SYMBOL計算機の論理構造とシステム・ソフトウェア

国分明男（電子技術総合研究所）

1. まえがき

最近，ソフトウェアのファームウェア化およびハードウェア化の研究が盛んになってきたが，“高級言語計算機”という語に代表されるように，それらの多くは言語プロセッサを対象としている。一方，今日のソフトウェアは言語プロセッサのみから成り立っているわけではない。大規模なOS，ユーティリティ，デバッギング・エイド，ライブラリィなど多種多様なものを含んでいる。

SYMBOL計算機は，計画ではそれらソフトウェアの殆んどすべてを，実際には強力な言語プロセッサとTSS用のOSをハードウェア化することによって，計算機の画期的性能向上を実証してみせるために製作された。製作プロジェクトは1960年代の中頃から1970年にかけてFairchild社のResearch Divisionで進められた。プロジェクトの中心はRex Rice，計算機の実質的な製作責任者にはIowa州立大学のA. V. Pohm教授のもとでPh. DをとってまもないW. R. Smithがなった。彼等は従来の計算機設計思想の改革を狙う野心に満ちていた。彼等の狙いは，Fairchild社という半導体集積回路メーカーにいれば当然明らかであったように，ハードウェアの価格がソフトウェアの価格に対して急激に低下していくという事実に裏打ちされていた。

この計算機は完成後，1971年の初めにIowa州立大学に移された。それから約2年間，ハードウェアの虫取り，ドキュメントの整備を行ない，磁気ドラムの増設，ソフトウェアOS（ハードウェアOSのみでもシステムは動作可能であるが，エラー・メッセージや料金計算等の細かいところまでサポートしていないので，それを増強するためのもの）の製作などをへて，性能測定用ハードウェア・モニタの追加を行なっている。Iowa州立大学では，SYMBOL計算機の評価プロジェクトが進められており，財政的にはNSF（National Science Foundation）によってサポートされている。プロジェクトは1971年からの5ヶ年計画になっているから，今年で終了の予定である。筆者はIowa州立大学でのプロジェクトに1974年の9月まで1年間滞在する機会を持った。以下では従来，あまり明らかでないこの計算機の論理構造とシステム・ソフトウェアを中心に報告したい。

2. SYMBOL計算機の構成

SYMBOL計算機は，単一バスを介して8台の専用プロセッサが接続されたマルチ・プロセッサ構成になっている。（ハードウェアのデバッグの際に用いるシステム診断・テスト・プロセッサまで含めると9台だが，このプロセッサは正常時には使用されない。）各プロセッサは特定の機能のみを持った専用プロセッサであるから，他のプロセッサの代りはできない。ユニークな点はメモリがメモリ・プロセッサの下に接続されていて，バスに直接出ないことである。したがって，メモリを必要とするプロセッサは，メモリ・プロセッサにメモリ・サービスを依頼する形で処理を進める。構成を図1に示そう。

- システム・スーパバイザ（SS）はシステム内の資源割り当て，プロセス管理を行なうプロセッサであり，最大32のプロセスまで制御できる。
- トランスレータ（TR）はALGOLに似たSYMBOLという言語（名前が一切不要で，実行時の動的配列などに特徴がある言語）のハードウェア・コンパイラであり，ソース・プログラムを逆ポーリッシュ形のオブジェクト・コードに変換する。
- セントラル・プロセッサ（CP）は，機械語よりはむしろ高級言語に近いレベルのオブジェクト・コードを実行する。
- I/Oプロセッサ（IO）は，I/Oバッファとバーチャル・メモリ空間の間でデータの移動を行なう。
- チャネル・コントローラ（CC）は，端末装置とI/Oバッファの間でデータの移動を行なう。
- メモリ・コントローラ（MC）は，バーチャル・メモリの制御，各ページ内のグループと称するブロックの動的割り付け，動的リンキング，メモリへのアクセスの制御を行なう。
- ドラム・コントローラ（DC）は，ページ・フォールトの処理の際，コア・メモリとドラム間のデータ転送の制御を行なう。
- メモリ・リクレイマー（MR）は，不用になったグループを回収して再使用が可能なようにする。

SYMBOL計算機は製作開始の時期から見て，今日ではハードウェア的に古い計算機となったが，アーキテクチャは多くの観点から新鮮さを失なっていない。しかし，製作時にROMが価格的に適当でなかったことなどもあって，マイクロプログラムを採用せず，ワイヤド論理を採用したため，構造が固定的になり，柔構造を求める最近の傾向に反する結果となってしまった。

使用したICはFairchild社のCTμL type I familyで，クロックは約200nS程度である。コア・メモリはサイクル・タイム2.5μSのものを用いている。容量は8K語（1語は64 bits）で，256語ずつのページ構成になっている。先頭のページはシステム・スーパバイザ(SS)のテーブル領域にあてられ，続く3ページは各端末に各ページあたり8語ずつ割り当てられた制御語の記憶領域である。（すなわち，各端末あたり24語，32端末分が用意されている。）残りの28ページはバーチャル・メモリのコア・フレームとなっている。ページの入れかえ

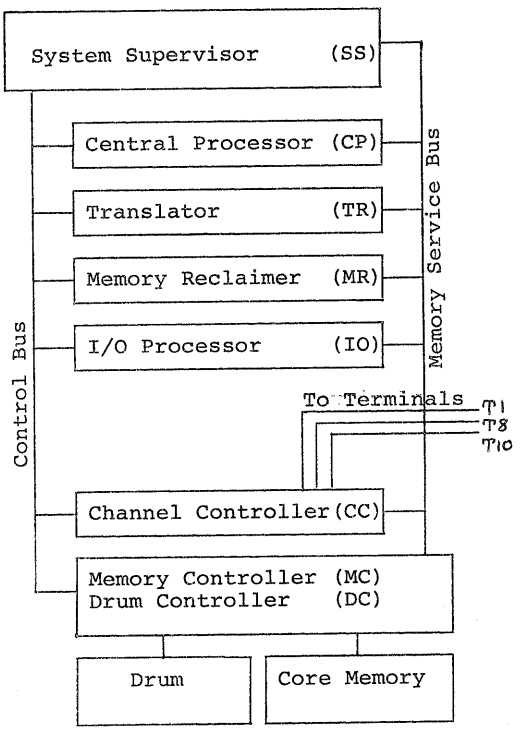


図1 SYMBOL計算機の構成

は各ページに対応する連想レジスタを用いて行なわれ，ドラムがバッキング・メモリとなる。ドラムには磁気ヘッドが1,024個ついており，平均アクセス・タイムが8msである。1トラックに4ページ記憶されるので，ドラム全体では4,096ページになる。物理的な端末は現在のところ3端末のみである。端末P1は“SYMBOL Terminal”とも呼ばれており，通常のteletypeに加えて，各種の状態インディケータとファンクション・キーが用意され，多くの場合，コマンドはファンクション・キーのどれかを押すだけでよい。また，この端末にはカードリーダとラインプリンタも付属している。P1中のどの装置から出し入れするかはマニュアルでもセットできるし，プログラムでも指定できる。端末P8はキャラクタ・ディスプレイ端末，P10は音響カップラーを介して接続されているTTYである。P8やP10にはSYMBOL Terminalのようなインディケータやキーがないので，それらをソフトウェアOSによって補っている。

3. メモリの論理構造

アーキテクチャの観点から特色があると見られるのは，メモリ管理の徹底的な自動化であろう。メモリ・コントローラ（MC）はドラム・コントローラ（DC）とメモリ・リクレイマー（MR）の助けを借りながら，チャネル・コントローラ以外のプロセッサにメモリ・サービスを提供する。メモリ・サービスは次の2つに大別される。(1)ページングの管理，(2)記憶領域のアロケーション，アクセス，デ・アロケーション（割り当ての解除），再生。

コア・メモリとドラム間のデータ転送は，システム・スーパバイザ（SS）の指導の下にドラム・コントローラ（DC）が行なう。1語64 bits に通常は2命令収容され，1命令32 bits 中24 bits がバーチャル・メモリのアドレスに当てられているが，そのうち16 bits はページ指定に用いられる。

メモリ命令

SYMBOLでは，ユーザが高級言語上で使うことができる便利なメモリとコア・メモリのように単純な動作しかしないメモリの間のギャップを，通常の計算機のようにコンパイル時にソフトウェアで橋渡しするのではなく，実行時にハードウェアで縮める。すなわち，セントラル・プロセッサ（CP）やI/Oプロセッサ（IO）は，プログラムの実行時にコア・メモリと直接やりとりするのではなく，メモリ・コントローラ（MC）からメモリ・サービスを受ける形で処理が進行する。この方式だと，単に読み書きを行なうのよりも見かけ上高度な機能をメモリに持たせることができる。図2に，SYMBOLで用意されているメモリ命令の一覧を示そう。

バーチャル・メモリ空間はページに分割されているが，各ページは更にグループと呼ばれる単位に分割されている。グループは物理的に連続している8語からなり，グループ間は任意長のストリングを形作るために必要に応じて論理的につながれる。（すなわち，ストリングはメモリ上で物理的に連続であるとは限らない。）メモリ命令によって，このストリング上での読み書き，両方向へのたどり，ストリングの発生・延長・削除ができる。ストリングは一次元であるが，ストリングにストリングをつなぐことによって，N次元にすることができる。

メモリ命令を使用するプロセスの具体例として，次のような場合を考えよう。プログラム中で数式を処理するために，セントラル・プロセッサ（CP）がオペランド・スタックの必要なことを検出したとしよう。CPはMCに“グループを

AG	Assign Group
IG	Insert Group
FF	Fetch and Follow
FR	Reverse Follow and Fetch
FL	Follow and Fetch
FD	Fetch Direct
SA	Store and Assign
SO	Store Only
SI	Store and Insert
SD	Store Direct
DE	Delete to End of String
DS	Delete String
DL	Delete Page List
RG	Reclaim Group

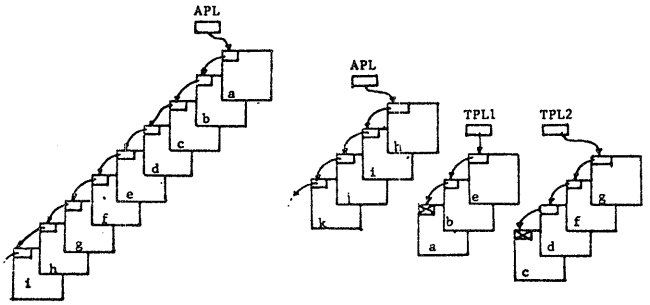
図2　メモリ命令の一覧

割り当てよ”命令（Assign Group）を送る。MCは未使用のメモリ領域中から，ある決まったアルゴリズムにもとづいて1グループを選び，そのグループの先頭アドレスをCPに返す。CPはそのアドレスを出発点として，データをグループ内の連続している各語に，書き込み・割り当て命令（Store and Assign）をくり返して書き込んでいく。この場合，MCは常に次の論理アドレスをCPに返すことになっているから，8語以上になると自動的に新しいグループが割り当てられ，それ以前のグループにリンクされる。したがって，論理的にいくらでもスタックを伸ばすことが可能である。（上述のストリングは，この場合，スタックとして使われている。）このことは，CPをアドレス計算の仕事から解放した。CPに必要なことは，返されてきた次のアドレスを次回のStore and Assign命令を行なうまで，保存しておくことだけである。このようにして作られたスタックからデータを読み出すには，逆方向スキャン用に逆たどり・読み出し命令（Reverse Follow and Fetch）が，正方向スキャン用に読み出し・たどり命令（Fetch and Follow）が用意されている。スタックが必要でなくなったとき，Assign Groupの段階でCPに与えられたアドレス（スタックの頂のアドレス）を，ストリング削除命令（Delete String）に添えてMCに送れば，その記憶ストリングはメモリ・リクレイマー（MR）によって再生された後，未使用のメモリとなる。

上に述べたスタックだけでなく，ソース・プログラム，オブジェクト・コード，ネーム・テーブル，スカラーおよびベクトル値，システム・スーパバイザ・プログラム，システム・ライブラリィなどの場合にも，同様にしてストリングに記憶される。上例のスタックを説明する際，CPからMCに命令と共に送るアドレス（前回にMCから得たアドレス）はコアにあるページを指していると仮定したが，コアにないときはMCはページ・アウト信号を返す。CPはそれを受けて各種レジスタの内容を退避させ，システム・スーパバイザ（SS）に次の仕事を求める。SSの指導の下に，ドラム・コントローラ（DC）がページをコアに読み込むと，SSはCPに中断していた仕事を割り当てる。このように，CPはページングの仕事からも解放されている。

メモリ・リスト

システムの初期状態ではメモリはすべて未使用だから，図3に示すように，バーチャル・メモリ中のすべてのページがリンクされて，使用可能ページ・リスト（Available Page List，APL）が作られている。ある端末（TSS向きに設計されているため，端末とユーザは同義語である。）からの仕事を処理するため，メモリ・コントローラ（MC）はその端末専用のページ・リスト（Terminal Page Lists，TPL）を3種類まで作ってやる。すなわち，MCはAPLから先頭ページを取り，TPLを作ってやる。TPLに与えられたページが使いつくされたら，MCは再びAPLの先頭からページを取り，TPLの先頭に移す。こ



a. Before allocation. b. After allocation of pages a-g.

図3 ページのアロケーション

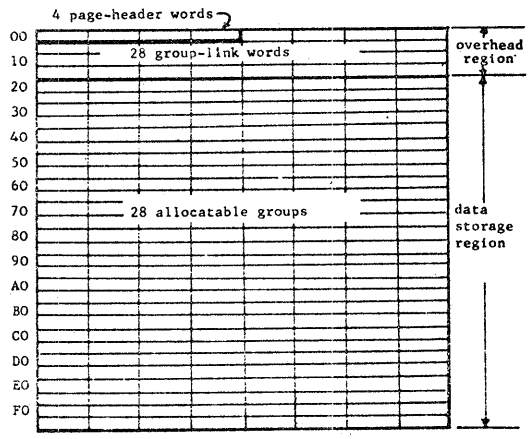


図4 ページのフォーマット

のようにページをTPLに移し続けていくと，TPLが増大するにつれて，APLが小さくなっていく。不要になったページをAPLに戻せるように再生するのはメモリ・リクレイマー（MR）の仕事であるが，ページ・レベルでは1つのページ・リスト全体が不要になった後でないと，解除・再生されない。そこで，ページ内での動的メモリ管理を次に見てみよう。

ページ内でのメモリ割り当て方式を説明するために，図4にページのフォーマットを示す。256語のページ中，先頭4語はページ・ヘッダー語（page header）と呼ばれ，ページ間のリンキングやページのアロケーションなどに使われる。続く28語は，そのページ内に存在する各グループに1語ずつ対応したグループ・リンキング用のポインタ語で，正方向，逆方向共にリンクできる。残りはすべて記憶領域であり，8語単位のグループが28ある。

あるプロセッサがAssign Group命令をMCに送ったとしよう。その結果，MCがAPLから新しいページをTPLにかえた場合を考える。MCはグループを要求しているプロセッサに，そのページの最初のグループの先頭アドレスを返す。そのTPLにおける次回以後のグループ割り当ては，ページ・ヘッダー語中にある割り当てカウンタを用いて，物理アドレスから見て次のグループを割り当てる方式で行なわれる。

使用可能グループ・リスト（Available Group List，AGL）を用いて，グループ割り当てを行なう機構も用意されている。このリストの開始ポインタはページ・ヘッダー語中にあり，一度割り当てられた後，解除・再生されたグループをたどれるようになっている。

新しいグループが必要なとき，MCはまず割り当てカウンタを調べて，TPLの先頭にあるページ内のグループがすべて割り当てつくされたかどうかを知る。もし，カウンタによって割り当てつくされていれば，次にそのページのAGLを調べる。（動的メモリ管理を行なっているので，そのページ内には一度割り当てられた後，解除・再生されたグループがあるかもしれないからである。）もし，AGL中にグループがあれば，MCは先頭グループを取り，メモリを要求したプロセッサにそのグループの先頭アドレスを送り，必要があれば，それ以前に割り

当てたグループにリンクしてやる。もし，AGLにもグループがなければ，APLから新しいページをもらう前に，TPL毎に1つ作られる使用可能グループを持つページ・リスト（Terminal Available Space List，TASL）を見る。TASL中にページがあれば，MCはこのページ（いくつかあれば先頭ページ）のAGLを見て，割り当てを実行する。もし，TASLに何もなければ，MCはAPLから新しいページをもらうことになる。

メモリを必要とするとき，プロセッサはAssign Group命令などにアドレスをつけてMCに送ることもできる。この場合，MCは指定されたアドレスのページを最初に見て，上述の割り当てアルゴリズムを実行する。

あるプロセッサが，MCにストリングの部分的または全面的削除（すなわち，割り当ての解除）を要求したとしよう。解除されたグループは，即座にメモリ・リクレイマー（MR）によって処理されるのではなく，TPL毎に1つ作られるガーベージ・リスト（Garbage List）に加えられて，システムのバック・グランドの仕事としてMRが処理するまで待つ。MRによって再生された後，それらのグループは各ページのAGLに加えられ，再び使用される。説明してきたメモリ・リストの一覧を図5に示す。

Name	Where Used	Type of Linking	Addition To List	Removal From List
Available Page List (APL)	One Per System	Forward Linked Tree	Top	Top
Terminal Page Lists (TPL)	One To Three Per Terminal	Forward Linking	Top Or Middle	Complete Transfer
Terminal Available Space List (TASL)	One Per Page List	Forward Linking	Top	Top
Available Group List (AGL)	One Per Page	Forward Linking And/Or Count	Top	Top Or Count
Strings	One Or More Per Page List	Forward And Backward	End Or Middle	Completion Or Partial Transfer
Garbage List (GL)	One Per Page List	Forward Linked Tree	Top	Top

図5　ページ・リストの一覧

4. オペレーティング・システム

OSの仕事として次のものがある。1) プロセス管理，2) 各種サービス（ハードウェアによって行なわれるサービスに加えて，ソフトウェアによって行なわれる補助的サービス），3) 資源割り当て。

SYMBOLのOSの大部分は，システム・スーパバイザ（SS）中にハードウェア化されている。ハードウェアOSによって行なわれる内容は，主として使用頻度が高いか，短いレスポンス・タイムを必要とするものから成り立っている。OSのソフトウェア部分（すなわち，ソフトウェアOS）はSYMBOL語で書かれ，ユーザ・プログラムを実行するのと同じセントラル・プロセッサ（CP）で実行される。

システム・スーパバイザ

SSの仕事は次の通りである。1) プロセスの変化に応じて，関連する端末の制御語の変更，2) ソフトウェアによってセットされたパラメータにもとづいて資源割り当ての決定，3) ハードウェアで用意されていない機能の場合，ソフトウェアへ仕事を渡すこと。

SSは図6に示されるように，サイクリック・プロセスと見ることができる。他のプロセッサからのサービス・リクエスト（SR）によって刺激されるまでRest Stateにとどまっているが，SRがあると，そのリクエストに適切なサービス・ルーチンを実行する。約70種のSRがSSによって処理される。たとえばSRとして，

1) SYMBOL Terminalで，あるファンクション・キーが押されたことをチャネル・コントローラ（CC）が報告した，
2) プロセッサのうちの1つが，現在の仕事を正常に終えたことを報告した，
3) インプット・バッファが一杯になったことを，CCが報告した，
4) ドラムの各トラックに4ページずつ記憶されているが，そのページの境界がヘッドの下を通りすぎたことを，ドラム・コントローラ（DC）が報告した，
5) プロセッサのうちの1つが，ページ・フォールトに直面したことを報告したなどがあげられる。

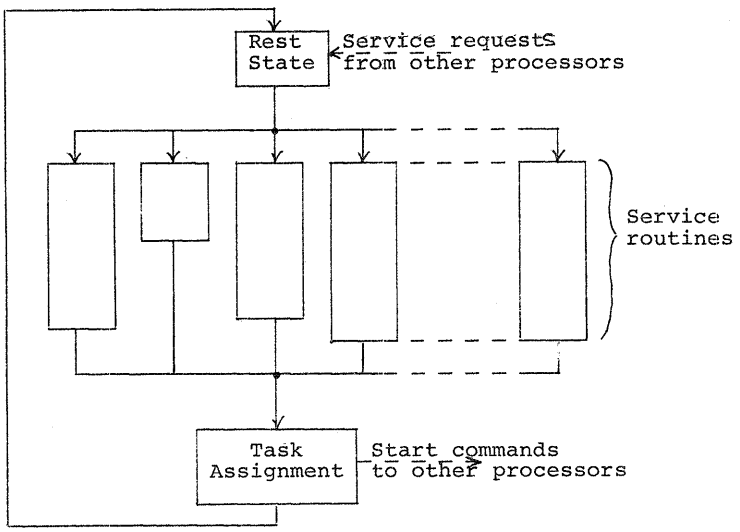


図6 システム・スーパバイザ中の制御の流れ

SSはCCを除いた各プロセッサのために，そのプロセッサによるサービスを必要とする端末のキューを持っている。タスク割り当ての際には，SSはidle状態にあるプロセッサのキューをスキャンして，割り当てられるべき端末を発見すると，そのプロセッサにその端末をサービスするようコマンドを出す。各サービス・ルーチンは十分速く実行されるの

で，実行中の割り込みやreentranceは考えられていない。サービス・ルーチンが速く実行されない例外的な場合もある。たとえば，SSがあるプロセッサに仕事を中止するようコマンドを出した後，そのレスポンス待ちの場合やSSがページ・フォールトに直面した場合がある。そのような場合には，SSはサービスしている端末の制御語にコンティニュエーション・コードとそのルーチン番号を書きこんで，直ちにRest Stateに戻る。遅れを生じる原因が解消されたら，SSは前の状態に復帰してSRを処理する。

サービス・ルーチンは通常，次のことを行なう。

1) サービスしていた端末の制御語中のプロセッサ状態に関する部分をイニシャライズする。
2) ページ・フォールトになった場合には，キュー中の該当する端末にブロックされていることを記録する。ページがコアに読みこまれて，ページング・タスクが終了したら，もうブロックされていないことを記録する。
3) バッファが一杯になった場合あるいは空になった場合には，I/Oバッファのポインタをスワップする。

また，キューからある端末を削除して別のキューに加えることなども行なう。

上に述べたサービス・ルーチンによってシステムはソフトウェアなしで動くことができるが，ハードウェアは，プログラム・エラー診断メッセージ，料金計算，普通の端末の場合のログ・イン／ログ・アウト，端末間通信，またはファイル・システムなどを処理しない。したがって，ソフトウェアでこれらが補われる。ソフトウェアで処理されるサービス・リクエスト（SR）として

1) プロセッサがプログラム・エラーを検出した（すなわち，シンタックス・エラーまたは実行エラーを検出した），
2) 与えられた処理時間またはメモリ・スペースを越えた，
3) ドラム読み出し／検査エラーを検出したなどがあげられる。

ハードウェアで処理される場合と異なって，SSのレスポンスはソフトウェアへ仕事を渡すことであり，すべて同一である。

システム・ソフトウェア

サービス・リクエスト（SR）には特定の端末によって起こるものと端末によらないもの（たとえば，リアル・タイム・クロック信号，制御パネル・スイッチ信号等のようなシステムSR）がある。ソフトウェアで処理されるシステムSRは端末TOの受け持ちと見なされている。端末TOは物理的な入出力部は何も持たない論理的な端末で，システム・ソフトウェアを実行するために用意されている。システムSRが生じたとき，SSは対応するSR番号をTOの制御語中に書き込んで，セントラル・プロセッサ（CP）のキューにTOを加える。TOに順番がまわってくると，SSは制御語中のSR番号を見て通常のCP割り当てルーチンからとび出る。そして，ソフトウェア・エントリ・ポイント・テーブル（システムの初期的に，ソフトウェア・ローダーによってロードされる。）から得られたソフトウェア・ルーチン・アドレスを制御語の実行アドレス・フィールドに入れる。更に，SR番号をクリアして，制御語中のTOアクティブ・フラグを立て，最後にSSはCPに通常のスタート・コマンドを出す。この時点からそのルーチンの終了まで，TOは他のすべての端末と同様にCPサービスをキューでとりあうことになる。TOアクティブ・フラグが立っているから，別のシステムSRが生じたとき，SSは終了までそれを無視する。システムSRはサービスされ

るまでリクエストをくり返すように作られているから，無視しても問題は生じない。

システムSRでないSR（すなわち，特定の端末に原因があるSR）を処理するのに，端末TOは適当でない。理由は，サービスされる端末は多数であるのに，TOはreentrantな構造になっていないし，サービスする内容にしばしばインタラクティブI/Oを含むから，処理時間が長い場合が多いであろう。長時間，ある端末のサービスのためにTOを占有されると，他の多くの端末がめいわくする。したがって，SYMBOLのOSではシステムSRでないSRを処理するために，TOを除いた各端末に専用のモニターを作成し，その上をソフトウェア・サービス・ルーチンが走る。すなわち，ハードウェア的には端末数は固定されているから，各端末のハードウェア上にソフトウェアでモニターという仮想の端末とユーザという仮想の端末を作り，サービス・ルーチンはモニター上を走る。端末の制御語を本来ならば各仮想端末にハードウェア的に用意できると良いのだが，各物理端末あたり1ブロックぶっしか用意されていないので，モニターとユーザの間でそれをスイッチして使わなければならない。しかし，スイッチする時点で問題が生じるので，現在はバーチャル・メモリ空間にモニターが補助制御語ブロックを2ブロック確保する。そして，一方をユーザ制御語用に，他方をモニター制御語用に使う。モニターとユーザ間にスイッチング・ハードウェアがないため，物理端末の制御語ブロックと補助ブロック間のデータ転送はTOが行なう。システムSRと同様に，SSはSRを発生した端末の制御語中にSR番号を書き込んだ後，その端末をCPキューにかえる。キュー中ほどの端末に順番がまわってくると，SSはSR番号に対応したサービス・ルーチン・アドレスをソフトウェア・エントリ・ポイント・テーブルから得て，TOの制御語中に書き込む。また，SSはSRを発生した端末の制御語中にTOアクティブ・フラグを立てる。その結果，その端末をサービスするためにCPスタート・コマンドが出された際，コマンド中の端末番号がOに置きかえられる。したがって，実際にサービスされるのはTOとなる。

SYMBOL語

OSのソフトウェア部分がSYMBOL語で書かれることは既に述べた。以下では，システム・ソフトウェアのために用意されているステートメントを紹介する。通常の計算機では実行時にスーパバイザ・モードとユーザ・モードに分けられていて，ある命令をユーザ・モードで実行すると割り込みがかかるようになっている場合が多いが，SYMBOLでは実行時には1つのモードのみであり，トランスレーション時に特権プログラム（システム・プログラム）と非特権プログラム（ユーザ・プログラム）に分けられる。ユーザ・プログラムではメモリに直接アクセスすることはできないが，システム・プログラムでは次章で述べたメモリ命令が使える外，“SYSTEM”と“TRAP”が使用できる。

SYSTEMステートメントによって，CPはidle状態になり，SSがジョブ・コントロール・プログラムを実行する。ジョブ・コントロール・プログラムは端末制御語中のフラグ等の変更，キューの内容の変更などを行なう。

TRAPステートメントがあると，トランスレーション中にTRをストップしてシステム・ソフトウェアへ仕事を渡す。SYMBOL語のループ文関係が計画では含まれていたが，実際にはインプリメントされなかったため，このステートメントを使ってトランスレータ（TR）をソフトで補ってやることができる。

SYMBOL語は汎用の高級言語であるから，それで書かれたシステム・ソフトウェアは当然読みやすい。また，コンパイルはハードウェアで行なうから，コンパイル時間を気にする必要がないほど十分に高速である。そのため，システム・ソフトウェアはディスク上で常にソース・プログラムの形で存在することができる。サービスのために，システム・ソフトウェアの一部分が必要になったときは毎回その部分をコンパイルして使用すればよいからである。

言語として非常に強力であるため，書きやすいのも当然である。たとえば，宣言が全く必要ないこと，配列は何次元でも動的に使えること，identifier に長さの制限がないこと，Format に加えて Mask 機能が強力であること，ALGOL 形のブロック構造の言語であること，ON割り込みの機能があることなど。また，システム・ソフトウェアには直接関係がないかもしれないが，演算精度をプログラムで指定することによって 99 桁まで自由に選べるのは特筆に値する。各種関数のサブルーチン類はこのようなハードウェアに対応して，演算精度を自由に選べるように作られている。

5. むすび

SYMBOL計算機の従来あまりふれられていない面を中心に述べた。ソフトウェアの大部分をハードウェア化した計算機として，SYMBOLは普通の計算機と比べると，非常にユニークである。このようにユニークなアプローチが，ハードウェア技術の進歩と共に広く受け入れられるかどうかは，コストのことまで含めて考えなければ分からないが，より高級な言語，より使いやすい計算機，しかも高速処理を求める限り，SYMBOLのインパクトは無視できないであろう。

最後に，筆者のSYMBOL評価プロジェクト滞在をアレンジして下さった石井 治部長，A. V. Pohm教授にお礼を申し上げます。また，米国出張を可能にして下さった黒川一夫部長，新田松雄室長にお礼を申し上げます。

参考文献

1. R. Rice and W. R. Smith : SYMBOL - A Major Departure from Classic Software Dominated Computing Systems, Proc. SJCC, Vol. 38 pp. 575 ~ 587, 1971.
2. W. R. Smith et al : SYMBOL - A Large Experimental System Exploring Major Hardware Replacement of Software, Proc. SJCC, Vol. 38, pp. 601 ~ 616, 1971.
3. G. D. Cheeley and W. R. Smith : The Hardware Implemented High Level Language for SYMBOL, Proc. SJCC, Vol. 38, pp. 563 ~ 573, 1971.
4. H. Richards, Jr. : SYMBOL 2R Programming Language Reference Manual, Cyclone Computer Laboratory, Iowa State University, Ames, Iowa, 1971.
5. R. J. Zingg and H. Richards, Jr. : SYMBOL : A System Tailored to the Structure of Data, Proc. NEC, Vol. 27, pp. 306 ~ 311, 1972.
6. W. R. Smith : System Supervision Algorithms for the SYMBOL Computer, Digest of Papers COMPCON 72, pp. 21 ~ 25, 1972.
7. J. W. Anderberg and C. L. Smith : High - Level Language Translation in SYMBOL 2R, Proc. High Level Language Computer Architecture Symposium, pp. 11 ~ 19, 1973. このシンポジウムにあと3件ある。