第Ⅳ章　リアルタイム制御処理システム

情報処理人材育成用指導書Ⅳ

第Ⅳ章　リアルタイム制御処理システム

学習目標

1．リアルタイム制御処理システムの典型的なハードウェア構成の概略を理解させる。
2．リアルタイム制御処理システムの一般的なソフトウェア構造について理解させる。
3．リアルタイム制御処理システム向きオペレーティングシステムの一般的な動作を理解させる。
4．マルチタスク構造のリアルタイム応用プログラムとその動作を理解させる。
5．リアルタイム制御処理システムにおける入出力の概略を理解させる。
6．アナログとディジタルの違い、A／D変換、D／A変換の概略を理解させる。
7．標準インタフェースの利点を知らせる。

内容のあらまし

リアルタイム制御処理システムでは、システムの中核機能となるコンピュータが制御対象である機械装置などの状態を時々刻々観察し、状態に応じて適切な指示を与える（制御する）、このようなコンピュータは制御対象である機械装置内に組み込まれていることも多い。例えば、車のエンジンはそのようなコンピュータで制御されている。

本章では、リアルタイム制御処理システムのハードウェア構成、コンピュータと制御対象である機械装置とのインタフェース、リアルタイム制御処理システムのソフトウェア構造、リアルタイム制御処理向きのオペレーティングシステム、リアルタイム応用プログラムの構造と動作の順で説明する。まず、この章全体の説明のあらましを以下の表に整理し示す。

節　・　項	内　容
1．リアルタイム制御処理	
（1）リアルタイム制御処理とは	要求時間以内での処理の完了、リアルタイム処理 外部事象（イベント）、リアルタイムOS
（2）事例	温度制御実験システム例、走行実験ロボット例
（3）ハードウェア構成	入力装置系、処理装置系、出力装置系 FAX装置事例のハードウェア構成
（4）応用ソフトウェアの機能と動作	FAX装置事例のソフトウェア機能（送受信処理）と構造と動作
2．リアルタイム制御処理の入出力インターフェース	
（1）入出力機構	CPUとI／O装置間のデータ授受 入出力インタフェース、システムバス、コントローラ
（2）A／D変換処理とD／A変換処理	ディジタル量によるI／O制御対象 アナログ量によるI／O制御対象、標本化、量子化
（3）割込み処理機構	コントローラからCPUへの非同期データ通知、割込み
（4）システムバスとインタフェースボード	インタフェースの標準化、外部インタフェース化
（5）標準インタフェース	IEEE－488（GPIB）、インタフェース RS－232Cインタフェース

節　・　項	内　容
3．リアルタイム制御処理システムのオペレーティングステム （1）リアルタイム制御処理システムにおけるモニタ機能 （2）リアルタイムオペレーティングシステム	リアルタイムモニタ、マルチタスクモニタ リアルタイムオペレーティングシステム タスクの状態遷移、タスクの切替え、スケジューラ プライオリティ方式、ラウンドロビン方式 ディスパッチャ、イベントドリブン方式 タスク間の同期、システムコール イベントフラグシステムコール
4．応用プログラムの構造 （1）リアルタイム制御処理の典型的な処理手順 （2）マルチタスク構造のリアルタイム応用プログラムの例 （3）リアルタイム応用プログラム例の動作	自動車の自動定速走行制御システムのモデル作成例題 システムの概要、システムの分析（入力装置、内部処理タスク、割込みハンドラの内部設計仕様） 例題モデルのプログラム設計仕様とコード例（速度監視タスクの流れ図、定速走行制御タスクの流れ図、割込みハンドラの流れ図、各タスクのCプログラム例） 例題モデルのタスク間の優先順位の設定 優先順位を背景としたタスクの動作図

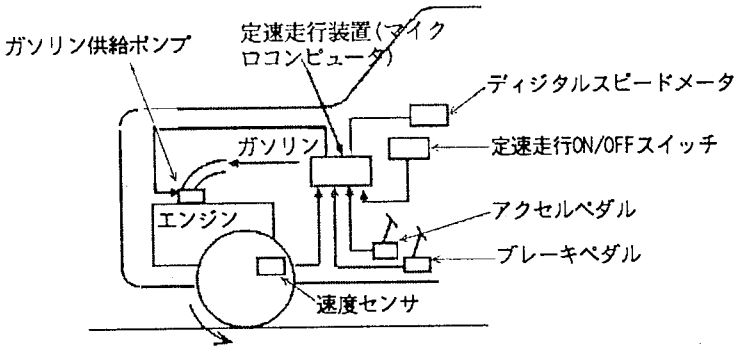
1．リアルタイム制御処理

（1） リアルタイム制御処理とは

対話型処理やオンライントランザクション処理と比較しながら、リアルタイム制御システムは、どんなことを行う情報処理システムかを以降で説明する。

自動車の制動や動力機構などがコンピュータ化によりソフトウェアで制御されることが多くなった。このような設計でも、急ブレーキのような緊急性の高い処理も瞬時に行える（図表Ⅳ－1）。

図表Ⅳ－1　コンピュータ組込みの自動車用の自動定速走行システムのモデル例



出典：第二種共通テキスト情報処理システム　中央情報教育研究所

このようにリアルタイム制御処理は、ハードウェアとソフトウェアをいかに適用し、要求された時間以内に目的を達成するかという処理方式の分野である。

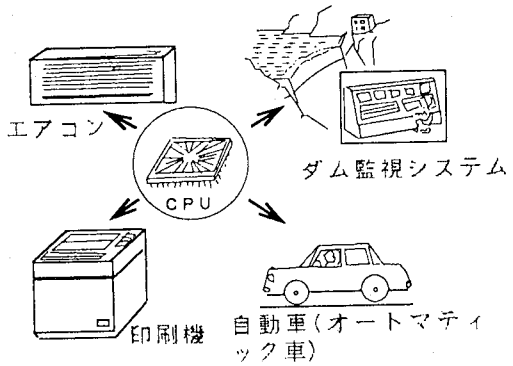
■ 要求された時間以内に処理の完了が要請される情報処理システムのオンライントランザクション処理や対話型処理もリアルタイム処理の一つである。

■ ここで述べていくリアルタイム制御処理では、さまざまなが外部事象が複数同時に発生し、限られた時間（通常はミリセカンドオーダ）のもとで処理しなければならないようなシステムを対象としている。

外部事象（イベントともいう。）とは、人間の入力したデータや指令によるものだけでなく、さまざまな環境の物理的・化学的変化の発生の検出を含む。

コンピュータは、その小型化やワンチップ化などによって安価に使用できるようになり、各種システムまたは製品の構成部品の一つとして使用されるようになってきている（図表Ⅳ－2）。

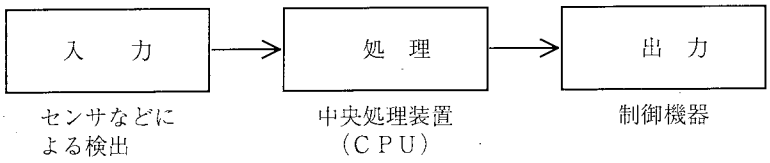
図表Ⅳ－2　コンピュータが使われている製品分野



出典：第二種共通テキスト情報処理システム　中央情報教育研究所

家庭用の電気製品のように構成部品の一部として使用される場合、ディスプレイもキーボードもないことが多い。したがって、そのハードウェア構成は、コンピュタ自身が外部情報を検出するための入力装置と、そのデータを時間以内に計算し判断する処理装置（通常、マイクロコンピュータ）と、外部に対して制御を行わなければならない出力装置とからなる（図表Ⅳ－３）。

図表Ⅳ－３　リアルタイム制御システムでの入力・処理・出力



また、処理装置には、外部事象に追従するように作成されたプログラムとリアルタイム制御に向いたオペレーティングシステム（OS）が必要となる。

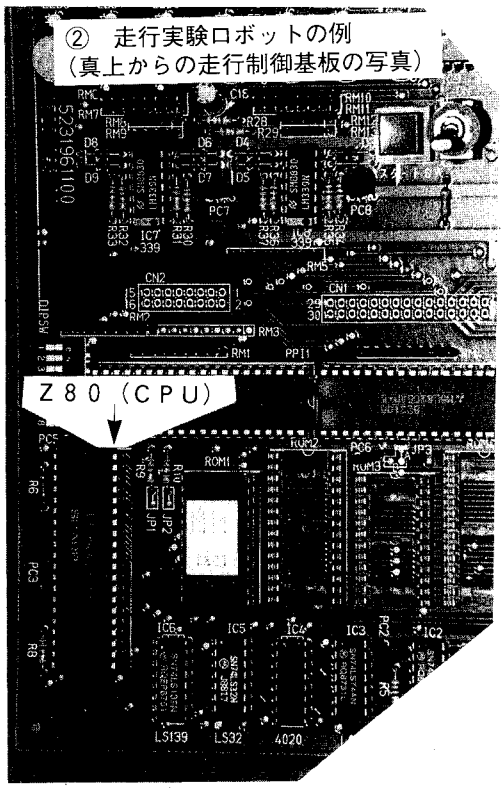
リアルタイム制御向きオペレーティングシステムは、一つには外部事象の発生に対して即座にタスク切替えを行う機能を有するという特徴をもつものである。現状ではパソコンやワークステーション用のOSのように標準化されておらず多種多様なものがある。機能の詳細等については３節で触れる。

（２）　事　例

リアルタイム制御処理システムは、大規模な管理制御システムから家庭で目にする小規模な電気製品まで実にさまざまな分野で使われている。

例えば、大規模なものでは、鉄道の運行管理システムや道路交通管制システムなどがあげられる。小なものでは、エアコン、FAX、電話機などがあげられる。ここでは、フィードバック制御の温度制御実験システムとマイクロコンピュータ組込みの走行実験ロボットを紹介しておく（図表Ⅳ－４）。

図表Ⅳ－４　リアルタイム制御処理システムの事例



（3） ハードウェア構成

典型的な事例－マイコンによるリアルタイム制御など－のハードウェア構成を示し、各構成要素の役割を説明する。

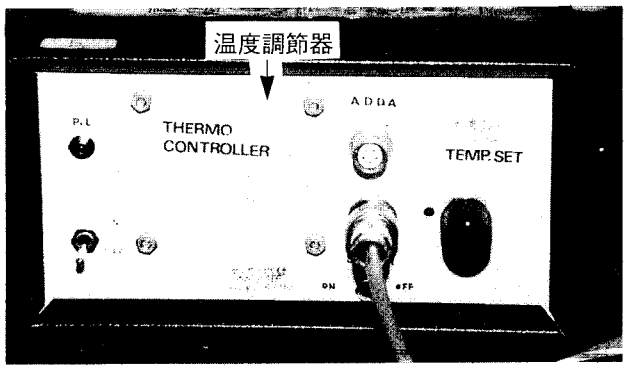
事例として家庭用の電気製品のように構成部品の一部として使用されている場合を考える。(1)項で述べているように（図表Ⅳ－3を参照)、そのハードウェア構成は入力装置、処理装置、出力装置の3部からなる。

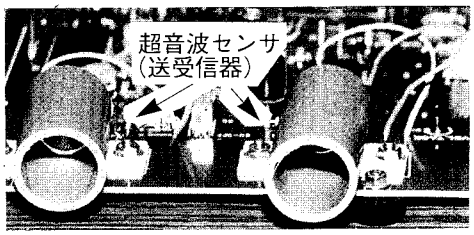
① 入力装置

外部事象の状態変化を検出する。例えば、温度センサなどのアナログデータ発生機器や、超音波センサなどの直接パルスを発生する機器や、切替えスイッチのような操作機器などがある（図表Ⅳ－5)。

・ アナログデータ発生（電圧、電流）の場合は、A／D変換を行って処理装置へ取り込まれる。詳しくは次節で説明する。

図表Ⅳ－5 入力装置例（温度センサ、超音波センサ）





② 処理装置

通常、ワンチップマイコンやパーソナルコンピュータなどが使用される。入出力装置とはインタフェースボードを介して、システムバスに接続される。

インタフェースボードからの入力情報を中央処理装置（CPU）に伝える方法は、ハードウェアの割り込みによる方法とソフトウェアから一定間隔に状態を見にいく二つの方法がある。

③ 出力装置

CPUからの命令によって、外部環境に対する制御を行う。例えば、パルスモータなどの駆動装置、温度調節器などである（図表Ⅳ－5を参照)。また、通信データ入力機器を動作させるために通信データを出力することもある。

・ 出力装置に対してアナログ量（電圧、電流）の必要な場合は、処理装置の出力をD／A変換してから出力する。詳しくは次節で説明する。

④ 構成例

ハードウェア構成の事例として簡単なFAX装置を紹介する（図表Ⅳ－6)。

主要な構成要素の役割を以下に示す。

ア．FAXモデム

FAXデータの送信と受信を一台の装置でおこなう。送信と受信を同時に行うことができない。

受信の場合、１バイト受信するごとにCPUにハードウェア割込みを発生させる。

送信の場合、モデムに１バイト書込むとモデムが回線に送信し、送信終了後CPUにハードウェア割込みを発生させる。

イ．イメージセンサ

送信原稿を走査し、１走査線分（１ドットライン）のイメージデータを読取る。

CPUから必要に応じてこのデータを読みにいく。

ウ．原稿検出

送信原稿を読取り部に挿入すると、ハードウェア割込みを発生させる。

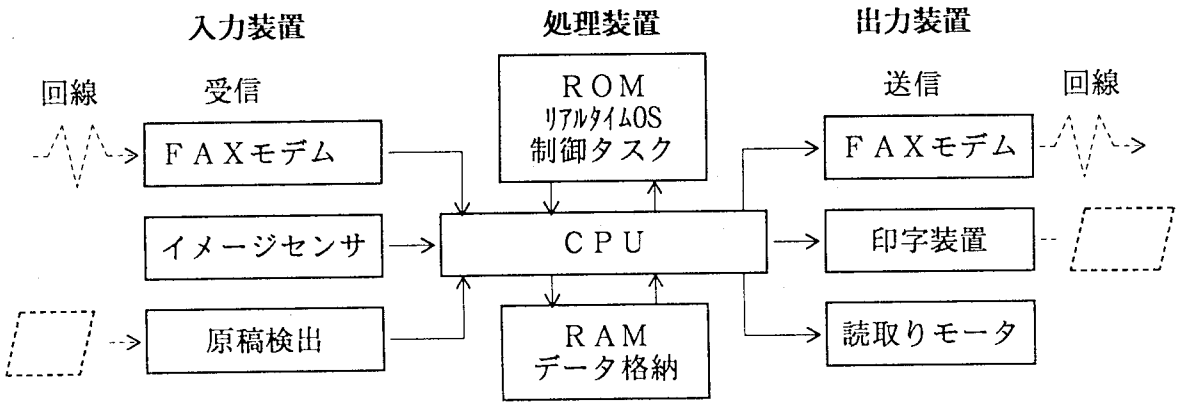
エ．読取りモータ

送信原稿をイメージセンサで読取るために、モータで原稿を移動させる。

オ．印字装置

受信データを印刷する。

図表Ⅳ－６　FAXの構成例



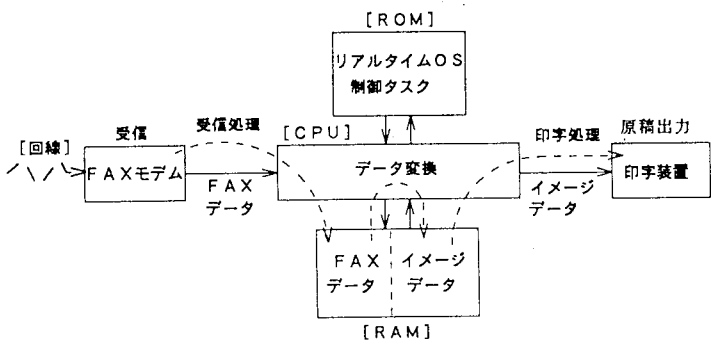
（４）　応用ソフトウェアの機能と動作

(3)項と同じ事例を用いながら、ROMに書かれているソフトウェアがどのような機能を果たし、どのように動作するかを説明する。

①　受信処理機能

FAXモデム（入力装置）が１バイト受信すると、CPUに外部割込みを発生させる。CPUは、外部割込みによりFAXモデムから１バイト分のデータを読み込み、RAMの受信データ格納領域へ蓄積する。この処理を最後のデータがくるまで繰り返す。受信がすべて完了すると、RAMに蓄積したFAXデータ（画像の圧縮データ）を復元し、そのデータを印字装置へ送り、印刷する（図表Ⅳ－７）。

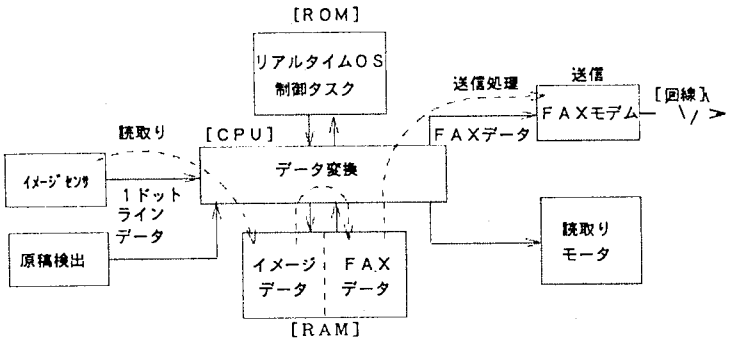
図表Ⅳ－７　FAXデータの受診処理



② 送信処理機能

送信原稿をFAXの読取り部に挿入するとCPUに受信処理と同様に外部割込みが発生する。CPUは、イメージセンサから1ドットライン分のイメージデータを読込み、そのデータをRAMのイメージデータ格納領域に蓄積し、読取りモータを1ドットライン分進める。この処理を全原稿を読み終えるまで繰り返す。読み込みがすべて完了するとイメージデータをFAXデータ（画像の圧縮データ）に変換し、FAXモデムに1バイト書込む。続いて、FAXモデムが回線に1バイト送り終えると外部割込みが発生するので、CPUは次のFAXデータを1バイト書込むという処理を繰り返す（図表Ⅳ－8）。

図表Ⅳ－8　FAXデータの送信処理



③ ソフトウェアの構造

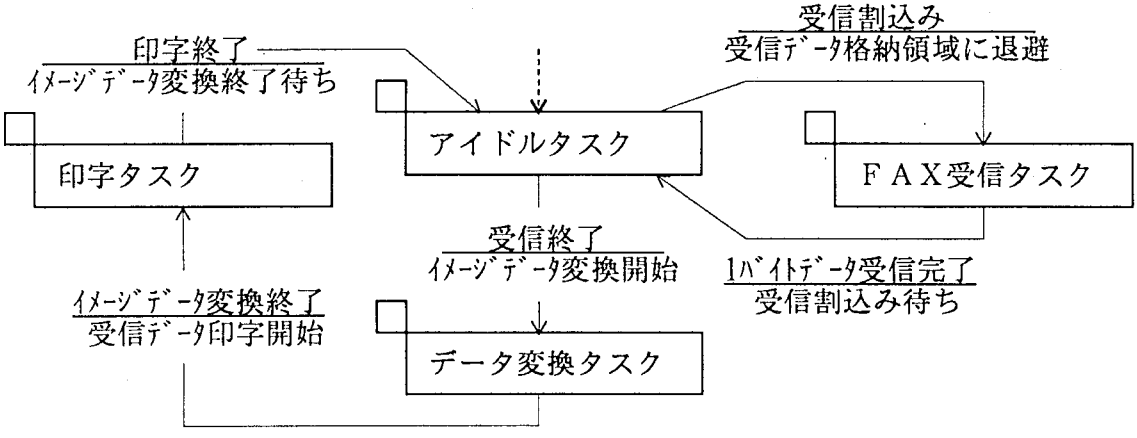
ここで、上記二つの受信処理と送信処理の実際上の動きがどのようになるかを考慮し、ソフトウェアの有り様を考えてみる。

二つの処理を考えたとき、FAXモデムが1台という理由から同時に起こり得ないのはFAXモデムの送信と受信だけで、その他の入力装置、出力装置やCPUで行うデータ変換などは、同時に進行させるべきである。例えば、FAXデータの受信中は受信割込みが数ミリセカンドオーダで発生するので、この発生間隔の空き時間に原稿のイメージデータを読込み、RAMに格納する処理が可能である。

このような処理を「割込み処理」だけで記述すると、上記の受信処理と送信処理の流れが混じり合い非常に複雑になる。このような場合、リアルタイムOSを前提に処理を表現する、FAX受信タスク、FAX送信タスク、印字タスク、原稿読取りタスク、データ変換タスク（FAXデータとイメージデータの相互変換）とに分けて作成すると、動作がわかり易くなる。プログラムを作成する側では、タスク毎に独立した処理で並行して動作すると考えてプログラムを作成すればよい。

この制御の流れの一部は、図表Ⅳ－9の状態遷移図のようになる。

図表Ⅳ－9　　FAX機能の状態遷移図例



2．リアルタイム制御処理の入出力インタフェース

コンピュータで制御を行う場合、制御対象となるＩ（Input）／Ｏ（Output）装置との間で自由にデータ（例えば、I／O装置に対する命令や、装置が返す処理結果である）の授受ができる必要がある。

このためにCPU装置とI／O装置との間には約束事を設けている。例えば、ハードウェアの物理的な形状、電気的な特性、信号線の**ハンドシェイク方法**（信号受け渡しのタイミングを知らせるための制御方法）、I／O装置に対する命令体系などさまざまな段階で存在する。そして、これらの約束事を総称して**入出力インタフェース**とよんでいる。

この節では、CPUアーキテクチャや**システムバス**に依存していくつもある入出力インタフェースの実現方式の一例を説明していく。

（1） 入出力機構

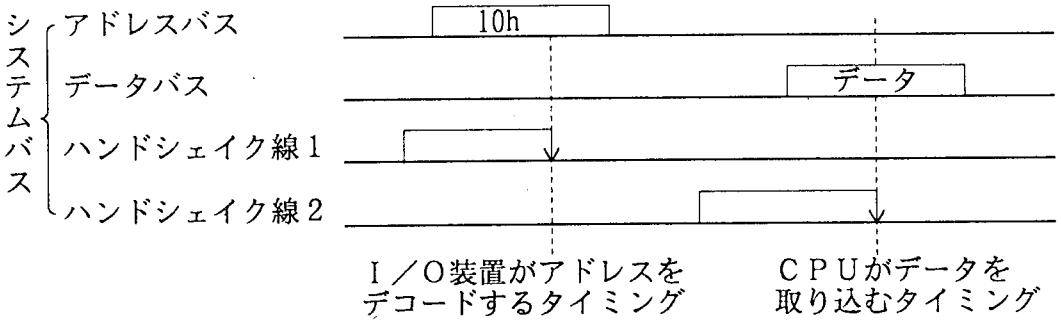
制御対象の状態変化をコンピュータに取り込むまでの過程、コンピュータが出す命令が制御対象に伝えられる過程について説明する。

CPUのI／O装置に対する命令を実行するためには、I／O装置のとの間でデータを授受するための信号線（**データバス**という）と、いくつかある場合のI／O装置を選択するための信号線（**アドレスバス**という）が必要となる。各信号線数は「サイズとか幅」とよばれ、８本や、16本や、あるいは32本のものが現在のところである。

データバス幅が８本の場合は一度に８ビットのデータが転送できる意味となり、アドレスバス幅が16本の場合はアクセス領域が64kバイトあることを意味する。

そこで、図表Ⅳ－10で入出力動作を大まかにみておこう。

図表Ⅳ－10　入力動作の簡単な例



出典：第二種共通テキスト情報処理システム　中央情報教育研究所

［ここでの約束事（システムバスインタフェース）］

① CPUは入力動作をしたい場合、ハンドシェイク線１をアクティブにしてから、アドレスバスにＩ／O装置のアドレスを出力する。

② I／O装置は、ハンドシェイク線１がアクティブになると、アドレスデコード（装置番号解読）を開始し、立ち下がりのタイミングでアドレスバスに出力されているアドレスを解読する。

③ I／O装置は、アドレスをデコードした結果、10hであれば、自分のアドレスと認識し、ハンドシェイク線２をアクティブにしてから、データバスにデータを出力する。

④ CPUはハンドシェイク線２がアクティブになった場合、その立ち下がりのタイミングでデータバスに出力されているデータを取り込む。

・ 実際のI／O装置の構成例について説明を補足しておく。磁気ディスクやMT装置などの機械的な装置が直接CPUに接続されているわけではなく、**コントローラ**とよばれるI／O装置制御用のIC（つまり、本来ならI／O装置制御用のチップとかデバイスとよぶ方が適切である）であることが多い。

（2） A／D変換処理とD／A変換処理

ここでは、コントローラと制御対象のＩ／Ｏ装置間の過程について説明をする。

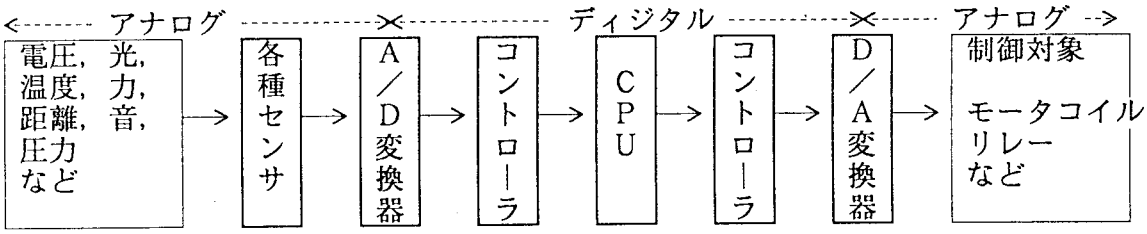
I／O装置がディジタル量（０と１のデータ）で制御できる場合は、CPUからコントローラに送られたディジタル信号をそのままI／O装置に透過させるだけでよい。

ところが、現実の自然界はアナログ中心の世界である。電圧や温度、光、力、距離、音などの物理量はすべてアナログ量となっている。このため、アナログ量で制御したい場合には、コントローラに送られるディジタル量をどこかでアナログ量に変換させなければならない。この変換をD／A変換（Digital to Analog conversion）という。

実際の複雑な制御としてエアコンの例で上述の過程を見てみる。エアコンは、室内の温度変化に応じてモータの回転数を制御していくことになる。具体的には、まず温度センサ（入力装置）によって検出されたアナログ量はD／A変換と逆の現象であるA／D（Analog to Digital conversion）変換によってディジタル量に変換し、CPUが解釈できる０と１の信号にする。CPUは、入力されたデータに応じて、回転数を制御するためのデータをコントローラに送り、電圧量というアナログ量への変換を通してモータを制御する。

整理すると、リアルタイム制御処理システムの典型的なパターンは、センサが検出したアナログ量をA／D変換によってCPUに取り込み、CPUで取り込んだデータを演算し、演算結果に応じた制御データをコントローラに送り、D／A変換を経て制御対象を動かすことである（図表Ⅳ－11）。

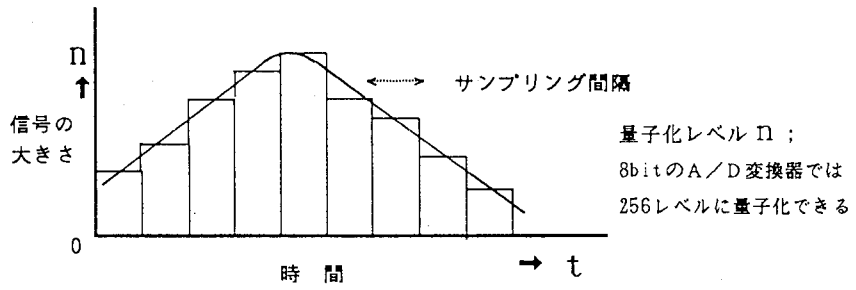
図表Ⅳ－11　リアルタイム制御処理システムのデータの流れ



・　A／D変換の処理について簡単に説明する。標本化（サンプリング）と量子化という２つ概念で説明される（図表Ⅳ－12）。前者は、連続したアナログ量をディジタル量にするために、信号を適当な時間をおいて取り出す処理をいう。後者は、取り出した信号を最も近い値に近似して置き換える処理をいう。この近似値化のレベルをビットを使って表し、８ビットとか12ビットのA／D変換器といった表現を使用する。

D／A変換は、A／D変換と逆の処理を行っている。

図表Ⅳ－12　標本化と量子化



（3） 割込み処理機構

ここでは、割込みを用いた入出力について説明する。

CPUとI／O装置制御用のコントローラとの間では、更に割込み処理とよばれる動作がハードウェアによって実現されている。まず、割込み処理が実現されている理由を二つ述べておく。

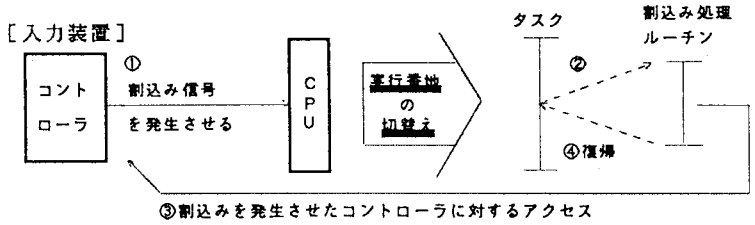
① CPUがコントローラに対して動作指示を送ってからI／O装置の動作が終了するまで、当然時間がかかる。この間、CPUがほかの処理が実行できれば、システムのスループットの向上が期待できる。

② 緊急性の大きい処理は、現在の処理を中断してでも素早く行う必要がある。リアルタイム制御処理システムに対して、迅速な応答が求められれば、CPUのコントローラに対するアクセスも時間遅れなしに行う必要がある。これらの要求を解決するために、I／O装置の状態変化をコントローラ側からCPUに対して非同期に通知できるようにしたものが割込みである。

割込み処理のおおまかな動作を図表Ⅳ－13に示す。

・ コントローラが割込みを発生させてからCPUが実行番地を「割り込み処理ルーチン」に切り替えるまでの処理がハードウェアで実現されており、その時間は無視できるほど小さいことである。

図表Ⅳ－13 割り込み処理の動作

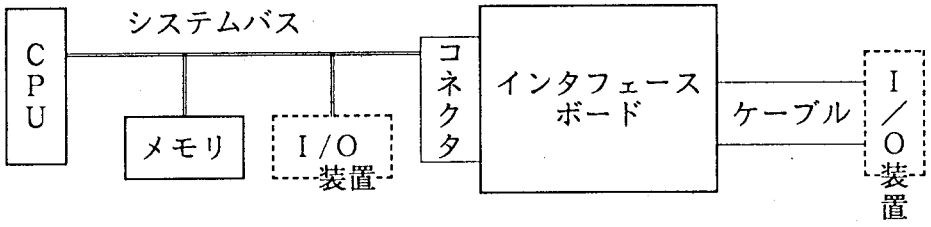


（4） システムバスとインタフェースボード

(1)節で説明した入出力インタフェースとしてのシステムバスを拡張性を考慮して、基板レベルで追加や交換できるような概念のものが**インタフェースボード**である。

実現されているインタフェースボードの機能構成は、図表Ⅳ－14のような形態となっている。図表中のコネクタ部分で抜き差しが可能となっており、制御が必要となるI／O装置に応じたインタフェースボードを付け替えるだけで、さまざまな状況のリアルタイム制御処理システムの構成に対応できるようになる。

図表Ⅳ－14 システムバスとインタフェースボード



（5） 標準インタフェース

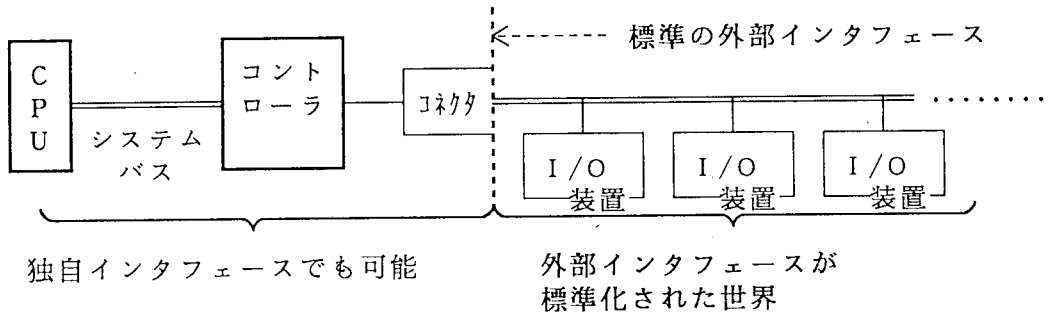
IEEE－488（GPIB）を例にとって、標準インタフェースの必要性、その使い方について説

明する。

前述のシステムバスのインタフェースが標準化されていれば、**標準インタフェース**に基づいたハードウェアの互換性が実現され、再利用性が高くなり経済性がよくなる。

このような背景から、現在では図表Ⅳ－15に示すような観点から外部に標準化されたインタフェースを持つ方法が考え出された。この**外部インタフェース**の概念による例では、RS－232Cやセントロニクス、SCSI（Small Computer System Interface）、IEEE－488など多様なI／O装置に応じた種々のものがある。

図表Ⅳ－15　外部インタフェースの概念



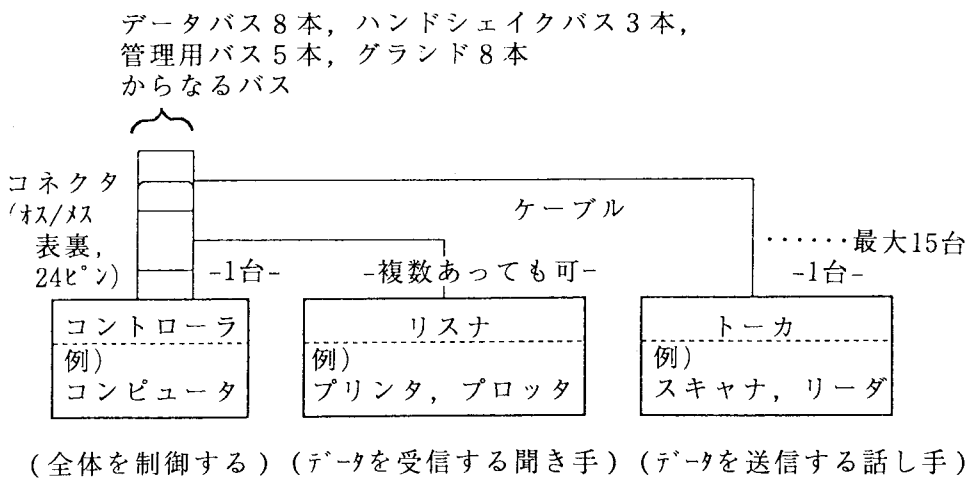
さまざまな約束事が定められているので、ＩＥＥＥ－４８８を例にその概要を述べる。

８ビットのパラレルインタフェースであり、最大15台のI／O装置が接続できる。システムバスの最大長は20m以内で、I／O装置間との間は４m以内と決められている。個々のI／O装置には０～31までの装置番号を付与し識別する。これらのI／O装置間でデータ転送を実現するために、コントローラ、リスナ、トーカという機能が割り当てられる（図表Ⅳ－16）。コントローラやトーカは１台に限られるが、１台でトーカとリスナになれるので送受信も可能になる。

注意すべきことは、IEEE－488にせよ、RS－232Cにせよ、「相手との間でデータ転送が行える」というレベルの規格であり、転送するデータの内容そのものに関しては一切規定していないことである。

例えば、IEEE－488をインタフェースとして持つプリンタにしても、メーカが違うと論理的なコマンドレベルのデータ（プリンタ自体の制御コード）が異なるというケースが多い。このような場合、プリンタを制御するためのソフトウェアドライバを作成すればよい。

図表Ⅳ－16　IEEE－488（GPIB）のシステム構成例



出典：第二種共通テキスト情報処理システム　中央情報教育研究所

3．リアルタイム制御処理のオペレーティングシステム

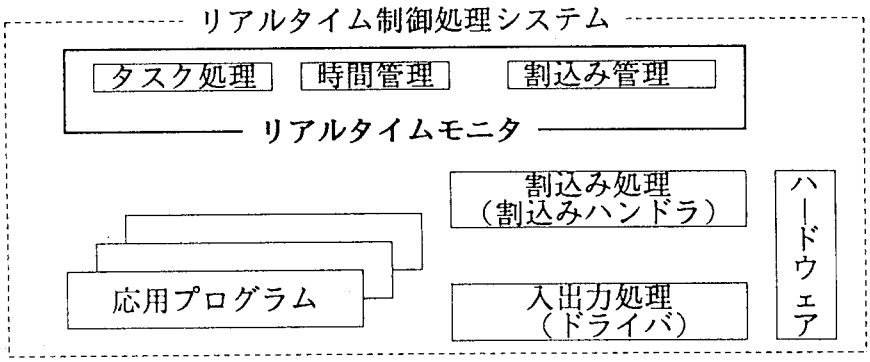
（1） リアルタイム制御処理システムにおけるモニタ機能

マイクロコンピュータを利用したリアルタイム制御処理システムのオペレーティングシステムにつて説明する。

マイコンには簡単なモニタもないことがある。しかし、ソフトウェアコストを削減するには、管理機能を担当するプログラム（図表Ⅳ－17）を応用プログラムから切り離してつくる方が賢明である。

- この管理機能のプログラムはリアルタイムモニタ、またはリアルタイムオペレーティングシステム（リアルタイムOS）などと呼ばれている。
- 両者の区別には明確な定義はない。一般に比較的小規模で必要最小限の機能を有した簡易なものに対してリアルタイムモニタとよんでいる。

図表Ⅳ－17　リアルタイム制御処理システムのソフトウェア構成例



リアルタイム制御処理システムでは、独立した複数の入出力装置から発生する外部事象を、決められた時間内に決められた処理を行い出力を行わなければならない。これを実現する応用プログラムは複雑となる。

管理機能部をもつリアルタイムモニタを使えば、①応用プログラムの複雑化を回避しながら、複雑な処理を扱うことができ、②プログラム全体の見通しがよくなり、③開発効率も向上させることができる。

- 複数のタスク制御だけができる（マルチタスク）単なるマルチタスクモニタとリアルタイムモニタの差異は、実行中のタスクの優先度よりも発生事象を待つタスクの優先度が高いならば、即座にタスク切替えが行われる点である。

（2） リアルタイムオペレーティングシステム

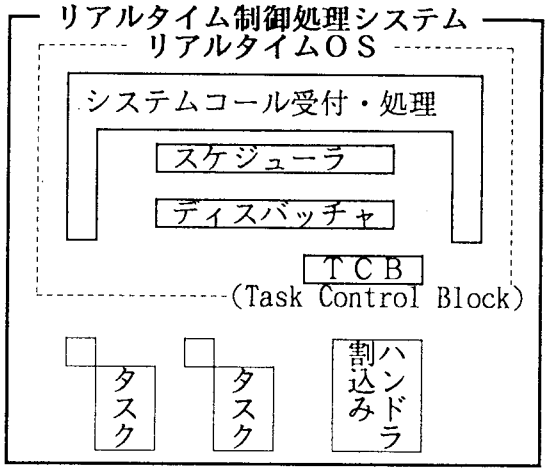
タスクの状態遷移、タスクの切替え、タスク間の同期について、具体的なリアルタイムモニタの下に説明する。また、そのシステムコールについても説明する。

具体例は図表Ⅳ－18～21、24に示す仕様のものを取り上げる。仕様の中にでてくる新しい用語の説明は、随時、関係の深いところで触れる。

図表Ⅳ－18　リアルタイムモニタの仕様

項　　目	仕　　様
管理できるタスク数	32個
優先度	0～31の32個
タスクの状態	RUN、READY、WAITの3状態
管理可能なイベント数	16個
スケジュール方式	優先度制御によるイベントドリブン方式

図表Ⅳ－19　リアルタイム制御処理システムのソフトウェア構成例



① タスクの状態遷移

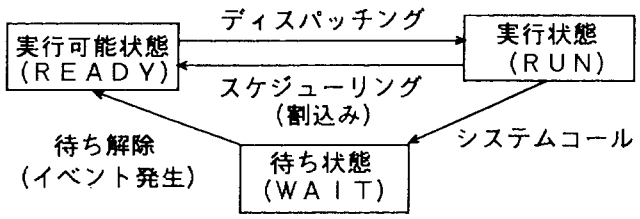
タスクの状態定義（図表Ⅳ－20）に従って、タスクが特に入出力などの動作完了待ちになると、実行可能な中で優先度の高いタスクをリアルタイムOSが選び出して実行する。また、入出力動作完了後は、元のタスクは再び実行可能状態の待ちとなる。

図表Ⅳ－20　タスクの状態

タスクの状態	説　　明
実行可能状態（READY）	タスクは実行する条件が整っているが、このタスクより優先度の高いタスクが実行中のために、実行権が与えられず、実行されていない状態
実行状態（RUN）	現在、実行権が与えられて実行中の状態。実行状態のタスクは一つしか存在できない。
待ち状態（WAIT）	実行するための必要条件が整っていないため実行できない状態。イベントの発生を待っている状態

このようなタスクの状態の変化（状態遷移）を繰り返している（図表Ⅳ－21）。
この状態数はマルチタスク実現のため基本的なものだけである。

図表Ⅳ－21　タスク状態遷移

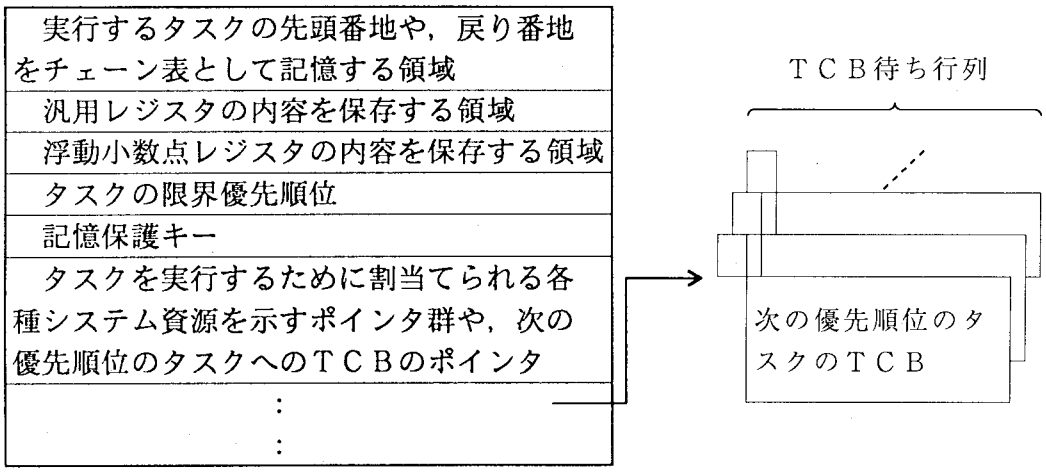


② タスクの切替え

タスクが状態遷移する要因は、タスク自体が呼び出すシステムコールである。また、直接タスクの状態を変化させるのは、リアルタイムOS部のスケジューラとディスパッチャ機能である。

・タスクの状態に関する諸情報はタスク毎のTCB（Task Control Block）とよばれるメモリ領域に格納されて管理される（図表Ⅳ－22）。

図表Ⅳ－22　TCBの主要な管理情報



・スケジューラは、TCB情報を下に、実行可能な状態であるタスクの中から実行すべきタスクを決定し、そのTCBをディスパッチャに渡す。

・ディスパッチャは、現在実行中のタスクの情報をそのTCBに保存する。次いで渡されたTCBに保存されている情報を取り出してCPUレジスタなどに設定し、そのタスクの実行を再開する。いわゆるタスクにCPUを与えることであるが、一般にはスケジューラと明確に分けて使われることは少ない。

スケジューラの決定アルゴリズムには、次の二つの代表的なものがある。

ア．優先度（プライオリティ）方式

予めタスクに優先度を決めておき、一番優先度の高いタスクを選択する方式である。

イ．ラウンドロビン方式

すべてのタスクが平等に選択される方式である。

リアルタイム制御では、発生事象に優先度があり、事象発生時に動作すべきタスクも発生した事象に対応したものでなければならないため、優先度方式が適している。

スケジューラ機能自体を実行する契機の与え方にも、次の二通りの方法がある。

ア．タイムスライス方式

一定時間ごとに、周期的に起動する方式である。

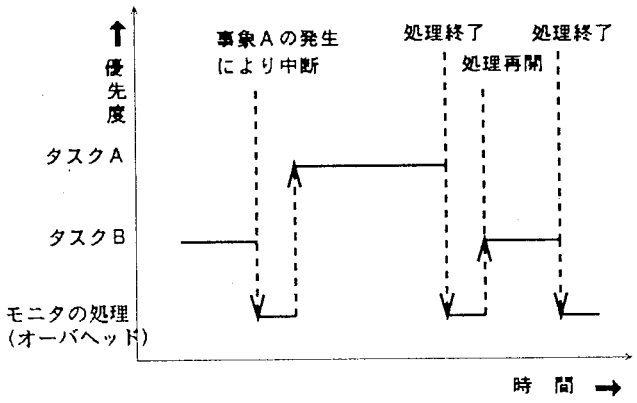
イ．事象駆動（イベントドリブン）方式

何らかの事象が発生し、タスクの状態に遷移が発生したときに起動する方式である。

リアルタイム制御では、ある事象が発生すると対応するタスクに速やかに切替える必要があるため、イベントドリブン方式が適している。

ここで、例題として、A，B二つのタスクについて、仕様にある優先度制御のイベントドリブン方式によるタスクの切替えの様子を示す（図表Ⅳ－23）。

図表Ⅳ－23　タスクの切替え



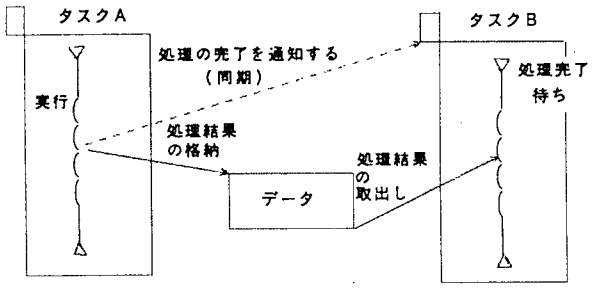
出典：第二種共通テキスト情報処理システム　中央情報教育研究所

③　タスクの同期

複数のタスクが連係した処理を実現する場合、タスクの待合わせによる同期をとる必要がある。

ここで、例題として、A，B二つのタスクの間に、この順序で処理結果の受け渡しが必要であるとする。この場合の連係処理の様子を示す（図表Ⅳ－24）。連係処理に当たって、あるタスクから他のタスクへデータを受け渡すことを特に**タスク間通信**とよぶ。

図表Ⅳ－24　タスクの同期



(出典:大原茂之著:「実践リアルタイムプログラミング技法」,オーム社,1991年)

④　システムコール

リアルタイムOSのいろいろな機能をタスク側で使うことによって、プログラミングはずっと容易になる。OSの機能は、目的のサービスの呼びだしといわれる**システムコール**という形で提供され、タスクで利用される。

・　システムコールは**ファンクションコール**とか**スーパバイザコール**とも呼ばれる。

ここで、例題として、タスクの同期をとる時に使用される**イベントフラグシステムコール**を取り上げる。この方法のメカニズムは次のようになっている。

ア．イベントフラグ（16ビット領域）と称する０と１の２値を持つものを使用する。

イ．各ビットは一つのイベントに対応しており、その割当はシステム設計時の取決めによる。複数タスクとの連係処理を記述できる。

イベント用システムコールの種類と機能の詳細例を示す（図表Ⅳ－25）。次節のプログラムの動作を理解するときにも参照する必要があることを触れておく。

図表Ⅳ－25　イベントフラグシステムコールの説明

（ただし、C言語インタフェースが前提となっている。）

	名　　称	機　　能
1	イベントフラグの生成	void creat__flag（short id） id：イベントフラグID 指定されたIDのイベントフラグ（16ビット）の領域をモニタ内に生成し、0で初期化する。
2	イベントフラグの削除	void delete__flag（short id） id：イベントフラグID 指定されたIDのイベントフラグを削除する。
3	イベントフラグ待ち	short wait__flag（short id, unsigned short flag、 wmode mode） id　：イベンドフラグID flag　：待合せイベントパターン mode　：待合せ条件（OR／AND） 指定されたIDのフラグが、待合せイベントパターンになるまで待つ。 待合せの条件として、いずれかON（＝OR）、すべてON（＝AND）を指定する。 ONに設定されたビットは、イベントフラグのリセットを行うまでリセットされない。 リターン値は、指定されたIDのイベントフラグの内容である。
4	イベントフラグのセット	void set__flag（short id, unsigned short flag） id　：イベントフラグID flag　：設定するイベントのビットパターン ビットが1ならばセット、0ならばnop、指定されたIDのフラグ（領域）を指定されたflagパターンでORする。
5	イベントフラグのリセット	void clear__flag（short id, unsigned short flag） id　：イベントフラグID flag　：リセットするイベントのビットパターン ビットが1ならばリセット、0ならばnop、指定されたIDのフラグ（領域）に対して、指定されたflagパターンのONになっているビットをリセットする（EORする。）。

4．応用プログラムの構造

（1） リアルタイム制御の典型的な処理手順

制御対象の状態変化を入力して、何らかの演算を行い、適切な指示を与える（制御情報を出力する。）というリアルタイム制御の典型的な処理手順を例題を引用し説明する。

自動車の自動定速走行制御システムの簡単なモデルの作成を考えてみる。マイクロコンピュータは制御対象である車のエンジン部に組み込まれている。－定速走行装置部がそれに当たる（図表Ⅳ－１のモデルの構成を参照）。

① システムの概要

簡易モデルへの要求仕様は次の通りである。

ア．モデルシステムの入力・出力装置は図表Ⅳ－26に示す。

イ．発進させ所定の速度まで加速した後、定速走行ON／OFFスイッチを押すと、その時点での速度（速度センサから）とガソリン供給量（アクセルペダルの変位量から）を記憶し、一定速度で自動を走行させる。

ウ．定速走行中、坂等で速度が変化した場合には、ガソリン供給ポンプを制御して一定速度を保つようにする。このために、定速走行装置は、一定時間間隔で速度センサから速度情報を受け取る。

エ．一定速度で走行中、ブレーキペダルを踏むか、再度、定速走行ON／OFFスイッチを押下すると、定速走行モードは解除される。

オ．スピードメータは、一定時間間隔で速度センサから速度情報を検出し、現在の車速を次の速度情報検出までディジタルで表示し続ける。

図表Ⅳ－26　入出力装置の構成概要

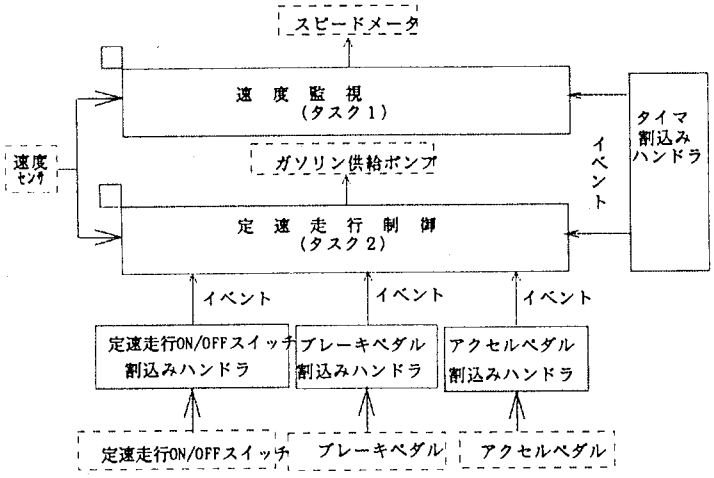
a．入力装置	・速度センサ ・定速走行ON／OFFスイッチ ・ブレーキペダル ・アクセルペダル
b．出力装置	・ガソリン供給ポンプ ・ディジタルスピードメータ

出典：第二種共通テキスト情報処理システム　中央情報教育研究所

② システムの分析

この簡易モデルの要求仕様から内部設計仕様を図表Ⅳ－27～30のように考える。

図表Ⅳ－27　制御系統とタスクの割当て



出典：第二種共通テキスト情報処理システム　中央情報教育研究所

図表Ⅳ－28　入力装置の機能要件

名　　称	機　　能
・速度センサ	タイヤの回転状態を検出し、現在の車速を得ることができる装置である。タスクから入力を行うことによって車速を獲得できる。
・定速走行ON／OFFスイッチ	スイッチを押すごとにハードウェア割りみが発生する。
・ブレーキペダル	ブレーキペダルを踏むと、ハードウェア割込みが発生する。
・アクセルペダル	アクセルペダルを踏むと、ハードウェア割込みが発生する。アクセルペダルの踏込み量がA／D変換されるため、タスクで読み取ることによって、その時点でのアクセル踏込み量を取得することができる。

出典：第二種共通テキスト情報処理システム　中央情報教育研究所

図表Ⅳ－29　内部処理タスクの機能定義

名　　称	機　　能
・速度監視 （タスク１）	タイマ割込みハンドラから、タイマイベントによって一定時間ごとに通知されるイベントを受け取り、速度センサから速度の取得を行う。スピードメータに速度を出力後、再びタイマイベント待ちになる。
・定速走行制御 （タスク２）	このシステムの要であり、ON／OFFスイッチを押すことで開始し、再びON／OFFスイッチが押されるまで、ガソリン供給ポンプの制御を行って、一定の速度を保つ。 ON／OFFスイッチは前述したように単にハードウェア割込みを発生させるだけなので、押されたON／OFFスイッチの押下を開始と見なすか終了と見なすのかは、内部でステータス管理を行っている。定速走行装置は、現在の速度が40km/hから120km/hの間で動作するため、速度範囲外の「ON」指示は無視される。坂道の登り降りで車速に変化が出た場合は、ガソリン供給ポンプの制御によって、加速、減速を行う。このため、定速走行中は一定時間ごとに車速情報を受け取る。速度制御は、ガソリン供給ポンプの制御だけ（エンジンブレーキまたは加速）で行い、急な下り坂で速度が上がりすぎてもブレーキ動作は行わない。 定速走行は、 ①　ブレーキペダルが踏まれる。 ②　ON／OFFスイッチが押される。 ③　急な上り坂、下り坂で車速が40km/hから120km/hの範囲外になる。 の条件で終了する。

出典：第二種共通テキスト情報処理システム　中央情報教育研究所

図表Ⅳ－30　割込みハンドラ定義一覧

	名　　称	機　　能
1	タイマ割込みハンドラ	一定時間おきに発生するタイマ割込みを受け取り、「速度監視タスク」と「定速走行制御タスク」にタイマイベントを通知する。
2	定速走行ＯＮ／ＯＦＦスイッチ割込みハンドラ	定速走行ON／OFFスイッチによるハードウェア割込み処理を行い、その結果を「定速走行制御タスク」にイベントとして通知する。
3	ブレーキペダル割込みハンドラ	ブレーキペダルが踏まれたことによるハードウェア割込み処理を行い、その結果を「定速走行制御タスク」にイベントとして通知する。
4	アクセルペダル割込みハンドラ	アクセルペダルが踏まれたことによるハードウェア割込み処理を行い、その結果を「定速走行制御タスク」にイベントとして通知する。

出典：第二種共通テキスト情報処理システム　中央情報教育研究所

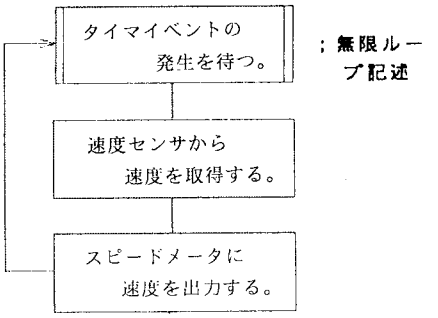
（2） マルチタスク構造のリアルタイム応用プログラムの例

(1)項の例題モデルに対して、マルチタスク構造の簡単なリアルタイム応用プログラム作成して見せる。

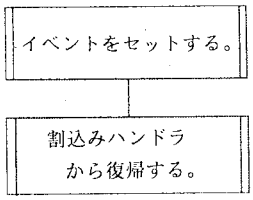
まず、簡易モデルのプログラム設計仕様として図表Ⅳ－31～33を説明する。その上で、Ｃ言語によるプログラム例を見せる。

- Ｃ言語プログラムの基本構成や特定の用語ついて最小限の説明をする。
- システムコールについて、改めて図表Ⅳ－25で確認する。
- 割込みハンドラは、装置ごとに存在し、タイマ割込みハンドラを含め４種類が存在する。

図表Ⅳ－31　速度監視（タスク１）の流れ図

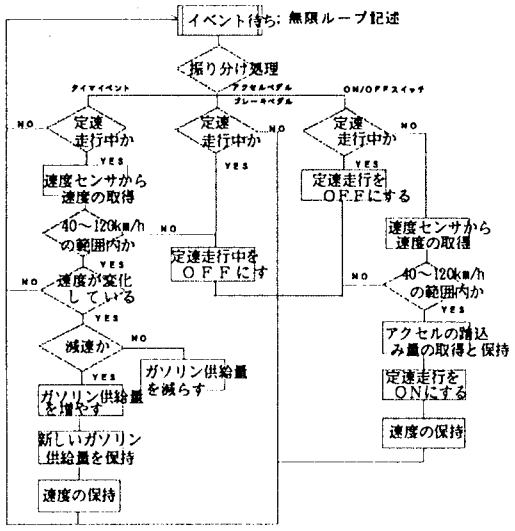


図表Ⅳ－32　割込みハンドラの流れ図



出典：第二種共通テキスト情報処理システム　中央情報教育研究所

図表Ⅳ－33　定速走行制御（タスク２）の流れ図



出典：第二種共通テキスト情報処理システム　中央情報教育研究所

＜定速走行制御(タスク２)のプログラム＞　；Ｃ言語コード

```
/******************          定義文              ******************/
#define BOOL    short
#define FALSE   0
#define TRUE    1
#define EVENT   7
/******************  関数のプロトタイプ宣言      ******************/
void task2( void );
/******************          タスク本体          ******************/
void task2( void )
{
    BOOL bASCFlag = FALSE;      /*定速走行フラグ(初期値はOFF)      */
    int  uSpeed, uASCSpeed;     /*スピードを代入する変数           */
    int  uAccel;                /*アクセルの踏込み量を格納する変数*/
    short sEvent;               /*イベント値                       */

    create_flag( EVENT );       /*イベントの作成(ｲﾍﾞﾝﾄ用ｼｽﾃﾑｺｰﾙ)  */
    for( ;sEvent = wait_flag( EVENT, 7, OR ); )
    {
        if( sEvent & 1 )        /*タイマイベント                   */
        {
            if( bASCFlag )      /*定速走行中かどうか調査           */
            {
                uSpeed = get_speed();/*速度センサから速度の取得    */
                if( uSpeed >= 40 && uSpeed <= 120) /*速度が40～    */
                                                   /*120km/hの間か*/
                {
                    if( uSpeed != uASCSpeed )
                    {
                        if( uSpeed > uSCSpeed )    /*加速している */
                            Control_pump( --uAccel); /*減速処理   */
                        else                       /*減速している */
                            Control_pump( ++uAccel); /*加速処理   */
                    }
                }
            }
            create_flag( EVENT, 1 );  /*イベントのクリア          */
        }
        else if( sEvent & 2 )         /*ブレーキ/アクセル         */
        {
            if( bASCFlag )            /*定速走行中かどうか調査    */
                bASCFlag = FALSE;     /*定速走行フラグをOFFにする */

            clear_flag( EVENT, 2 );   /*イベントのクリア          */
        }
        else if( sEvent & 4 )         /*ON/OFFボタン              */
        {
            if( bASCFlag )            /*定速走行中かどうか調査    */
                bASCFlag = FALSE;     /*定速走行フラグをOFFにする */
            else
            {
                uSpeed = get_speed();  /*速度センサから速度の取得  */
                if( uSpeed >= 40 && uSpeed <= 120 ) /*速度が40～   */
                                                    /*120km/hの間か*/
                {
                    uAccel = get_accel();/*アクセルの踏込み量の取得*/
                    bASCFlag = TRUE;     /*定速走行フラグをONにする*/
                    uASCSpeed = uSpeed;  /*速度の保持              */
                }
            }
            clear_flag( EVENT, 4 );  /*イベントのクリア           */
        }
        else                         /*その他のイベント           */
            clear_flag( EVENT, 0xfff8 ); /*イベントのクリア       */
    }
} /******************          タスク２終了        ******************/
```

＜速度監視(タスク１)のプログラム＞　；C言語コード

```
01 /*******************          定義文                ********************/
02 #define OR           0
03 #define EVENT_TIMER1 1
04 /*******************  関数のプロトタイプ宣言        ********************/
05 void task1( void );
06 /*******************          タスク本体            ********************/
07 void task1( void )
08 {
09    int uSpeed;                                    /*スピードを代入する変数*/
10
11    create_flag( EVENT_TIMER1 );                   /*イベントの作成          */
12                                                   /*      (ｲﾍﾞﾝﾄ用ｼｽﾃﾑｺｰﾙ)*/
13    for(;;)                                        /*無限ループ記述          */
14    {
15       wait_flag( EVENT_TIMER1, 1, OR );           /*イベント待ちを行う      */
16       uSpeed = get_speed();          /*速度センサから速度を取得する*/
17       out_speed( uSpeed );           /*スピードメータに速度を出力  */
18       clear_flag( EVENT_TIMER1, 1 );/*イベントのクリア            */
19    }
20 } /*******************          タスク１終了          ********************/
```

＜割込みハンドラのプログラム＞

```
01 /*    割り込みハンドラプログラムは，装置毎（定速走行ＯＮ／ＯＦＦス*/
02 /*  イッチ割込み，ブレーキペダル割込み，アクセルペダル割込みの３種*/
03 /*  類）に存在するが，処理が同じため一般化例を一つ示すことにする．*/
04 /*******************  関数のプロトタイプ宣言        ********************/
05 void Interrupt_HandlerX( void );
06 /*******************          ハンドラ本体          ********************/
07 void Interrupt_HandlerX( void )
08 {
09    set_flag( X, X );                    /*X=個別のイベント番号となる*/
10    return_interrupt();                  /*割込みからリターンする    */
11 } /*******************          ハンドラ終了          ********************/
```

（3）　リアルタイム応用プログラム例の動作

リアルタイムOSの下で上記(2)項の簡単なプログラムが、どのように実行されるかをトレースする演習を行う。

機能の「まとまり」からタスク分割された二つの速度監視タスク（タスク１）と定速走行制御タスク（タスク２）の間の優先順位の必要性について説明する。(機能につては、改めて図表Ⅳ－30を参照)

① タスク１は現在速度を人が認識するためにディジタル表示する。そのためタイマイベントは３～５秒間隔で発生させることでよい。仮に、タイマイベントが多少遅れてもシステム全体では危険になるものではない。

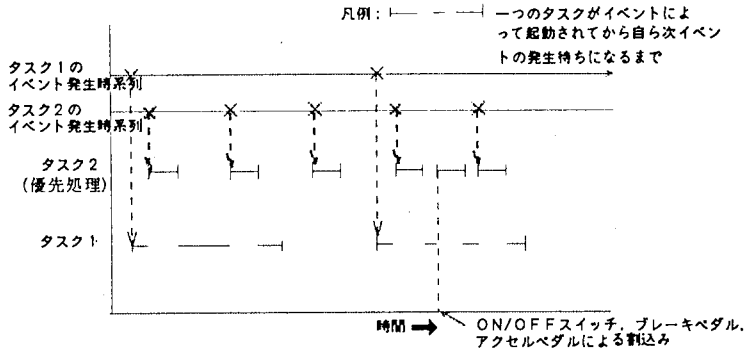
② タスク２は、システムの要となるもので、定速走行するようにガソリンの供給量を制御する。タスク１での表示間隔の間に極端な速度変動があってはならない（モデルの目標である)。そのことから、タスク２のタイマイベントの周期はタスク１のそれよりも小さくなる。

危険回避のため、ブレーキペダル、アクセルペダル、ON／OFFスイッチが押されたときは、緊急に自動定速走行システムを解除しなければならない。

③ これらより、タスク２はタスク１より優先順位を高くする必要がある。

④ 優先順位を使い、タスク２に対するイベントが発生するとタスク１が動作中の時でも、割り込んで処理を開始する－即時処理をすることを説明する。その間タスク１は処理を中断される（図表Ⅳ－35)。

図表Ⅳ－34　タスクの動作図例



出典：第二種共通テキスト情報処理システム　中央情報教育研究所

指導上の留意点

1．本章の説明にあたっては、この章に割けれる講義時間が短いことを踏まえ、他章との先修条件を十分考慮し、重複を避け簡潔に説明をすることが肝要である。

具体的な先修条件としては以下のような項目を考慮にいれるとよい。

・コンピュータの仕組み（コンピュータの構成要素　入出力装置）

・ソフトウェアの基礎（オペレーティングシステム）

・プログラミング能力（プログラム言語Ｃ入門程度）

また、本章を学習後、更に学習を進めていくことが望ましい項目にも触れておく。

・マイクロコンピュータ応用システム設計の基礎能力、特に、入出力機能の活用技法、論理的なインタフェースの設計、物理的なインタフェースの設計である。

2．タスクを基本単位としたOSの動作、リアルタイム応用プログラムの構造と動作に力点を置いて指導する。

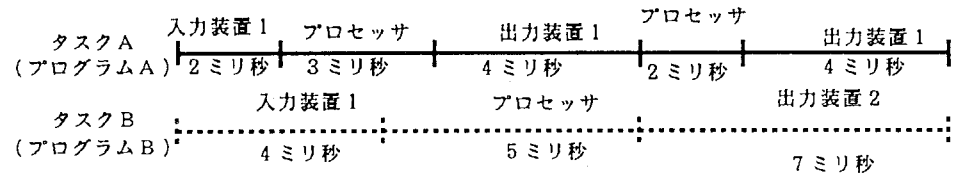
3．ハードウェアに関する部分は詳細に立ち入らず、機能を重点に説明すればよいが、入出力のメカニズムはコンピュータの基本的な仕組みとして重要である。

練　習　問　題

（問題１）　マルチタスク環境下で、下図のようにタスクＡ、Ｂを実行した。すべてのタスクの実行が終了するのに（　　a　　）ミリ秒かかるか。解答群の中から一つ選べ。

（ヒント：優先度方式のスケジュールとタスク動作の確認）

≪■各タスク単独での処理時間≫



≪■制約条件≫

１）プログラムＡはプログラムＢよりも優先度が高い。

２）プログラムは、ほかのプログラムが使用中の周辺装置を使用することができない。

３）プログラムＡは、プログラムＢが使用中のプロセッサに割り込んで使用することができる。

解答群

ア　16　イ　18　　ウ　20　　エ　22　　オ　24

参 考 文 献

1．中央情報教育研究所編；「第二種共通テキスト⑧　情報処理システム」、
日本情報処理開発協会、1994年
2．大原茂之他著；「実践リアルタイムプログラミング技法」、オーム社，1991年
3．廣瀬　健他著；「コンピュータソフトウェア辞典」、丸善社、1990年
4．佐々木　茂著；「解析マルチタスク」－システムの内部構造とプログラミング－、
CQ出版社、1988年
5．神本　武征著；「ディジタル計測制御」、オーム社、1990年