数式処理専用計算機FLATSのデバッグ・システムについて

稲田信幸（理研），鈴木正幸（東大・理），平木 敬（電総研），
清水謙多郎（東大・理），佐藤三久（東大・理）

1 はじめに

昭和54年度より，理研で開発中の数式及び記号処理向き計算機FLATS は，現在理研に搬入されマシン・デバッグ中である。これまで多くのLispマシンの素子がTTL中心なのに対して，FLATS ではECL100K及び10Kシリーズを使用し回路の高速化を図っている。ここではFLATSのアーキテクチュアやデータ・タイプ等については論じない。これまでに行ってきたデバッグの方法の概要について述べる。詳細については個別に別の機会に発表される予定である。本稿で言及するデバッグ・プログラムは筆着らと三井造船KKの間で共同で開発したものである。

2. システムの構成

FLATSマシンのゲートの数を正確に計算したことはないものの使用ICの数が，34000 個になっているため汎用の大型機クラスの規模となっている。故にこの程度の規模のマシンになると大型機の開発と多少なりとも似た製作のアプローチをとる必要が感じられる。開発当初よりデバッグやメインテナンスのし易さも念頭に入れておく必要があった。またロジック・アナライザやオシロスコープに頼った開発は，その信号の数（約25000本）から見ても不可能に近かった。それでハードウェアの一部にそれに替る機能を取り入れておく必要があった。

SVP（サービス・プロセサ）はFLATS の診断をUX-BUS を介して行なう他にFLATS からの要求（割込）に基づいてCHP-ADP 経由で入出力処理やページングを代行する。FLATS はSVPのバック・エンド・プロセサとして位置づけられ，SVPから見ればSVPのメモリ及び入出力装置として見える。FLATSのMMアクセスのためにマッピング・レジスタとウィンドが2つ用意されている。FLATS では約25000 本の信号を直接scan out することが可能で，図2.ではa,b,c すべてscan out できるが，scan in はlatchの出力aに対してだけできる。

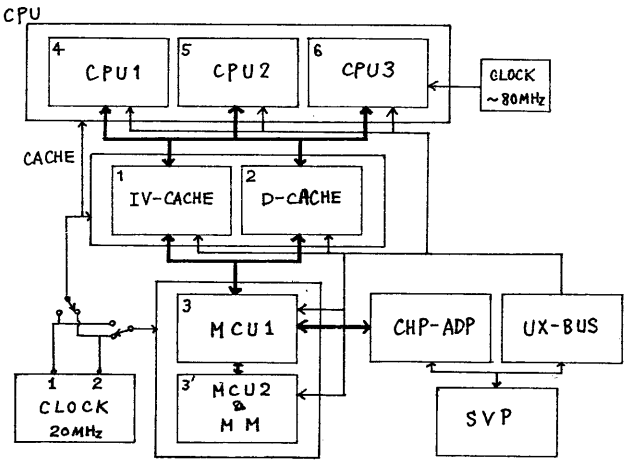


図1. FLATS システム構成

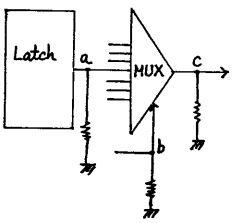


図2. Scan in/out の例
（Rはターミネータ）

FLATSはCPU，CACHE 及び主記憶を含むMCUで構成される。CPUはCPU1，CPU2 及びCPU3 の3筐体より成る。CACHE はIV-及びD-CACHE の2筐体より成る。MCUとMMで同じく2筐体を占め，全体は星形に配置され，筐体間の

線長を短かくするのに貢献している。CPU1には先行制御，パイプライン処理等のステートを管理する制御記憶(ICS)が256×40bitの高速RAMで実現されている。また命令のデコードのための制御記憶(VCS)が256×16bitの高速RAMで同じく実現されている。CPU1~3には命令の実行をコントロールする制御記憶(DCS)が同じく高速RAMで実現され，その大きさはアドレス0~255までは64bit巾，アドレス256~1023までは340bit巾である。新しい命令を追加しない限り，命令のデバッグはこのDCSを対象に行なわれる。

IV-CACHEは命令専用の命令キャッシュ(I-cache)と汎用レジスタ及びフレーム・レジスタ用のオペランド・キャッシュ(GV-cache)とから成る。I-cacheだけはwrite throughで8KB実装されており，通常は読み出し専用に用いられる。GV-cacheは3ポート同時に読み出し可能なレジスタ群で6KB実装されている。GV-cacheはデータ空間の主記憶にマッピングされるので，レジスタとして使用可能な量は最大16M語となり，プロセスの切り替え等に伴なうマルチ・スタックやレジスタ・ファイルの概念は，FLATSではハードウェアレジスタのGTOPとCFPの内容を切り替えるだけで済む。命令語のオペランドでアドレス可能な汎用レジスタ及びフレーム・レジスタの数は各々128組，127組である。

D-CACHEは命令がデータ空間をアクセスする場合と，CPU1内のCユニットでコントロール・スタック(Cstack)をアクセスする際に使用される。CPU3では32byte並列サーチをこのD-cacheに対して要求し，その判定を1マシン・サイクルで実行することができる。Cユニットは専用のレジスタCSTB(Control Stack Top Buffer)をもっていて，サイクルスチールでD-cacheにアクセスしている。

各CACHEへのアクセスはすべて仮想アドレスのまま(アドレス変換なしに)行なわれるのでキャッシュ上に存在する(on cache)場合には1マシン・サイクルで読み書きが可能となり，CPUのパイプラインを乱すことがない。キャッシュ上に存在しない場合にはmiss hitとなりMCUの助けを借りてブロック・インを待つ。

MCUはMemory Control Unitの略で，SVPとのインタフェースを受けもつ他に各キャッシュからのmiss hitの解消のためにMMより32byteのブロックを読み込んだり，キャッシュからブロックをMMへ書き出したりする。この時，仮想アドレスより実アドレスの変換をページ・マップ・テーブル(PMT)を使用して行なわれる。PMTに登録されていない仮想アドレスの参照はページ・フォールトとして検出される。FLATSの場合，cons領域のフリー・ポインタLPRの上下限のチェックはハードウェアで行っていないがページ・フォールトを契機にGBCが起動される。MMとキャッシュ間でブロックのswapを行なう毎にページ・ステータス・テーブル(PST)を更新している。

PMT及びPSTはキャッシュと同等の高速RAMが用いられ各々の大きさは256×10×33bit及び2048×16bitである。MM，PMT，PSTはCHP-ADP経由でSVP側から自由に読み書きができる。MMは最大128MBまで拡張可能であるが，現在は8MB実装されている。

3. デバッグの手順と作業分担

大規模なマシンになればなる程，順序だててデバッグをし，その成果を積み重ねてゆく必要がある。開発マシンの誤りの原因は，

1). ボード・部品・素子の不良，

2). 誤配線・実装の不良，

3). 設計・論理の誤り,
4). マイクロ・プログラム等のファームウェア の誤り,
5). テスト環境の誤り,
6). 被テスト・プログラムの論理的な誤り
等に分類できる。

図3はデバッグの流れを示しているが, FLATS の場合そのいくつかは並列に行なわれている。左側はFLATS マシンによる実テストで, 右側は計算機シミュレーションによる虚テストである。FLATS マシンは1台しかないので環境が整備されだすとマシンタイムの割当ても効率の良いデバッグ方法の重要なファクターとなってくる。

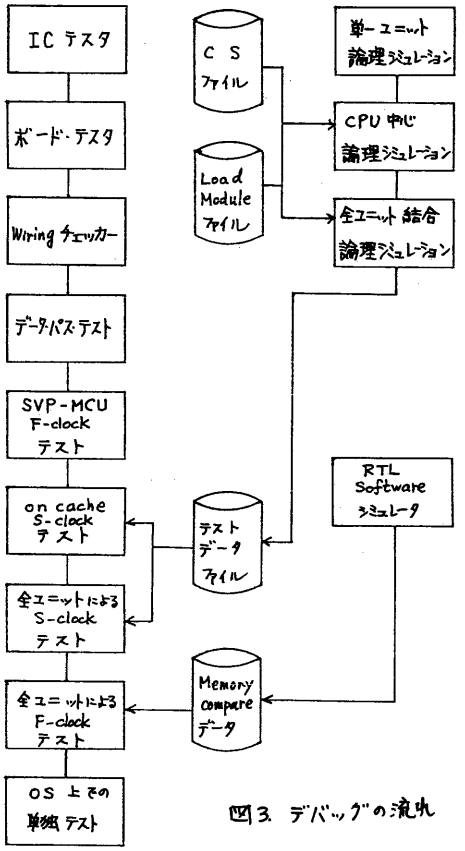


図3. デバッグの流れ

論理設計は理研側, 詳細設計・実装設計及び製作は三井側と定め, またデバッグについても分担を取り決めドッキングして作業にかかった。使用するICはICテスタにかけられ, 検査合格のものを原則として使用する。ボードも部品・素子を装着後ボード・テスタにかけている。

図4はFLATSのボードでサイズは10×16 (cm), ICの白く点に見えるのは検査されたICである。現実的にFLATS の場合, 初期不良ICの他に死んだICはこれまでのところ1個である。最初に製作された筐体はもうかなりの通電時間になっている。FLATSのボード内の論理が浅いため必然的にバック・パネルの配線の量は多くなり, 誤配線の確率も高くなる。三井側では布線CADを準備し, 慎重に配線したせいか誤配線による誤りは殆んどなかった。wiringのテストもボード・テスタを駆使して数日の作業量で終了している。ここまでは筐体にボードを差す以前のテスト工程である。

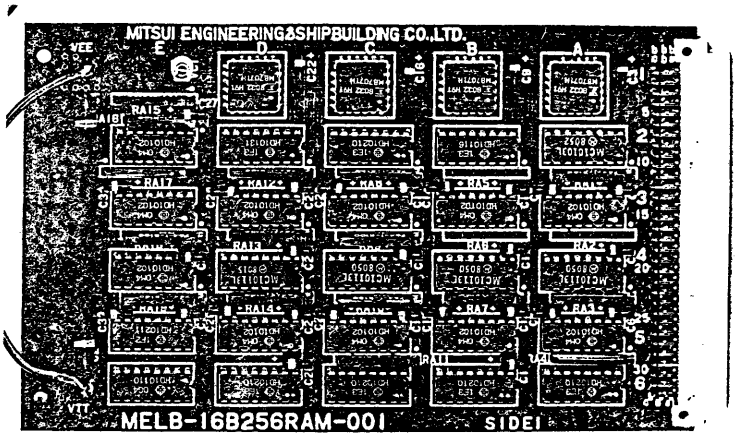


図4. FLATS ボードの一例 (100×160 mm)

SVPよりクロックを与える与えないの指示が可能で, 且つ2系統使用することができる。Sclock

というのはSingle clock モードのことで1マシン・クロック与えた後マシンはスタティックに止っている。すなわち回路の信号はすべてレベルとして見ることができ，早い話信号の1,0をテスタで測っても構わない。Fclockというのは通常の基本クロックで動作する状態である。現在デバッグ中は200nsに設定している。FLATSはNclockというモードがあり，n-1のFclockを与えた後，nクロック目はSclockと同一状態になる。Sclock, Nclockで注意しなければならないのは，主記憶にダイナミックRAMを使用しているので，クロックを止めたのではrefreshが行なわれずデータが消えてしまう。Sclock, Nclockテストの際には実際主記憶にはアクセスせず主記憶のコピーをSVP内にもっている。クロックが止っている時にUX-BUS経由で信号をscan outしたり，latchにscan inしたりできるので，シミュレーションによってクロック毎に吸い上げられたテスト・データをscan inし，scan outして結果比較することによりハードウェアのテストが行える。

4. シミュレーション

FLATSシステムはすべて論理CADシステム（DDL*）で記述され，設計当初より計算機によるシミュレーションによって論理の正しさを検証してきた。ダイナミックRAMとCPU3の除算回路以外はすべてシステム・クロックと同期して動作する。

FLATSの場合，論理的に意味のある単位（ユニット）毎にDDL*で記述されている。そのDDL*によるソース・ファイルのディレクトリを表1に掲げる。

XDIR MESTXT.DATA

表1. CPU1,2のDDL*ソース・ファイルのディレクトリ

1 A	10 DT	19 ICHMLD	28 MMWRT	37 U
2 BIT	11 DUMP	20 IPLRWM	29 P	38 UX
3 C	12 F	21 J	30 PMTMLD	39 V
4 CSLD	13 FUNCTION	22 M	31 PSTMLD	40 VCH
5 DCH	14 G	23 MAIL	32 R	41 VCHMLD
6 DCHMLD	15 H	24 MEMINIT	33 RBUBLD	42 W
7 DD	16 I	25 MEMLD	34 REGLD	43 XMEMLD
8 DE	17 ICC	26 MEMWRT	35 S	
9 DS	18 ICH	27 MMLD	36 SVP	

最初ユニット毎に高速なシミュレーションが可能である。そのユニットで必要な入出力をダミー・ルーチンを書いて結合することになる。例えば，CPU3の高速乗算回路は100万クロック以上もシミュレーションが行なわれた。もちろんこの数でもシミュレーションすべき数から見ればほんのわずかである。果して大型機などは完全にシミュレーションしきれない部分は何で補っているのだろうか。

次にキャッシュとMCUを擬似ユニットで記述してCPUの各ユニットを結合して論理シミュレーションが行われた。この段階になると各種制御記憶用のCSファイルをロードするダミー・ルーチン，被テスト・プログラムをロードするダミー・ルーチンが結合された。CSやLDMファイルの作成は別の支援プログラムによって成りたっている。CPUが記述されているので，あたかも全命令，全データがon cacheであるという仮定のハードウェア・テストと等価にシミュレーションができ，殆んどの命令がテストされ現在約230のマシン命令が備っている。論理の正しさの検証はシミュレータからのダンプ・リストあるいはトレースによる他，入出力擬似命令の実行によって直接8式の形でも確められた。当然FLATSのDCSを中心とするマイクロ・プログラムがデバッグの対象となり，ソース・プログラムは約15000行にはね上った。

DDL*でFLATSを記述した真の目的は，テスト・データをシミュレーションによって生成し，ハードウェアテストの入力としたかったからである。次の段階でMCU

及びCACHEの全ユニットを結合したテスト環境が整備された。理研ではFLATSが搬入になる以前よりCPU3をも含めたテスト環境が整備されていたが，テスト・データはCPU3なしの環境で作成しなければならない。ハードウェアの構成に環境を合わせる必要がある。CPU3なしのテスト環境は三井造船のVAX上に作成された環境をFLATS搬入とともに理研のM380に移植した。VAXで250クロックのシミュレーションに約1時間必要であったのが，M380上では1クロック当り1.3秒となり，SVP側のディスク容量の関係で約1700クロックを上限にテスト・データを生成している。被テスト・プログラム名の一覧を表2に掲げる。

表2. テスト・データ作成用のテスト・プログラム名のリスト

```
XDIR MESLDM.DATA
  1 AKZNZ      11 FACS24     21 MAP       31 RTTJ01     41 TSCSTB
  2 APPEND     12 FBLKIN     22 MAPC      32 RTTJ02     42 TSTCFP
  3 BLKOUT     13 FTHRU      23 MAPCAN    33 RTTJ03     43 VUNIT
  4 CALLR1     14 GETVX      24 MAPCAR    34 RTTJ04     44 VUNITX
  5 CGOTAG     15 GOTOR1     25 MAPCON    35 SEQ        45 ZALU
  6 CHKREG     16 IPFGO      26 MAPLST    36 SEQUEN     46 ZTSTX
  7 CPUINT     17 LDCSP      27 PAIRLS    37 SKZNZ
  8 CPUSH1     18 LDID       28 PURGVS    38 STRTHR
  9 EXPAND     19 LDMCR      29 REVAPP    39 TAKLST
 10 FACLST     20 LSTCHR     30 REVSIP    40 TEST
```

DDL*によるシミュレーションでは，数100命令どまりである。インタプリータや実行時サポート・ルーチンのデバッグをするには，レジスタ転送レベル(RTL)の命令のシミュレータが必要である。今回これまでの第1版のものを全面的に書き替えた。高速に実行できる代りに，クロックの概念，先行制御，パイプライン，及びサイクル・スチールといったハードウェアのパラレリズムは記述できない。しかしデバッグ時には図5のようなユーザが直接見たいハードウェア・レジスタの内容は命令終了時点でハードウェアとの一致が保証されている。一般にはある時点、

図5. RTLシミュレータのレジスタ表示

```
----- Registers and Pointers ------

      NCLK        PC       CSP       CFP      GTOP       LPR       RPR       HPR
      6905   00003A    000401    000100     00000    001450    001100    0017F3

       VR1L      SIZE      VR2L       DWL       DWH       DVA       VWL       VWH
   00003C3C  00000000  00000001  5E000958  9F00145C    003C3C  00003C3C  00000000

       DRL       DRH       IVA       IRL       IRH       IWL       IWH       IBR
   5E000958  9E000958    00003A  AD800000  E400003C  00000000  00000000  AD800000

       IMV       CVA       CRL       CWL      ACCL      ACCH       DPC       JPC
   0000C3C3    000402  80000036  80000036    000000   0000000    00003A    000063

      NILR        TR      RTTR      INTR       OPW      OPRW      GVAW        IL
   1E000900  1E000904         0         0  30000000        81        01         0
```

でのメモリの内容をチェックする方法が最善である。RTLシミュレーションではcacheのシミュレートは大変になる。命令をシリアルに切り出して実行していたのではcacheの内容が正確に反映されないからである。しかし，ハード命令の各cacheに対するpurgeを発っすればcacheの内容はすべてメモリにダンプされる。このあとでメモリの内容の比較によって結果の正しいことが確かめられる。

RTLシミュレーションでは毎秒10000命令以上シミュレートすることが可能でDDL*によるシミュレーションに比較して数万倍高速であるが，実クロックに対しては約1000倍程遅い。OS，インタプリータ，コンパイラ等が開発し易い様にユーザ・インタフェースの考慮されていて，例えば実行中に割込んでデバッグのモードを変えたり，レジスタの内容を見たり，メモリ・ダンプがとれるようになっている。

5. マシン・デバッグと環境

FLATSではCPU-CACHEとMCUに対して別々のclockを与えることが可能であるが現在は，同一系統しか使用していない。テストの中でSclockテストが一番きっい（エラーを発見するという意で）テストである。FLATSのscan outできる信号の殆んどをチェックしているからで，don't careのゲートですらチェックの対象となる。ECLのデータ・シートには，きり書かれていないものは逆にマシン・デバッグをすることによってDDL* のソースを修正している。

データ・パス試験では，memoryがある筐体の場合それへの全アドレスread/writeテストが行なわれ，latchにscan inしてscan outしてみるテストが行なわれ，マルチプレクサなどは入力を与えておいて切り替えのテストをする。この試験ではミクロ的な回路に目が向けられているのに対して，SVP-MCUのFclock試験では，SVPよりMCU内のメモリを初期したり，write/read/checkテストしたりしている。これはCHP-ADP経由で高速に行える。MCUのPMTのアドレス変換のロジックも確かめられた。この時，CPUやCACHEは止っていたので，SVPとFLATS双方からのアクセスによるアービトレーションがうまくいっているかは確かめることができなかった。

全命令や全データををcacheにロードしてやれば，on cacheでCPUの試験を行なうことができ，CPUの動作の確認，CPU-CACHE間の動作の確認がSclockテストで完全にチェックすることが可能である。主記憶へアクセスしないので，たとえrefreshが行なわれなくても構わない。このテストではMCUのテストは何も行なわれない。キャッシュのコントロール・メモリの方をすべてhitした状態にしてやる必要がある。あまり大きなテストはできないが，CPU1の先行制御やパイプライン処理などが確認でき，マシンとしてのテストは大いに進展したことになった。

命令やデータをcacheにロードしないで主記憶にロードし，PMTおよびPSTを適当にセットしてやれば，命令フェッチやデータの参照の際にmiss hitとなりMCUが主記憶からcacheにブロック・インして，miss hitを解消するはずである。MCUは11のユニットから成っていて大変複雑な動きをするので，テストは繁殖そのものであった。しかし，DDL*によるシミュレーションでは予定したとおりの動作をMCUがしているのが確かめられていたので，問題はSclockモードでは主記憶の内容が消えてしまうことに集中していた。幸い三井側で，SVP側でテスト・データのファイルより主記憶のデータのコピーをとってくる環境を整備された。FLATSはブロック・インの時，SVP側の主記憶のコピーからデータを転送されることになる。

現在，この環境のもとで表2のテスト・データをM380で生成し，MTで吸い上げマシン・デバッグを行っている。Sclockテストの場合，100クロックテストするのに約30分必要である。Nclockテストも同一環境のもとで実行できるが，実際にnを指定してもそれまでに，ブロック・インやブロック・アウトがある場合にはそこで中断しSVP側で主記憶のデータと一致させておく必要がある。通常は1000クロックまでNclock指定し，それからSclockモードで動作させたりしている。もし最後までNclockでテストした場合には，Cacheの内容を比較するコマンドでチェックしている。

この段階では，まだCPUがMCUなどにアクセスしている間にCHP-ADP経由でアクセスするという，FLATSとSVPがMCUを競合して使用していない。MCUのアービトレーションではSVP側に高いプライオリティを与えている。FLATSがFclock

でMCUを使用して動作している最中に，SVP側が主記憶を読みおすとか，PMT/PSTにロック をかけるとかができるか確かめなくてはならない。また，SVP側にFLATSの主記憶のコピーを置く必要もないので，大きなテスト・プログラムをFclockでは走行させてデバッグすることが可能である。結果が正しいかの検証にはRTLシミュレータのメモリ・ダンプと比較すればよい。現在はSclockテストの一部の環境を使用してFclockテストが行なわれている。MCUのアービトレーションが正常に動作することも確かめられている。このテストには，自分でreverse-appendを実行して比較のデータを作り，この時のLPRの値を記憶しておき，再度reverse-appendを実行してリストを作成し先程のリストとEQUALを呼んで比較し，全データ・スペースをpurgeしてcacheから主記憶におとし，結果が同等であれば再度実行しにゆくようにしておき，この時SVPよりメモリの内容をダンプしたり，PSTの内容をダンプしてMCUのアービトレーションを確認した。特にPSTの内容はpurgeの前後でおうりと変るのでFLATSが走っているか否かの判定にもなっている。図6で8つの数字は左側より命令空間ヤデータ空間に割当てた物理ページのステータスを表わしているPSTのダンプである。4ケタの16進数のうち下2ケタはにキャッシュにのっているブロック数を表わしているので，第4番目の数字がa401より2400に変っているのはCstack領域がpurgeされた直後のPSTのダンプを表わしている。

図6. Fclockテスト・コンソール表示の一部

Fclockの環境は，Sclockの環境に依存しているので環境の整備が待たれる。またRTLシミュレータのダンプ結果と比較するなどの作業が自動的に行なわれるのがよいと考えている。

現在Fclockまでの試験が行なわれているが，Sclock試験を通してのみ悪くなったICの発見やdon't careのステートまでのチェックが行なわれるので，Sclock試験の重要性は変化していない。

SVP側に入出力やページングが完備してくると，もはやFclock試験と呼ぶよりOSと呼んだ方がよくなるだろうし，インタプリータをのせればどのくらいのスピードで動いているのか実感するようになると思う。

6. 今後のデバッグ予定

CPU3はCPU1,2の拡張演算ユニットとして位置づけられる。高速演算回路を中心に構成されており，特に除算器は80MHzのクロックで非同期に動作する。このデバッグはこれまでの同期系の環境は使用できない。但し80MHzのクロックは可変なので，はじめ20MHzにしておき除々に上げてゆく方法がとられる。クロックを変える毎にDCSの方も変えなくてはならない。Sclockテストは不可能なのでFclockテストで結果を比較しながらのテストになるはずである。高速乗算器はスピードが本当に速いので，シフタは不要なくらいである。ユニフォームなハッシュ・アドレスを求めるために乗算器を使用している。

CPU3搬入時に，全筐体に温度センサーがつけられるので，今後Sclock試験はなるべく夜に実行する予定で，昼はマシンをもっと有効に使用する予定である。

7. おわりに

μアセンブラ・FAP・LOADERといったデバッグを支援したソフトウェアについてあるいは三井造船側で作成された支援ソフトウェアについては別の機会に発表される予定である。我々は計算機メーカから見れば素人には違いないが，FLATSの論理設計にDDL* を使用して記述し，大型機でシミュレーションを中心にデバッグが出来た点を強調したい。作業を分担して統合してゆくことは個々人の協調性がなければできないと思うが，環境づくりには皆一丸と協力し合えたのも功を奏したといわねばなるまい。

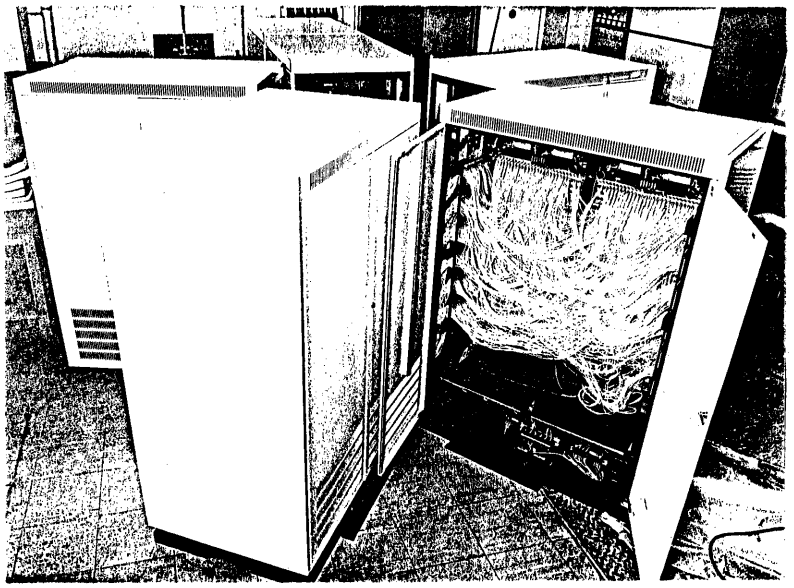
三井造船KKのFLATS担当グループをはじめ関係者に，また研究会を通して協力して下さった富士通KK及び富士通研究所の関係者に感謝の意を表します。

参考文献

[1] Goto, E. et al, "Design of a Lisp Machine - FLATS", ACM Symposium on Lisp and Functional Programming, (1982).

[2] 稲田信幸, "FLATSにおける記号処理命令について", 記号処理研究会資料 15-2, (1981).

[3] Shimizu, K., "Design and CAD Implementation of Formula Manipulation Machine, FLATS", master thesis, Dept. of Information Science, Univ. of Tokyo, (1982)



屋形に配置された各筐体 FLATS(CPU3を除く)