単化を試みても，時間がかかるばかりで成功する機会はほとんどないと思われる．そこで，ユーティリティ・ルーチンとして使用できるように，別個に論理関数簡単化のプログラムを作成した．この方が人間-機械系の観点からはすぐれている．結局，従来からの簡単化または最適設計の方法は，処理時間および設計の条件のきびしさからはとんど採用していない．

6. T-言語から B-言語への変換

変換プログラムは4パス形式であり，それぞれソース・プログラムの構文分析，シーケンスの処理，順序回路の設計およびオブジェクト・プログラムの生成を行なう (**Fig. 8**)．

6.1 ソース・プログラムの構文分析

まず，宣言部のモジュールとブロックをファイルへ移すと同時に，モジュールとブロック・テーブルを作成する．モジュールとブロック・テーブルは，プログラミング言語バリアブル・テーブルに相当する．ただし，各変数について上・下限が指定され，変数の種類が多いため1変数につき4語あて，500 変数（中型機設計で 300 足らずであった）のテーブル（計 2000 語）とし，磁気コアに常駐する．これらモジュールとブロックは変換されることなく B-言語にそのまま現われる．プロセジュア宣言があれば，ブロックの深さを1つ増し，より中のプロセジュアから処理を始める．ブロックの深さは 10 としている．関数部の関数はそのままファイルへ移す．順序部に開サブルーチンがあれば，それらをメインルーチンに組み入れファイルに移す．閉サブルーチンはそのままである．これらの処理を文法違反の監視手順とともに行なう．

6.2 シーケンスの処理

複合文，閉サブルーチン，メインルーチンの順番でその中の各文の処理を始める．各文の処理には文の文法違反の発見，レジスタ，演算器などへの接続と，ゲートの生成および命名，制御回路の生成および命名がある．ゲートの生成と命名では，まず，すでに同じゲートが作られているか否かを調べ，同じゲートが生成されているならば，新しく生成しないでそれを使う (ゲートの統合)．これらはマクロな意味での簡単化である．ゲートの統合のためにすべての代入命令をゲート・テーブルへ登録する．単純代入命令はそのまま，算術代入命令の加算は第1項と第2項をアルファベット順に並べ登録する．論理代入命令は非常に複雑な式が現われることがあり，あらゆる場合について統合することはむずかしい．そこで2項からなる論理式のみアルファベット順に並べ登録することにした。シフト代入命令は単純代入命令と同じ扱いである．1つの代入命令が現われるとゲート・テーブルを調べ，同じ代入命令があれば，すでに登録されたゲート名を使う．ゲート・テーブルは1ゲートに対し，10語必要とし，テーブルの大きさは少なくとも 1000 ゲート（計算機中のゲートの数の中型機で 500 を越えた）は必要である．したがって 10000 語を占有するためにディスクに入れた．このため処理時間がきわめて長くなり，デッバック用として他のテーブルを小さくして，ゲート・テーブルを磁気コアに入れたプロセサを作っている．T-言語表現では代入命令がもっとも多く代入命令に

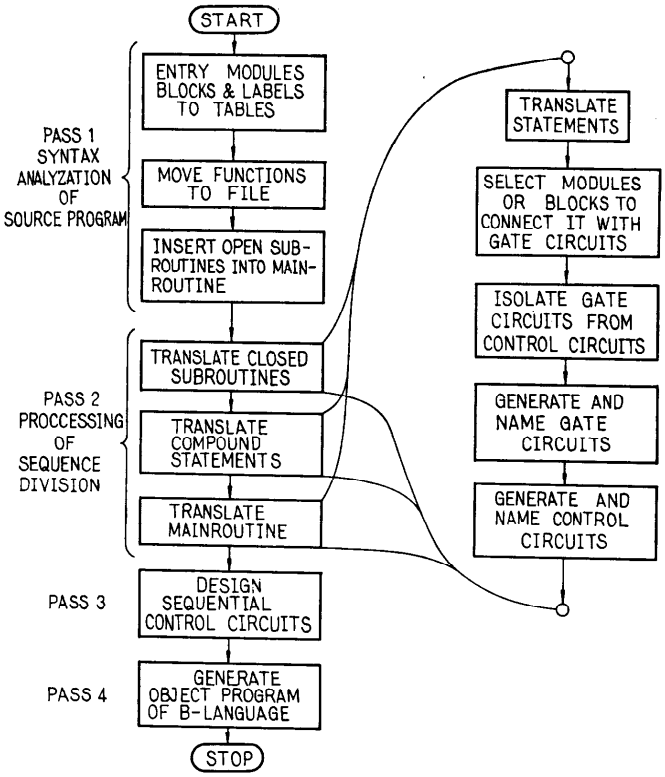


Fig. 8 Flow-chart to translate B-language from T-language

出合うとゲート・テーブルを参照するため，ゲート・テーブルを磁気コアとディスクに入れた場合の全体の処理時間を比較すると，約2倍弱の差があった．ゲート・テーブルに登録されたゲートがないならば新しく生成する．ゲートの生成は送り側と受け側のフリップ・フロップの種類により異なるため複雑である．変換上の問題点としてゲート名の命名がある．人手による設計では各ゲートに愛称を命名するが，設計言語にゲート名を記入することは非常に煩雑になり，しかも設計機械化の意図に反する．そこで，命名の規則を決定しなければならない．よく使われる方法は，送受のモジュール名の頭文字を合成する方法であるが，同名のゲートができる危険があり採用できない．筆者らの方法は，受け側のモジュール名にハイフンをつけ，その後に連続番号をつけることにした．これよりゲートの分類も可能となる．

次の問題は，制御回路の中間表現の作成である．T-言語の代入文のうち単純代入命令や論理代入命令など，1ユニット・タイムで完了する文は1状態の生成のみである，主記憶呼出しや演算命令など数ユニット・タイムを必要とする文については，順序型制御方式（または，マイクロ・プログラム方式）ではディレーを，割当型制御方式では数状態を生成する．終了信号により次のステップに進む非同期型であれば，終了信号を制御入力として待ち合わせ状態を生成する．サブルーチン・コール文も待ち合わせ状態となる．複合文では複合文の終わりが待ち合わせ状態となる．go to 文は順序型では OR のみでよい．割当型，またはマイクロ・プログラム型では次の状態を指定する．条件文は状態の生成は行なわない．順序型では制御入力を受けて制御信号の分岐回路が作る．割当型では制御入力による状態分岐を用意する．マイクロ・プログラム型では制御入力により条件マトリクスを作成する．

順序回路については，第2,3,4章に述べた．制御部の制御方式，SCB の範囲，制御レジスタの種類などの指定により，各サブ・ブロックの中間表現から入力方程式を導く．このとき，サブ・ブロック名，制御レジスタ名，デコーダ名，制御出力名などの命名はラベルに続いてハイフンをつけさらに連続番号を付して行なう．パス4である B-言語表現の段階ではオブジェ

クト・プログラムを生成する．ソース・プログラムのモジュール宣言およびブロック宣言はそのままオブジェクト・プログラムに移す．サブ・ブロックの制御回路はおのおのブロック（B-言語表現）で記述する．その後，ソース・プログラムの関数と生成されたゲート回路を関数の形で記述する．以上でT-言語はB-言語表現に変換される．変換プログラムはFORTRANとたはアッセンブラで記述し，FORTRANをおもに使い，アッセンブラはテーブル・サーチとバイト/ワード変換のみに使った．現在20ルーチンで約2800文である．ただし，制御方式のうちassignmentおよびmicroprogram方式は製作中である．処理速度は5～20文/分である．例として，命令数48個，演算レジスタ3個，インデックスレジスタ3個，入出力レジスタ4個を持つ中型機をT-言語で記述したところ約230文であり，変換時間は28分であった．

6.2 B-言語からM-言語への変換

B-言語はモジュール宣言文，ブロック宣言，関数文，ブロック・コール文により構成される．

M-言語はモジュール文のみで構成される．したがってB-言語からM-言語への変換は，それぞれの文をモジュールに変えることである．変換プログラムは3パス方式である（**Fig. 9**）．

6.2.1 ブロック宣言の変換

B-言語からM-言語への変換の最初の処理は，ブロックの壁を取り除くことである．M-言語ではすべてのモジュールは，その大小にかかわらず同列に扱われる．ブロックの中のブロックのうち，ブロック・コール文によりコールされないブロック（関数により連結されているのみ）は，blockとendを取り除くだけでよい．

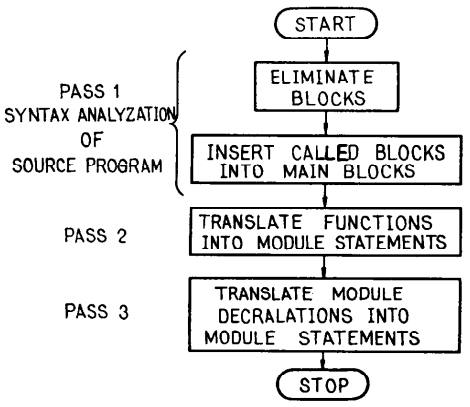


Fig. 9 Flow-chart to translate M-language to B-language.

6.2.2 ブロック・コール文の変換

ブロック・コール文は開サブルーチン・コール命令に相当するから，構文分析時にコールされたブロックの仮パラメータを実パラメータに置き換え本体に組み入れる．このとき，コールされたブロックにモジュール宣言があると，コールごとに別名のモジュールを宣言しなければならない．このモジュール名の命名の方法は，モジュール名のあとにハイフンをつけ，続いて連続番号を与えて命名する．多重構造のブロックになっている場合は，深いブロックから処理する．なお，T-言語から生成されたブロックはブロック・コール文は含まない．また，メイン・ブロックは一重のブロックをいくつか含むに過ぎない．

以上の操作により，ブロックは一重となり，モジュール宣言文と関数文の変換のみになる．

6.2.3 関数文の変換

実装設計を含めて機械化するならばfan-in. fan-out，および段数の制限，またはNAND, NORなどのモジュールによる構成など多くの問題点が続出する．しかし，筆者らの方法は論理関数をAND, OR, NOTのモジュールで表現するのみである．簡単化も試みない．その理由は，処理時間が大であること，人間-機械系による設計を目的とすることの2点である．

6.2.4 モジュール宣言文の変換

モジュール宣言により宣言されたモジュールは，標準的なインタフェースが定義されているからそれに従って変換する．同時に，あるモジュールに接続するすべての生成されたモジュールをさがし，入出力端子リストを完成する．この手順は簡単ではあるが，かなり時間がかかる．たとえば，もっとも簡単な例としてA, FともにRSフリップフロップ（すなわち，register A, F;）とし，A=Fという単純代入文について考える．まず，T-言語からB-言語への変換過程でA, F間のゲート（生成モジュール）として

A-1 AN (F-RS 3, SEQ-9 OR 0/A-RS 1)

A-2 AN (F-RS 4, SEQ-9 OR 0/A-RS 2)

が生成される．SEQは制御回路のデコーダ出力である．次に，B-言語からM-言語への変換過程でフリップ・フロップFについて，F-RS (〈リセット入力〉; 〈セット入力〉/A-1 AN 1, A-2 AN 2) なるモジュール文が生成される．RSはRSフリップ・フロップ，ANはANDである．FをF-RS (〈入出力接続リスト〉) に変換することは簡単であるが，入出力接続リ

ストを完成させることは少し時間を必要とする．B-言語から M-言語へ変換された生成モジュール文は，オブジェクト・ファイルにはいっているが，その個数は1000～10000（中型機の場合で3000あまりであった）になる．これらすべてのモジュール文の入出力接続リストをサーチしなければならないからである．別の方法として，T-言語から B-言語への変換過程で，宣言モジュールへの接続状態のファイルを作る方法があるが，ファイルが1つふえることと，最終的にはサーチが必要となるため前記の方法と大差はない．人手による設計においても，結線表として接続状態の接続する側と，接続される側との両方のデータを作成し，設計ミスを少なくするように配慮するが，この操作はかなりの時間を費やすことは明らかである．

結局，上記の手順は宣言モジュール名の各端子名と作成モジュールのオブジェクト・ファイル中の同名リストとのつき合わせである．中型機の設計例でディスクを使っても1宣言あたり 0.5 分くらいかかり，300モジュール程度あったため約3時間を必要とした．オブジェクト・ファイルが磁気コアにはいるならば飛躍的に速度は改善されるが，1モジュール文が40字（10語）程度となり，3000 モジュールあったため磁気コア（65 K であるが常駐 15 K，変換プログラム 32 K）にはいらなかった．なお，この変換プログラムは現在12ルーチン，約1500文である．処理速度は1～10文/分である．

7. むすび

計算機の設計機械化の目標は性能の仕様を与えて，費用/性能比を最小にする計算機を設計・製造することである．性能とは，ある仕事を何分でかたずけるというものであるが，現状での計算機の CAD の分野では，性能の仕様とは，方式設計または論理設計の仕様である．筆者らは，さきに提案した計算機設計言語の変換方法を検討し，変換プログラムを作ることを試みた．その結果，上記の評価関数の追求はおろか，従来の最適設計の理論もほとんど使えないことが判明した．しかし，大量の複雑なデータを取り扱うことから，設計者を解放する点では役立つと思われる．つまり，設計の事務処理としての機能は，CAD により十分代行できると思われる．今後は，実用的な最適設計の手順，実装設計も考慮した設計手順の開発を考える必要がある．これについては，かなりきびしい条件のもとでの最適設計手順を各種の条件について多数開発し，これらのうちいずれかを選択することにより，あらゆる場合の設計機械化が可能になるものと思われる．また，設計シミュレーションは不可欠であり，同じ言語で両方できることが望まれる．今後，この言語のシミュレータの製作も始める予定である．

なお，変換プログラムの作成には京都産業大学計算機科学研究所の TOSBAC-3400-40 を使用した．功力所長を初め多くの関係者に謝意を表わす．

参考文献

1) 戸田，岩下：論理関数簡単化の一方法．信学誌，Vol. 47, No. 10, pp. 1506-1511, Oct., 1964.
2) Miller R. E.; Switching theory I, II. John Wiley & Sons, 1965.
3) 元岡，他：計算機論理設計自動化の実験．信学会電子計算機研究会資料，May 1968.
4) Gerace G. B; Digital system design automation a method for designing a digital system as a sequential network system. IEEE Trans., C-17, pp. 1044-1061, Nov., 1968.
5) 萩原，黒住：計算機設計言語．情報処理，pp. 93-102, Feb., 1971.

（昭和45年10月21日受付）