お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

ユーザーズ・マニュアル

RENESAS

17K シリーズ

アーキテクチャ編

© NEC Corporation 1991

○本資料の内容は，後日変更する場合があります。
○文書による当社の承諾なしに本資料の転載複製を禁じます。
○本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的所有権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。
○当社は品質，信頼性の向上に努めていますが，半導体製品はある確率で故障が発生します。当社半導体製品の故障により結果として，人身事故，火災事故，社会的な損害等を生じさせない冗長設計，延焼対策設計，誤動作防止設計等安全設計に十分ご注意願います。
○当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定して頂く「特定水準」に分類しております。また，各品質水準は以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認の上ご使用願います。

標準水準：コンピュータ，OA機器，通信機器，計測機器，AV機器，家電，工作機械，パーソナル機器，産業用ロボット

特別水準：輸送機器（自動車，列車，船舶等），交通用信号機器，防災／防犯装置，各種安全装置，生命維持を直接の目的としない医療機器

特定水準：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等

当社製品のデータ・シート／データ・ブック等の資料で，特に品質水準の表示がない場合は標準水準製品であることを表します。当社製品を上記の「標準水準」の用途以外でご使用をお考えのお客様は，必ず事前に当社販売窓口までご相談頂きますようお願い致します。
○この製品は耐放射線設計をしておりません。

M7 94.11

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

●はじめに

今、商品設計をしている方へ
今、あなたはどんな商品を設計していますか？
それはもしかして小型化や高機能化や低コスト化したい製品ではありませんか？
それはもしかしてDTSやLCDを使った製品ではありませんか？
今までのコントローラに満足していますか？
あなたの設計を生かせるプロセッサを捜していませんか？

あなたは17Kシリーズをご存知ですか？
もし17Kシリーズをご存知でなかったら
一度本書を読んでみてください。
17Kシリーズがあなたの設計に役立つことを発見できるでしょう。

これから商品設計をされる方へ
あなたは今のプロセッサの機能をご存知ですか？
あなたは今のコントローラの性能をご存知ですか？

あなたは17Kシリーズをご存知ですか？
もし17Kシリーズをご存知でなかったら
一度本書を読んでみてください。
今度の商品の性能を1ランクアップさせていることに気が付くでしょう。

そして
本書を読んで、
17Kシリーズを使って
作りませんか？

●アーキテクチャとは　－まえがきにかえて－

まずは、おさだまり方法ですが、言葉の定義から。

> アーキテクチャ：コンピュータシステムの属性。
> つまりシステム構成や命令体系、アドレス方式、データ形式などをアーキテクチャという。

製品を設計する人にとってみれば、自分の考えている機能が実現でき（コストも計画の範囲内に抑えられ）ればどのようなコントローラでもいっこうに構わないはずです。だから、そのコントローラでできる最大のことが知りたいでしょう。
プログラマにとってみれば、既に使用するプロセッサが決っている場合がほとんどですので、与えられた資源（機能）をいかに使いこなすかが重要なテーマになってきます。だから、使用するプロセッサで今、何ができるのかが知りたいでしょう。

また、プロセッサは素材であると同時に、道具でもあります。周辺ハードウエアやマクロやライブラリが内蔵されているからです。使用するプロセッサにどんな道具が用意されているのかが知りたいでしょう。道具や素材を知ることは重要なことです。

民生市場では、コスト、サイズなどの制約により必ずしも提供可能なすべての機能が実現されているとは限りません。というより不要な機能は付いていないはずです。
専用マイコンでは、１つのアーキテクチャで複数のプロセッサを構成する、いわゆるファミリーを形成しています。したがって個々のプロセッサの機能、性能は微妙に違っている場合があります。しかしもし、個々の品種ごとにアーキテクチャが違っていたらどうなるでしょう。違う品種のプログラムを開発するたびに新しいアーキテクチャを覚えなければなりません。でも、最近の専用マイコンでは、１つのファミリーのアーキテクチャは１つになってきています。

いままで、個々のコントローラごとに実際に提供されている機能の詳細を記載した資料はありました。データ・シートがそれに当ります。これは、どちらかといえば、実際にコーディングを必要とするプログラマ向けでした。でもここから、コントローラの最大仕様を把握したり、その可能性を見つけることはなかなか難しいことです。つまり製品を設計する人には、もの足らない面があったのではないでしょうか？

そこで、本書では、製品を設計する人、つまりこれから設計する、もしくは設計を生かせるコントローラを捜している人のために、17Kシリーズの可能性を簡単に解説したいと思います。このためには、17Kシリーズの設計概念、アーキテクチャを知ってもらうのが一番いいと思うのです。
もちろん、プログラマの方の参考になるようにも務めました。17Kシリーズの設計概念＝アーキテクチャを知ってもらうことは、17Kシリーズに適した、17Kシリーズの機能をうまく生かしたプログラムを作成するための手助けにもなると考えるからです。

そしてできれば、本書は17Kシリーズと関係のあるすべての方に、実際に17Kシリーズをお使いの方にはデータ・シートとともに、まだ17Kシリーズをお使いでない方には、本書だけでもご覧いただけると幸いです。

（本書では、μPD17000シリーズに関するすべてを総称して17Kシリーズと呼んでいます。また本書で17Kシリーズといっても、すべての品種において共通の事柄であるとは限りません。μPD17000シリーズにはさまざまな品種があり、個々の品種は目的に応じて微妙に違うことがあります。個々の品種については品種ごとのデータ・シートをご覧ください。）

目次

●はじめに
●アーキテクチャとは －まえがきにかえて－

■第1章 現在、過去、未来

●17Kシリーズの歴史 ―― 1
◆DTSから、1970年代末
◆17Kシリーズの登場、1980年代後半
◆そして環境の整備、1990年代
●17Kシリーズの展開 ―― 4
◆17Kシリーズの応用分野
◆17Kシリーズの得意なこと
◆17Kシリーズの製品展開
●17Kシリーズが目指すもの ―― 6
◆設計者のために
◆プログラマのために
◆そして…
◇コラム：17Kシリーズのこわい話

■第2章 基礎知識、専門知識

●4ビット・シングルチップを理解するために ―― 10
◆まず、コンピュータとは
◆4ビット
◆なぜ4ビットなのか？
◆シングルチップとマルチチップ
◆専用マイコンと汎用マイコン
◇コラム：コンピュータの分類
●コンピュータを理解するために ―― 15
◆ハードウエアとソフトウエア
◆データとプログラム
◆アドレス
◆高水準言語と低水準言語
●コンピュータの専門知識 ―― 18
◆ニブルとバイト
◆プログラム・カウンタ
◆アドレス・バスとデータ・バス
◆インストラクション・デコーダ
◆アキュムレータ方式と汎用レジスタ方式
◆サブルーチンとマクロ
◆LIFOとFIFO

■第3章　小さなCPUとプログラム・メモリ

●17Kシリーズ・アーキテクチャの特徴 ―― 23

◆RISC指向と拡張命令

◆ASIC指向とASC

◆キャッチフレーズ

●CPUとインストラクション ―― 26

◆命令フォーマットとその種類

◆プログラム・カウンタ（PC）

◆アドレス・レジスタ（AR）

◆やっぱり電子計算機

◆ALU

◆プログラム・ステータス・ワード（PSW）

●プログラム・メモリ ―― 31

◆プログラム・メモリの特徴

◆プログラム・メモリの大きさ

◆プログラム・メモリの構成（セグメントとページ）

◆プログラムの分岐

◆間接分岐

◆アドレス・スタック・レジスタ

◆スタック・ポインタ

◆SPARELIB

■第4章　データ・メモリ

●データ・メモリ ―― 39

◆データ・メモリの構成とバンク

◆システム・レジスタとバンク・レジスタ

●データ・メモリと周辺回路 ―― 42

◆レジスタ・ファイルとウィンドウ・レジスタ

◆レジスタ・ファイルに割り当てられたフラグの例、DMA許可フラグ

◆汎用ポート

◆ポート・レジスタ

◆データ・バッファと周辺ハードウエア

◆I/O空間の大きさ

◆データ・バッファとプログラム・メモリ

■第5章　アドレシング

●汎用レジスタ —— 51

◆汎用レジスタとは

◆レジスタ・ポインタ（RP）

◆汎用レジスタを用いたアドレシング

◆汎用レジスタと即値の演算

●インデクス・アドレシングとインデクス・レジスタ —— 56

◆インデクス修飾

◆インデクス・レジスタ（IX）

●間接アドレシングとデータ・メモリ・ロウ・アドレス・ポインタ —— 58

◆間接アドレシング

◆データ・メモリ・ロウ・アドレス・ポインタ（MP）

●インデクス修飾と間接アドレシング —— 62

●配列 —— 64

◆4ビット以上のデータ

◆配列

■第6章　割り込み

●割り込みとは —— 66

◆ハードウエア割り込みとソフトウエア割り込み

◆割り込みの優先度と多重割り込み

◆17Kシリーズの割り込みとメモリ

◆VAG

◆割り込み時の動作

◆割り込みとスタック

■第7章 開発ツール

●AS17K —— 72

◆アブソリュート・マクロ・アセンブラ
◆読み易さと日本語
◆シンボル定義
◇コラム：言語処理ソフトウェアの話
◆疑似命令と制御命令
◆出力ファイル

●IE-17K —— 78

◆エミュレータの歴史
◆Mam'Chip方式
◆インサーキット・エミュレータによるディバグ
◆IE-17Kの内部とCLICEとインタフェース

●SIMPLEHOST —— 83

◆何ができるか
◆SIMPLEHOSTの実際
◇コラム：開発者の憂鬱

○コラム：プログラムの互換性 —— 86

◇コラムの続き1：プログラムの互換性、17Kシリーズでは
◇コラムの続き2：プログラムの互換性とフラグの再配置

■付録

●周辺ハードウエア —— 90

◆A／Dコンバータ
◆D／Aコンバータ
◆クロック・ジェネレータ・ポート
◆シリアル・インタフェース
◆周波数カウンタ
◆LCDコントローラ／ドライバ
◆キー・ソース・コントローラ／デコーダ
◆PLL周波数シンセサイザ
◆タイマ

●用語解説 —— 92
●索引 —— 96
●17Kシリーズの仕様 —— 99

17K series architecture

第1章

現在、過去、未来

17Kシリーズは、民生機器の多機能、多品種およびその他のさまざまな要求などに答えるために開発された新アーキテクチャ採用の4ビット・シングルチップ・マイクロコントローラです。ここでは、17Kシリーズの歴史から、現在、未来までを一望したいと思います。もしかしたら、良く分からない用語がでてくるかも知れませんが、その場合は、巻末の用語集を参照するか、『第2章 基礎知識、専門知識』を先にご覧ください。

● 17Kシリーズの歴史

まずは、17Kシリーズの歴史です。『ふるきをたずねて･･･』というほど大げさなものではありませんが、製品が生まれた背景を知ると愛着が沸いてくるものです。できればご一読を。

◆ DTSから、1970年代末

17Kシリーズの物語は、DTSから始まります（ご存知でしょうがDTSとはDigital Turning System：電子選局装置のことです）。

民生用受信機のディジタル化の先鞭としてDTSが一般に普及したのは1976年です。この当時のディジタル化といっても、ラジオの受信機やCB無線の選局に利用するための、ICで構成されたごく簡単な装置でした。

> でも、1976年といえば日本初のマイコンTK-80が発売され、バイキングが火星に着陸した年ですから･･･。それだけ民生市場がコストにきびしく、安く良いものでなければディジタル化しても意味がないということでしょうネ。

しかし、ディジタル化された機器が市場に受け入れられていくにつれて、大量に生産され、価格競走がおこり、単価が下がり、簡単に入手できるので、また大量に必要になるといった、おさだまりのパターンの繰り返しに陥りました。他の場合と同じく、この場合もブレーク・スルーの鍵は「高機能化」です。

でも、使う側としては、多機種への対応のことを考慮してさまざまな機能を盛り込んだDTSを、もしくは他製品のディジタル化に対応できるようにいろいろな種類を（しかも低コストで）要求します。

しかし作る側としても、新しい機能を盛り込んだDTSを作ろうと思えば新しい回路を組まなければならない。製品ごとに新しい機能を盛り込むたびに回路を組んでいたのでは、開発時間がかかる。1つの回路に一度に多くの機能を詰め込もうとすると、回路が複雑、大規模になり、仕様にあわなくなるのみならず、採算まであわなくなってくる。などの問題がICの構成では発生してきます。

この問題の解決策として、考え出されたのが、いってみればさらなるディジタル化で、ほんの数年前に開発されたマイクロプロセッサを組み込むことでした。マイクロプロセッサを組み込んでしまえば、使用頻度の低い機能をソフトウエアで提供することにより、他製品への展開が簡単になります。

こういった背景のもと、17Kシリーズの直接の祖先であるμPD1700シリーズが開発されたのは1978年でした。これは、4ビット・マイコンとPLL回路を1チップに内蔵したものでした。

μPD1700シリーズの開発当初の機能は選局だけの割合シンプルなものでした。しかし本格的にDTSが普及するにつれ、他の周辺ハードウエアも内蔵することにより、大規模化してきました。が、ちょうどマイクロプロセッサの開発とプロセッサの集積化技術の発展の流れが重なり、周辺ハードウエアを内蔵しても、LSI技術を使用しプロセッサを高集積化することで、プロセッサの面積を押さえてきました。当然1つのプロセッサは、だんだん多機能になっていきます。

μPD1700シリーズを開発当時、すでに命令語長16ビット、汎用レジスタ方式などのアーキテクチャが採用されていました。これらの言葉の意味についてはあとで解説します。1978年のこの年は、郵政省のキャプテンやFAXがで始めた頃です。FAXが一般に普及するのには10年かかりました。その10年の間にμPD1700シリーズはいくつ販売されたのでしょうか？

◆17Kシリーズの登場、1980年代後半

このμPD1700シリーズも年月を経過すると多機能化·大規模化も限界に達し、相当な余裕を持って設計されたμPD1700シリーズもこの業界特有の時間の流れの速さにだけは勝てずに、新しいアーキテクチャの元に17Kシリーズが開発されました。1987年のことです。
17Kシリーズの開発にあたっては、①μPD1700シリーズからの継承性を持つこと、②柔軟で大きな拡張性を持つこと、③コストパフォーマンスの向上を図ること、④プログラム開発環境の整備などが考慮されました。

ここで重要なのは、17Kシリーズが設計されてからまだ数年しかたっていない、ということです。日進月歩のどころか秒進分歩のこの世界においては、10年以上も前に設計されたものが現役でいることは、なかなか辛いことです。

まぁ、μPD1700シリーズがだいぶ前（コンピュータ業界にとっては大昔）の産物であることを考えたら、選手寿命は長い方です。これ以上昔のしがらみを引きずって、使いにくいものになる前に引退してもらったようなものです。そして、μPD1700シリーズが長い現役生活を終え、そのすべての資産を継承し、かつ新たなアーキテクチャでつい最近17Kシリーズが設計されたことは、ベテランのテクニックとルーキーの体力を持った新たなヒーローが誕生したような･･･、と思いませんか？ ちなみに、この年（1987年）にDATが発売され、NHKが24時間衛星放送を始めています。

17K シリーズが開発された当時の状況としては、そろそろパソコンが一般に普及し始め、海の向こうではマルチタスクOSのOS／2が発表されています。17K シリーズの周辺でも、多くの設計者がμPD1700 シリーズのアーキテクチャを習得し、既に作成されたプログラムやライブラリの数も無視できないほどに多くなっていました。したがって17K シリーズの設計において前述の『①μPD1700 シリーズからの継承性を持つこと』は、かなり重要な要素だったようです。でもこの問題は解決されていると思います。μPD1700 シリーズのアーキテクチャをご存知の方に本書をお読み戴ければ、多くのμPD1700 シリーズとの類似性を見つけられることでしょう。

◆そして環境の整備、1990年代

また、プログラム開発環境の整備に関しても重要性が高かったようです。
まず、プロセッサの開発と同時に分割アセンブルが可能なマクロ・アセンブラAS17Kと新しいプログラム評価方式を採用したインサーキット・エミュレータIE-17Kが発表されました。そして、1990年5月にはディバグ環境の改善のためのソフトウエアSIMPLEHOSTが発表されています。

IE-17Kは、Mam'Chip方式という新しいプログラム評価方式を搭載した小型のインサーキット・エミュレータです。また、SIMPLEHOST は、新世代インタフェース・ソフトウエア、MS-Windows上で動作するディバッグ・インタフェース・ソフトウエアです。
これらのプログラムを作る道具、開発言語、プログラム評価キットなどはあらかじめプロセッサの開発時に予定されていたものです。それまでは、これらの開発ツールはあまり重要視されていませんでした。プロセッサを開発するときには、開発ツールの重要性までは気が回らなかったのです。でも、17K シリーズは違っていました。発表はたしかにプロセッサより遅くなったかも知れませんが（でも、プロセッサより先にそのプロセッサのための開発ツールができていたって…）、開発ツールの重要性は開発時に考慮されていたそうです。だから、17K シリーズはデバイスと開発ツールの相性がぴったしです。この相性の良さは実際につき合ってみると良く分かります。
この先もプログラムの開発環境を良くするツールが開発されるかもしれません。人の要求は果てしなく、できるだけ良い環境で仕事をしたいのは当然ですから。17K シリーズの開発者たちはきっとさらに良い環境を提供してくれるでしょう。

> 1990年代の新しいキーワードにASICとRISCというのがあります。これからのコンピュータ業界のトレンドがここにあります。でも当然それだけではなく、実は17Kシリーズもこの2つのキーワードと深い関わりを持っているのです。ここでは説明しませんが、後で必ず説明しますので、この2つのキーワードはぜひ覚えておいてください。ちなみに17K シリーズのアーキテクチャには直接関わりはありませんが、ウィンドウ、インタフェース、環境も別の意味で重要なキーワードとなっています。ようするに17K シリーズでは90年代の重要なキーワードはすべて押さえているということです。

すでに17K シリーズはフルパワーで前進を始めているのです。ちょっと本書を読んでみてください。あなたの設計に17K シリーズがピッタシであることが発見できると思います。

●17K シリーズの展開

17Kシリーズが利用されている分野を少し紹介してみたいと思います。17Kシリーズの応用分野を知ることで、17Kシリーズの現在と未来が見えてくるのでは…。

◆17K シリーズの応用分野

μPD1700シリーズがDTS用から出発したのに対して、17KシリーズはADTS（Advaced DTS）用として出発しました。しかし現在では17Kシリーズの応用範囲はDTS関連にとどまらず、電子制御が必要なさまざまな製品に組み込まれ、幅広い展開をしています。ちょっと例を揚げただけでも…

ラジオ、テレビ、CD、炊飯器、ヘッドフォン・ステレオ、電子レンジ、BSチューナ、VTR、玩具、洗浄機、電子ジャー、リモコン、マウス、冷蔵庫、バッテリーチャージャー、一眼レフカメラのレンズ、電子カーペット、電子毛布、扇風機、ポット、低周波治療機…

と数え切れません。これだけさまざまな機械に組み込まれていると、今度は逆に17Kシリーズを組み込むことができない製品を捜したくなるほどです（でも17Kシリーズは応用範囲を絞った特定応用分野用のコントローラのはずだったんだけどなぁ）。

今までですと、これだけの製品に対応するには、何種類のプロセッサを用意し、何種類のアーキテクチャを理解しなければならないのかと、考えただけでも頭が痛くなります。
でもこれからは、アーキテクチャが統一されていれば、事情が違ってきます。1つのアーキテクチャを理解すれば事は足ります。アーキテクチャが同じならば、プログラムも同じ様な機能の部分は、あちこちで使い回しができますから。

◆17K シリーズの得意なこと

とこのように、さまざまな用途に拡散していった17Kシリーズですが、故郷、DTSを忘れてしまったわけではありません。依然として一番多くの種類が出荷されている品種がDTS用です。

DTSといっても昔のようにラジオだけでなく、オーディオ、TV、VTRなどの選局用にも組み込まれています。またVHFの周波数帯まで動作可能になるといった、従来の選局、バンド切換の基本機能の向上のみならず、周波数表示、プリセット・メモリの制御、CDのコントロールまで可能になっています（こうなってしまってもDTS用といってていいのか疑問ですが、基本は選局動作ですので、まぁいいでしょう）。DTS用として販売されているものでも、用途によってDTS以外の機能もたくさん内蔵している場合もあるということです。

たとえば、DTS用のある品種は、カーステレオ、ホームステレオ用のDTS LSIです。PLLシンセサイザ、FM放送受信用のプリスケーラ、LCDドライバなどを内蔵していますので1チップでDTSを構成することができます。FM、AM、TVなどの受信機を内蔵し、小型化が要求されるポータブル製品に、うってつけではないでしょうか？

また、海外では180以上ものチャンネルがあるTV、CATVなどがあります。こんなケースに今までのもので代用できますか？

さらに、17KシリーズにはIDC（Image Display Controller）を内蔵した品種があります。これは1画面に100文字近くを表示することができます。これはTV用DTSとして開発された品種に内蔵されています。TVやVTRの操作で、操作した内容を画面に表示するようにしておけば、操作性がよくなることは請け合いです。これと17Kシリーズの大容量ROMを組み合わせれば、オンライン・マニュアル機能付きの製品の設計も、もう簡単でしょう？

次にどんな分野で利用されているかとみると、リモコンでしょう。
『最近のAV機器の新機種設計ってさぁ、リモコンの設計に費やす労力が全体の15～30％を占めるんだよ。昔はただ単にチャンネルの切り換えができるだけだったリモコンが、今じゃユーザとのインタフェースをとるための道具になっているんだもんなぁ。手の中にユーザ・インタフェースを納められるように設計をするのは大変なんだよ。』

という話が本当にあったのかどうかは知りませんが、ここでも、問題となるのはユーザ・インタフェースです。『先ほどのIDCほどの機能はいらないが･･･』という方も、17Kシリーズをお使いください。もちろん従来のようにLCDドライバを内蔵している品種も存在します。そういえば、たしか赤外線リモコン用キャリア発生回路や赤外線リモコン受信プリアンプなどリモコンには欠かせない機能を内蔵した品種もあったなぁ。当然これらの品種は低電圧で動作しています。

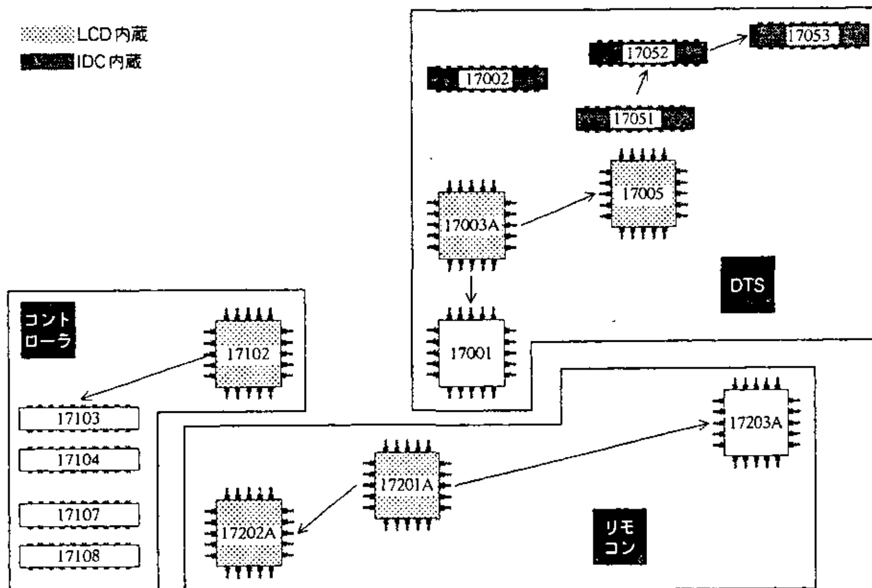
『本当に最近のリモコンは、操作案内がLCDに表示されたり、予約の内容をリモコンが記憶したりと、すごく高機能だなぁ』と思っていたらもうこんなのは古い話で、最新の情報では、他のリモコンの送信内容を記憶することができる学習リモコンがあるそうです。とくれば、もうこのリモコンに使用するコントローラは、SRAM内蔵の17Kシリーズしかありません。

と、ここまで17Kシリーズの高機能、高性能を唱ってきましたが、どうです？ 17Kシリーズはあなたの設計にぴったりのコントローラでしょう？ え、高機能すぎる？ 大丈夫、委せてください。実をいうと、17Kシリーズの本当に凄いところは、これらの高機能、高性能をまるで1つの部品のように付けたり、外したりできるということです。IDCもLCDもいらない。選局する訳ではない。リモコン送信する訳でもない。という方にも、『ほんとに小さなコントローラがあればいいんだよぉ』という方にも、17Kシリーズは使って戴けると思います。

17Kシリーズは、家電製品や玩具などの電子制御化にも使えます。たとえば、炊飯器、電子レンジなどのコントローラとしてです。最近の家電製品の広告では一頃のように『マイコンXX』というのがなくなってきました。しかし、マイコンを内蔵した製品が売れないから広告がなくなったのではなく、マイコンを内蔵して高機能、高操作性を実現するのが当り前になってしまったから、わざわざ『マイコンを使用している』という広告を打たないそうです。ですから壊れた家電製品を見かけたら、分解してみてください。きっと何らかのマイコンが使ってあると思います（でも、実際に分解してみる人いるのかなぁ）。

以上17Kシリーズの宣伝の一部でした。ついでに次に17Kシリーズのカタログにあった製品展開の図を掲げておきますのでよろしかったら参考にしてください。

◆17Kシリーズの製品展開



●17Kシリーズが目指すもの

17Kシリーズは、世界を目指す・・・、訳ではありませんが、海外でも結構引合いが多いそうです。でもここではそういう話ではなく、17Kシリーズの目的みたいなものを少し話してみたいと思います。17Kシリーズが目指すものを説明するためには、本書を最後まで読んでもらうのが一番いいのですが、いくつか専門的なことも理解してもらわなければならないので、ここでは基本的なことに絞って説明します。

◆設計者のために

17Kシリーズは民生機器に最適な4ビット・シングルチップ・マイクロコントローラです。現在ではさまざまな機器に組み込まれていますが、基本的にはDTS、家電、リモコンなどです。
あらかじめこれらの各分野に最適な基本部分を用意し、オプションを簡単に内蔵できることを目指しています。このように基本部分とオプションを分離することには、いくつかのメリットがあります（利点がなければ今まで一緒だったものをわざわざ分けませんて)。こうするとオプション部分の周辺ハードウエアの拡張性が向上します。まぁよく考えて見れば当り前の話です。新しい周辺ハードウエアが必要になったら、その部分だけを新たに作って継ぎ足せばいいのですから。

17Kシリーズは応用範囲を絞った特定応用分野用のマイコンです。そのために周辺回路をたくさん内蔵するわけです。逆にいえば、周辺回路をたくさん内蔵して始めて特定応用分野用のマイコンになるともいえます。また、分離できるということは、必要がなければ内蔵しなくてもいいということです。これにより、民生市場で最も重要視されるといっても過言ではない『コスト』に大きな影響を与えることができます（もちろん安くすることができるということですよネ、きっと）。17Kシリーズが目指すものは、不要な機能を提供しないことです。

17Kシリーズは、汎用マイコンと比べると、あきらかに特定分野を意識した専用マイコンであるといえます。しかし、普通の専用マイコンにありがちな他の目的には使いにくいという問題が発生しにくい専用マイコンであるともいえます。ですから特定分野に依存することなく、さまざまな分野に応用することが可能なのです。どうです、あなたの設計にぴったりでしょう。

当然、専用マイコンなのですから、DTSなどのあらかじめ用意された目的に使用する分には、なんら問題はないでしょう。17Kシリーズが目指すものは、常にあなたの設計にあった品種を提供することです。

◆プログラマのために

17Kシリーズは専用マイコンですから、結構たくさんの品種が存在します。それらの機能はおのずと違っています。ある製品はROMが24Kで、ある製品は16Kと。でもこれからプログラムを書くあなたやこれから本書を読むあなたは、そんなことは気にしなくてもいいのです。品種がいくつあろうと17Kシリーズのアーキテクチャは1つだけです。デバイスごとに記述する命令や指定の順序が違うなんてナンセンスなことです。17Kシリーズでは基本となる命令はすべて統一されています（いくつかデバイスに依存する命令も存在しますが、それらはデバイス固有の機能を使用するためのものです）。デバイスに依存しないプログラムを記述するための疑似命令やマクロも用意されています。17Kシリーズが目指すものは、デバイスに依存しないプログラムを書ける環境を提供することです。

最近では要求仕様が高度になってきているのでプログラムの開発も簡単ではないでしょう。でもプロセッサの開発当初から、そしてこれからさらに力を入れていく分野にプログラム開発環境の整備があります。17Kシリーズの開発環境はすでに現在でもどんな言語、プロセッサにも劣らないものがあります。17Kシリーズのツールは17Kシリーズのデバイスにベストマッチしています（あ、これは当り前か。17Kシリーズの開発ツールは17Kシリーズのプログラムを開発するために作られたのだから）。17Kシリーズの設計者たちは常にプログラマのことを考えています。

17Kシリーズはあなたに余分なことをさせません。ドキュメントも変数のリストも自動的に作成されます。17Kシリーズはプログラマに無駄なことをさせません。1つの修正をするのに、あっちを直してこっちを直して、とする必要はありません（要するにソースとオブジェクトの対応をとるのはツールがやってくれるということです）。17Kシリーズはプログラマに過大な要求をしません。誰でも簡単に使えるツールを提供します。もうあやしげな呪文を丸暗記する必要はありません。あなたがしたいことをコンピュータが呈示してくれますから。17Kシリーズが目指すものは、誰でも簡単に高度なプログラミングができる環境を提供することです。

こんな楽なプログラム作成をやってみません？

◆そして・・・

そして、17Kシリーズが目指すものは、民生用特定分野向けコントローラです。特定分野とは

DTS 赤外線リモコン タイニ・コントローラ 小型家電用コントローラ

などです、現在では。でも、実をいうとこれから先17Kシリーズが何に組み込まれていくか、よく分からないのです。最初はDTSとして出発したはずのものが、今ではこんなにさまざまな機器に組み込まれているのですから。『特定分野』といっても『特定分野で最適な力を発揮できるように設計された』という意味で、特定分野でなければ使えない、特定分野に依存している、という意味ではないのですから。
結局17Kシリーズが何に組み込まれるかを決定するのはあなたの設計です。17Kシリーズではいつでもあなたの設計にぴったりの品種を用意しているはずですから。

あと数年もしたら、プロセッサの開発者が予想もしなかったところで17Kシリーズを見つけることができるかも知れません。そしてもしかしたらそれは、本書を読んで17Kシリーズを使って製品を設計、プログラミングしたあなたが作ったかもしれないのです。

では、これから本書を読んで

17Kシリーズがあなたの設計にピッタリであることを

もしくは

17Kシリーズが製品の性能をアップしてくれること

をみつけてください。

◇コラム：17K シリーズのこわい話

17K シリーズの応用分野の中に一眼レフのカメラ、というのがありました。最近のカメラを触ってみたことがありますか？ 最近の一眼レフカメラは光学製品というよりもう立派な精密電子工学製品です。レンズは大型コンピュータで設計し、コンピュータで研磨する。ここにもう人手の入る余地はありません。本体には、マイコンを始め、IC、LSIがぎっしり詰まっています。現在の撮影モードなどは液晶で表示され、日付やデータやいろんな文字を映像と一緒にフイルムに写し込む機能まで付いています（ということは、この液晶のコントロールや写し込みのためにLSIが使用されているということです)。

これらのことはカメラのカタログを見ていればある程度は想像が付きます。本体にマイコンを使っていることは、『インテリジェントXX』という言葉をカタログから見つければ、だれでも想像が付きます。実際にあるカメラのカタログにはカメラの電子制御システムとして8ビット・マイクロコンピュータを使用していることを小さく唱っています。

でも、カメラの交換用レンズにまでマイコンが使用されているとは知りませんでした。レンズのさまざまなデータをレンズの中に納められたマイコンに記憶しておき、レンズがカメラと接続されたら、本体とのデータ通信を行ない、レンズのデータを本体に転送し、レンズに適切な撮影ができるように本体のマイコンを設定、という動作をするためです。このレンズに組み込まれたマイコンに 17K シリーズが使われているという噂です。

実際にいろいろな製品にマイコンが組み込まれているという『話』はあちこちで聞きます。でも実際に自分の知っているマイコンが自分の知っている製品に組み込まれているのを見ると、なにか懐かしいような、てれくさいような・・・。そして何よりも、ここまでマイコンが組み込まれるのか？、という驚きを禁じ得ません。こうなってくると、それこそ、マイコンが組み込まれない製品を捜すのが難しい様な気がします。

と、このようにマイコンは、どこに潜んでいるか解りません。
ほら、あなたの後ろにもマイコンが・・・
なんてはずもなく、あなたの前にもあるでしょう？
製品に組み込む前の 17K シリーズが。
これを使って、新しい製品を作りませんか？

17K series architecture

第2章

基礎知識、専門知識

17Kシリーズのアーキテクチャを理解することは簡単です。次の呪文を唱えてください。

『小さなCPUに豊富な周辺』

ホラ、もうあなたは17Kシリーズの専門家。もしこの呪文の意味を知りたかったらこの章を読んでみてください。コンピュータの基礎知識から17Kシリーズのアーキテクチャまで、いろいろと書いてありますから。

本章でももちろん17Kシリーズの話もしますが、どちらかと言えば一般論です。コンピュータとは、4ビット・マイコンとは、といった話や、コンピュータを理解するための基礎知識みたいな話がほとんどですので、基本的なことはご存知の方は次の章からご覧ください。

●4ビット・シングルチップを理解するために

まずは、コンピュータと4ビットの話です。今まで一度もコンピュータを見たことも触ったこともない方は、できればここからご覧ください。そうとう基本的なことから説明していますから。

◆まず、コンピュータとは

『コンピュータとは電子計算機です。』今時こんな説明で納得する人はいませんよね。では、『コンピュータとは電脳です。』これなら今っぽいでしょう。でも、パソコンやワークステーションをバリバリ使いこなしている人ならばともかく、17Kシリーズには、あまり相応しいとはいえません。あれ、でもそうすると17Kシリーズとは何なのでしょうか？ 他のコンピュータとはどう違うのでしょうか？

基本的にコンピュータができることは、データの移動、足し算、引算、0か1かの判断ぐらいのものです。これらの基本機能を組み合わせて高度な処理ができるようになっているのです。

その辺の事情に詳しい人に聞くと、『17Kシリーズは、4ビット・シングルチップ・マイクロ・コントローラである。』という答が返ってきます。確かに本書でも何気なくこの言葉を使ってきました。でも、『4ビット』とは、『シングルチップ』とは、どういう意味を持つのでしょうか？

まず『ビット』の問題からです。残念ながらコンピュータ関連製品を理解するのに、この『ビット』の概念を避けて通るわけにはいきません。だから、最初に説明してしまいます。『ビットとはあるかないか』です。え、いきなり話が哲学的になった、ですって。では、『ビットとは情報の記憶の単位です』これでどうです？ すこし科学的で何となく分かった気にはなるでしょう？

そもそもコンピュータとは、情報を記憶し、記憶した情報に基づいて何らかの動作を行ない、また新たな情報を記憶する機械なのです。どういうことかというと、たとえば、『1と2という数字を記憶し、足し算をして、3という数字を記憶する』ということを繰り返す機械なわけです。
コンピュータはものを記憶するのに『2進法』を使います。普段私たちが使っているのは10進法で、10個の数字を用いて数を勘定しています。でも2進法では、2個の数字、0と1しか使うことが許されません。先ほどの『あるかないか』です。なぜ、こんな面倒くさいことをしているかというと、まぁ理由はいろいろあるのですが、身も蓋もない説明の仕方をすれば、これが便利だからでしょう。
コンピュータは電気の力で動いています。だから電気を与えないとすぐさま動かなくなってしまうのです。この電気の電圧の高い低いで、あるかないかを記憶するようにしました。たとえば、電圧が高い場合はある、電圧が低い場合にはない、です。したがってコンピュータの記憶は『あるかないか』が使われているわけです。この『あるかないか』の単位がビットなわけです。

ちなみに、『あるかないか』だけではなく、うんと大きい、大きい、小さい、うんと小さい、といった多段階の記憶する素子の開発が進められているという話を聞いたことがあります。でもおそらく、こんなのが実用化されるのはうーんと未来か、もしかしたらいつまでたっても実用化されないかもしれません。

◆4ビット

では、4ビットとはなにを意味するのでしょうか。1つのビットで表わせる、記憶できることは、『あるかないか』に過ぎません。でもその『あるかないか』が4つ集まればどんなことを表現できるでしょうか? 『あるかないか』の組み合わせが4回にわたって行なえるのですから、ちょっと学生時代の順列・組み合わせを思い出して・・・。そうです、2の4乗で16とおりの組み合わせがあります。ですから16とおりの状態を表現できるはずです。ということは4ビットとは、16種類のデータを表現できることだともいえます。

ここで、4ビット・マイコンとは、何を意味するのでしょうか。これは、1度に処理できる情報の量が4ビットであるということです。先ほどコンピュータは、情報を記憶し、記憶した情報に基づいて何らかの動作を行ない、また新たな情報を記憶する機械、という説明をしました。この記憶-処理-記憶という一連の流れの中で、一回に処理される情報量が4ビットであるということです。

◆なぜ4ビットなのか?

先ほど17Kシリーズは4ビットのマイコンであるといいました。では、なぜ17Kシリーズは、4ビットなのでしょうか?

よくこのコンピュータの一度に処理できるビット数の概念を説明するのに、道路の車線を例にとった説明がされます。たとえば、1車線の道路ですと、一度に通れる車は1台だけです。これを2車線の道路にしますとこの道路を通れる車の量は2倍に脹れ上がります。コンピュータの基本の処理単位はビットですので、4ビットとは4車線であることになります。

交通量の多いところでは、車の流れを良くするために幅の広い道路を作ります。ならば、コンピュータも同じで幅の広い道路の方が情報の流れが良くなるのではないか？と思われたのではないでしょうか。たしかに4ビットより、8ビット、8ビットより16ビット、16ビットより32ビットの方が情報の流れは良くなります。事実ある分野においては、この情報量を競いあっています。

でも、17Kシリーズの応用分野では、必ずしも一度に処理できる情報量のみで競っているわけではありません。先ほどの道路の車線うんぬんという話は、実をいうとコンピュータの一般論としては通用するかもしれませんが、余り17Kシリーズに適した説明とはいえないのです。17Kシリーズに代表される世界をこのアナロジーに当てはめると、駐車場ではないかと。つまり、交通量は少なく、何台かの車が駐車ができれば十分、という世界です。

> つまり17Kシリーズの主目的は制御であり、データの転送ではないのです。データの転送も存在しないわけではないですが、フラグの判定の方が多い世界なのです。

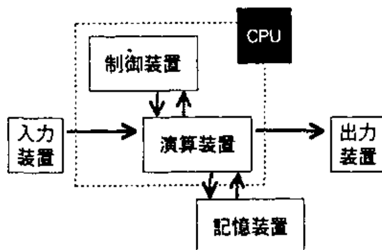
どこの世界に駐車場に8車線の100メートル道路を設置する人がいるのでしょうか？ましてや、17Kシリーズは、コスト・パフォーマンスの良さが要求される民生品の世界にいます。そんな無駄なことが許されるわけはないのです。

というわけで、17Kシリーズの世界、民生品の、制御系の世界では4ビットで十分という結論でした。ちなみに世界初のマイクロプロセッサは1971年に開発されたものですが、これも4ビットでした。

◆シングルチップとマルチチップ

シングルチップとはなにか、この問題はそんなに難しくありません。コンピュータの基本構成要素を1つのチップ上に組み込んでいるコンピュータを、シングルチップ・コンピュータといいます。

また先ほどの例に戻りますが、コンピュータとは、情報を記憶し、記憶した情報に基づいて何らかの動作を行ない、また新たな情報を記憶する機械、という説明をしました。では、記憶された新たな情報はどうするのでしょう。当然何らかの形でどこかにリアクションを返さなければ、せっかく動作した意味がありません。当然結果を出力する部分が必要になります。
また、あらかじめ記憶する情報がすべて揃っていれば構いませんが、そんなことはまれで、動作中に新たな情報が必要になる場合があります。したがって情報を入力する部分が必要になります。
そんなことから一般にコンピュータというものは、いくつかの部分から構成されていることがほとんどです。情報を入力する部分（入力装置）、入力された情報を記憶する部分（記憶装置）、入力された情報を処理する装置（演算装置）、処理された結果を出力する部分（出力装置）、そしてこれらの制御をする部分（制御装置）です（一般に制御装置と演算装置をあわせた部分を中央処理装置（CPU）といいます）。



これらのコンピュータの基本動作に必要なほとんどの部分を数センチ角の小さなチップに組み込んだものをシングルチップ・コンピュータといいます。そして、多くの場合、記憶装置を外部に持つものを、シングルチップとの対比において、マルチチップといいます。
シングルチップとマルチチップの関係は、こういった外面的な問題ですので、あまりどちらが優れているとか、どちらがどのようなことに向いているとかいうことはありません。しいていえば、製品に組み込む時には、シングルチップの方が余分な手間が省けるということぐらいです。
しかし、この問題に対しては17Kシリーズの立場は複雑で、分類上は当然『シングルチップ』です。しかし、17Kシリーズにおいては必要ならば、その外部に拡張RAMを接続することもできますので、場合によっては『完全なシングルチップ』というと、17Kシリーズの機能を正しく表現できない可能性もあります。

> ちなみに、コンピュータの開発初期、当時の技術では制御装置と演算装置を1つのチップに組み込むことができませんでしたので、いくつかの回路を接続して構成していました。CPUを始めて1つのチップに組み込んだのが、マイクロプロセッサと呼ばれるものです。これは電卓を簡単に作るために開発されたそうです。

◆専用マイコンと汎用マイコン

シングルチップとマルチチップの話は、どちらかというとコンピュータの見た目の話で、そのコンピュータが、1チップで構成されていようと、複数のチップで構成されていようと、基本的な機能に大きな差はありません（性能的にはいくらかの差が出るかもしれませんが）。しかし今度の専用マイコンと汎用マイコンの話は、直接機能に差が出てきます。というより、使い方に差が出てくるのです。

簡単にいってしまえば、専用マイコンとは、あらかじめ使用目的が想定されたマイコンをいいます。これに対して汎用マイコンとは、使用目的が想定されていないマイコンを指します。したがって、汎用マイコンで専用マイコンの肩代りをすることができないわけではありません。もともとマイコン＝コンピュータは、汎用性が売り物のはずです。にもかかわらず、専用マイコンというものが存在するのは、それなりの存在意義があるからです。

たしかに汎用マイコンは応用範囲が広いのがその特徴です。その結果品種が少ないことがあります。しかし逆にいえば、広すぎて目的にそぐわないケースや簡単には応用できないケースもしばしば発生します。ようするにおびに短し、たすきに長しというわけです。

それに引き替え専用マイコンは、『専用』なのですから目的にそぐわないわけがありません。あらかじめ、使用目的を想定することにより、使用目的にあった機能、環境を提供します。その結果品種が多くなります。たとえば、汎用マイコンを使った機器の設計をしたことがある人に、17Kシリーズを見ていただくと、汎用マイコンでは自分で1から作らなければならなかったような機能があらかじめ用意されていることに驚くでしょう。でもこれは、17Kシリーズのみならず専用マイコンの世界では当り前の話なのです。なぜって、そのための『専用』なのですから。

> 汎用の世界では80xxとか68xxxとかいうプロセッサが一般に有名ですが、専用の世界ではあまり有名なプロセッサがあるという話は聞きません。これはそれぞれの使用目的にも関連しています。汎用マイコンはその性格上パソコン等に『も』搭載され、必要以上にプロセッサの機能が重要視されていますが、専用マイコンは機器に組み込んで使用することがほとんどで、普通はその機器の部品の一部として扱われ、プロセッサの機能は組み込んだ機器で使用され、一般のユーザからその機能が見えるということがないからです。

◇コラム：コンピュータの分類

1口にコンピュータといってもいろいろあります。17Kシリーズのように製品に組み込んで使うものもあれば、空調の効いた専用の部屋に鎮座ましましているコンピュータをあります。あるいは、机の上でぼろぼろになりながら毎日の酷使に耐えているコンピュータもあります。それぞれ、みな用途が違い、何かを何かで代用するということはまずありません。
ここではコンピュータを用途と性能により分類してみたいと思います。

まず、1番最初に出てくるのがメインフレームです。メインフレームという言葉は『本体』ぐらいの意味なのですが、あまりにも一般的になってしまったので大型の汎用コンピュータのことをメインフレームと呼びます。特にどこで何のために使われるということはなく、しいていえば、大量の端末装置を持ち、大容量ファイルを必要とするシステムのホスト・コンピュータとして使用されます。たいていコンピュータを保守する人がそばに付き四六時中管理しているのが普通です。価格も数億円するもので、さらに保守に経費がかかることから、大手の企業しか購入できないのが実情です。

このメインフレームに対して、科学技術計算に特化し、高速な演算ができるようにしたコンピュータをスーパーコンピュータといいます。これは本当に計算をするだけのために作られたようなコンピュータで、最近まではまともなオペレーティング・システムさえもありませんでした（必要なかった）。1台が数十億円単位でもあり、単独で所有することはまずなく、大学や公共機関が購入し、それを時間貸しするといった利用形態が一般的です。

メインフレームが高価で、一般の企業では手が出ないという背景のもと考案されたのが、このミニコンで、処理能力ではメインフレームに劣りますが、コストパフォーマンスでは優るとも劣りませんでした。最近ではメインフレームの生産、販売の伸び率が落ちてきている一方、ミニコンなどの伸び率は目ざましいものがあります。

マイクロプロセッサの誕生を受けて、考案されたのがワークステーション、パソコンで、その誕生時は、処理能力でミニコンの足元にさえ及ばないものでしたが、コンピュータを個人が所有し、管理するということのインパクトは絶大なものがあり、最近の高級ワークステーションでは過去のミニコンの性能をかるく凌駕するものまであります。また、以前は開発、設計、事務処理のためにワークステーション、娯楽のためのパソコンという住み分けがありましたが今では、ワーク

ステーションの低価格化、パソコンの高機能化もあり、分類が難しくなってきています。

最後になりましたが17Kシリーズに代表される組み込み型のコンピュータです。このコンピュータが、今までのコンピュータと決定的に違うのは、組み込み型のコンピュータが使用者からまったく見えないということです。他のコンピュータは、多かれ少なかれ、使用者の意識の違い、という差はあれ、意識してコンピュータを使用しているということには違いありません。でも組み込み型のコンピュータは、コンピュータそのものが利用者から見えないものです。この手のコンピュータは、コンピュータと表現されることも珍しく、マイコンとかただ単にコントローラとか表現されることもしばしばです。当然この場合のマイコンはマイクロ・コンピュータであり、マイ・コンピュータ（もうこれはほとんど死語です）ではありません。
また、コントローラは当然プログラマブルです。17Kシリーズはコントローラです。
また、他のコンピュータがそれだけで一つの完結した製品であるのに対して、組み込み型のコンピュータは、半製品であるといっても過言ではありません。大抵の場合、組み込み型のコンピュータは、そのプログラムをROMに格納します。ROMにプログラムを格納する作業は、多くの場合コストなどの理由から、コンピュータのメーカと協力して行なわれます。
これらの理由からこの種類のコンピュータは、一般の人の目に付きにくくなっています。他のコンピュータと比べると、ニュースとなる機会も少なくなっています。でも実際には世界中で一番多く生産、使用されているのが、この種類のコンピュータです。あるコンピュータは、月産数十万個生産されているといわれています。

●コンピュータを理解するために

だいたいコンピュータとか4ビットとか、イメージしていただけましたか？ ここからはコンピュータを理解するための基礎知識講座です。もしお暇ならご覧ください。

◆ハードウエアとソフトウエア

実をいうと、この問題は言葉の定義が非常に難しいのです。現在最も無難な答が、『形があるものがハードウエア、形がないものがソフトウエア』というぐらいです。
よく口にされるのが、『コンピュータは、ソフトウエアがなければ動かない』という意味の言葉ですが、これは逆も真なり、で、『ソフトウエアがあってもハードウエアがなければ動かない』。ということは、ハードウエアとソフトウエアは車の両輪みたいなもので、どちらがかけても動かないということです。

> 最初は、背反の関係にあったこの2つの言葉ですが、最近のコンピュータの世界では、ハードウエアとソフトウエアの中間の存在としてファームウエアと言葉まであります。

昔のコンピュータ業界では、ソフトウエアはハードウエアの付属品という程度の認識でした。しかし現在では、ソフトウエアの重要性も認識されています。

> コンピュータ業界で、始めてハードウエアとソフトウエアを分離して販売したのは、アメリカの超巨大コンピュータ・メーカ（っていえばもう分かりますよね）で、1965年のことです。

専用マイコンの世界で、このハードウエアとソフトウエアが話題になるのは、少し別の角度からで、要求された仕様をハードウエアで満たすか、ソフトウエアで満たすかという問題が発生した場合です。この辺がまた難しいところで、要求された機能を満たすだけならば、ハードウエアでもソフトウエアでもいいということがあるのです。しかし、要求された性能を満たすためには、ハードウエアでなければだめだということもあります。

ハードウエア、ソフトウエアの場合とも長所、短所がありますが、専用マイコンの世界で好まれるのは、ハードウエア化です。これは一般的にハードウエア化した方が信頼性が高いからです。これに加え、17Kシリーズの場合においては周辺ハードウエアの追加が行ない易いという特徴があるため、よりハードウエア化が好まれています。

◆データとプログラム

少しコンピュータをかじったことがある人ならば誰でも知っている重要キーワード、データとプログラムです（でもあんなものかじっておなかを壊さなかったかな？）。

とりあえずこれは、コンピュータの処理の対象がデータ、処理の手順がプログラムと定義しておけばいいでしょう。でも、コンピュータの内部ではすべて0と1です。データとプログラムの区別はありません。だからプログラム中にデータを記述しておくことが簡単にできる代わりに、間違ったプログラムを記述すると、データをプログラムと思い込んでプログラムが意図したとおりに動かなくなる可能性があるわけです。

> 未来のコンピュータでは、このデータとプログラムの区別ができるかもしれませんが、今のコンピュータでは記憶する領域を分離するぐらいしか方法がありません。

実をいうと専用マイコンで、データとプログラムの記憶領域が区別されているのは、もっと根本的な問題があり、専用マイコンは、プログラムはROMに、データはROMとRAMに記憶させます。ROMに記憶するのは、専用マイコンのプログラムは、一度作成してしまえば書き換える必要がないからです。したがって、プログラムとプログラム中に記述した固定データはROMで構成されたプログラム・メモリに記憶され、プログラムの処理中に変更されるデータはRAMで構成されたデータ・メモリに記憶されます。

> ROM (Read Only Memory)：読み出し専用の記憶装置。最近では、紫外線を使用しデータを消去可能なUVEPROM(Ultra-Violet Erasable and Programmable ROM)、電気的な操作でデータを消去可能なUVEEPROM (Ultra-Violet Electorically Erasable Programmable ROM) などがありますが、どちらにしても開発段階でプログラムを変更するためのものです。
> RAM (Random Access Memory)：随時読み出し、書き込みが可能な記憶装置。

ちなみに17Kシリーズでは、プログラムの他に変更の必要のない固定データも一緒にROMに格納されます。

◆アドレス

では、コンピュータの内部に記憶されたデータというのは、どのような形で管理されているのでしょうか？

先ほど、『4ビット・マイコンは、一回に処理される情報量が4ビット』という説明をしました。ということは、1回に処理される情報量を基準として内部に記憶された情報の管理を行なえば、効率的に情報の管理ができるということです。

たとえば、書籍はページを単位として情報の管理を行ない、それぞれのページには順番に番号が振られています。あとで、あるページを参照したいときに、『あそこのページ』や『あれが書いてあったページ』などと表現しても、すぐには分からない場合がほとんどです。『XXページ』という表現をすれば、ほとんどの人に分かってもらえます。

また、別の例を挙げれば、人がどこかに出かけるときの行き先を表現するのに、『家を出てから右へいって、左へいって、何メートルあるいて』と表現する人は、余りいません。行き先に名前が付いていれば『XX郵便局』とか表現しますし、特に名前が付いていなくとも、全国共通の住所表記法である『XX町YY番地』という表現をすれば、誰にでも分かってもらえます。

同様にコンピュータでは、内部に記憶された情報の管理に『アドレス（番地）』という概念を使います。単位は、先ほどの『一度に処理できる情報量』です。

17Kシリーズでプログラムを記述する際には、できるだけこのアドレスを意識しなくてすむように設計されていますが、全く無視することはできません。なぜなら、原則としてプログラムの一番最初で、プログラムを格納する先頭アドレスを定義しなければならないし、プログラムの処理中に変更されるデータは自分で格納場所を確保しなければならないからです。

◆高水準言語と低水準言語

よく一般の人に誤解されているのが低水準言語という言葉です。コンピュータの世界では、高水準言語が機能が高くて、低水準言語が機能が低いということはありません。ここで、水準が高い、低いというのは、自然言語（人間が普通に話す言語）と比較して自然言語に近い方を高水準言語といっているだけです。先ほども触れましたが、コンピュータが扱えるのは0と1だけですので、理解できるのも一定の規則に規則に基づいて記述された0と1からなる言語だけです。これを専門用語で機械語と呼んでいます。

でも、これでは人間がほとんど全く理解できない（中にはコンピュータの種類と並んでいる数字だけで何をするプログラムなのかを理解してしまう人もごくわずかいることもいますが…）ので、開発された言語が17Kシリーズが採用しているアセンブリ言語です。これは原則として機械語と1対1で対応しているため、人が記述したとおりに機械語に変換されます。

これに対して高水準言語では、人に解り易い表現を用いるため、機械語と1対1で対応している部分は少ないといえます。また、翻訳プログラム（コンパイラ）を介して機械語に翻訳するため、アセンブリ言語と比較すると、大幅に冗長な機械語を生成する場合がほとんどです（少なくとも、ある程度適切に記述されアセンブルされた機械語よりコンパクトな機械語をコンパイラが生成することは不可能です。もし万が一そうである場合は、そのコンパイラが不正な翻訳を行なっている可能性があります。もちろん、著しく不適切な記述をされアセンブルされた機械語ならば、コンパイラが生成した機械語の方がコンパクトな場合もありますが…）。ようするに、アセンブリ言語の方がより人間の指示に忠実であるということです。

> 高水準言語が始めて開発されたのは1954年で、科学技術計算用のFORTRANという言語でした。しかし、当時はソフトウエアの技術もコンピュータの機能も決して高くはなく、高水準言語といっても当時のアセンブリ言語よりもより高水準であるというだけの話です。

機械語と1対1で対応しているため、可読性が低いのが問題点だったアセンブリ言語ですが、最近のソフトウエア技術の発達、疑似命令、マクロ機能などにより初期のアセンブリ言語より格段に可読性は高くなっています。

専用マイコンの世界では、ハードウエアに密着したプログラムを記述することが要求されるため、アセンブラが用いられる場合がほとんどです（最近C言語によるプログラムも出てきましたが、あまり一般的ではありません）。

●コンピュータの専門知識

ここまできたら、あと少し我慢していただいて、コンピュータの専門家になっていただくしかありません。コンピュータが理解できればこれから先恐いものなしで、17Kシリーズについても簡単に理解できます。

◆ニブルとバイト

この2つの内、バイトは一般的にも有名ですが、ニブルはあまり有名とはいえません。
8ビットを1単位とした数値をバイトといいます。4ビットを単位とした数値をニブルといいます。当然4ビット・マイコンでは、ニブルを単位として数値を表記したほうが、システムの諸機能を正しくイメージできるはずですが、一般的にはバイトの方が圧倒的に有名なので、バイトを単位と表現することもしばしばあります。

通常のワードの定義は、CPUが一度に処理できるビット数、なのですが、17Kシリーズでは多少事情が異なっています。17Kシリーズでは、一命令で16ビットのデータを扱うことができるので、1ワードは16ビットとなっています。
しかし、1つのアドレスに格納できるデータの大きさは4ビットなので、データ・メモリの大きさの表現には、ニブル（もしくはAA×4ビット）という表現を使い、プログラム・メモリの大きさの表現には、ワード（もしくはBB×16ビット）という表現を使います。
さらに、プログラム・メモリの場合は、ステップという単位も使用します。これは、プログラム・メモリには、データとプログラムの両方が格納されるためで、データの大きさの表現には、ワードを使用しますが、プログラムの（特に論理的な）大きさの表現には、何命令に相当するかを把握するためにステップという単位を使用します。でも、17Kシリーズでは、1命令の大きさも16ビットなので、ワードとステップの物理的な大きさは同じです。ただ、対象とニュアンスが少し違うだけです。

◆プログラム・カウンタ

プログラムとは、手順であると説明しました。最初から順番に実行するだけでよい手順もあるでしょうが、それ以外の場合が必要になる手順もたくさんあります。一般に次の3つのパターンを使えばすべての手順は記述できる、といわれています。

順次処理 分岐処理 反復処理

ようするに順番に実行する（順次)、条件によって実行したり、しなかったりを選択する（分岐)、同じ動作を繰り返す（反復）の3つのパターンだけで、プログラムを記述することができる、ということです。

順番に実行するだけなら、命令を読みとって、逐次それを実行すればいいのですが、場合によって実行しなかったり、同じ動作を繰り返したり、する場合には、現在実行している命令が、全体のなかでどういう位置付けなのかを把握していなければなりません。この仕事をするのがプログラム・カウンタ（Program Counter）です。

プログラム・カウンタは常に次に実行する命令のある箇所を記憶しています。

プログラムが順番に実行されようと、いきなり他の箇所のプログラムを実行しようと、プログラム・カウンタの値はCPU（制御装置）が管理し、自動的に変更しているのです。

◆アドレス・バスとデータ・バス

命令のある箇所が解ったら、命令をメモリから読み込みます。このときにCPUは命令のあるアドレスを指定しなければならないのですが、このメモリのある箇所を指定するために使用されるのが、アドレス・バスです。CPUはこのバスによって、メモリなどと接続されています。命令が格納されているアドレスをこのアドレス・バスに乗せて送り出す（といっても、アドレス・バスは一方通行なので送り出すことしかできません）と、格納されていた命令がデータ・バスに乗って送られてきます。

データ・バスに乗って送られてきた命令は、CPU内部で解析され、必要ならばメモリなどに書き込んだり、メモリなどから読み込んだりします（データ・バスは、アドレス・バスと違い一方通行ではありません)。

また、コントロール・バスというバスもあり、色々な制御信号が流れています。

> という構成が汎用マイコンなどでは、一般的だったのですが、最近ではちょっと違ってきています。アドレスやデータの情報を別々のバスに乗せて送り出せば、高速なバス・アクセスが可能、パイプラインなどの対応が容易などのメリットがあり、最近はこの方式が流行のようです。このように、内部において、複数のバスをひく方式をハーバート・アーキテクチャというそうです。

17Kシリーズでは、インストラクション・バスというバスがあり、アドレス・バスで指定されたメモリがプログラム・メモリならば、このインストラクション・バスに乗って命令が送られてきます。当然、アドレス・バスで指定されたメモリがデータ・メモリならば、データ・バスに乗ってデータが送られてきます。

また、17Kシリーズでは、インストラクション・バスにバンクなどの情報を乗せて、配線の減少を図っています。このように1つのバスに複数の情報を乗せて使うことを多重化利用のバスと呼びます。

バスの構成は、プロセッサによって、少しずつ違い、一概には説明できませんが、バス上を流れる信号の種類で分類するのが一般的なようです。

◆インストラクション・デコーダ

インストラクション・デコーダでは、命令の解析を行ないます。
アセンブラのプログラムは機械語に変換されROM内に格納されています。よくある説明では、機械語こそがCPUに理解できる唯一の言語である。となっていますが、これは他の言語、つまり高級言語やアセンブラなどは、CPUに理解させるために機械語に翻訳する必要があるからです。機械語とて、何が記述されているのか解析をしなければなりません。この機械語を実際に解析するのがインストラクション・デコーダの仕事です。機械語は、もうこれ以上他の言語に翻訳されないという意味で「唯一の」ということです。

人間が理解しやすい言語＝アセンブリ言語で書かれたプログラムは、アセンブラにより0と1の数字でのみ記述された機械語に変換（アセンブル）されます。アセンブルされた機械語は、メモリ上（専用マイコンでは通常ROM）に格納されています。普通は、ここから先の内部の動作を考慮する必要はなく、あとは機械まかせ、CPUまかせでいいのですが、ここではごく簡単に説明しておきましょう。

まず始めに、プログラムありき、です。これから実行するプログラムがどこにあるかを示しているのが、プログラム・カウンタですので、これが示すプログラム・メモリの内容を読み取り、インストラクション・デコーダに渡します。
インストラクション・デコーダは、渡された命令を解析し、必要ならば、解析の結果やオペランド・アドレスをALUなどに渡します。それから1つの命令が実際に実行されるわけです。

プログラムを読み取り、解析、実行という流れは特に17Kシリーズの特徴というわけではなく、すべてのCPUが実行していることです。

◆アキュムレータ方式と汎用レジスタ方式

演算および転送の方式には、2とおりの考え方があります。アキュムレータ方式と汎用レジスタ方式です。どちらの方式を採用するかによって、プログラムの書き方からその特性まで変わってきます。
アキュムレータ方式の特徴は、

> データをアキュムレータにロード（読み込み）し、アキュムレータとメモリ、アキュムレータとレジスタで演算を行ない、アキュムレータに格納されたデータをメモリにストア（格納）する。

といった動作を行ないます。この方式の長所は一般にROM効率にあるといわれています。被演算データの内の1つがアキュムレータに格納され、演算の結果もアキュムレータに格納されることを『常識』とすることで、命令のオペランドには、被演算データの内の1つだけを記述するだけでいいからです。命令として記述しなければならないことが減れば、それだけROM効率はよくなりそうです。このことから、この方式によるアドレスの指定を暗黙的 (Inharant) アドレシングといいます。

この利点の反面、ソース・プログラムの可読性は低くなり (2、3行通して眺めてみて始めて何の演算をしているのか理解できる)、アセンブラの可読性の悪さという弱点にますます拍車をかける結果となっています。

一方、汎用レジスタ方式の特徴は、

データ・メモリと汎用レジスタで演算を行ない、結果をメモリに格納する。

といった動作を行ないます。この方式では、命令のオペランドに汎用レジスタとデータ・メモリを一度に指定します。このことからこの方式によるアドレスの指定を1½アドレシングといいます。

したがって、長所、短所はちょうどアキュムレータ方式と反対で、ソース・プログラムの可読性が高くなる (1行読めばどういう処理を行なっているかは理解できる) 反面、命令として記述しなければならない項目が増え、それだけROM効率が悪くなるといわれています。

しかし、このROM効率の問題は、立場が逆転するほどのことはないとしても、世間で喧伝されているほどの差はないともいえます。アキュムレータ方式では、演算の前後に転送命令が必要です。このため、プログラムの記述行数が増え、ROM効率の差も埋まります。さらに速度の面まで考慮すれば、転送命令が必要のない分、高速に演算結果を得られることがあります。

最近の新しく設計されたプロセッサでは、汎用レジスタ方式を採用するプロセッサが増えています。という前振りをしたからには当然17Kシリーズは、汎用レジスタ方式を採用しています。

◆サブルーチンとマクロ

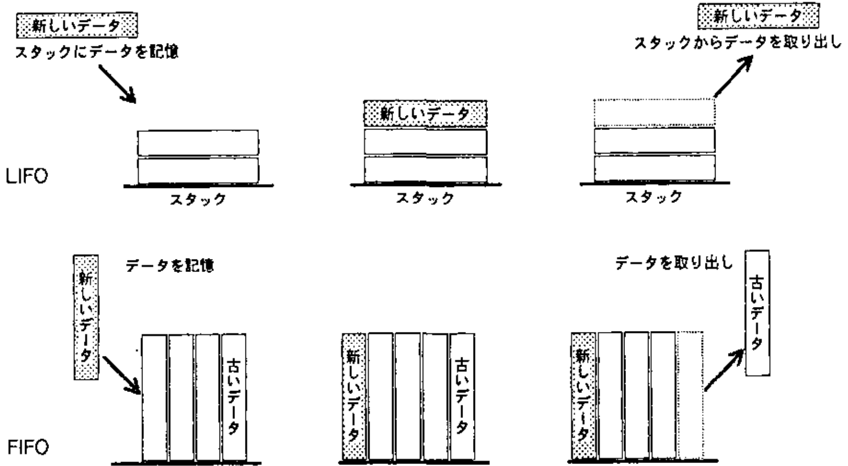
マクロとサブルーチンは、ソース・プログラム上では似ていますが、メモリ上では全く違います。これは、アセンブル・リストなどを見るとすぐに解ることですが、マクロは参照するたびごとにアセンブル・リスト上に現れてきますが、サブルーチンは何回参照しても現れるのは一度だけです。これは、メモリ上でも同じことで、マクロは定義した内容をそのまま、参照したところに展開しますが、サブルーチンは、定義した箇所に一度だけ展開し、あとは命令でジャンプしているだけです。

つまり、マクロはソース・プログラムを読み易くするためにアセンブラが提供する機能 (疑似命令) で、サブルーチンは、プログラム・メモリを飛び越えるためのCPUの機能 (命令) です。1つのマクロを多用するとプログラム・メモリが多量に消費されますが、1つのサブルーチンは何回使用してもプログラム・メモリの消費量はほとんど変わりありません。

◆LIFOとFIFO

両方ともデータの記憶方式を決めるものです。記憶したデータは、いつか使われる為に記憶されているわけです。この取り出し方を決めるのがLIFOとFIFOです。

LIFOといっても何のことか解らない方もいらっしゃるかもしれませんが、スタックというと、解っていただける方もいるのでは・・・。LIFOとはLast In First Outの略で、最後に入れたものを先に出すという動作をする記憶装置のことです。LIFOの例では、バネや干し草を使った図がいろいろな書籍などに掲載されているので、見たことがある人もいるでしょう。
これに対してFIFOとはFirst In First Outの略で、最初に入れたものを先に出すという動作をする記憶装置のことです。こちらは筒などを使った図が掲載されていることがあります。
別の表現をすれば、入口と出口が一緒なのがLIFOで、入口と出口が違うのがFIFOです。



LIFO、FIFOといっても要は単なる記憶装置でしかありません。通常のメモリと違うのは、通常のメモリは格納する場所を決めて格納し、任意の場所から格納したデータを取り出すのに対して、この手の記憶装置は、格納する場所をいちいち指定するのは面倒（一時的に記憶するだけ）なので、格納する場所ではなく、格納した順番を指定して格納します。そして、この順番の最後のデータから取り出すのがLIFO、最初のデータから取り出すのがFIFOです。

もう1つの特徴は、通常のメモリは一度格納したデータは、同じ場所に新たにデータを格納しないかぎり、いつまでも残っているのに対して、この手の記憶装置は一度データを取り出すと、データは残らないということです。一時記憶用なのですから、データを取り出したらあとは用はないはずですし、いつまでも不要なデータを残していたら、あっというまにが記憶装置がいっぱいになってしまい、使用できなくなってしまいます。

17K series architecture

第3章

小さなCPUとプログラム・メモリ

いよいよ、ここからは17Kシリーズ・アーキテクチャの本論です。今までと比較すると、いささか技術的な話になってしまう可能性があります。もし、解らない用語などがありましたら、巻末の用語解説などをご覧ください。

●17Kシリーズ・アーキテクチャの特徴

まずは、特徴からです。17Kシリーズの特徴は前に簡単に紹介しましたが、ここでは少し技術的な側面から説明したいと思います。

◆RISC指向と拡張命令

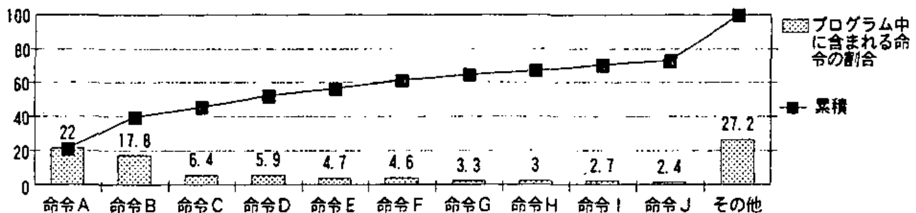
第1章で必ず説明するといいつつここまで説明を持ち越してしまったRISC（Reduced Instruction Set Computer）です。つまりCPUが用意している命令の数を整理､統合し､CPUが1命令を解析、実行するのにかかる時間を極力減らすことによりプロセッサの負担を軽減しようという考え方です。

RISCに対応する言葉として、CISCという言葉があり、これはComplex Instruction Set Computerという意味です。RISCの良さを強調するために、命令の複雑なコンピュータをやゆした言葉でもあり、あまりいい意味では使われません。

CPUのRISC化には長短両面があります。長所としては先ほどの、1命令当りの速度が速くなるということです。これはカタログ性能にも大きく影響するので、宣伝効果は絶大です。一方短所はというと、CPUが用意している命令の数が減っているため、アセンブラ、コンパイラの負担が大きくなるということです。

こういった、諸事情を踏まえつつも17KシリーズがRISC指向のアーキテクチャを導入したのは、それなりに裏付けがあってのことです。理由の1つは、17Kシリーズの祖先μPD1700シリーズのプログラムにおける命令の使用頻度に大きな偏りがあったことです。調査の結果によると、プログラムの70％以上がわずか10命令で構成されているとのことです。

μPD1700シリーズ命令使用頻度統計



また、RISCプロセッサの特徴の1つでもある固定長命令をすでにμPD1700シリーズで採用していたことも、要因の1つとなったようです。どういうことかというと、固定長ということは1命令で使用できる大きさが決っているということです。全体の大きさが決っていて、一部（=オペランド）を大きくしたかったら、他（=オペレーションコード）を削るしかありません。オペランドを大きくすれば、それだけ、広いアドレス空間を指定できるようになるのですから。

この結果、特に使用頻度の少ない命令の内、他の命令との組み合わせで、実現できる命令が削除されました。これにより、基本命令数は47命令となりました。この命令の数は、他のマイコンと比較しても大差ない数字です。実際、命令はその数が余り多くてもその使用に困りますし、逆に少ないと、いちいち基本的な機能からコーディングせねばならず、これもまた困り物です。結局は、プログラマが必要とする命令だけがあればいい（μPD1700シリーズと比較して削除された命令もありますが、プログラミングの便を考えて、追加された命令もあります）のですが、そう簡単にもいきません。

μPD1700シリーズは？：基本命令は94命令。

そこで、基本命令だけでは制御しにくい特殊な周辺ハードウエアなどが内蔵された時のことを考慮して命令の拡張性もまた、考慮されていました。
ちなみに、というか大変重要な話の様な気もするのですが、μPD1700シリーズのオペレーション・コードは6ビットでした。これが、17Kシリーズでは命令を減らした結果（最大）5ビットになったため、余った1ビットをオペランドに割り当てるようにしたそうです。その結果、μPD1700シリーズと比較して、アドレス空間の広さが倍になっています。

速くなって、かつ広いメモリ空間が得られるのですから、一石二鳥ということもできます。

基本命令の内の1つを拡張命令とすることで、残りの11ビットを使って命令を表現することで、命令を増やすことができるようにしたのです。当然11ビットしかないわけですし、基本命令の一部にもこの拡張命令を使用しているものがあるため、命令を増やす数にも限界がありますが、それでもオペランドがなければ1500近い命令を増やすことができますし、それぞれが4ビットのオペランドを必要としても、100近い命令を増やすことができます。基本命令の47命令でも十分なことを考えると、この拡張命令が足らなくなることは、永遠にないような気がします。

◆ASIC指向とASC

LSI（Lerge Scale IC）は普通、汎用LSIと特定用途向けLSI（ASIC；Application Specific IC）に分けられます。汎用LSIとは一般に販売しているLSIで、使用目的や使用者をまったく想定していません。これに対して、ASICは使用目的や、場合によっては使用者を特定したLSIです。そのため、特殊な機能を盛り込んだりすることができます。

さらに、ASICには、フルカスタムLSIとセミカスタムLSIという分類があり、フルカスタムLSIがほとんどすべてユーザ設計なのに対して、セミカスタムLSIはメーカーで用意した部品を使ってLSIを設計するといった違いがあります。セミカスタムLSIには、シングルチップ・マイコンの他、部品の完成度によりゲートアレイやスタンダードセルといった、いわゆるハードウエア・ロジック・システムがあります。

ハードウエア・ロジック・システムとシングルチップ・マイコンを比較すると、ハードウエア・ロジック・システムはハードウエアで要求仕様を実現させているのに対し、シングルチップ・マイコンはソフトウエアによって要求仕様の一部を実現することが可能になっています。

17Kシリーズの設計時、シングルチップ・マイコンには、次々と周辺ハードウエアが内蔵され、プロセッサが大規模化する傾向にありました。この対策として周辺ハードウエアをいくつかの機能ブロックにわけ、インテリジェント化し、全体をASICとして提供することが考えられたそうです。このインテリジェント化されたASICに17Kシリーズの開発者たちが付けた名称がASC（Application Specific Controller）だそうです。

◆キャッチフレーズ

17Kシリーズが開発された当時の設計方針は次の2つだそうです。

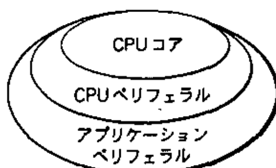
1.柔軟で大きな拡張性 2.コストパフォーマンスの向上

で、この2つの設計方針のもと、開発されたプロセッサのキャッチフレーズは

『小さなCPUに豊富な周辺』

となります。シングルチップ・マイコンでは、前述のように1つのチップにいろいろな周辺ハードウエアを内蔵します。周辺ハードウエアを変更することによりファミリー化を図り、色々な需要に対応するわけです。そして、すべての製品に共通な部分を『CPUコア』と呼びます。
CPUコアをできるだけ小さくすることにより、無駄な機能を使わなくても済むようにし、無駄なコストがかからないようにすることができるわけです。逆に、高機能を要求しても、周辺を内蔵していくことで対応できるはずです。

また、CPUを機能ごとに分割した個々のかたまりを機能ブロックといいます。
一般に他の専用マイコンは、ファミリーのすべてのマイコンに共通な『CPUコア』と品種によって構成が違う『ペリフェラル』という2つの階層を持っています。
ところが、17Kシリーズでは、『CPUコア』『CPUペリフェラル』『アプリケーション・ペリフェラル』という3つの階層を持っています。



インストラクション・デコーダ、ALU、
タイミング・ジェネレータ

レジスタ、プログラム・カウンタ

ROM、RAM、ADコンバータ

CPUコアには、インストラクション・デコーダ、ALU、タイミング・ジェネレータなどの機能ブロックがあります。最低限CPUを動かすためには、これだけでも構わないはずです（これだけの機能で実用にならないだろうとは思いますが・・・）。

そして、CPUペリフェラルには、レジスタ、PCなどが含まれます。ここでわざわざCPUコアと分け、CPU『ペリフェラル』と称しているのは、品種によっては、一部のレジスタを削ったり、PCの大きさを小さくしたりすることができるからです。

17Kシリーズのアプリケーション・ペリフェラルには、他の専用マイコンのペリフェラルに相当するROM、RAM、ADコンバータなどの機能ブロックがあります。

●CPUとインストラクション

人にあることを理解してもらうときは、そのものの根幹から説明するのが常套手段です。これにしたがって、CPUとインストラクション（命令）からです。

◆命令フォーマットとその種類

汎用の4ビット・マイコンは8ビット幅の可変長命令を用いていることが多いのですが、この方法は、命令の長さ、大きさを決めずに、オペランドに取りうるデータの大きさを8ビットに固定する方式です。この方式は、命令ごとに実行にかかる時間が違うことがほとんどです。また、オペランドのデータ幅が固定されているため、柔軟性に欠ける可能性があります。
これに対して、17Kシリーズの命令フォーマットは、μPD1700シリーズの昔から16ビット1語の固定長命令です。固定長命令の場合は命令の実行時間が一部をのぞいてほとんど同じになっています。
また、第2章でも説明したとおり、17Kシリーズでは、オペランドのアクセス方式として、アキュムレータ方式でなく、汎用レジスタ方式を採用しています。アドレスの指定も1½アドレシングです。したがって、17Kシリーズでは、メモリと16ビット即値を一度に演算することも可能になっています。これもμPD1700シリーズの昔から採用されていることです。

μPD1700シリーズと、それを継承した17Kシリーズで、この方式が採用されたのは、非生産的な転送命令をなくすこと、プログラムの読み易さ（Readability）を良くすること、ソフトウエアのバグを減らすことなどが考慮されたからだそーです。

17Kシリーズの命令をオペランドの数で分類すると、オペランドがない命令、1つのオペランドを持つ命令、2つのオペランドを持つ命令、という風に分類できます。2つのオペランドを持つ命令のほとんどは、2つ目のオペランドにレジスタか即値を指定します。
つまり、データ・メモリと汎用レジスタ、データ・メモリと即値を演算することは1命令で簡単にできますが、汎用レジスタと即値、汎用レジスタと汎用レジスタ、データ・メモリとデータ・メモリの演算を1命令で簡単に、というわけにはいかないのです。

17Kシリーズには47個といくつかの品種に依存した命令があります。これらのほとんどが5ビットのオペレーション・コードを持っています。同じニモニックでもオペランドの種類により、違うオペレーション・コードが生成されます。

17K シリーズの命令のオペランドで指定できるプログラム・メモリの大きさは

13ビット

です。でも、これに3ビットのセグメント・レジスタ（SGR）を付加することにより16ビットのアドレスがアクセスできます。つまり17K シリーズのすべてのプログラム・メモリを指定できるということです。17K シリーズの命令のオペランドで指定できるデータ・メモリの大きさは

7ビット

です。でも、これに4ビットのバンク・レジスタ（BANK）の値を付加することにより11ビットのアドレスがアクセスできます。

17Kのデータ・メモリ・アドレスの指定方法には大きく分けて4とおりの方法があります。

データ・メモリの直接アドレシング 汎用レジスタ・アドレシング データ・メモリのインデクス修飾アドレシング データ・メモリの汎用レジスタ間接アドレシング

詳細は『インストラクション・マニュアル』か『第5章　アドレシング』をご覧ください。

◆プログラム・カウンタ（PC）

17K シリーズのプログラム・メモリは64K ワードまで拡張可能です（このことに関してはプログラム・メモリの項目で説明します）。では、この64K ワードのプログラム・メモリの任意の1箇所（命令）を指定するためにはどうしたらよいのでしょうか？

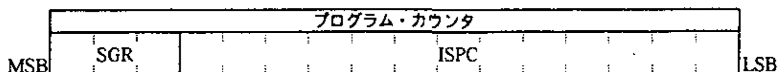
これも簡単ですね。$64 \times 1024 = 2^n$ ですから $n = 16$ ビットあれば、64K ワードのプログラム・メモリの任意の1箇所を指定することができます。

このプログラム・メモリの任意の1箇所を指定する（簡単にいえばこれから実行するプログラムがどこにあるかを示している）のが、プログラム・カウンタ（PC）です。

したがって17K シリーズのプログラム・カウンタは最大16ビットです。

ところで17K シリーズの命令のオペランドで指定できるプログラム・メモリのアドレスは何ビットでしょう？ 17K シリーズの命令は16ビットで1語の固定長ですから16ビットのオペランドを持てるわけがありません。答は13ビットです。

そこで、17K シリーズのプログラム・カウンタは最大の場合、2つに、3ビットのセグメント・レジスタ（SGR）と13ビットのセグメント内プログラム・カウンタ（ISPC ; Intra Segment Program Counter）に分かれています。



そして、他のマイコンのプログラム・カウンタですと、1つの命令が実行されるたびにプログラム・カウンタの値も1つ増加（インクリメント）されるのですが、17Kシリーズのプログラム・カウンタでは、1つの命令が実行されるたびにプログラム・カウンタの値も1つ増加される、のは変わりませんが、セグメント内プログラム・カウンタの値がいっぱいに（1FFFHから0000H）なってもセグメント・レジスタの値は増加されることはありません。

すくなくとも、システムの制御部に比較してメモリ容量の大きいプロセッサでは、いずれかの方法を用いメモリを分割しメモリの管理を行なっています。

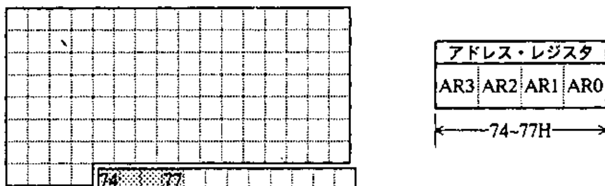
この結果、プログラム・メモリがどのように制御されるようになったかというと、17Kシリーズのプログラム・メモリは3ビットつまり8つのセグメントに分割され、一直線で連続的なメモリというのも、1つのセグメント内での話となりました。

ところで1990年現在、17Kシリーズで最大のプログラム・メモリは何Kワードでしょう。たしか先ほど書いた記憶があります。そうです24Kバイト＝12Kワードです。つまり、セグメント2枚しか使われていないのです。他の品種では、ほとんど1枚のセグメント内にすべてのプログラム、固定データが収まっています。では、セグメントが1枚もしくは1枚の半分という品種で、プログラム・カウンタがどうなっているかというと、セグメント・レジスタが存在しなかったり、セグメント内プログラム・カウンタが12ビットしかなかったりするわけです。
ここで簡単に、『プログラムの拡張性の為にセグメントがある』といってしまったら17Kシリーズを設計した人に怒られるかなァ･･･。どっちにせよ一直線で連続的なメモリ･･･というのはその程度のものです。それにプログラム・メモリ項目を読んでもらえれば、17Kシリーズでは簡単に64Kワードのアドレスを指定できるようになっていることが解ってもらえると思います。

◆アドレス・レジスタ（AR）

アドレス・レジスタ（AR）は、データ・メモリに存在するレジスタですが、説明の関係上早々とご登場願いました。アドレス・レジスタは、プログラム・メモリのアドレスを格納し、その内容をプログラマが操作できます。

つまり、プログラム・カウンタとアドレス・レジスタの違いは、プログラム・メモリの内容を格納する、のまでは同じですが、プログラム・カウンタはCPUが制御する（ジャンプやコールが行なわれると自動的に内容が変化する）のに対して、アドレス・レジスタはプログラマが意図的にその内容を操作し、使用するといった違いがあります。このアドレス・レジスタは、データ・メモリの74H～77Hにあり最大16ビットで構成されています。



アドレス・レジスタを使用する命令には、次のものがあります。

間接分岐
テーブル参照
スタック操作

プログラム・カウンタの大きさはプログラム・メモリの大きさにより決定されますが、アドレス・レジスタの大きさは、選択することができます。ただし、アドレス・レジスタがプログラム・カウンタより小さい場合は、プログラム・メモリのすべてのアドレスをアクセスできなくなる場合があります。

また、プログラム・カウンタは、命令を実行するたびに自動的にインクリメントされますが、アドレス・レジスタにそういう機能はありません。その代わりに、アドレス・レジスタをインクリメントする命令として

```
INC AR
```

という命令があります。

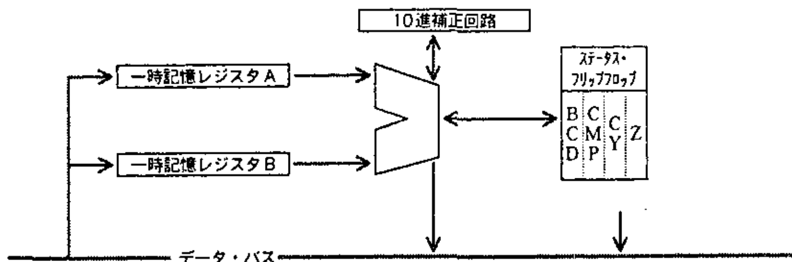
◆やっぱり電子計算機

コンピュータの最も計算機らしい部分の説明です。一般にコンピュータが抱かれているイメージ通りの仕事をするのがALUの仕事です。当然すべての品種が必要とする機能なので、CPUコアに存在します。
PSWは、データ・メモリに存在します。PSWは複数のフラグから構成され、すべてのフラグがすべての品種に存在するとは限りません。

CPUコアに存在するALUの説明とデータ・メモリに存在するPSWの説明が一緒になっているのを奇異に思われる方もいらっしゃるでしょうが…。
たとえば、汎用マイコンでは、データを格納するメモリと、CPUの実行を制御するために必要なメモリは、存在する場所が違うのが一般的です。それぞれのメモリの意味合いも、使い方も大幅に違うために普通は別々の場所に存在し、別々に説明してあります。
しかしシングルチップ・マイコンでは、1つのチップの中にCPUからメモリからI/Oまで、いっしょくたになって入っています。また、データ用のメモリの大きさも汎用マイコンなどと比べると小さいのが普通です。したがって、CPUの実行を制御するためのメモリとデータを格納するためのメモリも一緒になっていることが多いのです。
だからここでは、ALUとPSWを一緒に説明してしまいます。

◆ALU

ALUはArithmetic Logic Unitといい、4ビット・データの算術演算、論理演算、ビット判断、比較判断、回転処理などを行ないます。この仕事をするブロックをまとめてALUブロックといいます。17KシリーズのALUブロックは、ALU本体と、ALUの周辺ハードウエアとして、一時記憶用レジスタA、一時記憶用レジスタB、ALUの状態を制御するステータス・フリップフロップ、10進演算使用時の10進補正回路から構成されています。



ステータス・フリップフロップは

BCDフラグ用フリップフロップ CMPフラグ用フリップフロップ CYフラグ用フリップフロップ Zフラグ用フリップフロップ

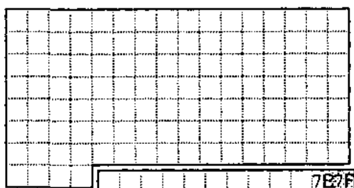
から構成されています。また、ALUの目立たない機能として、スタック・ポインタの操作があります。したがって、μPD1700シリーズと違い、スタック・ポインタを増減させる回路は必要ありません。

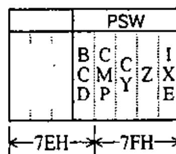
と、ここまではALUの説明だけだったのですが、ここからがデータ・メモリと関連してくるところです。このステータス・フリップフロップは、データ・メモリのシステム・レジスタ中のプログラム・ステータス・ワード（PSW）と1対1に対応しています。したがって、ALUで計算が行なわれた場合、ステータス・フリップ・フロップが自動的に書き換えられ、それにともなってプログラム・ステータス・ワードの内容も変化します。

◆プログラム・ステータス・ワード（PSW）

プログラム・ステータス・ワードは、ALUによる演算結果の状態を示すフラグとALUの実行条件を指定するフラグにより構成される5ビットのフラグの総称です。プログラム・ステータス・ワードはシステム・レジスタの7EHの3ビット目と7FHにあり、次のフラグで構成されています。

BCDフラグ（binary code decimal／hexadecimal） CMPフラグ（compare） CYフラグ（carry） Zフラグ（zero） IXEフラグ（*index addressing enable*）





このように、プログラム・ステータス・ワードはデータ・メモリ上にマッピングされているため、データ・メモリ操作命令で各フラグのセット、リセット等を行なうことができます。したがってプログラム・ステータス・ワードと1対1に対応しているステータス・フリップ・フロップもプログラム・ステータス・ワードの内容にしたがって変化します。

このうち、CMPフラグ、CYフラグ、Zフラグは、どの品種にも存在しますが、BCDフラグとIXEフラグは品種によっては存在しない場合があります。また、プログラム・ステータス・ワードの値はリセット時にすべて0にクリアされます。

ALUの実行条件を指定するフラグ

- BCDフラグ：　算術演算を2進で行なうか10進で行なうかを指定するためのフラグ
- CMPフラグ：　算術演算の結果をデータ・メモリ、汎用レジスタに格納するかしないかを指定するためのフラグ
- IXEフラグ：　インデクス修飾をするかしないかを指定するためのフラグ

ALUによる演算結果の状態を示すフラグ

- CYフラグ：　加減算実行後のキャリー、ボローの発生を示すフラグ
- Zフラグ：　演算の結果が0であることを示すフラグ

●プログラム・メモリ

コンピュータを構成する要素としては、CPU、プログラム、データ、I/Oなどがあります。17Kシリーズでは、プログラムはプログラム・メモリ、データはデータ・メモリに格納されています。またI/Oは、データ・メモリと深い関係にあります。ここでは、プログラム・メモリを中心に説明します。

◆プログラム・メモリの特徴

プログラム・メモリとはプログラムと定数データが格納される場所です。
現在の一般的なコンピュータは、あらかじめ作成されたプログラムによってその動作が確定されます。コンピュータの動作を決定するためのプログラムは、一度コンピュータに記憶されます。このプログラムを格納するための記憶場所をプログラム・メモリと呼びます。

> 昔のコンピュータは、昔の電話交換のように線をつなぎ変えることでプログラムを記憶していました（ほんと一に昔の話です）。これでは違う計算をする、つまりプログラムを変えるたびに線をつなぎ変えなければなりません。そこで（かどうかは知りませんが）現在では、プログラムを一度内部に記憶してから実行するようになっているのです。

専用マイコンのプログラムはROMという記憶装置に格納されます。これはメモリの製造工程で一度だけプログラムを書き込む（焼き付ける）ことができる記憶装置で、製造後にプログラムを変更することはできません。

これには、専用マイコンの使用目的は決っているので、汎用マイコンのように頻繁にプログラムを変更する必要がない、そのような状況でプログラムを変更するための装置を付けるという無駄なことはコスト上できない、などの理由があります。
一般にプログラム・メモリは、一直線の連続的なメモリとして構成されています。よくメモリの概念図としてメモリの構成を表形式で表現している図がありますが、あれは人間の理解を助けるために表形式で表現しているだけで、物理的には一直線になっています。
また、一直線のメモリはある単位ごとに「番地」が付けられています。

17Kシリーズのプログラム・メモリは、第2章のアドレスのところで説明したように、16ビット(=1ワード)単位で番地が付けられ、1つの番地に1つの命令が格納されています。そして実に

16ビット×64Kワード=1Mビット(128Kバイト)

まで拡張可能ですので、理論上は約65000命令のプログラムを記述することができることになっています。しかし、1990年現在でも17Kシリーズで実際に使用されたプログラム・メモリは24Kバイト程度という話ですので、128Kバイトを使い切ってしまうのは、いつのことでしょう?

μPD1700シリーズは?:プログラム・メモリは16ビット×2Kワード。

◆プログラム・メモリの大きさ

もうすでにプログラムを開発されている方は64Kワードものプログラムを開発しないといけないのかと、思われるかもしれません、64Kワードものプログラムの開発時間をとれるのかと疑問に思うかもしれません。
現在の開発環境で64Kワードものプログラムを作成するのは、いくら17Kシリーズの開発環境がすぐれているとしても、大変なことです。
でも将来的なものを考えた場合、このプログラム・メモリの大きさが大きすぎるとは言えないのです。その将来的なものを考えるキーワードとして次の2つがあります。

オンライン・マニュアル
ライブラリ

この2つに共通して言えるのが、プログラムを作るのはあなた一人ではないということです。専用マイコンのプログラムは、ROMに記憶されます。17Kシリーズではプログラムの他にプログラム作成時に決定された固定データも格納することができます。
最近のTV、VTRなどでは、次の操作のための案内を画面に表示します。この画面に表示するためのデータを格納するための空間としてプログラム・メモリを使用します。この考え方を発展させていけば、操作中に使用説明が参照できるように−−つまりオンライン・マニュアルが必要になります。オンライン・マニュアルのデータ量を考えた場合、実際にコーディングが必要なプログラム量は、微々たるものでしょう。

また、17Kシリーズでは組み込みマクロやマクロ・ライブラリなどの提供が積極的に進められています。提供されたライブラリがなくとも既に作成されたサブルーチンをもとにユーザ・ライブラリが簡単に作成できるようになっています。このようないわゆるライブラリ的な使い方をした場合、1つのプロジェクトであなたがコーディングしなければならないステップ数はわずかかもしれませんが、マクロやライブラリを多用した場合に作成されるオブジェクト・サイズは以外と大きなものになる可能性があります。

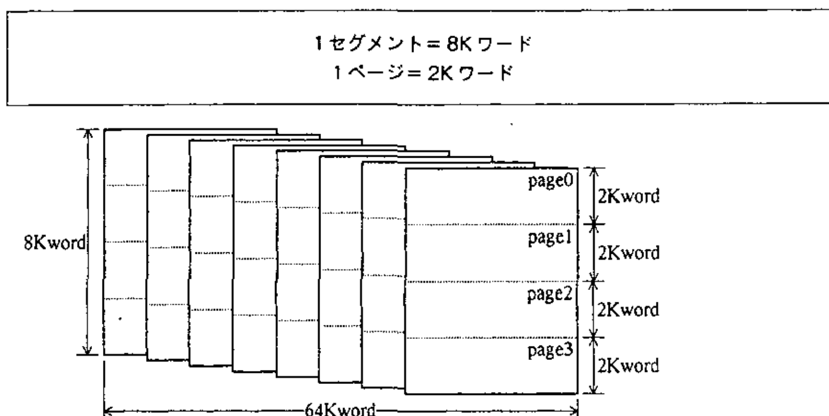
これは、開発ツールのところで説明すべきことですが、17Kシリーズはプログラムの開発の容易さ、作成したプログラムの読みやすさを大事にしています。開発の容易さや読みやすさのためにマクロやライブラリの多用は欠かせない要素の1つです。

◆プログラム・メモリの構成（セグメントとページ）

17Kシリーズのプログラム・メモリが64Kワードまで実装可能なことは解りました。しかしすべての製品で64Kワードものプログラム・メモリが必要な訳ではありません。512ワードもあれば十分な製品もあるでしょう。

当然、必要のないプログラム・メモリが載った品種を使うわけにはいきません。また、64Kワードすべてを簡単に指定できるわけでもありません。このためのプログラム・メモリの制御単位が『セグメント』であり、『ページ』なわけです。17Kシリーズのプログラム・メモリは8つのセグメントという制御単位で分割されます。また、各セグメントは4つのページという単位で区切られます。

1セグメント＝8Kワード 1ページ＝2Kワード



◆プログラムの分岐

通常プログラムの分岐には2種類あります。直接分岐と間接分岐です。分岐先を直接オペランドに指定して分岐するのが直接分岐、分岐先をどこかのアドレスに保存し、保存したアドレスをオペランドに記述するのが間接分岐、と読んで字のごとしの機能なのです。

17Kシリーズの場合、分岐先を保存するのは、アドレス・レジスタです。

BR命令による直接分岐の場合、同じセグメント内でしか分岐することができません。これは、BR命令のオペランドが13ビットの大きさしかないためです。13ビットで指定できる数値は2^{13}=8192ですから、BR命令では8Kワードの範囲、つまりセグメント内でしか分岐することができないのです。

同様に、CALL命令による直接分岐の場合は、さらに条件がきびしく、同じセグメント内の上位2Kワードの範囲にしか分岐することができません。ならばCALL命令のオペランドの大きさは何ビットでしょう？ 答は11ビットです。
13ビットのセグメント内プログラム・カウンタの上位2ビットには『ページ・ビット』という名前が付いていますが、これは、セグメント・レジスタと違い、命令が実行されるたびにインクリメントされ、セグメント内では一直線で連続的なメモリを構成しています。CALL命令で、同じセグメント内の上位2Kワードの範囲にしか分岐できないのは、オペランドが11ビットの大きさしかなく、プログラム・カウンタのページ・ビットには自動的に00Hが格納されるからです。
そして、このページ・ビットの値ごとにページ0からページ3までの4ページに1つのセグメントを区切るわけです。

当然、これらの直接分岐の問題は、間接分岐を使えば解決するのですが、その前にもう少し、この直接分岐について、考えてみたいと思います。

BR命令による直接分岐の場合、分岐先のページにより生成されるオペレーション・コードが違うという問題がありますが、このことが問題となるのはディバグ時に生成されたオブジェクト・コードに対して手でパッチをあてる場合ですので、パッチによる修正でSIMPLEHOSTを使えばこの問題は発生しません。

このことに関して詳しくはインストラクション・マニュアル等をご覧ください。

CALL命令による直接分岐の場合、その呼び出し番地、つまりサブルーチンの先頭アドレスはページ0（上位2Kワード）でなければなりません。この問題は高級言語を使ってきた方ですと、重要な問題に感じられるかもしれませんが、アセンブラをご存知の方ならば、それほど問題とは感じられないかもしれません。この問題はプログラミング・テクニックで大方解決できる問題だからです。
つまりここに、メモリの物理的な配置までコントロールできる（しなければならない？）アセンブラとそういったことはどうでもいい高級言語の違いがここにあるのです。
まず、アセンブラの分岐命令ですが、ほとんど局所的です。つまり分岐距離は短くなっています。たとえ遠方に分岐する場合でも、先にサブルーチンを定義すれば（=モジュールの後方=メモリの上位の方)、サブルーチンのほとんどをページ0に納めることができることになります。もしこの方法でも、サブルーチンの先頭アドレスがページ0に収まり切らない場合は、そのプログラムの4分の1以上がサブルーチンであるという可能性があります。その場合には、それこそ直接分岐を使えばいいのです。

プログラムを一度でも作成したことがある方ならば、理解してもらえると思いますが、一度にプログラムが分岐、参照しなければならない範囲というのは限られています。どれだけシステムが遠方への分岐、参照の方法を用意していても、1つの論理的なプログラム・ブロックで必要なのはごくわずかです。特にアセンブラではこの傾向が顕著になっています。

ちなみに、17Kシリーズのアセンブラである AS17K では、メモリの上位の方に配置されたモジュールに変更がなければ、アセンブル処理を行なわず、中間オブジェクト・ファイルを使いリンク処理に入るため全体のアセンブル処理が極端に速くなる可能性があります。このことからも、サブルーチンを前方に集めることが推奨されています。

◆間接分岐

間接分岐は、特に問題がなく、アドレス・レジスタに分岐先を格納すれば、本当に任意の番地に（セグメントに関係なく）分岐することが可能です。

BR命令やCALL命令による間接分岐の場合、プログラム・カウンタには、アドレス・レジスタのすべての値がコピーされます。つまり、プログラム・カウンタにあるセグメント・レジスタの値もアドレス・レジスタの値がコピーされます。当然ページ・ビットの値も変化します。
それよりも、間接分岐で重要なのは、データ・メモリの値により、分岐先を選択するテーブル・ジャンプが可能なことです。

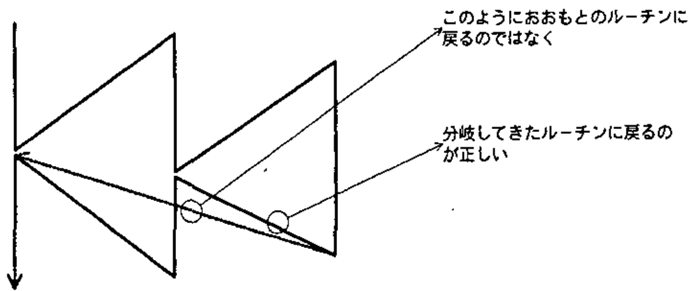
◆アドレス・スタック・レジスタ

先ほどから急にでてきたBR命令とCALL命令ですが、この2つがどういう風に違うか？ 本書は17Kシリーズの命令の解説を目的としているわけではないので、簡単に説明しますが、BR命令は一方通行、CALL命令はUターン可能、という違いがあります。
一方通行の方はこの際ほっておくとして、Uターンするためにはどうしたらよいか？ がこの項目のテーマです。実はこの問題も答が解ってしまえば簡単。出かけるときに、帰る場所を記憶しておけばいいのです。そのための記憶場所が、アドレス・スタック・レジスタです。17Kのスタック・レベルは最大16レベルあります。

μPD1700シリーズは？：スタック・レベルは最大3レベルです。

CALL命令が実行されると、CPUは現在のプログラム・カウンタの値に1を足した値をアドレス・スタック・レジスタに格納します。それからCALL命令のオペランドで指定されたアドレスもしくはアドレス・レジスタが示すアドレスをプログラム・カウンタにコピーし、プログラムは分岐先に移ります。
CALL命令からのUターンには、RET命令を使用しますが、RET命令が実行されると、CPUはCALL命令で格納された（と思われる）アドレス・スタック・レジスタの値をプログラム・カウンタにコピーし、プログラムはCALL命令の分岐前の箇所に戻ります、普通は。
普通でない場合というのが、どういう場合かというと、CALL命令で分岐したあと、RET命令で戻るまでの間に、このアドレス・スタック・レジスタの値を変更してしまった場合です。そうすると、元いた箇所に戻れる保証はなくなります。

で、このアドレス・スタック・レジスタの構成ですが、第2章で説明したスタックです。最後に入れたものを最初に出す（LIFO)、という動作をする記憶機構です。
これは、アドレス・スタック・レジスタで何をしなければいけないかを考えれば、当然スタック構成にしなければならないことは明白です。つまり、CALL命令で、分岐している最中にもう一度CALL命令を実行したらどうなるかを考えて見てください（CALL命令を一度だけに限るのもこの問題を解決する手段の1つではありますが・・・）。
分岐している最中にもう一度CALL命令を実行したら、その2番目CALL命令の分岐先から戻るのは、1番目のCALL命令の次の命令ではなく、2度目のCALL命令の次の命令です。



このためには、最後に記憶した内容が簡単に取り出せる方が都合がいいのです。

また、アドレス・スタック・ポインタのもう1つの機能として、アドレス・レジスタの内容をスタックに退避、復帰することができます。このために使用されるのがPUSH命令とPOP命令です。

◆スタック・ポインタ

というのが、ソフトウエアの方から見た話でして、実際問題メモリには、LIFOもFIFOもありません。あるのは記憶しているかいないかだけです。そこで、次に取り出す箇所を記憶しているのが、スタック・ポインタです。極論すれば、LIFOかFIFOかという違いは、この次に取り出す命令を記憶しているポインタの動作の違いだけです。スタック動作をするからスタック・ポインタと名前を付けているのに過ぎません。

このアドレス・スタック・レジスタとスタック・ポインタが物理的にどうなっているかというと、アドレス・スタック・レジスタは最大

16ビット×16ワード

で、スタック・ポインタは最大

4ビット

です。当然最大というからには、これよりも少ない品種があってもいいはずです。ただ、アドレス・スタック・ポインタのビット長（最大16ビットというやつ）はプログラム・カウンタのビット数と同じになっているはずです。これも当然といえば当然で、アドレス・スタック・ポインタはプログラム・カウンタの値を格納するためのものですから。

同様にスタック・ポインタの最大4ビットというのも、アドレス・スタック・レジスタの任意の位置を指し示す（ポイントする）ために存在するのですから、レベル数（16ワード）の任意の位置を指し示すビット数になっているはずです。

ちなみに、17Kシリーズには（どのコンピュータにもありますが）ALUという算術演算などをするブロックがあり、スタック・ポインタの増減はALUが行なっています。17Kシリーズは4ビットのマイコンですからALUで一度に処理できるビット数も4ビットでして、このALUで増減を行なうスタックのレベルも4ビットに制限されているわけです。

◆SPARELIB

8つのセグメントに分割されたうちの1つのセグメントをシステム・セグメントと呼び、SPARELIB（スペアリブ：Specific Application REsident LIBrary）と呼ばれるライブラリが格納される品種が予定されています。

このライブラリは、

特定の応用分野のためのサブルーチン
制御が複雑な周辺ハードウエアの制御用サブルーチン
上位品種ではハードウエアで実現されている機能を
プログラムで実現させるためのサブルーチン

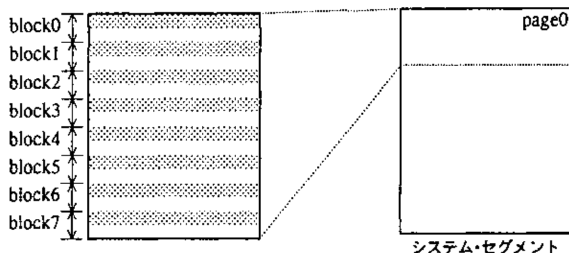
などのサブルーチンが格納される予定です。

システム・セグメントのページ0はさらに8つのブロックという単位で区切られます。

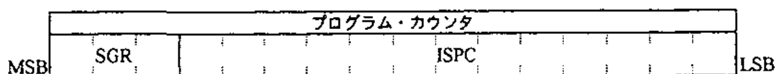
このSPARELIBというライブラリを簡単にアクセスするための命令があります。通常の直接分岐では、セグメントを超えて分岐することができないので、間接分岐を使いますが、これはアドレス・レジスタに分岐先を格納する必要があるため、1命令では実行できません。しかしこの命令は、どのセグメントにあっても1命令でシステム・セグメントに分岐し、SPARELIBを利用することができます。このことからSPARELIBのマクロ、サブルーチンは、普通はOSが備えている『システム・コール機能』と同様の感覚で使用できます。この命令は、SYSCAL（システム・サービス要求）と名付けられています。

システム・セグメントの各ブロックの先頭の16ワードがSYSCAL命令の入口となります。

1ブロック＝256ワード（128エントリ）



SYSCAL命令が実行されると、プログラム・カウンタは次のようになります。



セグメント・レジスタにはシステム・セグメントに割り当てられたセグメント番号が格納されます。SYSCAL命令はプログラムであらかじめ予定された一種の割り込み（ソフトウエア割り込み）です。このことに関しては、割り込みの項目をご覧ください。

17K series architecture

第4章

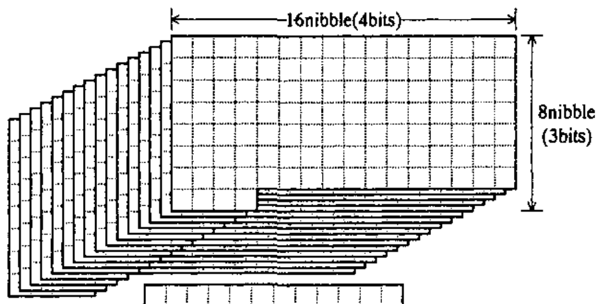
データ・メモリ

●データ・メモリ

データ・メモリは演算・制御のための変化するデータが格納される場所です。

専用マイコンのプログラムは、ROMに格納されました。でもこれだけですとあらかじめ決められた動作を繰り返すことしかできません。入力されたデータを格納する場所や、データ同士を計算するための作業用の場所などが必要です。これらの動作のための作業場所がデータ・メモリです。さまざまに変化するデータを記憶するために、RAMで構成されます。

一般にデータ・メモリは、一直線の連続的なメモリとして構成されています。しかし、17Kシリーズでは、人間の理解の助けになるように、下位4ビットを行（ロウ）、上位3ビットを列（カラム）としてとった『表形式』で表現しています。



そして、17Kシリーズのデータ・メモリは、目的別にいくつかのブロックに分割されます。これらのキーワードには次のようなものがあります。

> バンク、データ・バッファ、システム・レジスタ、レジスタ・ファイル、
> コントロール・レジスタ、ポート・レジスタ、汎用レジスタ

◆データ・メモリの構成とバンク

17Kシリーズの命令のオペランドで指定できるプログラム・メモリのアドレスは、最大13ビットでした。でも、プログラム・メモリの拡張性のために16ビットものアドレス空間を用意していました。では、データ・メモリの拡張性のためには何を用意しておいてくれたのでしょうか？

17Kシリーズの命令のオペランドで指定できるデータ・メモリのアドレスは、7ビットということになっています。したがって一度にアクセスできるデータ・メモリ空間としては、$2^7 = 128$ニブルということになります。このアドレス空間を拡張するために考案されたのが『バンク』という概念だそうで、データ・メモリをアクセスする場合に、オペランドで指定した7ビットに『バンク』の4ビットを付加すれば、11ビットのアドレス空間をアクセスすることが可能になり、拡張性も申し分なしです。

というわけで、17Kシリーズのデータ・メモリは4ビット＝16種類のバンクという単位で分割されます。そして、各バンクは128ニブルの大きさがあるわけですから、

128ニブル×16バンク＝8192ビット（1Kバイト）

まで実装可能ですので、理論上は約2000個の4ビットデータを格納することができるようになったわけです。しかし各バンクの70〜7FHの16ニブルには、メモリは実装されることはなく、システム・レジスタとポート・レジスタが割り当てられています。これにより実際に実装可能なメモリの容量は

（128－16）ニブル×16バンク＝7168ビット

となります。メモリが実装されない各バンクの70〜7FHの内70〜73Hの4ニブルにはポート・レジスタが割り当てられ、これにより実装可能なI／Oポートは、

4ニブル×16バンク＝256ビット

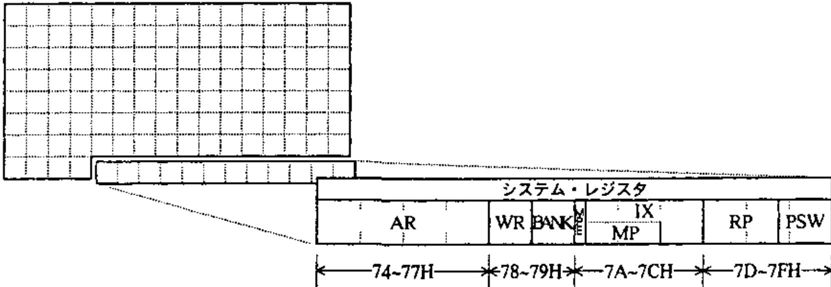
です。でも、残りの74〜7FHの12ニブルの12×16・・・とはいかずに、システム・レジスタという性格上、各バンク共通となっています。これは考えてみれば当り前の話です。システム・レジスタはシステムの情報が格納されるところですので、バンクごとに内容が違っていたらおかしなことになってしまいます。

μPD1700シリーズは？：データ・メモリは64ニブル×4バンク。

と、この様にバンクを利用することにより、広大なデータ・メモリ空間を提供してきてくれた17Kシリーズですが、これから将来（ずっと先の未来の話でしょうが）データ・メモリ空間が足らない、という事態が絶対おこらない、とはいえません。何せ進歩の速いこの世界のことですから。そんな時に、今までの資産をかなぐり捨てて、新しい製品に手を出す前にちょっと思い出してください。17Kシリーズのデータ・メモリ、RAMの大きさは、直接コントロールできるのが、7168ビットであって、周辺ハードウエアとしてRAMを接続すれば、さらにRAM容量を増やすことができるということを。実際にどうやってRAMを増設するかは・・・、今は内緒です。

◆システム・レジスタとバンク・レジスタ

さてその各バンクを通じて、たった1つしかないシステム・レジスタの話です。前述のように、システム・レジスタは、74〜7FHの合計12ニブルのレジスタの総称で、CPUの制御に直接関係するレジスタ群です。



システム・レジスタには次のようなフラグ、ポインタ、レジスタがあります。

> アドレス・レジスタ（AR)、ウィンドウ・レジスタ（WR)、バンク・レジスタ（BANK)、
> メモリ・ポインタ・イネーブル・フラグ（MPE)、インデクス・レジスタ（IXH、IXM、IXL）
> メモリ・ロー・アドレス・ポインタ（MPH、MPL)、レジスタ・ポインタ（RPH、RPL）
> プログラム・ステータス・ワード（PSW）

このうち一番最初のアドレス・レジスタと一番最後のプログラム・ステータス・ワードはすでに説明しました。
システム・レジスタの内容は、リセットがかかると、ウィンドウ・レジスタと、ハード的に1に固定されているビットをのぞいて、0がセットされます。でも、ウィンドウ・レジスタはリセット信号の影響を受けずに前の値を保持しています。
システム・レジスタの各フラグ、ポインタ、レジスタは、割り込みようスタックを持っている場合があります。少なくとも、割り込みがサポートされている品種では、IXEとBANKの割り込み用スタックがあります。

さて、このシステム・レジスタにバンク・レジスタというものがあることが解りました。バンクの利用により広大なデータ・メモリ空間が手に入ったことは事実ですが、あいかわらず17Kシリーズの命令で直接アクセスできるのは128ニブルのみです。したがって、場合によっては、バンクの切り換えを行なう必要があります。そのためのレジスタがこのバンク・レジスタです。
システム・レジスタの内容は各バンク共通になっている、という説明をしました。ということは当然バンク・レジスタの内容も共通ということです。これは当り前の話です、このバンク・レジスタの値を書き換えることで、バンクの切り換えを行なうのですから。

バンク・レジスタは、データ・メモリにありますので、データ・メモリ操作命令で、操作することができます。ただし、17Kシリーズ用のアセンブラであるAS17Kには、バンク操作用の組み込みマクロ命令として、

```
BANKn
```

という命令があり任意のバンクを指定することができます。プログラムの可読性を考慮した場合、組み込みマクロ命令を記述することが推奨されています。

またすべてのデータ・メモリのアドレスは00H～7FHですので、すべてのバンクのアドレスから任意のアドレスを指定したい場合は、バンク番号とアドレスの両方を指定する必要があります。AS17Kでは、データ・メモリにシンボルを割り付けて使用することが可能ですが、1命令で直接アクセスできるのは、バンク内のみです。したがって、シンボルが割り付けられたバンクと現在操作中のバンクが違う場合でも、エラーの発生や自動的なバンクの切り換えなどは行なわれません。プログラマの責任でバンクの切り換えを行なってください。

●データ・メモリと周辺回路

17Kシリーズは、周辺ハードウエアの増加などにも柔軟に対応できる拡張性があること…、という話が何度も出てきました。では、この目標がどのような形で実現されているのかを見てみましょう。

17Kシリーズは、たくさんの周辺ハードウエアを内蔵することができるように設計されています。なぜたくさんの周辺ハードウエアを内蔵するかというと、専用マイコンではそれが必要とされるからです。いかにしてたくさんの周辺ハードウエアを内蔵できるようにしたかというと、データ・メモリ上にマッピングする、という方式（Mapped I/O）をとったからです。このため、データ・メモリと周辺ハードウエアの関係は切っても切れない強い関係にあります。

専用マイコンに必要とされる周辺ハードウエアは、その高機能化とともに増加する傾向にあります。物理的に内蔵できる数も半導体プロセス技術の進歩により増加しています。でも物理的に内蔵できるのと、それらをすべて制御できるのとは、意味合いが違います。マイコンが制御できる周辺ハードウエアの数は、アーキテクチャにより決定されるといっても過言ではないのです。
一般的にはマイコンで周辺ハードウエアを制御する方法には次のような方法があります。

周辺ハードウエア制御専用の命令を設ける
周辺ハードウエア用の外部端子を作る
データメモリ上にマッピングする

このうち、『周辺ハードウエア制御専用の命令を設ける』のは、17Kシリーズの命令セットを決める時点で内蔵する周辺ハードウエアを決定しなければなりません（もしくは必要が予想される命令をあらかじめ用意しておく）。この方法は、必ず内蔵する周辺ハードウエアが存在すればプログラム上のステップ数を減らすことができますが、内蔵するかしないか分からない周辺ハードウエアのために命令数をむやみに増やすのは適切とはいえないと思います（特に17Kシリーズにおいては、不要な、使用頻度の低い命令は削除し（RISC指向）、CPUの負担軽減を図っているのですから）。また、これでは、新しい（プロセッサの設計時には予想されなかった）周辺ハードウエアを接続する場合に、『柔軟』に対応できないのは明白です。

『周辺ハードウエア用の外部端子を作る』のは、パッケージの大きさが大きくなってもかまわないならば、この方法でもいいでしょう。でも民生用の世界ではそんなことはまれでしょう。より小さな部品が要求され、部品が何パーセント大きくなるとコストが何パーセントかかる、という世界だそうですから。また、17Kシリーズでは、たくさんの周辺ハードウエアを内蔵することを前提とし、かつその周辺ハードウエアがそれぞれ異なる複数の品種を用意できるようになっています。そういった状況で、特定の周辺ハードウエア専用の外部端子を作ることは、なかなかできません。

そこで、『データ・メモリ上にマッピングする』方法ということのようです。データ・メモリを操作することで、周辺ハードウエアを制御します。命令のステップ数は減らすことができますし、パッケージもあまり大きくなりません。

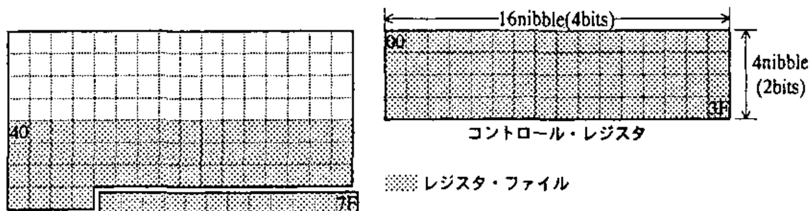
メモリ空間とI/O空間：一般的には、アドレス線の先にメモリが接続されていればメモリ空間、I/Oが接続されていればI/O空間です。

◆レジスタ・ファイルとウィンドウ・レジスタ

レジスタ・ファイル（RF）は、CPUの周辺ハードウエアの制御を行なうためのレジスタの集合です。レジスタ・ファイルを用いて制御できる周辺ハードウエアには次のようなものがあります。

A/Dコンバータ、D/Aコンバータ、シリアル・インタフェース、LCDコントローラ/ドライバ、汎用出力ポート、クロック・ジェネレータ・ポート（CGP）、PLL周波数シンセサイザ、周波数カウンタ、スタック、タイマ、割り込み、CE端子

レジスタ・ファイルは2つの部分に分けることができます。1つめは、上位64ニブルのいわゆるコントロール・レジスタと呼ばれる部分です。もう1つは、下位64ニブルのデータ・メモリと共有している部分です。



物理的には、コントロール・レジスタのアドレス空間とデータ・メモリのアドレス空間は別の位置にあり、アドレスとしては、コントロール・レジスタが00H〜3FH、データ・メモリとの共有部が40H〜7FHとなっています。

これは、レジスタ・ファイルの構成（どこに何が割り当てられているか）が、個々の品種によって違う（可能性がある）ためです。コントロール・レジスタとデータ・メモリのアドレス空間を分離することにより、品種ごとに最適なCPUの周辺ハードウエア制御用のレジスタを提供できるのです。

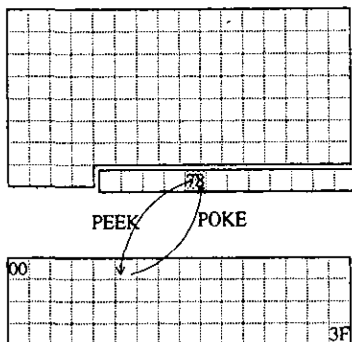
ウィンドウ・レジスタ（WR）は、データ・メモリのアドレス空間とは別の位置にあるコントロール・レジスタ（=周辺ハードウエア制御用レジスタ）とのデータのやり取りを行なうための窓です（でもウィンドウ・レジスタ自身はデータ・メモリにあるため、レジスタ・ファイルのデータ・メモリとの共有部とのデータのやり取りも当然できます）。
たとえば、レジスタ・ファイルにデータを書き込みたい場合は、ウィンドウ・レジスタにデータを置いて

```
POKE <レジスタ・ファイルのアドレス>,WR
```

とすれば、ウィンドウ・レジスタの内容がレジスタ・ファイルに書き込まれます。また、レジスタ・ファイルのデータを読み出したい場合は、

```
PEEK WR <レジスタ・ファイルのアドレス>
```

とすれば、ウィンドウ・レジスタにレジスタ・ファイルの内容が読み出されます。



ウィンドウ・レジスタはリセット信号を受けても前の値を保持しています。

しかし、このままでは、フラグ操作の際に、コントロール・レジスタのフラグとデータ・メモリのフラグの操作で別の命令を使用することになり、プログラム記述上好ましくありません。
そこで、アセンブラでは、コントロール・レジスタのアドレスを80H～0BFHと割り当てることで、プログラムの際にコントロール・レジスタとデータ・メモリのフラグの操作を同じ命令（たとえば、SETn組み込みマクロ命令）で記述することができるようになっています。ちなみにレジスタ・ファイルに割り当てられたフラグ、ポインタ、レジスタには次のようなものがあります。

ポート・グループI/Oフラグ（PbpGIO）、ポート・ビットI/Oフラグ（PbpBIO）、 割り込み要求フラグ（IRQxxx）、割り込み許可フラグ（IPxxx）、割り込み要求状態（INTxxx） スタック・ポインタ（SP）、DMA許可フラグ（DMAEN）

ポート・グループI/Oフラグ（PbpGIO）とポート・ビットI/Oフラグ（PbpBIO）は、『ポート・レジスタ』の項目ででてきます。割り込み要求フラグ（IRQxxx）と割り込み許可フラグ（IPxxx）と割り込み要求状態（INTxxx）は、『割り込み』の項目ででてきます。スタック・ポインタは、すでに『プログラム・メモリの制御』の項目ででてきました。

◆レジスタ・ファイルに割り当てられたフラグの例、DMA許可フラグ

ここではレジスタ・ファイルに割り当てられたフラグなどに関して少し説明しておきたいと思います。DMA許可フラグ（DMAEN）です。

周辺ハードウエアとのデータのやり取りの方法としてDMA（Direct Memory Access）という方法があります。これは、データ・バッファやポート・レジスタと違いデータの出入口を指定するもの、ではありません。
通常周辺ハードウエアとのデータのやり取りは、データの出入口にデータをおいて、CPUの命令を使用して、データのやり取りを行ないます。DMAでは、周辺ハードウエアからデータの要求があると、CPUの命令を中断し、指定されたプログラム・メモリ・アドレスの内容を周辺ハードウエアに転送します。

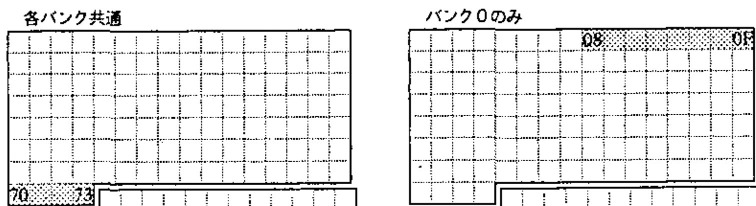
DMAは、高速に周辺ハードウエアにデータを転送する必要があるときに使用しますが、データの転送をいそがない場合には、この機能が働いては困ることもあります。このため、DMAを使用するかしないかを選択するためのフラグがあります。それが、このDMAEN（DMA ENable）です。このフラグが、セット（1）されていればDMAを使用します。このフラグがリセットされているとDMAは使用されません。

◆汎用ポート

ポートとはプログラムによってその状態を制御または読み取ることのできる外部端子の総称で、CPUと外部回路とのインタフェースのために設けられているものです。
『虫』のようなプロセッサの周りに出た『足』のような外部端子をイメージしてもらえば、理解し易いと思います。ですがワンチップ・マイコンの場合、必ずポートがプロセッサの外部に出ているとは限らない（ポートの先に内蔵の周辺ハードウエアなどが接続されている内部ポートがある）ので、この辺がワンチップ・マイコンのポートの理解の難しいところです。

> 汎用ポートがあるということは、当然専用ポートというものが存在するはずです。でも、この場合専用ポートとは周辺ハードウエアから直接プロセッサの周りに出た外部端子を指すので、何が周辺ハードウエアとして内蔵されているかによってどんな専用ポートがあるのかは違います。この辺のデータの詳細はデータ・シートをご覧ください。

17Kシリーズの汎用ポートとしては、データ・メモリ上に設けられた外部ハードウエアとのインタフェースのためのポート・レジスタの他に、外部、内部を問わず、周辺ハードウエアとのデータ転送のためにデータ・バッファという機能が用意されています。また、頻繁にアクセスする周辺ハードウエアがあれば、周辺ハードウエアをデータ・メモリの任意のアドレスにマッピングすることができます。



まず、汎用ポートをその大きさから分類すると、ニブル・ポート、バイト・ポート、ワード・ポート、ロングワード・ポートという風に分類できます。
ポート・レジスタはニブル・ポートとしてしか利用できませんが、データ・バッファはバイト・ポート、ワード・ポート、ロングワード・ポートとして使用できます。

また、その機能から分類すると、入力ポート、出力ポート、入出力ポートという風に分類されます。入出力ポートとは、ソフトウエアにより入力か出力を指定できるポートを指すのですが、その指定の方法によってさらに３種類に分類できます。ポートの大きさごとに入出力の指定ができる『グループ単位I／Oポート』と、ビット単位で入出力の指定ができる『ビット単位I／Oポート』と、入出力の操作により、入出力が切り替わる『トグルI／Oポート』です。
このうち、ビット単位で入出力の指定ができるのは、ポート・レジスタだけです。

という話は文章にしてもなかなか解らないので、表にします。表にすると次のようになります。

		ポート・レジスタ	データ・バッファ			備考
		ニブル・ポート	バイト・ポート	ワード・ポート	ロングワード・ポート	
入力ポート		品種により固定				
出力ポート						
入出力ポート	グループ単位I／Oポート					P_{bp}GIOで指定
	ビット単位I／Oポート		——	——	——	P_{bp}BIOで指定
	トグルI／Oポート	入出力の操作により入出力が切り替え				
大きさ		4nibbles × 16banks =256bits	8bits × 64address =512bits	16bits × 32address =512bits	32bits × 32address =1024bits	

（斜線）プログラムで入出力を設定

こうしてみると、一目瞭然でしょう。

◆ポート・レジスタ

ポート・レジスタは、各汎用ポートの出力データの設定や入力データの読み込みを行なうためのレジスタです。どういうことかというとポート・レジスタにデータを設定することにより、対応する各端子から信号を出力できます。また、ポート・レジスタの内容を読み出すことにより、対応する各端子に加えられている信号レベルを検出できます。

ポート・レジスタは、データ・メモリの各バンクの70H～73Hに割り当てられ、最大

4ニブル×16バンク＝256ビット

で構成されています。したがって、ポート・レジスタの内容は、データ・メモリ操作命令で操作できます。ということは当然フラグ操作組み込みマクロ命令でも操作できる、ということです。

ポート・レジスタは、ニブル・ポートとしてしか利用できません。1つのバンクに4ニブル、16ビットのポートが存在しますが、一度に処理できるのは4ビットまでです。

先ほども説明しましたが、あるポート・レジスタが、入力用、出力用、入出力用のいずれなのかは、品種によって違います。また、入出力用のポート・レジスタの場合、ビット単位、グループ単位、トグルのいずれなのかも、品種によって違います。しかし、入出力用のポートで、ビット単位かグループ単位で入出力の指定ができる場合、プログラマの責任で入出力の指定を行なわなければなりません。そのためのフラグがコントロール・レジスタにある

ポート・グループI／Oフラグ（PbpGIO）
ポート・ビットI／Oフラグ（PbpBIO）

です。それぞれのフラグが、コントロール・レジスタのどの位置にあるかは、これも品種によって違います。

◆データ・バッファと周辺ハードウエア

周辺ハードウエアを接続すれば、その分周辺ハードウエアとのデータの転送が増えます。17Kシリーズは4ビット・マイコンですから特に手だてを講じなければ1命令では4ビットづつしか転送できません。また、プログラム・メモリに大量のデータを格納するように設計されているため、プログラム・メモリからデータを読み出す必要があります。
これらの問題を解決するために設けられたのが、データ・バッファです。このデータ・バッファを使うと、1命令で周辺ハードウエアとのデータのやり取りができますし、簡単にプログラム・メモリのデータの読み出しができます。

まず、データ・バッファの1つ目の目的、周辺ハードウエアに転送するデータ、周辺ハードウエアから送られてくるデータを格納するためのバッファとしての役割の説明からです。データ・バッファで、データ転送を行なうことのできる周辺ハードウエアには、次のようなものがあります。

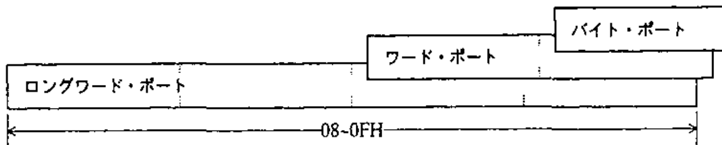
A／Dコンバータ、D／Aコンバータ、シリアル・インタフェース、LCDコントローラ／ドライバ、汎用出力ポート、クロック・ジェネレータ・ポート（CGP）、PLL周波数シンセサイザ、キー・ソース・コントローラ／デコーダ、周波数カウンタ、アドレス・レジスタ

データ・バッファはバンク0のデータ・メモリのアドレス08H〜0FHに割り当てられ、最大、

8ニブル×4ビット＝32ビット

で構成されています。したがって、データ・バッファの内容は、データ・メモリ操作命令で操作できます。

データ・バッファは、ニブル・ポートをのぞく、バイト・ポート、ワード・ポート、ロングワード・ポートのいづれの大きさでも使用ができます。しかし、データ・バッファは1つで、バイト・ポート、ワード・ポート、ロングワード・ポートの位置は重なっているので、1命令で処理できる最大ビット数は32ビットです。また、7ビットのデータを転送する場合でも、バイト・ポートを使用し、8ビットのデータが転送されます。



17Kシリーズの周辺ハードウエアには、それぞれデータ転送用のレジスタ（周辺レジスタ）があり、この周辺レジスタにはそれぞれアドレス（周辺アドレス）が割り付けられています。データ・バッファから周辺ハードウエアにデータを転送する場合は、この周辺アドレスを指定して、データの転送を行ないます。

たとえば､周辺ハードウエアにデータを書き込みたい場合は、データ・バッファにデータを置いて

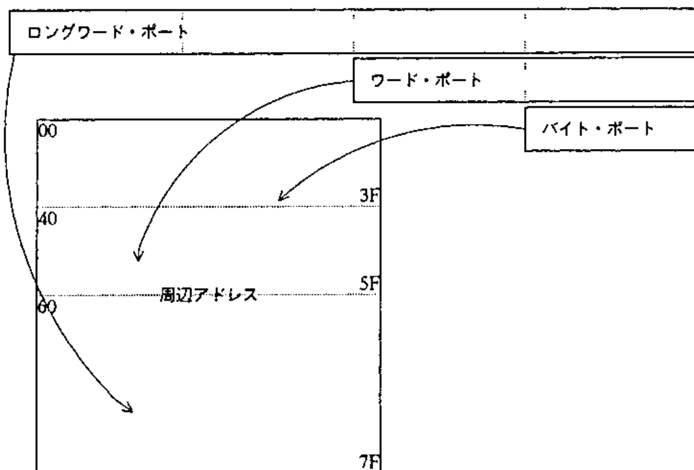
PUT <周辺アドレス>,DBF

とすれば（DBF＝データ・バッファ）、データ・バッファの内容が周辺ハードウエアに書き込まれます。また、周辺ハードウエアのデータを読み出したい場合は、

GET DBF,<周辺アドレス>

とすれば、データ・バッファに周辺ハードウエアの内容が読み出されます。

それぞれのポートに割り当てられている周辺アドレスはあらかじめ決められています。バイト・ポートは周辺アドレスの00H〜3FHの64アドレスに、ワード・ポートは40H〜5FHの32アドレスにロングワード・ポートは60H〜7FHの32アドレスに割り当てられています。



先ほども説明しましたが、あるデータ・バッファが、入力用、出力用、入出力用のいづれなのかは、品種によって違います。また、入出力用のポート・レジスタの場合、ビット単位、グループ単位、トグルのいずれなのかも、品種によって違います。

しかし、入出力用のポートで、グループ単位で入出力の指定ができる場合、プログラマの責任で入出力の指定を行なわなければなりません。そのためのフラグがコントロール・レジスタにある

ポート・グループI／Oフラグ（PbpGIO）

です。このフラグが、コントロール・レジスタのどの位置にあるかは、これも品種によって違います。

◆I／O空間の大きさ

このように、17KシリーズのI／O空間は広大な大きさとなっています。いままでに挙げたI／O空間の大きさを累計してみると、

ニブル・ポート 4ビット×16バンク＝256ビット バイト・ポート 8ビット×64アドレス＝512ビット ワード・ポート 16ビット×32アドレス＝512ビット ロングワード・ポート 32ビット×32アドレス＝1024ビット

と、最大2304ビットものI／O空間を持つことができるようになっています。

μPD1700シリーズは？：I／O空間は4ビット×16アドレス。

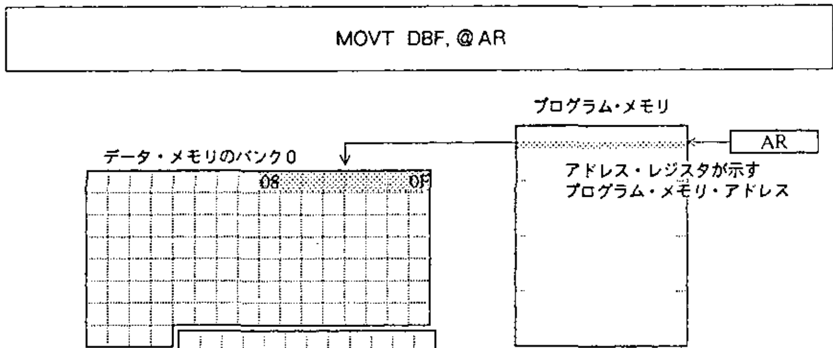
◆データ・バッファとプログラム・メモリ

では、間に余分な説明がいくつか入りましたが、データ・バッファの2つ目の目的、プログラム・メモリからの固定データの読みだしです。

たとえば、LCDに表示するデータやPLL周波数シンセサイザの分周値などの定数データをプログラム作成時に書き込んでおいても、特別なROM内データ参照プログラム、データ変換アルゴリズムなどを設計、コーディングする必要がなくなります。このように、プログラム・メモリ(ROM内)の定数データなどを読み込むことを17Kシリーズでは、テーブル参照といいます。データ・バッファを使用することにより、他のプロセッサより簡単に、テーブル参照が行なえます。

実際アセンブラ上でどのようにテーブル参照を行なうかというと、

```
MOVT DBF, @AR
```



とプログラム・メモリのアドレス・レジスタ(AR)を指定すれば、データ・バッファにアドレス・レジスタで指定されたプログラム・メモリの内容が読み出されます。

これはいってみれば、プログラム・メモリをも周辺ハードウエアの様にみたてて、データ転送(当然読み込みのみですが)を行なうようなものです。

μPD1700シリーズは?:テーブル参照は不可。

17K series architecture

17K series architecture

第5章

アドレシング

17Kシリーズのデータ・メモリのアクセスの方法には以下の4つがあります。

直接アドレシング

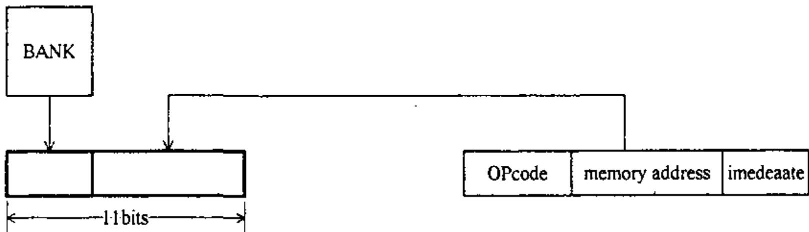
汎用レジスタ・アドレシング

インデクス・アドレシング

汎用レジスタ間接アドレシング

このうち、直接アドレシング以外の3つのアドレシングについては、このあと随時説明していきますので、ここでは直接アドレシングについて少し説明したいと思います。

直接アドレシングとは、何のことはない普通のデータ・メモリの指定の方法を指します。ただし、17Kシリーズでは、バンク・レジスタがありますので、命令のオペランドとバンク・レジスタの値を合わせた実アドレスが生成されます。



●汎用レジスタ

17Kシリーズの説明の最初からいきなり出てきていた汎用レジスタですが、いままでまったく説明もせずに使ってきました。ここまできてやっと説明できます。汎用レジスタとは、何にでも使えるレジスタです。え、説明になっていないって? たしかにそうですね。でも汎用レジスタの説明は難しいんですよ。何せ、どこにあるかも特定できない幽霊の様なレジスタですから。でも、気を取り直して説明します。

◆汎用レジスタとは

汎用レジスタとは、データ・メモリ上に配置されるレジスタです。でも汎用レジスタとして専用のハードウエアがあるわけではなく、レジスタ・ポインタ(RP)というポインタで指定されるデータ・メモリの16ニブルにそういう名前を付けているだけです。

汎用レジスタ方式を唱った他のマイコンの説明では、何にでも使えるレジスタ、という説明で納得してもらえることもあります。この汎用レジスタという言葉の裏には、今までのマイコンでは、用途が決められた専用レジスタといくつかのデータ・レジスタ（これを汎用レジスタと称することもあります）という構成が一般的だったという事実があるからです。この用途が決められたレジスタを持つプロセッサに対して、まったく用途を限定しないレジスタを複数個持つプロセッサを汎用レジスタ方式といったわけです。したがって、用途が限定されないのだから、何にでも使える、という理屈が罷り通っていたようです。

汎用マイコンの世界での話ですが、専用レジスタがあるプロセッサで有名なのが80xxxシリーズ、汎用レジスタ方式を採用したプロセッサで有名なのが68xxxシリーズですが、68xxxの汎用レジスタと17Kシリーズの汎用レジスタではだいぶイメージが違います。別物と思った方が理解し易いでしょう。

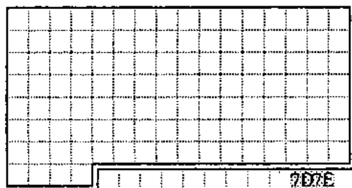
基本的に、17Kシリーズのデータの演算、転送はデータ・メモリと即値との間で行なわれます。でも、これだけでは不便ですので、データ・メモリとレジスタとの間で演算、転送ができるようになっています。このレジスタが汎用レジスタで、汎用レジスタとデータ・メモリの間で演算、転送ができる方式を汎用レジスタ方式と呼んでいます。
でも、先ほど汎用レジスタはデータ・メモリにある、といったばかりです。ということはデータ・メモリ同士で演算ができることになります。
つまりデータ・メモリAとデータ・メモリBで演算を行ないデータ・メモリAに結果を格納する、ということが1命令でできるのです。

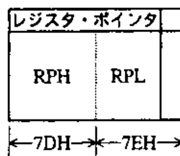
このことは、アキュムレータ方式のマイコンでプログラミングをしてきた方には新鮮に感じられるかもしれません。アキュムレータ方式では、データ・メモリAの値をアキュムレータに格納し、アキュムレータとデータ・メモリBの間で演算し、演算の結果が格納されているアキュムレータの内容をデータ・メモリに保存する、といった動作が必要だからです。

たびたび汎用の話で恐縮ですが、汎用マイコンでレジスタの大きなプロセッサが好まれるのは、通常のメモリとレジスタでは、データのアクセスに要する時間が5倍から10倍くらい違ってくることに原因があるようです。でも17Kシリーズなどの専用マイコンでは、レジスタもメモリも一緒のケースが多く、アクセスに要する時間はほとんど問題になりません。

◆レジスタ・ポインタ（RP）

さて、問題はこの汎用レジスタがどこにあるかですが、レジスタ・ポインタ（RP）の内容により違います。では、このレジスタ・ポインタが、どうなっているかというと、レジスタ・ポインタはデータ・メモリのシステム・レジスタの7DHと7EH（ビット3-1）にあります。

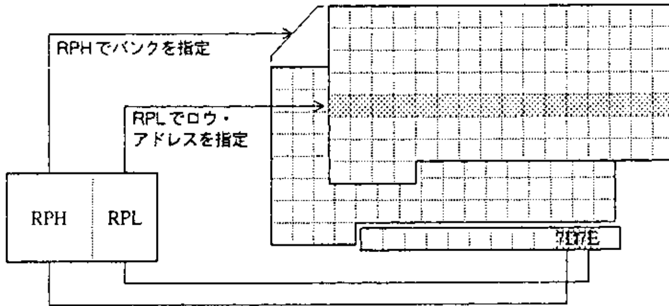




これがどういうことを意味するかというと、汎用レジスタの位置はレジスタ・ポインタを操作することにより自由に変更できるということです。このことから、データ・メモリ同士で演算ができるということがいえるわけです。たとえ、汎用レジスタがデータ・メモリにあろうとも品種によって固定されていたら、さすがにデータ・メモリ同士で・・・、という（拡大解釈に近い）ことはいえません。

μPD1700シリーズは？：汎用レジスタのロウ・アドレスは0Hに固定。

レジスタ・ポインタは、最大7ビットあります。下位4ビット（7DH）で汎用レジスタとして使用するバンクを指定し、上位3ビット（7EH）でデータ・メモリのロウ・アドレスを指定します。バンクとロウ・アドレスを別々に指定する場合に、バンクを指定する下位4ビット（7DH）をRPH、ロウ・アドレスを指定する上位3ビット（7EH）をRPLという場合があります。

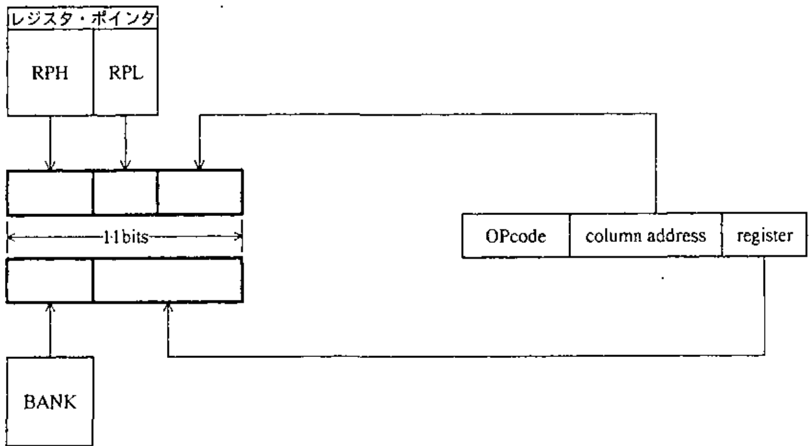


ここで重要なのは汎用レジスタを指定するレジスタ・ポインタの情報としてバンクの情報も含まれている、ということです。つまり汎用レジスタとして指定できるデータ・メモリは、どのバンクにあってもよく、現在使用中のバンクに影響を受けないということです。
これは、バンクAのデータ・メモリBが汎用レジスタに指定されていれば、バンクCのデータ・メモリDとの間で演算、転送ができる、ということで、バンクを超越した演算、転送が可能、ということです。

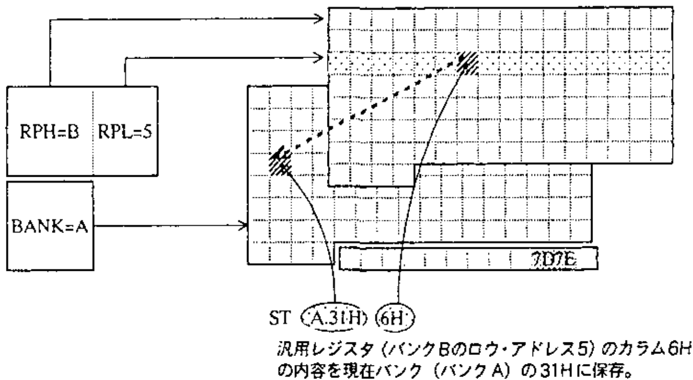
◆汎用レジスタを用いたアドレシング

汎用レジスタがデータ・メモリのどこにあってもよく、現在使用中のバンクに影響を受けないならば、当然汎用レジスタをオペランドに指定することのできる命令では、17Kシリーズのすべてのデータ・メモリ・アドレスを生成しなければなりません。

汎用レジスタをオペランドに指定できる命令には、ADD／ADDC／SUB／SUBC／AND／OR／XOR／LD／ST／RORCなどがあります。これらの命令を使用すると、アドレスとして、次のような11ビットのアドレスが生成されます。



このようにして生成されたアドレスの間で演算、転送を行ない、結果を汎用レジスタで指定されたアドレスに格納します。当然、BANKと、RPHの値が違っていようと同じであろうと、正しく処理が行なわれます。



汎用レジスタ（バンクBのロウ・アドレス5）のカラム6H
の内容を現在バンク（バンクA）の31Hに保存。

ここで、唯一問題になりそうなことは、異なるバンクにある汎用レジスタと即値との演算の命令がないことです。もし現在使用中のバンクに、汎用レジスタがあれば、汎用レジスタであるかないかに関わらず、データ・メモリとして扱えばいいのですから。

というところで、いきなりこの項目は終了です。汎用レジスタを用いたアドレシングで特徴的なことは、①データ・メモリ同士で演算できる。②バンクを超えて演算できる、ということでした。異なるバンクにある汎用レジスタと即値の演算についての解決は次週まで・・・、なんてことはありません。次の項目でちゃんと説明します。

◆汎用レジスタと即値の演算

この問題で一番単純な解決方法は、バンクを切り換え、データ・メモリと即値の間で演算を行ない、再びバンクを切り換えることです。しかしこれでは、プログラム・ステップの増加を招く原因となり得ます（一度や二度なら構わないでしょうが）。

では、2つ目の解決方法として、演算を行ないたい即値を一度現在使用中のバンクのデータ・メモリに格納し、格納したデータ・メモリと汎用レジスタの間で演算を行なうことです。これにも実は問題があり、即値を格納するためのデータ・メモリと格納のためのステップが、余分にかかることになります。

ここで、ちょっと発想を転換して、汎用レジスタと即値を演算しなければならないケースはどういう場合でしょうか？

実をいうと汎用レジスタと即値を演算しなければならない必然性というのは少ないと思います（多かったら命令として存在しているはずです）。汎用レジスタを使うということは、これからデータ・メモリとの間で何らかの演算の行なうことを意味しているのですから、わざわざ汎用レジスタと即値を演算してからさらにデータ・メモリとの間で演算する必要はあまりなく、即値が必要なデータ・メモリと直接演算すればいいはずです。

また、データ・メモリを一時的なデータの保存場所と演算用の作業場所と分けて考えた場合、演算用の作業場所がそれほど大量に（バンクをまたがって）必要になるとは、あまり考えられず、大部分のデータ・メモリにおいては演算が行なわれる可能性が低いということもできます。

という同情的見解はこの際ほっておいて、汎用レジスタと即値の演算の可能性がないわけではありません。一番可能性があるのは、汎用レジスタを使用した間接転送の時で、この場合、汎用レジスタの内の1ニブルを即値を用いて増減することが考えられます。

そこで、汎用レジスタと即値の間の演算の3つ目の解決方法として、被演算データが1ニブルの場合の最も効率的な演算方法です。
17Kシリーズのデータ・メモリ上にはシステム・レジスタがあり、その中にはコントロール・レジスタをアクセスするためのウィンドウ・レジスタが存在します。汎用レジスタがデータ・メモリのどこを指しても良いのならば、ウィンドウ・レジスタが存在するロウ・アドレス7Hのここを指してもいいはずです。システム・レジスタは各バンク共通ですので、どのバンクからも同じ様に見えるはずです。そこで、汎用レジスタをシステム・レジスタのある位置に移動し（RPL＝7)、ウィンドウ・レジスタと即値を演算すれば、現在どのバンクにいようとも、レジスタ・ポインタがどのバンクを指していようとも汎用レジスタと即値の間で演算ができたことになります。

●インデクス・アドレシングとインデクス・レジスタ

インデクス・アドレシングの説明をする前に、PSWのIXEの説明をしなければなりません。PSWは先ほど出てきましたがIXEの説明はインデクス修飾をするかしないかを指定するためのフラグ、といったおざなりに説明で逃げていました。ここまで来たらもう逃げているわけにはいきません。でもその前にインデクス修飾とは何か？ ということを説明しなければなりません。

◆インデクス修飾

インデクス修飾とは、バンク及び命令のオペランドで直接指定されたデータ・メモリ・アドレスを『修飾』する機能で、この様なデータ・メモリのアドレシングをインデクス・アドレシングといいます。

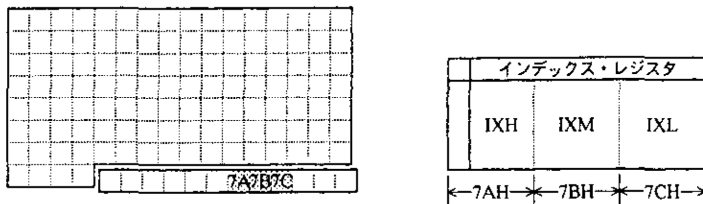
インデクス修飾は、一種のデータ・メモリへの間接アドレシングです。前の項目で説明した汎用レジスタを用いたアドレシングでは、レジスタ・ポインタと命令のオペランドで指定した値を組み合わせることで、特定のデータ・メモリ・アドレスを指定しました。インデクス・アドレシングでは、インデクス・レジスタと命令のオペランドで指定した値をOR演算することで、特定のデータ・メモリ・アドレスを指定します。

> 他のマイコンでインデクス修飾といった場合、オペランドのアドレスと、レジスタのアドレスの加算を意味します。しかし17Kシリーズにおいては、オペランドのアドレスとレジスタのアドレスのOR演算を行ないます。したがって、17Kシリーズのインデクス修飾を特にORインデクス修飾と呼んで区別する場合があります。なぜ、加算でなくOR演算なのか･･･、という話はできませんが、ヒントは『コスト』です。

この様なデータ・メモリに対する修飾を行なうか行なわないかを指定するためのフラグがIXEです。IXEがセット（1）されているときに、データ・メモリをアクセスすると、命令のオペランドで直接指定したデータ・メモリ・アドレスとインデクス・レジスタの値がOR演算され、演算の結果として生成されたデータ・メモリ・アドレス（これを実アドレスというそうです）に対して命令が実行されます。

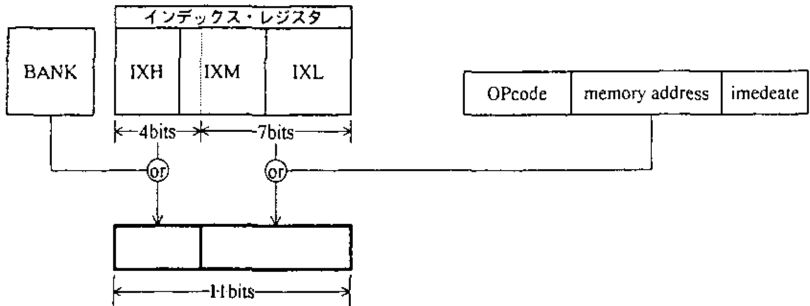
◆インデクス・レジスタ（IX）

まずは、インデクス・レジスタ（IX）のある場所ですが、当然データ・メモリのシステム・レジスタの、7AH～7CHに割り当てられ、最大11ビットです。



このインデクス・レジスタを使った実アドレスの生成についてもう少し細かく説明すると、命令のオペランドで指定された7ビットと、インデクス・レジスタの下位7ビットがOR演算され、実アドレスの下位7ビットとなります。そしてバンク・レジスタの4ビットとインデクス・レジスタの上位4ビットがOR演算され、実アドレスの上位4ビットとなります。

という風に文章で書くとさもむずかしそうに聴こえるのですが、図にすると、



という風になり、要するに今までのデータ・メモリのアドレシング（バンクと命令のオペランドをあわせる）にインデクス・レジスタを足した（正確にはOR演算）しただけです。こんなに簡単なことなのです、インデクス・アドレシングの『原理』は。

ちなみに、このインデクス・アドレシングの適用対象は、『IXE = 1の場合に、命令のオペランドで直接指定され、データ・メモリとしてアクセスされるデータ・メモリ』です。つまり何がいいたいかというと、汎用レジスタを介してアクセスされるデータ・メモリや、汎用レジスタとしてアクセスされるデータ・メモリは含まれないということです。インデクス修飾が行なわれるのは、ADD／ADDC／SUB／SUBC／AND／OR／XOR／LD／ST／MOV／SKE／SKGE／SKLT／SKNE／SKT／SKFなどの命令の場合のみです。これらの命令で指定されるデータ・メモリ・アドレスは、IXEの値によって違ってきます。これ以外は、IXEの値には影響されません。
と、『原理』はさほど難しくないインデクス・アドレシングですが、『使い方』には多少のテクニックがいります。ここでは少し例も交えて説明しましょう。

例1 : BANK = 0、IXH = 1、IXM = 0、IXL = 0、IXE = 1で

ADD 12H, # 3

という命令を実行した場合

[BANK = 0] OR [IX (10-7) = 0010B] = 0010B = 2

[1stOP = 0010010B] OR [IX (6-0) = 0000000B] = 0010010 = 12H

となり、バンク2の12Hの値に3が加算されます。この様に異なるバンクにあるデータ・メモリと即値の間で演算ができます。

例2 : BANK = 0、IXH = 0、IXM = 1、IXL = 0、IXE = 1で

ADD 12H, # 3

という命令を実行した場合

[BANK = 0] OR [IX (10-7) = 0000B] = 0000B = 0

[1stOP = 0010010B] OR [IX (6-0) = 0010000B] = 0010010B = 22H

となり、バンク0の22Hの値に3が加算されます。この様に同じバンクの違うロウ・アドレスとの演算ができます。

例3 :BANK = 0、IXH = 0、IXM = 0、IXL = 0、IXE = 1ですと、どのような命令を記述しても、IXE = 0の場合と同じ結果が得られます。ここで、

```
MOV 0H, #3  ;①
INC IX
MOV 0H, #5  ;②
```

という命令を実行した場合①では、

[BANK = 0] OR [IX (10-7) = 0000B] = 0

[1stOP = 0000000B] OR [IX (6-0) = 0000000B] = 0H

というアドレスが生成され、バンク0の12Hには3が格納されますが、②では

[BANK = 0] OR [IX (10-7) = 0000B] = 0

[1stOP = 0000000B] OR [IX (6-0) = 0000001B] = 1H

というアドレスが生成され、バンク0の13Hには5が格納されます。この様に連続するアドレスに次々にデータを格納することができます。ここでINC IXは、IXを1つ増加するための命令です。格納開始番地として、IXと命令のオペランドの両方を0にしていますが、どちらも任意の番地を指定することができます。

この例では、説明のため、同じ様な命令を①と②に記述しましたが、アセンブラでは、マクロを使用すればもっと簡単に美しく記述できるはずです。

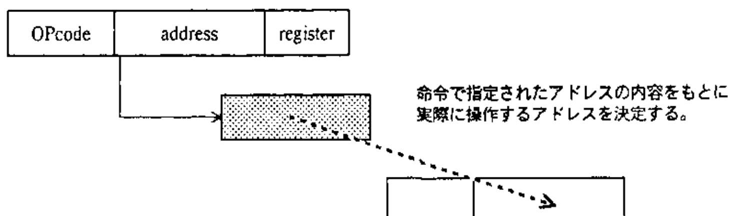
●間接アドレシングとデータ・メモリ・ロウ・アドレス・ポインタ

データ・メモリ・ロウ・アドレス・ポインタ（MP）を利用した汎用レジスタ間接アドレシングも、前の項目の汎用レジスタを用いたアドレシング、インデクス・アドレシングと同様に特定のデータ・メモリ・アドレスを指定するための1つの方法です。

◆間接アドレシング

まずは、間接アドレシングの説明です。『間接』というキーワードは、プログラム・メモリの項目でも出てきました。『間接分岐』という言葉でした。間接分岐は、分岐する時に命令のオペランドを使って分岐するのではなく、アドレス・レジスタに保存された値を使って分岐していました。間接アドレシングも、イメージとしては同様で、データ・メモリの特定のアドレスを指定するのに、レジスタの値を使ったり、演算したりするのではなく、あるアドレスに保存された値を使って、特定のアドレスを指定する方法です。

間接アドレッシングの基本



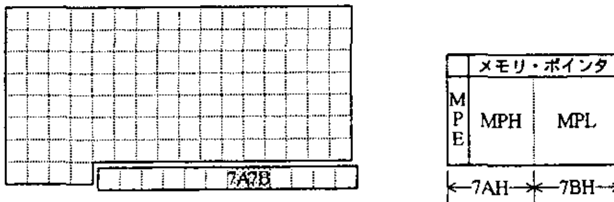
17Kシリーズで間接アドレシングを行なうためにデータ・メモリ・アドレスの内容を保存する場所は、汎用レジスタだけです。したがって17Kシリーズで、間接アドレシングといえば、汎用レジスタ間接アドレシングだけです。

で、汎用レジスタ間接アドレシングに関係するフラグが1つあります。それがMPE（Memory Pointer Enable）フラグです。インデクス・アドレシングに関係するIXEフラグが、インデクス修飾をするかしないかを指定するためのフラグだったのに対し、MPEは多少ニュアンスが違います。汎用レジスタ間接アドレシングをするかしないかを指定するのではなく、汎用レジスタ間接アドレシングの際にデータ・メモリ・ロウ・アドレス・ポインタを使用するかしないかを指定するためのフラグがMPEです。

◆データ・メモリ・ロウ・アドレス・ポインタ（MP）

データ・メモリ・ロウ・アドレス・ポインタ（以下MP）は、レジスタ・ポインタと同じ様な感じでバンクとロウ・アドレスの情報を持っています。レジスタ・ポインタが汎用レジスタのあるバンクとロウ・アドレスの情報を持っているのに対して、MPは間接アドレシングの参照先のアドレスがあるバンクとロウ・アドレスの情報を持っています。

では、MPのある場所ですが、これもデータ・メモリのシステム・レジスタの、7AHと7BHに割り当てられ、最大8ビットです。そして、システム・レジスタの7AH（ビット3）には先ほどのMPEがあります。



ということはインデクス・レジスタとMPは同じ1つのデータ・メモリを共有していることになります。これは決して（少なくともこの部分は）誤植や書き間違いでは、ありません。ましてや、システム・レジスタが2枚あるわけでもないのです。というわけで、インデクス・レジスタとMPで別々の値を保持することはできません。

汎用レジスタ間接アドレシングができる命令は2つだけです。

```
MOV @r, m
MOV m, @r
```

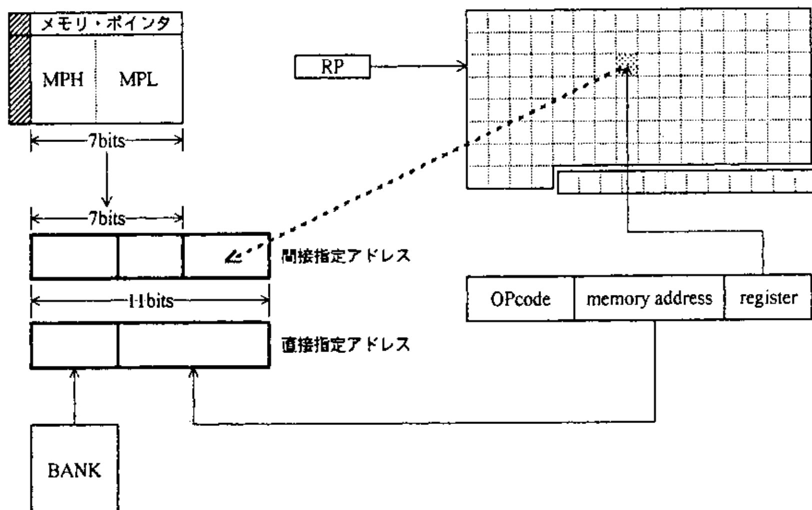
この2つの命令を総称して汎用レジスタ間接転送命令というそうです。ということは、汎用レジスタ間接アドレシングを利用してできることはデータの転送だけであるということです。ちなみに、この2つの命令は転送の方向が逆になっているだけで、動作としてはほとんど同じです。ここでは、@rで指定される方を間接側、mで指定される方を直接側と名付け、説明を行ないます。

17Kシリーズの汎用レジスタ間接転送で、まず始めに決めなければならないことは、どこからどこに転送するか、ということです。MPEの内容により間接側のバンクとロウ・アドレスが違います。

MPE＝1の場合

MPE＝1の場合、間接側のバンクとロウ・アドレスを生成するのに、MPの値を使用します。これにより、間接側と直接側のバンクとロウ・アドレスが違うという可能性が発生します。この汎用レジスタ間接転送を『ななめ転送』という場合があります。

命令のオペランドで指定されたカラム・アドレスは、レジスタ・ポインタで指定されたバンクとロウ・アドレスと一緒になって汎用レジスタ上のアドレスを指します。この汎用レジスタ上のアドレスの内容を元に、間接側のカラム・アドレスを生成します。他のアドレスの『内容』からアドレスを生成するから『間接』アドレシングなわけです。このようにして生成されたカラム・アドレスは、MPで指定されたバンクとロウ・アドレスと一緒になって間接側となる11ビットのアドレスを生成します。



一方直接側のアドレスの生成は簡単で、命令のオペランドで指定されたデータ・メモリ・アドレスは、バンク・レジスタで指定されるバンクと一緒になって直接側となる11ビットのアドレスを生成します。

図では、汎用レジスタが描かれていますが、重要なのは、汎用レジスタがどこにあるかではなく、汎用レジスタ中のアドレスに間接側のカラム・アドレスを決定するデータが格納されていることです。

例1：BANK＝0、MPH＝0、MPL＝0DH、MPE＝1、RPH＝1、PRL＝2

バンク1の23Hには6が格納、で

MOV　12H, @03H

という命令を実行した場合

[直接側]　　バンク0の12H

[間接側]　　カラム・アドレスは汎用レジスタと第2オペランドにより指定される

汎用レジスタはBANK＝1、ロウ・アドレス＝2、カラム・アドレス＝3

参照されるアドレスは、バンク1の23H

したがって、間接側のカラム・アドレス＝6

バンクとロウ・アドレスはMPにより指定される。

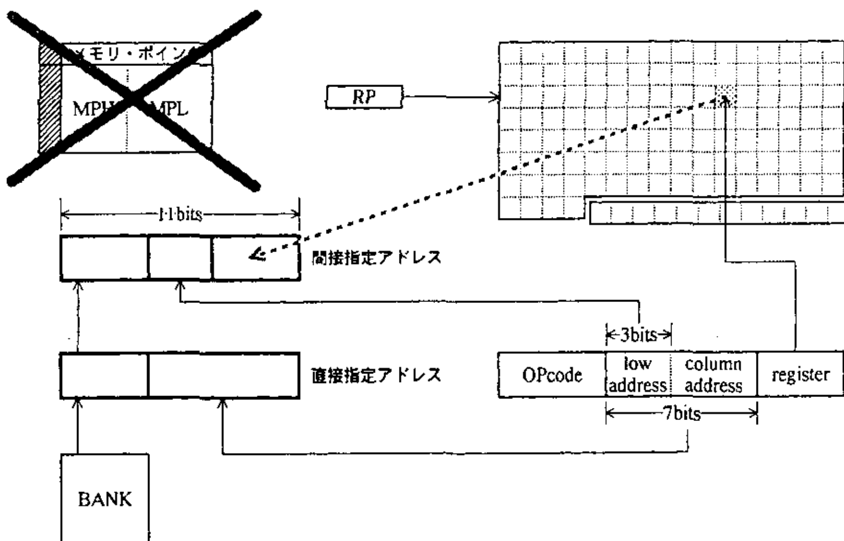
MP＝0000 1101

したがって、間接側のBANK＝1、ロウ・アドレス＝5

間接側のアドレスはバンク1の56H

MPE＝0の場合

MPE＝0の場合、間接側のバンクとロウ・アドレスを生成するのに、命令のオペランドを使用します。これにより、間接側と直接側のバンクとロウ・アドレスは常に一致します。この汎用レジスタ間接転送を『同一ロウ転送』という場合があります。



命令のオペランドで指定されたカラム・アドレスから間接側のカラム・アドレスを間接アドレシングで、生成するところまでは同じですが、そこから先が多少違います。

生成されたカラム・アドレスは、バンク・レジスタで指定されるバンクと命令のオペランドで指定されたロウ・アドレスと一緒になって間接側となる11ビットのアドレスを生成します。

直接側のアドレス生成はMPE＝1の場合と同じで、命令のオペランドで指定されたデータ・メモリ・アドレスとバンク・レジスタで指定されるバンクから直接側となる11ビットのアドレスを生成します。

要するにMPE＝0の場合の直接側と間接側のバンクとロウ・アドレスの情報源は一緒です。命令のオペランドで指定されたカラムと、間接アドレシングで生成されたカラムの間で転送が行なわれるだけです。

例2：BANK＝0、MPH＝0、MPL＝0DH、MPE＝0、RPH＝1、PRL＝2
　　バンク1の23Hには6が格納、で
　　　　MOV　12H, @03H
　　という命令を実行した場合
[直接側]　　バンク0の12H
[間接側]　　カラム・アドレスは汎用レジスタと第2オペランドにより指定される
　　　　　　　汎用レジスタは
　　　　　　　BANK＝1、ロウ・アドレス＝2、カラム・アドレス＝3
　　　　　　参照されるアドレスは、バンク1の23H
　　　　　　したがって、間接側のカラム・アドレス＝6
　　　　　　バンクとロウ・アドレスは第1オペランドにより指定される。
　　　　　　したがって、間接側のアドレスはバンク0の16H

●インデクス修飾と間接アドレシング

インデクス・レジスタとMPで別々の値を保持することはできません。でもインデクス修飾と間接アドレシングは同時に使用できます。インデクス修飾と間接アドレシングを同時に使用すると、便利なことがある反面、アドレスの生成やメモリの共有の点で注意を要する場合があります。

インデクス修飾をするかしないかを指定するフラグIXEが0の場合、つまりインデクス修飾をしない場合、間接アドレシングは、前の項目で説明したとおりの動作をします。この場合は、インデクス・レジスタとMPがデータ・メモリを共有していることを気にする必要はありません。データ・メモリの7AH、7BHを使用するのは、MPだけということになりますから。

IXEフラグが1の場合、つまりインデクス修飾をする場合に間接アドレシングをすると、いささかアドレスの生成が複雑になりますので、ここで簡単に説明して見たいと思います。アドレスの生成が少々複雑になるにも関わらず、このようなことができるのは、それなりの理由があるからです。以下はIXEフラグが1の場合の話です。

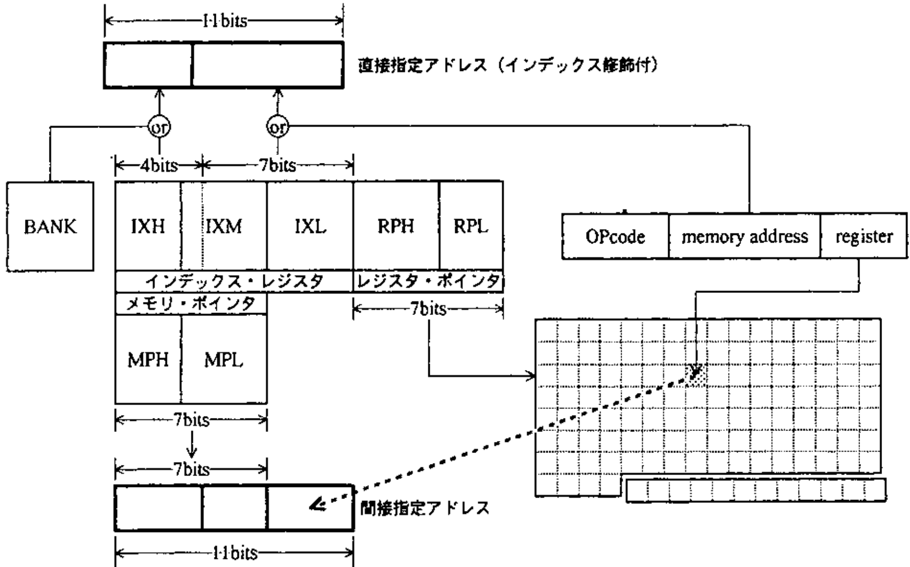
MPE＝0の場合

この場合、インデクス・レジスタとMPがデータ・メモリを共有していることを気にする必要はありません・データ・メモリの7AH、7BH、7CHを使用するのは、インデクス・レジスタだけということになりますから。
アドレス生成においては、（間接アドレシングで生成される）間接側のカラム・アドレスをのぞいてすべてのアドレスがインデクス修飾されます。

MPE＝1の場合

アドレス生成において、間接側のアドレスは、MPと汎用レジスタの内容により生成されますので、インデクス修飾は行なわれません。直接側のアドレスは、オペランドにより指定されますので、インデクス修飾が行なわれます。

間接側のアドレスがMPにより生成され、直接側のアドレスがインデクス修飾される（＝インデクス・レジスタの値が使用される）ということは、間接側と直接側で同じ値（MPとインデクス・レジスタ）を使用し、別々のアドレス生成処理（バンクとロウ・アドレスの指定と、OR演算）が行なわれるということです。これは、一見アドレス生成処理が複雑になるだけのように思えます。



なぜ、このような状態が存在するかというと、先ほどもいったように便利なことがあるからです。通常のインデクス・アドレシング、間接アドレシングでは、データ・メモリ間の転送の際にどちらか一方はオペランドでデータ・メモリ・アドレスを直接指定する必要がありました。これをIXE＝1、MPE＝1とすることで、転送元（ソース）と転送先（ディスティネーション）を両方とも間接指定をすることができるようになります。

例：　BANK＝1、IX＝0、RPL＝7、IXE＝1、MPE＝1で

```
LOOP:   OR    PSW, #0001B  ; IXE←1
        MOV   40H, @0CH
        AND   PSW, #1110B  ; IXE←0
        INC IX             ; IX←IX+1
        SKGE IXM,2         ; IXMが2未満なら
        BR    LOOP
```

配列とは、同じ大きさの変数の集まりをいいます。高級言語ではあらかじめ配列の機能が用意されていることがほとんどですが、アセンブラでは配列の機能が用意されていることはまずありません。コンピュータが持っている自然なサイズの要素を持つ配列を作るのはアセンブラでも簡単です。1つのアドレスに1つのデータを格納していけばいいのですから。難しいのは自然なサイズ以外の要素を持つ配列を作ることです。一部の、古い高級言語では自然なサイズの大きさ以外の配列を作ろうとすると、アセンブラなみのテクニックを要するものもありました。

コンピュータが持っている自然なサイズとは、4ビットのコンピュータならば4ビットの大きさ、8ビットのコンピュータならば8ビットの大きさのことです。また違う大きさの変数の集まりは、C言語では構造体、Pascalではレコードです。

◆4ビット以上のデータ

配列の本題に入る前に、すごく基本的なことを抑えておきましょう。17Kは4ビットのマイコンでした。4ビットでは、16種類のデータを表現することができます。17Kシリーズの主目的は制御であり多桁の演算ではありませんが、2桁の演算くらいは、制御の世界でもよく行ないます。

一般に一度に演算できる範囲以上の値（4ビットの場合ならば16以上）の演算を行なうには、1つの数値をいくつかのアドレスに分けて格納し、1つの演算を行なうにも、複数のアドレスを操作して演算を行ないます。

4ビット以上の値の演算、たとえば17Hと29Hの足し算を行なう場合には、次の様な操作が必要になります。

```
M1      MEM     0.01H           ; 8ビットの値の演算結果　下位桁
M2      MEM     0.02H           ; 8ビットの値の演算結果　上位桁

        MOV     M1, #07H        ; 17Hをメモリに記憶
        MOV     M2, #01H
        ADD     M1, #09H        ; 29Hを加算、結果をM1、M2へ
        ADDC    M2, #02H
```

一方、17Kのアセンブラでは、NIBBLEn疑似命令（n＝1～8）を使用することで、複数ニブルの大きさのメモリを定義することができますが、

```
N1      NIBBLE2         0.01H

        ADD     N1, #21H
```

という演算を行なうと

```
T.ASM(5) : V ERROR 015 : Illegal second operand value
```

というエラーが発生します。これは『2番目のオペランドの大きさが不適切である』というメッセージで、16以上の値を指定すると発生します。したがって、次のような演算を行ないます。

```
N1      NIBBLE2         0.01H

        ADD     N1, #1H
        ADDC    N1+1, #2H
```

N1と1を加算し、N1のとなりのアドレスに2と前の命令でキャリーが出たらキャリーを加算しなさい、というプログラムです。

一般に、任意ニブルの大きさの加算のためのマクロは次のように定義できます。

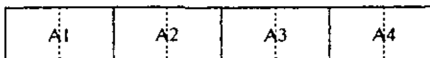
```
ADDX    MACRO   M, I                ; 任意ニブルの加算マクロ
                                    ; M = MEM型
        ADD     M, I AND 0FH

CNT     SET     1                   ; MEM型シンボルの評価値の上位4ビットには
        REPT    .DM.M SHR 12        ; [実際のニブル数-1] が格納されている
                ADDC    M+CNT, I SHR (4 * CNT) AND 0FH
          CNT   SET CNT + 1
        ENDR
        ;
        ENDM
```

REPT－ENDRの回数は、MEM型シンボルの評価値の上位4ビットに、[実際のニブル数－1] が格納されていることを利用して決定しています。これにより、2ニブルなら1回、5ニブルならば4回の加算が行なわれます。

◆配列

さて、問題の配列ですが、ここでは仮に8ビットの大きさを持つ変数が4個集まった配列を考えるとします。



そして、N番目の要素にMを加算する場合には、前述の任意ニブルの加算のマクロを使用し（別にマクロを使用しなくてもいいのですが、マクロを使えばより汎用的なルーチンになります）、

```
N2      NIBBLE2         0.00H

        OR      PSW, #0001B     ; IXE is enable

        MOV     IXH, #0         ; (IX)=N
        MOV     IXM, #0         ; 同一バンク、同一ロウの
        MOV     IXL, #N         ; N番目の要素にアクセスする

        ADD     IXL, IXL        ; (IX) = (IX) * 2
        ADDC    IXM, IXM        ; 要素の大きさの分 (8ビット=2ニブル) だけ
        ADDC    IXH, IXH        ; IXが指し示す位置を変更

        ADDX    M2, M           ; 任意ニブルの加算のマクロを呼び出し
```

というプログラムにします。ここでは、インデックス・レジスタを配列の要素を指定するために使っています（もともとはそのためのインデックス・レジスタですから）。

17K series architecture

第6章

割り込み

●割り込みとは

割り込みとは、ある処理の流れを中断させ、別の処理を実行させる機能です。割り込みの特徴は、何といっても『いつ発生するか分からない』という点にあります。割り込みにより実行される別の処理を割り込みルーチンといいますが、割り込みがいつ発生するか分からないのですから、割り込みルーチンもいつ実行されるか分かりません。いつ実行されるか分からないということは、どんな状況で実行されるかも分からないということです（本当はある程度想像は付くのですが、あくまで想像です）。

さらに、ある処理を中断して、割り込みルーチンを実行させたのですから、割り込み処理ルーチンが終ったらすべての情報を元に戻しておかないと中断された処理は怒ります。怒るだけならまだしも、割り込み処理ルーチンに変更された情報を元にプログラムを実行しますので、いきなり暴走を始める可能性さえあります。

というわけで、割り込みルーチンの最初では、割り込みの前で使用していた情報を別の場所に格納しておき、割り込みルーチンの最後では格納した情報を元に戻す、という作業を行ないます。ちなみにこのときの格納と元に戻す作業を『退避』と『復元』という場合があります。

こうすることにより、どんな状況でも実行でき、割り込みが終了したらすべての情報が元に戻っている、という割り込みルーチンを記述することができるのです。

◆ハードウエア割り込みとソフトウエア割り込み

汎用マイコンでは、ソフトウエア割り込みとハードウエア割り込みという分類の方法があります。ハードウエアの信号による割り込みをハードウエア割り込み、ソフトウエアからの要求（ということは、プログラムに記述され、意図的に割り込みルーチンを使用）による割り込みをソフトウエア割り込みといいます。

が、今までの専用マイコンは、ハードウエア割り込みしかありません（もしくは圧倒的にハードウエア割り込みが多い）でしたので、通常は割り込みといえば、ハードウエア割り込みを指します。この場合のハードウエアとは、周辺ハードウエアを指し、周辺ハードウエアからの割り込み信号を割り込み要求といいます。

しかし17Kシリーズでは、システムで用意されているルーチンを実行する機能（SYSCAL命令）があります。この場合は、普通にプログラムに命令として記述され、意図的に割り込みルーチンを使用するのですから、ソフトウエア割り込みとなります。

このあらかじめ用意されたルーチンを実行するためのソフトウエア割り込みを17Kシリーズでは『システム割り込み』と呼んでいます（この他にシステム・セグメントに割り当てられたハードウエア割り込みもシステム割り込みというそうです）。

当然、普通のプログラムと一緒に記述した普通の割り込みルーチンを実行させる普通のソフトウエア割り込みも存在します。

◆割り込みの優先度と多重割り込み

割り込みが起った原因を割り込み要因といいます。一般に、複数の周辺ハードウエアを持つプロセッサは、複数の割り込み要因を持つことがあります。周辺ハードウエアを持たないプロセッサでも、割り込み用の端子を複数持ち、この端子の数だけ割り込み要因を持つことが考えられます。割り込み要因が一つだけならば構わないのですが、複数の割り込み要因持つものでは何らかの形で、同時に割り込みが起った場合や、割り込み中に割り込みが起った場合のことを考慮しなければなりません。

µPD1700シリーズでは？：割り込み要因数は1つだけ。
同時に割り込みが起ることや割り込み中の割り込みなどは、考慮しない、というのも一つの手段ではありますが、その様なコンセプトは低機能な割り込み機能しかサポートされてない、と判断してもいいと思います（コスト上の問題からこのような選択がされることもありますが…）。

複数の割り込み要因をサポートし、かつ同時に割り込みが起ることや割り込み中の割り込みを考慮しているプロセッサでは、この問題を解決するために、割り込み要求があってもすぐに割り込みが実行されないようになっていることがほとんどです。割り込み要求を発行しても、条件が整っていなければ割り込みが受け付けられないこともありますし、割り込みの要求が無視されることもあります。当然、割り込みが受け付けられなければその割り込みは待たされる（保留）ことになりますし、割り込み要求が無視されれば、割り込みルーチンは実行されません。要求を出し、その要求が受け付けられ（許可され）て、始めて割り込みルーチンが実行されるのです。

複数の割り込み要求が同時に発行される場合を考えると、次のようなケースが考えられます。

各割り込み要因が同時に割り込み要求を発行（IRQxxx = 1） 割り込み要求の発行に時間差はあるが、 割り込みが可能になったのが同時（INTE = 1 or IPxxx = 1）

このようなときのために17Kシリーズでは、割り込み要因ごとにそれぞれ優先度が決っています。この割り込みが同時に発生した場合の優先順位をハードウエアで決定するための回路が『優先順位決定回路』です。これは割り込みが同時発生した場合に、優先度の高い方から受け付けるためです。このとき、割り込み要求が発行された割り込み要因は、これ以上割り込みができないようになります。(IRQxxx)

でも、これだけでは不便な場合があります。同時に発生した割り込みの優先順位をハードウエアで決定されると、あとから変更することができません。あとから変更することができないということは、プログラム中で割り込みの優先順位を変更できないということです。

17Kシリーズでは、このような時のために、割り込みを禁止することができます。

ハードウエアで決定された優先順位が高い割り込みでも、割り込み要求があっても、割り込みを受け付けなければ、ハードウエアで決定された優先順位の低い割り込みでも先に受け付けることができます。このように禁止できる割り込みをマスカブル割り込み、逆に禁止できない割り込みをノンマスカブル割り込みといいます。言葉を変えれば、ノンマスカブル割り込みは何時でも割り込める割り込みで、マスカブル割り込みは場合によっては割り込めない割り込みです。ある割り込みが禁止できるかできないか（マスカブルかノンマスカブルか）は、製品によって異なっています。

ハードウエアで決定された優先順位を変えるための機構ですから、プログラム中で変更ができるようになっているはずです。(IPxxx)

では、めでたく割り込みが受け付けられたと仮定して、割り込みルーチン中で、さらに別の割り込みを受け付ける場合の話です。まず、このようなことができる方式を多重割り込み、といいます。ただし、割り込みルーチン中で、別の割り込みを受け付けられるのは、実行中の割り込みルーチンの優先度より高い優先度の割り込みが発生した場合のみです。実行中の割り込みルーチンの優先度より低い優先度の割り込みが発生した場合は、優先度の低い割り込みはその割り込みを保留されます。

この多重割り込みの優先度は、プログラムの割り込みルーチン中に決定することができます。

◆17Kシリーズの割り込みとメモリ

ということで、17Kシリーズの割り込みは、割り込みの優先度を変更することが可能で、多重割り込み方式を採用し、割り込み要因数も多重割り込みの多重度もアーキテクチャ上の上限はないことになっています。という持って回ったいいかたをするのは、多重度に関しては、戻り番地の管理をスタックで行なう関係上、スタックの制限を受け、最高16レベルまでとなるからです。

17Kシリーズの割り込みで使用されるメモリ等として次のようなものがあります。
それぞれ簡単に説明したいと思います。

INTEフラグ、IPxxxフラグ、IRQxxxフラグ、INTxxxフラグ、IEGxxxフラグ

まず、INTEフラグ。INTEとはINTerrupt Enableのことで、マスカブル割り込みの許可、不許可を選択するためのフラグです（当り前のことですが、ノンマスカブル割り込みは許可、不許可を選択できません)。

このフラグがセット（1）されるとすべてのマスカブル割り込みが許可され、リセット（0）されると禁止されます。INTEフラグの操作は専用の命令があり、EI命令とDI命令です。DI命令が実行されるとその命令サイクル中にマスカブル割り込みは禁止されます。EI命令が実行されるとEI命令の次の命令が終った時点でマスカブル割り込みは許可されます。

EI命令だけ、このような時間差攻撃をするのは、当然理由があるからです。

割り込みが受け付けられた時は当然EI状態です。でも17Kシリーズでは割り込みルーチンの最初ではDI状態になります。これを元に戻すために割り込みルーチンからの復帰命令（RTI命令）の前にEI命令を実行するようにします。

もし、EI命令のサイクル中に割り込みが入ったら、せっかく退避、復元した情報が意味を持たなくなります。このようなことを繰り返すと、動作不良の原因ともなりかねません。そこで、EI命令を次の命令、つまりRTI命令の実行が終るのを待って、マスカブル割り込みを許可状態にするのです。

次は、IPxxxフラグです。IPとは、Interrupt Permissionのことで、割り込みの許可、不許可を選択するためのフラグです。え、INTEフラグと同じ説明ですって？ではこの2つのフラグの違いを説明すると、INTEフラグがすべての割り込みの許可、不許可を選択するのに対して、IPxxxフラグは各割り込み要因ごとの割り込みの許可、不許可を選択します。17Kシリーズでは、このフラグで割り込みを禁止できる割り込みをマスカブル割り込みと呼んでいます。
このフラグがセット（1）の状態だと対応する割り込み要求は受け付けられ、リセット（0）の状態だと対応する割り込み要求は保留されます（つまりIPxxx＝0では、対応する割り込み要求は、発行はされても、受け付けられない)。
当然、対応する割り込み要求を受け付けられる状態にするためには、すべての割り込みが受け付けられる状態、つまりINTE＝1でなければなりません。コントロール・レジスタにありますので、ウィンドウ・レジスタとPEEK／POKE命令で変更することができます。このフラグを操作することにより、ハードウエアで決定された優先順位を変更することができます。

IRQxxxフラグは、Interrupt ReQuest、つまり割り込み要求関係のフラグです。割り込み要求の何を示すかというと、割り込み要求が発行されるとセット（1）され、割り込み要求が受け付けられるとリセット（0）されます。
このフラグもIPxxxと同様、各割り込みの要因ごとに割り当てられているフラグですので、このフラグの内容を調べることにより、各割り込み要因ごとの発行、受け付け状況を知ることができます。このフラグがある場所もIPxxxと同様、コントロール・レジスタですので、PEEK命令を実行すると、ウィンドウ・レジスタにIRQxxxフラグの内容が読み出され、フラグの内容を調べることができます。

逆にPOKE命令を使用し、IPxxxフラグを変更するとどうなるか？これは、1を書き込むと、その割り込み要求が発行されたと同じ意味になります。これはプログラムで意図的に割り込みを発行させることですから、『ソフトウエア割り込み』ということになります。当然条件が整えば割り込みが受け付けられ割り込みルーチンが実行されます。
では、既に1になっているところに0を書き込むと（0のところに0を書き込んだって…)。既に1になっているということは、割り込み要求が発行されている、ということです。でも0でないということは、まだその割り込みが受け付けられていない、ということです。これを強制的に0に変更するのですから、結果的には割り込み要求が発行されたにも関わらずまだ受け付けられていない割り込みをキャンセルすることになります。

この他にも、割り込みの状態を知るためのフラグとしてINTxxxというフラグがあります。このフラグは、割り込み端子と対応しており、このフラグにより、割り込み信号の状態を知ることができます。割り込み信号の状態を知ってもあまり意味がない場合には、このフラグは割り当てられないこともあります。

IEGxxxフラグは、Interrupt EdGe direction、つまり周辺ハードウエアからの割り込み信号のエッジの方向を選択するためのフラグです。
このフラグが、セット（1）の状態だと信号の立ち下がりエッジで割り込み要求が発行され、リセット（0）の状態だと信号の立ち上がりエッジで割り込み要求が発行されます。マスカブル割り込みの要因ごとに割り当てられているはずのフラグですが、割り込み要因数が品種ごとに違うので、このフラグの数も品種ごとに違いますし、品種によっては、エッジの方向が固定されていたり、信号のレベルにより割り込みが発行されるものもあるため、割り込み要因数とIEGxxxの数が一致するとは限りません。
このフラグの目的は、信号のエッジの方向を選択し、どのタイミングで割り込み要求を発行するかを選択することです。このフラグも、コントロール・レジスタにありますので、PEEK／POKE命令で変更することができます。

◆VAG

割り込みルーチンの先頭アドレスをベクタ・アドレス、ベクタ・アドレスが固定されている割り込み方式をベクタ割り込みといいます。
17Kシリーズは、基本的にはベクタ・アドレスはハードウエアで固定となっています。しかし、世の中には状況により柔軟な対応や高速な応答が求められることもあるのです。というわけで、そんな時のためにも17KシリーズはVAGを用意しています。
VAGは、フラグでもメモリ機構でもありません。ベクタ・アドレスを生成する機構がVAG（Vector Address Generator）です。

◆割り込み時の動作

割り込みが受け付けられると、割り込みルーチンへ分岐する前に、CPUは次の動作を行ないます。

INTEと割り込みに対応するIRQxxxをリセット（0）
スタック・ポインタの内容をデクリメント（−1）
プログラム・カウンタの内容をスタック・ポインタで指定されるアドレス・スタック・レジスタへ格納
VAGから出力されるベクタ・アドレスをプログラム・カウンタに格納（一部の品種のみ）
BANK、IXEなどを専用の割り込みスタックへ格納

これだけの処理を1サイクル中に行ないます。ただしこの場合の1サイクルは通常のサイクルと違い、『割り込みサイクル』といいます。したがって、割り込みが受け付けられてから、割り込みルーチンへ分岐するまでに1サイクルを有します。

逆に、割り込みルーチンからの復帰命令が実行されるとCPUは、割り込みルーチンからの復帰命令のサイクル中に次の動作を行ないます。

割り込み時にスタック・ポインタで指定されていたアドレス・スタック・レジスタの内容をプログラム・カウンタに格納
専用の割り込みスタックからBANK、IXEなどに格納
スタック・ポインタの内容をインクリメント（＋1）

◆割り込みとスタック

ここでいうアドレス・スタック・レジスタとは、『プログラムの分岐』のところで出てきたアドレス・スタック・レジスタと同じものです。スタック・ポインタも同様です。しかし、割り込みスタックは今まで出てきませんでした。のでここで説明しましょう。割り込み（や分岐）の時には次に実行する予定だった命令のある番地、プログラム・カウンタの値をアドレス・スタック・レジスタに格納しました。割り込みが終了したあと、その値が必要になるからです
では、退避する値はプログラム・カウンタだけでいいのでしょうか？ システムすべての情報が割り込みの前の状態に戻ればいいのですから、プログラム・カウンタだけでもいい場合もありますが、そんなことはまれです、割り込みをサポートしている品種では。
システムのすべて、とまではいかなくとも割り込み時に最低限の情報を退避するための記憶場所として、割り込みスタックがあります。この割り込みスタックに退避することができる情報としては次のようなものがあります。

AR、BANK、IX、RP、PSW、IPxxx

しかしこのうち自動的に、割り込み時に退避、割り込みからの復帰時に復元されるのは、BANKとIXEだけです。これらのレジスタはハードウエアで退避され、退避されたあとのレジスタの内容はリセット（0）されています。

あとの情報は、自動的には退避、復元は行なわれません。すべてプログラマの責任で退避、復元しなければなりません。

第7章 開発ツール

歴史のところでも少し出てきましたが、17Kシリーズほどツールの重要性をプロセッサの設計時に考慮したマイコンはありません。17Kシリーズのツールは4ビット・マイコンの開発環境として最高レベルにあります。ここでは、これらの開発ツールについて見ていきたいと思います。

現在の17Kシリーズの開発ツールとして挙げられるのは次の3つです。

AS17K, IE-17K, SIMPLEHOST

AS17Kはアセンブラ、IE-17Kはインサーキット・エミュレータ、SIMPLEHOSTはディバッガ・インタフェース・ソフトウエアです。

● AS17K

まずは17Kシリーズの開発ツールの一番目。何はなくともこれがなければプログラムの開発は始まらないというアセンブラです。17KシリーズのアセンブラはAS17Kといいます。覚え易いでしょ。ではまずアセンブラの解説から。

アセンブラとは、アセンブリ言語で記述されたプログラム（ソース・プログラム）を機械語に変換するためのソフトウエアです。アセンブリ言語で記述されたプログラムをアセンブラを用い機械語に変換することをアセンブルするといいます。機械語に変換されたプログラムは、（とりあえず）ファイルに格納されます。このファイルをオブジェクト・ファイルといいます。専用マイコンのためのプログラムの記述言語はたいていアセンブル言語です。ハードウエアに密着した（無駄のない）プログラムが要求されるからです。

普通、アセンブラはプロセッサに依存したものがほとんどです。いい替えればプロセッサごとにアセンブラが存在するわけです。たとえばAというプロセッサ用のアセンブラはAというプロセッサ用のオブジェクト・ファイルしか出力しません。もし、Aというプロセッサ用のアセンブラが出力したオブジェクト・ファイルをBというプロセッサで動かしたら、まず間違いなく動作しません（その前に動かす方法があるかな？）。

Aというプロセッサ用のアセンブラがAというプロセッサ用のオブジェクト・ファイルしか出力しないのは、間違いありませんが、Aというプロセッサ用のアセンブラがCというプロセッサ上で動いていることはあります。これをクロス開発といいます。17Kシリーズがそうですし、専用マイコンのほとんどがクロス開発を行なっています。専用マイコンでは、プログラムが動作する環境（実行環境）とプログラムを開発する環境（開発環境）が同じことはほとんどありません。

では、ファミリー展開が要求される専用マイコンの世界では、アセンブラはいくつ必要になるのでしょうか？ アーキテクチャが統一されているプロセッサでも、1つではまかないきれないのが一般的です。最悪の場合、個々の品種ごとにアセンブラが存在する可能性さえあるのです。当然、あたらしい品種がメーカーにより作成されたら、その品種用のアセンブラが作成されるのをじっと待つという事態になりえます。

この問題を、プロセッサの開発時に開発ツールのことまで考えていた17Kシリーズでは、どのようにして解決したかというと、シリーズすべてに共通なアセンブラ本体と、個々の品種に依存した部分を集めた『デバイス・ファイル』に分けたのです。このアセンブラがAS17Kです。新しい品種ができてもアセンブラ本体を作り直す必要はない、ということは、新しい品種ができてもすぐにアセンブラが手に入る、ということです。新しいデバイス・ファイルを作成してもらえばいいのですから。

◆アブソリュート・マクロ・アセンブラ

デバイス・ファイルという画期的な方式により17Kシリーズの全品種において使用することが可能になったAS17Kですが、個々の品種に対する特徴も、いくつか注目すべき点があります。まず1番最初は、『アブソリュート』です。

アブソリュートとは絶対的、という意味です。つまり機械語に変換する際に、命令を割り付けるアドレスをあとから決定するのではなく、変換時に決定してしまうのです。

アブソリュートとでないアセンブラといえば、リロケータブル（再配置可能）・アセンブラですが、リロケータブル・アセンブラにおいては、アセンブル時にはすべてのアドレスは決定されず、リンク時にすべてのアドレスが決定されます。そのためにはソース・プログラムとアドレスが決定されたオブジェクト・ファイルの間に、中間オブジェクト・ファイルというファイルが必要になります。つまり、リロケータブル・アセンブラはソース・プログラムを読み込んで、中間オブジェクト・ファイルを出力します。そしてリンカはアセンブラが出力した中間オブジェクト・ファイルを読み込んで、オブジェクト・ファイルを出力するわけです。
一度、アドレスが決定されていない中間オブジェクト・ファイルを出力するが故に、ソース・プログラムを分割し、分割したファイル単位（これをモジュールといいます）でアセンブル処理ができるのです。そして、分割したファイル単位でアセンブルされ出力された中間オブジェクト・ファイルは、リンカにより（未決定のアドレスを決定し）、結合されます。

これに対して、普通のアブソリュート・アセンブラは、アセンブル時にすべてのアドレスが決定され、中間オブジェクト・ファイルを出力する必然性がありません。となると、オブジェクト・ファイルを出力するために必要な情報をすべて読み込んでアセンブル処理する必要があります。すると、ソース・ファイルを分割することができない、という結論になります。

しかしAS17Kは違います。ソース・プログラムを分割することができます。モジュールごとのアセンブル処理はできませんが、モジュールごとの中間オブジェクト・ファイルを出力することができます。これにより、出力された中間オブジェクト・ファイルとソース・プログラムの作成日時を比較し、ソース・プログラムの方が新しいモジュールとそのモジュール以降のプログラムのみをアセンブルの対象とすることができます。変更のなかった中間オブジェクト・ファイルをできるだけアセンブルの対象としないことで、アセンブル時間の短縮を図ります。

> なぜ、モジュール『以降』なのかというと、AS17Kが出力する中間オブジェクト・ファイルで決定されていないアドレスはモジュール間にまたがって分岐するアドレス情報だけで、ほとんどのアドレスは決定されているからです。あるモジュールで変更があった場合、アドレス情報も変更されていることがほとんどです。当然変更のあったモジュール以降のアドレスにも影響を及ぼす可能性が十分あり、アセンブルが必要だからです。

ちなみに、ソース・プログラムを分割して記述できることは、プログラムの構造化の機能と相まってAS17Kのプログラミングに大きな影響を与えています。

マクロとはもともとは巨視的、という意味ですが、プログラミング言語の世界では、命令の集まったものを意味します。定形的な命令の繰り返しをマクロ化することで、プログラミング効率がさらに良くなります。ただし、マクロはあるだけでは余り意味はなくマクロ機能を強力にサポートしていて始めてマクロが使えるのです。という説明を行なったからには当然AS17Kのマクロは『使え』ます。

◆読み易さと日本語

もともとアセンブラは、数字の羅列だけの機械語では理解できない、という背景のもと作られました。したがってアセンブリ言語で記述されたプログラムは（機械語などとは比べ物にならないほど）読み易くなっています。でも、プログラミング言語以外も含む他の言語と比較した場合、決して読み易いとはいえません。

基本的にはプログラムを記述するということは、処理の論理的展開を記述することです。ましてやアセンブラは機械語と1対1で対応をとることを義務づけられています。しかし読み易さを考えた場合、できるだけ自然言語に近い方が読み易いに決っています。ここにアセンブラのジレンマがあります。読み易さだけを考慮すれば（論理的でないこともある）自然言語に近付ければいいのですが、論理的展開を記述し、機械語と1対1で対応付けなければならないことを考えると、あまり大幅な言語体系の変更はできません。

そんな中でより読み易いアセンブラを記述するための試みが構造化であり、マクロであり、コメントなのです。構造化により全体の流れを把握し、マクロにより細かな処理の集まりを『巨視的』に捉え＝他の言葉に置き換え、コメントによりプログラミング言語としては省略されがちな詳細な説明を付け加えておく、といった処理を現在のアセンブラでは行ないます。というより持てる力の大部分をこのための処理に割いています。

初期のアセンブラが機械語と1対1の言葉の置き換えをする単なる変換機という役割が強かったのに対し、現在のアセンブラは、機械の論理を人間が理解し易いようにするための装置といった役割が強くなってきている、ともいえます。

『機械の論理を人間が理解し易いように』ということをする装置を普通『インタフェース』というと思ったんだけど、アセンブラがインタフェースだという話はあまり聞かないなぁ。

そういった状況で、英語圏の人以外はいささかのハンデを背負っています。プログラミング言語をどの（自然）言語で記述するか、という問題です。コンピュータやアセンブリ言語が英語圏で開発されたのは周知の事実です。しかし、だからといってアセンブリ言語が英語だけ扱えればいい、という理屈は通用しません。せっかくソフトウエア工学が発達し、より人間が理解し易い形でさまざまな情報をアセンブラが出力するのに、アセンブラが出力した結果をさらに英語から他の言語へ変換する作業をしなければならないというのも、変な話です。というよりその前に、アセンブラが理解できる言語が英語を元にしたアセンブリ言語だけであるということ事態に問題があるような気がします。

プログラミング言語の母国語化については、『プログラミング言語に使われる簡単な英語くらい理解できないで、プログラムを記述すること…』、といった話が以前ありましたが、GOTOぐらいの英語だけならばともかく、ドキュメントになるくらいのコメントをソース中に埋め込み、いくつかの処理をまとめて名前を付け、プログラムの論理ブロックごとに全体の流れを理解するための文章を記述しなければならない場合に、いくつかの予約語のみを他言語で記述することを強要するのは、いささかナンセンスだといわなければなりません。あ、もちろんこれは、予約語以外のコメント等を母国語で記述できる場合の話ですが。

といったこととはおそらく関係なく、AS17Kではプログラムの記述に日本語が使えます。それも、コメントやデータとして使用できる、といった中途半端なレベルではなく、シンボル名（いわゆる変数など）、マクロ名、レーベル名として日本語を使うことができます（当然コメントやデータに使用できることはいうまでもありません）。
ちなみに、エラーメッセージやアセンブラの処理過程もほとんどすべて日本語で出力されます。さらにいえば、バージョン1.1以降のAS17Kでは、予約語を他の言葉（もちろん日本語を含む）に置き換えられます。このような使い方はあまりお勧めはできませんが。

AS17Kは、PC-9801のMS-DOS上で動作し、標準テキスト・ファイルを入力ファイルとします。ということは、好みのかな漢字変換ソフトウエアと好みのテキスト・エディタを使用してソース・プログラムを記述することができるということです。日本語入力にはいささかの手間がかかるかもしれませんが、あとの保守などを考えた場合、日本語でプログラムを記述することをお勧めします。

この文章は別に英語圏の人にまで日本語で記述することを強要しているわけではありません。当然ソース・プログラムは英語のみで記述することもできますし、お好みならばエラーメッセージやアセンブラの処理過程も英語で出力することができます。要するに母国語もしくは母国語と同じくらい理解し易い言語で、ソース・プログラムを記述するのが好ましい、ということです。英語と日本語以外を母国語としているかたはごめんなさい。

◆シンボル定義

AS17Kでは、データ・メモリやプログラム・メモリの値を使用するときには、命令のオペランドにメモリのアドレスを直接記述することはせずに、一度データ・メモリやプログラム・メモリのある箇所に名前を付けてから使用します。この名前をシンボル名といいます。なぜこんなことをするかというと、データ・メモリやプログラム・メモリのアドレスといった数字の羅列を理解するより、意味のある名前を付けた方が解り易いからです。命令を数字で記述するより名前を付けてその名前を記述した方が解り易いのと同じ意味です。

これだけでも十分だと思うのですが、もう1つくらい理由を挙げるならば、プログラムの記述ミスを減らすためです。命令のオペランドとして意味のある名前を記述するということは、常に記述したアドレスに何が入っているかを把握することです。少なくともアドレスを直接記述して自分の頭の中で『あそこのアドレスにはあのデータが入っているからここにこれを記述すると･･･』と考えるよりは、間違う可能性が低いと思うのですが。

ちなみに、AS17Kでは、シンボル名に253文字まで使用できます。つまり、十分に意味のある名前を付けることができるということです。昔のXXXXという言語の用に8文字しか名前が付けられないようでは、意味のある名前を付けたくたって･･･。

◇コラム：言語処理ソフトウェアの話

一般にコンピュータの言語処理においては2つの考え方があります。

自由な記述が可能 文法が極めて厳格

今までのコンピュータの言語処理ソフトウエアでは、文法が極めて厳格、というのが多かった様です。でもこれは何らかの哲学があってそうしていたわけではなく、厳密な定義をしていないと、言語解析ソフトウエアが意味不明として処理してくれなかっただけなのです。AS17Kでは、『文法が極めて厳格』の方に分類されます。でもこれは今までのように、言語解析が難しいからではなく、いくつかのメリットがあると判断されたためです。

修正が楽 読み易い 論理的な間違いが検出し易い

などです。修正が楽というのは、自由な記述が可能な言語では、基本的には自由な修正が可能です。でも、場合によってその自由度が変わることもあるのです。1つの修正に対して、文法が極めて厳格な場合修正の方法は1とおりですが、自由な記述が可能な場合あるときは3とおりの修正しか許されずあるときは1367とおりの修正が許されている、ということも考えられます。文藝作品を書くのならば、自由な記述が可能な方が、修正も楽でしょうが、プログラミング言語のように、論理的処理を記述しなければならない場合は、文法が極めて厳格で、修正内容が一義的に決ってしまった方が、楽なのではないでしょうか？

文法が極めて厳格ならば次に記述されるタイプが決られています。ということは人間が読むときに次に記述される内容を予想できるということです。たとえばアセンブラなら、命令のあとにはオペランドがくることが基本です。これが命令の前や後にオペランドがあったら、たとえこれを言語処理ソフトウエアは理解できても、人間が理解するには相当な努力を要求されるはずです。

これは、言語処理ソフトウエアにとっても同様で、次に記述されるタイプが決められているということは、決められたタイプ以外のものがきた場合は、間違いが発生している可能性が高い、ということです。

ということで、コンピュータの世界の言語処理では、文法が極めて厳格な方がいい場合もあるということでした。こういうAS17Kとあまり関係なさそうな話を長々としてきたのは、プログラムの質を向上させるために、17Kシリーズは、色々な努力をしているということを、説明したかったからです。
すべてのプログラマが、昔のように、1ステップを削ることのみに熱を上げ、トリッキーな手法を使うことに喜びを感じているのならば、自由な記述が可能な言語の方が受けいられるかも知れませんが、今はそういう時代ではありません。普通の人が普通の業務としてプログラムを記述しなければならない時代なのです。そんなときに、プログラムの質を向上させるために、誰でも同じ様な質のプログラムを記述できるような環境や、ツールや、言語を提供しているのが、17Kシリーズなわけです。

◆疑似命令と制御命令

アセンブラは、機械語を生成するためのソフトウエアです。では何を元に機械語を生成するかというと、ソース・プログラムです。でもソース・プログラムには、機械語に変換する必要のない情報も記述します。その中の1つが疑似命令です(ちなみに機械語に変換される命令はただ単に『命令』といいます。機械語に変換される命令であることを強調したい場合には『機械語命令』などという場合があります)。
疑似命令そのものは機械語に変換されませんが、命令を機械語に変換するのに重要な役割を担っています。先ほどのシンボル定義も疑似命令ですし、マクロを定義するのも疑似命令です。さらに命令をアセンブルするかしないか、同じ部分を何回アセンブルするか、などを指定することができます。
つまり疑似命令とは、どのような機械語を出力するかを指定するための(疑似的な)命令群である、ということができます。アセンブリ言語で記述された命令をプロセッサにおいて用意された機械語に1対1で変換することが仕事のアセンブラとしては、変換すること自体は正確にできて当り前、この疑似命令が商品としての生命線ともいえます。
当然AS17Kもこの疑似命令を強力にサポートしています。どれほど強力なのかは・・・、AS17Kのユーザーズ・マニュアルをご覧ください。

AS17Kには、疑似命令の他に制御命令というのがあります。
疑似命令では、アセンブラがどのような機械語を出力するかを指定しました。制御命令では、アセンブラがどのようなリストを生成するかを指定します。これは他のアセンブラにはあまり見られないものです。他のアセンブラでも、アセンブル・リストを出力するかしないか、ぐらいの指定はできます。でも普通は疑似命令の一部として機能し、それだけの役割しか与えられていません。それなのにAS17Kで、わざわざ『制御命令』という分類を作りだしているのは、それだけAS17Kが出力するリスト類が豊富で、リストの出力様式などを制御するための命令が強力だということです。

◆出力ファイル

アセンブラは、ソース・プログラムを機械語に変換するためのソフトウエアです。17Kシリーズのプログラムはに格納されますので、ROMに格納するための用意が必要です。プログラムをROMに格納するためには、HEXファイルかPROMに格納する必要があります。プログラムをPROMに格納するには、PROMライタを使用しますが、PROMライタに入力できるようにしたファイルが必要になります。したがって、AS17Kでは、HEXファイルとPROMファイルを出力できます。

AS17Kは分割アセンブルすることが可能であることは説明しました。したがってこのために必要な中間オブジェクト・ファイルを出力することもできます。

そして、AS17Kの出力ファイルで一番インパクトがあるのが、ドキュメントでしょう。ソース・プログラム中でせっかくコメントやシンボルを日本語で記述し、論理的なブロックに分割しその概要までを記述したのです。これはいってみれば立派なドキュメントの一種ともいえます。でもそれだけでは凄く良いソース・プログラムとしか評価されません。これらの凄く良いソース・プログラムから本当の日本語の（もちろん日本語で記述してあればの話ですが）ドキュメントとその目次まで自動生成してしまうのです、AS17Kは。17Kシリーズの流儀にのっとってソース・プログラムが記述してあれば、もう上司から『ドキュメントはどうした』といわれ、『これから作ります』と答えなくても良いようになります。

当然、これだけのドキュメントまで作ってくれるのですから、他のリスト類も抜かりはありません。マップ・ファイルも未定義シンボルの一覧もクロスレファレンス・リストもアセンブルのリポートも必要ならば出力してくれます。

あれ何か抜けているような・・・。そうだ、アセンブル・リストを忘れていました。プログラムの間違いを修正するにはこれがなくてはネ。ちゃんと生成されるオブジェクト・コードもそのロケーション・アドレスもエラーの有無もネストのレベルも一緒に出力されます。

これだけの情報が出力されるんですけど、SIMPLEHOSTを使ってしまうと、これらをプリンタに出力する機会も減ってしまうんです。すべてコンピュータの画面上で確認できるようになりますから。

● IE-17K

IE-17Kはインサーキット・エミュレータです。これを使って何をするかというと、プログラムの間違いを取り除きます。プログラムの間違い（バグ）を取り除く作業をディバグといい、ディバグをするための道具をディバッガといいます。

製品となるものにプログラムといえども間違いがあることは許されません。製品にする前にすべてのバグを取り除かなければならないのですが、その前にプログラムに間違いがあることを発見しなければなりません。そのためには、一度実行させてみるのが一番です。

先ほど専用マイコンは、実行環境と開発環境が違う、という話をしました。汎用マイコンだと実行環境と開発環境が同じという場合も結構多いので、オブジェクト・プログラムができたら、開発用のコンピュータで即実行という安直な方法が取れないわけではありません（本当は汎用マイコンでもディバグ用のソフトウエアを起動し、その制御下で実行します）。
でもシングルチップ・マイコンのほとんどはプログラムをROMに格納します。汎用マイコンのように、外部RAM上で、簡単に実行させるわけにはいかないのです。だからといって、プログラムを作成するたびにROMに格納して実行してみるわけにもいきません。ご存じのとおり、プログラムのROM化には膨大な時間とコストがかかるのですから。

では、ROMに格納せずにどうするか？ ROMで動いているように見せかける、しかありません。このための装置がエミュレータです。エミュレータの定義としては、今開発中の装置からみて、実際のチップと同じ動作をする装置、といったところです。
汎用マイコンでも実行環境と開発環境が違う場合にはエミュレーションで対応する場合があります。この場合のほとんどはソフトウエア・エミュレーションです。しかし、専用マイコンでは、この方法はまず取られません。なぜなら、シングルチップの、専用マイコンのプログラムのディバグは電気特性など細かな点まで実行環境と同じであり、実際の製品に内蔵させて動作することまで要求されるからです。

シングルチップのディバグにも対応できるように作られたエミュレータが、インサーキット・エミュレータです。インサーキット・エミュレータを使えば、実際のハードウエアに乗っているようにソフトウエアのディバグが行なえます。

◆エミュレータの歴史

エミュレータにもそれなりの紆余曲折があります。17Kシリーズのエミュレータは特殊な方式を使っています。この特殊な方式を説明するためにも、簡単に今までのエミュレータの歴史をふりかえってみたいと思います。この話はプロセッサの開発者の苦労話くらいの気持ちでお聞きください。

プログラムをROMに格納することができない、この問題からエミュレータの歴史は始まります。ではROMに格納できないならばプログラム・メモリをRAMまたはPROMで構成すればいい。これは結構誰でも思い付くことです。でも昔は外部から命令を入力できなかったり、実際にRAM等で構成しても高いものだったり、サイズが大きくなったりで、実用的ではありませんでした。
ROMを取り除き、外部から命令を入力できるようにしたエミュレータが使用できるようになっても、価格的に高いものでありことは変わりありません。また、エミュレータを作る側としては、品種の多さに対応するためにあらかじめ最大アーキテクチャ品種のエミュレータを作ります。これではすでにそのエミュレータで使える品種が決められてしまい、新しい品種には簡単に対応できません。さらに最大アーキテクチャが決っているのですから、エミュレータ以上の機能を持った品種が開発される可能性がほとんどゼロという、今にして17Kシリーズを見ながら考えるととんでもない方法でした（でも当時はこれでも仕方がなかったようです）。

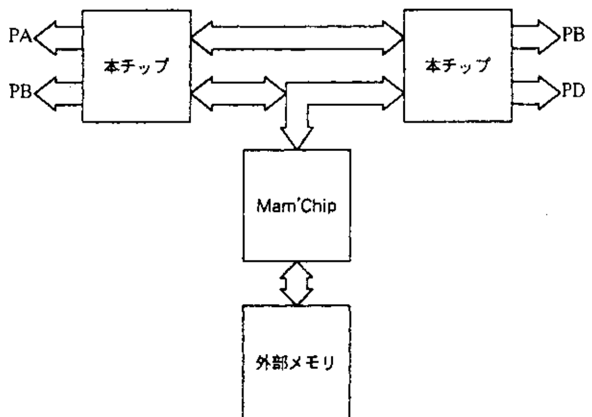
では、エミュレータの周辺回路は外部回路で拡張すればいい、と考えたようです。こうすると新しい品種に対する問題はある程度は解決します。がこれでもまだベストではありません。価格もさほど目立って下がったわけではありませんし、周辺回路を増やすたびにサイズが大きくなります。それよりもこの方式の最大の問題点は、外部回路をディスクリートで構成したためエミュレータと実際に使うチップ（本チップ）の特性等が一致しなかったことです。

ではいっそのこと本チップとエミュレーション・チップを一緒にしてしまえばいい、という大胆な発想が生まれたのはアメリカでした。でも、ディバグ用に余分な端子が必要だったり、外部メモリを接続するために汎用ポートを使用するためエミュレータでは使用できないものが本チップでは使用できるといった問題があり、他のプロセッサのエミュレーション方式に採用されるまでには到りませんでした。なんせ価格が最大の関心となる民生市場ですから、そんな大胆な…。

そこで、17Kシリーズの開発者たちが考え付いたのが『Mam'Chip』方式だそうです。

◆Mam'Chip方式

Mam'Chip（マム・チップ）の最大の特徴は、どんな特殊な周辺がつまれても本チップとまったく同じに動作すること、といわれています。でもこれはMam'Chipを知っている人にいわせると当り前だそうです。なにせMam'Chipには本チップが使われているのですから。



図をみても解るとおり、1つのチップをエミュレータするのに2つの本チップを使用しています。2つの本チップは、ポートを使用してデータ・バスとアドレス／インストラクション・バスで接続されています。こうすることにより無駄なポートというものが存在しなくなります。アドレス／インストラクション・バスは問題のMam'Chipにも接続されています。このチップを通して外部プログラム・メモリを接続しています。ここにプログラムを格納します。
ちなみに、Mam'Chipは普通のエミュレータの周辺回路をゲート・アレイで組んだだけとのことで、価格的にも期待できます。

この方式ならば、特性まで一致したディバグができますし、本チップのハードウエアが変更になっても新しい品種が作られてもすぐにエミュレータを手にいれることができますしし、使えない端子もない、といいことづくめです。

◆インサーキット・エミュレータによるディバグ

というわけでIE-17Kのエミュレーション方式の良さは理解していただけたと思います。でも、こんなことは実に内部的な話ですのであまり詳しくは覚えなくてもいいです。『Mam'Chip方式』という名前と無駄がない新しい方式だということだけ覚えておいてください。

わざわざインサーキット・エミュレータを使うのですから、プログラムを実行させるだけではもの足りません。ということで、普通のインサーキット・エミュレータには、プログラムのディバグ機能も付いています。

プログラムのディバグに要求される機能としては、まずプログラムの実行。これは先ほどから説明しているように、エミュレータを使って実行できます。
次はプログラム・メモリの制御、というよりプログラムの実行の制御です。普通に実行させるだけならば、エミュレータを使うだけでいいかも知れませんが、ディバグをするためには、プログラムを1ステップづつ実行したり、任意の番地から実行したりする機能が必要になります。プログラム・メモリの制御ができるからにはデータ・メモリの制御もできます。任意のアドレスに任意のデータを格納することができます。プログラム・メモリとデータ・メモリの制御を組み合わせれば、任意の範囲のプログラムを任意のデータで実行できます。
最後になりましたが、ディバグには欠かせない機能として、ブレークとトレースです。
ブレークは、プログラムを止める機能ですが、ただ単に止めるだけではあまり意味がありません。特定の条件が成立したらプログラムを自動停止させる機能です。ブレークするための条件としては次のものが使用できます。

> プログラム・メモリ・アドレス、データ・メモリ・アドレス、データ・メモリの内容、
> レジスタ・ファイルの内容、スタック・レベル、外部端子の状態、割り込み、DMA、
> 命令コード、命令実行数、条件成立回数

トレースは、実行状態を追跡する機能です。プログラムが最初から終りまでまっすぐに実行されるだけならばさほどのメリットはありませんが、普通プログラムは、さまざまに分岐します。それも条件によって、あっちへ飛んだり、こっちへ飛んだりです。そんなときに、いつ、どこのプログラムが実行されたかを時間を基準に記録すれば、簡単にプログラムの流れを把握することができます。トレースする項目としては次のものが記録されます。

> プログラム・メモリ・アドレス、実行した命令コード、スキップされた命令、
> 書き込まれたデータ・メモリ・アドレス、書き込まれたデータ・メモリの内容、
> IE-17Kのロジック・アナライザ端子の状態、各実行命令の相対時間

あと、ディバグとはいえないかも知れませんが、IE-17Kには、カバレージという機能もあります。これは、ディバグというよりプログラムの検査に使います。

これは、以前はROMサイズに制限のあるシングルチップ・マイコンの専売特許だったのですが、いまでは、汎用マイコンのプログラムをディバグするためのソフトウエアにもカバレージの機能が付いています。この場合はプログラムの命令ごとの実行回数を記録するだけですが、IE-17Kのカバレージ対象は、プログラム・メモリとデータ・メモリで、プログラムの命令ごとの実行回数とデータ・メモリに何が書き込まれたかを記録します。

◆IE-17Kの内部とCLICEとインタフェース

IE-17Kは、実物の外観を見ていただければ解るとおり、A4サイズです。このA4サイズの中に3枚のボードが入っています（正確には2枚半）。スーパーバイザ・ボードとメモリ・ボードとSEボードです。

このうち、スーパーバイザ・ボードとメモリ・ボードはすべての品種において共通ですが、SEボードは品種によって違います。なぜならSEボードには先ほどの本チップとMam'Chipで構成されたエミュレータが乗るからです。実はSEボードは、単独でシステム評価用ボードとしても使用することができます。SEボードにはIE-17Kと接続して使うときのためのプログラム・メモリとしてCMOSスタティックRAMが搭載されています。また単独で使用するときのためにPROMが搭載される場合もあります。
またメモリ・ボードには、プログラムのトレースやカバレージのためのメモリが搭載されています。トレース・メモリの大きさは32Kステップあります。
スーパーバイザ・ボードには16ビットCPU V30が乗っています。このCPUによりインタプリタを駆動し、IE-17Kでのディバグが簡単になるわけです。このインタプリタをCLICEといいます。

CLICE（クライス）はCommand Language for In-Circuit Emulatorの略で文字どおりインサーキット・エミュレータのためのインタプリタです。CLICEは、コマンド言語というくらいですから、コマンドの連続実行や繰り返しといった機能まで備えています。

基本的にIE-17Kは、ホスト・コンピュータとRS-232Cのシリアル回線をつないで使用するのですが、このときユーザ・インターフェースとなるのがCLICEです。ただしあとで説明するSIMPLEHOSTがホスト・コンピュータ上で稼働している場合はSIMPLEHOSTがユーザ・インタフェースとなります。

RS-232Cのシリアル回線は2チャンネルあり、1つはホスト・コンピュータとの接続に使用しますが、もう1つはPROMプログラマなどを接続します。当然、ホスト・コンピュータとのプログラムの双方向転送ができますし、PROMプログラマを接続すれば、PROMファイルの形式で、プログラム・メモリの内容を出力できます。

また、IE-17Kは14チャンネルのパターン・ジェネレータを内蔵しています。プログラム可能なステップ数は8Kステップで、1μsから13333μsで実行可能です。

SIMPLEHOST（シンプルホスト）はSIMPLE and High level Operation Support Toolsの略で、ディバッガ・インタフェース・ソフトウエアです。マイコンの開発環境に詳しい方でも、この言葉を聞いた方は少ないのではないでしょうか？ なにせおそらくSIMPLEHOSTで一番最初に使われた言葉でしょうから。

いままで、IE-17Kを始め、インサーキット・エミュレータが高機能なものでした。しかし、一部のインサーキット・エミュレータは、高機能である反面低レベル・インタフェースしか持ちませんでした。IE-17KはCLICEというインタプリタを搭載し、他のインサーキット・エミュレータに比べれば、格段の高レベル・インタフェースを維持してきました。でもまだ、始めての人でも学習しなくても使える、使いながら学習できる、ということはありませんでした。これを実現したのが、SIMPLEHOSTです。
CLICEとSIMPLEHOSTでは、その操作において大きな違いがあります。CLISEはキーボードを使いますし、SIMPLEHOSTではマウスを主に使います。これはどちらの方が優れているとかいう問題ではありません。どちらが要求されているかの問題です。他のインサーキット・エミュレータに慣れた方は、CLICEをお使いになれば、その高機能に驚かれるでしょうし、初心者の方は、SIMPLEHOSTをお使いになれば、そのやさしさに驚かれるでしょう。どちらでもお好きな方を選択してください。どちらでも、できることはほとんど同じですから。
ただし、初心者の方はSIMPLEHOSTをお勧めします。なにせ学習しなくても使えるというのは、大きなメリットだと思いますから。

◆何ができるか

SIMPLEHOSTを使うと、マニュアル・フリーで、できることがみえ、できたことがみえます。
SIMPLEHOSTは、MS-Windows上で動作するソフトウエアです。したがって、インタフェースの大部分をMS-Windowsに準拠します。

SIMPLEHOSTは専門家でない人がプログラムのディバグを行なうことを想定して、だれでも簡単に操作できるようになっています。だれでも簡単に操作できるためには、詳しい人におうかがいをたてたり、分厚いマニュアルを索くことなく、操作したいことが操作したいときにすぐに分かるようでなければなりません。
またSIMPLEHOSTでは、コマンドが画面に表示されています。さらに、コマンドの実行結果が人にやさしい形で表示されています。先ほどアセンブラの項目で、さまざまな情報が出力されることを説明しました。これだけの情報を一瞬に理解するのは大変なことです。でもこれをSIMPLEHOSTは可能にしてくれます。

◆SIMPLEHOSTの実際

SIMPLEHOSTは5つのウィンドウで構成されます。ListingとMemoryとTriggerとTraceとPPGです。これらの内の基本となるウィンドウがListingです。
Listingでは、プログラムの修正がソース・イメージ行なえます（CLICEでは、ダンプリストを修正していました）。プログラムの実行も（を指令、実際に実行するのはIE-17KですけどSIMPLEHOSTを使っていると見えない）、Listingの仕事です。Listingでは、プログラム・カウンタの位置が手の形のアイコンを使って、ソース・イメージと一緒に表示されています。また、プログラムの構造を表示するのも、任意の部分を省略して表示するのも、プログラム・メモリのカバレージ状態を表示するのも、アドレス条件を設定するのも、Listingの仕事です。

Memoryの仕事は、データ・メモリの内容を表示することです。データ・メモリだけではありません。フラグやレジスタ・ファイルの内容も表示します。でも表示するだけではありません。任意のアドレスの値を書き換えることもできます。また、データ・メモリのカバレージ状態を表示することもできます。
Memoryで、データ・メモリの値を書き換え、Listingでプログラムを実行すれば、またその実行結果がMemoryに表示されます。

Triggerの仕事は、ブレーク条件、トレース条件を設定することです。メモリの書き込み動作による条件や、スタック・ポインタのレベルの範囲による条件、モニタ・アドレス・レジスタで指定されたメモリの内容による条件、プローブ・ターミナルの検出レベルによる条件、割り込みサイクルの発生による条件、DMAサイクル発生による条件、カウンタの使用／不使用、トレースのオン／オフ、ブレークのオン／オフ、など実にたくさんの条件を設定できます。これらの条件をMS-Windowsのインタフェースを用い対話的に設定できるのです。Triggerでブレーク条件を設定し、Listingでプログラムを実行すれば、ブレーク条件が成立したところで自動的に停止します。

Traceの仕事は、トレース項目の設定、トレース結果の表示です。あらかじめ用意されているトレース項目を組み合わせてトレース新しいトレース項目を作ったり、いっぱい出力されたトレース結果を見やすいように絞り込んだりするのがTraceの仕事です。Triggerでトレース条件を設定していれば、その範囲内だけでトレースさせることができますし、Memoryでデータを書き換えていればその結果を反映します。そしてListingで実行をすると、実際にトレース結果を保存し（実際に保存するのは、IE-17Kのトレース・メモリ）、表示します。表示されているトレース結果をマウスで指定すれば、Listingの該当部分を反転表示します。

もう1つのPPGの仕事は、パルス・パターンの編集と出力です。

◇コラム：開発者の憂鬱

シーン1
★「◇君、あのプログラムできた？」
◇「あそこのルーチンができれば完成なんですが・・・」
★「で、何パーセントできたの？」
◇「何パーセントといわれても・・・。あとルーチン1つだけですけど・・・」
★「ぢゃあ、ドキュメントは？」
◇「プログラムができていないのにドキュメントなんてできてません！」

シーン2
◇「▲君、あのルーチンできた？」
▲「まだです。」
◇「いつごろできるの？」
▲「わかりません。」
◇「どこまでできたの？」
▲「ぼくぶんけいなんですよ！」

という会話が実際にあったかどうかは知りませんが（だぶんなかったろうなァ、いくらなんでもこんなこと）、でもシーン1の方は割と実感のある話でしょ？ まぁ冗談はともかく、これから先、こんなことがない、とはだれもいいきれません。
実際、ソフトウエアの技術者が2000年には何万人足りない、という予測の話を良く聞きます。いままで一度もプログラムを組んだことがない人が、1つの製品の制御関連のすべてを任されることもありうるわけです。
人材の不足はベテランの技術者にも影響を与えます。未経験者への指導に時間を割かれ、自分の割り当てのプログラムを記述する時間にも事欠くようになり、それらのしわ寄せはドキュメントなどのプログラム以外の付属物に来ます。

さらに、こんなことをいうのは、今プログラムを組んでいる人に対して失礼かも知れませんが、技術者が足らなくなれば、他産業から人を補充しなければなりません。これから、実際に仕事をしながら、プログラミングの勉強をしなければならない人も現場に投入されてくるわけです。すべての製品の品質は人材によって決定されます。プログラマの質の低下はプログラムの質の低下に現れます。

こんなときに「がんばってネ」といっているだけのメーカの製品を買う人はいません。というわけで、17Kシリーズでは、これらの問題に対していろいろなサポートをしているそうです。まず、不本意ながらプログラムを記述しなければならなくなった新人さんのための、プログラムの質という問題に対しては、①簡単なツールを提供する、②質そのものを向上できるようにする、などのことのようです。
①の『簡単なツールを提供する』はおそらく、SIMPLEHOSTのことでしょう。学習しなくても使える、使いながら学習できるツールのようですから。
②の『質そのものを向上できるようにする』はたぶん、アセンブラのことだと推測されます。一部のマニアのための『自由な記述が可能な言語』ではなく、だれでも同じ様なプログラムが記述できる『文法が極めて厳格な言語』を採用しているそうですから。

では、不本意ながらしわ寄せを受けなおかつ黙々とドキュメントを記述しているベテランのための、ドキュメントの記述時間という問題に対しては、Computer Asisted Documentation という考え方をしているみたいです。日本語で記述されたソースから、ドキュメントを自動抽出し、プログラムの完成日がドキュメントの完成記念日、というようなことができるみたいですから。

いずれにしても、これからプログラムを記述しなければならない開発者、プログラマにとっては、大変なことです。ですから17Kシリーズを使って、少しでも楽をしましょう。人が楽をするだけのためにコンピュータは存在するわけですから。

○コラム：プログラムの互換性

環境が違うプログラムに互換性はない、というのがコンピュータ業界の通例となっています。でも、このプログラムの互換性の問題は、立場によって、視点によって、その問題のあるべき点が大幅に違います。

よく世間の話題に上るのは、パソコンのプログラムに互換性がない、機種が違うと同じことをするプログラムが動かない、といったことです。
これはパソコンのプログラムが、パソコンの限られた資源で要求されたすべての仕様を実現し、他製品との差別化を図ろうとしていることに原因があります。この結果、ハードウエア性能のギリギリまで使用し、他のハードウエアでは対応できない、ということになります。また、すべてのパソコンに共通のプログラミング規約を作っても、性能の低いパソコンに規約をあわせることになり、性能の低いプログラムしかでてこなくなる可能性もあります。

これは、基本的にはプログラムを使用するユーザの視点です。つまりパソコンを使っているユーザがプログラマに対して、VTRのビデオソフトのように、どの機械（コンピュータ）でも同じように動くプログラムを作れ、といった要求をしているのです。この要求に対応するためには、どのコンピュータでも同じオブジェクト・プログラムを使う、つまりプログラムの互換性をオブジェクト・レベルでとらなければなりません。しかし、これは現在のパソコンの性能、環境では難しい話です。

これが、ワークステーションのプログラムとなると少し事情が違ってきます。ワークステーションでは、その性能を惜しげもなく使うことができます。そして、もともとワークステーションはプログラムの開発環境という位置付けであったために、ソース・プログラムでの互換性がとれればいい、といった考え方が主流です。つまり、互換性はソース・プログラムでとり、機種依存、プロセッサ依存の部分はコンパイラで吸収する、といった考え方です。

> プログラムを開発できる人がワークステーションを管理しているはずですから、ソース・プログラムをコンパイルすることぐらいすぐしてくれます。パソコンではこの様にはいきません。すべてのユーザがプログラムを開発できる環境にあるとはいえないからです（特に最近ではプログラミングをしないユーザが大半を占めています）。

このように、ワーク・ステーションでは、高度なハードウエア資源とプログラムの開発環境により、ソース・プログラムで互換性がとれ、互換性の問題は（コンピュータの中では）少ない方です（ちなみに、どの機械でも使える可能性が高いことを、可搬性（Portability）が高い、といいます）。

と、ここまでは汎用マイコンの話でした。
専用マイコンのプログラムの互換性の問題では、汎用マイコンのようにユーザの視点がでてくることはありません。専用マイコンのプログラムを使う人はそれをプログラムだとは意識していません。当然、機械とそれを動かすためのプログラムという分類はしませんから、プログラムの互換性という問題が使う人からでてくる訳がありません（たとえ間違って、他の機械で動かそうと思い立ってもROMに格納されたものをどうやって・・・）。

専用マイコンのプログラムの互換性の問題がでてくるのは、プログラムを作るプログラマからです。つまり、同じ仕様で、違う品種のプログラムをする時や、1つのルーチンを複数の品種で使う時に、以前に作ったプログラムを流用したり、一度作ったプログラムをライブラリとして使用したい、という当り前の要求があるのです。

この要求に応えるのは、並み大抵のことではないはずです。専用マイコンは、パソコン以上に、品種によって大幅にハードウエア資源が違います（違うから『専用』なのです）。ROMやRAMの構成が違うことは当り前です。同じ機能を実現するのにも、ハードウエアを使ったり、ソフトウエアで実現したりします。ハードウエア資源が大幅に違う以上、オブジェクト・レベルで互換性をとることは不可能です。
開発環境が揃っているといって、ワークステーションのようにソース・プログラムで互換性をとるにしても、プログラムによってハードウエアを呼び出すルーチンが必要だったり、機能をまるごとソフトウエアで実現するルーチンを記述しなければならない、などの大きな違いがあり、翻訳プログラムなどで品種による違いを吸収するのにも限度があります。

というわけで、今までは専用マイコンのプログラムの互換性は、品種に依存する部分はコーディングのし直し、プログラム構造のみを流用してコーディングのし直し、最悪の場合は基本設計のみを流用してすべてのコーディングの最初から、などということが結構ありました。では17Kシリーズでは・・・。

◇コラムの続き1：プログラムの互換性、17Kシリーズでは

現在のところ、17Kシリーズのプログラムもソース・プログラムでとるようになっています。基本的には、他の専用マイコンの互換性と同じ様な問題を抱えています。しかし、他の専用マイコンと比較すればプログラムの互換性に大きな気配りをしているといえます。その例をいくつかあげると、

デバイス・ファイル SPARELIB フラグの再配置の可能性

というキーワードがでてきます。デバイス・ファイルもSPARELIBもすでにでてきました。『フラグの再配置の可能性』もこのすぐあとのコラムででてきます。

では、まず『デバイス・ファイル』から。デバイス・ファイルがでてきたのは、アセンブラの項目でした。17KシリーズのアセンブラのAS17Kシリーズの説明のところで、『個々の品種に依存した部分を集めたデバイス・ファイル』という説明をしました。新しい品種ができてもすぐにアセンブラが手に入るというメリットを強調しました。AS17Kシリーズ+デバイス・ファイルで、個々の品種に対応したオブジェクト・ファイルを生成するという意味ですが、これはプログラムの互換性にも大きく関わってきます。
デバイス・ファイルは、個々の品種に依存した部分を集めたもの、ですがこれだけでは何も説明していないに等しい説明です。個々の品種に依存した部分とはどこを指すのでしょうか？ まず、プログラム・メモリ（ROM）やデータ・メモリ（RAM）のサイズが挙げられます。内蔵された周辺ハードウエアも、これを制御するための特別の命令も、これを制御するためのレジスタの位置も個々の品種に依存します。

たとえば、新しい周辺ハードウエアが内蔵された品種が追加されたとします。これを制御するためのレジスタは、コントロール・レジスタに配置されますが、それぞれのレジスタには名前が付いています。この名前が定義されているのも、デバイス・ファイルです。品種が違っても同じ機能のレジスタには同じ名前が付いています。プログラムからこの名前でレジスタを使用する限り、たとえ品種が違って位置が変わっていようとも、プログラムの互換性は、ソース・レベルでとれていることになります。名前を実際のアドレスに変換するのは、アセンブラの仕事ですから。
このようにデバイス・ファイルは、プログラムの多品種への展開をサポートすることもできます。

では、品種が違って今まであった周辺ハードウエアがなくなり、プログラムで実現しなければならなくなったら、プログラムの互換性をどうやってとるか? この問題の解決のために活躍するのが『SPARELIB』です。SPARELIBがでてきたのはプログラム・メモリの説明のところでした。その説明の中に。このライブラリに格納されるサブルーチンとして『上位品種ではハードウエアで実現されている機能をプログラムで実現させるためのサブルーチン』という記述があります。
もうお分かりですね。上位品種では、ハードウエアって実現し、結果をデータ・メモリに格納します。下位品種では、このデータ・メモリとSPARELIBに格納されたルーチンを使用して、結果をこのデータ・メモリに格納します。こうしてプログラムからは結果が格納されたこのデータ・メモリだけをアクセスするようにすれば、違う品種でも同じプログラムで同じ結果を得ることができます。
ただし、これにはいくつかの条件があり、SPARELIBがサポートされ、そのサポートされたSPARELIBに下位品種でなくなったハードウエアをソフトウエアで実現するサブルーチンが格納されていることです。
このようにSPARELIBは、品種間の周辺ハードウエアの有無を吸収することもできます。

◇コラムの続き2：プログラムの互換性とフラグの再配置

という訳でやっと後回しにされた『フラグの再配置の可能性』です。
フラグというのは、1ビットであるかないかを指定するだけのものです。でも他のマイコンと比べると、4ビットではこのフラグの判定が多いのです。アドレスは4ビット単位でした。4ビット単位の方が効率がいいからです。そういった中で、1ビット単位のフラグのセット(1)、リセット(0)が簡単にできなければなりません。また17Kシリーズでは、コントロール・レジスタとデータ・メモリのアドレス空間は別の位置にあり、コントロール・レジスタとデータのやり取りをする場合は、専用の命令(PEEK／POKE命令)を使用しなければなりません。そういった中でも、コントロール・レジスタとデータ・メモリの区別なくフラグの操作ができた方がいいに決っています。

複数のフラグを操作する場合に、同一のアドレスに定義してあるか、異なるアドレスに定義してあるかで、記述する命令の数が変わってくるのです。
また、1つのフラグを操作する場合でも、そのフラグがコントロール・レジスタにあるか、データ・メモリにあるかで、今度は記述する命令の内容まで変わってきます。

対象の位置で記述するプログラムを変えるような行為は、プログラムの作成効率を落すだけでなく、バグの原因ともなります。また、当然このような記述をしたプログラムでは、この項目のテーマであるプログラムの互換性を損なう結果となるのは明白です。フラグの定義位置が違うプログラムの間で互換性がとれなくなります。さらにいえば、同じプログラムの中でもフラグの定義位置を変更すれば、プログラムはそのままでは動かなくなり命令の記述の変更が必要になる可能性があるのです。

このように、たとえフラグといえども、その定義位置に依存したプログラムを記述することは避けなければなりません。フラグの定義位置が変わる可能性が、つまり『フラグが再配置』される可能性があるからです。そこで、17Kシリーズではいくつかの手段を用意しています。フラグ操作組み込みマクロ命令とフラグ型のシンボルです。

組み込みマクロ命令を使用することにより、フラグの定義してあるアドレスに関係なく、1つの組み込みマクロ命令で、複数のフラグを操作することができます。フラグが同一アドレスに定義してあれば1つの命令を展開しますし、異なるアドレスに定義してあるならば、フラグの数だけ命令を展開します。つまり、必ず最も短いオブジェクトを生成する訳です。

組み込みマクロ命令には次のようなものがあります。SETn（フラグのセット）、CLRn（フラグのクリア）、SKTn／SKFn（フラグのテスト）、NOTn（フラグの反転）、INITFLG（4ビット単位のフラグの初期化）

フラグ型のシンボルを使用することにより、ビット位置と共にそのフラグがデータ・メモリに位置するのかコントロール・レジスタに位置するのかといった情報が定義されます。このシンボル（とフラグ操作組み込みマクロ命令）を介してフラグをアクセスすれば、データ・メモリだろうがコントロール・レジスタだろうが関係なくアクセスするプログラムを記述することができます。

このように『フラグの配置の可能性』、つまりフラグ型のシンボルと組み込みマクロ命令は、ユーザ・ライブラリの汎用性を保証することもできます。

17K series architecture

第8章

付録

●周辺ハードウエア

17Kシリーズは、たくさんの周辺ハードウエアが内蔵できることが特徴の1つでした。17Kシリーズが内蔵できる周辺には次のようなものがあります。ここではそれぞれについてごく簡単にこれらの説明をしてみたいと思います。ただし、ここまで本書を読んで17Kシリーズを理解してくれた方なら分かると思いますが、すべての品種にすべての周辺が乗っているとは限りません。また各種スペックも品種によって違う場合がありますので、詳細は各品種のデータ・シート等をご覧ください。

A/Dコンバータ、D/Aコンバータ、クロック・ジェネレータ・ポート、
シリアル・インタフェース、周波数カウンタ、LCDコントローラ/ドライバ、
キー・ソース・コントローラ/デコーダ、PLL周波数シンセサイザ、タイマ

◆A/Dコンバータ

A/Dコンバータは、外部のアナログ電圧をディジタル信号として取り込むことができます。コントロール・レジスタで入力端子を選択し、選択された端子の電圧と比較電圧生成ブロック(Rストリング方式D/Aコンバータ)の電圧とをソフトウエアで比較することにより逐次比較方式のD/Aコンバータとして使用できます。

◆D/Aコンバータ

D/Aコンバータは、ディジタル信号をアナログ信号として使用することができます。17KシリーズのD/Aコンバータは、クロック生成ブロックとデューティ設定ブロックから構成され、これと外部のLPF(Low Pass Filter)を接続することによりデューティが可変(PWM方式)なD/Aコンバータとして使用できます。

◆クロック・ジェネレータ・ポート

クロック・ジェネレータ・ポートは、デューティ可変信号か周波数可変信号のどちらかを出力できます。VDP/SG設定ブロックにより、デューティ、周波数の設定と出力信号の設定を行ないます。

◆シリアル・インタフェース

シリアル・インタフェースは、外部との8ビット単位のシリアル・データ転送ができます。シリアル・インタフェースは2系統で構成され、同時に使用することができます。シリアル・インタフェース1は、2線式と3線式を選択できますが、シリアル・インタフェース2は3線式のみです。また通信方式として、2線式ではシリアル・バス方式を選択することもできますが、3線式はシリアルI/O方式のみです。

◆周波数カウンタ

周波数カウンタは、チューナの中間周波数（IF ; Intermediate Frequency）のカウントや外部信号のパルス幅の検出ができます。周波数のカウントをするか、パルス幅の検出をするか、（もしくは汎用ポートとして使用するか）の選択、周波数カウントのスタート／ストップの制御などをコントロール・レジスタで行ないます。

◆LCDコントローラ／ドライバ

LCDコントローラ／ドライバは、ドット単位やセグメント単位のLCD表示を行なうことができます。LCDコントローラ／ドライバは、コモン信号出力タイミング制御、セグメント信号／キー・ソース信号出力タイミング制御、キー・ソース信号出力制御ブロックなどから構成されています。

◆キー・ソース・コントローラ／デコーダ

キー・ソース・コントローラ／デコーダは、キー・ソース信号出力端子とキー入力端子のマトリクスによりキー・マトリクスを構成できます。

◆PLL周波数シンセサイザ

PLL（Phase Locked Loop）周波数シンセサイザは、MF、HF、VHF帯の周波数を位相差比較方式により一定周波数にロックすることができます。

17KシリーズのPLL周波数シンセサイザは次のブロックから構成されています。

PLL周波数シンセサイザ用ハードウエア、デュアル・モジュラス・プリスケーラ（最大250MHz）、プログラマブル・ディバイダ、位相比較器、チャージ・ポンプ、LPF用アンプ

これらのブロックと、外部のLPF（Low Pass Filter）と電源制御発信機（VCO）を接続することによりPLL周波数シンセサイザとして使用できます。

◆タイマ

タイマ機能は、正確な時間管理が必要なプログラムで使用します。

一定時間ごとにセットされる『タイマ・キャリー・フリップ・フロップ』と一定時間ごとに割り込みをかける『タイマ割り込み』があります。それぞれの時間間隔はコントロール・レジスタにより独立して設定できますので、2系統の時間管理ができます。

記号等の意味
- G 一般に良く使う用語です。
- K 17Kシリーズにおいて固有名詞、もしくは特殊な意味を持つ用語です。
- 参 関連項目があります。参照してください。

A-Z

◆**ALU(Arithmetic Logic Unit)【エーエルユー】** G算術演算、論理演算、ビット判断、比較判断、回転処理などを行なうための機能ブロック。コンピュータの最も計算機らしい部分。[参PSW]

◆**AS17K【えーえすいちななけい】** K17Kシリーズ用のアセンブラである。個々の品種に依存した部分をデバイスファイルとすることで、17Kシリーズのすべての品種をこれでアセンブルできる。

◆**ASC(Application Specific Controller)【エーエスシー、特定用途向けコントローラ】** Gシングルチップ・マイコンの周辺ハードウェアをいくつかの機能ブロックに分け、インテリジェント化したASICに付けられた名称。

◆**ASIC(Application Specific IC)【エーシック、特定用途向けIC】** G使用目的や使用者を特定したLSI。使用目的を特定することにより特殊な機能を盛り込むことができる。[参LSI]

◆**CISC(Complex Instruction Set Computer)【シスク】** Gコンピュータのアーキテクチャを示す用語。RISCとの対応において従来からあるタイプのCPUを総称してCISCと呼ぶ。[参RISC、アーキテクチャ]

◆**CLICE(Command Language for In-Circuit Emulator)【クライス】** KIE-17Kを操作するためのコマンド群とそのコマンドをコントロールするための言語を総称する言葉。インタプリタ。オペランドの指定は逆ポーランド表記。[参IE-17K]

◆**CPU(Cental Processing Unit)【シーピーユー、中央処理装置】** Gコンピュータの制御装置と演算装置をまとめたものをCPUという。CPUが1つのチップにのったものをマイクロ・プロセッサという。

◆**FIFO(First In First Out)【ファイフォ、先入れ先出し】** Gデータの記憶方式の1つ。新しいデータから最初に使っていく方式。[参LIFO]

◆**IC(Integrated Circuit)【アイシー、集積回路】** G数ミリ角のチップ上にトランジスタやダイオードなどの部品を多数集めたもの。コンピュータを小型化するための必需品。[参LSI]

◆**In-Circuit Emulator【インサーキット・エミュレータ】** Gマイコンの開発支援装置の1つ。ソフトウエアのディバッグを実際のハードウエア上で実行することができる。

◆**IE-17K【あいいーいちななけい】** K17Kシリーズ用のインサーキット・エミュレータである。本チップを2個使ったMam'Chipやインサーキット・エミュレータ用の言語CLICEが特徴。[参Mam'Chip、CLICE]

◆**I/O(Input and Output)【アイオー、入出力】** G情報の入力と出力を総称していう言葉。コンピュータの基本である。入力装置としてキーボード、出力装置としてディスプレイが有名。

◆**LCD(Liqid Crystal Display)【エルシーディー】** G液体でありながら結晶のような特性をもった物質を液体結晶というがこの特性を応用した表示装置。CRTディスプレイに比べ消費電力が少ない、小型化が可能などの特徴を持つ。

◆**LIFO(Last In First Out)【ライフォ、後入れ先出し】** Gデータの記憶方式の1つ。古いデータから最初に使っていく方式。スタックのこと。スタックに格納することをPUSH、スタックから取り出すことをPOPという。[参FIFO]

◆**Listing【リスティング】** KSIMPLEHOSTのウィンドウの1つ。ソースイメージを表示し、プログラムの実行を制御するためのウィンドウ。[参SIMPLEHOST]

◆**LSI(Large Scale Ic)【エルエスアイ、大規模集積回路】** GICの一種。一般に1つのチップ上に1000個以上の素子を組み込んだものを指す。100個以下だとSSI、100個以上1000個以下だとMSI、10万個以上だとVLSIという場合がある。

◆**Mam'Chip【マムチップ】** K17Kシリーズのエミュレーション方式。またはそのために使用するチップのこと。本チップを2個とこのMam'Chipを使って1つのチップをエミュレーションする。

◆**Memory【メモリ】** KSIMPLEHOSTのウィンドウの1つ。データ・メモリの内容を表示し、データ・メモリの内容を変更するためのウィンドウ。[参SIMPLEHOST]

◆**PC(Program Counter)【ピーシー】** Gプログラムの実行制御を専門とするカウンタ。次に実行する命令のあるアドレスが格納されている。1回命令が実行されるとPCの値も1増加する。

◆**PPG(Programable Pattern Generator)【ピーピージー】** KSIMPLEHOSTのウィンドウの1つ。パルス・パターンの編集と出力を行なう。[参SIMPLEHOST]

◆PSW(Program Status Word)【ピーエスダブリュ】GALUによる演算の結果を示すフラグとALUの実行条件を指定するためのフラグで構成される。プログラムの実行状況を示すための一連のビット。[関ALU]

◆RAM(Random Access Memory)【ラム、随時読み出し、書き込み可能記憶装置】Gコンピュータの記憶装置で情報の変更できるものをいう。SRAM、DRAMなどの種類がある。[関メモリ、ROM]

◆RISC(Reduces Instruction Set Computer)【リスク】Gコンピュータのアーキテクチャを示す用語。命令の数を減らすことで高速化を図ったCPUのこと。[関CISC、アーキテクチャ]

◆ROM(Read Only Memory)【ロム、読み出し専用記憶装置】Gコンピュータの記憶装置で情報の変更できないものをいう。ROMに情報を書き込むためには特別な装置がいる。EPROMやEEPROMなどがある。[関メモリ、RAM]

◆SIMPLEHOST(Simple & High level Operation support tools)【シンプルホスト】17 17Kシリーズ用のディバッガ・インタフェース・ソフトウェアである。パソコンのMS-Windows上で動作し、高水準のマン-マシン・インタフェースを提供するとともに、視覚的にディバッグを行なうことができる。

◆SPARELIB(Specific Application REsident LIBrary)【スペアリブ】17特定応用分野のためのサブルーチンなどが格納されたライブラリ。システム・セグメントのページ0にあることになっている。

◆Trace【トレース】17SIMPLEHOSTのウィンドウの1つ。プログラムの実行結果を保存し、時間を基準に表示する。[関SIMPLEHOST]

◆Trigger【トリガー】17SIMPLEHOSTのウィンドウの1つ。ブレーク、トレース条件を設定するためのウィンドウ。[関SIMPLEHOST]

あ行

◆アーキテクチャー(Architecture)Gコンピュータシステムの属性。つまりシステムや命令体系、アドレス方式、データ形式などをいう。建築の設計がアーキテクチャであるのと同じように、コンピュータの設計もアーキテクチャである。

◆アキュムレータ(Accumulator)Gコンピュータの演算装置にあり、累算ができるレジスタを指す。ちなみにコンピュータの演算、転送の方式でオペランドの片方をアキュムレータに置いてから演算する方式をアキュムレータ方式という。

◆アドレシング(Adressing)Gアドレスの指定方式をいう。直接アドレシング、間接アドレシング、インデクス・アドレシングなどがある。

◆アドレス(Address)G番地と訳される。コンピュータの記憶装置を制御するために付けられた位置情報のことである。

◆アドレス・スタック・レジスタ(Address Stack Register)17アドレス・データを格納するためのスタック構造のレジスタ。プログラムの分岐が起こると、このレジスタにプログラム・アドレスが格納される。[関スタック・ポインタ]

◆アドレス・レジスタ(Address Register)Gプログラム・メモリのアドレスを格納するためのレジスタシステム・レジスタにある。

◆インストラクション(Instruction、命令)GCPUに実行させる命令をインストラクションという。PCには次に実行するインストラクションのあるアドレスが格納されている。

◆インタフェース(Interface)G情報のやり取りのことをいう。機械と機械の間の情報のやり取りもインタフェースだが、機械と人間の情報のやり取りもインタフェースである。[関SIMPLEHOST]

◆インデクス・レジスタ(Index Register)17データ・メモリをORインデクス修飾するためのレジスタ。システム・レジスタにあり、一部がメモリ・ポインタと重なっている。[関メモリ・ポインタ]

◆ウィンドウ・レジスタ(Window Register)17コントロール・レジスタとのデータのやり取りを行なうためのレジスタ。コントロール・レジスタを操作するためには、ウィンドウ・レジスタにデータを置いて、PEEK／POKE命令を使用する。システム・レジスタにある。[関コントロール・レジスタ]

◆エミュレータ(Emulator)Gあるシステム上で別のシステムを模倣することをエミュレーションという。このエミュレーションをするための装置がエミュレータである。[関インサーキット・エミュレータ]

か行

◆高水準言語(High Level Language)Gコンピュータ言語において高水準言語とは、より人間が理解しやすい言語である。これに対して、コンピュータが理解しやすい言語を低水準言語という。

◆コーディング(Coding)Gプログラムの設計図である仕様書にしたがってプログラミング言語を使用してプログラムを記述することをコーディングという。

◆コンピュータ(Computer)Gデータを入力すると、プログラムにしたがって演算を行ない、データを出力する機械をコンピュータという。コンピュータは、入出力装置、演算装置、制御装置、記憶装置で構成さ

れている。

◆コントロール・レジスタ(Control Register)[G][K]一般的に周辺ハードウエア制御用レジスタを指す。17Kシリーズではデータ・メモリと物理的に別空間にあるレジスタ・ファイルの一部のこと。[参レジスタ・ファイル]

さ行

◆サブルーチン(Subroutine)[G]プログラム中の同じ部分をまとめて、1つにしたものをサブルーチンという。サブルーチンを使用することで、プログラム全体が簡潔になる。[参マクロ]

◆システム・レジスタ(System Register)[G][K]一般的にシステムを制御するためのレジスタを指す。17Kシリーズではデータ・メモリの74H～7FHにあるレジスタ群のこと。

◆周辺ハードウェア(Hardweare Periferal)[G]専用マイコンを専用マイコンたらしめんとする回路のこと。1チップマイコンでは、CPUと同じチップ上に存在する。

◆シングルチップ・マイコン(Single Chip Micro Computer)[G]一つのチップ上にコンピュータに必要な機能(制御装置、演算装置、入出力装置、記憶装置)を盛り込んだもの。制御等によく使われる。[参マルチチップ・マイコン]

◆スタック・ポインタ(Stack Pointer)[G]スタックにデータを記憶していったとき、次に取り出すデータを記憶しているのが、スタック・ポインタである。[参LIFO]

◆ソフトウエア(SoftWare)[G]もともとは、コンピュータを動かすためのプログラムを指したのだが、現在ではビデオの内容もレコードの内容も何かをするためのテクニックも目で見えないものはすべてソフトウェアである。[参ハードウエア、プログラム]

た行

◆チップ(Chip)[G]トランジスタやダイオードなどが取り付けられた半導体の1塊をチップという。一つのチップ上にコンピュータに必要な機能を盛り込めば、シングルチップ・マイコン。[参シングルチップ・マイコン]

◆ディバッグ(Debug)[G]コンピュータのプログラム上の誤りをバグというが、このバグを取り除く作業をディバッグという。コーディング上の誤りを発見するのはたやすいが、構造上設計上の誤りを取り除くのは至難の技である。

◆データ・バッファ(Data Buffer)[K]17Kシリーズでは、バンク0の08H～0FHにあるバッファのこと。周辺ハードウェアとのデータのやり取りができる。

◆データ・メモリ(Data Memory)[K]プログラム中で変更することが予想されるデータが格納されるメモリ。したがってRAMで構成されている。[参RAM]

◆トレース(Trace)[G]プログラムを追跡することをトレースするという、ディバッグにおいては、任意の区間を追跡することで、プログラムが流れに間違いがないかを発見することが可能である。インサーキット・エミュレータには必須の機能である。

な行、は行

◆ニブル(Nibble)[G]4ビット・マイコンの基本的なデータの記憶単位。1ニブルは4ビットである。4ビットで表わせる情報は1～16まで。[参ビット]

◆バイト(Byte)[G]コンピュータの基本的な情報の単位。1バイトは8ビットである。8ビットで表わせる情報は1～255まで。[参ビット]

◆配列(Array)[G]一般には同じ大きさの同じ型の変数の集まりをいう。配列の特徴は要素に付けた番号で変数をアクセスできることである。

◆バス(Buss)[G]コンピュータの内部での信号の伝送路をバスという。信号の種類によりアドレス・バス、データ・バス、コントロール・バス、インストラクション・バスなどがある。[参コンピュータ]

◆ハードウエア(HardWare)[G]もともとは金物という意味だが、現在ではソフトウエアと対で使い目で見える装置、ぐらいの意味で使用する。[参ソフトウエア]

◆バッファ(Buffer)[G]緩衝用記憶装置の意味。処理速度が違う2つの装置の間でデータの漏れなどが起こらないようにする装置。

◆バンク・レジスタ(Bank Register)[K]17Kシリーズではデータ・メモリを最大16のバンクに分割して使用できる。バンク・レジスタは現在どのバンクを参照しているかを示すレジスタ。システム・レジスタにある。[参システム・レジスタ]

◆汎用レジスタ(General Purpose register)[G][K]一般には特別な用途を持たないレジスタを指す。コンピュータの演算、転送の方式でこの汎用レジスタとメモリの間で演算をする方式を汎用レジスタ方式という。

◆ビット(Bit)[G]コンピュータの一番基本的な情報の処理単位。もともとはBinaly Digitの略語である。1ビットで表わせる情報は0か1のどちらかである。[参バイト、ニブル]

◆ブレーク(Break)[G]プログラムを途中で止めることをブレークするという。ディバッグにおいては、任意の位置でプログラムを止めることで、システムの

状態を検査することができる。インサーキット・エミュレータには必須の機能である。

◆プログラマ(Programmer)[G]プログラムを作る人をプログラマという。西暦2000年には何万人の単位で不足するといわれている。誰でもプログラムを作ることはできるが、売り物になるプログラムを作ることは難しい。

◆プログラム(Program)[G]コンピュータを動かすための手順を記述したものをプログラムという。一般に専用マイコンのプログラムはROMに格納するが、汎用マイコンのプログラムはRAMに格納することが多い。[関ソフトウエア]

◆プログラム・メモリ(Program Memory)[17]プログラムが格納されるメモリ。専用マイコンはほとんどの場合ROMで構成されている。ここに固定データを格納する場合もある。[関マイコン、ROM]

◆ポート・レジスタ(Port Register)[G][17]一般的に外部端子に接続されたレジスタの総称。17Kシリーズでは、データ・メモリの70H～73Hにあるレジスタのこと。

ま行

◆マイクロプロセッサ(Micro Processer)[G]CPUを1つのLSIで構成したものマイクロプロセッサという。マイクロプロセッサが開発されたのは、1971年のこと。[関CPU]

◆マイコン(Micro Computer)[G]現在ではマイクロ・プロセッサに記憶装置と入出力装置を接続して、コンピュータを構成したものをマイコンといい、マイクロ・コンピュータの略語である。機器に組み込むための専用コンピュータを指すことが多い。[関コンピュータ]

◆マクロ(Macro)[G]巨視的という意味で、ある処理をまとめて見やすくしたものをいう。見やすく、理解しやすくすることが目的なので実体に変わりはない。[関サブルーチン]

◆マルチチップ(Multi-Chip)[G]シングルチップとの対応において記憶装置を同じチップ上に組み込まないで外部に接続して使うマイコンを指す。[関シングルチップ・マイコン]

◆メモリ(Memory)[G]コンピュータの記憶装置をメモリという。このメモリの内で情報の書き替えができるものをRAM、情報の書き替えができないものをROMという。[関ROM、RAM]

◆メモリ・ポインタ(Memory Pointer)[17]汎用レジスタ間接アドレシングする際のバンクとロウ・アドレスを格納したレジスタ。システム・レジスタにあり、一部がインデクス・レジスタと重なっている。[関インデクス・レジスタ]

や行、ら行、わ行

◆レジスタ(Register)[G]命令やデータを一時的に記憶しておくための装置をいう。アキュムレータ、PCもレジスタの一種である。[関アキュムレータ、PC]

◆レジスタ・ファイル(Register File)[G][17]一般的にCPU周辺のハードウエアの制御を行なうためのレジスタを指す。17Kシリーズではデータ・メモリの40H～7FHとコントロール・レジスタのこと。[関データ・メモリ、コントロール・レジスタ]

◆レジスタ・ポインタ(Register Pointer)[17]17Kシリーズでは汎用レジスタのある位置を示すためのポインタ。システム・レジスタにある。

◆割り込み(Interrupt)[G]現在行なっている処理を中断して、別の処理を行なうことを割り込みという。別の処理から戻ったら、元の処理があたかも何もなかったように動いていなければならない。

記号等の意味
囲 用語解説にも簡単な説明があります。

A～Z、他

μPD1700シリーズ 2
4ビット 10
A/Dコンバータ 43、47、90
ADTS 4
ADコンバータ 25
ALU 25、29、囲
AR 28、41
AS17K 3、72、囲
ASC 24、25、囲
ASIC 3、24、囲
BANK 41
BCD 31
CDのコントロール 4
CE端子 43
CGP 43、47
CISC 27、囲
CLICE 82、囲
CMP 31
CPU 12、囲
CPUコア 25
CPUペリフェラル 25
CY 31
D/Aコンバータ 43、47、90
DBF 48
DI 68
DMA 45
DMAEN 44
DMA許可フラグ 44
DTS 1、4、8
EI 68
FIFO 22、囲
FM放送受信用 4
GET 48
HEXファイル 78
I/O空間 43、49
IDC 5
IE-17K 3、78、囲
IEGxxx 70
INC 29
INTE 68
INTxxx 44、69
IPxxx 44、68、69
IRQxxx 44、67、69
IX 56
IXE 31
IXH 41
IXL 41
IXM 41
LCDコントローラ/ドライバ 43、47、91
LCDドライバ 4、5
LCDに表示するデータ 49
LIFO 22、囲
Listing 84、囲
LSI 24、囲
Mam'Chip 3、80、囲
MappedI/O 42
Memory 84、囲
MOVT 49
MP 58
MPE 41、59
MPH 41
MPL 41
MS-Windows 3
OR演算 56
PbpBIO 44
PbpGIO 44
PEEK 44
PLL回路 2
PLL周波数シンセサイザ 43、47、91
PLL周波数シンセサイザの分周値 49
PLLシンセサイザ 4
POKE 44
PPG 84、囲
PROMファイル 78
PSW 30、41、囲
PUT 48
Portability 86
RAM 16、25、囲
Readability 26
RF 43
RISC 3、23、囲
ROM 16、25、囲
RP 51
RPH 41
RPL 41
RTI 68
SEボード 82
SIMPLEHOST 3、83、囲
SP 44

SPARELIB 37、87、88、囲
SRAM内蔵 5
SYSCAL 37、66
TK-80 1
Trace 84、囲
Trigger 84、囲
VAG 70
WR 41、43
Z 31

あ行

アキュムレータ方式 20、囲
アセンブルのリポート 78
アセンブル・リスト 78
アドレス 16、囲
アドレス・スタック・レジスタ 35、囲
アドレス・バス 19
アドレス・レジスタ 28、41、47、囲
アブソリュート 73
アプリケーション・ペリフェラル 25
あるかないか 11
インサーキット・エミュレータ 3、78
インストラクション・デコーダ 20、25
インストラクション・バス 19、囲
インデクス・アドレシング 51
インデクス修飾 56
インデクス・レジスタ 41、56、囲
ウィンドウ・レジスタ 41、43、囲
エミュレータ 79、囲
演算装置 12
オンライン・マニュアル 5、32

か行

開発環境 2
学習リモコン 5
拡張性 2
拡張命令 23
家電製品 5
家電用コントローラ 8
可搬性 86
玩具 5
間接アドレシング 58
間接分岐 33
キャリア発生回路 5
記憶装置 12
疑似命令 77
キー・ソース・コントローラ/デコーダ 47、91
組み込み型 14
グループ単位I/Oポート 46
クロス開発 72
クロスレファレンス・リスト 78
クロック・ジェネレータ・ポート 43、47、90
継承性 2
高水準言語 17、囲
構造化 74
コストパフォーマンス 2
異なるバンクにある汎用レジスタと即値の演算 54
コメント 74
コントローラ 5、10
コントロール・バス 19
コントロール・レジスタ 43、囲

さ行

サブルーチン 21、囲
システム・コール機能 37
システム・サービス要求 37
システム・レジスタ 41、囲
システム割り込み 66
周波数カウンタ 43、47、91
周波数表示 4
周辺回路 42
周辺ハードウエア 47、囲
周辺ハードウエア制御専用の命令を設ける 42
周辺ハードウエア用の外部端子を作る 42
出力装置 12
シリアル・インタフェース 43、47、90
シングルチップ 10、12、囲
シンボル定義 76
炊飯器 5
スタック 43
スタック・ポインタ 36、44、囲
スーパーコンピュータ 14
スーパーバイザ・ボード 82
制御装置 12
制御命令 77
赤外線リモコン 8
赤外線リモコン受信プリアンプ 5
赤外線リモコン用 5
セグメント 33
専用ポート 45
専用マイコン 13
ソフトウエア 15、囲
ソフトウエア割り込み 66

た行

タイニ・コントローラ 8
タイマ 43、91
タイミング・ジェネレータ 25

直接分岐 33
中央処理装置 12
中間オブジェクト・ファイル 73
直接アドレシング 51
低水準言語 17
ディジタル化 1
ディバグ 80、囲
データ 16
データ・バス 19
データ・バッファ 45、47、囲
データ・メモリ 39、囲
データ・メモリ上にマッピングする 43
データ・メモリ同士で演算 54
データ・メモリ・ロウ・アドレス・ポインタ58、59
デバイス・ファイル 73、87
電子制御化 5
電子レンジ 5
転送命令 26
同一ロウ転送 61
ドキュメント 78
トグルI/Oポート 46

な行、は行

ななめ転送 60
ニブル 18、囲
ニブル・ポート 46
日本語 75
入力装置 12
ノンマスカブル割り込み 68
バイト 18、囲
バイト・ポート 46
パイプライン 19
配列 64、囲
パソコン 14
ハードウエア 15、囲
ハードウエア割り込み 66
ハーバート・アーキテクチャー 19
バンク 39
バンクを超えて演算 54
バンク・レジスタ 41、囲
汎用コンピュータ 14
汎用出力ポート 43、47
汎用ポート 45
汎用マイコン 13
汎用レジスタ 51、囲
汎用レジスタ・アドレシング 51
汎用レジスタ間接アドレシング 51
汎用レジスタ方式 20
ビット単位I/Oポート 46
ファームウエア 15
フラグの再配置の可能性 87、88
プリスケーラ 4
プリセット・メモリの制御 4
プログラム 16、囲
プログラム・カウンタ 19、25、27
プログラム・ステータス・ワード 30、41
プログラム・メモリ 31、囲
ページ 33
ポート・グループI/Oフラグ 44、47
ポート・ビットI/Oフラグ 44、47
ポート・レジスタ 46、囲

ま行

マイクロ 10
マイクロプロセッサ 2、14、囲
マクロ 21、74、囲
マクロアセンブラ 3
マスカブル割り込み 68
マップ・ファイル 78
マルチチップ 12、囲
未定義シンボルの一覧 78
ミニコン 14
民生用受信機 1
命令使用頻度統計 23
命令フォーマット 26
メインフレーム 14
メモリ空間 43
メモリ・ポインタ・イネーブル・フラグ 41
メモリ・ボード 82
メモリ・ロー・アドレス・ポインタ 41、囲

や行、ら行、わ行

読み易さ 26
ライブラリ 32
リモコン 5
リロケータブル 73
レジスタ 25、囲
レジスタ・ファイル 43、囲
レジスタ・ポインタ 41、51、52、囲
ロングワード・ポート 46
ワークステーション 14
ワード・ポート 46
割り込み 43、囲
割り込み許可フラグ 44
割り込み要求状態 44
割り込み要求フラグ 44

● 17K シリーズの仕様

命令サイクル	:	1 μs
プログラム(ROM)容量	:	(16ビット×64Kワード)
データRAM容量	:	(4ビット×1.8ニブル)
I／Oポート	:	4×64＋8×64 16×32＋32×32＝2304ビット
スタック・レベル	:	16レベル
割り込み要因数	:	無制限
多重割り込みレベル	:	16レベル（スタックによる制限）
最小端子数	:	18ピン
ノンマスカブル割り込み	:	1
テーブル参照	:	可能
間接分岐	:	全アドレス空間
直接分岐	:	セグメント内（8Kワード）
スタック操作	:	可能
アドレシング	;	直接アドレシング 汎用レジスタ・アドレシング インデクス修飾・アドレシング 汎用レジスタ間接アドレシング
周辺データ転送	:	8／16／32ビット並列転送
PROM化	:	可能
プログラム評価方式	:	セルフ・エミュレーション方式
OS／ライブラリ化	:	可能
CPUブロック化	:	可能
基本命令数	:	47命令
拡張命令数	:	3296命令

キリトリ

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］ 17Kシリーズ ユーザーズ・マニュアル アーキテクチャ編

（IEU-769（第1版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）（ ）
ご住所（ ）
お電話番号（ ）
お仕事の内容（ ）
お名前（ ）

1．ご評価（各欄に○をご記入ください）

項目	大変良い	良い	普通	悪い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（ ）					
（ ）					

2．わかりやすい所（第 章，第 章，第 章，第 章，その他 ）

理由［ ］

3．わかりにくい所（第 章，第 章，第 章，第 章，その他 ）

理由［ ］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは

NEC販売員，特約店販売員，NEC半導体ソリューション技術本部員，

その他（ ）

ご協力ありがとうございました。

下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

NEC半導体インフォメーションセンター
FAX：(044)548-7900