情報科学実験 1
4. 教育用計算機 KUE-CHIP2(基礎編 1)

J2200071 齊藤 隆斗
実験グループ: A 班

実験実施日: 2024 年 4 月 18 日
レポート提出日: 2024 年 4 月 25 日
提出期限: 2024 年 4 月 25 日

1　実験で作成したプログラムとその説明

1.1　ACC の値を IX の値回足す、即ち $ACC \times IX \rightarrow ACC$ を行うプログラム

1.1.1　プログラム

ADDR	DATA	OPECODE
00	75	ST ACC, (03H)
01	03	
02	C0	EOR ACC, ACC
03	B5	ADD ACC, (03H)
04	03	
05	AA	SUB IX , 1
06	01	
07	31	BNZ 03H
08	03	
09	0F	HLT

1.1.2　説明

大まかな処理の流れは、入力を ACC と IX に格納して、出力を ACC に格納するという形になっている. プログラムのアイディアは、0 に ACC を IX 回足すことによって、ACC × IX → ACC を実現するという発想である. まず、ACC を、出力の際に使用することになるので、入力として格納した ACC のデータをメモリのデータ領域の 03H に退避しておく. そして、ACC のデータを 0 に初期化する. その後、ACC に、退避しておいたデータを足したあとに IX の値を 1 だけ減少させる. IX の値が 0 になるまで繰り返すことでこの操作を IX 回繰り返すことができる.

1.2　メモリに格納してある 3 つのデータを加算するプログラム

1.2.1　プログラム

ADDR	DATA	OPECODE
00	C0	EOR ACC, ACC
01	B5	ADD ACC, (00H)
02	00	
03	B5	ADD ACC, (01H)
04	01	
05	B5	ADD ACC, (02H)
06	02	
07	0F	HLT

1.2.2 説明

メモリのデータ領域の 00H, 01H, 02H に入力を格納して、出力 (足した結果) を ACC に格納するという流れとなっている. ACC に累積和を格納し、それを更新していくというアイディアである. まず、ACC の値を 0 に初期化しておく. そして、ACC と 00H に格納したデータを足したものを ACC に再び格納する. そして同じように、操作後の ACC と 01H に格納したデータを足したものを ACC に格納する. 03H に対しても同様の操作を行えば、ACC には、3 つのデータ領域 00H, 01H, 02H に格納されたデータを足したものが格納されていることになる.

1.3 1 から N までの整数の加算を行うプログラム

1.3.1 プログラム

ADDR	DATA		OPECODE
00	6C		LD IX, [80H]
01	80		
02	62		LD ACC, 0
03	00		
04	B1	LOOP:	ADD ACC, IX
05	AA		SUB IX, 1
06	01		
07	33		BP LOOP
08	04		
09	74		ST ACC, [81H]
0A	81		
0B	0F		HLT

1.3.2 説明

メモリのデータ領域の 80H に入力を格納し、81H に出力を格納するという流れである. アイディアとしては、ACC に N, N-1,. . ., 2, 1 を次々に足していくことで、N の総和を実現している. まず、IX に N を格納し、ACC に 0 を格納し、初期化を行う. その後、ACC と IX を足したものを ACC に格納し、IX の値を 1 だけ減少させる. IX には n-1 が格納されることになる. これを IX の値が 1 以上である間だけ繰り返す. 最後の 1 を ACC に足した後、IX の値は 0 になるため繰り返しが終了する. これによって、まず ACC に N を足し、その後 N-1 を足し、. . .、その後 2 を足し、その後に 1 を足して処理を終了するという操作を実装している. この操作を終えた後の ACC には N の総和が格納されており、この ACC のデータをメモリのデータ領域の 81H に格納してプログラムを終了する.

1.4 多倍長の加算を行うプログラム

1.4.1 プログラム

ADDR	DATA		OPECODE
00	6C		LD IX , [C0H]
01	C0		
02	20		RCF
03	66	LOOP:	LD ACC, [IX+7FH]
04	7F		
05	96		ADC ACC, [IX+8FH]
06	8F		
07	76		ST ACC, [IX+9FH]
08	9F		
09	AA		SUB IX , 1
0A	01		
0B	33		BP LOOP
0C	03		
0D	0F		HLT

1.4.2 説明

インデックス修飾とキャリーフラグを使用することで、多倍長の加算を実現する. このプログラム例では 4Byte 同士の足し算行っているので説明もこの場合について行う. メモリのプログラム領域の C0 に何 Byte での加算を行うかの入力データを、80H, 81H, 82H, 83H に加算で使用される被演算子の入力データを、90H, 91H, 92H, 93H にもう一方の入力データを格納し、これらを足し算した結果を A0H, A1H, A2H, A3H に格納するという流れである. ここで、より小さい番地に、演算に使用するデータのより上位のビットが格納されることに注意する. 具体的には、80H を MSB、83H を LSB として加算を行うということである. プログラムのアイディアは、まず、データの下位バイトから 1Byte ずつ足し算を行っていく. そして、桁が溢れた場合はキャリーフラグを 1 にセットし、次回の上位ビットの足し算で追加的にこれを足す. これを繰り返すことによって、データ全体での足し算を実現するというものである. ここで、プログラムについて見ていく. まず、何バイトの加算を行うかのデータを IX に格納する. この例では 4Byte であるから、IX には 4 が格納される. そしてキャリーフラグをリセット (0 にセット) する. 最初の 1Byte では繰り上がりは無関係であるからである. 次に、83H のデータを ACC に格納し、ACC + (93H のデータ) + (キャリーフラグ) の結果を ACC に再代入し、ACC のデータを 93H に格納する. これで、期待する出力の最初の 1Byte を得ることができる. また、この加算を行った際に桁が溢れた場合はキャリーフラグを 1 に、そうでない場合は 0 にセットする. その後、IX の値を 1 減少させる. この操作が終了したら、82H のデータを ACC に格納し、ACC + (92H のデータ) + (キャリーフラグ) の結果を ACC に再代入し、ACC のデータを 92H に格納する. この操作を繰り返し、IX の値が 0 になったならば繰り替えしを終了し、プログラムを停止する. 繰り返しは、ACC(80H のデータ) + (90H のデータ) + (キャリーフラグ) の加算が終了した直後に停止する.

1.5 1 バイトの乗算を行うプログラム

1.5.1 プログラム

ADDR	DATA		OPECODE
00	C0		EOR ACC, ACC
01	74		ST ACC, [82H]
02	82		
03	74		ST ACC, [83H]
04	83		
05	74		ST ACC, [F0H]
06	F0		
07	64		LD ACC, [81H]
08	81		
09	74		ST ACC, [F1H]
0A	F1		
0B	6C		LD IX, [80H]
0C	80		
0D	4A	LOOP:	SRL IX
0E	35		BNC SKIP
0F	1D		
10	20		RCF
11	64		LD ACC, [83H]
12	83		
13	94		ADC ACC, [F1H]
14	F1		
15	74		ST ACC, [83H]
16	83		
17	64		LD ACC, [82H]
18	82		
19	94		ADC ACC, [F0H]
1A	F0		
1B	74		ST ACC, [82H]
1C	82		
1D	FA	SKIP:	CMP IX, 0
1E	00		
1F	39		BZ FIN
20	2D		
21	64		LD ACC, [F1H]
22	F1		
23	41		SLA ACC

```
24      74              ST  ACC,  [F1H]
25      F1
26      64              LD  ACC,  [F0H]
27      F0
28      45              RLA ACC
29      74              ST  ACC,  [F0H]
2A      F0
2B      30              BA  LOOP
2C      0D
2D      0F      FIN :   HLT
```

1.5.2 説明

メモリのプログラム領域の 80H, 81H に入力データ (乗算の被演算子) が格納されて、出力データ (乗算の結果) を 82H, 83H の 2Byte に格納されるという流れである. この出力データについても、より小さい番地にデータのより上位のビットが格納されることに注意する. このプログラムのアイディアは、乗算の筆算を 2 進数で行うというところにある. 例として、1001×1010 について考える.

$$\begin{aligned} 1001 \times 1010 &= (1 \times 1000 + 0 \times 100 + 0 \times 10 + 1 \times 1) \times 1010 \\ &= 1 \times 1010 \times 1000 + 0 \times 1010 \times 100 + 0 \times 1010 \times 10 + 1 \times 1010 \times 1 \\ &= 1 \times 1010000 + 0 \times 101000 + 0 \times 10100 + 1 \times 1010 \end{aligned}$$

となる. これを見ると、1001 の 1 が立っている桁それぞれについて、それに対応する左シフトを 1010 に対して行ってそれを足していくことで掛け算を実装することができる. ここでプログラムについて見ていく. まず、メモリのプログラム領域の 82H, 83H, F0 を 0 に、そして 81H のデータを F1 に格納し、80H のデータを IX に格納し、初期化する. そして次の処理をプログラムが停止するまでくりかえす.

まず、IX に対して SRL を行って、1 桁目の値が溢れてキャリーフラグがセットされる. その後、BNC によって 1 桁目が 0 か 1 かを調べる. これは出力に対して加算を行う必要があるのかを判定するのに必要である.

(i) キャリーフラグが 0 であったとき出力に対して加算は行わないので、次の桁の加算に備えて F0H, F1H を左にシフトしておく. しかし、この時点ですべての桁が 0 であったらこれ以降加算は行わないということなのでループを抜けてプログラムを停止することに注意する. 左シフトを行うために、まず、F1H を SLA によって左シフトを行う. このときに溢れた桁はキャリーフラグとして保存される. 次に F0H を RLA によって左シフトする. このときにシフトされた F0H の 1 桁目に先程保存したキャリーフラグを割り当てる. これによって、全体として左シフトを行うことができる.

(ii) キャリーフラグが 1 であったとき出力に対して加算を行う必要がある. この時点でシフトを行う必要はない. なぜならば 1 回目の繰り返しではそもそも 1 桁目についての加算を行えば良いのでシフトを必要としない. 2 回目以降の繰り返しでは、すでに 1 回目の繰り返しによって、シフトが行われているからである. よって、素直に出力とシフトされたデータの加算を行えば良い. まず、83H のデータと F1H のデータを加算する. このときに桁が溢れた場合、キャリーフラグが 1 にセットされる. 次に 82H と F0H のデータの加算を行う. このとき、先程のキャリーフラグも加算されることに注意する. これで加算は終了したので、この次の繰り返しに向けてシフトを行う必要がある. これは (i) と同様に行えば良い.

2　考察課題

2. 例題の乗算プログラムを参考にして、1 バイトの除算プログラムを作成し、解説せよ.

2.0.1　プログラム

ADDR	DATA	OPECODE
00	75	ST ACC, (03H)
01	03	
02	C0	EOR ACC, ACC
03	A2	SUB ACC, 1
04	01	
05	B2	ADD ACC, 1
06	01	
07	AD	SUB IX, (03H)
08	03	
09	32	BZP 05H
0A	05	
0B	BD	ADD IX, (03H)
0C	03	
0D	0F	HLT

2.0.2　説明

IX ÷ ACC → ACC ··· IX を行うプログラムを実装した. プログラムの大まかな流れは、ACC に被除数、IX に除数を入力して、ACC に商、IX に余りを出力する. プログラムのアイディアは被除数から除数を何回だけ引くことができるのかということに注目することで商を求め、余りは被除数からその商だけ除数を引いたときの数のことであると見ることである. プログラムについて見ていくと、まず、ACC に格納した除数をメモリのデータ領域に退避させておき、ACC を 0 にセットする. その後 ACC の値から 1 を引くことで ACC を-1 にセットする. そして、その後 1 を加えて ACC を 0 にする. これは繰り返しの際に SUB という命令でフラグの状態を利用したいからである. そして、IX から、除数を引いて、まだ負でないなら ACC の値を 1 増加させて、IX から除数を再び引く. 負でないなら, ... というようにこの操作を繰り返す. IX が負になったら、その (ACC に格納された値 +1) × (除数) > (被除数) となってしまうので繰り返しを停止する. 繰り返し後に IX にメモリ領域の 03H のデータを足しているのはそのままだと余分に 1 回だけ引かれてしまっているのでもう一度 IX を足して正しい余りを出力するためである.

4. SBC 命令の動作について説明し、それを用いて多倍長の引き算プログラムを作成せよ. ただし、計算結果が負になる場合は考えなくても良い.

2.0.3　プログラム

ADDR	DATA		OPECODE
00	6C		LD IX , [C0H]
01	C0		
02	20		RCF
03	66	LOOP:	LD ACC, [IX+7FH]
04	7F		
05	86		SBC ACC, [IX+8FH]
06	8F		
07	76		ST ACC, [IX+9FH]
08	9F		
09	AA		SUB IX , 1
0A	01		
0B	33		BP LOOP
0C	03		
0D	0F		HLT

2.0.4 SBC 命令の説明

SBC 命令は、SBC A B のとき、 (A) − (B) − CF → A を行う. このとき、ボローが発生したときに CF が立つ. ここで、ボローとは存在しない桁からの繰り下がりを意味している. 上の処理を見てわかるように、この CF が SBC 命令の減算にも影響が及ぶ. この機能を利用して引き算における繰り下がりを実装する.

2.0.5 プログラムの説明

プログラムの流れとしては、何バイトのデータを減算するかのデータと実際に減算をする際に使用するデータを入力として受け取り、減算した結果を出力する. プログラムのアイディアとしては、大まかにはデータの下位バイトから減算を行っていき、SBC を利用して、ボローが発生した場合に次回の上位バイトの減算の際に被減数から 1 を引くことで繰り下がりを実装している. ここでプログラムについて見ていく. まず何バイトの引き算を行うかのデータを入力する. その後キャリーフラグを 0 にセットする. これは最初の 1 バイトの引き算では下位バイトからの繰り下がりは考慮しなくても良いからである. この後次の操作を繰り返す. ACC に被減数のデータの対応するバイトを ACC に格納する.IX の値が繰り返しの度に 1 つずつ減っていくので格納するデータも繰り返しの度に下位バイトから上位バイトへとズレていく. 次に ACC の値から減数のデータの対応するバイトを減算する. 被減数のデータも同様に下位バイトから上位バイトへとズレていく. このとき 1 回目の繰り返しの場合、キャリーフラグはリセットされているが 2 回目以降は繰り下がりが発生していた場合、直前のバイトの繰り下がりも考慮して更に 1 を引く必要がある. この減算の際、(被減数-CF) よりも減数の方が大きい場合、繰り下がりが発生するためボローが発生する. よって、CF に 1 がセットされる. これは次の繰り返しの際に次の減数のバイトから 1 を引くかどうかに影響を与える. この後、出力データとして ACC の値が格納され、IX から 1 が引かれる.IX が正である間は繰り返しこの処理を繰り返す. 即ち、IX が 1 であるときは処理を実行するので 80H と 90H を減算のデータとして利用し、出力を A0H に格納するため期待通りの繰り返しの挙動となっている. 結果として、N バイトのとき、A0H, ..., (N-1)+A0H に減算の結果が出

力される.