アセンブラ入門

Outline

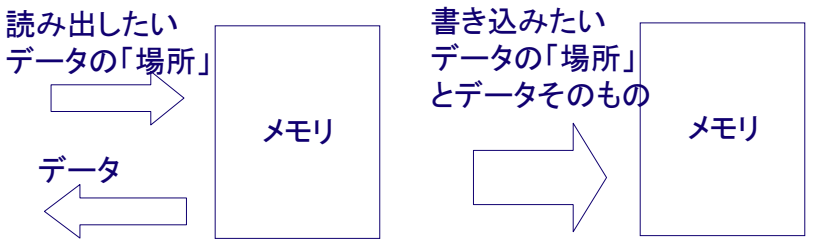
1. メモリとは
2. 条件分岐のプログラム
3. 繰り返し処理のプログラム

メモリとは

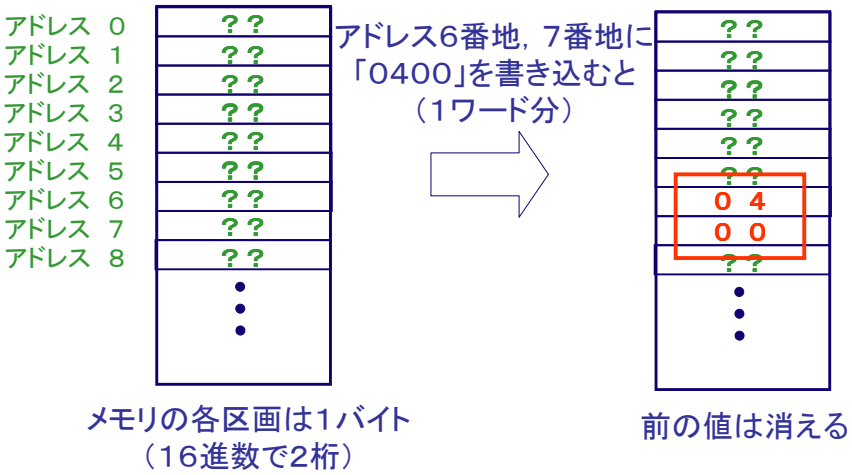
- デジタルデータの記憶を行うLSIチップ
- デジタルデータを覚えさせたり，取り出したりの機能がある

メモリ

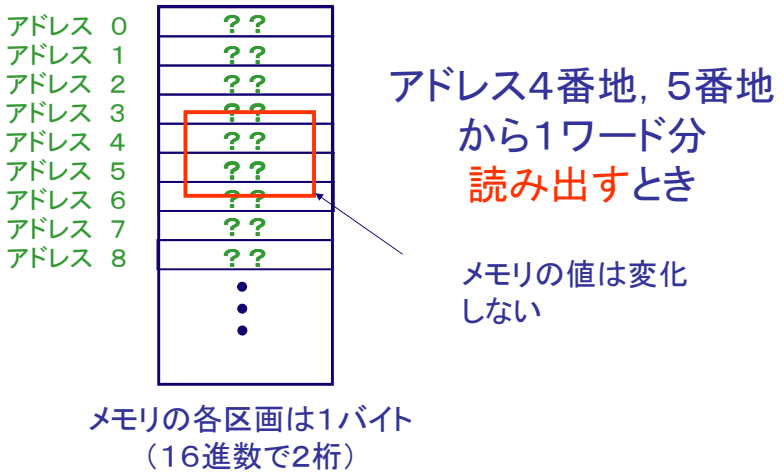
- 読み出し
- 書き込み



メモリへの書き込み



メモリからの読み出し

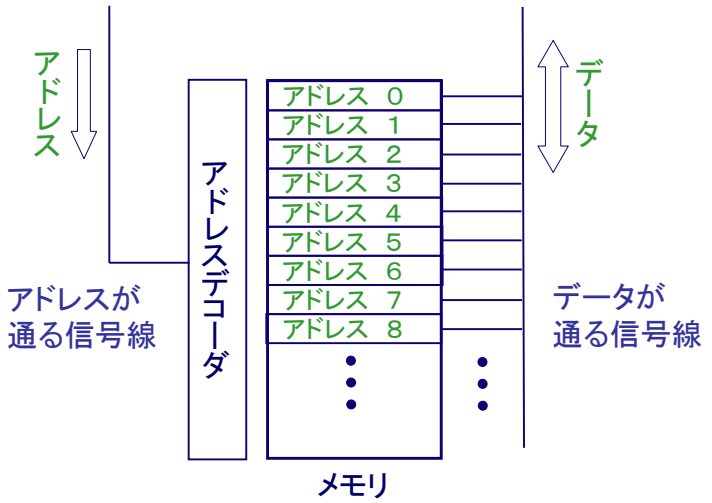


バイト，ワード，ロングワード

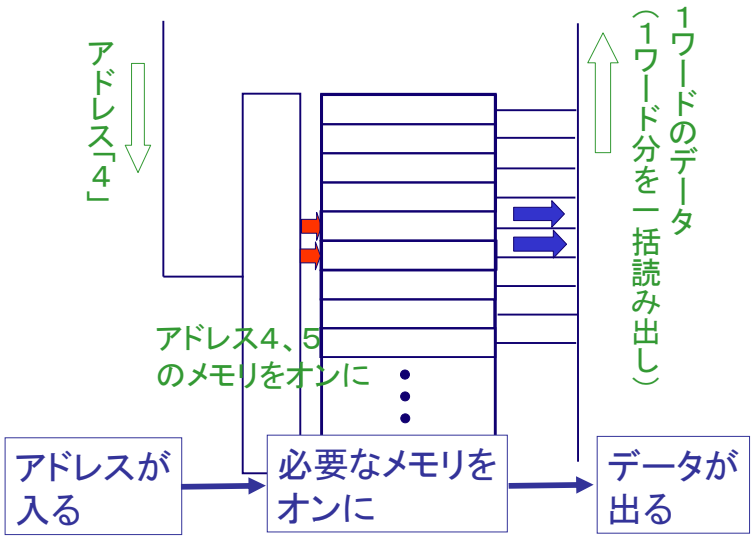
- バイト：　16進数で2桁
  0x00 ～ 0xff
- ワード：　16進数で4桁（＝2バイト）
  0x0000 ～ 0xffff
- ロングワード：　16進数で8桁（＝4バイト）
  0x00000000 ～ 0xffffffff

この授業では、16進数を多用する．
16進数には、適宜頭に「0x」を付ける

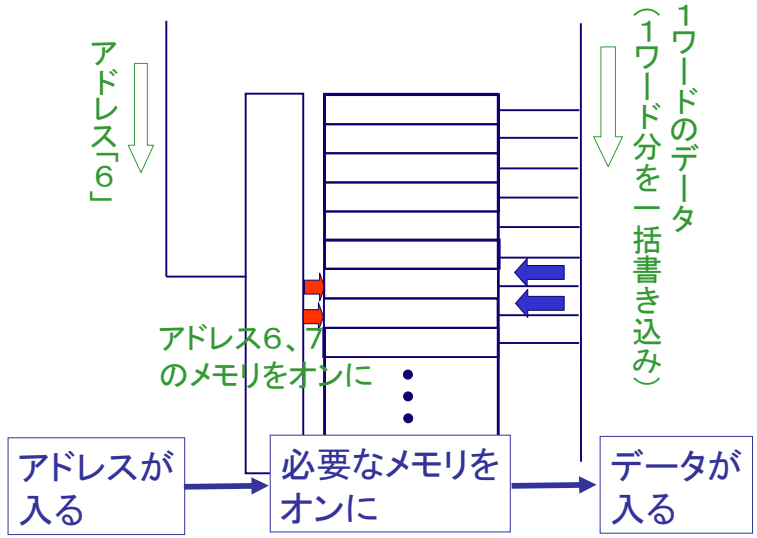
メモリの仕組み



アドレス4番地から，1ワード分読み出す



アドレス6番地に，1ワード分書き込む

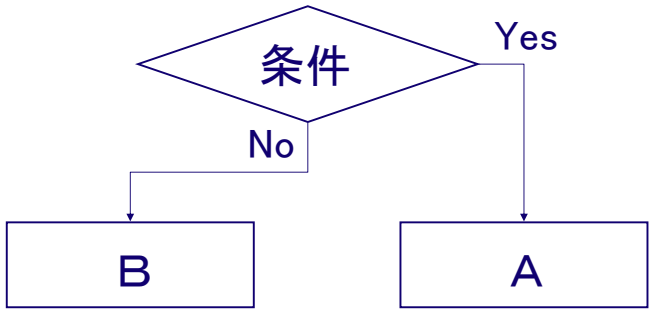


例題1．x ≦ 5 での分岐

- 条件分岐の例として，次の例を考える

$$y = 8 \times x \quad (x > 5 \text{のとき})$$
$$y = 0 \quad (x \leq 5 \text{のとき})$$

条件分岐とは



- 「ある条件」が成り立てばAを、成り立たなければBを実行

x ≦ 5 での分岐

実行結果の例

ここでは，x, y はともに4バイトのデータ

```
000000: 70 05 b0 b9 00 00 00 2c 64 00 00 14 20 39 00 00
000010: 00 2c e7 88 23 c0 00 00 00 30 60 00 00 08 42 b9
000020: 00 00 00 30 48 48 4e 72 00 00 00 00 00 00 00 07
000030: 00 00 00 38 00 00 00 00 00 00 00 00 00 00 00 00
000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

x

y

$$y = 8 \times x \quad (x > 5 \text{のとき})$$

$$y = 0 \quad (x \leq 5 \text{のとき})$$

見方

メモリの中身を表示： 16進表記，1バイト単位

アドレス0x000000 から
0x00000f までの中身は・・・

この通り

```
000000: 70 05 b0 b9 00 00 00 2c 64 00 00 14 20 39 00 00
000010: 00 2c e7 88 23 c0 00 00 00 30 60 00 00 08 42 b9
000020: 00 00 00 30 48 48 4e 72 00 00 00 00 00 00 00 07
000030: 00 00 00 38 00 00 00 00 00 00 00 00 00 00 00 00
000040: 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

プログラム本体そのものが
入っているエリア

ここでは，x, y はともに4バイトのデータ

```
000000: 70 05 b0 b9 00 00 00 2c 64 00 00 14 20 39 00 00
000010: 00 2c e7 88 23 c0 00 00 00 30 60 00 00 08 42 b9
000020: 00 00 00 30 48 48 4e 72 00 00 00 00 00 00 00 07
000030: 00 00 00 38 00 00 00 00 00 00 00 00 00 00 00 00
000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

x

y

未使用

データが入っているエリア

x > 5 での分岐

C言語

```
static long int x = 7;
static long int y = 0;

int main()
{
  if ( x > 5 ) {
    y = 8 * x;
  }
  else {
    y = 0;
  }

  return 0;
}
```

等価

等価

68000アセンブラ言語

```
.data
x:
        dc.l 7
y:
        dc.l 0
.text
        moveq.l #5,%d0
        cmp.l x,%d0
        bcc else1
        move.l x,%d0
        lsl.l #3,%d0
        move.l %d0,y
        bra endif1
else1:
        clr.l y
endif1:

        .dc.w 0x4848
        stop #0
.end
```

関数の定義は，
今後の授業で触れる（今回は触れない）

68000アセンブラ言語

```
.data
x:
        .dc.l 7
y:
        .dc.l 0
.text
        moveq.l #5,%d0
        cmp.l x,%d0
        bcc else1
        move.l x,%d0
        lsl.l #3,%d0
        move.l %d0,y
        bra endif1
else1:
        clr.l y
endif1:

        .dc.w 0x4848
        stop #0
.end
```

データエリアの確保

x, y（ともに4バイトデータ）
のためのデータエリアを確保せよ

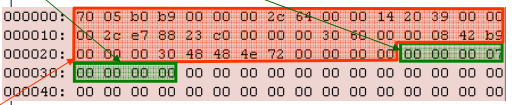
プログラム本体

68000アセンブラ言語

```
.data
x:
        .dc.l 7
y:
        .dc.l 0
.text
        moveq.l #5,%d0
        cmp.l x,%d0
        bcc else1
        move.l x,%d0
        lsl.l #3,%d0
        move.l %d0,y
        bra endif1
else1:
        clr.l y
endif1:

        .dc.w 0x4848
        stop #0
.end
```

最初の時点
（プログラム全体をメモリ上にロードした時点であり，
プログラムを実際に実行する前）



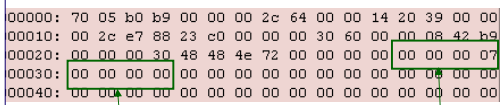
メモリの中身

```
.data
x:
        .dc.l 7
y:
        .dc.l 0
.text
        moveq.l #5,%d0
        cmp.l x,%d0
        bcc else1
        move.l x,%d0
        lsl.l #3,%d0
        move.l %d0,y
        bra endif1
else1:
        clr.l y
endif1:

        .dc.w 0x4848
        stop #0
.end
```

「**4バイト**をデータエリア内に確保．
最初は「**0x0000 0007**」にしておく．
x というラベルを付ける」という指示

「**4バイト**をデータエリア内に確保．
最初は「**0x0000 0000**」にしておく．
y というラベルを付ける」という指示



y　　x

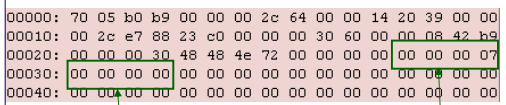
プログラム全体をメモリ上に
ロードした時点で，x, y の値がセットされる

```
.data
x:
        .dc.l 7
y:
        .dc.l 0
.text
        moveq.l #5,%d0
        cmp.l x,%d0
        bcc else1
        move.l x,%d0
        lsl.l #3,%d0
        move.l %d0,y
        bra endif1
else1:
        clr.l y
endif1:

        .dc.w 0x4848
        stop #0
.end
```

「**4バイト**をデータエリア内に確保．
最初は「**0x0000 0007**」にしておく．
x というラベルを付ける」という指示

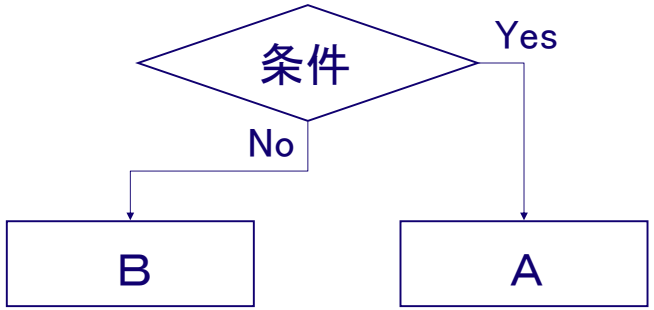
「**4バイト**をデータエリア内に確保．
最初は「**0x0000 0000**」にしておく．
y というラベルを付ける」という指示



y　　x

プログラム全体をメモリ上に
ロードした時点で，x, y の値がセットされる

条件分岐とは



- 「ある条件」が成り立てばAを、成り立たなければBを実行

```
.data
x:
        .dc.l 7
y:
        .dc.l 0
.text
        moveq.l #5,%d0
        cmp.l x,%d0
        bcc else1
        move.l x,%d0
        lsl.l #3,%d0
        move.l %d0,y
        bra endif1
else1:
        clr.l y
endif1:

        .dc.w 0x4848
        stop #0
.end
```

x, y, をメモリエリア中に確保

ロングワード
1ロングワードは4バイト

プログラム中で使用

.l ロングワード（4バイト）
.w ワード（2バイト）
.b バイト（1バイト）

```
.data
x:
        .dc.l 7
y:
        .dc.l 0
.text
        moveq.l #5,%d0
        cmp.l x,%d0
        bcc else1
        move.l x,%d0
        lsl.l #3,%d0
        move.l %d0,y
        bra endif1
else1:
        clr.l y
endif1:

        .dc.w 0x4848
        stop #0
.end
```

x と 5 の比較（D0 を使用）

比較結果による分岐

x > 5 のとき実行される部分

そうでないときに実行される部分

```
.data
x:
        .dc.l 7
y:
        .dc.l 0
.text
        moveq.l #5,%d0
        cmp.l x,%d0
        bcc else1
        move.l x,%d0
        lsl.l #3,%d0
        move.l %d0,y
        bra endif1
else1:
        clr.l y
endif1:

        .dc.w 0x4848
        stop #0
.end
```

もし x＞5 ならば

実行順

① moveq.l #5,%d0
② cmp.l x,%d0
③ bcc else1　分岐しない
④ move.l x,%d0
⑤ lsl.l #3,%d0　x > 5 のとき実行される部分
⑥ move.l %d0,y
⑦ bra endif1　ラベル endif1 へ分岐せよという指示

ジャンプ

else1:
        clr.l y
endif1:　スキップ

⑧ .dc.w 0x4848　以後省略
        stop #0
.end

もし x≦5 ならば

実行順

```
.data
x:
        .dc.l 0
y:
        .dc.l 0
.text
        ① moveq.l #5,%d0
        ② cmp.l x,%d0
        ③ bcc else1
           move.l x,%d0
           lsl.l #3,%d0
           move.l %d0,y
           bra endif1
else1:
        ④ clr.l y
endif1:
        ⑤ .dc.w 0x4848
          stop #0
.end
```

分岐する

ラベル else1へ分岐せよという指示

ジャンプ

スキップ

そうでないときに実行される部分

以後省略

x > 5 での分岐

```
static long int x = 7;
static long int y = 0;

int main()
{
  if ( x > 5 ) {
    y = 8 * x;
  }
  else {
    y = 0;
  }

  return 0;
}
```

条件式

条件が成り立つ場合に実行される部分

条件が成り立たない場合に実行される部分

プログラムの実行順

```
.data
x:
        .dc.l 7
y:
        .dc.l 0
.text
        ① moveq.l #5,%d0
        ② cmp.l x,%d0
        ③ bcc else1
        ④ move.l x,%d0
        ⑤ lsl.l #3,%d0
        ⑥ move.l %d0,y
        ⑦ bra endif1
else1:
        ④ clr.l y
endif1:
        .dc.w 0x4848
        stop #0
.end
```

① 「5」をデータレジスタ D0 に格納

② x の値とデータレジスタ D0 を比較

③ 比較結果により else1 に分岐（条件分岐）

④ x の値をデータレジスタ D0 に格納

⑤ データレジスタ D0 の値を8倍にする

⑥ データレジスタ D0 の値を y に格納

⑦ endif1 に分岐

x > 5 のとき

④ y の値を0にする

そうでないとき

例題2．繰り返し

- 繰り返しの例として，次の例を考える

$$s = \sum_{i=1}^{3} i$$

繰り返し

```
static long int i = 1;
static long int s = 0;

int main()
{
  for ( i = 1; i <= 3; i++ ) {
    s = s + i;
  }

  return 0;
}
```

等価

等価

```
.data
i:
        .dc.l 1
s:
        .dc.l 0
.text
start1:
        /* i = 1, 2, 3 */
        cmp.l #3,i
        bhi break1

        move.l i,%d0
        add.l %d0,s
        addq.l #1,i
        bra start1
break1:

        .dc.w 0x4848
        stop #0
.end
```

関数の定義は，
今後の授業で触れる（今回は触れない）

繰り返し

```
static long int i = 1;
static long int s = 0;

int main()
{
  for ( i = 1; i <= 3; i++ ) {
    s = s + i;
  }

  return 0;
}
```

この部分は繰り返し処理
（for 文による繰り返し）

- i = 1 から開始
- i を 1 ずつ足しながら，「i <= 3」が成り立たなくなったら終了

繰り返し

- ある条件が満たされるまで，同じ処理を繰り返す
- ループ変数 （ループカウンタ）を使うことが多い

ループ変数： 繰り返しの回数を数える変数

- インクリメント　値を1増やす
- デクリメント　値を1減らす

繰り返し

実行結果の例

ここでは，i, s はともに4バイトのデータ

```
000000: 0c b9 00 00 00 03 00 00 00 2c 62 00 00 18 20 39
000010: 00 00 00 2c d1 b9 00 00 00 30 52 b9 00 00 00 2c
000020: 60 00 ff de 48 48 4e 72 00 00 00 00 00 00 00 04
000030: 00 00 00 06 00 00 00 00 00 00 00 00 00 00 00 00
000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

i

s

$$s = \sum_{i=1}^{3} i$$

繰り返し

繰り返しの終了条件
「i <= 3」が成り立たない

i と 3 の比較

i > 3
のときはジャンプ

```
.data
i:
        .dc.l 1
s:
        .dc.l 0
.text
start1:
        /* i = 1, 2, 3 */
        cmp.l #3,i
        bhi break1

        move.l i,%d0
        add.l %d0,s
        addq.l #1,i
        bra start1
break1:

        .dc.w 0x4848
        stop #0
.end
```

繰り返し

i と 3 の比較

ジャンプ
繰り返しを続ける

i ≦ 3
のとき

```
.data
i:
        .dc.l 1
s:
        .dc.l 0
.text
start1:
        /* i = 1, 2, 3 */
        cmp.l #3,i
        bhi break1

        move.l i,%d0
        add.l %d0,s
        addq.l #1,i
        bra start1
break1:

        .dc.w 0x4848
        stop #0
.end
```

繰り返し

i と 3 の比較

ジャンプ
繰り返しを続ける

i > 3 のときは
ジャンプ

そうでないときに
実行される部分

```
.data
i:
        .dc.l 0
s:
        .dc.l 0
.text
        /* i = 1, 2, 3 */
        moveq.l #1,%d0
        move.l %d0,i
start1:
        moveq.l #3,%d0
        cmp.l i,%d0
        blt break1

        move.l i,%d0
        add.l %d0,s
        addq.l #1,i
        bra start1
break1:

        .dc.w 0x4848
        stop #0
```