1.1.3　コンピュータシステムによる情報処理の階層構造

(a)　コンピュータシステムにおけるハードウェア/ソフトウェア・トレードオフ

1.1.2 項(b)で述べたように，コンピュータシステムのトレードオフを具体的に決定するものは，コンピュータの機能の実現におけるハードウェアとソフトウェアの機能分担の割合や方式である。本書で特に断わりなく“トレードオフ”という用語を使用した場合には，この“**コンピュータシステムにおけるハードウェアとソフトウェアの機能分担**方式”のことを示している。特に，これを他のトレードオフと区別する必要のある場合や強調すべきところでは**ハードウェア/ソフトウェア・トレードオフ**という。

コンピュータシステムの機能を実現する具体的な処理方式や処理機構の設計や選択においては，必ずハードウェア/ソフトウェア・トレードオフを決定しなければならない。部分的なトレードオフを決めなければシステム全体の機能は実現できないし，また逆に部分的な機能を実現する際にシステム全体のトレードオフにも配慮せねばならない。

(b)　情報処理の階層構造

コンピュータシステムによる情報処理過程は，図 1.3 に示すように，処理の機能レベルによって階層化できる。そして，応用問題の機能レベルとコンピュータハードウェアで実現している機能レベルとには大きな差がある。この機能差のことを**セマンティックギャップ**(semantic gap)という。応用問題がコンピュータハードウェア上で実行されるまでには，次に掲げるようなセマンティックギャップを埋める機能を実現しなければならない。

(1)　応用問題を**アルゴリズム**(algorithm)化する。すなわち，コンピュータ上で問題を解決する計算(処理)手順としてのアルゴリズムを開発する。この過程を実現する機能は**プログラミングパラダイム**(programming paradigm；プログラミング方法の枠組み)とか**計算モデル**という。

(2)　アルゴリズムを**プログラム**(program)化する。すなわち，**プログラミング**(programming)する(プログラムを作る)。この過程の機能は**プログラミング言語**(programming language)によって実現する。

(3)　プログラムを処理してコンピュータハードウェア上で実行可能な**マシン語**(machine language)(コンピュータが理解できる言語，**マシン命令**ともいう)に翻訳する。この機能は，1.1.4 項(c)で述べる言語処理プログラムやオペレーティングシステム(OS)などの**システムプログラム**(system program)

モリ系と，モデム，**SCSI**(Small Computer System Interface；スカジー)，ネットワーク接続などの入出力インタフェース系との，大きく2種類に分類できる。

⑥ 赤外線や電波を介してコンピュータどうしあるいはコンピュータ本体と周辺装置との接続を無線(wireless；ワイヤレス)化することが始まり，携帯電話の普及とともに，ネットワークの無線化の端緒となっている。

⑦ マシン命令の時間的多重処理(実行)であるパイプライン処理と複数のマシン命令を同時実行する空間的な多重処理(実行)を組み合わせた**命令レベル並列処理**の採用が(マイクロ)プロセッサアーキテクチャとして主流となっている。

⑧ 特定用途向きLSIである**ASIC**(Application Specific IC)が種々開発され，高機能な汎用マイクロプロセッサとは別の市場を築いている。ほとんどの電子機器には機器制御用の特定プログラムをあらかじめチップ上に実装したASICプロセッサが組み込まれ，これらのASICプロセッサを“組み込みプロセッサ(embedded processor)”，“マイクロコントローラ(microcontroller)”あるいは“1チップマイコン”などという。

⑨ コンピュータの誕生以来その応用分野として注目されてきた**AI**(Artificial Intelligence；人工知能)がコンピュータ性能の高度化にしたがって再びブームとなり，AI技術の可能性と限界を精力的にチェックしている。日本では，“第5世代コンピュータ”と呼ぶAI向きコンピュータシステムの国家的開発プロジェクト(1981～1992年)も推進されている。

(h)　第6世代(1990年代～2000年代)：ユビキタス時代

(1)　32ビットあるいは64ビットマイクロプロセッサを構成するVLSIには，10^7個以上(数千万個)のトランジスタを1チップ上に集積するものも現れている。これら最高性能マイクロプロセッサのクロック周波数は数GHzに達している。また，VLSIチップ上にプロセッサのほかにメインメモリとしてのDRAMなども混載する**システムLSI**(**システムオンチップ**(system on chip)ともいう)も出現している。一方で，パソコンや電子機器のモバイル利用が広がるにつれ，それらの主要ハードウェア部品であるマイクロプロセッサの要件に**低消費電力**が加わり，性能と消費電力とのトレードオフもコンピュータアーキテクチャ設計における重要な評価指標になっている。

(2)　DRAMの集積度についてはムーアの法則が依然として有効であり，DRAMで構成するメインメモリの実装容量も，高性能ワークステーションや

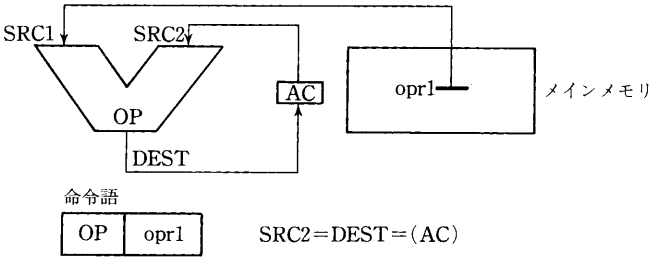

図 2.11 1アドレス形式

レジスタオペランドである命令形式を **R-R-R 形式**という。

（2） **2アドレス形式**（図 2.10 参照）：2個のうちどちらかのソースオペランドと1個のデスチネーションオペランドを兼用することによって，2個のオペランドで1命令語を構成する。2個のオペランド（SRC1-SRC2（＝DEST）の順）をメモリオペランドかレジスタオペランドのいずれにするかによって，① **M-M 形式**；② **R-M 形式**か **M-R 形式**；③ **R-R 形式**；の3種類に細分できる。

（3） **1アドレス形式**（図 2.11 参照）：レジスタの代りとして**アキュムレータ**（accumulator；累算器，AC）という1ワードの特別な格納装置をオペランドとしてあらかじめ暗黙的に決めておき，命令語でのオペランドの明示的な指定を省略する。命令語中では1個のオペランド（SRC1）だけを明示する。1個のソースオペランドとデスチネーションオペランドをアキュムレータとし，もう1個のソースオペランドのみを命令内で指定する命令形式を指すのが普通である。1アドレス形式を採用したコンピュータを“アキュムレータマシン”という。

（4） **1・1/2アドレス形式**（図 2.12 参照）：2アドレス命令形式において，片方のオペランドがレジスタオペランドである場合を特にいう。レジスタオペランドはメモリオペランドよりも短くてすむので，(2) の2アドレス形式の欠点が自立たない。1個しかない AC の代りに，複数個あるうちの1個のレジスタを指定するので，(3) の1アドレス形式に比べてオペランド指定の柔然性は増す。(2) と (3) の命令形式の融合方式である。(2) の2アドレス形式のうちの **R-M 形式**（opr 1 がレジスタオペランド，opr 2 がメモリオペランド，図 2.12 の場合）と **M-R 形式**（opr 1 がメモリオペランド，opr 2 がレジスタオペランド）がこの 1・1/2 アドレス形式にあたる。

い，あらかじめ命令セットアーキテクチャとして決めておく。そして，(a)で述べた ⓐ～ⓒ を実現できる種々のアドレス指定モードをアーキテクチャとして実現して，特に ② の実行時の可変性に関する相反する要件の両立を図る。

“メモリオペランドとメモリアドレスとの対応付けだけを行って，それらを互いに独立して付ける”方法は 7.2 節で詳述する**仮想メモリ**によっても実現している。現代のコンピュータでは，プロセッサアーキテクチャとしてのアドレス指定モードとメモリアーキテクチャとしての仮想メモリとの両方によって，プロセッサで指定するメモリオペランドとメインメモリに付けたメモリアドレスとで相反する要件となっている ① ② ((a) 参照) を解決している。

メモリオペランドをもとに生成したメインメモリ空間のアドレス(メモリアドレス)を**実効アドレス** (effective address) という。アドレス指定モードは“命令語中のオペランド(アドレス情報)から実効アドレス(メモリアドレス)を生成(計算)する方法”でもある。

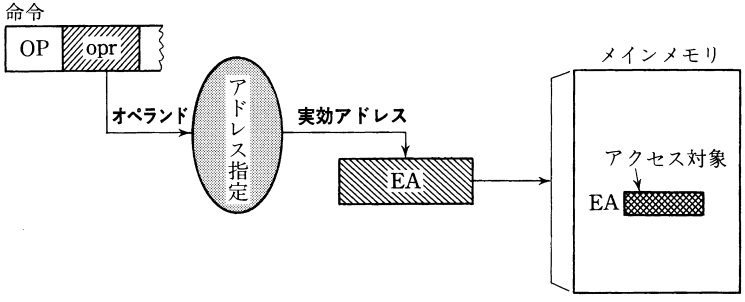

図 2.14 アドレス指定モードによる実効アドレスの生成

図 2.14 に示すように，アドレス指定モードを実現するために，オペランド (opr) から実効アドレス (EA) を生成するアドレス指定機能をハードウェア機構 (アドレス指定機構) として備える。アドレス指定モードは，① 命令語中で明示指定する；② 命令種類や命令形式ごとに特定の方式 (モード) をあらかじめ決めておく (暗黙指定)；などによる。また，実際には，ほとんどのコンピュータがメモリアーキテクチャとして仮想メモリ方式 (7.2 節参照) を採用しており，アドレス指定機構は仮想メモリを実現する機構の前段に置く。

アドレス指定モードが有効であるメインメモリ (メモリオペランド) に比べると，レジスタのアドレス空間 (個数) は小さく固定であるので，レジスタオペランドはアドレス指定モードによらずに直接レジスタ番号として指定する。

アクセス対象＝Mem[EA]，EA＝Reg[opr]

命令を書き換える(再コンパイルする)ことなく，レジスタ内容をプログラムで操作することによって実行時に(動的に)実効アドレスを変えることができる。また，1回目の実効アドレスの読み出しが(メインメモリではなく)レジスタへのアクセスとなるので，間接アドレス指定の短所である長いアクセス時間を短縮できる。

次の(3)～(5)は**相対アドレス指定**である。

(3)　**インデックス**：ベースアドレスとしてオペランド(oprB)を直接使用し，ディスプレースメントをオペランド(oprD)によって指定したレジスタにある値とする(レジスタ間接)相対アドレス指定モードである(図2.22参照)。

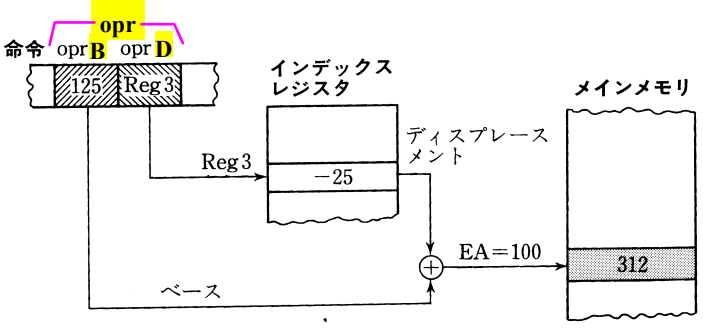

図2.22　インデックスアドレス指定

アクセス対象＝Mem[EA]，EA＝oprB＋Reg[oprD]

静的(コンパイル時)に決める固定ベースアドレスと，動的(実行時)に計算・変更できる可変ディスプレースメントによる相対アドレス指定を行う。均質(同一属性)データの集まり(データブロック)の各データ要素への順次アクセスに**インデックス**(index；添字)を利用するので，ディスプレースメントを格納するレジスタを**インデックスレジスタ**という。インデックスレジスタには命令実行ごとにインデックスを指定数(普通はデータ長のバイト数)だけ増減する自動インクリメント(increment)/デクリメント(decrement)機能を装備する。① 汎用レジスタ(2.1.2項(b)の(1)参照)をインデックスレジスタとして流用する；② 汎用レジスタとは別に専用のインデックスレジスタを設ける；の2方式がある。図2.23に示すように，同一マシン命令をくり返し実行するたびに，等間隔に置いた均質データを順次オペランドとして指定できる。

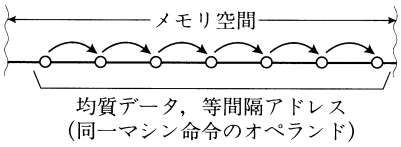

図 2.23　インデックスアドレス指定の利用

インデックスアドレス指定は (a) で述べたメモリアドレス指定例の ⓑ を実現する。

（4）　**ベース**：(3) のインデックスアドレス指定におけるレジスタの役割 (ディスプレースメントの格納) をベースアドレスの格納に変えたアドレス指定である (図 2.24 参照)。

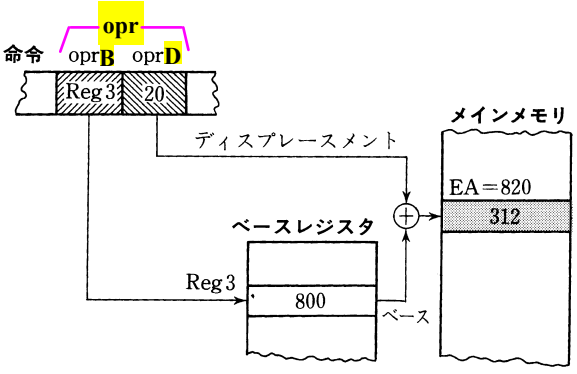

図 2.24　ベースアドレス指定

$$\text{アクセス対象}=\text{Mem}[\text{EA}],\quad \text{EA}=\text{Reg}[\text{opr}_{\mathbf{B}}]+\text{opr}_{\mathbf{D}}$$

ベースアドレスが可変となり，ディスプレースメントは固定となる。ベースアドレスを格納するレジスタを**ベースレジスタ**という。ベースレジスタの構成にも，① 汎用レジスタを流用する；② 専用のベースレジスタを設ける；の 2 方式がある。ベースアドレス指定は，図 2.25 に示すように，① **リロケーション** (relocation) というプログラムの再配置 (配置変え，プログラムのメインメモリ上での格納場所を移動すること)；② ブロック内のデータ個々の相対アドレスはコンパイル時に決まるが，データブロックのベースアドレスは実行時に決まる場合，たとえば現在実行中の手続きや関数が使用しているメモリブロックの指定；などの機能の実現に利用できる。ベースアドレス指定は (a) で述べた

ンメモリに格納してあるマシン命令列を順次実行する。命令実行サイクルにおける各ステージの役割について図2.27に沿って説明してみよう。図2.27では，左側にハードウェア構成の概略を示してあり，右側に命令実行サイクルの各ステージと各ステージが主として使用するハードウェア機構とを併記してある。

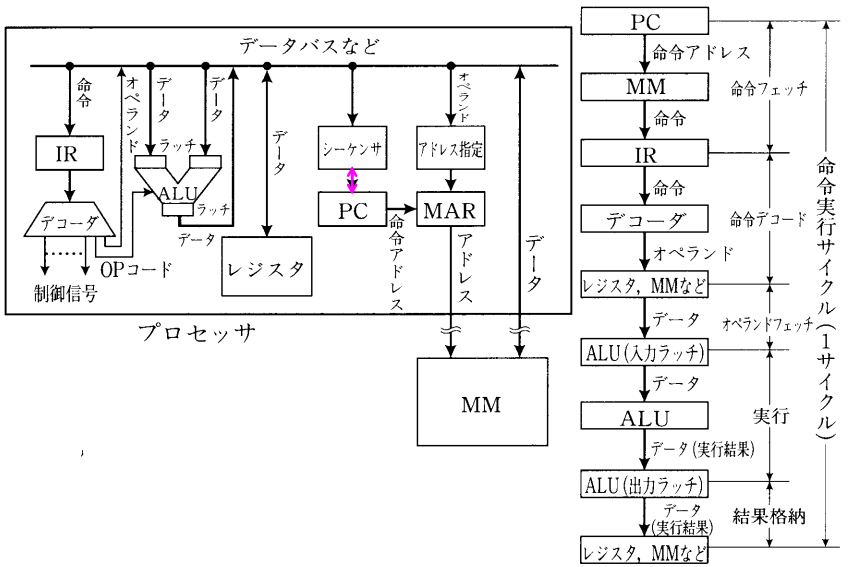

図2.27　命令実行サイクルとハードウェア構成例

（a）　命令フェッチ

プログラムカウンタ(PC)に置いた命令アドレスをメモリアドレスレジスタ(MAR)に設定し，命令の読み出しをメインメモリ(MM)に指令する。読み出した命令が命令レジスタ(IR)に設定されるまでのステージである。

“命令をメインメモリより読み出し，プロセッサ内のIRに格納する”ことを**命令フェッチ**(fetch；取り出し)という。

（b）　命令デコード

フェッチした命令は命令形式にしたがって符号化(エンコード)してあるので，それをデコード回路(デコーダ)によって復号(デコード)することでOPコードとオペランドの抽出さらには制御信号の生成などを行うステージである。

スーパスカラに対して，命令レベル並列性を（実行時ではなく）コンパイル時に（静的に）行う命令レベル並列処理もある。この命令レベル並列処理では，コンパイラが抽出した命令レベル並列性をそのまま長い命令長のマシン命令（**VLIW** (Very Long Instruction Word) あるいは“超長形式命令語”という）に埋め込み，そのVLIWに埋め込んだ複数命令（演算）機能を並列実行する。この命令レベル並列処理方式を採るコンピュータを**VLIW コンピュータ**という。

命令レベル並列処理方式をハードウェア/ソフトウェア・トレードオフの観点で分類すると，① スーパスカラアーキテクチャ：ハードウェア重視；② VLIW アーキテクチャ：ソフトウェア（コンパイラ）重視；となる。

5.2.3　制御機構とオペレーティングシステム

コンピュータの実行を管理制御するソフトウェア機能は**オペレーティングシステム (OS)** という。OSは制御機構ハードウェアと直接かかわり合うソフトウェア機能であり，制御アーキテクチャの設計では制御機構とOSとの機能分担について十分配慮しなければならない。

（a）　プロセッサとプロセス

プロセッサはマシン命令を実行する物理的（ハードウェア）機構であり，命令実行サイクルを繰り返しながらメインメモリに置いてあるマシン命令列を順次フェッチして（取り出して）実行する。

このとき，実行に先立って（実行前に）あらかじめメインメモリに置いてあって，“ある機能を実現するために実行するプログラム（すなわち動的にできるマシン命令列やそれらが使用するデータの集まり）”を**プロセス** (process) あるいは**タスク** (task) という（本書では“プロセス”を使う）。

プロセッサは実行制御対象の物理的な単位であり，プロセスはそれの論理的な単位である。プロセッサでプロセスを実行するためには，プロセスをメインメモリにあらかじめ置いておき（“プロセス割り付け”という），プロセッサとプロセスを対応付けることが必要となる。OSは，① メインメモリにプロセスを割り付ける；② プロセッサにプロセスを割り当てる；を担当する。

（b）　マルチタスキング

ノイマン型コンピュータでは，プロセッサは単一であり，一時には唯一プロセスしか実行できない。そこで，プロセッサの利用時間（“プロセッサ時間”という）を細かく（数十ミリ秒が普通）分割し，それらをOSが各プロセスの実行に割り当てるプロセッサ制御方式が**時分割制御**である。

（e）　プロセス制御ブロック

プロセスを管理し，その実行を制御するための情報（の格納場所）を**プロセス制御ブロック**（**PCB**, Process Control Block）あるいは**プロセステーブル**という。PCBはプロセスの状態を示しており，プロセスの切り替え時にはPCBを切り替えればよい。PCBは，① プロセス識別番号（ID）；② 優先順位やライフタイム（そのプロセスを生成してからの経過時間）などのスケジューリング情報；③ メモリ管理情報やプロセッサ管理情報の格納先アドレスとその大きさ（サイズ）；などから構成する。

（f）　プロセススイッチ

OSは，① 実行中状態のプロセスを実行可能状態にする；② あらかじめ決めておいた一定のスケジューリング方針（**スケジューリングアルゴリズム**（scheduling algorithm）という，（g）で詳述）にしたがって，実行可能状態のプロセスから1個を選定し，それを実行中状態にする；を行い，プロセスを切り替える。これを**プロセススイッチ**（process switch）という。

プロセススイッチは具体的にはPCBやそのほかのハードウェア機構の状態の切り替えである。プロセススイッチに際して退避が必要となる各種のハードウェア状態を**プロセスコンテクスト**（process context）あるいは**プロセッサ状態ワード**（**PSW**（Processer Status Word），**プログラム状態ワード**（Program Status Word）ということもある）という。プロセスコンテクストあるいはPSWは，① プロセッサ状態（（i）参照）；② コンディション（条件，2.2.6項（f）参照）；③ プログラムカウンタ（PC）；④ 汎用レジスタ；⑤ そのプロセスが使用しているメインメモリ内容の完全なコピー（“メモリイメージ（memory image）”という）；⑥ メインメモリについての管理情報；⑦ 割り込みの優先度；などによって構成する。

①～⑦のうち①～③だけを“PSW”と，そのPSWに④～⑦を加えて“プロセスコンテクスト”と，それぞれ定義して，PSWとプロセスコンテクストとを区別することもある。後の5.3.1項（c）の割り込み処理では，この定義にしたがって，①～③のPSWと①～⑦（①～③のPSWと④～⑦）のプロセスコンテクストとを区別して説明している。

プロセススイッチは，実際には，“OSによるプロセスコンテクスト（PSWを含む）のメインメモリ（内のPCBあるいはプロセステーブル）への退避およびメインメモリからの回復の処理”である。

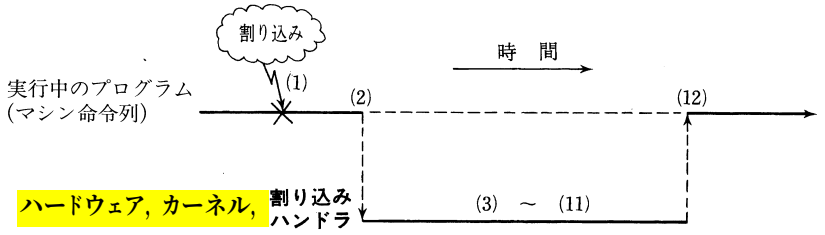

図 5.22 割り込み処理

(6) **割り込みハンドラ** (handler) への分岐。

(7) (今まで実行中であった) プロセスコンテクスト ((4) で退避した PSW を含む) のメインメモリ (実際には，メインメモリ内の PCB あるいはプロセステーブル領域，5.2.3 項 (e) 参照) への退避。

(8) 割り込みハンドラによる割り込み要因ごとの処理。

(9) (7) で退避しておいたプロセスコンテクストの回復。

(10) (4) で退避しておいた PSW (PC を除く) の回復。

(11) 他の割り込みを許可 (割り込み可能状態への移行)。

(12) 割り込み受付時点への復帰。

まず，割り込みが発生 (1) すると，OS カーネルで一般的な割り込みへの対応 (前処理) (2)～(4) を行う。次に，割り込み要因を識別 (5) すると，それに対応する**割り込みハンドラ**に分岐 (6) する。そして，割り込みハンドラは個別の要因ごとに定めておいた処理 (7) (8) を行う。最後に，割り込みハンドラによる要因ごとの処理 (8) が終了すると，カーネルに戻って，割り込みに対する一般的な後処理 (9)～(11) を行い，中断していた (割り込まれた) プロセスに復帰 (12) する。なお，割り込み要因によっては，中断プロセスに復帰しないこともある。

(2)～(12) の手順 (ルーチン) を**割り込み処理**あるいは**割り込み処理ルーチン**という。一連の割り込み処理ルーチンにおいて，割り込み要因ごとに実行時に決まる (動的な) 処理機能 ((7)～(9)) が**割り込みハンドラ** (による処理) である。

割り込み処理中は原則として割り込み禁止とし，他の割り込みが発生しても現在の割り込み処理が終了するまで ((4)～(10) の間)，新たな割り込みの受付を待たせる。この原則にしたがわない場合や割り込み処理中に別の割り込みが発生した場合の割り込み処理については (h) で詳述する。

割り込み処理の手順は実際には制御アーキテクチャとしてハードウェア機構

とソフトウェア(OS)との機能分担によって実現する。一般的には，(1)～(6)と(10)～(12)は高速処理が必要なので，OSカーネルのもとでハードウェア機構(割り込み処理機構，次の5.3.2項でその一部について詳述)が分担する。一方，(7)～(9)(割り込みハンドラによる処理)は割り込み要因ごとに処理内容が異なる場合への対処などの問題適応性が必要となるので，OS(ソフトウェア)が分担する。また，割り込み要因によっては，(8)での割り込みハンドラがユーザ(OS以外の)プロセスとして動く場合もある。

(d)　割り込みハンドラ

割り込みハンドラが行う(狭義の)割り込み処理は，次に述べるように，割り込み要因によって異なる。

(1)　**内部割り込み**に対する割り込み処理：① **命令実行例外**に対しては，例外を起こした原因をユーザプログラム(割り込み要因となったマシン命令を含むプログラム，これは割り込まれたプログラムでもある)に通知したり，OSがその原因となった事象を取り除く。たとえば，ページフォールト(7.2.4項(a)で詳述)はユーザプログラムの責任ではないので，OSが原因となった事象(メインメモリに実行しようとしたマシン命令がない)を取り除き(当該マシン命令が含まれるブロックをメインメモリに置き)，ユーザプログラムを再開してやる。ページフォールト以外の命令実行例外はユーザプログラムの責任であり，例外事象(割り込み要因)を引き起こしたユーザプログラムにOSが対応(たとえば，メッセージを出して当該ユーザプログラムを強制終了するなど)する。② **SVC命令**や**ブレークポイント命令**による割り込みに対しては，ユーザプログラム(プロセッサ状態ではユーザモード)によるOS(プロセッサ状態では特権モード)の呼び出しであるので，それに答えて呼び出し(依頼)内容を実行する。

(2)　**外部割り込み**に対する割り込み処理：① **入出力割り込み**と**タイマ割り込み**に対しては，その事象(外部割り込み)を待っている実行待ち状態プロセスを実行可能状態にする(**ウェイクアップ**(wake-up)という，5.2.3項(d)参照)。② ハードウェア障害に対しては，OS自身がメッセージ表示，停止，リブート(reboot；再起動)などを行う。

(3)　**リセット**に対する割り込み処理：原則としてOSをリブートする。

割り込みハンドラはいくつかのあらかじめ(制御アーキテクチャの設計時に)決めておいた割り込み要因のそれぞれに対してOSが用意しているソフトウェア(OSプログラム)である。

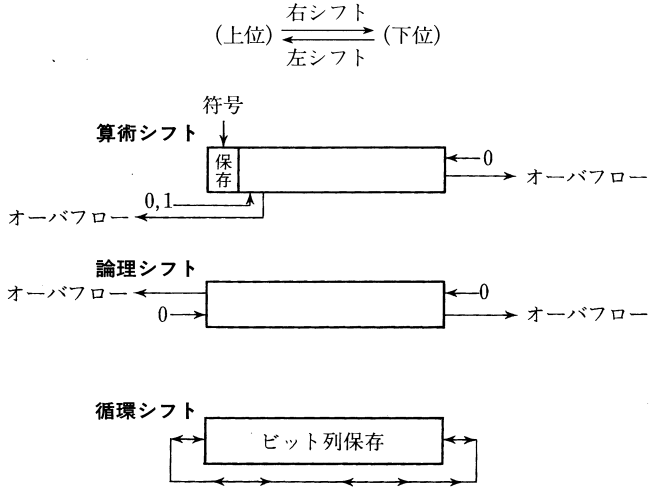

図 2.31 シフト

をもっているデータである場合が多い。ビット列の意味の配慮すなわち対象ビット列のデータ型によって，次のように細分できる。

- **算術シフト**：数値データを対象とし，符号(ビット)を保存しつつシフトする。したがって，結果データも数値である。n ビット算術シフトは元の数値に 2^n を乗じる(左シフト)あるいは除する(右シフト)整数演算と同じ結果を与える。負数の右シフト(除算)の場合には，負数の内部表現形式(3.2.1 項で詳述)によっては，(符号ビットを除く)最上位(最左)ビットから“0”ではなく“1”を補充しなければならない(6.1.1 項(g)で詳述)。
- **論理シフト**：論理値データを対象とし，対象データすべてを独立した論理値からなるビット列データとして操作する。結果データも論理値データである。
- **循環シフト**：論理値データを対象とする論理シフトの一種であるが，シフト操作によってもシフト対象となるビット列情報を失わない。すなわち，最上

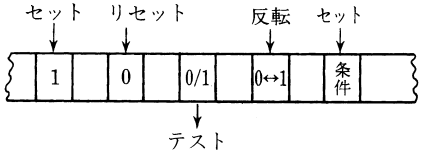

図 2.32 ビット列操作

付け替えればよく，また，絶対値が同じで符号が異なる(正と負の)数値を直感的に把握し記述できる；② 負符号と減算記号の数学的意味が同じであるので，数式の記述が自然にできる；という長所と，一方で，③ 同一絶対値に対する異種演算(特に加算と減算)を符号によって使い分けねばならない；という短所とをもつ。この①② の長所と ③ の短所が“符号-絶対値表現を，人間による 10 進数表現では使うが，コンピュータ内部での 2 進数表現では使わない”理由である。

符号-絶対値表現に対して，コンピュータ内部での 2 進数(特に負数)表現に適した数表現として，符号ビットも数値ビット(列)に連結する(最上位)ビットとして等価に(含めて，一緒に)扱う**補数表現**(complement representation)がある。

符号と絶対値を一体化した数値として表す補数表現の特徴は，① 正負の判定操作以外では，符号(ビット)と数値(ビット列)とを識別する必要がない；② 正数と負数とをまったく同じように扱えるので加算と減算とを区別する必要がない；という長所と，③ 負数の表現には補数への変換操作が必要であり，また，負数の絶対値を(人間は)直感的に把握できない；という短所である。③ の短所はコンピュータでは無視できるので，特に，①② の長所が“補数表現をコンピュータ内部での負数の 2 進数表現として使う”理由である。

一般的に，符号を含めて数値を表現する r 進数の補数表現では，符号は最上位桁(2 進数では MSb)の数字で示す。正数の符号“＋”は“0”，補数表現した負数の符号“－”は“$r-1$”である。

2 進数の補数表現では，正数の MSb は“0”，補数表現した負数の MSb は“1”となる。

2 進数の補数表現には，① **1 の補数表現**((c) で詳述)；② **2 の補数表現**((d) で詳述)；の 2 種類がある。

(c) 1 の補数表現

図 3.5 に示すように，ある正数 R の符号桁(“0”で表現)を含めた数値の r 進数表現の各桁を $(r-1)$ から減算して得る数表現 $\overline{R}$ は R を符号反転して得る負数 $-R$ を表現しており，これを **$(r-1)$ の補数表現**($(r-1)$'s complement representation)という。逆に，負数 $-R$ の $(r-1)$ の補数表現は正数 R を表す。ある数 R と絶対値が同じで符号が逆の $(r-1)$ の補数表現で表す数 $\overline{R}$ をその数の **$(r-1)$ の補数** あるいは **擬補数** という。10 進数の場合は“9 の補数”であり，2 進数の場合は **1 の補数**(1's complement)である。r 進数表現する

$$\overline{N}=2^4-1-6=9 \tag{3.14}$$

より，N の1の補数 $\overline{N}$ は $(9)_{10}$ で，4ビット2進整数表現すると $(1001)_2$ ($(0110)_2$ の各ビットの反転) である。

1の補数(表現)には，① 2進数表現の各ビットを反転($0 \Leftrightarrow 1$)するだけで得ることができるので1の補数化機構は簡単である(6.1.2項(f)参照)；という長所と，② 加減算を行うときに加減算結果の補正操作(+1の加算)が必要となる場合がある(6.1.1項(e)で詳述)；という短所がある。

(d)　2の補数表現

n 桁の整数部と m 桁の小数部をもつ R から次の式(3.15)で求める数表現 $\overline{R}$ は R を符号反転して得る負数 $-R$ を表現しており，これを **rの補数表現** (r's complement representation) という。逆に，負数 $-R$ の r の補数表現は正数 R を表す。ある数 R と絶対値が同じで符号が逆の r の補数表現で表す数 $\overline{R}$ をその数の **r の補数** あるいは **真補数** という。ただし，式(3.15)～(3.19)では，R と $\overline{R}$ は10進数値で表し，それぞれの符号桁(最上位桁)もそれぞれの数値部と連結・一体化して考える。

$$\overline{R}=r^n-R \tag{3.15}$$

式(3.15)によると，R の r の補数 $\overline{R}$ の数値(10進数)は r 進数表現した R の小数部の長さ(桁数) m によらずに求まる。

10進数の場合は“10の補数”であり，2進数の場合は **2の補数** (2's complement) である。

ある数 R (r 進数表現) の r の補数 $\overline{R}$ (r 進数表現) は次のような手順で求める(図3.6に例示)。

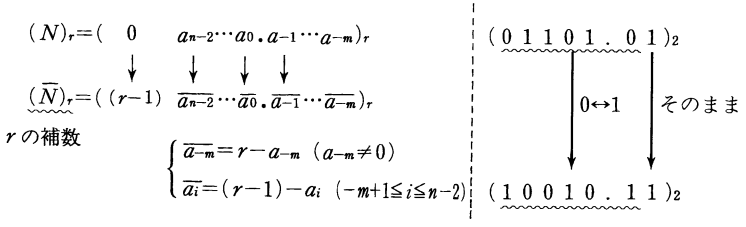

図 3.6　r の補数表現と2の補数表現の例

(1)　小数点位置に無関係に(小数点位置はそのままで)，最下位桁から順に次の(2)～(4)によって r の補数の各桁を得る。

（2） まず，最下位桁からの0の連なり（連続する0）はそのままにしておく。

（3） 最初の0でない最下位桁を r から減算することによって r の補数のその桁を得る。

（4） その次の上位桁からは，$(r-1)$ から各桁を減算することによって補数の各桁を得る。これを繰り返す。

実際には，式(3.10)と(3.15)より

$$\overline{R}=\overline{\overline{R}}+r^{-m} \tag{3.16}$$

である。すなわち，m 桁の小数部をもつ数の r の補数表現は $(r-1)$ の補数表現に r^{-m} を加算して得ることができる。

ある数 R（2進数表現）の2の補数 $\overline{R}$（2進数表現）を求める手順は次のようになる（図3.6右に例示）。

（1） R のLSbから順に2の補数 $\overline{R}$ の各ビットを(2)～(4)によって得ていく。

（2） LSbから連続する0をそのままにしておく。

（3） 最初に出現する1もそのままにしておく。

（4） その次のビットからは各ビットを反転して $(0\Leftrightarrow 1)$ 2の補数 $\overline{R}$ の各ビットを得る。

2進数表現すると n ビットの整数部と m ビットの小数部をもつ実数 R とその2の補数 $\overline{R}$（2進数表現）との関係は，式(3.15)より，R と $\overline{R}$ を10進数値で表すと，

$$\overline{R}=2^n-R \tag{3.17}$$

である。式(3.17)によると，R の2の補数 $\overline{R}$ の数値（10進数）は2進数表現した R の小数部の長さ（ビット長）m によらずに求まる。たとえば，R が4ビット2進数表現で $(0111)_2$，すなわち R が $(7)_{10}$ の場合，

$$\overline{R}=2^4-7=9 \tag{3.18}$$

より，R の2の補数 $\overline{R}$ は $(9)_{10}$ で，4ビット2進整数表現すると $(1001)_2$ である。

また，式(3.16)より，2進数表現では，

$$\overline{R}=\overline{\overline{R}}+2^{-m} \tag{3.19}$$

である。

したがって，2進数表現した n ビット整数 N の1の補数 と2の補数 $\overline{N}$（2進数表現）との関係は式(3.19)で $m=0$ として，

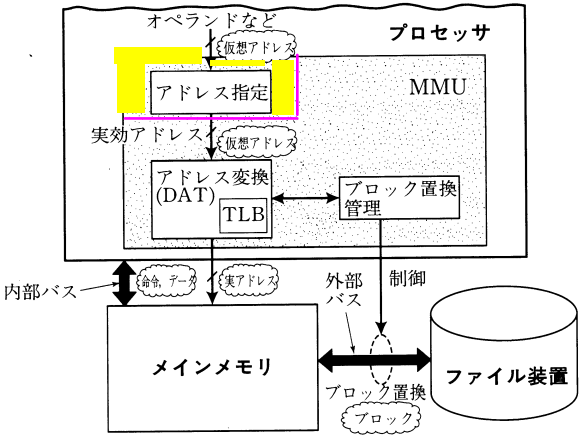

図 7.14　仮想メモリ機構

モード，2.2.3 項参照) にしたがい，MMU の前段に置いたアドレス指定機構によって実効アドレスとなる。実効アドレスも仮想アドレスである。

(3)　MMU は“実効アドレス (仮想アドレス) を含むブロックが実メモリ (実アドレス空間，メインメモリ) 上にマッピングしてある (存在する) かどうか”について，アドレス変換テーブルを引くことによって調べる。

①　あれば，MMU 内の**アドレス変換機構** ((b) 参照) がアドレス変換テーブルによって仮想アドレスを実 (物理) アドレスに変換する。

②　なければ，ⓐ MMU が“メモリアクセス例外”あるいは“ページ (セグメント) フォールト”という要因による割り込み (5.3 節で詳述) によって，“アクセスしようとしたアドレスがメインメモリ (実メモリ) にない (**フォールト** (fault) という，7.2.4 項 (a) 参照)”ことを OS に知らせる；ⓑ OS は，MMU の**ブロック置換管理機構** ((c) 参照) と協調して，メインメモリ内の不要なブロックを決定する；ⓒ メインメモリ上の不要なブロックとバックアップメモリ (ファイル装置) にある必要な仮想アドレスを含むブロックとを置換する (入れ替える)；ⓓ ① を実行する；を行う。

(b)　アドレス変換機構

アドレス変換機構は，① 実メモリにアクセス対象を含むブロックが存在するかしないかを検査する機構；② 仮想 (論理) アドレス → 実 (物理) アドレス変換を実現する機構；によって構成する。

仮想アドレスを実アドレスに変換する**アドレス変換** (address translation)

数 $\bar{N}$ には次の関係がある (3.2.1 項 (c) の式 (3.12) の再掲)。

$$N+\bar{N}=2^{p}-2^{-q}=(\overbrace{111\cdots11}^{(p+q)\text{ビット}})_2 \qquad (6.3)$$

図 6.5 に示すように，式 (6.3) は“ある数とその 1 の補数との和は 0 である”ことを示している。

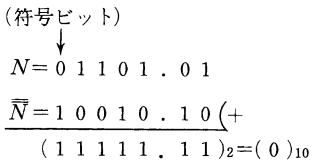

図 6.5　ある数 N とその 1 の補数 $\bar{N}$ との関係

1 の補数表現固定小数点数の加減算は，次に示すように，符号を含めて 1 の補数表現にした減数による加算手順で行える。すなわち，被減数 M と減数 N との減算

$$S=M-N$$

は被減数 M と“減数 N の 1 の補数 $\bar{N}$”との加算

$$S=M+\bar{N}$$

となる (図 6.6 に例示)。

負数を 1 の補数表現する場合の 2 進数 M と N の加減算は次の手順の“加算”となる。

(1)　符号ビットも含めて (負数の場合は 1 の補数表現で) 被加数 M と加数 N を加算する。

(2)　最上位ビット (MSb) からの桁上げ (**エンドキャリ** (end carry) という) があれば，(1) の演算結果にその桁上げ 1 を加え，桁上げそのものは無視する。すなわち，エンドキャリを (1) の演算結果に加えてそれを**補正**する。

(3)　残った結果が符号を含めた演算結果 (負数は 1 の補数表現) S である。

この手順 (2) の補正で用いる MSb からのエンドキャリを**循環桁上げ** (end-around carry，**エンドアラウンドキャリ**) という (図 6.6 の例参照)。

負数を 1 の補数表現する加 (減) 算におけるオーバフローは演算結果 S が 1 の補数の表現範囲である。

$$-(2^{p-1}-1)\leqq S\leqq 2^{p-1}-1 \quad (\text{たとえば } p=8 \text{ では，} -127\leqq S\leqq 127) \qquad (6.4)$$

の外にある場合である。オーバフローは被加数と加数の符号が同一の加算の場

いう長所と，② 循環桁上げ(エンドアラウンドキャリ)があれば補正を行わねばならない；という短所がある。特に，② は循環桁上げの有無によって補正(循環桁上げの加算)の有無を演算機構で切り替える必要があり，次の(f)の2の補数表現による加減算に比べてハードウェア機構の規模は大きくなる。

(f)　2の補数表現固定小数点数の加減算

ある2進数 N (p ビットの整数部と q ビットの小数部をもつ) とその2の補数 $\overline{N}$ には次の関係がある (3.2.1 項 (d) の式 (3.17) の再掲)。

$$N+\overline{N}=2^p=(\overbrace{1000\cdots00}^{(p+q)\text{ビット}})_2 \tag{6.5}$$

図 6.8 に示すように，式 (6.5) は“ある数とその2の補数との和は最上位ビット (MSb) からの桁上げ (エンドキャリ) を無視する ($(p+q)$ ビット分だけを見る) と0である”ことを示している。

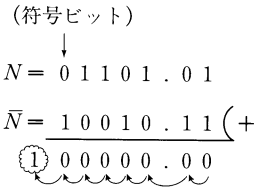

図 6.8　ある数 N とその2の補数 $\overline{N}$ との関係

2の補数表現を用いた固定小数点数の加減算は，次に示すように，符号を含めて2の補数表現にした減数の加算手順で行える。すなわち，被減数 M と減数 N との減算

$$S=M-N$$

は被減数 M と“減数 N の2の補数 $\overline{N}$”との加算

$$S=M+\overline{N}$$

となる (図 6.9 に例示)。

負数を2の補数表現する場合の2進数 M と N の加減算は次の手順の加算となる。

(1)　符号ビットも含めて (負数の場合は2の補数表現で) 被加数 M と加数 N を加算する。

(2)　最上位ビット (MSb) からの桁上げ (**エンドキャリ**) は無視する (図 6.9 の例参照)。

アクセス対象＝Mem[EA]，EA＝Reg[opr]

命令を書き換える(再コンパイルする)ことなく，レジスタ内容をプログラムで操作することによって実行時に(動的に)実効アドレスを変えることができる。また，1回目の実効アドレスの読み出しが(メインメモリではなく)レジスタへのアクセスとなるので，間接アドレス指定の短所である長いアクセス時間を短縮できる。

次の(3)～(6)は**相対アドレス指定**である。

(3)　**インデックス**：ベースアドレスとしてオペランド(opr B)を直接使用し，ディスプレースメントをオペランド(opr D)によって指定したレジスタにある値とする(レジスタ間接)相対アドレス指定モードである(図2.22参照)。

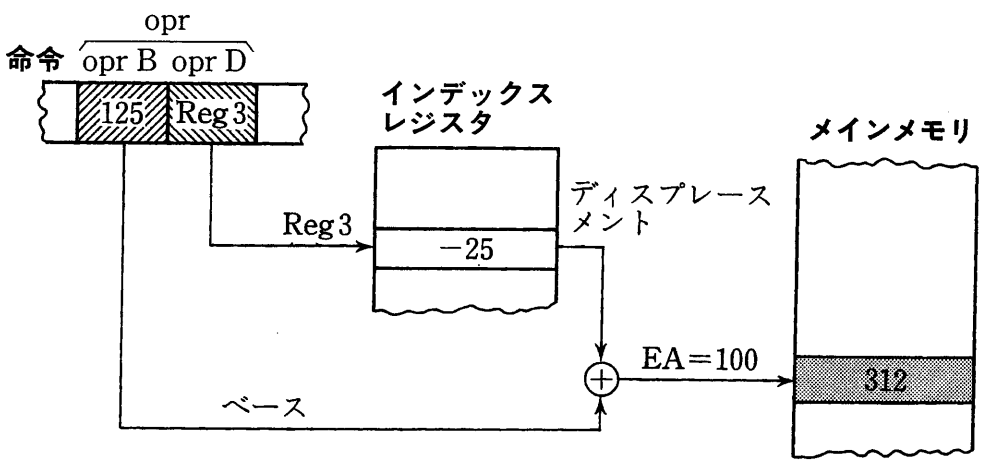

図 2.22　インデックスアドレス指定

アクセス対象＝Mem[EA]，EA＝opr B＋Reg[opr D]

静的(コンパイル時)に決める固定ベースアドレスと，動的(実行時)に計算・変更できる可変ディスプレースメントによる相対アドレス指定を行う。均質(同一属性)データの集まり(データブロック)の各データ要素への順次アクセスに**インデックス**(index；添字)を利用するので，ディスプレースメントを格納するレジスタを**インデックスレジスタ**という。インデックスレジスタには命令実行ごとにインデックスを指定数(普通はデータ長のバイト数)だけ増減する自動インクリメント(increment)/デクリメント(decrement)機能を装備する。① 汎用レジスタ(2.1.2項(b)の(1)参照)をインデックスレジスタとして流用する；② 汎用レジスタとは別に専用のインデックスレジスタを設ける；の2方式がある。図2.23に示すように，同一マシン命令をくり返し実行するたびに，等間隔に置いた均質データを順次オペランドとして指定できる。

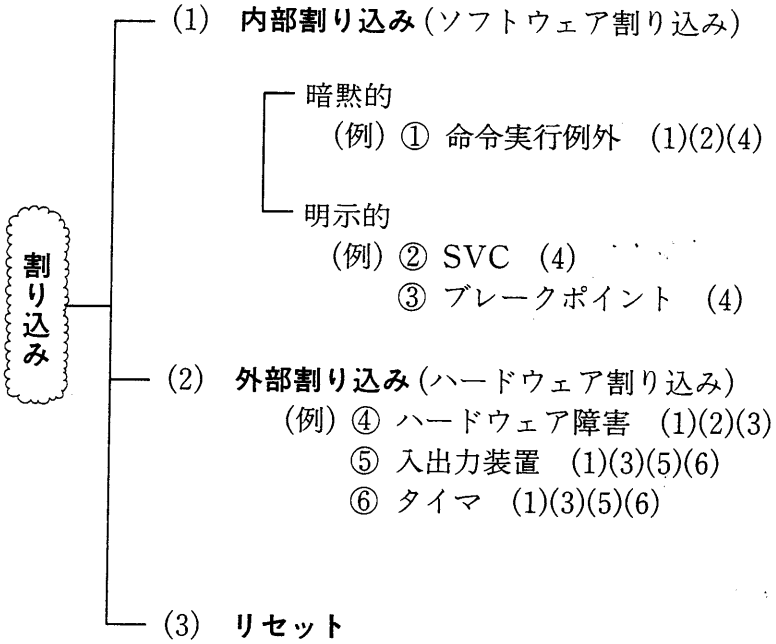

図 5.21 割り込みの分類

のでソフトウェア割り込みともいう。さらに内部割り込みはマシン命令機能として明示してある(明示的)かそうでない(暗黙的)かで細分類できる。次に列挙する内部割り込み要因例のうち，① は暗黙的であり，② と ③ は明示的である。

① **命令実行例外**：本来の命令機能以外の事象の発生による。命令実行サイクル(2.2.5 項参照)の命令フェッチ，命令デコード，オペランドフェッチ，命令実行，結果格納の各ステージで発生する。次のような具体的要因がある。

- メモリアクセス例外：**ページフォールト**(7.2.4 項(a)参照)や **TLB ミス**(7.2.3 項(c)参照)など。
- **アクセス保護違反**：プロセッサ状態がユーザモードでの特権命令の実行(“特権命令違反”という，5.2.3 項(i)参照)など。
- 不正命令：不正あるいは未定義の OP コード指定など。
- 不正オペランド：メモリアドレス境界違反など。
- **演算例外**：ゼロ除算(ゼロを除数とする除算)，オーバフロー(演算結果の格納装置からのあふれ)など。

② **SVC**(**スーパバイザコール**(SuperVisor Call)，**システムコール**(system call))：OS を呼び出してユーザプログラム(プロセス)から OS(カーネル)に制御を移すマシン命令 SVC の実行による。**トラップ**(trap)命令ともいう。

③ **ブレークポイント**(breakpoint)：ブレークポイント(プログラムの中断点)を OS に指示するブレークポイント命令の実行による。ブレークポイント

(e) イーサネットの規格

イーサネットは現代に最も普及しているLAN規格である。ベースバンド伝送方式(9.2.2項(a)の(11)参照)で，10 M～1 Gbpsの伝送速度を実現している。

“狭義のイーサネット”はOSI参照モデルの第1層(物理層)規格を指す。広義のイーサネットは第1層(物理層)とCSMA/CDによるアクセス制御などを規定する第2層(データリンク層)とを併せた規格を指す。普通“イーサネット”というとこの“広義のイーサネット”を指す。

イーサネットは1980年に米国のXerox, Digital Equipment (DEC), Intelの3社が共同でUNIXワークステーションに標準装備するLAN規格として策定している。その後，IEEEが**IEEE 802.3**として規格化している。IEEE 802.3では，物理層とその上位のデータリンク層および両層のインタフェースについて定めている。IEEE 802.3規格にしたがってイーサネットを分類してみよう。イーサネットは伝送速度によって次の(1)～(4)に大別でき，それぞれはさらに伝送距離や伝送媒体などによって細分できる。この分類は物理層仕様の相違による分類である。表9.1にも一覧にしてまとめてある。

表9.1 イーサネットの規格

通　称	イーサネット			ファストイーサネット	ギガビットイーサネット		テンギガビットイーサネット
規　格	10BASE5	10BASE2	10BASE-T	100BASE	1000BASE-T	1000BASE-X	10GBASE-X
IEEE規格	802.3	802.3 a	802.3 i	802.3 u	802.3 ab	802.3 z	802.3 ae
伝送速度(bps)	10 M	10 M	10 M	100 M	1 G	1 G	10 G
伝送媒体	同軸ケーブル	同軸ケーブル	UTP	-T2/4：UTP -TX：UTP, STP -FX：光ファイバ	UTP	-LX/SX：光ファイバ -CX：STP†1	光ファイバ
ネットワークトポロジ	バス	バス	スター	スター	スター	スター	スター
伝送距離(m)	500	185	100	-T2/4/X：100 -FX：412†2	100	-LX/SX：550†3 -CX：25	最大40 K
全2重通信	なし	なし	あり	-TX, -FX：あり	あり	あり	あり

(†2 全2重通信路：2 K)　(†1 平衡型銅線)
(†3 -LXのSMF：5 K)

（1） 仮数部を p だけ左(右)にシフトする。

（2） 指数部の値を p だけ減じる(加える)。

仮数部が純小数(整数)でない一般の浮動小数点数も同様に，シフト操作によって正規化することができる。

（c） 指数の数表現

コンピュータ内部での浮動小数点数表現2進数の指数(部)は固定小数点数と同じように，① 符号-絶対値表現；② 1の補数表現；③ 2の補数表現；のいずれかで表す。

ここで，図3.10に例を示すように，浮動小数点数表現の指数部に**バイアス**(bias，げた履き)**値**という整数定数を加えて，指数部には正整数だけを指定するようにしてしまう。これを**バイアス表現**あるいは**げた履き表現**という。バイアス値としてはそのコンピュータ内部で表現可能な最小指数(負数)の絶対値を選ぶ。これによって，指数(部)はゼロ以上の正整数で表現でき，符号ビットや補数表現は不要となる。コンピュータから演算結果を取り出すときなどの必要時に指数部からバイアス値を減じて実際値に戻せばよい。

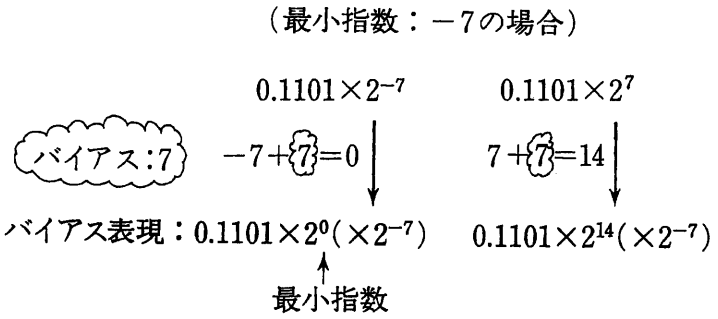

図 3.10 バイアス表現の例

指数(部)をバイアス表現(バイアス値は B)する場合，仮数 m，指数 e で2進数浮動小数点数表現した実数 R の10進数値は，

$$R=m\times 2^{e-B} \tag{3.27}$$

である。たとえば，指数部が8ビットで，それで表す最小指数が −127 ならば，式(3.27)において $B=127$ で，

$$R=m\times 2^{e-127} \tag{3.28}$$

となる。

（d） 浮動小数点数の範囲と精度

式(3.25)で表す正規化した浮動小数点数表現 r 進数の仮数 m

の範囲は，m を符号ビットなしの正の純小数(正規化による)で表現する場合

$$r^{-1} \leqq m < 1 \tag{3.29}$$

である。特に，式(3.26)と図3.7で表す正規化した浮動小数点数表現2進数の場合には，

$$0.5 \leqq m < 1 \tag{3.30}$$

である。r 進数，2進数いずれの場合も，仮数 m の範囲は仮数部の長さ p にかかわらず一定である。

浮動小数点数表現の仮数は“有効数字(有効桁，有効ビット)”を表している。2進数の場合，p ビットの仮数部では，2^p (概数，厳密には仮数の数表現で異なる)個の仮数が表現できる。仮数の範囲は式(3.29)や(3.30)で示したように一定であるから，仮数部の長さは一定の範囲にある仮数の個数，すなわち，それで表す浮動小数点数そのものの精度を決める。(b)で述べた“正規化”とは，“仮数部の有効数字(のビット数)すなわち精度を最も高く(最大に)する”ことでもある。

一方，浮動小数点数表現した実数 R そのものの範囲はその指数 e で決まる。指数部の長さを q ビット(図3.7参照)とし，e を符号ビットなしの正整数で表現する場合，e の範囲は $0 \leqq e \leqq 2^q - 1$ となる。このとき，実数 R の範囲は，式(3.30)によって，

$$0.5 \leqq R < 2^{2^q - 1} \tag{3.31}$$

となる。

仮数部の長さ p が24ビット，指数部の長さ q が8ビットの浮動小数点数表現で表す実数 R の範囲は，

$$0.5 \leqq R < 2^{255} \tag{3.32}$$

である。符号ビットなしで32ビット(1ワード)すべてを整数部とする固定小数点数で表現する正整数 N の範囲

$$0 \leqq N < 2^{32} \tag{3.33}$$

と比べると格段に広い(大きい)。

浮動小数点数表現の精度は仮数部の長さによって決まる。**単精度**とは仮数部と指数部とを合わせて1ワード(普通は32ビット)の浮動小数点数表現をいう。2ワード以上の場合は**多倍精度**(2ワードの場合は単に**倍精度**)という。

浮動小数点数表現においては，指数部の長さによって決まる範囲と仮数部の長さによって決まる精度との間にトレードオフ関係がある。すなわち，指数部を長く(短く，以下カッコ内に対応)すれば範囲が広く(狭く)なるが，一方で，仮数部が短く(長く)なって精度が低く(高く)なる。

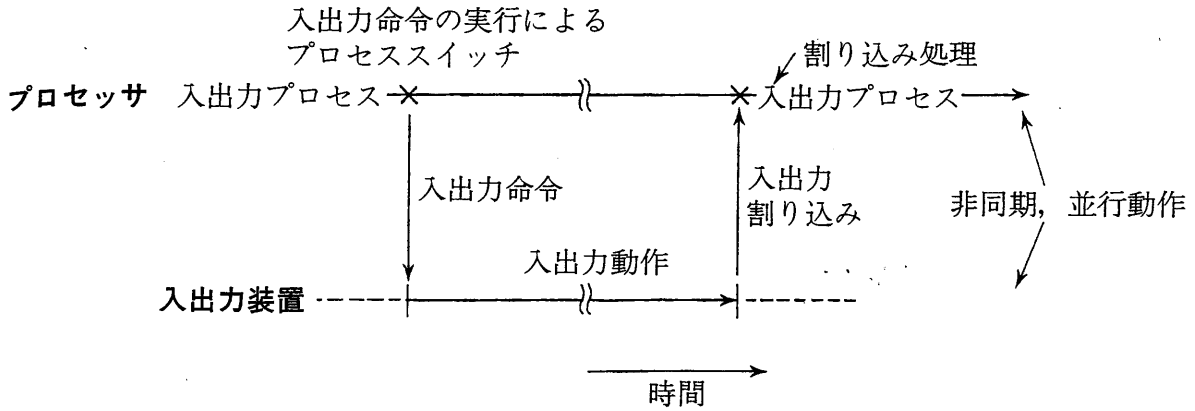

図 5.26 入出力割り込み

うマシン命令を実行すれば，そのタイミングで同期はとれる。これとは逆に，プロセッサからの入出力動作指令によって（プロセッサとは非同期に）動作している入出力装置がプロセッサに自分の状態を知らせるための手段が**入出力割り込み**である。

図 5.26 に示すように，入出力割り込みは入出力動作の完了や異常などの入出力装置の状態をプロセッサ側に通和するためにある。これによってプロセッサやメインメモリと入出力装置とが独立に並行して動作可能となる。

入出力装置を制御する（入出力動作を行う）プロセスを“入出力プロセス”という。プロセッサで実行する入出力プロセスを管理・制御する OS と入出力装置の動作（入出力動作）との関係を時間経過にしたがって説明すると次のようになる。

まず，プロセッサが入出力命令を実行すると，OS は，① 入出力命令を実行した入出力プロセス（＊）を実行中状態から実行待ち状態にする；② プロセススイッチによって別のプロセスを実行中にする（切り替える）；を行う。一方，入出力装置は，③ プロセッサからの入出力指令によって入出力動作を開始する；④ 入出力動作を完了するとプロセッサに対して入出力割り込みを起こしてそれを通知する；を行う。プロセッサ(OS)は，⑤ 入出力割り込みを受け付けて，その割り込み処理を始める；⑥ 入出力割り込み（という事象）を待っていた入出力プロセス（＊）を実行待ち状態から実行可能状態にする（ウェイクアップする）；を行う。③ と ④ の間の入出力動作中には，プロセッサは入出力装置とは非同期に（独立して）別のプロセスを実行できる。

入出力割り込みは“入出力装置が非同期に並行動作するプロセッサに同期をとってもらう”機能の実現である。入出力割り込みを用いた入出力制御機能の詳細については 8.2 節で述べる。

$$\bar{\bar{N}}=\bar{N}+1 \tag{3.20}$$

となる。ただし，式(3.20)では，$\bar{N}$ と $\bar{\bar{N}}$ は10進数値で表し，それぞれの符号ビット(MSb)もそれぞれの数値部と連結・一体化して考える。式(3.20)より，2進数表現したある整数 N の2の補数 $\bar{N}$ (2進数表現)は“N の1の補数 $\bar{\bar{N}}$ に $+1$ (1を加算)する”操作で得ることができる。

2進数表現した整数 N の2の補数 $\bar{N}$ (2進数表現)を求める手順(前述の(1)～(4))を式(3.20)にしたがって書き直すと，次のようになる。

(1) N の各ビットを反転($0 \Leftrightarrow 1$)して，まず，1の補数 $\bar{\bar{N}}$ を得る。

(2) $\bar{\bar{N}}+1$ ($\bar{\bar{N}}$ に1を加算する)によって2の補数 $\bar{N}$ を得る。

2の補数(表現)には，1の補数(表現)と比較して，① 加減算を行うときに補正は不要である(6.1.1項(f)で詳述)；という長所と，② いったん1の補数を得て，それに $+1$ の加算操作を行って2の補数を得るので，2の補数化機構では加算器が必要となる(6.1.2項(f)参照)；という短所がある。現代のコンピュータでは，②の短所よりも①の長所を重視して，“固定小数点数による負数の数表現は2の補数(表現)で行う”のが普通である。

(e) 固定小数点数の範囲と精度

n ビットの整数部(符号ビットを含む)と m ビットの小数部をもつ固定小数点数表現(2進数表現)では，補数表現を使うと，次の式(3.21)と(3.22)の範囲にある(すべてではなく有限個の)実数 R を表現できる。

(1) 1の補数表現による実数 R の範囲は

$$2^{-m}-2^{n-1} \leqq R \leqq 2^{n-1}-2^{-m} \tag{3.21}$$

となる。式(3.21)の範囲に，1の補数による固定小数点数表現で表せる2進実数 R は $(2^{n+m}-1)$ 個ある。なお，1の補数表現では“$+0$”と“-0”の表現が異なるが，数学的な違いはない。

(2) 2の補数表現による実数 R の範囲は

$$-2^{n-1} \leqq R \leqq 2^{n-1}-2^{-m} \tag{3.22}$$

となる。式(3.22)の範囲に，2の補数による固定小数点数表現で表せる2進実数 R は 2^{n+m} 個ある。

(1)(2)いずれの固定小数点数表現の場合でも，“精度は 2^{-m} で一定”である。

また，n ビットの2進整数の固定小数点数表現(整数表現)では，補数表現を使うと，次の式(3.23)と(3.24)の範囲にある(すべてのかつ有限個の)整数 N を表現できる。

(1) 1の補数表現による整数 N の範囲は

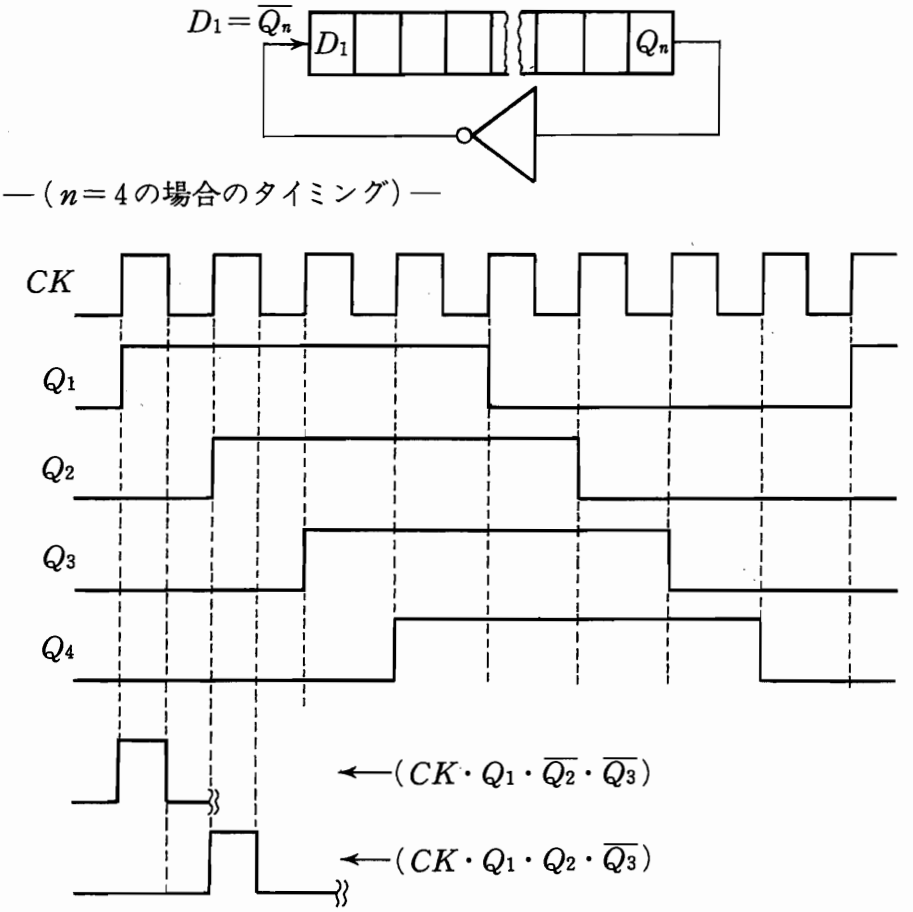

図 4.20　nビットジョンソンカウンタの構成と動作タイミング（n=4 の場合）

4.2.4　順序回路による基本ハードウェア機構の構成例

実際にコンピュータのハードウェア機構を構成する基本順序回路を紹介してみよう。

（a）　タイミング生成回路

コンピュータ（のハードウェア）の動作を制御（同期）するマスタ（原）クロックパルスから位相のずれたタイミングパルスを生成する回路を**タイミング生成回路**という。**タイミングパルス**はクロック周波数を分周した周波数をもつが，そのパルス幅はマスタクロックと同じである。

基本的なタイミングパルスは，前の 4.2.3 項 (d) で述べたジョンソンカウンタなどを利用して次のような手順で作ることができる（図 4.21 参照）。

（1）　周波数 f，デューティ比（“1”を示すパルスの時間と“0”を示すそれとの比）1：1のマスタクロックパルスを$2n$分周し，周波数$\frac{f}{2n}$，デューティ比1：1のパルスを作る。

（2）　このデューティ比を1：$(4n-1)$にし，マスタクロックのパルス幅と

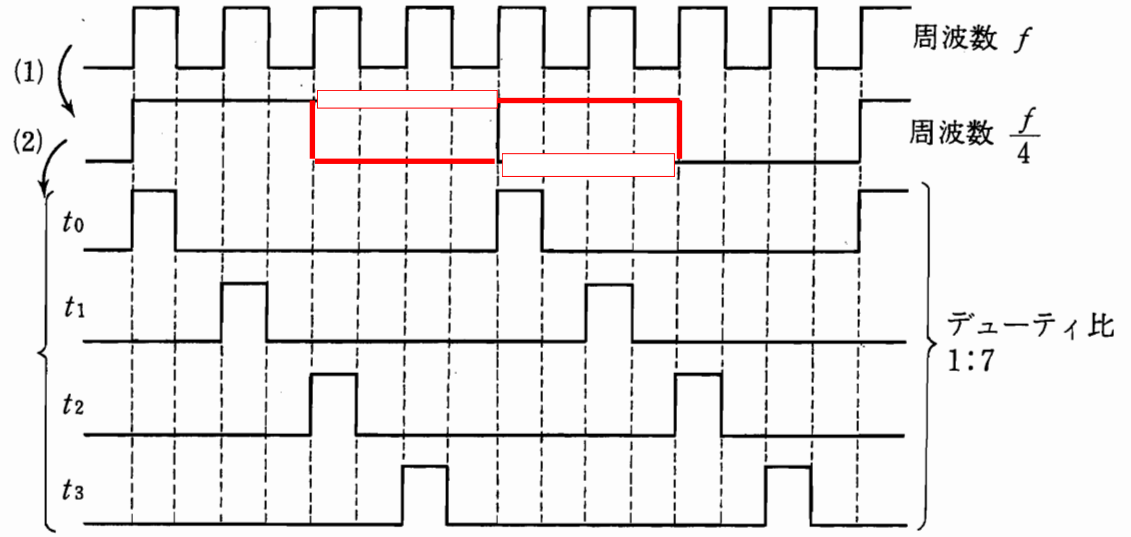

図 4.21 タイミングパルスの生成手順 ($n=2$)

同じにする。

マスタクロックから作るタイミングパルスはコンピュータの制御機構 (特に同期式制御機構，5.1.3 項 (c) 参照) のタイミング合わせ (同期動作) に使う。

周波数が f のマスタクロックの周期 (cycle；1 クロックサイクル) は $\frac{1}{f}$ 秒であり，これを**マシンサイクル** (machine cycle) という。マシンサイクルはコンピュータの動作を同期させるために使う最小単位時間 (間隔) である。

(b) パルスエッジ検出回路

クロックに同期していないパルスのエッジ (edge；立ち上がりと立ち下がり) を検出する回路を**パルスエッジ検出回路**という。非同期信号の同期化などのコンピュータの基本ハードウェア機構 (特に制御機構) の構成には不可欠である。

パルスエッジ検出回路は，図 4.22 に示すように，2 個の D フリップフロップを使用して構成できる。

(c) アービタ

“あるハードウェア機構や装置へ複数のアクセスが同時に生じる”ことを“アクセス競合”という。たとえば，メインメモリへプロセッサと入出力装置とが同時にアクセスする場合や，プロセッサ内の ALU とレジスタがバスを同時に使用する場合など，ハードウェア利用の多重化を図っている現代のコンピュータではアクセス競合が発生する回数は非常に多い。このアクセス競合をある戦略で調停し解決する制御機構を**アービタ** (arbiter，調停機構) という。アービタは順序回路の一種であり，非同期に生じる信号を順序付けする。