資　料

ローカルエリアネットワーク端局用
リアルタイムモニタ
取 扱 い 説 明 書

昭 和 59 年 3 月

JIPDEC

財団法人　日本情報処理開発協会

財団法人 図書 1.11.6 日本情報処理開発協会

JIPDEC
58
X006

この資料は、日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて、昭和58年度に実施した「マイクロコンピュータの利用に関する共通的な技術開発」の一環としてとりまとめたものであります。

目　　次

1. 概　　要

1.1 開 発 目 的

５７年度に開発された"光ファイバを用いた簡易型ローカルエリアネットワークシステム""インテリジェントディスクファイル管理ユニット"と５６年度に開発された"インテリジェント・ディスクユニット"を使用することを条件として、複数の端局を有するデータ収集システム・プロセス制御システム・ソフトウェア開発システムの分散処理システムを構築する際の基礎となるリアルタイムモニタを開発する。

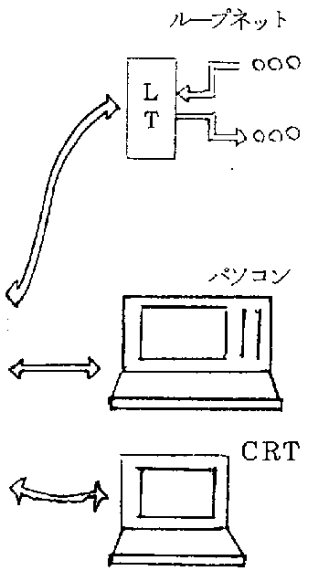
1.2 開発ソフトウェア概要

AZTEC C86コンパイラ(CP/M86版)を採用し、パーソナルコンピュータを用いてアプリケーションであるテストプログラムまでrom化(read only memory)した次図のようなネットワーク端局用プログラムを製作した。

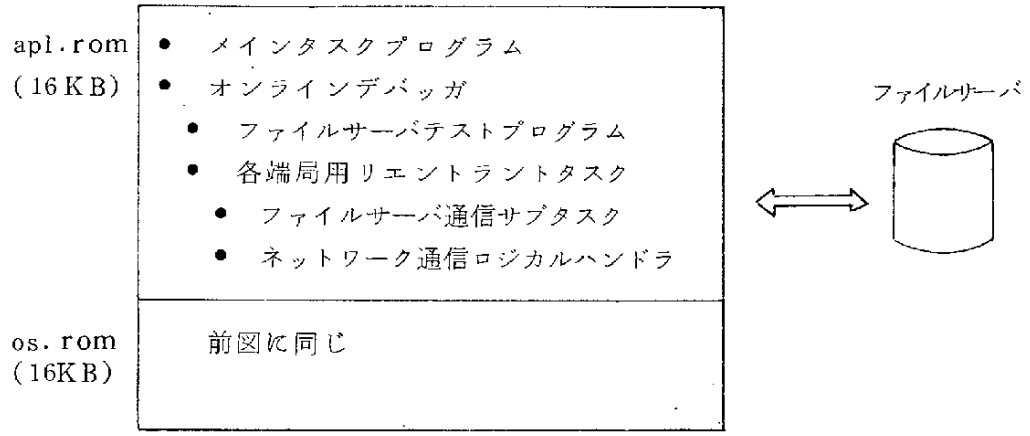
apl.rom (16KB)
- ネットワーク通信テストプログラム
- ファイルサーバ通信テストプログラム
- オンラインデバッガ
  - ファルサーバ対象 save・load 処理
  - ネットワーク通信ロジカルハンドラ(nxxx)

os.rom (16KB)
- ネットワーク通信サブタスク
  - LST−LT通信ハンドラ
- プログラムローダ
- コンソールロジカルハンドラ(accept)
  - コンソールフイジカルハンドラ(excp)

◎ タスク制御カーネル
- ○ 電源投入タスク制御開始
- ○ ソフト割込マクロ処理
- ○ ハード割込入出力制御



ファイルサーバ端局用のプログラムは次のようになっている。



アプリケーションプログラム（タスクプログラム）からソフトウェア割込を用いてタスク制御カーネルに連絡するマクロには次のものがある。

○ 同期制御 …… fpost, waitf, post, waitp

○ タイマ制御 …… stime, gtime

○ 入出力制御 …… excp, setdi

○ 割込制御 …… setsrf, setsrp, waiti

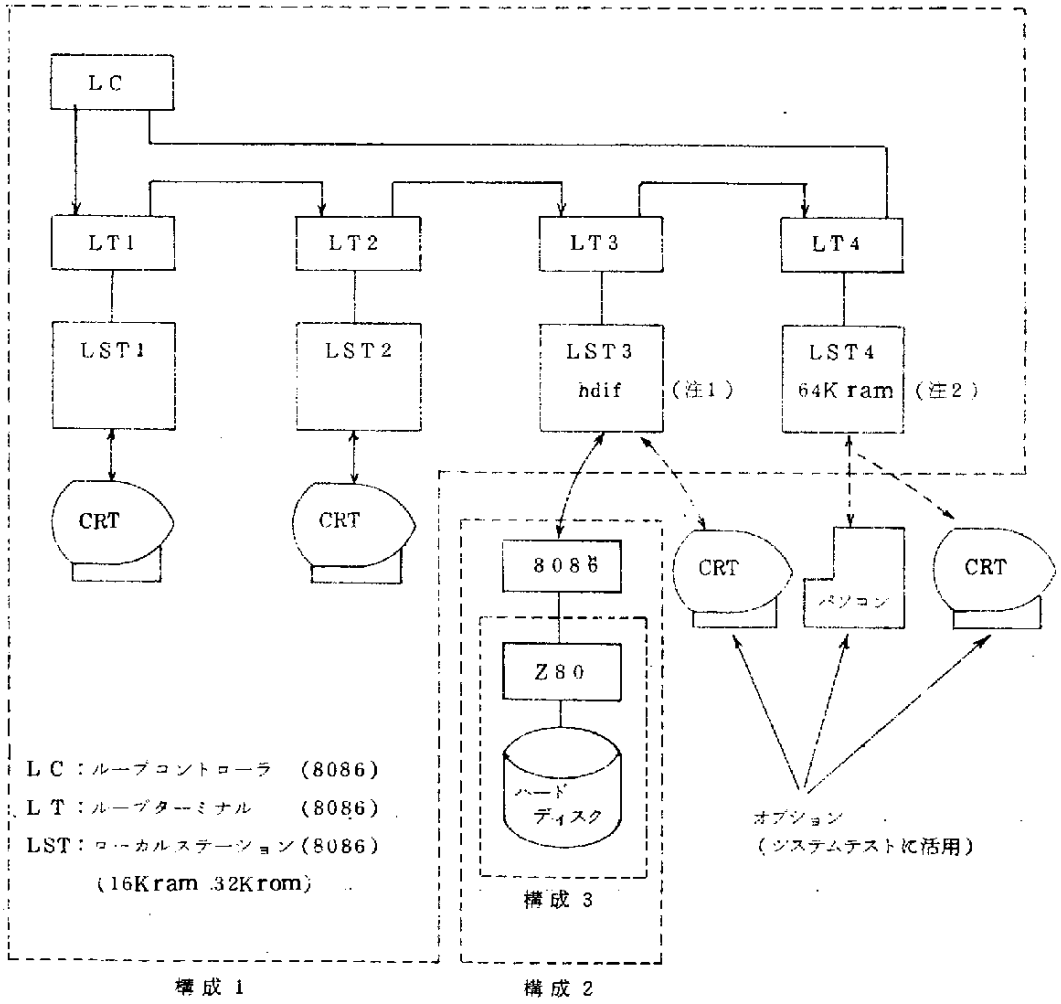
○ 事象確認 …… waitm, rflag, cflag

○ 資源制御 …… enq, deq

○ タスク制御 …… attach, dttach, chap, next, exit

なお前図のように、アプリケーション側apl.romとモニタ側os.romとわける方法ばかりでなく、全体を1本にまとめるsys.romも作成できるし、アプリケーション側をram(random access memory)としておき、パーソナルコンピュータからプログラムをローディングすることも可能としている。

1.3　ハードウェア構成



構成１：光ファイバを用いた簡易型ローカルネットワークシステム（５７年度）

構成２：インテリジェント・ディスクファイル管理ユニット（５７年度）

今回このものをファイルサーバと呼んでいる。

構成３：インテリジェント・ディスク・ユニット（５６年度）

（注１）　ファイルサーバ接続用インターフェイス（今回増設）

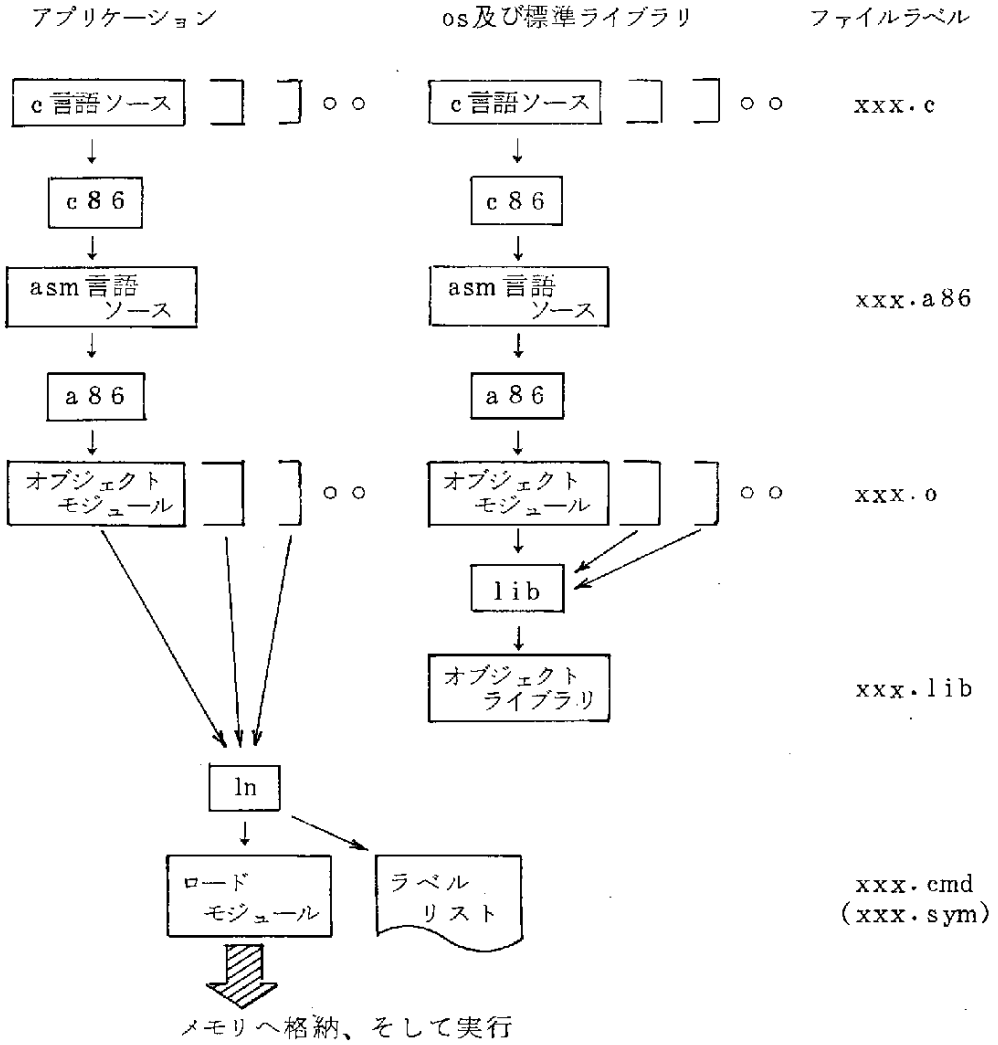
（注２）　ram 空間でのアプリケーション動作確認用 ram 基板（今回増設）

1.4 AZTEC C コンパイラ概要

58.10時点にて唯一のアッセンブリ言語ソースを出力する8086 C コンパイラである。カーニハンとリッチーの言語標準仕様を満足しており、コンパイラ・アッセンブラ・ライブラリユティリティ・リンカを持っており、ランタイムライブラリをすべてC言語及びアッセンブリ言語にて公開しているものである。

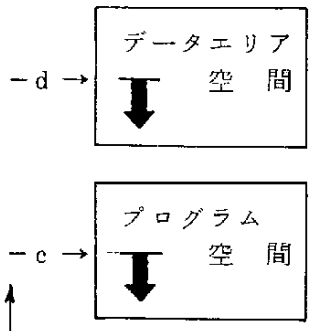
(1) コンパイルとリンケージ

n本のソースプログラムから次の様にロードモジュールが作られる。



(2) オブジェクトプログラムのメモリ空間

AZTEC C のターゲットマシンは一般的なパーソナルコンピュータ（CP/M86 or MSDOS）であるため、全メモリがram空間であると想定し、メモリ効率とスピードを上げるためにセグメントレジスタの管理を行なっていない。（他社のCコンパイラでも同じ様子である）そのため次のようにメモリ空間をデータ用とプログラム用に完全にわけている。



（注1） 8086セグメントレジスタ
ds：データセグメント
ss：スタックセグメント
es：エクストラセグメント
cs：コードセグメント

└リンカ（ln）のオプション機能であり、データ及びコードのロードポイントをオフセット値として設定できる。

アプリケーションプログラムをrom(read only memory) にするとなると次の様な問題が発生し、対処が必要となる。

① C言語仕様では実行開始のデータ空間はNULL（Zero）である。

→ モニタ動作開始時点にてメモリクリアが必要

② 文字定数と初期値はデータ空間に設定される。

→ プログラム空間に乗せておき、開始前にデータ空間へ移動

③ C言語のアプリケーションの先頭はmain( )関数である。

→ 電源ONにてアッセンブラプログラムを動かし、セグメントレジスタセット後main( )関数を呼ぶ必要がある

④ 出力ファイルは×××.CMDであり、アッセンブラの×××.HEXではない。

→ ×××.CMDからromに焼く方法が必要

⑤　64KB以上のプログラムとデータの扱い。

→ モニタマクロにて64KB空間渡りのサポート必要

(3)　C言語の変数

C言語の変数には次の3種が用意されている。モニタの理解とアプリケーション作成の助けとすべく簡単に記す。

①　内部変数（auto変数又はスタック変数）

関数の中で定義される変数でスタックエリアが用いられる。関数の入口にて確保され関数の出口にて消滅する作業用変数であって、C言語の最も使い易い特長的な変数である。

②　static変数

メモリの定位置が割当てられる変数で、定義したソースファイル上のどの関数からでも自由に読み書きできる変数である。異なるファイルからは参照できない。又、リンカ（ln）のラベル表には出力されない。

③　外部変数（common変数）

異なるソースファイルからの共通変数、アッセンブリ言語プログラムとの共通エリア定義に用いられるものである。リンカ（ln）のラベル表には出力されるのでシステム上重要な変数に用いられる。

(4)　本説明書にて用いるC言語特有の表現

①　main( )　……　最初に実行される関数

②　×××(arg1、arg2•••)　……　引数を持っての関数の呼出し

③　var＝×××(arg•••)　……　関数呼出し後の関数値の採用

④　wcb.cd　……　wcb構造体の中のcd変数内容

⑤　& wcb　……　wcbエリアのアドレス

⑥　大文字　……　#define定義した定数とグローバルな構造体

⑦　小文字　……　変数および関数名

2. タスク制御カーネル

リアルタイムモニタの核（カーネル）となる部分であり、一般的な制御方式にこだわらず、ユニークな発想を加えてrom上にて動くタスク制御カーネルを作成した。大きな特長として次のものがある。

(1) 一般的なpost(ecb, postcode)・wait(ecb)のようなecb(event conrol block)を採用せず、タスク番号(Ti)主体におくPost(Ti, postcode)・waitp(& wcb)を採用してプログラムを見易くしている。

(2) 制御システムにあっては非常に有効なビットONによるタスク間通信をfpost(Ti, postbit)・waitf(waitbit)によって実現している。

(3) setsr(set subroutine)マクロを用意して、実行中に割込処理ルーチンが登録でき、割込発生をipost（ポストコードをキューイングする方法）とfpost（タスクにビットONを通知する方法）の両方にて対処できるようにしている。

(4) 入出力ハンドラの上位階層ハンドラをリエントラントサブタスクとして作成できる。……これはC言語の最大特徴かも知れない。

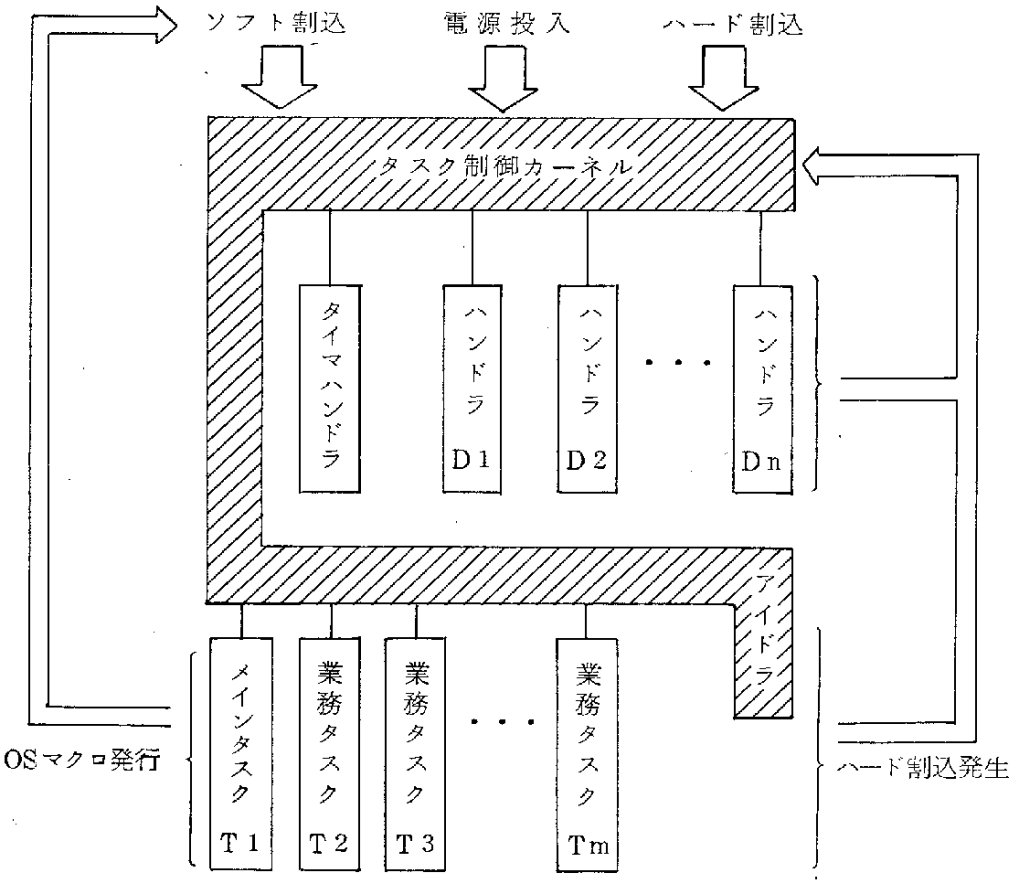
(5) ほとんどをC語でプログラミングし、アッセンブリ言語のコーディングは割込関係のみとしている。……C言語によるテーブル定義は非常に助かっている。

2.1 タスク制御カーネル仕様

リアルタイムモニタの核となる部分であり、ユーザが業務処理を行うタスクプログラムと入出力処理を行うハンドラプログラムの管理を行う。

(1) プログラムの位置構成

タスク制御カーネル（狭義のOSと表現する場合がある）は、電源投入イニシアライズとハード割込とソフト割込（OSマクロという）を扱い、次図の様な構成位置を示す。



（注1） アイドラはハード割込を待つコーディングであり、通常アイドルタスクとも言われる。

(2) タスクプログラム

ＯＳより制御を受ける単位のプログラムで、通常１本のタスクが１個の業務を受け持つ。タスクの性格としては次のものがある。

① タスク単位にスタックエリアが必要。

割込発生によってすべてのレジスタはそのスタックエリアに退避される。

② タスクはタスク番号を持つ。

このタスク番号を使って相手タスクと通信する。

タスク番号とは tcb(task control block)テーブルの中での自分の存在場所番号である。

③、タスクはプライオリティ番号を持つ。

0～255の中の数であり、0に近いほど優先度が高く、早くcpu時間がサービスされる。

(3) ハンドラプログラム

入出力動作を扱うプログラムであり、タスクプログラムの中のexcpマクロによって入出力動作の開始が指示され、入出力割込によってcpuタイムがサービスされるプログラムである。

ハンドラの性格としては次のものがある。

① ハンドラのプログラム開始アドレスはdcb(device control block)テーブルの自分の存在場所の1部に登録される。

② ハンドラプログラムのデータエリアはdcbの中にある。

入出力動作の進行状況は自分で管理する必要がある。

③ スタックエリアとしてはシステムスタックエリアが使われる。

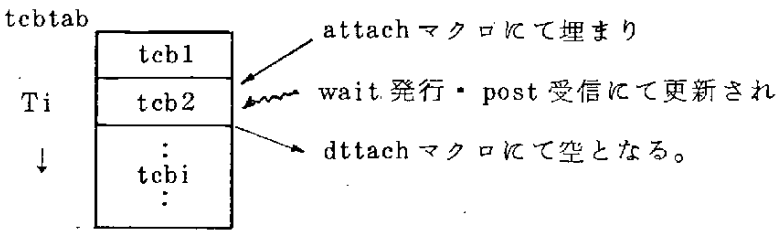
④ ハンドラは割込みが発生する毎にcpuを渡されるため、入出力動作の途中なのか終了なのかを厳密に扱わないといけない。完全終了にて初めて起動をかけたタスクに通知される。

(4) ＯＳテーブルの概要

タスク制御と入出力制御(ハンドラ制御)を行うために、本ＯＳでは次のテーブルを用意している。ＯＳマクロ理解を助けるために概要を示す。

① tcb(task control blok)テーブル

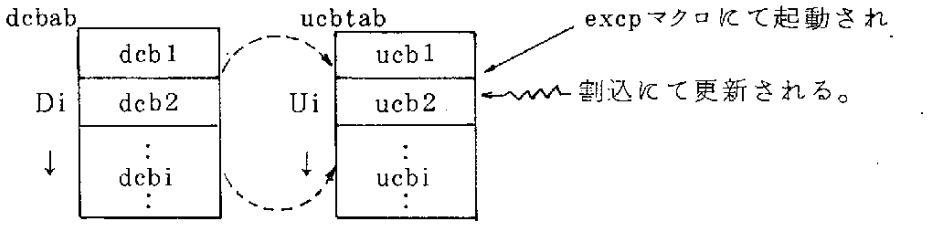
タスクの状態を示すテーブルであり、タスク番号順に割当てられ、必要タスク分(ＴＣＢＳＩＺＥ)確保される。



② dcb(device control block)テーブル

ucb(unit device control block)テーブル

ハンドラアドレスを登録し、入出力装置の状態を示すテーブルであり、ＯＳより電源投入時にコールされるtabset( )モジュールの中で登録する。（DCBSIZE、UCBSIZE）



（注1） ハンドラはdcb 1個に1モジュール結合される。

（注2） 1個のdcbにn個のucbが登録できる。

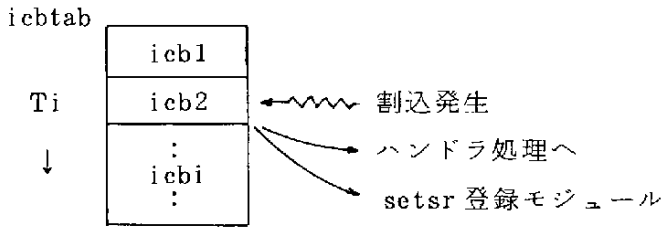
（注3） ucbのないdcbも登録できる。

（注4） excpマクロでは、dcb番号とucb番号を指してＯＳへリンクする。

③ icb(interrupt control block)テーブル

割込み要因を登録するテーブルであり、ハンドラが結合される時には、そのdcb番号を登録し、ハンドラが結合されない時には実行時にsetsrのモジュールが登録される。

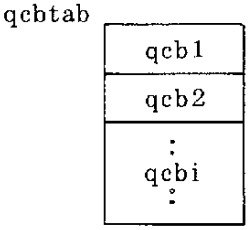
このテーブルもＯＳより電源投入時にコールされるtabset( )モジュールの中で登録する。（ICBSIZE）



（注1） 普通のＯＳではこのicb情報はdcbに格納されており、本システムは特殊である。

④ qcb(queue control block)テーブル

post, excp, enq, stime 等の待行列を格納するエリアであり、QCBSIZE にて、余裕をもった必要個数を確保する。



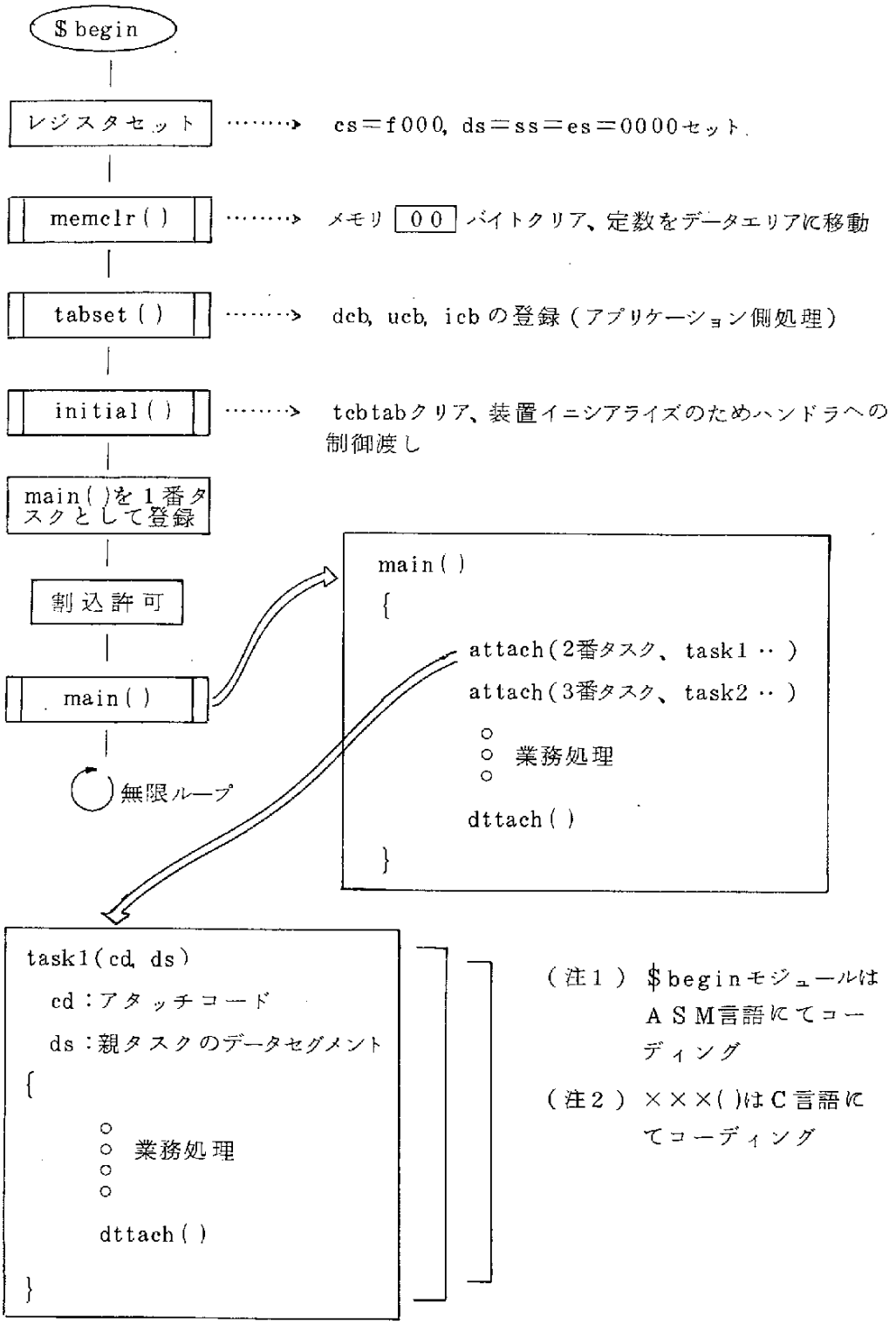
(注1) tcb, dcb, ucb, icb よりキューチェインされる。

(5) 電源投入からアプリケーションへの制御渡り

intel 8086 cpu は $(F000)_{16}$ セグメントの$(FFF0)_{16}$ オフセットの番地より命令を実行する。その場所には

```
jmpf  $begin(f000:seg)
```

が格納されており、アプリケーションプログラムに制御が渡るまでのフローチャートを示すと次の様になる。


$begin
レジスタセット
cs＝f000, ds＝ss＝es＝0000セット
memclr( )
メモリ 00 バイトクリア、定数をデータエリアに移動
tabset( )
dcb, ucb, icb の登録（アプリケーション側処理）
initial( )
tcbtabクリア、装置イニシアライズのためハンドラへの制御渡し
main( )を１番タスクとして登録
割込許可
main( )
無限ループ
main( )
{
attach(2番タスク、task1 ‥)
attach(3番タスク、task2 ‥)
業務処理
dttach( )
}
task1(cd, ds)
cd：アタッチコード
ds：親タスクのデータセグメント
{
業務処理
dttach( )
}
（注１） $beginモジュールはＡＳＭ言語にてコーディング
（注２） ×××( )はＣ言語にてコーディング

(6) サンプルプログラムと＃defineファイル

ＯＳの理解を早めるためにサンプルプログラムを提示する。

このサンプルは、最初に実行されるmain( )関数（タスク番号１番で最優先プライオリテイ０を持つタスク）で、タスク番号２と３を起動し、(esc)キーインを割込にてとらえてオンラインデバッガ（debug( )）を呼び出し、デバッガのpost, fpost コマンドにてタスク間通信を行う例である。

表面上のcrtの動作として

① 電源投入で、タスク２番・３番が動き出し

```
task 1   run
task 2   run
```

② (esc)キーインでデバッガが呼び出され

```
***  online debugger  ***
＝コマンド入力
   :

＝fpst, 2, 1 ◀━  fpostの代行
```

③ ２番タスクのwaitfが解除されて

```
fpost receive & debugger post
```

④ post(1, 0x001b)にてデバッガが呼び出され

```
***  online debugger  ***
＝post, 3, 1 ◀━  postの代行
```

⑤ ３番タスクのwaitpが解除されて

```
post receive PC 0001  0001  0000
```

⑥ 次の (esc) キーンを待つ……②へ

（注１） すべてのアプリケーションプログラムはシステム定数を用意している＃define ファイルを＃include " aplstr.c " しないといけない。

サンプルプログラム

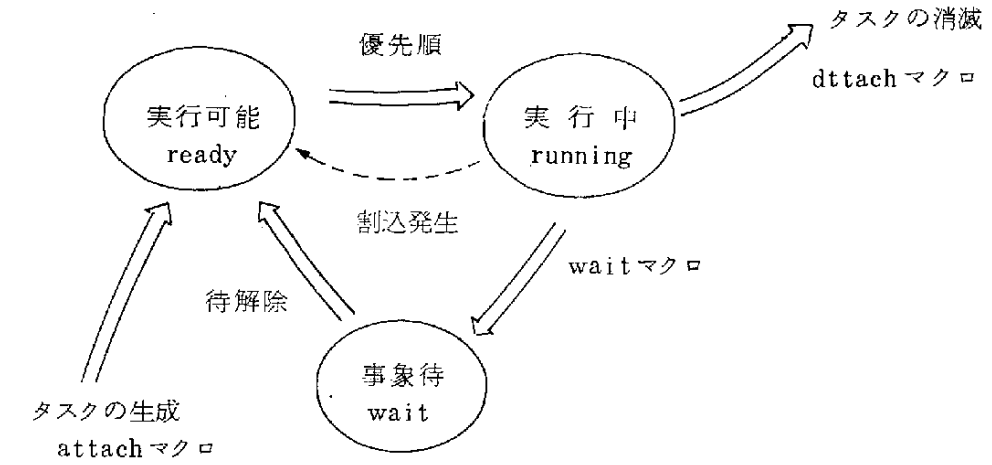
```
 1: /*** mcc4.c 59.2.14 <- 59.2.14 ***/
 2: /***   debugger, post, fpost test   ***/
 3: /* operation      1. esc key-in    debugger on
 4:                    2. fpst,2,1      message
 5:                    3. post,3,1      message          */
 6:
 7: #include "aplstr.c"
 8:
 9: #define TASK1   2
10: #define TASK2   3
11:
12: static int stack1[300], stack2[300];
13:
14: main()
15: {
16:         int timer();
17:         int task1(), task2();
18:         struct WCB wa;
19:
20:         setsr(ICBTIME, timer, PC, SHD);          /* timer start */
21:         attach(TASK1, 10, 0x0010, task1, &stack1[299]);
22:         attach(TASK2, 11, 0x0020, task2, &stack2[299]);
23:
24: trigger:
25:         setdi(CONSOLEI, SET);
26:         waitm(0x6000, &wa);
27:         switch (wa.cd) {
28:                 case 0x001b:    debug();         /* esc */
29:                                 break;
30:                 case 0x0003:    exit();          /* control-c */
31:                                 break;
32:         }
33:         goto trigger;
34: }
35:
36: task1(code, ds)
37: unsigned code, ds;
38: {
39:         struct WCB wb;
40:         printf("task 1 run¥n");
41: loop:
42:         waitf(0x0001);
43:         printf("fpost receive & debugger post¥n");
44:         post(1,0x001b);
45:         goto loop;
46:
47: }
48:
49: task2(code, ds)
50: unsigned code, ds;
51: {
52:         int wa;
53:         struct WCB wc;
54:         printf("task 2 run¥n");
55: p1:
56:         waitp(&wc);
57:         printf("post receive pc=%4x %4x %4x¥n", wc.stn, wc.cd, wc.ds);
58:         goto p1;
59: }
```

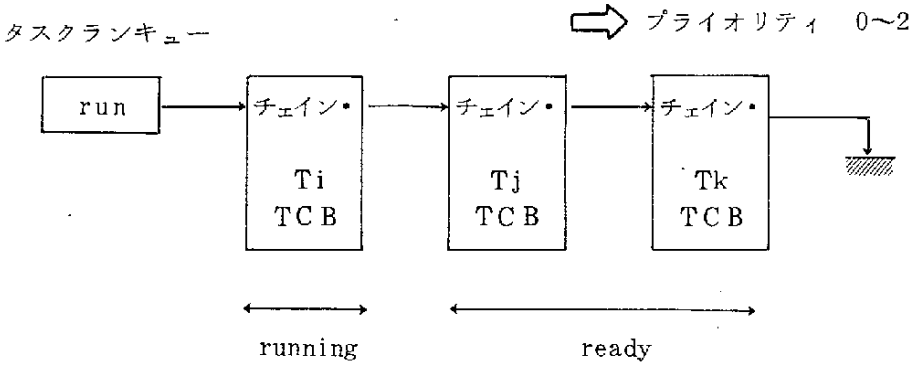
define ファイル

```
 1: /*  aplstr.c  59.2.29 <- 59.2.15  */
 2:
 3: /*   --- os table size ---    */
 4: #define TCBSIZE 7
 5: #define DCBSIZE 3
 6: #define UCBSIZE 2
 7: #define ICBSIZE 8
 8: #define QCBSIZE 24
 9: #define VECADR  0x0080
10: #define EOIPORT 0x00e0
11: #define EOIBIT  0x0020
12: #define CONSOLE  1       /* dcb */
13: #define CONSOLEI 1       /* icb in */
14: #define CONSOLEO 2       /* icb out */
15: #define ICBTIME  3       /* timer */
16:
17: /*   --- os constant ---    */
18: #define NULL 0
19: #define SET 0            /* setdi */
20: #define REP 1
21: #define RES 2
22: #define FP  0            /* setsr */
23: #define PC  1
24: #define SHD 0
25: #define DIR 1
26: #define W   0            /* enq deq */
27: #define R   1
28: #define T   0
29: #define D   1
30: #define I   2
31: #define U   3
32:
33: /*   --- waitp, waiti receive block ---    */
34: struct WCB {
35:         int stn;
36:         unsigned cd;
37:         unsigned ds;
38: };
```

2.2　タスク制御カーネルマクロ

(1)　タスクの状態遷移





（注1）　実行中にwaitマクロを発行すると、チェイン＝0xffffとなり、ランキューよりはずされる。

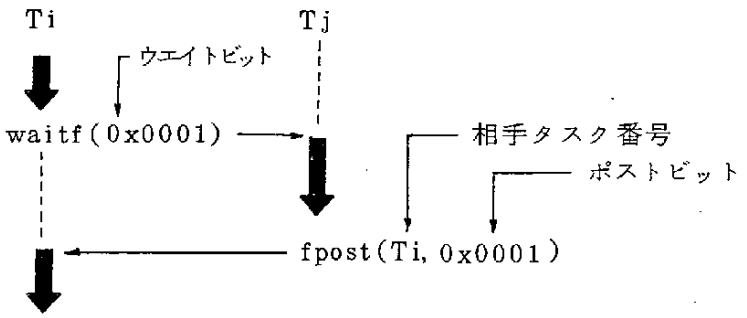
（注2）　wait中に待が解除されると、プライオリテイを見てランキューに結合される。runningのところか、途中か、チェインの最後に。（最後の時はchain＝0x0000）

（注3）　タスクとして登録可能な数は最大255である。

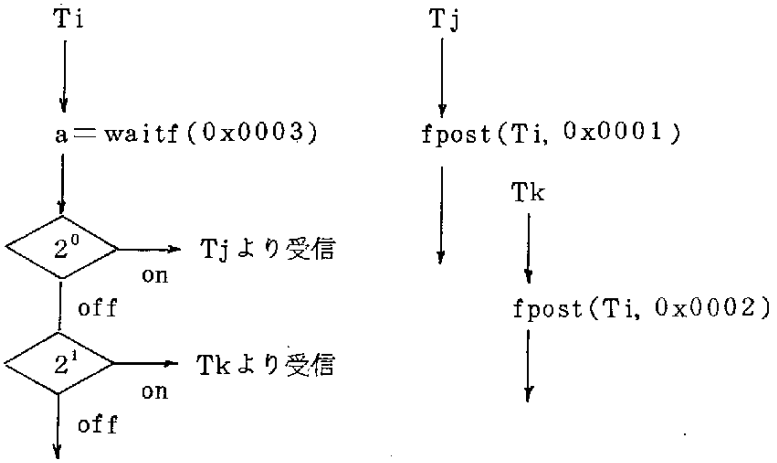
（注4）　以後の説明においてTiはタスク番号（TCB番号）、Diはデバイス番号（DCB番号）、Uiはユニット番号（UCB番号）、Iiはインタラプト番号（ICB番号）を示し、通常＃defineにてユニークな名称を大文字にて定義する。

(2) タスク間通信

① waitf と fpost …… ポストビットを用いるタスク通信

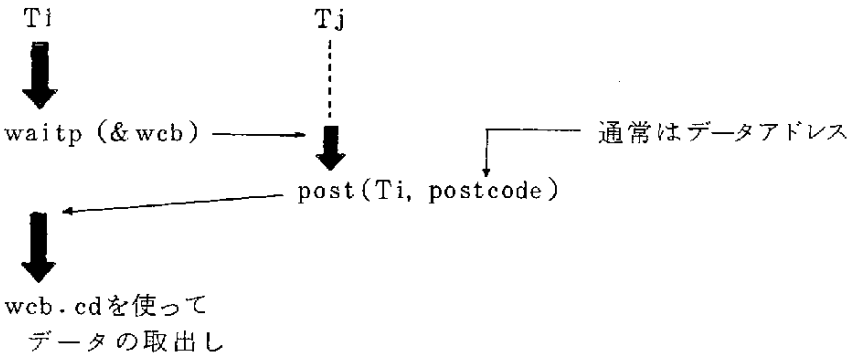


- waitf のビット位置と fpost のビット位置がマッチングした時、wait は解除されタスクは実行可能となる。
- ポストビットワードはＴＣＢの中に確保されており、fpost にて on となり wait が解除されて off となる。
- 次の例は２本のタスクよりポストビットを受ける例である。



- ポストビットはＴＣＢのエリアの中で重ねられるので、ひとつのビットに waitf にて取り上げる前に fpost を発行すると、２度のものが１度として解釈される。便利な時もあるし、不便の時もあるので注意を要す。

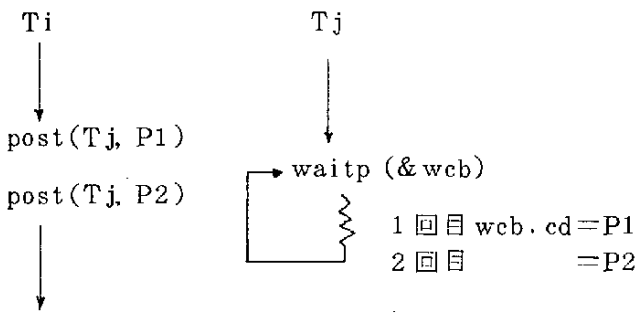
② waitp と post……ポストコードをもってのタスク通信



- wcbエリア（受取りエリア）……

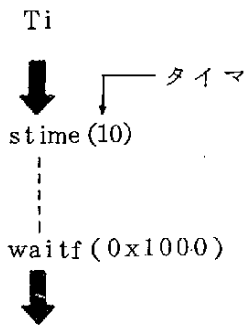
stn	post 発行元タスク番号
cd	postcode
ds	Tjのデータセグメント

- ポストコードはwaitpにて取りあげるまでqcbtabにキューイングされている。次の例はTjが2回まわる。



(3) タイマ管理

③ stime と waitf ……タイムアップ待



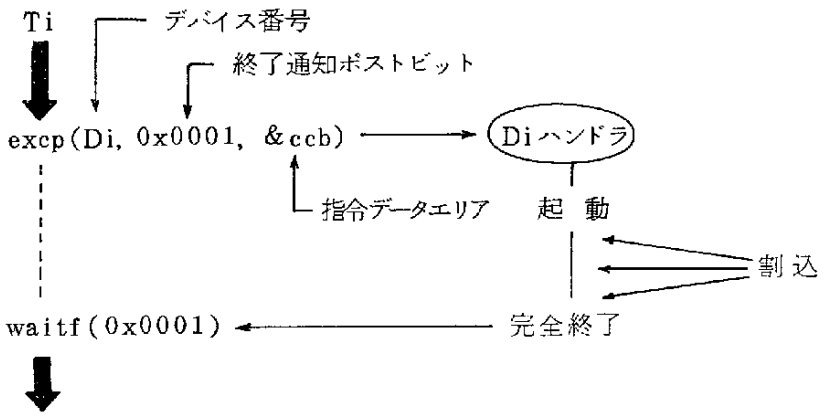
osタイマが1秒間隔なら10秒後にwaitf が抜ける。

- タイマ待は必ずウエイトビットの 2^{12} ビットを用いること。
- stime (0)はタイマリセットとなる。

④ 残り時間＝ gtime ( )…… stime (t)後の残り時間取出し

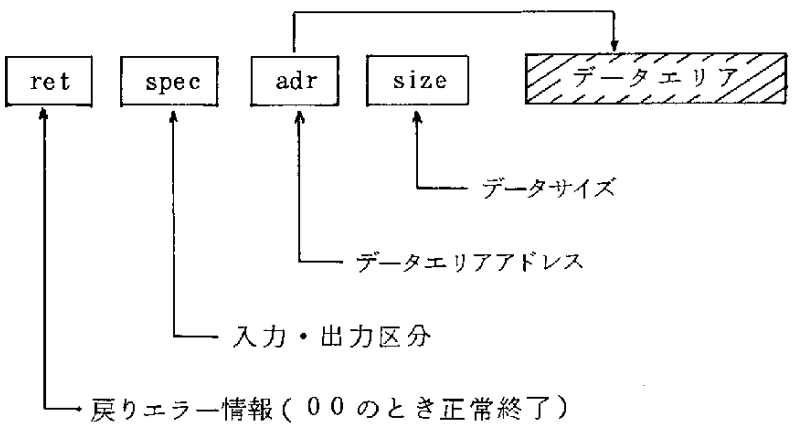
(4) 入 出 力 管 理

⑤ excp と waitf ……デバイス起動と終了時



- ccb(command control block)は excp の指令データとハンドラからの戻り情報を受取るために利用される。

一般的には次の様なエリアとなろう。（ハンドラ仕様にて異なる）



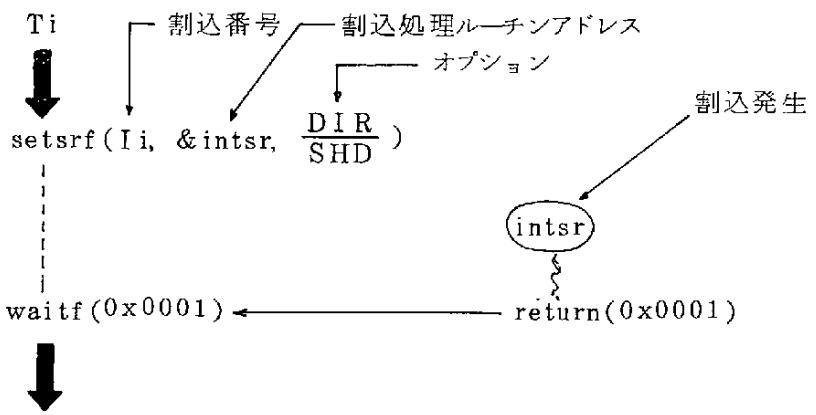
⑥　setdi と waiti ……起動割込待

- waiti は機能的には waitp と同じである。異なるのはタスク側からの通信か、割込側からの通信かということである。
- オプションにはSET(1回のみの割込受付)、REP(RESがかかるまでの割込受付)、RES(PEP、SETのリセット)がある。
- ハンドラにて発行する postcode を web.cd にて受取ることができる。

(5)　割　込　管　理

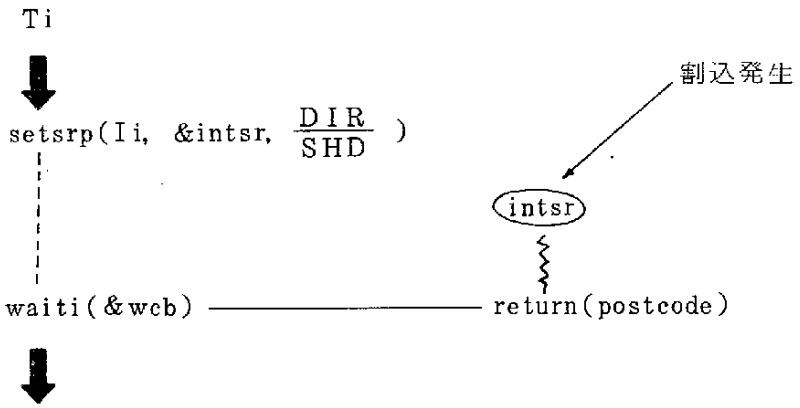
タスク実行中に割込処理ルーチンをセットして割込みをとらえる方法であり、制御用にはきわめて有効な手段である。

⑦　setsrf と waitf …… fpost での割込通知



- 割込処理ルーチンのリターンコードが fpost マクロになる。
- オプションにはDIR(最優先にてsetsrf発行元タスクに戻る)とSHD(タスクスケジュールを通ってタスクに戻る)がある。

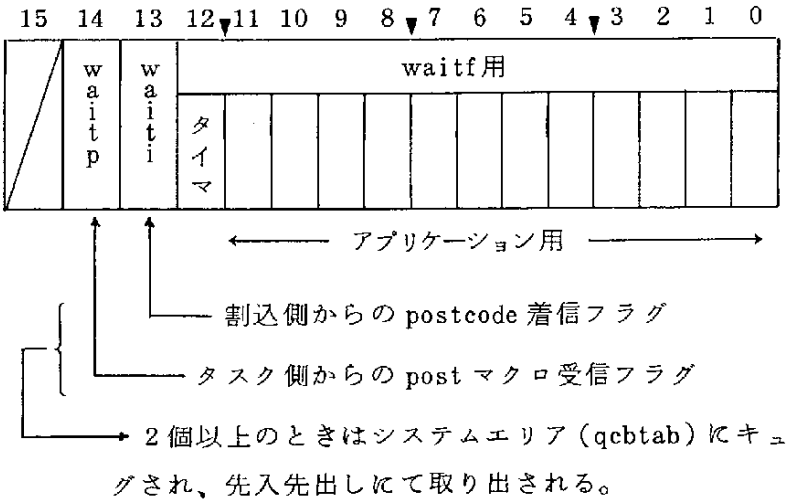
⑧　setsrp と waiti …… post での割込通知



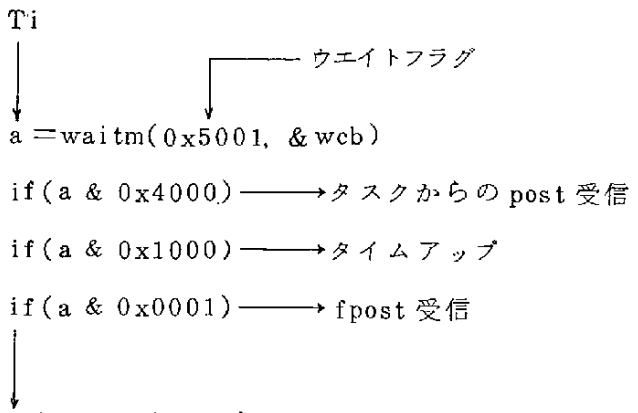
- postcodeをwcb.cd にて受取ることができ、何度も割込が発生したときは postcode が qcbtab（キューエリア）にキューイングされる。
- 割込処理ルーチンを解除したい（無効にする）ときは、setsrp(Ii, NULL, DIR/SHD）を発行する。ただし、post されている分はシステムキュー上に残っているため waiti にて取り出す必要はある。

(6)　複数事象の発生待と確認

ＴＣＢに確保されているポストフラグワードは次の形式をもっている。



⑨ waitm……複数事象待



- waitm(0x6000) のときは &wcb が 1 個しかないから、waitp が優先されwaiti が待たされる。

⑩ flag = rflag( ) …… ポストフラグワードの取出し
TCBのエリアは変化しない。

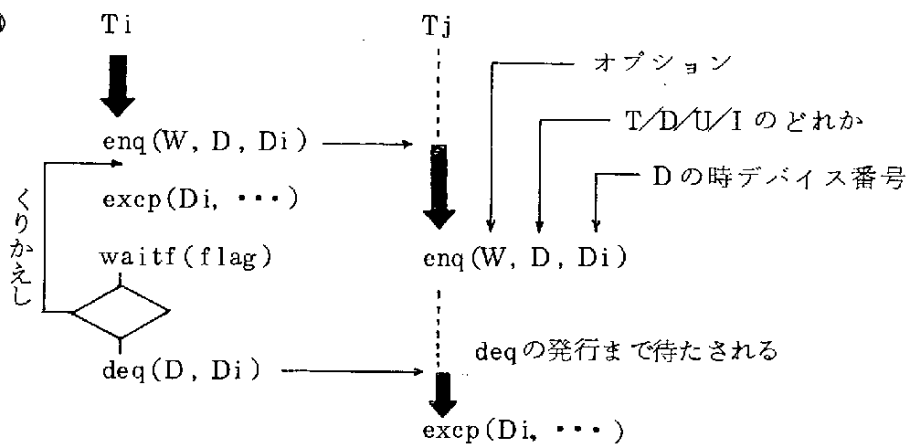
⑪ cflag(clearflag) …… ポストフラグワードの消しこみ
waitf用のみ消すことができる。

(7) リソース管理

TCB・DCB・UCB・ICBの中にはそれぞれRCB(resource control block)を持っている。このRCBを使って、タスク・デバイス・ユニットデバイス・インタラプト・その他のリソースを管理する。

⑫ enq と deq ……占有と解除

⑬



- オプションにはW（既に占有されている時は解除まで待つ）とR（占有されているときはリターンコードを１とする、占有した時は０とする）の２種がある。
- 上記の例はexcpがcrt 1行分の出力として、n行分他のタスクにじゃまされずに出力したい場合の例である。

(8) タスク管理

⑭ chap(Ti, priority) ……タスク優先順位の変更

⑮ next( ) ……自分のタスクをランチェインの最後に付ける。
ループ待をする時に他タスクに制御を渡すときに用いる。

⑯ exit( ) ……タスク制御の終了
CP/M86にあっては次のシステムコマンド受付けとなる。

⑰ attach ……タスクの生成

```
attach
attach (Ti, pri, code, &task, &stack)
                                  └ スタックエリアアドレス
                          └ タスク開始アドレス
                    └ attachコード
              └ プライオリティ（0～255）
        └ タスク番号
```

- 生成されたタスクには、attachコードと発行元のds（データセグメントアドレス）が通知される。
- 64KB空間外のタスクにはattachfがある。

```
attachf (Ti, pri, code, & task, cs, & stack, ds)
                        コードセグメントアドレス ┘
                                  データセグメントアドレス ┘
```

⑱ attach( )……自分のタスクの消滅

これにより発行したタスクTiのTCBは空となる。再度attachマクロを用いればTCBは埋まることとなる。

⑲ except(Di)……実行中のexcp動作の強制終了

異常検出タスクから発行するもので、実行中のexcp動作を強制終了させるときに用いる。このマクロにはwaitはない。

(9) OSマクロ一覧

- タスク間通信
  - ① fpost(Ti, postflag) …… waitf(waitflag)
  - ② post (Ti, postcode) …… waitp(&wcb)
- タイマ管理
  - ③ stime(t) …… waitf(0x1000)
  - ④ 残時間＝gtime( )
- 入出力管理
  - ⑤ excp(Di, postflag, &ccb) …… waitf(waitflag)
  - ⑥ setdi(Ii, SET/REP/RES) …… waiti(&wcb)
- 割込管理
  - ⑦ setsrf(Ii, &intsr, DIR/SHD) …… waitf(waitflag)
  - ⑧ setsrp(Ii, &intsr, DIR/SHD) …… waiti(&wcb)
- 複数事象発生待と確認
  - ⑨ waitm(waitflag, &wcb)
  - ⑩ flag＝rflag( )
  - ⑪ cflag(clearflag)
- リソース管理
  - ⑫ enq(W/R, T/D/U/I, Xi)
  - ⑬ deq(T/D/U/I, Xi)

- タスク管理

⑭ chap(Ti, pri)

⑮ next( )

⑯ exit( )

⑰ attach(Ti, pri, code, & task, & stack)

⑱ dttach( )

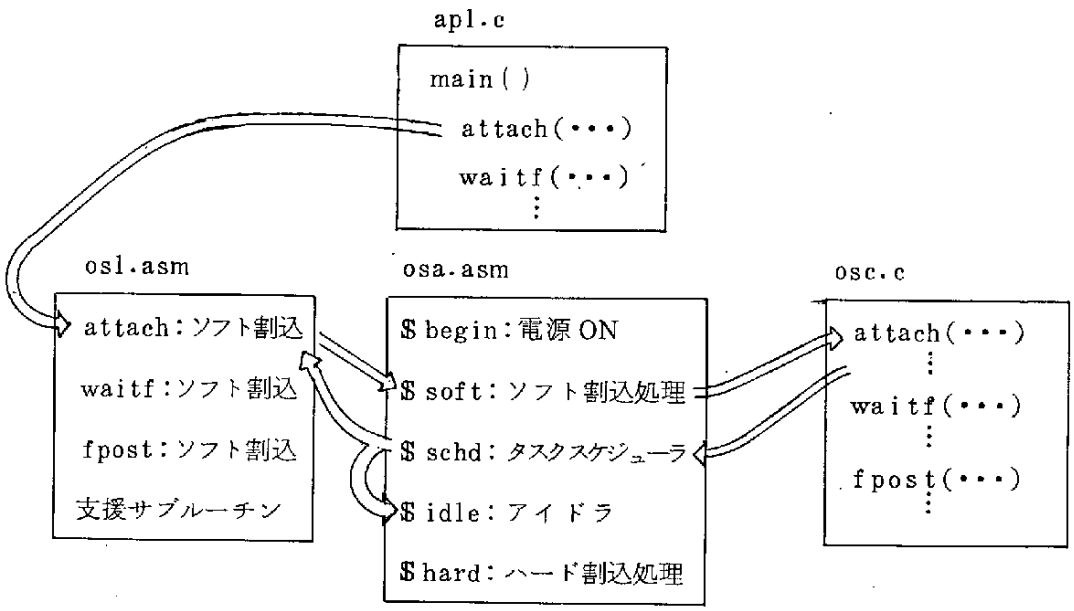
⑲ except(Di)

2.3 タスク制御カーネルプログラム

アッセンブリ言語(asm)のコーディングを最少におさえて、ほとんどをC言語にてコーディングした。その概要を示す。

(1) ソースファイル(プログラム言語)のからみ

カーネル実現のためにosc.c、osa.asm、osl.asmの3本のファイルがあり、アプリケーションをapl.cとすると次の様な構成となる。



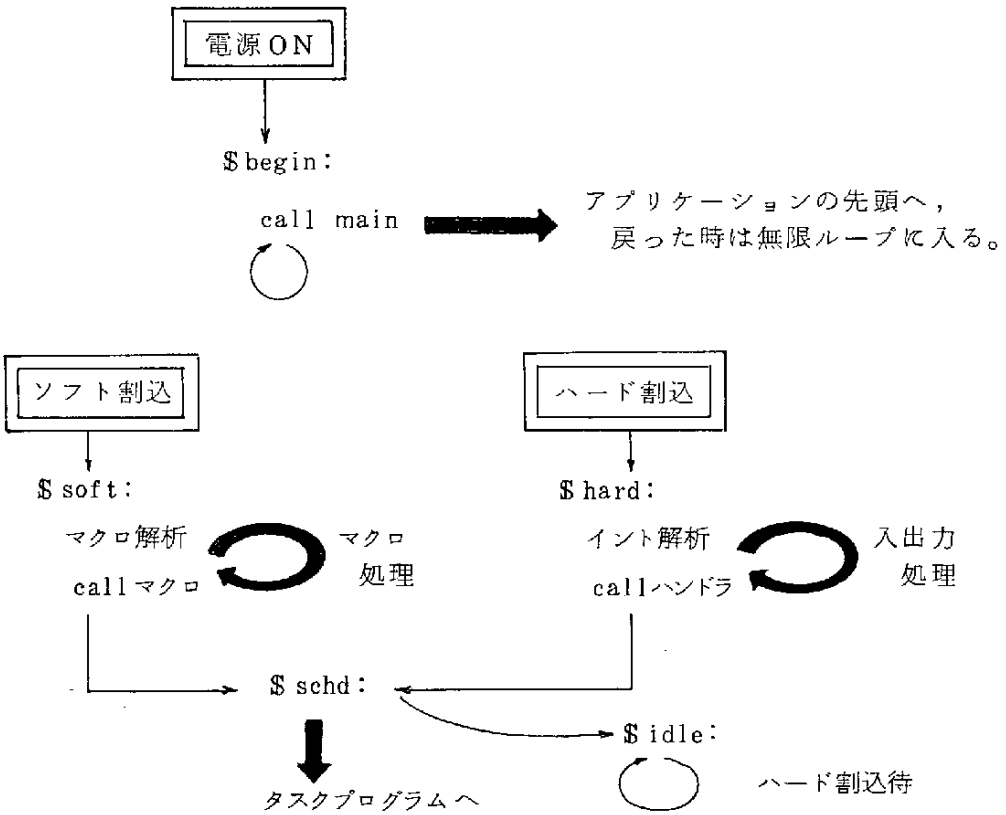
(注1) タスクスケジューラに入るとタスクのプライオリティに従って優先度の高いタスクに制御が渡る。上記のattachではコール元に戻る。

① osl.asm

asm言語の小さなサブルーチンの集まったコーディングであり、ＣＨレジスタにマクロコードを乗せ、ＣＬレジスタにマクロ引数個数を乗せてソフト割込255をかける役目をしているファイルである。

② osa.asm

制御の流れと8086レジスタの退避・回復を行うasm言語のコーディングであり次の流れとなる。



③ osc.c

osマクロ処理をするプログラムであり、tcbtab・dcbtab・ucbtab・icbtab・qcbtab等のosテーブルをすべてＣ言語プログラムにて処理している。

(2) 支援サブルーチン

① osl.asmに存在するサブルーチンでアプリケーションにて使えるもの

- blockmv(&recv, &send, length) …… 移　動
- clear(&area, length, value) …… クリア
- msti() …… 割込許可
- mcli() …… 割込禁止
- aputp(port, value) …… ポート出力
- value＝agetp(port) …… ポート入力
- cs＝getcs() …… code segmentの取出し
- ds＝getds() …… data segmentの取出し
- byte＝memrb(seg, offset) …… バイト取出し
- word＝memrw(seg, offset) …… ワード取出し
- memwb(seg, offset, byte) …… バイト書き込み
- memww(seg, offset, word) …… ワード書き込み

(注) osl.asmは常にアプリケーションとリンケージされるため、OS存在64KB空間外で自由に使えるものである。

② osc.cに存在するサブルーチンでハンドラが使えるもの

- mmstime(T/D/U, &tcb, time) …… タイマーセット
- mmrtime(T/D/U, &tcb) …… タイマーリセット
- 残り時間＝mmgtime(T/D/U, &tcb) …… 残タイム取出し

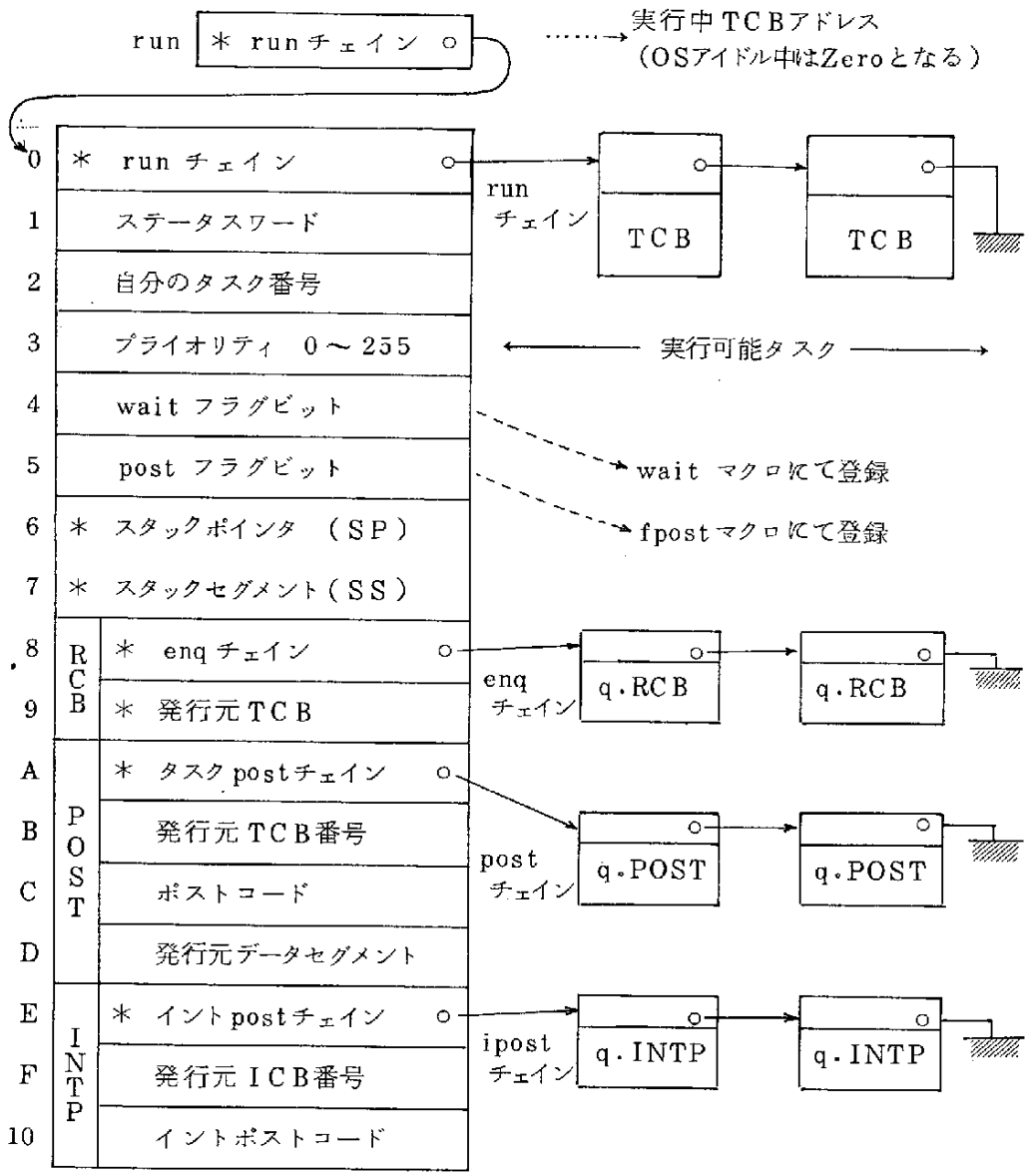
(注) osc.cの中であるため、os存在64KB空間にて使用すること。

2.4 タスク制御カーネルのテーブル

タスク制御と入出力制御を管理しているＯＳテーブルの詳細を記す。

Ｃ言語のstruct定義はこのようなテーブルの記述にはきわめてすぐれている。

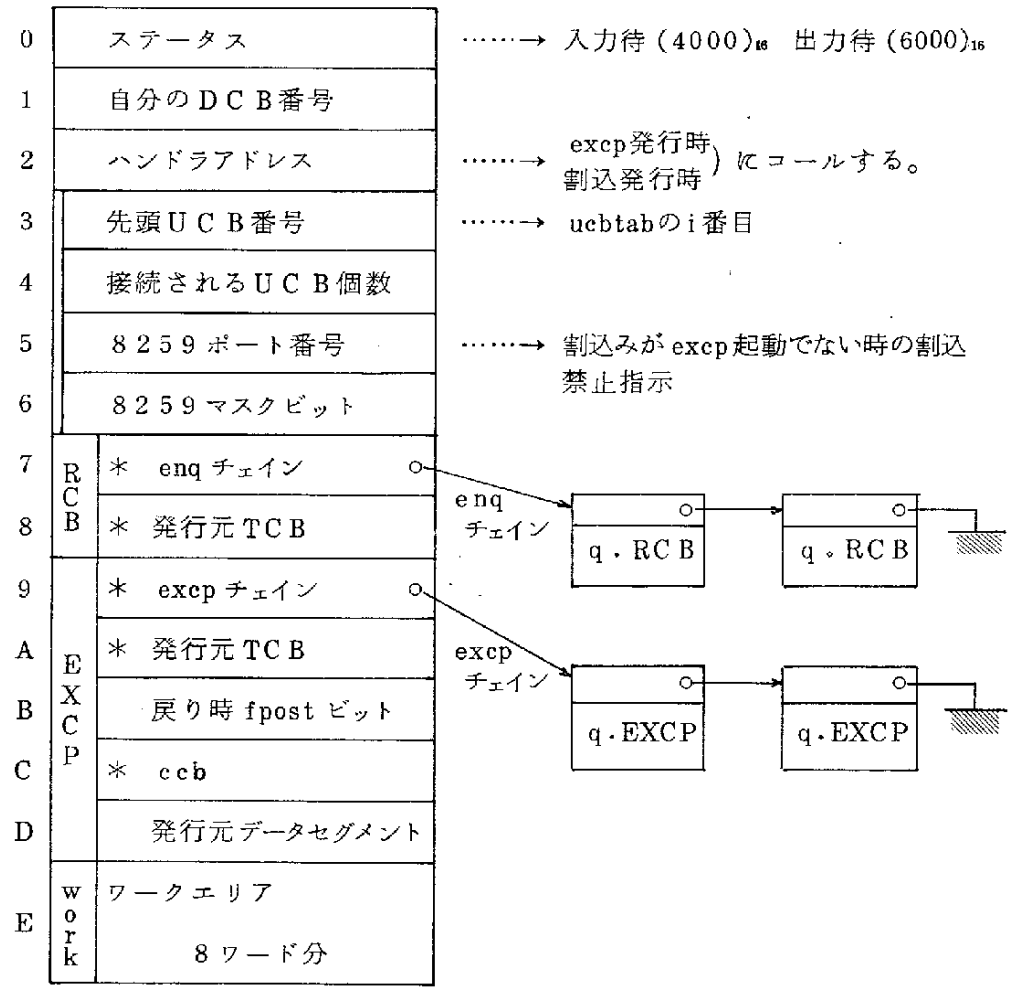
(1) ＴＣＢ (Task Control Block) ····· tcbtab(i)



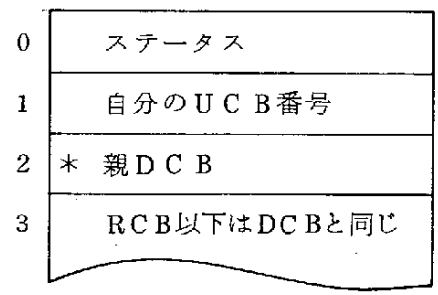
（注１） ＊印はアドレスを示す。

（注２） ＲＣＢ、ＰＯＳＴ、ＩＮＴＰ先入先出方式を取る。

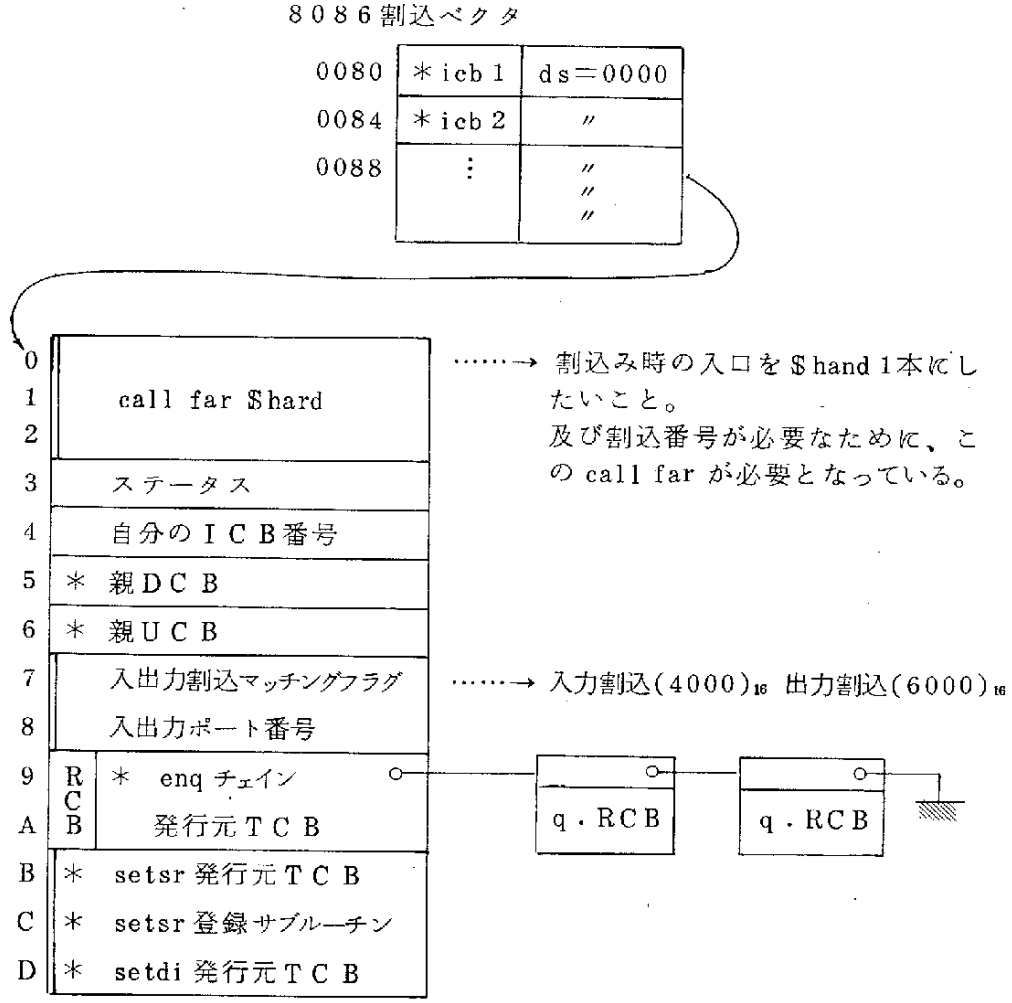
(2) ＤＣＢ（Device Control Block） …… dcbtab(i)



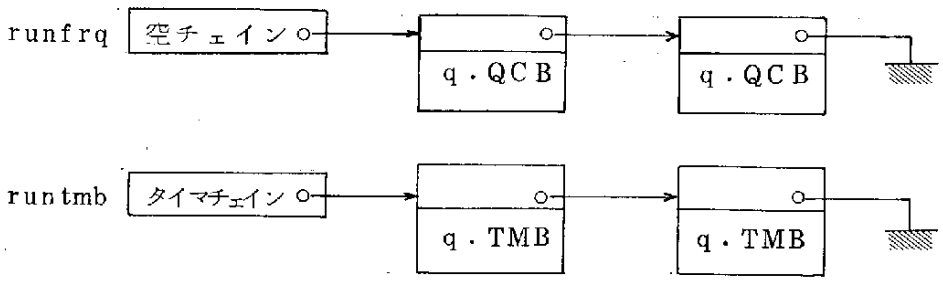
(3) ＵＣＢ（Unit Control Block） …… ucbtab(i)



(4) ＩＣＢ（Interrupt Control Block）…… icbtab(i)



(5) ＱＣＢ（Queue Control Block）…… qcbtab(i)

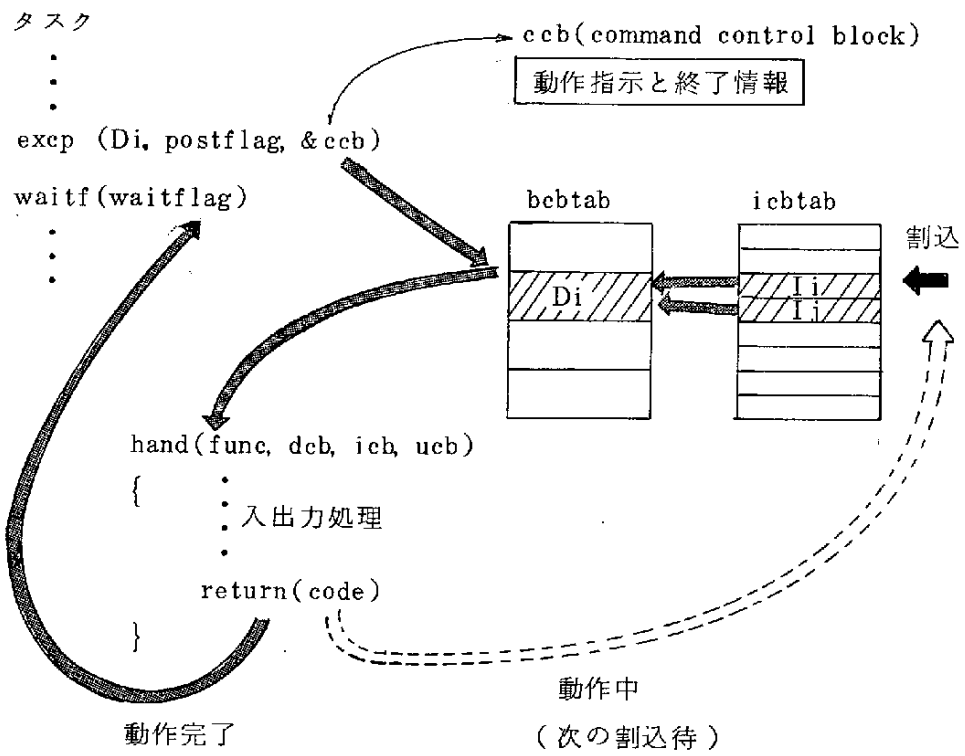


(注) enq, post, ipost, excpはqcbtab(i)に格納される。

3. 入出力ハンドラ仕様

excp マクロによって起動がかかり、ハード割込みによって動作が進行する入出力処理プログラム（ハンドラという）の概要を説明する。

(1) excpマクロとハンドラの位置



- ハンドラの開始アドレスはＤＣＢの中に定義されている。
- 割込とデバイスの接続はＩＣＢの中で定義されている。
- ccbのフォーマットは各種装置によって異なってよく、ハンドラの性質によって決定してよい。
- ハンドラはまずexcpマクロによって示されるccb内容に従って装置に初期動作を指示する。必要ならばccbの内容を自分のＤＣＢ中のワークエリアに取り込み、以後の割込動作の進行にそなえる。リターンcodeは重要であり、進行中（code≠0）か、動作完了（code＝0）の区分に使用される。

(2) ハンドラの形状と処理内容

ＯＳから処理要求コードをfuncに乗せて、ハンドラをコールするのでその要求にあった処理を行う。Ｃ言語のフォーマットにて示す。

```
hand(func, dcb, icb, ucb)
int func;
struct DCB  *dcb;
struct ICB  *icb;
struct UCB  *ucb;
{
    switch(func) {
      case 1：電源投入時（dcb参照）……
                                        }装置の初期設定
      case 2：電源投入時（icb参照）……
      case 3： excp発行時 …… 動作開始処理
                入力割込要求のとき return(4000)にて戻す
                出力割込要求のとき return(6000)にて戻す
      case 4：割 込 発 生 …… 動作続行か動作終了か
                動作終了のとき return（００００）にて戻す
                次の割込期待のとき return（4000 or 6000）
      case 5：タイムアップ …… 異常終了
                case 3、４の時に設定したタイムのオーバー
      case 6： excpt発行時 …… 強制終了
      case 7： setdi発行中の割込発生
                return(code)にpostcodeを乗せる。
  }
}
```

㈱ 通常のプログラムと異なり、１回のexcpマクロと多数回の割込によってハンドラーは動かされるため、プログラムの組み方には充分注意が必要である。

3.1　コンソールハンドラ

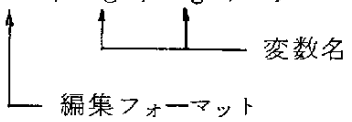
システムコンソールとなるcrtの入出力処理を行うもので、ロジカルベースとフィジカルベースの2段階のハンドラを用意している。

ロジカルハンドラはwaitfマクロを内蔵するリエントラントサブルーチンであり、コールしたタスクのスタック上にてexcpマクロを用いてフィジカルハンドラを起動している。

(1)　ロジカルハンドラ

128バイトの入出力バッファを用意して、コンソールdcbを占有(enq, deqマクロ発行)し、タスク間の干渉をなくして処理している。

①　printf("fmt", arg1, arg2, …)……編集出力

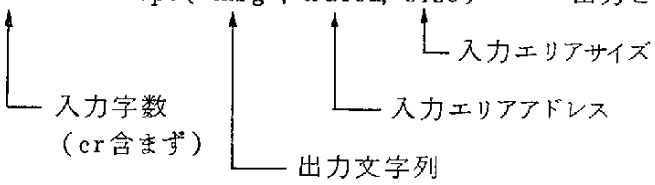


（注1）　B.W.カーニハン仕様と同一形式である。

（注2）　AZTEC　Cの標準ライブラリを入出力部分について改造した。

（注3）　スタックとして約500バイトを使用している。

②　size = accept("msg", &area, size)……出力と入力



（注1）　入力エリアが満杯になるかCr／Lfが入力された時点にて動作を終了する。

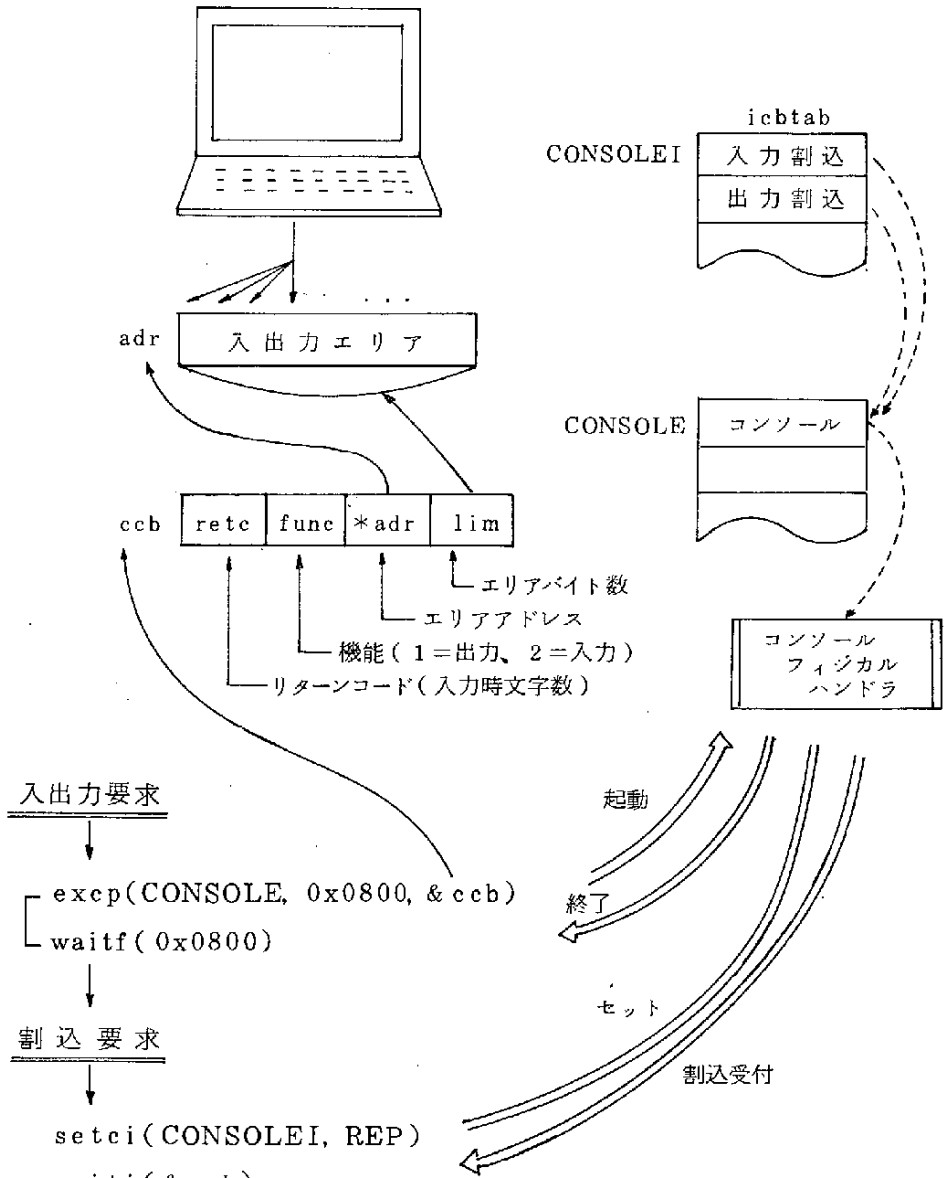
（注2）　Cr/Lfストップの時にはメモリに$(00)_{16}$バイトを設定する。

（注3）　msgは省略可能であり、その時は入力処理のみとなる。

（注4）　&areaも省略可能である。当然出力処理のみとなる。

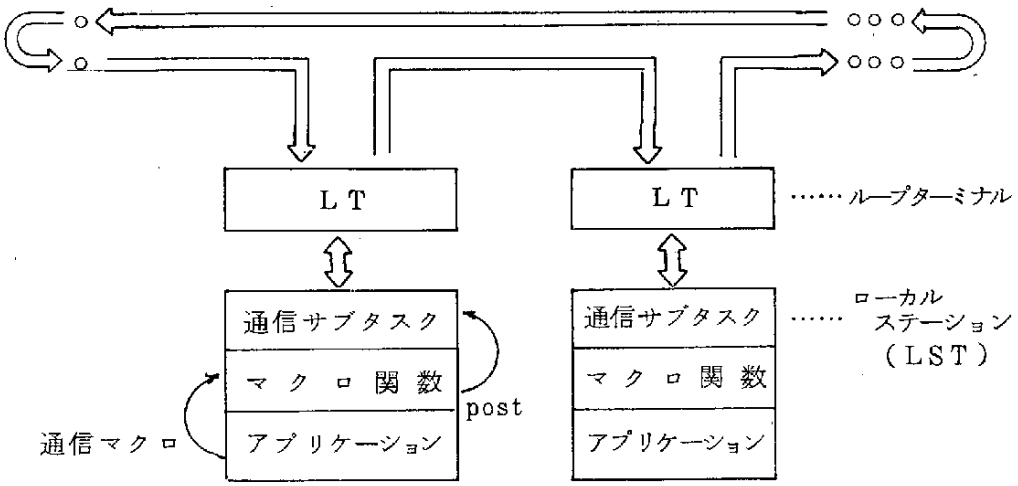
(2) フィジカルハンドラ

excpマクロにて起動がかかり、crtの入力割込と出力割込をとらえて文字列の処理を行う。又、キーイン割込みがとらえるようにsetdiマクロもサポートしている。



3.2　ループネットワーク通信ハンドラ

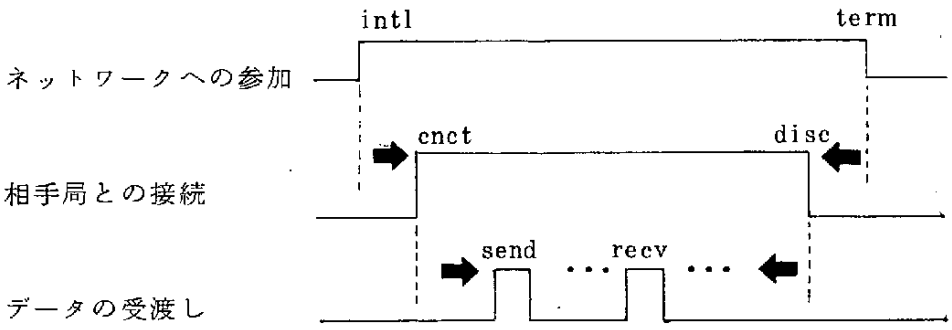
ループネットワーク端局間の通信処理を行うもので、ロジカルベースのマクロ関数と通信プロトコル処理を行うフィジカルベースのサブタスクがある。



ロジカルベースのマクロ関数はwaitpマクロを内蔵するリエントラントサブルーチンであり、通信マクロの指示を通信サブタスクに通知する。

通信サブタスクはそのマクロを受けてＬＴへ伝文フレームを出力したり、相手局からの伝文フレームを割込にてＬＴから受取る処理を行う。

なお、ループネットワークにおいては厳密に次のような３階層の通信プロトコルが存在する。



(注)　ネットワーク管理伝文はintlとtermの間にて許される。

(1) 通信マクロ …… ロジカルベース・ハンドラマクロ

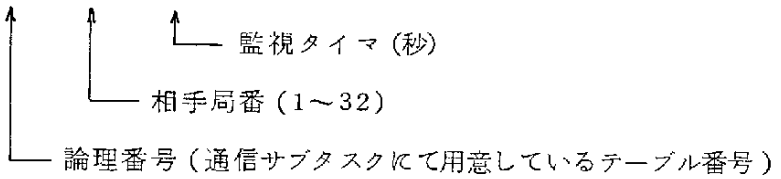
- ループネットワークへの参加と離脱

① nintl( ) …… 自局LSTをオンラインにする

② nterm( ) …… 自局LSTをオフラインにする

- 相手局との接続要求と接続断

③ ncnct(ln, targ, time) …… 相手局との接続

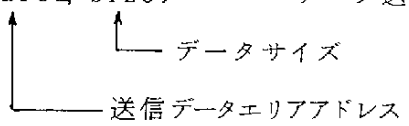


④ ndise(ln, time) …… 相手局との接続断

⑤ targ=ncnctr(ln, time) …… 相手局から接続要求待

- データの送受信

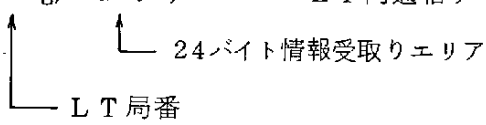
⑥ nsend(ln, time, &area, size) …… データ送信



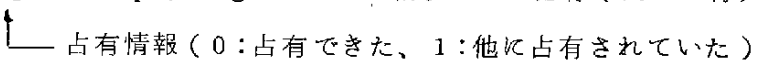
⑦ size=nrecv(ln, time, &area, size) …… データ受信

- ネットワーク管理伝文

⑧ nhealth(targ, &area) …… LT内通信テーブルの取出し



⑨ sign=nenqr(targ) …… 相手LTの占有(リターン付)



⑩ nenqw(targ) …… 相手LTの占有(ウエイト付)

⑪ ndeq (targ) …… 相手LTの占有解除

⑫ nwrite(targ, data) ……相手ＬＴコモンエリアへの書き込み

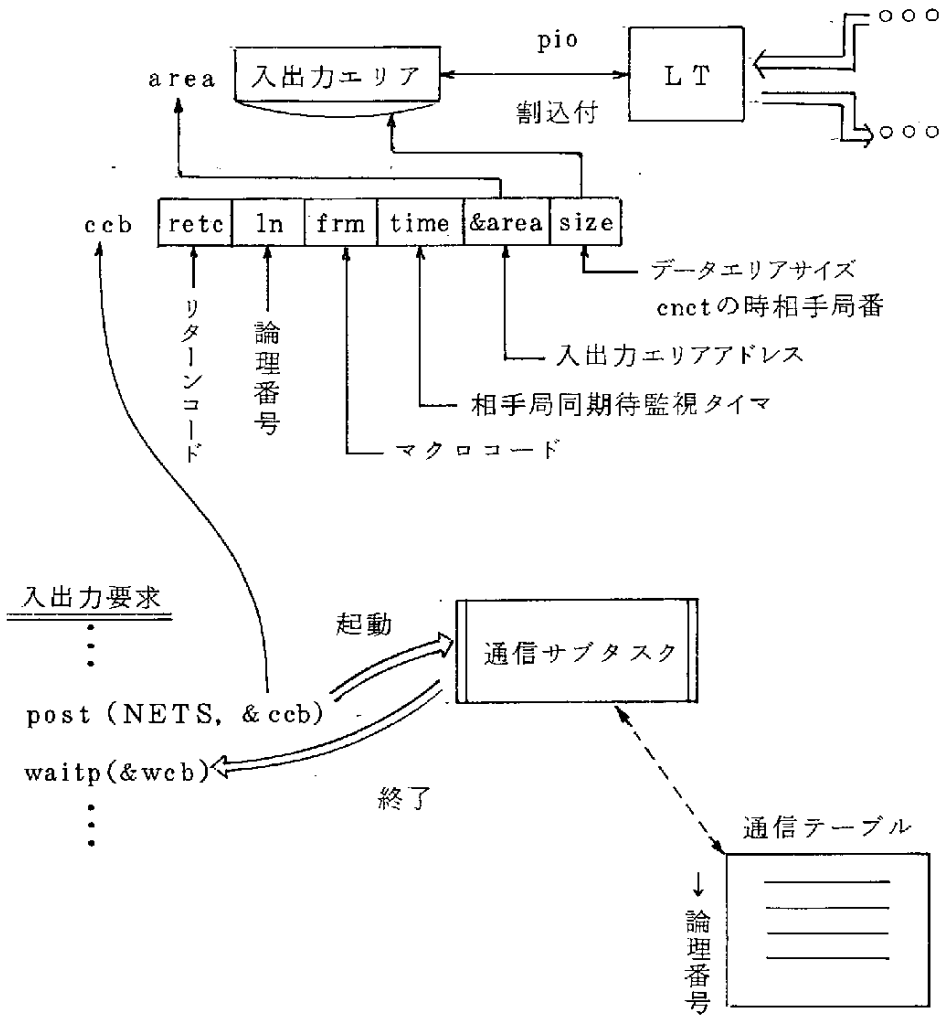
└ 1バイトデータ

⑬ data＝nread(targ) ……自局ＬＴコモンエリアからの取出し

└ 1バイトデータ

(2) 通信サブタスク …… フィジカルベース・ハンドラ

通信マクロ関数およびアプリケーションからのpostマクロによって起動され、ＬＳＴ−ＬＴ間の通信を行う。

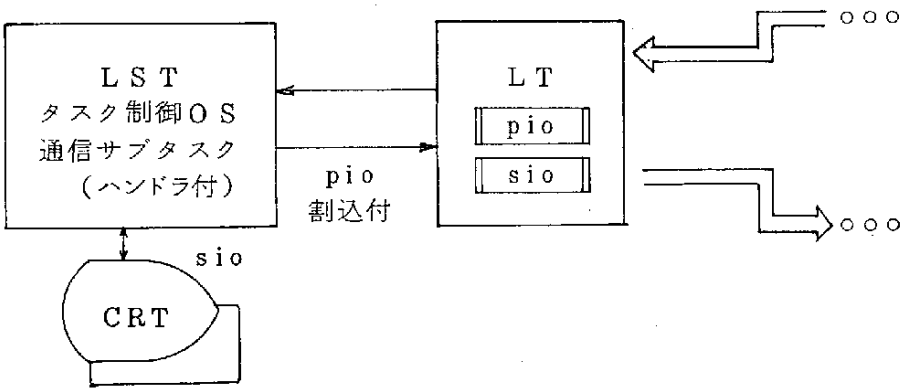


(注) 通信テーブルには相手局番・親タスク番号・ccbアドレス・監視タイマ等が設定され、ＬＴからの割込み、伝文受取、タイマ割込によって保守される。

3.3　ループネットワークＬＳＴ－ＬＴプロトコル

５７年度開発"光ファイバを用いた簡易型ローカルエリアネットワークシステム"にあってはＬＳＴとＬＴはメモリシェアされており、ＬＳＴからＬＴへの割込は存在したが、ＬＴからＬＳＴへの割込みは存在していなかった。

今回はポート結合（pio及びsio通信）のため双方向の割込みは必須であり、かつネットワーク管理用の伝文も扱うこととなったため、５７年度開発プログラムを解読し修正を加えて用いている。



(1)　ＬＴとの接続・接続断

①	intl	10	00	タイマ	⇄	11	00	er
②	term	60	00	00 00	⇄	61	00	er

(2)　ネットワーク管理伝文

③	health	70	s	00	⇄	71	s	テーブルデータ（24B）
④	enqr	90	s	00	⇄	91	s	f	f < 0：yes / 1：no
⑤	deq	A0	s	00	⇄	A1	s	00
⑥	com_read	B0	s	00	⇄	B1	s	d
⑦	com_write	C0	s	d	⇄	C1	s	00

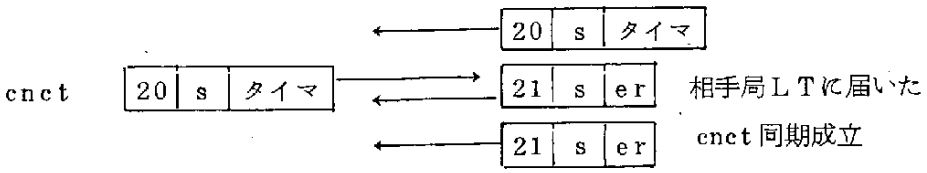
s：相手局番

d：データ

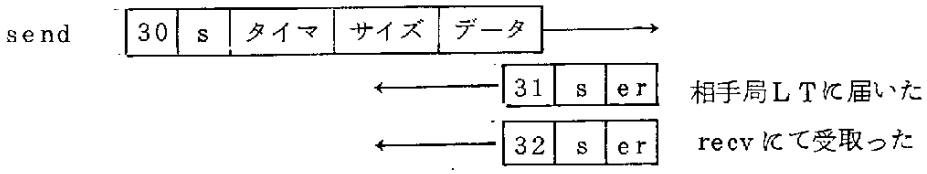
er：エラーコード(00のときOK)

(3) データ通信伝文

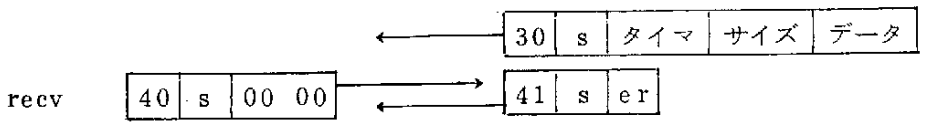
⑨ 相手局との接続



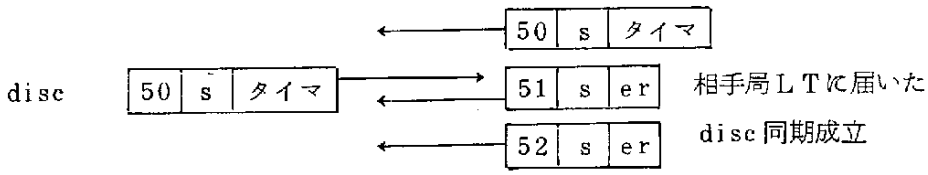
⑩ データ送信



⑪ データ受信



⑫ 相手局との接続断



4. オンライン・デバッガ

(1) 特　　長

① デバッガはオンライン・リアルタイムモニタの一部として存在し、ユーザー側からは必要都度デバッガモードに入ることができる。

② デバック対象プログラムにデバック用コードを付加するなど、デバッガ起動を前提した作業は不要である。

③ テーブル形メモリーダンプ、関数シミュレーションなどC言語プログラムデバックを対象としたコマンドを用意している。

④ データ・ベース局とのデータ送受信などＬＡＮの特性を生かしたコマンドを用意している。

(2) デバッガコマンド群

	コマンド	内　　容
1	d	メモリーダンプ
2	p	メモリーパッチ
3	r	メモリークリア
4	c	コマンドカタログ関係
5	f	関数実行関係
6	p	疑似ポートＩ／Ｏ関係
7	p	実ポートＩ／Ｏ関係
8	it	イント発生
9	post	タスク・ポスト
10	fpst	タスク・ポスト（フラグ）
11	load	データ・ロード
12	save	データ・セーブ
13	run	プログラムのロードと実行
14	•	デバッガの停止

(3) デバッガコマンドの基本項目

① デバッガの起動

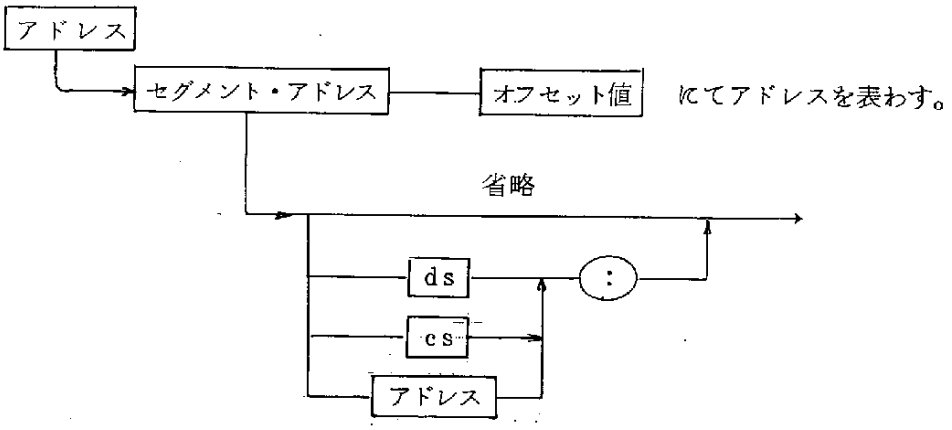
ＥＳＣ　キー押下（ＣＲＴの場合）

CTRL ― Ｅ押下　（パソコンの場合）

② コマンド受付けプロンプト

＝ がデバッガのプロンプトであり、この状態でコマンド入力が可能

③ アドレス入力の共通フォーマット



省略 ＝ データ・セグメント
ds ＝ データ・セグメント
cs ＝ コード・セグメント

④ 入 力 形 式

- アドレス関係、ポートＩ／Ｏデータ等は16進表示形式にて入力
- 行、列番号は10進数にて入力
- ロード、セーブのアルファ名、関数シミュレーションのデータセットはキャラクタ形式にて入力

(4) デバッガコマンド説明

① シリアル形式メモリーダンプ

＝ d, アドレス, $(\frac{b}{w})$

(b：バイト単位 / w：ワード単位)

次に

＝ dにより、次のアドレスからメモリーダンプを継続する。

② テーブル形成メモリーダンプ

＝ d, [アドレス], ($\frac{b}{w}$), ($\frac{h}{v}$), [行数], [列数]

[列数] ← 1行当りの項目数

[行数] ← 表示すべき行数

h：横型（1次元テーブルなど）

v：縦型（2次元テーブル、構造体など）

③ メモリーパッチ

＝ p, [アドレス], ($\frac{b}{w}$)

⇩

アドレス　××−××−××−××−××

⇩ ← 現在の内容を5個分表示しパッチ可能となる

××−××−××

⇩ ← 表示形式と同一フォーマットでパッチデータを入力

…… ××−

← パッチデータの最後に付けることで次のアドレスにパッチがかけられる。

④ メモリ・クリア

＝ r, [アドレス], [終了オフセット], ××

×× ← バイト単位のクリアパターン

アドレスから終了オフセットまでをクリアパターンで埋める。

⑤ コマンド・カタログ関係（ダンプコマンドのカタログ）

＝ c, この後はメモリダンプコマンドに準ずる。

← カタログに登録（最大5コマンドまで）

＝ cg

└── カタログ済コマンドの実行

＝ cp

└── カタログ内容の表示

＝ ce

└── カタログの消却

⑥ 関数実行前データセット

＝ fs, (＊i / ＊c) (1/2), [セットデータ]

[セットデータ] ── 関数実行前データ

(1/2) ── ワークエリア番号

(＊i / ＊c) ── ＊i：整数型、＊c：文字ストリング

○ 関数の引数がデータ・アドレスを指す場合、あらかじめfsコマンドによってデータを設定する。

⑦ 関 数 実 行

＝ f, [関数アドレス] (引数1, 引数2, ……引数8)

(引数1, 引数2, ……引数8) ── 16進形式にて

(引数1, 引数2, ……引数8) ── fsコマンドでセットしたデータ・アドレスを表わす時は、＊i1～2、＊c1～2で引数を入力。

[関数アドレス] ── 実行する関数のアドレス（オフセット値）

Return ＝××××

Return ＝×××× ── 関数からのリターン情報を表示する。

⑧ 疑似ポート I／O関係

16個のポートシミュレーションテーブルを用いて、ポートI／Oテストを行う。アプリケーション・プログラムでは、下記のポートI／O関数によりシミュレーションが可能となる。

portget （ポート番号）
　└ポートからの取り出し
portput （ポート番号、データ）
　└ポートへのデータ書き込み

＝ pw, [ポート番号], [データ]
　　└ポートに書き込むワード・データ
　└ポート・テーブルに登録

＝ Pr, [ポート番号]
　└ポート・テーブルから該当するポート・データを取り出し表示する。

＝ pr
　└ポート・テーブルに登録済みの全てのポート・データを表示

＝ pd, [ポート番号]
　└ポート・テーブルより該当するポートを消却する。

⑨ 実ポートI／O関係

＝ pi, [ポート番号]
　└実ポートよりデータを取り出し表示する。

＝ po, [ポート番号], [データ]
　└実ポートにデータを書き込む。

⑩ イントシミュレーション

＝ it, [イント番号]
　　└20～27
　└イントの20～27を発生させる。

⑪ タスク・ポスト

＝ post, [タスク番号], [ポスト・コード・アドレス]
　└該当タスクにポストをかける。（ポスト・コードタイプ）

⑫ フラグポスト

＝ fpst, [タスク番号], [ポストフラグ]
　└該当タスクにポストをかける。（ポスト・フラグタイプ）

⑬ データ・ロード

＝ load, [アドレス], [ファイル名]

- [ファイル名] ― ファイル・サーバに登録した名称
- [アドレス] ― ロード開始アドレス
- load ― データ・ベース局のファイル・サーバよりデータをメモリにロードする。

⑭ データ・セーブ

＝ save, [アドレス], [終了アドレス], [ファイル名]

- [終了アドレス] ― セーブ終了アドレス（オフセット値）
- [アドレス] ― セーブ開始アドレス
- save ― データ・ベース局のファイル・サーバにメモリデータをセーブする。

⑮ プログラムのロードと実行

＝ run, [アドレス], [ファイル名]

- [アドレス] ― ロード開始アドレス（オフセット値）
- run ― ファイル・サーバよりプログラムをロードし、かつ実行する。

⑯ デバッガの停止

＝ .

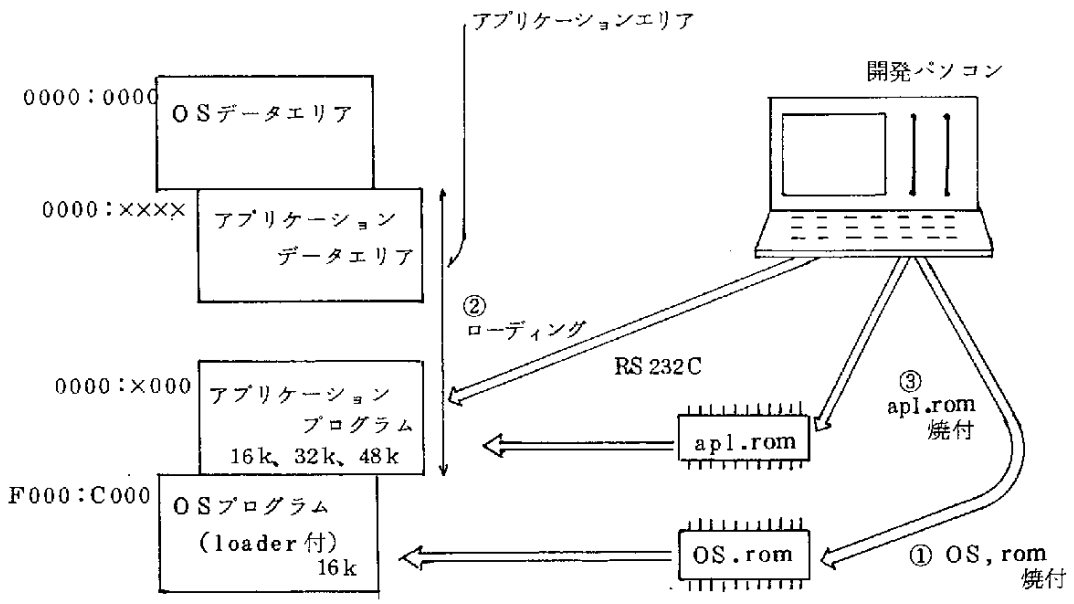
- . ― デバッガ・モードを終了する。

5. OS .romとアプリケーション. romの作成

ＯＳ開発段階ではアプリケーション（実際はテストプログラム）とＯＳが合体されてロードモジュール（rom形式）が作られると非常に助かる。しかし、入出力ハンドラを含めたＯＳが固定化できた後は、アプリケーション側のみコンパイルしてディバックできるようにした方が操作面では非常に楽である。

ＯＳにパソコンからのプログラムローダ（loader）を乗せ、ＩＣＥ（インサーキットエミュレータ）のram、又はターゲットマシンのram上にて、ディバッキングを行ない、その後アプリケーションをrom にする方法を説明する。

(1)　rom化手順とターゲットマシンメモリ

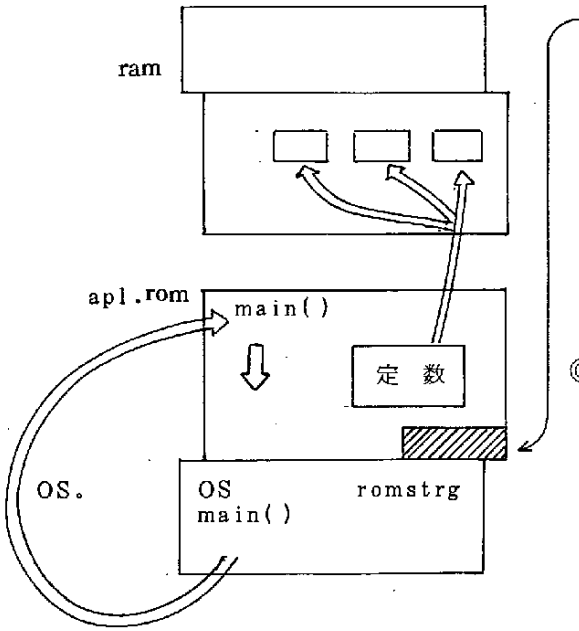


（注１）　PROM 2764 は８KB であるが 8086 にあっては奇数バイト用、偶数バイト用となるため最少２個、16KB のモジュールが最少となる。

（注２）　データ及びプログラムのロードポイント指定にはリンカ（ln）の－D、－Ｃオプションを用いる。

(2) OS.romからapl.romへの制御の渡し

Cコンパイラの出力オブジェクトでは文字定数・初期値がデータエリア空間(ram)に設定されるため、romベースの制御用システムでは使えない。そこで我々は"romファイル作成ユティリティ"を用意してプログラム空間に定数・初期値を持つようにし、romstrg関数によって復元している。



プログラム空間最後の16バイト

①プログラム開始アドレス

②定数テーブルの先頭

③データエリアのセグメントアドレス

④ 〃 オフセットアドレス

⑤ 〃 バイト数

◎ romstrg関数では上記情報を使ってメモリをクリアし、データエリアに定数を配置し、開始アドレスを持って戻る。

プログラムローダを持たないOS.romのmain列

```
 1:  main()
 2:  {
 3:          int (*prog)();
 4:          struct WCB wa;
 5:
 6:          mcli();
 7:          prog = romstrg(0xf000, 0xbff0);
 8:          msti();
 9:          (*prog)();                  /* apl.main call */
10:          exit();
11:
12:          /* dummy generation module */
13:          printf();
14:          accept();
15:  }
```

(3) OS.romを構成するソースプログラムモジュール

	ソース名	モジュール概要	
①	aplstr.c	定数定義	ソースファイルに#includeされる。
②	str.c(aplstr)	OSテーブル定義	
③	osc.c(str)	OSカーネルC言語プログラム	
④	osa.a	電源投入、ソフトハード割込処理	
⑤	osl.a	OSマクロの割込リンク	
⑥	osp.c(aplstr)	printfモジュール	
⑦	osscst.c(aplstr)	acceptモジュール	
⑧	toupper.c	文字属性判定ルーチン (AZTEC版)	
⑨	lsubs.a	32ビット演算ルーチン (AZTEC版)	
⑩	string.c	文字列処理(AZTEC版)	
⑪	hmcc2.c(str)	dcb, icb定義とコンソールハンドラ	
⑫	nets.c(str)	ループネットワーク通信サブタスク	
⑬	loader(aplstr)	パソコンからのプログラムローダ	
⑭	osrom.c	アプリケーション.romへの連結	

(注1) ( )内は#includeファイルを表わす。

(注2) デバッガはOS空間16KBに入らないため除いている。

(注3) ⑥⑦⑧⑨の関数アドレスをアプリケーションプログラムに知らせると、アプリケーション側にて自由に使える。

(注4) ×××.cはC言語プログラム、×××.aはASM言語プログラムを示す。

(4) apl.romを構成するモジュール

os.romにて入出力ハンドラ(prntf・accept含む)を用意しているために、アプリケーションに焦点をしぼったプログラム構成ができる。

	ソース名	モジュール処理
①	aplstr.c(coname)	定数定義
②	osname.c	OS.romの関数アドレス
③	osl.a	OSマクロの割込リンク
④	osd.c(aplstr)	デバッガ
⑤	×××.c(aplstr)	mainの存在するアプリケーション

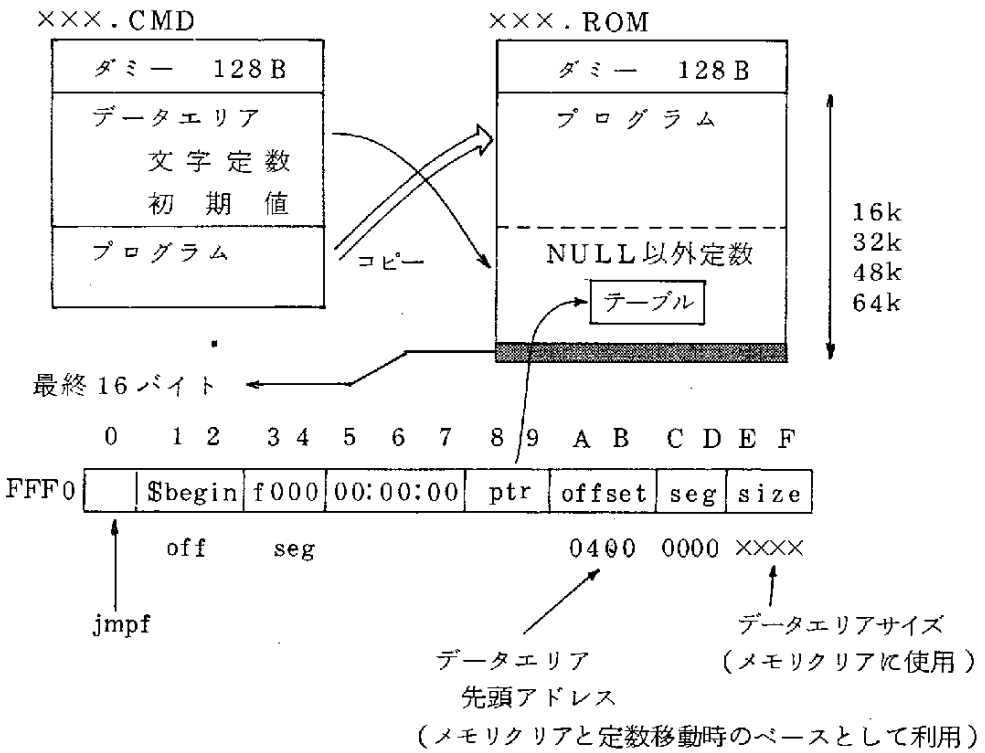
（注1） osl.aは本来osnameにて定義できるものであるが、64KB以外の時にはソフト割込をおこさせるために必須のものである。

6. ユティリティ

本リアルタイムモニタを作りながら作成したユティリティでアプリケーション開発に不可欠のものを紹介する。

(1) romファイルの作成 …… C言語

Cコンパイラ出力のロードモジュール（×××.CMD）ファイルからデータエリアの定数をプログラムエリアに移して、romへ書き込みできる×××.ROMファイルを作成する。



（注1） テーブルには、rom.adr, rom.adr, word.size が格納される。

（注2） NULL以外とは、4バイト連続NULL以外ということである。

（注3） 8086では電源投入によってF000セグメントFFF0アドレスより命令が実行される。上記 jmpf 命令はrom上必須である。

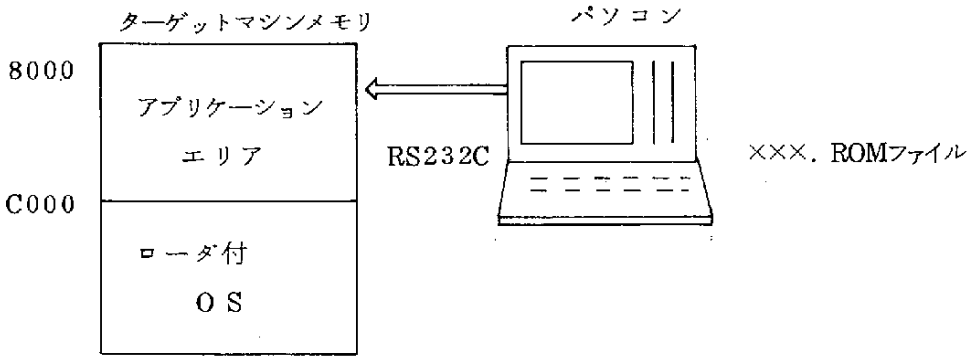
（注4） $beginはOSの入口を指し、そこでメモリクリアと定数移動を行う。

(2) romへのプログラム焼き込み …… BASIC言語

×××.ROMファイルよりPECKERβ（アバールコーポレーション製）を用いてromに焼く。PROM IC2764は8KBのメモリを持つが左バイト、右バイトと別々になっているため、プログラム空間としては16KB単位となり、最低2個は焼くこととなる。

(3) ターゲットマシンへのプログラム転送 …… BASIC言語

ターゲットマシンにローダ内蔵のOS.romがあるときにパソコン側より、アプリケーションプログラムを送る場合に用いる。



(4) ラベルの名前順・番地順リスト作成……C言語

Cコンパイラのリンカは×××.symファイルに、前順のそれも1列のベタリストしか出力しない。これではデバッグの役に立たないため用紙全面を使う名前順・番地順のラベルリストを作成する。

(5) プログラムリストの精書 …… CP/M, C言語

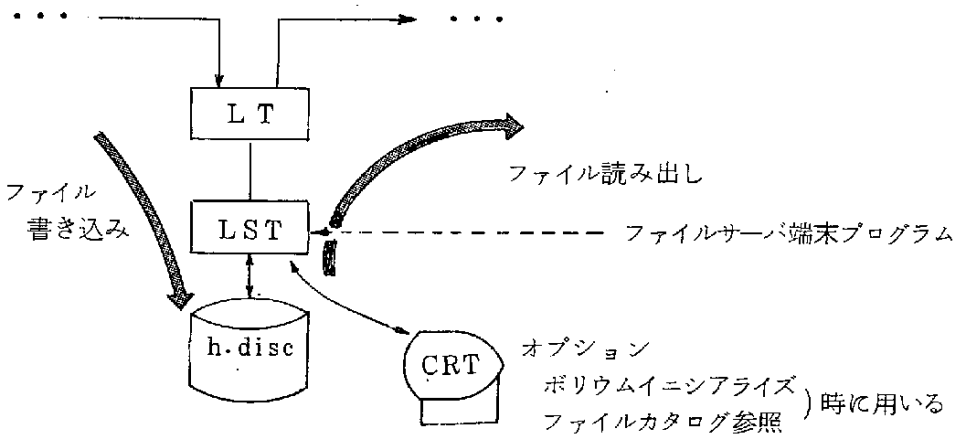
関数とデータのクロスリファレンスリストを付けてプログラムリストを作成する。プログラム保存にはきわめて有益となる。

(6) パソコンのcrt代用 …… BASIC言語

ターゲットマシンに付けるcrtが無い場合に、パソコンそのものがcrtの代行をするパソコン側のプログラムである。

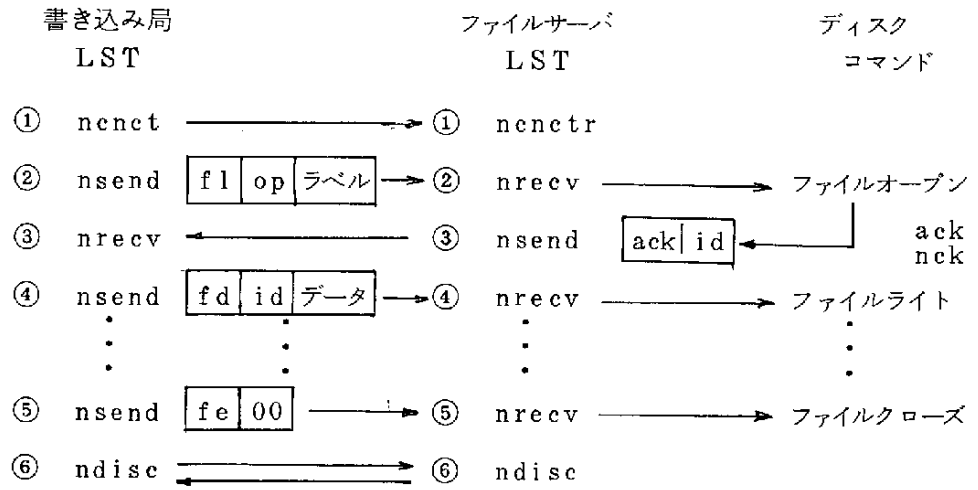
7. ファイルサーバ端局プログラム

ファイルサーバはネットワーク全体のファイルを管理しようとするものであるがデータの書き込み、データの読み出しはネットワークの各端局が主導権を握るため、ファイルサーバ側のプログラムはきわめて受動的なものとなる。



(1) ファイル受信プロトコル

通信マクロとデータフォーマットを下記に示す。

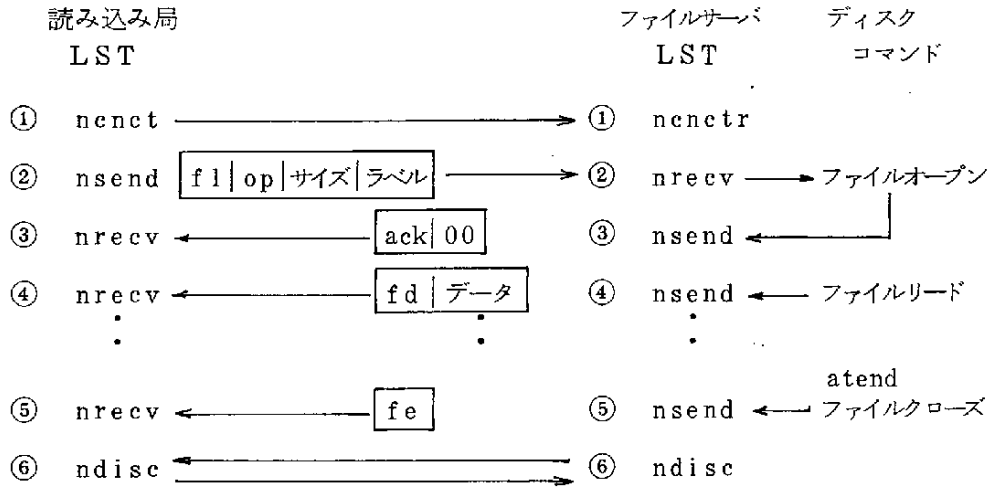


（注1） ファイルオープン時にファイルが存在すればディリートし、新しいファイルを作成する。

（注2） fl, fd, fe, ackは1バイトの定数を示す。

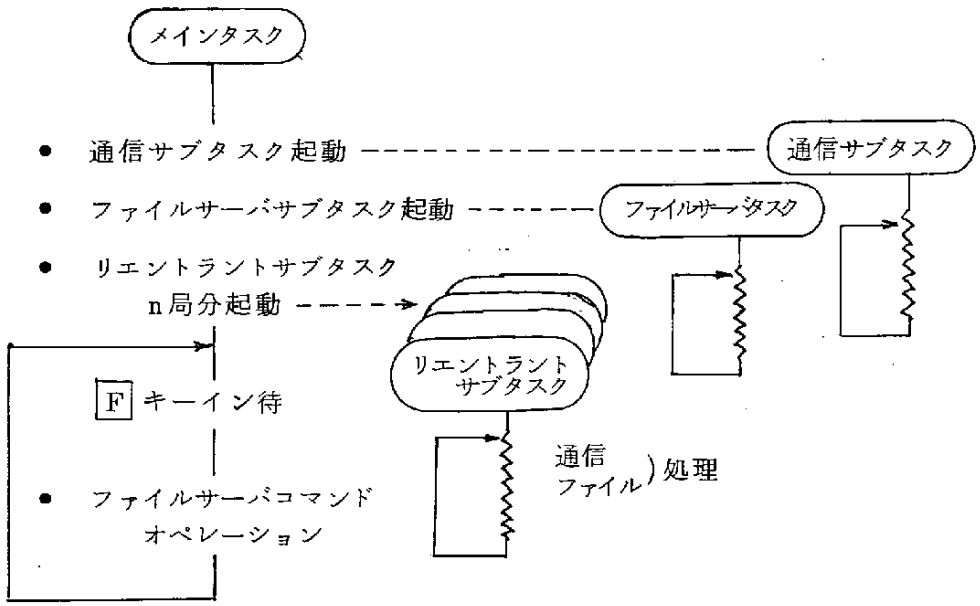
(2) ファイル送信プロトコル

通信マクロとデータフォーマットを下記に示す。



(3) プログラム概要

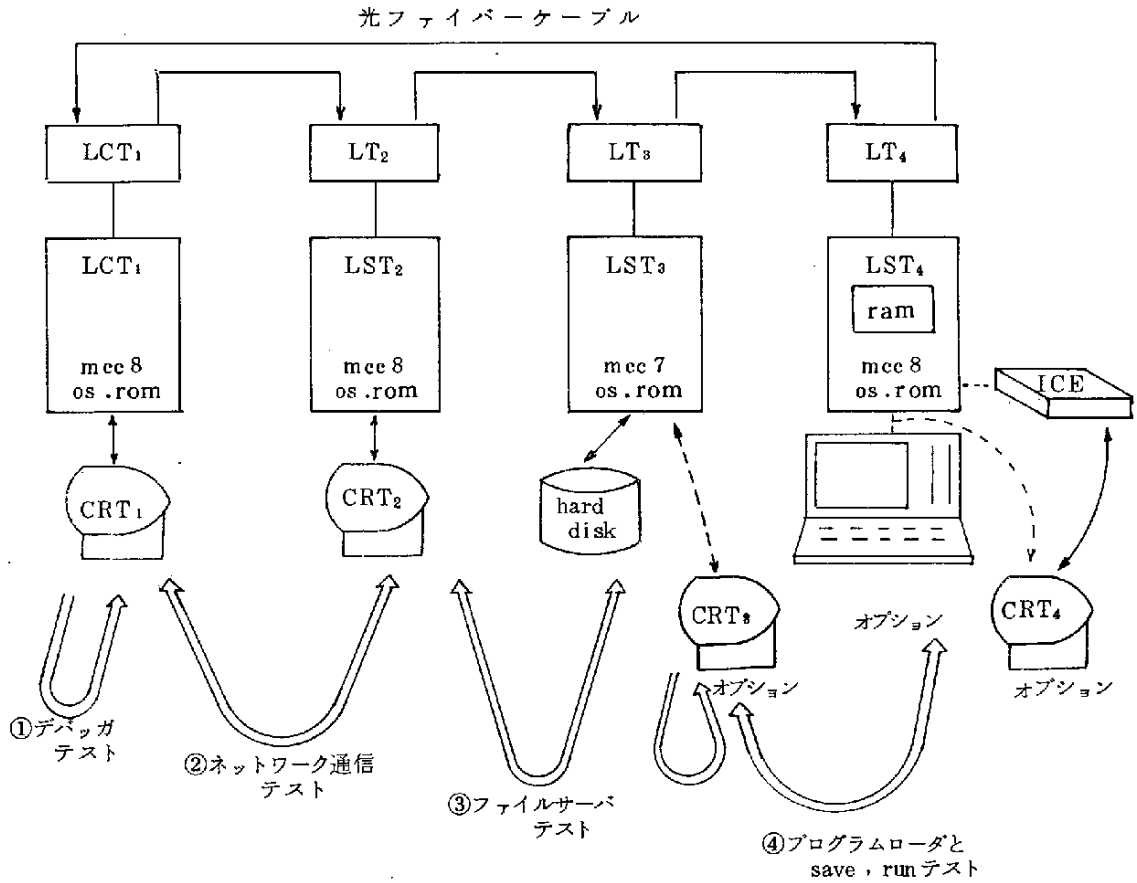
ネットワーク端局用にリエントラントサブタスク（プログラムは1本で端局数だけタスクが走る）を用意して具現化している。



（注1） ファイルサーバコマンドは操作説明にて行う。

8. システムテストプログラムとその操作

(1) ハード構成とテスト概略



（注1） ５６年５７年に開発された装置にはcrt 2台しか接続されていないがスムーズにテストしようとするとcrt 4台が必要である。

（注2） LSTにはRS232Cポートが１個しか用意されていないため、パソコンよりプログラムをロードする時はcrtと接続変えをしないといけない。

（注3） mcc8, mcc7はアプリケーションプログラムromの名称であり、mcc8はテストプログラムがmcc7にはファイルサーバ端局プログラムが格納されている。

(2) 電源投入操作

OS.rom(16KB)としてパソコンからのプログラムローダ付を採用している。それがためＬＳＴの電源投入にあたってはcrtを接続しておく必要があり、電源ＯＮと同時に次のメッセージを出力する。

```
Program Load (y/n)=n
```

アプリケーションがromにある場合にはn crと答えるが、その前にネットワークステーション（ＬＴ）の電源を投入、又はリセットする必要がある。

(3) システムテスト（ＭＣＣ８）の操作

アプリケーションプログラムが動き出すと次の表示となる。

```
*** Loop Network test on C.OS ***
my station (1-32)=
target station (1-32)=
Master                          Slave
11 pattern data send            12 pattern data recv
21 pattern data send->recv      22 pattern data recv->send
31 crt data send                32 crt data recv
41 crt data send->recv          42 crt data recv->send
51 macro test                   52 macro test
61 file write                   62 file read
99 end                          98 debugger -- fpst,3,1
Menu Number =
```

１１～５１が自局側動作であり、１２～５２が相手局側の動作となる。

６１はファイルサーバにcrtデータを送り、６２にてファイルサーバのデータをcrtに表示させる。

① デバッガテスト

９８にてデバッガに起動をかけ、各種コマンドのテストを行う。

終了時点にて fpst , 3, 1 をキーインすると上記メニューに戻る。

② ネットワーク通信テスト

メニュー１×～５×が対象となり、１１・２１の時にはフレームバイトのパターンとフレームサイズと送信回数が設定できる。５１の時にはネット

ワーク通信ロジカルハンドラのマクロをそのままキーインして相手局との交信テストを行う。

③ ファイルサーバテスト

61にてデータを送りこみ、62にてその確認を取る。又、デバッガのsaveコマンドとloadコマンドによっても行うことができる。

④ プログラムローダとsave, runテスト

パソコン上にて" post・fpost n回テストプログラム"（MCC6）をロードポイント4000からのロードモジュールを作成し、OS・rom上のローダにてrom空間にローデし、デバッガの中のsaveコマンドにてファイルサーバに移し、runコマンドにて実行させる。

(4) ファイルサーバ端局プログラム（MCC7）の操作

F割込キーインにて次の表示となる。

```
*** Hard Disk Test menu ***

1. Volume initial
2. Mount
3. File make
4. File erase
5. Catalogue Print
6. File close
7. Dismount
8. Hex format I/O Test
9. End

Request =
```

ネットワークのファイルサーバとするには1、2が必要である。システム終了にて5で確認を取り、7で終了させる。詳細チェックには8を用いて16進データをファイルサーバに送るテストを行う。

禁 無 断 転 載

昭和５９年 ３ 月　発行

発行所　財団法人日本情報処理開発協会
東京都港区芝公園３－５－８
機械振興会館内
TEL（４３４）８２１１（代表）

印刷所　株式会社　昌 文 社
東京都港区芝５－２６－３０
TEL（４５２）４９３１（代表）