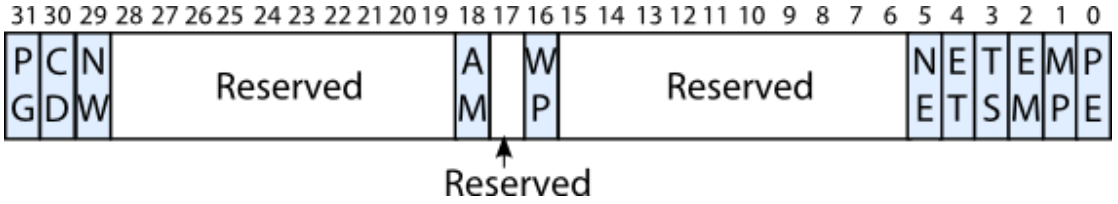
0から作るOS開発 カーネルローダその3 プロテクティッドモードへの移行とA20

前回までの内容

これまでで、

- 32ビットのプロテクティッドモードへ移行し、4GBのメモリ空間を利用する手順
  - プロテクティッドモード用のメモリアクセス設定を行う
  - プロテクティッドモードをONする
  - 4GBのメモリ空間を利用できるようにする
- プロテクティッドモードでは論理アドレス（セグメントセレクタ、オフセット）でメモリアクセスする
- プロテクティッド用のメモリアクセス設定はGDTを作り、LGDT命令でGDTをロードする
- GDTはセグメントディスクリプタで構成されおり、Nullディスクリプタ、コードディスクリプタ、データディスクリプタを作る
- セグメントセレクタでどのセグメントディスクリプタを

CR0レジスタはCPUの動作モードと状態を制御するレジスタです。CR0レジスタの制御フラグを見てみましょう



CR0レジスタ		
ビット	ビット名称	説明
0	PE	Protection Enableビット。このビットをONにするとプロテクティッドモードへ移行します
1	MP	Monitor Co-Processorビット。モニタ・コプロセッサビット WAIT命令を実行したときの動作を変更することができます 0：TSビットが0でも1でも関係なく無視する 1：TSビットが1であればコプロセッサ使用不可能例外を発生させる
2	EM	Emulationビット。エミュレーションビット 浮動小数点演算（x87 FPU）命令を実行したときの動作を変更することができます 0：x87 FPUを持っているので命令実行可能 1：命令実行時x87 FPUを持っていないのでコプロセッサ使用不可例外が発生する ソフトウェアによってエミュレーションを行う ※細かい条件はIntelのCPU仕様書をご確認ください
3	TS	Task Switchビット。タスクスイッチビット CPUはタスクスイッチする時にこのビットを1にします ※EM、MPビットの設定によっては影響があります
4	ET	Extended Typeビット。拡張タイプビット 0：80287以前のCPU 1：80387以後のCPU
5	NE	数値演算エラービット 0：x87 FPUエラーレポート無効 1：x87 FPUエラーレポート有効
16	WP	Write Protectビット。書き込み保護ビット 0：リング0のプログラムが読み取り専用のユーザ空間に書き込むことができる 1：リング0のプログラムが読み取り専用のユーザ空間に書き込むことを禁止
18	AM	Alignment Maskビット。アライメントマスクビット 0：自動アライメントチェック無効 1：自動アライメントチェック有効
29	NW	Not Write throughビット。ノットライトスルービット 0：CDビットが0の場合キャッシュをライトバックまたはライトスルーが有効 1:CDビットとNWビットの組み合わせはCPUのマニュアル10-17ページを参照ください
30	CD	Cache Disableビット。キャッシュ無効ビット 0：NWビットが0の場合キャッシュ操作が有効 1:CDビットとNWビットの組み合わせはCPUのマニュアル10-17ページを参照ください
31	PG	PaGingビット。ページングビット 0：ページング無効 1：ページング有効 ページングについては仮想メモリ管理にて後述します

CR0制御レジスタの0ビット目を1にするとプロテクティッドモードに移行します

CR0制御レジスタの注意点としてCR0レジスタの予約領域（図のReserved）は

CR0にもともと書き込んであった値を書き込み必要があります

ちょっと日本語が難しいですが、CR0の値を変更する際には

CR0の値を1回読みだしてから、書き込む必要があります

プロテクティッドモードに移行例をアセンブラ言語で見てみます

```
        , R                R   ジスタの  を読み  します
R       ,                     トを R    で   します
      R ,                  そのまま R に    込みます
```

MOV命令を使用してCR0を読みだしてPEビットを1にしてから書き込んでいます

CR0制御レジスタは32ビットのレジスタですので、32ビット汎用レジスタEAXを使用しています

以外に簡単でしたが、ここでやっかいな問題があります

CR0制御レジスタのPEビットをONした直後、前回ご紹介したGDTを利用したメモリアクセス方法に

変わってしまいます。単純にMOV命令でCS、DSレジスタのセグメントセレクタを変えようとしてもN.G.となります

ここがまた難しいのですが、順を追って見ていきます。このホームページ通りの手順でいくとCSレジスターの

セグメントセレクターは0x00で、DSレジスタセグメントセレクタは0x00でNullディスクリプタを選択している

状態となっています。ですので

```
S,                  ll  ィス     タを選  して  る状  なので . .
S,                  ll  ィス     タを選  して  る状  なので . .
```

は期待した通りの動作はしません。人間の普通の感覚では問題なく動くかと思いますが、

CPUは大変賢いので、PEビットをONにする頃には、先の命令を何行かすでに読み込んでいる状態となります

いわゆるパイプライン処理を行なっています。この例で言えばすでにMOV CSとMOV DSの命令は読み込みこまれて

いる状態で後は実行を待つのみとなっています。ところが、PEモードをONにしたためにいきなりプロテクティッドモードに

なってしまったので、0x08の値を読みに行く時にGDTを使って0x08が格納されているアドレスを読みに行ってしまいます

このプログラム例はリアルモードのままを想定して作ってしまっています。が、CPUはGDTを使います

GDTを使うのでCSを先に見に行くと0x00が書いてあるのでNullディスクリプタを使用してしまい、

サイズが0のセグメントにアクセスしてエラーとなってしまいます

プロテクティッドモードにする前に先にセグメントレジスタのセグメントセレクタの設定をしてあげる必要があります

しかし、プロテクティッドモードにする前にセグメントレジスタの値を変えてしますとリアルモードで上手く動きません

ここで卵がさきか鶏が先かの状態になってしまいますが、Intelの方もちょうどよい命令を作ってくださっていらっしゃいます

JMP命令を使います

```
              , R                  R  ジスタの  を読み  します
     R        ,                         トを R    で  します
            R ,                   そのまま R に    込みます
    J             S     de    r            S は          って  ます
```

JMP命令時”にコードセグメントに設定する値:ジャンプ先のアドレス”と命令すると、ジャンプと同時にCSレジスタの

値を書き換えてくれます。更に都合が良いことに、JMP命令はパイプライン処理として読み込んでいた数行先の命令を

全部クリアしてくれます。この例ではコードセレクタとして0x08を指定しています。0x08はコードセグメントディスクリプタです

次にJMP命令で飛ぶ32ビットのプログラムを見てみます

32ビットプログラム

32ビットのプログラム例を示します

```
//////////////////////////////////////////////
 S  r i g  r  e  ed   de
//////////////////////////////////////////////
 ITS
  de    r
              ,   T    S       T   S には         って  ます
            SS,                      タセ       トセ     タを  れて  ます
             S,
             S,
             S,
             S,

             S ,                      スタ      イ  タも       します
```

最初の

```
 ITS
```

でアセンブラが32ビットのコードを作ってくれます

```
,   T   S       T   S には       って ます
SS,                タセ     トセ   タを れて ます
 S,
 S,
 S,
 S,

 S ,                スタ     イ タも     します
```

CSはJMP命令で書き換えたので、他のセグメントセレクタを初期化します

CS以外はデータを扱うセレクタですので、コードセグメントセレクタとして0x10で初期化します

ESPも念のため初期化しています。アドレスはどこでもいいと思います（例のアドレス少し悪い例かもしれません）

これで32ビットのプロテクティッドモードに移行することができました

しかし、32ビットモードといってもアドレス0x000FFFFF以上にはアクセスすることができません

ハードウェア制御とA20

CPUはアドレスバスという32本（32ビットの場合）の配線でメモリとつながっています

CPUがメモリにアクセスするときに、アドレスバスのA0からA31までの32本の電圧をそれぞれHighにしたりLowにしたり

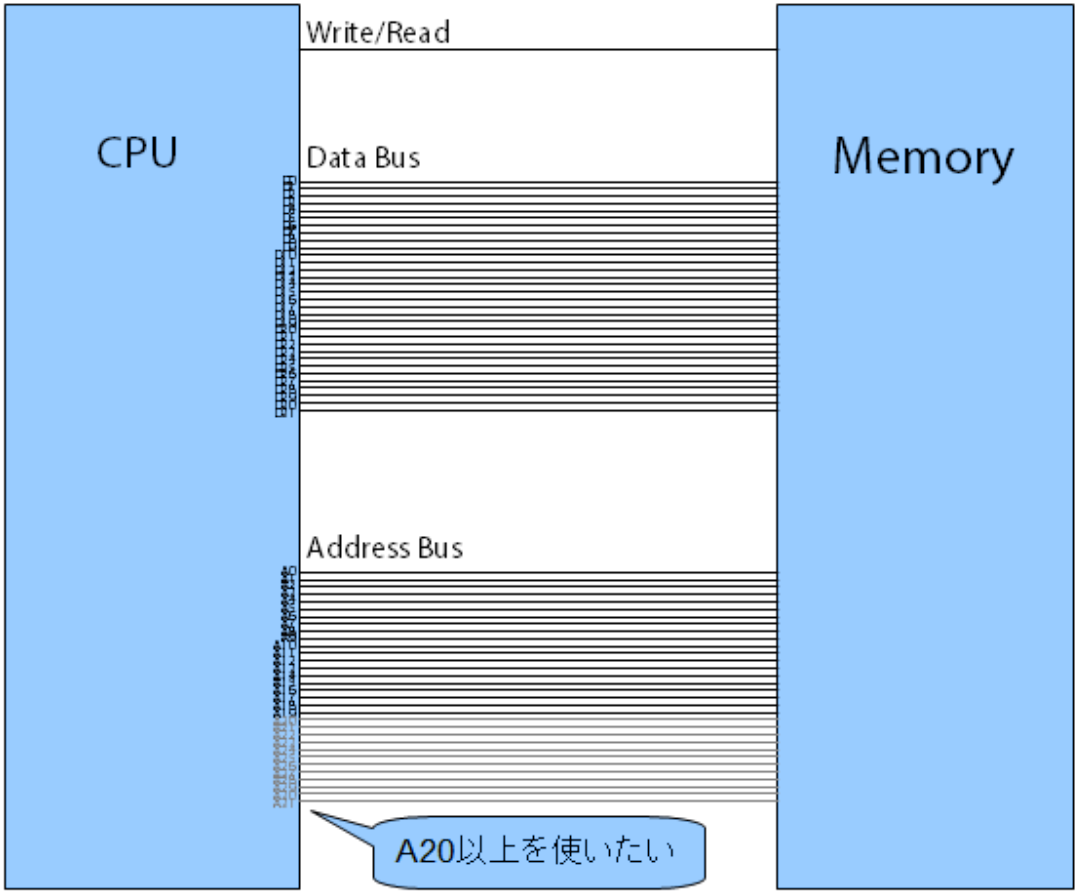
その組み合わせでアドレスを指定します。アドレスバスが32本ありますが、昔のCPUはA19までしか使用できません

ですので、昔のCPUがアクセスできるアドレスは0x000FFFFFまででしたが、新しくA20からA31を追加して

0xFFFFFFFFまでアクセスできるようにしたようです。現在のCPUでも昔のCPUとの互換性のため、リアルモードの間は

A0からA19までのアドレスバスしか使用できませんが、A20以上を使えるように限定を解除することができます

このため一般的にはA20を解除するといったような表現が使われていますので、このサイトもA20とタイトルをつけています



この図のようにA20以上のアドレスバスを有効にして、32ビッのアドレスを使えるようにするのが今回の目的です

（実際にはCPUとメモリの間にはメモリコントローラが存在します）

A20以上のアドレスバスを有効にするにはいくつか方法があります

- システムコントロールポートAを変更する

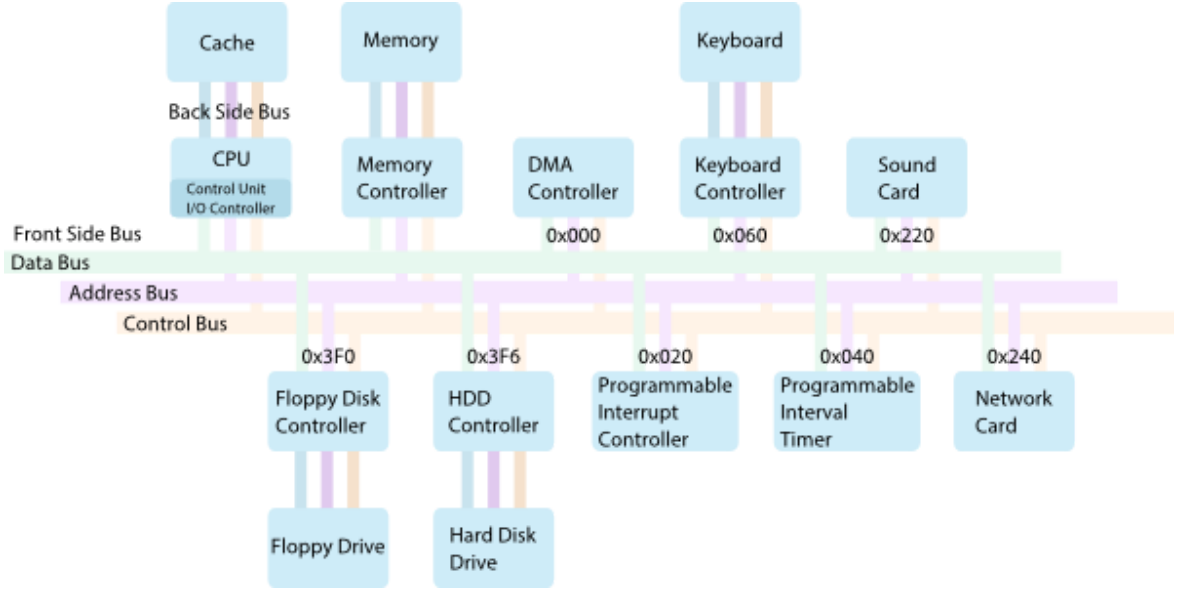
- BIOSの処理を利用して変更する
- キーボードコントローラの制御を変更する

ここでは3番目の”キーボードコントローラの制御を変更する”を紹介しようと思います

システムアーキテクチャ（フロントサイドバス）

キーボードコントローラを制御するために、CPU周りのシステムアーキテクチャ、特にシステムバスが

どうなっているのかを見ていきたいと思います



CPUは上図のように、フロントサイドバス（Front Side Bus）とバックサイドバス（Back Side Bus）に繋がっています

フロントサイドバスはメモリコントローラ（Memory Controller）、DMA（Direct Memory Access）コントローラ、

キーボードコントローラ（Keyboard Controller）、フロッピーディスクコントローラ（Floppy Disk Controller）などの

I/O（Input/Output：入出力）デバイスと接続されています。一方で、バックサイドバスにはキャッシュ（Cache）コントローラに

接続されています。フロントサイドバスには大きく分けると、

データバス（Data Bus）、アドレスバス（Address Bus）、制御バス（Control Bus）に分けることができます

データバス（Data Bus）

データバスは64ビットのバスとなりますが、現状は32ビットのプロテクティッドモードとなりますので、データバスは主に32ビットのデータを

CPUはやり取りします。もちろん、16ビットで動作するリアルモードでは16ビットのバスとしてCPUはやり取りします

アドレスバス（Address Bus）

アドレスを指定するバスとなります。”アドレス”と言いましてもこれまで出てきたアドレスとは異なります。少し複雑ですが、

今までのアドレスは”メモリ（Memory）内のアドレス”のことです。ここでのアドレスはバス上の”I/Oポートアドレス”となりますので、

注意してください。プログラムで指定したアドレスは”メモリ内のアドレス”のデータとして、CPUの制御ユニット（Control Unit）と

I/Oコントローラ（I/O Controller）はメモリコントローラ（Memory Controller）の”I/Oポートアドレス”を指定して

メモリコントローラに送信されます。データを受け取ったメモリコントローラはアドレスを解釈してメモリにアクセスします

このように今まで出てきたアドレスは特に”ポートアドレス”について意識せず使用することができました

しかし、メモリコントローラ以外のI/Oデバイスにアクセスするときは意識的に”I/Oポートアドレス”を指定してアクセスする必要があります

”I/Oポートアドレス”は各デバイス毎に決まっています。例えば、フロッピードライブ（Floppy Drive）を制御する

フロッピーディスクコントローラの”I/Oポートアドレス”は0x3F0です（0x3F0は代表的なアドレスとなります。実際は

フロッピーディスクコントローラにアクセスできる”I/Oポートアドレス”は複数あります）

ですので、フロッピーディスクを制御するときは”I/Oポートアドレス”0x3F0等にアクセス、

またキーボードコントローラ（Keyboard Controller）を制御するときは”I/Oポートアドレス”0x060等にアクセスする

必要があります

制御バス（Control Bus）

制御信号をとりあえずまとめて制御バスに分類しました。制御といってもいろいろありますが、例えば、I/Oデバイスの

コントローラに書き込みしたいときには、書き込み信号を出してコントローラに知らせます（命令実行時にハードウェアが自動で出してくれます）

また、ハードディスクからデータの転送が終わったことを知らせる割り込み信号もあります

この信号の制御はI/Oコントローラを介してCPUが制御したり、デバイスから信号を受け取ったりします

フロントサイドバスにデータバス、アドレスバス、制御バスがあり、そのバスに各I/Oデバイスがぶら下がっています

ではフロントサイドバスで制御できるI/Oバスを見ていきましょう

I/Oデバイス

フロントサイドバスに接続されているI/Oデバイスは下記となります

（マザーボード、接続機器によって異なります）

I/Oデバイスのポートアドレスマップ				
アドレス範囲	下位4バイト 0x0-0x3	中下位4バイト 0x4-0x7	中上位4バイト 0x8-0xB	上位4バイト 0xC-0xF
0x0000-0x000F	DMAコントローラ チャンネル0-3			
0x010-0x00F	-			
0x020-0x02F	PICマスタ	-		
0x030-0x03F	-			
0x040-0x04F	PIT	-		
0x050-0x05F	-			
0x060-0x06F	キーボード・マウス（0x60） スピーカ（0x61）	キーボード・マウス（0x64）	-	
0x070-0x07F	RTC（0x70：ビット0-6） NMI（0x70：ビット7） RTC（0x71）	-		
0x080-0x08F	DMAページレジスタ0-2（0x81-0x83）	DMAページレジスタ3（0x87）	DMAページレジスタ4-6（0x89-0x8B）	DMAページレジスタ7（0x8F）
0x090-0x09F	-			
0x0A0-0x0AF	PICスレーブ（0xA0-0xA1）	-		
0x0B0-0x0BF	-			
0x0C0-0x0CF	DMAコントローラ チャンネル4-7（0xC0-0xDF）、1-16バイト			
0x0D0-0x0DF	DMAコントローラ チャンネル4-7（0xC0-0xDF）、16-32バイト			
0x0E0-0x0EF	-			
0x0F0-0x0FF	フローティングポイントユニット（FPU/NPU/Math Coprocessor）			
0x100-0x10F	-			
0x110-0x11F	-			
0x120-0x12F	-			
0x130-0x13F	SCSIホストアダプタ（0x130-0x14F）、バイト1-16			
0x140-0x14F	SCSIホストアダプタ（0x130-0x14F） バイト17-32		SCSIホストアダプタ（0x140-0x15F） バイト1-16	

0x150- 0x15F	SCSIホストアダプタ(0x140-0x15F)、バイト17-32		
0x160- 0x16F	-		クワンタネリ(4番目の)IDEコントローラ マスタドライブ
0x170- 0x17F	セカンダリ(2番目の)IDEコントローラ マスタドライブ		-
0x180- 0x18F	-		
0x190- 0x19F	-		
0x1A0- 0x1AF	-		
0x1B0- 0x1BF	-		
0x1C0- 0x1CF	-		
0x1D0- 0x1DF	-		
0x1E0- 0x1EF	-		ターシェリ(3番目の)IDEコントローラ マスタドライブ
0x1F0- 0x1FF	プライマリ(1番目の)IDEコントローラ マスタドライブ		-
0x200- 0x20F	ジョイスティックポート		-
0x210- 0x21F	-		
0x220- 0x22F	サウンドカード		
	SCSIホストアダプタ(0x220-0x23F)、バイト1-16		
0x230- 0x23F	SCSIホストアダプタ(0x220-0x23F)、バイト17-32		
0x240- 0x24F	サウンドカード		
	ネットワークカード(NE2000以外)	-	
	NE2000ネットワークカード(0x240-0x25F)、バイト1-16		
0x250- 0x25F	NE2000ネットワークカード(0x240-0x25F)、バイト1-16		
0x260- 0x26F	サウンドカード		
	ネットワークカード(NE2000以外)	-	
	NE2000ネットワークカード(0x260-0x27F)、バイト1-16		
0x270- 0x27F	-	プラグ・アンド・プレイシステムデバイス	LPT2(パラレルポート2)
	-		LPT3(パラレルポート3)
	NE2000ネットワークカード(0x260-0x27F)、バイト17-32		
0x280- 0x28F	サウンドカード		
	ネットワークカード(NE2000以外)	-	
	NE2000ネットワークカード(0x280-0x29F)、バイト1-16		
0x290- 0x29F	NE2000ネットワークカード(0x280-0x29F)、バイト17-32		

0x2A0- 0x2AF	ネットワークカード（NE2000以外）	－	
	NE2000ネットワークカード（0x2A0-0x2BF）、バイト1-16		
0x2B0- 0x2BF	NE2000ネットワークカード（0x2A0-0x2BF）、バイト17-32		
0x2C0- 0x2CF	－		
0x2D0- 0x2DF	－		
0x2E0- 0x2EF	－	COM4（シリアルポート4）	
0x2F0- 0x2FF	－	COM2（シリアルポート2）	
0x300- 0x30F	サウンドカード/ MIDIポート（0x300-0x301）	－	
	ネットワークカード（NE2000以外）	－	
	NE2000ネットワークカード（0x300-0x31F）、バイト1-16		
0x310- 0x31F	NE2000ネットワークカード（0x300-0x31F）、バイト17-32		
0x320- 0x32F	サウンドカード/ MIDIポート（0x320-0x321）	－	
	NE2000ネットワークカード（0x320-0x33F）、バイト1-16		
	PC/XTハードディスクコントローラ	－	
0x330- 0x33F	サウンドカード/ MIDIポート（0x330-0x331）	－	
	NE2000ネットワークカード（0x320-0x33F）、バイト17-32		
	SCSIホストアダプタ（0x330-0x34F）、バイト1-16		
0x340- 0x34F	SCSIホストアダプタ（0x330-0x34F）、バイト17-32		
	SCSIホストアダプタ（0x340-0x35F）、バイト1-16		
	ネットワークカード（NE2000以外）	－	
	NE2000ネットワークカード（0x340-0x35F）、バイト1-16		
0x350- 0x35F	SCSIホストアダプタ（0x340-0x35F）、バイト17-32		
	NE2000ネットワークカード（0x340-0x35F）、バイト17-32		
0x360- 0x36F	テープアクセラレータカード （0x360）	－	クワンタネリ（4番目の）IDEコントローラ スレイブドライブ
	ネットワークカード（NE2000以外）		
	NE2000ネットワークカード（0x360-0x37F）、バイト1-16		
0x370- 0x37F	テープアクセラレータカード （0x370）	セカンダリ（2番目の）IDEコントローラ スレイブドライブ	LPT1（パラレルポート1） （カラーシステム）
	－	LPT2（パラレルポート2） （モノクロシステム）	
	NE2000ネットワークカード（0x360-0x37F）、バイト17-32		
0x380- 0x38F	－	サウンドカード （FMシンセサイザー）	－

<table>
<tr><td>0x390-
0x39F</td><td colspan="3">-</td></tr>
<tr><td>0x3A0-
0x3AF</td><td colspan="3">-</td></tr>
<tr><td>0x3B0-
0x3BF</td><td colspan="3">VGA/モノクロビデオ</td></tr>
<tr><td>0x3C0-
0x3CF</td><td colspan="3">VGA/EGAビデオ</td></tr>
<tr><td>0x3D0-
0x3DF</td><td colspan="3">VGA/CGAビデオ</td></tr>
<tr><td rowspan="2">0x3E0-
0x3EF</td><td>テープアクセラレータカード
(0x3E0)</td><td>-</td><td>COM3(シリアルポート3)</td></tr>
<tr><td colspan="2">-</td><td>ターシェリ(3番目の)IDEコントローラ
スレイブドライブ(0x3EE-
0x3EF)</td></tr>
<tr><td rowspan="2">0x3F0-
0x3FF</td><td colspan="2">フロッピードライブコントローラ</td><td>COM1(シリアルポート1)</td></tr>
<tr><td>テープアクセラレータカード
(0x3F0)</td><td>プライマリ(1番目の)IDEコントローラ
スレイブドライブ(0x3F6-
0x3F7)</td><td>-</td></tr>
<tr><td>0x0400-
0x0537</td><td colspan="3">-</td></tr>
<tr><td>0x0537-
0x0CEF</td><td colspan="3">-</td></tr>
<tr><td>0x0CF0-
0x0CFF</td><td colspan="2">-</td><td>PCI
アドレスレジスタ</td><td>PCI
データレジスタ</td></tr>
</table>

この表は大まかなポートアドレスで、例えばキーボードコントローラであれば、制御・ステータスレジスタ(0x064)、

データレジスタ(0x060)などと細かくアドレスが分かれています

次にこれらのI/Oデバイスにアクセスする方法を見ていきます

I/Oデバイスへのアクセス

I/Oデバイスへアクセスする特別な命令があります

デバイスからの読み込みはIN命令、デバイスへの書き込みにはOUT命令を使います

IN命令ではALレジスタにデバイスから読み込んだ値を、OUT命令ではALレジスタの値をデバイスに書き込みます

IN命令のソースコード例を示します

```
w i
    I      L,                    の  バイス  ら読み込み   Lに  納
    T ST   L,                    ト を  スト
    J        w i                 ト    でなければ
```

この例では0x64のアドレスを指定して、0x64のデバイス(キーボードコントローラ)から値を読み込みます

この命令を実行時にアドレスバスに0x64が出力され、制御バスには読み込み信号が出力されます

0x64と読み込み信号を受けたキーボードコントローラは自分のレジスタの値をデーババスに出力し

I/Oコントローラがデータバスから読み取り、ALレジスタにその値を格納します

```
        L,                          L ジスタに   を 納
UT        ,  L                      L ジスタの を  の バイスに   込み
```

この例では0x64のアドレスを指定して、0x64のデバイスにALレジスタの値を書き込みます

この命令実行時にアドレスバスに0x64が出力され、制御バスには書き込み信号が出力されます

同時に、データバスにALレジスタの値を出力されます。0x64と書き込み信号を受けたキーボードコントローラは

自分のレジスタにデータバスに出力されている値を格納します

8042キーボードコントローラ

キーボードコントローラはIntelの8042キーボード仕様でコントロールするICです

コントローラの仕様はサイトを検索するとチップを開発している会社などから発行されているものなどがでてきます

また、 IBMのテクニカルマニュアルを抜粋したもの もあり、全部のコマンドは載っていませんがわかりやすいかと思います

キーボードコントローラは2つのレジスタを持っていて、リード時とライト時で役割が変わります

キーボードコントローラのポートアドレス			
ポートアドレス	リード/ライト	名前	説明
0x060	リード	リードバッファレジスタ	キーボードコントローラからデータを読み込むレジスタです
0x060	ライト	ライトバッファレジスタ	キーボードコントローラにデータを書き込むレジスタです
0x064	リード	ステータスレジスタ	キーボードコントローラのステータスを読み込むレジスタです
0x064	ライト	制御コマンドレジスタ	キーボードコントローラに制御するコマンドを書き込むレジスタです

データレジスタはキーボードコントローラからのデータ読み込みとCPUからのデータ書き込みをするときに使用します

アドレス0x64はIN命令時とOUT命令時で意味合いが変わります

IN命令時はステータスレジスタとして動作し、コントローラのステータス情報を読取ることができます

読み取ったステータスレジスタの値の各ビットは次の意味を持っています

キーボードコントローラのステータスレジスタ		
ビット	名前	説明
ビット0	アウトプットバッファフル	0：キーボードコントローラのアウトプットバッファにデータが無いことを意味しています 1：アウトプットバッファにデータが存在していて、まだリードされていない状態です。リードバッファレジスタから値を読み込みとこのビットが0になります

ビット1	インプットバッファフル	0：キーボードコントローラのインプットバッファ（0x060または0x064）が空の状態 1：キーボードコントローラのインプットバッファにデータが書き込まれた状態であるが、まだコントローラが値を読み取っていない状態。コントローラが書き込まれた値を読み取るとこのビットは0になります
ビット2	システムフラグ	制御コマンドを書き込み、コマンドが成功すると、制御コマンドに応じて0または1にセットされます 例えばコントローラの中にはセルフテストが成功したときに1をセットします また、パワーONリセットしたときにはこのビットが0にセットされます
ビット3	コマンド/データフラグ	0：最後に書き込みしたのがデータ（0x060に書き込みをした） 1：最後に書き込みしたのがコマンド（0x064に書き込みをした）
ビット4	禁止（ロック）フラグ	キーボードコントローラのデータアウトプットバッファにデータが書き込まれるたびに、このビットが更新されます このフラグはキーボードのインヒビット（禁止）キーが押されている状態に1になります インヒビットキーが押されている状態ではキーボードコントローラはキーが押されても何も反応しません 0：インヒビットキーが押されている状態です 1：インヒビットキーが離されている状態です
ビット5	送信（書き込み）タイムアウトフラグ	0：キーボードコントローラからの送信（書き込み）が完了した状態 1：キーボードコントローラからの送信（書き込み）が完了していない状態。または送信エラーが起こった状態。ソフトウェアで一定時間タイムアウトを監視します
ビット6	受信（読み込み）タイムアウトフラグ	0：キーボードコントローラが受信（読み込み）を完了した状態 1：キーボードコントローラが受信（読み込み）を完了していない状態。または受信エラーが起こった状態。ソフトウェアで一定時間タイムアウトを監視します
ビット7	パリティエラー	0：キーボードコントローラから受信（読み込み）した最後のバイトは奇数パリティ 1：キーボードコントローラから受信（読み込み）した最後のバイトは偶数パリティ キーボードコントローラは通常奇数パリティで送信してきます

OUT命令時は制御コマンドレジスタとして動作し、制御コマンドを書き込みコントローラの動作を決定します

0x064の制御コマンドレジスタに次の制御コマンドを書き込みします

（各コマンドの詳細とその他のコマンドはキーボードドライバで説明します）

キーボードコントローラの制御コマンド		
コマンド	名前	説明
0x20	キーボードコントローラコマンド読み込み	キーボードコントローラから制御コマンドを読み込みます
0x60	キーボードコントローラコマンド書き込み	制御コマンドをキーボードコントローラに書き込みます。次に0x60にコマンドのデータを書き込みます
0xAA	セルフテスト	セルフテストを行います
0xAB	インターフェイステスト	インターフェイステストを行います
0xAC	診断情報出力	診断情報を出力します
0xAD	キーボード無効	キーボードを無効にします
0xAE	キーボード有効	キーボードを有効にします
0xC0	リードインプットポート	コントローラはインプットポートを読み取った値をアウトプットバッファに書き込みます
0xD0	リードアウトプットポート	コントローラはアウトプットポートを読み取った値をアウトプットバッファに書き込みます
0xD1	ライトアウトプットポート	コントローラは次に0x60に受信したデータをアウトプットポートに書き込みます
0xDD	A20アドレスライン有効	A20以上のアドレスラインを有効にします ※標準コマンドではないので対応していないコントローラがあります
0xDF	A20アドレスライン無効	A20以上のアドレスラインを無効にします ※標準コマンドではないので対応していないコントローラがあります

0xE0	リードテストインプット	TEST0、TEST1ピンの状態を読み取り、アウトプットバッファにデータを書き込みます
0xFE	システムリセット	キーボードをリセットします

A20を有効にする場合はこの制御コマンドレジスタにコマンドをOUT命令で書き込み、

ステータスレジスタをIN命令で読み込み制御していきます

8042キーボードコントローラでA20を有効にする

キーボードコントローラでA20以上のアドレスラインを有効にするには、制御コマンド表にある0xDDを

使って書き込みをすればいいのではと思いますが、このコマンドはコントローラを作っている会社独自のコマンドで

対応しているコントローラはあまりありません。ですので、違う方策を使っていきます

より一般的な方法は標準コマンドを使用する方法ですが、難解です

キーボードコントローラにはアウトプットポートを持っていて、各ポートの出力を変更することができます

出力を変更するコマンドは0xD1ライトアウトプットポートコマンドで出力を制御することができます

各ポートの出力を変えることが下記の変更が可能となります

キーボードコントローラのアウトプットポート		
ビット	名前	説明
ビット0	システムリセット	0：システムをリセットします 1：システムは通常動作状態
ビット1	A20	0：A20以上のアドレスラインを無効 1：A20以上のアドレスラインを有効
ビット2-3	未使用	-
ビット4	インプットバッファフル	0：キーボードコントローラのインプットバッファ（0x060または0x064）が空の状態 1：キーボードコントローラのインプットバッファにデータが書き込まれた状態であるが、まだコントローラが値を読み取っていない状態。コントローラが書き込まれた値を読み取るとこのビットは0になります
ビット5	アウトプットバッファエンプティ	0：アウトプットバッファにデータが存在していて、まだリードされていない状態です。リードバッファレジスタから値を読み込みとこのビットが0になります 1：キーボードコントローラのアウトプットバッファにデータが無いことを意味しています
ビット6	キーボードクロックライン	0：キーボードのクロックラインはハイインピーダンス 1：キーボードのクロックラインをLow固定
ビット7	キーボードデータライン	0：キーボードのデータラインはハイインピーダンス 1：キーボードのデータラインはLow固定

今回やりたいことはA20以上のアドレスラインを有効にしたいだけですので、他のポート出力は変更したくありません

ですので、一度アウトプットポートの出力内容を制御コマンド0xD0リードアウトプットポートで読みだしてから

ビット1のA20を1：有効にして0xD1ライトアウトプットポートで書き込みA20のみポートの出力を変更します

ライトアウトプットポート

アウトプットポートへの書き込み例です

```
        L,                          イトアウト    ト    トコ    ドを L  ジスタに  納
UT          ,  L                    アド  ス    に Lの    タを送

    L,                              を    にする    タ
UT          ,  L                    にアウト    ト    トに    した    を    込む
```

この例では0xD1ライトアウトプットポートコマンドを0x64制御コマンドレジスタに書き込みし、

アウトプットポートに書き込みたい値を0x60ライトバッファレジスタに書き込みしています

（この例の手続きは正しくありません）

リードアウトプットポート

アウトプットポートの読み込み例です

```
        L,                          ドアウト    ト    トコ    ドを L  ジスタに  納
UT          ,  L                    アド  ス    に Lの    タを送

I       L,                          ドバ    ァ  ジスタにアウト    ト    トの
                                    込まれるので  読み込む
```

この例では0xD0リードアウトプットコマンドを0x64制御コマンドレジスタに書き込みした後、

キーボードコントローラは0x60にアウトプットポートを読み込んだ値を0x60リードバッファに書き込みます

それをIN命令で0x60リードバッファレジスタから読み込んでALレジスタの格納しています

A20を有効にするまとめ

A20を有効にする手続きは

- 0xADキーボード無効コマンドでキーボードを一旦無効化
- 0xD0リードアウトプットポートコマンドでアウトプットポートを読み込む
- 読み込んだ値のビット1を1にしてA20を有効にするアウトプットポート出力データを作る
- 0xD1ライトアウトプットポートコマンドでA20を有効にする出力データを書き込む
- 0xAEキーボード有効コマンドでキーボードを有効にする

となります。コマンドを書き込んだ後はコントローラがコマンドの処理を完了するまで時間がかかりますので、

ウェイトをする必要があります。

それではA20を有効にするアセンブラソースを見てみます

```
/////////////////////////////////////////////////

    le

/////////////////////////////////////////////////
 ITS
   le
       LI                              り込み禁

        LL      w i                 ウ  イト  理
              L,                          ド無
       UT         ,  L

        LL      w i                 ウ  イト  理
              L,                       ドアウト    ト    トコ    ド送
       UT         ,  L

        LL      w i                 アウト    ト    トの       込まれるまでウ  イト
      I       L,                        ドバ    ァ  ジスタ読み込み
       USH                          読み込んだ    タをスタ    に

        LL      w i                 ウ  イト  理
              L,                     イトアウト    ト    トコ    ド送
       UT         ,  L

        LL      w i                 ウ  イト  理
                                           読み込んだアウト    ト    トの  を  に  す
       R      L,                          トを にする
       UT         ,  L                   にする       タを送

        LL      w i                 ウ  イト  理
              L,                          ド
       UT         ,  L

        LL      w i                 ウ  イト  理
      STI                             り込み
      R T

  w i                               ウ  イト  理関
       I      L,                   ス    タス  ジスタ読み込み
       T ST   L,                   送  したコ    ド  理     した  を
                                   シス          で
       J        w i                   して  な     は
       R T

  w i                               アウト    ト    トの       込まれるまでウ  イト  理関
       I      L,                   ス    タス  ジスタ読み込み
       T ST   L,                         込まれた  イ     トバ    ァ    を
       J        w i                   込まれて  な     は
       R T
```

注意点として、この処理はリアルモードで処理されることを想定して作成しています

以上でA20以上のアドレスラインを有効にし、プロテクティッドモードに移行することができました

次はカーネル起動準備について説明します

カーネルローダその2 プロテクティッドモードとGDT

カーネルローダその4 カーネルをロードする

mail@softwaretechnique.jp