900, 900/L, 900/H, 900/L1, 900/H2　CPUコアの違いについて

TLCS-900ファミリには、CPUコアとして① 900, ② 900/L, ③ 900/H, ④ 900/L1, ⑤ 900/H2の5種のCPUコアがあり、それぞれ表1に示す点が異なります。

表 1 CPU 相違点

CPU / 相違内容	900	900/L	900/H, 900/L1	900H2
最大アドレスバス幅	24 ビット	←	←	←
最大データバス幅	16 ビット	←	←	32 ビット
命令キューバッファ	4 バイト	←	←	12 バイト
命令セット	TLCS-900	TLCS-900 から 以下の命令を削除 NORMAL MAX 以下の命令を追加 MIN	TLCS-900 から 以下の命令を削除 NORMAL MAX	TLCS-900 から以下の命令を削除 NORMAL MAX LDX
分岐命令実行時のコードフェッチ	分岐条件が真の時のみ、分岐先コードをフェッチする	←	←	分岐条件に関係なく、常に分岐先コードを先行フェッチする
マイクロ DMA	4 チャネル	←	←	8 チャンネル
CPU 特権モード	システムモード、ノーマルモード	システムモードのみ	←	←
CPU レジスタモード	MIN モード、 MAX モード (リセット後MIN モード)	← (リセット後MAX モード)	MAX モードのみ	←
割り込み方式	リスタート方式	ベクタ方式	←	←
ノーマルスタックポインタ XNSP	あり	なし	←	←
割り込みネスティングカウンタ INTNEST (主に OS 用)	なし	あり	←	←

1. 概　要

TLCS-900/L1シリーズは、東芝オリジナルの高性能16ビットCPUを内蔵しています。

このCPUと多様なI/O機能ブロック (タイマ、シリアルI/O、ADなど) をワンチップ化したTLCS-900/L1シリーズは、多彩な応用分野に適用可能です。

TLCS-900/L1のCPUは、16ビットCPUながら、内蔵汎用レジスタは32ビット構成のレジスタバンク方式を採用しているため、特に組み込み制御などに最適です。

TLCS-900/L1のCPUの特長は以下のとおりです。

(1) TLCS-90 (東芝オリジナル8ビットCPU) の拡張アーキテクチャ
- 命令ニモニック&レジスタセットのレベルで上位互換

(2) 汎用レジスタマシン
- 8本のレジスタがすべてアキュームレータとして使用可能

(3) レジスタバンク方式 (4本のレジスタをバンク化)
- 32ビット幅 4バンク

(4) 広大なリニアアドレス空間 (16Mバイト) と豊富なアドレッシングモード (9種)

(5) ダイナミックバスサイジングシステム
- 8ビットまたは16ビットの外部データバスが混在可能

(6) 直交性のある豊富な命令セット
- 8/16/32ビットのデータ転送/演算命令
- 16ビット乗除算命令
  16×16 → 32ビット　(符号なし/付き)
  32÷16 → 16ビット…16ビット　(符号なし/付き)
- ビット演算を含む多様なビット処理命令
- Cコンパイラ対応命令
- フィルタ演算命令 : 積和演算/モジュロ増加命令

(7) 高速処理
- 4バイトの命令キューバッファを内蔵したパイプラインシステム
- 32ビットALU

2. CPUの動作モード

900/L1は、システムモード固定です。システムモードでは使用できる命令およびレジスタの種類に制限はありません。

システムモードでCPUが取り扱い可能なリソースには、以下のものがあります。

1) 汎用レジスタ :
   - 4本の32ビット汎用レジスタ×4バンク
   - 4本の32ビット汎用レジスタ (スタックポインタ : XSPを含む)
2) ステータスレジスタ (SR)
3) プログラムカウンタ (PC) : 32ビット
4) コントロールレジスタ : マイクロDMAのパラメータ設定レジスタなど
5) CPUの全命令
6) 内蔵I/Oレジスタのすべて
7) 実装されているメモリのすべて

3. レジスタ

3.1 レジスタ構成……16 MBのプログラム空間/16MBのデータ空間

レジスタ構成を図3.1.1 に示します。
汎用レジスタ4本 (32ビット幅)×4バンク
+
汎用レジスタ4本 (32ビット幅)
+
プログラムカウンタ (32ビット幅)
+
ステータスレジスタ

モード切り替えについて

リセットにより、ステータスレジスタSRの<MAX>ビットは、1に初期化され、マキシマムモードになります。TLCS-900/L1は、マキシマムモード固定です。

スタックポインタについて

システムモード用レジスタ (XSP) のみ持っており、リセットにより100Hにセットされます。

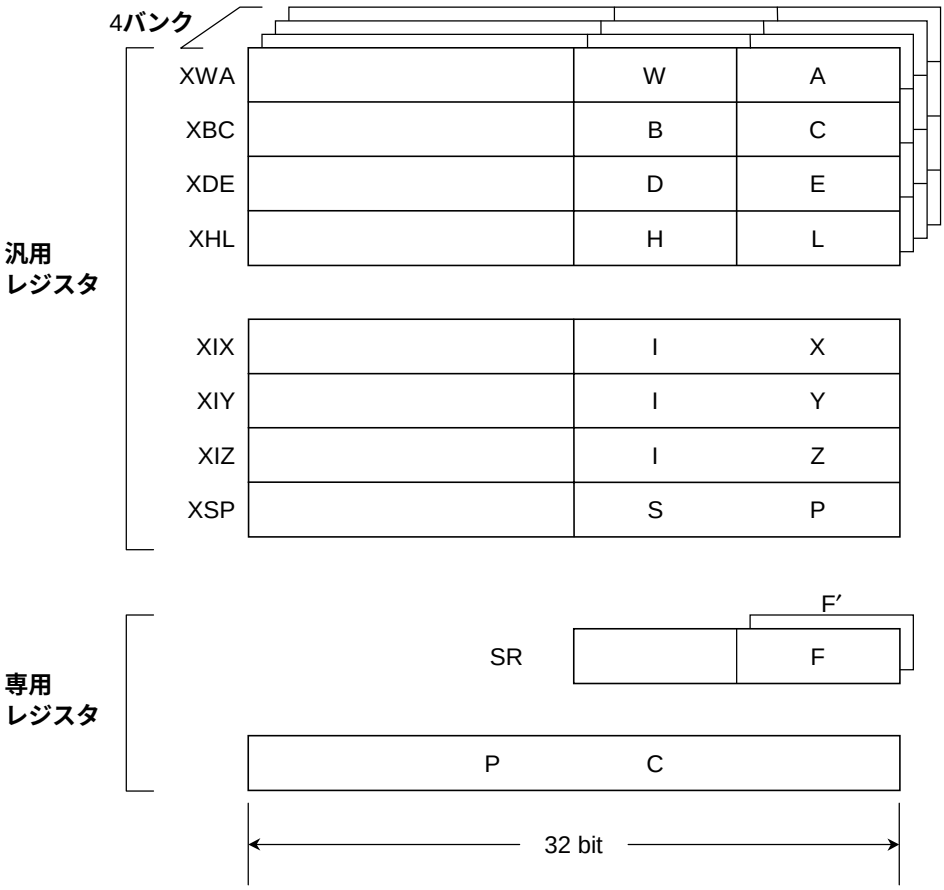


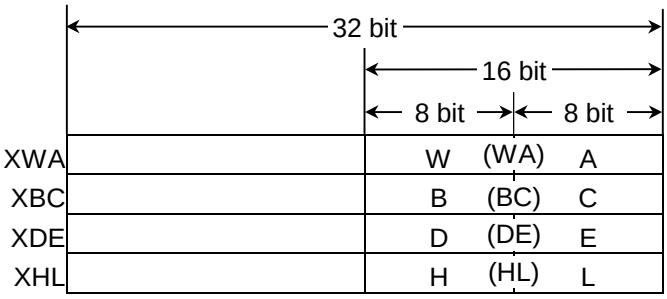
図 3.1.1 **レジスタ構成** (16MB **プログラム**)

3.2 レジスタの詳細

3.2.1 汎用バンクレジスタ

マキシマムモードでは、4バンクよりなる下図に示す4本の32ビット汎用レジスタとして使用できます。それぞれのバンクにおけるレジスタ構成は下図のとおりです。

4本の32ビットレジスタ (XWA, XBC, XDE, XHL) は汎用レジスタであり、アキュムレータ、インデックスレジスタとして使用できます。それぞれの32ビットレジスタは16ビットレジスタ (WA, BC, DE, HL) としても使用でき、32ビットレジスタの下位16ビットに割り付けられています。



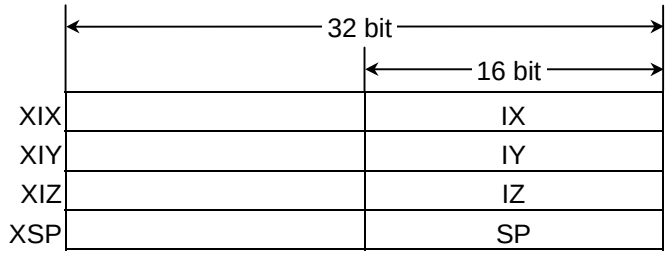
注) ( ) : カッコ内は16ビットレジスタの名称です。

16ビットレジスタはアキュムレータおよびインデックスアドレッシングモードのインデックスレジスタ、ディスプレースメントレジスタとして使用できます。さらに16ビットレジスタは2つの8ビット汎用レジスタ (W, A, B, C, D, E, H, L) としても使用することができ、アキュムレータなどになります。

3.2.2 32ビット汎用レジスタ

4本の32ビット汎用レジスタ (XIX, XIY, XIZ, XSP) があり、レジスタ構成は下記のとおりです。

このレジスタもアキュムレータ、インデックスレジスタ、ディスプレースメントレジスタとして使用でき、16ビットレジスタ/8ビットレジスタ (8ビットレジスタとして使用する場合のレジスタ名称は後述) として取り扱うことができます。



スタックポインタ

レジスタ(XSP)はスタックポインタと呼ばれ、割り込み、CALL命令実行時のスタックポインタとして動作します。

リセットにより、XSPは100Hに初期化されます。

3.2.3 ステータスレジスタ (SR)

ステータスレジスタには、CPUの状態(CPU動作モード、レジスタ構成など)と、演算結果の状態を表すフラグが、格納されます。このレジスタは上位バイトと下位バイトの2つの部分で構成されています。 ステータスレジスタの上位バイト(ビット8～15)はCPUの状態を表します。下位バイト(ビット0～7)は通常フラグレジスタ(F)と呼ばれ、演算結果の状態を表しています。TLCS-900/L1は2つのフラグレジスタ(F/F´)をもっており、EX命令で交換を行うことができます。

(1) ステータスレジスタSRの上位バイト

15	14	13	12	11	10	9	8
SYSM	IFF2	IFF1	IFF0	MAX	RFP2	RFP1	RFP0

① SYSM (SYStem Mode):

システムモードかノーマルモードかを示します。900/L1では、システムモードに固定されています。

0	**ノーマルモード**
1	**システムモード** (900/L1 **はこのモードに固定される**)

② IFF2～IFF0 (Interrupt mask F/F 2～0) :

割り込みレベルのマスクレジスタです。割り込み要求レベルは、1～7まであり、7が、最も優先順位の高い割り込み要求レベルです。

リセットにより111に初期化されます。割り込みが受け付けられたとき、その割り込みレベルより1だけ高い値がセットされます。

000	**レベル** 1 **以上の割り込みを許可**
001	**レベル** 1 **以上の割り込みを許可**
010	**レベル** 2 **以上の割り込みを許可**
011	**レベル** 3 **以上の割り込みを許可**
100	**レベル** 4 **以上の割り込みを許可**
101	**レベル** 5 **以上の割り込みを許可**
110	**レベル** 6 **以上の割り込みを許可**
111	**レベル** 7 **の割り込み(ノンマスカブル割り込み)だけを許可**

↑ **同じです。**

ただし、レベル7の割り込み受け付け時は、111がセットされます。なお、EI命令はTLCS-90と異なり、実行後直ちに有効となります。

③ MAX ($\overline{\text{MIN}}$imum /MAXimum) :

レジスタバンクのレジスタ幅と、プログラムカウンタPCの幅を指定するビットです。

0	**ミニマムモード**
1	**マキシマムモード** (900/L1**は、このモードで使用してください**)

リセットにより、900/L1では1、(マキシマムモード)に初期化されます。
900/L1はミニマムモードをサポートしていません。0をライトしないでください。

④ RFP2～RFP0 (Register File Pointer2～0) :

現在使用しているレジスタファイル(レジスタバンク)のナンバーを示しています。リセットにより000に初期化されます。

この値はレジスタバンクの切り替え命令 (3種)で操作することができます。なお、マキシマムモードのとき、RFP2は0に固定され、下記命令でRFP2が1になるような操作をしても、0のままです。

- LDF imm ; RFP←imm (0～3)
- INCF ; RFP←RFP + 1
- DECF ; RFP←RFP – 1

(2) **フラグレジスタ**F

7	6	5	4	3	2	1	0	
S	Z	0	H	0	V	N	C	:R/W

① S (Sign flag)

サインフラグ。
演算結果が負のとき1がセットされ、正のとき0がセットされます。
(演算結果の最上位ビット(MSB)の値がコピーされます。)

② Z (Zero flag)

ゼロフラグ。
演算結果がゼロのとき1にセットされ、それ以外のときは0にセットされます。

③ H (Half carry flag)

ハーフキャリーフラグ。
演算の結果、ビット3からビット4へキャリーまたはボローが発生したとき1にセットされ、それ以外のときは0にセットされます。ただし、32ビット演算命令の場合は、不定値がセットされます。

④ V (parity/over-flow flag)

パリティ/オーバフローフラグ

演算の種類によってパリティを示す場合とオーバフローを示す場合があります。

パリティ(P): 演算の結果、1にセットされているビットの数が奇数のとき0にセットされ、偶数のとき1にセットされます。ただし、32ビット演算命令の場合は、不定値がセットされます。

オーバフロー(V): 演算の結果、オーバフローなしのとき0にセットされ、オーバフローのとき 1 にセットされます。

⑤ N (Negative)

ADD/SUBフラグ。

加算(ADDなど)命令実行後0にセットされ、減算(SUBなど)命令実行後1にセットされます。

このフラグは、DAA(10進補正)命令実行時に使用されます。

⑥ C (Carry)

キャリーフラグ。

演算の結果、キャリーまたはボローが発生したとき1にセットされ、それ以外のときは0にセットされます。

ステータスレジスタのリード/ライト方法

ビット 0〜15 **のリード**	① PUSH SR POP dst
ビット 0〜15 **のライト**	① POP SR
ビット 15 **のみ** <SYSM>	**システムモード固定のため常に** 1 **にセットされています。**
ビット 14〜12 **のみ** <IFF2 : 0>	① EI num num **の値がライトされます。**
ビット 11 **のみ** <MAX>	**マキシマムモード固定**(1 **にセット**)**で使用してください。** 0 **をライトしないでください。**
ビット 10〜8 **のみ** <RFP2 : 0>	① LDF imm ② INCF ③ DECF
ビット 7〜0 **のみ**	① PUSH F/POP F ② EX F, F′ ③ **演算命令などの実行により間接的にフラグがセットされます。**

3.2.4 プログラムカウンタ(PC)

プログラムカウンタは、次に実行するメモリアドレスを示すポインタです。マキシマムモードでは、32ビットのPCとなりますが、プログラム空間は各製品のアドレスピンの数によって決まります。アドレスピンが24本(A0～A23)の場合には、最大16M バイトのプログラム空間がリニア空間としてアクセス可能となります。この場合には、PCの上位8ビット(bit 24～31)は無視されます。

リセット後のPC

リセットによりベクタベース番地(FFFF00H)に格納されているリセットベクタをリードした後、その値をPCにセットし、そのベクタ以降のプログラムをリードし、実行します。

3.2.5 コントロールレジスタ (CR)

コントロールレジスタは、マイクロDMAの動作を制御するレジスタと、割り込みのネスト値をカウントする割り込みネスティングカウンタより成っています。コントロールレジスタは、 LDC命令で、アクセスできます。

コントロールレジスタは以下のとおりです。

	<DMAS0>		DMA ソースレジスタ
	<DMAS1>		
	<DMAS2>		
	<DMAS3>		
	<DMAD0>		DMA デスティネーションレジスタ
	<DMAD1>		
	<DMAD2>		
	<DMAD3>		
	DMAM0	(DMAC0)	DMA モード/カウンタレジスタ
	DMAM1	(DMAC1)	
	DMAM2	(DMAC2)	
	DMAM3	(DMAC3)	
		(INTNEST)	割り込みネスティングカウンタ

( ) : ワードレジスタ (16ビット) 名
< > : ロングワードレジスタ (32ビット) 名

マイクロDMAについては TLCS-900/L1個別製品の説明の章に詳細が記載されています。

3.3 レジスタバンクの切り替え

レジスタバンクは以下の3つに大別されます。
カレントバンクレジスタ
プレビアスバンクレジスタ
アブソリュートバンクレジスタ

カレントバンクは、レジスタファイルポインタ<RFP> (ステータスレジスタ :SRのビット8～10)が指し示しているレジスタバンクです。カレントバンクにあるレジスタは、前節で説明した汎用レジスタとして、使用されます。この<RFP>の内容を変更することにより、他のレジスタバンクがカレントレジスタバンクとなります。

プレビアスバンクは、レジスタファイルポインタ<RFP>から1を減じた値により指し示される、レジスタバンクです。例えば、カレントバンクがバンク3のときには、バンク2がプレビアスバンクとなります。プレビアスバンクのレジスタは、ダッシュの付いたレジスタ名(WA', BC', DE', HL' など)で取り扱われ、カレントバンクとの入れ替えをEX命令(EX A, A' など)で行うことができます。

すべてのバンクレジスタ(カレントバンク、プレビアスバンクを含む)には、バンクを表わす数値(アブソリュートバンクナンバー)が付けられており、この数値付きのレジスタ名を使って、全バンクのレジスタを使用することができます。この方法でアクセスできるレジスタ(全レジスタ)を、アブソリュートバンクレジスタと呼びます。

TLCS-900のCPUは、カレントバンクのレジスタをワーキングレジスタとして動作させたときに、最大のパフォーマンスを出すように、設計されています。上述のとおり、CPUは他バンクのレジスタを使用することも可能ですが、他バンクのレジスタを使用すると、パフォーマンスが若干低下します。CPUの最大効率を引き出すために、TLCS-900では、レジスタバンクを簡単に切り替える機能が付いています。

このバンク切り替え機能により
- CPUは最大効率で動作可能となる。
- プログラムのコード量低減が可能となる。
- 特に、割り込みサービスルーチンのコンテキストスイッチとして使用した場合に、応答スピード/コード量で有利。

などの優位性が得られます。

バンクの切り替えは、下記の命令を用いて行います。

LDF imm : イミディエートの内容を<RFP>にセット imm : 0～3
INCF : <RFP>を1増加させる。
DECF : <RFP>を1減少させる。

LDF命令のイミディエート値は、0～3になります。また、INCF/DECF命令を用いて、キャリー/ボローが発生したときには、そのキャリー/ボローは無視され、<RFP>の値は巡回されます。たとえば、バンク3でINCFを実行するとバンク0となり、バンク0でDECFを実行するとバンク3になります。従って、不用意な INCF/DECF命令は、レジスタバンクの内容を破壊する恐れがあります。

● レジスタバンクの使用例

TLCS-900/L1のレジスタはバンク構成をとっており、各バンクは処理用途や割り込みレベルによって使い分けることができます。以下にその例を示します。

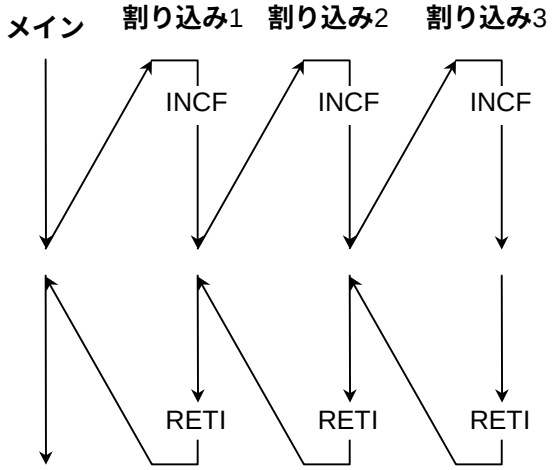
<例1> 各レジスタバンクを、それぞれの割り込み処理ルーチン専用に割り当てる使用例

レジスタバンク0 = メインプログラムと下記以外の割り込み処理用
レジスタバンク1 = INT0 処理専用
レジスタバンク2 = タイマ0 処理専用
レジスタバンク3 = タイマ1 処理専用

例えば、メインプログラム実行中、タイマ1の割り込みが発生し、その処理ルーチンへ分岐した場合、その処理ルーチンは、下記のようになり、レジスタのPUSH/POPが、不要になります。

```
LDF  3      ; レジスタバンクを3にする
  :
  :
  :
RETI        ; <RFP> (レジスタバンク)も含めて割り込み以前の状態に戻す。
```

<例2> 各レジスタバンクを、それぞれの割り込みレベルのネスティング専用に割り当てる例



(注1) この使用例では、割り込みのネスティングが、レジスタバンクの数 (4) を超えた場合、<RFP>は、000になり、レジスタバンク0の内容を破壊します。

(注2) INCF命令は、<RFP>←<RFP>+1を行う命令です。

3.4 汎用レジスタのアクセス

TLCS-900/L1の命令は、バイト単位で可変長になっています。カレントバンクのレジスタは、最短コード長でアクセスできます。また、命令コードが1バイト長くなりますが、すべての汎用レジスタをアクセスすることも可能です。すべての汎用レジスタとは、下記のとおりです。

① カレントバンクの汎用レジスタ

QW	(Q	WA)	QA	< X	WA >	W	(W	A)	A
QB	(Q	BC)	QC	< X	BC >	B	(B	C)	C
QD	(Q	DE)	QE	< X	DE >	D	(D	E)	E
QH	(Q	HL)	QL	< X	HL >	H	(H	L)	L

( ) : **ワードレジスタ** (16**ビット**)**名**
< > : **ロングワードレジスタ** (32**ビット**)**名**

② プレビアスバンクの汎用レジスタ

QW′	(Q	WA′)	QA′	< X	WA′>	W′	(W	A′)	A′
QB′	(Q	BC′)	QC′	< X	BC′ >	B′	(B	C′)	C′
QD′	(Q	DE′)	QE′	< X	DE′ >	D′	(D	E′)	E′
QH′	(Q	HL′)	QL′	< X	HL′ >	H′	(H	L′)	L′

③ 32**ビット汎用レジスタ**

QIXH	(Q	IX)	QIXL	< X	IX >	IXH	(I	X)	IXL
QIYH	(Q	IY)	QIYL	< X	IY >	IYH	(I	Y)	IYL
QIZH	(Q	IZ)	QIZL	< X	IZ >	IZH	(I	Z)	IZL
QSPH	(Q	SP)	QSPL	< X	SP >	SPH	(S	P)	SPL

④ **アブソリュートバンクレジスタ**

QW0	(QWA	0)	QA0	<XWA	0 >	RW0	(RWA	0)	RA0
QB0	(QBC	0)	QC0	<XBC	0 >	RB0	(RBC	0)	RC0
QD0	(QDE	0)	QE0	<XDE	0 >	RD0	(RDE	0)	RE0
QH0	(QHL	0)	QL0	<XHL	0 >	RH0	(RHL	0)	RL0
QW1	(QWA	1)	QA1	<XWA	1 >	RW1	(RWA	1)	RA1
QB1	(QBC	1)	QC1	<XBC	1 >	RB1	(RBC	1)	RC1
QD1	(QDE	1)	QE1	<XDE	1 >	RD1	(RDE	1)	RE1
QH1	(QHL	1)	QL1	<XHL	1 >	RH1	(RHL	1)	RL1
QW2	(QWA	2)	QA2	<XWA	2 >	RW2	(RWA	2)	RA2
QB2	(QBC	2)	QC2	<XBC	2 >	RB2	(RBC	2)	RC2
QD2	(QDE	2)	QE2	<XDE	2 >	RD2	(RDE	2)	RE2
QH2	(QHL	2)	QL2	<XHL	2 >	RH2	(RHL	2)	RL2
QW3	(QWA	3)	QA3	<XWA	3 >	RW3	(RWA	3)	RA3
QB3	(QBC	3)	QC3	<XBC	3 >	RB3	(RBC	3)	RC3
QD3	(QDE	3)	QE3	<XDE	3 >	RD3	(RDE	3)	RE3
QH3	(QHL	3)	QL3	<XHL	3 >	RH3	(RHL	3)	RL3

バンク0
バンク1
バンク2
バンク3

( ) : **ワードレジスタ** (16**ビット**)**名**
< > : **ロングワードレジスタ**(32**ビット**)**名**

4. アドレッシングモード

TLCS-900/L1には、9種類のアドレッシングモードがあります。これらは、ほとんどの命令と組み合わされて、CPUの処理能力を向上させています。

アドレッシングモードは、下記のようになっています。これは、TLCS-90のアドレッシングモードのすべてを包含しています。

No.	アドレッシングモード	記　述
1.	**レジスタ**	reg8 reg16 reg32
2.	**イミディエート**	n8 n16 n32
3.	**レジスタ間接**	(reg)
4.	**レジスタ間接** **プリデクリメント**	(–reg)
5.	**レジスタ間接** **ポストインクリメント**	(reg+)
6.	**インデックス**	(reg + d8) (reg + d16)
7.	**レジスタインデックス**	(reg + reg8) (reg + reg16)
8.	**絶対** (**ダイレクトアドレッシング**)	(n8) (n16) (n24)
9.	**相対**	(PC + d8) (PC + d16)

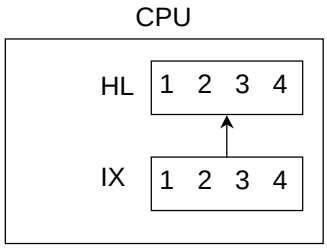
reg 8 : W, A, B, C, D, E, H, L, などの全8ビットレジスタ
reg 16 : WA, BC, DE, HL, IX, IY, IZ, SP, などの全16ビットレジスタ
reg 32 : XWA, XBC, XDE, XHL, XIX, XIY, XIZ, XSP, などの全32ビットレジスタ
reg : XWA, XBC, XDE, XHL, XIX, XIY, XIZ, XSP, などの全32ビットレジスタ
d8 : 8ビットディスプレースメント (–80H～+7FH)
d16 : 16ビットディスプレースメント (–8000H～+7FFFH)
n8 : 8ビット定数 (00H～FFH)
n16 : 16ビット定数 (0000H～FFFFH)
n32 : 32ビット定数 (00000000H～FFFFFFFFH)

(注1) 相対アドレッシングモードは、下記の命令でのみ使用可能です。
LDAR, JR, JRL, DJNZ, CALR

(1) **レジスタ**

オペランドは、指定されたレジスタになります。

例 : LD　HL, IX

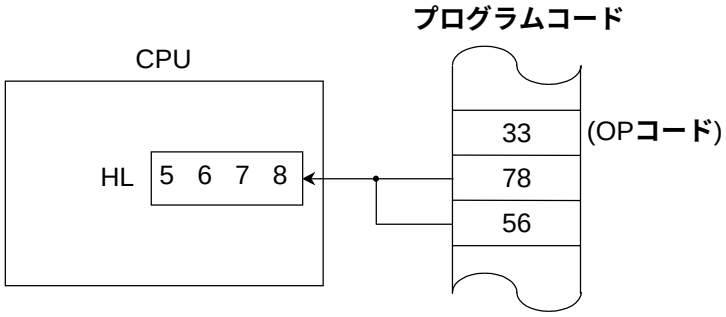


IXレジスタの内容1234Hが、HLレジスタへロードされます。

(2) **イミディエート**

オペランドは、その命令コード中になります。

例 : LD　HL, 5678H

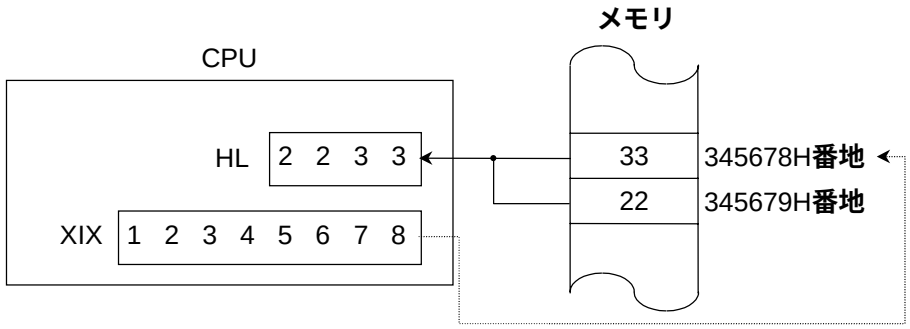


イミディエートデータ5678Hが、HLレジスタへロードされます。

(3) レジスタ間接

オペランドは、レジスタの内容で指定されるメモリ番地になります。

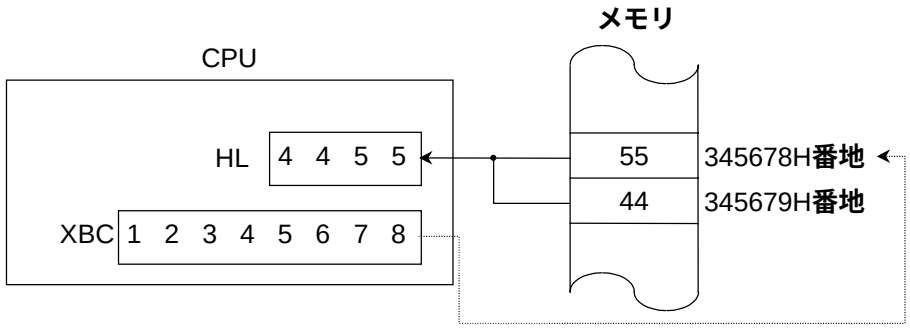
例1 : LD　HL, (XIX)



345678H番地のメモリデータ2233Hが、HLレジスタへロードされます。

バンクレジスタ (XWA, XBC, XDE, XHLレジスタ)をアドレッシングに使うと、ビット0～23の値が、そのままアドレスバスに出力されます。

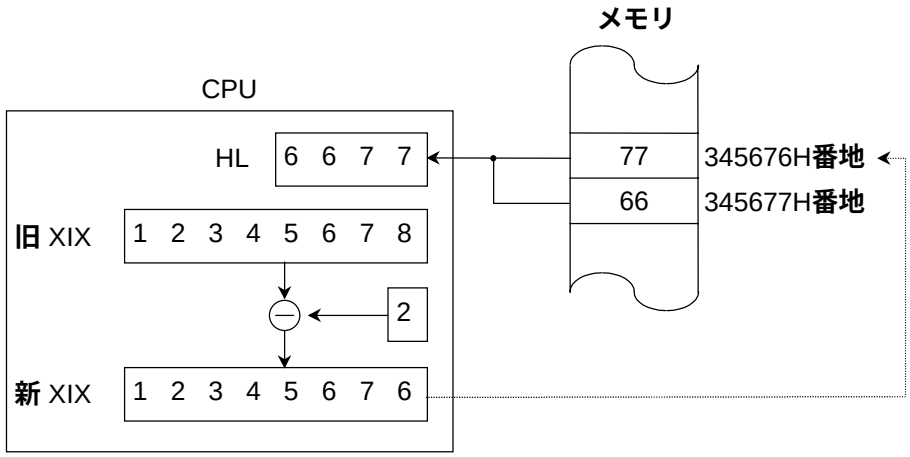
例2 : LD　HL, (XBC)



(4) レジスタ間接プリデクリメント

まず、レジスタ内容がオペランドサイズ分、減算されます。そして、オペランドは、減算されたレジスタの内容で指定されるメモリ番地になります。

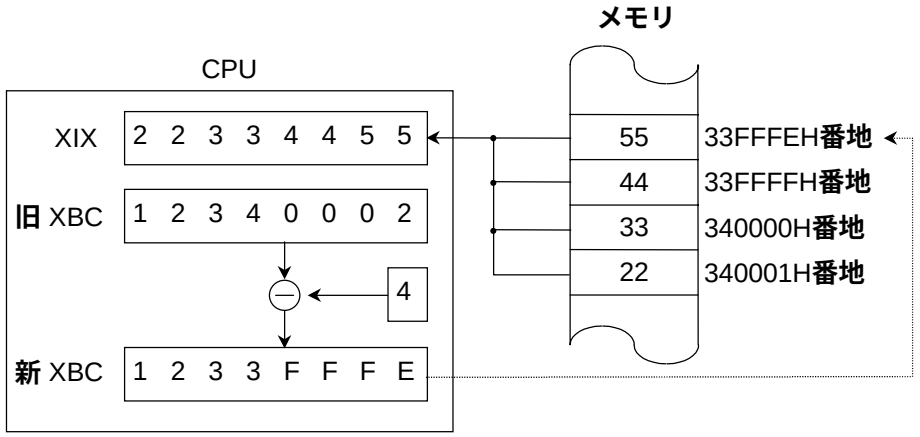
例1 : LD　HL, (–XIX)



プリデクリメント数は、下記のようになります。

オペランドサイズが　バイト　(8ビット)　　のとき = –1
オペランドサイズが　ワード　(16ビット)　のとき = –2
オペランドサイズが　ロング　(32ビット)　のとき = –4

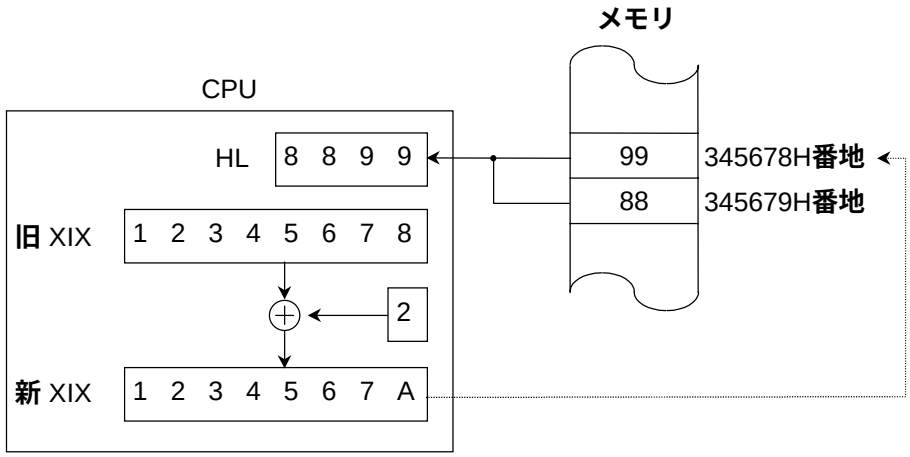
例2 : LD　XIX, (–XBC)



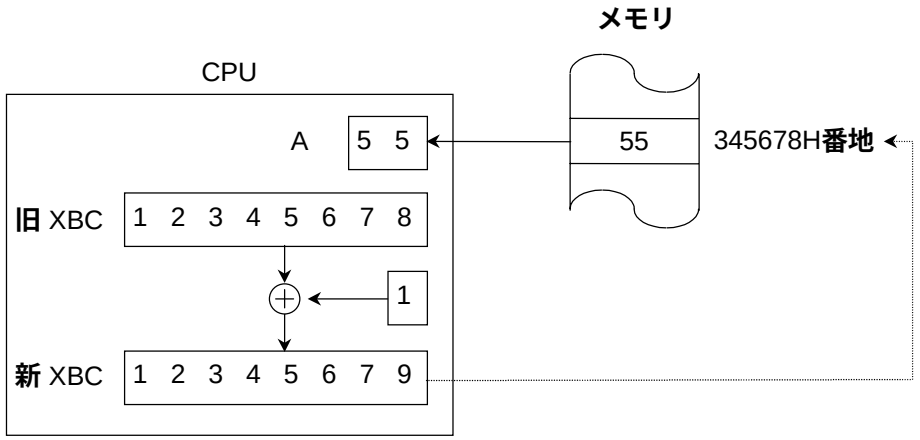
(5) レジスタ間接ポストインクリメント

オペランドは、レジスタの内容で指定されるメモリ番地になります。その後、レジスタの内容がオペランドサイズ分、加算されます。

例1 : LD HL, (XIX +)



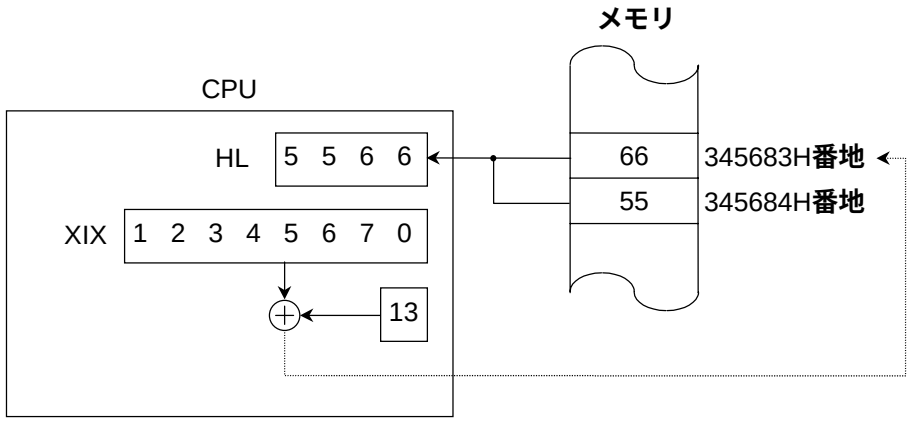
例2 : LD A, (XBC +)



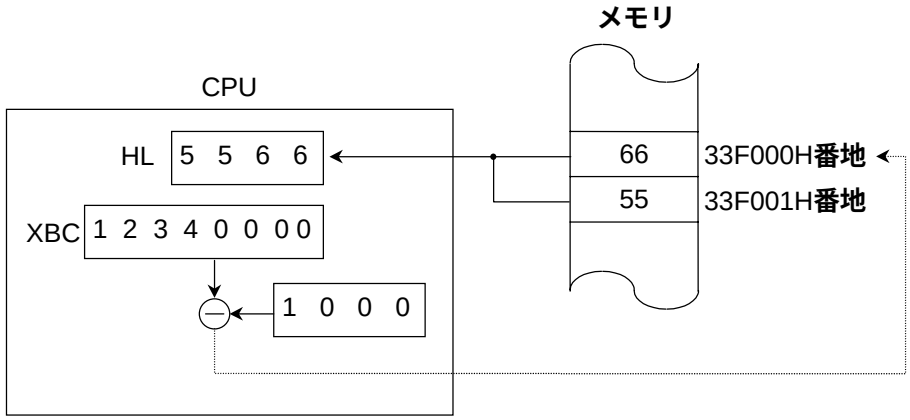
(6) インデックス

オペランドは、命令コード中の8ビットまたは16ビットのディスプレースメント値と、指定されたレジスタの内容を加算して得られるメモリ番地になります。

例1 : LD　HL, (XIX + 13H)



例2 : LD　HL, (XBC – 10000H)

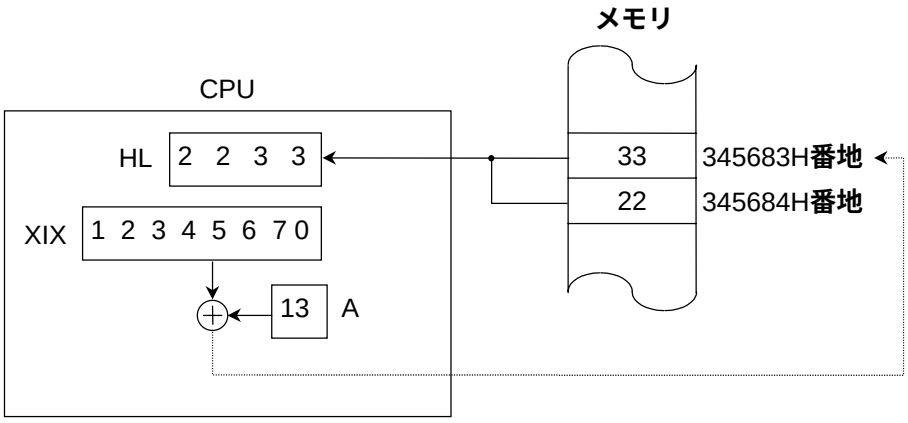


なお、ディスプレースメント値の範囲は、–8000H～+7FFFHに限られます。

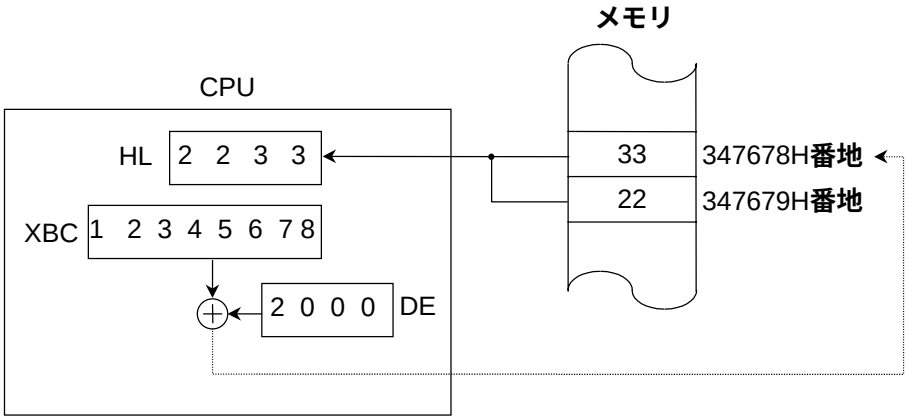
(7) レジスタインデックス

オペランドは、ディスプレースメント (符号付の8ビットまたは16ビット整数) に指定されたレジスタと、ベースに指定されたレジスタの内容を加算して得られるメモリ番地になります。

例1 : LD　HL, (XIX + A)



例2 : LD　HL, (XIX + DE)

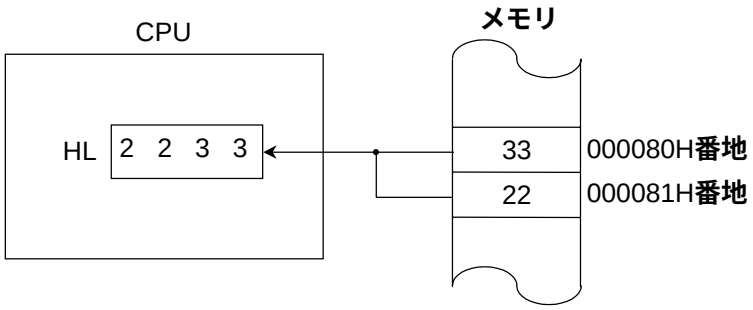


(8) **絶対**

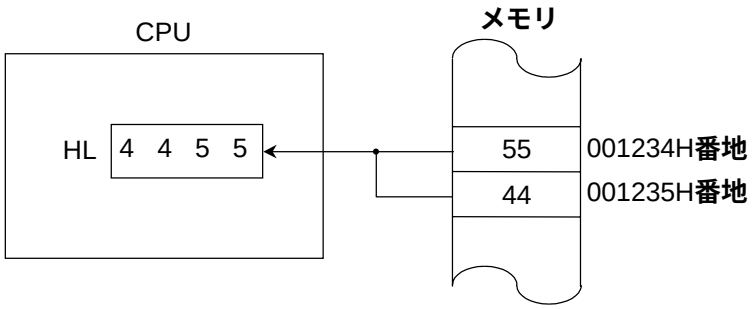
オペランドは、命令コード中の1～3バイトで指定されるメモリ番地になります。000000H～0000FFH 番地は、1バイトで指定できます。000000H～00FFFFH番地は、2バイトで指定でき、000000H～FFFFFFH番地は、3バイトで指定できます。

この中で、1バイトで指定できるメモリ領域(0H～FFH)の256バイト空間へのアドレッシングを、ダイレクトアドレッシングモードと呼んでいます。ダイレクトアドレッシングモードは、プログラムメモリが節約でき、実行時間も短縮できます。

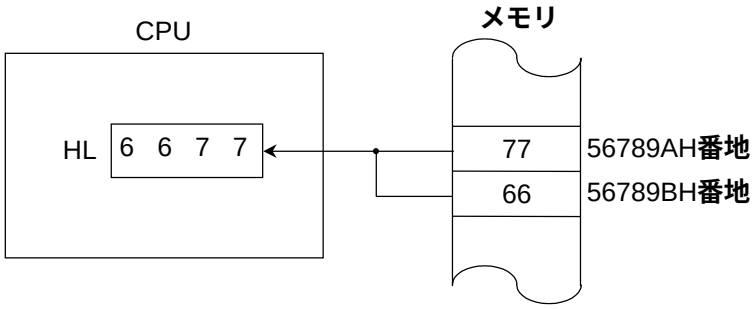
例1 : LD　HL, (80H)



例2 : LD　HL, (1234H)



例3 : LD　HL, (56789AH)



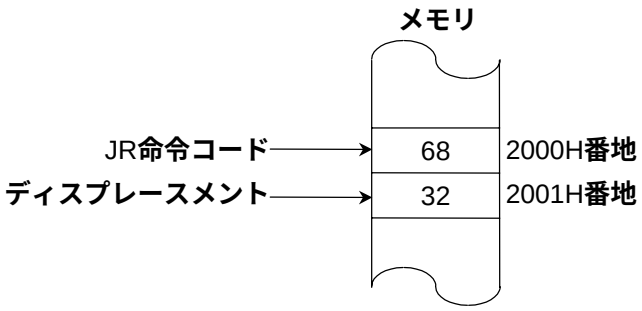
(9) **相対**

オペランドは、現在実行中の命令コードのある番地と、8ビットまたは16ビットのディスプレースメント値を加算して得られるメモリ番地になります。
このアドレッシングモードを使用できる命令は、限定されており、下記の5種類です。

LDAR	R, $ + 4 + d16	
JR	cc, $ + 2 + d8	
JRL	cc, $ + 3 + d16	
CALR	$+3 + d16	
DJNZ	r, $ + 3 + d8	($: 命令コードの先頭番地)

なお、ディスプレースメントのオブジェクトコード値の計算は、命令の種類によって、補正値が異なり(+2～+4)ます。

例1 : JR　2034H



上記の例での、ディスプレースメントのオブジェクトコード値は、

2034H – (2000H + 2)

であり、32Hとなります。

5. 命　令

TLCS-900/L1には、豊富なアドレッシングモードとともに、強力な命令セットを持っています。
基本命令は、大別して以下の9グループに分類されます。

- 転送命令 (8/16/32ビット)
- 交換命令 (8/16ビット)
- ブロック転送/ブロックサーチ命令 (8/16ビット)
- 算術演算命令 (8/16/32ビット)
- 論理演算命令 (8/16/32ビット)
- ビット操作命令 (1ビット)
- 特別演算、CPU制御命令
- ローテート、シフト命令 (8/16/32ビット)
- ジャンプ、コール、リターン命令

表5.1 に、TLCS-900/L1の基本命令を示します。
付録Aに「命令の詳細」を、付録Bに「命令一覧表」を、付録Cに「命令コードマップ」を示します。
また、付録Dに「TLCS-90との相違点」を示します。

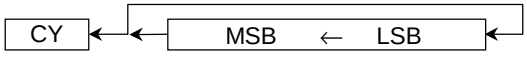
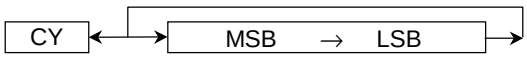
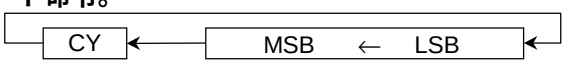
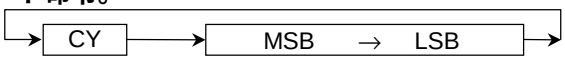
表5.1 TLCS-900/L1基本命令

LD	dst, src	データ転送命令。dst←src
PUSH	src	srcのデータをスタックへPUSHします。 SP←SP–size: (SP) ←src
POP	dst	スタックからdstへデータをPOPします。 dst←(SP) : SP←SP+size
LDA	dst, src	srcの実効アドレスをdstにセットします。
LDAR	dst, PC+dd	PC相対アドレス値をdstにセットします。dst←PC+dd
EX	dst1, dst2	dst1とdst2のデータを交換します。
MIRR	dst	dstのビットパターンをミラー反転します。
LDI		ブロック転送命令。
LDIR		ブロック転送命令(リピート)。
LDD		ブロック転送命令。
LDDR		ブロック転送命令(リピート)。
CPI		ブロック比較命令。
CPIR		ブロック比較命令(リピート)。
CPD		ブロック比較命令。
CPDR		ブロック比較命令(リピート)。
ADD	dst, src	加算命令。 dst←dst+src
ADC	dst, src	拡張加算命令。 dst←dst+src+CY
SUB	dst, src	減算命令。 dst←dst－src
SBC	dst, src	拡張減算命令。 dst←dst－src–CY
CP	dst, src	比較命令。 dst－src
AND	dst, src	論理積命令。 dst←dst AND src
OR	dst, src	論理和命令。 dst←dst OR src
XOR	dst, src	排他的論理和命令。 dst←dst XOR src
INC	imm, dst	増加命令。 dst←dst+imm
DEC	imm, dst	減少命令。 dst←dst－imm
MUL	dst, src	符号なし乗算命令。 dst←dst (low) ×src
MULS	dst, src	符号付き乗算命令。 dst←dst (low) ×src
DIV	dst, src	符号なし除算命令。 dst (low) ←dst÷src dst (high) ←余り 0による除算またはオーバフローで、Vフラグがセットされます。
DIVS	dst, src	符号付き除算命令。 dst (low) ←dst÷src dst (high) ←余り; 符号は常に被除数と同じです。 0による除算またはオーバフローで、Vフラグがセットされます。

MULA	dst	**符号付き積和演算命令。** dst ← dst+ (XDE) × (XHL–) 32bit 32bit 16bit 16bit
MINC1	num, dst	**モジュロインクリメント** (+1)。
MINC2	num, dst	**モジュロインクリメント** (+2)。
MINC4	num, dst	**モジュロインクリメント** (+4)。
MDEC1	num, dst	**モジュロデクリメント** (–1)。
MDEC2	num, dst	**モジュロデクリメント** (–2)。
MDEC4	num, dst	**モジュロデクリメント** (–4)。
NEG	dst	2**の補数。**dst←0－dst
CPL	dst	1**の補数。**dst←not dst
EXTZ	dst	**ゼロ拡張。**dst**の上位データを**0**にセットします。**
EXTS	dst	**符号拡張。**dst**の下位データの**MSB**値を上位にコピーします。**
DAA	dst	10**進補正。**
PAA	dst	**ポインタ補正。**dst**が奇数のとき、**+1**を加えて偶数にします。** if dst (0) =1 then dst←dst+1.
LDCF	bit, src	src<bit>**の値を、**C**フラグにコピーします。**
STCF	bit, dst	C**フラグの値を、**dst<bit>**へコピーします。**
ANDCF	bit, src	src<bit>**の値と**C**フラグの**AND**をとり、**C**フラグへ格納します。**
ORCF	bit, src	src<bit>**の値と**C**フラグの**OR**をとり、**C**フラグへ格納します。**
XORCF	bit, src	src<bit>**の値と**C**フラグの**XOR**をとり、**C**フラグへ格納します。**
RCF		C**フラグを**0**に、リセットします。**
SCF		C**フラグを**1**に、セットします。**
CCF		C**フラグの値を、反転します。**
ZCF		Z**フラグの反転値を、**C**フラグへコピーします。**
BIT	bit, src	**ビットテスト命令。**Z**フラグ**←not src<bit>
RES	bit, dst	**ビットリセット命令。** dst<bit>←0
SET	bit, dst	**ビットセット命令。** dst<bit>←1
CHG	bit, dst	**ビットチェンジ命令。** dst<bit>←not dst<bit>
TSET	bit, dst	**ビットテスト**&**セット命令。** Z**フラグ** ←not dst<bit> dst<bit> ←1

BS1F	A, dst	dst中にある最初の1のビットを、Forward (LSB) からサーチし、そのビット番号をAレジスタにセットします。
BS1B	A, dst	dst中にある最初の1のビットを、Backward (MSB) からサーチし、そのビット番号をAレジスタにセットします。
NOP		ノーオペレーション。
EI	imm	割り込みフラグの設定をします。 IFF←imm
DI		マスカブル割り込みを禁止します。 IFF←7
PUSH	SR	ステータスレジスタをPUSHします。
POP	SR	ステータスレジスタをPOPします。
SWI	imm	ソフトウエア割り込み。
HALT		CPUを停止します。
LDC	CTRL−REG, reg	レジスタの内容を、CPUのコントロールレジスタへコピーします。
LDC	reg, CTRL−REG	CPUのコントロールレジスタの内容を、レジスタへコピーします。
LDX	dst, src	抜き取り転送命令。dst←src

LINK	reg, dd	スタックフレームを生成します。
		PUSH reg
		LD reg, XSP
		ADD XSP, dd

UNLK	reg	スタックフレームを削除します。 LD XSP, reg POP reg
LDF	imm	レジスタバンクを指定します。 RFP←imm
INCF		新しいレジスタバンクへ移ります。 RFP←RFP+1
DECF		以前のレジスタバンクへ戻ります。 RFP←RFP−1
SCC	cc, dst	コンディションコードによるdstのセットを行います。 if cc then dst ←1 else dst ←0.
RLC	num, dst	ローテート命令。 
RRC	num, dst	ローテート命令。 
RL	num, dst	ローテート命令。 
RR	num, dst	ローテート命令。 

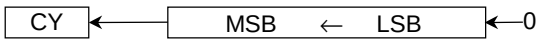
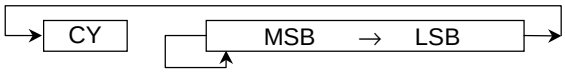
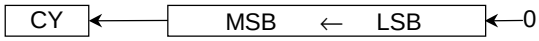
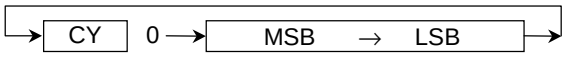
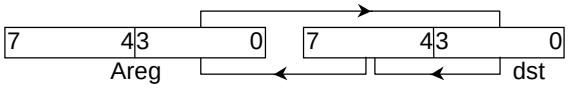
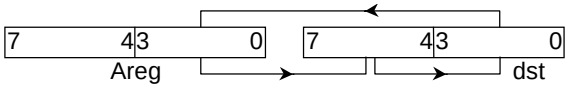
SLA	num, dst	**シフト命令。** 
SRA	num, dst	**シフト命令。** 
SLL	num, dst	**シフト命令。** 
SRL	num, dst	**シフト命令。** 
RLD	dst	**ローテートデジット命令。** 
RRD	dst	**ローテートデジット命令。** 
JR	cc, PC+d	**相対ジャンプ命令** (8**ビットディスプレースメント**)。 if cc then PC←PC+d.
JRL	cc, PC+dd	**相対ロングジャンプ命令** (16**ビットディスプレースメント**)。 if cc then PC←PC+dd.
JP	cc, dst	**ジャンプ命令。** if cc then PC←dst.
CALR	RC+dd	**相対コール命令** (16**ビットディスプレースメント**)。 PUSH PC: PC←PC+dd.
CALL	cc, dst	**コール命令。** if cc then PUSH PC: PC←dst.
DJNZ	dst, PC+d	**デクリメント&相対ジャンプ命令。** dst←dst−1 if dst≠0 then PC←PC+d.
RET	cc	**リターン命令。** if cc then POP PC.
RETD	dd	**リターン&引き数領域の削除命令。** RET XSP←XSP+dd
RETI		**割り込み用リターン命令。** POP SR&PC

表5.2 命令一覧

BWL	LD	reg, reg	BWL	INC	imm3, reg	---	NOP	
BWL	LD	reg, imm		DEC	imm3, mem.B/W			
BWL	LD	reg, mem						
BWL	LD	mem, reg				---	EI	[imm3]
BW-	LD	mem, imm				---	DI	
BW-	LD	(nn), mem	BW-	MUL	reg, reg	-W-	*PUSH	SR
BW-	LD	mem, (nn)		*MULS	reg, imm	-W-	*POP	SR
				DIV	reg, mem	---	SWI	[imm3]
				*DIVS		---	HALT	
BWL	PUSH	reg/F				BWL	*LDC	CTRL-R, reg
BW-	PUSH	imm	-W-	*MULA	reg	BWL	*LDC	reg, CTRL-R
BW-	PUSH	mem				B--	*LDX	(n), n
			-W-	*MINC1	imm, reg			
BWL	POP	reg/F	-W-	*MINC2	imm, reg	--L	*LINK	reg, dd
BW-	POP	mem	-W-	*MINC4	imm, reg	--L	*UNLK	reg
			-W-	*MDEC1	imm, reg	---	*LDF	imm3
			-W-	*MDEC2	imm, reg	---	*INCF	
-WL	LDA	reg, mem	-W-	*MDEC4	imm, reg	---	*DECF	
-WL	LDAR	reg, PC + dd				BW-	*SCC	cc, reg
			BW-	NEG	reg			
			BW-	CPL	reg	BWL	RLC	imm, reg
			-WL	*EXTZ	reg		RRL	A, reg
B--	EX	F, F'	-WL	*EXTS	reg		RL	mem. B/W
BW-	EX	reg, reg	B--	DAA	reg		RR	
BW-	EX	mem, reg	-WL	*PAA	reg		SLA	
							SRA	
							SLL	
-W-	*MIRR	reg	BW-	*LDCF	imm, reg		SRL	
				*STCF	A, reg			
				*ANDCF	imm, mem.B	B--	RLD	[A,] mem
				*ORCF	A, mem.B	B--	RRD	[A,] mem
BW-	LDI			*XORCF				
BW-	LDIR							
BW-	LDD		---	RCF		---	JR	[cc,] PC + d
BW-	LDDR		---	SCF		---	JRL	[cc,] PC + dd
			---	CCF		---	JP	[cc,] mem
			---	* ZCF		---	CALR	PC + dd
BW-	CPI					---	CALL	[cc,] mem
BW-	CPIR		BW-	BIT	imm, reg			
BW-	CPD			RES	imm, mem.B	BW-	DJNZ	[reg], PC + d
BW-	CPDR			SET				
				*CHG		---	RET	[cc]
				TSET		---	*RETD	dd
BWL	ADD	reg, reg				---	RETI	
	ADC	reg, imm	-W-	*BS1F	A, reg			
	SUB	reg, mem		*BS1B				
	SBC	mem, reg						
	CP	mem, imm.B/W						
	AND							
	OR							
	XOR							

B = Byte (8 bit), W = Word (16 bit), L = Long–Word (32bit)

* : TLCS-90**より追加された命令であることを示します。**

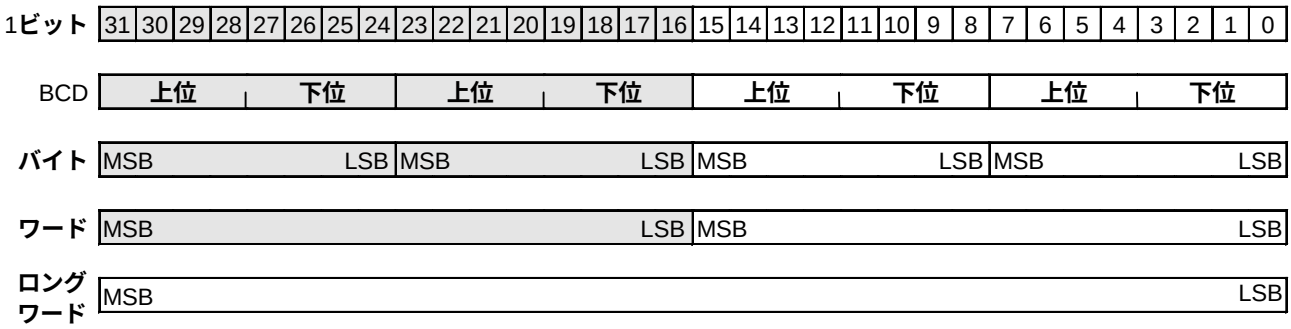
[ ] : **省略可能を示します。**

6. データ構成

TLCS-900/L1は、1/4/8/16/32ビットの各種サイズのデータを扱うことができます。

(1) レジスタのデータフォーマット

<データイメージ>



(注1) 網掛け部分のアクセスは、網掛けなしの部分のアクセスに比べて、命令コード長が1バイト長くなります。

(2) メモリのデータフォーマット

<データイメージ>

(注2) メモリ上でワードデータ/ロングワードデータを扱う場合、ワードアライメントの制限はありません。偶数番地または奇数番地のどちらからでも配置できます。

(注3) スタックエリアへデータをセーブする命令(PUSH)を実行した場合、まずスタックポインタをデクリメントした後、データをセーブします。

```
例: PUSH  HL; XSP      ←XSP – 2
              (XSP)    ←L
              (XSP + 1) ←H
```

これは、プリデクリメントのアドレッシングモードの場合も同じです。
TLCS-90では、その動作が逆で、セーブした後、スタックポインタをデクリメントします。

```
例: PUSH  HL; (XSP – 1) ←H
              (XSP – 2) ←L
              XSP       ←XSP – 2
```

(3) ダイナミックバスサイジング

TLCS-900/L1では、8ビットデータバスと16ビットデータバスを、ダイナミック(各バスサイクルごと)に切り替えることができます。これを、ダイナミックバスサイジングと呼びます。

この機能により、TLCS-900/L1では、外部に8ビットデータバス幅のメモリと16ビットデータバス幅のメモリを、混在して拡張することが可能です。

チップセレクト/ウェイトコントローラを内蔵した製品では、それにより各アドレス領域ごとに外部データバス幅の制御を行うことが可能です。

表 6.1 ダイナミックバスサイジング

オペランドデータ幅	オペランドスタート番地	メモリ幅データ幅	CPU アドレス	CPU データ	
				D15 – D8	D7 – D0
8 ビット	2n + 0 (偶数)	8 ビット	2n + 0	xxxxx	b7 – b0
		16 ビット	2n + 0	xxxxx	b7 – b0
	2n + 1 (奇数)	8 ビット	2n + 1	xxxxx	b7 – b0
		16 ビット	2n + 1	b7 – b0	xxxxx
16 ビット	2n + 0 (偶数)	8 ビット	2n + 0	xxxxx	b7 – b0
			2n + 1	xxxxx	b15 – b8
		16 ビット	2n + 0	b15 – b8	b7 – b0
	2n + 1 (奇数)	8 ビット	2n + 1	xxxxx	b7 – b0
			2n + 2	xxxxx	b15 – b8
		16 ビット	2n + 1	b7 – b0	xxxxx
			2n + 2	xxxxx	b15 – b8
32 ビット	2n + 0 (偶数)	8 ビット	2n + 0	xxxxx	b7 – b0
			2n + 1	xxxxx	b15 – b8
			2n + 2	xxxxx	b23 – b16
			2n + 3	xxxxx	b31 – b24
		16 ビット	2n + 0	b15 – b8	b7 – b0
			2n + 2	b31 – b24	b23 – b16
	2n + 1 (奇数)	8 ビット	2n + 1	xxxxx	b7 – b0
			2n + 2	xxxxx	b15 – b8
			2n + 3	xxxxx	b23 – b16
			2n + 4	xxxxx	b31 – b24
		16 ビット	2n + 1	b7 – b0	xxxxx
			2n + 2	b23 – b16	b15 – b8
			2n + 4	xxxxx	b31 – b24

xxxxx : リード時は、そのバスの入力データが無視されることを示します。ライト時は、そのバスがハイインピーダンスで、そのバスのライトストローブ信号はノンアクティブのままであることを示します。

(4) 内部データバスの構成

TLCS-900/L1では、CPUと内蔵メモリ(内蔵ROMまたは内蔵RAM)は16ビットの内部データバスで接続されています。内蔵メモリは、0ウェイトで動作します。

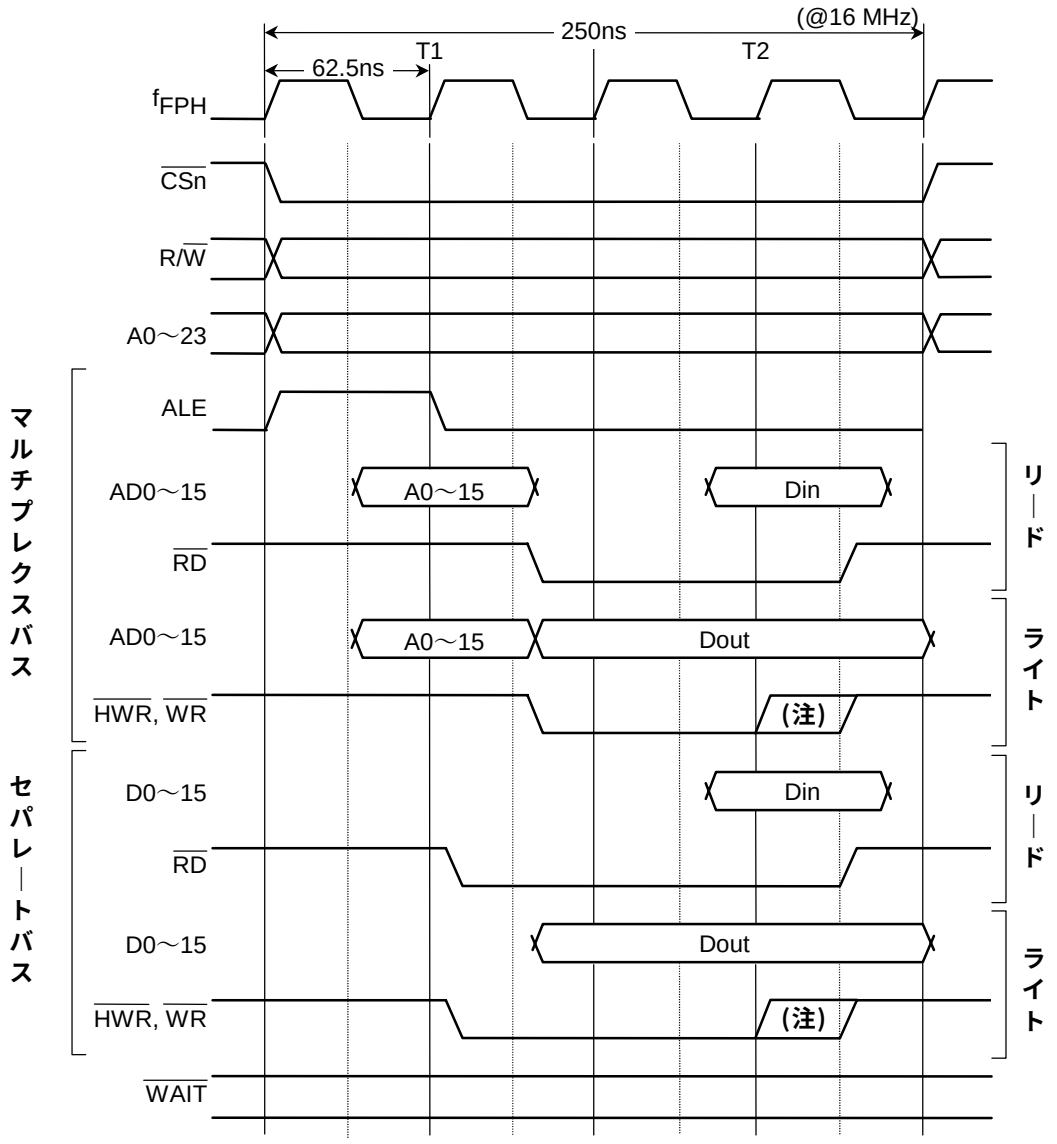
一方、内蔵I/Oとは8ビットの内部データバスで接続されています。これは、内蔵I/Oのアクセススピードがほとんどシステム全体の動作スピードに影響を与えないからです。システム全体の動作スピードは、プログラムメモリのアクセススピードに大きく依存します。

7. 基本タイミング

TLCS-900/L1には、以下の基本タイミングがあります。

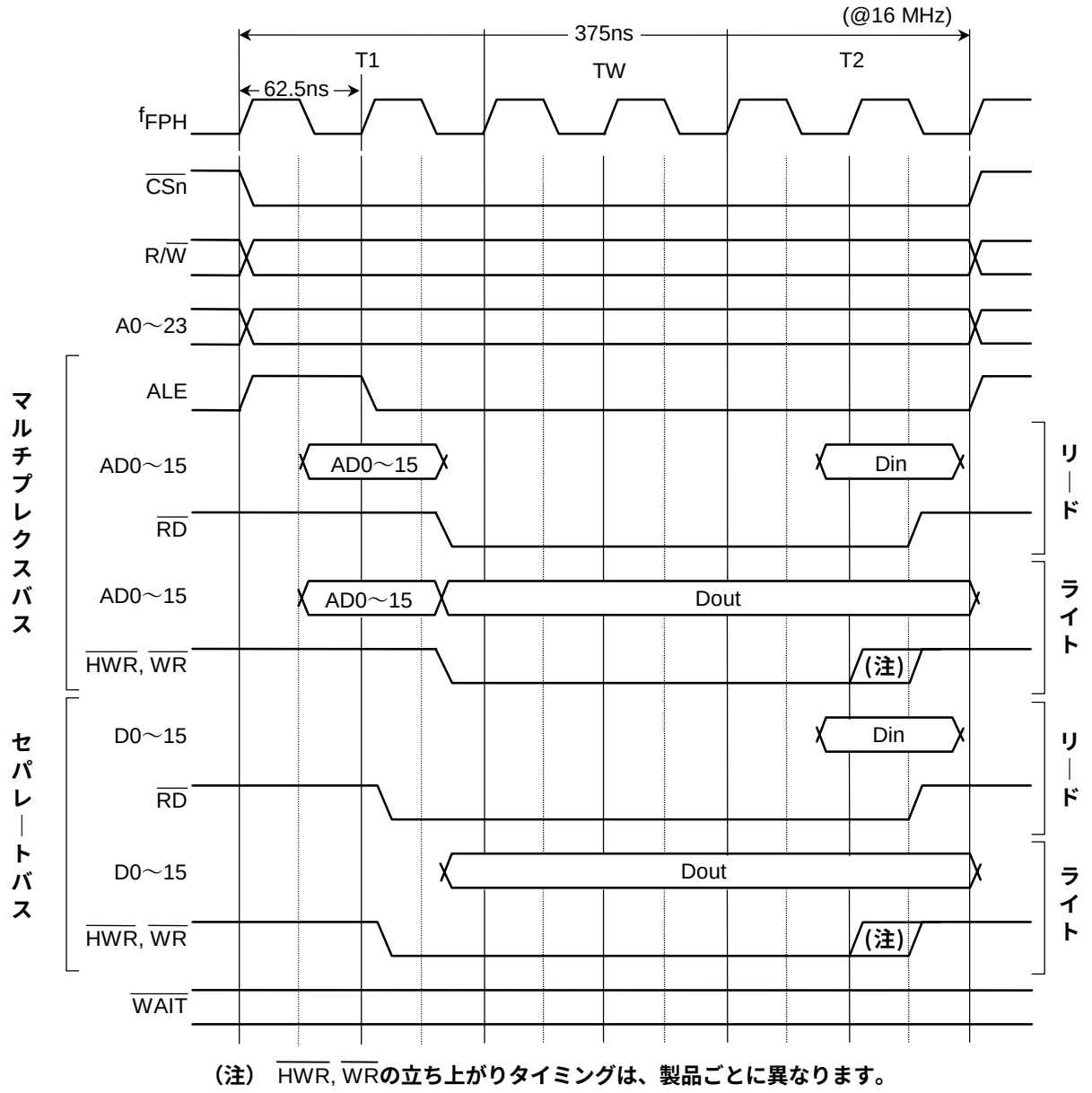
- リードサイクル
- ライトサイクル
- ダミーサイクル
- 割り込み受け付けタイミング
- リセット

図7.1～7.8に、それぞれの基本タイミングを示します。



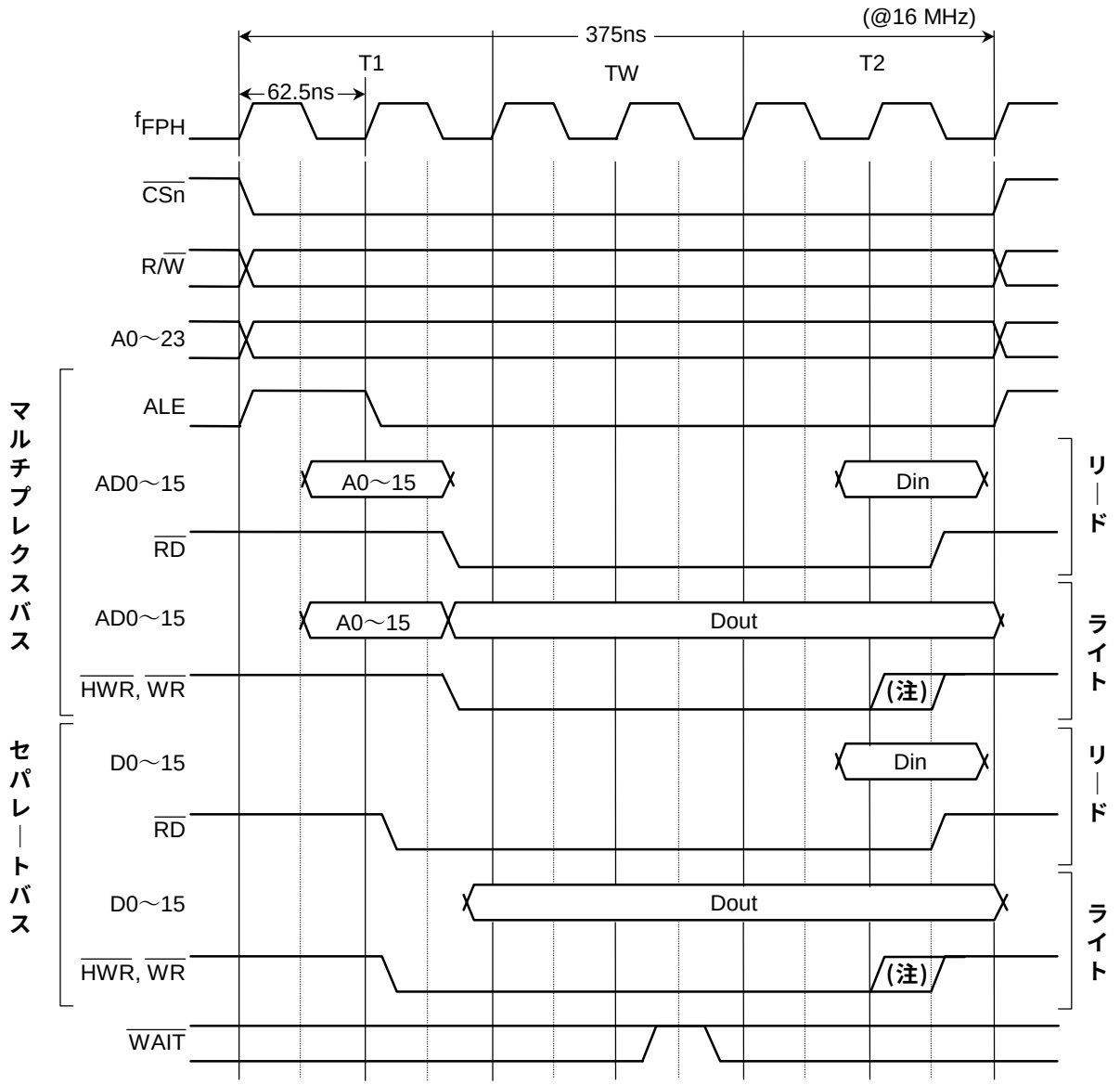
（注） $\overline{\text{HWR}}$, $\overline{\text{WR}}$の立ち上がりタイミングは、製品ごとに異なります。

図 7.1 0 WAIT **のリードサイクル/ライトサイクル**



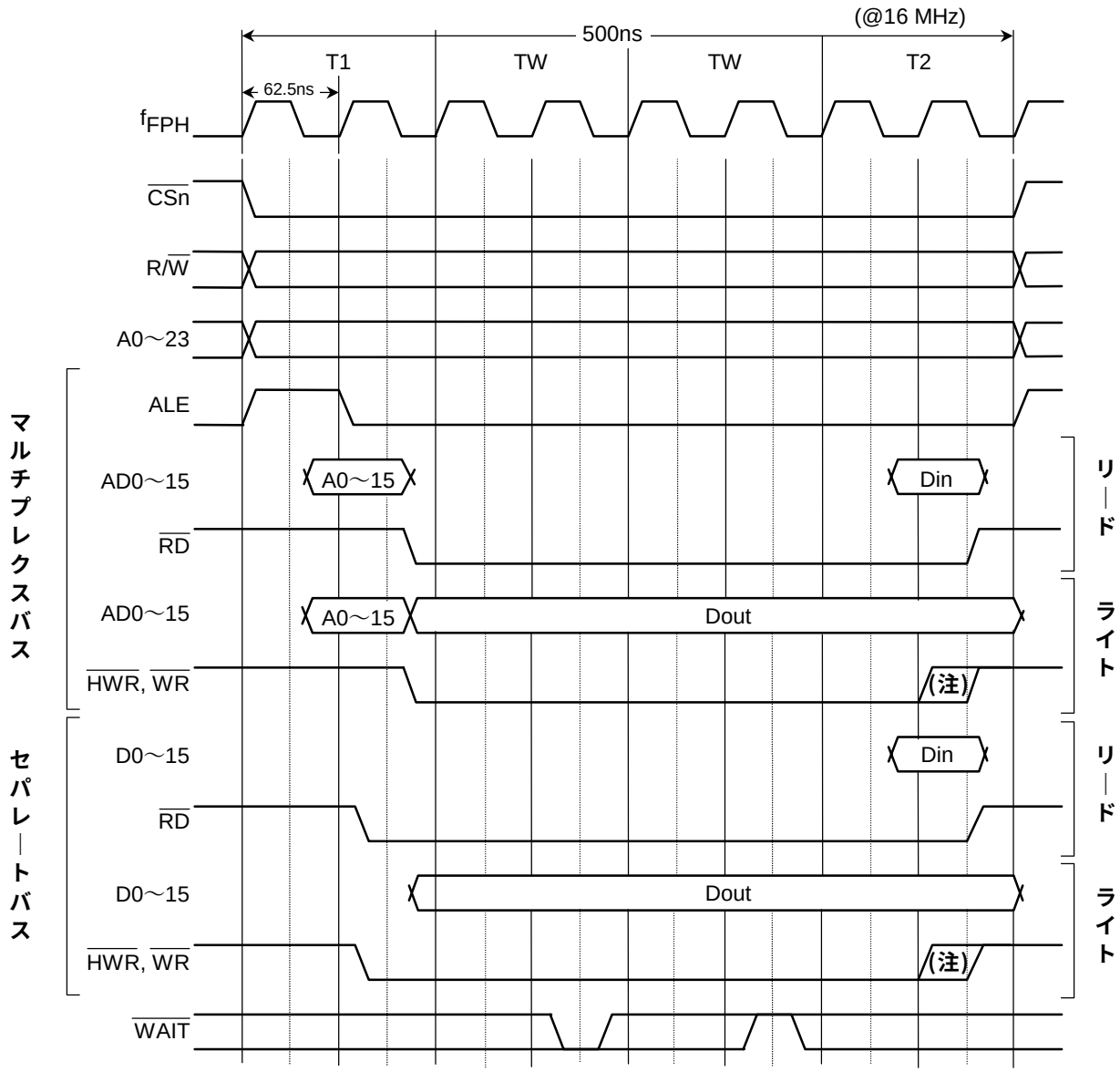
（注） HWR, WRの立ち上がりタイミングは、製品ごとに異なります。

図 7.2　1WAIT **のリード/ライトサイクル**



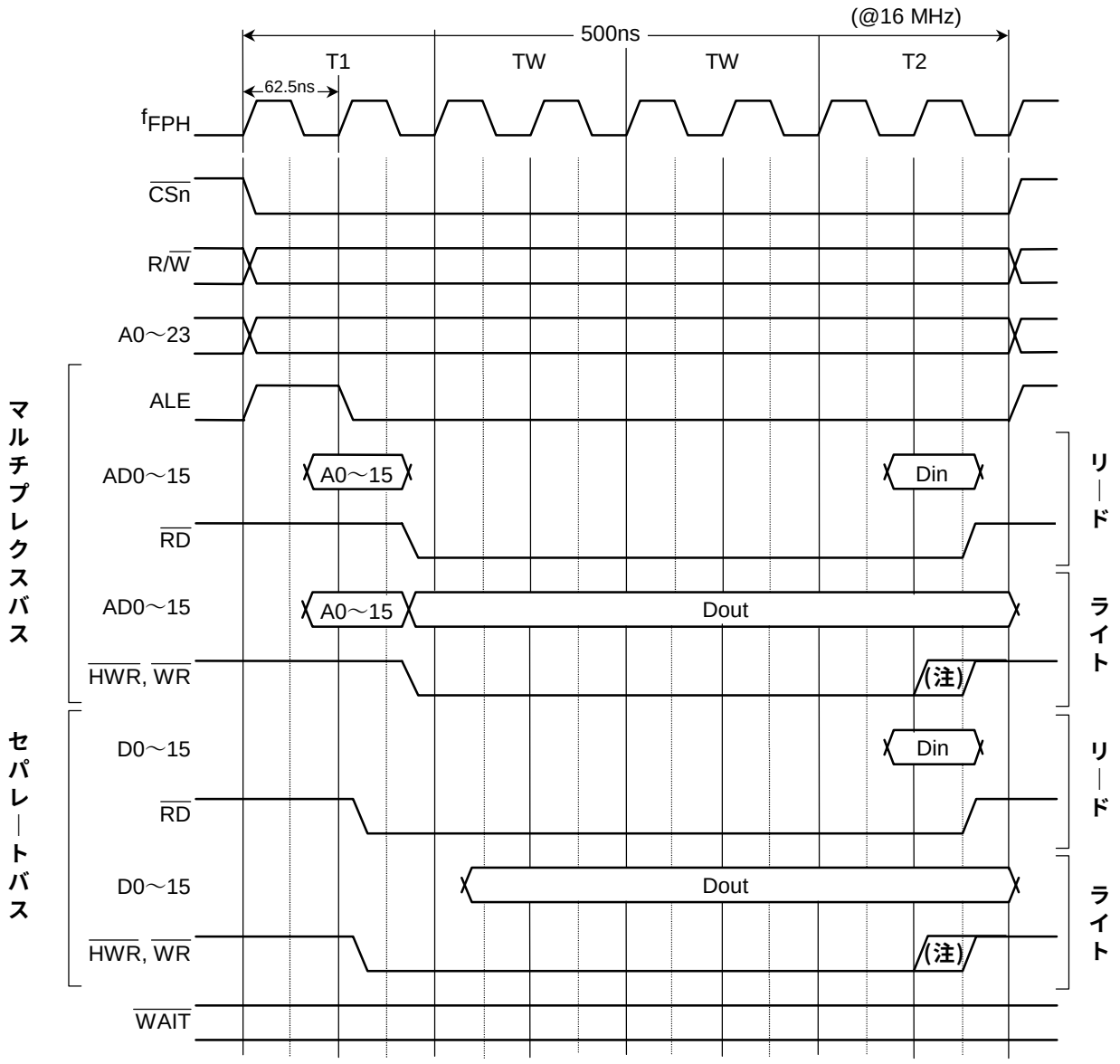
(注) $\overline{\text{HWR}}$, $\overline{\text{WR}}$**の立ち上がりタイミングは、製品ごとに異なります。**

図 7.3 1WAIT + n **のリード/ライトサイクル** (n = 0 **の場合**)



（注） $\overline{HWR}$, $\overline{WR}$の立ち上がりタイミングは、製品ごとに異なります。

図 7.4　1WAIT + n **のリード/ライトサイクル** (n = 1 **の場合**)



（注） HWR, WRの立ち上がりタイミングは、製品ごとに異なります。

図 7.5 2WAIT のリード/ライトサイクル

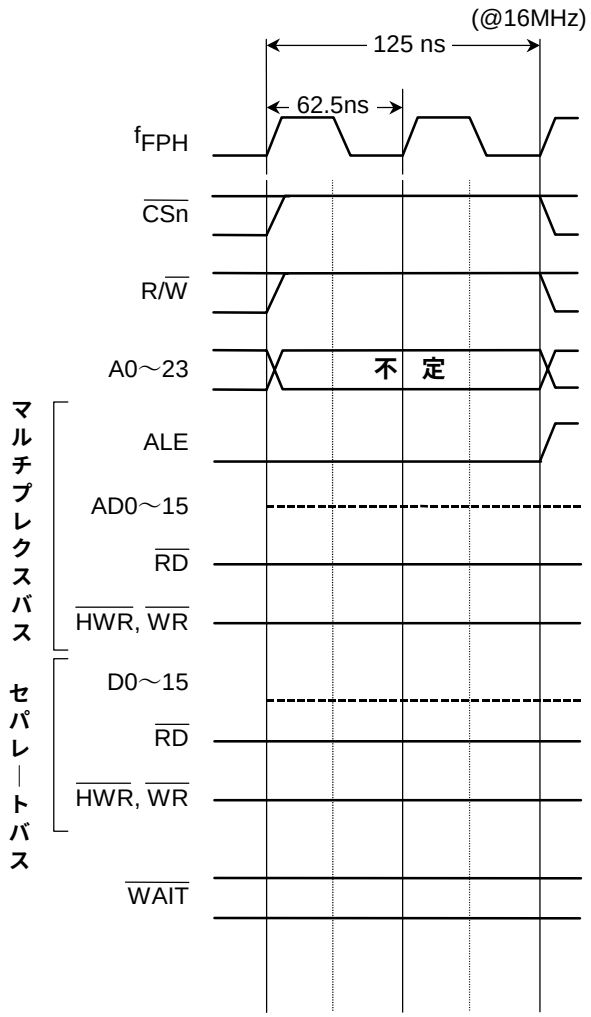
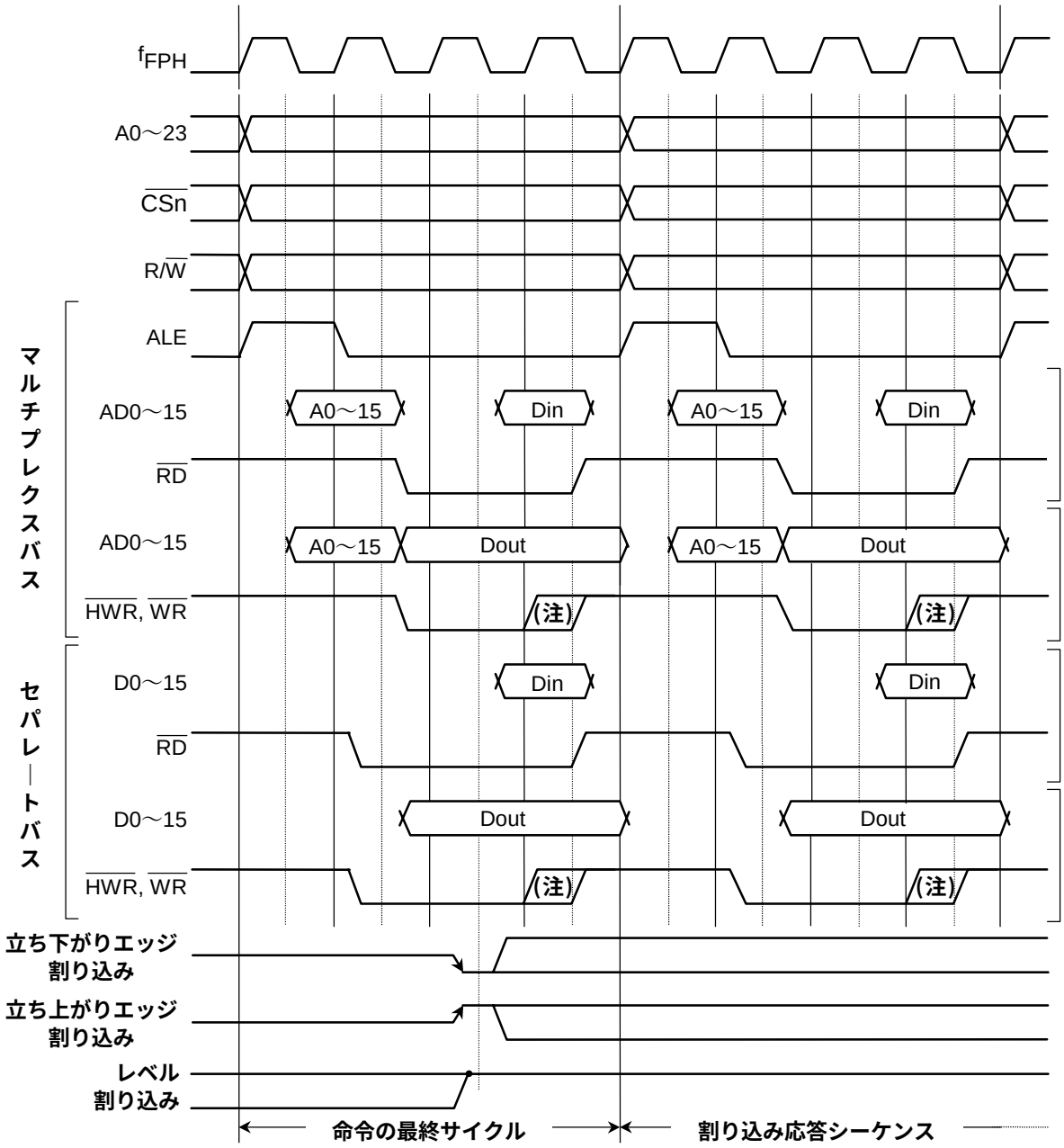


図 7.6 1 **ステートのダミーサイクル**



(注) このタイミング図は、典型例です。実際は、CPU内部のバスインタフェースユニットの働きにより、外部バスタイミングと内部割り込み受け付けタイミングは、一対一に対応しません。
また、$\overline{\text{HWR}}$, $\overline{\text{WR}}$の立ち上がりタイミングは、製品ごとに異なります。

図 7.7 割り込み受け付けのタイミング

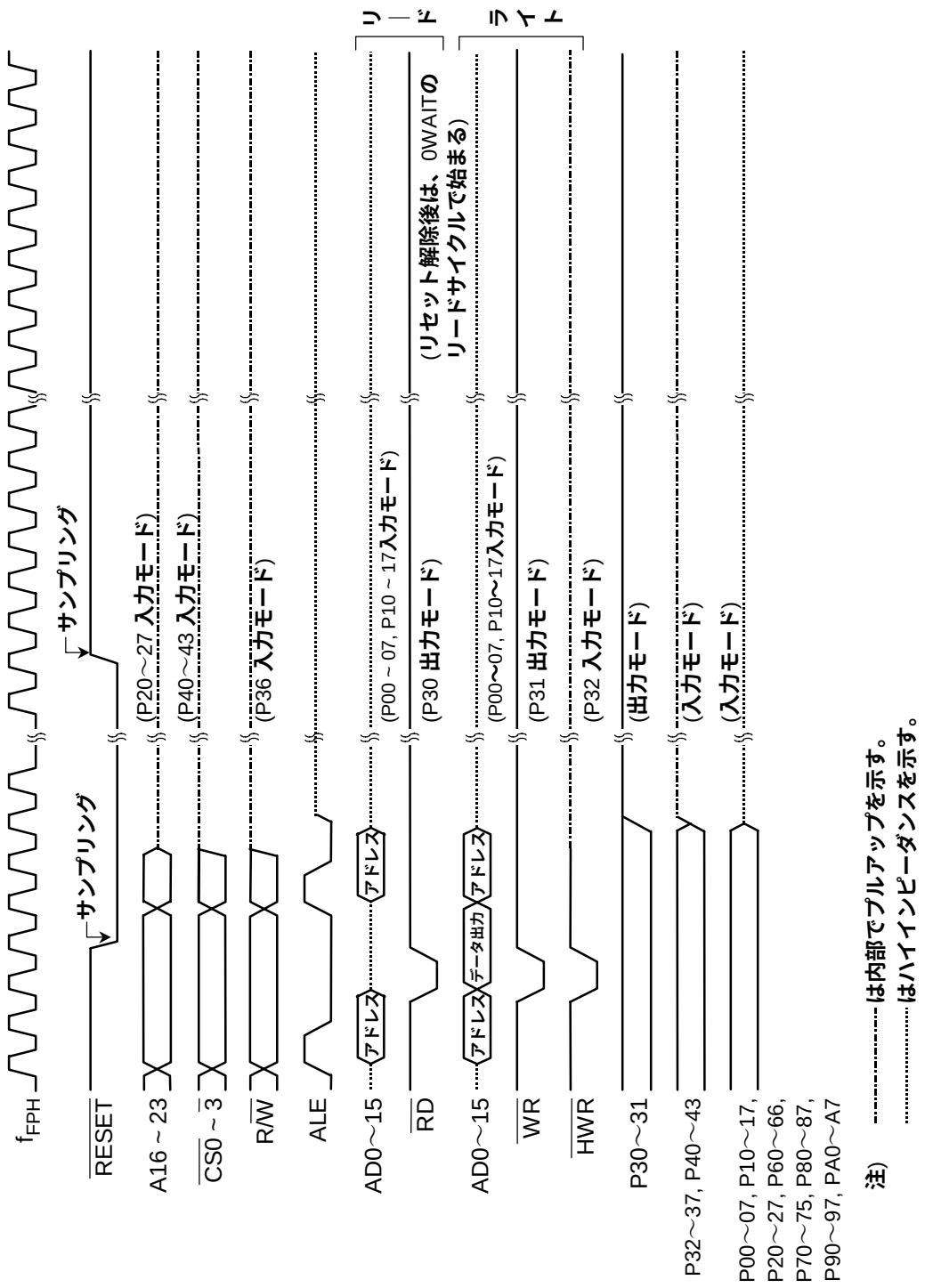


図 7.8 **リセットのタイミング例** (**内蔵** ROM **動作** : TMP91CW12)

付録A. 命令の詳細

■ 命令リスト

① 転送

LD	PUSH	POP	LDA	LDAR			

② 交換

EX	MIRR						

③ ブロック転送/ブロックサーチ

LDI	LDIR	LDD	LDDR	CPI	CPIR	CPD	CPDR

④ 算術演算

ADD	ADC	SUB	SBC	CP	INC	DEC	NEG
EXTZ	EXTS	DAA	PAA	MUL	MULS	DIV	DIVS
MULA	MINC	MDEC					

⑤ 論理演算

AND	OR	XOR	CPL				

⑥ ビット操作

LDCF	STCF	ANDCF	ORCF	XORCF	RCF	SCF	CCF
ZCF	BIT	RES	SET	CHG	TSET	BS1	

⑦ 特別演算，CPU制御

NOP	EI	DI	PUSH_SR	POP_SR	SWI		
HALT	LDC	LDX	LINK	UNLK	LDF	INCF	DECF
SCC							

⑧ ローテート，シフト

RLC	RRC	RL	RR	SLA	SRA	SLL	SRL
RLD	RRD						

⑨ ジャンプ，コール，リターン

JP	JR	JRL	CALL	CALR	DJNZ	RET	RETD
RETI							

■ 本文中の記号の説明

本文中で使われている各記号の意味は、下記のとおりです。

dst	デスティネーション。データの転送先または演算結果の格納先を示します。
src	ソース。データの転送元または演算データの読み出し元を示します。
num	ナンバー。数値であることを示します。
condition	コンディション。フラグによる条件記号を示します。
R	8/16/32 ビットのカレントバンクレジスタを含む 8 本の汎用レジスタ 8 ビットの場合: W, A, B, C, D, E, H, L の 8 本のみ 16 ビットの場合: WA, BC, DE, HL, IX, IY, IZ, SP の 8 本のみ 32 ビットの場合: XWA, XBC, XDE, XHL, XIX, XIY, XIZ, XSP の 8 本のみ
r	8/16/32 ビットの汎用レジスタ (4, 5 ページ後の「レジスタマップ」で示されたすべてのレジスタ)
r16	16 ビットの汎用レジスタ (4, 5 ページ後の「レジスタマップ」で示されたすべての 16 ビットレジスタ)
r32	32 ビットの汎用レジスタ (4, 5 ページ後の「レジスタマップ」で示されたすべての 32 ビットレジスタ)
cr	8/16/32 ビットの CPU の全コントロールレジスタ DMAS0〜3, DMAD0〜3, DMAC0〜3, DMAM0〜3, INTNEST
A	A レジスタ(8 ビット)
F	フラグレジスタ(8 ビット)
F′	裏フラグレジスタ(8 ビット)
SR	ステータスレジスタ(16 ビット)
PC	プログラムカウンタ(ミニマムモード=16 ビット,マキシマムモード=32 ビット)
(mem)	8/16/32 ビットのメモリ内のデータ
mem	実効アドレス値
<W>	オペランドサイズがワードのとき W の記述が必要であることを示します。
[ ]	カッコ内のオペランドの記述が、省略できることを示します。
#	8/16/32 ビットの即値データ
#3	3 ビットの即値データ; 0〜7 or1〜8 ・・・・・・・・・・・・・・・・・ 短縮コード用
#4	4 ビットの即値データ; 0〜15or1〜16
d8	8 ビットのディスプレースメント; –80H〜+7FH
d16	16 ビットのディスプレースメント; –8000H〜+7FFFH
cc	コンディションコード
CY	キャリーフラグ
Z	ゼロフラグ
(#8)	ダイレクトアドレッシング ; (00H)〜(0FFH) ・・・・・・・ 256 バイト空間
(#16)	64KB エリアアドレッシング ; (0000H)〜(0FFFFH)
(-r32)	プリデクリメントアドレッシング
(r32+)	ポストインクリメントアドレッシング
$	その命令が置かれている先頭アドレス

■ オブジェクトコード中の記号の説明

オブジェクトコードの中で使われている各記号の意味は、下記のとおりです。

Z
ZZ
ZZZ
S

オペランドサイズ指定コード

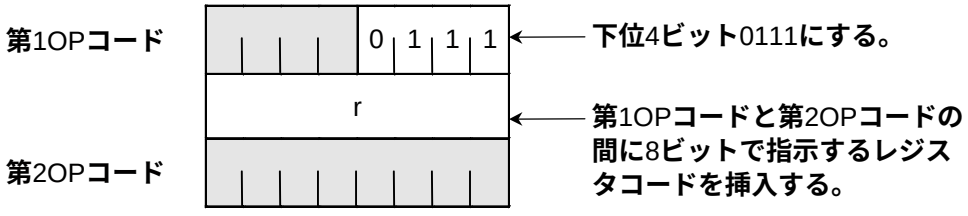
	バイト	ワード	ロング
Z	0	1	–
ZZ	00	01	10
ZZZ	010	011	100
S	–	0	1

R
r

レジスタ指定コード

コード	バイト	ワード	ロング
000	W	WA	XWA
001	A	BC	XBC
010	B	DE	XDE
011	C	HL	XHL
100	D	IX	XIX
101	E	IY	XIY
110	H	IZ	XIZ
111	L	SP	XSP

補足: r で示されるレジスタは、上記以外に、拡張コードを使うと、すべてのレジスタを指定することができます。なお、この場合、実行ステート数は+1 増加します。下記に、そのフォーマットを示します。



なお、r のコード値は、
ワードレジスタとしてアクセスするときは 2 の倍数
ロングレジスタとしてアクセスするときは 4 の倍数
8 ビットで指示されるレジスタは、後述のレジスタマップを参照してください。

m

メモリアドレッシングモード指定コード

(XWA)	=	-0- -0000				
(XBC)	=	-0- -0001				
(XDE)	=	-0- -0010				
(XHL)	=	-0- -0011				
(XIX)	=	-0- -0100				
(XIY)	=	-0- -0101				
(XIZ)	=	-0- -0110				
(XSP)	=	-0- -0111				
(XWA+d8)	=	-0- -1000	d<7:0>			
(XBC+d8)	=	-0- -1001	d<7:0>			
(XDE+d8)	=	-0- -1010	d<7:0>			
(XHL+d8)	=	-0- -1011	d<7:0>			
(XIX+d8)	=	-0- -1100	d<7:0>			
(XIY+d8)	=	-0- -1101	d<7:0>			
(XIZ+d8)	=	-0- -1110	d<7:0>			
(XSP+d8)	=	-0- -1111	d<7:0>			
(#8)	=	-1- -0000	#<7:0>			
(#16)	=	-1- -0001	#<7:0>	#<15:8>		
(#24)	=	-1- -0010	#<7:0>	#<15:8>	#<23:16>	
(r32)	=	-1- -0011	r32′	00		
(r32+d16)	=	-1- -0011	r32′	01	d<7:0>	d<15:8>
(r32+r8)	=	-1- -0011	000000	11	r32	r8
(r32+r16)	=	-1- -0011	000001	11	r32	r16
(-r32)	=	-1- -0100	r32′	zz		
(r32+)	=	-1- -0101	r32′	zz		

<7:0> = **データのビット範囲を示す。この例はビット0からビット7までの8ビット長のデータを意味する。**

r32: 32**ビット幅レジスタ**
r16: **符号付**16**ビット幅レジスタ**
r8: **符号付**8**ビット幅レジスタ**

zz = **インクリメント/デクリメント数の指定コード**

00: ±1
01: ±2
10: ±4
11: (**未定義**)

r32′ = **レジスタコードの上位**6**ビット**

cc

コンディションコード

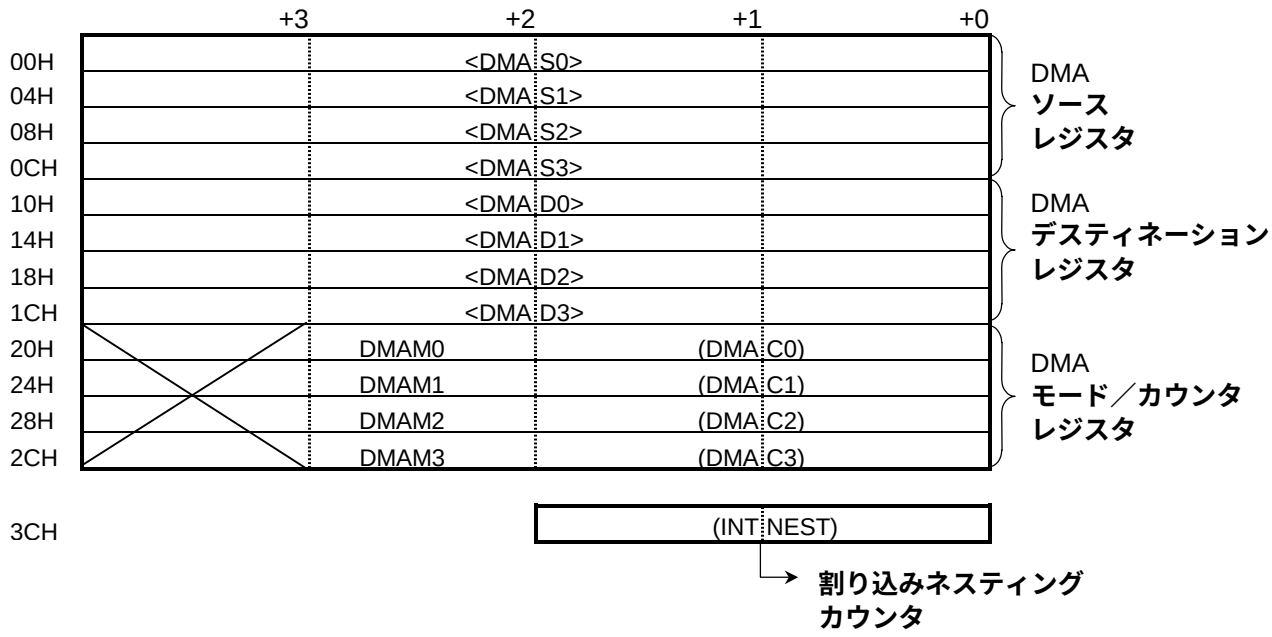
コード	記号	意味	条件式
0000	F	always False	–
1000	(なし)	always True	–
0110	Z	Zero	Z = 1
1110	NZ	Not Zero	Z = 0
0111	C	Carry	C = 1
1111	NC	Not Carry	C = 0
1101	PL or P	PLus	S = 0
0101	MI or M	MInus	S = 1
1110	NE	Not Equal	Z = 0
0110	EQ	EQual	Z = 1
0100	OV	OVerflow	P/V = 1
1100	NOV	No OVerflow	P/V = 0
0100	PE	Parity is Even	P/V = 1
1100	PO	Parity is Odd	P/V = 0
1001	GE	Greater than or Equal (signed)	(S xor P/V) = 0
0001	LT	Less than (signed)	(S xor P/V) = 1
1010	GT	Greater Than (signed)	[Z or (S xor P/V)] = 0
0010	LE	Less than or Equal (signed)	[Z or (S xor P/V)] = 1
1111	UGE	Unsigned Greater than or Equal	C = 0
0111	ULT	Unsigned Less Than	C = 1
1011	UGT	Unsigned Greater Than	(C or Z) = 0
0011	ULE	Unsigned Less than or Equal	(C or Z) = 1

■ レジスタマップ r (マキシマムモード)

	+3	+2	+1	+0	
00H	QW0 (QWA 0)	QA0 <XWA 0>	RW0 (RWA 0)	RA0	バンク0
04H	QB0 (QBC 0)	QC0 <XBC 0>	RB0 (RBC 0)	RC0	
08H	QD0 (QDE 0)	QE0 <XDE 0>	RD0 (RDE 0)	RE0	
0CH	QH0 (QHL 0)	QL0 <XHL 0>	RH0 (RHL 0)	RL0	
10H	QW1 (QWA 1)	QA1 <XWA 1>	RW1 (RWA 1)	RA1	バンク1
14H	QB1 (QBC 1)	QC1 <XBC 1>	RB1 (RBC 1)	RC1	
18H	QD1 (QDE 1)	QE1 <XDE 1>	RD1 (RDE 1)	RE1	
1CH	QH1 (QHL 1)	QL1 <XHL 1>	RH1 (RHL 1)	RL1	
20H	QW2 (QWA 2)	QA2 <XWA 2>	RW2 (RWA 2)	RA2	バンク2
24H	QB2 (QBC 2)	QC2 <XBC 2>	RB2 (RBC 2)	RC2	
28H	QD2 (QDE 2)	QE2 <XDE 2>	RD2 (RDE 2)	RE2	
2CH	QH2 (QHL 2)	QL2 <XHL 2>	RH2 (RHL 2)	RL2	
30H	QW3 (QWA 3)	QA3 <XWA 3>	RW3 (RWA 3)	RA3	バンク3
34H	QB3 (QBC 3)	QC3 <XBC 3>	RB3 (RBC 3)	RC3	
38H	QD3 (QDE 3)	QE3 <XDE 3>	RD3 (RDE 3)	RE3	
3CH	QH3 (QHL 3)	QL3 <XHL 3>	RH3 (RHL 3)	RL3	
D0H	QW′ (QWA′)	QA′ <XWA′>	W′ (WA′)	A′	プレビアスバンク
D4H	QB′ (QBC′)	QC′ <XWA′>	B′ (BC′)	C′	
D8H	QD′ (QDE′)	QE′ <XDE′>	D′ (DE′)	E′	
DCH	QH′ (QHL′)	QL′ <XHL′>	H′ (HL′)	L′	
E0H	QW (QWA)	QA <XWA>	W (WA)	A	カレントバンク
E4H	QB (QBC)	QC <XBC>	B (BC)	C	
E8H	QD (QDE)	QE <XDE>	D (DE)	E	
ECH	QH (QHL)	QL <XHL>	H (HL)	L	
F0H	QIXH (QIX)	QIXL <XIX>	IXH (IX)	IXL	
F4H	QIYH (QIY)	QIYL <XIY>	IYH (IY)	IYL	
F8H	QIZH (QIZ)	QIZL <XIZ>	IZH (IZ)	IZL	
FCH	QSPH (QSP)	QSPL <XSP>	SPH (SP)	SPL	

( ) : **ワードレジスタ**(16 **ビット**)**名**
< > : **ロングワードレジスタ**(32 **ビット**)**名**

■ **コントロールレジスタマップ** cr



( ) : **ワードレジスタ**(16 **ビット**)**名**
< > : **ロングワードレジスタ**(32 **ビット**)**名**

ADC dst, src

< Add, with Carry **キャリー付き加算** >

動　作：dst ← dst + src + CY

説　明：dstの内容とsrcの内容とキャリーフラグCYの内容が加算され、dstへ転送されます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	○	ADC	R, r	1 1 z z 1 r 1 0 0 1 0 R
○	○	○	ADC	r, #	1 1 z z 1 r 1 1 0 0 1 0 0 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	ADC	R, (mem)	1 m z z m m m m 1 0 0 1 0 R
○	○	○	ADC	(mem), R	1 m z z m m m m 1 0 0 1 1 R
○	○	×	ADC<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 0 0 1 #<7:0> #<15:8>

フラグ：

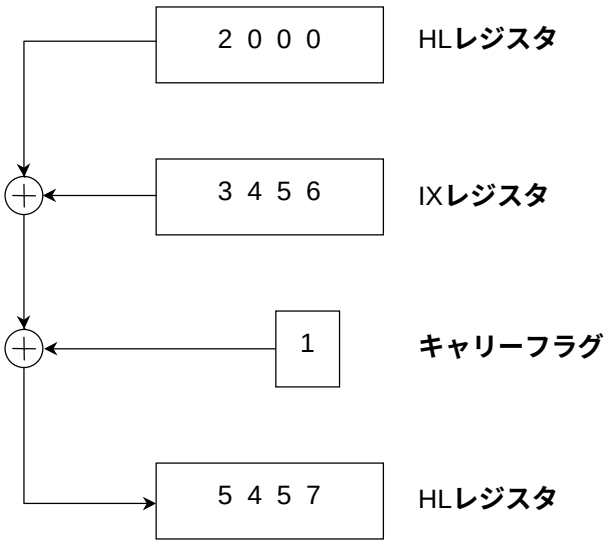
S	Z	H	V	N	C
*	*	*	*	0	*

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 演算結果、ビット3からビット4へキャリーが発生したときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は、不定値がセットされます。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 0にクリアされます。
C = 演算結果、最上位ビットからキャリーが発生したときは1、それ以外のときは0がセットされます。

実行例：HLレジスタが2000H,IXレジスタが3456H,キャリーフラグCYが1のとき、

```
ADC HL,IX
```

を実行すると、HLレジスタは5457Hになります。



ADD dst, src

< Add **加算** >

動　作：dst ← dst + src

説　明：dstの内容とsrcの内容が加算され、dstへ転送されます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	○	ADD	R, r	1 1 z z 1 r 1 0 0 0 0 R
○	○	○	ADD	r, #	1 1 z z 1 r 1 1 0 0 1 0 0 0 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	ADD	R, (mem)	1 m z z m m m m 1 0 0 0 0 R
○	○	○	ADD	(mem), R	1 m z z m m m m 1 0 0 0 1 R
○	○	×	ADD<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 0 0 0 #<7:0> #<15:8>

フラグ：

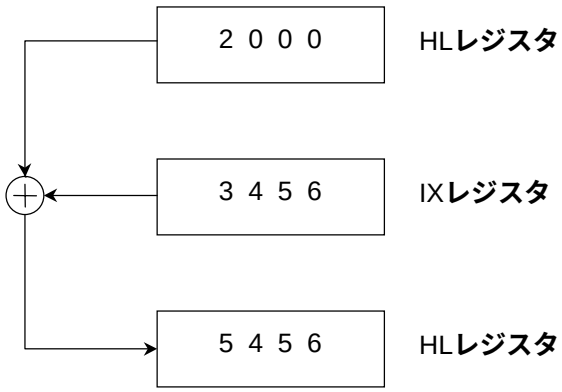
S	Z	H	V	N	C
*	*	*	*	0	*

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 演算結果、ビット3からビット4へキャリーが発生したときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は、不定値がセットされます。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 0にクリアされます。
C = 演算結果、最上位ビットからキャリーが発生したときは1、それ以外のときは0がセットされます。

実行例：HLレジスタが2000H,IXレジスタが3456Hのとき、

```
ADD HL,IX
```

を実行すると、HLレジスタは5456Hになります。



AND dst, src

< And **論理積** >

動　作：dst ← dst AND src

説　明：dstの内容とsrcの内容が論理積演算され、dstへ転送されます。

(真理値表)

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

詳　細：

サイズ			ニモニック		コード
バイト	ワード	ロング			
○	○	○	AND	R, r	1 1 z z 1 r 1 1 0 0 0 R
○	○	○	AND	r, #	1 1 z z 1 r 1 1 0 0 1 1 0 0 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	AND	R, (mem)	1 m z z m m m m 1 1 0 0 0 R
○	○	○	AND	(mem), R	1 m z z m m m m 1 1 0 0 1 R
○	○	×	AND <W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 1 0 0 #<7:0> #<15:8>

フラグ：

S	Z	H	V	N	C
*	*	1	*	0	0

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 1にセットされます。
V = 演算結果のパリティ(1の数)が偶数のときは1、奇数のときは0がセットされます。ただし、オペランド長が32ビットの場合は、不定値がセットされます。
N = 0にクリアされます。
C = 0にクリアされます。

実行例：HLレジスタが7350H,IXレジスタが3456Hのとき、

AND HL,IX

を実行すると、HLレジスタは3050Hになります。

	0111	0011	0101	0000	← HLレジスタ (実行前)
AND)	0011	0100	0101	0110	← IXレジスタ (実行前)
	0011	0000	0101	0000	← HLレジスタ (実行後)

ANDCF num, src

< And Carry Flag **キャリーフラグとの1ビット論理積** >

動　作：CY ← CY AND src<num>

説　明：キャリーフラグCYの内容とsrcのビットnumの内容が論理積演算され、キャリーフラグCYへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	ANDCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 0 0 0 0 0 0 0 #4
○	○	×	ANDCF	A, r	1 1 0 z 1 r 0 0 1 0 1 0 0 0
○	×	×	ANDCF	#3, (mem)	1 m 1 1 m m m m 1 0 0 0 0 #3
○	×	×	ANDCF	A, (mem),	1 m 1 1 m m m m 0 0 1 0 1 0 0 0

補　足；ビットnumがAレジスタで指定された場合、Aレジスタの下位4ビットの値がビットnumとして使われます。オペランドがバイトのとき、ビットnumの下位4ビットの値が8～15の場合、演算結果は不定になります。

フラグ：

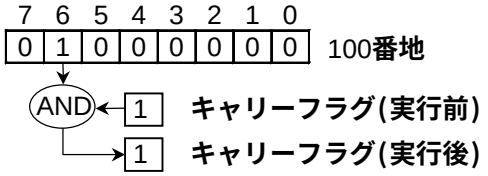
S	Z	H	V	N	C
–	–	–	–	–	*

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = キャリーフラグCYの内容とsrcのビットnumの内容の論理積演算された値がセットされます。

実行例：100番地のメモリの内容が01000000B(2進数)で、キャリーフラグCYが1のとき、

ANDCF 6,(100H)

を実行すると、キャリーフラグは1になります。



BIT num, src

< Bit test 1ビットのテスト >

動　作：Zフラグ ← src<num>の反転値

説　明：srcのビットnumの反転値が、Zフラグへ転送されます。

詳　細：:

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	BIT	#4, r	1 1 0 z 1 r 0 0 1 1 0 0 1 1 0 0 0 0 # 4
○	×	×	BIT	#3, (mem)	1 m 1 1 m m m m 1 1 0 0 1 #3

フラグ：

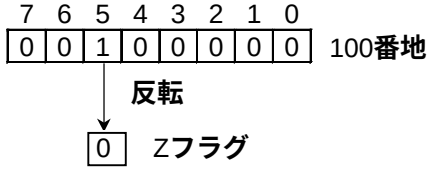
S	Z	H	V	N	C
×	*	1	×	0	–

S = 不定値がセットされます。
Z = src<num>の反転値がセットされます。
H = 1にセットされます。
V = 不定値がセットされます。
N = 0にリセットされます。
C = 変化なし。

実行例：100番地のメモリの内容が00100000B(2進数)のとき、

BIT 5,(100H)

を実行すると、Zフラグは0になります。



BS1B dst, src

< Bit Search 1 Backward **ビットパターン中の最初の1を後方向からサーチ** >

動　作：dst ← srcを後方向サーチした結果

説　明：srcのビットパターン中の最初の1が後方向から(MSBからLSBへ)サーチされ、その1の位置のビットナンバーが、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	×	BS1B	A, r	1 1 0 1 1 | r | 0 0 0 0 1 1 1 1

補　足：オペランド中のdstはAレジスタ、srcはワード長のレジスタに限られます。なお、サーチしたビットパターン中に1がなかった場合、Aレジスタの値は不定になり、Vフラグが1にセットされます。

フラグ：

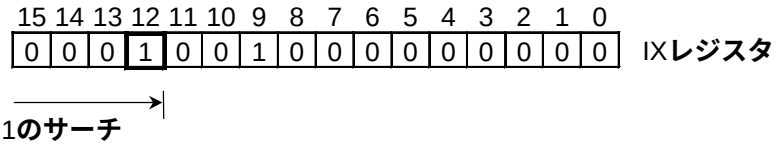
S	Z	H	V	N	C
–	–	–	*	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = srcの内容がオールゼロの場合(ビットパターン中に1がない場合)は1、それ以外のときは0がセットされます。
N = 変化なし。
C = 変化なし。

実行例：IXレジスタが1200Hのとき

```
BS1B A,IX
```

を実行すると、Aレジスタは0CHになります。



BS1F dst, src

< Bit Search 1 Forward **ビットパターン中の最初の1を前方向からサーチ** >

動　作：dst ← srcを前方向サーチした結果

説　明：srcのビットパターン中の最初の1が前方向から(LSBからMSBへ)サーチされ、その1の位置のビットナンバーが、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	×	BS1F	A, r	1 1 0 1 1 | r | 0 0 0 0 1 1 1 0

補　足：オペランド中のdstはAレジスタ、srcはワード長のレジスタに限られます。なお、サーチしたビットパターン中に1がなかった場合、Aレジスタの値は不定になり、Vフラグが1にセットされます。

フラグ：

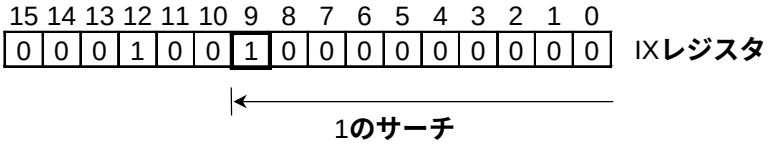
S	Z	H	V	N	C
–	–	–	*	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = srcの内容がオールゼロの場合(ビットパターン中に1がない場合)は1、それ以外のときは0がセットされます。
N = 変化なし。
C = 変化なし。

実行例：IXレジスタが1200Hのとき、

```
BS1F A,IX
```

を実行すると、Aレジスタは09Hになります。



CALL condition, dst

< Call **サブルーチンコール** >

動　作：if ccが真 then XSP ← XSP – 4, (XSP) ← 32ビット PC,PC ← dst.

説　明：オペランドのconditionが真の場合、スタック領域へプログラムカウンタPCの内容が退避され、dstで示されたプログラム番地へジャンプします。

詳　細：

ニモニック		コード
CALL	#16	0 0 0 1 1 1 0 0 #<7:0> #<15:8>
CALL	#24	0 0 0 1 1 1 0 1 #<7:0> #<15:8> #<23:16>
CALL	[cc,] mem	1 m 1 1 m m m m 1 1 1 0 c c c c

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：スタックポインタXSPが100Hのとき、8000H番地のメモリにある命令、

CALL 9000H

を実行すると、メモリの0FCH番地にリターンアドレス8003H(ロングデータ)がライトされ、スタックポインタXSPは0FCHになり、9000H番地へジャンプします。

CALR dst

< Call Relative **相対サブルーチンコール** >

動　作：XSP ← XSP − 4, (XSP) ← 32ビット PC, PC ← dst.

説　明：スタック領域へプログラムカウンタPCの内容が退避され、dstで示されたプログラム番地へ相対ジャンプします。

詳　細：

ニモニック		コード
CALR	$+3+d16	0 0 0 1 1 1 1 0
		d<7:0>
		d<15:8>

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

CCF

< Complement Carry Flag **キャリーフラグの反転** >

動　作：CY ← CYの反転値

説　明：キャリーフラグCYの内容が、反転されます。

詳　細：

	ニモニック	コード
	CCF	0 | 0 | 0 | 1 | 0 | 0 | 1 | 0

フラグ：

S	Z	H	V	N	C
–	–	X	–	0	*

S = 変化なし。
Z = 変化なし。
H = 不定値がセットされます。
V = 変化なし。
N = 0にリセットされます。
C = 自分自身の反転値がセットされます。

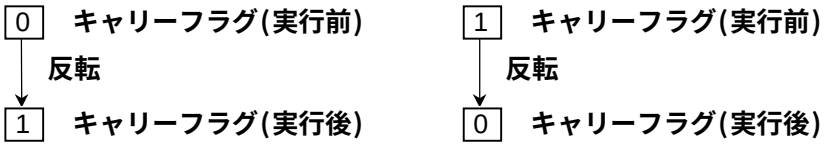
実行例：キャリーフラグCYが0のとき

CCF

を実行すると、キャリーフラグCYは1になります。さらに、

CCF

を実行すると、キャリーフラグCYは0になります。



CHG num, dst

< Change 1**ビットの反転** >

動　作：dst<num> ← dst<num>の反転値

説　明：dstのビットnumの値が反転されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	CHG	#4, r	1 1 0 z 1 r 0 0 1 1 0 0 1 0 0 0 0 0 # 4
○	×	×	CHG	#3, (mem)	1 m 1 1 m m m m 1 1 0 0 0 #3

フラグ：

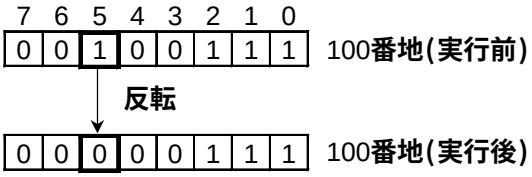
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：100番地のメモリの内容が00100111B(2進数)のとき、

CHG 5,(100H)

を実行すると、100番地のメモリの内容は00000111B(2進数)になります。



CP src1, src2

< Compare **比較** >

動　作：src1 – src2

説　明：src1の内容とsrc2の内容が比較され、その結果がフラグレジスタFに反映されます。

詳　細：:

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	○	CP	R, r	1 1 z z 1 r 1 1 1 1 1 0 R
○	○	×	CP	r, #3	1 1 0 z 1 r 1 1 0 1 1 #3
○	○	○	CP	r, #	1 1 z z 1 r 1 1 0 0 1 1 1 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	CP	R, (mem)	1 m z z m m m m 1 1 1 1 1 0 R
○	○	○	CP	(mem), R	1 m z z m m m m 1 1 1 1 1 R
○	○	×	CP<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 1 1 1 #<7:0> #<15:8>

補　足：オペランド中の#3は、0～7を示します。

フラグ：

S	Z	H	V	N	C
*	*	*	*	1	*

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は、不定値がセットされます。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 1にセットされます。
C = 演算結果、最上位ビットからボローが発生したときは1、それ以外のときは0がセットされます。

実行例：HLレジスタが1234H,IXレジスタが1234Hのとき、

CP HL,IX

を実行すると、ZフラグとNフラグは1にセットされ、SフラグとHフラグ、Vフラグ、Cフラグは0にクリアされます。

CPD src1, src2

< Compare Decrement **逆方向のブロック部分サーチ** >

動　作：src1 – src2, BC ← BC – 1

説　明：src1とsrc2の内容が比較されます。その後、BCレジスタの内容が–1されます。なお、src1はAレジスタまたはWAレジスタに限られ、src2のアドレッシングモードはポストデクリメントのレジスタ間接に限られます。

詳　細：:

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	×	CPD	[A/WA, (R–)]	1 0 0 z 0 R 0 0 0 1 0 1 1 0

補　足：[ ]の内側のオペランドの記述を省略すると、A,(XHL–)を指定したことになります。

フラグ：

S	Z	H	V	N	C
*	*	*	*	1	–

S = src1–src2の演算結果の最上位ビットの値がセットされます。
Z = src1–src2の演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = src1–src2の演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。
V = 命令実行後、BCレジスタの値が0のときは0、それ以外のときは1がセットされます。
N = 1にセットされます。
C = 変化なし。

実行例：XIXレジスタが00123456H,BCレジスタが0200Hのとき、

CPD A,(XIX–)

を実行すると、Aレジスタの内容と123456H番地の内容が比較され、XIXレジスタは00123455H,BCレジスタは01FFHになります。

CPDR src1, src2

< Compare Decrement Repeat **逆方向のブロックサーチ** >

動　作：src1 – src2, BC ← BC – 1, Repeat until src1 = src2 or BC = 0

説　明：src1とsrc2の内容が比較されます。その後、BCレジスタの内容が–1され、src1 = src2またはBC = 0でなければ、再び前記動作を繰り返します。なお、src1はAレジスタまたはWAレジスタに限られ、src2のアドレッシングモードはポストデクリメントのレジスタ間接に限られます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	CPDR	[A/WA, (R–)]	1 0 0 z 0 R 0 0 0 1 0 1 1 1

補　足：[ ]の内側のオペランドの記述を省略すると、A,(XHL–)を指定したことになります。

フラグ：

S	Z	H	V	N	C
*	*	*	*	1	–

S = src1–src2の演算結果の最上位ビットの値がセットされます。
Z = src1–src2の演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = src1–src2の演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。
V = 命令実行後、BCレジスタの値が0のときは0、それ以外のときは1がセットされます。
N = 1にセットされます。
C = 変化なし。

実行例：Aレジスタが55H,XIXレジスタが00123456H,BCレジスタが0003Hで、
メモリの123456H番地の内容が11H、
メモリの123455H番地の内容が22H、
メモリの123454H番地の内容が33Hのとき、
CPDR A, (XIX－)
を実行すると、メモリの123456H番地, 123455番地, 123454H番地をリードした後、この命令の実行がBC = 0の条件で終了し、XIXレジスタは00123453H, BCレジスタは0000Hになります。

CPI src1, src2

< Compare Increment **正方向のブロック部分サーチ** >

動　作：src1 – src2, BC ← BC – 1

説　明：src1とsrc2の内容が比較されます。その後、BCレジスタの内容が–1されます。なお、src1はAレジスタまたはWAレジスタに限られ、src2のアドレッシングモードはポストインクリメントレジスタ間接に限られます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	CPI	[A/WA, (R+)]	1 0 0 z 0 R 0 0 0 1 0 1 0 0

補　足：[ ]の内側のオペランドの記述を省略すると、A,(XHL+)を指定したことになります。

フラグ：

S	Z	H	V	N	C
*	*	*	*	1	–

S = src1–src2の演算結果の最上位ビットの値がセットされます。
Z = src1–src2の演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = src1–src2の演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。
V = 命令実行後、BCレジスタの値が0のときは0、それ以外のときは1がセットされます。
N = 1にセットされます。
C = 変化なし。

実行例：XIXレジスタが00123456H,BCレジスタが0200Hのとき、

```
CPI A, (XIX+)
```

を実行すると、Aレジスタの内容と123456H番地の内容が比較され、XIXレジスタは00123457H,BCレジスタは01FFHになります。

CPIR src1, src2

< Compare Increment Repeat **正方向のブロックサーチ** >

動　作：src1 – src2, BC ← BC – 1, Repeat until src1 = src2 or BC = 0

説　明：src1とsrc2の内容が比較されます。その後、BCレジスタの内容が–1され、src1 = src2またはBC = 0でなければ、再び前記動作を繰り返します。なお、src1はAレジスタまたはWAレジスタに限られ、src2のアドレッシングモードはポストインクリメントのレジスタ間接に限られます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	CPIR	[A/WA, (R+)]	1 0 0 z 0 R 0 0 0 1 0 1 0 1

補　足：[ ]の内側のオペランドの記述を省略すると、A,(XHL+)を指定したことになります。

フラグ：

S	Z	H	V	N	C
*	*	*	*	1	–

S = src1–src2の演算結果の最上位ビットの値がセットされます。
Z = src1–src2の演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = src1–src2の演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。
V = 命令実行後、BCレジストの値が0のときは0、それ以外のときは1がセットされます。
N = 1にセットされます。
C = 変化なし。

実行例：Aレジスタが33H,XIXレジスタが00123456H,BCレジスタが0200Hで、
メモリの123456H番地の内容が11H,
メモリの123457H番地の内容が22H,
メモリの123458H番地の内容が33Hのとき、
CPIR A,(XIX+)
を実行すると、メモリの123456H番地、123457番地、123458H番地をリードした後、この命令の実行がsrc1 = src2の条件で終了し、XIXレジスタは00123459H、BCレジスタは01FDHになります。

CPL dst

< Complement 1**の補数** >

動　作：dst ← dstの1の補数

説　明：dstの1の補数(各ビットの0/1反転)値が、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	CPL	r	1 1 0 z 1 r 0 0 0 0 0 1 1 0

フラグ：

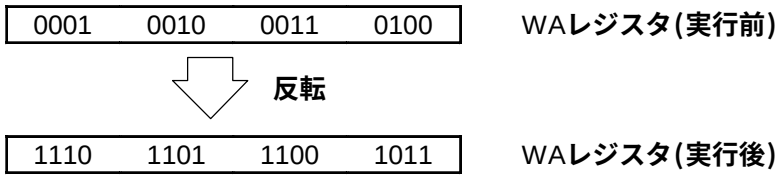
S	Z	H	V	N	C
–	–	1	–	1	–

S = 変化なし。
Z = 変化なし。
H = 1にセットされます。
V = 変化なし。
N = 1にセットされます。
C = 変化なし。

実行例：WAレジスタが1234Hのとき、

CPL WA

を実行すると、WAレジスタはEDCBHになります。



DAA dst

< Decimal Adjust Accumulator 10**進補正** >

動　作：dst ← dstの10進補正

説　明：dstの内容がCフラグとHフラグ、Nフラグの状態により、10進補正されます。この命令は、加算命令または減算命令の後に、その演算結果を、2進化10進(BCD)表現に補正するためのものです。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	×	×	DAA	r	1 1 0 0 1 r 0 0 0 1 0 0 0 0

演算	DAA **命令を実行する前の** N **フラグ**	DAA **命令を実行する前の** C **フラグ**	dst **の上位** 4 **ビット**	DAA **命令を実行する前の** H **フラグ**	dst **の下位** 4 **ビット**	**加算される数**	DAA **命令を実行した後の** C **フラグ**
	0	0	0～9	0	0～9	00	0
	0	0	0～8	0	A～F	06	0
	0	0	0～9	1	0～3	06	0
ADD	0	0	A～F	0	0～9	60	1
	0	0	9～F	0	A～F	66	1
ADC	0	0	A～F	1	0～3	66	1
	0	1	0～2	0	0～9	60	1
	0	1	0～2	0	A～F	66	1
	0	1	0～3	1	0～3	66	1
SUB	1	0	0～9	0	0～9	00	0
SBC	1	0	0～8	1	6～F	FA	0
NEG	1	1	7～F	0	0～9	A0	1
	1	1	6～F	1	6～F	9A	1

補　足：INCまたはDEC命令の演算に対しては、10進補正できません。これらの命令では、Cフラグが変化しないからです。

フラグ：

S	Z	H	V	N	C
*	*	*	*	–	*

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 演算結果、ビット3からビット4へキャリーが発生したときは1、それ以外のときは0がセットされます。
V = 演算結果のパリティ(1の数)が偶数のときは1、奇数のときは0がセットされます。
N = 変化なし。
C = 演算結果、最上位ビットからキャリーが発生した場合、または演算前にキャリーが1であった場合は1、それ以外のときは0にセットされます。

実行例：Aレジスタが59Hで、Bレジスタが13Hのとき、

```
ADD A,B
DAA A
```

を実行すると、Aレジスタは72Hになります。

DEC num, dst

< Decrement **減少** >

動　作：dst ← dst – num

説　明：dstの内容からnumの内容が減算され、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	DEC	#3, r	1 1 z z 1 r 0 1 1 0 1 #3
○	○	×	DEC<W>	#3, (mem)	1 m 0 z m m m m 0 1 1 0 1 #3

補　足：オペランド中の#3は1～8を示し、オブジェクトコードは1～7, 0に対応します。

フラグ：

S	Z	H	V	N	C
*	*	*	*	1	–

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 1にセットされます。
C = 変化なし。
(注) DEC #3,rでオペランドサイズがワードまたはロングの場合は、すべてのフラグは変化しません。

実行例：HLレジスタが5678Hのとき、

```
DEC 4, HL
```

を実行すると、HLレジスタは5674Hになります。

DECF

< Decrement Register File Pointer **レジスタバンクの–1切り替え** >

動　作：RFP<2:0> ← RFP<2:0> – 1

説　明：ステータスレジスタSR内のレジスタファイルポインタRFP<2:0>の内容が–1されます。なお、RFP2は0に固定されます。

詳　細：

ニモニック	コード
DECF	0 0 0 0 1 1 0 1

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：RFP<2:0>の内容が2のとき、

```
DECF
```

を実行すると、RFP<2:0>の内容は1になります。

DI

< Disable Interrupt **割り込み禁止** >

動　作：IFF<2:0> ← 7

説　明：ステータスレジスタSR中の割り込み許可フラグIFF<2:0>の内容が7になります。この命令の実行後、割り込みは、ノンマスカブル割り込み(割り込み要求レベルが7のもの)しか受け付けなくなります。

詳　細：

ニモニック	コード
DI	0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

DIV dst, src

< Divide **符号なし除算** >

動　作：dst<下位半分> ← dst ÷ src,dst<上位半分> ← 余り(符号なし)

説　明：dstの内容がsrcの内容で符号なし除算され、商がdstの下位半分へ、余りがdstの上位半分へ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	DIV	RR, r	1 1 0 z 1 r 0 1 0 1 0 R
○	○	×	DIV	rr, #	1 1 0 z 1 r 0 0 0 0 1 0 1 0 #<7:0> #<15:8>
○	○	×	DIV	RR, (mem)	1 m 0 z m m m m 0 1 0 1 0 R

(注)　RRついては次ページ参照

補　足：演算サイズがバイトのときは、dst(下位バイト)←dst(ワード)÷src(バイト)、dst(上位バイト)←余り
演算サイズがワードのときは、dst(下位ワード)←dst(ロング)÷src(ワード)、dst(上位ワード)←余り
になります。オペランドのdstの記述は、被除数のサイズに合わせてください。
オーバフローしたとき(Vフラグが1にセットされたとき)は、dstへは不定値がセットされています。

フラグ：

S	Z	H	V	N	C
–	–	–	V	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 0で割った場合、または、商が格納用dstのビット長で表現できる数を超えた場合は1、それ以外のときは0がセットされます。
N = 変化なし。
C = 変化なし。

実行例：XIXレジスタが12345678Hで、IYレジスタが89ABHのとき、

DIV XIX,IY

を実行すると、商は21DAH、余りは0FDAHとなり、XIXレジスタは0FDA21DAHになります。

補　足：DIV RR,rとDIV RR,(mem)命令のRRは、下記に示す表のようになります。

演算サイズがバイト
(8 ビット←16 ビット÷8 ビット)

RR	コード R
WA	001
BC	011
DE	101
HL	111
IX	指定不可!
IY	
IZ	
SP	

演算サイズがワード
(16 ビット←32 ビット÷16 ビット)

RR	コード R
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

DIV rr,#命令のrrは、下記に示す表のようになります。

演算サイズがバイト
(8 ビット←16 ビット÷8 ビット)

rr	コード r
WA	001
BC	011
DE	101
HL	111
IX	C7H : F0H
IY	C7H : F4H
IZ	C7H : F8H
SP	C7H : FCH
	1 バイト目　2 バイト目

(注) その他のワードレジスタも IX～SP 同様の拡張コード方式で すべて指定可能。

演算サイズがワード
(16 ビット←32 ビット÷16 ビット)

rr	コード r
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

(注) その他のロングワードレジスタも 拡張コード方式ですべて指定可能

DIVS dst, src

< Divide Signed **符号付き除算** >

動　作：dst<下位半分> ← dst ÷ src,dst<上位半分> ← 余り(符号付き)

説　明：dstの内容がsrcの内容で符号付き除算され、商がdstの下位半分へ、余りがdstの上位半分へ転送されます。

詳　細：:

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	×	DIVS	RR, r	1 1 0 z 1 r 0 1 0 1 1 R
○	○	×	DIVS	rr, #	1 1 0 z 1 r 0 0 0 0 1 0 1 1 #<7:0> #<15:8>
○	○	×	DIVS	RR, (mem)	1 m 0 z m m m m 0 1 0 1 1 R

(注)　RRについては次ページ参照

補　足：演算サイズがバイトのときは、dst(下位バイト)←dst(ワード)÷src(バイト)、dst(上位バイト)←余り
演算サイズがワードのときは、dst(下位ワード)←dst(ロング)÷src(ワード)、dst(上位ワード)←余り
になります。オペランドのdstの記述は、被除数のサイズに合わせてください。
なお、余りの符号は、被余数と同じになります。
オーバフローしたとき(Vフラグが1にセットされたとき)は、dstへは不定値がセットされています。

フラグ：

S	Z	H	V	N	C
–	–	–	*	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 0で割った場合、または、商が格納用dstのビット長で表現できる数を超えた場合は1、それ以外のときは0がセットされます。
N = 変化なし。
C = 変化なし。

実行例：XIXレジスタが12345678Hで、IYレジスタが89ABHのとき、

DIVS XIX,IY

を実行すると、余りは16EEH、商はD89EHとなり、XIXレジスタは16EED89EHになります。

補　足：DIVS RR,rとDIVS RR,(MEM)命令のRRは、下記に示す表のようになります。

演算サイズがバイト
(8 ビット←16 ビット÷8 ビット)

RR	コード R
WA	001
BC	011
DE	101
HL	111
IX	指定不可!
IY	
IZ	
SP	

演算サイズがワード
(16 ビット←32 ビット÷16 ビット)

RR	コード R
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

DIVS rr,#命令のrrは、下記に示す表のようになります。

演算サイズがバイト
(8 ビット←16 ビット÷8 ビット)

rr	コード r
WA	001
BC	011
DE	101
HL	111
IX	C7H : F0H
IY	C7H : F4H
IZ	C7H : F8H
SP	C7H : FCH
	1 バイト目　2 バイト目

(注) その他のワードレジスタも IX～SP 同様の拡張コード方式で、すべて指定可能。

演算サイズがワード
(16 ビット←32 ビット÷16 ビット)

rr	コード r
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

(注) その他のロングワードレジスタも拡張コード方式ですべて指定可能。

DJNZ dst1, dst2

< Decrement and Jump if Non Zero **減少&ジャンプノンゼロ** >

動　作：dst1 ← dst1 – 1.　if dst1 ≠ 0, then PC ← dst2.

説　明：dst1の内容が–1され、その値がゼロでなければ、dst2で示されたプログラム番地へ相対ジャンプします。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	DJNZ	[r,]$+3/4+d8	1 1 0 z 1 r 0 0 0 1 1 1 0 0 d<7:0>

(注) rが拡張コードで指定されるときは $+4+d8、それ以外のときは$+3+d8 となります。

補　足：[ ]の内側のオペランドのr,の記述を省略すると、Bレジスタが指定されたものと解釈されます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：Aレジスタが12Hで、Wレジスタが03Hのとき、

```
LOOP:  ADD A, A
       DJNZ W, LOOP
```

を実行すると、3回ループして、Aレジスタは24H → 48H → 90H,Wレジスタは02H → 01H → 00Hになります。

EI num

< Enable Interrupt **割り込み許可** >

動　作：IFF <2:0> ← num

説　明：ステータスレジスタSR中の割り込み許可フラグIFF<2:0>の内容が、numになります。この命令の実行後、CPUの割り込み受け付けレベルはnumになります。

詳　細：

ニモニック		コード
EI	[#3]	0 0 0 0 0 1 1 0 0 0 0 0 0 #3

補　足：オペランドの値は、0～7まで指定できます。オペランドの記述を省略すると、EI 0と認識されます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

EX dst, src

< Exchange **交換** >

動　作：dst ↔ src

説　明：dstの内容とsrcの内容が、交換されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	×	×	EX	F, F′	0 0 0 1 0 1 1 0
○	○	×	EX	R, r	1 1 z z 1 r 1 0 1 1 1 R
○	○	×	EX	(mem), r	1 m z z m m m m 0 0 1 1 0 R

フラグ：

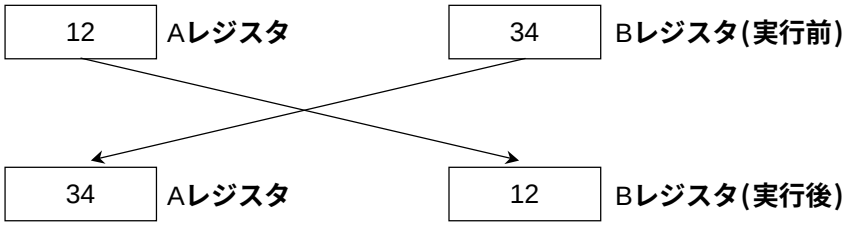
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。
(注) EX F,F'を実行すると、フラグはすべて変化します。

実行例：Aレジスタが12Hで、Bレジスタが34Hのとき、

EX A, B

を実行すると、Aレジスタは34H、Bレジスタは12Hになります。



EXTS dst

< Extend Sign **符号拡張** >

動　作：dst <上位半分> ← dst <下位半分>の符号ビット

説　明：dstの下位半分の符号ビット(オペランドサイズがワードのときはビット7、ロングワードのときはビット15)が、dstの上位半分の全ビットに転送(コピー)されます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
×	○	○	EXTS	r	1 1 z z 1 r / 0 0 0 1 0 0 1 1

フラグ：

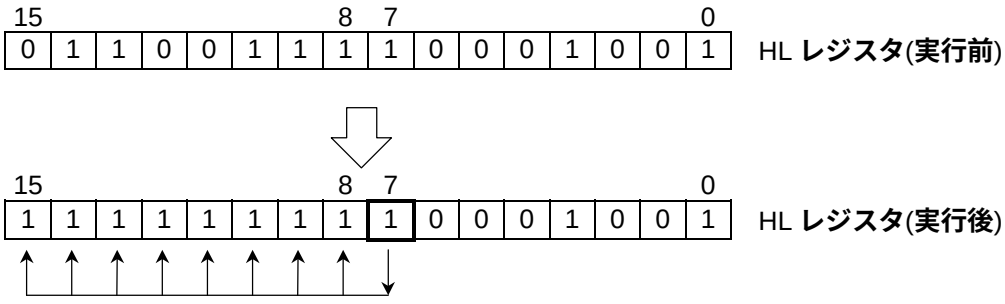
S	Z	H	V	N	C
–	–	–	–	–	–

Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：HLレジスタが6789Hのとき、

EXTS HL

を実行すると、HLレジスタはFF89Hになります。



EXTZ dst

< Extend Zero **ゼロ拡張** >

動　作：dst<上位半分> ← 0

説　明：dst上位半分がゼロにクリアされます。通常オペランドサイズが異なる演算を行う前、オペランドサイズを揃えるために使用します。

詳　細：

サイズ			ニモニック		コード
バイト	ワード	ロング			
×	○	○	EXTZ	r	1 1 z z 1 r 0 0 0 1 0 0 1 0

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：HLレジスタが6789Hのとき、

```
EXTZ HL
```

を実行すると、HLレジスタは0089Hになります。

XIXレジスタが12345678Hのとき、

```
EXTZ XIX
```

を実行すると、XIXレジスタは00005678Hになります。

HALT

< Halt CPUの停止 >

動　作：CPUの停止

説　明：命令の実行が停止されます。再開は、割り込みを受け付けることによって行われます。

詳　細：

ニモニック	コード
HALT	0 0 0 0 0 1 0 1

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

INC num, dst

< Increment **増加** >

動　作：dst ← dst + num

説　明：dstの内容とnumの内容が加算され、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	INC	#3, r	1 1 z z 1 r 0 1 1 0 0 #3
○	○	×	INC<W>	#3, (mem)	1 m 0 z m m m m 0 1 1 0 0 #3

補　足：オペランド中の#3は1～8を示し、オブジェクトコードは1～7, 0に対応します。

フラグ：

S	Z	H	V	N	C
*	*	*	*	0	–

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1それ以外のときは0がセットされます。
H = 演算の結果、ビット3からビット4へキャリーが発生したときは1、それ以外のときは0がセットされます。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 0にクリアされます。
C = 変化なし。
(注) INC #3,rでオペランドサイズがワードまたはロングの場合は、すべてのフラグは変化しません。

実行例：WAレジスタが1234Hのとき、

```
INC 5,WA
```

を実行すると、WAレジスタは1239Hになります。

INCF

< Increment Register File Pointer **レジスタバンクの+1切り替え** >

動　作：RFP<2:0> ← RFP<2:0> + 1

説　明：ステータスレジスタSP内のレジスタファイルポインタRFP<2:0>の内容が+1されます。なお、RFP2は0に固定されます。

詳　細：

ニモニック	コード
INCF	0 0 0 0 1 1 0 0

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：RFP<2:0>の内容が2のとき、

```
INCF
```

を実行すると、RFP<2:0>の内容は3になります。

JP condition, dst

< Jump **ジャンプ** >

動　作：If ccが真　then PC ← dst.

説　明：オペランドのconditionが真の場合、dstで示されたプログラム番地へジャンプします。

詳　細：

ニモニック		コード
JP	#16	0 0 0 1 1 0 1 0 #<7:0> #<15:8>
JP	#24	0 0 0 1 1 0 1 1 #<7:0> #<15:8> #<23:16>
JP	[cc,] mem	1 m 1 1 m m m m 1 1 0 1 c c c

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：下記の命令

```
JP 2000H
```

を実行すると、無条件で2000H番地へジャンプします。
XIXレジスタが00123456Hで、

```
JP C,XIX+2
```

を実行すると、キャリーフラグの値が1の場合、123458H番地へジャンプします。

JR condition, dst

< Jump Relative **相対ジャンプ** >

動　作：If ccが真　then PC ← dst.

説　明：オペランドのconditionが真の場合、dstで示されたプログラム番地へ相対ジャンプします。

詳　細：

	ニモニック	コード
JR	[cc,]$+2+d8	0 1 1 0 c c / d<7:0>
JRL	[cc,]$+3+d16	0 1 1 1 c c / d<7:0> / d<15:8>

フラグ：

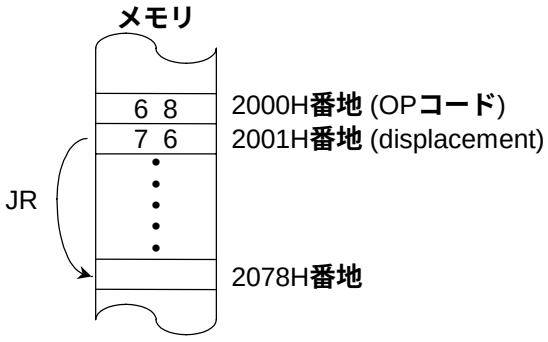
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：2000H番地のメモリにある下記の命令、

JR 2078H

を実行すると、無条件で2078H番地へ相対ジャンプします。
なお、上記命令のオブジェクトコードは、68H:76Hになります。



LD dst, src

< Load **転送** >

動　作：dst ← src

説　明：srcの内容が、dstへ転送されます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	○	LD	R, r	1 1 z z 1 r 1 0 0 0 1 R
○	○	○	LD	r, R	1 1 z z 1 r 1 0 0 1 1 R
○	○	○	LD	r, #3	1 1 z z 1 r 1 0 1 0 1 #3
○	○	○	LD	R, #	0 z z z 0 R #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	LD	r, #	1 1 z z 1 r 0 0 0 0 0 0 1 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	LD	R, (mem)	1 m z z m m m m 0 0 1 0 0 R
○	○	○	LD	(mem), R	1 m 1 1 m m m m 0 1 z z 0 R
○	○	×	LD<W>	(#8), #	0 0 0 0 1 0 z 0 #8 #<7:0> #<15:8>

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	×	LD<W>	(mem), #	1 m 1 1 m m m m 0 0 0 0 0 0 z 0 #<7:0> #<15:8>
○	○	×	LD<W>	(#16), (mem)	1 m 0 z m m m m 0 0 0 1 1 0 0 1 #16<7:0> #16<15:8>
○	○	×	LD<W>	(mem), (#16)	1 m 1 1 m m m m 0 0 0 1 0 1 z 0 #16<7:0> #16<15:8>

フラグ：

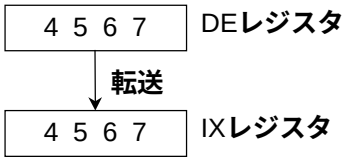
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：DEレジスタが4567Hのとき、

LD IX, DE

を実行すると、IXレジスタは4567Hになります。



LDA dst, src

< Load Address **実効アドレスの転送** >

動　作：dst ← srcの実効アドレス

説　明：srcの実効アドレス値が、dstへ転送されます。

詳　細：

サイズ			ニモニック		コード
バイト	ワード	ロング			
×	○	○	LDA	R, mem	1 m 1 1 m m m m 0 0 1 s 0 R

補　足：本命令は、ADD命令と動作が似ていますが、dstの指定がsrcの指定とは別にできる点が特長です。主に、Cコンパイラなどでポインタを扱う場合に使用します。

フラグ：

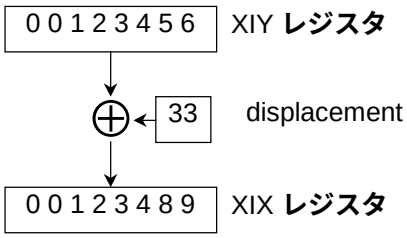
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：XIYレジスタが00123456Hで

LDA XIX, XIY+33H

を実行すると、XIXレジスタは00123489Hになります。



LDAR dst, src

< Load Address Relative **相対アドレスの転送** >

動　作：dst ← srcの相対アドレス値

説　明：srcで指定された相対アドレス値が、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	○	LDAR	R, $+4+d16	

1	1	1	1	0	0	1	1
0	0	0	1	0	0	1	1
d<7:0>							
d<15:8>							
0	0	1	s	0		R	

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

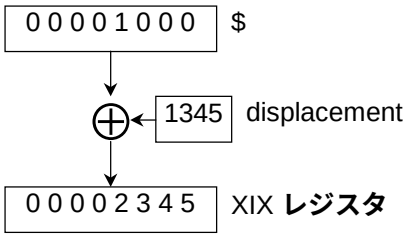
S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：1000H番地のメモリにある下記の命令、

LDAR XIX, $+1345H

を実行すると、XIXレジスタは00002345Hになります。$は、その命令が置かれている先頭番地を示します。

なお、上記命令のオブジェクトコードは、F3H: 13H: 41H: 13H: 34Hになります。



LDC dst, src

< Load Control **コントロールレジスタの転送** >

動　作：dst ← src

説　明：srcの内容が、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	LDC	cr, r	1 1 z z 1 r / 0 0 1 0 1 1 1 0 / cr
○	○	○	LDC	r, cr	1 1 z z 1 r / 0 0 1 0 1 1 1 1 / cr

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：WAレジスタが1234Hのとき、

LDC DMAC0, WA

を実行すると、コントロールレジスタDMAC0は1234Hになります。

LDCF num, src

< Load Carry Flag **キャリーフラグへの1ビット転送** >

動　作：CY ← src<num>

説　明：srcのビットnumの内容が、キャリーフラグCYへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	LDCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 0 1 1 0 0 1 0 #4
○	○	×	LDCF	A, r	1 1 0 z 1 r 0 0 1 0 1 0 1 1
○	×	×	LDCF	#3, (mem)	1 m 1 1 m m m m 1 0 0 1 1 #3
○	×	×	LDCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 0 1 1

補　足：ビットnumがAレジスタで指定された場合、Aレジスタの下位4ビットの値がビットnumとして使われます。オペランドがバイトのとき、ビットnumの下位4ビットの値が8～15の場合、キャリーフラグの値は不定になります。

フラグ：

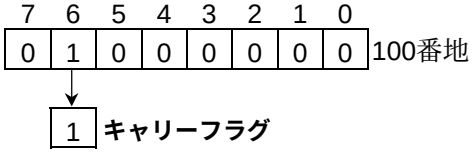
S	Z	H	V	N	C
–	–	–	–	–	*

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = srcのビットnumの内容がセットされます。

実行例：100番地のメモリの内容が01000000B (2進数) のとき、

LDCF 6, (100H)

を実行すると、キャリーフラグは1になります。



LDD dst, src

< Load Decrement **逆方向のブロック部分転送** >

動　作：dst ← src, BC ← BC − 1

説　明：srcの内容が、dstへ転送されます。その後、BCレジスタの内容が−1されます。なお、srcおよびdstのアドレッシングモードはポストデクリメントのレジスタ間接に限られます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	×	LDD<W>	[(XDE−),(XHL−)]	1 0 0 z 0 0 1 1 0 0 0 1 0 0 1 0
○	○	×	LDD<W>	(XIX−),(XIY−)	1 0 0 z 0 1 0 1 0 0 0 1 0 0 1 0

[ ] はその内側の記述が省略可能であることを示しています。

フラグ：

S	Z	H	V	N	C
−	−	0	*	0	−

S = 変化なし。
Z = 変化なし。
H = 0にクリアされます。
V = 命令実行後、BCレジスタの値が0のときは0、それ以外のときは1がセットされます。
N = 0にクリアされます。
C = 変化なし。

実行例：XIXレジスタが00123456Hで、XIYレジスタが00335577HでBCレジスタが0700Hのとき、

```
LDD (XIX−), (XIY−)
```

を実行すると、335577H番地の内容が123456H番地へ転送され、XIXレジスタは123455H、XIYレジスタは00335576H、BCレジスタは06FFHになります。

LDDR dst, src

< Load Decrement Repeat **逆方向のブロック転送** >

動　作：dst ← src, BC ← BC – 1, Repeat until BC = 0

説　明：srcの内容が、dstへ転送されます。その後、BCレジスタの内容が–1され、その値が0でなければ、再び前記動作を繰り返します。なお、srcおよびdstのアドレッシングモードはポストデクリメントのレジスタ間接に限られます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	×	LDDR<W>	[(XDE–),(XHL–)]	1 0 0 z 0 0 1 1 0 0 0 1 0 0 1 1
○	○	×	LDDR<W>	(XIX–),(XIY–)	1 0 0 z 0 1 0 1 0 0 0 1 0 0 1 1

[ ] はその内側の記述が省略可能であることを示しています。

補　足：割り込み要求は、1データを転送するごとにサンプリングされます。

フラグ：

S	Z	H	V	N	C
–	–	0	0	0	–

S = 変化なし。
Z = 変化なし。
H = 0にクリアされます。
V = 0にクリアされます。
N = 0にクリアされます。
C = 変化なし。

実行例：XIXレジスタが00123456Hで、XIYレジスタが00335577Hで、BCレジスタが0003Hのとき、

```
LDDR (XIX–), (XIY–)
```

を実行すると、

335577H番地の内容が123456H番地へ、
335576H番地の内容が123455H番地へ、
335575H番地の内容が123454H番地へ

転送され、XIXレジスタは00123453H、XIYレジスタは00335574H、BCレジスタは0000Hになります。

LDF num

< Load Register File Pointer **レジスタバンクの設定**>

動　作：RFP<2: 0> ← num

説　明：numの値が、ステータスレジスタSR内のレジスタファイルポインタRFP<2:0>へ転送されます。
なお、RFP2は0に固定されています。

詳　細：

ニモニック	コード
LDF #3	0 0 0 1 0 1 1 1 0 0 0 0 0 #3

補　足：オペランドの値は、0～3まで指定できます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

LDI dst, src

< Load Increment **正方向のブロック部分転送** >

動　作：dst ← src, BC ← BC – 1

説　明：srcの内容が、dstへ転送されます。その後、BCレジスタの内容が–1されます。なお、srcおよびdstのアドレッシングモードはポストインクリメントのレジスタ間接に限られます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	×	LDI<W>	[(XDE+),(XHL+)]	1 0 0 z 0 0 1 1 0 0 0 1 0 0 0 0
○	○	×	LDI<W>	(XIX+),(XIY+)	1 0 0 z 0 1 0 1 0 0 0 1 0 0 0 0

[ ] はその内側の記述が省略可能であることを示しています。

フラグ：

S	Z	H	V	N	C
–	–	0	*	0	–

S = 変化なし。
Z = 変化なし。
H = 0にクリアされます。
V = 命令実行後、BCレジスタの値が0のときは0、それ以外のときは1がセットされます。
N = 0にクリアされます。
C = 変化なし。

実行例：XIXレジスタが00123456Hで、XIYレジスタが00335577Hで
BCレジスタが0700Hのとき、

```
LDI (XIX+), (XIY+)
```

を実行すると、335577H番地の内容が123456H番地へ転送され、XIXレジスタは00123457H、XIYレジスタは00335578H、BCレジスタは06FFHになります。

LDIR dst, src

< Load Increment Repeat **正方向のブロック転送** >

動　作：dst ← src, BC ← BC – 1, Repeat until BC = 0

説　明：srcの内容が、dstへ転送されます。その後、BCレジスタの内容が–1され、その値が0でなければ、再び前記動作を繰り返します。なお、srcおよびdstのアドレッシングモードはポストインクリメントのレジスタ間接に限られます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	LDIR<W>	[(XDE+),(XHL+)]	1 0 0 z 0 0 1 1 0 0 0 1 0 0 0 1
○	○	×	LDIR<W>	(XIX+),(XIY+)	1 0 0 z 0 1 0 1 0 0 0 1 0 0 0 1

[ ] はその内側の記述が省略可能であることを示しています。

補　足：割り込み要求は、1データを転送するごとにサンプリングされます。

フラグ：

S	Z	H	V	N	C
–	–	0	0	0	–

S = 変化なし。
Z = 変化なし。
H = 0にクリアされます。
V = 0にクリアされます。
N = 0にクリアされます。
C = 変化なし。

実行例：XIXレジスタが00123456Hで、XIYレジスタが00335577Hで、
BCレジスタが0003Hのとき、
　LDIR (XIX+), (XIY+)
を実行すると、
　335577H番地の内容が123456H番地へ、
　335578H番地の内容が123457H番地へ、
　335579H番地の内容が123458H番地へ、
転送され、XIXレジスタは00123459H、XIYレジスタは0033557AH、BCレジスタは0000Hになります。

LDX dst, src

< Load eXtract **抜き取り転送** >

動　作：dst ← src

説　明：srcの内容が、dstへ転送されます。この命令コードは、1バイトおきに有効コードが割り当てられており、16ビットデータバスモードで、8ビットデータバスメモリ上のコードをフェッチして、転送動作を行うための命令です。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	×	×	LDX	(#8), #	1 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 #8 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0

補　足：第2、第4、第6命令コードの値が00Hでなくても、正しく命令は動作します。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

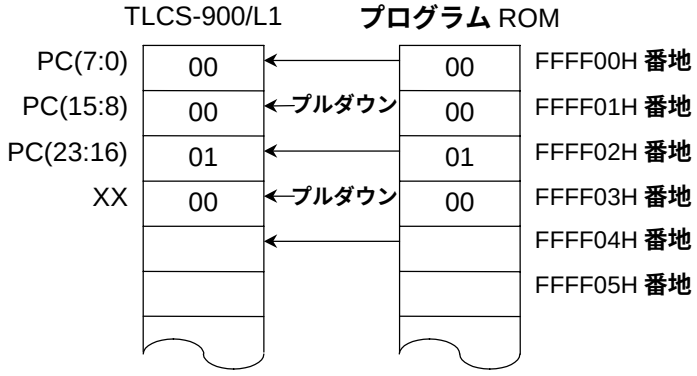
本命令は、CPUがリセット後プログラムをフェッチする際に900/L1に設定されているバス幅が16ビットで、外部プログラムROMのバス幅が8ビットの場合に使用します。
下記表に、その条件を示します。

製品名	AM0 端子	AM1 端子	プログラム ROM のバス幅	その他のメモリのバス幅	LDX 命令
TMP91C815	0	0	8 ビット	8 ビット	使用しない
	1	0	8 ビット	8/16 ビット	使用する
			16 ビット	8/16 ビット	使用しない
	1	1	禁止	—	—

実行例：TMP91C815を例に、AM0, 1 = 1, 0、プログラムROM以外のメモリが16ビットで、8ビット幅プログラムROMから実行させる場合を示します。リセット後は、16ビット幅のデータバスのモードでリセットベクタをリードします。このため、8ビット幅のデータバスを持つ外付けメモリでプログラムをスタートする場合、上位側のデータバスD8～15端子にプルアップ/ダウンを接続することによりリセットベクタのPC(15:8)値を入力する必要があります。

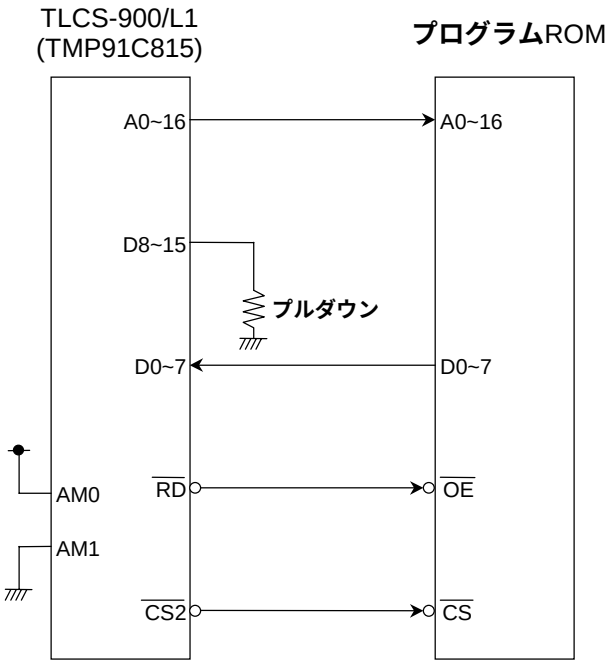
例えばリセットベクタを010000H番地にする場合、プログラムROMのFFFF00H番地に010000Hを置きD8～15端子をプルダウンします。それによりPC(15:8)値として00Hが入力できるようになります。そして、プログラムROMの010000H番地にLDX命令を置きます。

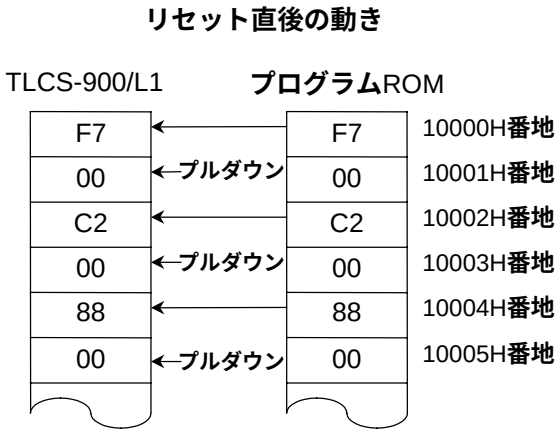
リセット直後の動き



LDX (0C2H), 88H

上記の命令を実行すると、内蔵のプログラマブルチップセレクト/ウェイトコントローラの0C2H番地のコントロールレジスタに88Hのデータが書き込まれ、その結果、CS2空間は、8ビット幅のデータバスで2WAITモード になり、次の命令から8ビットのバス幅でプログラムをフェッチし実行します。





(注意) リセットベクタPC(15:8)を入力するためのD8～15端子に接続するプルアップ/ダウンはD8～15端子からの、データ出力と衝突し、消費電流が増加します。
従って、それが問題な場合は、上記処理終了後、プルアップ/ダウンをカットすることが必要です。

LINK dst, num

< Link **スタックフレームの生成** >

動　作：(–XSP) ← dst, dst ← XSP, XSP ← XSP + num

説　明：dstの内容がスタック領域へ退避されます。次に、スタックポインタXSPの内容がdstへ転送されます。最後にスタックポインタXSPの内容とnumの内容(符号付き)が加算され、スタックポインタXSPへ転送されます。この命令はスタック領域にローカルな変数エリアを–numバイトだけ確保するときに使用します。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	×	○	LINK	r, d16	1 1 1 0 1 r 0 0 0 0 1 1 0 0 d<7:0> d<15:8>

フラグ：

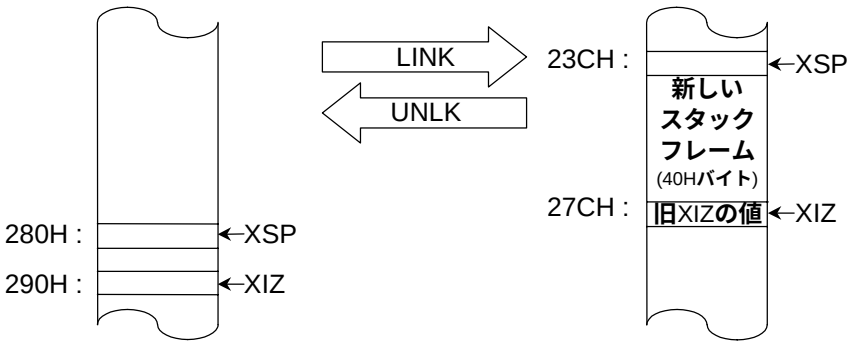
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：スタックポインタXSPが280Hで、XIZレジスタが290Hのとき、

LINK XIZ, –40H

を実行すると、メモリの27CH番地にデータ00000290H(ロングデータ)が書き込まれ、XIZレジスタは27CHに、スタックポインタXSPは23CHになります。



MDEC1 num, dst

< Modulo Decrement 1 **モジュロ減少**1 >

動　作：if (dst mod num) = 0 then dst ← dst + (num − 1) else dst ← dst − 1.

説　明：dstのモジュロnumが0の場合、dstにnum−1が加算されます。それ以外の場合、dstから1が減算されます。この命令は、循環構造のメモリテーブルポインタを操作するとき、使用します。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	×	MEDEC1	#, r	1 1 0 1 1 | r | 0 0 1 1 1 1 0 0 #<7:0>-1 #<15:8>

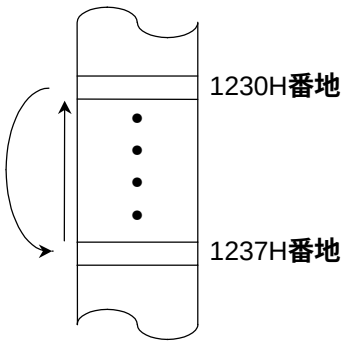
補　足：オペランドの#は、2のn乗(nは1〜15)の値に限られます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：IXレジスタを1230H〜1237Hの範囲内で循環デクリメントする例です。
IXレジスタが1231Hのとき、
　MDEC1 8, IX
を実行すると、IXレジスタは1230Hになり、再び
　MDEC1 8,IX
を実行すると、IXレジスタのモジュロ8(8で割った余り)が0なので、IXレジスタに8 − 1が加算され、
IXレジスタは1237Hになります。



MDEC2 num, dst

< Modulo Decrement 2 **モジュロ減少**2 >

動　作：if (dst mod num) = 0 then dst ← dst + (num – 2) else dst ← dst – 2.

説　明：dstのモジュロnumが0の場合、dstにnum – 2が加算されます。それ以外の場合、dstから2が減算されます。この命令は、循環構造のメモリテーブルポインタを操作するとき、使用します。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	×	MDEC2	#, r	1 1 0 1 1 r 0 0 1 1 1 1 0 1 #<7:0>-2 #<15:8>

補　足：オペランドの#は、2のn乗(nは2～15)の値に限られます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

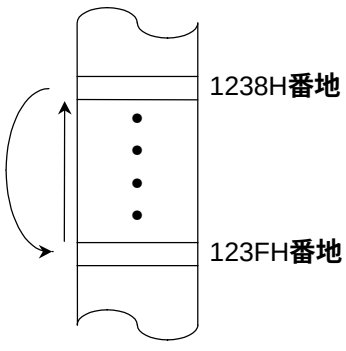
実行例：IXレジスタを1238H～123FHの範囲内で循環デクリメントする例です。
IXレジスタが123AHのとき、

```
MDEC2 8,IX
```

を実行すると、IXレジスタは1238Hになり、再び

```
MDEC2 8,IX
```

を実行すると、IXレジスタのモジュロ8(8で割った余り)が0なので、IXレジスタに8 – 2が加算され、
IXレジスタは123EHになります。



MDEC4 num, dst

< Modulo Decrement 4 **モジュロ減少**4 >

動　作：if (dst mod num) = 0 then dst ← dst + (num – 4) else dst ← dst – 4.

説　明：dstのモジュロnumが0の場合、dstにnum – 4が加算されます。それ以外の場合、dstから4が減算されます。この命令は、循環構造のメモリテーブルポインタを操作するとき、使用します。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	×	MDEC4	#, r	1 1 0 1 1 r 0 0 1 1 1 1 1 0 #<7:0>-4 #<15:8>

補　足：オペランドの#は、2のn乗(nは3～15)の値に限られます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

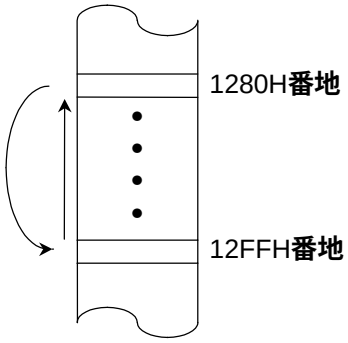
実行例：IXレジスタを1280H～12FFHの範囲内で循環デクリメントする例です。
IXレジスタが1284Hのとき、

```
MDEC4 80H,IX
```

を実行すると、IXレジスタは1280Hになり、再び

```
MDEC4 80H,IX
```

を実行すると、IXレジスタのモジュロ80H(80Hで割った余り)が0なので、IXレジスタに80H – 4が加算され、IXレジスタは12FCHになります。



MINC1 num, dst

< Modulo Increment 1 **モジュロ増加**1 >

動　作：if (dst mod num) = (num – 1) then dst ← dst – (num – 1) else dst ← dst + 1.

説　明：dstのモジュロnumがnum – 1の場合、dstからnum – 1が減算されます。それ以外の場合、dstに1が加算されます。この命令は、循環構造のメモリテーブルポインタを操作するとき、使用します。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
×	○	×	MINC1	#, r	

コード								
1	1	0	1	1	r			
0	0	1	1	1	0	0	0	
#<7:0>-1								
#<15:8>								

補　足：オペランドの#は、2のn乗(nは1～15)の値に限られます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

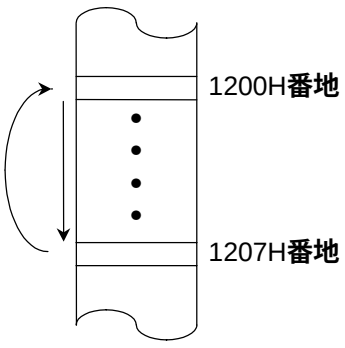
実行例：IXレジスタを1200H～1207Hの範囲内で循環インクリメントする例です。
IXレジスタが1206Hのとき、

```
MINC1 8, IX
```

を実行すると、IXレジスタは1207Hになり、再び

```
MINC1 8, IX
```

を実行すると、IXレジスタのモジュロ8 (8で割った余り)が8 – 1なので、IXレジスタから8 – 1が減算され、IXレジスタは1200Hになります。



MINC2 num, dst

< Modulo Increment 2 **モジュロ増加2** >

動　作：if (dst mod num) = (num – 2) then dst ← dst – (num – 2) else dst ← dst + 2.

説　明：dstのモジュロnumがnum – 2の場合、dstからnum – 2が減算されます。それ以外の場合、dstに2が加算されます。この命令は、循環構造のメモリテーブルポインタを操作するとき、使用します。

詳　細：

サイズ バイト	ワード	ロング	ニモニック	コード
×	○	×	MINC2 #, r	

1	1	0	1	1	r		
0	0	1	1	1	0	0	1
#<7:0>-2							
#<15:8>							

補　足：オペランドの#は、2のn乗(nは2～15)の値に限られます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

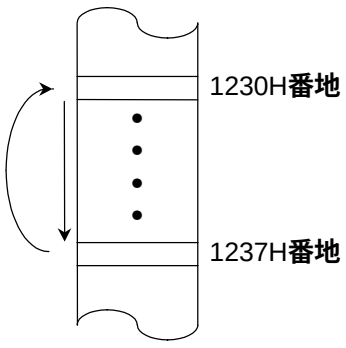
実行例：IXレジスタを1230H～1237Hの範囲内で循環インクリメントする例です。
IXレジスタが1234Hのとき、

```
MINC2 8,IX
```

を実行すると、IXレジスタは1236Hになり、再び

```
MINC2 8,IX
```

を実行すると、IXレジスタのモジュロ8 (8で割った余り)が8 – 2なので、IXレジスタから8 – 2が減算され、IXレジスタは1230Hになります。



MINC4 num, dst

< Modulo Increment 4 **モジュロ増加**4 >

動　作：if (dst mod num) = (num – 4) then dst ← dst – (num – 4) else dst ← dst + 4.

説　明：dstのモジュロnumがnum – 4の場合、dstからnum – 4が減算されます。それ以外の場合、dstに4が加算されます。この命令は、循環構造のメモリテーブルポインタを操作するとき、使用します。

詳　細：

サイズ バイト	ワード	ロング	ニモニック	コード
×	○	×	MINC4 #, r	

1	1	0	1	1	r		
0	0	1	1	1	0	1	0
#<7:0>-4							
#<15:8>							

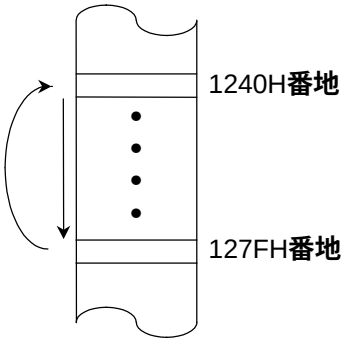
補　足：オペランドの#は、2のn乗(nは3～15)の値に限られます。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：IXレジスタを1240H～127FHの範囲内で循環インクリメントする例です。
IXレジスタが1278Hのとき、
　MINC4 40H,IX
を実行すると、IXレジスタは127CHになり、再び
　MINC4 40H,IX
を実行すると、IXレジスタのモジュロ40H (40Hで割った余り)が40H – 4なので、IXレジスタから
40H – 4が減算され、IXレジスタは1240Hになります。



MIRR dst

< Mirror **ミラー交換** >

動　作：dst<MSB : LSB> ← dst<LSB : MSB>

説　明：dstの内容が、ビットパターンのイメージでミラー交換されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	×	MIRR	r	1 1 0 1 1 r 0 0 0 1 0 1 1 0

フラグ：

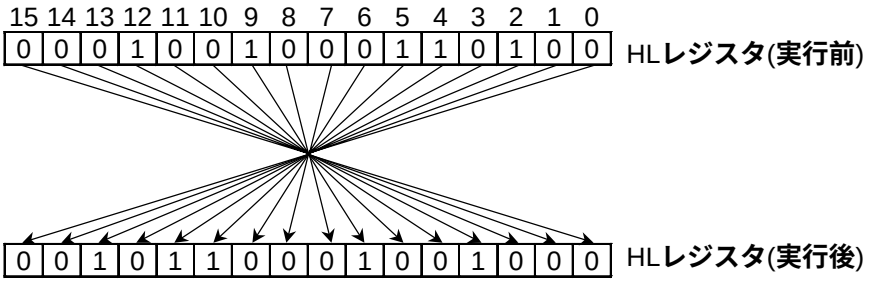
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：HLレジスタが0001 0010 0011 0100B(2進数)のとき、

MIRR HL

を実行すると、HLレジスタは0010 1100 0100 100B(2進数)になります。



MUL dst, src

< Multiply **符号なし乗算** >

動　作：dst ← dst<下位半分> × src (符号なし)

説　明：dstの下位半分の内容とsrcの内容が符号なし乗算され、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	MUL	RR, r	1 1 0 z 1 r 0 1 0 0 0 R
○	○	×	MUL	rr, #	1 1 0 z 1 r 0 0 0 0 1 0 0 0 #<7:0> #<15:8>
○	○	×	MUL	RR, (mem)	1 m 0 z m m m m 0 1 0 0 0 R

補　足：演算サイズがバイトのときは、dst(ワード)←dst(バイト)×src(バイト)、演算サイズがワードのときは、dst(ロング)←dst(ワード)×src(ワード)になります。オペランドのdstの記述は、演算結果のサイズに合わせてください。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：IXレジスタが1234Hで、IYレジスタが89ABHのとき、

```
MUL XIX, IY
```

を実行すると、IXレジスタの内容とIYレジスタの内容が符号なし乗算され、XIXレジスタは09C9FCBCHになります。

注　意：MUL RR,rとMUL RR,(mem)命令のRRは、下記に示す表のようになります。

演算サイズがバイト
(16 **ビット**← 8 **ビット** ×8 **ビット**)

RR	**コード** R
WA	001
BC	011
DE	101
HL	111
IX	**指定不可**!
IY	
IZ	
SP	

演算サイズがワード
(32 **ビット** ← 16 **ビット** ×16 **ビット**)

RR	**コード** R
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

MUL rr,#命令のrrは、下記に示す表のようになります。

演算サイズがバイト
(16 **ビット**← 8 **ビット** ×8 **ビット**)

rr	**コード** r
WA	001
BC	011
DE	101
HL	111
IX	C7H : F0H
IY	C7H : F4H
IZ	C7H : F8H
SP	C7H : FCH
	1 バイト目　2 バイト目

(注) **その他のワードレジスタも** IX～SP **と同様の拡張コード方式で、すべて指定可能。**

演算サイズがワード
(32 **ビット** ← 16 **ビット** ×16 **ビット**)

rr	**コード** r
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

(注) **その他のロングワードレジスタも拡張コード方式ですべて指定可能。**

MULA dst

< Multiply and Add **符号付き積和演算** >

動　作：dst ← dst + (XDE) × (XHL), XHL ← XHL − 2

説　明：XDEレジスタで示されたメモリデータ(16ビット)とXHLレジスタで示されたメモリデータ(16ビット)の内容が符号付き乗算され、その結果(32ビット)とdstの内容(32ビット)が加算され、dst(32ビット)へ転送されます。その後、XHLレジスタの内容が−2されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	×	MULA	rr	1 1 0 1 1 r 0 0 0 1 1 0 0 1

補　足：オペランドのdstの記述は、演算結果のサイズ(ロング)に合わせてください。

フラグ：

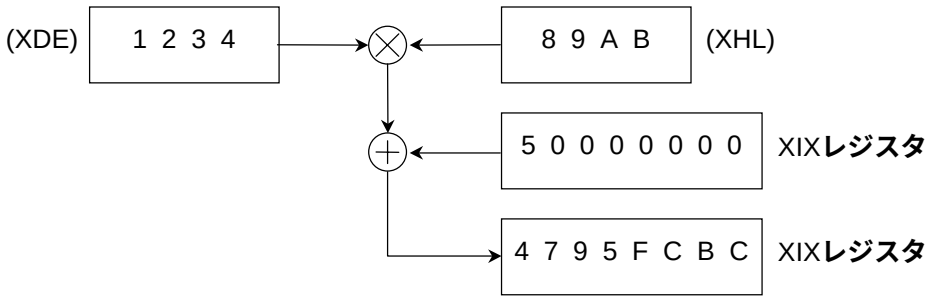
S	Z	H	V	N	C
*	*	–	*	–	–

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 変化なし。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 変化なし。
C = 変化なし。

実行例：XIXレジスタが50000000Hで、XDEレジスタが100H,XHLが200H,100H番地のメモリの内容(ワード)が1234H,200H番地のメモリの内容(ワード)が89ABHのとき、

MULA XIX

を実行すると、XIXレジスタは4795FCBCH、XHLレジスタは1FEHになります。



MULS dst, src

< Multiply Signed **符号付き乗算** >

動　作：dst ← dst<下位半分> × src (符号付き)

説　明：dstの下位半分の内容とsrcの内容が符号付き乗算され、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	MULS	RR, r	1 1 0 z 1 r 0 1 0 0 1 R
○	○	×	MULS	rr, #	1 1 0 z 1 r 0 0 0 0 1 0 0 1 #<7:0> #<15:8>
○	○	×	MULS	RR, (mem)	1 m 0 z m m m m 0 1 0 0 1 R

補　足：演算サイズがバイトのときは、dst(ワード)←dst(バイト)×src(バイト)、演算サイズがワードのときは、dst(ロング)←dst(ワード)×src(ワード)になります。オペランドのdstの記述は、演算結果のサイズに合わせてください。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：IXレジスタが1234Hで、IYレジスタが89ABHのとき、

```
MULS XIX, IY
```

を実行すると、IXレジスタの内容とIYレジスタの内容が符号付き乗算され、XIXレジスタはF795FCBCHになります。

補　足：MULS RR,rとMULS RR,(mem)命令のRRは、下記に示す表のようになります。

演算サイズがバイト
(16 **ビット** ← 8 **ビット** ×8 **ビット**)

RR	**コード** R
WA	001
BC	011
DE	101
HL	111
IX	**指定不可**!
IY	
IZ	
SP	

演算サイズがワード
(32 **ビット** ← 16 **ビット** ×16 **ビット**)

RR	**コード** R
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

MULS rr,#命令のrrは、下記に示す表のようになります。

演算サイズがバイト
(16 **ビット** ← 8 **ビット** ×8 **ビット**)

rr	**コード** r
WA	001
BC	011
DE	101
HL	111
IX	C7H : F0H
IY	C7H : F4H
IZ	C7H : F8H
SP	C7H : FCH
	1 **バイト目** 2 **バイト目**

(**注**) **その他のワードレジスタも** IX～SP **と同様の拡張コード方式で、すべて指定可能。**

演算サイズがワード
(32 **ビット** ← 16 **ビット** ×16 **ビット**)

rr	**コード** r
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

(**注**) **その他のロングワードレジスタも拡張コード方式ですべて指定可能。**

NEG dst

< Negate 2**の補数** >

動　作：dst ← 0 – dst

説　明：ゼロからdstの内容が減算され、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	NEG	r	1 1 0 z 1 r 0 0 0 0 0 1 1 1

フラグ：

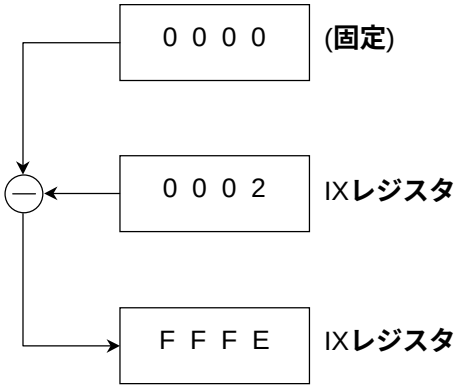
S	Z	H	V	N	C
*	*	*	*	1	*

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 1にセットされます。
C = 演算結果、最上位ビットからボローが発生したときは1、それ以外のときは0がセットされます。

実行例：IXレジスタ0002Hのとき、

NEG IX

を実行すると、IXレジスタはFFFEHになります。



NOP

< No Operation **何もしない** >

動　作：何もしない

説　明：何もせず、次の命令の実行に移ります。この命令は、オブジェクトコードが00Hです。

詳　細：

ニモニック	コード
NOP	0 0 0 0 0 0 0 0

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

OR dst, src

< Or **論理和** >

動　作：dst ← dst OR src

説　明：dstの内容とsrcの内容が論理和演算され、dstへ転送されます。

(真理値表)

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

詳　細：

サイズ			ニモニック		コード
バイト	ワード	ロング			
○	○	○	OR	R, r	1 1 z z 1 r 1 1 1 0 0 R
○	○	○	OR	r, #	1 1 z z 1 r 1 1 0 0 1 1 1 0 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	OR	R, (mem)	1 m z z m m m m 1 1 1 0 0 R
○	○	○	OR	(mem), R	1 m z z m m m m 1 1 1 0 1 R
○	○	×	OR<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 1 1 0 #<7:0> #<15:8>

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	0

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 0にセットされます。
V = 演算結果のパリティ(1の数)が偶数のときは1、奇数のときは0がセットされます。ただし、オペランド長が32ビットの場合は、不定値がセットされます。
N = 0にクリアされます。
C = 0にクリアされます。

実行例：HLレジスタが7350H,IXレジスタが3456Hのとき、

OR HL, IX

を実行すると、HLレジスタは7756Hになります。

	0111	0011	0101	0000	← HLレジスタ (実行前)
OR)	0011	0100	0101	0110	← IXレジスタ (実行前)
	0111	0111	0101	0110	← HLレジスタ (実行後)

ORCF num, src

< OR Carry Flag **キャリーフラグとの1ビット論理和** >

動　作：CY ← CY OR src<num>

説　明：キャリーフラグCYの内容とsrcのビットnumの内容が論理和演算され、キャリーフラグCYへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	ORCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 0 0 1 0 0 0 0 #4
○	○	×	ORCF	A, r	1 1 0 z 1 r 0 0 1 0 1 0 0 1
○	×	×	ORCF	#3, (mem)	1 m 1 1 m m m m 1 0 0 0 1 #3
○	×	×	ORCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 0 0 1

補　足：ビットnumがAレジスタで指定された場合、Aレジスタの下位4ビットの値がビットnumとして使われます。オペランドがバイトのとき、ビットnumの下位の値が8～15の場合、演算結果は不定になります。

フラグ：

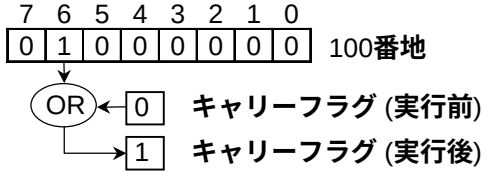
S	Z	H	V	N	C
–	–	–	–	–	*

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = キャリーフラグCYの内容とsrcのビットnumの内容の論理和演算された値がセットされます。

実行例：100番地のメモリの内容が01000000B(2進数)で、キャリーフラグCYが0のとき、

ORCF 6, (100H)

を実行すると、キャリーフラグは1になります。



PAA dst

< Pointer Adjust Accumulator **ポインタ偶数補正** >

動　作：if dst <LSB> = 1 then dst ← dst + 1

説　明：dstのLSB(最下位ビット)が1の場合、dstに1が加算されます。dstのLSBが0の場合、何もしません。
この命令は、dstの内容を偶数にするためのものです。TLCS-900では、メモリ中の16ビットデータ、または、32ビットデータをアクセスするとき、そのデータが偶数番地から配置されている方が、奇数番地からの場合に比べて、バスサイクル数が1回少なくなります。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	○	PAA	r	1 1 z z 1 r 0 0 0 1 0 1 0 0

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：XIZレジスタが00234567Hのとき、

```
PAA XIZ
```

を実行すると、XIZレジスタは+1されて00234568Hになります。

POP dst

< Pop **スタック領域からの転送** >

動　作：dst ← (XSP+)

バイト時　：dst←(XSP), XSP ← XSP + 1
ワード時　：dst←(XSP), XSP ← XSP + 2
ロング時　：dst←(XSP), XSP ← XSP + 4

説　明：まず、スタックポインタXSPで示されたメモリ番地の内容が、dstへ転送されます。次に、スタックポインタXSPがオペランドサイズのバイト長だけ加算されます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	×	×	POP	F	0 0 0 1 1 0 0 1
○	×	×	POP	A	0 0 0 1 0 1 0 1
×	○	○	POP	R	0 1 0 s 1 R
○	○	○	POP	r	1 1 z z 1 r 0 0 0 0 0 1 0 1
○	○	×	POP<W>	(mem)	1 m 1 1 m m m m 0 0 0 0 0 1 z 0

フラグ：

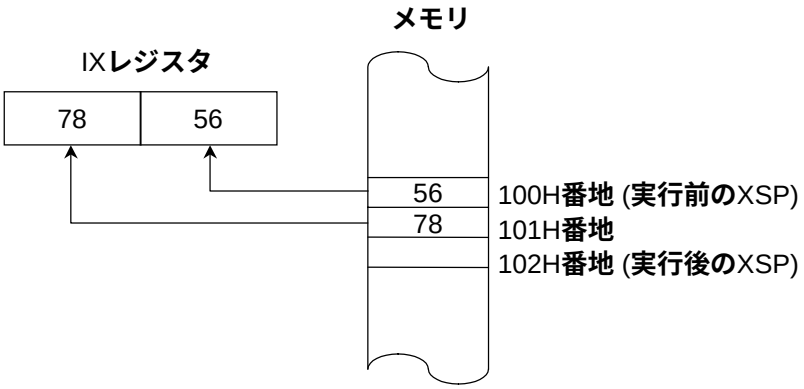
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。
(注) POP Fを実行すると、フラグはすべて変化します。

実行例：スタックポインタXSPが0100Hで、100H番地の内容が56H、101H番地の内容が78Hのとき、

```
POP IX
```

を実行すると、IXレジスタは7856Hになります。また、スタックポインタXSPは、0102Hになります。



POP SR

< Pop SR **ステータスレジスタのスタック領域からの転送** >

動　作：SR ← (XSP+)

説　明：スタックポインタXSPで示されたメモリ番地の内容が、ステータスレジスタへ転送されます。
その後、スタックポインタXSPの内容が+2されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
×	○	×	POP	SR	0 0 0 0 0 0 1 1

フラグ：

S	Z	H	V	N	C
*	*	*	*	*	*

S=
Z=
H=
V=
N=
C=
} スタックポインタXSPで示されたメモリ番地の内容がセットされます。

補　足：本命令の実行は「DI」状態で行ってください。この命令が実際に実行されるタイミングは、この命令がフェッチされるタイミングより、数ステート遅れます。これは、本CPUが命令キュー(4バイト)とパイプライン処理方式を採用しているためです。
また、本命令はステータスレジスタの全ビットを書き替える動作を行いますので、値を書き替えてはいけないビット(MAXビットは1しかライトしてはいけません)は、書き替わることがないようにしてください。

PUSH SR

< Push SR **ステータスレジスタのスタック領域への転送** >

動　作：(–XSP) ← SR

説　明：スタックポインタXSPの内容が–2されます。その後、ステータスレジスタSRの内容が、スタックポインタXSPで示されたメモリ番地へ転送されます。

詳　細：

サイズ			ニモニック		コード
バイト	ワード	ロング			
×	○	×	PUSH	SR	0 0 0 0 0 0 1 0

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

PUSH src

< Push **スタック領域への転送** >

動　作：(–XSP) ← src

バイト時： XSP ← XSP – 1, (XSP) ← src
ワード時： XSP ← XSP – 2, (XSP) ← src
ロング時： XSP ← XSP – 4, (XSP) ← src

説　明：まず、スタックポインタXSPがオペランドサイズのバイト長だけ減算されます。次に、スタックポインタXSPが示すメモリ番地へ、srcの内容が転送されます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	×	×	PUSH	F	0 0 0 1 1 0 0 0
○	×	×	PUSH	A	0 0 0 1 0 1 0 0
×	○	○	PUSH	R	0 0 1 s 1 R
○	○	○	PUSH	r	1 1 z z 1 r 0 0 0 0 0 1 0 0
○	○	×	PUSH <W>	#	0 0 0 0 1 0 z 1 #<7:0> #<15:8>
○	○	×	PUSH <W>	(mem)	1 m 0 z m m m m 0 0 0 0 0 1 0 0

フラグ：

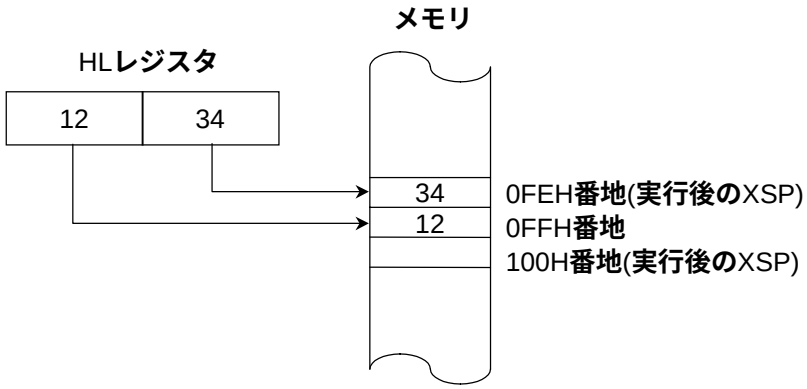
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：スタックポインタXSPが0100Hで、HLレジスタが1234Hのとき、

PUSH HL

を実行すると、00FEH番地は34Hに、00FFH番地は12Hになります。また、スタックポインタXSPは、00FEHになります。



RCF

< Reset Carry Flag **キャリーフラグのリセット** >

動　作：CY ← 0

説　明：キャリーフラグCYが0にリセットされます。

詳　細：

ニモニック	コード
RCF	0 | 0 | 0 | 1 | 0 | 0 | 0 | 0

フラグ：

S	Z	H	V	N	C
–	–	0	–	0	0

S = 変化なし。
Z = 変化なし。
H = 0にリセットされます。
V = 0にリセットされます。
N = 変化なし。
C = 0にリセットされます。

RES num, dst

< Reset 1**ビットのリセット** >

動　作：dst <num> ← 0

説　明：dstのビットnumが0にリセットされます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	RES	#4, r	1 1 0 z 1 r 0 0 1 1 0 0 0 0 0 0 0 0 #4
○	×	×	RES	#3, (mem)	1 m 1 1 m m m m 1 0 1 1 0 #3

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：100番地のメモリの内容が00100111B(2進数)のとき、

RES 5, (100H)

を実行すると、100番地のメモリの内容は00000111B(2進数)になります。

RET condition

< Return **リターン** >

動　作：if ccが真　then 32ビット　PC ← (XSP), XSP ← XSP + 4.

説　明：オペランドのconditionが真の場合、スタック領域からプログラムカウンタPCへリターンアドレスがPOPされます。

詳　細：

ニモニック		コード
RET		0 0 0 0 1 1 1 0
RET	cc	1 0 1 1 0 0 0 0 1 1 1 1 c c

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：スタックポインタXSPが0FCHで、0FCH番地のメモリの内容が9000H(ロングデータ)のとき、

```
RET
```

を実行すると、スタックポインタXSPは100Hになり、9000H番地にジャンプ(リターン)します。

RETD num

< Return and Deallocate **リターン&パラメータ領域の削除** >

動　作：32ビットPC ← (XSP), XSP ← XSP + 4, XSP ← XSP + num

説　明：まず、スタック領域からプログラムカウンタPCへリターンアドレスがPOPされます。次に、スタックポインタXSPにnum(符号付)の値が加算されます。

詳　細：

ニモニック	コード
RETD d16	0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 d<7:0> d<15:8>

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

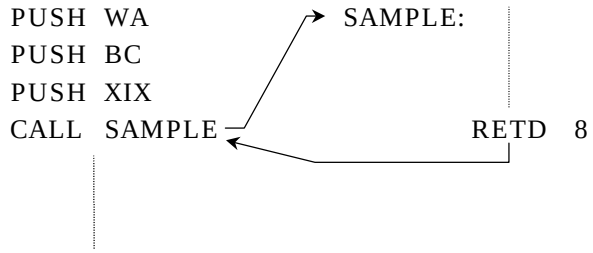
S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：スタックポインタXSPが0FCHで、0FCH番地のメモリの内容が9000H(ロングデータ)のとき、

```
RETD 8
```

を実行すると、スタックポインタXSPは0FCH + 4 + 8 → 108Hになり、9000H番地にジャンプ(リターン)します。

下記にRETD命令の活用例を示します。この例では、サブルーチンを呼ぶ前に8バイトのパラメータをスタックにPUSHし、サブルーチン処理終了後、RETD命令で、その使用済のパラメータ領域を削除しています。



RETI

< Return from Interrupt **割り込み処理からのリターン** >

動　作：SR ← (XSP), 32ビットPC ← (XSP + 2),
XSP ← XSP + 6
上記動作後、割り込みネスティングカウンタINTNESTを–1します。

説　明：スタック領域から「ステータスレジスタSRとプログラムカウンタPC」へデータがPOPされます。
上記動作後、割り込みネスティングカウンタINTNESTを–1します。

詳　細：

ニモニック	コード
RETI	0 0 0 0 0 1 1 1

フラグ：

S	Z	H	V	N	C
*	*	*	*	*	*

S = スタック領域からPOPした値になります。
Z = スタック領域からPOPした値になります。
H = スタック領域からPOPした値になります。
V = スタック領域からPOPした値になります。
N = スタック領域からPOPした値になります。
C = スタック領域からPOPした値になります。

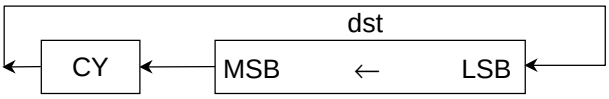
RL num, dst

< Rotate Left **キャリーフラグを含む左ローテート** >

動　作：{CY & dst ← CY & dstの左ローテート値} Repeat num回

説　明：キャリーフラグCYとdstの連結した内容が左へローテートされます。num回繰り返されます。

説明図：



詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	○	RL	#4, r	1 1 z z 1 r 1 1 1 0 1 0 1 0 0 0 0 0 #4
○	○	○	RL	A, r	1 1 z z 1 r 1 1 1 1 1 0 1 0
○	○	×	RL<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 0 1 0

補　足：ローテート回数numがAレジスタで指定された場合、Aレジスタの下位4ビットの値がローテート回数として使われます。0を指定すると、16回ローテートされます。
dstがメモリの場合、ローテート回数は1回に制限されます。

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	*

S = ローテート後のdstの最上位ビットの値がセットされます。
Z = ローテート後のdstの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = ローテート後、dstのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は不定値がセットされます。
N = 0にリセットされます。
C = ローテート後の値がセットされます。

実行例：HLレジスタが6230Hで、キャリーフラグCYが1のとき、

```
RL 4, HL
```

を実行すると、HLレジスタは230BH、キャリーフラグCYは0になります。

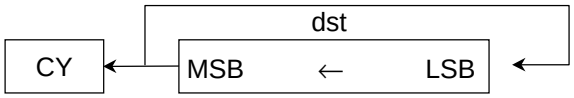
RLC num, dst

< Rotate Left without Carry **キャリーフラグを含まない左ローテート** >

動　作：{CY← dst <MSB>, dst ← dstの左ローテート値} Repeat num回

説　明：dstのMSBの内容がキャリーフラグCYへ転送され、dstの内容が左へローテートされます。num回繰り返されます。

説明図：



詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	RLC	#4, r	1 1 z z 1 r 1 1 1 0 1 0 0 0 0 0 0 0 # 4
○	○	○	RLC	A, r	1 1 z z 1 r 1 1 1 1 1 0 0 0
○	○	×	RLC<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 0 0 0

補　足：ローテート回数numがAレジスタで指定された場合、Aレジスタの下位4ビットの値がローテート回数として使われます。0を指定すると、16回ローテートされます。
dstがメモリの場合、ローテート回数は1回に制限されます。

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	*

S = ローテート後のdstの最上位ビットの値がセットされます。
Z = ローテート後のdstの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = ローテート後、dstのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は不定値がセットされます。
N = 0にリセットされます。
C = 最終のローテート前のdstの最上位ビットの値がセットされます。

実行例：HLレジスタが1230Hのとき、

```
RLC 4, HL
```

を実行すると、HLレジスタは2301H、キャリーフラグCYは1になります。

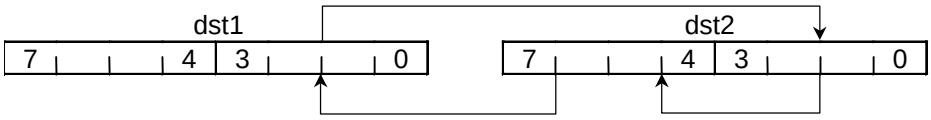
RLD dst1, dst2

< Rotate Left Digit 4**ビットの左回転** >

動　作：dst1<3:0> ← dst2<7:4>, dst2<7:4> ← dst2<3:0>,dst2<3:0> ← dst1 <3:0>

説　明：dst1の下位4ビットと、dst2の内容が、4ビット単位で左へローテートされます。

説明図：



詳　細：

サイズ			**ニモニック**		**コード**
バイト	**ワード**	**ロング**			
○	×	×	RLD	[A,] (mem)	1 m 0 0 m m m m 0 0 0 0 0 1 1 0

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	–

S = ローテート後のAレジスタの最上位ビットの値がセットされます。
Z = ローテート後のAレジスタの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = ローテート後、Aレジスタのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。
N = 0にリセットされます。
C = 変化なし。

実行例：Aレジスタが12Hで、100H番地のメモリの内容が34Hのとき、

RLD A, (100H)

を実行すると、Aレジスタは13H, 100番地のメモリの内容は42Hになります。

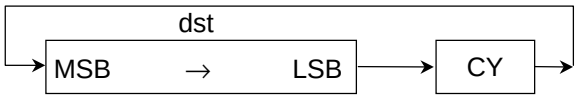
RR num, dst

< Rotate Right **キャリーフラグを含む右ローテート** >

動　作：{CY & dst ← CY & dstの右ローテート値} Repeat num回

説　明：キャリーフラグCYとdstの連結した内容が右へローテートされます。num回繰り返されます。

説明図：



詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	○	RR	#4, r	1 1 z z 1 r 1 1 1 0 1 0 1 1 0 0 0 0 # 4
○	○	○	RR	A, r	1 1 z z 1 r 1 1 1 1 1 0 1 1
○	○	×	RR<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 0 1 1

補　足：ローテート回数numがAレジスタで指定された場合、Aレジスタの下位4ビットの値がローテート回数として使われます。0を指定すると、16回ローテートされます。
dstがメモリの場合、ローテート回数は1回に制限されます。

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	*

S = ローテート後のdstの最上位ビットの値がセットされます。
Z = ローテート後のdstの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = ローテート後、dstのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は不定値がセットされます。
N = 0にリセットされます。
C = ローテート後の値がセットされます。

実行例：HLレジスタが6230Hで、キャリーフラグCYが1のとき、

```
RR 4, HL
```

を実行すると、HLレジスタは1623H、キャリーフラグCYは0になります。

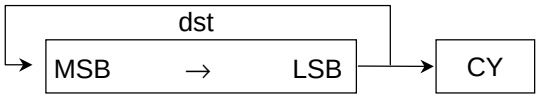
RRC num, dst

< Rotate Right without Carry **キャリーフラグを含まない右ローテート** >

動　作：{CY ← dst <LSB>、dst ← dstの右ローテート値} Repeat num回

説　明：dstのLSBの内容がキャリーフラグCYへ転送され、dstの内容が右へローテートされます。
num回繰り返されます。

説明図：



詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	RRC	#4, r	1 1 z z 1 r 1 1 1 0 1 0 0 1 0 0 0 0 # 4
○	○	○	RRC	A, r	1 1 z z 1 r 1 1 1 1 1 0 0 1
○	○	×	RRC<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 0 0 1

補　足：ローテート回数numがAレジスタで指定された場合、Aレジスタの下位4ビットの値がローテート回数として使われます。0を指定すると、16回ローテートされます。
dstがメモリの場合、ローテート回数は1回に制限されます。

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	*

S = ローテート後のdstの最上位ビットの値がセットされます。
Z = ローテート後のdstの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = ローテート後、dstのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は不定値がセットされます。
N = 0にリセットされます。
C = 最終のローテート前のdstの最上位ビットの値がセットされます。

実行例：HLレジスタが1230Hのとき、

```
RLC 4, HL
```

を実行すると、HLレジスタは0123H、キャリーフラグCYは0になります。

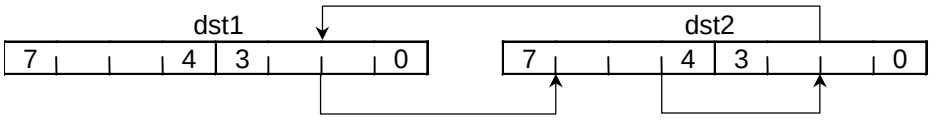
RRD dst1, dst2

< Rotate Right Digit 4**ビットの右回転** >

動　作：dst1<3:0> ← dst2<3:0>, dst2<7:4> ← dst1<3:0>, dst2<3:0> ← dst2<7:4>

説　明：dst1の下位4ビットと、dst2の内容が、4ビット単位で右へローテートされます。

説明図：



詳　細：

サイズ			**ニモニック**		**コード**
バイト	**ワード**	**ロング**			
○	×	×	RRD	[A,] (mem)	1 m 0 0 m m m m 0 0 0 0 0 1 1 1

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	–

S = ローテート後のAレジスタの最上位ビットの値がセットされます。
Z = ローテート後のAレジスタの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = ローテート後、Aレジスタのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。
N = 0にリセットされます。
C = 変化なし。

実行例：Aレジスタが12Hで、100H番地のメモリの内容が34Hのとき、

RRD A, (100H)

を実行すると、Aレジスタは14H, 100番地のメモリの内容は23Hになります。

SBC dst, src

< Subtract with Carry **キャリー付き減算** >

動　作：dst ← dst − src − CY

説　明：dstの内容からsrcの内容とキャリーフラグCYの内容が減算され、dstへ転送されます。

詳　細：

サイズ バイト	ワード	ロング	ニモニック		コード
○	○	○	SBC	R, r	1 1 z z 1 r 1 0 1 1 0 R
○	○	○	SBC	r, #	1 1 z z 1 r 1 1 0 0 1 0 1 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	SBC	R, (mem)	1 m z z m m m m 1 0 1 1 0 R
○	○	○	SBC	(mem), R	1 m z z m m m m 1 0 1 1 1 R
○	○	×	SBC<w>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 0 1 1 #<7:0> #<15:8>

フラグ：

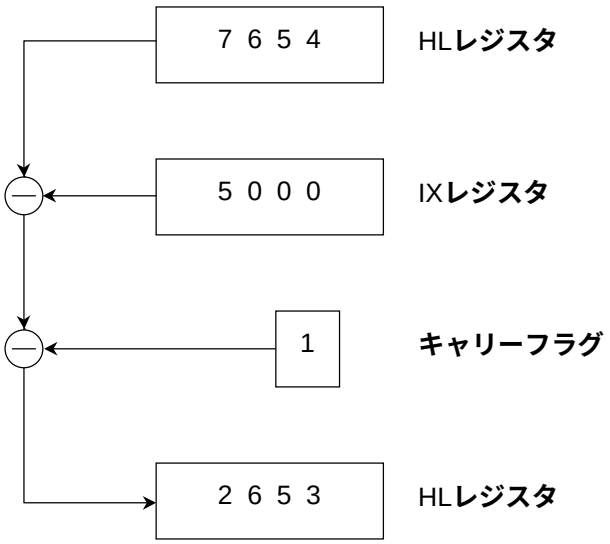
S	Z	H	V	N	C
*	*	*	*	1	*

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は、不定値がセットされます。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 1にセットされます。
C = 演算結果、最上位ビットからボローが発生したときは1、それ以外のときは0がセットされます。

実行例：HLレジスタが7654H,IXレジスタが5000H、キャリーフラグCYが1のとき、

SBC HL, IX

を実行すると、HLレジスタは2653Hになります。



SCC condition, dst

< Set Condition Code **コンディションコードによる値のセット** >

動　作：if ccが真、then dst ← 1 else dst ← 0.

説　明：オペランドのconditionが真の場合、1がdstへ転送され、偽の場合、0がdstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	SCC	cc, r	1 1 0 z 1 r 0 1 1 1 c c

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：Vフラグの内容が1のとき、

SCC OV, HL

を実行すると、HLレジスタは0001Hになります。

SCF

< Set Carry Flag **キャリーフラグのセット** >

動　作：CY ← 1

説　明：キャリーフラグCYが1にセットされます。

詳　細：

ニモニック	コード
SCF	0 | 0 | 0 | 1 | 0 | 0 | 0 | 1

フラグ：

S	Z	H	V	N	C
–	–	0	–	0	1

S = 変化なし。
Z = 変化なし。
H = 0にリセットされます。
V = 変化なし。
N = 0にリセットされます。
C = 1にセットされます。

SET num, dst

< Set 1**ビットのセット** >

動　作：dst <num> ← 1

説　明：dstのビットnumが1にセットされます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	SET	#4,r	1 1 0 z 1 r 0 0 1 1 0 0 0 1 0 0 0 0 #4
○	×	×	SET	#3, (mem)	1 m 1 1 m m m m 1 0 1 1 1 #3

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：100番地のメモリの内容が00000000B(2進数)のとき、

SET 5, (100H)

を実行すると、100番地のメモリの内容は00100000B(2進数)になります。

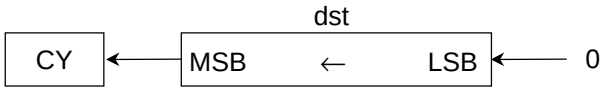
SLA num, dst

< Shift Left Arithmetic **算術左シフト** >

動　作：{CY ← dst<MSB>, dst ← dstの左シフト値、dst<LSB> ← 0} Repeat num回

説　明：dstのMSBの内容がキャリーフラグCYへ転送され、dstの内容が左へシフトされ、0がdstのLSBへ転送されます。これがnum回繰り返されます。

説明図：



詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	SLA	#4, r	1 1 z z 1 r 1 1 1 0 1 1 0 0 0 0 0 0 # 4
○	○	○	SLA	A, r	1 1 z z 1 r 1 1 1 1 1 1 0 0
○	○	×	SLA <W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 1 0 0

補　足：シフト回数numがAレジスタで指定された場合、Aレジスタの下位4ビットの値がシフト回数として使われます。0を指定すると、16回シフトされます。
dstがメモリの場合、シフト回数は1回に制限されます。

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	*

S = シフト後のdstの最上位ビットの値がセットされます。
Z = シフト後のdstの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = シフト後、dstのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。
ただし、オペランド長が32ビットの場合は不定値がセットされます。
N = 0にリセットされます。
C = 最終のシフト前のdstの最上位ビットの値がセットされます。

実行例：HLレジスタが1234Hのとき、

```
SLA 4, HL
```

を実行すると、HLレジスタは2340H、キャリーフラグCYは1になります。

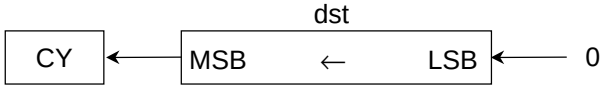
SLL num, dst

< Shift Left Logical **論理左シフト** >

動　作：{CY ← dst<MSB>, dst ← dstの左シフト値、dst<LSB> ← 0} Repeat num回

説　明：dstのMSBの内容がキャリーフラグCYへ転送され、dstの内容が左へシフトされ、0がdstのLSBへ転送されます。これがnum回繰り返されます。

説明図：



詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	SLL	#4, r	1 1 z z 1 r 1 1 1 0 1 1 1 0 0 0 0 0 # 4
○	○	○	SLL	A, r	1 1 z z 1 r 1 1 1 1 1 1 1 0
○	○	×	SLL<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 1 1 0

補　足：シフト回数numがAレジスタで指定された場合、Aレジスタの下位4ビットの値がシフト回数として使われます。0を指定すると、16回シフトされます。
dstがメモリの場合、シフト回数は1回に制限されます。

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	*

S = シフト後のdstの最上位ビットの値がセットされます。
Z = シフト後のdstの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = シフト後、dstのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。
ただし、オペランド長が32ビットの場合は不定値がセットされます。
N = 0にリセットされます。
C = 最終のシフト前のdstの最上位ビットの値がセットされます。

実行例：HLレジスタが1234Hのとき、

```
SLL 4, HL
```

を実行すると、HLレジスタは2340H、キャリーフラグCYは1になります。

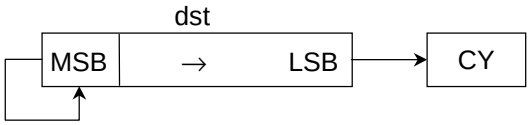
SRA num, dst

< Shift Right Arithmetic **算術右シフト** >

動　作：{CY ← dst<MSB>, dst ← dstの右シフト値、dst<MSB>は固定} Repeat num回

説　明：dstのLSBの内容がキャリーフラグCYへ転送され、dstの内容が右へシフト(MSBは固定)されます。これがnum回繰り返されます。

説明図：



詳　細：

バイト	サイズ ワード	ロング	ニモニック		コード
○	○	○	SRA	#4, r	1 1 z z 1 r 1 1 1 0 1 1 0 1 0 0 0 0 # 4
○	○	○	SRA	A, r	1 1 z z 1 r 1 1 1 1 1 1 0 1
○	○	×	SRA<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 1 0 1

補　足：シフト回数numがAレジスタで指定された場合、Aレジスタの下位4ビットの値がシフト回数として使われます。0を指定すると、16回シフトされます。
dstがメモリの場合、シフト回数は1回に制限されます。

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	*

S = シフト後のdstの最上位ビットの値がセットされます。
Z = シフト後のdstの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = シフト後、dstのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。
ただし、オペランド長が32ビットの場合は、不定値がセットされます。
N = 0にリセットされます。
C = 最終のシフト前のdstの最下位ビットの値がセットされます。

実行例：HLレジスタが8230Hのとき、

```
SRA 4, HL
```

を実行すると、HLレジスタはF823H、キャリーフラグCYは0になります。

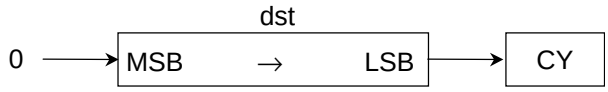
SRL num, dst

< Shift Right Logical **論理右シフト** >

動　作：{CY ← dst<LSB>, dst ← dstの右シフト値、dst<MSB> ← 0} Repeat num回

説　明：dstのLSBの内容がキャリーフラグCYへ転送され、dstの内容が右へシフトされ、0がdstのMSBへ転送されます。これがnum回繰り返されます。

説明図：



詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	SRL	#4, r	11zz1 r 11101111 0000 #4
○	○	○	SRL	A, r	11zz1 r 11111111
○	○	×	SRL<W>	(mem)	1m0zmmmm 01111111

補　足：シフト回数numがAレジスタで指定された場合、Aレジスタの下位4ビットの値がシフト回数として使われます。0を指定すると、16回シフトされます。
dstがメモリの場合、シフト回数は1回に制限されます。

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	*

S = シフト後のdst最上位ビットの値がセットされます。
Z = シフト後のdstの内容がゼロのときは1、それ以外のときは0がセットされます。
H = 0にリセットされます。
V = シフト後、dstのパリティ(1の数)が偶数のときは1、それ以外のときは0がセットされます。
ただし、オペランド長が32ビットの場合は不定値がセットされます。
N = 0にリセットされます。
C = 最終シフト前のdstの最下位ビットの値がセットされます。

実行例：HLレジスタが1238Hのとき、

```
SRL 4, HL
```

を実行すると、HLレジスタは0123H、キャリーフラグCYは1になります。

STCF num, dst

< Store Carry Flag **キャリーフラグからの1ビット転送** >

動　作：dst<num> ← CY

説　明：キャリーフラグCYの内容が、dstビットnumへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	STCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 1 0 0 0 0 0 0 #4
○	○	×	STCF	A, r	1 1 0 z 1 r 0 0 1 0 1 1 0 0
○	×	×	STCF	#3, (mem)	1 m 1 1 m m m m 1 0 1 0 0 #3
○	×	×	STCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 1 0 0

補　足：ビットnumがAレジスタで指定された場合、Aレジスタの下位4ビットの値がビットnumとして使われます。オペランドがバイトのとき、ビットnumの下位4ビットの値が8～15の場合、オペランドの値は変化しません。

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：100番地のメモリの内容が00Hで、キャリーフラグCYが1のとき、

STCF 5, (100H)

を実行すると、100番地のメモリの内容は00100000B(2進数)になります。

SUB dst, src

< Subtract **減算** >

動　作：dst ← dst − src

説　明：dstの内容からsrcの内容が減算され、dstへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	○	SUB	R, r	1 1 z z 1 r 1 0 1 0 0 R
○	○	○	SUB	r, #	1 1 z z 1 r 1 1 0 0 1 0 1 0 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	SUB	R, (mem)	1 m z z m m m m 1 0 1 0 0 R
○	○	○	SUB	(mem), R	1 m z z m m m m 1 0 1 0 1 R
○	○	×	SUB <w>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 0 1 0 #<7:0> #<15:8>

フラグ：

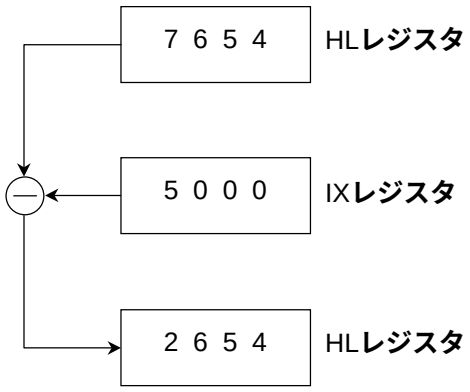
S	Z	H	V	N	C
*	*	*	*	1	*

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 演算結果、ビット3からビット4へボローが発生したときは1、それ以外のときは0がセットされます。ただし、オペランド長が32ビットの場合は、不定値がセットされます。
V = 演算結果、オーバフローが発生したときは1、それ以外のときは0がセットされます。
N = 1にセットされます。
C = 演算結果、最上位ビットからボローが発生したときは1、それ以外のときは0がセットされます。

実行例：HLレジスタが7654H,IXレジスタが5000Hのとき、

SUB HL, IX

を実行すると、HLレジスタは2654Hになります。



SWI num

< Software Interrupt **ソフトウエア割り込み** >

動　作：① XSP ← XSP – 6
② (XSP) ← SR
③ (XSP + 2) ← 32ビットPC
④ PC←(ベクタベース番地 + num × 4)
補足：ベクタベース番地は、派生品ごとに定義されます。

説　明：スタック領域へ、ステータスレジスタSRの内容と、このSWI命令の次の番地を示しているプログラムカウンタPCの内容が退避され、ベクタベース番地 + num × 4番地の内容で示される飛び先へジャンプします。

詳　細：

ニモニック	コード
SWI [#3]	1 1 1 1 1 #3

補　足：オペランドの値は、0～7まで指定できます。オペランドの記述を省略すると、SWI 7と解釈されます。

参　考：ステータスレジスタSRの構成は下記のとおりです。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SYSM	IFF2	IFF1	IFF0	MAX	RFP2	RFP1	RFP0	S	Z	0	H	0	V	N	C

フラグ：

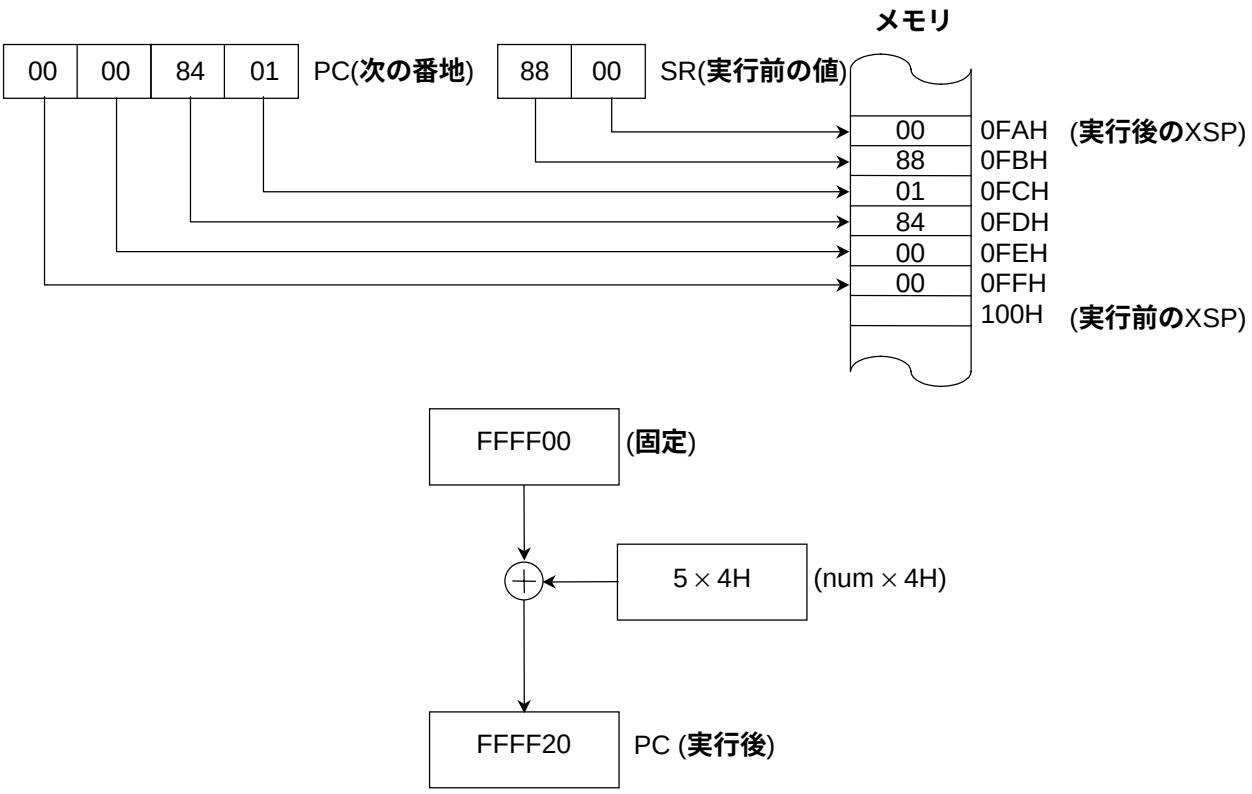
S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：スタックポインタXSPが100H、ステータスレジスタSRが8800Hで、8400H番地のメモリにある、

SWI 5

を実行すると、00FAH番地のメモリに旧ステータスレジスタSRの内容8800Hが、00FCH番地のメモリにプログラムカウンタPCの内容00008401Hがライトされ、FFFF20H番地の内容で示されるメモリ番地へジャンプします。



TSET num, dst

< Test and Set 1**ビットのテストとセット** >

動　作：Zフラグ ← dst<num>の反転値
dst <num> ← 1

説　明：dstのビットnumの反転値が、Zフラグへ転送されます。その後、dstのビットnumが1にセットされます。

詳　細：

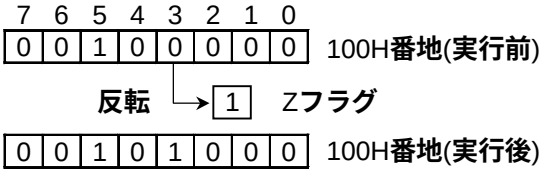
サイズ			**ニモニック**		**コード**
バイト	**ワード**	**ロング**			
○	○	×	TSET	#4, r	1 1 z z 1 r 0 0 1 1 0 1 0 0 0 0 0 0 # 4
○	×	×	TSET	#3, (mem)	1 m 1 1 m m m m 1 0 1 0 1 #3

フラグ：

S	Z	H	V	N	C
×	*	1	×	0	–

S = 不定値がセットされます。
Z = src<num>の反転値がセットされます。
H = 1にセットされます。
V = 不定値がセットされます。
N = 0にセットされます。
C = 変化なし。

実行例：100番地のメモリの内容が00100000B(2進数)のとき、
TSET 3, (100H)
を実行すると、Zフラグは1になり、100H番地のメモリの内容は00101000B(2進数)になります。



UNLK dst

< Unlink **スタックフレームの削除** >

動　作：XSP ← dst, dst ← (XSP+)

説　明：dstの内容が、スタックポインタXSPへ転送されます。その後、スタック領域からロングワードデータがdstへPOPされます。この命令は、LINK命令とペアで使用します。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック	コード
×	×	○	UNLK r	1 1 1 0 1 r 0 0 0 0 1 1 0 1

フラグ：

S	Z	H	V	N	C
–	–	–	–	–	–

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = 変化なし。

実行例：LINK命令実行後、

```
UNLK XIZ
```

を実行すると、スタックポインタXSPとXIZレジスタは、LINK命令実行前と同じ値になります(詳しくは、LINK命令のページを参照してください)。

XOR dst, src

< Exclusive or **排他的論理和** >

動　作：dst ← dst XOR src

説　明：dstの内容とsrcの内容が排他的論理和演算され、dstへ転送されます。

(真理値表)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

詳　細：

サイズ			ニモニック		コード
バイト	ワード	ロング			
○	○	○	XOR	R, r	1 1 z z 1 r 1 1 0 1 0 R
○	○	○	XOR	r, #	1 1 z z 1 r 1 1 0 0 1 1 0 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	XOR	R, (mem)	1 m z z m m m m 1 1 0 1 0 R
○	○	○	XOR	(mem), R	1 m z z m m m m 1 1 0 1 1 R
○	○	×	XOR<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 1 0 1 #<7:0> #<15:8>

フラグ：

S	Z	H	V	N	C
*	*	0	*	0	0

S = 演算結果の最上位ビットの値がセットされます。
Z = 演算結果がゼロのときは1、それ以外のときは0がセットされます。
H = 0にセットされます。
V = 演算結果のパリティ(1の数)が偶数のときは1、奇数のときは0がセットされます。ただし、オペランド長が32ビットの場合は、不定値がセットされます。
N = 0にクリアされます。
C = 0にクリアされます。

実行例：HLレジスタが7350H,IXレジスタが3456Hのとき、

XOR HL, IX

を実行すると、HLレジスタは4706Hになります。

	0111	0011	0101	0000	← HLレジスタ (実行前)
XOR)	0011	0100	0101	0110	← IXレジスタ (実行前)
	0100	0111	0000	0110	← HLレジスタ (実行後)

XORCF num, src

< Exclusive Or Carry Flag **キャリーフラグとの1ビット排他的論理和** >

動　作：CY ← CY XOR src<num>

説　明：キャリーフラグCYの内容とsrcのビットnumの内容が排他的論理和演算され、キャリーフラグCYへ転送されます。

詳　細：

サイズ バイト	サイズ ワード	サイズ ロング	ニモニック		コード
○	○	×	XORCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 0 1 0 0 0 0 0 # 4
○	○	×	XORCF	A, r	1 1 0 z 1 r 0 0 1 0 1 0 1 0
○	×	×	XORCF	#3, (mem)	1 m 1 1 m m m m 1 0 0 1 0 #3
○	×	×	XORCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 0 1 0

補　足：ビットnumがAレジスタで指定された場合、Aレジスタの下位4ビットの値がビットnumとして使われます。オペランドがバイトのとき、ビットnumの下位4ビットの値が8～15の場合、演算結果は不定になります。

フラグ：

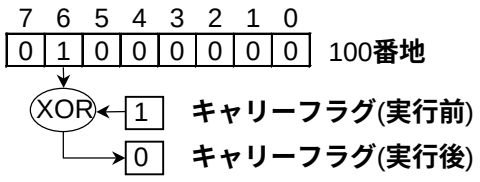
S	Z	H	V	N	C
–	–	–	–	–	*

S = 変化なし。
Z = 変化なし。
H = 変化なし。
V = 変化なし。
N = 変化なし。
C = キャリーフラグCYの内容とsrcのビットnumの内容の排他的論理和演算された値がセットされます。

実行例：100番地のメモリの内容が01000000B(2進数)で、キャリーフラグCYが1のとき、

XORCF 6, (100H)

を実行すると、キャリーフラグは0になります。



ZCF

< Zero flag to Carry Flag Zフラグをキャリーフラグへ転送 >

動　作：CY ← Zフラグの反転値

説　明：Zフラグの反転値が、キャリーフラグCYへ転送されます。

詳　細：

ニモニック	コード
ZCF	0 0 0 1 0 0 1 1

フラグ：

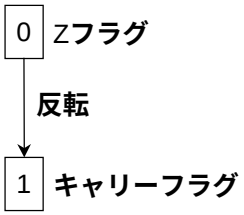
S	Z	H	V	N	C
–	–	X	–	0	*

S = 変化なし。
Z = 変化なし。
H = 不定値がセットされます。
V = 変化なし。
N = 0にリセットされます。
C = Zフラグの反転値がセットされます。

実行例：Zフラグが0のとき、

ZCF

を実行すると、キャリーフラグCYは1になります。



付録 B. 命令一覧表

■ 本文中の記号の説明

1. サイズ欄

B	オペランドサイズがバイト (8 ビット)
W	オペランドサイズがワード (16 ビット)
L	オペランドサイズがロング (32 ビット)

2. ニモニック欄

R	8/16/32 ビットのカレントバンクレジスタを含む 8 本の汎用レジスタ 8 ビットの場合: W, A, B, C, D, E, H, L 16 ビットの場合: WA, BC, DE, HL, IX, IY, IZ, SP 32 ビットの場合: XWA, XBC, XDE, XHL, XIX, XIY, XIZ, XSP
r	8/16/32 ビットの汎用レジスタ
cr	8/16/32 ビットの CPU の全コントロールレジスタ DMAS0〜3, DMAD0〜3, DMAC0〜3, DMAM0〜3, INTNEST
A	A レジスタ(8 ビット)
F	フラグレジスタ(8 ビット)
F'	裏フラグレジスタ(8 ビット)
SR	ステータスレジスタ(16 ビット)
PC	プログラムカウンタ(32 ビット)
(mem)	8/16/32 ビットのメモリ内のデータ
mem	実効アドレス値
<W>	オペランドサイズがワードのとき W の記述が必要であることを示します。
[ ]	カッコ内のオペランドの記述が、省略できることを示します。
#	8/16/32 ビットの即値データ
#3	3 ビットの即値データ; 0〜7 or 1〜8 ・・・・・・・・・・・・・・・・ 短縮コード用
#4	4 ビットの即値データ; 0〜15 or 1〜16
d8	8 ビットのディスプレースメント ;–80H〜+7FH
d16	16 ビットのディスプレースメント ;–8000H〜+7FFFH
cc	コンディションコード
(#8)	ダイレクトアドレッシング ; (00H)〜(0FFH) ・・・・・・ 256 バイト空間
(#16)	64KB エリアアドレッシング ; (0000H)〜(0FFFFH)
$	その命令が置かれている先頭アドレス

3. **コード欄**

Z	**オペランドサイズを表す指定コードです。** **バイト** (8 **ビット**) = 0 **ワード** (16 **ビット**) = 2 **ロング** (32 **ビット**) = 4
ZZ	**オペランドサイズを表す指定コードです。** **バイト** (8 **ビット**) = 00H **ワード** (16 **ビット**) = 10H **ロング** (32 **ビット**) = 20H

4. **フラグ欄** (SZHVNC)

–	**フラグは変化しません。**
*	**命令の実行によって、フラグは変化します。**
0	**フラグは** 0 **にクリアされます。**
1	**フラグは** 1 **にセットされます。**
P	**命令の実行によって、フラグは変化します。(パリティフラグとして働きます)**
V	**命令の実行によって、フラグは変化します。(オーバフローフラグとして働きます)**
X	**フラグには、不定値がセットされます。**

5. **命令長欄**

命令長を、バイト単位で示しています。

+#	**イミディエートデータ長を加えてください。**
+M	**アドレッシングコード長を加えてください。**
+#M	**イミディエートデータ長+アドレッシングコード長を加えてください。**

6. **ステート欄**

命令の実行処理時間を、ステート単位で、8ビット, 16ビット, 32ビット処理の場合を順に示しています。

1**ステート**=2×1/f_{FPH}

■ TLCS-900/L1**命令一覧表** (1/10)

(1) **転送**

命令群	サイズ	ニモニック		コード(16 進)		機能	SZHVNC	命令長	ステート
LD	BWL	LD	R, r	C8+zz+r	:88+R	R ← r	- - - - - -	2	2. 2. 2
	BWL	LD	r, R	C8+zz+r	:98+R	r ← R	- - - - - -	2	2. 2. 2
	BWL	LD	r, #3	C8+zz+r	:A8+#3	r ← #3	- - - - - -	2	2. 2. 2
	BWL	LD	R, #	20+zz+R	:#	R ← #	- - - - - -	1+#	2. 3. 5
	BWL	LD	r, #	C8+zz+r	:03:#	r ← #	- - - - - -	2+#	3. 4. 6
	BWL	LD	R, (mem)	80+zz+mem	:20+R	R ← (mem)	- - - - - -	2+M	4. 4. 6
	BWL	LD	(mem), R	B0+mem	:40+zz+R	(mem) ← R	- - - - - -	2+M	4. 4. 6
	BW-	LD<W>	(#8), #	08+z	:#8:#	(#8) ← #	- - - - - -	2+#	5. 6. -
	BW-	LD<W>	(mem), #	B0+mem	:00+z:#	(mem) ← #	- - - - - -	2+M#	5. 6. -
	BW-	LD<W>	(#16), (mem)	80+zz+mem	:19:#16	(#16) ← (mem)	- - - - - -	4+M	8. 8. -
	BW-	LD<W>	(mem), (#16)	B0+mem	:14+z:#16	(mem) ← (#16)	- - - - - -	4+M	8. 8. -
PUSH	B--	PUSH	F	18		(-XSP) ← F	- - - - - -	1	3. –. –
	B--	PUSH	A	14		(-XSP) ← A	- - - - - -	1	3. –. –
	-WL	PUSH	R	18+zz+R		(-XSP) ← R	- - - - - -	1	–. 3. 5
	BWL	PUSH	r	C8+zz+r	:04	(-XSP) ← r	- - - - - -	2	4. 4. 7
	BW-	PUSH<W>	#	09+z	:#	(-XSP) ← #	- - - - - -	1+#	4. 5. –
	BW-	PUSH<W>	(mem)	80+zz+mem	:04	(-XSP) ← (mem)	- - - - - -	2+M	6. 6. –
POP	B--	POP	F	19		F ← (XSP+)	* * * * * *	1	4. –. –
	B--	POP	A	15		A ← (XSP+)	- - - - - -	1	4. –. –
	-WL	POP	R	38+zz+R		R ← (XSP+)	- - - - - -	1	–. 4. 6
	BWL	POP	r	C8+zz+r	:05	r ← (XSP+)	- - - - - -	2	5. 5. 7
	BW-	POP<W>	(mem)	B0+mem	:04+z	(mem)←(XSP+)	- - - - - -	2+M	7. 7. –
LDA	-WL	LDA	R,mem	B0+mem	:10+zz+R	R ← mem	- - - - - -	2+M	–. 4. 4
LDAR	-WL	LDAR	R, $+4+d16	F3:13:d16:20+zz+R		R ← PC+d16	- - - - - -	5	–. 7. 7

(2) **交換**

命令群	サイズ	ニモニック		コード(16 進)		機能	SZHVNC	命令長	ステート
EX	B--	EX	F, F′	16		F ↔ F′	* * * * * *	1	2. –. –
	BW-	EX	R, r	C8+zz+r	:B8+R	R ↔ r	- - - - - -	2	3. 3. –
	BW-	EX	(mem), R	80+zz+mem	:30+R	(mem) ↔ R	- - - - - -	2+M	6. 6. –
MIRR	-W-	MIRP	r	D8+r	:16	r<0:MSB> ← r<MSB:0>	- - - - - -	2	–. 3. –

■ TLCS-900/L1**命令一覧表** (2/10)

(3) **ブロック転送/ブロックサーチ**

命令群	サイズ	ニモニック	コード(16 進)	機能	SZHVNC	命令長	ステート
LDxx	BW–	LDI<W> [(XDE+), (XHL+)]	83+zz :10	(XDE+) ← (XHL+) BC ← BC-1	- - 0①0 -	2	8. 8. –
	BW–	LDI<W> (XIX+), (XIY+)	85+zz :10	(XIX+) ← (XIY+) BC ← BC-1	- - 0①0 -	2	8.8. –
	BW–	LDIR<W> [(XDE+), (XHL+)]	83+zz :11	repeat (XDE+) ←(XHL+) BC ← BC-1 until BC=0	- - 000 -	2	7n+1
	BW–	LDIR<W> (XIX+), (XIY+)	85+zz :11	repeat (XIX+)←(XIY+) BC ← BC-1 until BC=0	- - 000 -	2	7n+1
	BW-	LDD<W> [(XDE-), (XHL-)]	83+zz :12	(XDE-) ← (XHL-) BC ← BC-1	- - 0①0 -	2	8.8. –
	BW-	LDD<W> (XIX-), (XIY-)	85+zz :12	(XIX-) ← (XIY-) BC ← BC-1	- - 0①0 -	2	8.8. –
	BW-	LDDR<W> [(XDE-), (XHL-)]	83+zz :13	repeat (XDE-) ←(XHL-) BC ← BC-1 until BC=0	- - 000 -	2	7n+1
	BW-	LDDR<W> (XIX-), (XIY-)	85+zz :13	repeat (XIX-) ←(XIY-) BC ← BC-1 until BC=0	- - 000 -	2	7n+1
CPxx	BW-	CPI [A/WA, (R+)]	80+zz+R :14	A/WA - (R+) BC ← BC-1	*② ①1-	2	6. 6. –
	BW-	CPIR [A/WA, (R+)]	80+zz+R :15	repeat A/WA - (R+) BC ← BC-1 until A/WA=(R) or BC=0	*②*①1-	2	6n+1
	BW-	CPD [A/WA, (R-)]	80+zz+R :16	A/WA - (R-) BC ← BC-1	*②*①1-	2	6. 6. –
	BW-	CPDR [A/WA, (R-)]	80+zz+R :17	repeat A/WA - (R-) BC ← BC-1 until A/WA=(R) or BC=0	*②*①1-	2	6n+1

(注1) ①： 命令実行後BC = 0ならP/Vフラグは0に、それ以外は1にセットされます。
②： A/WA = (R)ならZフラグは1に、それ以外は0にセットされます。

(注2) CPI, CPIR, CPD, CPDR命令でオペランドを省略したときは、A,(XHL+/–)を指定したことになります。

■ TLCS-900/L1**命令一覧表** (3/10)

(4) **算術演算**

命令群	サイズ	ニモニック		コード(16 進)		機能	SZHVNC	命令長	ステート
	BWL	ADD	R, r	C8+zz+r	:80+R	R ← R+r	* * * V0 *	2	2. 2. 2
	BWL	ADD	r, #	C8+zz+r	:C8:#	r ← r+#	* * * V0 *	2+#	3. 4. 6
ADD	BWL	ADD	R, (mem)	80+zz+mem	:80+R	R ← R+(mem)	* * * V0 *	2+M	4. 4. 6
	BWL	ADD	(mem), R	80+zz+mem	:88+R	(mem) ← (mem)+R	* * * V0 *	2+M	6. 6. 10
	BW-	ADD<W>	(mem), #	80+zz+mem	:38:#	(mem) ← (mem)+#	* * * V0 *	2+M#	7. 8. –
	BWL	ADC	R, r	C8+zz+r	:90+R	R ← R+r+CY	* * * V0 *	2	2. 2. 2
	BWL	ADC	r, #	C8+zz+r	:C9:#	r ← r+#+CY	* * * V0 *	2+#	3. 4. 6
ADC	BWL	ADC	R, (mem)	80+zz+mem	:90+R	R ← R+(mem)+CY	* * * V0 *	2+M	4. 4. 6
	BWL	ADC	(mem), R	80+zz+mem	:98+R	(mem) ← (mem)+R+CY	* * * V0 *	2+M	6. 8. 10
	BW-	ADC<W>	(mem), #	80+zz+mem	:39:#	(mem) ← (mem)+#+CY	* * * V0 *	2+M#	7. 8. –
	BWL	SUB	R, r	C8+zz+r	:A0+R	R ← R – r	* * * V1 *	2	2. 2. 2
	BWL	SUB	r, #	C8+zz+r	:CA:#	r ← r – #	* * * V1 *	2+#	3. 4. 6
SUB	BWL	SUB	R, (mem)	80+zz+mem	:A0+R	R ← R – (mem)	* * * V1 *	2+M	4. 4. 6
	BWL	SUB	(mem), R	80+zz+mem	:A8+R	(mem) ← (mem) – R	* * * V1 *	2+M	6. 6. 10
	BW-	SUB<W>	(mem), #	80+zz+mem	:3A:#	(mem) ← (mem) – #	* * * V1 *	2+M#	7. 8. –
	BWL	SBC	R, r	C8+zz+r	:B0+R	R ← R – r – CY	* * * V1 *	2	2. 2. 2
	BWL	SBC	r, #	C8+zz+r	:CB:#	r ← r – # – CY	* * * V1 *	2+#	3. 4. 6
SBC	BWL	SBC	R, (mem)	80+zz+mem	:B0+R	R ← R–(mem)–CY	* * * V1 *	2+M	4. 4. 6
	BWL	SBC	(mem), R	80+zz+mem	:B8+R	(mem) ← (mem)-R–CY	* * * V1 *	2+M	6. 6. 10
	BW-	SBC<W>	(mem), #	80+zz+mem	:3B:#	(mem) ←(mem)–# –CY	* * * V1 *	2+M#	7. 8. –
	BWL	CP	R, r	C8+zz+r	:F0+R	R – r	* * * V1 *	2	2. 2. 2
	BW-	CP	r, #3	C8+zz+r	:D8:#3	r – #3	* * * V1 *	2	2. 2. –
CP	BWL	CP	r, #	C8+zz+r	:CF:#	r – #	* * * V1 *	2+#	3. 4. 6
	BWL	CP	R, (mem)	80+zz+mem	:F0+R	R – (mem)	* * * V1 *	2+M	4. 4. 6
	BWL	CP	(mem), R	80+zz+mem	:F8+R	(mem) – R	* * * V1 *	2+M	4. 4. 6
	BW-	CP<W>	(mem), #	80+zz+mem	:3F:#	(mem) – #	* * * V1 *	2+M#	5. 6. –
	B--	INC	#3, r	C8+r	:60+#3	r ← r+#3	* * * V0 -	2	2. –. –
INC	-WL	INC	#3, r	C8+zz+r	:60+#3	r ← r+#3	- - - - - -	2	–. 2. 2
	BW-	INC<W>	#3, (mem)	80+zz+mem	:60+#3	(mem) ← (mem)+#3	* * * V0 -	2+M	6. 6. –
	B--	DEC	#3, r	C8+r	:68+#3	r ← r – #3	* * * V1 -	2	2. –. –-
DEC	-WL	DEC	#3, r	C8+zz+r	:68+#3	r ← r – #3	- - - - - -	2	–. 2. 2
	BW-	DEC<W>	#3, (mem)	80+zz+mem	:68+#3	(mem)←(mem) – #3	* * * V1 -	2+M	6. 6. –
NEG	BW-	NEG	r	C8+zz+r	:07	r ← 0 – r	* * * V1 *	2	2. 2. –
EXTZ	-WL	EXTZ	r	C8+zz+r	:12	r<high> ← 0	- - - - - -	2	–. 3. 3
EXTS	-WL	EXTS	r	C8+zz+r	:13	r<high> ← r<low.MSB>	- - - - - -	2	–. 3. 3
DAA	B - -	DAA	r	C8+r	:10	**加減算後の10進補正**	* * * P - *	2	4. –. –
PAA	-WL	PAA	r	C8+zz+r	:14	if r<0>=1 then INC r	- - - - - -	2	–. 4. 4

(注1) INC/DEC命令では、#3のコード値が0のときは、+8/–8として機能します。
(注2) TLCS-90のADD R,r命令(ワード型)は、S/Z/Vフラグが変化しません。

■ TLCS-900/L1**命令一覧表** (4/10)

命令群	サイズ	ニモニック	コード(16 進)	機能	SZHVNC	命令長	ステート
MUL	BW-	MUL RR, r	C8+zz+r :40+R	RR ← R×r	- - - - - -	2	11.14. –
	BW-	MUL rr, #	C8+zz+r :08:#	rr ← r×#	- - - - - -	2+#	12.15. –
	BW-	MUL RR, (mem)	80+zz+mem :40+R	RR ← R×(mem)	- - - - - -	2+M	13.16. –
MULS	BW-	MULS RR, r	C8+zz+r :48+R	RR ← R×r ;signed	- - - - - -	2	9.12. –
	BW-	MULS rr, #	C8+zz+r :09:#	rr ← r×# ;signed	- - - - - -	2+#	10.13. –
	BW-	MULS RR, (mem)	80+zz+mem :48+R	RR ← R×(mem) ;signed	- - - - - -	2+M	11.14. –
DIV	BW-	DIV RR, r	C8+zz+r :50+R	R ← RR÷r	- - - V - -	2	15.23. –
	BW-	DIV rr, #	C8+zz+r :0A:#	r ← rr÷#	- - - V - -	2+#	15.23. –
	BW-	DIV RR, (mem)	80+zz+mem :50+R	R ← RR÷(mem)	- - - V - -	2+M	16.24. –
DIVS	BW-	DIVS RR, r	C8+zz+r :58+R	R ← RR÷r ;signed	- - - V - -	2	18.26. –
	BW-	DIVS rr, #	C8+zz+r :0B:#	r ← rr÷# ;signed	- - - V - -	2+#	18.26. –
	BW-	DIVS RR, (mem)	80+zz+mem :58+R	R ← RR÷(mem) ;signed	- - - V - -	2+M	19.27. –
MULA	-W-	MULA rr	D8+r :19	**符号付き積和演算** rr ← rr+(XDE)×(XHL) 32bit 32bit 16bit 16bit XHL ← XHL – 2	* * - V - -	2	–. 19. –
MINC	-W-	MINC1 #, r (#=2**n) (1<=n<=15)	D8+r :38:# – 1	**モジュロインクリメント** ; +1 if (r mod #)=(# – 1) then r←r –(# – 1) else r←r+1	- - - - - -	4	–. 5. –
	-W-	MINC2 #, r (#=2**n) (2<=n<=15)	D8+r :39:# – 2	**モジュロインクリメント** ; +2 if (r mod #)=(# – 2) then r←r –(# – 2) else r←r+2	- - - - - -	4	–. 5. –
	-W-	MINC4 #, r (#=2**n) (3<=n<=15)	D8+r :3A:# –4	**モジュロインクリメント** ; +4 if (r mod #)=(# – 4) then r←r –(# – 4) else r←r+4		4	–. 5. -
MDEC	-W-	MDEC1 #, r (#=2**n) (1<=n<=15)	D8+r :3C:# –1	**モジュロデクリメント** ; –1 if (r mod #)=0 then r←r+(# – 1) else r←r – 1	- - - - - -	4	–. 4. –
	-W-	MDEC2 #, r (#=2**n) (2<=n<=15)	D8+r :3D:# –2	**モジュロデクリメント** ; –2 if (r mod #)=0 then r←r+(# – 2) else r←r – 2	- - - - - -	4	–. 4. –
	-W-	MDEC4 #, r (#=2**n) (3<=n<=15)	D8+r :3E:# –4	**モジュロデクリメント** ; –4 if (r mod #)=0 then r←r+(# – 4) else r←r – 4	- - - - - -	4	–. 4. –

(注) MUL, MULS, DIV, DIVS命令のオペランドRRは、演算サイズの倍のレジスタを指定することを示しています。演算サイズがバイトのとき(8×8ビット, 16÷8ビット)は、ワードレジスタ(16ビット)指定し、演算サイズがワードのとき(16×16ビット, 32÷16ビット)は、ロングワードレジスタ(32ビット)を指定します。

■ TLCS-900/L1**命令一覧表** (5/10)

(5) **論理演算**

命令群	**サイズ**	**ニモニック**		**コード**(16 **進**)		**機能**	SZHVNC	**命令長**	**ステート**
	BWL	AND	R, r	C8+zz+r	:C0+R	R ← R and r	* * 1P00	2	2. 2. 2
	BWL	AND	r, #	C8+zz+r	:CC:#	r ← r and #	* * 1P00	2+#	3. 4. 6
AND	BWL	AND	R, (mem)	80+zz+mem	:C0+R	R ← R and (mem)	* * 1P00	2+M	4. 4. 6
	BWL	AND	(mem),R	80+zz+mem	:C8+R	(mem) ← (mem) and R	* * 1P00	2+M	6. 6. 10
	BW-	AND<W>	(mem), #	80+zz+mem	:3C:#	(mem) ← (mem) and #	* * 1P00	2+M#	7. 8. –
	BWL	OR	R, r	C8+zz+r	:E0+R	R ← R or r	* * 0P00	2	2. 2. 2
	BWL	OR	r, #	C8+zz+r	:CE:#	r ← r or #	* * 0P00	2+#	3. 4. 6
OR	BWL	OR	R, (mem)	80+zz+mem	:E0+R	R ← R or (mem)	* * 0P00	2+M	4. 4. 6
	BWL	OR	(mem),R	80+zz+mem	:E8+R	(mem) ← (mem) or R	* * 0P00	2+M	6. 6. 10
	BW-	OR <W>	(mem), #	80+zz+mem	:3E:#	(mem) ← (mem) or #	* * 0P00	2+M#	7. 8. –
	BWL	XOR	R, r	C8+zz+r	:D0+R	R ← R xor r	* * 0P00	2	2. 2. 2
	BWL	XOR	r, #	C8+zz+r	:CD:#	r ← r xor #	* * 0P00	2+#	3. 4. 6
XOR	BWL	XOR	R, (mem)	80+zz+mem	:D0+R	R ← R xor (mem)	* * 0P00	2+M	4. 4. 6
	BWL	XOR	(mem),R	80+zz+mem	:D8+R	(mem) ← (mem) xor R	* * 0P00	2+M	6. 6. 10
	BW-	XOR <W>	(mem), #	80+zz+mem	:3D:#	(mem) ← (mem) xor #	* * 0P00	2+M#	7. 8. –
CPL	BW-	CPL	r	C8+zz+r	:06	r ← not r	- - 1 - 1 -	2	2. 2. –

■ TLCS-900/L1**命令一覧表** (6/10)

(6) **ビット操作**

命令群	サイズ	ニモニック		コード(16 進)		機能	SZHVNC	命令長	ステート
LDCF	BW-	LDCF	#4, r	C8+zz+r	:23:#4	CY ← r<#4>	- - - - - *	3	3. 3. –
	BW-	LDCF	A, r	C8+zz+r	:2B	CY ← r<A>	- - - - - *	2	3. 3. –
	B--	LDCF	#3, (mem)	B0+mem	:98+#3	CY ← (mem)<#3>	- - - - - *	2+M	6. –. –
	B--	LDCF	A, (mem)	B0+mem	:2B	CY ← (mem)<A>	- - - - - *	2+M	6. –. –
STCF	BW-	STCF	#4, r	C8+zz+r	:24:#4	r<#4> ← CY	- - - - - -	3	3. 3. –
	BW-	STCF	A, r	C8+zz+r	:2C	r<A> ← CY	- - - - - -	2	3. 3. –
	B--	STCF	#3, (mem)	B0+mem	:A0+#3	(mem)<#3> ← CY	- - - - - -	2+M	7. –. –
	B--	STCF	A, (mem)	B0+mem	:2C	(mem)<A> ← CY	- - - - - -	2+M	7. –. –
ANDCF	BW-	ANDCF	#4, r	C8+zz+r	:20:#4	CY ← CY and r<#4>	- - - - - *	3	3. 3. –
	BW-	ANDCF	A, r	C8+zz+r	:28	CY ← CY and r<A>	- - - - - *	2	3. 3. –
	B--	ANDCF	#3, (mem)	B0+mem	:80+#3	CY ← CY and (mem)<#3>	- - - - - *	2+M	6. –. –
	B--	ANDCF	A, (mem)	B0+mem	:28	CY ← CY and (mem)<A>	- - - - - *	2+M	6. –. –
ORCF	BW-	ORCF	#4, r	C8+zz+r	:21:#4	CY ← CY or r<#4>	- - - - - *	3	3. 3. –
	BW-	ORCF	A, r	C8+zz+r	:29	CY ← CY or r<A>	- - - - - *	2	3. 3. –
	B--	ORCF	#3, (mem)	B0+mem	:88+#3	CY ← CY or (mem)<#3>	- - - - - *	2+M	6. –. –
	B--	ORCF	A, (mem)	B0+mem	:29	CY ← CY or (mem)<A>	- - - - - *	2+M	6. –. –
XORCF	BW-	XORCF	#4, r	C8+zz+r	:22:#4	CY ← CY xor r<#4>	- - - - - *	3	3. 3. –
	BW-	XORCF	A, r	C8+zz+r	:2A	CY ← CY xor r<A>	- - - - - *	2	3. 3. –
	B - -	XORCF	#3, (mem)	B0+mem	:90+#3	CY ← CY xor (mem)<#3>	- - - - - *	2+M	6. –. –
	B - -	XORCF	A, (mem)	B0+mem	:2A	CY ← CY xor (mem)<A>	- - - - - *	2+M	6. –. –
RCF	- - -	RCF		10		CY ← 0	- - 0 - 00	1	2
SCF	- - -	SCF		11		CY ← 1	- - 0 - 01	1	2
CCF	- - -	CCF		12		CY ← not CY	- - X - 0*	1	2
ZCF	- - -	ZCF		13		CY ← not Z**フラグ**	- - X - 0*	1	2
BIT	BW-	BIT	#4, r	C8+zz+r	:33:#4	Z ←not r<#4>	X * 1X0 -	3	3. 3. –
	B - -	BIT	#3, (mem)	B0+mem	:C8+#3	Z ←not (mem)<#3>	X * 1X0 -	2+M	6. –. –
RES	BW-	RES	#4, r	C8+zz+r	:30:#4	r<#4> ← 0	- - - - - -	3	3. 3. –
	B - -	RES	#3, (mem)	B0+mem	:B0+#3	(mem)<#3> ← 0	- - - - - -	2+M	7. –. –
SET	BW-	SET	#4, r	C8+zz+r	:31:#4	r<#4> ← 0	- - - - - -	3	3. 3. –
	B - -	SET	#3, (mem)	B0+mem	:B8+#3	(mem)<#3> ← 0	- - - - - -	2+M	7. –. –
CHG	BW-	CHG	#4, r	C8+zz+r	:32:#4	r<#4> ← not r<#4>	- - - - - -	3	3. 3. –
	B - -	CHG	#3, (mem)	B0+mem	:C8+#3	(mem)<#3> ← not (mem)<#3>	- - - - - -	2+M	7. –. –
TSET	BW-	TSET	#4, r	C8+zz+r	:34:#4	Z ←not r<#4> : r<#4>←1	X * 1X0 -	3	4. 4. –
	B - -	TSET	#3, (mem)	B0+mem	:A8+#3	Z ←not (mem)<#3> (mem)<#3> ←1	X * 1X0 -	2+M	7. – –
BS1	-W-	BS1F	A, r	D8+r	:0E	A ← 1**サーチ** r ;Forward	- - -①- -	2	–. 3. –
	-W-	BS1F	A, r	D8+r	:0F	A ← 1**サーチ** r ;Backward	- - -①- -	2	–. 3. –

(注1) ①： サーチするビットが見つかったとき0にセットされ、それ以外のときは1にセットされAレジスタは不定値になります。

■ TLCS-900/L1**命令一覧表** (7/10)

(7) **特別演算、**CPU**制御**

命令群	**サイズ**	**ニモニック**		**コード**(16 **進**)		**機能**	SZHVNC	**命令長**	**ステート**
NOP	- - -	NOP		00		no operation	- - - - - -	1	2
EI	- - -	EI	[#3]	06	:#3	**割り込み許可フラグの設定** IFF←#3	- - - - - -	2	3
DI	- - -	DI		06	:07	**割り込み禁止** IFF←7	- - - - - -	2	4
PUSH	-W-	PUSH	SR	02		(– XSP) ← SR	- - - - - -	1	–. 3. –
POP	-W-	POP	SR	03		SR ← (XSP+)	* * * * * *	1	–. 4. –
SWI	- - -	SWI	[#3]	F8+#3		**ソフトウェア割り込み** PUSH PC&SR JP (FFFF00H+4×#3)	- - - - - -	1	19
HALT	- - -	HALT		05		CPU**停止**	- - - - - -	1	6
LDC	BWL	LDC	cr,r	C8+zz+r	:2E:cr	cr ← r	- - - - - -	3	3. 3. 3
	BWL	LDC	r, cr	C8+zz+r	:2F:cr	r ← cr	- - - - - -	3	3. 3. 3
LDX	B - -	LDX	(#8), #	F7: 00: #8: 00: #: 00		(#8) ← #	- - - - - -	6	8. –. –

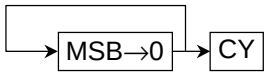
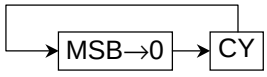
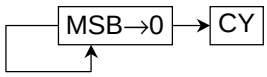
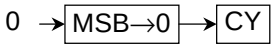
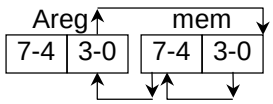
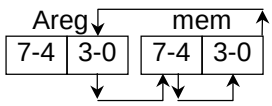
LINK	- - L	LINK	r, d16	E8+r	:0C:d16	PUSH r
						LD r, XSP
						ADD XSP, d16
							- - - - - -	4	–. –. 8

UNLK	- - L	UNLK	r	E8+r	:0D	LD XSP, r POP r	- - - - - -	2	–. –. 7
LDF	- - -	LDF	#3	17	:#3	**レジスタバンクの設定** RFP ←#3(**リセット時**0)	- - - - - -	2	2
INCF	- - -	INCF		0C		**レジスタバンクの切り替え** REP ←RFP+1	- - - - - -	1	2
DECF	- - -	DECF		0D		**レジスタバンクの切り替え** REP ←RFP – 1	- - - - - -	1	2
SCC	BW-	SCC	cc, r	C8+zz+r	:70+cc	if cc then r ←1 else r ←0	- - - - - -	2	2. 2. –

(注1) EI命令でオペランド#3の記述を省略すると0を指定したものとみなされます。
(注2) SWI命令でオペランド#3の記述を省略すると7を指定したものとみなされます。

■ TLCS-900/L1**命令一覧表** (8/10)

(8) **ローテート、シフト**

命令群	サイズ	ニモニック		コード(16 進)		機能	SZHVNC	命令長	ステート
RLC	BWL	RLC	#4, r	C8+zz+r	:E8:#4		* * 0P0 *	3	3+n/4
	BWL	RLC	A, r	C8+zz+r	:F8		* * 0P0 *	2	3+n/4
	BW-	RLC<W>	(mem)	80+zz+mem	:78		* * 0P0 *	2+M	6.6
RRC	BWL	RRC	#4, r	C8+zz+r	:E9:#4		* * 0P0 *	3	3+n/4
	BWL	RRC	A, r	C8+zz+r	:F9		* * 0P0 *	2	3+n/4
	BW-	RRC<W>	(mem)	80+zz+mem	:79		* * 0P0 *	2+M	6.6
RL	BWL	RL	#4, r	C8+zz+r	:EA:#4		* * 0P0 *	3	3+n/4
	BWL	RL	A, r	C8+zz+r	:FA		* * 0P0 *	2	3+n/4
	BW-	RL<W>	(mem)	80+zz+mem	:7A		* * 0P0 *	2+M	6.6
RR	BWL	RR	#4, r	C8+zz+r	:EB:#4		* * 0P0 *	3	3+n/4
	BWL	RR	A, r	C8+zz+r	:FB		* * 0P0 *	2	3+n/4
	BW-	RR<W>	(mem)	80+zz+mem	:7B		* * 0P0 *	2+M	6.6
SLA	BWL	SLA	#4, r	C8+zz+r	:EC:#4		* * 0P0 *	3	3+n/4
	BWL	SLA	A, r	C8+zz+r	:FC		* * 0P0 *	2	3+n/4
	BW-	SLA<W>	(mem)	80+zz+mem	:7C		* * 0P0 *	2+M	6.6
SRA	BWL	SRA	#4, r	C8+zz+r	:ED:#4		* * 0P0 *	3	3+n/4
	BWL	SRA	A, r	C8+zz+r	:FD		* * 0P0 *	2	3+n/4
	BW-	SRA<W>	(mem)	80+zz+mem	:7D		* * 0P0 *	2+M	6.6
SLL	BWL	SLL	#4, r	C8+zz+r	:EE:#4		* * 0P0 *	3	3+n/4
	BWL	SLL	A, r	C8+zz+r	:FE		* * 0P0 *	2	3+n/4
	BW-	SLL<W>	(mem)	80+zz+mem	:7E		* * 0P0 *	2+M	6.6
SRL	BWL	SRL	#4, r	C8+zz+r	:EF:#4		* * 0P0 *	3	3+n/4
	BWL	SRL	A, r	C8+zz+r	:FF		* * 0P0 *	2	3+n/4
	BW-	SRL<W>	(mem)	80+zz+mem	:7F		* * 0P0 *	2+M	6.6
RLD	B - -	RLD	[A,](mem)	80+mem	:06		* * 0P0 -	2+M	14. –. –
RRD	B - -	RRD	[A,](mem)	80+mem	:07		* * 0P0 -	2+M	14. –. –

(注1) #4/Aによるシフト回数の指定では、モジュロ16(0～15)が使われます。コード0は、16回シフトになります。

(注2) TLCS-90のRLCA/RRCA/RLA/RRA/SLAA/SRLA/SLLA/SRLA命令は、S/Z/Vフラグが変化しません。

(注3) ステート数の計算で、小数点以下は切り上げます。

■ TLCS-900/L1**命令一覧表** (9/10)

(9) **ジャンプ、コール、リターン**

命令群	**サイズ**	**ニモニック**	**コード**(16 **進**)	**機能**	SZHVNC	**命令長**	**ステート**
JP	- - -	JP #16	1A :#16	PC ← #16	- - - - - -	3	5
	- - -	JP #24	1B :#24	PC ← #24	- - - - - -	4	6
	- - -	JR [cc,]$+2+d8	60+cc :d8	if cc then PC ← PC+d8	- - - - - -	2	5/2(T/F) (注1)
	- - -	JRL [cc,]$+3+d16	70+cc d16	if cc then PC ← PC+d16	- - - - - -	3	5/2(T/F)
	- - -	JP [cc,]mem	B0+mem :D0+cc	if cc then PC ← mem	- - - - - -	2+M	7/4(T/F)
CALL	- - -	CALL #16	1C :#16	PUSH PC : JP #16	- - - - - -	3	9
	- - -	CALL #24	1D :#24	PUSH PC : JP #24	- - - - - -	4	10
	- - -	CALR $+3+d16	1E :d16	PUSH PC : JP $+3+d16	- - - - - -	3	10
	- - -	CALL [cc,]mem	B0+mem :E0+cc	if cc then PUSH PC : JP mem	- - - - - -	2+M	12/4(T/F)
DJNZ	BW-	DJNZ [r,]$+3/4+d8	C8+zz+r :1C:d8	r ← r – 1(**省略でB レジスタ**) if r≠0 then JR $+3+d8	- - - - - -	3	6(r≠0) 4(r=0)
RET	- - -	RET	0E	POP PC	- - - - - -	1	9
	- - -	RET cc	B0 :F0+cc	if cc then POP PC	- - - - - -	2	12/4(T/F)
	- - -	RETD d16	0F :d16	RET : ADD XSP, d16	- - - - - -	3	11
	- - -	RETI	07	POP SR&PC	* * * * * *	1	12

(注1) (T/F)はTrue/False時のステート数を表しています。

■ TLCS-900/L1**命令一覧表** (10/10)

(10) **アドレッシングモード**

分類	**モード**	**ステート(追加分)**
R	R	+0
r	r	+1
(mem)	(R)	+0
	(R+d8)	+1
	(#8)	+1
	(#16)	+2
	(#24)	+3
	(r)	+1
	(r+d16)	+3
	(r+r8)	+3
	(r+r16)	+3
	(– r)	+1
	(r+)	+1

(11) **割り込み**

モード		**動作**	**ステート**
汎用割り込み処理		PUSH PC PUSH SR IFF←**受け付けレベル**+1 INTNEST←INTNEST +1 JP (FFFF00H + **ベクタ**)	18
マイクロ DMA	I/O to MEM	(DMADn+) ← (DMASn)	8. 8. 12
	I/O to MEM	(DMADn-) ← (DMASn)	8. 8. 12
	MEM to I/O	(DMADn) ← (DMASn+)	8. 8. 12
	MEM to I/O	(DMADn) ← (DMASn-)	8. 8. 12
	I/O to I/O	(DMADn) ← (DMASn)	8. 8. 12
	Counter	(DMASn) ← (DMASn+1)	–. –. 5

(注) 割り込み動作の詳細は、各製品の「割り込み」の章をご覧ください。

付録C. 命令コードマップ (1/4)

1バイトオペコード命令

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP		PUSH SR	POP SR		HALT	EI n	RETI	LD (n), n	PUSH n	LDW (n), nn	PUSHW nn	INCF	DECF	RET	RETD dd
1	RCF	SCF	CCF	ZCF	PUSH A	POP A	EX F, F'	LDF n	PUSH F	POP F	JP nn	JP nnn	CALL nn	CALL nnn	CALR PC+dd	
2	LD R,n								PUSH RR							
3	LD RR, nn								PUSH XRR							
4	LD XRR,nnnn								POP RR							
5									POP XRR							
6	JR cc,PC+d															
	F	LT	LE	ULE	PE/OV	M/MI	Z	C	(T)	GE	GT	UGT	PO/NOV	P/PL	NZ	NC
7	JRL cc,PC+dd ↑															
8	src. B								src. B							
	(XWA)	(XBC)	(XDE)	(XHL)	(XIX)	(XIY)	(XIZ)	(XSP)	(XWA+d)	(XBC+d)	(XDE+d)	(XHL+d)	(XIX+d)	(XIY+d)	(XIZ+d)	(XSP+d)
9	src. W ↑								src. W ↑							
A	src. L ↑								src. L ↑							
B	dst ↑								dst ↑							
C	src. B							reg. B r	reg. B							
	(n)	(nn)	(nnn)	(mem)	(-xrr)	(xrr+)			W	A	B	C	D	E	H	L
D	src. W ↑							reg. W rr	reg. W							
									WA	BC	DE	HL	IX	IY	IZ	SP
E	src. L ↑							reg. L xrr	reg. L							
									XWA	XBC	XDE	XHL	XIX	XIY	XIZ	XSP
F	dst ↑							LDX (n), n	SWI n							
									0	1	2	3	4	5	6	7

(注1) 空白部分のコードは、未定義命令です。

(注2) コード01H、04Hにはダミーの命令が実装されています。使用しないでください。

付録C. 命令コードマップ (2/4)

1バイト目：reg

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				LD r,#	PUSH r	POP r	CPL BW r	NEG BW r	MUL rr,#	MULS rr, #	DIV rr, #	DIVS BW rr, #	LINK L r	UNLKL r	BS1F A, r	BS1B W A, r
1	DAA B r		EXTZ WL r	EXTS WL r	PAA WL r		MIRR W r			MULA W r			DJNZ BW r, d			
2	ANDCF #, r	ORCF #, r	XORCF #, r	LDCF #, r	STCF BW #, r				ANDCF A, r	ORCF A, r	XORCF A, r	LDCF A, r	STCF BW A, r		LDC cr, r	LDC r,cr
3	RES #, r	SET #, r	CHG #, r	BIT #, r	TEST BW #, r				MINC1	MINK2 #, r	MINC4 W		MDEC1	MDEC #, r	MDEC4 W	
4				MUL	R, r			BW				MUL	R, r			BW
5				DIV	R, r			BW				DIVS	R, r			BW
6	8	1	2	INC 3	#3, r 4	5	6	7	8	1	2	DEC 3	#3, r 4	5	6	7
7	F	LT	LE	ULE	PE/OV	M/MI	Z	SCC C	cc, r (T)	GE	GT	UGT	PO/NOV	P/PL	NZ	BW NC
8				ADD	R,r							LD	R, r			
9				ADC	R, r							LD	r, R			
A				SUB	R, r				0	1	2	LD 3	r, #3 4	5	6	7
B				SBC	R,r							EX	R, r			BW
C				AND	R, r				ADD r, #	ADC r, #	SUB r, #	SBC r, #	AND r, #	XOR r, #	OR r, #	CP r, #
D				XOR	R,r				0	1	2	CP 3	r, #3 4	5	6	BW 7
E				OR	R, r				RLC #, r	RRC #, r	RL #, r	RR #, r	SLA #, r	SRA #, r	SLL #, r	SRL #, r
F				CP	R, r				RLC A, r	RRC A, r	RL A, r	RR A, r	SLA A, r	SRA A, r	SLL A, r	SRL A, r

r ： 1バイト目のコードで指定されるレジスタを示します(すべてのCPU内部レジスタが指定可能です)。
R ： 2バイト目のコードで指定されるレジスタを示します(カレンとレジスタ8本のみ指定可能です)。
B ： オペランドサイズは、バイト型です。
W ： オペランドサイズは、ワード型です。
L ： オペランドサイズは、ロング型です。

(注) コード1AH、1BH、3BH、3FHにはダミーの命令が実装されています。使用しないでください。

付録C. 命令コードマップ (3/4)

1**バイト目：**src (mem)

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					PUSH BW (mem)		RLD A, (mem)	RLD B								
1	LDI	LDIR	LDD	LDDR BW	CPI	CPIR	CPD	CPDR BW		LD BW (nn),(m)						
2	LD R, (mem)															
3	EX (mem), R			BW					ADD (mem), #	ADC	SUB	SBC	AND	XOR	OR	CP BW
4	MUL R, (mem)			BW								MULS R, (mem)				BW
5	DIV R, (mem)			BW								DIVS R, (mem)				BW
6	INC #3, (mem)			BW								DEC #3, (mem)				BW
	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7
7									RLC (mem)	RRC	RL	RR	SLA	SRA	SLL	SRL BW
8	ADD R, (mem)											ADD (mem), R				
9	ADC R, (mem)											ADC (mem), R				
A	SUB R, (mem)											SUB (mem), R				
B	SBC R, (mem)											SBC (mem), R				
C	AND R, (mem)											AND (mem), R				
D	XOR R, (mem)											XOR (mem), R				
E	OR R, (mem)											OR (mem), R				
F	CP R, (mem)											CP (mem), R				

B ： オペランドサイズは、バイト型です。
W ： オペランドサイズは、ワード型です。

付録C. 命令コードマップ (4/4)

1**バイト目：**dst (mem)

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LD B (m), #		LD W (m), #		POP B (mem)		POP W (mem)									
1					LD B (m),(nn)		LD W (m),(nn)									
2	LDA R, (mem) W								ANDCF A, (mem)	ORCF A, (mem)	XORCF A, (mem)	LDCF A, (mem)	STCF B A, (mem)			
3	LDA R, (mem) L															
4	LD (mem) R, B															
5	LD (mem) R, W															
6	LD (mem) R, L															
7																
8	ANDCF #3, (mem) B								ORCF #3, (mem) B							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
9	XORCF #3, (mem) B								LDCF #3, (mem) B							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
A	STCF #3, (mem) B								TSET #3, (mem) B							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
B	RES #3, (mem) B								SET #3, (mem) B							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
C	CHG #3, (mem) B								BIT #3, (mem) B							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
D	JP cc, mem															
	F	LT	LE	ULE	PE/OE	M/MI	Z	C	(T)	GE	GT	UGT	PO/NOV	P/PL	NZ	NC
E	CALL cc, mem ↑															
F	RET cc ↑								(**1バイト目のコードは**B0H)							

B ： オペランドサイズは、バイト型です。
W ： オペランドサイズは、ワード型です。
L ： オペランドサイズは、ロング型です。

付録D TLCS-90との相違点

シリーズ / 項目	TLCS-90	TLCS-900/L1
CPU アーキテクチャ 内蔵 ROM/内蔵 RAM 内蔵 I/O 外部データバス	8ビット CPU 8ビットデータバス 8ビットデータバス 8ビットデータバス	16 ビット CPU 16 ビットデータバス 8 ビットデータバス 8 ビット/16 ビットデータバス (混在可能)
プログラム空間 (MMU 付を除く) データ空間	64KB 16MB(バンク)	16MB(リニア) 16MB(リニア)
命令セット/命令ニモニック	TLCS-90	TLCS-90+α α= 16 ビット乗除算命令、 ビット操作命令の強化, 32 ビット転送 / 演算命令, C コンパイラ用命令, レジスタバンク操作命令 etc.
命令コード (オブジェクトコード)	TLCS-90 独自	TLCS-900 独自 (TLCS-90 と違います。)
アドレッシングモード	TLCS-90	TLCS-90+α α= (-Reg), (Reg+), (Reg+disp16), (Reg+Reg16), (nnn)
汎用レジスタ	TLCS-90	TLCS-90+α α= 32 ビット化 レジスタバンク化 システムスタックポインタの追加 [900 CPU のみ]
フラグ (F)	S Z I H X V N C	S Z 0 H 0 V N C I フラグは、SR レジスタの IFF2〜0 に拡張。X フラグは、削除。
リセット	PC←0000H (SP は不変)	PC←(ベクタベース番地) XSP←100H
内蔵 ROM アドレス 内蔵 RAM アドレス 内蔵 I/O アドレス ダイレクトアドレッシング領域 (n)	0000H〜 〜FFxxH FFxxH〜FFFFH FF00H〜FFFFH	不定 不定 000000H〜000FFFH 000000H〜0000FFH
割り込み 割り込みスタートアドレス セーブされるレジスタ マスクレジスタ マスクレベル	 0000H + (8×V) PC と AF IFF 0〜1	 (ベクタベース番地+4×V) PC と SR IFF2〜0 0〜7

項目＼シリーズ	TLCS-90	TLCS-900/L1
命令		
①ADD R, r (**ワード型**)	S, Z, V **フラグは変化しません。** ［16 **ビットのレジスタ間加算以外では、**S, Z, V **フラグは変化します。**］	S, Z, V **フラグは変化します。**
②A **レジスタのシフト**	「RLCA RRCA RLA RRA SLAA SRAA SLLA SRLA 」 **命令では、**S, Z, V **フラグは変化しません。** 「RLC A RRC A RL A RR A SLA A SRA A SLL A SRL A 」 **命令では、**S, Z, V **フラグは変化します。**	S, Z, V **フラグは変化します。**

補　足：TLCS-900/L1は、基本的にTLCS-90との互換性を考慮しつつ、TLCS-90のCPUを16ビット化したシリーズです。
内蔵されているI/Oは、TLCS-90と完全互換性がありますが、CPUについては、合計6種類の直接対応する命令がないので、TLCS-90用に作られたプログラムを移植するときは、下記のような等価命令に置き換えが必要です。

TLCS-90 にあり TLCS-900/L1 にない命令	TLCS-900/L1 での等価命令
EXX	EX BC, BC′ EX DE, DE′ EX HL, HL′
EX AF, AF’	EX A, A′ EX F, F′
PUSH AF	PUSH A PUSH F
POP AF	POP F POP A
INCX	(32 ビット INC 命令)
DECX	(32 ビット DEC 命令)

また、TLCS-900/L1は、TLCS-90と同じ命令でも一部機能強化されている命令があり、オペランドでの指定項目が増えているものがあります。下記に、それらを示します。

TLCS-90	TLCS-900/L1
INC reg INC mem	INC imm3, reg INC imm3, mem
DEC reg DEC mem	DEC imm3, reg DEC imm3, mem
RLC reg RRC reg RL reg RR reg SLA reg SRA reg SLL reg SRL reg	RLC imm, reg RRC imm, reg RL imm, reg RR imm, reg SLA imm, reg SRA imm, reg SLL imm, reg SRL imm, reg