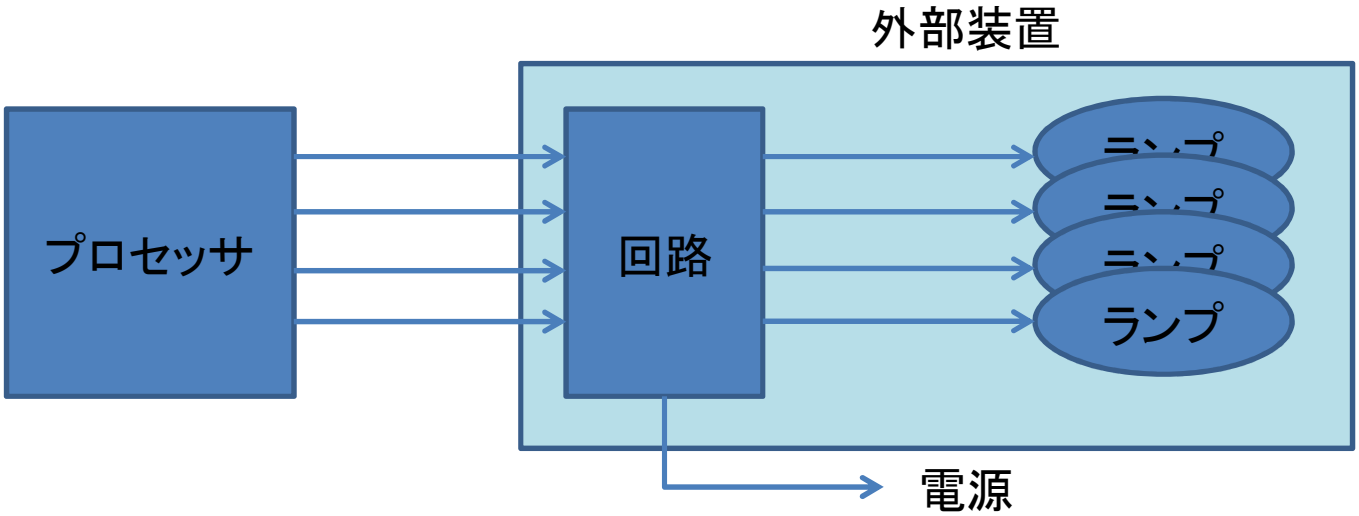
入出力の概念と用語

入力と出力の装置

- 入出力装置(I/O) :
  コンピュータと外部装置を接続する方法



データ転送

外部装置の1次的な機能：データ転送

プロセッサと外部装置間のデータ交換メカニズム

制御機能：2次的な機能

データはどのように通信されるのか

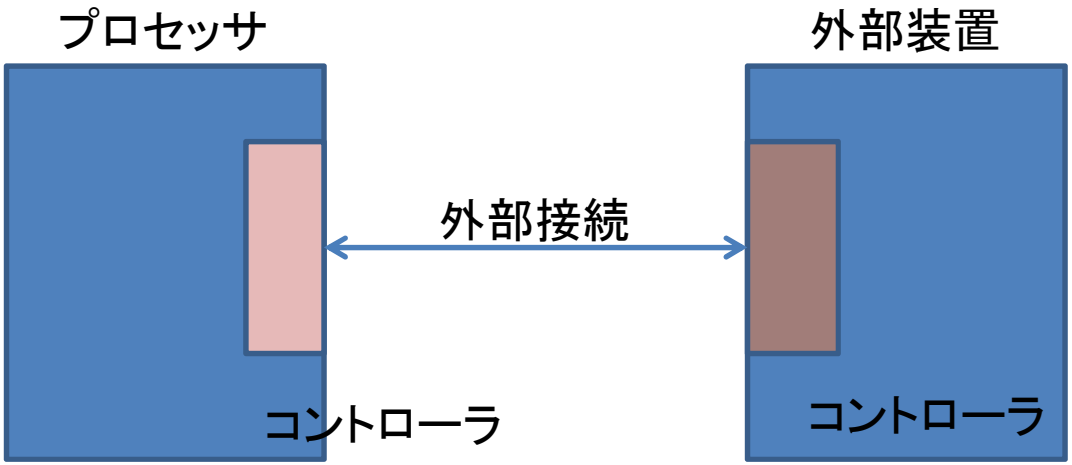
転送はどのように制御されるのか

高速な転送にはどのような技術が必要か

インターフェースコントローラ

外部装置との接続時使われる電圧や符号化方法の相互変換

外部装置で利用される電圧や，データの符号化
装置のタイプ，データの転送される速度，採用される接続方法，CPUからの距離に依存



シリアルとパラレルデータ転送

- 並列型インターフェイス
  - 多くの接続線により複数ビットを1度に転送
  - 転送速度：高速，ハードウエア量：大
  - インターフェイス幅
- 直列型インターフェイス
  - 1度に1ビットを転送
  - 転送速度：低速，ハードウエア量：小

自己同期型データ

プロセッサと入出力装置：独立したクロックで動作

送り手がどのように符号化したかを，受け手が正確に決定できる情報を，インターフェイスを介して送られる信号に内包されている場合：ハンドシェイク

全二重と半二重通信

- 全二重通信
  双方向の通信を同時に行えるインターフェイス
  2つの並列装置から構成
- 半二重通信
  同時には一方向の通信しか行えないインターフェイス
  双方向通信のためには，接続線の供用が必要

インターフェイスの遅延と流量

遅延(レイテンシー):あるビットが送られて，それが受け取られるまでの遅延時間

流量:単位時間当たりの転送ビット数

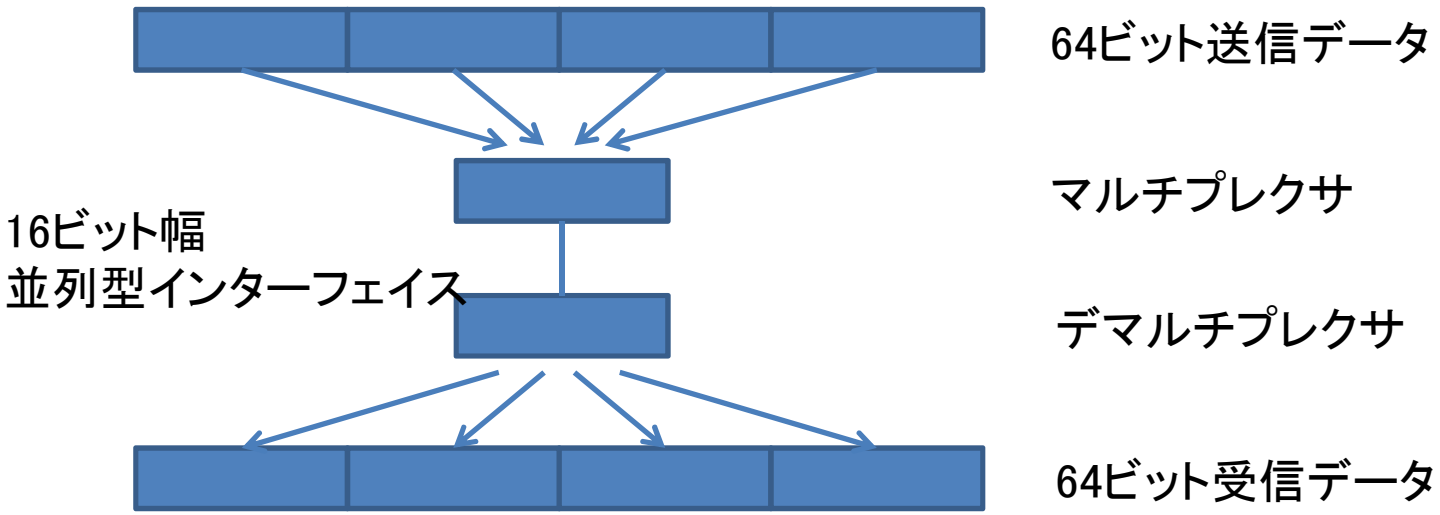
Mbps:Mbit/s, MBps:Mbyte/s

多重化

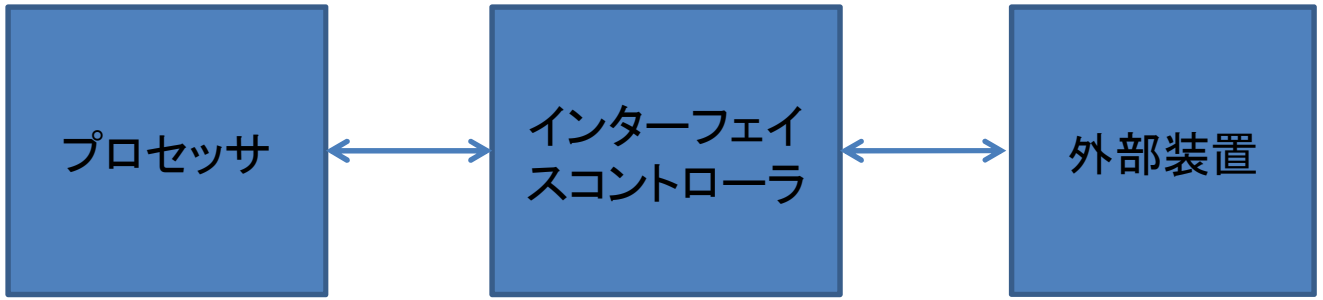
流量：並列型インターフェイス ＞ 直列型
並列性の向上 ⇒ データ転送の向上

限られたハードウエア資源（ICチップのピン数など）

多重化による性能向上：マルチプレクサ，デマルチプレクサ



- １つの外部インタフェースに複数の装置
- 入出力のプロセッサからの見え方



プログラミングインターフェイスの提供
　その下にある装置の動作を正確に知る必要はない

バスとバスアーキテクチャ

バスの定義

2つ以上の機能ユニットが制御信号やデータを送ることを可能にする，ディジタル的な通信機構

メモリバス：

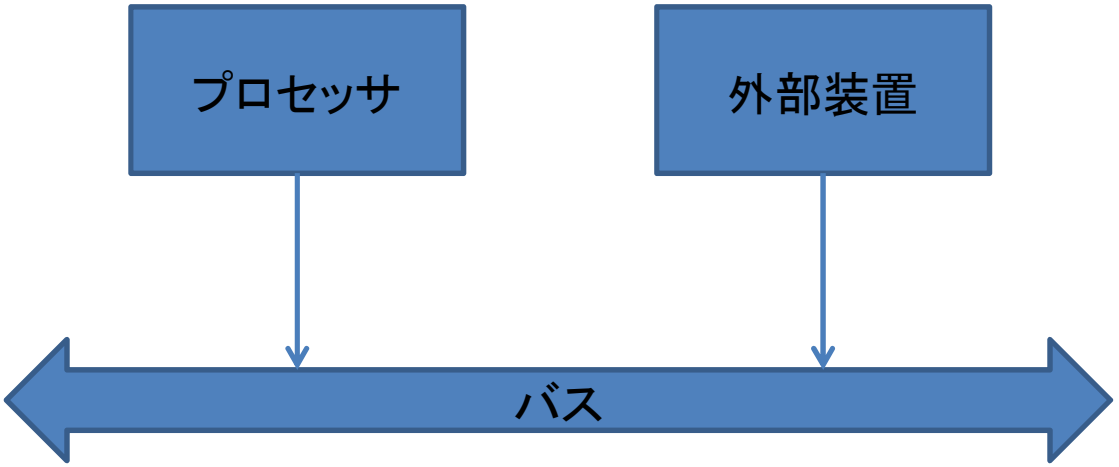
　プロセッサとメモリシステム間の接続を意図

IOバス：

　プロセッサと入出力装置間の接続を意図

プロセッサ，入出力，バス

- 機能ユニット感を接続するのに利用される，コンピュータシステム内の接続機構
- 1つのコンピュータは1つ以上のバスを含む
- プロセッサやメモリ，外部入出力装置との間を接続



専有的バスと標準バス

専有的バス

標準バス

共有バスとアクセスプロトコル

大半のバスは共有型

⇒ 1つのバスがプロセッサと一群の入出力装置との接続に利用される

アクセスプロトコル:

接続された装置がどのようにしてバスが利用可能か，他の装置により使用中かを判断

どのようにしてバスの使用権を得るか

- 複数のバス

  コンピュータシステム内には複数のバスが装備

  プロセッサチップ内にも1つ以上のバスを含む

- 受動的で並列的な機構

  もっとも単純なバス：電子部品を含まない ⇒受動型

  バスに接続される装置は，バスを介して通信するための
  ハードウエアを実装

- 物理的接続

  1つのシリコンチップ上に焼き付けられた小さな接続線

  複数の接続線からなるケーブル

  回路基盤上の平行線一式 ⇒ マザーボード

- バスインターフェース

  マザーボードと外部装置：ソケット部品を利用

- アドレス(A)ライン，コントロール(C)ライン，データ(D)ライン

  バスを構成する接続線

フェッチ-ストアパラダイム

メモリシステム：メモリからの値のリード，値のライト

バス　：入出力装置との間のデータ転送

　　：フェッチ-ストアパラダイムを提供

バスを介してフェッチあるいはストアを行う手順

フェッチ	ストア
Cラインを使ってバスへのアクセス権の獲得	Cラインを使ってバスへのアクセス権の獲得
アドレスをAラインに	アドレスをAラインに
Cラインによりフェッチ操作の要求	値をDラインに
Cラインにより操作の完了待機	Cラインにストア操作を提示
データをDラインから読み込み	Cラインにより操作の完了待機
他の装置によるバスのアクセスを許可	他の装置によるバスのアクセスを許可

- バスの幅

- 多重化
  データの多重化とアドレスの多重化
  アドレスとデータの多重化

  利点：ハードウエア資源の削減
  欠点：処理時間の増大
  　　洗練されたバスプロトコルが必要
  　　より複雑なバスインターフェイスが必要

- バス幅とデータのサイズ
  - バス幅に合致するデータの転送：1サイクル
  - バス幅より大きいと：複数のサイクル
- バスのサイズ，汎用レジスタのサイズ，ALUや機能ユニットが単一のサイズを利用することによる利点
- アドレスのサイズを他の項目と同じにすることによる利点

バスアドレス空間

メモリバス

プロセッサと1つ以上のメモリとの接続

バスインターフェイスはバスプロトコルを実装

プロセッサからはプログラミングインターフェイスを提供

バスはアドレス空間を提供

プロセッサは外部装置を直接見ることができない

個々のメモリは特定のアドレスの集合に対応するよう配置

プロセッサのフェッチ要求

⇒すべてのメモリは要求を受け入れ

アドレスが一致したメモリのみ応答

潜在的なエラー

メモリインターフェイスが実装している概念的な手順

```
Rをメモリが割り当てられたアドレス範囲とする
永久に{
  要求が発生するまでバスを監視;
  if(要求がRの中のアドレスを指定){
    要求に応答;
  } else {
    要求を無視;
  }
}
```

潜在的なエラー

- アドレス衝突

  複数のインターフェイスが間違えて配置され，同じアドレスに対して応答する

  ほとんどのバスプロトコルは，アドレス衝突の検出機構を含む

- 未定義アドレス

  インターフェイスが配置されていないアドレスに対してプロセッサがアクセス

  ほとんどのバスプロトコルでは，時間切れの仕組みを採用

アドレス配置とソケット

システム構成例１

メモリシステムはマザーボードに実装する小さな回路基板として実現

すべてのメモリボードは同一，マザーボードに実装するためにあらかじめ設定は不要

メモリボード装着用ソケットに，重複しないメモリアドレスが割当てられている

システム構成例２

コンピュータ立ち上げ時にMMUがソケットに番地を割り当て

MMUはどのソケットにメモリが実装されているか決定し，アドレスを割り当てる

複数のバスか1つのバスか

高性能コンピュータ（メインフレームなど）

複数のバスが搭載：

メモリバス，高速入出力バス，低速入出力バス

PCなど

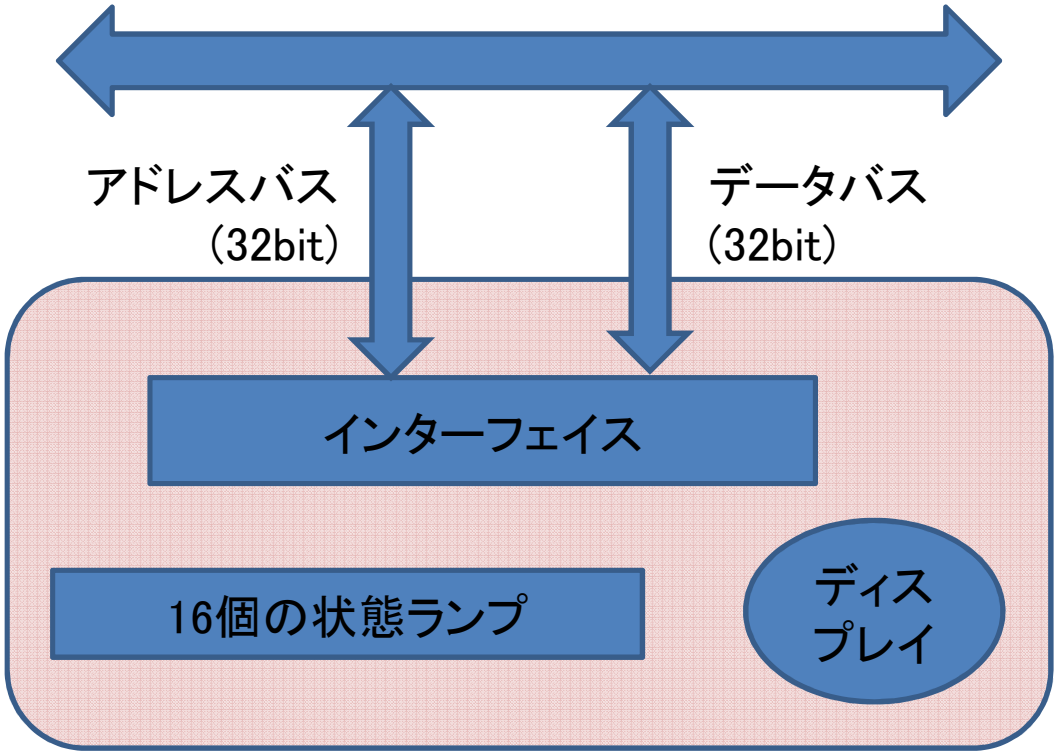
単独のバスを使用

低いコストと高い汎用性

単一のバスインターフェイスで構成

- フェッチ-ストアを装置に用いる
  - プロセッサと入出力装置の間の1次的な接続で，バスに対する捜査は，フェッチ-ストアパラダイムを使用
  - 装置の制御も，フェッチ-ストアパラダイムを使用

- フェッチ-ストアを使った装置制御の例
  - 16個の状態ランプを有する簡単なハードウエア装置
  - 装置が提供する機能
    - ディスプレイをつける
    - ディスプレイを消す
    - ディスプレイの明るさをセット
    - i番目の状態ランプを点灯/消灯する



アドレス	操作	意味
100 -103	ストア	非0の値でディスプレイをつけ，0で消す
100 -103	フェッチ	ディスプレイが消えているとき0を，ついていると非0
103 -107	ストア	明るさの変更，データの下位4ビットで明るさを設定
108 -111	ストア	下位16ビットでそれぞれ状態ランプを操作，0でランプを消灯，1でランプを点灯

注意：アドレスはバイトアドレス

アドレス集合に対する意味の割り当て例

Bアドレス	操作	意味
100–103	ストア	非0の値でディスプレイをつけ，0で消す
100–103	フェッチ	ディスプレイが消えているとき0を，ついていると非0
104–107	ストア	明るさの変更，データの下位4ビットで明るさを設定
108–111	ストア	下位16ビットでそれぞれ状態ランプを操作，0でランプを消灯，1でランプを点灯

- インタフェースの操作を言語的に表現すると

  if(アドレス==100 && 操作==ストア && データ != 0)

  　　ディスプレイをつける

  if(アドレス==100 && 操作==ストア && データ == 0)

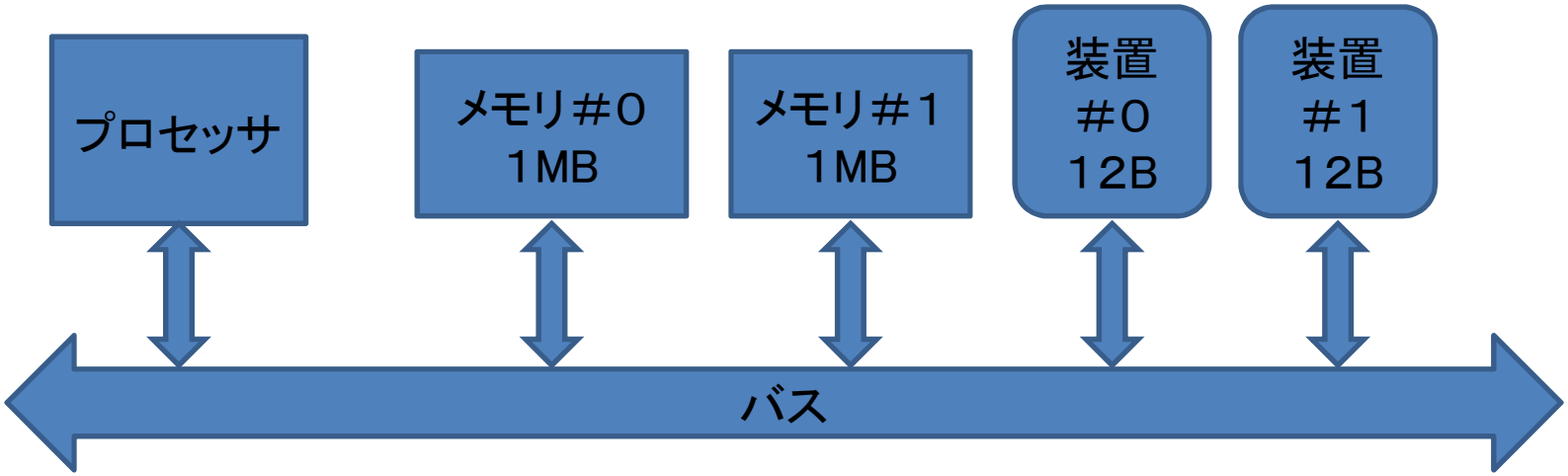
  　　ディスプレイを消す

非対称な割り当て

- 与えられたアドレスに対して，フェッチ，ストアの両方の操作が定義されている場合：対称な割り当て
  - 先の例では，101，102番地に対するフェッチ操作を定義していない．⇒非対称な割り当て
  - 100番地に対してはフェッチ，ストアともにできるが，101，102番地はフェッチしようとするとエラー

メモリと装置の統一的なアドレッシング

単一のバスで，メモリと入出力装置，双方へアクセスするアーキテクチャ



装置	アドレス
メモリ＃０	000000－0FFFFF
メモリ＃１	100000－1FFFFF
装置＃０	200000－20000B
装置＃１	20000C－200017

- アドレス空間の穴
  先のアドレスの割り当て：アドレス範囲は連続的
  ⇒ アドレス空間に穴がない

  多くのアーキテクチャ
  アドレス空間の低位部：メモリ
  高位部：装置

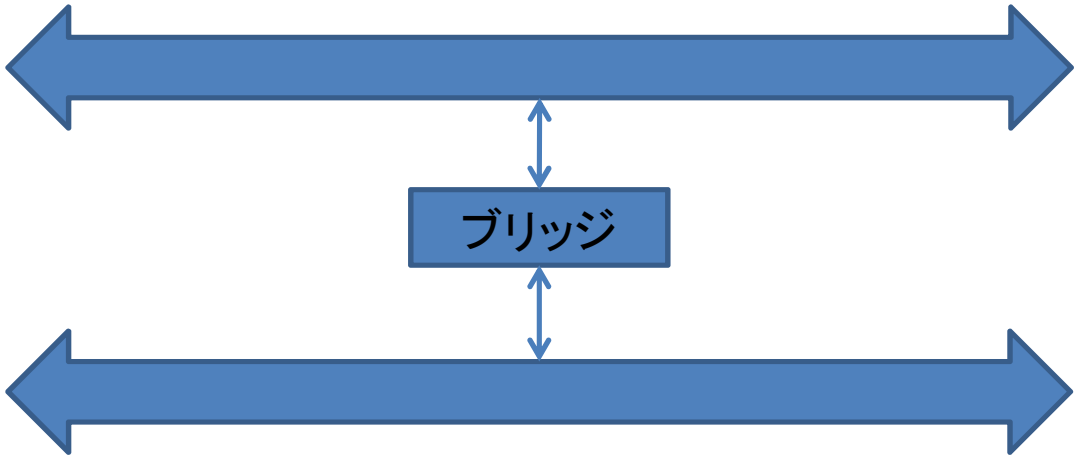
- アドレスマップ

- バスに対するプログラムインタフェース

```
int *ptr;
prt=(*int) 400;
*ptr=1;
```

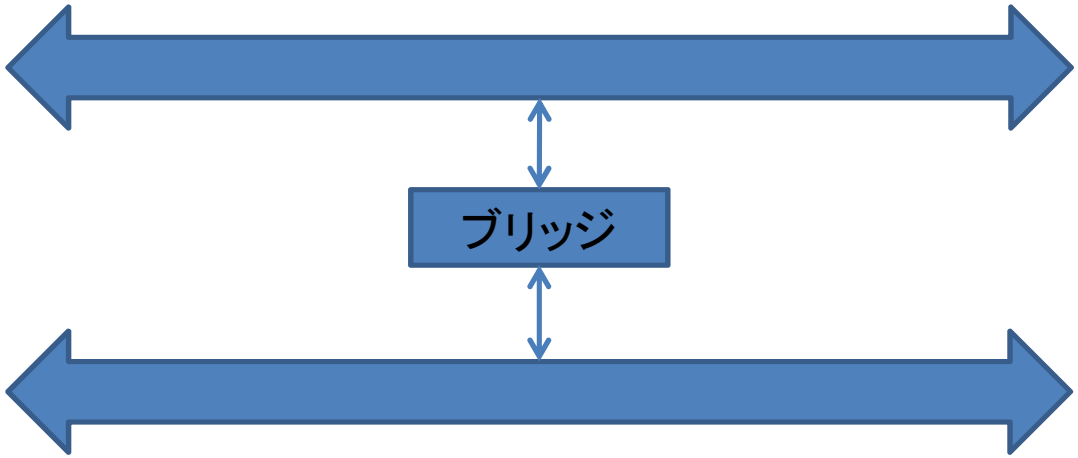
1つのバスを持つプロセッサ:
　メモリ操作命令により，装置の制御
複数のバスを持っているプロセッサ
　それぞれにアクセスするために特別な命令

- 2つのバスの橋渡し
  - 単一バスのコンピュータ
    - 単純性と低コスト
  - 複数バスのコンピュータ
    - 多様な装置を接続できる自由度
  - ⇒複数バスを装備する，高価でない方法
  - 2つのバスを連結するブリッジの採用



ブリッジ

- 装置やメモリと同様，ブリッジにはそれぞれのバスのアドレスが割り当てられる
- プロセッサからの要求には，直接は応答しない
- 1つのバスからの要求は，他のバスへアドレスを変換し，要求を転送
- バスからの応答についても同様



主バスと補助バス

ブリッジ：1つのバスのアドレス空間と，一方のバスのアドレス空間の1対1のマッピング

1つのバスのアドレスの集合が，他方のアドレス空間に見えるようにする

すでにバスを持つコンピュータに新しい装置を追加する場合

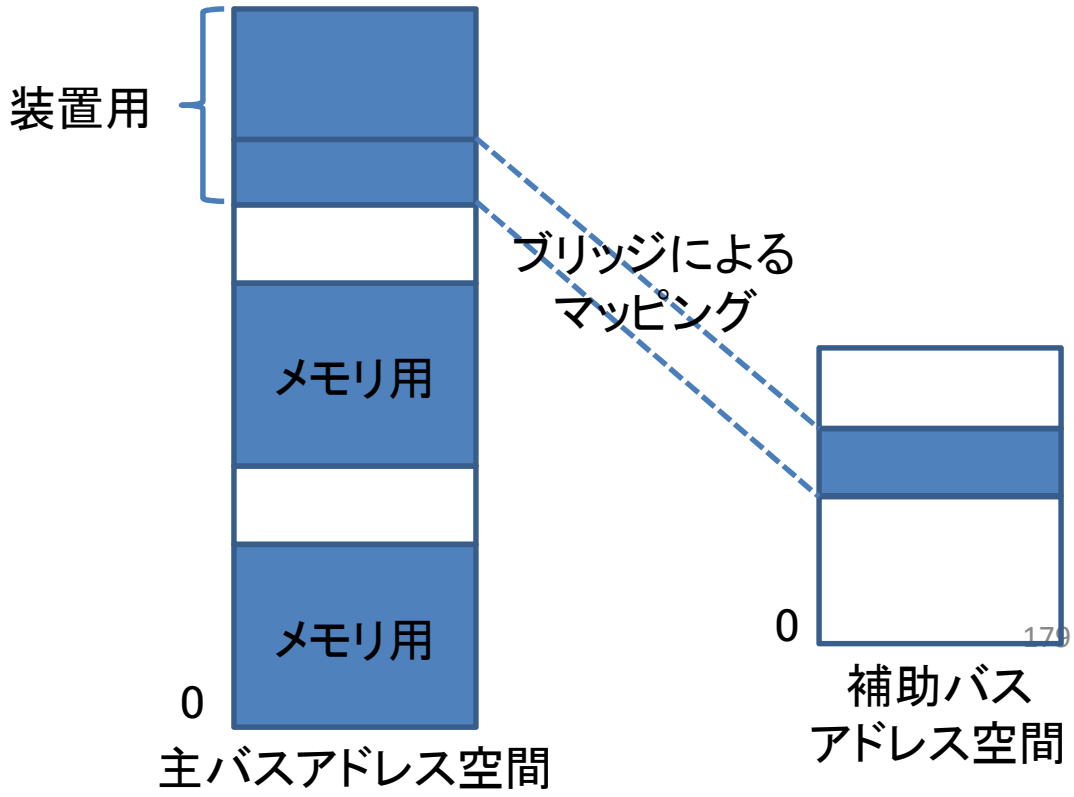
新しい装置のインターフェイスが，バスに合致しないとき

システムに対して，すでにあるバス（主バス）から新しい装置を接続する補助バスへのブリッジを使用

利点１：コンピュータの所有者がハードウエアを理解しなくとも，補助バスに新しい装置を接続できる点

利点２：ほとんどのバス用にブリッジが用意されているので，改めて開発する費用はない

– 双方のアドレスは0から始まっている
– 補助バスのアドレス空間のうち，一部をマッピングの対象としている
– 補助バスにマッピングされた装置は，主バスに接続されたように見える.



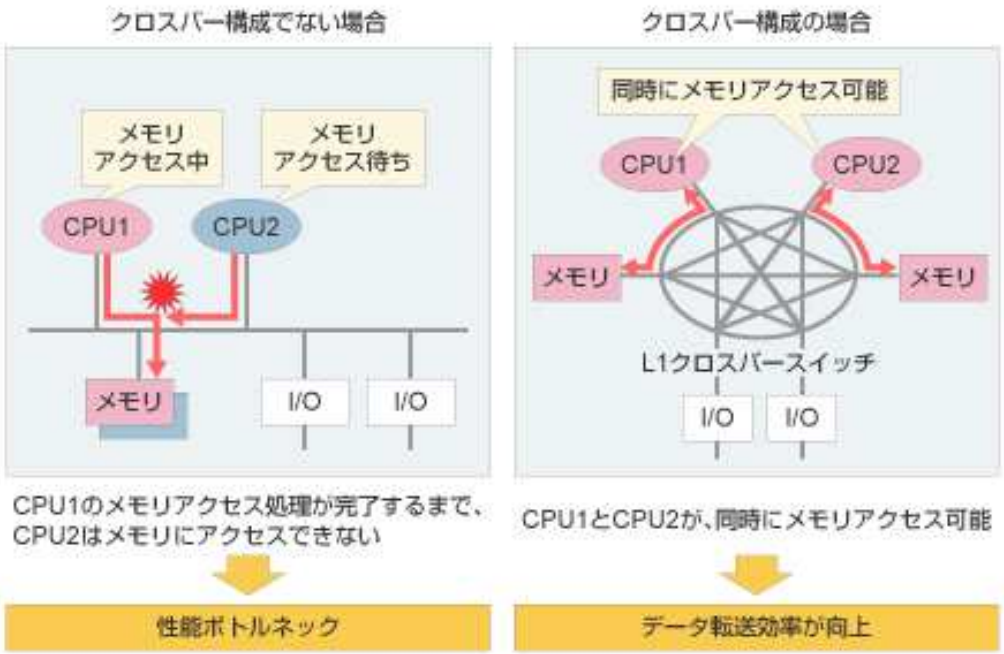
スイッチング機構

バスは一時に1つの転送しかできない

同時に複数の転送を可能にする機構 ⇒ スイッチ機構

ハードウエア量：入力N, 出力M

バス＝N+M ⟺ クロスバースイッチ＝N×M



プログラム駆動入出力と割込み駆動入出力

入出力パラダイム

1. 入出力装置はバスに接続し，装置に割当てられたアドレスに対して，フェッチとストア操作を発行し，プロセッサは装置とやり取り
2. 各装置はどのような制御操作を有しているか
3. プロセッサで稼働しているアプリケーションプログラムは，ハードウエアの詳細を知ることなく，どのように装置にアクセスするのか
4. プロセッサと入出力装置間のやり取りは，システムの全体的な性能に影響を及ぼさないのか

プログラム駆動入出力

外部装置は，フェッチとストア操作に対して，ハードウエアを制御する基本的なディジタル回路からなる

CPUはすべての詳細を取り扱う

プリンタの場合

1. 紙を進め
2. 印字ヘッドを移動し
3. 印字する文字を選び
4. 文字を打ち出す

CPUからの命令に反応する基本的な回路から成り立っている場合，装置はインテリジェントではない

⇒ 入出力がプログラム駆動である.

同期

インテリジェントでない装置

一連のコマンドを覚えておく必要がある

装置の回路は，プロセッサがコマンドを送信したときに，それぞれの命令を実行する

CPUの動作速度は入出力装置より高速な動作

⇒ プログラム制御の入出力は同期が必要

命令を発行するたびに，プロセッサは装置とやり取りをして次の指令を受け取れる状態かどうかを調べる必要がある

ポーリング

プロセッサが採用する入出力装置との同期の基本的な形

プロセッサが次の動作をする前に，装置に対して現在の操作が完了したかどうかを，繰り返し確認する

プリンタにより印字する場合

- プリンタが紙を進めるようにする
- 紙が進んだことを検出するためにポーリング
- 印字ヘッドを行の先頭に移動
- 印字ヘッドが票の先頭に移動したかポーリング
- …

ポーリングのコード

入出力装置：バスに接続されており，バスはフェッチ・ストアパラダイムに従っている

ポーリングはフェッチ操作による

1つ以上のアドレスが装置に割当てられ，CPUのフェッチ操作によりCPUに装置の現在の状態を伝える

バイトアドレス	操作	意味
0-3	ストア	非ゼロの値で紙送りを起動
4-7	ストア	非ゼロの値でヘッドを行の先頭に移動
8-11	ストア	印字する文字（下位バイト）
12-15	ストア	非ゼロの値で印字機構を起動
12-15	フェッチ	稼働中：非ゼロのとき，装置は起動中

```
int *p;                 /* バスの開始アドレスが 0x110000 に割り当てられて
                           いると仮定 */
p=0x110000;             /* 装置の最下位アドレス p+1=0x110004 */
*p=1;                   /* 紙送りを起動 */
while(*(p+3)!=0);       /* 紙送り完了をポーリング */
*(p+1)=1;               /* 印字ヘッドの移動を指示 */
while(*(p+3)!=0);       /* 印字ヘッドの移動をポーリング */
*(p+2)='c';             /* 'c'という文字を選択 */
while(*(p+3)!=0);       /* 文字選択完了をポーリング */
…
```

制御・状態レジスタ (CSR:Control and Status Reg.)

ストア操作に対応 : 制御レジスタ

フェッチ操作に対応 : 状態レジスタ

ビット単位での : 意味の割り当て

フェッチ操作とストア操作を同じアドレスに割当て

フェッチ操作を状態情報の要求とコントロール操作の両方に解釈するものも

マウス :

フェッチ操作によりマウスからのデータ読み込みと同時に，ハードウエアをリセット

プロセッサの使用とポーリング

利点：簡易なディジタル回路により実現可能

欠点：計算のオーバヘッドの発生

ポーリングを行うシステムの処理速度が，入出力装置の速度に(のみ)依存

高速なプロセッサを用いても改善なし

第１世代，第２世代，第３世代のコンピュータ

- 第１世代から第２世代のコンピュータへの移行時
- プロセッサの処理速度の大幅な向上

  ⇒ 入出力装置との速度差増大

世代	説明
第1	真空管使用の回路構成
第2	トランジスタ使用の回路構成
第3	割込み使用の入出力制御

割込み駆動入出力

ポーリングの時間の浪費:

入出力装置の動作中に，プロセッサが処理を継続

割込み駆動入出力の導入

- 入出力装置のハードウエアの変更
- バスのアーキテクチャと機能の追加
- プロセッサのアーキテクチャの変更
- プログラミングパラダイムの変更

- 入出力装置のハードウエアの変更

  単にプロセッサの制御の下での動作ではなく，稼働後は独立動作が必要．動作完了後はプロセッサへの通知機能

- バスのアーキテクチャと機能の追加

  プロセッサによる装置への操作の起動を指示するだけでなく，操作完了時には装置からの操作完了通知を転送⇒双方向通信の必要性

- プロセッサのアーキテクチャの変更

  通常の処理を一時停止し，装置の操作を可能とする仕組みの導入

- プログラミングパラダイムの変更

  ポーリング：同期的なプログラミングスタイル

  割込み：非同期的なプログラミングスタイル

ハードウエアの割込み機構

プロセッサは装置を起動

通常の命令の実行を継続

入出力装置がサービスを必要としたとき，バスを通して割込み信号を送信

プロセッサは命令の実行を一時停止

後に実行を復活するために必要な状態情報を退避

ハードウエアはプロセッサに対して，どの装置が割込みをかけたかを通知

割込み処理が終了すると，状態情報を読み込み通常の命令を再開

割込みとフェッチ-実行サイクル

プログラマ：

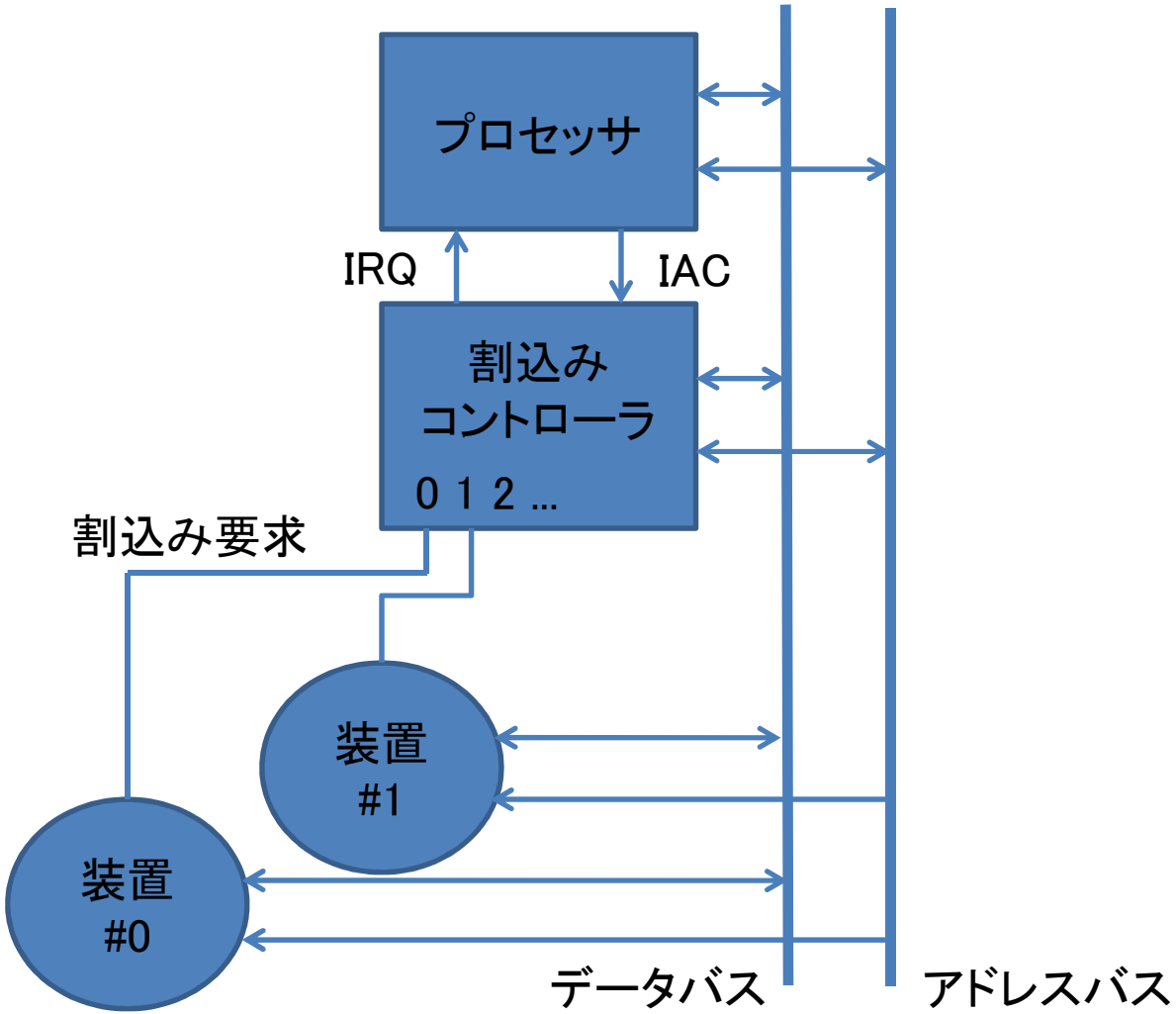
割込みの発生を無視して，通常の処理プログラムを記述

繰り返し{
　検査：もしどれかの装置が割込みを要求したら，割込み処理を実行し，ループの次の繰り返しをつづける
　フェッチ：プログラムが格納されている番地からプログラムの次のステップをアクセス
　実行：プログラムのステップを実行
}

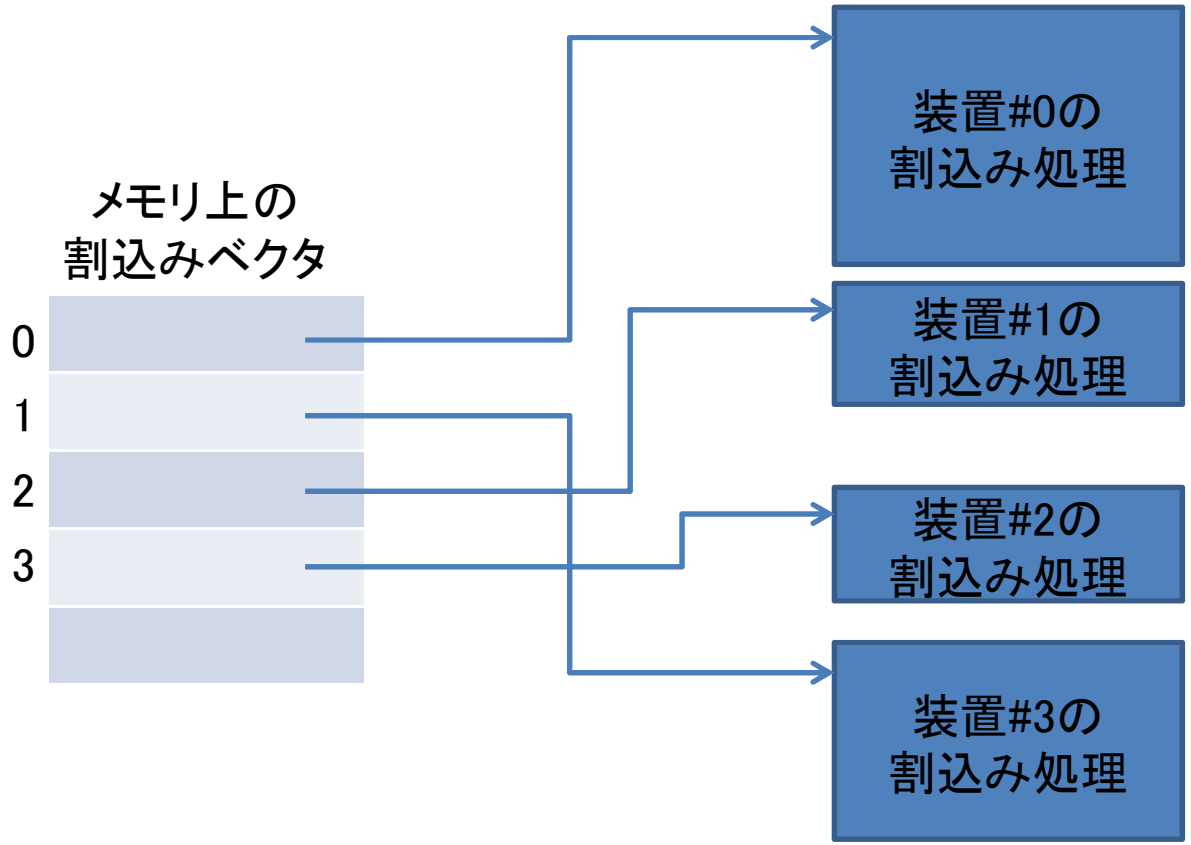
割込みの処理

- 現在の実行状態の退避
  レジスタを含む完全な状態情報の退避
- 割込み装置の確認
- 割込みルーチンの実行
- バスの割込み信号のクリア
- 現在の実行状態の復帰
  割込みからの復帰命令

割込みコントローラ



割込みベクタ



割込みハンドラ

- 初期化と割込みの許可/不許可
  - プロセッサの起動（割込み不許可状態）
  - 割込みベクタ表の初期化

  割込み許可命令の実行

- 割込みコードが割り込まれることの防止

  割込み処理中の割込みの禁止

割込みの複数レベル

多重レベルの割込み

割込みの多重優先度

プロセッサが優先レベルKで稼働中，K+1以上のレベルが割り当てられた装置からのみの割込みを許可

優先度0：割込み処理なし

1つの優先レベルでは1つの割込み処理

割込みベクタの割り当てと優先度

- 小規模なシステムでの固定的な手動割当て
  装置の回路上のディップスイッチの設定
- 汎用システムでの柔軟な自動割り当て

コンピュータの立ち上がり時に，バス上の装置の検索と，割込みベクタと優先度の設定

動的バス接続と活線装着可能な装置

割込みベクタと優先度が，立ち上り時に割当てられ，コンピュータの稼働時はその状態を保っているのが前提

コンピュータ稼働時に動的に，脱着される装置

USBの例

- USBはコンピュータ上の主バスの上の副バス
- コンピュータ立ち上がり時にUSBに割込みベクタが割当てられ，ハンドラがメモリ上に配置される
- 装置を装着時：USBは割込みを発生，プロセッサはハンドラを実行
- ハンドラにより装着装置を認識し，2次ハンドラをロード
- 装置のサービス開始時：USBは割込みを発生させ制御をUSBハンドラに写し，どの装置がサービスを必要としているか認識
- 適切な2次ハンドラの呼び出し

割込みが優れている点

割込みにより，ポーリングを使うコンピュータよりプログラミングが容易となり，高い入出力性能を得られる.

反面，割込みが多発するシステムでは，現在の状態の情報を退避する必要オーバヘッドが無視できない状況も発生

- 賢い装置と入出力性能の改善

一連の操作を装置自身で行うことのできる装置

⇒スマート装置

スマートでない装置

1. プロセッサはディスクの回転を起動する
2. 規定の速度に達したとき，ディスクが割り込む
3. プロセッサはディスクのヘッドを指定した場所へ移動することを指定する
4. ヘッドが指定された場所に達したとき，ディクスが割り込む
5. データを転送するためにリード操作を命ずる
6. 転送が完了したとき，ディスクが割り込む

スマート装置

プロセッサはディスク上の場所の指定と，メモリ上の場所の指定をして，リード操作を発する

ディスクは操作のステップを遂行し，操作を完了したときに割り込む

入出力エラーへの対応

ディスクが回転しない

記録面に欠陥がありディスクを読めない，など

エラーが恒久的なものか，一時的なものかの調査と判断

繰り返し操作を実行する必要

ダイレクトメモリアクセス：DMA

入出力装置がプロセッサを介さずにデータをメモリに転送すること

スマート装置が，直接メモリにアクセス

操作の各ステップでは，プロセッサへの割込みは発生させない

バッファの数珠繋ぎ

高速なネットワーク通信において，データを連続して読み込む場合

ネットワークに接続した装置が，一時に1つの処理

- 装置は1つのパケットが到着後，割込み
- 割込みの間，プロセッサは次のパケットを記憶するためのバッファの確保と，パケットの読み込みを装置に指示
- これらの一連の操作は，次のパケットが到着する前に処理が必要

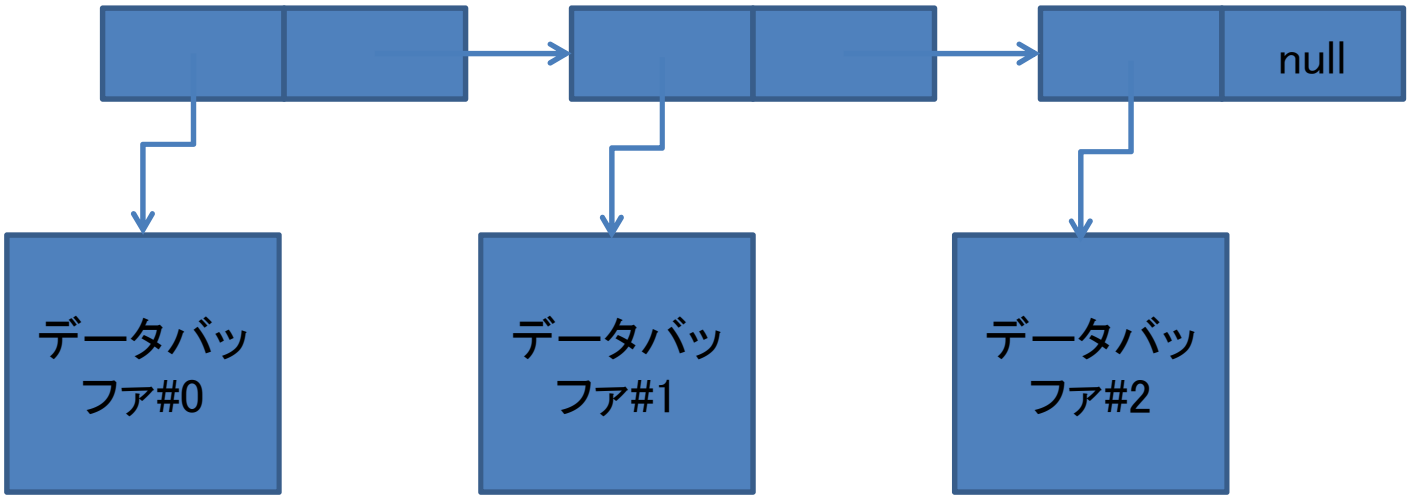
複数の装置が同時に割込みを行う場合，処理できない場合が存在

バッファの数珠つなぎ

複数のバッファを用意し，連結リストをメモリに構成

プロセッサはリストを入出力装置に渡し，装置がそれぞれのバッファを満たすことができるようにする．

装置に渡されるアドレス



撒き読み込み操作と集め書き出し操作

撒き読み込み操作

到来するデータの大きいブロックを分割して，複数の小さなバッファに格納すること

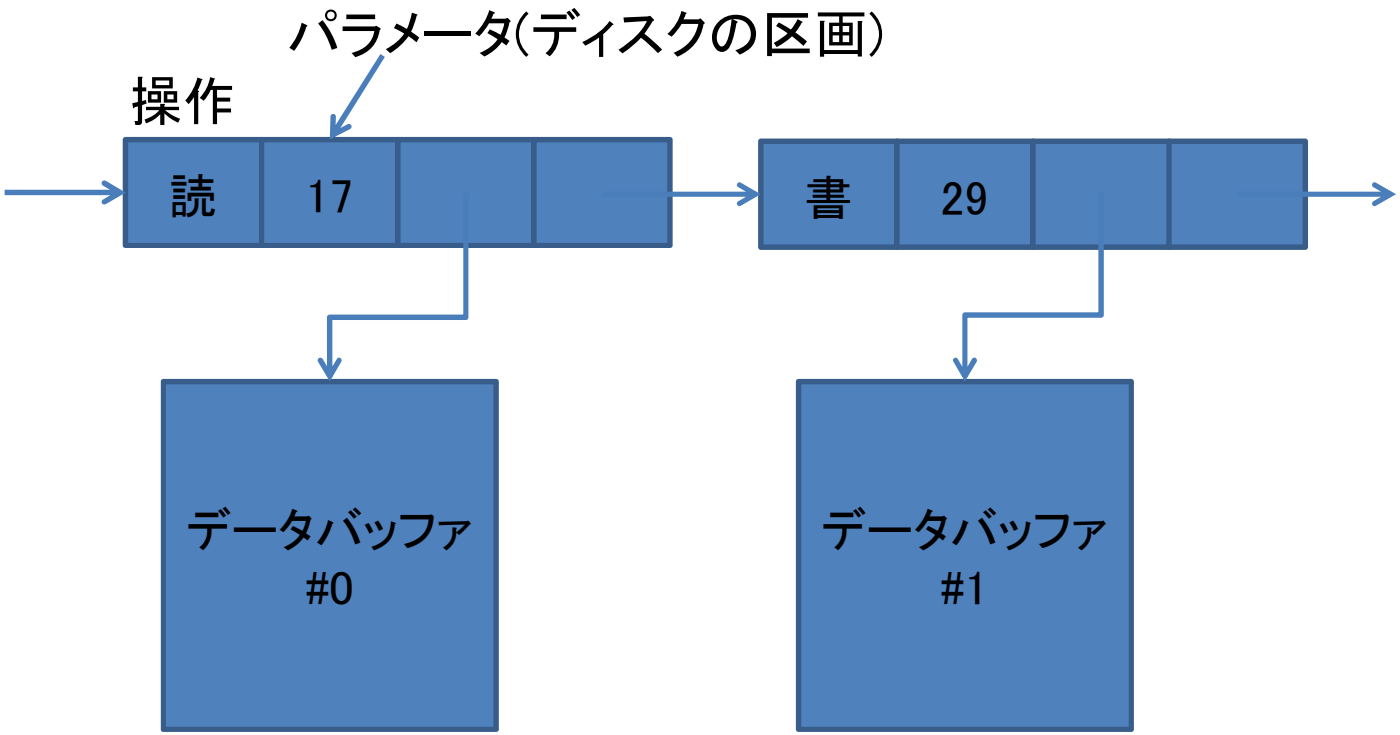
集め書き出し操作

複数の小さなバッファのデータを，まとめ上げて1つの出力ブロックとすること

操作の数珠繋ぎ

装置が複数の操作を行える場合

現在の操作完了直後に，新しい操作を開始



装置，入出力に対するプログラマの視点

デバイスドライバの定義

- アプリケーションプログラムと外部ハードウエアの間のインターフェイスを提供するプログラム
- コンピュータシステムは各外部装置のためのデバイスドライバを装備
- 所定の装置にアクセスするアプリケーションプログラムは同じドライバを使用
- デバイスドライバはコンピュータの操作システムの一部
- ドライバはバス上で装置と通信し，装置の制御・状態レジスタを理解し，装置からの割り込みを処理

装置独立，カプセル化と隠蔽

- デバイスドライバの目的 ⇒ 装置独立性の確保
- ハードウエアの詳細な知識は不要
- デバイスドライバがハードウエアの細部をカプセル化（隠蔽）する

  例：プリンタの変更

  アプリケーションプログラムの変更は不要

  デバイスドライバの入れ替え

デバイスドライバの概念的な部品

- デバイスドライバ
  - バス上で通信するコード
  - 装置の細部の処理を担うコード
  - アプリケーションプログラムとの通信を行うコード
  - OSとの通信

- ドライバの構成
  - 割り込みが発生するときに起動するハンドらで構成される下半分
  - 入出力操作を要求するアプリケーションプログラムで起動する関数で構成される上半分
  - ２つの部分を調整するのに必要な状態情報を持つ共有変数一式

- 上半分と下半分

  ハードウエア：低いレベル

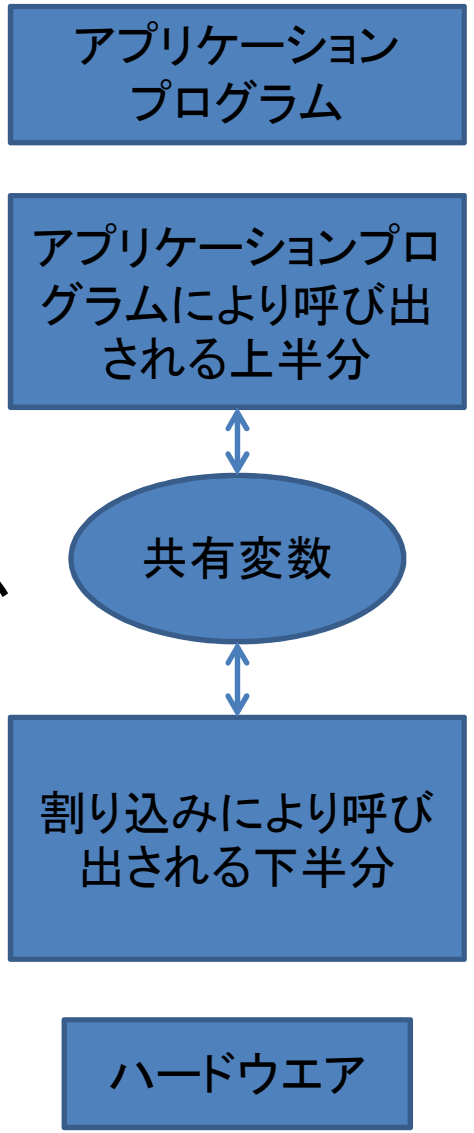
  アプリケーションプログラム：

  　　高いレベル

  最上位の階層：

  　　アプリケーションプログラム

  最下層：ハードウエア



2つのタイプの装置

使用するインターフェイスの型により2つのカテゴリに分類可能

- 文字指向の装置

  一度に1バイトのデータを転送

  文字の送受信ごとに割り込みを発生

  例）キーボードとコンピュータ間のインターフェイス

- ブロック指向の装置

  一度にデータブロック全体を転送

  例）ディスク装置とコンピュータ間のインターフェイス

  ディスクのセクタと同一のブロックサイズ

  ネットワークインターフェイス

  1パケットと同一サイズ（ブロックサイズは可変）

待ち行列を使った出力操作

初期化

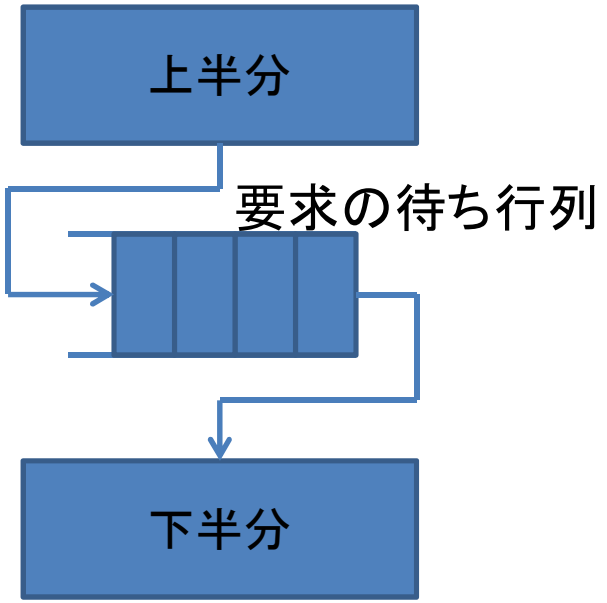
1. 入力待ち行列を空に初期化

上半分（アプリケーションプログラムが書き出しを実行）

1. 待ち行列にデータ項目を登録
2. CSRを使って割り込みを要求
3. アプリケーションプログラムに戻る

下半分（割り込み発生）

1. 待ち行列が空なら，装置の割り込みを止める
2. 待ち行列が空でないなら，データ項目を取り出し，出力を開始
3. 割り込みから戻る



割込みの強制

- プロセッサが装置に割り込みを強いるように設定できるよう，装置にはCSRビットが実装されている
- もし装置が稼働中でないならば，CSRビットの設定により割り込みが発生
- 稼働中ならば，稼働中の処理を待ち割り込みを発生

上半分は，装置が稼働中かどうかを判断する必要はない

データを待ち行列に書き出すとともに，CSRをセットするだけでよい

待ち行列化された入力操作

初期化

1. 入力待ち行列を空に初期化
2. 装置に割り込みを発生させる

上半分（アプリケーションプログラムが読み込みを実行）

1. 待ち行列が空なら，アプリケーションプログラムを止める
2. 入力待ち行列の次のデータ項目を取り出す
3. アプリケーションプログラムにデータ項目を返す

下半分（割り込み発生）

1. 待ち行列が満杯でないなら，別の入力操作を開始
2. アプリケーションプログラムが停止中なら，実行を開始
3. 割り込みから戻る

双方向の転送をサポートする装置

双方向装置：入力と出力，双方を許す装置

双方向装置の取り扱い

- 2つの別々の装置として

  内向きと外向きの2つの待ち行列

  1つの転送方向を処理する2つのドライバで構成

- 2種類のコマンドを処理する単一の装置として

  要求の待ち行列は1つ，それぞれの要求が方向を指示

  例）ディスク装置

“非同期なプログラミングの枠組み”対“同期的なプログラミングの枠組み”

- ポーリング : 同期的

  初めから終わりまでコードを通して制御の受け渡しが行われる
- 割り込みを処理するデバイスドライバ : 非同期的

アプリケーションプログラムから見た入出力

- 高級言語の利用
- 入出力操作の表現

⇒ プログラミング言語が提供する抽象化を行う

ディスク装置の取り扱い：ファイル

ディスプレイ装置の取り扱い：ウインドウ

など

実行時入出力ライブラリ

- コンパイラは入出力操作を行うためにライブラリ関数を呼び出すコードを生成
- 実行時にプログラムは適切なライブラリ関数とリンクされる

実行時ライブラリ

ライブラリ/OS二分論

- デバイスドライバはOSに位置
- 実行時ライブラリはOSの外
- ハードウエア装置上の3つソフトウエアの層

OSが提供する入出力操作

- C言語など

  OSインターフェイスはアプリケーションプログラムで直接利用可能

  プログラマは，入出力ライブラリを利用するか，OSレベルへのシステムコールをするかを選択可能

操作	意味
open	装置を利用可能にする
read	装置からアプリケーションプログラムにデータを転送
write	アプリケーションプログラムから装置にデータを転送
close	装置の利用を停止
seek	装置の新しいデータの位置に移動
ioctl	様々な制御機能

入出力操作のコスト

- 実行時ライブラリ内で関数を起動する際のコスト
  →手続きを呼び出すときと同様
- 制御：システムコールを通してOS関数の起動
  →極めて高いオーバーヘッド
  - OSはアプリケーションより大きな特権で稼働
    ⇒特権レベルの変更が必要
  - アプリケーションプログラムの仮想アドレス空間からOSのアドレス空間に，メモリ空間の変更が必要
  - その間のデータのコピーが必要

バッファリングの重要な概念

- バッファリング :

  入出力のシステムコール回数を減らすため

  バッファでデータを蓄積し

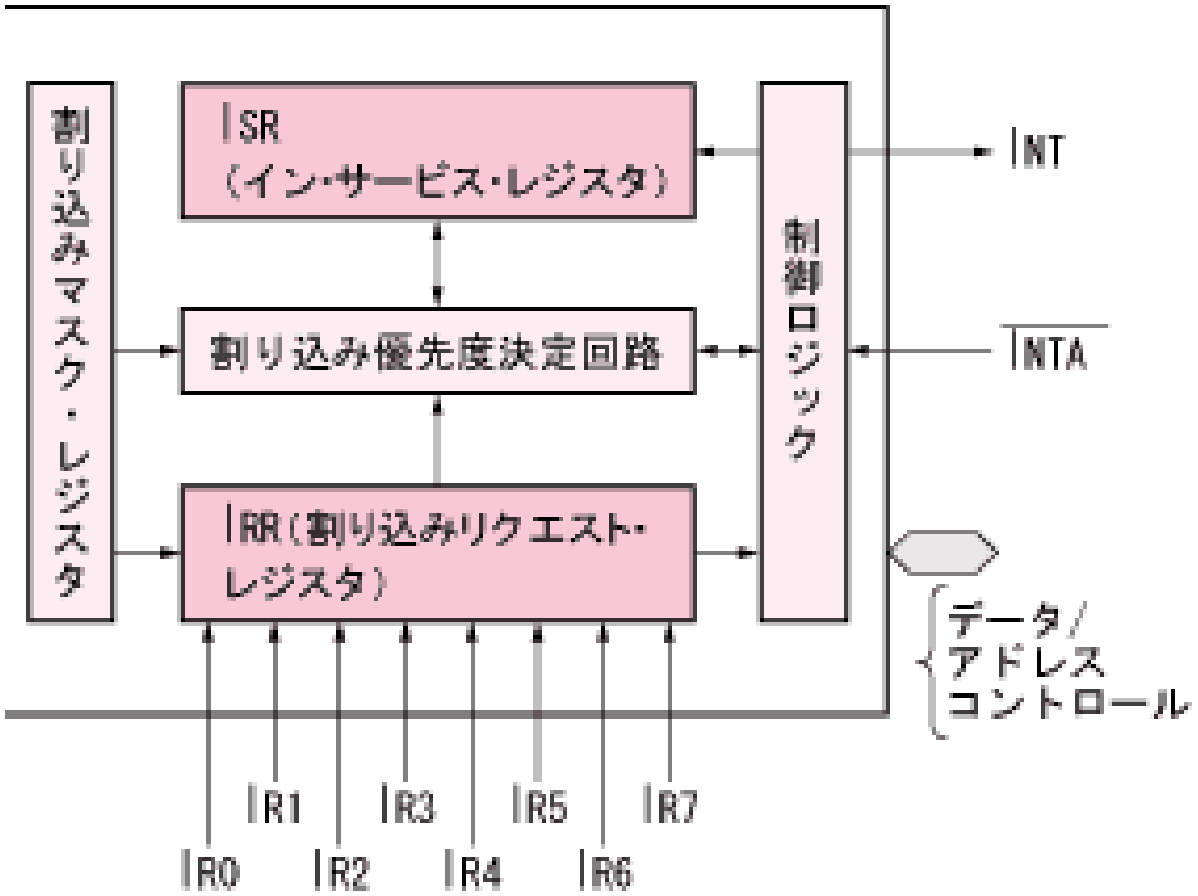
  システムコールが起こるたびにより多くのデータを転送

UNIX標準入出力ライブラリ

機能	意味
fopen	バッファの準備
fgetc	バッファ化された1バイトの入力
fread	バッファ化された複数バイトの入力
fwrite	バッファ化された複数バイトの出力
fprintf	バッファ化された整形付複数バイトの出力
fflush	バッファ化された出力のフラッシュ（掃出し）
fclose	バッファの使用の打ち切り

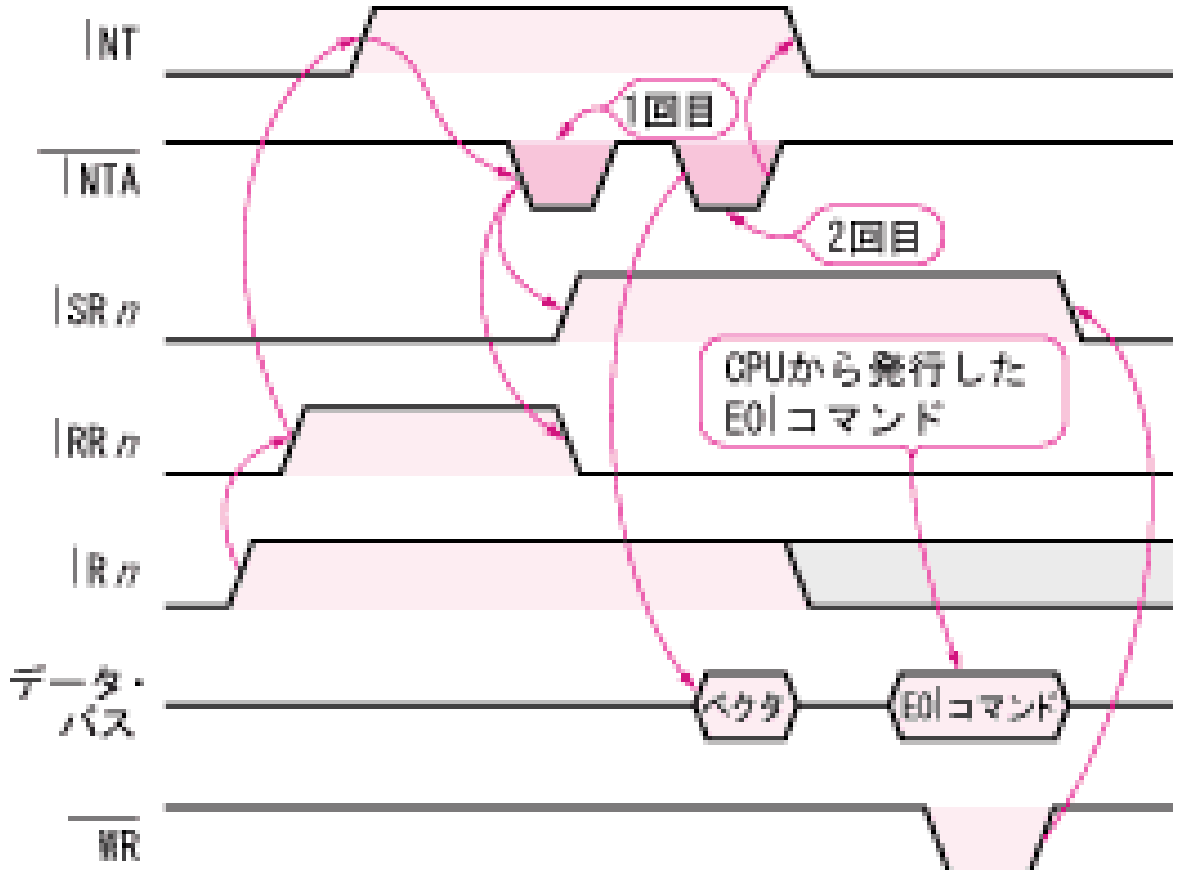
8259割り込みコントローラ

ブロック図



8259割り込みコントローラ

タイムチャート



8259割り込みコントローラ

回路例