アセンブリ言語

割り込み，リアルモード

情報工学系
権藤克彦

割り込み (interrupt)

割り込みとは？

- 割り込み(interrupt)
  - CPU外部で発生したイベントを非同期的にCPUに伝える仕組み.
  - イベント発生時に声をかけてもらう＝割り込み.
  - いつ発生するか予測できないことに即座に反応して処理できる.

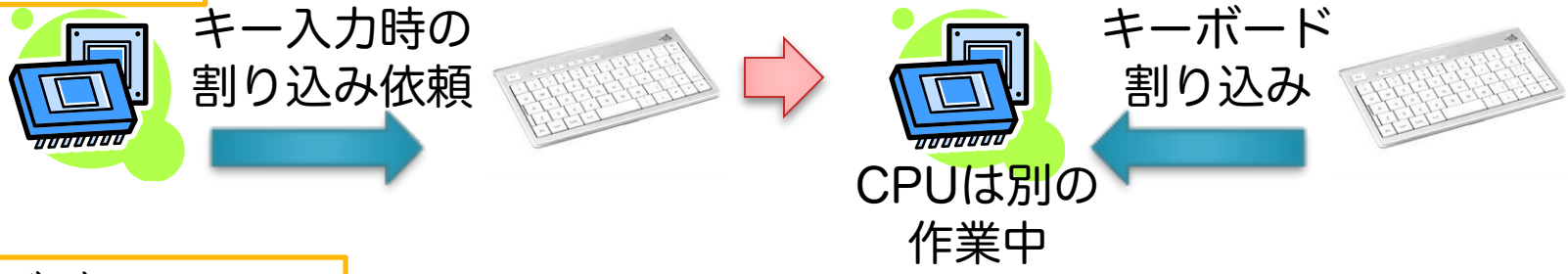
- 割り込みはOSの実現にとても重要.
  - cf. コンパイラでは不要.

非同期とは？

多くの場合，非同期方式が好ましい．
例外はあるが．

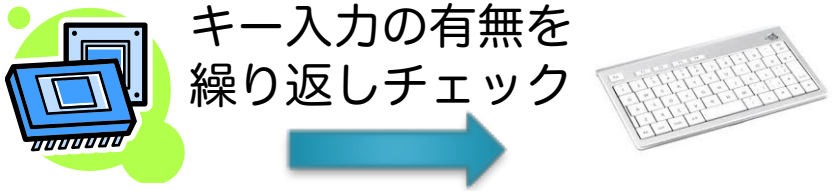
- 非同期(asynchronous)
  - 待ち合わせ（同期）しないこと．
- 割り込みは非同期．
  - cf. ビジーウェイト(busy wait)，ポーリング(polling)

割り込み



ビジーウェイト

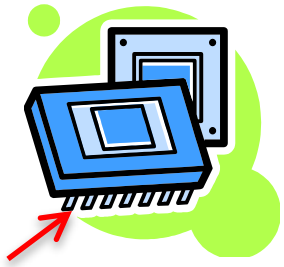
ポーリング



ビジーウェイトはCPUを
100%消費してループでチェック．
ポーリングは別作業もしつつ
定期的にチェック．

割り込みピン(interrupt pin)

- CPUは（外部）割り込みを割り込みピンで受け取る.
  - 例：x86-64ではINTRピン（80386ではB7ピン）.
- CPUのピン
  - 外部と電気的に接続するCPUの接点．電圧変化で0か1を伝達.



Intel Core i7のピン

http://angel-halo.jugem.jp/?eid=1073316

割り込みの分類

単に「割り込み」と言ったら
こちらを指すのが普通.

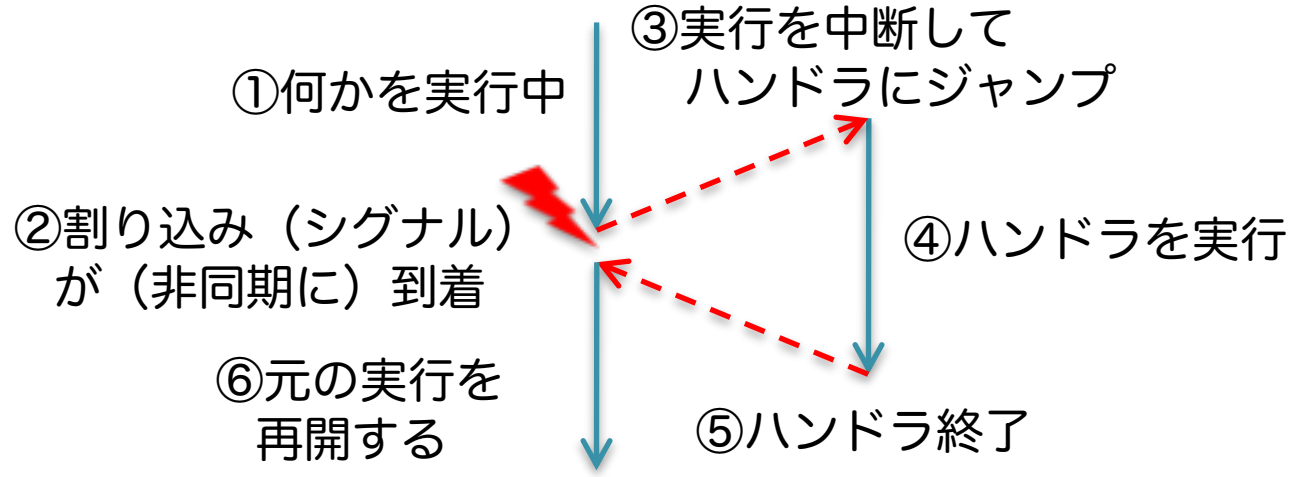
- ハードウェア割り込み
  - CPUの外部からの非同期的な割り込み.
    - 例：キーボード押下.

- ソフトウェア割り込み
  - CPU内部で発生させる（同期的な）割り込み.
    - 例：ゼロ除算.
  - 呼称：例外(exception)，トラップ(trap)，フォルト(fault)
    - 意味はCPUベンダごとに異なる
  - ソフトウェア割り込みを発生させる命令もある.
    - 例：int命令（後述）

システムプログラミングで学習

割り込みとUNIXシグナル（１）

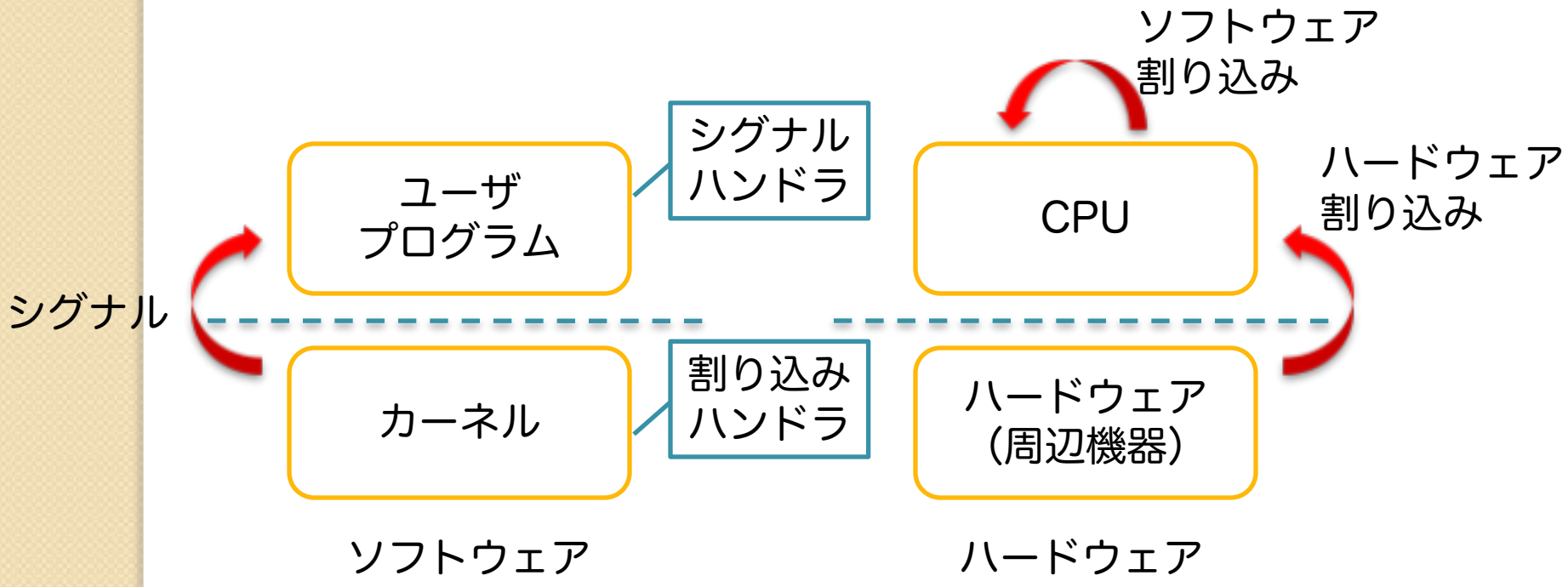
- 割り込みとUNIXシグナルは似ている.
  - （非同期に）イベントが到着する.
  - そのイベント処理のための手続き（割り込みハンドラ，シグナルハンドラ）が実行される.
  - 終了すると，元の実行を（何事もなかったように）再開・実行.



関数呼び出しとは異なり，
ハンドラを呼び出すコードは無い.

割り込みとUNIXシグナル（2）

- 発生（処理）する場面が違う.
  - 割り込みはCPUレベルの話.
    - 割り込みハンドラはカーネルが実行.
  - UNIXシグナルはOSがAPIとして提供するサービス.
    - シグナルハンドラはユーザプログラムが実行.



フェッチ実行サイクル（再）

実はこの部分あり！

- 割り込み処理
  - 割り込みがあれば，その割り込みを処理する．
  - 割り込み処理中もフェッチ実行サイクルを回る．
- フェッチ(fetch)
  - プログラムカウンタが指す機械語命令をメモリからCPUに読み込む．
- デコード(decode)
  - 読み込んだ命令を解析して，実行の準備をする．
  - 次の機械語命令を指すようにプログラムカウンタの値を増やす．
- 実行 (execute)
  - 読み込んだ機械語命令を実行する．

フェッチ実行サイクル中で割り込みの有無をチェック．

簡単な方から説明

ソフトウェア割り込み（1）：命令

int3はデバッガで，intはシステムコール呼び出しによく使う．

形式	例	説明
int 3	int3	デバッガのブレークポイント．1バイト長．
int *imm8*	int $4	ソフトウェア割り込み．
into	into	OF=1なら割り込む．
iret	iret	割り込みからのリターン
bound *r16, m16&16* bound *r32, m32&32*	bound %dx, 4(%bp) bound %edx, 4(%ebp)	配列のインデックス境界をチェック．
ud2	ud2	無効オペコード例外を発生．

int
bound
ud2

OF	SF	ZF	AF	PF	CF

iret

OF	SF	ZF	AF	PF	CF
x	x	x	x	x	x

ソフトウェア割り込み（2）：発生条件

- CPUが命令実行時に「ある条件」を検出すると発生.
  - 例：0による除算が発生.

- x86-64のソフトウェア割り込みの例：
  - 除算エラー，ブレークポイント，オーバーフロー，無効オペコード
  - ページフォルト，一般保護例外，セグメント不在

ソフトウェア割り込み（3）：例

- 一部のソフトウェア割り込みは，ユーザ空間ではUNIXシグナルとして見える．

foo.c

```
int main (void)
{
    asm ("int3");
}
```

```
% gcc -g foo.c
% gdb ./a.out
(gdb) run
Program received signal SIGTRAP,
Trace/breakpoint trap.
main () at foo.c:4
4       }
(gdb)
```

foo2.c

```
int main (void)
{
    asm ("ud2");
}
```

```
% gcc -g foo2.c
% gdb ./a.out
(gdb) run
Program received signal SIGILL,
    Illegal instruction.
main () at foo2.c:3
3   asm ("ud2");
(gdb)
```

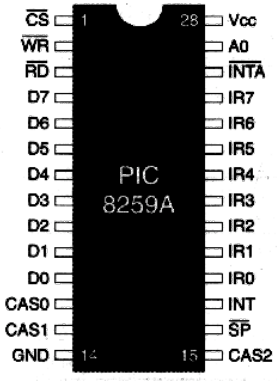
割り込み番号，割り込みベクタ

- 割り込み番号(interrupt number)
  - 割り込みを区別する番号．割り込みは割り込み番号を持つ．
  - x86-64ではなぜか割り込みベクタ(interrupt vector)と呼ぶ．
  - （ソフトウェア割り込みにも割り込み番号がある．）
- CPUは割り込み番号を受け取る．
  - 割り込みピンで割り込み通知後で，CPUは割り込みコントローラ(PIC)からデータバス経由で割り込み番号を受け取る．

CPU

割り込みがあったよ

割り込み番号は3ね



PIC

リアルモードの

OADG BIOSマニュアルより

PC/AT互換機の割り込み番号の例（1）

- PC/AT互換機(PC/AT compatible machine)
  - IBMのPC/AT（1984年製）と互換のパソコン.
  - 現在のいわゆるパソコンはPC/AT互換機を拡張したもの.

番号	説明
0	0による除算
1	シングルステップ
2	NMI割り込み
3	ブレークポイント
4	オーバーフロー
5	画面印刷
6, 7	予約
8	タイマー
9	キーボード
A	H/W割り込み
B	H/W割り込み
C	H/W割り込み
D	予約
E	フロッピーディスク
F	H/W割り込み

ハードウェア割り込み

番号	説明
10	ディスプレイ入出力
11	装置構成情報
12	メモリサイズ情報
13	ディスクドライブ入出力
14	非同期通信入出力
15	システム入出力
16	キーボード入出力
17	プリンタ入出力
18	予約
19	予約
1A	時刻
1B	キーボードBreakアドレス
1C	ユーザタイマー割り込み
1D - 1F	予約

BIOSコール（後述）

リアルモードの

OADG BIOSマニュアルより

PC/AT互換機の割り込み番号の例（2）

番号	説明
20 – 6F	DOSファンクションコール
70	リアルタイムクロック
71	予約
72	H/W割り込み
73	H/W割り込み
74	マウス
75	コプロセッサ
76	ハードディスク
77	H/W割り込み
78 – FF	予約

ハードウェア割り込み

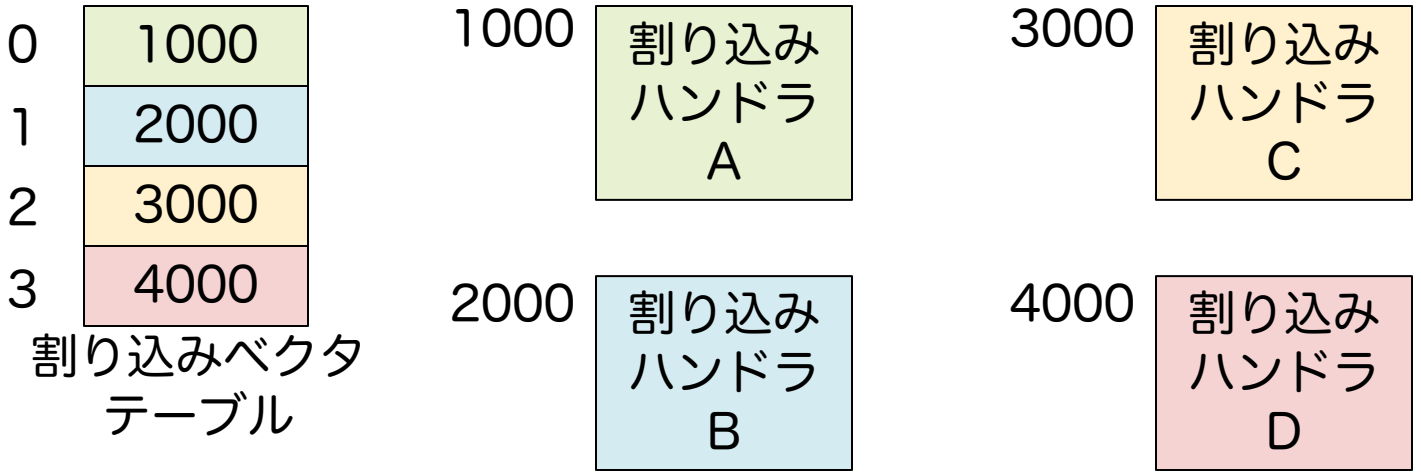
割り込みハンドラ

- 割り込みハンドラ(interrupt handler)
  - 割り込み発生時に，CPUが自動的に呼び出す関数（手続き）．
  - ISR (interrupt service routine)とも呼ぶ．
  - 通常の関数とは関数呼び出し規約が異なる．
    - 例：割り込みハンドラの場合，エラーコードや%eflagsの値がスタック上に積まれる．
  - 通常の関数とは，お作法が少し異なる．
    - 割り込みハンドラ中で実行をブロック（例：sleep）してはいけない．
    - 割り込みハンドラの実行はなるべく短時間で終了すべき．
    - 割り込みハンドラはリエントラントではない関数を呼んではいけない．
      - リエントラント(reentrant)＝ある関数の実行中に，さらにその関数を非同期に呼び出すことができる関数の性質．
      - 例：多くの場合，printf を呼んではいけない．競合状態がおこる．
    - ．．．．

競合状態はシステムプログラミングで学ぶ．

割り込みベクタテーブル

- 割り込みベクタテーブル(interrupt vector table)
  - 割り込み番号と割り込みハンドラの対応を保持する配列.
  - 割り込み番号が i で，この配列を v とすると，v [i] は呼び出すべき割り込みハンドラの先頭アドレスになる.

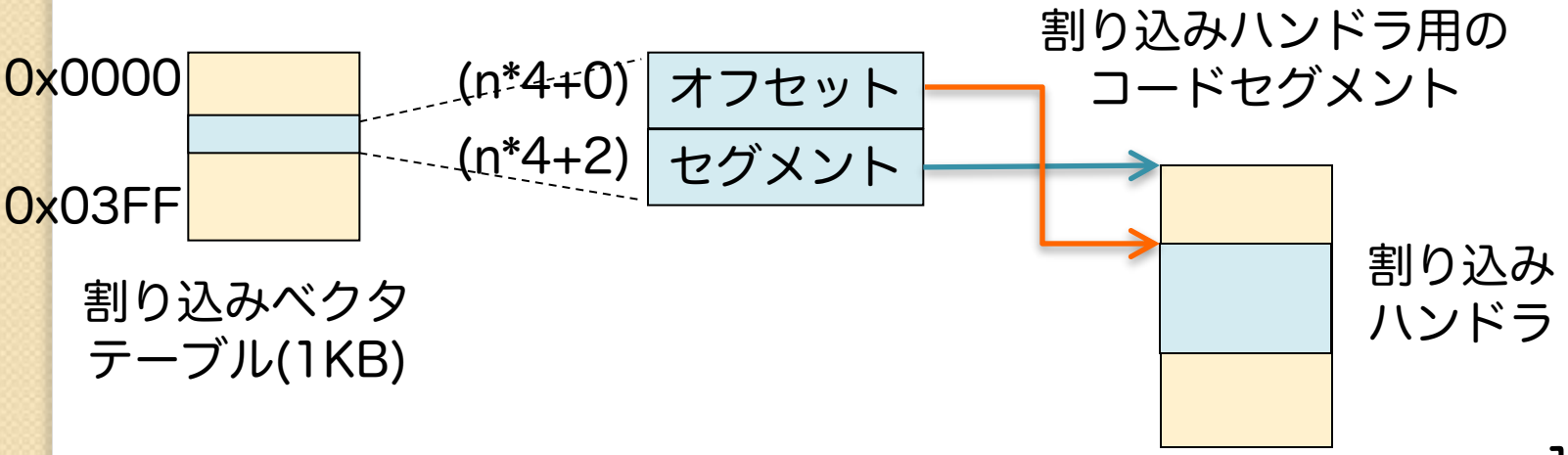


必要に応じて，プログラマは割り込みベクタテーブルの中身をあらかじめ修正する.

割り込み番号２の割り込みが発生すると割り込みハンドラＣをCPUが呼び出す.

i386の割り込みベクタテーブル（リアルモード）

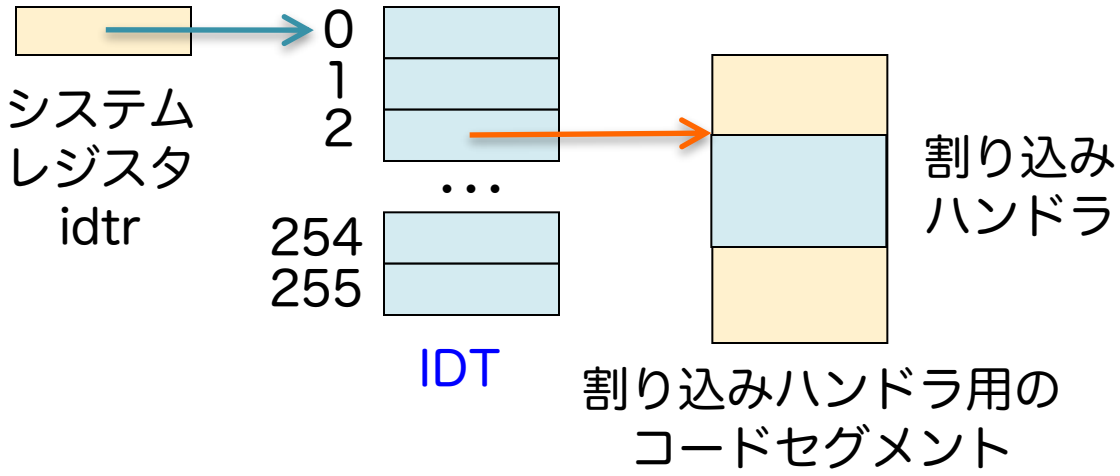
- 存在するメモリアドレスは0x0000～0x03FF.
  - 4バイトのfarポインタが256個．(256*4=1KB)
- 各farポインタは割り込みハンドラの先頭番地を指す.
  - 割り込み番号 n に対応する割り込みハンドラの,
    - セグメントは (n*4+2)番地，オフセットは(n*4+0)番地に格納される.
  - 割り込み番号 n の割り込みハンドラは int $n で起動可能.



i386の割り込みベクタテーブル（保護モード）

少々複雑なので
詳細は省略.

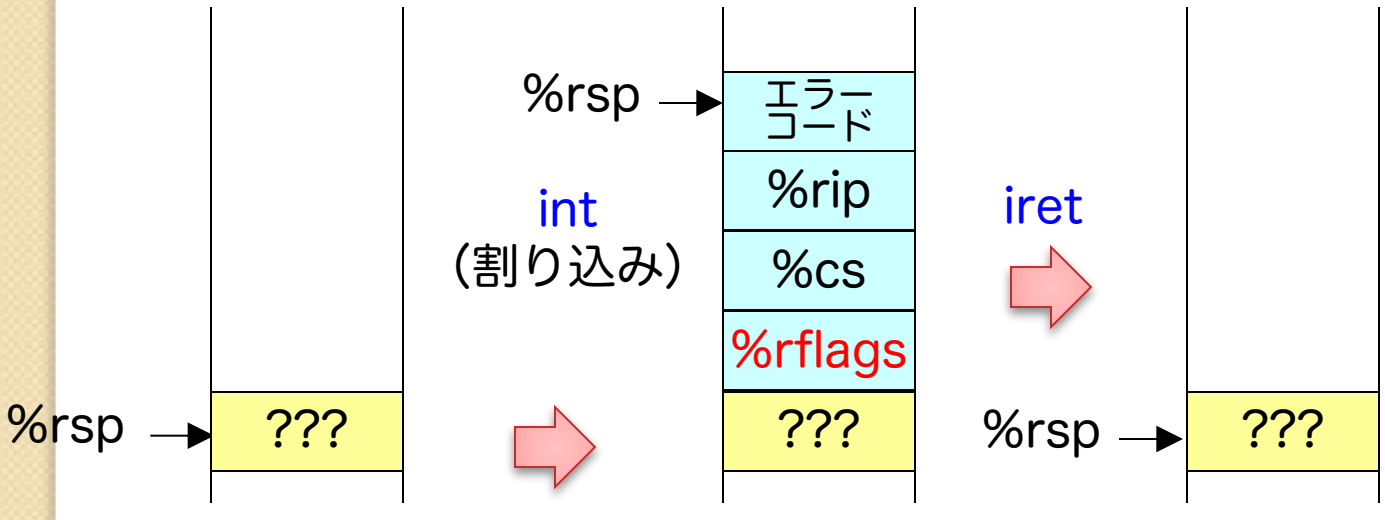
- リアルモードのものとは別形式.
  - IDT (interrupt descriptor table)
  - 8バイトのエントリを最大256個登録可能.
  - 好きな場所に配置可能(0番地でなくても良い）.



IA-32eだとちょっと異なる

割り込み時のスタックの状態（1）

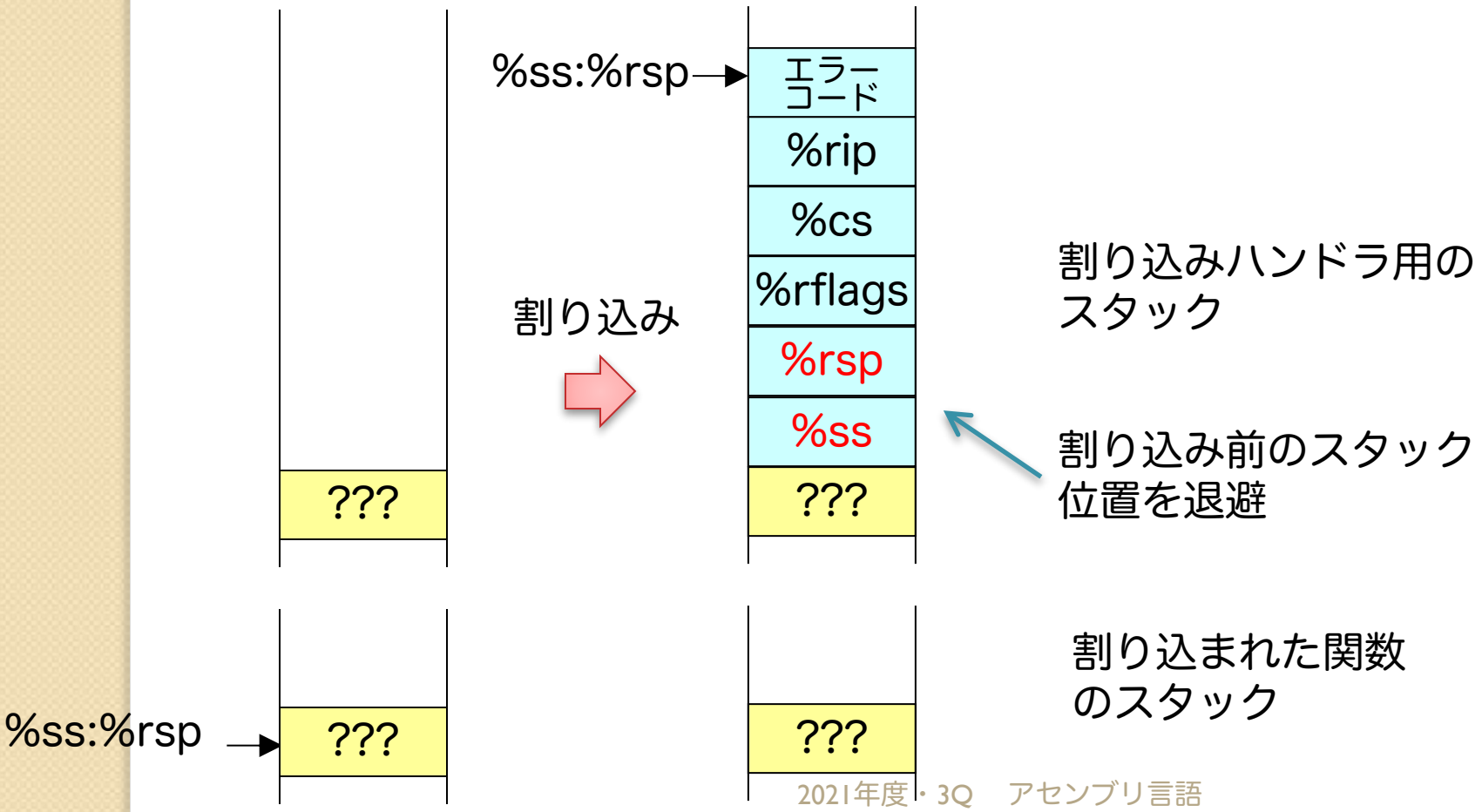
- 割り込みハンドラに制御が移った際のスタック
  - スタック切り替えが無い場合.



- エラーコードをスタックに積まない割り込みもある.
- iret は %rflagsをポップする点だけが，retと異なる.

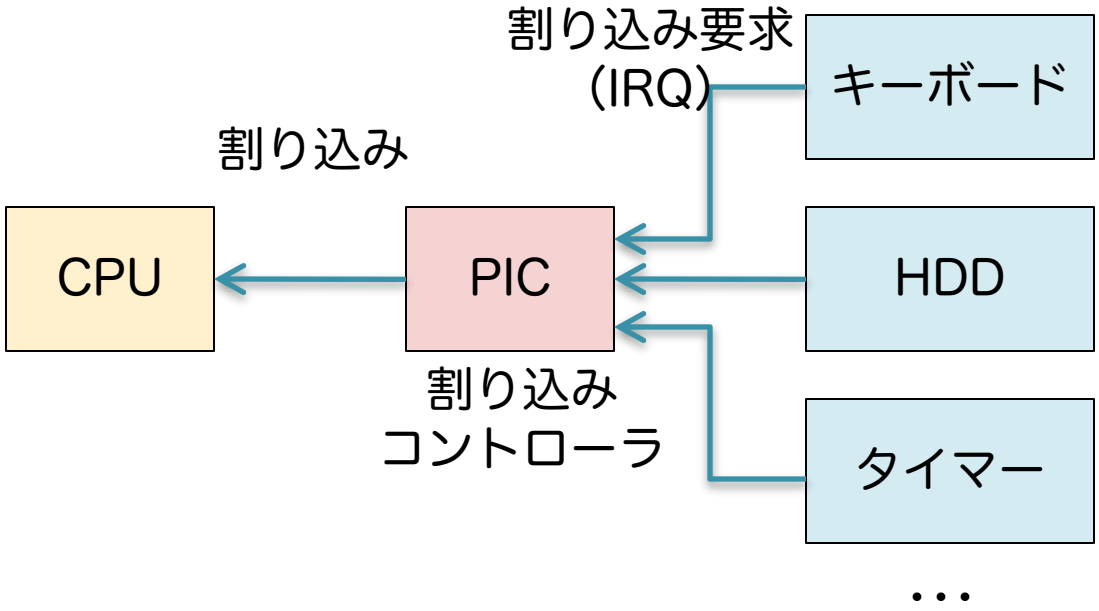
割り込み時のスタックの状態（2）

- 割り込みハンドラに制御が移った際のスタック
  - スタック切り替えがある場合.



割り込みコントローラ

- 割り込みコントローラ
  - PIC (programmable interrupt controller)
  - 入出力装置からの割り込みを集約して，CPUに伝える.
  - ハードウェア割り込みの処理には，PICの設定が必要.
  - 例：Intel 8259A

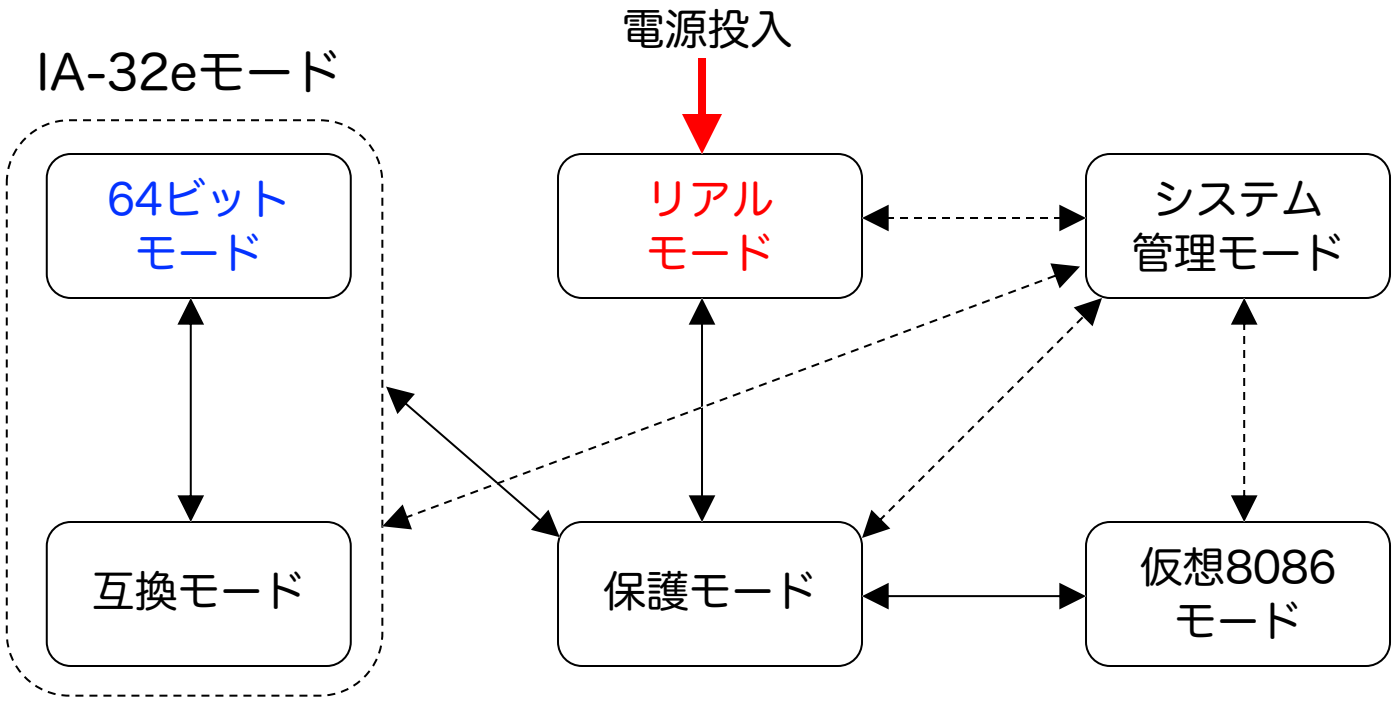


リアルモード (real-address mode)

正式には実アドレスモード

リアルモードと64ビットモード

- x86-64は次の6つの動作モードを持つ.
  - 電源投入直後はリアルモード.
  - OS管理下では64ビットモード.



リアルモード

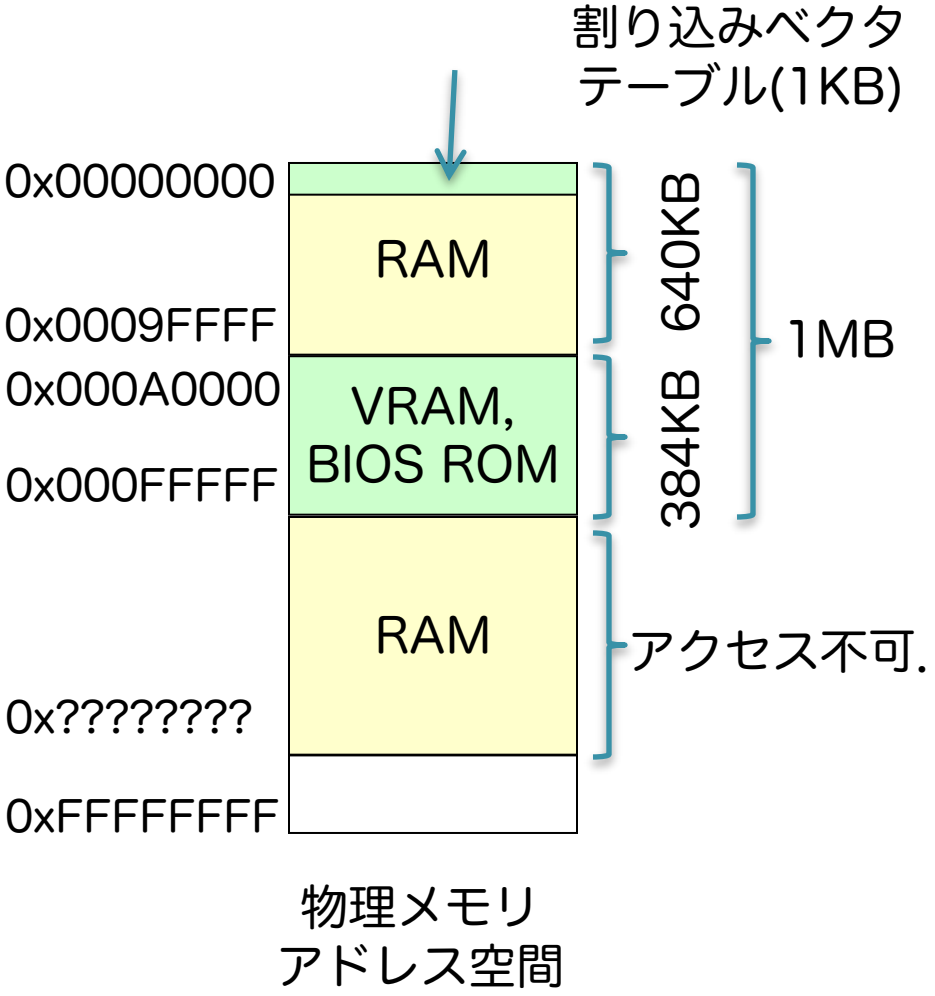
HDDの内容も破壊可能.
つまり，とても危険.

- 保護がないので何でもできる.
  - 特権命令や入出力命令を実行できる.
  - BIOSコールも呼び出せる.
  - ページングはオフ．生の物理メモリにアクセス可能.
- 8086プロセッサ互換のため，機能は貧弱.
  - 物理アドレス空間は 20ビット（1MB）.
  - 最大セグメントサイズは 64KB.
  - デフォルトのアドレス・データサイズは16ビット.
    - 32ビットアドレスへのアクセスは可能.
  - 保護モード・ページングに移行するのは（可能だが）面倒.
    - LILOやGRUBなどのブートローダを使うと比較的簡単.

PC/AT互換機の

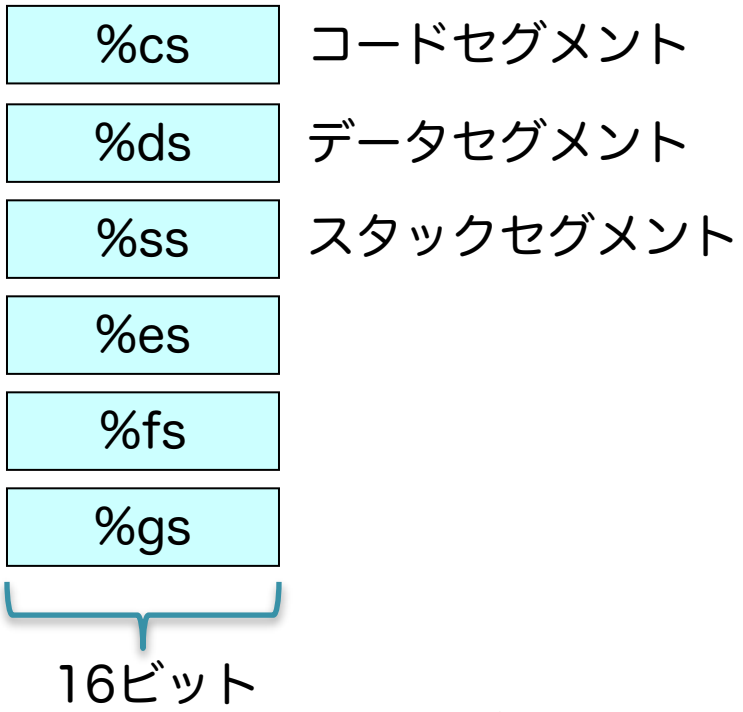
システムメモリマップ（リアルモード）

- 先頭640KBはRAM.
  - その先頭1KBは割り込みベクタテーブル.
- 次の384KBはVRAMとBIOS ROM（後述）.
- アクセスできるのは先頭1MBだけ.



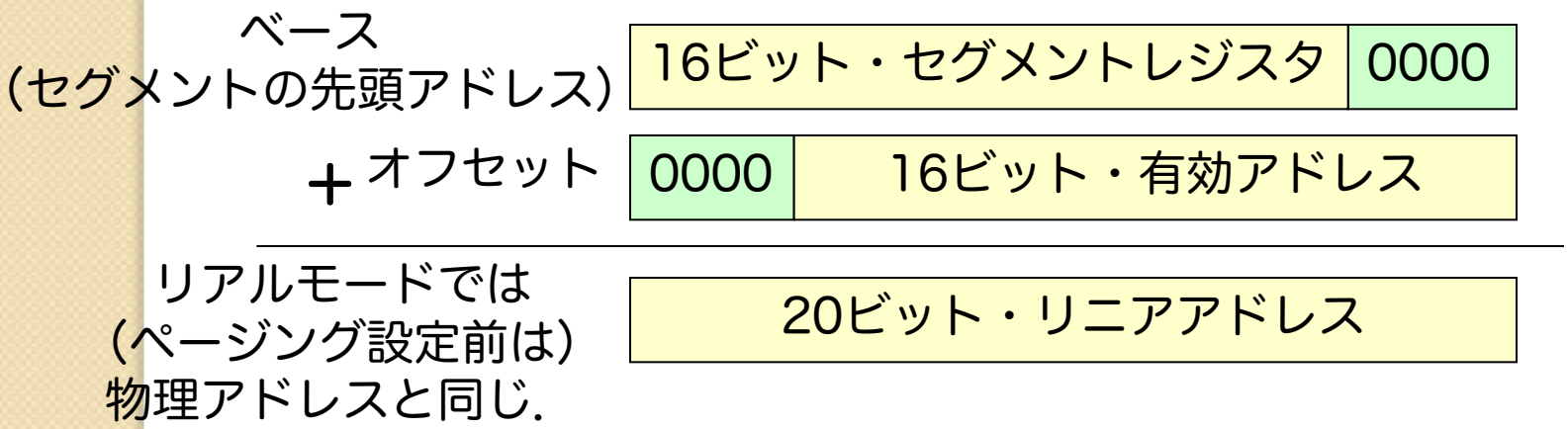
セグメントレジスタ（リアルモード）

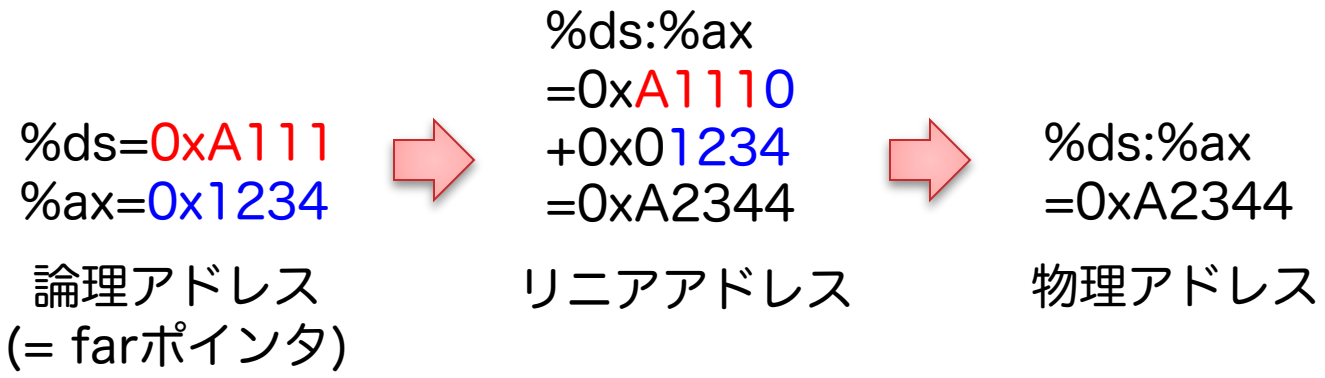
- 16ビット長
- 最大64KBのセグメントの先頭を指す.
  - 4ビット左にシフトしてから.



リアルモードのアドレス計算

- セグメントレジスタの値を4ビット左シフトしてから，オフセットアドレスを足す．

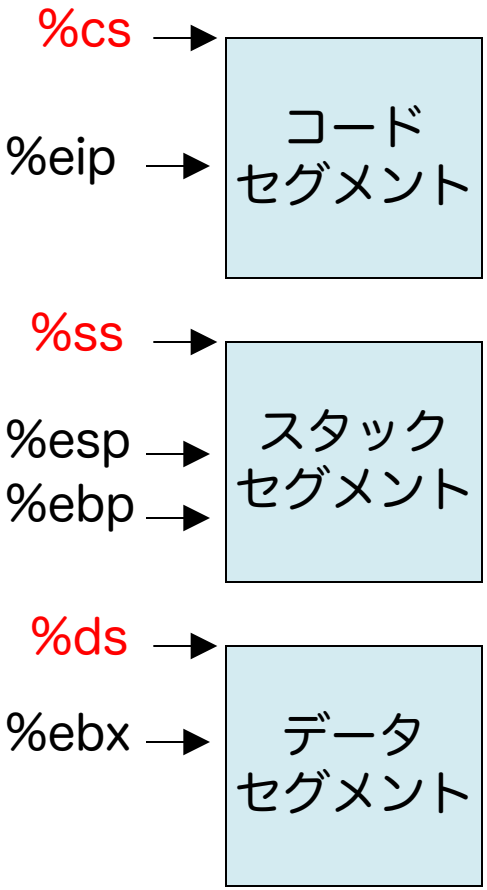




デフォルトのセグメント選択規則（1）

- 次の規則で，CPUは自動的にセグメントを選択する．

使用するレジスタ	使用されるセグメント	デフォルトで選択するもの
%cs	コードセグメント	%eipが指す機械語命令
%ss	スタックセグメント	%espや%ebpが指すスタック参照． push/pop命令．
%ds	データセグメント	その他のデータ参照すべて． ただし，スタックやストリング命令の 転送先の場合を除く．
%es	%esが指すデータ セグメント	ストリング命令の転送先．

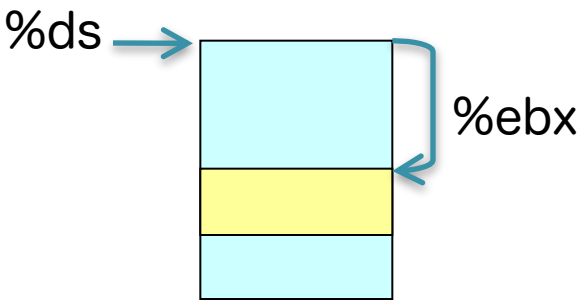


デフォルトのセグメント選択規則（2）

- 例：(%ebx)に対しては，%dsが自動的に選択される.

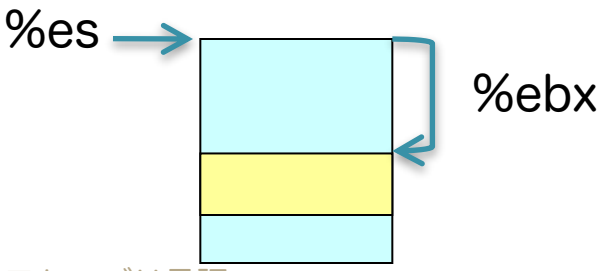
どちらも同じ

```
movl        (%ebx), %eax
movl  %ds:(%ebx), %eax
```



- 例えば「%es:」と指定すればデフォルトの選択を無効にできる.
- 無効にできない場合：
  - %csがフェッチする機械語命令.
  - %esが指すストリング命令の転送先.
  - %ssによるpush/pop操作.

```
movl  %es:(%ebx), %eax
```



Intelのマニュアルでは「プリフィックス」

Mac OS Xでは，lock/inc などとスラッシュで区切る.

命令プレフィックス(instruction prefix)

- 5種類ある.

プレフィックスの種類	プレフィックス
セグメントを指定	cs, ds, ss, es, fs, gs
データサイズを指定	data16, data32
アドレスサイズを指定	addr16, addr32
バスをロック	lock
ストリング命令用	rep, repe, repne, repnz, repz

- 使用例：

```
   40        inc  %eax
f0 40     lock inc  %eax
```

バスをロック
(inc命令の実行がアトミックになる)

```
   8b 10     mov        (%eax),  %edx
   8b 10     movl  %ds:(%eax),  %edx
64 8b 10     mov   %fs:(%eax),  %edx
```

デフォルトのセグメント以外にアクセスできる.

データサイズ・プレフィックス

.code32はデフォルトのデータとアドレスのサイズが32ビットであることをGNUアセンブラに伝える.

- 使用例：

data32.s

```
.code32
.text
push            $0x1234
pushl           $0x1234
pushw           $0x1234
data16 push $0x1234
```

```
% gcc –c data32.s
% objdump -d –Msuffix data32.o
    68 34 12 00 00   pushl   $0x1234
    68 34 12 00 00   pushl   $0x1234
66 68 34 12          pushw   $0x1234
66 68 34 12          pushw   $0x1234
```

data16.s

```
.code16
.text
push            $0x1234
pushl           $0x1234
pushw           $0x1234
data32 push $0x1234
```

```
% gcc -c code16.s
% objdump -d -Mi8086,suffix data16.o
    68 34 12          pushw   $0x1234
66 68 34 12 00 00   pushl   $0x1234
    68 34 12          pushw   $0x1234
66 68 34 12 00 00   pushl   $0x1234
```

同じ機械語命令(例：68）の解釈が16ビットモードか32ビットモードかで変化.

アドレスサイズ・プレフィックス

- 実効アドレスのサイズを変更する.

addr32.s

```
.code32
.text
push          4(%ebp)
pushl         4(%ebp)
pushw         4(%bp)
addr16 pushw  4(%bp)
```

```
% gcc –c addr32.s
% objdump -d –Msuffix addr32.o
ff 75 04                pushl
0x4(%ebp)
ff 75 04                pushl
0x4(%ebp)
67 66 ff 76 04  addr16 pushw    4(%bp)
67 66 ff 76 04  addr16 pushw    4(%bp)
```

addr16.s

```
.code16
.text
push          4(%ebp)
pushl         4(%ebp)
pushw         4(%bp)
addr32 pushl  4(%ebp)
```

```
% gcc –c addr16.s
% objdump -d -Mi8086,suffix addr16.o
    67 ff 75 04  addr32 pushw 0x4(%ebp)
67 66 ff 75 04  addr32 pushl  0x4(%ebp)
       ff 76 04            pushw     4(%bp)
67 66 ff 75 04  addr32 pushl  0x4(%ebp)
```

.code16, .code16gcc, .code32

- データとアドレスのサイズを指定するアセンブラ命令.
- .code16gcc は，Cの関数を呼ぶ時に使用（後述）.

アセンブラ命令	説明
.code16	デフォルトのデータとアドレスのサイズが16ビットのコードを生成.
.code16gcc	.code16と同じ．ただし，call，ret，enter，leave，push，pop命令だけはデフォルトを32ビットにする.
.code32	デフォルトのデータとアドレスのサイズが32ビットのコードを生成.

hlt（1）

- hlt は命令の実行を停止する.
  - ただし，割り込み，NMI，リセットにより実行を再開.

命令	例	説明
hlt	hlt	命令の実行を停止する

実行を再開したくない場合は
jmp命令と組み合わせて使う.

```
1:hlt; jmp 1b
```

NMI (non-maskable interrupt)
割り込みをマスク（割り込み不可に）できない割り込み.

hlt（2）

- hltは特権命令なので，ユーザ空間では使えない．

```
int main ()
{
   asm ("1:hlt; jmp 1b");
}
```

```
% gcc -g foo.c
% gdb ./a.out
(gdb) run
Program received signal EXC_BAD_ACCESS,
Could not access memory.
Reason: 13 at address: 0x00000000
main () at foo.c:3
3       asm ("1:hlt; jmp 1b");
(gdb)
```

CygwinではSIGILL，LinuxではSIGSEGVが出る．