計算機アーキテクチャ 61－1
（1986. 3. 14）

TRONリアルタイムオペレーティングシステム
—アーキテクチャと今後の展望—

坂村 健
Ken Sakamura
（東京大学理学部・情報科学科）

1. はじめに

マイクロプロセッサが誕生したのは1970年代初めであるが、その進歩は著しく、応用分野は今後もますます広がるであろう。数多くのマイクロプロセッサが人間の生活空間の細部に入り込み、それらが相互に通信し、分散処理を行ないながら、より快適な生活を人間に提供するようになると思われる[1]。1990年代には50～100万トランジスタ以上の集積度を持つＶＬＳＩチップ出現すると予測されているが、その進んだ半導体技術を利用すれば、将来必要とされるマイクロプロセッサの処理能力に対応することは十分可能である。

しかしながら、現在のマイクロプロセッサは、まだ応用からの要求にとって十分な能力を提供しているとは言い難い。つまり、現在のマイクロプロセッサは、ハードウエア資源の乏しかった頃に作られたアーキテクチャをそのまま引きずっており、多くの部分で歪みを生じているということである。このまま半導体技術が進展すると、ハードウエア資源とアーキテクチャとのギャップがますます大きくなり、ハードウエアの能力を生かし切ることができなくなる恐れがある。応用分野からのトップダウンの要求が高まり、半導体技術の進歩によるボトムアップの性能向上がなされても、その中間に位置するアーキテクチャが古いままであると、システム全体としてのバランスが非常に悪くなってしまう。よりＶＬＳＩを意識し、応用からの要求に応え、90年代にターゲットを合わせた、新しいマイクロプロセッサのアーキテクチャ体系が欲しい。

TRONプロジェクト[2][3]は、このような問題点を解決すべく、マイクロプロセッサから応用分野にわたる一貫したコンピュータ体系を、全く新しく作り直そうというものであり、1984年6月から始まった。将来のマイクロプロセッサの応用分野やユーザからくる要求と、1990年代のＶＬＳＩの技術予測とを調べ、従来のアーキテクチャからの互換性にとらわれない新しいアーキテクチャを提唱する。このアーキテクチャ体系は、リアルタイム特性に優れ、ＶＬＳＩベース、分散処理指向といった特長をもち、1990年代における一つの標準となることを目指す。

TRONプロジェクトの基本的な考え方をまとめると、次のようになる。

・1990年代の技術水準をターゲットとした設計を進める。

現在のマイクロプロセッサのアーキテクチャは8ビット時代との互換性をいまだに引きずっているものが多く、ＶＬＳＩ化を始めから考慮したものではない。ＶＬＳＩ化で得られる豊富なハードウエア資源を、古いアーキテクチャに無理に付け加えたような形になっており、無駄が多い。TRONプロジェクトでは、始めからＶＬＳＩの存在を前提とした設計を行なう。

・リアルタイム処理を前提とする。

従来の計算機の使い方は、バッチ処理やＴＳＳ処理に見られるように、「人間の方が計算機を待つ」ものであった。しかし、今後は自然界を含めた外部環境を制御し、人間の生活をより快適にするために多くの計算機が使用される。この場合は、「計算機の方が人間（または外部環境からの刺激）を待つ」といった形態になる。これがリアルタイム性の本質である。リアルタイム性は、分散処理、通信などの応用においても必須のものである。しかし、リアルタイム処理では計算機の稼働率がさがるため、同じ処理能力を得るにはより高性能の計算機が必要になる。TRONプロジェクトではリアルタイム処理を必須のものと考え、始めからそれを考慮して設計を行なった。

・究極のノイマン型計算機を目指す。

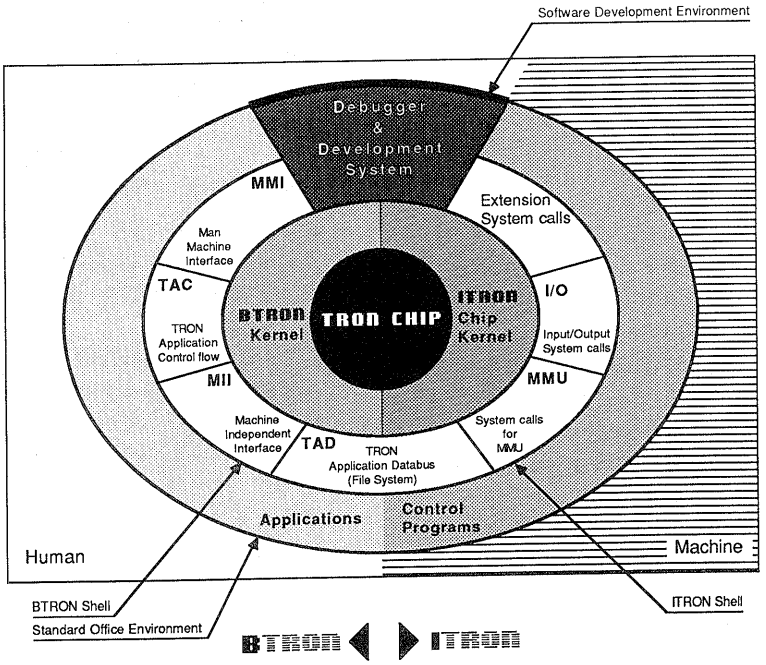
TRONプロジェクトでの研究の対象は、あくまでも汎用的なノイマン型計算機である。非ノイマン型アーキテクチャの研究はさかんに行なわれており、中にはノイマン型計算機の可能性をいたずらに否定する意見もある。しかし、ノイマン型計算機の汎用性や技術の完成度などを見た場合、1990年代になっても社会の中心になって使われるのがノイマン型計算機であることは間違いない。TRONプロジェクトの目標は、ＶＬＳＩの特徴を生かせるような究極のノイマン型計算機のモデルを提供することである。

・計算機の各階層の設計を同時に行ない、トータルアーキテクチャを目指す。

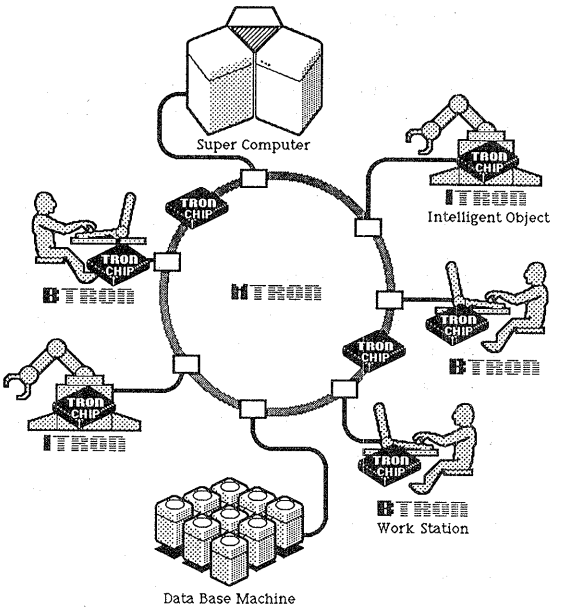
TRONプロジェクトでは、マイクロプロセッサからネットワークまでの設計を、同時に行なっている。したがって、ＯＳでネックになる部分はプロセッサの命令で強化したり、ネットワークの構築のために必要なシステムコールをＯＳに導入したりといった、広い範囲にわたる設計のフィードバックが可能である。このように、設計段階において階層間にまたがるチューニングまで行なえるため、実質的な計算機の速度であるアプリケーションの実行速度を大幅に向上させることが可能である。

TRONプロジェクトでは、4つのサブプロジェクトを同時に進行させている。プロジェクト全体の核となるのが、1990年代の半導体技術を生かせるようなアーキテクチャを持つマイクロプロセッサ、TRONCHIP[4]である。TRONCHIPは、1990年代のアプリケーションやＯＳの実行に最も適したマイクロプロセッサとなる。次にＯＳの階層として、組み込み型制御用のリアルタイムＯＳであるITRON（Industorial-TRON）[5][6]とワークステーション向きのＯＳであるBTRON（Business-TRON）[7][8]がある。ＯＳの階層を応用ニーズに応じて2つに分けたのは、組み込み用コンピュータとワークステーションコンピュータの違いが汎用性の限界を越えていると判断したからである。簡単に言えば、対人間の処理を行なうのがBTRON、対機械、対外部環境の処理を行なうのがITRONである。さらに、複数のITRONやBTRONをネットワークで有機的に結合し、人間の生活する環境全体を総合的に操作するのがMTRON（Macro-TRON）である。[図1]にTRONプロジェクト全体の階層構造図を、[図2]にMTRONの模式図を示す。

TRONプロジェクトの特徴の一つとして、設計基準、すなわちアーキテクチャと実現手段、すなわちインプリメントを分離したことがある。ここで、アーキテクチャという言葉を定義しておこう。TRONプロジェクトの中では、G.M.Amdahlが定義した「アーキテクチャ」[9]の意味を拡張し、「プログラマから見た命令セットのみならず、オペレーティングシステム、マン=マシン・インターフェースまで含めた、システムの各層の属性を規定するもの」といった意味で「アーキテクチャ」という言葉を用いている。具体的には、TRONCHIPの命令セットから、ITRONやBTRONのシステムコール、BTRONのマン=マシン・インターフェースまでのすべてのインタフェース仕様がTRONのア



[図1] TRONプロジェクトの階層構造図
Hierarchical Structure of TRON Project



[図2] MTRONの模式図
Image of MTRON Structure

ーキテクチャに含まれる。アーキテクチャとインプリメント手法の分離により、ある階層より下のインプリメンテーションが全く変わっても、それより上位の階層はそのまま使用できるという利点がある。すなわち、階層ごとに互換性が保たれている。また、TRONプロジェクトで重要なことは、アーキテクチャは規定するが性能に関する規定は行なわないことである。したがって、その部分がインプリメンタ間の自由競争の部分となる。ユーザから見ると、使い勝手は同じだが性能は異なるという製品が、メーカーの違いや時期の違いによって何種類も提供されるわけである。つまり、TRONプロジェクトは、統一化と同時に公平な技術競争を行なうための土俵を提供するものであると言える。

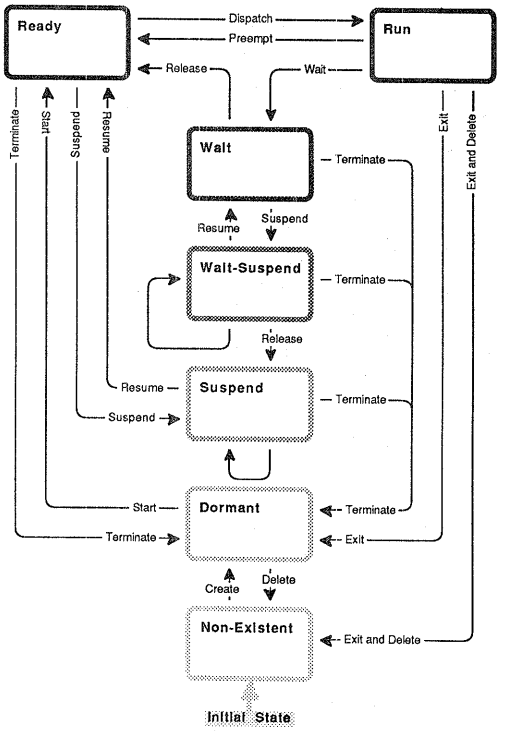
本論文の以下の章では、TRONプロジェクトのうちのITRON（Industorial-TRON）について、その概要を説明する。ITRONの今後の拡張や発展についての考え方を述べ、その中でMMU版のITRON（ハードウエアのメモリ管理機構をサポートするITRON）、32ビットプロセッサ版のITRON、TRONCHIP上のITRONなどについて具体的に述べる。

2. ITRON

ITRONは、機器組み込み用のリアルタイム・マルチタスクオペレーティングシステムである。ITRONのシステムコール一覧表を[表1]に、ITRONの状態遷移図を[図3]に示す。ITRONは、ITRONチップ核とITRONチップ核周辺から構成されている。ITRONチップ核は、タスク関連操作、タスク間同期・通信機能、割り込み処理機能、例外処理機能、メモリ管理機能といった、リアルタイム、マルチタスク処理の基本となる部分である。一方、ITRONチップ核周辺は、システムコール拡張機能、入出力機能、ファイル管理機能などといった、システム構成によって変化の大きくなる部分である。ITRONチップ核周辺を組み合わせることによって、いろいろなシステム構成に対応することができる。

ITRONチップ核周辺は、さらにITRON外核とITRONユーティリティタスクから構成される。チップ核周辺の機能のうち、システムコールの実行として実現されるのがITRON外核であり、タスクの形で実現されるのがITRONユーティリティタスクである。

ITRON外核に含まれる機能は、システムコール拡張機能、MMU（ハードウエアメモリ管理機構）操作機能、ファイル、入出力管理機能である。



[図3] ITRONのタスク状態遷移図
Task State Transition of ITRON

1．タスク管理	
cre_tsk	タスクを生成する
sta_tsk	タスクを起動する
del_tsk	タスクを削除する
def_ext	終了時処理ルーチンを定義する
ext_tsk	自タスクを正常終了する
exd_tsk	自タスクを正常終了後、削除する
abo_tsk	自タスクを異常終了させる
ter_tsk	他タスクを強制的に異常終了させる
chg_pri	タスク優先度を変更する
rot_rdq	タスクのレディキューを回転する
tcb_adr	タスクのＴＣＢアドレスを得る
tsk_sts	タスクの状態を見る
sus_tsk	タスクを強制待ち状態へ移行する
rsm_tsk	強制待ち状態のタスクを再開する
slp_tsk	タスクを待ち状態へ移行する
wai_tsk	タスクを一定時間待ち状態に移行する
wup_tsk	待ち状態のタスクを起床させる
can_wup	タスクの起床要求を無効にする
cyc_wup	タスクを周期起床する
can_cyc	タスクに出された周期起床要求を無効にする

2．同期・通信機能	
cre_flg	イベントフラグを生成する
del_flg	イベントフラグを削除する
set_flg	イベントフラグをセットする
wai_flg	イベントフラグの条件満足を待つ
flg_adr	イベントフラグアクセスアドレスを得る
cre_sem	セマフォを生成する
del_sem	セマフォを削除する
sig_sem	セマフォに対する信号操作（Ｖ命令）
wai_sem	セマフォに対する待ち操作（Ｐ命令）
sem_adr	セマフォアクセスアドレスを得る
cre_mbx	メイルボックスを生成する
del_mbx	メイルボックスを削除する
snd_msg	メッセージを送信する
rcv_msg	メッセージを受信する
mbx_adr	メイルボックスアクセスアドレスを得る

3．割り込み処理	
def_int	割り込みハンドラを定義する
ret_int	割り込みハンドラから復帰する
set_int	割り込みマスクをセットする(ITRON/68Kのみ)
dis_int	割り込みを禁止する(ITRON/86のみ)
ena_int	割り込みを許可する(ITRON/86のみ)
fet_dat	割り込みハンドラ内で使うＤＳをセットする(ITRON/86のみ)
get_dvn	割り込みを発生したデバイス番号を知る(ITRON/86のみ)

4．例外処理	
def_exc	例外処理ハンドラを定義する
ret_exc	例外処理ハンドラから復帰する

5．メモリ管理	
cre_mpl	メモリプールを生成する
del_mpl	メモリプールを削除する
get_blk	メモリブロックを獲得する
rel_blk	メモリブロックを解放する
mpl_adr	メモリプールアクセスアドレスを得る

6．時間管理	
set_tim	システムクロックを設定する
get_tim	システムクロックの値を読み出す

[表1] ITRONチップ核システムコール一覧表
System-calls of ITRON Chip Kernel

2.1. ITRONの特徴

ITRONは、リアルタイム処理時の効率を最大にすることを目標にして設計されており、次のような特徴を持つ。

・プロセッサ毎に独立して ITRONの実装設計を行なう。

ITRONは、複数のマイクロプロセッサを対象とした標準ＯＳである。標準ＯＳといった場合に最初に考えられるのがマシンを仮想化することであるが、リアルタイムＯＳとしての最高の性能を得るためには、どうしてもプロセッサの能力を最大に生かすようなアーキテクチャが必要であり、マシンの仮想化やＯＳの極端な仮想化は避けなければならない。さらに、組み込み用という用途を考えると、オブジェクトの互換性に対する要求はそれほど強くはない。むしろ、システムコールの名前や機能の統一といった、教育の面からの互換性が重要なことが多い。したがって、ITRONの標準仕様では、各システムコールの種類や名前、パラメータの種類などを規定するにとどめ、より細かい部分は ITRONを実装するプロセッサ毎に決めるようになっている。プロセッサ間の相違が特に大きくなるところは、割り込み処理、例外処理関係である。

現在、ITRONの実装を直接考慮しているプロセッサは、インテル iAPX86 系（ ITRON/86）、モトローラ M68000 系（ ITRON/68k）のプロセッサ、　ナショナルセミコンダクタ NS32000 系のプロセッサ、およびTRONCHIPである。

マシンの仮想化を行なっていないため、プロセッサ間のプログラム移植性が必要な場合には、高級言語を使用する。ITRONで標準的に使用する高級言語は言語Ｃであり、アセンブリ言語からの ITRONインタフェースとともに言語Ｃからの ITRONインタフェースも規定されている。

・いろいろな機能を持つシステムコールが豊富に用意されている。

例えば、ITRONでは同期手段としてイベントフラグ、セマフォ、および各タスクに付随した同期機構（ slp_tsk，wup_tsk システムコールで使用）など、考えられるほとんどのメカニズムを提供している。このため、応用に最適なメカニズムを選択することによる性能向上やプログラムの書きやすさが期待できる。

・高速な応答が可能である。

リアルタイム応用において応答速度に特に影響する部分は、タスクの切り替え（ディスパッチング）を行なう部分と割り込み処理ハンドラを起動する部分である。後で述べる適応化と関連するが、ITRONではディスパッチ時に入れ換えを行なう汎用レジスタを指定する機能や、複合システムコールの機能などにより、高速のディスパッチングができるように考慮されている。

また、外部割り込み発生時にはＯＳを介することなくユーザの定義した割り込み処理ハンドラを起動できるようになっており、この部分のオーバーヘッドは基本的にゼロである。ただし、割り込み処理ハンドラで使用するレジスタは、ユーザ側で退避する必要がある。

・適応化を考慮している。

ITRONは一つのＯＳアーキテクチャであるが、組み込み用ということを考慮し、柔軟で適応性の強いアーキテクチャになっている。システムコールの取捨選択、ディスパッチ時に入れ換えを行なうレジスタの指定、実行（可能）状態のタスクがない時の低消費電力モードの使用、システムコール実行におけるエラーチェックの有無、などの選択の自由度がユーザに開放されており、これを用いて適応化を行なうことができる。

例えば、メモリアドレスを指定するパラメータがあった場合、その値がセグメンテーションエラー、バスエラーなどのメモリ関係のエラーを起こす可能性があるかどうかをソフトウエアでチェックすると、システムコールの実行にとってかなりのオーバーヘッドになる。一方、ユーザの書いたプログラムが正しければ、パラメータがエラーを起こすことはないはずである。以上のような状況では、デバッグ時に多少のオーバーヘッドがあっても完全なエラーチェックを行なっておき、デバッグが終了してからエラーチェックをはずすようにするのが一番望ましい。ITRONアーキテクチャでは、エラーチェックの有無にも自由度を持たせているため、このようなことが可能である。

・教育を重視している。

ITRONのシステムコールの名前や機能はいくつかの基準に基づいて系統的に決められており、理解が容易になっている。例えば、ITRONのシステムコール名は‘xxx_yyy’型の7文字であり、‘xxx’が操作の方法、‘yyy’が操作の対象（オブジェクト）を表わすのが基本である。また、ITRONでは自タスクの操作をするシステムコールと他タスクの操作をするシステムコールが明確に分離されており、この特徴は、終了、削除のシステムコール（del_tsk，exd_tsk）、異常終了のシステムコール（abo_tsk，ter_tsk）、強制待ち状態へ移行させるシステムコール（sus_tsk）などに見られる。[注1]

2.2. 複合システムコール

ITRONの適応化の考え方をシステムコール新設にまで拡張したものとして、複合システムコールの概念がある。

連続して使用される頻度の多いシステムコールの流れがあった場合に、それを一つのシステムコールに統合することができれば、システムコール呼び出しなどに伴うオーバーヘッドが減り、性能を向上させることができる。複合システムコールは、このような原理に基づいていくつかのシステムコールを合成し、実行速度や使いやすさの向上をめざしたもの

[注1] del_tsk は他タスクを削除するシステムコール、exd_tsk は自タスクを終了、削除するシステムコールである。del_tsk で自タスクをパラメータとして指定した場合に exd_tsk の機能を実行するような仕様も考えられるが、自タスクの操作と他タスクの操作をシステムコールレベルで分離するという思想のため、exd_tsk を del_tsk とは別に設けている。また、自タスクを異常終了するシステムコールが abo_tsk、他タスクを異常終了するシステムコールが ter_tsk となっているが、これもあえて別のシステムコールとしたものであり、自タスクに対して ter_tsk を発行した場合にはエラーとなる。さらに、ITRONでは、強制待ち状態は「他タスクの sus_tsk システムコール実行の結果による待ち状態」と定義されており、自タスクに対して sus_tsk を実行することはできない。これは、インプリメント上の制約によるものではなく、自タスクの操作と他タスクの操作を明確に分離するという ITRON の思想を反映したためである。

である。

チップ核に相当するレベルの複合システムコールとしては、割り込み処理ハンドラからのリターンとタスク起床を行なうシステムコール　iret_tsk, iret_wup が標準的に用意されている。さらに、外核に相当するレベルの複合システムコールとして、MMU操作用の複合システムコールやファイル関係の複合システムコールがある。

ITRONの複合システムコールの特徴を以下に述べる。

- 複合システムコールは、実行速度や使いやすさの向上のためにいくつかのシステムコールを合成したものである。複合システムコールは他のシステムコールの組み合わせで代用することができるため、ITRONで規定した最小限の仕様には含まれず、インプリメント側の判断で導入できるものである。また、アーキテクチャ上の直交性はなく、プロセッサへの依存性も大きい。なお、ここで「直交性がない」とは、一つの機能を実現するのに複数のやり方が存在することを言う。

- 複合システムコールは、考え方の上では複数のシステムコールを合成したものになっているが、実際のインプリメントはどのように行なっても構わない。逆に、インプリメント方法を工夫して効率を上げることが望ましい。

- 複合システムコールの名前は、'xyyy_zzz' の形の8文字とする。このうち、'x' は何に関する複合システムコールかを表わす。'x' は、割り込み関係では 'i'、MMU関係では 'm' となる。

ITRONチップ核の複合システムコールを[表2]に示す。

iret_tsk	割り込み処理からの復帰と割り込みタスク起床を行なう（ITRON/86のみ）
iret_wup	割り込み処理からの復帰とタスク起床を行なう（ITRON/68Kのみ）

[表2] ITRONチップ核の複合システムコール一覧表
Complex System-calls of ITRON Chip Kernel

3. ITRONの発展

一般的な計算機システムの形態には、シングルプロセッサとマルチプロセッサの区別、メモリ管理方式の区別に従って、[表3]に示すようにいろいろなものが考えられる。ITRONはどのような形態のシステムにも載せられることが理想であるが、実際問題として各形態間の隔たりはあまりにも大きく、核の拡張だけではこれらの違いのすべてを吸収することはできない。

そこで、ITRONでは[表3]の各形態を一応別のものとしてとらえ、①～⑥の番号順に開発を進める方針をとっている。もちろん、①～⑥の全体に流れる思想は一貫させ、システムコールもできる限り共通化する。

ITRONチップ核は、[表3]の①、つまりシングルプロセッサ、メモリ保護機構なしの形態に対して、リアルタイムOSの仕様を提示したものである。次に提示されるのが、②、つまりメモリ保護機構付きのシステム形態に対するリアルタイムOSの仕様であるが、これに対応するITRONを ITRON/MMU と呼ぶ。ITRON/MMU は、ハードウエアのメモリ保護機構をサポートするITRONである。

	シングルプロセッサ	マルチプロセッサ
メモリ保護機構なし	①	③
メモリ保護機構あり	②	⑤
仮想記憶サポート	④	⑥

① シングルプロセッサ、メモリ保護機構なし
② シングルプロセッサ、メモリ保護機構あり
③ マルチプロセッサ 、メモリ保護機構なし
④ シングルプロセッサ、仮想記憶サポート
⑤ マルチプロセッサ 、メモリ保護機構あり
⑥ マルチプロセッサ 、仮想記憶サポート

[表3] 計算機システムの構成の分類（参考文献[15]より）
Variations of Computer Systems

3.1. ITRON/MMU

ここでは、①メモリの保護、②論理空間と物理空間の分離やプログラムのリロケーション、といった二つの機能をハードウエアで実現するものを、MMU（メモリ管理機構）と呼ぶ。①と②は、本来別の目的を持つ機能であるが、ほとんどのプロセッサのMMU機能では、論理空間を多重化する方法によって①と②を同時に実現するようになっており、物理空間と論理空間を分離することが実質的なメモリ保護の手段となっている。つまり、実現の面からは①と②を切り離して考えることができない。

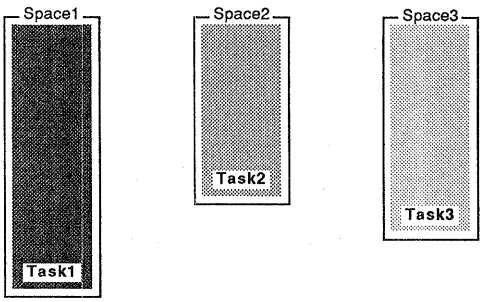
しかし、論理空間と物理空間の分離を行なった場

合、従来の ITRONと ITRON/MMU のアーキテクチャ面での隔たりが大きくなり、MMU版への移行がスムーズにはいかないという問題点が生ずる。また、組み込みシステムでは、タスクと論理空間との対応、すなわちメモリ構成に関してもいろいろな要求があり、必ずしも論理空間の多重化が望ましいわけではない。例えば、タスク（プロセス）間の独立性が強く、実行されるプログラムが完全に動的に決まる開発システム向きの用途では、タスクと論理空間を1対1に対応させるのが最も適当であり、大部分の開発ＯＳではこの方法がとられている。しかし、組み込み用途ではタスク間の従属関係が強い場合が多く、いちいち共有メモリを利用するよりも関連タスクを同一空間とした方がかえってすっきりする場合がある。そもそも、マルチタスクＯＳの導入には、もともと一つのプログラム（タスク）であった処理を複数のタスクに分割することにより、ソフトウエアの生産性を上げようというねらいがあった。その場合には、必ずしもタスク間のメモリ保護は必要なく、単にタスク単位の並行処理の機能だけを利用したいことが多かったわけだ。このような場合には、「実行の単位」としてのタスクと「アドレス、保護の単位」としての論理空間とは独立に考えられる方が望ましい。また、組み込み機器向きという ITRONの用途を考えると、すべてのオブジェクトコードをリンク、リロケーションしておき、①の機能のみMMUで実現できればよいことが多い。

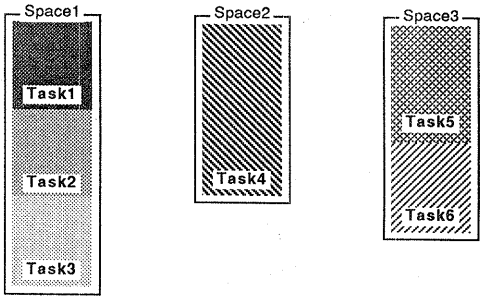
そこで、 ITRON/MMU では、組み込みシステムとしてのいろいろな要求に対応するため、タスクの生成、削除といったタスク操作の指定と論理空間の生成、削除といった論理空間操作の指定を全く独立したシステムコールで行なうようにした。その結果、メモリ構成の自由度が大きくなり、①②の両方の目的の達成度をユーザのレベルで選択できるようにもなった。これは、 ITRONの適応化の思想に基づいたものである。

一方、システム構成の自由度を増やすことは、一つ一つのシステムコールのレベルを低くしてプリミティブなものにすることにつながる。その結果、同じ処理を行なうためにも多くのシステムコールを発行しなければならず、メモリ管理の自由度をあまり必要としないユーザには向かない面がある。

ITRON/MMU では、 先にあげた複合システムコールを設けることでこの問題に対処する。 ITRON/MMU にはMMU関係の複合システムコールが用意されており、タスクと論理空間が1対1に対応した標準的なメモリ構成をとる場合には、こちらを使う方が簡単な操作で済む。 ITRON/MMU で提供されるメモリ空間のモデルを[図4]に示す。



ITRON/MMUで複合システムコールを用いた場合のタスクと論理空間との対応
（一般的な開発ＯＳでのタスクと論理空間の対応）



ITRON/MMUにおいて複合システムコールを用いず、メモリ構成の自由度を生かした場合のタスクと論理空間との対応

[図4] メモリ構成のモデル
Models of Memory Structure

ITRON/MMU におけるMMUの機能とシステムコールとの対応は、次のようになる。

- ITRONチップ核のシステムコール

  ITRONチップ核のシステムコールは ITRON/MMU でもそのまま利用でき、名前、 機能、パラメータなどを含めてMMUなしの ITRONのシステムコールと共通になっている。ただし、これらのシステムコールでは、多重論理空間などのMMUの機能は全く使用しない。例えば、 ITRON/MMU の cre_tsk システムコールは、新しいタスクを生成するだけで、MMUに関する操作は行なわない。したがって、従来の ITRONとの互換性は、メモリ空間の問題まで含めて完全に保たれている。

  MMUなしの ITRONで指定していたアドレス関係のパラメータについては、 ITRON/MMU

では、原則として論理アドレスの指定という意味になる。

- **ITRON**/MMU　で追加されたシステムコール（複合システムコールを除く）

これらのシステムコールは、MMUに関する操作のみを行なうシステムコールである。具体的には、新しい論理空間を生成するもの、論理アドレスと物理アドレスの対応をとるもの、メモリの保護属性を設定するもの、などがある。

- MMU複合システムコール

メモリ管理とは直接関係のないシステムコールに対し、多重論理空間などのMMUの機能をサポートするように変更を加えたものである。例えば、タスク生成のシステムコール cre_tsk にMMUサポート機能を付け加えた **ITRON**/MMU の複合システムコールとして mcre_tsk があり、mcre_tsk ではタスクの生成に加えて新しい論理空間の生成やタスクと論理空間の対応をとる処理などを行なう。

ユーザから見ると、論理空間を別にしたタスクを生成する場合には mcre_tsk を使用し、論理空間を共通にしたタスクを生成する場合には cre_tsk を使用すればよいことになり、両者の使い分けができる。

ITRON/MMU のシステムコール一覧表を[表4]に、**ITRON**/MMU の複合システムコール一覧表を[表5]に示す。

cre_spc, del_spc は、論理空間全体に対する操作を行なうものである。それに対して cre_map, sha_map, trf_map, del_map は、論理空間中のあるアドレスに対する操作を行なうものである。インプリメント上は、cre_spc により新しいページテーブル（セグメントテーブル）領域の確保とそれに対する論理空間ID番号の割り当てを行ない、cre_map, trf_map, sha_map により個々のページテーブルエントリーの操作を行なうということになる。MMU関連の複合システムコールの名前は、'myyy_zzz' の形になっている。

現在、**ITRON**/MMU の実装を直接考慮しているプロセッサは、インテル iAPX286 系（**ITRON**/MMU286）、モトローラ M68000 系（**ITRON**/MMU68k）、ナショナルセミコンダクタ NS32000 系、および**TRONCHIP**である。

マイクロプロセッサのメモリ管理アーキテクチャは、OS空間（あるいはシステム共通空間）と、ユーザタスク空間（あるいはタスク固有空間）の区別をどのように行なうか、セグメント方式かページング方式か、などといった項目によって、いくつかのモデルに分けられる。しかし、ユーザの論理空間をプロセス（タスク）毎に分離して多重論理空間となっている点、および（簡易）リング保護を行なっている点はプロセッサ間で共通である。**ITRON**/MMU を実装するメモリ管理アーキテクチャのモデルとしては、この多重論理空間、（簡易）リング保護といった二つの特徴を持つものを想定しており、大部分のマイクロプロセッサの上に、共通の考え方で **ITRON**/MMU を載せることができる。ただし、セグメントマシン、ページングマシンといったメモリ管理アーキテクチャの違いによって、パラメータの種類やシステムコールの機能の詳細は異なる。

3.2. 32ビットプロセッサ対応

ITRONの標準仕様はプロセッサのビット数には独立に決められているので、プロセッサが32ビットになってもアーキテクチャには特に変更はない。

32ビットのマイクロプロセッサでは、ビット幅の

cre_spc	新しい論理空間を生成する
del_spc	論理空間を削除する
exd_spc	自タスクを終了、削除し、論理空間も削除する
spc_adr	論理空間管理ブロックのアクセスアドレスを得る
cre_map	物理空間から論理空間への割り付けを行なう
trf_map	ある論理空間から別論理空間への再割り付けを行なう
sha_map	論理空間の共有割り付けを行なう
del_map	論理空間への割り付けを解除する
map_adr	マッピング管理ブロックのアクセスアドレスを得る
set_prt	メモリ保護属性をセットする
get_dat	システムデータの読みだしを行なう

[表4] **ITRON**/MMU システムコール一覧表
System-calls of ITRON/MMU

mcre_tsk	独立した論理空間を持つタスクを生成する
mdel_tsk	独立した論理空間を持つタスクを削除する
mexd_tsk	独立した論理空間を持つ自タスクを終了、削除する
mget_blk	指定したアドレスにメモリを獲得する
mrel_blk	指定した論理空間のメモリブロックを解放する
msnd_msg	異なる論理空間へメッセージを送信する
mrcv_msg	異なる論理空間からメッセージを受信する

[表5] **ITRON**/MMU の複合システムコール一覧表
Complex System-calls of ITRON/MMU

拡大以外に、メモリ管理機構の導入、レジスタ数の増加、高機能命令の導入、コプロセッサの接続などが行なわれている。このうち、アーキテクチャレベルの変更を要するのはメモリ管理機構であり、これについては前に述べた。

レジスタ数の増加は、リアルタイムOSにとっては、ディスパッチ時間の増加といったマイナス要因になることが多い。しかし、ITRONではディスパッチ時に入れ換えを行なうレジスタを指定することができるので、不必要なレジスタの退避、復帰を行なう必要がない。逆に、ユーザの側で各タスクで使用するレジスタをずらしていくことにより、多くのレジスタを有効に利用することが可能になる。例えば、従来のリアルタイムOSでは、レジスタが16個あるプロセッサで各タスクが10個のレジスタを使用するプログラムを走らせる場合、ディスパッチ毎に6個ずつのレジスタを無駄に退避、復帰しなければならなかった。しかし、ITRONでは、各タスクで使用する10個のレジスタのみを入れ換えるように指定できるため、レジスタ数が多くなっても全く無駄な動作をしなくて済む。ディスパッチされない6個のレジスタは、タスク間の共通変数や緊急度の高いタスクの専用レジスタとして使用することができる。

高機能命令の導入は、アーキテクチャを変更することなくITRONの性能を向上させるのに役立つ。具体的にITRONの実装に役立つのはキュー操作命令、コンテキストスイッチ命令などである。

浮動小数点演算プロセッサなどのコプロセッサの接続に対しては、タスク生成時にコプロセッサ使用のオプションを設けることで対処している。

4. TRONCHIPとの関係

ITRONと並行して進められているTRONCHIPプロジェクトでは、ITRONを最も効率良く実行できる32ビット汎用マイクロプロセッサの開発を行なっている。

TRONCHIPの設計においては、iAPX86やM68000などの既存のプロセッサにITRONの実装を行なった結果をもとに、ITRONの実装上ネックになりやすいところを解析し、それをカバーする形で命令セットを決定した。

TRONCHIPの命令のうち、ITRONの実行を直接考慮して決められている高機能命令は、具体的には次の二つである。

・高速ディスパッチ用命令

タスク切り替えの際には、タスク固有の資源であるレジスタを入れ換える必要がある。この時に有効なのは複数のレジスタ群を転送する命令であり、レジスタの退避と復帰を効率よく行なうことができる。複数レジスタの転送命令は、TRONCHIPではもちろん、既存のプロセッサでも導入されている。

しかし、レジスタ退避エリアを一般のメインメモリ上にとると、メモリアクセスにかなりの時間がかかり、十分な性能が出ない。アクセス時間を短縮するためにはキャッシュメモリを用いる方法もあるが、普通のキャッシュでは、レジスタ退避エリアが必ずキャッシュ上に置かれるという保証がなく、最悪の場合を保証すべきリアルタイムOSには向かない。

そこで、TRONCHIPでは、チップバスアーキテクチャのレベルでキャッシュとは別のレジスタ退避用高速メモリを設ける予定である。この高速メモリは、通常のアドレッシングでアクセスする汎用メモリとは異なり、アドレス変換を行なわず、単純化された専用のプロトコルによってアクセスされる点が特徴である。専用のプロトコルにより、アドレス変換、キャッシュとの比較、システムバスへのメモリアクセス要求といったオーバーヘッドがなくなるため、一般的なメモリアクセスよりもはるかに高速のアクセスが可能となる。

TRONCHIPの高速ディスパッチ用命令では、この高速メモリをレジスタ退避用のエリアとして利用することにより、複数レジスタの入れ換え操作を高速、確実に行なうことができ、ディスパッチ時間を数μSにすることが可能である。（なお、現在のITRON/68kでは、8MHzの68000で100μS程度のディスパッチ時間である。）また、TRONCHIPの集積度が上がった場合には、退避用の高速メモリをTRONCHIPに内蔵することにより、さらに高速のレジスタ退避がチップ内のみで行えるようになる。

・キュー操作用命令

ITRONでは優先度ベースのスケジューリングを行なっており、優先度を持ったキューの管理をすることが必要である。キュー操作の処理時間は、タスク数が増加するにしたがって長くなるため、この部分もかなり性能に影響する。また、キュー操作では複雑なポインタの入れ替えを伴うため、割り込みやマルチプロセッサに対する考慮も必要になる。TRONCHIPでは、このような

キュー操作に適した命令として、キューをサーチする命令、キューに新しいエントリを挿入する命令、キューからエントリを削除する命令を備えている。これらの命令はそのまま ITRONのインプリメントに役立つようにできており、 ITRONの性能を大幅に向上させることができる。

5. BTRONを使った ITRONの開発

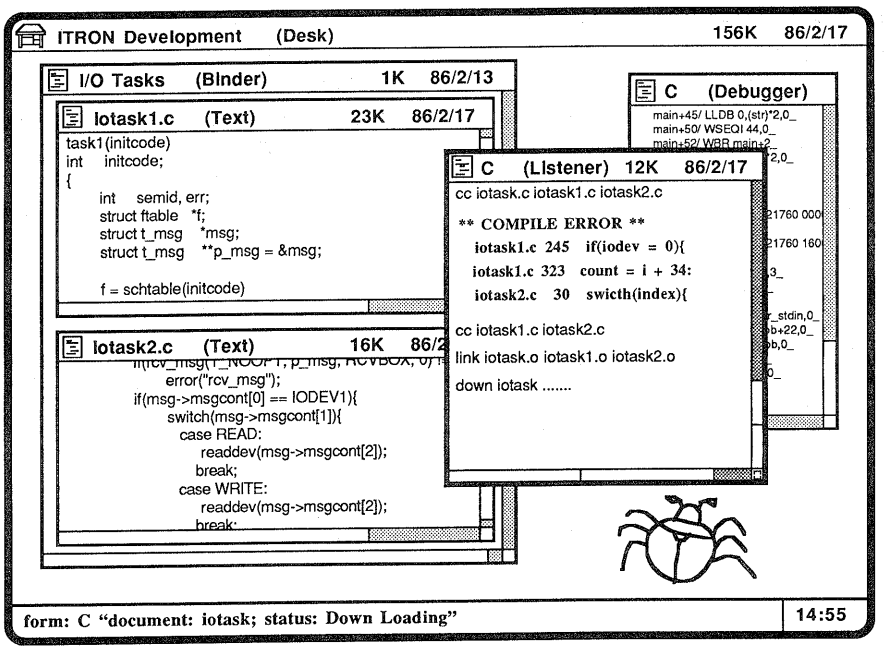
BTRONはワークステーション用のＯＳであり、 ITRONの開発システムにもなり得る。BTRONを使って ITRONアプリケーションの開発を行なうことにより、統合化、マルチウインドウといった高度な環境での開発が可能になる。

一般に、ビジュアル・インタラクションを使った操作環境は、 高度なマン=マシン・インタフェースを実現できる反面、すべての操作は、画面に見えているものをユーザが選択することによって行わなければならないことが多い。したがって、定型的な操作であっても、それをマクロコマンドとして登録することができない。例えば、プログラム開発の場合、ソースプログラムをエディトした後のコンパイル、アセンブル、リンク、ダウンロードなどを一つ一つユーザが指示してやらなければならない。インタラクティブな操作は非定型的な操作には向いているが、定型的な操作には向いていない。

この欠点を解決するため、 BTRONでは実身（従来のＯＳのファイルに相当するもの）に付箋という処理の指示書を貼ることができる。付箋とは、ファイル間の処理の流れを記述したものであり、メニューにより選択された時、データの内容が変更された時、などの時点で付箋に書かれた処理を起動することができる。したがって、付箋にエディト、コンパイル、リンク、ダウンロードといった処理の流れを記述しておくことにより、インタラクティブな環境の中で、ソースプログラムの変更を自動的にオブジェクトにまで反映させることが可能になる。 ITRONアプリケーションプログラムを開発中のBTRONの画面イメージを[図5]に示す。

また、BTRONによる ITRONの開発を容易にするため、BTRONのファイルと ITRONのファイルは互換性がある。

通常、BTRONのような開発システムでのファイルにはファイルの柔軟性（サイズの自由度など）が要求され、 ITRONのようなリアルタイムシス



[図5] BTRONによる ITRONの開発
Development of ITRON with BTRON

テムのファイルにはリアルタイム性（連続ブロックの確保など）が要求されており、両者の要求は正反対のものである。そこで、BTRONとITRONに共通のファイルシステムでは、構成ブロック数の値を可変にすることでこれに対応した。構成ブロック数とは、何個までの論理ブロックを物理的に連続したエリアに置くかといった数であり、ファイル毎に異なる値をとる。構成ブロック数を1にしておけば、ファイルの生成、変更、移動などによってフラグメンテーションが生じた場合でも無駄なエリアが少なくて済み、開発ＯＳ向きのファイル構成となる。一方、構成ブロック数を非常に大きな値にしておけば、実質的に連続ファイルになり、リアルタイム応用向きのファイル構成となる。パラメータ値の指定だけで自由度に対する要求とリアルタイム性に対する要求の両方を満たすことが可能になった。

BTRONのファイルに含まれる管理情報のうち、ユーザ名、アクセス権などについての情報はITRONではサポートしない。この点についてはITRONのファイルがBTRONのファイルのサブセットとなっているからである。

6．終わりに

本論文では、ITRONを含むTRONプロジェクト全体の概要、ITRONとTRONプロジェクト全体の関係について説明し、ITRONの最近の拡張や開発システムなどについて述べた。

現在、複数のメーカーによって、多くのマイクロプロセッサ上でITRONの実験研究やインプリメントが行われているが、これだけ多品種のマイクロプロセッサ上にインプリメントされたリアルタイムＯＳは、世界でも稀なものと思われる。さらに、国際的な標準化ＯＳの一つとしてITRONを提案する作業も進められている。

ITRONとのそのファミリーの機能や性能は現在でも高い評価を受けているが、ITRON/MMUへの拡張、TRONCHIPへの実装、BTRONを使った統合開発システムの整備などによって、よりすぐれた性能とより充実した機能を持つ1990年代の標準リアルタイムＯＳに成長するであろう。TRONプロジェクトが最終的に完成した時点では、このプロジェクトが日本の産学共同によるオリジナルな成果として世界に誇るに足るものになると確信している。

最後に、本プロジェクトの主旨にご賛同して頂き、日頃ご援助、ご協力頂いている日本電気、日立製作所、富士通、松下電器産業、三菱電機を中心とする多くの関係者の方々に感謝の意を表したい。

参考文献

[1] 坂村 健：電脳都市，冬樹社

[2] 坂村 健：TRONプロジェクト，マイクロコンピュータ応用国際コンファレンス（IMAC）'84

[3] 坂村 健：TRONトータルアーキテクチャ，アーキテクチャワークショップインジャパン'84

[4] Sakamura Ken: "Development of TRON Chip: A single chip VLSI computer architecture in the 1990's", Proceedings of IFIP VLSI 85

[5] 坂村 健：リアルタイムオペレーティングシステム- ITRON，オペレーティングシステム研究会資料 24-10（1984）

[6] 坂村 健：リアルタイム・オペレーティングシステム ITRON，日本ロボット学会誌 Vol.3 No.5（1985）

[7] 坂村 健：BTRON スーパー・パーソナル・コンピュータ，計算機アーキテクチャ研究会資料 57-4（1985）

[8] 坂村 健：BTRONにおける統一的操作モデルの提案，情報処理 Vol.26 No.11（1985）

[9] Amdahl,G.M., G.A.Blauw, and F.P.Brooks, Jr.: "Architecture of the IBM System/360" IBM J.Res.Dev., Vol.8, No.2（1964）

[10] 矢部 栄一、竹山 寛、堀越 健一：ITRON/68k，計算機アーキテクチャ研究会資料 61-2（1986.3）

[11] 工藤 健治、下原 明、安田 秀徳、本田 昭人：ITRON/MMU286，計算機アーキテクチャ研究会資料 61-3（1986.3）

[12] 坪田 秀夫、中村 和夫、榎本 龍弥：NS32032上へのITRONインプリメントの検討，計算機アーキテクチャ研究会資料 61-4（1986.3）

[13] 門田 浩、古城 隆：ITRON 32ビットへ向けて，計算機アーキテクチャ研究会資料 61-5（1986.3）

[14] 櫛木 好明: マイクロプロセッサ用オペレーティングシステムインタフェースMOSI（IEEE855）とITRONの国際標準化について，計算機アーキテクチャ研究会資料 61-6（1986.3）

[15] 日本電子工業振興協会: マイクロコンピュータに関する調査報告書 - リアルタイム組み込みＯＳ（ＴＲＯＮ）仕様の概要と将来動向，60-A-221（1985）