情報科学実験 1
5. 教育用計算機 KUE-CHIP2(基礎編 2)

J2200071 齊藤 隆斗
実験グループ: A 班

実験実施日: 2024 年 5 月 9 日
レポート提出日: 2024 年 5 月 15 日
提出期限: 2024 年 5 月 16 日

1　実験で作成したプログラムとその説明

1.1　N 個の数の加算: 指定したアドレスから始まる N バイトの値の和を求めるプログラムを作成せよ.

入力
180: 14 23 31 42
190: 04

出力
1A0: AA

1.1.1　プログラム

ADDR	DATA	OPECODE
00	C0	EOR ACC, ACC
01	C9	EOR IX , IX
02	B7	ADD ACC, (IX+80) [:LP]
03	80	
04	BA	ADD IX , 1
05	01	
06	FD	CMP IX , (90)
07	90	
08	31	BNZ [:LP]
09	02	
0A	75	ST ACC, (A0)
0B	A0	
0C	0F	HLT

1.1.2　説明

プログラムは、ACC に入力データの値を (80) から (83) まで順番に足していくことで総和を計算するという流れになっている.
プログラムを詳細に見ていく. このプログラムでは、ACC は累積和を格納しておくために、IX は何回足し合わせたのかをカウントするために使われている. まず、ACC の値と IX の値を 0 で初期化をする. その後、ACC に (80) のデータを足し合わせる.1 回分のデータを足し合わせたので、IX の値を 1 増加させる.IX の値がまだ 04 になっていないため、同様の処理を IX が 04 になるまで繰り返す. 繰り返しを抜けたら、処理を停止する. これによって、与えられた 4 つの入力データを足すことができる.

1.2 フィボナッチ数列: {1, 1} に続くフィボナッチ数列の値を N 個求めるプログラムを作成せよ.

入力
180: 01 01
190: 0A (フィボナッチ数列の個数)

出力 (入力として与えた個数のフィボナッチ数列)
180: 01 01 02 03 05 08 0D 15 22 37

1.2.1 プログラム

ADDR	DATA	OPECODE
00	C0	EOR ACC, ACC
01	6A	LD IX , 2
02	02	
03	67	LD ACC, (IX+7F) [:LP]
04	7F	
05	B7	ADD ACC, (IX+7E)
06	7E	
07	77	ST ACC, (IX+80)
08	80	
09	BA	ADD IX , 1
0A	01	
0B	FD	CMP IX , (90)
0C	90	
0D	31	BNZ [:LP]
0E	03	
0F	0F	HLT

1.2.2 説明

プログラムは、すでにあるフィボナッチ数列の最後の 2 つの和を存在する数列の最後のアドレスの次のアドレスに格納するという操作を繰り返すことによってフィボナッチ数列を生成していくという流れである. プログラムについて見ていくと、ACC は足し算の結果を一時的に格納しておくために使用される. また、IX はその時点でフィボナッチ数列がいくつあるのかをカウントしてる. まず、ACC の値を初期化する. 次に IX の値を 2 で初期化する. これはすでにフィボナッチ数列 {1, 1} が存在するためであるからである. ここから繰り返しに入る. 一回目のループについて考える. まず、ACC に (81) のデータを格納する. 次に (80) のデータを

ACC に足し合わせる．足し合わせた ACC のデータを (82) に格納する．この時点でフィボナッチ数列は 1 つ増加しているため、IX の値を 1 増加させる．この時点でほしい長さのフィボナッチ数列が得ることができていれば処理を抜けることができるが、まだ IX の値は 3 であるからこの処理は再び繰り返される．この繰り返しを行っていき、入力として与えた長さの数列を得ることができればループを抜けて、処理を停止する．このようにしてフィボナッチ数列を生成するプログラムを実装した．

1.3 データ挿入: 昇順に整列されたデータに 1 バイトのデータを挿入する.

入力
180: 11 23 36 58 77 89 (整列済データ)
1A0: 42/05/A0 (挿入データ. 3 通り)
1B0: 06 (整列済データ長)

出力
180: 11 23 36 42 58 77 89

1.3.1 プログラム

ADDR	DATA	OPECODE
00	C0	EOR ACC, ACC
01	C9	EOR IX , IX
02	6D	LD IX , (B0)
03	B0	
04	65	LD ACC, (A0) [:LP]
05	A0	
06	AA	SUB IX , 1
07	01	
08	32	BZP [:SKIP]
09	0D	
0A	75	ST ACC, (80)
0B	80	
0C	0F	HLT
0D	20	RCF [:SKIP]
0E	87	SBC ACC, (IX+80)
0F	80	
10	3D	BC [:SKIP]
11	17	
12	65	LD ACC, (A0)

```
13      A0
14      77      ST  ACC,  (IX+81)
15      81
16      0F      HLT
17      67      LD  ACC,  (IX+80)  [:SKIP]
18      80
19      77      ST  ACC,  (IX+81)
1A      81
1B      30      BA  [:LP]
1C      04
```

1.3.2 説明

プログラムの流れは次のようになる. 整列済データの大きい方から順に挿入データと比較していき、対象のデータと挿入データを比較し、(対象のデータ) > (挿入データ) であるとき、対象のデータを一つ先のアドレスへ移す. 対象のデータと挿入データを比較し、(対象のデータ) ≤ (挿入データ) であるとき、比較したデータの直後に挿入データを挿入して、処理を停止する. ただし、挿入データが整列済データの最も小さいデータよりも小さいと判定された場合は整列済データの最も小さいデータの直前に挿入データを格納する. このようなフローでデータの挿入を実現した. プログラムのアイデアで理解したことについて述べる.CMP 命令であると符号付きで減算が行われてしまうため、補数による計算が行われる. しかし、89 は 2 進数で表現すると 10001001 となり、最上位ビットが符号を表すことを考えると正しく表現できていない. したがって、CMP 命令以外の符号なしで比較を行う命令を利用する必要がある. 今回は、符号なしで減算をする命令である SBC を用いた. SBC を利用することによって、符号なしの比較ができる. SBC による減算で、A と B を比較するとき、$A - B$ を行い、B の方が大きいとキャリーフラグが立ち、A の方が大きい、または等しい場合、キャリーフラグは立たない. よって、符号付きで計算する CMP のかわりに符号なしで計算する SBC によってキャリーフラグが立つか立たないかで大小の判定をおこなった.

1.4 挿入ソート: 挿入ソートによって N バイトのデータを昇順に整列する.

入力
180: 36 11 23 58 42 89 (未整列データ)
190: 06 (データ長)

出力 (整列済データ)
180: 11 23 36 42 58 89

作業領域
1A0: (整列済みの配列に追加するデータを一時的に格納)
1B0: (整列済みの配列の長さを一時的に格納)

1.4.1 プログラム

ADDR	DATA	OPECODE
00	6A	LD IX , 1
01	01	
02	7D	ST IX , (B0)
03	B0	
04	67	LD ACC, (IX+80) [:LP]
05	80	
06	75	ST ACC, (A0)
07	A0	
08	65	LD ACC, (A0) [:LP]
09	A0	
0A	AA	SUB IX , 1
0B	01	
0C	32	BZP [:SKIP]
0D	12	
0E	75	ST ACC, (80)
0F	80	
10	30	BA [:SKIP]
11	23	
12	20	RCF
13	87	SBC ACC, (IX+80)
14	80	
15	3D	BC [:SKIP]
16	1D	
17	65	LD ACC, (A0)
18	A0	
19	77	ST ACC, (IX+81)
1A	81	
1B	30	BA [:SKIP]
1C	23	
1D	67	LD ACC, (IX+80)
1E	80	
1F	77	ST ACC, (IX+81)
20	81	
21	30	BA [:LP]

```
22      08
23      6D      LD IX, (B0)
24      B0
25      BA      ADD IX, 1
26      01
27      7D      ST IX, (B0)
28      B0
29      FD      CMP IX, (90)
2A      90
2B      31      BNZ [:LP]
2C      04
2D      0F      HLT
```

1.4.2 説明

このプログラムにおいて、データ挿入のプログラムをサブルーチンとして使用した. このサブルーチンを使用してアドレスの小さい方から挿入ソートによってソートしていき、ソート済のデータ配列を 1 つずつ大きくしていくという流れプログラム作成した. このプログラムでは、作業領域として (A0) と (B0) を利用する. これらはそれぞれ、整列済みの配列に追加するデータを一時的に格納するため、整列済みの配列の長さを一時的に格納するために用いる. プログラムについて見ていく. まず、IX と B0 に、1 を格納する. これは (80) にデータがあるためこれを一つのソート済みデータとも考えられるからである. 次に、以下の処理を繰り返す. ソート済データの直後のデータを ACC と A0 に格納する. これはデータ挿入のサブルーチンで必要な情報である. ここで、先程のデータ挿入サブルーチンを使用し、データ挿入をおこなうことでソート済データの長さを一つ大きくする. この際、サブルーチンで、処理を停止する命令のかわりに指定された命令に飛ぶようにプログラムを変更していることに注意する. サブルーチン後の処理は、まず、整列済データが 1 増加したため、IX と B0 の値を 1 増加させる. この時点で IX の値がデータ長でないならば処理を繰り返す. IX の値がデータ長となり、すべてのデータが整列済となれば、繰り返しを抜け、プログラムを停止する. このように、データ挿入プログラムと IX を利用した繰り返しによって挿入ソートを実装した.

2 考察課題

1. 課題 2 の 1 バイトのデータ挿入プログラムを拡張し、2 バイトで表現されるデータの挿入プログラムを作成せよ.

入力

180: 1234 2481 4721 6382 7837 A381 (整列済データ)
1A0: 3521/0304/BC07 (挿入データ. 3 通り)
1B0: 06 (整列済データ長)

出力

180: 1234 2481 3521 4721 6382 7837 A381

作業領域
1C0: (IX の値を一時的に保存する際に使用する.)

2.0.1 プログラム

ADDR	DATA	OPECODE
00	C0	EOR ACC, ACC
01	C9	EOR IX , IX
02	6D	LD IX , (B0)
03	B0	
04	AA	SUB IX , 1 [:LP]
05	01	
06	32	BZP [:SKIP]
07	11	
08	65	LD ACC, (A1)
09	A1	
0A	75	ST ACC, (81)
0B	81	
0C	65	LD ACC, (A0)
0D	A0	
0E	75	ST ACC, (80)
0F	80	
10	0F	HLT
11	20	RCF [:SKIP]
12	7D	ST IX , C0
13	C0	
14	B9	ADD IX , IX
15	65	LD ACC, (A1)
16	A1	
17	87	SBC ACC, (IX+81)
18	81	
19	65	LD ACC, (A0)
1A	A0	
1B	87	SBC ACC, (IX+80)
1C	80	

```
1D    6D    LD IX, (C0)
1E    C0
1F    3D    BC [:SKIP]
20    2B
21    B9    ADD IX, IX
22    65    LD ACC, (A1)
23    A1
24    77    ST ACC, (IX+83)
25    83
26    65    LD ACC, (A0)
27    A0
28    77    ST ACC, (IX+82)
29    82
2A    0F    HLT
2B    7D    ST IX, (C0) [:SKIP]
2C    C0
2D    B9    ADD IX, IX
2E    67    LD ACC, (IX+81)
2F    81
30    77    ST ACC, (IX+83)
31    83
32    67    LD ACC, (IX+80)
33    80
34    77    ST ACC, (IX+82)
35    82
36    6D    LD IX, (CO)
37    C0
38    30    BA [:LP]
39    04
```

2.0.2 説明

プログラムのおおまかな流れは課題 2 と同様である. 変更した点は、挿入によってデータをずらす際に 2 バイトのデータを 2 バイトずつずらしたことと、SBC 命令によって符号なし減算を考える際に下位バイトから減算を行い、この減算によるキャリーフラグを考慮して上位バイトの減算を行ったことの 2 つである.

2. N の階乗で、値が 2 バイトで収まるものを全て出力するプログラムを作成せよ.

入力

1A0: 0001

出力

1A0: 0001 0001 0002 0006 0018 0078 02D0 13B0 9D80

作業領域

170: (被乗数を一時的に保存)
172: (乗数を一時的に保存)
180: (あと何回足すべきかをカウント)
190: (乗算の際に上位バイトの計算結果を一時的に保存)
191: (乗算の際に下位バイトの計算結果を一時的に保存)

2.0.3 プログラム

ADDR	DATA	OPECODE
00	C0	EOR ACC, ACC
01	75	ST ACC, (90)
02	90	
03	75	ST ACC, (91)
04	91	
05	6A	LD IX, 1
06	01	
07	7D	ST IX, (72) [:LP]
08	72	
09	7D	ST IX, (80)
0A	80	
0B	B9	ADD IX, IX
0C	67	LD ACC, (IX+9F)
0D	9F	
0E	75	ST ACC, (71)
0F	71	
10	67	LD ACC, (IX+9E)
11	9E	
12	75	ST ACC, (70)

```
13      70
14      C0      EOR ACC, ACC
15      75      ST ACC, (90)
16      90
17      75      ST ACC, (91)
18      91
19      C0      EOR ACC, ACC [:LP]
1A      65      LD ACC, (71)
1B      71
1C      95      ADC ACC, (91)
1D      91
1E      75      ST ACC, (91)
1F      91
20      65      LD ACC, (70)
21      70
22      95      ADC ACC, (90)
23      90
24      75      ST ACC, (90)
25      90
26      3D      BC [:SKIP]
27      41
28      6D      LD IX, (80)
29      80
2A      AA      SUB IX, 1
2B      01
2C      7D      ST IX, (80)
2D      80
2E      31      BNZ [:LP]
2F      19
30      6D      LD IX, (72)
31      72
32      B9      ADD IX, IX
33      65      LD ACC, (91)
34      91
35      77      ST ACC, (IX+A1)
36      A1
37      65      LD ACC, (90)
```

```
38      90
39      77      ST  ACC, (IX+A0)
3A      A0
3B      6D      LD  IX , (72)
3C      72
3D      BA      ADD IX , 1
3E      01
3F      30      BA  [:LP]
40      07
41      0F      HLT [:SKIP]
```

2.0.4 説明

出力は (A0) からのデータ領域に格納していく. $0! = 1$ であるから、最初の 2 バイトのデータは 0001 である. 次に、$1! = 1 \times 0!$ であるから、1 と 0! をかけた結果を次の 2 バイトに格納する. 次に、$2! = 2 \times 1!$ であるから、2 と 1! をかけた結果を次の 2 バイトに格納する. 次に、$3! = 3 \times 2!$ であるから、3 と 2! をかけた結果を次の 2 バイトに格納する. このように、$N!$ を求める際にすでに計算した $(N-1)!$ を格納してある 2 バイト前のデータを利用してデータ領域を 2 バイトずつ進めていく. 乗算のプログラムは KUE-CHIP2[基礎編 1] で扱った例題を参考にした. ここで、2 バイトを超える階乗の結果が出てきた時に処理を終了したいので、乗算のプログラム内で必要となる ADC を利用した. 上位バイトにおいて、この ADC による加算でオーバーフローが発生、即ちキャリーフラグが立ったら処理を終了するように実装した.