パルテノン研究会 第3回 ASIC デザインコンテスト
規定課題「8 ビットマイクロプロセッサ KUE-CHIP2」

本規定課題では、8 ビットマイクロプロセッサ KUE-CHIP2[1] [2] を実現する。以下では実現すべき KUE-CHIP2 の命令セットアーキテクチャについて規定する。なお、規定ではパルテノン用に一部追加修正[3] をした。

1 内部レジスタ構造とメモリ空間

1.1 内部レジスタ構造

(1) アキュムレータ ACC:
　演算に使用する 8 ビットのレジスタ。演算のオペランドや演算結果を保持する。

(2) インデックスレジスタ IX:
　演算に使用する 8 ビットのレジスタ。演算のオペランドや演算結果を保持する。また、修飾アドレス指定のときのアドレス修飾にも利用される。

(3) フラグ FLAG:
　桁上げフラグ CF、ネガティブフラグ NF、桁あふれフラグ VF、零フラグ ZF の 4 ビットからなるレジスタ。命令の実行結果に従ってセット / リセットされる。

(4) プログラムカウンタ PC:
　8 ビットのレジスタ。次に実行すべき命令のアドレスを保持する。パワーオンリセットによりクリア (全ビット 0 と) される。

以上のレジスタは簡単のため SFL ではすべて reg_wr（パワーオンリセットでクリアされるレジスタの宣言）で宣言すること。

1.2 メモリ空間

メモリ空間は 512 バイトでバイト単位に指定される。0 バイト目から 255 バイト目はプログラム領域とよばれ、プログラムまたはデータを格納する領域として利用する。256 バイト目から 511 バイト目まではデータ領域と呼ばれ、データを格納する領域としてのみ利用する。

プログラム領域またはデータ領域を指定するアドレスの最上位ビット (第 8 ビット目) は命令コード内のビットから生成される。このビットで、プログラム領域またはデータ領域のいずれにアクセスするかが決められる。なお、すべてのメモリ空間は KUE-CHIP2 の外部にある。

1.3 データ語の構成

KUE-CHIP2 のデータ語は最上位ビット (第 7 ビット) が符号に対応し、2 の補数表示で負の数を表す。

2 命令セット

2.1 命令形式

KUE-CHIP2 の命令サイズは 1 バイトまたは 2 バイトであり、以下の 4 種類の形式で構成される。

(1) タイプ 1(1 バイト形式) (NOP,HLT,OUT,IN,RCF,SCF)
　命令コード (5 ビット) + ドントケア (3 ビット)

(2) タイプ 2(2 バイト形式) (Bcc)
　命令コード (4 ビット) + 条件コード (4 ビット) + 分岐アドレス (8 ビット)

(3) タイプ 3(1 バイト形式) (Ssm,Rsm)
　命令コード (4 ビット) + 第 1 オペランド (1 ビット) + 巡回モード (1 ビット) + シフトモード (2 ビット)

(4) タイプ 4(1 バイトまたは 2 バイト形式) (残りの命令)
　命令コード (4 ビット) + 第 1 オペランド (1 ビット) + 第 2 オペランドのアドレスモード (3 ビット)
　　{ + 第 2 オペランド (8 ビット)}
　(注) 第 2 オペランドのアドレスモードの指定により 1 バイトまたは 2 バイトとなる。

---

[1] KUE-CHIP2 は九州大学大学院総合理工学研究科安浦寛人教授らにより情報工学教育用に開発された 8 ビットマイクロプロセッサである。
[2] kue-design@astem.or.jp、「僕らのマイクロプロセッサ開発記 (1 ~ 5)」、『bit』、1993 年 10 月号 ~ 1994 年 2 月号
[3] IBUF_FLG_CLR 端子を削除した (3)。その他記述をより明確な表現にした。

2.2　アドレスモード

KUE-CHIP2 のオペランドのアドレスモードを以下に示す。

レジスタ指定 (第 **1** オペランド及び第 **2** オペランド)	
ACC	: アキュムレータ
IX	: インデックスレジスタ
アドレス指定 (第 **2** オペランド)	
d	: 即値アドレス
[d]	: 絶対アドレス (プログラム領域を指定)
(d)	: 絶対アドレス (データ領域を指定)
[IX+d]	: インデックス修飾アドレス (プログラム領域を指定)
(IX+d)	: インデックス修飾アドレス (データ領域を指定)

アキュムレータ ACC とインデックスレジスタ IX の 2 つのレジスタ指定と、メモリアクセス時のアドレス指定ができる。メモリアクセス時のアドレス指定には 5 つのモードがあり、即値アドレス、絶対アドレス (プログラム領域指定とデータ領域指定)、インデックスレジスタによる修飾アドレス (プログラム領域指定とデータ領域指定) がある。

即値アドレスではオペランドの内容がそのままデータとなる。絶対アドレスではオペランドの内容がデータの格納されているアドレスを示す。インデックス修飾アドレスではオペランドの内容とインデックスレジスタ IX の内容を加算して決まる値がオペランドのアドレスとなる。インデックス修飾アドレスを計算する際のオーバフローは無視される。

2.3　命令セット

命令語は 1 バイトあるいは 2 バイトで構成され、命令はプログラム領域の 0 バイト目から格納される。命令の 2 バイト目は 1 バイト目の次のアドレスに置かれ、分岐命令が成功する場合を除いて次のアドレスの命令が次に実行される。

命令は 19 種で、入出力命令 2、シフト命令 2、算術論理演算命令 8、フラグセット命令 2、ロード命令、ストア命令、ブランチ命令、 NOP 命令、停止命令からなる。

各命令の簡単な動作は以下のとおりである。

[タイプ **1**]	
NOP (No OPeration)	何もしない。
HLT (HaLT)	停止命令。
OUT (OUTput)	出力命令。 ACC の内容を OBUF へ移す。
IN (INput)	入力命令。 IBUF の内容を ACC へ移す。
RCF (Reset CF)	桁上げフラグ CF をリセットする。
SCF (Set CF)	桁上げフラグ CF をセットする。
[タイプ **2**]	
Bcc (Branch cc)	分岐命令。分岐条件 cc は分岐する条件を指定。
[タイプ **3**]	
Ssm (Shift sm)	シフト命令。 sm はシフトモードを指定。
Rsm (Rotate sm)	巡回シフト命令。 sm はシフトモードを指定。
[タイプ **4**]	
LD (LoaD)	ロード命令。メモリの内容をレジスタへ移す。
ST (STore)	ストア命令。レジスタの内容をメモリへ移す。
SBC (SuBtract with Carry)	減算命令。桁上げフラグ CF を考慮する。
ADC (ADd with Carry)	加算命令。桁上げフラグ CF を考慮する。
SUB (SUBtract)	減算命令。桁上げフラグ CF を考慮しない。
ADD (ADD)	加算命令。桁上げフラグ CF を考慮しない。
EOR (Exclusive OR)	ビット毎の排他的論理和演算命令。
OR (OR)	ビット毎の論理和演算命令。
AND (AND)	ビット毎の論理積演算命令。
CMP (CoMPare)	比較命令。

命令コードと動作の詳細を以下に示す。下記の命令コードに違反した命令に対する設計は不要である。以下の説明で`{ACC}`などは ACC の内容を表す[4]。

命令形式	略記号	命令コード	命令の動作
タイプ **1**	NOP	0 0 0 0 0 - - -	何もしない
	HLT	0 0 0 0 1 - - -	停止
	OUT	0 0 0 1 0 - - -	`{ACC}->DBo,1->OBUF_WE`
	IN	0 0 0 1 1 - - -	`{DBi}->ACC,1->IBUF_RE`
	RCF	0 0 1 0 0 - - -	`0->CF`

[4] op2 が、即値アドレスの場合はその値

	SCF	0 0 1 0 1 - - -	1->CF
タイプ 2	Bcc	0 0 1 1 cc	条件が成立すれば、分岐アドレス ->PC
タイプ 3	Ssm	0 1 0 0 A 0 sm	{op1}->{shift}->op1, はみ出したビット ->CF
	Rsm	0 1 0 0 A 1 sm	{op1}->{rotate}->op1, はみ出したビット ->CF
タイプ 4	LD	0 1 1 0 A B	{op2}->op1
	ST	0 1 1 1 A B	{op1}->op2
			(注)ST 命令の場合、 B=100,101,110,111 のみ可能。
	SBC	1 0 0 0 A B	{op1}-{op2}-{CF}->op1
	ADC	1 0 0 1 A B	{op1}+{op2}+{CF}->op1
	SUB	1 0 1 0 A B	{op1}-{op2}->op1
	ADD	1 0 1 1 A B	{op1}+{op2}->op1
	EOR	1 1 0 0 A B	{op1}eor{op2}->op1
	OR	1 1 0 1 A B	{op1}and{op2}->op1
	AND	1 1 1 0 A B	{op1}or{op2}->op1
	CMP	1 1 1 1 A B	{op1}-{op2}

ここで、 op1、 op2 は各々第 1、第 2 オペランドである。また A、 B は各々第 1 オペランドと第 2 オペランドのアドレスモードを指定している。第 1 オペランドは

A=0	ACC
A=1	IX

で指定される場所であり、第 2 オペランドは以下のアドレスモードにより指定される場所である[5] 。

B=000	ACC
B=001	IX
B=01-	d
B=100	[d]
B=101	(d)
B=110	[IX+d]
B=111	(IX+d)

シフトモード sm を以下に示す。

RA	0 0	Right Arithmetically
LA	0 1	Left Arithmetically
RL	1 0	Right Logically
LL	1 1	Left Logically

分岐条件 cc を以下に示す。

A	0 0 0 0	Always	
VF	1 0 0 0	on oVerFlow	VF=1
NZ	0 0 0 1	on Not Zero	ZF=0
Z	1 0 0 1	on Zero	ZF=1
ZP	0 0 1 0	on Zero or Positive	NF=0
N	1 0 1 0	on Negative	NF=1
P	0 0 1 1	on Positive	(NF or ZF)=0
ZN	1 0 1 1	on Zero or Negative	(NF or ZF)=1
NI	0 1 0 0	on No Input	IBUF_FLG_IN=0
NO	1 1 0 0	on No Output	OBUF_FLG_IN=1
NC	0 1 0 1	on Not Carry	CF=0
C	1 1 0 1	on Carry	CF=1
GE	0 1 1 0	on Greater than or Equal	(VF eor NF)=0
LT	1 1 1 0	on Less Than	(VF eor NF)=1
GT	0 1 1 1	on Greater Than	((VF eor NF) or ZF)=0
LE	1 1 1 1	on Less than or Equal	((VF eor NF) or ZF)=1

シフト命令 Ssm と巡回シフト命令 Rsm の動作を以下に示す。 SLA と SLL では、実行後の各フラグの状態のみが異なる。

SRA 命令

レジスタ (ビット)	:b7 b6 b5 b4 b3 b2 b1 b0	CF フラグ
実行前	:d7 d6 d5 d4 d3 d2 d1 d0	ドントケア
実行後	:d7 d7 d6 d5 d4 d3 d2 d1	d0

---

[5] このように変数が表す場所と内容を区別する表記法とそうしない表記法があるが、ここでは前者の表記法をとった。

SLA 命令

レジスタ (ビット)	:b7 b6 b5 b4 b3 b2 b1 b0	CF フラグ
実行前	:d7 d6 d5 d4 d3 d2 d1 d0	ドントケア
実行後	:d6 d5 d4 d3 d2 d1 d0 0	d7

SRL 命令

レジスタ (ビット)	:b7 b6 b5 b4 b3 b2 b1 b0	CF フラグ
実行前	:d7 d6 d5 d4 d3 d2 d1 d0	ドントケア
実行後	: 0 d7 d6 d5 d4 d3 d2 d1	d0

SLL 命令

レジスタ (ビット)	:b7 b6 b5 b4 b3 b2 b1 b0	CF フラグ
実行前	:d7 d6 d5 d4 d3 d2 d1 d0	ドントケア
実行後	:d6 d5 d4 d3 d2 d1 d0 0	d7

RRA 命令

レジスタ (ビット)	:b7 b6 b5 b4 b3 b2 b1 b0	CF フラグ
実行前	:d7 d6 d5 d4 d3 d2 d1 d0	cf
実行後	:cf d7 d6 d5 d4 d3 d2 d1	d0

RLA 命令

レジスタ (ビット)	:b7 b6 b5 b4 b3 b2 b1 b0	CF フラグ
実行前	:d7 d6 d5 d4 d3 d2 d1 d0	cf
実行後	:d6 d5 d4 d3 d2 d1 d0 cf	d7

RRL 命令

レジスタ (ビット)	:b7 b6 b5 b4 b3 b2 b1 b0	CF フラグ
実行前	:d7 d6 d5 d4 d3 d2 d1 d0	ドントケア
実行後	:d0 d7 d6 d5 d4 d3 d2 d1	d0

RLL 命令

レジスタ (ビット)	:b7 b6 b5 b4 b3 b2 b1 b0	CF フラグ
実行前	:d7 d6 d5 d4 d3 d2 d1 d0	ドントケア
実行後	:d6 d5 d4 d3 d2 d1 d0 d7	d7

次に、各命令の実行に影響するフラグ及び命令実行後のフラグの設定条件 (実行後の状態) を以下に示す。

命令		実行への影響				実行後の状態			
		CF	VF	NF	ZF	CF	VF	NF	ZF
NOP	No OPeration	-	-	-	-	-	-	-	-
HLT	HaLT	-	-	-	-	-	-	-	-
OUT	OUTput	-	-	-	-	-	-	-	-
IN	INput	-	-	-	-	-	-	-	-
RCF	Reset Carry Flug	-	-	-	-	0	-	-	-
SCF	Set Carry Flug	-	-	-	-	1	-	-	-
SRA	Shift Right Arithmetically	-	-	-	-	b0	0	N	Z
SLA	Shift Left Arithmetically	-	-	-	-	b7	V	N	Z
SRL	Shift Right Logically	-	-	-	-	b0	0	N	Z
SLL	Shift Left Logically	-	-	-	-	b7	0	N	Z
RRA	Rotate Right Arithmetically	o	-	-	-	b0	0	N	Z
RLA	Rotate Left Arithmetically	o	-	-	-	b7	V	N	Z
RRL	Rotate Right Logically	-	-	-	-	b0	0	N	Z
RLL	Rotate Left Logically	-	-	-	-	b7	0	N	Z
LD	LoaD	-	-	-	-	-	-	-	-
ST	STore	-	-	-	-	-	-	-	-
SBC	SuBtract with Carry	o	-	-	-	C	V	N	Z
ADC	ADd with Carry	o	-	-	-	C	V	N	Z
SUB	SUBtract	-	-	-	-	-	V	N	Z
ADD	ADD	-	-	-	-	-	V	N	Z
EOR	Exclusive OR	-	-	-	-	-	0	N	Z
OR	OR	-	-	-	-	-	0	N	Z
AND	AND	-	-	-	-	-	0	N	Z
CMP	CoMPare	-	-	-	-	-	V	N	Z
BA	Branch Always	-	-	-	-	-	-	-	-
BVF	Branch on oVerFlow	-	o	-	-	-	-	-	-
BNZ	Branch on Not Zero	-	-	-	o	-	-	-	-
BZ	Branch on Zero	-	-	-	o	-	-	-	-
BZP	Branch on Zero or Positive	-	-	o	-	-	-	-	-

BN	Branch on Negative	-	-	o	-	-	-	-	-
BP	Branch on Positive	-	-	o	o	-	-	-	-
BZN	Branch on Zero or Negative	-	-	o	o	-	-	-	-
BNI	Branch on No Input	-	-	-	-	-	-	-	-
BNO	Branch on No Output	-	-	-	-	-	-	-	-
BNC	Branch on No Carry	o	-	-	-	-	-	-	-
BC	Branch on Carry	o	-	-	-	-	-	-	-
BGE	Branch on Greater than or Equal	-	o	o	-	-	-	-	-
BLT	Branch on Less Than	-	o	o	-	-	-	-	-
BGT	Branch on Greater Than	-	o	o	o	-	-	-	-
BLE	Branch on Less Than or Equal	-	o	o	o	-	-	-	-

ここで記号の意味は以下の通りである。

実行への影響
- o : 影響する。
- - : 影響しない。

実行後の状態
- C : 桁上げまたは桁落ちが発生した場合に 1 が設定される。
- V : オーバフローが発生した場合に 1 が設定される。
- N : 演算結果の第 7 ビットの値が設定される。
- Z : 演算結果が 0 の場合に 1 が設定される
- b7: 第 1 オペランドの第 7 ビットの値が設定される。
- b0: 第 1 オペランドの第 0 ビットの値が設定される。
- 0 : 0 が設定される。
- 1 : 1 が設定される。
- - : 変化しない。

3 外部端子

KUE-CHIP2 は以下の外部入出力端子を持つ。これらの与えられた外部インターフェースは変更してはならない。

- **VDD**
  電源の +5.0V を接続する。
- **VSS**
  電源のグラウンド (0V) を接続する。
- **m_clock**
  マスタークロック
- **p_reset**
  チップ内のすべての記憶素子を非同期的にリセット (論理値 0 に) する。

上記 4 つの外部端子については、 PARTHENON で自動生成される端子であり、 SFL では記述しない。

- **DBi(0:7) (Data Bus for input)**
  IBUF またはメモリからのデータ (命令を含む) が入力される。

- **DBo(0:7) (Data Bus for output)**
  OBUF またはメモリへ書き込むデータが出力される。
- **AB(0:8) (Address Bus)**
  アクセスすべきメモリのアドレスを出力する。
- **MEM_WE (MEMory Write Enable)**
  メモリへ書き込む契機を出力する。このとき、 AB にはアドレスが、 DBo には書き込むべきデータが出力される。
- **MEM_RE (MEMory Read Enable)**
  メモリから読み込む契機を出力する。このとき、 AB にはアドレスが出力され、 DBi の値が入力される。
- **IBUF_FLG_IN (IBUF FLaG IN)**
  IBUF のフラグ IBUF_FLAG の値が入力され、分岐命令で参照される。 IBUF_FLAG のセットは IBUF にデータが存在することを示している。
- **IBUF_RE (IBUF Read Enable)**
  IN 命令によりアサートされる。この契機により IBUF の内容が DBi へ出力されるものとする。また、この契機により IBUF のフラグ IBUF_FLAG がリセットされるものとする。なお、 IBUF_FLAG のセットは IBUF へデータが用意された時に行われるものとする。

- **OBUF_FLG_IN (OBUF FLaG IN)**
  OBUF のフラグ OBUF_FLAG の値が入力され、分岐命令で参照される。 OBUF_FLAG のセットは OBUF にデータが存在することを示している。
- **OBUF_WE (OBUF Write Enable)**
  OUT 命令によりアサートされる。この契機により DBo の内容が OBUF に書き込まれるものとする。また、この契機により OBUF のフラグ OBUF_FLAG がセットされるものとする。なお、 OBUF_FLAG のリセットは OBUF のデータが使われた時に行われるものとする。
- **start**
  KUE_CHIP2 の動作を開始する契機を与える端子。

MEM_RE と MEM_WE、 IBUF_RE と MEM_RE、 OBUF_WE と MEM_WE は各々同時にアクティブとなってはならない。

与えられた外部インターフェースの SFL 記述を以下に示す。なお、 KUE-CHIP2 の SFL 記述でのモジュール名は KUE_CHIP2 とする。

```
declare KUE_CHIP2 {
    input    DBi<8> ;
    output   DBo<8> ;
    output   AB<9> ;
    instrout MEM_WE;
    instrout MEM_RE;
    instrin  IBUF_FLG_IN;
    instrout IBUF_RE;
    instrin  OBUF_FLG_IN;
    instrout OBUF_WE;
    instrin  start;
}
```

4 ベンチマークプログラム

性能の客観的な評価を行う基準として、 4.1 および 4.2 に示す 2 つのベンチマークプログラムの実行時間 (マシンサイクル数) をシミュレータで測定して、設計データに添付することが望ましい。ただし、実行時間のみが設計の良さを判定する基準ではない。

4.1 課題 1. バブルソートによる整列

256 個の逆順に並んだデータ 127(7FH),126(7EH),...,0(00H),-1(FFH),-2(FEH),...,-128(80H) を下記のプログラムでソートする。

[プログラムの説明]

バブルソートによって、データ領域の DATA(00H 番地) から始まる、 N(80H 番地) バイトのデータを昇順に整列する。ここで、データは、 2 個から 256 個の範囲とする。ただし、データが 256 個の場合は N を 0 とする。なお、ベンチマークプログラムでは、ソートの完了判定に桁上げフラグ cf を利用している。

[プログラムの入出力例]

```
入力:
 80 : 08（ソートするデータの長さ）
100 : 10 FF 40 80 C0 D8 7F CD  （ソートするデータ）
出力:
100 : 80 C0 CD D8 FF 10 40 7F  （ソート結果）
```

[プログラムのコード]

```
*** KUE-CHIP2 Assembler ver.1.0 by H.Ochi ***
 * Bubble Sorting
 * Programmed by Akira Uejima, May. 3, 1992
 * Data(singned) on data page
  00 :               DATA:   EQU      00H
 * Data Length(byte) on program page
  80 :               N:      EQU      80H
 * Work Area(loop counter) on program page
  90 :               WORK1:  EQU      90H
 * Work Area(swap area) on program page
```

```
 91 :               WORK2:  EQU       91H

 00 : 6C 80         LD      IX,       [N]
 02 : AA 01         SUB     IX,       1
 04 : 7C 90         ST      IX,       [WORK1]
 06 : C9     LP1:   EOR     IX,       IX
 07 : 20            RCF
 08 : 67 00  LP2:   LD      ACC,      (IX+DATA)
 0A : F7 01         CMP     ACC,      (IX+DATA+1)
 0C : 3F 19         BLE               SKIP
 0E : 74 91         ST      ACC,      [WORK2]
 10 : 67 01         LD      ACC,      (IX+DATA+1)
 12 : 77 00         ST      ACC,      (IX+DATA)
 14 : 64 91         LD      ACC,      [WORK2]
 16 : 77 01         ST      ACC,      (IX+DATA+1)
 18 : 2F            SCF
 19 : BA 01  SKIP:  ADD     IX,       1
 1B : FC 90         CMP     IX,       [WORK1]
 1D : 31 08         BNZ               LP2
 1F : 35 29         BNC               FIN
 21 : 6C 90         LD      IX,       [WORK1]
 23 : AA 01         SUB     IX,       1
 25 : 7C 90         ST      IX,       [WORK1]
 27 : 31 06         BNZ               LP1
 29 : 0F     FIN:   HLT

                    END
```

4.2 課題 2. CRC コードの計算

課題 1 の入力データならびに出力データ (ともに 256 バイト) に対して CRC(Cyclic Redundancy Check) コードを下記のプログラムによって生成する。

[プログラムの説明]

シフト / ローテート命令や論理演算命令を使用して、データ領域の DATA(00H 番地) から始まる N(80H 番地) バイトのデータの CRC コードを計算する。データの最大長は 256 バイトである。ただし、データ長が 256 バイトの場合は、バイト数 N を 0 とする。尚、 CRC コードの生成多項式として、 CCITT X.25 規格の $X^{16}+X^{12}+X^{5}+1$ を用いている。

[プログラムの入出力例]

```
入力:
 80 : 08（データのバイト数）
100 : 62 FF 75 C0 75 C1 C9 7D（データ）
出力:
 C0 : B0 8A（CRC）
```

[プログラムのコード]

```
*** KUE-CHIP2 Assembler ver.1.0 by H.Ochi ***
 * Calculate CRC(Cyclic Redundancy Check) Code
 * Programmed by Akira Uejima, May. 3, 1992
 * Data on date page
  00 :        DATA:  EQU              00H
 * Data Length(byte) on program page
  80 :        N:     EQU              80H
 * Resultant CRC on program page
  C0 :        C1:    EQU              0C0H
  C1 :        C2:    EQU              0C1H
 * Work Area on program page
  F0 :        WORK:  EQU              0F0H

  00 : 62 FF         LD      ACC,     0FFH
  02 : 74 C0         ST      ACC,     [C1]
```

```
 04 : 74 C1         ST      ACC,     [C2]
 06 : C9            EOR     IX,      IX
 07 : 7C F0         ST      IX,      [WORK]
 09 : 64 C0  LP1:   LD      ACC,     [C1]
 0B : C7 00         EOR     ACC,     (IX+DATA)
 0D : 74 C0         ST      ACC,     [C1]
 0F : 6A 08         LD      IX,      8
 11 : 64 C1  LP2:   LD      ACC,     [C2]
 13 : 43            SLL     ACC
 14 : 74 C1         ST      ACC,     [C2]
 16 : 64 C0         LD      ACC,     [C1]
 18 : 45            RLA     ACC
 19 : 74 C0         ST      ACC,     [C1]
 1B : 35 27         BNC              SKIP
 1D : C4 42         EOR     ACC,     [P1]
 1F : 74 C0         ST      ACC,     [C1]
 21 : 64 C1         LD      ACC,     [C2]
 23 : C4 43         EOR     ACC,     [P2]
 25 : 74 C1         ST      ACC,     [C2]
 27 : AA 01  SKIP:  SUB     IX,      1
 29 : 33 11         BP               LP2
 2B : 6C F0         LD      IX,      [WORK]
 2D : BA 01         ADD     IX,      1
 2F : FC 80         CMP     IX,      [N]
 31 : 7C F0         ST      IX,      [WORK]
 33 : 31 09         BNZ              LP1
 35 : 64 C1         LD      ACC,     [C2]
 37 : C2 FF         EOR     ACC,     0FFH
 39 : 74 C1         ST      ACC,     [C2]
 3B : 64 C0         LD      ACC,     [C1]
 3D : C2 FF         EOR     ACC,     0FFH
 3F : 74 C0         ST      ACC,     [C1]
 41 : 0F            HLT
* CRC Generator Polynomial on program page
* CCITT(X^16 + X^12 + X^5 + 1) -> 1  0001  0000  0010  0001
*                                     1     0     2     1
 42 : 10     P1:    PROG             10H
 43 : 21     P2:    PROG             21H

                    END
```