電気電子物理工学実験Ⅲ
マイクロプロセッサ

課題: 二進化十進数(BCD)乗算

二進化十進数とは

十進数の1桁は0から9までの値となり、4ビット二進数で表せる

1バイト8ビットの下位4ビットに十進数の下位桁、上位4ビットに十進数の上位桁を入れて、1バイトで2桁の十進数を表す

・・・二進化十進数(Binary Coded Decimal, BCD)

例　十進数 $49_{(10)}$ ⇒ そのまま二進数にすると $00110001_{(2)} = 31_{(BCD)}$

$4_{(10)}$　$9_{(10)}$

0	1	0	0	1	0	0	1

⇒ $01001001_{(2)}$　二進数

$73_{(10)}$　十進数

$49_{(BCD)}$　BCD

★16進数とは

4ビット二進数は値が$0_{(10)}$以上$15_{(10)}$以下となり、$0_{(10)}$から$9_{(10)}$までは数字、$10_{(10)}$から$15_{(10)}$までをアルファベットAからFで表すもの

8ビット二進数は2桁16進数で表せる

二進化十進数の加算

1桁の加算結果が10以上になる場合あり

➡ 上位桁(左隣桁)への桁上げと、9以下の値へ修正が必要

十進数の加算

	15
+	28
	43

BCD数の加算

	0	0	0	1	0	1	0	1
+	0	0	1	0	1	0	0	0
	0	0	1	1	1	1	0	1

$=15_{(BCD)}$

$=28_{(BCD)}$

$=3D_{(BCD)}$

$10_{(10)}$以上

10を減じて、上位桁を1増やす

0	1	0	0	0	0	1	1	$=43_{(BCD)}$

★なお、「10を減じて、上位桁を1増やす」のは-10+16=+6なので、6を加算すれば同じこと

さらに注意

1桁の加算結果が9以下であっても、
上位桁に桁上げしている場合があるので注意

十進数の加算

	19
+	28
	47

BCD数の加算

	0	0	0	1	1	0	0	1
+	0	0	1	0	1	0	0	0
	0	1	0	0	0	0	0	1

$=19_{(BCD)}$
$=28_{(BCD)}$
$=41_{(BCD)}$

9以下

1桁の加算結果が16以上のとき、二進数加算で十進数上位桁(二進数5ビット目)に桁上げが起こる

➡ 本来は10以上で桁上げすべきだが、16で桁上げしているため、結果が6だけ小さな値となっている ⇒ 6を加算

二進化十進数の加算のまとめ

1桁の加算結果について

- 下位桁(下位4ビット)が10以上の場合

  ⇒ 下位桁に6を加算

- 下位桁(下位4ビット)から上位桁(5ビット目)に桁上げがある場合

  ⇒ 下位桁に6を加算

⇓

下位桁のみの和が$10_{(10)}$以上ならば6を加算

十進数和　二進数和(16進)　補正

十進数和	二進数和(16進)	補正
0	00	不要
1	01	不要
2	02	不要
3	03	不要
4	04	不要
5	05	不要
6	06	不要
7	07	不要
8	08	不要
9	09	不要

十進数和	二進数和(16進)	補正
10	0A	+6
11	0B	+6
12	0C	+6
13	0D	+6
14	0E	+6
15	0F	+6
16	10	+6
17	11	+6
18	12	+6

二進化十進数の加算方法

① 下位桁だけを取り出し、加算実行
② 加算結果が10以上のとき、6を加算
③ 上位桁についても(①の結果を含めて)同様に加算

	19
+	28
	47

	0	0	0	1	1	0	0	1
	0	0	1	0	1	0	0	0

$=19_{(BCD)}$
$=28_{(BCD)}$

①

	0	0	0	0	1	0	0	1
+	0	0	0	0	1	0	0	0
	0	0	0	1	0	0	0	1

②

+	0	0	0	0	0	1	1	0
	0	0	0	1	0	1	1	1

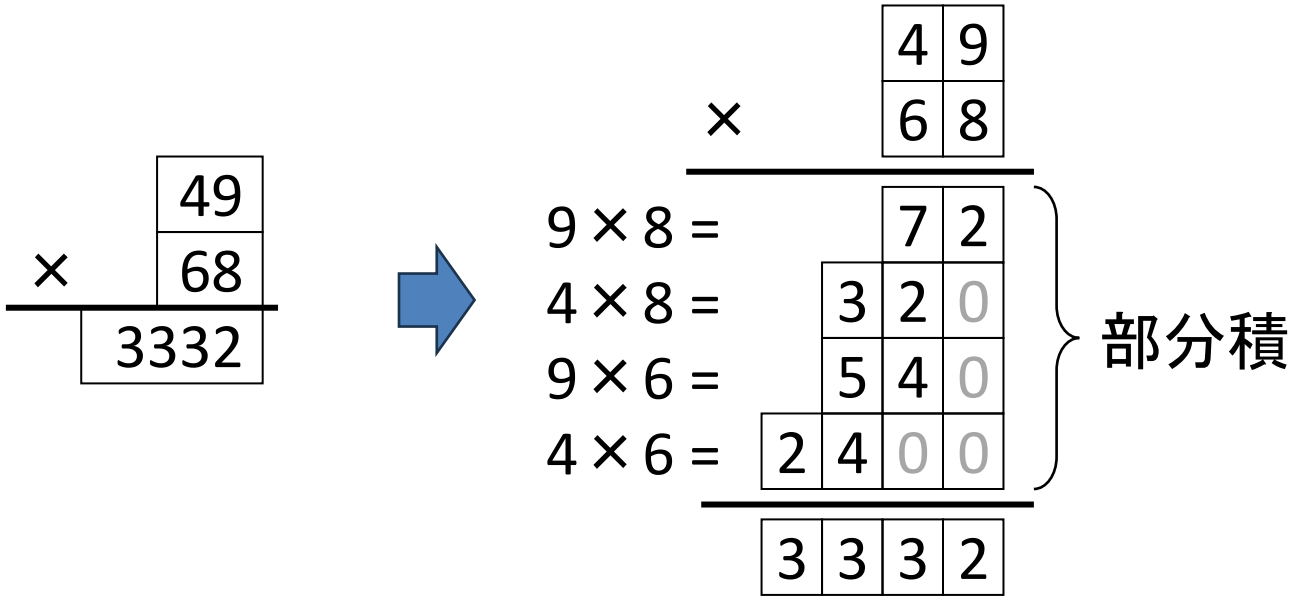
	0	0	0	1	0	1	1	1
+	0	0	0	1	0	0	0	0
	0	0	0	1	0	0	1	1	1
+	0	0	1	0	0	0	0	0
③	0	0	1	0	0	0	1	1	1

$=47_{(BCD)}$

★さらに左隣桁に桁上げする可能性を考慮 (97+28など)
③の加算実行後にCフラグを確認

二進化十進数の乗算

十進数の1桁ごとに乗算を実施・・・積は2桁
桁重みを考慮して桁を合わせて部分積を加算



十進数の1桁どうしの乗算

0×0 = 0
〜
9×9 = 81

} 二進数として乗算した結果

0	0	0	0	0	0	0	0

〜 = 0

0	1	0	1	0	0	0	1

$= 81_{(10)}$

積(二進数)の上位桁(上位4ビット)は0〜5の6通りだけ

➡ 上位桁の値に応じて修正

0	0	0	0	0	0	0	0	= 0
0	0	0	1	0	0	0	0	$= 16_{(10)}$
0	0	1	0	0	0	0	0	$= 32_{(10)}$
0	0	1	1	0	0	0	0	$= 48_{(10)}$
0	1	0	0	0	0	0	0	$= 64_{(10)}$
0	1	0	1	0	0	0	0	$= 80_{(10)}$

➡ 置換

0	0	0	0	0	0	0	0	= 0
0	0	0	1	0	1	1	0	$= 16_{(BCD)}$
0	0	1	1	0	0	1	0	$= 32_{(BCD)}$
0	1	0	0	1	0	0	0	$= 48_{(BCD)}$
0	1	1	0	0	1	0	0	$= 64_{(BCD)}$
1	0	0	0	0	0	0	0	$= 80_{(BCD)}$

➡ その後、下位桁を加算し、BCDに修正する

十進数の1桁どうしの乗算

例　6×5 = 30

			0	1	1	0	= 6
×			0	1	0	1	= 5
0	0	0	1	1	1	1	0	= $30_{(10)}$

BCDでは$1E_{(BCD)}$ ➡ $30_{(BCD)}$ に直す

① 上位4ビットが'0001' ➡ $16_{(BCD)}$

	0	0	0	1	0	0	0	0	= $16_{(10)}$
➡	0	0	0	1	0	1	1	0	= $16_{(BCD)}$

② 下位4ビットを補正

	0	0	0	0	1	1	1	0	= $14_{(10)}$ ≧ $10_{(10)}$
+	0	0	0	0	0	1	1	0	= 6
	0	0	0	1	0	1	0	0	= $14_{(BCD)}$

③ ①と②の下位4ビットを加算

	0	0	0	0	0	1	0	0	
+	0	0	0	0	0	1	1	0	= 6
	0	0	0	0	1	0	1	0	

④ 下位4ビットを再度補正

	0	0	0	0	1	0	1	0	≧ $10_{(10)}$
+	0	0	0	0	0	1	1	0	= 6
	0	0	0	1	0	0	0	0	= $10_{(BCD)}$

⑤ ②の上位4ビットを加算

	0	0	0	1	0	0	0	0	
+	0	0	0	1	0	0	0	0	
	0	0	1	0	0	0	0	0	

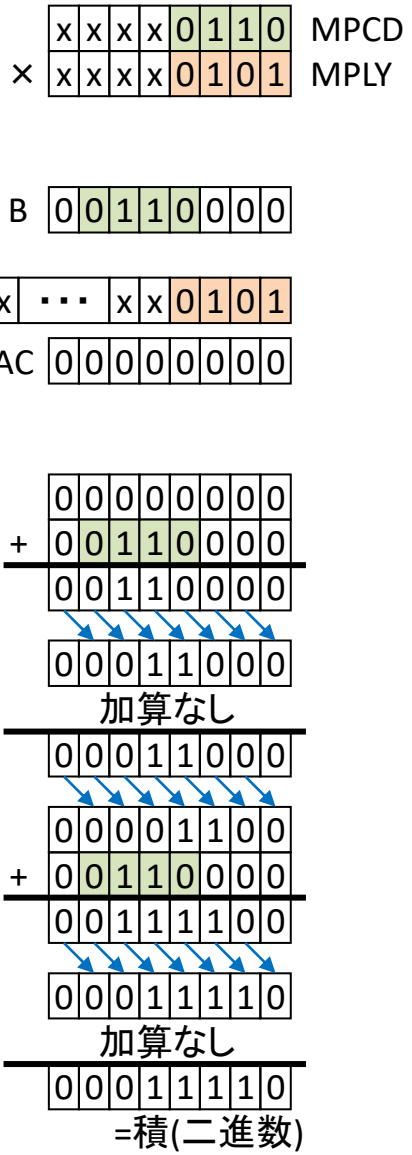
⑥ ①の上位4ビットを加算

	0	0	1	0	0	0	0	0	
+	0	0	0	1	0	0	0	0	
	0	0	1	1	0	0	0	0	= $30_{(BCD)}$

十進数の1桁どうしの乗算

```
MUL:    LD      AC, MPCD
        AND     AC, $0F     #被乗数MPCD(下位4ビットのみ使用)
        SLR     AC
        SLR     AC
        SLR     AC
        LD      B, AC       #Bレジスタ = 被乗数(3ビット左シフト)
        LD      AC, MPLY
        LD      WA, AC      #WAレジスタ = 乗数(MPLY)
        XOR     AC, AC      #ACレジスタ = 積、0に初期化

M0:     SRR     WA          #MPLYを右にシフトする
        JNC     M1
        ADD     AC, B       #Cフラグが1ならば、被乗数を加える
M1:     SRR     AC          #部分積和を右にシフト
        SRR     WA          #MPLYを右にシフト
        JNC     M2
        ADD     AC, B       #Cフラグが1ならば、被乗数を加える
M2:     SRR     AC          #部分積和を右にシフト
        SRR     WA          #MPLYを右にシフト
        JNC     M3
        ADD     AC, B       #Cフラグが1ならば、被乗数を加える
M3:     SRR     AC          #部分積和を右にシフト
        SRR     WA          #MPLYを右にシフト
        JNC     M4
        ADD     AC, B       #Cフラグが1ならば、被乗数を加える
M4:     LD      P0, AC      #ACレジスタ=積(二進数)、メモリ(P0)に保存
        #ここに来たときACレジスタは積(二進数)
```



#BCD補正

```
        AND     AC, $F0     #下位4ビットを0にする
        CPR     AC, $00     #上位4ビットを0000と比較(全8ビットを00000000と比較)
        JZ      M20         # 0000に等しい場合は変換不要
M5:     CPR     AC, $10     #上位4ビットを0001と比較(全8ビットを00010000と比較)
        JNZ     M6
        LD      AC, $16     # 0001に等しい場合は16(BCD)に変換
        JMP     M20
M6:     CPR     AC, $20     #上位4ビットを0010と比較(全8ビットを00100000と比較)
        JNZ     M7
        LD      AC, $32     # 0010に等しい場合は32(BCD)に変換
        JMP     M20
M7:     #途中省略
        #ここに来たときACレジスタは積(二進数)の上位桁の変換結果
```

8ページ ①

#次ページのラベルM20へ続く

```
M20:    LD      PH, AC      #ACレジスタ=上位桁変換結果、メモリ(PH)に保存
        LD      AC, P0      # P0: 積(二進数)
        AND     AC, $0F     #ACレジスタ=積(二進数)の下位4ビット
        CPR     AC, 10      # AC<10ならば以下の加算をスキップ
        JC      M21
        ADD     AC, 6       #10以上ならば6を加算
M21:    LD      P0, AC      # P0: 積(二進数)の下位4ビットのBCD補正結果
        AND     AC, $0F
        LD      B, AC       #Bレジスタ=下位4ビット補正結果の下位桁
        LD      AC, PH
        AND     AC, $0F     #ACレジスタ=上位桁変換結果の下位4ビット
        ADD     AC, B       #下位4ビットどうしを加算
        CPR     AC, 10      # AC<10ならば以下の加算をスキップ
        JC      M22
        ADD     AC, 6       #10以上ならば6を加算
M22:    LD      B, AC       #Bレジスタ=下位桁BCD補正結果
        LD      AC, P0
        AND     AC, $F0     #ACレジスタ= (2) の結果の上位4ビット
        ADD     AC, B
        LD      B, AC
        LD      AC, PH
        AND     AC, $F0     #ACレジスタ= (1) の結果の上位4ビット
        ADD     AC, B       #下位桁補正結果と加算
        #ここに来たときACレジスタは積(BCD)

MPCD:   db      0
MPLY:   db      0
PH:     db      0
PL:     db      0
P0:     db      0
```

8ページ

2

3

4

5

6

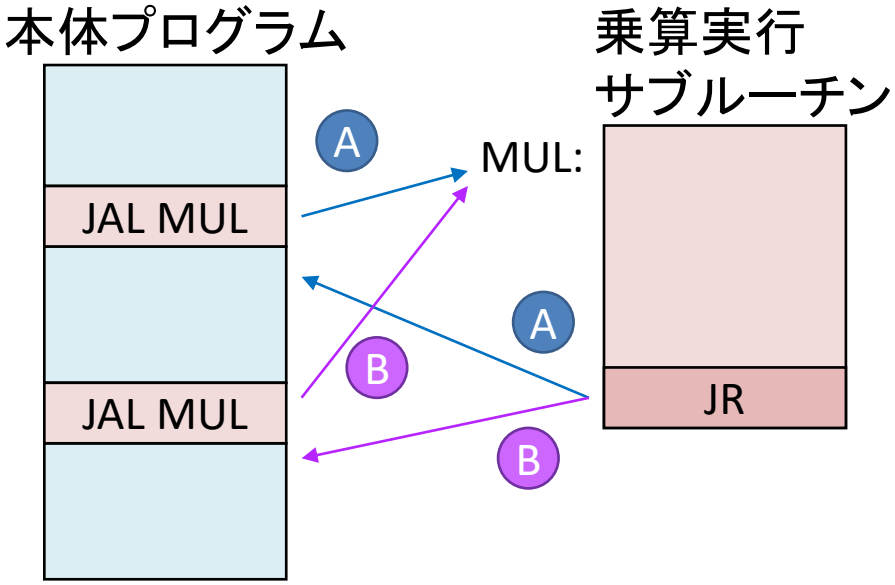
■乗算をサブルーチンとして呼び出す

1桁どうしの乗算を複数回実行

➡ 乗算をサブルーチンとして、必要なところで呼び出す

サブルーチンとは

- プログラム中の複数の箇所からジャンプして来る
- 処理が完了したら元の場所(の次のアドレス)へ戻る



- JAL (Jump And Link)
戻りアドレスを保存(Link)して
サブルーチンへジャンプ

- JR (Jump for Return)
保存している戻りアドレスへ
ジャンプ

■サブルーチン関連の命令定義

JAL arg

nbyte 3
opcode xxxxxxxx
0: MAR <- PC, PC <- inc
1: MDR <- mem
2: IR <- MDR
3: MAR <- PC, PC <- inc
4: MDR <- mem, MAR <- PC, PC <- inc
5: MDRH <- MDR, MDR <- mem, X <- PC
6: PC <- MDRW

JAL label

alias JAL arg

JR

alias JMP X

JR命令はJMP Xと同じ

戻りアドレス保存にレジスタXを利用

注意

・サブルーチンからさらにサブルーチンを呼び出すことはできない

・サブルーチン内でレジスタXを書き換えてはならない

■乗算サブルーチンの使用方法

```
        :
        LD      AC, X1
        LD      MPCD, AC
        LD      AC, B1
        LD      MPLY, AC
        JAL     MUL
        :

MUL:
        LD      AC, MPCD
        AND     AC, $0F
        :
        AND     AC, $F0
        ADD     AC, B
        JR
MPCD:   db 0
MPLY:   db 0
P0:     db 0
PH:     db 0
PL:     db 0
```

← 被乗数(MPCD)、乗数(MPLY)を設定
(ここではX1, B1の下位桁どうしを乗算)

← JALでサブルーチン呼び出し

← ここに戻ってきたとき、積はACに入っている

← 乗算が終了したらJRでサブルーチンから戻る

← サブルーチン中で使用するメモリ領域