

心理学実験における MS-DOS (3)

— 音声データ録再生ライブラリ SB16I/O の作成¹⁾ —

下木戸 隆 司²⁾

MS-DOS は構造がシンプルでプログラミングが容易であることや、厳格な時間制御が可能であるといった点から、心理学実験にとって優れた特徴を有している (e.g., 下木戸, 2000, 2001)。しかしその反面、利用可能なメモリや一度に扱うことのできるデータのサイズに制限があり、動画や音声といった大容量のデータを扱う場合には効率が悪いという問題がある。それに加えて、PC-9800シリーズや PC/AT互換機の機体本体に備わっている音声処理装置が貧弱なため、人の声を高品質で録音したり、再生したりすることが難しいという問題もある。仮に高性能なサウンド・カードを増設してこうした欠点を補ったとしても、これらの装置にはその機能を利用するためのプログラム・インターフェースがユーザに開放されていなかったり、されていたとしても互換性の面で扱いづらいものであることが多く、ユーザが実際にこうした装置を制御するのはこれまでなかなか困難であった。そのため音声処理装置を自作して、そのプログラム・インターフェースを自身で用意するといったこともしばしば行われた。しかしこうした行為には相応の工学的知識を必要とするため、一般には敷居が高く、一部の人しか実行できなかった。

一方、音声データを扱いにくかったMS-DOSと比べると、Windows 環境ではかなり状況が改善されている。

1) SB16I/O が動作するためには、Creative 社の Sound Blaster 16が利用可能な状態になっている必要がある。SB16I/O を使用することによって生じた不具合に対して、筆者は賠償及び保障に関する一切の責任を負い得ないのでご了承いただきたい。技術的な問い合わせやバグレポートは、電子メールにて筆者宛 (E-mail: m47031a@cc.nagoya-u.ac.jp) に送付いただければ幸いである。SB16I/O の作成に際して、Ethan Brodsky 氏のドキュメントやソース・プログラムが大いに参考になった。ここに深く感謝いたします。

2) 名古屋大学大学院教育発達科学研究科研究生

Windows ではサウンド・カードのような音声処理装置を利用するためのプログラム・インターフェースがユーザーに提供されており、また音声の録音・再生を行うアプリケーションも豊富に用意されているため、音声処理の工学的知識や Windows の知識をあまり有さない初心者であっても比較的容易に音声データを扱うことができる。しかしその反面、MS-DOS が持っていたプログラミングの容易さやリアルタイム処理の実効性に関しては Windows ではほとんど省みられなくなってしまっている。Windows はシステム自体の構造が MS-DOS と比べるとはるかに複雑であり、ユーザの制御を受け付けない部分もあるため、厳格な時間制御が要求される実験用途には使用しにくいといった問題がある。

心理学実験という観点からみると、MS-DOS と Windows の長所・短所は好対照をなしている。MS-DOS は意図通りの制御が可能で厳格な時間制御が実施できる反面、音声や動画等のマルチメディア・データへの対応があまりよくない。Windows はマルチメディア・データの扱いが容易である反面、厳格な時間制御を苦手としている。つまり MS-DOS と Windows には、制御の精密さと利便性のトレード・オフが存在することになる。これらの点はどちらも重要であり、無条件にどちらかが優先されるべきものではない。しかしもし MS-DOS 環境で音声データを手軽に扱うことが可能になれば、MS-DOS と Windows 双方の長所を併せ持つことになり、便利であるに違いない。本稿ではそのような主旨から、MS-DOS 環境で比較的容易に音声の録音・再生を実行するプログラム・ライブラリの作成を試みるものである。

実際にパーソナル・コンピュータを用いて音声データを扱う際のやり方や注意点に関しては、三輪 (1991) や内田 (1993) に詳しく論じられている。本稿では、音声データがパーソナル・コンピュータでどのように扱われるのか、音声データを扱う装置にはどのようなものがあるのか、さらには心理学実験で音声データを扱う際の注意点に関しても議論を行う。ところで音声データをパー

ソナル・コンピュータで扱うためには、音の物理的特性をコンピュータで扱うことのできるコードに変換する必要がある。この手続きはデジタル化と呼ばれるが、音声データがどのようにデジタル化されるのかを知つておくことは心理学実験を行う際に有用であろう。音声データのデジタル化を取り上げる前に、本稿ではまず最初に音の特性について簡単に説明する。

音の特性

音を聴いたときに感じられるその心理的特性には強さ、高さ、音色がある。これらの心理的特性にはそれぞれ対応する物理的特性があり、音の強さには音圧が、音の高さには音の周波数が、音色には音の波形や音響スペクトルの分布等が関係することが知られている。

音の強さ

音は空気を媒質として伝達される。音源が振動することでエネルギーを受け、媒質である空気中の分子が変位すると、それによってその部分の空気の圧力が変化する。この変化は音の強さに比例して大きくなるため、測定の容易な空気圧の変化をもって音の強さの指標とすることが一般に行われている。この空気圧の変動分は音圧と呼ばれている。人間が聴取可能な音の範囲は膨大であるため、音圧の記述には対数尺度が用いられる。なかでも音圧と基準音圧の比の対数を取って20倍したもの($20\log_{10} P_v/P_0$)は音圧レベル(sound pressure level; SPL)と呼ばれ、よく用いられている。単位はデシベル(dBSPL)である。基準音圧(P_0)は0 dBSPLであり、空気圧でいうところの20マイクロ・パスカル(μPa)に相当する。人間の聴取可能な音圧は0~130 dBSPLの範囲であるとされている。

音の高さ

音は縦波であり、空気密度の粗密に応じて空気中の分子が振動することで伝達される。この振動は正弦波によって表現することができ、その際、振動は波の上下運動で表現されることになる。上下運動が一通り完了する単位はサイクルと呼ばれ、1サイクルに要した時間間隔は周期と呼ばれる。周期の逆数は音の振動数を示したもので、周波数と呼ばれている。単位はヘルツ(Hz)である。1 Hzは周期が1秒の振動数のことをいう。音の周波数と知覚される音の高さ(ピッチ)には密接な関係があり、周波数の低いものは低い音として聴こえ、周波数の高いものは高い音として聴こえる。人間の可聴周波数は20~20,000 Hzの範囲であるとされている。

音色

音はその構造によって純音と複合音に分けられる。純音は空気の圧力が正弦波的に変化するもので、音叉を叩いたときに得られる音が代表的である。純音は構造が単純で基本的なものであるため、心理学実験では利用されることが多い。複合音は純音以外の音を指し、周期性の有無によってさらに二つに区分される。周期性を有するものは楽音と呼ばれ、周期性をもたないものは雑音と呼ばれている。なかでも無規則な振動が同じ強さで一律に含まれるものは白色雑音と呼ばれ、こちらも心理学実験ではしばしば利用されている。複合音は構造が複雑であるが、フーリエ変換を施すことで正弦波の組み合わせによって表現することが可能になる。そうして分解された周波数成分はスペクトルと呼ばれる。離散的フーリエ変換を時間的に連続して行い、時間、周波数、強さの三次元をグラフで表したもののがスペクトログラムであり、音響分析ではよく利用されている。

音声データのデジタル化

音は連続的なデータであるため、コンピュータ内に取り込むためにはそれを離散化した数値で置き換える必要がある。これをデジタル化という。デジタル化には、音声データを一定の時間ごとにアナログ・デジタル変換し、一定のビット数のデータに置き換えるというやり方が採られる。具体的には、どのくらいの頻度でデータを取り込むのか(標本化)、またどのくらいの細かさでデータを置き換えるのか(量子化)、どのような形式で記録するのか(符号化)、といった事項が問題になる。音声データをデジタル化することでコンピュータ資源を活用でき、高度な計算力を必要とする音響分析を行ったり、音声刺激提示のタイミングを精密に制御したりすることが可能になる。

標本化

電気信号は一定間隔ごとに離散化されるが、その際のデータを取り込む頻度を標本化周波数と呼ぶ。この値は1秒間にどれだけの割合でデータの取り込みを行うかを示したものである。標本化周波数の逆数は標本化が行われる時間間隔を示しており、周波数が高くなるほど時間間隔は短くなる。この時間間隔は標本化周期と呼ばれている。仮に標本化周波数を10 kHzとすると、これは1秒間に10,000回の割合で、つまり0.1ミリ秒間隔でデータを取り込むことを意味する。標本化周波数が高くなればなるほど、いいかえれば標本化周期が短くなればなるほど、元の音声を再現できる能力が増すものの、その分データの容量が大きくなるという性質がある。

取り込まれる音声の周波数が標本化周波数よりも高い場合、標本化周波数の1/2の値（ナイキスト周波数）より上の高周波成分が低周波成分に紛れ込んでしまい、結果的に意図しない信号が含まれることになる。これはエイリアシング・ノイズと呼ばれ、音声を取り込む際の大きな問題となる。逆に、元の音声の周波数がナイキスト周波数より低いものであれば、元の音声を忠実に再現できることは標本化定理によって保証されている。そのため実際に音声データを取り込む際には、意図する標本化周期の2倍以上の値を用いてナイキスト周波数より下の低周波成分の情報を保持すると同時に、低域通過フィルタを用いて高周波成分を除去する方法が用いられている。

量子化

標本化によって一定間隔ごとに取り込まれた電気信号を有限の桁数の数値に変換することをいう。標本化はデータを取り込む際の頻度を決定するものであるのに対し、量子化はデータを置き換える際の範囲の大きさを決定するものである。実際にはビット数で示される。仮に量子化ビット数を16ビットとすると、これは 2^{16} (0~65,535)の範囲の数値で元データを置き換えることを意味する。量子化ビットの値が大きくなるほど、肌理の細かな置き換えが行われることとなり、分解能が高くなる。それによって元の音声が忠実に再現される能力も上がるが、その分データの容量も大きくなる。

連続した電気信号を有限の数値で置き換える過程で必然的に誤差が生じることになる。この誤差は量子化誤差と呼ばれる。量子化誤差は量子化ビットが大きくなるほど小さくなるが、電気信号が微弱な場合、入力ゲインを調整してアナログ・デジタル変換器が要求する範囲に適合させることによっても小さくすることができる。

符号化

量子化されたデータを単純に0/1の2進数によって記録する方法はパルス符号変調（Pulse Code Modulation; PCM）と呼ばれ、音楽CD等に利用されている。この際にデータ圧縮技術が適用される場合もあり、例えばMPEG (Motion Picture Experts Group) オーディオ形式は、高压縮率・高品質の音声ファイル・フォーマットとして広く利用されている。符号化時に圧縮が行われないものはリニアPCMと呼ばれる。

音声ファイル・フォーマット

コンピュータによって取り込まれた音声データは、実際にはファイル形式で保存されることが多い。こうしたファイル形式には様々なフォーマットがあり、WAVE

形式(WAV)やAIFF形式(AIF), Creative Voiceファイル形式(VOC), MPEGオーディオ形式(MP3), Realオーディオ形式(RA)等が代表的である。これらの音声ファイルは、対応している量子化ビット数やデータを符号化する方法に違いが見られるものの、ファイル構造では比較的共通しており、ヘッダ部とデータ部に区分することができる。ヘッダ部では標本化周波数や量子化ビット、チャンネル数といったデジタル化された音声データの品質に関する情報が記録され、データ部には実際にデジタル化された音圧データが時系列的に記録されている。

音声データの入出力

音声入出力装置

コンピュータは実際にどのようにして音声データを扱っているのだろうか。いいかえれば、どのような過程を経て音声データが再生されたり、録音されたりするのだろうか。音声データを扱う装置には、マイクやスピーカのような電気音響変換器や低域通過フィルタやアナログ・デジタル変換器のようなデジタル信号処理装置等がある。サウンド・カードと呼ばれる音声入出力装置はアンプやアナログ・デジタル変換器やデジタル・アナログ変換器を備えており、安価ながらも比較的高品質な音声処理を行うことができるため、パソコン・コンピュータの拡張機器として広く利用されている。最近ではPC/AT互換機のチップ・セットに最初から用意されているものも存在する。

以下にこれらの装置について簡単に説明する。

マイク 音圧を電気信号に変換する装置である。この装置には構造面から、コンデンサ・マイクとダイナミック・マイクに大別される。コンデンサ・マイクは対応する周波数の範囲が広く、周波数特性が平坦であるという利点があるものの、振動や湿度に弱いという欠点がある。ダイナミック・マイクはコンデンサ・マイクに比べると対応する周波数の範囲が狭い反面、比較的安価で壊れにくいという利点がある。

アンプ 入力される電気信号が微弱な場合、それを増幅する手続きがある。これを行う装置がアンプであり、用途の違いによって、マイク・アンプ(プリ・アンプ)やパワー・アンプ(メイン・アンプ)等と呼ばれている。マイク・アンプは、マイクで変換された電圧(数mV程度)をライン・レベル(200mV~2V程度)に増幅するものであり、パワー・アンプはライン・レベルの電圧をスピーカ・レベル(数V~数百V)にまで増幅する。

低域通過フィルタ (Low Pass Filter; LPF) 標本

化の際に生じるエイリアシング・ノイズを取り除く目的で使用される。低域通過フィルタは指定された周波数（遮断周波数）より下の低周波成分を通過させ、それより上の高周波成分を遮断する。このようなフィルタにはいろいろなものがあるが、通過域が平坦なバーワース・フィルタや遮断周波数からの優れた減衰特性を有する連立チェビシェフ・フィルタ等がよく用いられている。

アナログ・デジタル変換器 (Analog-to-Digital Converter; ADC) 連続的に変化する電気信号を一定間隔ごとに逐次数値に変換する装置である。量子化の際の分解能や入力される電気信号の電圧範囲、同時変換可能なチャンネル数の違い、変換する際の方法の違い等によって様々な種類がある。

デジタル・アナログ変換器 (Digital-to-Analog Converter; DAC) ADCとは逆に、数値データを連続する電気信号に変換する装置である。これも出力される電圧の範囲や同時変換可能なチャンネル数の違い、変換方法の違い等によって様々な種類がある。

スピーカ 電気信号を機械的な振動に変換し、音圧として空気中に放射する。心理学実験ではダイナミック・スピーカと呼ばれるものが用いられることが多い。口径の大きいものほど低い音を再生するのに適しており、小さいものほど高い音を再生するのに適している。スピーカには強力な永久磁石が使用されているため、磁力の影響を受けやすいものを近づけないようにする必要がある。

ヘッドホン 耳介に直接接触させて音声を伝達させるものである。これらのうち耳孔に挿入して使用するものはイヤホンと呼ばれる。ヘッドホンは音源からの距離を一定にできる利点があるため、心理学実験ではよく利用される。

音声データはどのように録再生されるのか

ここでは音声データがパーソナル・コンピュータでどのように取り込まれ、どのように再生されるのか、具体的に例を挙げて説明する。Figure 1にその概略を示す。パーソナル・コンピュータでは音声処理を行う装置が機体本体に実装されていないケースがあり、多くの場合、音声処理はサウンド・カードと呼ばれる拡張機器が代行することになる。サウンド・カードはコンピュータ本体の拡張バスに挿入されることで、コンピュータ本体のI/Oを介して情報をやり取りする。

まず人によって発せられた音声は、マイクを通して電気信号に変換される。この電気はADCが扱うことできる電圧の範囲に対してあまりにも微弱なので、アンプによって増幅された状態でADCに送られる。送られた電気信号はADCによってデジタル化されるが、この時

点ではエイリアシング・ノイズが混入された状態であるため、LPFによって不必要的高周波成分が除去される。その後、ADCによってデジタル化されたデータはコンピュータ本体のI/Oを介してメモリやCPUに転送される。その際には、DMA (Direct Memory Access) 転送と呼ばれる方法が用いられることが多い。この方法は、CPUを介すことなく直接メモリに情報を転送するもので、大容量のデータを高速に伝達することができるという利点がある。これに対して、CPU自身が直接データをサウンド・カードから読み込むことも可能であるが、この方法はCPUに負荷をかけ、標本化周波数の高い大容量のデータには向いていないため、今日ではほとんど使用されていない。そうしてメモリに送られたデータはその後、CPUによってフロッピー・ディスクやハード・ディスクに書き込まれ、音声データの録音が完了する。

一方で、音声データの再生は録音の逆の手順を踏む。まずCPUによってファイルからメモリに読み出された音声データは、DMA転送によってサウンド・カードに直接送られる。デジタル化されたデータはDACによって連続データへと変換される。DACによって変換された電気信号はLPFによって高周波成分が除去された後、必要に応じて増幅される。そうして最終的にスピーカやヘッドホンに送られ、実際の音声として再生される。

以上に述べたような手順を経て、音声データは録音されたり、再生されたりする。サウンド・カードはこれらの音声入出力装置を搭載し、音声処理に特化したものである。なかでもCreative社のSound BlasterシリーズやNEC社のPC-9801-86等が有名である。サウンド・カードが実際にどのようなものであるのかを説明するため、次にCreative社のSound Blaster 16について取り上げる。本稿で紹介するプログラム・ライブラリであるSB16I/Oは、このSound Blaster 16を利用して音声データの取り込みや再生を行うものである。

Sound Blaster 16

Sound Blaster 16は1992年にCreative社によって販売されたサウンド・カードである。現在では製造が終了されているものの、PC/AT互換機の事実上の標準装置として広く普及したため、今日でも入手は比較的容易である。主な特徴としてはFM (Frequency Modulation) 音源とPCM音源を搭載し、8ビットと16ビットの分解能を有すること、5kHz~44.1kHzまでの標本化周波数に対応していること、2チャンネルの音声を扱うことができ、ステレオ音に対応していること、ISA (Industry Standard Architecture) バスに挿入して

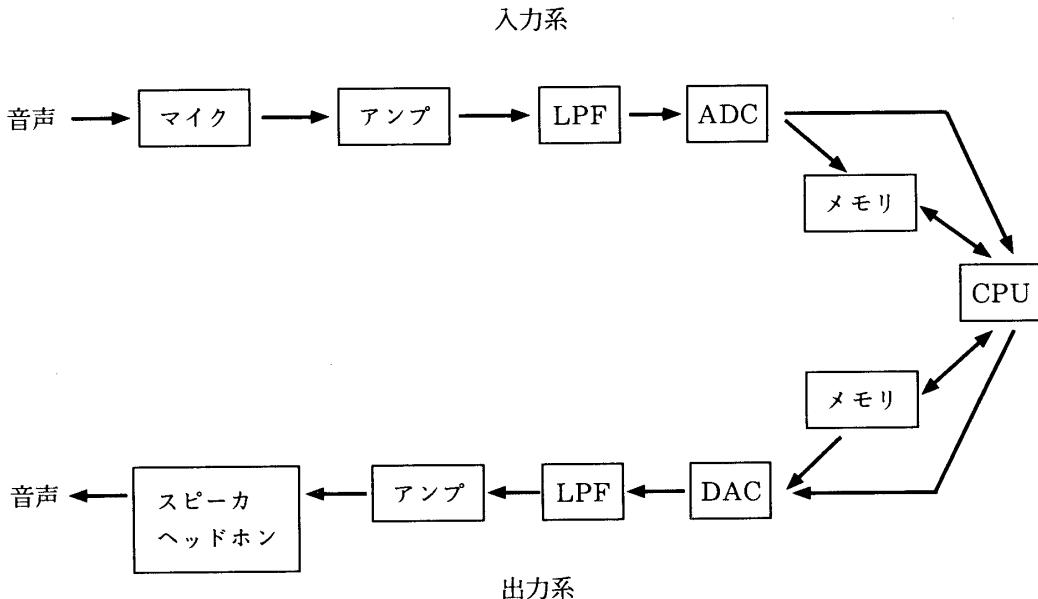


Figure 1 音声データの入出力の流れ

使用すること等が挙げられる。サウンド・カードは大きく分けて、音声処理チップ、コーデック、入出力端子、音源チップから構成されているが、以下に Sound Blaster 16が搭載しているこれらのパーツについて簡単に述べる。

Sound Blaster 16は、“Advanced Signal Processor (ASP)”と称する音声処理チップを搭載している(Heimlich, Golden, Luk, & Ridge, 1994)。この ASP はコンピュータ本体の CPU にかわって音声処理を行うものであり、一般にいわれるところの DSP (Digital Sound Processor) あるいはデジタル・コントローラに相当する。コンピュータ本体とのデータのやり取りは I/O ポートを介して行う。本稿で紹介する音声データの録音再生プログラム・ライブラリである SB 16I/O は、この ASP を制御することによって音声データの取り込みや再生を行っている。

Sound Blaster 16内部には音声データのデジタル化やデジタル・データのアナログ化を行う ADC および DAC が搭載されている。通常はこれらは一つのチップに収められており、コーデック (COder-DECoder; CODEC) もしくはアナログ・コントローラと呼ばれている。コーデックの性能はサウンド・カードが扱える音声の品質にダイレクトに影響するが、Sound Blaster 16のコーデックは 8 および 16 ビットの分解能と 5 kHz~44.1 kHz の標準化周波数に対応している。

Sound Blaster 16の入出力用の端子は入力、出力に 2

本ずつ用意されており、ライン・イン、マイク・イン、ライン・アウト、スピーカ・アウトで構成されている。実際の録音・再生時にはこれらの端子にマイクやスピーカのプラグを接続して使用する。マイクやスピーカのボリュームや入力ゲイン等の設定は Sound Blaster 付属の MIXERSET.EXE を起動して行うと便利である。

Sound Blaster 16は FM 音源チップを搭載しており、22音までの同時発生が可能である。この FM 音源は MIDI (Musical Instrument Digital Interface) データの演奏にも使用されている。FM 音は正弦波の周波数を変調させることで様々な音声を産みだすものであるが、歴史的にはこの FM 音源を搭載することでサウンド・カードが普及し、発展してきたともいえる。

Sound Blaster 16はベストセラーとして長期間にわたって販売されたためか、様々なバリエーションが存在する。WAVE テーブル・シンセサイザーの有無、CD-ROM 用の IDE (Integrated Drive Electronics) や SCSI (Small Computer System Interface) インターフェースの有無、プラグ・アンド・プレイへの対応の有無や、音声処理チップ自体にも種類によっていくつかの違いがある。そのため、一概に “Sound Blaster 16” と一括りにできない面もあるが、基本的な機能に関してはほぼ同等であるため、本稿ではこれらの違いをとくに区別しないこととする。

ところで Sound Blaster 16にはユーザを悩ませる問題も知られている。なかでもシステム・リソースとの相

性の問題や録音時の音質劣化の問題は有名である。Sound Blaster 16はシステム・リソースとの相性が激しく、特定の割り込み番号（IRQ）やDMA チャンネルが割り当てられていないと正常に認識されない場合がある。その場合にはこれらのリソースを最優先して Sound Blaster 16に割り当てなければならない。録音時に音質が劣化する問題に関しては、これは Sound Blaster 16の回路上にスピーカ用のアンプが搭載されているため、ここから発生したノイズが録音時に音声データに混入することで生じる。これらの問題はいずれも Sound Blaster 16のハードウェア構造に起因するものである。

心理学実験で音声データを扱う際の注意点

音声データを心理学実験で扱う際には、周囲の騒音や電磁波から遮蔽された環境で実験を行うことが肝要である。音の知覚に影響を及ぼす要因、例えば音の大きさ、高さ、音色に関しては、それぞれに対応する音圧レベル、周波数、音の波形や周波数のスペクトルを統制し、一定にする必要がある。実際には、これらの心理的特性は完全に統制しきれない面もあるが、実験者はこうした要因が極力一定になるような環境を設定し、実験を行うべきであろう。その他にも音声データの使用用途によって留意しなければならない点がある。以下に、音声データを聴覚刺激として用いる場合と反応指標として用いる場合について論じる。

刺激として用いる際の問題

音声データを実験刺激として使用する場合は、刺激そのものの音以外のものが混入しないように配慮することが重要である。こうした余計な音の混入は比較的容易に生じうる。例えば、使用するスピーカやマイクが劣化していれば、それによって意図しない音の歪みが生じる。刺激そのものの提示時間が短ければ、音の鳴りはじめと鳴り終わりに伴って生じるクリック音が刺激音と重なって提示されてしまう。録音時の不注意によってノイズが刺激音に混入してしまうこともある。刺激音そのものが純粋に提示されることは音声装置の性質上まずありえないが、実験者が十分に配慮することで余計な音が混入り、刺激音の音響スペクトルが変化してしまうのある程度おさえることができる。音響スペクトルの変化は知覚される音の印象を容易に変化させてしまうため、それが起こらないように注意する必要がある。

次に音声データの提示時間に関してである。人の聴覚系は短時間の応答を苦手としているため、雑音ではない意味のある音として知覚されるためには最低でも100ms

以上の音を提示する必要があるといわれている。また提示時間が短い場合、先に述べたように音の鳴りはじめと鳴り終わりに発生するクリックが刺激音と重なってしまい、刺激音の音響スペクトルに大きな影響を及ぼしてしまうことも留意しなければならない。

最後に、マイクを通して音声刺激を作成する場合、周囲のノイズと一緒に取り込んでしまわないよう努める必要がある。マイクは感度が高く、周囲の音や電磁波の影響を受けやすいため、意図通りの音質が得られない可能性がある。とくに無音状態ではこうした影響を受けやすいので注意されたい。最近では刺激音をコンピュータで生成し、編集するケースも増えてきているが、発話などの自然音をコンピュータに取り込む際には、余計なノイズが刺激音に混じらないように配慮すべきである。

反応として用いる際の問題

心理学実験ではしばしば、提示された語を音読させたり、提示された図や色の名称を呼称させたりして、その際の反応潜時を測定することが行われる。この場合、発音された音のエネルギーがある一定の値に達した時点を持って反応潜時とすることが多い。それを行う装置は一般にボイス・キーと呼ばれている。しかしながらボイス・キーには問題も多く、反応潜時の測定装置としては必ずしも適切ではないことが指摘されている (e.g., Kello & Kawamoto, 1998; 都築, 1999)。

まずボイス・キーは呼気や咳、どもり、いいまちがい等の影響を受けやすい。こうした要因は正確な反応潜時の推定を妨げるため、これらが含まれる試行は実際の研究では分析から省かれることが多い。データの精度を確保するためにはこれらの要因は重大な問題となる。対策として、例えば呼気に関していえば、これが問題になるのはマイクの感度が高すぎる場合が多いことから、マイクの位置や入力ゲインの設定を適切なものに調節しておくことでだいぶ改善することができる。咳やどもり、いいまちがいに関しては、被験者がこうした発声を安易に行わないように教示や練習段階でチェックしておくことが重要である。

これらの他に発音される音素特有の問題もある。具体的には、日本語では“さ行”や“は行”に代表される無声摩擦音 (/θ/, /s/, /ʃ/, /ç/, /h/) や、“か行”に代表される無声破裂音 (/p/, /t/, /k/), “ら行”に代表される弾音 (/r/) の反応潜時が過剰に推定されてしまうことが知られている (佐久間・伏見・辰巳, 1997; 都築, 1999)。無声摩擦音では発音された音の音圧レベルが低いため、ボイス・キーが音の立ち上がりを検出にくく、結果的に遅延が生じてしまう。それに対し破裂

音では、発音するために声道内に呼気をため込む必要があり、その間は無音状態が続くため、ボイス・キーの検出が遅れてしまうのである。こうした問題を回避するために、従来の研究では刺激の語頭音素を条件間で均一にしたり、一定の時間間隔をおいてから音読させたりするといった手法が用いられてきた。それらとは別にボイス・キーに頼らない反応潜時測定の方法もある。音声データをデジタル化してそのままコンピュータに取り込み、音響分析にかけるやり方はその一つである (e.g., Kello & Kawamoto, 1998; 都築, 1999)。音の立ち上がりを一度検出したら残りのデータを捨て去ってしまうボイス・キーに対し、音声データを取り込む方法はその後何度も分析にかけることができるため、反応潜時推定の再現性という点からより望ましいといえる。

心理学実験では、発話プロトコルを得る目的で音声データを記録する場合もある。その際も、周囲の騒音や電子機器の電磁波の影響をなるべく受けないように配慮することが重要である。

SB16I/O

本稿で紹介する SB16I/O は、Creative 社のサウンド・カード Sound Blaster 16を利用して音声データの録音や再生を行う。そのため SB16I/O が動作するためには Sound Blaster 16 のドライバや設定ツールが正しくインストールされている必要がある³⁾。Sound Blaster 16 の割り込み番号 (IRQ) や DMA チャンネル、I/O ポートといったリソース・パラメータは SB16I/O のなかで指定する (付録 1 参照)。このパラメータが正しく指定されない限り、SB16I/O は動作しないので注意されたい。

SB16I/O は 5 kHz~44.1 kHzまでの標本化周波数に対応しており、8あるいは16ビット・モノラルの音声データを扱うことができる⁴⁾。ただし実際に利用できるファイルは RAW 形式のものに限られる。RAW 形式は一般的にはあまり利用されていないファイル・フォーマットであるが、構造がシンプルなため、プログラム上で扱いやすいという利点がある。

3) Sound Blaster 16 のドライバや、インストールの仕方を記述したドキュメントは Creative 社の Web サイト (URL: <http://www.creative.com/>) から入手することができる。必要に応じて参照されたい。

4) ただし version 1.0 の時点では PC-9800 シリーズには対応していない。現段階では SB16I/O の利用は PC/AT 互換機環境だけに限られる。

利用の仕方

SB16IO.C ファイル (付録 1 参照) を C プログラム中でインクルードすることによって、SB16I/O が提供する機能を利用することができます。コンパイラは Borland 社の Turbo C/C++ version 4.0J を想定しているが、以前のバージョンや、必要な修正を施せば他のコンパイラでも利用可能かと思われる。コンパイル時のメモリ・モデルはとくに限定していない。

例として、音声データの録音・再生を行う SBTEST.C (付録 2 参照) を取り上げて説明する。このプログラムは音声データの取り込みと再生を 5 回分だけ行うものである。プログラムをコンパイルするには、コマンドライン上で “TCC SBTEST.C” と入力してやればよい。正しく行われれば “SBTEST.EXE” という実行ファイルが生成されているはずである。

使用上の注意

前に述べたように、SB16I/O が扱うことのできる音声ファイルは RAW 形式のものに限られる。それ以外の音声ファイルには対応していないので注意していただきたい。そのため RAW 形式以外の音声ファイルを刺激として用いる場合には、事前にファイル・フォーマットを RAW 形式に変換しておく必要がある。こうしたファイル形式の変換には、MS-DOS 環境では SOX 等の音声ファイル・コンバータが便利である。

次に音声データの音質に関してであるが、SB16I/O では Sound Blaster 16 自体の構造上の問題のために、音声処理中にノイズが混入することがある。とくに 8 ビットでの録音・再生時にはこうしたノイズが入り込みやすいので、音質にこだわるのであれば 16 ビットで音声処理を行うことをおすすめする。

SB16I/O は音声データの入出力に際し、割り込み処理を行っている。そのためミリ秒単位のタイマ・プログラムである DOSTIMER のように、比較的短時間の間隔で割り込みを行うものと SB16I/O を併用する場合には、音声処理が意図通りに行われない可能性がある。そのような状況で音声処理を行う場合には、必然性がとくにないのであれば、他のプログラムの割り込み処理を停止しておく必要があるかもしれません。

以上、SB16I/O を利用する上でとくに留意すべき点について取り上げたが、その他の細かな注意点に関しては、SB16I/O に付属するドキュメント・ファイルを参照していただきたい。

Table 1 SB16/O が提供する主な関数

ファイルの読み込み／保存	
int LoadRawData (unsigned char sndbit, char *filename)	Raw ファイルの読み込み
int SaveRawData (char *filename)	Raw ファイルの書き込み
録音／再生	
int RecordData (unsigned char sndbit, float rectime, unsigned int samplefreq)	録音開始
int PlayData (unsigned int samplefreq)	再生開始
音声処理制御	
void StopSoundProc (void)	録再生終了
void RestartSoundProc (void)	(前の設定と同じで) 録再生開始
void PauseSoundProc (void)	録再生一時中断
void ContinueSoundProc (void)	中断解除
void WaitSoundProc (void)	録再生終了待ち
void ShutdownSB16 (void)	SB16I/O の終了 (必須)

討 論

本稿では MS-DOS 環境で音声データを扱う方法や、その際の注意点について論じてきた。しかし本稿で紹介した方法は、旧式のサウンド・カードを用いるものであるため、高品質の音声データを録音したり、再生したりする必要がある場合には最適なものであるとはいえない。音声処理を取り巻くハードウェアの変化はめざましく、今日では音楽 CD や DAT (Digital Audio Tape) の音質を優に上回る 24 ビット 96 kHz の録音・再生が可能なサウンド・カードや、機体本体のチップに音声処理装置を搭載した PC/AT 互換機も登場しており、音声データを扱う際の選択肢は以前よりもはるかに増している。使用する音声データはそれほど高音質でなくともよいが、時間制御には厳格なものを要求するのであれば、本稿で取り上げた方法は最適なものとなりうるかもしれない。しかし音質、時間制御ともに質の高いものを求めるのであれば、MS-DOS 環境下で高性能な DSP ボードを介し、音声データの入出力を行うという方法もありうる。最終的にどういった装置や方法を用いるのかは、実験の性質や目的と照らし合わせ、ユーザ自身が選択することが重要である。

引 用 文 献

- Kello, C.T., & Kawamoto, A.H. 1998 Runword: An IBM-PC software package for the collection and acoustic analysis of speeded naming responses. *Behavior Research Methods, Instruments, & Computers*, 30, 371-383.
- Heimlich, R., Golden, D.M., Luk, I., & Ridge, P.M. アスキー書籍編集部(訳) 1994 SOUND BLASTER オフィシャルブック アスキー出版社
- 三輪譲二 1991 パソコン活用シリーズ8 パソコン音声処理 昭晃堂
- 佐久間尚子・伏見貴夫・辰巳格 1997 音声波の観察による仮名の音読潜時の測定—音読潜時は語頭音の調整法により大きく異なる— 神経心理学, 13, 126-136.
- 下木戸隆司 2001 心理学実験における MS-DOS—ミリ秒単位のタイマ・ルーチン DosTimer を中心として— 名古屋大学大学院教育発達科学研究科紀要(心理発達科学), 48, 315-341.
- 下木戸隆司 2002 心理学実験における MS-DOS(2)—ビットマップ・ファイル表示ライブラリ BMPLOADeR の作成— 名古屋大学大学院教育発

資料

達科学研究科紀要（心理発達科学），49，247-276.
竹川忠男 1973 聴覚実験法 大山正（編） 心理学研究法 2 実験 I 東京大学出版会
都築誉史 1999 音読データの音素レベルにおける分析—認知心理学における音読潜時測定の諸問題— 応用社会学研究（立教大学社会学部紀要），41，51-57.

内田照久 1993 音声を扱うためのパソコン・コンピュータを用いた心理学実験装置—音声言語認知、音楽認知、両耳分離聴取、クロス・モダリティの研究のために— 名古屋大学教育学部紀要教育心理学科，40，227-237.

(2003年9月30日 受稿)

ABSTRACT

Using MS-DOS to Conduct Psychological Experiments: III. Sound Files I/O

Takashi SHIMOKIDO

Sound data have been widely used not only as experimental stimuli, but also as behavior indices. This article introduced a MS-DOS program routine, which was able to record and play back 8/16 bit sound data. This program (i.e. SB16I/O) was required for Sound Blaster 16 as a sound processing device and available to be called by C programs. In addition, issues of sound processing and handling sound data in psychological experiments were discussed.

付録 1 SB16IO.C (version 1.0)

```
=====
心理学実験用音声データ入出力ルーチン
SB16I/O version 1.0 for (Borland Turbo C)
2003/09/26 by Takashi Shimokido.

コンパイル：
tcc *.c

=====

#include <alloc.h>
#include <conio.h>
#include <dos.h>
#include <mem.h>
#include <stdlib.h>
#include <stdio.h>

#define lo(value) ((unsigned char)((value) & 0x00FF))
#define hi(value) ((unsigned char)((value) >> 8))
#define TRUE    1
#define FALSE   0
#define FAILURE -1

#define SAVE_CHUNK_SIZE 8192
#define LOAD_CHUNK_SIZE 8192
#define BLOCK_LENGTH    256

/* Sound Blaster 16 用リソース */
/* この中身は Sound Blaster 16 の設定に準じます */
/* 必要に応じて適宜書き換えてください */

#define BASEIO 0x220          /* Sound Blaster 16 I/O ポート・アドレス */
#define IRQ    2                /* 割り込み番号 */
#define DMA16  5                /* 8 ビット DMA チャンネル */
#define DMA8   3                /* 16 ビット DMA チャンネル */

/* -----
 * ===== 変数の宣言 ===== */
/* 割り込み処理用変数 */
volatile long intcount;
volatile int done;
volatile char curblock;
volatile long samplesremaining;

/* Sound Blaster16 DSP I/O ポート */
int resetport;
int readport;
int writeport;
int pollport;
int poll8port;
int poll16port;

/* Sound Blaster16 DSP コマンド */
char pause_command;
```

資料

```
char continue_command;
char exit_command;

/* 割り込みコントローラ I/O ポート etc */
int pic_rotateport;
int pic_maskport;
char int_controller;
char irq_intvector;
char irq_stopmask;
char irq_startmask;
char irq_intvector;

/* DMA コントローラ I/O ポート etc */
int dma_maskport;
int dma_clrprrport;
int dma_modeport;
int dma_baseaddrport;
int dma_countport;
int dma_pageport;
char dma_startmask;
char dma_stopmask;
char dma_mode;

/* DMA 転送用アドレス */
unsigned long buf_addr8;
unsigned char buf_page8;
unsigned int buf_ofs8;
unsigned long buf_addr16;
unsigned char buf_page16;
unsigned int buf_ofs16;

/* バッファ関連 */
void far *memarea = NULL;
int memareresize;
int buf_length;           /* バッファサイズ */
int block_length;         /* ブロックサイズ */

unsigned char (far *bufptr_8)[2][BLOCK_LENGTH] = NULL;      /* 1 バイト転送用バッファ */
int (far *bufptr_16)[2][BLOCK_LENGTH] = NULL;                /* 2 バイト転送用バッファ */

/* DMA 転送用パラメータ */
typedef enum {input, output} mode;
mode iomode;

long numsamples;
long datasize;
long curoffset;
int handle;
long datasize_init;

int handlerinstalled;

/* Sound Blaster 16 リソース・パラメータ */
int baseio;
unsigned char irq;
unsigned char dma;
unsigned char soundbit;

/* XMS用パラメータ */
typedef struct {
    unsigned long length;
```

```

    unsigned int sourcehandle;
    unsigned long sourceoffset;
    unsigned int desthandle;
    unsigned long destoffset;
} MOVEPARAMS;

MOVEPARAMS record_moveparams;
MOVEPARAMS save_moveparams;
MOVEPARAMS play_moveparams;

void far *xms_driver = NULL; /* XMS ドライバへのポインタ */

void interrupt (*oldintvector) (void) = NULL;

void far (*handler) (void) = NULL;

/* ===== 関数の宣言 ===== */

void WriteDSP(unsigned char value);
unsigned char ReadDSP(void);
int ResetDSP(void);
void InitDSP(unsigned long length, unsigned int samplingrate);
int InitSB16(mode io);
void ShutdownSB16(void);

void RestartSoundProc(void);
void PauseSoundProc(void);
void ContinueSoundProc(void);
void StopSoundProc(void);
void WaitSoundProc(void);

void SetHandler(void far *proc);
void InstallHandler(void);
void UninstallHandler(void);
void ExitProcSB16(void);
void far PlayHandler(void);
void far RecordHandler(void);
void CopyBlock(char blocknum);

void GetBuffer8(unsigned char far **bufptr, unsigned int length);
void GetBuffer16(int far **bufptr, unsigned int length);
void FreeBuffer8(unsigned char far **bufptr);
void FreeBuffer16(int far **bufptr);

void InitXMS(void);
unsigned int GetVersionXMS(void);
unsigned int GetFreeMemXMS(void);
int AllocateXMS(int far *handle, unsigned int size);
int ReallocateXMS(int handle, unsigned int newsize);
int FreeXMS(int far *handle);
int MoveXMS(MOVEPARAMS far *params);

int SaveRawData(char *filename);
int LoadRawData(unsigned char sndbit, char *filename);
int RecordData(unsigned char sndbit, float rectime, unsigned int samplefreq);
int PlayData(unsigned int samplefreq);

***** SB16I/O の設定 *****

```

資料

```
*****  
/* ===== SB16I/O 初期設定 ===== */  
  
int InitSB16(mode io) {  
  
/* Sound Blaster 16 用リソース・パラメータの設定 */  
    baseio = BASEIO;  
    irq = IRQ;  
  
/* Sound Blaster 16 DSP I/O ポートの設定 */  
    resetport = baseio + 0x006;  
    readport = baseio + 0x00A;  
    writeport = baseio + 0x00C;  
    poll18port = baseio + 0x00E;  
    poll16port = baseio + 0x00F;  
  
    switch (soundbit) {  
        case 8:  
            dma = DMA8;  
            pollport = poll18port;  
            break;  
        case 16:  
            dma = DMA16;  
            pollport = poll16port;  
            break;  
    }  
  
/* DSP のリセット */  
    if (!ResetDSP()) {  
        return(FALSE);  
    }  
  
/* 割り込みコントローラの設定 */  
    if (irq < 8) {  
        int_controller = 1;  
        pic_rotateport = 0x20;  
        pic_maskport = 0x21;  
        irq_intvector = 0x08 + irq;  
    }  
    else {  
        int_controller = 2;  
        pic_rotateport = 0xA0;  
        pic_maskport = 0x21;  
        irq_intvector = 0x70 + irq - 8;  
    }  
    irq_stopmask = 1 << (irq % 8);  
    irq_startmask = ~irq_stopmask;  
  
/* DMA コントローラの設定 */  
    if (dma < 4) {  
        dma_maskport = 0x0A;  
        dma_clptrport = 0x0C;  
        dma_modeport = 0x0B;  
        dma_baseaddrport = 0x00 + 2 * dma;  
        dma_countport = 0x01 + 2 * dma;  
    }  
    else {  
        dma_maskport = 0xD4;  
        dma_clptrport = 0xD8;
```

心理学実験における MS-DOS(3)

```
dma_modeport    = 0xD6;
dma_baseaddrport = 0xC0 + 4 * (dma % 4);
dma_countport   = 0xC2 + 4 * (dma % 4);
}

switch (dma) {
    case 0:
        dma_pageport = 0x87;
        break;
    case 1:
        dma_pageport = 0x83;
        break;
    case 2:
        dma_pageport = 0x81;
        break;
    case 3:
        dma_pageport = 0x82;
        break;

    case 5:
        dma_pageport = 0x8B;
        break;
    case 6:
        dma_pageport = 0x89;
        break;
    case 7:
        dma_pageport = 0x8A;
        break;
}

dma_stopmask  = dma % 4 + 0x04;          /* 000001xx */
dma_startmask = dma % 4 + 0x00;          /* 000000xx */

iomode = io;

switch (iomode) {
    case input:
        dma_mode = dma % 4 + 0x54;
        break; /* 010101xx */
    case output:
        dma_mode = dma % 4 + 0x58;
        break; /* 010110xx */
}

/* 割り込みハンドラ、プログラム終了処理の登録 */
InstallHandler();
atexit(ExitProcSB16);

return(TRUE);
}

/* ===== SB16I/O 終了処理 ===== */

void ShutdownSB16(void) {

    StopSoundProc();

    if (handlerinstalled) {
        UninstallHandler();
    }
}
```

資料

```

ResetDSP();

SetHandler(NULL);

/* XMS メモリの解放 */

    if (dma < 4) {
        FreeBuffer8(&(unsigned char far *)bufptr_8);
    }
    else {
        FreeBuffer16(&(int far *)bufptr_16);
    }
    FreeXMS(&handle);

}

/* ===== プログラム終了処理 ===== */

void ExitProcSB16(void) {

    if (irq < 8) {
        outp(0x20, 0x20);
    }
    else {
        outp(0xA0, 0x20);
        outp(0xA0, 0x0b);
        if (inp(0xA0) == 0) {
            outp(0x20, 0x20);
        }
    }
}

WriteDSP(pause_command);           /* DSP 処理中断 */

outp(dma_maskport, dma_stopmask); /* DMA チャンネルのマスク */

if (handlerinstalled) {
    UninstallHandler();          /* 割り込みハンドラの解放 */
}

ResetDSP();

}

/* ===== ハンドラのセット ===== */

void SetHandler(void far *proc) {

    handler = proc;

}

/* ===== 割り込み処理 ===== */

void interrupt inthandler(void) {

    intcount++;

}

```

心理学実験における MS-DOS(3)

```

if (handler != NULL) {
    (*handler)();
}

samplesremaining -= block_length;
curblock = !curblock;
if (samplesremaining < 0) {
    done = TRUE;
    WriteDSP(pause_command);
}

inp(pollport); /* 割り込み終了の合図 */

if (irq < 8) {
    outp(0x20, 0x20);
}
else {
    outp(0xA0, 0x20);
    outp(0xA0, 0x0b);
    if (inp(0xA0) == 0) {
        outp(0x20, 0x20);
    }
}
}

/* ===== 割り込みハンドラのセット ===== */
void InstallHandler(void) {

    disable(); /* 割り込み禁止 */
    outp(pic_maskport, (inp(pic_maskport) | irq_stopmask)); /* IRQ のマスク */

    oldintvector = getvect(irq_intvector); /* 割り込みベクタの保存 */
    setvect(irq_intvector, inthandler); /* 割り込みベクタのセット */

    outp(pic_maskport, (inp(pic_maskport) & irq_startmask)); /* IRQ のアンマスク */
    enable(); /* 割り込み開始 */

    handlerinstalled = TRUE;
}

/* ===== 割り込みハンドラの解放 ===== */
void UninstallHandler(void) {

    disable(); /* 割り込み禁止 */
    outp(pic_maskport, (inp(pic_maskport) | irq_stopmask)); /* IRQ のマスク */
    setvect(irq_intvector, oldintvector); /* 割り込みベクタの復帰 */
    enable(); /* 割り込み開始 */

    handlerinstalled = FALSE;
}

*****

```

資料

Sound Blaster 16 DSP の設定

```

***** DSP の設定 *****

/* ===== DSP の設定 ===== */

void InitDSP(unsigned long length, unsigned int samplingrate) {

    done = FALSE;
    samplesremaining = length;
    curblock = 0;

    samplesremaining -= block_length;

    if (dma < 4) {
        pause_command = 0xD0;
        continue_command = 0xD4;
        exit_command = 0xDA;
    }
    else {
        pause_command = 0xD5;
        continue_command = 0xD6;
        exit_command = 0xD9;
    }

/* DMA コントローラの設定 */
    outp(dma_maskport, dma_stopmask);
    outp(dma_clrptrport, 0x00);
    outp(dma_modeport, dma_mode);

    if (dma < 4) {
        outp(dma_baseaddrport, lo(buf_ofs8));           /* オフセットアドレスの指定 */
        outp(dma_baseaddrport, hi(buf_ofs8));           /* 低バイト→高バイト */
        outp(dma_countport, lo(buf_length - 1));       /* 転送するブロックのサイズ */
        outp(dma_countport, hi(buf_length - 1));       /* 低バイト→高バイト */
        outp(dma_pageport, buf_page8);
    }
    else {
        outp(dma_baseaddrport, lo(buf_ofs16));         /* オフセットアドレスの指定 */
        outp(dma_baseaddrport, hi(buf_ofs16));         /* 低バイト→高バイト */
        outp(dma_countport, lo(buf_length - 1));       /* 転送するブロックのサイズ */
        outp(dma_countport, hi(buf_length - 1));       /* 低バイト→高バイト */
        outp(dma_pageport, buf_page16);
    }
    outp(dma_maskport, dma_startmask);

    switch (iomode) {
        case input:                                /* 入出力の指定 */
            WriteDSP(0x42);
            break;
        case output:
            WriteDSP(0x41);
            break;
    }
    WriteDSP(hi(samplingrate));                  /* 標本化周波数の指定 */
    WriteDSP(lo(samplingrate));                 /* 高バイト→低バイト */

    if (dma < 4) {
        switch (iomode) {
            case output:                         /* DMA の設定 */
                WriteDSP(0xC6);
                break;                            /* 8-bit D/A, A/I, FIFO */
        }
    }
}

```

心理学実験における MS-DOS(3)

```
    case input:
        WriteDSP(0xCE);
        break;                      /* 8-bit A/D, A/I, FIFO */
    }

    WriteDSP(0x00);             /* 符号なし モノラル */
    WriteDSP(lo(block_length - 1)); /* 転送するブロック */
    WriteDSP(hi(block_length - 1)); /* 低バイト→高バイト */
}
else {
    switch (iomode) {
        case output:           /* DMA の設定 */
        WriteDSP(0xB6);
        break;                 /* 16-bit D/A, A/I, FIFO */
        case input:
        WriteDSP(0xBE);
        break;                 /* 16-bit A/D, A/I, FIFO */
    }

    WriteDSP(0x10);             /* 符号あり モノラル */
    WriteDSP(lo(block_length - 1)); /* 転送するブロック */
    WriteDSP(hi(block_length - 1)); /* 低バイト→高バイト */
}
}

/* ===== DSP の書き込み ===== */
void WriteDSP(unsigned char value) {

    while (inp(writeport) & 0x80);      /* ビット 7 がクリアされるまで待つ */
    outp(writeport, value);
}

/* ===== DSP の読み込み ===== */
unsigned char ReadDSP(void) {

    unsigned int value;

    while (!(inp(readport) & 0x80));   /* ビット 7 がセットされるまで待つ */
    value = inp(readport);

    return(value);
}

/* ===== DSP のリセット ===== */
int ResetDSP(void) {

    int i;

    outp(resetport, 1);
    outp(resetport, 0);
    i = 100;

    while ((ReadDSP() != 0xAA) && i--);
}
```

資料

```
    return(i);

}

/* ===== 音声処理開始 ===== */
void RestartSoundProc(void) {

    curoffset = 0;                                /* データ転送用パラメータのリセット */
    curblock = 0;
    samplesremaining = numsamples;
    datasize = datasize_init;
    done = FALSE;

    WriteDSP(continue_command);

}

/* ===== 音声処理一時中断 ===== */
void PauseSoundProc(void) {

    WriteDSP(pause_command);

}

/* ===== 音声処理再開 ===== */
void ContinueSoundProc(void) {

    WriteDSP(continue_command);

}

/* ===== 音声処理終了 ===== */
void StopSoundProc(void) {

    WriteDSP(exit_command);

}

/* ===== 音声処理終了待ち ===== */
void WaitSoundProc(void) {

    while (!done) {

    }

}

***** 音声データの取り込み *****
```

心理学実験における MS-DOS (3)

```
/* ===== 音声データの保存 ===== */
int SaveRawData(char *filename) {

    FILE *f;
    char chunk[SAVE_CHUNK_SIZE];

    if ((f = fopen(filename, "wb")) == NULL) {
        fprintf(stderr, "%s の書き込みに失敗しました。%n", *filename);
        return(EXIT_FAILURE);
    }

    /* XMS 転送用パラメータの設定: handle → chunk */
    save_moveparams.sourcehandle = handle;
    save_moveparams.sourceoffset = 0;
    save_moveparams.desthandle = 0;          /* コンベンショナル領域 */
    save_moveparams.destoffset = (long)(&chunk);

    if (dma < 4) {
        memset(&chunk, 0x80, SAVE_CHUNK_SIZE);
    }
    else {
        memset(&chunk, 0x00, SAVE_CHUNK_SIZE);
    }

    while (datasize) {
        save_moveparams.length =
            (datasize > SAVE_CHUNK_SIZE) ? SAVE_CHUNK_SIZE : datasize;
        MoveXMS(&save_moveparams);

        fwrite(&chunk, 1, (size_t)save_moveparams.length, f);
        save_moveparams.sourceoffset += save_moveparams.length;
        datasize -= save_moveparams.length;
    }

    fclose(f);
    return(EXIT_SUCCESS);
}

/* ===== 音声データの取り込み ===== */
int RecordData(unsigned char sndbit, float rectime, unsigned int samplefreq) {

    numsamples = rectime * samplefreq;
    soundbit = sndbit;

    switch (soundbit) {
        case 8:
            dma = DMA8;
            datasize_init = numsamples;
            datasize = datasize_init;

            GetBuffer8(&(unsigned char far*)bufptr_8, BLOCK_LENGTH);
            break;
        case 16:
            dma = DMA16;
            datasize_init = numsamples * 2;
            datasize = datasize_init;
    }
}
```

資料

```

        GetBuffer16(&(int far *)bufptr_16, BLOCK_LENGTH);
        break;
    }

    InitXMS();

    if (AllocateXMS(&handle, (unsigned int)((datasize / 1024) + 1))) {
        curoffset = 0;

        SetHandler(RecordHandler);
        InitSB16(input);
        InitDSP(numsamples, samplefreq);
    }
    else {
        fprintf(stderr, "メモリの確保に失敗しました。%n"
                "          %u バイトのメモリが必要です。%n", datasize);
        return(EXIT_FAILURE);
    }

    return(EXIT_SUCCESS);
}

/* ===== ハンドラ処理： 録音 ===== */
void far RecordHandler(void) {

    if (curoffset < datasize) {

        if (dma < 4) {
/* XMS 転送用パラメータの設定： bufptr_8 → handle */
            record_moveparams.length = ((curoffset + BLOCK_LENGTH) <= datasize)
                ? (BLOCK_LENGTH * 2) : (datasize - curoffset);
            record_moveparams.sourcehandle = 0; /* コンベンショナル領域 */
            record_moveparams.sourceoffset = (long)((*bufptr_8)[curblock]);
            record_moveparams.desthandle = handle;
            record_moveparams.destoffset = curoffset;

            MoveXMS(&record_moveparams);
            curoffset += BLOCK_LENGTH;
        }
        else {
/* XMS 転送用パラメータの設定： bufptr_16 → handle */
            record_moveparams.length = ((curoffset + BLOCK_LENGTH * 2) <= datasize)
                ? (BLOCK_LENGTH * 2) : (datasize - curoffset);
            record_moveparams.sourcehandle = 0; /* コンベンショナル領域 */
            record_moveparams.sourceoffset = (long)((*bufptr_16)[curblock]);
            record_moveparams.desthandle = handle;
            record_moveparams.destoffset = curoffset;

            MoveXMS(&record_moveparams);
            curoffset += BLOCK_LENGTH * 2;
        }
    }
}
*****
```

音声データの再生

```
*****
/* ===== 音声ファイルの読み込み ===== */

int LoadRawData(unsigned char sndbit, char *filename) {

    FILE *f;
    char chunk[LOAD_CHUNK_SIZE];

    soundbit = sndbit;

    if ((f = fopen(filename, "rb")) == NULL) {
        fprintf(stderr, "%s の読み込みに失敗しました。%n", *filename);
        return(EXIT_FAILURE);
    }

    fseek(f, 0, SEEK_END);           /* ファイルの終端まで移動し */
    datasize = ftell(f);           /* データのサイズを測定 */
    fseek(f, 0, SEEK_SET);           /* ファイルの先頭に移動 */

    if (dma < 4) {
        numsamples = datasize;       /* 8 ピット転送 */
    }
    else {
        numsamples = datasize / 2;   /* 16 ピット転送 */
    }

    InitXMS();

    if (!AllocateXMS(&handle, (unsigned int)((datasize / 1024) + 1))) {
        fprintf(stderr, "メモリの確保に失敗しました。%n"
                "          %u バイトのメモリが必要です。%n", datasize);
        return(EXIT_FAILURE);
    }
    else {
        /* XMS 転送用パラメータの設定: chunk → handle */
        play_moveparams.sourcehandle = 0; /* コンベンショナル領域 */
        play_moveparams.sourceoffset = (long)(&chunk);
        play_moveparams.desthandle = handle;
        play_moveparams.destoffset = 0;

        do {
            play_moveparams.length = fread(&chunk, 1, LOAD_CHUNK_SIZE, f);
            MoveXMS(&play_moveparams);

            play_moveparams.destoffset += play_moveparams.length;
            datasize -= play_moveparams.length;
        }
        while (datasize > 0);

        fclose(f);
    }

    return(EXIT_SUCCESS);
}

void CopyBlock(char blocknum) {
```

資料

```
int length;

if (dma < 4) {
    if ((curoffset + BLOCK_LENGTH) <= numsamples) {
        length = BLOCK_LENGTH;
    }
    else {
        length = (int)(numsamples - curoffset);
        memset((void *)(&(*bufptr_8)[blocknum][length / 2]), 0x80,
               BLOCK_LENGTH - (length / 2));
    }
}

/* XMS 転送用パラメータの設定: handle → bufptr_8 */
play_moveparams.length = length;
play_moveparams.sourcehandle = handle;
play_moveparams.sourceoffset = curoffset;
play_moveparams.desthandle = 0; /* コンベンショナル領域 */
play_moveparams.destoffset = (long)((*bufptr_8)[blocknum]);

MoveXMS(&play_moveparams);
curoffset += BLOCK_LENGTH;

}

else {
    if ((curoffset + BLOCK_LENGTH) <= numsamples) {
        length = BLOCK_LENGTH * 2;
    }
    else {
        length = (int)(numsamples - curoffset) * 2;
        memset((void *)(&(*bufptr_16)[blocknum][length / 2]), 0x00,
               (BLOCK_LENGTH - (length / 2)) * 2);
    }
}

/* XMS 転送用パラメータの設定: handle → bufptr_16 */
play_moveparams.length = length;
play_moveparams.sourcehandle = handle;
play_moveparams.sourceoffset = curoffset * 2;
play_moveparams.desthandle = 0; /* コンベンショナル領域 */
play_moveparams.destoffset = (long)((*bufptr_16)[blocknum]);

MoveXMS(&play_moveparams);
curoffset += BLOCK_LENGTH;
}

}

/* ===== 音声ファイルの再生 ===== */

int PlayData(unsigned int samplefreq) {

    switch (soundbit) {
        case 8:
            dma = DMA8;
            GetBuffer8(&(unsigned char far *)bufptr_8, BLOCK_LENGTH);
            curoffset = 0;

            memset((void *)bufptr_8, 0x80, sizeof(*bufptr_8));
            break;
        case 16:
            dma = DMA16;
            GetBuffer16(&(int far *)bufptr_16, BLOCK_LENGTH);
    }
}
```

心理学実験における MS-DOS(3)

```
curoffset = 0;

memset((void *)bufptr_16, 0xFF, sizeof(*bufptr_16));
break;
}
CopyBlock(0);
CopyBlock(1);

SetHandler(PlayHandler);
InitSB16(output);
InitDSP(numsamples, samplefreq);

return(EXIT_SUCCESS);
}

/* ===== ハンドラ処理： 再生 ===== */
void far PlayHandler(void) {

if (curoffset < numsamples) {
    CopyBlock(curblock);
} else {
    if (dma < 4) {
        memset((void *)((*bufptr_8)[curblock]), 0x80, BLOCK_LENGTH * 2);
    }
    else {
        memset((void *)((*bufptr_16)[curblock]), 0x00, BLOCK_LENGTH * 2);
    }
}
}

/*********************メモリ関連********************/
/* ===== リニアアドレス取得 ===== */
unsigned long getlinearaddr(void far *p) {

unsigned long addr;

addr = (unsigned long)FP_SEG(p) * 16 + (unsigned long)FP_OFF(p);
return(addr);
}

/* ===== バッファの確保 ===== */
void GetBuffer8(unsigned char far **bufptr, unsigned int length) {

memareasize = 8 * length;
memarea = malloc(memareasize);
if (memarea == NULL) {
    exit(EXIT_FAILURE);
}
}
```

資料

```

*bufptr = (unsigned char far *)memarea;

if (((getlinearaddr(memarea) % 65536) + length * 2) > 65536) {
    *bufptr += 2 * length;           /* */
}

buf_addr8 = getlinearaddr(*bufptr);
buf_page8 = buf_addr8 / 65536;
buf_ofs8 = buf_addr8 % 65536;

/* DMA 転送先アドレス */
buf_length = length * 2;
block_length = length;

}

void GetBuffer16(int far **bufptr, unsigned int length) {

    memareasize = 8 * length;
    memarea = malloc(memareasize);
    if (memarea == NULL) {
        exit(EXIT_FAILURE);
    }
    *bufptr = (int far *)memarea;

    if (((((getlinearaddr(memarea) >> 1) % 65536) + length * 2) > 65536) {
        *bufptr += 2 * length;           /* Pick second half to avoid crossing boundary */
    }

    buf_addr16 = getlinearaddr(*bufptr);
    buf_page16 = buf_addr16 / 65536;
    buf_ofs16 = (buf_addr16 >> 1) % 65536;

/* DMA 転送先アドレス */
    buf_length = length * 2;
    block_length = length;

}

/* ===== バッファの解放 ===== */
void FreeBuffer8(unsigned char far **bufptr) {

    *bufptr = NULL;
    free((void *)memarea);

}

/* ===== バッファの解放 ===== */
void FreeBuffer16(int far **bufptr) {

    *bufptr = NULL;
    free((void *)memarea);

}

*****

```

XMS 関連

```
*****  

#pragma option -w-  

/* ===== 初期化 ===== */  

void InitXMS(void) {  

    asm {  

        mov ax, 0x4310  

        int 0x2F  

        mov word ptr [xms_driver], bx  

        mov word ptr [xms_driver+2], es
    }  

}  

/* ===== バージョン取得 ===== */  

unsigned int GetVersionXMS(void) {  

    asm {  

        mov ah, 0x00  

        call [xms_driver]
    }  

}  

/* ===== メモリ確保 ===== */  

unsigned int GetFreeMemXMS(void) {  

    asm {  

        mov ah, 0x08  

        call [xms_driver]  

        mov ax, dx
    }  

}  

/* ===== メモリ割り当て ===== */  

int AllocateXMS(int far *handle, unsigned int size) {  

    asm {  

        mov ah, 0x09  

        mov dx, size  

        call [xms_driver]  

        les di, [handle]  

        mov es:[di], dx
    }  

}  

/* ===== メモリ再割り当て ===== */
```

```

int ReallocateXMS(int handle, unsigned int newsize) {
    asm {
        mov ah, 0x0F
        mov bx, newsize
        mov dx, handle
        call [xms_driver]
    }
}

/* ===== メモリ解放 ===== */
int FreeXMS(int far *handle) {
    asm {
        mov ah, 0x0A
        les di, [handle]
        mov dx, es:[di]
        call [xms_driver]
        mov word ptr es:[di], 0
    }
}

/* ===== データ転送 ===== */
int MoveXMS(MOVEPARAMS far *params) {
    asm {
        push ds
        mov ah, 0x0B
        lds si, [params]
        call [xms_driver]
        pop ds
    }
}

#pragma option -w

```

付録2 SBTEST.C

```

/*
SB16I/O のサンプル (SBTEST.C)
2003/09/26 by Takashi Shimokido.

音声データを記録し、再生します（×5回）

```

心理学実験における MS-DOS (3)

```
コンパイル：  
TCC SBTEST.C  
  
-----*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <conio.h>  
#include <string.h>  
#include <dos.h>  
#include <time.h>  
#include "sb16io.c"  
  
struct trial {  
    char filename[16];           /* 記録する音声ファイル名 */  
};  
  
struct trial data[32];  
  
int main(void);  
  
int main(void) {  
  
    int i;  
    char dumnum[3] = " ";  
  
    unsigned char sndbit = 16;      /* 量子化ビット */  
    float rectime = 2.5;           /* 録音時間 */  
    unsigned int samplefreq = 8000;  /* 標本化周波数 */  
  
    for (i = 0; i < 5; i++) {  
        strcpy(data[i].filename, "rec");  
        itoa(i, dumnum, 10);  
        strcat(data[i].filename, dumnum);  
        strcat(data[i].filename, ".raw");  
    }  
  
    RecordData(sndbit, rectime, samplefreq);  
    PauseSoundProc();  
  
    clrscr();  
    gotoxy(10, 10);  
    printf("何かキーを押してください。vn");  
    while (!kbhit());  
    clrscr();  
  
    for (i = 0; i < 5; i++) {  
        gotoxy(10, 10);  
        printf("%d 試行を %s に記録します。vn", i + 1, data[i].filename);  
        RestartSoundProc();  
        WaitSoundProc();  
  
        SaveRawData(data[i].filename);  
        clrscr();  
    }  
  
    ShutdownSB16();
```

資料

```
for (i = 0; i < 5; i++) {
    LoadRawData(sndbit, data[i].filename);
    PlayData(samplefreq);
    PauseSoundProc();

    gotoxy(10, 10);
    printf("%s を再生します。%n", data[i].filename);

    ContinueSoundProc();
    WaitSoundProc();

    clrscr();
    ShutdownSB16();
}

printf("終了します。%n");
return(EXIT_SUCCESS);
}
```