8. アセンブリ言語によるプログラミング演習

—Web 版 CASL シュミュレータ向け—

1 目的

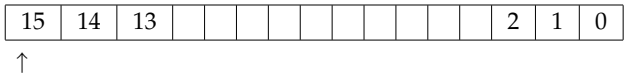
アセンブリ言語はハードウェアの命令とほぼ 1 対 1 に対応している言語で、この言語で書かれたプログラムは機械語に変換されて実行されます。したがって、ハードウェアを理解するためには最も適しています。現在では、アセンブリ言語で実際にプログラムを書くことはほとんどありませんが、情報処理に携わるものにとって、一度はアセンブリ言語によるプログラムの作成経験をもつことは重要です。この経験を通してハードウェアの理解がより高められるでしょう。

本実験では情報処理技術者試験で採用している仮想計算機 COMET II のアセンブリ言語 CASL II を使用したプログラミングを体験し、ハードウェアの理解を深めることを目的としています。

2 ハードウェア COMET II

2.1 COMET II の仕様

データ単位 1 語を単位とします。1 語は 16 ビットで、構成は次のようになります。



↑

符号（負：1，正：0）

アドレス 主記憶の容量は 65536 語（2^{16}）で、そのアドレスは 0〜65535 番地です。

数値 16 ビットの 2 進数で表現します。負数は、2 の補数で表現します。ここでは「−5」の例を示します。

```
0000000000000101 ← 5 の 2 進数
       ↓         ← ビットを反転
1111111111111010
+              1 ← 1 を加算
----------------
1111111111111011 ← -5 の 2 進数
```

レジスタ GR（16 ビット），SP（16 ビット），PR（16 ビット），FR（3 ビット）の 4 種類があります。

GR（汎用レジスタ，General Register）

GR0〜GR7 の 8 個があり、算術, 論理, 比較, シフトなどの演算に用います。このうち、GR1〜GR7 は、指標レジスタとしてアドレスの装飾にも用います。

FR（フラグレジスタ，Flag Register）

左側から、OF（Overflow Flag）, SF（Sign Flag）, ZF（Zero Flag）の順に並んだ 3 個のビットからなり、演算命令の結果によって値が設定されます。この値は分岐命令で使用されます。

OF	演算結果が 1 語に収まらなくなった時に 1、それ以外は 0 。
SF	演算結果が負のとき 1、それ以外は 0 。
ZF	演算結果が 0 のとき 1、それ以外は 0 。

SP（スタックポインタ，Stack Pointer）

スタックの最上段のアドレスを保持しています。

PR（プログラムレジスタ，Program Register）

次に実行すべき命令のアドレスを保持しています。

2.2 命令

COMET II の機械語命令の一覧を付録 A に示します。ただし、いくつかの命令は省略されています（28 命令のうち 24 命令掲載）。

2.3 文字の符号表

「JIS X 0201 ラテン文字・片仮名用 8 ビット符号」で規定する文字の符号表を使用します。この符号表の一部（カタカナ部分は省略）を次に示します。

行＼列	2	3	4	5	6	7
0	間隔	0	@	P	‘	p
1	!	1	A	Q	a	q
2	”	2	B	R	b	r
3	#	3	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	’	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
A	*	:	J	Z	j	z
B	+	;	K	[	k	{
C	,	<	L	¥	l	|
D	−	=	M	]	m	}
E	.	>	N	^	n	~
F	/	?	O	_	o	

表の見方 1 文字は 8 ビットからなり、上位 4 ビットを「列」で、下位 4 ビットを「行」で表します。

例えば、3, F, Z のビット構成は、16 進表記で、それぞれ 33, 46, 5A となります。

3　アセンブリ言語 CASL II

ここでは、CASL II の仕様について説明します。

3.1　命令の形式

1 つの命令は 1 行で書き、この行を「命令行」と呼びます。命令行は、次のような形式になます。

ラベル	Tab	命令コード	Tab	オペランド

【例】

		LD		GR1,LBL
LBL		DC		20

ラベル　その命令のアドレスを他の命令から参照するための名前です。長さは 1〜8 文字で、先頭の文字は大文字でなければなりません。それ以降の文字は、英大文字または数字のどちらかです。

コメント　行中にセミコロン「；」があると、そこから行の終わりまでがコメントとなります。行の先頭に；がある場合、または、；の前に空白しかない場合は、行全体がコメントとなります（「注釈行」と呼ぶ）。

3.2　命令の種類

CASL II の命令は、4 種類の**アセンブラ命令**（START, END, DS, DC）、4 種類の**マクロ命令**（IN, OUT, RPUSH, RPOP）、**機械語命令**（COMET II の命令）からなります。これらの命令の詳細については、それぞれ付録にまとめてあるので参照してください。

機械語命令	付録 A
アセンブラ命令	付録 B
マクロ命令	付録 C

4　CASL II プログラミング基礎

Web 上の CASL II シミュレータを開いて下さい。

`https://www.officedaytime.com/dcaslj/`

4.1　プログラムの作成・実行手順

1. ソース欄にコードを入力する
2.「アセンブル」ボタンでアセンブルする
3.「▶」ボタンで実行する
4. プログラム終了時のレジスタ（GR0〜GR7 など）を確認する

4.2　プログラムの構造

プログラムのおおまかな構造は次のようになります。START 命令の行はラベルが必須なので、以下の「ラベル名」には何かラベルを入れて下さい。（ラベルについては 3.1 節を参照）

```
ラベル名    START
            START から RET まで実行される
            RET
            DC, DS 命令
MSG         DC      'msg'
            END
```

4.3　プログラムの作成・実行の練習

4.3.1　プログラムの作成

4.2 節で説明したプログラムの構造に、4.4.1 節の【方法 1】にある LD 命令のサンプルを追加して、プログラムを作ってみましょう。

```
PROG1   START           ; メインプログラム開始
        LD      GR0,CON ;"GR"の後は数字のゼロ
        RET             ; メインプログラム終了
CON     DC      6       ; 定数 6 を作る
        END             ; プログラム終了
```

4.3.2　プログラムの実行

4.1 節で説明した流れで実行してみましょう。終了時に、GR0 に 6 が入ってることを確認して下さい。

4.3.3　命令の動き

付録 A を参照してプログラムの動きを確認してみます。

LD 命令の説明を確認してみましょう。「**（実行アドレス）**を r に読み込む」と書いてある部分は、**表記について**の部分に従うと、上記のプログラム内の LD 命令の書き方を、次のように当てはめることができます。

- 実行アドレスとは、adr と x の内容との論理加算値であり、adr には CON が指定されており、x は省略されているため、CON が表しているアドレス、つまり、「CON のラベルが付いている行のアドレス」を意味している。
- 「(実行アドレス)」に付いている括弧（）とは、括

弧内のアドレスに格納されている内容を表しており、ここでは「ラベル CON の行に格納されている内容、つまり整数 6」を意味している。

- r とは、GR0～GR7 までを指定することができ、ここでは GR0 を指定している。

以上のことから、「**（実行アドレス）**を r に読み込む」という動きは、ここでは「整数 6 を GR0 に読み込む」という意味になります。

4.3.4　他の命令についての動作確認

上記の要領で、次節「レジスタ操作」から 4.8 節「サブルーチン」のプログラムを作って（不要な命令行を取り除いて、新しい命令を追加するだけで完成する）、命令の動きを確認してみましょう。

4 節の内容が一通り理解できたら、6 節に進んで課題に取り組みましょう。（5 節のサンプルプログラムは必要に応じて、参考にして下さい。）

4.4　レジスタ操作

データを操作する場合は必ずレジスタを用います。

4.4.1　レジスタに値を入れる

GR0 に値を入れるには、次のような方法があります。

【方法 1】　LD 命令

```
        LD    GR0,CON

CON     DC    6
```

【方法 2】　LAD 命令

```
        LAD   GR0,8
```

【方法 3】　スタック

```
        PUSH  10
        POP   GR0
```

4.4.2　レジスタ間でコピーする

GR1 の値を GR0 にコピーする方法を示します。

【方法 1】　LD 命令と ST 命令

```
        ST    GR1,WK
        LD    GR0,WK

WK      DS    1
```

【方法 2】　LAD 命令

```
        LAD   GR0,0,GR1
```

【方法 3】　スタック

```
        PUSH  0,GR1
        POP   GR0
```

【方法 4】　LD 命令

```
        LD    GR0,GR1
```

4.4.3　レジスタの値を加減算する

GR3 に 1 を加算するには次のような方法があります。

【方法 1】　ADDA 命令

```
        ADDA  GR3,CON1

CON1    DC    1
```

【方法 2】　LAD 命令

```
        LAD   GR3,1,GR3
```

【方法 3】　スタック

```
        PUSH  1,GR3
        POP   GR3
```

【方法 4】　リテラル

命令の adr 部に「＝定数」という形式で書くと、その定数を DC 命令で自動的に用意してくれます。この書き方を「**リテラル**」と言います。リテラルの「定数」の部分には、10 進数, 16 進数, 文字定数が書けます。

方法 1 で、リテラルを使うと次のようになります。

```
        ADDA  GR3,=1
```

4.4.4　レジスタの符号を調べる

符号を調べるには、フラグレジスタ FR の値がどう変化しているかを見ます。FR については、2.1 節を参照して下さい。

FR を変化させるには、FR が設定される命令（付録 A に「FR を設定する」と書かれている命令）を使います。GR2 の符号を調べるには次の方法などがあります。

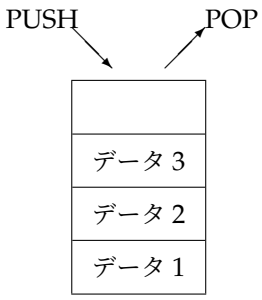
【方法 1】　CPA 命令

```
        CPA   GR2,CON0    ;0 と比較する

CON0    DC    0
```

4.5　スタックの操作

スタックとは、データを格納できる領域の一種です。下図のように、スタックは、領域の最上段にデータを格納（PUSH）し、最上段からデータを取り出す（POP）構造になっています。



スタックにはいろいろな使い方があります。次にその一例を示します。

4.5.1　レジスタの内容を入れ替える

【例】　GR1 と GR3 を入れ換える。

```
PUSH  0,GR1
PUSH  0,GR3
POP   GR1
POP   GR3
```

4.5.2　レジスタを退避する

レジスタが不足したときには一時退避しなければなりません。そのためにスタックが使われます。

【例】　GR1～GR3 をスタックを使って退避/復元する。

```
GR1=10, GR2=20, GR3=30 に設定する

PUSH  0,GR1
PUSH  0,GR2
PUSH  0,GR3

GR1、GR2、GR3 に別の値を入れる

POP   GR3
POP   GR2
POP   GR1

GR1=10, GR2=20, GR3=30 が復元される
```

また、GR1～GR7 をまとめてスタック操作する場合は、**マクロ命令 RPUSH, RPOP** を使うことができます（付録 C 参照）。

4.6　シフトとビット操作

CASL II には語をビット単位で扱える命令があります。これらの命令の基本操作を説明します。

4.6.1　レジスタの内容を 2^n 倍する

2 進数の値は、シフト命令で左へ n ビットシフトすると 2^n 倍され、右へシフトすると $1/2^n$ 倍されます。

【例】　GR0 を 2 倍、GR1 を 1/2 倍する。

```
      SLA   GR0,1
      SRA   GR1,1
```

4.6.2　レジスタの内容を 10 倍する

$10 = 8 + 2 = 2^3 + 2^1$ なので、8 倍したものと 2 倍したものを加算して求めます。

```
      SLA   GR1,1    ;GR1 ← GR1*2（元の 2 倍）
      ST    GR1,WK   ;GR1 → WK
      SLA   GR1,2    ;GR1 ← GR1*4（元の 8 倍）
      ADDA  GR1,WK   ;GR1 ← GR1+WK

WK    DS    1
```

4.7　分岐処理

4.7.1　指定回数分繰り返す

【例】　領域 A 以降の 10 語を加算する。

カウンタと番地修飾を同じレジスタで行います。この繰り返しは、カウンタを 9 から 0 へと減らしていく方法です。

```
      LAD   GR0,0     ; 和が入る GR0 をクリア
      LAD   GR1,9     ; カウンタ i に 9 を入れる
LOOP  ADDA  GR0,A,GR1 ;GR0 ← GR0+A(i)
      SUBA  GR1,=1    ;i ← i-1
      JPL   LOOP      ;if(i>0) LOOP へジャンプ
      JZE   LOOP      ;if(i=0) LOOP へジャンプ

A     DC    2,4,6,8,10,1,3,5,7,9
```

次のような方法もあります。この繰り返しは、カウンタを 0 から 9 へと増やしていく方法です。

```
      LAD   GR0,0
      LAD   GR1,0
LOOP  ADDA  GR0,A,GR1   ;GR0 ← GR0+A(i)
```

```
        LAD     GR1,1,GR1     ;i ← i+1
        CPA     GR1,=10       ;if i<終値
        JMI     LOOP          ;LOOP へジャンプ

A       DC      2,4,6,8,10,1,3,5,7,9
```

4.8 サブルーチン

サブルーチンを呼び出すには CALL 命令を使います。

4.8.1 サブルーチンへデータを引き渡す

【方法 1】 特定のレジスタに入れる。

次のプログラムは、GR0 に N を入れてサブルーチンに渡し、サブルーチンでは 1 から N までの和を求めて、和を GR0 に入れています。この方法は渡すデータが少ないときに使われます。

```
SUM1    START
        LD      GR0,N         ;GR0 ← N の内容（5）
        CALL    SUB
        RET
N       DC      5

SUB     ST      GR0,WK
        LAD     GR0,0
LOOP    ADDA    GR0,WK
        LD      GR1,WK
        SUBA    GR1,=1
        ST      GR1,WK
        JNZ     LOOP
        RET
WK      DS      1
        END
```

【方法 2】 パラメータリストのアドレスを渡す。

サブルーチンに渡すデータを連続する記憶域に置き、その先頭アドレスを渡します。渡すデータが多いときに有効です。

```
SUM2    START
        LAD     GR1,N     ;GR1 ←「N のアドレス」
        CALL    SUB
        RET
          ; 連続したアドレスに 3 個の定数を定義
N       DC      10
        DC      5
        DC      20

    ; アドレス GR1+0 に入っている値を GR2 に入れる
SUB     LD      GR2,0,GR1   ;GR2 ← 10
    ; アドレス GR1+1 に入っている値を GR3 に入れる
        LD      GR3,1,GR1   ;GR3 ← 5
    ; アドレス GR1+2 に入っている値を GR4 に入れる
        LD      GR4,2,GR1   ;GR4 ← 20

        LAD     GR0,0
        ADDA    GR0,GR2
        ADDA    GR0,GR3
        ADDA    GR0,GR4
        RET
        END
```

5 CASL II プログラミング入門

ここでは、CASL II プログラミングの基本的技法を用いたサンプルプログラムを示します。

5.1 ゼロ抑制処理

このプログラムは、8 けたの数字データを入力して、ゼロ抑制処理を行った後、結果を表示するプログラムです。ゼロ抑制処理とは、先頭にある'0' を、'0' 以外の数字が現れるまで間隔文字（スペース）に置き換える処理のことです。

プログラムは以下のような動きをします。

1. 【入出力例】
   入力データ：００００１００４
   結果の表示：△△△△１００４
2. 入力桁数が 8 桁でないのときは、エラー表示をし再入力を要求する。
3. 全けた'0' のデータを入力したときは、最後の'0' だけを表示する。
4. 復帰符号のみの入力で、プログラムは終了する。

```
PROG51  START BEGIN
BEGIN   IN    INBUF,LENG    ; データの入力

        LD    GR3,LENG
        CPA   GR3,C0        ; 復帰符号の入力?
        JZE   LAST
        CPA   GR3,C8        ; 入力桁数=8?
        JZE   GOOD
```

```
        OUT   ERMSG,C9        ; 入力エラー
        JUMP  BEGIN

GOOD    LD    GR0,ZERO        ; レジスタ初期設定
        LAD   GR1,0           ; 同上
        LD    GR2,SPACE       ; 同上

LOOP    CPA   GR1,C7          ;7 桁分終了?
        JPL   OUTPUT
        JZE   OUTPUT
        CPL   GR0,INBUF,GR1   ; 入力データ='0'?
        JNZ   OUTPUT
        ST    GR2,INBUF,GR1   ; ゼロ抑制
        LAD   GR1,1,GR1       ; インデックス更新
        JUMP  LOOP

OUTPUT  OUT   INBUF,C8        ; 結果の表示
        JUMP  BEGIN
LAST    RET

LENG    DS    1
INBUF   DS    80
ZERO    DC    '0'
SPACE   DC    ' '
C0      DC    0
C7      DC    7
C8      DC    8
C9      DC    9
ERMSG   DC    '--ERROR--'
        END
```

5.2 最大値と最小値を求める

このプログラムは、GR1 に格納されたアドレスから始まり、GR2 で示される語数からなる領域中の数値の最大値及び最小値を求めるサブルーチンです。

```
MAXMIN ST     GR2,GOSU    ; 語数→ GOSU
       LD     GR0,0,GR1   ;GR0 ←先頭の数値
       ST     GR0,MAX
       ST     GR0,MIN
       LAD    GR3,1       ; 語数のカウンタ

S1     CPA    GR3,GOSU    ; カウンタと語数の比較
       JPL    SRET
       JZE    SRET
       LAD    GR1,1,GR1   ; インデックス更新
       LD     GR0,0,GR1   ;GR0 ←次の数値

       CPA    GR0,MAX     ; 最大値と比較
       JMI    S2          ;GR0<MAX
       JZE    S3          ;GR0=MAX
       ST     GR0,MAX     ; 最大値を更新
       JUMP   S3

S2     CPA    GR0,MIN     ; 最小値と比較
       JPL    S3          ;GR0>MIN
       JZE    S3          ;GR0=MIN
       ST     GR0,MIN     ; 最小値を更新

S3     LAD    GR3,1,GR3   ; カウンタを 1 加算
       JUMP   S1

SRET   LD     GR2,MAX
       LD     GR3,MIN
       RET

GOSU   DS     1
MAX    DS     1
MIN    DS     1
       END
```

6 実験レポートについて

以下の項目でまとめてください。

1. 目的
2. 実装：全ての課題で実装したプログラムコードを報告してください。
3. 解説：実装したプログラムコードがどのような処理となっているのか、サブルーチンごとに、図（UML アクティビティ図, フローチャートなど）で説明してください。最低でも、課題の中から 3 個を選び説明してください。
4. 結果：全ての課題で実行した結果を、シミュレータ内での最終的な出力結果が分かるように報告してください。
5. 低水準言語の需要：現代社会において、アセンブリ言語のような低水準言語が必要とされるのはどのような場合なのかを調べて報告してください。調べた際の情報源を参考文献に掲載してください。
6. 結論
7. 参考文献：Web 上の参考文献を載せる場合は、Web サイトの名前, URL, 参照日を報告してください。

7　実験課題

【課題 1】　次のようなサブルーチン作り、動作を確認しなさい。

1. 10×(GR2)+(GR3)→ $GR0$
2. サブルーチンを実行した時、サブルーチン内で値が変更されるレジスタの内容は保存される（GR0 は除く）。
3. メインプログラムで GR2 に 1 を、GR3 に 8 を入れて、このサブルーチンを呼び出してみる。

補足説明

課題 1　次のような処理手順でプログラムを考えます。

1. 保存するレジスタの値を退避する。
2. GR2 の値を 10 倍する。
3. GR2 と GR3 の値を加算する。
4. 加算結果を GR0 に入れる。
5. レジスタの値を元に戻す。

【課題 2】　GR1 に n を入れて呼び出されたとき n^2 を GR0 に求めるサブルーチンを作り、動作を確認しなさい。

1. $n^2 = \sum_{i=1}^{n}(2i-1)$ によって n^2 を求める。
2. サブルーチンを実行した時、サブルーチン内で値が変更されるレジスタの内容は保存される（GR0 は除く）。
3. メインプログラムで GR1 に 6 を入れて、このサブルーチンを呼び出してみる。

補足説明

課題 2　n^2 を求める数式 $\sum_{i=1}^{n}(2i-1)$ は、「1 から $2n-1$ までの奇数を加算する」と考えることができます。したがって、次のような繰り返し処理をするサブルーチンを作ります。

1. 保存するレジスタの値を退避する。
2. $2n-1$ を求めてカウンタとするレジスタに入れる。
3. 結果を入れるレジスタ GR0 を 0 にリセットする。
4. GR0 にカウンタを加算する。
5. カウンタを 2 減らす。
6.「カウンタ >0」の間 4.〜5. を繰り返す。
7. レジスタの値を元に戻す。

【課題 3】　GR1 に正整数 M を入れ、GR2 に正整数 N を入れ、呼び出しされたとき、M と N の最大公約数を GR0 に求めるサブルーチンを作り、動作を確認しなさい。

1. 次のレジスタを作業領域とする。GR_{WKM}： 整数 M の作業用領域，　GR_{WKN}： 整数 N の作業用領域
2. サブルーチンを実行した時、サブルーチン内で値が変更されるレジスタの内容は保存される（GR0 は除く）。
3. M を 30、N を 84 として、メインプログラムからこのサブルーチンを呼び出してみる。

補足説明

課題 3　最大公約数はユークリッドの互除法によって求めることができます。剰余を求める命令がないため、代わりに減算を用いる方法を使います。

1. 保存するレジスタの値を退避する。
2. M（GR1）の値を GR_{WKM} に入れる。
3. N（GR2）の値を GR_{WKN} に入れる。
4. 結果を入れるレジスタ GR0 を 0 にリセットする。
5. GR_{WKN} と GR_{WKM} を比較する。
6. 等しい場合、10. に行く。
7. GR_{WKM} の方が大きい場合、9. に行く。
8. GR_{WKN} の方が大きい場合（つまり条件なしの場合）、GR_{WKN} から GR_{WKM} を引き、5. へ行く。
9. GR_{WKM} から GR_{WKN} を引き、5. へ行く。
10. GR_{WKN} の値を GR0 に入れる。
11. レジスタの値を元に戻す。

【課題 4】　GR1 に整数 M（正でも負でもよい）を入れ GR2 に正整数 N を入れて呼び出し出されたとき、M×N を計算して結果を GR0 に格納するサブルーチンを作り、動作を確認しなさい。

1. 次のレジスタを作業領域とする。
   GR_{WKM}： 整数 M の作業用領域、シフトして（2^n 倍して）使う。
   GR_{WKN}： 整数 N の作業用領域、シフトして（2^{-n} 倍して）使う。
2. サブルーチンを実行した時、サブルーチン内で値が変更されるレジスタの内容は保存される（GR0 は除く）。
3. M を 4、N を 5 として、メインプログラムからこのサブルーチンを呼び出してみる。
4. M を −5、N を 2 として、メインプログラムからこのサブルーチンを呼び出してみる。

補足説明

課題 4　乗算 (M×N) は次のように値を 2 進数にした筆算で考えます。

```
M = 4      0100     ←にシフト
N = 5    × 0101     →にシフト
          ------

            0100
           0000
          0100
       + 0000
         ------

        0010100  ⇒  20
```

上図のように、N を右にシフトしていき、N の最下位のビットが 1 ならば M を加算、0 ならば 0 を加算するという処理になります。

したがって、次のようなサブルーチンを作ります。

1. 保存するレジスタの値を退避する。
2. M（GR1）の値を GR_{WKM} に入れる。
3. N（GR2）の値を GR_{WKN} に入れる。
4. 結果を入れるレジスタ GR0 を 0 にリセットする。
5. GR1 に 1 を入れ、GR_{WKN} との AND を求める。
   （つまり GR_{WKN} の最下位のビットが 0 か 1 か調べる）
6. 5. の結果が 0 の場合、8. に行く。
7. GR0 に GR_{WKM} を加算する。
8. GR_{WKM} を値を左に 1 シフトする。
9. GR_{WKN} を値を右に 1 シフトする。
10. 9. の結果 GR_{WKN} が 0 になるまで、5.〜9. を繰り返す。
11. レジスタの値を元に戻す。

【課題 5】　GR1 に x を入れ GR2 に y を入れて呼び出されたとき、x^y を GR0 に求めるサブルーチンを作り、動作を確認しなさい。

1. 課題 4 で作成したサブルーチンを繰り返し呼び出して計算する。
2.「$x \geq 1, y \geq 1$」である値に対して、メインプログラムからこのサブルーチンを呼び出してみる。

補足説明

課題 5　例えば、x^y が 3^4 の場合は、次のように課題 4 のサブルーチンを繰り返し呼び出して計算します。

```
M*N(GR0)  M(GR1)   N(GR2)
 3   ←    1    ×    3
 9   ←    3    ×    3
27   ←    9    ×    3
81   ←   27    ×    3
                   ---

          計 4 回
```

したがって、次のようなサブルーチンを作ります。

1. 保存するレジスタの値を退避する。
2. カウンタとするレジスタに y の値を入れる。
3. 課題 4 のサブルーチンに渡す値の初期値を、
   M（GR1）は 1、N（GR2）は x とする。
4. 課題 4 のサブルーチンを呼び出す。
5. 得られた計算結果を M とする。
6. カウンタを 1 減らす。
7.「カウンタ ＞ 0」の間 4.〜6. を繰り返す。
8. レジスタの値を元に戻す。

付録 A　機械語命令一覧表

表記について　**r, r1, r2** — 汎用レジスタ（GR0～GR7），**adr** — アドレス（0～65535），**x** — 指標レジスタ（GR1～GR7），
[　] — 括弧内は省略可能，**(　)** — 括弧内のレジスタまたはアドレスに格納されている内容，
実効アドレス — adr と x の内容との論理加算値またはアドレス

命令	命令コード	オペランド	説明
ロード (LoaD)	LD	r, adr [, x]	**（実効アドレス）**を r に読み込む。結果より FR を設定する。
	LD	r1, r2	（r2）を r1 に読み込む。結果より FR を設定する。
ストア (STore)	ST	r, adr [, x]	（r）を実効アドレスに格納する。
ロードアドレス (Load ADdress)	LAD	r, adr [, x]	**実効アドレス**を r に読み込む。
算術加算 (ADD Arithmetic)	ADDA	r, adr [, x]	オペランドの書式が次の 2 通りある。 **r, adr [, x] の場合** (r) と (実効アドレス) に指定された演算を施し、結果を r に格納する。 **r1, r2 の場合** (r1) と (r2) に指定された演算を施し、結果を r1 に格納する。 これらの命令は、演算結果によって FR を設定する。
	ADDA	r1, r2	
算術減算 (SUBtract Arithmetic)	SUBA	r, adr [, x]	
	SUBA	r1, r2	
論理積 (AND)	AND	r, adr [, x]	
	AND	r1, r2	
論理和 (OR)	OR	r, adr [, x]	
	OR	r1, r2	
排他的論理和 (eXclusive OR)	XOR	r, adr [, x]	
	XOR	r1, r2	
算術比較 (ComPare Arithmetic)	CPA	r, adr [, x]	(r) と (実効アドレス)、または (r) と (r2) の比較を行い、結果を FR（の中の SF と ZF）に設定する。 【SF の値】　＞：0，＝：0，＜：1 【ZF の値】　＞：0，＝：1，＜：0
	CPA	r1, r2	
論理比較 (ComPare Logical)	CPL	r, adr [, x]	
	CPL	r1, r2	
算術左シフト (Shift Left Arithmetic)	SLA	r, adr [, x]	**符号を除き**、(r) を実効アドレスのビット数だけ左または右にシフトする。 シフトの結果、空いたビットには、左シフトのときは 0、右シフトのときは符号と同じ値が入る。
算術右シフト (Shift Right Arithmetic)	SRA	r, adr [, x]	
論理左シフト (Shift Light Logical)	SLL	r, adr [, x]	**符号を含み**、(r) を実効アドレスのビット数だけ左または右にシフトする。 シフトの結果、空いたビットには 0 が入る。 SLA, SRA, SLL, SRL は、演算結果によって FR を設定する。
論理右シフト (Shift Right Logical)	SRL	r, adr [, x]	
正分岐 (Jump on PLus)	JPL	adr [, x]	FR の値によって、実効アドレスに分岐する。 分岐しないときは次の命令に進む。
負分岐 (Jump on MInus)	JMI	adr [, x]	
非零分岐 (Jump on Non Zero)	JNZ	adr [, x]	
零分岐 (Jump on ZEro)	JZE	adr [, x]	
オーバーフロー分岐 (Jump on OVerflow)	JOV	adr [, x]	
無条件分岐 (unconditional JUMP)	JUMP	adr [, x]	無条件に実効アドレスに分岐する。
プッシュ (PUSH)	PUSH	adr [, x]	実効アドレスをスタックに格納する。
ポップ (POP)	POP	r	スタックから取り出した値を r に読み込む。
コール (CALL subroutine)	CALL	adr [, x]	実効アドレスで指定したサブルーチンに分岐する。（その際、現在の PR をスタックに格納する。）
リターン (RETurn from subroutine)	RET		サブルーチンを終了し、コールした場所に分岐する。（その際、スタックから取り出した値が示すアドレスに分岐する。）

付録 B　アセンブラ命令一覧表

命令コード	オペランド	説明
START	[実行開始ラベル]	プログラムの先頭を定義する。「実行開始ラベル」を指定した場合はそのラベルのアドレスから、省略した場合は次の命令から開始する。 この命令行のラベルは必須である。
END		プログラムの終わりを定義する。この命令行にラベルは付けられない。
DS	語数	指定した「語数」(0 以上の 10 進数）の領域を確保する。 語数を 0 とした場合、領域は確保しないが、ラベルは有効である。
DC	定数 [, 定数] ・・・	「定数」で指定したデータを（連続する）語に格納する。 定数には以下の 4 種類がある。 **10 進定数**（書き方：n） n で指定した 10 進数値を、1 語分の 2 進数データとして格納する。 $-32768 \leq n \leq 32767$ ではない（値が 16 ビットで表現できない範囲）ときは、下位 16 ビットを格納する。 **16 進定数**（書き方：#h） h で指定した 16 進数値を、1 語分の 2 進数データとして格納する。 h は 4 けたの 16 進数とする（0000≤h≤FFFF)。 **文字定数**（書き方：’文字列’） 文字列の文字数分の連続する領域を確保し、最初の文字は第 1 語の下位 8 ビットに、2 番目の文字は第 2 語の下位 8 ビットに、・・・と順に文字データとして格納する。各語の上位 8 ビットには 0 のビットが入る。 **アドレス定数**（書き方：ラベル） ラベルに対応するアドレスを、1 語の 2 進数データとして格納する。

付録 C　マクロ命令一覧表

命令コード	オペランド	説明
IN	入力領域, 入力文字長領域	入力装置から文字データを読み込む。入力された文字は、「入力領域」の先頭から順に 1 文字が 1 語に対応するように格納される。入力された文字の長さは「入力文字長領域」に格納される。
OUT	出力領域, 出力文字長領域	出力装置に文字データを表示する。「出力領域」に格納されているデータを、「出力文字長領域」に格納されている文字の長さだけ表示する。
RPUSH		GR の内容を、GR1, GR2, ・・・, GR7 の順序でスタックに格納する。
RPOP		スタックの内容を取り出し、GR7, GR6, ・・・, GR1 の順序で GR に格納する。

参考文献

[1] CASL 入門, http://ww3.tiki.ne.jp/ tno2/shikaku/casl_content.htm

[2] CASL II シミュレータ, 情報処理技術者試験センター, http://www.jitec.jp/

[3] 甘利直幸, CASL プログラミング（オーム社, 1986）

[4] 廣松恒彦, 山口和子, COMET/CASL（オーム社, 1988）