ＡＣＥ１．１システムとその基本ＯＳ

ACE 1.1 SYSTEM AND A DESIGN CONCEPT OF IT'S BASIC OS

藤井捐介　飯塚肇　古谷立美

Kensuke FUJII　Hajime IIZUKA　Tatumi FURUYA

電子技術総合研究所　電子計算機部

Computer Division , Electrotechnical Laboratory

[1] はしがき

ＡＣＥ(Adaptive Computer Experiment)システムは電子技術総合研究所で開発中のモジュール構造複合計算機の実験システムである。その設計思想，並びにアーキテクチャの概要については既に報告したが，あらたに，３プロセッサモジュールのＡＣＥ１．１システムを完成したので，本稿ではその概要を述べた後，現在開発中の基本ＯＳについて報告する。

[2] ＡＣＥ1.1 システム

ＡＣＥ１．１システムは図１のような構成を持っている。図中の点線内がＡＣＥ０．１からＡＣＥ１．１になって追加された部分である。また点線内は入出力制御システムで，市販のミニコンピュータシステムを用いている。以下，ＡＣＥ１．１の主な特徴，ＡＣＥ０．１に対する改良点等について述べるが，ＡＣＥシステムの詳細については文献[1]～[4]を参照されたい。

(1) プロセッサモジュール

新しく追加したプロセッサモジュール(ＰＲ－２，３)は全く同一の仕様を持っているが，ＰＲ－１との互換性はない。その理由はＰＲ－２，３のマイクロ処理ユニットとして，パターンプロジェクトで開発中のＬＳＩマイクロ処理ユニットＰＵＬＣＥと同一のアーキテクチャを採用したためである。これによって，ソフトウエア開発は当面若干の不便を生じるが，近い将来のＰＵＬＣＥ部分のＬＳＩへの交換，およびプロセッサモジュール数の拡大に備えるためには，ファームウエア量の少ない現在のうちに移行する方が良いと判断したものである。

また，ＰＲ－２，３では実装形態の大幅な変更を行なった。すなわち，ＰＲ－１では片面に実装および配線をするユニバーサル基盤を用いたため，最短配線を行ないにくく，電気的特性等に若干問題があったので，今回は表面実装，裏面配線で，全ソケット方式の基盤を採用した。図２は，その写真であるが，各ＰＲはこの基盤９枚で構成されている。

(2) Ｃ－バスコントローラ

Ｃ－バスコントローラ(ＣＣ－２)は機能的には#１と同一であるが，実装形態は新ＰＲと同一にして，論理の整理を行なった。

(3) 相互排斥制御モジュール

ＰＲ間の同期制御ならびに相互排斥を制御するために開発した特殊モジュールで，[4]章で詳しく述べる。

[3] 主なハードウエアサポート

上述したように，ＡＣＥシステムの詳細については，既に発表した文献を参照してもらう必要があるが，以下で述べる相互排斥制御モジュールおよび基本ＯＳの基本となっているプロセッサモジュール(ＰＲ－２，３)の構成，アドレス空間とプロセッサモジュール間の通信に関する概念を簡単に述べる。

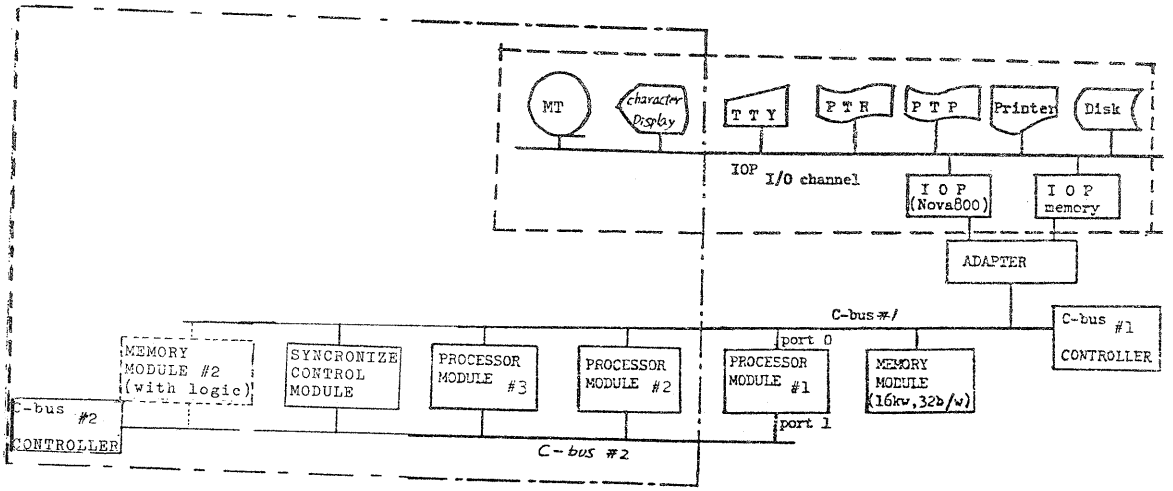


図 1 - (a) ACE システム 構成図

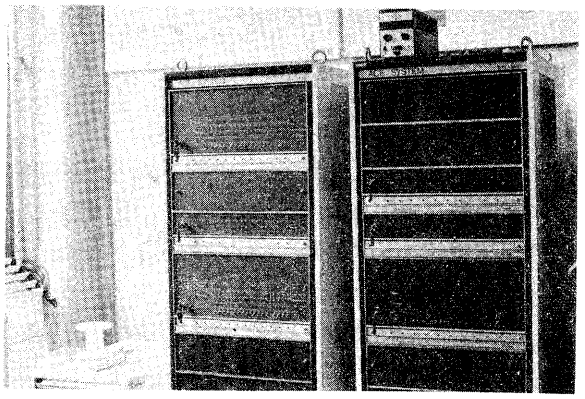


図1 - (b) ACE システム 概観

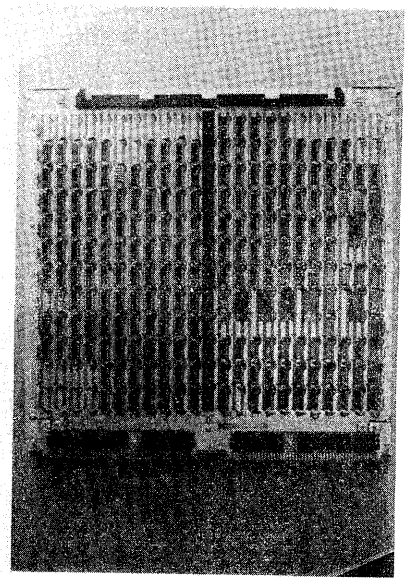


図2 - (a) PR-2, 3の基盤 (表面)

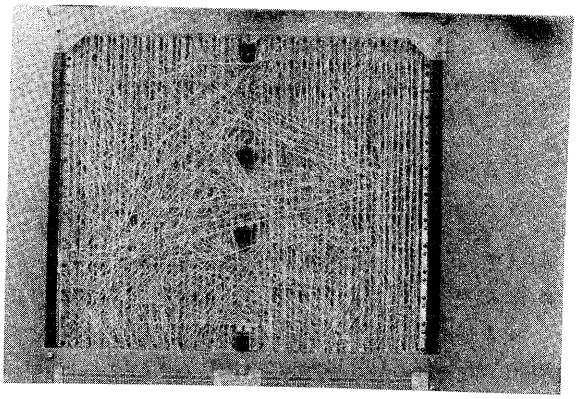


図2 - (b) PR-2,3 の基盤 (裏面)

（1） プロセッサモジュールの構成

現在，ACEシステムではNOVAを含めて3種類のプロセッサモジュールが存在している。図3にPR-2/3プロセッサモジュールの構成を示す。図3から明らかなようにPR-2,3はPULCE（Pips Universal Computing Element），SCU（Sequence Control Unit），μC（μ- Controller）の3ユニットより成っている。これ等のユニット内のレジスタについては[5]章の（6）で少し触れる。

（2） アドレス空間

ACEシステムでは，1つのバス上に2^{25}バイトのGA（Grobal Address）の空間を有しており，プロセッサモジュール内のLA（Local Address）はセグメント単位でGA上のアドレス空間にハードウエアでマッピングされる。マッピングは，μC内のSD（Segment Descriptor）とアドレスコンバータによって行なわれる。したがって，SDを適当にセットしてやることによって 各LAがGAに変換された時，をを分離したり，共有したりすることができる。

（3） プロセッサモジュール間の通信

各プロセッサモジュールは，128B/blockのLMB（Local Memory Block）を8ブロック有しており，このLMを他のプロセッサモジュールからアクセスすることが可能である。外部のアクセスは，プロセッサモジュール内のポートごとに存在する2つのレジスタ（FRA/SRA：First/Second Recognized Reg.）にGAをセットすることによって可能となる。つまり，細かいアクセスに関する指定を除けば

(a) アクセスする側：SDにGAをセットする。

(b) アクセスされる側：FRA/SRAにGAをセットする。

ことによってプロセッサモジュール間の通信が可能になる。したがって，アクセスされる側が全て同一のGAをFRA/SRAにセットしておけばデータのブロードキャストが可能であるし，さらに細かいアクセスに関する指定を行なえば，アクセスされる側に割込を発生させることも可能である。

以下で述べる相互排斥制御モジュールと基本OSは，これ等のハードウエアサポートを有効に用いて機能拡張を達成している。

[4] 相互排斥制御モジュール

相互排斥制御モジュール（MECM：Mutual Exclusion Control Module）は，共有資源の管理をサポートするために製作したもので，バス上に1つの独立したモジュールとして結合され，バス上の他のモジュールは，これに対してリード/ライトといった一般のデータ転送命令を発することにより初期値1のセマホアを実現出来る。このモジュールを使って相互排斥を実現する場合は，同期用の特殊な命令を必要としないことと，同期動作が他のモジュールと独立に行なえる利点があり，システム構成が変わり得るACEシステムには特に便利である。

（1） MECMの構成

一般に相互排斥を達成するためには，共有資源に対応させてロックバイトやセマホアと呼ばれる物理的実体を使うことがあるが，MECMでは各資源に対応させるものとしてロックセル（LC）と呼ぶセルを用意する。図4がMECMの構成であり，LCの集合であるメモリ，制御回路，加算回路より成る。LCは，共有資源の使用状態を示すT&S（Test and Set）と待ち行列を管理するためのカウンタ（CTR）より成る。各セルにはアドレスが与えられていて，相互排斥を行なうモジュールは，これをアドレスして次に示す同期用命令を発する。同期用の命令とそれに対するMECMの動作は次のとうりである。

(a) リード1：（LCへのリード命令）
アドレスされたLCの内容がC-バスに読出され，その後MECMではLCのCTRに1を加え，T&Sをセットする。

(b) リード2：（LCへのリード命令）
アドレスされたLCの内容がC-バスに読出され，その後T&Sがセットされる。

(c) ライト：（LCへのライト命令）
アドレスされたLCのT&Sがリセットされる。ここで(a)と(b)の区別にはアドレスの1ビットが使用される。

（２） 相互排斥の実現

ＭＥＣＭは，ＡＣＥのブロードキャスト通信方式を利用して，初期値１のセマホアを実現する。ＡＣＥのブロードキャスト通信方式とは，[3]章 (3) でも少し触れたが，データを送るプロセッサモジュールが，データ転送に先立ち，アドレスをバスに送り出し，バス上の他のモジュールはそれを見て自分がデータを受信するか否かをバスコントローラに知らせる。この様にしてデータ転送に参加するモジュールが決った後，データは通信に参加すると答えた全モジュールに送られるというものである。

図５は，相互排斥の実現法を示したものである。共有資源を使用するモジュールは，共有資源に対応したＬＣにリード１命令を出し，ＬＣの内容を読み込む。そして，ＴＢＳがセットされていれば，共有資源が空き状態と解釈し共有資源を使用する。一方，ＴＢＳがセットされていたら，自分の順番が廻って来るまで待つ。自分の順番が廻って来たことは，共有資源の使用を終えたモジュールが出すライト命令でわかる。ライト命令を出すモジュールは，自分がリード１命令を出した時に読み込んだＣＴＲの値に１を加えて送り出す。待っているモジュールは，これを見て自分がリード１で読み込んだＣＴＲの値と一致するかを調べ，一致すれば自分の番と判断し，リード２を出して共有資源を使用する。

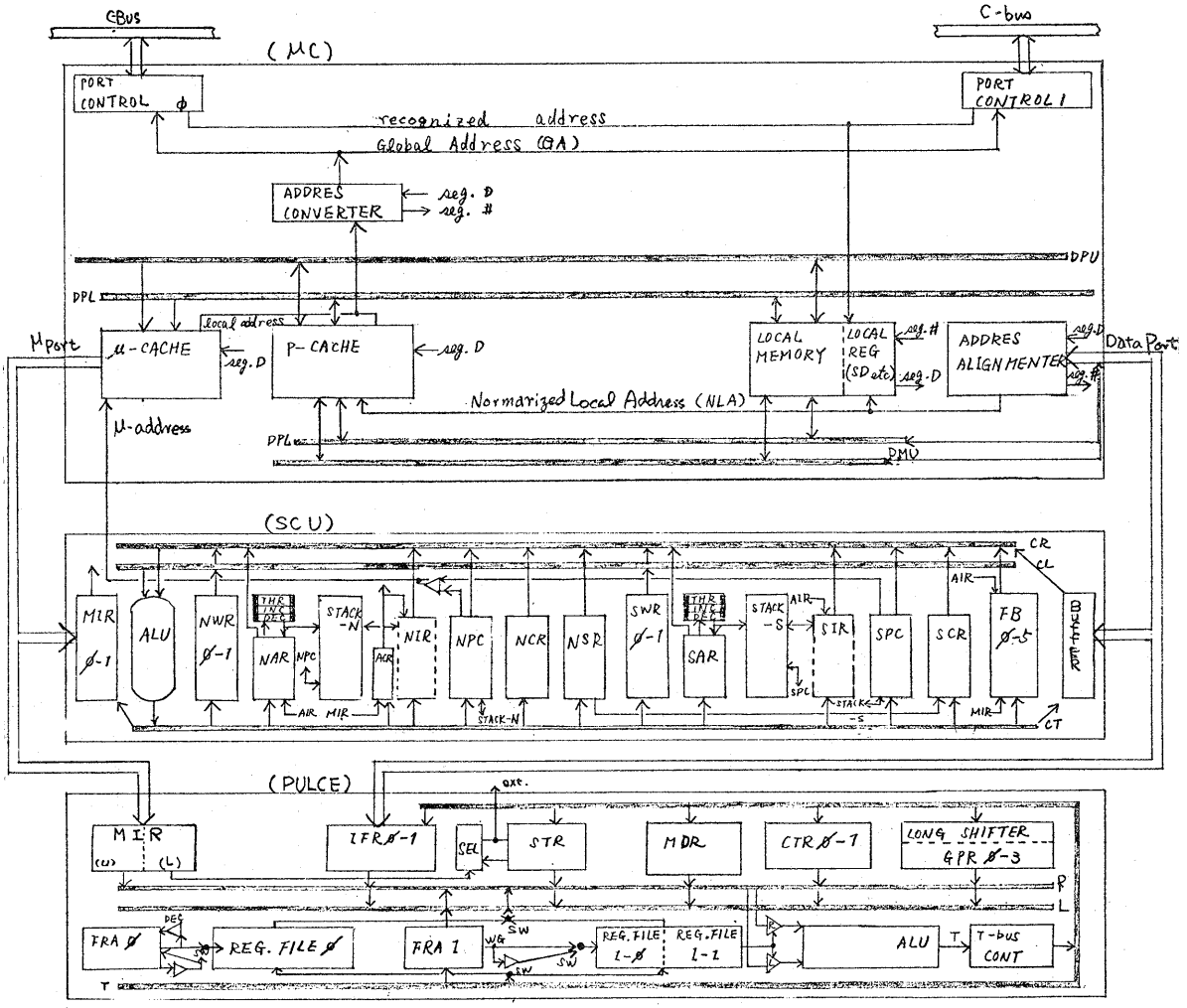


図３　ＰＲ－２，３の構成図

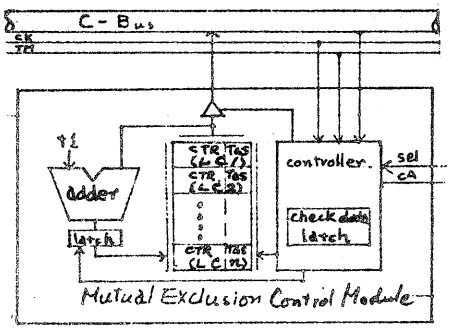


図4 MECMの構成図

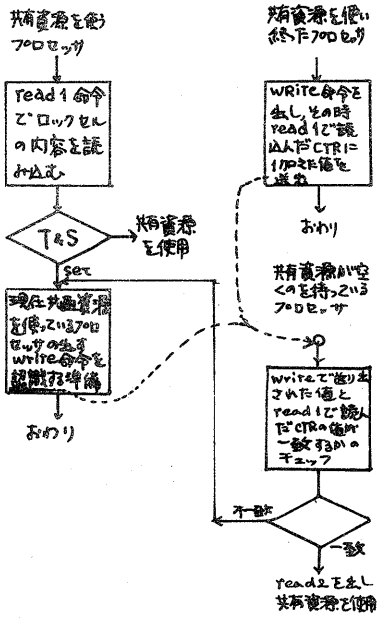


図5 相互排斥の実現法

[5] 基本OS

現在，基本OSは方式設計の段階でまだ完成していないが，設計方針などがほぼ定ったので以下に現状報告を記す。

(1) 設計方針

基本OSの主な設計方針は，以下の3点である。

(a) プロセッサモジュールはマイクロ命令とダイナミックマイクロプログラミングを採用している。したがって，特殊な中間言語を設定して，このレベルのみをユーザに解放する場合とユーザにマイクロプログラムのレベルを解放する場合とが考えられる。筆者等の研究室ではACEシステムを種々な実験に用いる予定であり，ユーザにマイクロプログラムを解放することは必須である。

(b) 入出力プロセッサのNOVAにあるRDOS (Real Time Disc Operating System) の機能を有効に利用することによって基本OSのオーバヘッドを減少させる。

(c) ACEシステムは自由な構成をとれるが，基本OSでは図1に示した構成のサポートおよび新しいハードウエアモジュールの追加に対する配慮のみに限定する。

(2) 基本OSの実現方法

OSを実際に作成するにあたり，ハードウエアの機能をどのように有効に使用すべきか問題となる。当基本OSの場合も，OSの核（OSK：OS Kernel と略記する）を，

(a) 特定の1プロセッサモジュール上にゆだねる (dedicate) か。

(b) 特定の1プロセッサモジュールに限らず，全てのプロセッサモジュールにゆだねるか。

と言う2つのアプローチが考えられ問題となった。(a)のアプローチは，専用OSKプロセッサが存在し，その他のプロセッサモジュールを言語プロセッサやユーザ用プロセッサとして管理すると言う感じである。したがって，各プロセッサからのOSKへの要求は全て外部からの通信によって行なわれる。一方，(b)のアプローチは，OSKがアクセスする共通エリアに相互排斥と標準インタフェースを確立し，OSKが任意のプロセッサモジュールで同時に動作可能とし，プロセスからの要求の多くを内部（ローカルに）で吸収しようと言うものである。

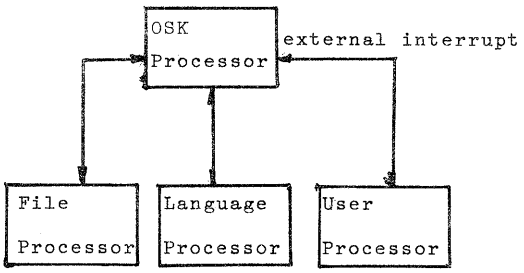
現在，筆者等は(b)のアプローチを取ることにしている。その主な理由は，

- (a)のアプローチによって専用OSKプロセッサを設けても，他の各プロセッサモジュールにも種々の割込が発生するから，何

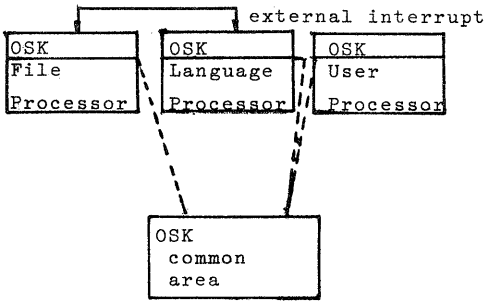
等かの形で割込処理を行なうOSKが必要である。

- OSKが特定の1プロセッサモジュールに固定されることによって，その他のプロセッサから，OSKプロセッサへの要求を何等かの手段で伝えねばならない。当然，OSKプロセッサとその他のプロセッサとの間で，要求および要求の結果に対する返答のため何回かの通信が必要となる。
- 筆者等の考えでは，NOVAに入出力やユーテリティプログラムの大部分をまかせる予定なので，OSKはそれほど重たくはならず，1プロセッサモジュールをOSKにのみゆだねるのはもったいない。そこで，OSKプロセッサの下でもプロセスを動作させようとすると，内部のプロセスからの要求と外部のプロセッサからの要求を同時に扱う必要が生じ，専用OSKプロセッサと言う意味があいまいになって来る。
- もし，仮に各プロセッサモジュールにOSKをゆだねることが出来れば，各プロセッサモジュールはローカルにOSKを有すると同時に，プロセス全体を共通エリア内のテーブルで統一して管理でき，自分に無い機能を必要とする時に限り，外部へ要求を出せば良くなる。もちろんOSKに対し各プロセスからどの位の頻度で要求が出るか問題である。
- (b)のアプローチの欠点は，ACEのように異ったアーキテクチャを有するプロセッサモジュールが存在する場合，ローカルに全くOSKと同一の機能を必要とすれば，それぞれ異った命令で書かれたOSKを作成せねばならぬ点である。また，上述したように共通エリア上で複数のOSKがシリアライズされる可能性があり，アイドル状態のOSKが生れる。前者に対しては，各プロセッサモジュール上にOSKの全機能を具備させる必要がない場合も存在するし，必要な場合でも(a)と似たアプローチを取りOSKに外部からの要求を受付ける入口を設ければ良い。後者に対しては，共通エリアをなるべく同時にアクセス出来るように分割し，その各々にロックセルを割当てる予定である。

以上のことを概念的にまとめたのが図6である。



(a) 専用OSKプロセッサのアプローチ



(b) 各プロセッサにOSKをゆだねるアプローチ

図6 OSKを実現する2つのアプローチ

(3) 空間の切換え

アドレス空間は，[3]章(2)で述べた機能を用いてOSKとプロセス，プロセスとプロセスの空間を分離/共有する。ACEシステムにおけるアドレス空間は，各プロセッサモジュール内で基準となっているLAをSDにセットされた内容によってGAへ変換し，固定される。

このように，各アドレス空間はSDによって規定されるから，プロセスとプロセスのアドレス空間の分離/共有は，プロセスごとにOSKがSDを管理することによって可能となる。そこで，このSD自体を管理するOSKとプロセスのアドレス空間の切換え(分離)を具体的にどのように行なうか次に述べる。

PR-2/3には，S-ステートとN-ステートと呼ばれるステートがあり，15のSDは

共通に両ステートで用いられる。当然，各々のステートで15のSDを自由に用いれることが望ましい。OSKでは，図7に示すような方法で，S-ステートとNステートのSD(空間)を切換えている。

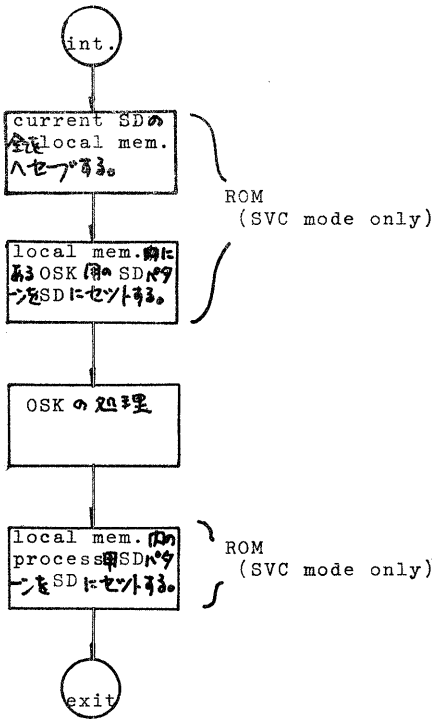


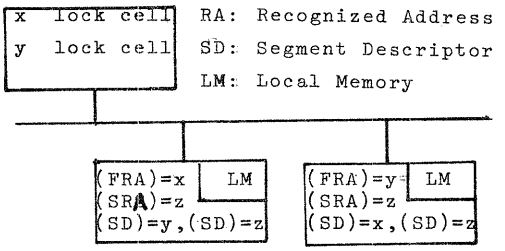
図7 空間の切換え (N→S→N)

ACEシステムにおいて，セグメント・ゼロ(ROM，LR：Local Register)へのアクセスはアドレス変換を受けず，かつS-ステートのみでアクセス可能である。また，LMはLR内にあるBD(Block Descriptor)をセットすることによって，N-ステートからのアクセスを禁止できる。但し，ROMは現在，Nステートからでもアクセス可能なため，改良する予定である。

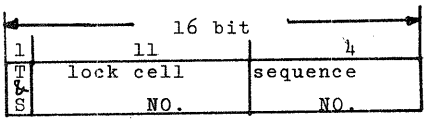
(4) モジュール間通信と相互排斥

前述したプロセッサモジュール間の通信と相互排斥制御モジュールを利用した通信形態を仮に直接通信と呼んでいる。直接通信は，ユーザプロセス間のプライベートな通信よりも，OSK間の連絡，入出力の要求，入出力終了報告など公共的な通信に利用する。

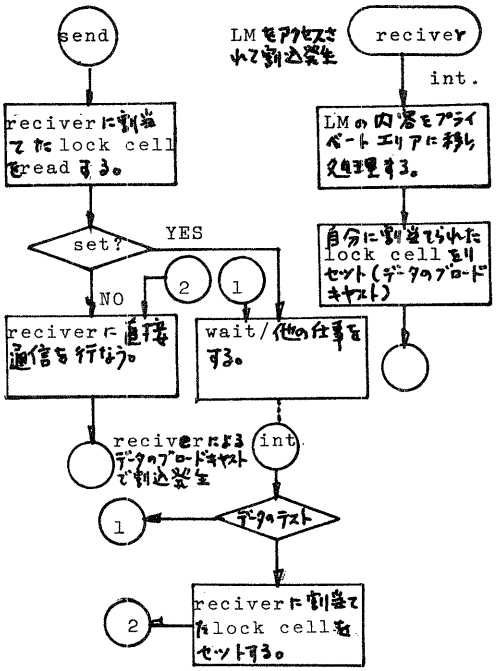
OSKでは，各プロセスから同時に発生する共通資源に対するアクセスを，ある期間シリアライズする必要がある。そこで，相互排斥制御モジュールの機能を用いて，図8に示す方法でシリアライゼーションを実現する。



(a) 直接通信のエンバイロメント



(b) ブロードキャストされるデータ



(c) 直接通信の実現

図8 直接通信の実現

図8は，プロセッサモジュール間の通信に対するものであるが，これと同様な方法でOSKの共通エリアのシリアライゼーションも行なう。

(5) OSKの構成

OSKは，上述した空間の切換えと直接通信の手段によって，[5]章(2)の(b)のアプローチを取ることが可能となる。図9に，OSK全体のブロック図を示す。

現在，OSKはPR-2，3用としてインプリメントする予定である。したがってPR-1や新しいアーキテクチャの異るモジュールの追加に対しては，外部割込に入口を設け，OSKの機能を外部からも利用できるようにする。例えば新しいプロセッサモジュールの追加に対しては，OSK用のプリミティブを解放し，特殊ハードウエアの追加に対しては，入出力デバイスの一種として扱うなどである。既存のPR-1に対しては，PR-1側に単純なイニシエーター&ターミネータを設け，プロセスの起動はOSK側から与え，プロセスの実行中はOSKプリミティブを図のext. Primitiveによって利用させる予定である。

図9から明らかなように，OSKの構成は従来のオペレーティングシステムの核の構成と何等変わる所はない。このように，従来のOSで用いられた技法をそのまま適用するだけで容易にOSKを作成できることは好ましいことなのかもしれない。また，システムサイドのユーティリティプログラム(コマンダ，アセンブラ，ローダ，メモリマネージャ，入出力ハンドラ等)をNOVAに置くことによって，プロセッサモジュール内で動作するプロセスを全てユーザ用のものに限定することが出来た。このことは，マイクロプログラムを頻繁に入換えたくないと言う希望をかなり満してくれる。つまり，プロセスの切換えは，自発，誤りによる以外は行はわないようにすることが出来るからである。

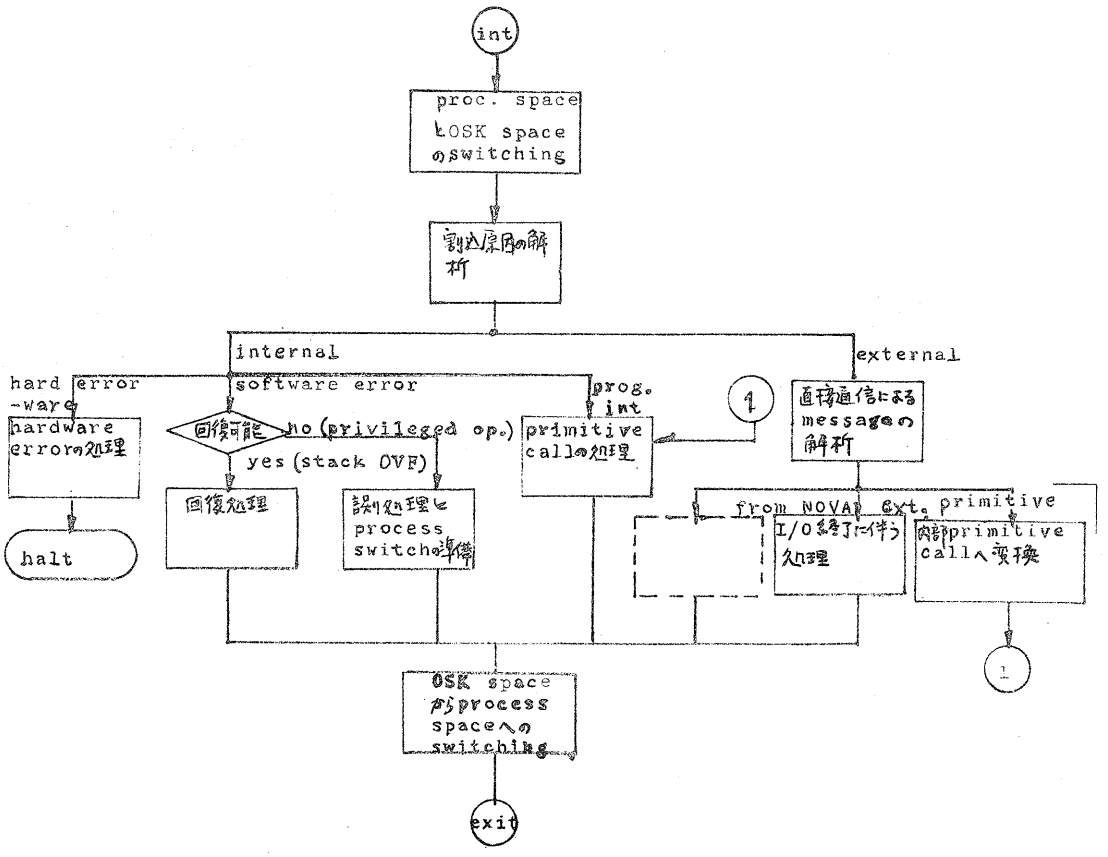


図9　OSKの構成図

(6) OSK用の主な共通テーブル

OSK用の主な共通テーブルとして，PRT (Processor Resource Table)，FRT(Firm-ware Resource Table)，μPCB(μ-Process Control Block)，CHT (cHannel Table)，MMT(Memory Management Table)などを考えている。

PRTは，プロセッサモジュールごとに1つのアイテムを有する。各PRTアイテムは，対応するプロセッサモジュールのシステムイニシアライズ時の初期値を保有する。また，現在実行中のプロセスに対応するcurrent PCBへのポインタも有している。PRTは，プロセッサモジュールのアーキテクチャの相違を吸収するため，2つの部分に分れる。図10に，PRTアイテムの概略を示す。

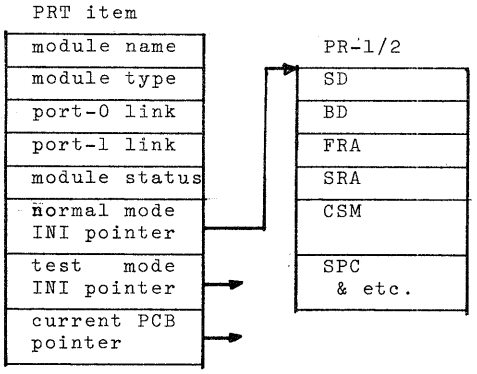


図 10 PRT item の形式

このように，OSKではプロセスを全く切換えないのではなく，前述したように自からブロックの状態になり得るため，プロセスの切換えに対する機能は存在している。表1に，PR-2/3上で動作するプロセスから他のプロセスへ，またはOSKへ切換わる時にどれだけのレジスタを入換える必要があるかを参考までのために示しておいた。この表から明らかなように，プロセスからOSKへの空間に変わると，INNRの85バイト分とSDの全て(一部分でもよい)の104バイトが入換わる。プロセスからプロセスでは，更に上述の分にNSTRの62バイトを加えたものを入換えねばならない。

FRTは，主メモリにロードされたマイクロプログラムごとに1つのアイテムを有し，マイクロプログラムに関する種々な情報を含んでいる。また，μPCBはメインプロセスとなるものは最初FRTアイテムによってイニシアライズされ，プロセスに関する状態情報などを保有している。この2つのテーブルの概略を図11に示しておく。

その他のテーブルについては，ここでは省略する。

name	個	B	INI	OSK	process	
LR Local Reg.						104B
SD	16	64	○	△	○	
BD	8	16	○			
FRA	2	4	○			
SRA	2	4	○			
ECR	2	8				
CSTR	1	8	○			
CSM	1	8	○			
LMB	8	128	△			1024B
SSTR S-stae reg.						44B
SAR	1	2	○			
SIR	1	2	○			
SPC	1	2	○			
SCR	1	2	○			
SWE	2	4				
stack	16	32				
NSTR N-state reg.						62B
NPC	1	2	○		○	
IFR	2	4			○	
NSR	1	2	○		○	
NCR	1	2	○		○	
NWR	2	4			○	
NAR	1	2	○		○	
NIR	1	2	○		○	
FB	6	12			○	
stack	16	32			○	
INNR Internal reg.						85B
GPR	4	8		○	○	
FAR	2	4	○	○	○	
MDR	1	2	○	○	○	
STR	1	2	○	○	○	
CTR	2	4		○	○	
RF0	16	32		○	○	
RF1	16	32		○	○	

表1 空間の切換えとレジスタの関係

(7) OSKプリミティブ

詳細は未定であるが，おおよそ表2に示したように，入出力用，プロセス制御用，その他のプリミティブを設ける予定である。これについては，説明するまでもないので省略する。表中で特殊なプリミティブはsend/recive directである。

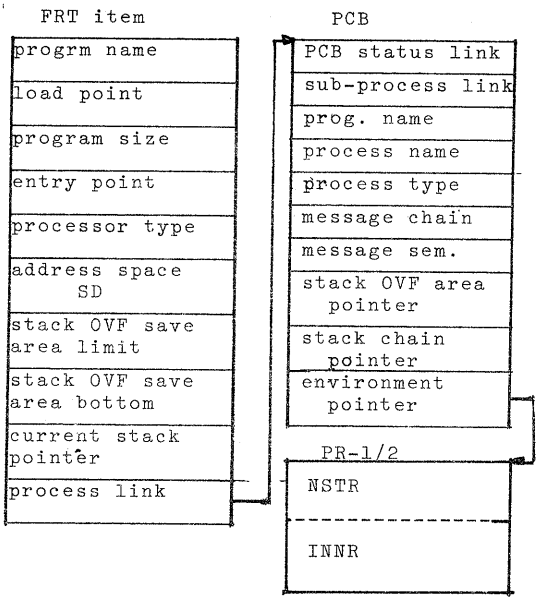


図 11　FRT item と μPCB の形式

I/O primitive	
open channel	擬似チャネルのオープン
close channel	〃　クローズ
start I/O	入出力の要求
sense CH status	入出力終了と状態のチェック
process control primitive	
create	サブタスクの生成
delete	タスクの消滅
wake-up	〃　起動
block	〃　停止
send	メッセージの転送
recive	〃　受け取る
term	プログラムの終了
other primitive	
P-operation	
V-operation	
send/recive direct	他のプロセッサモジュールのLMを利用
etc.	

表 2　OSK Primitive

(8)　ユーテリティプログラム

前述したように，システムサイドのユーテリティプログラムを全てNOVAに置く。これ等をマイクロプログラムで記述してPR上に置いても良いが，そのためにプロセスの切換えが強制的に行なわれたのでは得策でない。NOVA上のプログラムを仮にシステムプロセスと呼んでいるが，実際にはRDOSのユーザインタフェイスとマルチタスクの機能を用いて動作させる予定である。

(9)　将来の拡張

現在のOSKの機能だけでは，単にマイクロプログラムをACEシステム内にロードし，実行させることが出来るに過ぎない。しかし，マイクロプログラムをサポートする以上は，マイクロプログラムのデバッグに適した環境を整えてやることが望ましい。例えば，現在のままでは，LM，LR，SSTRは全てOSKによって管理されているため，プロセス内から直接にアクセスすることは出来ない。

そこで将来の拡張としては，OSK同志の空間を分離し，全く独立した空間内で，同時に別のモードのOSKを動作させることを考えている。そして，新しい考え方ではないが，これも既存の技法を用いて，分離された空間内に仮想マシンを実現させ，S－ステートのプログラムも実行可能にしたいと考えている。そのため，図10にtest modeに切換える時に必要な初期値を示すポインタを置いておいた。

[5]　結び

ACEシステムは現在ハードのレベルで一段落し，基本的な仕事はソフトに移りつつある。ハードの方は細かい微調整を残すのみで，次の仕事として新しいモジュールの結合計画がねられている。さいごに，筆者等の研究に対し御討論，御協力いただいた計算機方式研究室の諸氏ならびに日頃御指導いただく西野博二部長，石井　治部長，黒川一夫部長に感謝いたします。また，ACEシステムの製作・調整を行なっていただいている日本ミニコンピュータの皆様，ソフトウェア作成に協力していただいている中山照章氏に感謝いたします。

[6]　参考文献

[1]～[3]　飯塚他：ACE，マイクロプロセッサユニットのアーキテクチャ他，情報処理学会計算機アーキテクチャ研究会資料 3～5 (1975. 10. 1)

[4] H. IIZUKA et al：ACE－A NEW MODULAR COMPUTER ARCHITECTURE, 2nd USA-JAPAN Computer Conference (1975)