** (1) 1 + 2 + ... + N の計算

*** CPU のカード (レジスタ)

```
PC: 00
A: 0
B: 0
C: 0
```

*** CPU にできること

- PC が差す番地の指示を読み込む
  - PC の値は自動的に +1 される
- 読み込んだ指示に従う
  - カード (レジスタ) の値 ← 指定された値
  - カード (レジスタ) の値 ← 指定された番地の値
  - カード (レジスタ) の値 → 指定された番地の値
  - カード (レジスタ) の値を使って計算 (四則演算・論理演算・テスト)
  - (テストの結果によって) PC の値 ← 指定された値

*** 事前準備

50 番地に N の値を格納しておく。

*** メモリに書かれた「指示」 (プログラム)

```
00: A ← 0                    (合計値)
01: B ← 1                    (ループのインデックス)
02: A ← A + B
03: B と (50 番地) を比較    (テスト)
04: もし同じなら…… PC ← 07
05: B ← B + 1
06: PC ← 02
07: A → (51 番地)
08: 終了をOSに通知
09:
```

*** 計算に利用するメモリ (データ)

```
50: 3                        (N の値 (入力))
51: 6                        (計算結果 (出力)
52:
53:
54:
```

** (2) 1 + 2 + ... + N の計算と、1 * 2 * ... * M の計算 (2 プロセス版)

*** CPU のカード (レジスタ)

```
PC: 00
A: 0
B: 0
C: 0
```

*** CPU にできること

- PC が差す番地の指示を読み込む
  - PC の値は自動的に +1 される
- 読み込んだ指示に従う
  - カード (レジスタ) の値 ← 指定された値
  - カード (レジスタ) の値 ← 指定された番地の値
  - カード (レジスタ) の値 → 指定された番地の値
  - カード (レジスタ) の値を使って計算 (四則演算・論理演算・テスト)
  - (テストの結果によって) PC の値 ← 指定された値

*** 事前準備

50 番地に N の値を格納しておく。
100 番地に M の値を格納しておく。

*** メモリに書かれた「指示」 (プログラム)

```
00: A ← 0                    (合計値)
01: B ← 1                    (ループのインデックス)
02: A ← A + B
03: B と (50 番地) を比較
04: もし同じなら…… PC ← 07
05: B ← B + 1
06: PC ← 02
07: A → (51 番地)
08: 終了をOSに通知
09:

10: A ← 0                    (結果の値)
11: B ← 1                    (ループのインデックス)
12: A ← A * B
13: B と (100 番地) を比較
14: もし同じなら…… PC ← 17
15: B ← B + 1
16: PC ← 12
17: A → (101 番地)
18: 終了をOSに通知
19:
```

*** 計算に利用するメモリ (データ)

```
50: 3                        (N の値 (入力))
51: 6                        (計算結果 (出力))
52:
53:
54:
    :
    :
100: 4                       (N の値 (入力))
101: 24                      (計算結果 (出力))
102:
103:
104:
```

** (3) (1 + 2 + ... + N) + (1 + 2 + ... + M) の計算 (2 スレッド版)

*** CPU のカード (レジスタ)

```
PC: 00
A: 0
B: 0
C: 0
```

*** CPU にできること

- PC が差す番地の指示を読み込む
  - PC の値は自動的に +1 される
- 読み込んだ指示に従う
  - カード (レジスタ) の値 ← 指定された値
  - カード (レジスタ) の値 ← 指定された番地の値
  - カード (レジスタ) の値 → 指定された番地の値
  - カード (レジスタ) の値を使って計算 (四則演算・論理演算・テスト)
  - (テストの結果によって) PC の値 ← 指定された値

*** 事前準備

50 番地に N の値を格納しておく。
51 番地に M の値を格納しておく。
52 番地に 0 を格納しておく。

スレッド 1 の C レジスタを 0 にしておく。
スレッド 2 の C レジスタを 1 にしておく。

*** メモリに書かれた「指示」 (プログラム)

```
00: A ← 0                    (合計値)
01: B ← 1                    (ループのインデックス)
02: A ← A + B
03: B と (50 + C 番地) を比較
04: もし同じなら…… PC ← 07
05: B ← B + 1
06: PC ← 02
07: A ← A + (52 番地)
08: A → (52 番地)
09: 終了をOSに通知
```

*** 計算に利用するメモリ (データ)

```
50: 3
51: 4
52: 16
53:
54:
```

**** (4) C コンパイラが生成した x86 アセンブリ言語**

```
int N = 6;
int result;

int main(int argc, char **argv) {
  int sum = 0;
  int i;
  for (i = 0; i < N; i++) {
    sum += i;
  }
  result = sum;
  return (0);
}
```

```
N:
        .long   6                               変数 N のデータ領域
        .comm   result,4,4                      変数 result のデータ領域

main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    $0, -4(%ebp)                    sum = 0
        movl    $0, -8(%ebp)                    i = 0
        jmp     .L2
.L3:
        movl    -8(%ebp), %edx                  edx = i
        addl    %edx, -4(%ebp)                  sum = sum + edx
        addl    $1, -8(%ebp)                    i = i + 1
.L2:
        movl    N@GOTOFF(%eax), %edx            edx = N
        cmpl    %edx, -8(%ebp)                  edx == i?
        jl      .L3                             i < edx なら L3 へジャンプ
        movl    result@GOT(%eax), %eax          eax = result の番地
        movl    -4(%ebp), %edx                  edx = sum
        movl    %edx, (%eax)                    (eax が差す番地) ← edx
        movl    $0, %eax
        leave
        ret
```