１．アセンブラ入門（１）

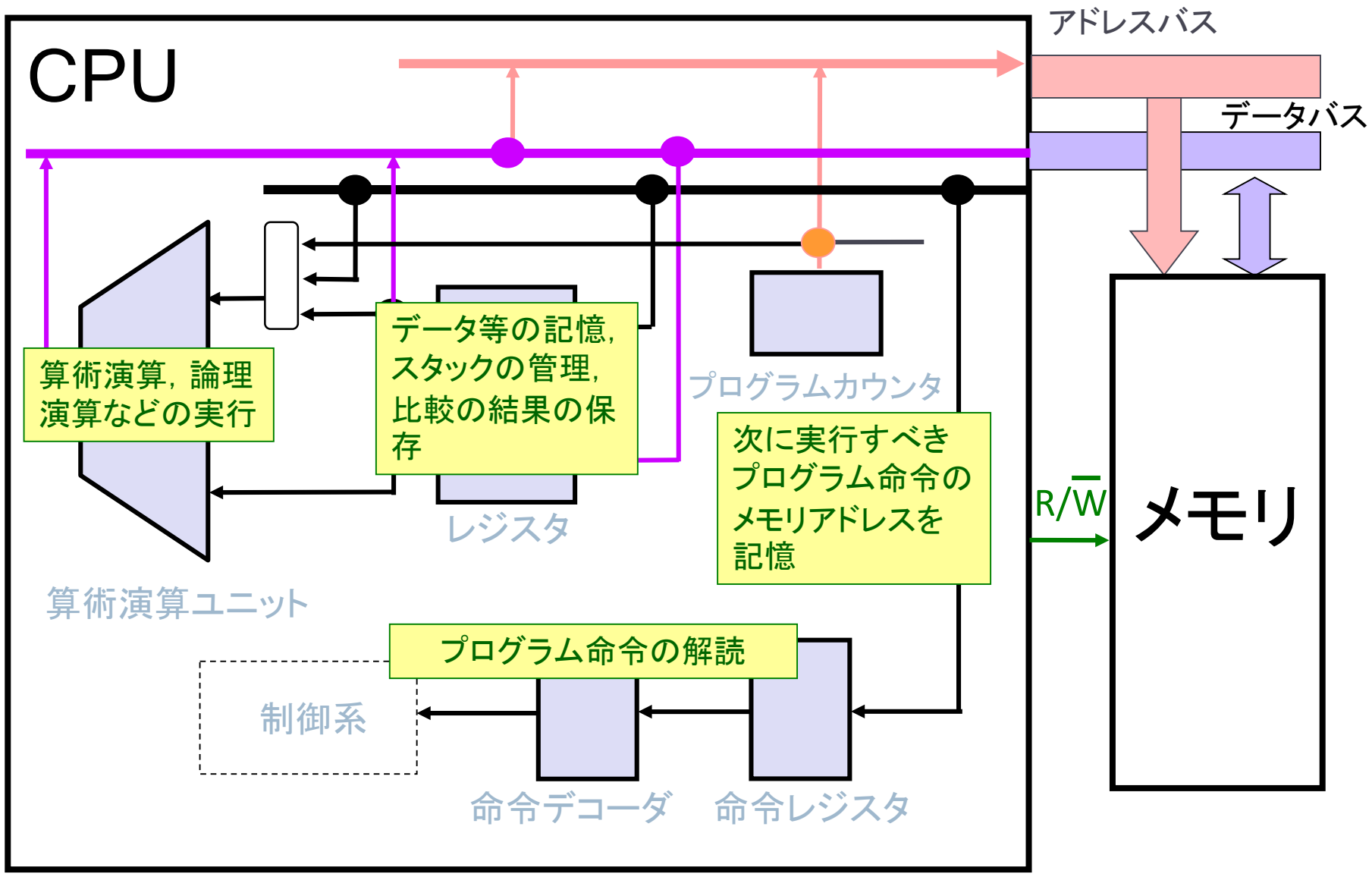
- 到達目標
- 計算の仕組みの理解
- 68000系CPUのレジスタ構成
- オペランドのサイズ
- アドレッシングモード
- 命令セット概要
- 2数の和の計算

到達目標

ソフト実験１，２の準備として，アセンブラ（アセンブリ言語）でプログラムを書けるようになることを目標とする．具体的には，

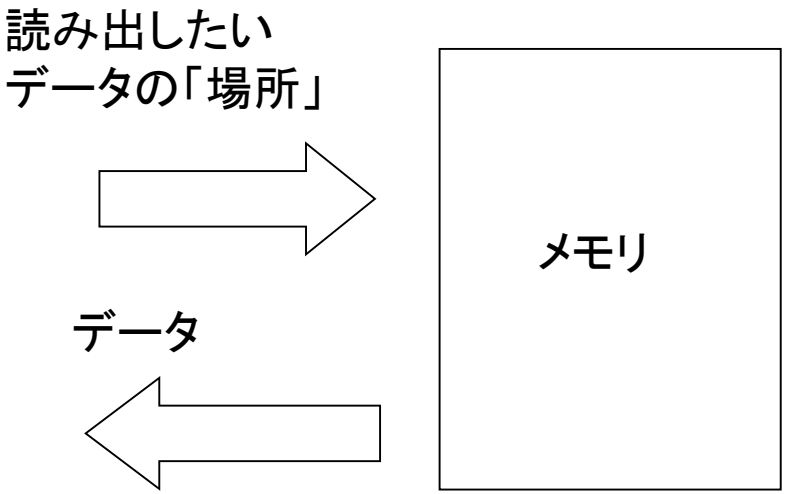
- 計算の仕組みの理解
- 68000系プロセッサのアドレッシング，命令セットの理解
- 条件分岐，繰り返しの方法の理解
- サブルーチン呼び出しの理解

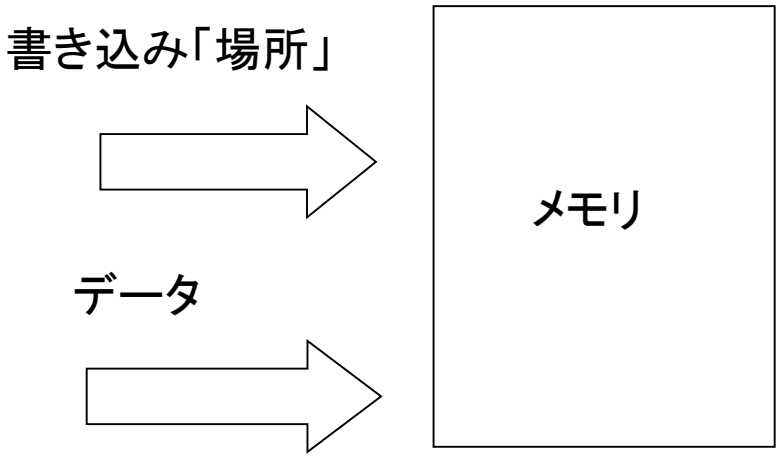
計算の仕組みの概要


CPU
アドレスバス
データバス
算術演算，論理演算などの実行
データ等の記憶，スタックの管理，比較の結果の保存
プログラムカウンタ
次に実行すべきプログラム命令のメモリアドレスを記憶
R/W
メモリ
レジスタ
算術演算ユニット
プログラム命令の解読
制御系
命令デコーダ
命令レジスタ

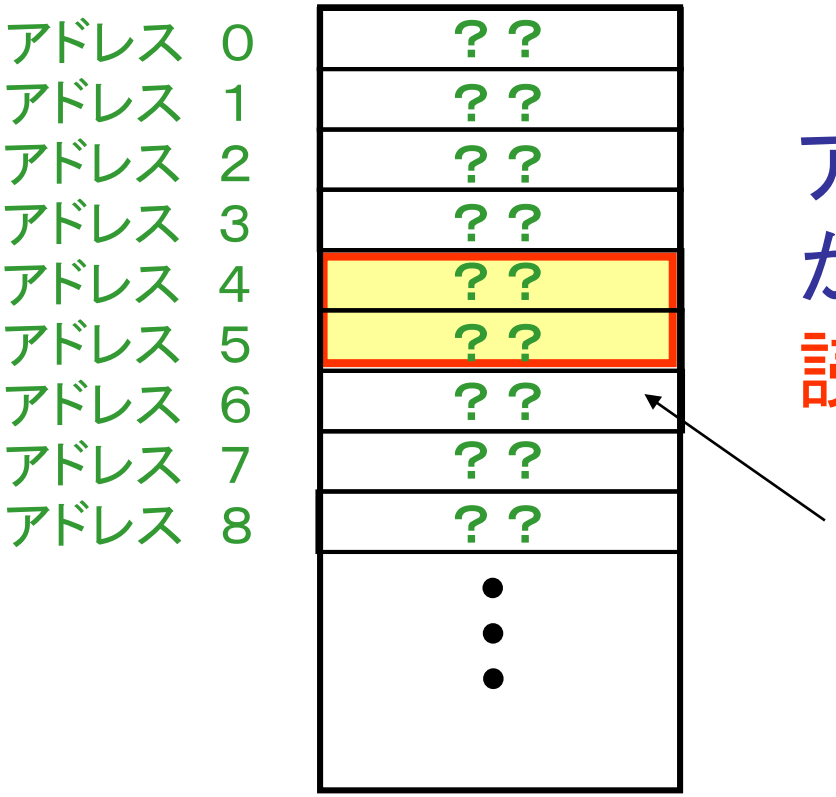
メモリ

- 読み出し
- 書き込み





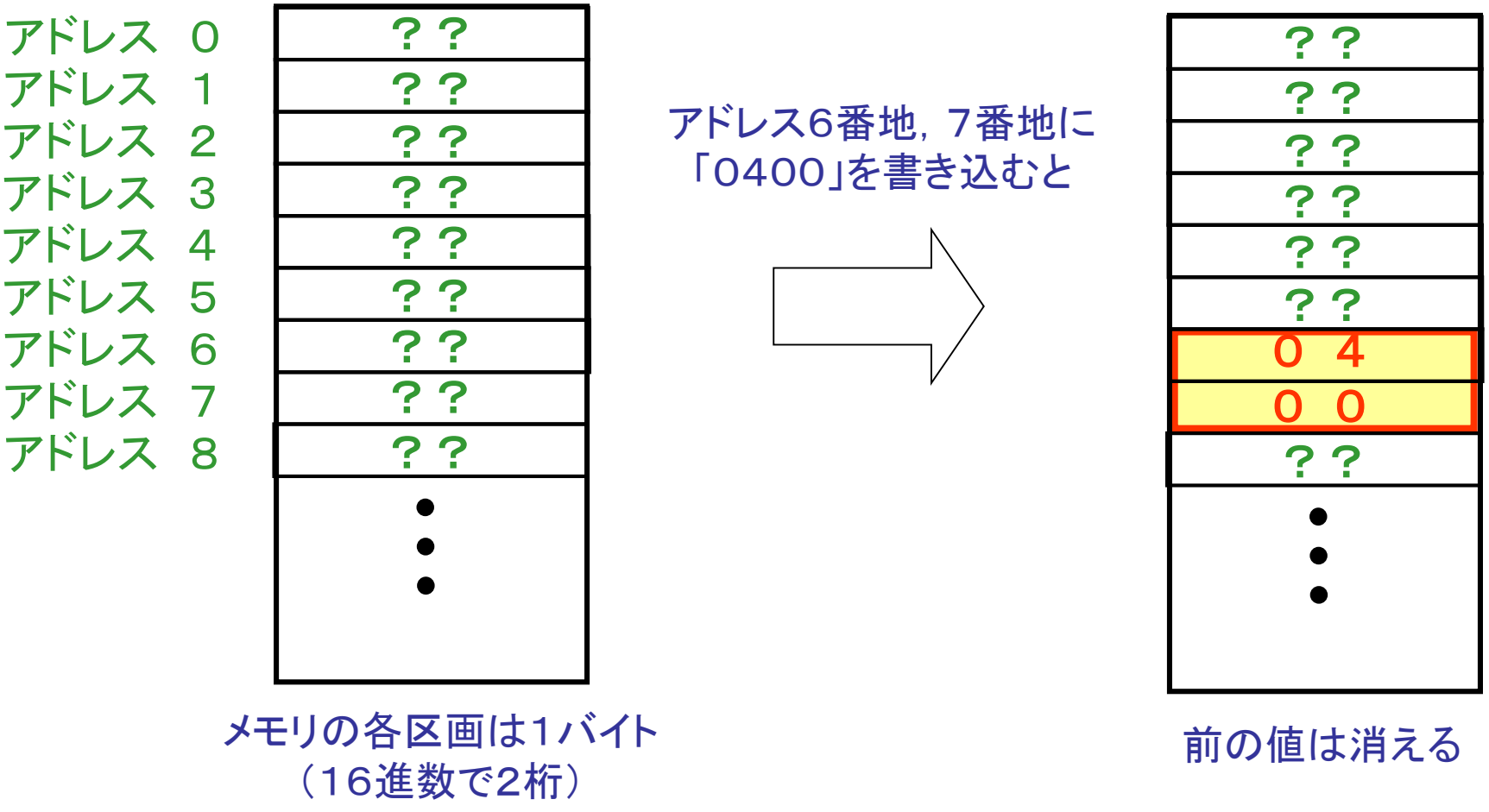
メモリからの読み出し



アドレス4番地，5番地から1ワード（2バイト）分読み出すとき

メモリの値は変化しない

メモリへの書き込み


アドレス 0
アドレス 1
アドレス 2
アドレス 3
アドレス 4
アドレス 5
アドレス 6
アドレス 7
アドレス 8
？？
？？
？？
？？
？？
？？
？？
？？
？？
アドレス6番地，7番地に
「0400」を書き込むと
？？
？？
？？
？？
？？
？？
0 4
0 0
？？
メモリの各区画は1バイト
（16進数で2桁）
前の値は消える

abs ファイル（1/2）

ソースファイル
テキストエディタなどで作成

```
/* sample     wa.s */

.section .data
x:      .dc.w   10
y:      .dc.w   20
z:      .ds.w   1

.section .text
        move.w  x, %d0
        add.w   y, %d0
        move.w  %d0, z

        stop #0x2700
.end
```

足し算のプログラム

アセンブル（m68k-as）

メモリのどこに何を置くかを書いたファイル

```
S00600004844521B
S2140004003039000004 18D0790000041A33C0000008
S20A000410041C4E722700DA
S20A000418000A00140000BB
S5030003F9
S804000400F7
```

abs ファイル（自動生成）

メモリにロード

```
000400: 30 39 00 00 04 18 D0 79 00 00 04 1A 33 C0 00 00
000410: 04 1C 4E 72 27 00 00 00 00 0A 00 14 00 00 00 00
000420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

メモリの中身

absファイル(2/2)

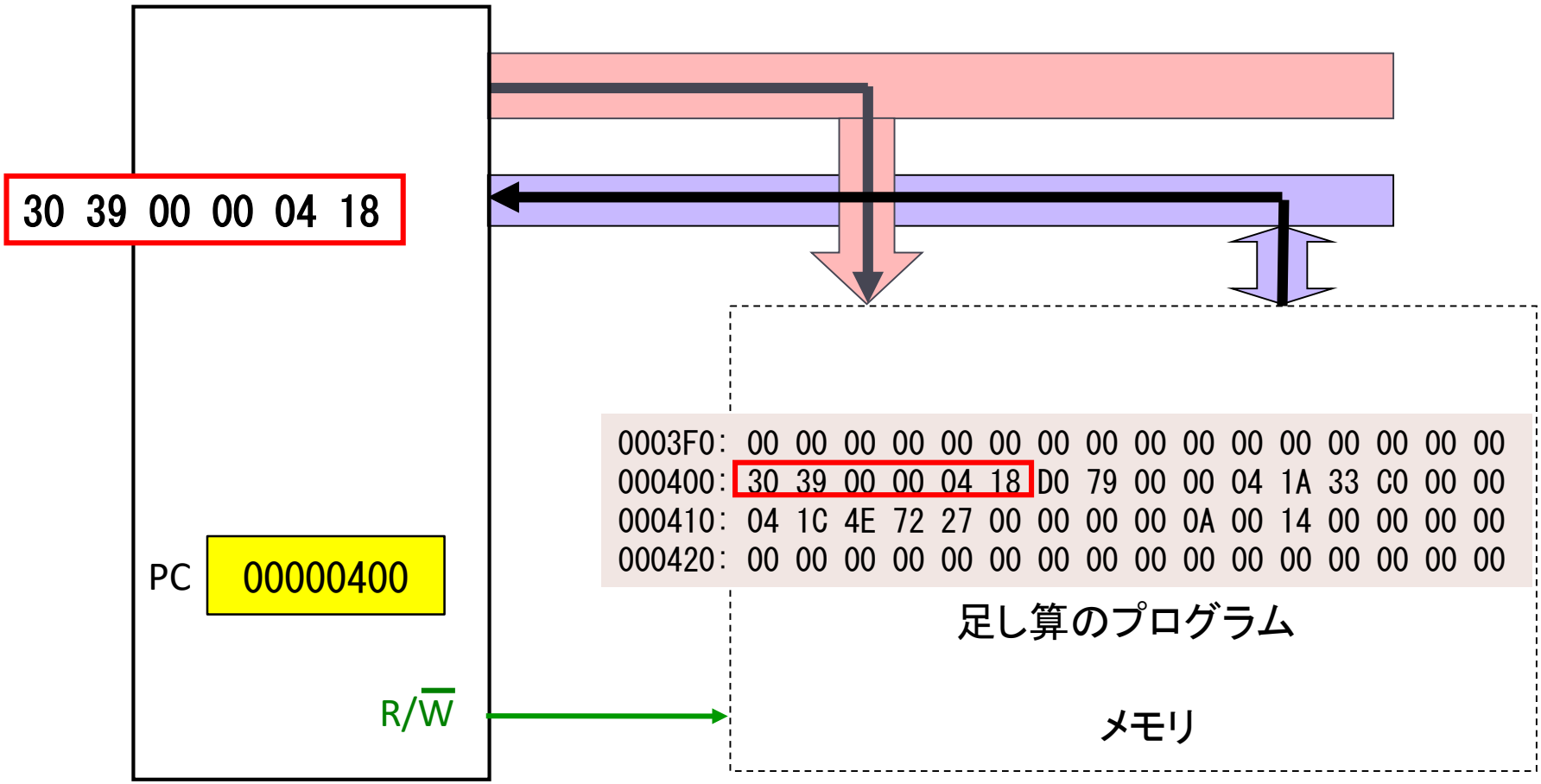
メモリにロード

```
0003F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000400: 30 39 00 00 04 18 D0 79 00 00 04 1A 33 C0 00 00
000410: 04 1C 4E 72 27 00 00 00 00 0A 00 14 00 00 00 00
000420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

メモリアドレス　　メモリの中身

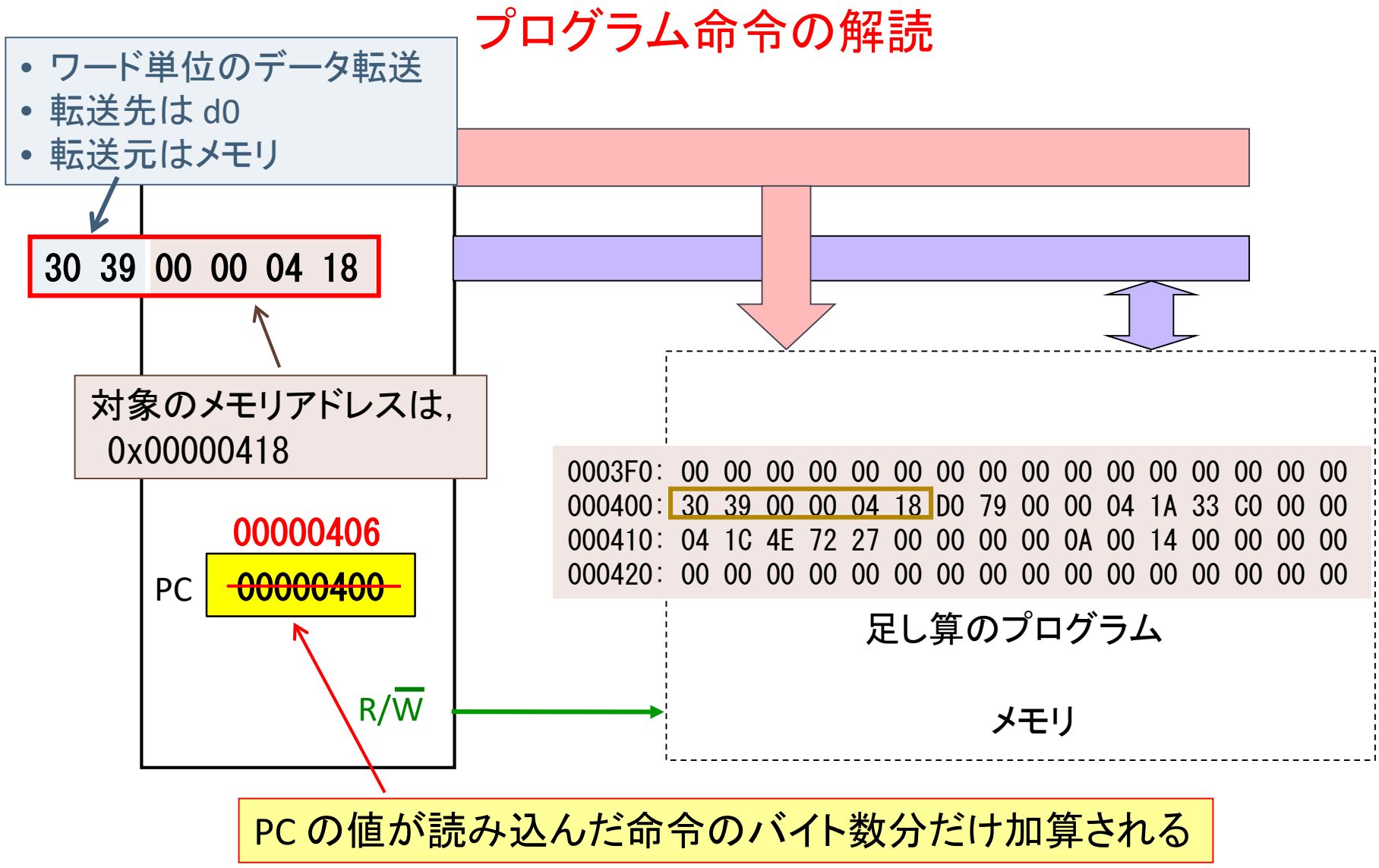
プロセッサの動作(命令フェッチ)

メモリ中のプログラム命令を CPU に読み込む



注意: この授業では, 説明を簡単にするために,
1命令分が1度に読み込まれるとして説明としている.

プロセッサの動作(命令デコード)

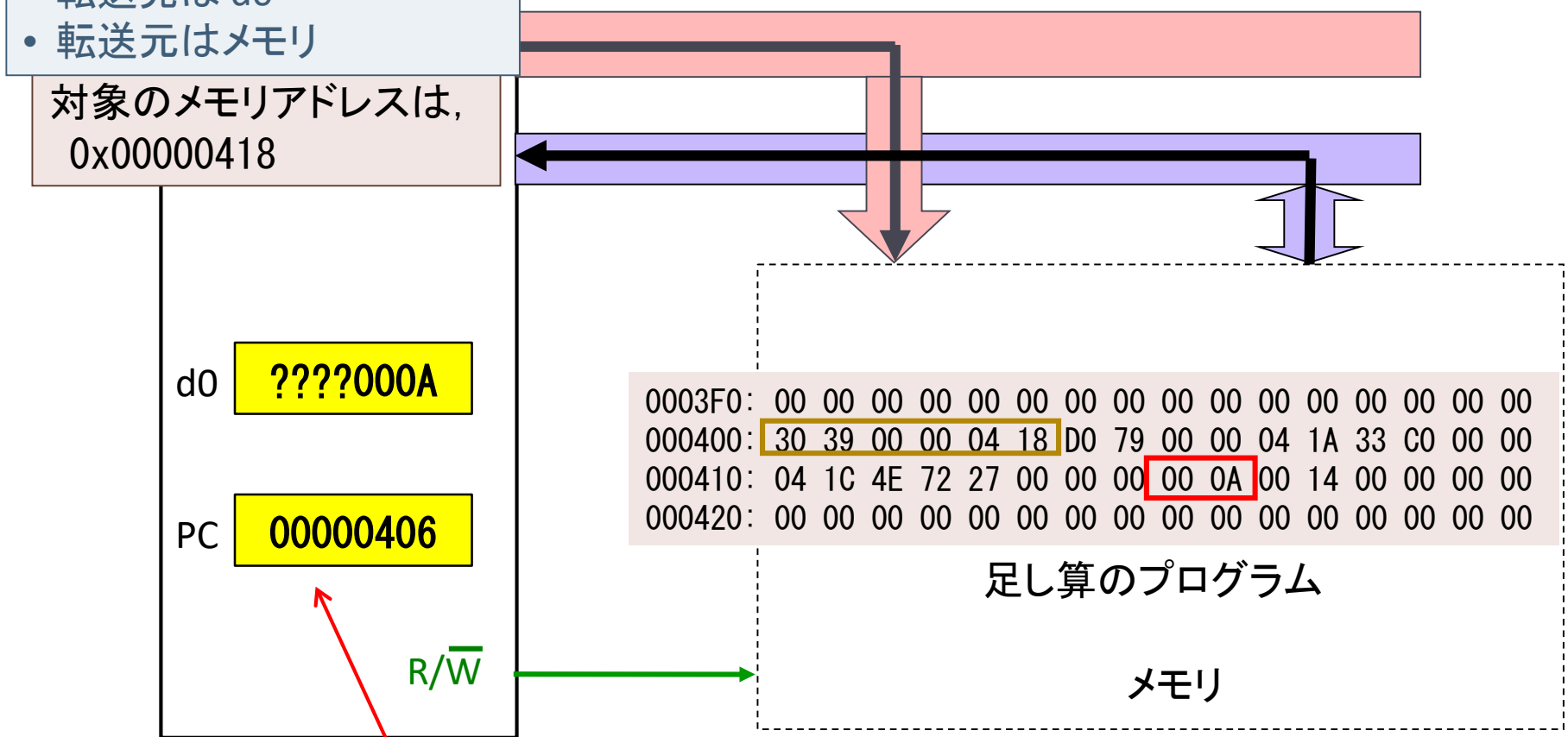


プロセッサの動作(命令実行)

プログラム命令の実行

- ワード単位のデータ転送
- 転送先は d0
- 転送元はメモリ

対象のメモリアドレスは,
0x00000418



次に実行すべき命令が格納されている番地を示す.
命令実行により PC の値が変わる命令はジャンプ命令.

68000系プロセッサのレジスタ構成

- **データレジスタ** D0～D7　　32bit

  メモリから取り出されたデータの加工や，メモリへ記憶されるべきデータの加工をするために使用されるレジスタ.

- **アドレスレジスタ** A0～A6　　32bit

  アドレスを保持するレジスタ.

- **スタックポインタ** A7　　32bit

  スーパバイザモードでは，A7 はスーパバイザスタック（SSP），**ユーザモード**では，ユーザスタックポインタ（USP）にアクセスすることになる.

  これらの利用に関しては，「関数呼び出し」「特権命令と割り込み処理」の回で解説.

- **プログラムカウンタ** PC　32bit（実際に使われているのはビット 0～23）

  次に実行すべき命令が書いてあるメモリ番地を記憶.

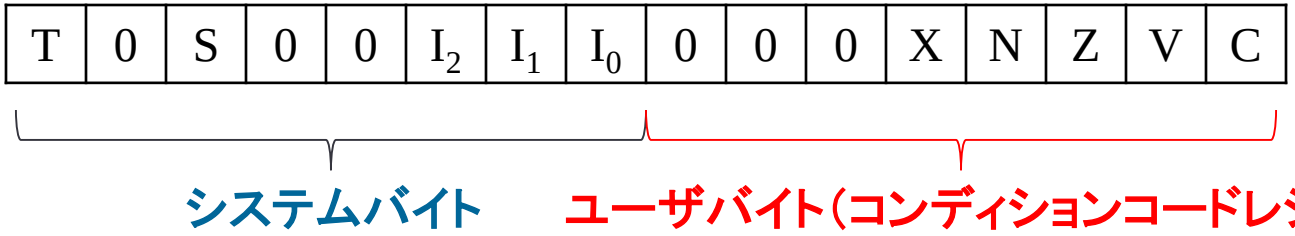
- **ステータスレジスタ** SR　　16bit

  プロセッサ自身に関する特別な制御のために利用.

68000系プロセッサのレジスタ構成（SR）

ステータスレジスタ

※ 0 に設定されているビットは未使用



T : 1 にセットされていると，命令実行ごとにスーパーバイザモードへ移行し，トレース処理を行う（プログラムのデバッグに使用）．

S : スーパーバイザモードなのか（S=1），ユーザモードなのか（S=0）を保持．

I : I_2 I_1 I_0 の3ビット（3桁の2進数と見よ）で走行レベルを指定．割り込みのマスクに利用．

X : 多倍長演算に使用されるフラグ．本実験では使用しない．

N : 演算結果が負（Negative）になると，セットされる（ 1 になる）．

Z : 演算結果がゼロ（Zero）になると，セットされる．

V : 演算結果で，オーバーフローが発生すると，セットされる．

C : 演算結果で，桁上がり（Carry），あるいは，減算で桁下がりが発生すると，セットされる．

オペランドのサイズ

実行する命令によって操作される対象（オペランド）のサイズ

バイト（B）:	8ビット
ワード（W）:	16ビット
ロングワード（L）:	32ビット

オペランドのサイズが決まっている命令と，サイズを指定するものとがある.

アドレスレジスタとオペランドサイズ

アドレスレジスタに対する操作は，W または L のみが許される

サイズ W での使用：

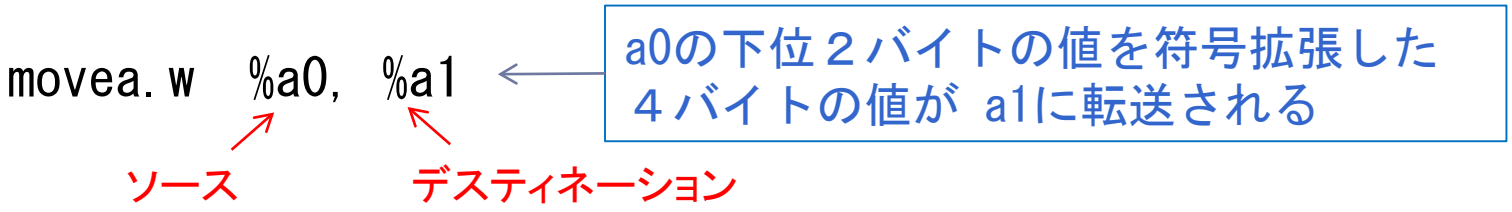
- ソースオペランドにアドレスレジスタが指定された場合は，符号拡張した32ビットの値としてアドレスの演算が行われる．

  ex. 0x7FFF ⇒ 0x00007FFF （ビット15が 0 ゆえ）

  0x8000 ⇒ 0xFFFF8000 （ビット15が 1 ゆえ）

  （ 0x は16進数を表す．アセンブラによっては $ のもある）

- デスティネーションオペランドにアドレスレジスタが指定された場合，符号拡張された32ビットの値が転送される．



サイズ L での使用：32ビットのすべてが操作対象

インデックスレジスタとしての使用：

サイズ W の場合符号拡張した32ビットの数値として扱われる．

データレジスタとオペランドサイズ

データレジスタに対する操作は，すべてのデータサイズが許される．

B では，下位8ビット，W では下位16ビット，L では32ビット全体が対象．
B, W ではレジスタの上位部分は無視され，**デスティネーションオペランド**（後述）として用いられた場合でも変化しない．

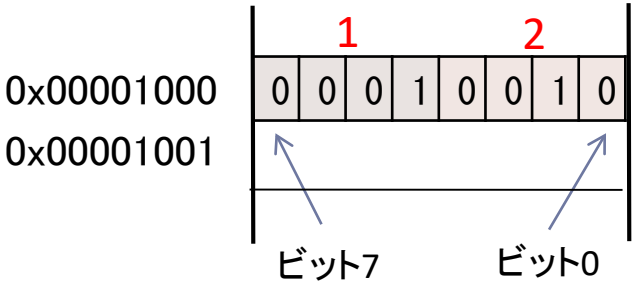
インデックスレジスタとしても使用することができ，アドレスレジスタと同様に，データサイズが W の場合は符号拡張され32ビットの値として扱われる．

メモリ内のデータ構成（下位はどっち？）

サイズ B の場合

偶数／奇数番地いずれにもアクセスできる.

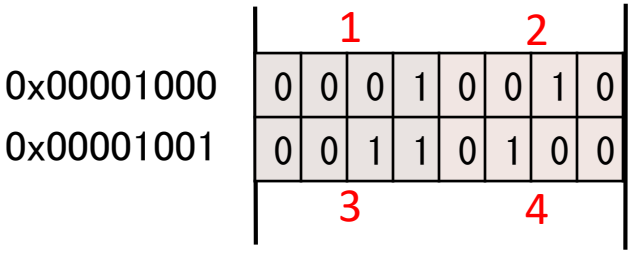
ex. 0x12（1 バイト）を0x1000番地に書き込む（転送する）命令を実行.



サイズ W の場合

上位バイトが偶数番地，下位バイトが奇数番地.
偶数番地からしかアクセスできない.

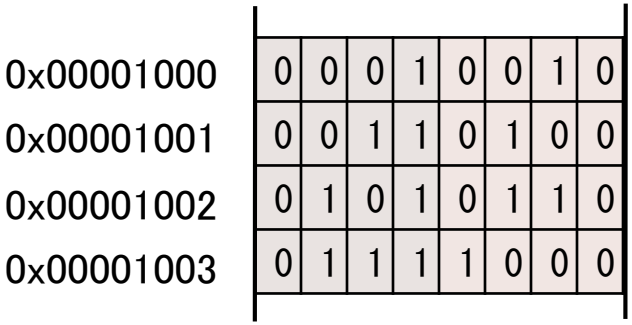
ex. 0x1234（1 ワード）を0x1000番地に書き込む（転送する）命令を実行.



サイズ L の場合

偶数番地からしかアクセスできない.

ex. 0x12345678（1 ロングワード）を0x1000番地に書き込む（転送する）命令を実行.



アドレッシングモード（1/7）

イミディエイトデータ，クイックイミディエイト以外は，対象の場所の示し方．
転送命令であれば『その場所の内容』への操作．
ジャンプ命令であれば『その場所』へのジャンプ．

- イミディエイトデータ
- 絶対アドレス
- データレジスタ直接
- アドレスレジスタ直接
- アドレスレジスタ間接
- ポストインクリメントアドレスレジスタ間接
- プリデクリメントアドレスレジスタ間接
- インデックス付きアドレスレジスタ間接
- ディスプレースメント付きアドレスレジスタ間接
- ディスプレースメント付きプログラムカウンタ相対
- インデックス付きプログラムカウンタ相対
- CCR / SR
- クイックイミディエイト

※ 注意：命令，ソースオペランド，デスティネーションオペランド毎にサポートしているモードの種類は異なる．

各自調べよ

アドレッシングモード(2/7)

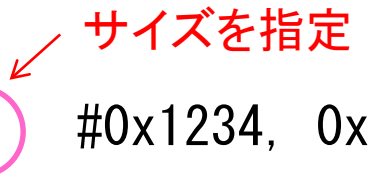
イミディエイトデータ 〔書式: #数値〕

対象となるデータを直接指定するモードで, # を先頭に付ける.

ex. #0x1234 ← 0x1234 という数値

絶対アドレス 〔書式: 数値〕

メモリ番地で場所を指定するモード.
2バイトで指定した場合は, 符号拡張された32ビットの値として使用される.



ex. move.w #0x1234, 0x00001000 ; 0x1000番地に0x1234を転送

アドレッシングモード(3/7)

データレジスタ直接 〔書式: %dn　n=0～7〕

場所として，データレジスタを指定するモード.

ex. move.b　%d1, 0x1000

0x1000は２バイトデータである. 符号拡張して32ビットの値0x00001000とし，この番地にd1レジスタの内容の下位１バイトを転送.

アドレッシングモード(4/7)

アドレスレジスタ直接 〔書式: %an　n=0～7〕

場所として，アドレスレジスタを指定するモード.

バイトサイズはサポートされない.

デスティネーションオペランドに指定すると，アドレスレジスタの32ビット全体が対象となる. したがって，サイズWの値であっても，符号拡張された32ビットの値がアドレスレジスタに格納される.

```
ex.  movea.w    #0x1234,  %a1          ;   a1 = 0x00001234 となる
     movea.w    #0x8000,  %a1          ;   a1 = 0xFFFF8000 となる

     a1 = 0x12345678 とすると,
     move.w     %a1,  0x1000           ;   0x1000番地の値は0x56
                                       ;   0x1001番地の値は0x78

     move.l     %a1,  0x1000           ;   0x1000番地の値は0x12
                                       ;   0x1001番地の値は0x34
                                       ;   0x1002番地の値は0x56
                                       ;   0x1003番地の値は0x78
```

アドレッシングモード(5/7)

アドレスレジスタ間接 〔書式: (%an)　n=0～7〕

場所として, アドレスレジスタ an の中身が示すメモリ番地を指定するモード.

ex. 0x2000番地の値は 0x12, 0x2001番地の値は 0x34 とする.

```
a1 = 0x00002000 とすると,
move.w   (%a1), 0x1000          ;  0x1000番地の値は0x12
                                ;  0x1001番地の値は0x34
a1 = 0x00001000 とすると,
move.b   0x2000, (%a1)          ;  0x1000番地の値は0x12

a1 = 0x00001000 とすると,
jmp (%a1)                       ;  0x1000番地にジャンプ
```

アドレッシングモード（6/7）

ポストインクリメントアドレスレジスタ間接　〔書式：(%an)+　n=0～7〕

場所として，アドレスレジスタ an の中身が示すメモリ番地を指定するモード．
ただし，**命令実行後，アドレスレジスタの値は扱ったデータサイズに応じて加算される（Bなら1，Wなら2，Lなら4）．**

ex. 0x2000番地から２バイトの値は 0x1234 とする．

a1 = 0x00002000 とすると，

```
move.w    (%a1)+, 0x1000        ;  0x1000番地 ← 0x1234
                                ;  a1の値は0x00002002
```

プリデクリメントアドレスレジスタ間接　〔書式：-(%an)　n=0～7〕

命令実行前に，アドレスレジスタ an は扱うべきデータサイズに応じて減算され，場所として，減算後の an の中身が示すメモリ番地を指定するモード．

ex. 0x2000番地から２バイトの値は 0x1234 とする．

a1 = 0x00002002 とすると，

```
move.w    -(%a1), 0x1000        ; 0x1000番地 ← 0x1234
                                ; a1の値は0x00002000
```

アドレッシングモード（7/7）

インデックス付きアドレスレジスタ間接 〔書式：d8(%an,%IX)　n=0～7〕

IX（インデックスレジスタ）は，a0～a7, d0～d7 のいずれか．
IX のサイズはIXの後に .w または .l を付けて指定（省略するとl）．
d8 は1バイトの数値（-128 ～ +127）．
場所として，[an の中身の値] + [IX の中身の値] + d8 なるメモリ番地を指定するモード．

※ アドレス計算はすべて符号拡張された 32 ビットの値として計算

ex.　a1 = 0x00020000,　d1 = 0x00008000 とする．

```
move.b    0xFF(%a1,%d1.w),  %d0

          0x00017FFF番地の値 → d0
```

```
a1 → 00020000
d1 → FFFF8000
d8 → FFFFFFFF
-------------
     00017FFF
```

```
move.b    0xFF(%a1,%d1.l),  %d0

          0x00027FFF番地の値 → d0
```

```
a1 → 00020000
d1 → 00008000
d8 → FFFFFFFF
-------------
     00027FFF
```

プログラムの書き方と LIS ファイル

wa.LIS

```
                        2 .list
                        3 /* sample program   wa.s */
                        4
                        5 .section .data
000418 000A             6 x:      .dc.w    10
00041a 0014             7 y:      .dc.w    20
00041c 0000             8 z:      .ds.w    1
                        9
                       10 .section .text
000400 3039 0000        11        move.w   x, %d0
       0418             11
000406 D079 0000        12        add.w    y, %d0
       041a             12
00040c 33C0 0000        13        move.w   %d0, z
       041c             13
                       14
000412 4E72 2700        15        stop #0x2700
                       16 .end
```

コメント．/* と */ で挟む

空行はプログラムに影響なし

ラベル．ラベル名は x．
末尾に：を付ける

ワードサイズで10を配置

ワードサイズで領域を1つ確保

ラベルは，対応するアドレスとして解釈される

stop 命令．
CPU の動作を停止する．

アセンブラ疑似命令(1/2)

.dc{.b .w .l }　　データ列

データをメモリ内に配置. 一つのデータのサイズが .b などで指定される.

ex. values:　.dc.w　1, 0xffff

ラベル values がたとえば, 0x1000 番地に関連付けられるとすると, 0x1000, 0x1001番地が 0x0001 で, 0x1002, 0x1003番地が 0xffff.

Message: .dc.b　'a', 'b', 0

ラベル Message がたとえば, 0x2000 番地に関連付けられるとすると, 0x2000番地が 0x61, 0x2001番地が 0x62, 0x2002番地が 0.

.ds{.b　.w　.l}　　式

指定したサイズの領域を連続「式」の値個確保する.

.equ　シンボル名　式

シンボルを「式」の値に設定する. シンボルが定数のように扱える.

(.org　式　←ロケーションアドレスの設定. 本実験では使わない方が良い.)

アセンブラ疑似命令(1/2)

.end

ソースプログラムの終わりを示す．省略するとファイルの末が終わり．

（複数のライブラリモジュールをリンクする際は，この疑似命令が含まれているモジュールがメインプログラムとみなされる．）

.section　タイプ

プログラムをデータ領域とコード領域などに分けて記述するために使用．タイプは

.text　：コード領域　　.data　：初期値のあるデータ領域

.bss　：初期値のないデータ領域

であり，次の .section 命令までは，そのタイプが指定される．

基礎ソフト実験，ソフト実験１～３で用いるコマンド m68k-as では，デフォルトではリンカを用いて，

text 領域を 0 番地から，次に data 領域，次に bss 領域

と配置されるようなっている．text 領域を 0x400 番地からにするには，

m68k-as -t 400　ソースファイル名

とする．

プログラム作成上の注意

- 68000は，reset後，まず，メモリの0～3番地の値をスーパバイザスタックポインタに転送し，4～7番地の値をプログラムカウンタに設定する．つまり，0～3番地の内容がスーパバイザスタックの底の番地，4～7番地の内容がプログラムの開始番地である．
  ただし，ソフト実験１以降の実機を使った実験では，reset 後は，ROM に書かれている IPL が起動するようになっており，指定したプログラム（abs ファイル）を読み込み，0x400番地からプログラムを実行するようになっている．
- 命令コードを奇数番地から配置すると，その命令を読み込んだ時にアドレスエラーが発生する．偶数番地から配置されるようにすること．
  .ds などで奇数バイトの領域を確保したりすると，起こる可能性がある（アセンブラによっては自動的に偶数番地から配置するものもある）．
- 使用するアセンブラによって，ソースラインの形式や疑似命令は若干異なる．