32ビット仮想コンソール

システム仕様

パート7:その他のコンソール・コンポーネント

資料作成日2023.01.08年Carra著

これは何だ?

このドキュメントは、Vircon32システム仕様のパート7です。この一連のドキュメントは、Vircon32システムを定義し、その機能と動作を詳細に記述した完全な仕様を提供します。

この仕様の主な目標は、Vircon32システムとは何か、および準拠していると見なされるためにゲームシステムがどのように実装される必要があるかについての標準を定義することです。また、Vircon32は仮想システムであるため、これらの文書の重要な第2の目標は、独自のVircon32実装を作成するための知識を誰にでも提供することです。

Vircon32について

Vircon32プロジェクトはCarraによって独自に作成されました。Vircon32システムとその関連資料(ドキュメント、ソフトウェア、ソースコード、アート、およびその他の関連要素を含む)は、元の作成者が所有しています。

Vircon32は無料のオープンソースプロジェクトで、誰でもゲーム機をプレイし、そのためのソフトウェアを開発できるようにすることを目的としている。この詳細については、使用可能な各ソフトウェアに含まれているライセンステキストを参照してください。

このドキュメントについて

このドキュメントは、Creative Commons Attribution 4.0 License(CC BY 4.0)に基づいて提供されています。ライセンスの全文は、Creative CommonsのWebサイトで読むことができます。

<https://creativecommons.org/licenses/by/4.0/>

まとめ

仕様のパート7では、これまで指定されていなかったコンソールの残りの内部コンポーネントを定義しています。このドキュメントでは、それぞれの動作、通信、内部変数、および操作の一般的なプロセスについて説明します。

パート6と同様に、残りのチップをこの1つの文書にグループ化するために、特定のチップの各セクションは、代わりにこの文書のサブセクションとして行われます。

[1 Timer 3](#_30j0zll)

[2 Random Number Generator (RNG) 9](#_1fob9te)

[3 Null device 12](#_3znysh7)

[4 RAM 13](#_2et92p0)

[5 BIOS 15](#_tyjcwt)

## 1タイマー

タイマーは、コンソールのグローバルタイミングを制御するチップです。新しいサイクルと新しいフレーム信号を生成し、コンソール内のすべてのコンポーネントに送信します。これに加えて、タイマーは、制御ポート要求を介して、現在の日時に関する情報をCPUに提供することもできます。

### 1.1外部接続

タイマーはコンソールを構成するチップの1つにすぎないため、独立して動作することはできません。この図は、他のコンポーネントとの通信をすべて示しています。

これらの各接続については、以下のセクションで個別に説明します。

#### 1.1.1新しいサイクルおよび新しいフレーム信号

タイマーは内部で新しいサイクル信号と新しいフレーム信号を生成し、これらの信号を他のコンソールエレメントが受信できるように出力します。これについては、このドキュメントのセクション1.6で説明されています。

#### 1.1.2コントロール信号

すべてのコンソールコンポーネントと同様に、タイマーはリセットのための信号を受信します。新しいフレームおよび新しいサイクル信号は、タイマー自体によって送信されている場合でも、他のコンソールコンポーネントと同様に、ここでも「受信」されたとみなすことができます。制御信号に対する応答は、このドキュメントのセクション1.7に詳述されています。

#### 1.1.3制御バス

タイマーは、デバイスID=0のスレーブ・デバイスとしてコントロール・バスに接続されます。これにより、バス・マスター(CPU)は、タイマーによって公開されるコントロール・ポートでの読取りまたは書込み操作を要求できます。ポートおよびそのプロパティのリストについては、後のセクションで説明します。

### 1.2現在の日付と時刻

タイマーには、現在の日付と時刻で更新され続ける内部クロックがあります。これをどのように管理するかを定義するのは実装次第です:ハードウェア実装は、コンソールがオフのときでも機能を維持する内部バッテリを備えたクロックを特徴とすることができます。代わりに、ソフトウェア実装は、コンソールがオンにされるたびに外部ソースから日付と時刻を更新することができます。

#### 1.2.1内部日付形式

タイマーは、内部的には現在の日付を32ビットワードとして表します。この内部形式では、次のように2つのフィールドがグループ化され、それぞれが16ビットの符号なし整数としてエンコードされます。

上位16ビットは、単に現在の年の数値(例:2022)です。下位16ビットは、その年に現在経過した日数を表します。したがって、たとえば3日間経過した場合、1月4日になります。番目です。

日付形式のこれら2つの整数フィールドの有効範囲は次のとおりです。

* **現在の年:**0～65535
* **ElapsedYearDays:**0から=1月1日セントから364=12月31日セント(\*)は

(\*):現在の年がうるう年と見なされる場合、ElapsedYearDaysの上限範囲は365年まで延長され、2月29日が含まれます。番目年の新しい日付として使用します。

#### 1.2.2内部時刻形式

特定の日付内では、タイマーはその日の現在の時刻も提供します。この時刻は、内部的には通常の32ビット整数値として表され、現在の日の経過秒数が格納されます。

1日は24 x 60 x 60=86400秒であるため、1日の経過秒数を表すこの形式の有効範囲は、0=00:00:00～86399=23:59:59になります。

### 1.3時刻の更新処理

時間が経過すると、タイマーは内部の現在の日付と時刻を更新し続ける必要があります。内部の時刻形式の解像度は1秒であるため、更新が必要になるのは1秒が経過するたびだけです。

これを決定するためにタイマーが持つ最も近い参照は、新しいフレーム信号です。コンソールフレームは正確に60 Hzで発生するため、日付と時刻を更新するために、タイマーは新しいフレーム信号を生成するたびにこのプロセスをトリガーします。新しいフレームを現在の日付と時刻に追加するために使用されるロジックは、次のアルゴリズムに示すように、タイマーの内部カウンタの範囲オーバーフローを伝播することです:

このロジックでは、現在の年がうるう年かどうかを判別する必要があることに注意してください。これにより、現在の年の日数を365日にする必要があるか、366日にする必要があるかが判別されます。

### 1.4内部変数

タイマーは、内部状態のさまざまな側面を保存する変数のセットを特徴としています。これらの変数はそれぞれ32ビット値として保存され、以前に表示された現在の日付形式を除き、仕様のパート2で説明されている同じデータ形式(整数、ブールなど)を使用してすべて解釈されます。ここでは、すべての内部変数をリストし、詳細を示します。

| 現在の日付 | **初期値:**(\*)は |
| --- | --- |
| **形式:**日付(Date) | **有効範囲:**1月1日セント、0～12月31日セント,65535#### |

この値は、1.2.1章で記述される日付フォーマットで、現在の日付を示します。

(\*)その初期値は、コンソール起動時の現在の日付によって決定されます。

| 現在の時刻 | **初期値:**(\*)は |
| --- | --- |
| **形式:**整数 | **有効範囲:**0から86399 |

この値は1.2.2章で記述されるように符号化された現在の時刻を示す。

(\*)その初期値は、コンソール起動時の現在の時刻によって決定されます。

| フレームカウンタ | **初期値:**0 |
| --- | --- |
| **形式:**整数 | **有効範囲:**負でない任意の整数値 |

この値は、最後にコンソールの電源を投入またはリセットしてから経過したフレーム数を示します。

| サイクルカウンタ | **初期値:**0 |
| --- | --- |
| **形式:**整数 | **有効範囲:**0から249999 |

この値は、現在のフレーム内で経過したCPUサイクル数を示します。

### 1.5制御ポート

このセクションでは、スレーブ・デバイスとしてのCPU制御バスへの接続を介して、タイマーによって公開される制御ポートのセットについて説明します。公開されるすべてのポートとその基本プロパティを次の表に示します:

| 公開されたコントロールポートのリスト | | | |
| --- | --- | --- | --- |
| 外部アドレス | 内部アドレス | ポート名 | R/Wアクセス |
| 000時間 | 00時間 | 現在の日付 | 読み取り専用 |
| 001時間 | 01時間 | カレントタイム(Current Time) | 読み取り専用 |
| 002時間 | 02時間 | フレームカウンタ | 読み取り専用 |
| 003時間 | 03時間 | サイクルカウンタ | 読み取り専用 |

#### 1.5.1ポートの読み取り/書き込み要求に対する動作

他のチップとは異なり、タイマーポートは実際には読み取り専用レジスタとしてモデル化できます。これらのポートを読み取ると、このセクションで説明するように、関連する値のみが取得されます。

実行されるアクションに加えて、制御バス通信の一部としてリクエストに対して成功/失敗レスポンスを提供する必要があることに注意してください。このレスポンスは常に、読取りリクエストでは成功、書込みリクエストでは失敗になります。この最後の場合、タイマーはそれ以上のアクションを実行せず、CPUはHWエラーをトリガーします。

現在の日付ポート

オン読み取る要求:

タイマーは、内部変数「Current date」の現在の値を提供します。

[Current Time]ポート

オン読み取る要求:

タイマーは、内部変数「Current time」の現在の値を提供します。

フレームカウンタポート

オン読み取る要求:

タイマは、内部変数「フレームカウンタ」の現在の値を提供する。

サイクルカウンタポート

オン読み取る要求:

タイマーは、内部変数「Cycle counter」の現在の値を提供します。

### 1.6タイミング信号の発生

グローバルコンソールのタイミングを制御するには、タイマーが時間を正確に測定できる必要があります。このために使用される特定のメカニズムは、実装によって定義されます。ハードウェア実装では発振器を使用できますが、ソフトウェアバージョンでは十分な解像度のシステムタイマーを使用できます。

コンソールがオンの間、タイマーは60 Hzの周波数で一定間隔で新しいフレーム信号を生成します。この周波数は、コンソールが長時間動作しているときに同期の問題を回避するために、できるだけ正確にする必要があります。

2つの連続した新しいフレーム信号の間に、タイマーは250,000の新しいサイクル信号を生成する必要があります。新しいフレーム信号とは異なり、新しいサイクル信号は一定の間隔で生成される必要はありません。それらは、それを含むフレーム内の任意のレートで、不規則であっても生成できます。実装は、与えられた各フレーム内の新しいサイクル信号の特定のタイミングを自由に決定できます。

### 1.7操縦信号への応答

コンソール内のすべてのコンポーネントと同様に、制御信号がトリガーされるたびに、タイマーはそれを受信し、そのイベントを処理するための応答を生成します。各制御信号に対して、タイマーは次のアクションを実行することによって応答します:

#### リセット信号:

* 内部変数「フレームカウンタ」、「サイクルカウンタ」は初期値に設定されます。
* 内部変数「現在の日付」と「現在の時刻」は、それぞれ現在の日付と時刻を表すように設定されています。必要に応じて、タイマーは実装の日付と時刻の情報にアクセスし、セクション1.2で説明されている内部形式に変換します。
* タイマーは、初期フレームを開始するための新しいフレーム信号を生成します。

#### フレーム信号:

* セクション1.3で説明されているように、タイマーは時間更新プロセスを起動します。
* 内部変数「Cycle counter」が0に設定されています。

#### サイクル信号:

* 内部変数「Cycle counter」が1だけインクリメントされる。

## 2乱数発生器(RNG)

RNGチップには、擬似乱数のシーケンスを生成する機能しかありません。これは、運やランダムな行動をシミュレートする必要があるゲームで役立ちます。この種の機能はソフトウェアで実装するのは非常に簡単ですが、それ専用のチップを持つことは、すべての実装が同意する必要がある標準の生成アルゴリズムをすべてのVircon32システムに提供するため、依然として価値があります。

### 2.1外部接続

RNGはコンソールを構成するチップの1つにすぎないため、単独で動作することはできません。この図は、他のコンポーネントとの通信をすべて示しています。

これらの各接続については、以下のセクションで個別に説明します。

#### 2.1.1コントロール信号

すべてのコンソールコンポーネントと同様に、RNGはリセット、新しいフレーム、新しいサイクルの信号を受信します。これらの信号に対する応答は、このドキュメントのセクション2.5に詳述されています。

#### 2.1.2制御バス

RNGは、デバイスID=1のスレーブ・デバイスとしてコントロール・バスに接続されます。これにより、バス・マスター(CPU)は、RNGによって公開されるコントロール・ポートに対する読み取りまたは書き込み操作を要求できます。ポートとそのプロパティのリストについては、後のセクションで詳しく説明します。

### 2.2内部変数

RNGは、その内部状態を格納する単一の変数を特徴とします。この変数は32ビット値として格納され、仕様のパート2で説明されているのと同じデータ形式(整数、ブールなど)を使用して解釈されます。ここでは、この内部変数のリストと詳細を示します。

| 現在の値 | **初期値:**1個 |
| --- | --- |
| **形式:**整数 | **有効範囲:**1～7 FFFFFFEh |

この値は、擬似ランダム生成アルゴリズムによって使用される現在のシードを表します。また、シーケンスの一部として提供される次の値でもあります。

### 2.3制御ポート

このセクションでは、スレーブ・デバイスとしてのCPUコントロール・バスへの接続を介してRNGによって公開されるコントロール・ポートのセットについて説明します。次の表に、公開される単一のポートとその基本プロパティを示します。

| 公開されたコントロールポートのリスト | | | |
| --- | --- | --- | --- |
| 外部アドレス | 内部アドレス | ポート名 | R/Wアクセス |
| 100時間 | 00時間 | 現在の値 | 読み取り/書き込み |

#### 2.3.1ポートの読み取り/書き込み要求に対する動作

RNGの制御ポートは、単なるハードウェア・レジスタではありません。このポートに対する読取り/書込み要求によってトリガーされる影響は、値の読取りまたは書込みとは異なる場合があります。この項では、これらの動作について説明します。

実行されるアクションに加えて、このポートに対する読み取りおよび書き込み要求に対して制御バスに提供される応答は、常に成功します。

[Current Value]ポート

オン読み取る要求:

RNGは、内部変数「現在の値」の現在の値を提供します。その後、RNGは、2.4項で説明したシーケンス生成プロセスをトリガして、シーケンス内の次の値で変数を更新します。

オン書く要求:

RNGは、受信した値が内部変数「Current value」の範囲外であるかどうかをチェックします。範囲外である場合、要求は無視されます。有効な値の場合、RNGは内部変数「Current value」を受信した値で上書きします。これにより、このポートに対する次の読み取り要求が作成され、新しい書き込み値が提供されます。

### 2.4シーケンス生成処理

擬似乱数を生成するアルゴリズムは、通常、初期シード値に基づいています。取得された値のシーケンスは、その特定の初期シード値から完全に決定されます。RNGで使用されるシードは、その内部変数「現在の値」です。生成ステップごとに、その変数はシーケンス内の次の値で上書きされます。その後、更新された値は次の生成ステップのシードとして機能します。

すべてのVircon32実装は、同等の生成アルゴリズムを使用する必要がある。これは、すべてのシステムが同じ初期シード値に対して同じシーケンスを生成する必要があることを意味する。RNGの場合、参照アルゴリズムは次のとおりである。

このアルゴリズムでは、Value64と呼ばれる一時的な64ビット整数変数が使用されることに注意してください。これは、32ビット値のみを使用する場合、乗算によって生成されるオーバーフロービットの処理方法がプラットフォームによって異なるためです。64ビット変数を使用して乗算を実行することで、一貫性のある結果を確保できます。

これらのアルゴリズムに使用される数値は、C++11のminstd\_rand実装と同じものが選択されます。

### 2.5操縦信号への応答

コンソール内のすべてのコンポーネントと同様に、制御信号がトリガーされるたびに、RNGはそれを受信し、そのイベントを処理するための応答を生成します。各制御信号に対して、RNGは次のアクションを実行することによって応答します:

#### リセット信号:

* 内部変数「Current value」が初期値に設定されます。
* コントロールポートの書き込み動作で説明されているように、内部変数の変更に関連付けられた追加の効果はすぐに適用されます。

#### フレームおよびサイクルシグナル:

* RNGは、特定の実装の詳細が必要としない限り、これらの信号に反応する必要はありません。

## 3 Nullデバイス

このデバイスは実際のコンソールチップではないため、これまでのところどの文書にも記載されていません。代わりに、存在しないコントロールポートへのアクセス要求が常に失敗して応答されることを保証するために、いくつかの実装で有用なオプションのダミーチップです。

仕様のパート2で説明したように、コントロールバスのグローバルアドレスは3ビット(デバイスID)+8ビット(そのデバイスのポートのローカルアドレス)で構成されています。これは、コントロールバスのアドレス空間が事実上8つの個別のデバイスIDに分割されることを意味します。ただし、コントロールバスには7つのデバイス(ID 0～6)しか接続されていないため、デバイスID=7は未使用のままであり、どのスレーブデバイスにも対応しません。

ヌル・デバイスを最後の既存のデバイスIDに接続することによって、実装がすべてのポート読み取り/書き込み要求を同じ方法で処理し、すべての要求を対応するデバイスに送信して処理することが可能になります。

### 3.1外部接続

ヌルデバイスの唯一の機能は、ID=7のスレーブデバイスとしてコントロールバスに接続され、受信した要求に失敗して応答することです。ダミーデバイスとして、実際のコンソール機能を実行するチップとは通信せず、制御信号を受信する必要もありません。

### 3.2制御ポート

ヌルデバイスは、実際にはポートを公開しません。代わりに、制御バスに障害応答を提供することによって、ポート範囲全体(700h～7 FFhの外部アドレス)内の要求を自動的に拒否します。

## 4 RAM

RAMチップメモリは、主にパッシブコンポーネントです。RAMチップに含まれるメモリの管理と、RAMチップに対する読み取り/書き込み要求の処理以外の機能は実行しません。したがって、RAMチップには内部変数や制御ポートはありません。

### 4.1外部接続

RAMはコンソールを構成するチップの1つにすぎないため、単独で動作することはできません。この図は、他のコンポーネントとの通信をすべて示しています。

これらの各接続については、以下のセクションで個別に説明します。

#### 4.1.1コントロール信号

すべてのコンソールコンポーネントと同様に、RAMチップはリセット、新しいフレーム、新しいサイクルの信号を受信します。これらの信号に対する応答は、このドキュメントのセクション4.3に詳述されています。

#### 4.1.2メモリバス

RAMチップは、デバイスID=0のスレーブデバイスとしてメモリバスに接続されています。これにより、バスマスタ(CPU)は、RAMチップによって公開されているメモリアドレスに対する読み取りまたは書き込み操作を要求できます。RAMメモリアドレスの範囲とプロパティについては、後のセクションで説明します。

### 4.2 RAMメモリ

RAMメモリは、32ビットワードの番号付きシーケンスです。サイズは4 MWords=4 x 1024 x 1024ワードです。RAMはランダムアクセスメモリの略で、これらの個々のワードは個別に読み取って変更できます。

RAMメモリは永続的ではありません。コンソールの電源が入っている間だけ内容を保存できます。含まれているデータは、コンソールの電源が切れると削除されます。

#### 4.2.1メモリ接続

RAMチップはメモリバスに接続され、メモリの内容をCPUに公開します。RAMはメモリバス内でデバイスID=0を使用するため、RAMメモリのアドレス範囲は次のようになります。

内部アドレス🡪0～3 FFFFFh(=4 x 1024 x 1024-1)

外部アドレス🡪00000000h～003 FFFFFh

#### 4.2.2メモリのパフォーマンス

特定の性能レベルは述べられないが、RAMメモリは、すべてのサイクルが、その同じサイクル内で現在の命令を完了するのに間に合うように、CPUのすべての要求に応答できるように、十分に高速であると仮定される。

### 4.3操縦信号への応答

コンソール内のすべてのコンポーネントと同様に、制御信号がトリガーされるたびに、RAMチップはそれを受信し、そのイベントを処理するための応答を生成します。制御信号のそれぞれに対して、RAMチップは次のアクションを実行することによって応答します:

#### リセット信号:

* すべてのRAMメモリアドレスは値0に設定されます。

#### フレームおよびサイクルシグナル:

* RAMチップは、特定の実装の詳細が必要としない限り、これらの信号に反応する必要はない。

## 5 BIOS画面

BIOSチップは、RAMと同様に、主にBIOS ROMへのアクセスを管理するパッシブコンポーネントです。そのため、BIOSチップには内部変数や制御ポートはありません。特定の実装の詳細で必要とされない限り、制御信号を受信する必要もありません。

BIOSはBasic Input/Output Systemの略で、起動やエラー処理などの低レベルの操作を管理するROMにインストールされたルーチンのセットを呼び出す一般的な方法です。Vircon32の場合、BIOS ROMはカートリッジROMと構造が似ていて、同じ3つの部分(プログラムROM、ビデオROM、オーディオROM)が含まれていますが、いくつかの追加の制限があります。

### 5.1外部接続

BIOSはコンソールを構成するチップの1つにすぎないため、単独で動作することはできません。この図は、他のコンポーネントとの通信をすべて示しています。

これらの各接続については、以下のセクションで個別に説明します。

#### 5.1.1メモリバス

BIOSチップは、デバイスID=1のスレーブデバイスとしてメモリバスに接続されています。これにより、バスマスタ(CPU)は、BIOSチップによって公開されているメモリアドレスに対して読み取りまたは書き込み操作を要求できます。BIOSメモリアドレスの範囲とプロパティについては、後のセクションで説明します。

#### 5.1.2 GPUおよびSPU

GPUとSPUは、それぞれBIOSのビデオROMとオーディオROMにアクセスする必要があります。GPUとSPUからの接続は、直接アクセスとして考えられています:各チップはこれらのコンテンツを自由に読み取ることができます。それでも、このアクセスを設定および管理するための実際のメカニズムは、実装によって定義されます。

### 5.2 BIOSメモリ

BIOSルーチンを実行するには、CPUはプログラムROMに接続して、BIOSに格納されているメモリワードを読み取る必要があります。プログラムROMは、32ビットワードのシーケンスを含む読み取り専用メモリ領域です。

BIOSチップはメモリバスに接続され、プログラムROMの内容をCPUに公開できるようになっています。BIOSチップはメモリバス内でデバイスID=1を使用するため、Nワードを含むBIOSプログラムROMの場合、BIOSメモリのアドレス範囲は次のようになります。

0からN-1までの内部アドレス🡪

🡪の外部アドレス10000000hから(10000000h+N-1)。

BIOSプログラムROMのサイズは、BIOSごとに異なります。1からBIOSプログラムROMのサイズ制限である1024 x 1024までの任意の数のワードを含めることができます。

#### 5.2.1メモリのパフォーマンス

具体的なパフォーマンスレベルは記載されていませんが、BIOS ROMの3つのコンポーネントは、次の機能を実行できるほど高速であると想定されています。

* サイクルごとに、プログラムROMメモリは、CPUが要求するすべてのワードを、その同じサイクル内で現在の命令を完了するのに間に合うようにCPUに提供することができる。
* すべてのフレームで、ビデオROMメモリは、同じフレーム内のすべての描画操作を完了するために必要なすべてのピクセルをGPUに提供できます。
* すべてのフレームにおいて、オーディオROMメモリは、SPUがその同じフレーム内ですべてのサウンド生成を完了するために必要なすべてのサンプルをSPUに提供することができる。

### 5.3 BIOSの要件

すべてのプログラム、ビデオ、およびオーディオROMに適用される一般的な要件に加えて、Vircon32 BIOSの内容は、いくつかのより制限的な要件の対象となります。これらは、各タイプのサブセクションにグループ化されて、次にリストされています:

#### 5.3.1 BIOSコンテンツの要件

* ビデオROMが存在し、テクスチャが1つだけ含まれている必要があります。
* オーディオROMが存在し、サウンドが1つだけ含まれている必要があります。
* オーディオROMのサイズ制限は1024 x 1024サンプルです。
* そのプログラムROMのサイズ制限は1024 x 1024ワードです。
* そのプログラムROMには、各コンソールの電源投入またはリセットの直後に呼び出されるスタートアップルーチンが含まれている必要があります。このルーチンのエントリポイントは、プログラムROMのアドレス4にある必要があります。
* そのプログラムROMには、CPUがハードウェアエラーをトリガーした直後に呼び出されるエラーハンドラルーチンが含まれている必要があります。このルーチンのエントリポイントは、プログラムROMのアドレス0にある必要があります。

#### 5.3.2 BIOS起動ルーチンの要件

* BIOSテクスチャの領域0～255にテキストフォントを定義する必要があります。このフォントの詳細については、セクション5.4を参照してください。
* BIOSテクスチャの領域256を、完全に不透明な白の単一ピクセルとして定義する必要があります。つまり、4つのRGBAコンポーネントは255になります。
* 画像と音声を含む何らかのイントロ画面の表示はオプションですが、お勧めします。これは、カートリッジを使用する必要なく、特定の実装における誤動作を廃棄するのに役立ちます。
* 最後のステップとして、カートリッジが存在するかどうかをチェックする必要があります。存在する場合は、プログラムROMの先頭に実行を転送します。
* カートリッジが存在しないことが検出された場合は、画面にその旨が表示されます。

#### 5.3.3 BIOSエラー・ハンドラーの要件

* 少なくとも、発生したハードウェアエラーのタイプを画面に表示する必要があります。
* エラーに関する拡張情報(たとえば、命令ポインタレジスタの値)の表示はオプションですが、推奨されます。
* すべての処理が完了したら、CPUを停止して実行を停止する必要があります。

### 5.4 BIOSテキストフォント

BIOSテキストフォントは256文字で構成されており、BIOSテクスチャの領域0～255で定義されています。これらの各領域は、幅10ピクセルx高さ20ピクセルで、ホットスポットが左上のピクセルに配置されている必要があります。BIOSは常にインストールされているため、すべてのプログラムは、テキストを書き込むためにBIOSテキストフォントが使用可能であると安全に想定できます。

BIOSフォントの文字マッピングは、主にWindowsコードページ1252(Latin-1としても知られています)とそのサブセットISO 8859-1に基づいています。次の表は、00hからFFhまでの各領域で定義する必要がある文字を示しています。各セルの青の4桁は、その位置にある文字のUnicodeコードポイントを示しています。

|  | 0 | 1個 | 2個 | 3個 | 4個 | 5個 | 6個 | 7個 | 8個 | 9個 | ア | イ | Cキー | Dキー | Eキー | Fキー |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 00 | しない  使用済み | しない  使用済み | しない  使用済み | しない  使用済み | しない  使用済み | しない  使用済み | しない  使用済み | しない  使用済み | しない  使用済み | (該当日本語なし)  0009 | (該当日本語なし)  000A弾 | しない  使用済み | しない  使用済み | (該当日本語なし)  000Dインチ | ⬚ 2B1A号 | ネットワーク 25 AF弾 |
| 10個 | (空) | ░ 2591 | ▒ 2592 | ▓ 2593 | オキシダーゼ 2588 | - 2500 | │ 2502 | 」 2518 | フランス 2510 | Γ 250度C | └ 2514 | ▲ 251度C | □ 2524 | リン 2534 | ┬ 252度C | できない 253度C |
| 20個 | (SPの場合)  0020 | あ! 0021 | 」 0022 | 番号 0023 | ドル 0024 | %未満 0025 | &(& 0026 | 「」 0027 | (の 0028 | )を 0029 | ※印 002A弾 | +キー 002バイト | 002度C | - 002Dインチ | です。 002(予) | /(該当日本語 002F(航空機) |
| 30人 | 0 0030 | 1個 0031 | 2個 0032 | 3個 0033 | 4個 0034 | 5個 0035 | 6個 0036 | 7個 0037 | 8個 0038 | 9個 0039 | (該当日本語なし 003A弾 | 003バイト | <<数値> 003度C | =<数値> 003Dインチ | > 003(予) | は? 003F(航空機) |
| 40人 | @様 0040 | ア 0041 | イ 0042 | Cキー 0043 | Dキー 0044 | Eキー 0045 | Fキー 0046 | Gキー 0047 | エイチ 0048 | アイ 0049 | ジェイ 004A弾 | Kキー 004バイト | Lキー 004度C | エム 004Dインチ | エヌ 004(予) | オ 004F(航空機) |
| 50人 | Pキー 0050 | Qキー 0051 | Rキー 0052 | エス 0053 | Tキー 0054 | ウ 0055 | Vキー 0056 | Wキー 0057 | X軸 0058 | ワイ 0059 | Zキー 005A弾 | [を 005バイト | \ 005度C | ]をクリックし 005Dインチ | ^キー 005(予) | \_する 005F(航空機) |
| 60人 | 「 0060 | ア 0061 | イ 0062 | ウ 0063 | dキー 0064 | eキー 0065 | フ 0066 | ト 0067 | hキー 0068 | アイ 0069 | jキー 006A弾 | キー 006バイト | l(エル) 006度C | エム 006Dインチ | n個 006(予) | オ 006F(航空機) |
| 70歳 | p値 0070 | qキー 0071 | アール 0072 | 秒 0073 | ト 0074 | ウ 0075 | vキー 0076 | wキー 0077 | () 0078 | y軸 0079 | Zキー 007A弾 | {を 007バイト | 007度C | }をクリックし 007Dインチ | ～い 007(予) | しない  使用済み |
| 80人 | ユーロ 20交流 | しない  使用済み | 「」 201A弾 | フ 0192 | 」 201(予) | ………… 2026 | † 2020 | ‡ 2021 | ƒ 02C6弾 | ‰ 2030 | ギガ 0160 | 「」 2039 | より 0152 | しない  使用済み | 〇 017Dインチ | しない  使用済み |
| 90人 | しない  使用済み | 「 2018 | ´ 2019 | 」 201度C | 」 201Dインチ | •サービス 2022 | - 2013 | - 2014 | ～い 02 DC(直流) | (TM) 2122 | シュ 0161 | 「」 203A弾 | ---- 0153 | しない  使用済み | ジュ 017(予) | Ÿ 0178 |
| A0(航空機) | しない  使用済み | 「 00A1(航空機) | セント 00A2弾 | ポンド 00A3弾 | スッ 00A4(航空機) | 円 00A5(航空機) | 同上 00A6弾 | § 00A7(航空機) | ィ 00A8インチ | ©著作権 00A9インチ | ェ 00年AA | 」 00航空機 | ¬ 00 AC弾 | ュ 西暦00年 | ョ 00 AE(航空機) | - 00 AF弾 |
| B0(該当日本語なし) | 度 00バイト0 | ア 00年B 1号 | ² 00バイト2 | 3分の1 00B3弾 | ´ 00B4弾 | ミュー 00B5弾 | ¶ 00B6弾 | ·<日付> B 7(航空機)00 | ソファ 00バイト8 | ¹ B 9(航空機)00 | º 00年BA | 」 00 BB弾 | 1/4 紀元前00年 | 1/2 00 BD弾 | 3/4 00年BE | ¿ 紀元前00年 |
| C0キー | 都市 00C0(航空機) | ア 第00条第1項 | ≫ 00C2弾 | テ 00C3弾 | ルッ 00C4弾 | Å 第00条C 5 | Æ 00C6弾 | 八つ 00C7(航空機) | 水 C 8(航空機)00 | ェ C 9(航空機)00 | ………… 00カリフォルニア州 | 00 CB弾 | 左 00 cc弾 | ブラウザ 00 CD弾 | び 00年CE | ジュークボックス 00 CFについて |
| D0(日) | ミ 00年D 0 | ム 00年D 1 | メ 00年D 2 | モ 00D3インチ | ヤ 00D4弾 | ユ 00D5インチ | ◯ 00D6インチ | × 00年D 7 | Ø(R 00D8インチ | であろう 00D9インチ | ミック 00 DA(航空機) | 乙 00 DB以上 | ウー 00 DC(直流) | キーボード 00 DD日 | ウィキメディアの曖昧 00 DE(航空機) | ß 00 DF弾 |
| E0キー | ア 00年E 0 | ア 00E1(航空機) | ～ 00E2(航空機) | アン 00年E 3 | äは 00E4(航空機) | Ø 00E5日本語版 | ミ 00E6日本語版 | チ 00E7日本語版 | è 00E8日本語版 | エ 00年E 9日 | エ 00年EA | エ 00年EB | μ 00欧州経済共同体 | 醇^ 00年ED | イッ 00年EE | イウ 00エフ |
| F0キー | 日 00年F 0 | エーン 00F1(航空機) | オー 00F2(航空機) | オゥ 00F3(航空機) | おう 00F4(航空機) | ◯ 00F5(航空機) | ヨ 00F6日本語版 | ÷(秒) 00F7(航空機) | ◯ 00F8(航空機) | ウ 00F9(航空機) | ウー 00フランス | ウ 00 FB弾 | ü様 FC 00台 | ý 00 FDについて | トファシチニブ 00フィート | □ 00 FF弾 |

この表で使用されている色は、次のように解釈されます。

|  | **白:**通常の印刷可能文字。それぞれのコードポイントとして表示されます。 |
| --- | --- |
|  | **赤:**未使用の文字。すべて空である必要があります。 |
|  | **黄:**空白文字。スペースは完全に空である必要があります。残りのスペースも空であるか、実装によって選択されたように表示されます。 |
|  | **緑:**これらの位置は、コードページ1252の印刷不可能なコントロールコードに対応していますが、BIOSフォントでは、テキストのみを使用して長方形のフレームとグリッドを描画するために設計された一連の文字に置き換えられています。 |

### 5.5標準BIOS

異なる作成者によって作成された任意の数のBIOS実装が存在する可能性があります。それらのそれぞれは、このドキュメントに記載されている要件に準拠している限り、有効なBIOSと見なすことができます。ただし、ROMファイルに「Vircon32標準BIOS」という名前が付けられている標準BIOSは1つだけです。他のBIOS実装は、実装とユーザーが標準BIOSと区別できるように、異なる名前を使用する必要があります。

すべてのVircon32実装は、デフォルトでカスタムBIOSを使用している場合でも、オプションとして標準BIOSを含めることをお勧めします。そうすることで、開発者がテストやデバッグのためにこれに頼る必要がある場合に備えて、すべての実装がハードウェアエラーの処理方法に合意する手段を持つことになる。

(パート7の終わり)