第７回　アセンブラプログラミング２（配列とループ）　予習資料

第7回授業の前にあらかじめ目を通し、問題を解いておくこと。
前回に続いて、今回も機械命令を組合せてプログラムにすることを試みる。

［1］ 繰り返し（ループ）

a.　Java もしくは C 言語で、10 回繰返して“Hallo“を表示するループを書け。 次にそれを流れ図（フローチャート）に書いてみよ。但し“Hallo“を表示する部分は1つの箱として書いてよい。

b.　上記の流れ図を、COMET-II 上のアセンブラプログラムとして書いてみよ。但し“Hallo“を表示する部分は1つの箱として書いてよい。

c.　下記は、上記の 1 から 100 までを加える手順をアセンブリプログラムに書き直したものである。［　　］を埋める正しい組み合わせはどれか。但し、変数 S、I はプログラムの最後に DS 命令で確保され、また定数（上限）MAX は DC 命令で領域確保されて値 100 に初期化されているとする。

① ア CPA GR3,100　イ ADDA GR5,GR4　ウ LD GR5,1
② ア CPA GR3,0　イ ADDA GR4,GR5　ウ LAD GR5,1
③ ア CPA GR3,MAX　イ ADDA GR4,GR5　ウ LAD GR5,1
④ ア CPA GR3,MAX　イ ADDA GR5,GR4　ウ LD GR5,1

```
01        LAD  GR3,0    ; S=0
02        ST   GR3,S
03        LAD  GR3,1    ; I=1
04        ST   GR3,I
05   L1   LD   GR3,I    ; I<=MAX?
06        [  ア  ]
07        JPL  L2       ;   jump to exit
08        LD   GR4,S    ;   S=S+I
09        LD   GR5,I
10        [  イ  ]
11        ST   GR4,S
12        LD   GR4,I    ;   I=I+1
13        [  ウ  ]
14        ADDA GR4,GR5
15        ST   GR4,I
16        JUMP L1       ; jump to top
```

d.　上記のプログラムで、いくつかの命令が無駄に使われている。下記の指摘のうち、誤っているものを選べ

① 1行目と2行目の、変数 S の 0 への初期化は、変数 S の領域確保のためのアセンブリ命令 DS を、定数設置のためのアセンブリ命令 DC 0 に置き換えることによって、2行共に省略することができる

② 8行目から11行目の、S=S+I の計算は、変数 S も I も一旦汎用レジスタに移してからレジスタ同士の加算をしているが、そのうち1つはメモリに置いたまま ADDA 命令の（レジスタ＋メモリ）のパターンを使えば、1行短

縮できる

③ 5行目の LD GR3,I は、その前に4行目で ST GR3,I をしているので、まだ I の値が汎用レジスタ GR3 に残っているはずである。だから削除して構わないので、1行短縮できる

④ 全体に見て、変数 I をメモリにおいているが、I は後から使わないのでメモリにおく必要はなく、汎用レジスタの1つを固定的に I のために割当てて使うことができる。これにより、4行目の ST 命令、5行目の LD 命令、9行目の LD 命令、12行目の LD 命令、15 行目の ST 命令が省略できる

[補足] 上記のようにして作ったアセンブリプログラムを、COMET Ⅱ のシミュレータ上で動作させてみよ。 ただし、MAX を 100 にすると、ステップ実行で最後まで行うのが大変なので、まずは MAX を 4 ぐらいにセットしてステップ実行してみるとよい。全てがうまく行ったら、MAX を 100 にして結果が正しい(5050)かどうかを確認せよ

[補充問題] 時間と体力に余裕のある学生は、似たようなプログラムをいくつか作ってみるとよい。

e. 上記(2)で工夫した、命令節約(プログラム短縮)の結果を、シミュレータ上で動作させて、期待通りの動作が得られるか確認せよ。上記(2)では短縮し残した部分があるので、自分で工夫して更にプログラムの短縮をトライし、それの動作をシミュレータ上で確認してみるとよい。

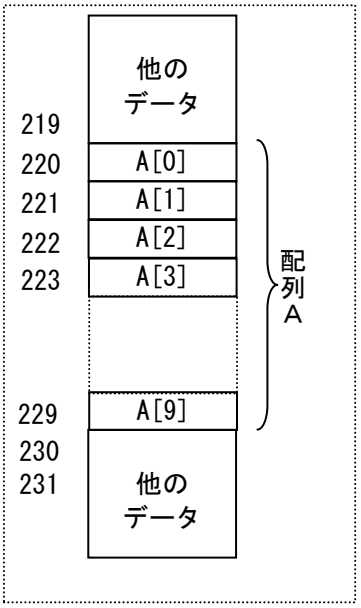
f. 次のプログラムをアセンブリプログラムに直してみよ

```
s = 0;
for (i=1; i<=10; i=i+2) {
    s = s + i;
}
```

g. 次のプログラムをアセンブリプログラムに直してみよ

```
x = 1;
while (x <= 10) {
  x = x + x;
}
```

［2］　配列のプログラミング



［補足］ 配列は、繰り返しのプログラムにおいて必ず出てくるプログラム要素である。同じ型の要素を複数個並べたものであり、メモリ上では連続した領域に取られる。たとえば要素が整数の場合、COMET では1つの整数はメモリ上の1つの場所(アドレス)を占めるので、右の図のように、先頭のアドレス(＝この場合 220 番地)から1ずつアドレスが進みながら置かれてゆく。この例では配列 A は整数型の 10 個の要素を持つので、220 番地から 229 番地を占めている。

(注: 個々の要素が整数ではなくて、もっと大きな場所を占める場合、たとえば1つの要素が5個のアドレスを占めるような場合では、要素のアドレスは5ずつ進み、A[0]が 220～224 番地なら、A[1]は 225～229 番地、A[2]は 230～234 番地、のように置かれ、A[9]は 265～269 番地に置かれることになる。)

このような配列にアクセスするには、要するに欲しいもののアドレスをうまく計算すればよいのであり、いろいろな方法が考えられる。 一般に、アドレスを≪計算≫し、その結果をアドレスとしてメモリにアクセスすることになるので、「間接アドレッシング」の仕組を使うと便利である。COMET II では「メモリ間接アドレッシング(オペランドで指定したメモリ内容をアドレスと解釈して、そのアドレスの指すメモリをアクセスする)」が装備されていないので、レジスタ内容による間接アドレッシング、具体的には「インデックス(指標)アドレッシング」を用いる。

具体的には、上記の例の A[2]を読み出すならば、汎用レジスタ GRn に値 2 を入れておき、 LD 220,GRn のようにしてアクセスできる。GRn は GR0～7 のどのレジスタでもよい。これによって、命令中のオペランドの値 220 と GRn の内容 2 が加算され、メモリの 202 番地がロードされる。 当然ながら、GRn の内容を0～9の範囲で変えることによって、配列の任意の要素がアクセスできる。

(注意: GRn の値が 0～9 を超えた場合、ハードウェアは単に 220+GRn をアドレスとし、何らのチェックをしないので、とんでもないアドレスをロードすることになる。)

また、220 を数字で書くのではなく、DS アセンブラ命令に付けたラベルを書くことができる。 右のように、領域(＝配列の場所)A として 10 語分(＝10 整数分)を DS 命令で確保し、その領域の先頭がたまたま 220 番地だったとすれば、ラベル A には 220 が対応するので、 LD A,GR3 によって、アドレス(220+GR3)の指す場所からロードされる。もし領域 A の場所が 734 番地(適当な数である)から 10 個分であれば、LD A,GR3 によって(734+GR3)の指す場所からロードされる。いずれにせよ配列 A の先頭から GR3 個目の要素を取出すことになる。

```
        ...     ←GR3 に値 3
        LD    GR5,A,GR3
        ...
        ...
A       DS 10     ←丁度 220 番地
MAX     DC 10
X       DC 0
        ...
```

a.　上記補足説明を参考にして、 X = A[2] + A[4]　を求めるプログラムを考える。 正しいのはどれか

①
```
LAD   GR3,2
LD    GR4,A,GR3
LAD   GR3,4
ADDA  GR4,A,GR3
ST    X,GR4
A     DS    10
X     DC    0
```

②
```
LAD   GR3,2
LD    GR3,A,GR4
LAD   GR3,4
ADDA  GR4,A,GR3
ST    X,GR4
A     DS    10
X     DC    0
```

③
```
LAD   GR3,2
LD    GR4,A,GR3
LAD   GR4,4
ADDA  GR4,A,GR3
ST    X,GR4
A     DS    10
X     DC    0
```

④
```
LAD   GR3,2
LD    GR3,A,GR4
LAD   GR4,4
ADDA  GR3,A,GR4
ST    X,GR4
A     DS    10
X     DC    0
```

b.　上記補足説明を参考にして、配列 A の要素をすべて加えるプログラムを作る。［　　］に入る文の正しい組合せはどれか

```
        LAD   GR5,0          ; GR5 を変数 S の途中結果として使う
        LAD   GR3,0          ; GR3 を変数 I として使う
L1      [　　ア　　]   ;
        JPL   L2
        [　　イ　　]   ;
        JUMP  L1
L2      ST    GR5,S          ; 和の結果を変数 S にしまう
MAX     DC    9              ; 定数 9
S       DC    0              ; 変数 S
A       DS    10             ; 配列 A[10]
```

```
S = 0;
for (I=0; I<=9; I++) {
    S = S + A[I];
}
```

① ア　CPA　GR3,MAX　　イ　ADDA GR5,A,GR3
② ア　CPA　GR3,MAX　　イ　ADDA GR3,A,GR5
③ ア　CPA　MAX,GR3　　イ　ADDA GR5,A,GR3
④ ア　CPA　MAX,GR3　　イ　ADDA GR3,A,GR5

［補足］
上記(3)と(4)で作ったアセンブリプログラムを、COMET II シミュレータ上で動作させてみよ。 但し、いずれも配列要素の値が必要なので、シミュレータ上での配列のアドレスを求めた上で、正しいメモリ上の場所に数値を入れよ。たとえば A[0]=0, A[1]=1, … A[9]=9 などと設定してみるとよいだろう。

［補充問題］ 時間と体力に余裕のある学生は、似たようなプログラムをいくつか作ってみよ。

c.　配列の初期化をしてみよう。 整数要素 10 個からなる配列 A に、A[0]=0, A[1]=1, A[2]=2, ... A[9]=9 となるように値をセットするには、どうすればよいか。

```
for (i=0; i<10; i++) {
    A[i] = i;
}
```

d.　整数要素 10 個からなる配列を、3つ用意する。それぞれを A, B, Cとするとき、ベクトルの加算のように、すべての I(0≦I≦9) について C[I]=A[I]+B[I] とするようなプログラムを作れ。

```
for (I=0; I<10; I++) {
    C[I] = A[I] + B[I];
}
```

e.　メインメモリ上の領域(配列)A から、領域(配列)B へ、1000 語分(1語は COMET II の1語とする)をコピーするアセンブリ言語プログラムを作れ。 ラベル A や B は領域(配列)の先頭要素に付いている。各々の領域は、コピーする個数(1000語分)よりは大きいものとする。 ちなみに、C 言語のライブラリに memcpy や bcopy と呼ばれるコピー関数があるが、それと同様の機能である。

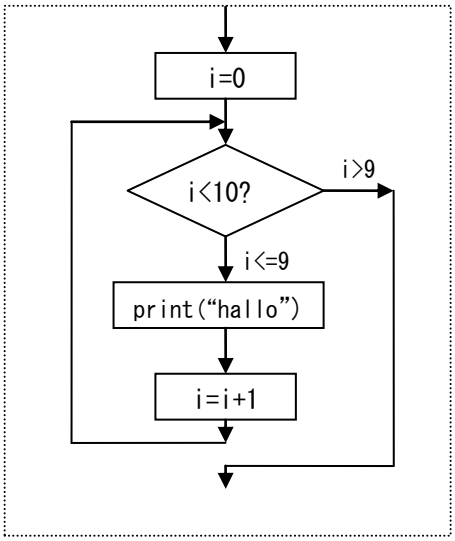
f.　上と同様に、メインメモリ上の領域 A から、領域 B へ、最大 1000 語分コピーするが、もし途中に値0が出てきたらそこでコピーを中止せよ(0自身はコピーする)。つまり、領域Aの先頭から、0でない要素を次々とコピーし、初めて0が出てきたときにその直後でコピーを止める。もし領域 A の先頭が0であったなら、その0だけをコピーする。このような動作をするアセンブリ言語プログラムを作れ。 ちなみに、C言語のライブラリに strcpy と呼ばれる文字列コピー関数があるが、それと似た機能である。

≪解答例≫
[1]

a.

```
void main() {
  int i;
  for (i=0; i<=9; i++) {
    printf( “Hallo” );
  }
}
```

```
class hallo {
 public static void main(String[] args) {
   int i;
    for (i=0; i<=9; i++) {;
    System.out.print( “Hallo” );
   }
 }
}
```



流れ図(フローチャート)は右図

b.

```
01      LAD  GR3,0    ; i=0
02      LAD  GR4,9    ; GR4=9
03      LAD  GR5,1    ; GR5=1
04  L1  CPA  GR3,GR4 ; i-9>0 ?
05      JPL  L2       ; jump if i-9<0
06      print(“hallo“)
07      ADDA GR3,GR5 ; i=i+1
08      JUMP L1      ; jump to top
09  L2  次の命令
```

c.　③　06 行は I と MAX の比較で、I は GR3 に入っているので、CPA GR3,MAX（か逆か）。 10 行は S=S+I だが S は GR4 に、I は GR5 に入っているので、ADDA GR4,GR5 か、ADDA GR5,GR4 なのだが、11 行で結果を GR4 から S にストアしているので、結果を GR4 に残すために ADDA GR4,GR5 である。 13 行は I=I+1 で、I は GR4 にあるので、LDA GR5,1 である。

d.　①は正しい。初期化さえすればよい時は DC 命令で 0 をセットしておいても同じ効果。
②は正しい。08 行目をやめて、10 行目を ADDA GR5,S、11 行目を ST GR5,S とすればよい。
③は間違い。05 行目に到達する道筋は、この説明にある 04 行目からの時と、その他に 16 行目から JUMP してくる時があり、16 行目から来る時は I の値が GR3 にあるとは限らない(現に 06 行目で足算をした時に GR3 に足算の結果が入ってしまう)
④は正しい。一般にこのような I や、計算の途中結果などは、レジスタだけに置いて、メモリに書き戻すことをしないのが、高速化のための良い工夫である。

e.（略）

f.　の補足　唯一の違いは i=i+2 であるので、その部分だけ変えればよい。 余談だが、Pentium など CPU の命令体系に、「1を足す(Increment)」という命令があるものがある。Increment を使って+1 を実現すると楽なのだが、+2 は面倒を見てくれないので2回 Increment するか、素直に2を加えるか、どちらかであろう。

g.　の補足　これは、流れ図(フローチャート)から書き起こす必要がある。while 文の動作を考えて、書き起こして欲しい。

[2]

a.　①

配列要素 A[2] を読み出す作業は、インデックスアドレシングを用いると、GR3 に 2 を入れておいて、 LD　GR4, A, GR3 のようにすればよい。この LD 命令の意味は、配列領域 A の先頭アドレス A と、GR3 の内容 2 を、足した値(A+2)をアドレスとしてメモリにアクセスし、その内容を持ってきて GR4 に入れよ、である。もし A の値が実際には 220 番地にアセンブルされるとすると、220+2 = 222 番地の中にある値を GR4 に入れることになる。
(注意)GR3 と GR4 を逆にすると、違う意味になるので注意。
同様に、A[4]を読み出すのも、GR3 に 4 を置いておいて、LD　GR4, A, GR3　のようにすればよい。

b.　③
配列全部の要素を加え合わせるには、I を1つずつ増やしながら、 S = S + A[I] を行えばよい。アの部分はループの脱出条件を書く比較命令になる。次が JPL 命令になっているから、 CPA u, v とすれば (u > v) の時に脱出する。(I > 9) で脱出したいのだから　CPA　GR3, MAX。
イは、GR3 を I として使うことにしてあり、また足し算の結果を GR5 から S にしまっているので、イの足し算命令は
ADDA　GR5, A, GR3　とする。(A に GR3 でインデックス修飾した場所をアクセスし、それと GR5 を足して、結果の和を GR5 にしまう)

c.　の補足: 要するに A[I]に I の値をストアすればよい。

```
        LAD   GR3,0          ; GR3 を変数 I として使う
L1      CPA   GR3,MAX        ;
        JPL   L2
        ST    GR3,A,GR3      ; A の GR3 番目の要素に、GR3 の値をしまう
        JUMP  L1
L2      次の命令
MAX     DC    9              ; 定数 9
A       DS    10             ; 配列 A[10]
```

d.　の補足: 配列 A、B、C があって、 C[I] = A[I] + B[I] を計算する。

```
        LAD   GR3,0          ; GR3 を変数 I として使う
L1      CPA   GR3,MAX        ;
        JPL   L2
        LD    GR4,A,GR3      ; A の GR3 番目の要素を GR4 に取り出す
        ADDA  GR4,B,GR3      ; GR4 に、B の GR3 番目の要素を加えて、GR4 にしまう
        ST    GR4,C,GR3      ; C の GR3 番目の要素に、GR4 の値をしまう
        JUMP  L1
L2      次の命令
MAX     DC    9              ; 定数 9
A       DS    10             ; 配列 A[10]
B       DS    10             ; 配列 B[10]
C       DS    10             ; 配列 C[10]
```

e.　省略

f.　ループの途中で、「もし値が 0 なら脱出」という条件分岐を加える。

```
        LAD   GR3,0          ; GR3 を変数 I として使う
L1      CPA   GR3,MAX        ;
        JPL   L2
```

```
        LD    GR4,A,GR3     ; A の GR3 番目の要素を GR4 に取り出す
        ST    GR4,A,GR3     ; A の GR3 番目の要素に GR4 の値をしまう
        LAD   GR5,0
        CPA   GR4,GR5       ; GR4 と GR5(=0)を比較する
        JZE   L2            ; もし 0 ならループを脱出する
        JUMP  L1
L2      次の命令
MAX     DC    999           ; 定数 999
A       DS    1000          ; 配列 A[1000]
B       DS    1000          ; 配列 B[1000]
```

途中で脱出する時、先に ST 命令をして A⇒B へのコピーを済ませてから脱出しているところがミソであるが、気持ちが悪いという人がいるかもしれない。それならば、ST する前に脱出するが、脱出先を L2 とせず、L3 としてそこで ST 命令を行い、そのあと L2 に続くようにすればよい。

≪情報処理技術者試験問題から類似・関連問題≫

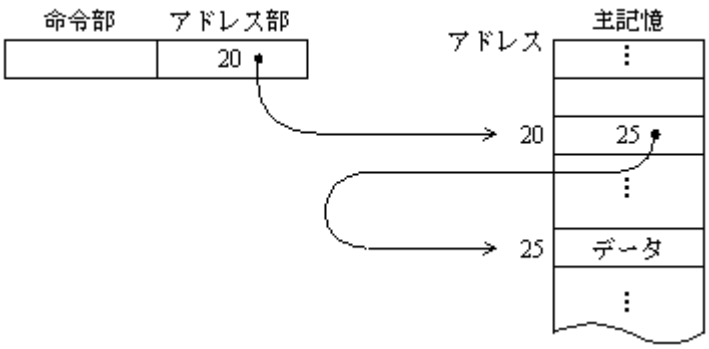
1） 主記憶へのアクセスを伴う演算命令を実行するとき，命令解読とオペランド読出しの間に行われる動作はどれか。（基本 20 春 18）

ア 実効アドレス計算　　イ 入出力装置起動
ウ 分岐アドレス計算　　エ 割込み発生

2） アドレス指定方式のうち，命令読出し後のメモリ参照を行わずにデータを取り出すものはどれか。（基本 16 春 17）

ア 間接アドレス　　イ 指標付きアドレス　　ウ 即値オペランド　　エ 直接アドレス

3） 主記憶のデータを図のように参照するアドレス指定方式はどれか。（基本 16 秋 18）



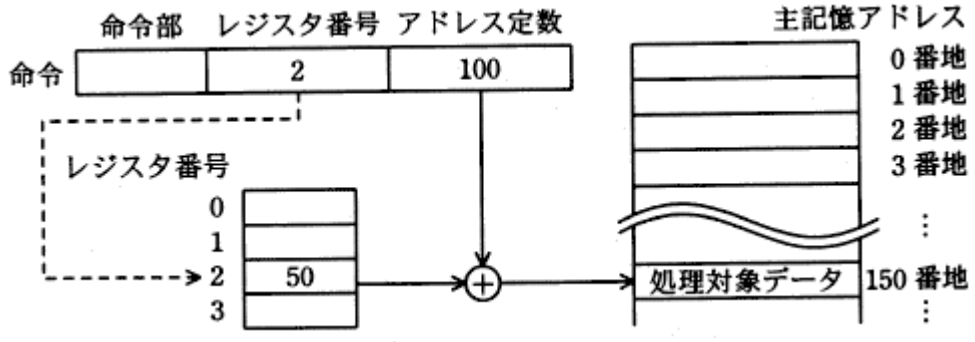
ア 間接アドレス指定　　イ 指標アドレス指定
ウ 相対アドレス指定　　エ 直接アドレス指定

4） インデックス修飾によってオペランドを指定する場合，表に示す値のときの実効アドレスはどれか（基本 17 秋 18）

インデックスレジスタの値　　10
命令語のアドレス部の値　　100
命令が格納されているアドレス　　1000

ア 110　　イ 1010　　ウ 1100　　エ 1110

5） 図に示すアドレス指定方式はどれか。（基本 19 秋 18）



ア　指標付きアドレス指定方式　　イ　相対アドレス指定方式

ウ　直接アドレス指定方式　　エ　レジスタ間接アドレス指定方式

6）　命令のオペランド部において，プログラムカウンタの値を基準とし，その値からの変位で実効アドレスを指定する方式はどれか。(基本 14 春 19)

ア　インデックスアドレス指定　　イ　絶対アドレス指定　　ウ　相対アドレス指定　　エ　ベースアドレス指定

≪解答≫

1）ア　　命令読出し⇒命令解読⇒命令実行⇒(PC=PC+1) なのだが、そのうち命令解読の後、命令実行の中で必要になる処理が幾つかある。具体的には、算術論理演算命令では、 ①オペランドの(実効)アドレスの計算 ⇒ ②オペランドの読み出し ⇒ ③実行(計算) ⇒ ④結果の書込み(格納) がある。 ロード命令やストア命令では③が実質的に空になる。 COMET の比較命令 CPA では、④で汎用レジスタに書かない(フラグレジスタに書くが、これは算術演算命令 SUBA でも同じである)。 また、ジャンプ系の命令では、オペランドが飛び先のアドレスであるが①②を行い、③④が無い代わりに PC(プログラムカウンタ)の値を飛び先アドレスに書き直す。この場合は当然(PC=PC+1)は行わない。

2）ウ　　3）ア　　4）ア　　5）ア　　6）ウ