

nX-U8/100 コア

インストラクションマニュアル

CMOS 8ビットマイクロコントローラ

はじめに

このマニュアルは、ラピスセミコンダクタオリジナル 8 ビット 1 チップマイクロコントローラの CPU コアとして用いられる nX-U8/100 コアの命令セットについて述べています。本文中は nX-U8/100 コアを略して U8 コア、または U8 と表現する場合もございますのでご了承ください。nX-U8/100 コア用いた最初の製品として、ML610711 があります。

このマニュアルでは、nX-U8/100 コアの基本アーキテクチャを前提として解説を行っています。お使いになる機種によっては実際に使用できるメモリ容量に制限がある場合があります。制限事項につきましては、各機種のユーザーズマニュアルをご参照ください。

nX-U8/100 をコアとする製品群に関連するマニュアルとして、本書のほかに以下のマニュアルがあります。あわせてお読みください。

■MACU8 アセンブラパッケージ ユーザーズマニュアル

リロケータブルアセンブラ、リンカ、ライブラリアン、オブジェクトコンバータ
の操作方法の説明およびアセンブリ言語仕様の説明

■CCU8 ユーザーズマニュアル

コンパイラの操作方法の説明

■Dr. 610xxx ユーザーズマニュアル

エミュレータ Dr. 610xxx の操作方法説明

■DTU8 デバッガユーザーズマニュアル

デバッガ DTU8 の操作方法説明

このマニュアルは3つの章と付録より構成されます。

各章の概要は次のとおりです。

第1章 アーキテクチャ

nX-U8/100の基本アーキテクチャについて説明します。レジスタやプログラムで扱うCPU資源を解説した後、特徴的な機能、制限およびプログラミングに関しての留意点について述べます。第2章および第3章を理解するために必要とされる基本事項について記述しています。

第2章 アドレッシングモード

レジスタやメモリに対するアクセスを指定する記述方法およびその詳細な動作について説明します。

第3章 命令の詳細

命令の機能とその詳細な動作および命令コードについて解説します。命令の解説はアルファベット順に並べてあります。

付録 命令一覧

nX-U8/100 コア命令の、オペランド表記と命令コードを、機能別に列挙しています。

また、本マニュアルでは説明をわかりやすくするために、次のような表現方法を用いています。

■値の表現

数値の終わりに“H”が付加されている場合、その値は16進数であることを示します。たとえば1000Hと記述した場合は16進数の1000(10進では4096)を表します。

■単位の表現

数値の単位が“バイト”で表現されている場合、8ビットデータであることを意味します。

数値の単位が“ワード”と表現されている場合、16ビットデータであることを意味します。

数値の単位が“ダブルワード”と表現されている場合、32ビットデータであることを意味します。

数値の単位が“クワッドワード”と表現されている場合、64ビットデータであることを意味します。

■範囲の表現

A-Bという表現は、AおよびBを含む値範囲を表します。同様の目的でA-Bを減算と混同しないと思われる個所で用いることがあります。

目次

1	アーキテクチャ	1-1
1.1	概要	1-1
1.1.1	特長	1-1
1.2	CPU資源とプログラミングモデル	1-2
1.2.1	レジスタ	1-3
1.2.1.1	汎用レジスタ	1-4
1.2.1.2	ベースポインタおよびフレームポインタ	1-4
1.2.2	コントロールレジスタ	1-5
1.2.2.1	プログラム・ステータスワード (PSW)	1-5
1.2.2.2	プログラムカウンタ (PC)	1-7
1.2.2.3	コードセグメントレジスタ (CSR)	1-7
1.2.2.4	リンクレジスタ (LR , ELR1 , ELR2 , ELR3)	1-8
1.2.2.5	CSR退避レジスタ (LCSR , ECSR1 , ECSR2 , ECSR3)	1-9
1.2.2.6	PSW退避レジスタ (EPSW1 , EPSW2 , EPSW3)	1-9
1.2.2.7	スタックポインタ (SP)	1-10
1.2.2.8	EAレジスタ (EA)	1-10
1.2.2.9	ARレジスタ (AR)	1-10
1.2.2.10	データセグメントレジスタ (DSR)	1-11
1.3	メモリ空間	1-12
1.3.1	プログラム・メモリ空間	1-12
1.3.2	ベクタテーブル領域	1-13
1.3.2.1	リセットベクタ領域	1-13
1.3.2.2	割込みベクタ領域	1-14
1.3.2.3	ベクタテーブルの記述方法	1-15
1.3.3	プログラムメモリ領域	1-16
1.3.4	DSRプリフィックスコードについて	1-16
1.3.5	データメモリ空間	1-17
1.3.5.1	データ型	1-18
1.3.5.2	アドレス割り付け	1-19
1.3.5.3	ワードバウンダリ	1-19
1.3.5.4	ROMウィンドウ機能	1-20
1.3.6	メモリモデル	1-20
1.3.7	割込み動作について	1-22
1.3.7.1	割込み成立の条件	1-22
1.3.7.2	ノンマスカブル割込み (NMI)	1-23
1.3.7.3	マスカブル割込み (MI)	1-24
1.3.7.4	ソフトウェア割込み (SWI)	1-25
1.4	例外レベルと退避レジスタの取り扱いについて	1-26
1.5	ノンマスカブル割込みに関する注意	1-32

1.6	割込み禁止状態について	1-33
1.7	スタックの変化について	1-34
2	アドレッシングモード	2-1
2.1	アドレッシングモード	2-1
2.2	レジスタアドレッシング	2-1
2.3	メモリアドレッシング	2-2
2.3.1	レジスタ間接アドレッシング	2-3
2.3.2	ダイレクトアドレッシング	2-6
2.4	即値アドレッシング	2-7
2.5	プログラムメモリアドレッシング	2-8
3	命令の詳細	3-1
3.1	概要	3-1
3.2	nX-U8/100コアの命令セット機能別分類表	3-2
3.3	命令実行時間について	3-11
3.4	各命令の説明	3-23
	ADD ERn , ERm	3-24
	ADD ERn , #imm7	3-25
	ADD Rn , obj	3-26
	ADD SP , #signed8	3-27
	ADDC Rn , obj	3-28
	AND Rn , obj	3-29
	B Cadr	3-30
	B ERn	3-31
	Bcond Radr	3-32
	BL Cadr	3-34
	BL ERn	3-35
	BRK	3-36
	CMP ERn , ERm	3-37
	CMP Rn , obj	3-38
	CMPC Rn , obj	3-39
	CPLC	3-40
	DAA Rn	3-41
	DAS Rn	3-42
	DEC [EA]	3-43
	DI	3-44
	DIV ERn , Rm	3-45
	EI	3-46

EXTBW <i>ERn</i>	3-47
INC [EA]	3-48
L <i>ERn, obj</i>	3-49
L <i>QRn, obj</i>	3-51
L <i>Rn, obj</i>	3-52
L <i>XRn, obj</i>	3-54
LEA <i>obj</i>	3-55
MOV <i>CERn, obj</i>	3-56
MOV <i>CQRn, obj</i>	3-57
MOV <i>CRn, obj</i>	3-58
MOV <i>CRn, Rm</i>	3-59
MOV <i>CXRn, obj</i>	3-60
MOV <i>ECSR, Rm</i>	3-61
MOV <i>ELR, ERm</i>	3-62
MOV <i>EPSW, Rm</i>	3-63
MOV <i>ERn, ELR</i>	3-64
MOV <i>ERn, ERm</i>	3-65
MOV <i>ERn, #imm7</i>	3-66
MOV <i>ERn, SP</i>	3-67
MOV <i>obj, CERm</i>	3-68
MOV <i>obj, CQRm</i>	3-69
MOV <i>obj, CRm</i>	3-70
MOV <i>obj, CXRm</i>	3-71
MOV <i>PSW, obj</i>	3-72
MOV <i>Rn, CRm</i>	3-73
MOV <i>Rn, ECSR</i>	3-74
MOV <i>Rn, EPSW</i>	3-75
MOV <i>Rn, PSW</i>	3-76
MOV <i>Rn, obj</i>	3-77
MOV <i>SP, ERm</i>	3-78
MUL <i>ERn, Rm</i>	3-79
NEG <i>Rn</i>	3-80
NOP	3-81
OR <i>Rn, obj</i>	3-82
POP レジスタリスト	3-83
POP <i>obj</i>	3-85
PUSH レジスタリスト	3-86
PUSH <i>obj</i>	3-88
RB <i>Dbitadr</i>	3-89
RB <i>Rn, bit_offset</i>	3-90
RC	3-91
RT	3-92

RTI	3-93
SB <i>Dbitadr</i>	3-94
SB <i>Rn . bit_offset</i>	3-95
SC	3-96
SLL <i>Rn , obj</i>	3-97
SLLC <i>Rn , obj</i>	3-98
SRA <i>Rn , obj</i>	3-99
SRL <i>Rn , obj</i>	3-100
SRLC <i>Rn , obj</i>	3-101
ST <i>ERn , obj</i>	3-102
ST <i>QRn , obj</i>	3-104
ST <i>Rn , obj</i>	3-105
ST <i>XRn , obj</i>	3-107
SUB <i>Rn , Rm</i>	3-108
SUBC <i>Rn , Rm</i>	3-109
SWI <i>#snum</i>	3-110
TB <i>Dbitadr</i>	3-111
TB <i>Rn . bit_offset</i>	3-112
XOR <i>Rn , obj</i>	3-113

4 付録	4-1
4.1 インストラクション表	4-1
4.2 A2コアとA3コアのインストラクションマニュアル記載内容の相違	4-7

1アーキテクチャ

1.1 概要

1.1.1 特長

nX-U8/100 コアは以下のような特長を持っています。

- 豊富な命令セット
転送，算術演算，比較，論理演算，ビット操作，ビット論理演算，ジャンプ，
条件ジャンプ，コール・リターンスタック操作，算術シフト
- 豊富なアドレッシングモード
レジスタアドレッシング
レジスタ間接アドレッシング
スタックポインタアドレッシング
コントロールレジスタアドレッシング
EA レジスタ間接アドレッシング
汎用レジスタ間接アドレッシング
ダイレクトアドレッシング
レジスタ間接ビットアドレッシング
ダイレクトビットアドレッシング
- メモリ空間
プログラムメモリ空間 (ROM)
最大 32K ワード (0000H - FFFFH) * 16 セグメント
データメモリ空間(RAM)
最大 64K バイト (0000H - FFFFH) * 256 セグメント
- 割込み
エミュレータ専用割込み
ノンマスカブル割込み
マスカブル割込み
ソフトウェア割込み

1.2 CPU 資源とプログラミングモデル

U8 は、最大 1M バイトのプログラムメモリ空間と、最大 16M バイトのデータメモリ空間を持っています。全メモリ空間は、64K バイト単位の物理セグメントで区切られており、物理セグメント 0 の空間(0:0000H-0:FFFFH)と、物理セグメントが 1 以上の空間(1:0000-FF:FFFF) では、メモリ構成がづぎのように異なります。

セグメント 0 の空間は、32K ワードのプログラムメモリ領域と、64K バイトのデータメモリ領域が独立して存在し、それぞれプログラムカウンタ (PC) とアドレスレジスタ (AR) を介してその内容を指定します。ただし、AR により指定された領域が ROM ウィンドウ領域であった場合は、AR の内容でプログラムメモリ領域を指定します。

セグメント 1 以上の空間は、アドレッシング対象が 64K バイトを超える空間であり、同一空間上にプログラムメモリ領域とデータメモリ領域がアサインされています。

セグメント 1 以上の空間のプログラムメモリ領域は、コードセグメントレジスタ (CSR) を上位 4 ビット、PC を下位 16 ビットとする 20 ビット (CSR:PC) でその内容を指定します。

セグメント 1 以上の空間のデータメモリ領域は、データセグメントレジスタ (DSR) を上位 8 ビット、AR の内容を下位 16 ビットとする 24 ビットアドレス (DSR:AR) で指定します。

U8 のメモリ空間の概念図を以下に示します。

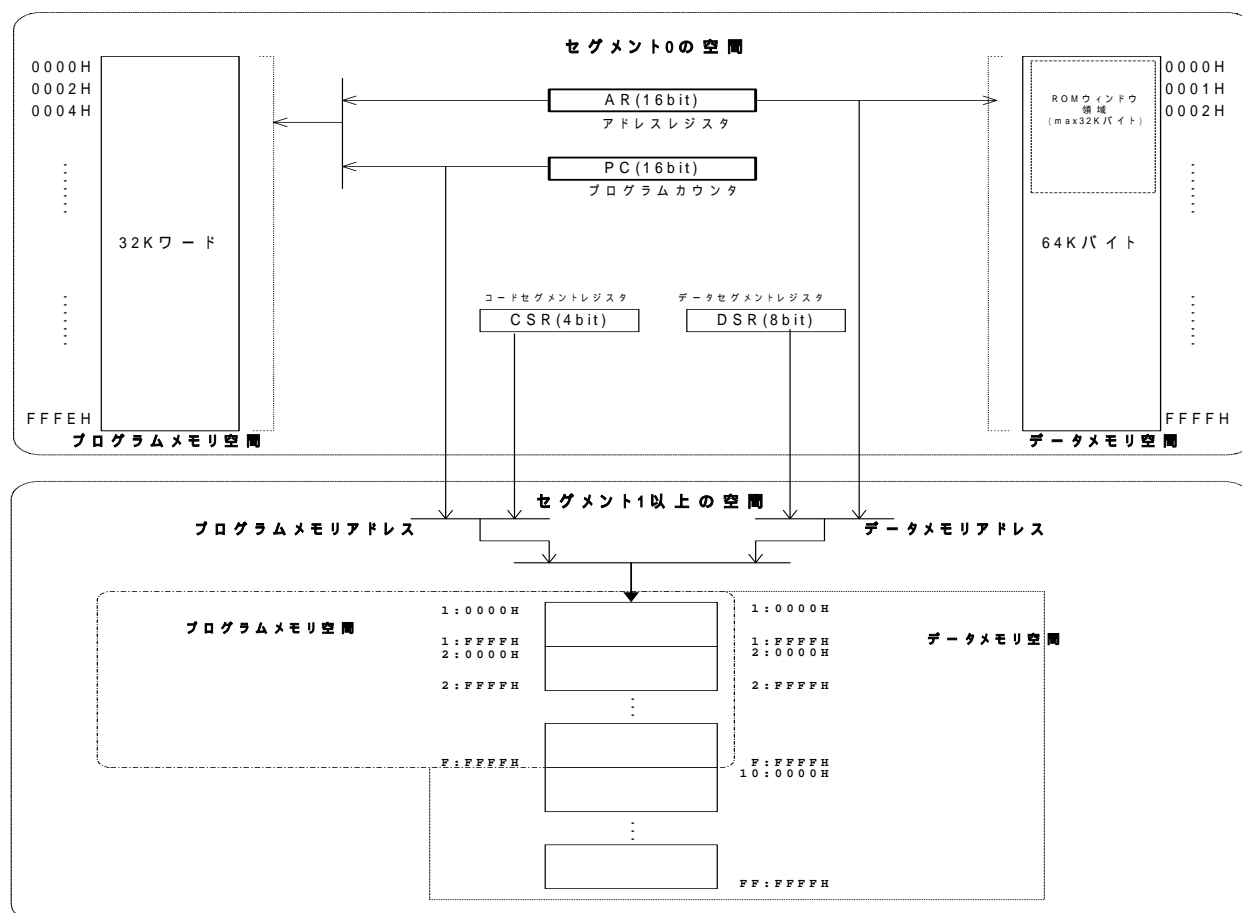


図1-1: U8 のメモリ空間

1.2.1 レジスタ

U8 では汎用レジスタを中心とした処理方式を採用します。レジスタ構成は、その特徴によって、図 1 - 2 のように汎用レジスタ、コントロールレジスタに分けることができます。

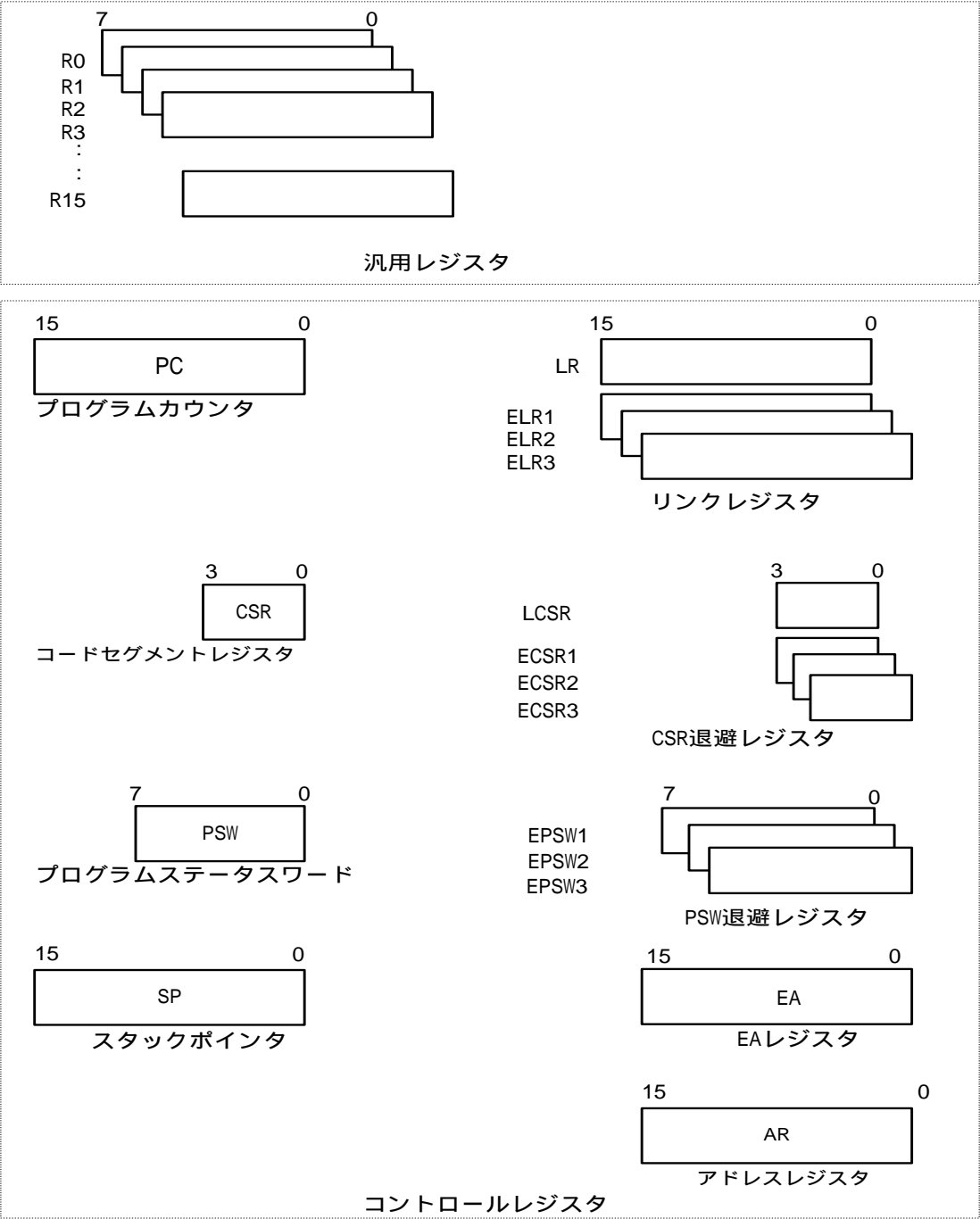


図 1 - 2 : レジスタ構成

1.2.1.1汎用レジスタ

演算の中心となる 16 本のバイト型レジスタです。

レジスタはバイト長ですが、アドレッシングモードの使い分けにより、連続するレジスタを結合して、8 本のワード型レジスタ(ER_n)、4 本のダブルワード型レジスタ(XR_n)、または 2 本のクワッドワード型レジスタ(QR_n)としてアクセスすることができます。割込み発生時のデータの退避はソフトウェアでおこないます。具体的には、割込みサブルーチンの先頭で PUSH 命令を使用して任意のレジスタを退避します。また POP 命令を使用して退避したデータを復帰します。

				7	0
XR0	ER0	R0			
		R1			
	ER2	R2			
		R3			
XR4	ER4	R4			
		R5			
	ER6	R6			
		R7			
XR8	ER8	R8			
		R9			
	ER10	R10			
		R11			
XR12	ER12	R12	BP (下位バイト)	ベースポインタ	
		R13	BP (上位バイト)		
	ER14	R14	FP (下位バイト)	フレームポインタ	
		R15	FP (上位バイト)		

図 1 - 3 : 汎用レジスタ

例 汎用レジスタの使用

```

MOV R0 , #7      ; バイト型
L   ER0 , [EA+]   ; ワード型
L   XR0 , [EA]     ; ダブルワード型
ST  QR0 , [EA]    ; クワッドワード型
SB  R3.2          ; ビット型

```

1.2.1.2ベースポインタおよびフレームポインタ

C コンパイラ使用時に、グローバルポインタとして、ER12 がベースポインタ (BP)、ER14 がフレームポインタ (FP) として使用されます。BP および FP は、汎用レジスタが使用できるアドレッシング以外に、専用のアドレッシングでアクセスすることができます。詳細は第 2 章：アドレッシングモードの項を参照して下さい。

1.2.2 コントロールレジスタ

プログラムの流れを制御し、現在の状態を保持するレジスタ群です。コントロールレジスタは18個あり、各々専用の機能を有しています。これらのレジスタが保持するのは、一般にプログラムコンテキストと呼ばれる情報群の核をなすものです。

1.2.2.1 プログラム・ステータスワード (PSW)

	7	6	5	4	3	2	1	0
PSW	C	Z	S	OV	MIE	HC	ELEVEL	

命令の実行結果の状態が格納される8ビットのレジスタです。プログラムの状態を保持または指定するフラグおよびフィールドから構成されます。PSWの内容は、割り込み時にPSW退避レジスタ(EPSW)に自動退避されます。EPSWに退避されたPSWの値はRTI命令の実行によりPSWに復帰します。

PSWは、CPUの演算状態を保持する5つのフラグと、割り込み制御に関する1つのフラグ、および現在の割り込みのレベルを示す2ビットのフィールド(ELEVEL)から構成され、プログラムにより値を任意に設定することができます。リセット時には0になります。各フラグおよびフィールドの動作を以下に述べます。

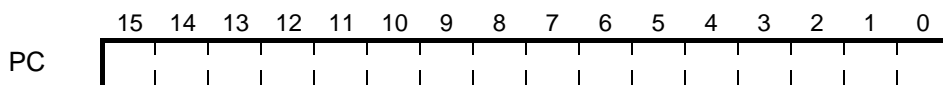
- ビット7: キャリフラグ (C)
算術演算命令、シフト命令、および比較命令の実行結果、ビット7あるいはビット0からのキャリの発生、およびビット7へのボローの発生があると1になり、なければ0になります。
また、SC/RC/CPLC命令により、直接セット・リセット・反転、および条件分岐時命令によってテストすることができます。
- ビット6: ゼロフラグ (Z)
演算の結果が0であることを示します。
演算命令および転送命令の実行結果が0の場合1になり、それ以外の時に0になります。また条件分岐命令によってテストすることができます。
- ビット5: サインフラグ (S)
演算結果が負の数であることを示します。算術、比較演算及び論理演算命令実行の結果、実行結果の符号ビットが1の場合に1になり、そうでない場合0になります。
- ビット4: オーバーフローフラグ (OV)
符号付き演算におけるキャリまたはボローを保持します。算術演算および比較演算などにおいて、2の補数で表現される範囲を越えた場合に1になり、そうでない場合0になります。

- ビット 3 : マスタインタラプトイネーブルフラグ (MIE)
マスカブル割込み全体の許可 / 禁止を制御します。1 の場合に割込み許可になり、0 の場合割込み禁止になります。マスカブル割込みが成立すると、本フラグは割込み移行サイクル中に 0 になります。なお、EI/DI 命令により MIE をセット/リセットすることができます。
- ビット 2 : ハーフキャリフラグ (HC)
10 進演算を実現するために用います。算術演算命令、比較命令実行の結果、ビット 3 またはビット 11 からのキャリまたはボローがあると 1 になり、そうでない場合 0 になります。
- ビット 1-0 : 割込みレベル (ELEVEL)
現在実行中の割込みレベルを保持するフィールドです。
割込みレベルは、割込みの優先度を示す 0-3 の整数で、割込みの種類別にその値が定められています。割込みの種類と割込みレベルについての関係は、“1.3.7 割込み動作について”を参照して下さい。
割込みレベルは大きいほど優先順位が高くなります。
実行中に何らかの割込み要求があると、U8 は要求があった割込みのレベルと、現在の ELEVEL の値を比較し、要求された割込みのレベルが ELEVEL よりも等しいか大きい場合に割込みが発生します。
割込みの種類と優先順位については、“1.3.7 割込み動作について”を参照して下さい。

1.2.2.1.1 命令と PSW のフラグの変化

PSW のフラグ変化を伴う命令と、その実行により変化するフラグについては、第 3 章“3.2 nX-U8/100 コアの命令セット機能別分類表”、または第 4 章のインストラクション表を参照してください。

1.2.2.2 プログラムカウンタ (PC)



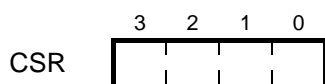
PC は、次に実行するプログラムコードのアドレスを保持する 16 ビットのカウンタです。PC はプログラムメモリからのプログラムコードのフェッチ直後にカウントアップされ、この繰り返しはプログラム実行の流れを作ります。分岐命令では新しいプログラムコードのアドレスが設定されます。

プログラムコードはワード境界に配置されるため、PC の更新は +2 ずつ行われ、LSB は常に 0 が入ります。

リセット直後の PC は、リセット要因に対応するベクタ値となります。割込み時は、実行再開アドレスが割込みレベルに対応する 1 つのリンクレジスタに自動退避されます。

退避されたこの値は、RTI 命令実行により PC に復帰します。

1.2.2.3 コードセグメントレジスタ (CSR)



CSR は現在実行しているプログラムコードが属するセグメントを保持するための 4 ビットのレジスタです。セグメントは 0-15 まで指定することができます。

ひとつのセグメントには 0 - FFFFH のセグメント内オフセットアドレスが割り振られており、PC でオフセットアドレスを指定します。

すなわち全プログラムメモリ空間は、CSR を上位 4 ビット、PC を下位 16 ビットとする 20 ビット (CSR:PC) で指定されます。

アドレッシング対象を決定するためのアドレス計算は 16 ビットのオフセットアドレスで行われ、この際生じるオーバフローやアンダフローは無視されます。したがってこれにより CSR が変化することはありません。同様に PC のオーバフローにより CSR は更新されません。よって CSR 書き換え手段無しにプログラム実行が、セグメント境界をまたいで進行することはありません。

CSR は、次の場合にのみ書き換わります。

割込み発生時	0 になります。
リセット時	0 になります。
B Cadr 命令	命令中で指定された CSR が書き込まれます。
BL Cadr 命令	命令中で指定された CSR が書き込まれます。
RTI 命令	PSW 中の ELEVEL で指し示す ECSR の、いずれかの内容が復帰します。
RT 命令	LCSR の内容が復帰します。
POP PC 命令	スタックメモリの内容が復帰します。

割込みが許可されると、現在の CSR の値は、PSW の ELEVEL フィールドが指し示す 1 つの CSR 退避レジスタ (ECSR1,ECSR2,ESCR3 のいずれか) に自動退避されます。

CSR のリセット直後の値は 0 となります。

1.2.2.4 リンクレジスタ (LR , ELR1 , ELR2 , ELR3)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LR																
ELR1																
ELR2																
ELR3																

リンクレジスタ (LR , ELR1-3) は PC を退避するための 16 ビット×4 本のレジスタで、サブルーチン用 (LR) と、例外処理用 (ELR1-3) の 2 種類があります。いずれのレジスタにも LSB には常に 0 が入ります。

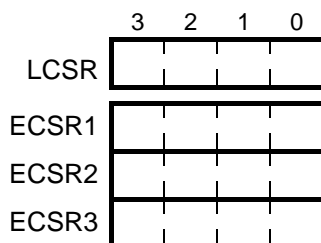
LR は BL 命令によりサブルーチンコールがあった場合に選択され、BL 命令実行中にサブルーチン終了後の戻り番地が自動退避されます。LR の内容は、RT 命令によって PC に復帰します。上位ルーチンへの復帰は、アプリケーションプログラムの実行状態により、RT 命令を使用する方法と、POP 命令を使用する方法する場合があります。詳細は“ 1.4 例外レベルと退避レジスタの取り扱いについて ”を参照して下さい。

ELR1-3 には、割込みが発生した場合に、割込みから復帰する PC が退避されます。ELR1-3 のどのレジスタに退避されるかは、割込みレベルによって決定します。具体的には、割込みの種類に対応する割込みレベル値を指数とする、いずれかひとつのレジスタが選択されることになります。割込みの種類と割込みのレベルについての関係は、“ 1.3.7 割込み動作について ”を参照して下さい。プログラム内で ELEVEL が変化すると、それが指し示す ELR も変化します。したがって PSW を書き換える場合は、現在の ELEVEL と ELR の関係に注意を払う必要があります。

注意

ELR3 はエミュレータおよびデバッガが専用に取り扱うレジスタのため、アプリケーションプログラム中に操作しないで下さい。アプリケーションプログラム中に ELR3 を操作する命令を実行した場合、プログラムの動作は予想できませんのでご注意ください。

1.2.2.5 CSR 退避レジスタ (LCSR, ECSR1, ECSR2, ECSR3)



CSR 退避レジスタは、CSR を退避するための 4 ビット×4 本のレジスタで、サブルーチン用(LCSR)と、例外処理用 (ECSR1、ECSR2、ECSR3) の 3 種類があります。

LCSR は BL 命令によりサブルーチンコールがあった場合に選択され、BL 命令実行中に戻りセグメント値が自動退避されます。LCSR の内容は、RT 命令によって CSR に復帰します。

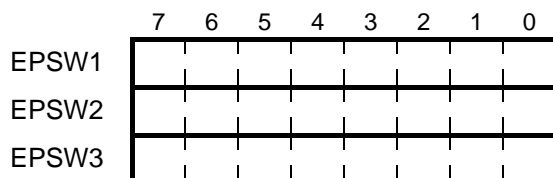
サブルーチンからメイン関数への復帰は、アプリケーションプログラムの実行状態により、RT 命令を使用する方法と、POP 命令を使用する場合があります。詳細は“ 1.4 例外レベルと退避レジスタの取り扱いについて ”を参照して下さい。

ECSR1、ECSR2、および ECSR3 には、割込みが発生した場合にいずれか 1 つのレジスタが選択され、そこに割込みから復帰するセグメントが退避されます。どのレジスタが選択されるかは割込みの種類によって決定します。具体的には、割込みの種類に対応する割込みレベル値を指数とする、いずれかひとつのレジスタが選択されることになります。割込みの種類と割込みのレベルについての関係は“ 1.3.7 割込み動作について ”を参照して下さい。プログラム内で ELEVEL が変化すると、それが指し示す ECSR も変化します。したがって PSW を書き換える場合は、現在の ELEVEL と ECSR の関係に注意を払う必要があります。

注意

ECSR3 はエミュレータおよびデバッガが専用に取り扱うレジスタのため、アプリケーションプログラムに操作しないで下さい。です。アプリケーションプログラム中に ECSR3 を操作する命令を実行した場合、プログラムの動作は予想できませんのでご注意ください。

1.2.2.6 PSW 退避レジスタ (EPSW1, EPSW2, EPSW3)

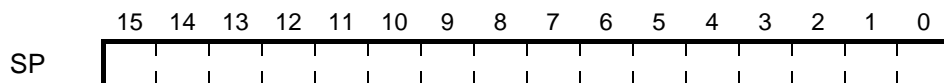


EPSW1、EPSW2、および EPSW3 は、割込み発生時に PSW を退避するための 8 ビットのレジスタです。割込みが許可されると、移行サイクル中に、PSW の ELEVEL フィールドの値を指数とするひとつのレジスタが選択され、PSW が自動退避されます。この値は RTI 命令によって PSW に復帰します。

注意

EPSW3 はエミュレータおよびデバッガが専用に取り扱うレジスタのため、アプリケーションプログラム中で操作しないで下さい。アプリケーションプログラム中に EPSW3 を操作する命令を実行した場合、プログラムの動作は予想できませんのでご注意ください。

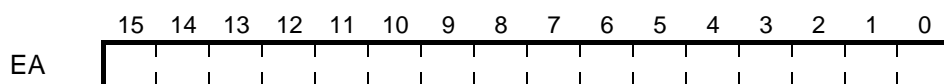
1.2.2.7 スタックポインタ (SP)



SP は、PUSH/POP 命令でレジスタの復帰/退避するための、スタックの先頭アドレスを保持する 16 ビットのレジスタです。

SP を介したスタックへのデータの退避・復帰は常にワード単位で行います。ワード型のデータをスタックに退避する場合、ハードウェアは SP を 2 デクリメントした後に、スタック上にデータを書き込みます。復帰する場合は、現在のスタック上のワード型データを読み出した後、SP を 2 インクリメントします。SP が奇数の場合は、そのアドレスから始まるデータが操作の対象になります。SP は独立したレジスタとして存在し、専用の命令でアクセスすることができます。リセット直後には、プログラムメモリの 0000H 番地の内容を下位バイト、0001H 番地の内容を上位バイトとする値が SP にセットされます。

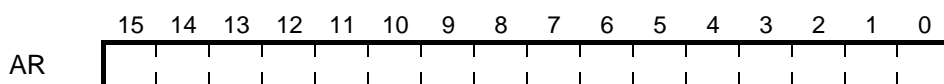
1.2.2.8 EA レジスタ (EA)



EA はメモリアドレスを保持する 16 ビットのレジスタです。メモリにアクセスする命令のいくつかは、EA を間接的に使用して、目的とするアドレスを指定することができます。

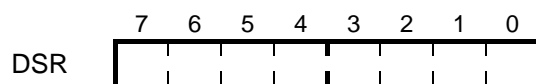
セグメント 0 の空間では、EA の内容そのものがメモリアドレスとなります。セグメント 1 以上の空間では、後述するデータセグメントレジスタ (DSR) を上位 8 ビット、EA を下位 16 ビットとする 24 ビットアドレス (DSR:EA) で、U8 のセグメント 1 以上の空間の全領域を指定することができます。EA は、専用のロード命令でプログラム中で値を任意に書き換えることができます。また、PUSH/POP 命令によって現在の値をスタック上に待避/復帰することが可能です。

1.2.2.9 AR レジスタ (AR)



AR は、メモリアドレスを一時的に保持する 16 ビットのレジスタで、メモリアクセス命令実行中に使用されます。なお、AR は、U8 コアのみが使用可能であり、ユーザプログラム中で取り扱えません。

1.2.2.10 データセグメントレジスタ (DSR)



DSR はデータセグメントを保持するための 8 ビットのレジスタで、セグメント 1 以上の空間のデータセグメントを指定します。セグメントは 0-255 まで指定することができます。

ひとつのデータセグメントには 0-0FFFFH のセグメント内オフセットアドレスが割り振られており、AR でオフセットアドレスを指定します。

すなわち全データメモリ空間は、DSR を上位 8 ビット、EA または AR を下位 16 ビットとする 24 ビットで指定することになります。

メモリアクセス命令中で、データセグメントの値が直接指定された場合は、命令実行中に DSR の内容は更新され、アドレッシング対象は指定されたセグメント値のデータメモリとなります。オペランドとして”DSR”が指定された場合は、アドレッシング対象は現在の DSR が指すセグメント内のデータメモリとなります。

DSR が省略された場合は、現在の DSR の値に関わらず、物理セグメント 0 空間のデータメモリがアドレッシング対象となります。

以下にメモリアクセス命令の実行例を示します。

L	R0	,5:1234H	;	DSR を 5 に更新した後、セグメント 1 以上の空間の 5:1234H の内容を、R0 にロード。
;				
LEA		55AAH		
ST	R0	,3:[EA+]	;	DSR を 3 に更新した後、セグメント 1 以上の空間の 3:55AAH に、R0 の内容をストア。
				EA をインクリメント
ST	R1	,3:[EA+]	;	DSR を 3 に更新した後、セグメント 1 以上の空間の 3:55ABH に、R1 の内容をストア。
				EA をインクリメント
ST	R2	,3:[EA+]	;	DSR を 3 に更新した後、セグメント 1 以上の空間の 3:55ACH に、R1 の内容をストア。
				EA をインクリメント
;				
L	R0	,5:1234H	;	DSR を 5 に更新した後、セグメント 1 以上の空間の 5:1234H の内容を、R0 にロード。
L	R1	,1234H	;	セグメント 0 の空間のデータメモリ 1234H の内容を、R1 にロード
L	R2	,01235H	;	DSR を 0 に更新した後、セグメント 0 の空間の 1235H の内容を、R2 にロード
;				
LEA	AA55H		;	
L	R5,DSR :	[EA+]	;	現在の DSR が指すセグメントの、AA55H の内容を、R5 にロード
				EA をインクリメント
L	R6,DSR:	[EA+]	;	現在の DSR が指すセグメントの、AA56H の内容を、R6 にロード
				EA をインクリメント

DSR の内容は、リセット直後は 0 になります。

1.3 メモリ空間

U8 の全メモリ空間は、64K バイトを 1 つの物理セグメントとして、最大 256 個の物理セグメントで構成されます。

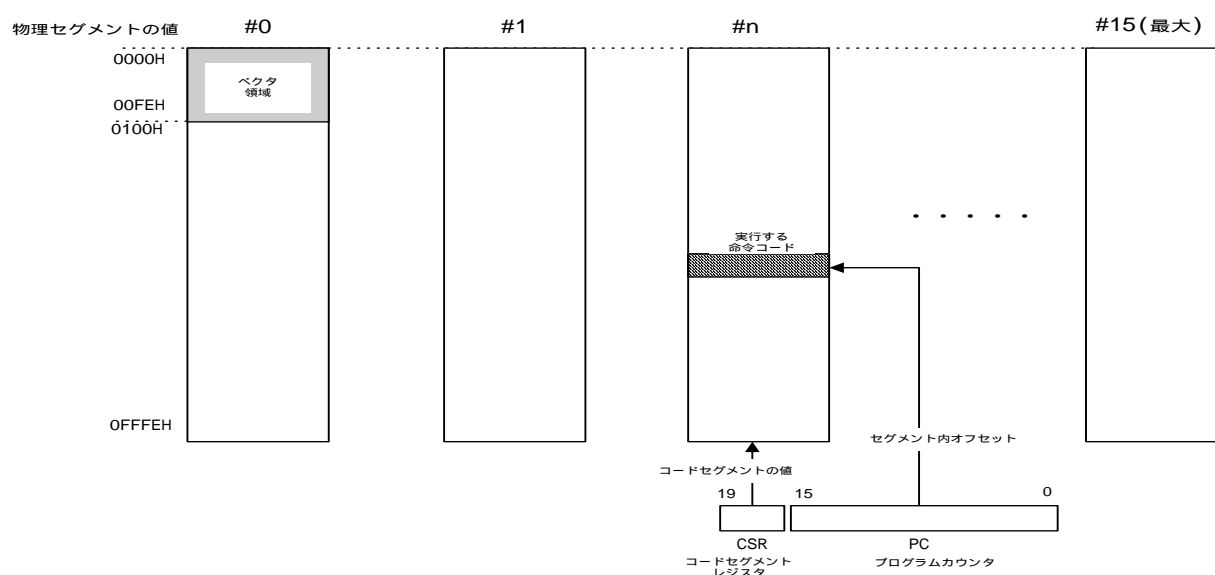
物理セグメント内にはプログラムメモリ空間と、データメモリ空間が存在し、その容量は、プログラムメモリ空間は最大 1M バイト (0:0000H-F:FFFFH)、データメモリ空間は最大 16M バイト (0:0000H-FF:FFFFH) です。それぞれの空間の構造は、物理セグメント 0 (0:0000H-0:FFFFH) と 1 以上 (1:0000H-FF:FFFFH) で異なります。ここではプログラムメモリ空間とデータメモリ空間の構造について述べます。

1.3.1 プログラム・メモリ空間

U8 のプログラム・メモリ空間は 32K ワードのセグメント 16 個で構成され、全体で 1M バイトの容量を持ちます。プログラム・メモリは、実行される命令コード (プログラム・コード) あるいは読み出し専用データ (テーブル・データ) を配置します。

実行中のプログラムコードは、CSR を上位 4 ビット、PC を下位 16 ビットとする 20 ビット (CSR:PC) で指定されます。CSR により選択されるセグメントをコード・セグメントと呼びます。命令実行での PC のインクリメントや、相対ジャンプによる PC に対するディスプレースメントの加減で生じるオーバーフローやアンダフローは無視され、これにより CSR が変化することはありません。また、ROM ウィンドウ機能を用いると、セグメント 0 空間の指定された領域に対して RAM アドレッシングが使用できます。

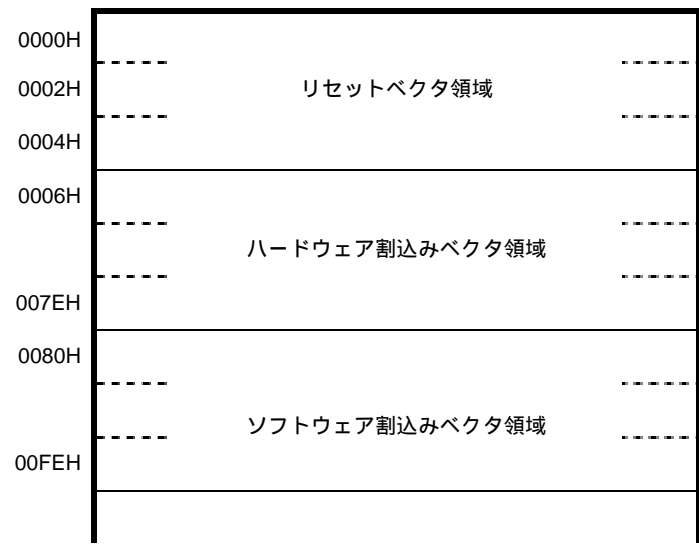
ひとつのセグメントには、0 から 0FFFFEH のセグメント内オフセット・アドレスが割り振られています。アドレッシング対象を決定するためのアドレス計算は、16 ビットのオフセットアドレスで行われ、この際生じるオーバーフローやアンダフローは無視されます。以下にプログラム・メモリ空間の概要を示します。



1.3.2ベクタテーブル領域

プログラムメモリ空間の 0:0H - 0:0FEH は、リセットおよび割込み時に用いられる処理プログラムのエントリアドレス（ベクタ）を格納するベクタテーブル領域です。

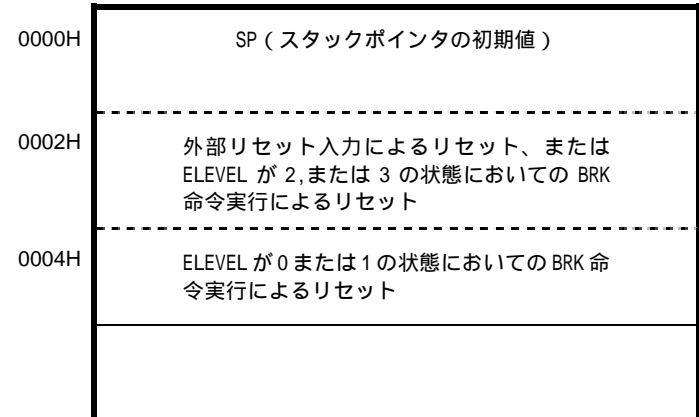
各ベクタは、偶数番地から配置されるワード型のデータです。処理プログラムに移行する時、CSR の値はハードウェアにより 0 にリセットされます。従って処理プログラムのエントリアドレスはセグメント 0 にのみ存在します。



ベクタテーブル領域

1.3.2.1リセットベクタ領域

ベクタテーブルの最初の 3 本は、リセット要因に対応したリセットベクタが割り当てられています。ベクタのアドレスとリセット要因は次の通りです。



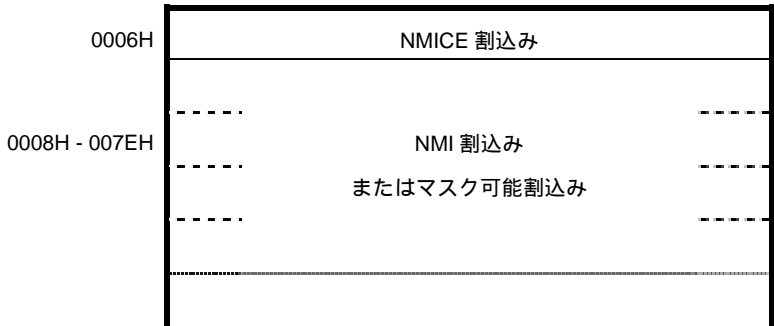
リセットベクタ領域

1.3.2.2割込みベクタ領域

1.3.2.2.1ハードウェア割込み領域

割込み要因は、機種を持つ周辺機能により異なります。本コアには 2 本のマスク不可（NMICE、NMI）割込みベクタと、マスク可能割込みベクタ（MI）が割り付けられます。

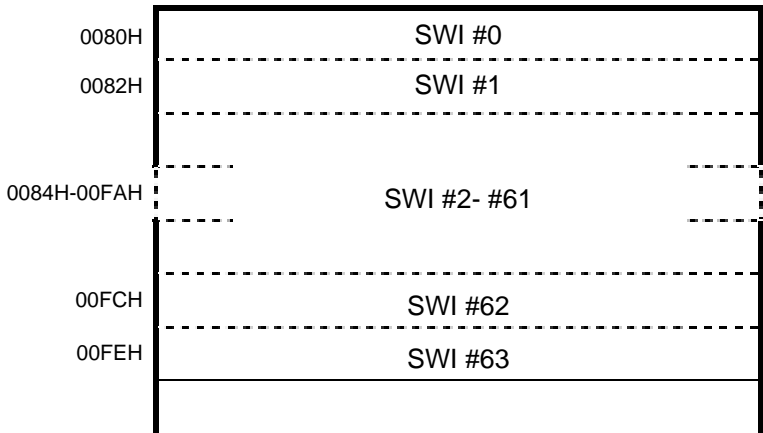
MI 割込みベクタは最大 59 本割り付けることができます。



ハードウェア割込み領域

1.3.2.2.2ソフトウェア割込み領域

SWI 命令のためのベクタ領域です。ベクタのアドレスと、それに対応する SWI 命令は次の通りです。



ソフトウェア割込み領域

1.3.2.3 ベクタテーブルの記述方法

アセンブラでは、dw 疑似命令のオペランドに処理プログラムのエントリアドレスを表すラベルを記述します。ベクタテーブル領域定義のプログラム記述例を次に示します。なお、リセット入力によるリセット以外のベクタ領域を、ベクタとして使用しない場合、通常のプログラムコードを記述して構いません。

```

;-----
;reset vector table
;-----
    cseg at 0000h
        dw      spinit      ;スタックポインタの初期アドレス
        dw      start      ;pc の初期値
        dw      brk        ;brk 命令によるベクタアドレス

    org      0008h
        dw      nmi_entry   ;ノンマスカブル割込み
        dw      Int1_entry  ;マスカブル割込み#0
        dw      Int2_entry  ;マスカブル割込み#1
        :
        :
        :

;-----
;software interrupts
;-----
    cseg at 0080h
swi_0:
        dw      sw0_entry   ;ソフトウェア割込み#1
swi_1:
        dw      sw1_entry   ;ソフトウェア割込み#2
        :
        :
        :

;-----
;start of main procedure
;-----
start:                                     ;プログラムの先頭
        :
        :
        :

```

1.3.3 プログラムメモリ領域

セグメント 0 の空間のプログラム・メモリ領域と、セグメント 1 以上の空間のプログラム・メモリを使用する場合のプログラミング上の論理的な違いはありません。リンカなどを適切に用いて、対象の機種に実装されている内部プログラム・メモリの領域や、外部プログラムメモリ領域にメモリを実装して下さい。

1.3.4 DSR プリフィックスコードについて

物理セグメント 1 以上のデータメモリ空間のアクセスは、DSR プリフィックスコードを実行することで実現することができます。DSR プリフィックスコードと、それに対応する動作は以下のとおりです。

命令フォーマット	動作
1110_0011_iiii_iiii	DSR に、iiii_iiii で示される 8bit 即値をライトする。
1001_0000_ddd_1111	DSR に、ddd で指定される汎用レジスタ値をライトする。
1111_1110_1001_1111	現在の DSR の値を有効にする。

DSR プリフィックスコードは、単独命令として実行するとプログラムの意図しない動作を行う可能性があるため、アセンブラでは常にメモリアクセス命令と組み合わせて使用します。アセンブラでの記述方法については、第 2 章 2.3 “メモリアドレッシング” の項を参照してください。

DSR プリフィックスコードは、その直後に配置した 1 命令だけに有効です。すなわち、DSR プリフィックスコードが直前にないメモリアクセス命令は、現在の DSR の値にかかわらず物理セグメント 0 がアクセス対象となります。

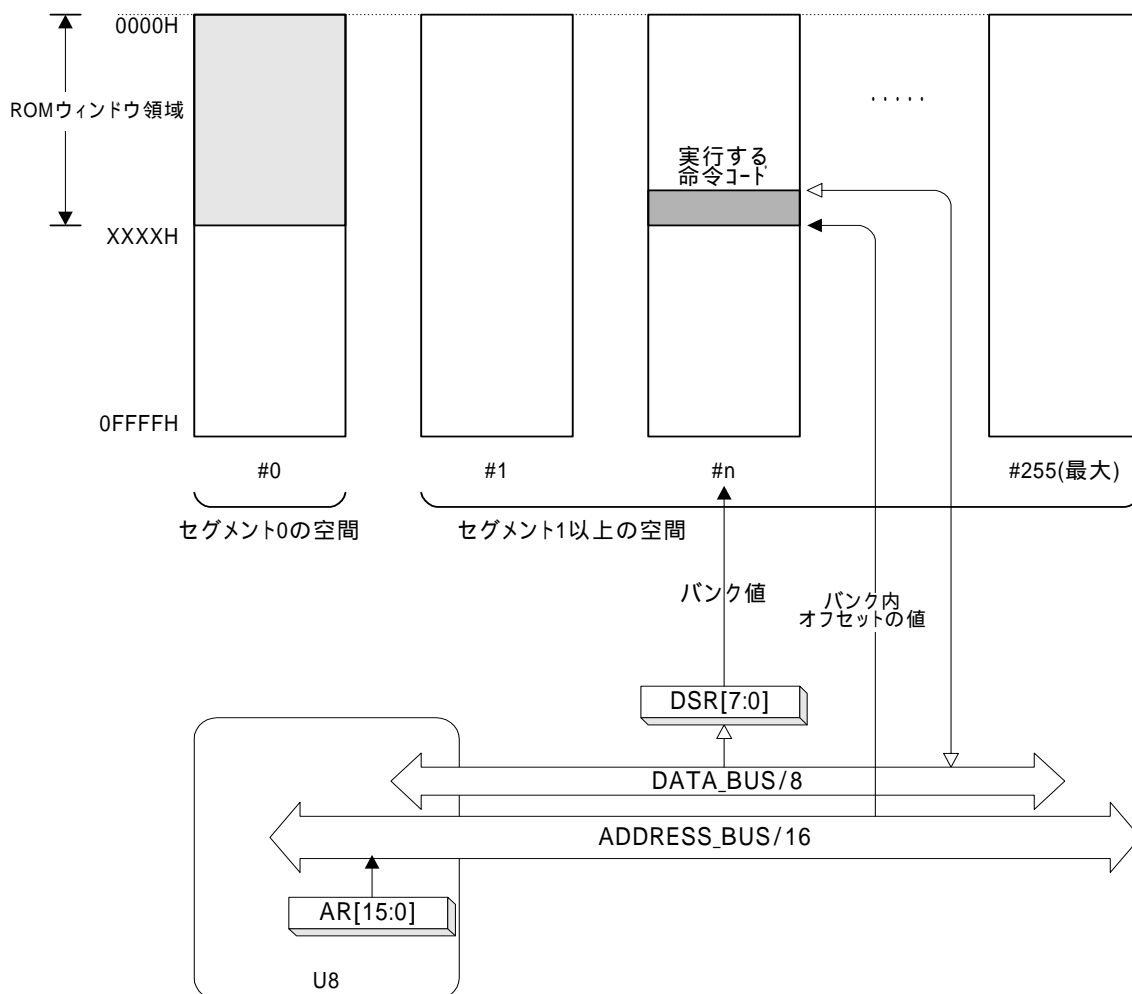
また、DSR プリフィックスコードとその直後の命令の間は、全ての割込みが受け付けられない割込み禁止状態となります。“1.5 割込み禁止状態について”を参照してください。

1.3.5 データメモリ空間

U8 のデータメモリ空間は 64K バイトのセグメント 256 個で構成され、全体で 16M バイトの容量を持ちます。データメモリは、通常読み書き可能なデータを配置します。

データメモリの内容は、DSR を上位 8 ビット、AR を下位 16 ビットとする 24 ビット (DSR:AR) アドレスで指定されます。DSR により選択されるセグメントをデータセグメントと呼びます。U8 のセグメント 0 の空間は、ROM ウィンドウ領域とデータ領域で構成されます。ROM ウィンドウ領域は、ROM 領域のデータを RAM のアドレッシングを用いてアクセスするために開いた窓の存在する領域です。内部データ・メモリのマッピングされていない領域に対し窓が開き、この窓を通して同じアドレスのテーブル・データを読み出すことができます。

データメモリ空間の概要を以下に示します。



1.3.5.1 データ型

ここでは、U8 の命令で使用可能なデータ型について述べます。

符号無しバイト型

バイト型の命令で操作できるデータ型です。0-255 の値範囲を持ちます。この型に対する算術演算で、0-255 の範囲からオーバフローやアンダフローが生じた場合にはキャリ (C) が 1 になり、結果は 256 でモジュロを取った値になります。この型に対するビット演算は、ビット毎に行われます。1 バイトのデータの各ビットには、MSB をビット 7、LSB をビット 0 とするビット位置を表す番号が付けられています。

符号付きバイト型

バイト型の命令で操作できるデータ型です。最上位ビットを符号ビットと見なした 2 の補数表現で、-128 - +127 の値範囲を持ちます。この型に対する算術演算で、-128 - +127 の値範囲からのオーバフローやアンダフローが生じた場合には、オーバフロー・フラグ (OV) が 1 になります。

符号無しワード型

ワード型の命令で操作できるデータ型です。0-65535 の値範囲を持ちます。メモリ上ではビット 7-0 の下位バイトが下位アドレスに、ビット 15-8 の上位バイトが上位アドレスに割り付けられます。データ・メモリ空間ではワードバウンダリのために下位バイトの置かれる下位アドレスは常に偶数アドレスとなります。この型に対する算術演算で、0-65535 の値範囲からのオーバフローやアンダフローが生じた場合には、キャリ (C) が 1 になり結果は 65536 でモジュロをとった値になります。この型に対する論理演算はビット毎に行われます。1 ワードのデータの各ビットには MSB をビット 15、LSB をビット 0 とするビット位置を表す番号が付けられています。

符号付きワード型

ワード型の命令で操作できるデータ型です。最上位ビットを符号ビットとみなした 2 の補数表現で、-32768 - +32767 の値範囲を持ちます。

メモリ上では、ビット 7-0 の下位バイトが下位アドレスに、ビット 15-8 の上位バイトが上位アドレスに割り付けられます。データメモリ空間では、ワードバウンダリのために下位バイトに置かれる下位アドレスは常に偶数アドレスとなります。この型に対する算術演算で、-32768 - +32767 の値範囲からのオーバーフローやアンダフローが生じた場合には、オーバフロー・フラグ (OV) が 1 になります。

ビット型

ビット操作命令でアクセスされるデータ型です。0 と 1 のいずれかの値を持ちます。ビット型のレジスタとメモリ上の全てのビットでこのデータ型を表現できます。バイト型のレジスタや、メモリへのアドレッシングを基に 0-7 のビット位置指定を加えたオペランド記述で指定します。対象の 1 ビットに対して、転送、ビットテスト&ジャンプなどができます。

1.3.5.2 アドレス割り付け

メモリ上のアドレス割り付けは、バイト単位で行われています。

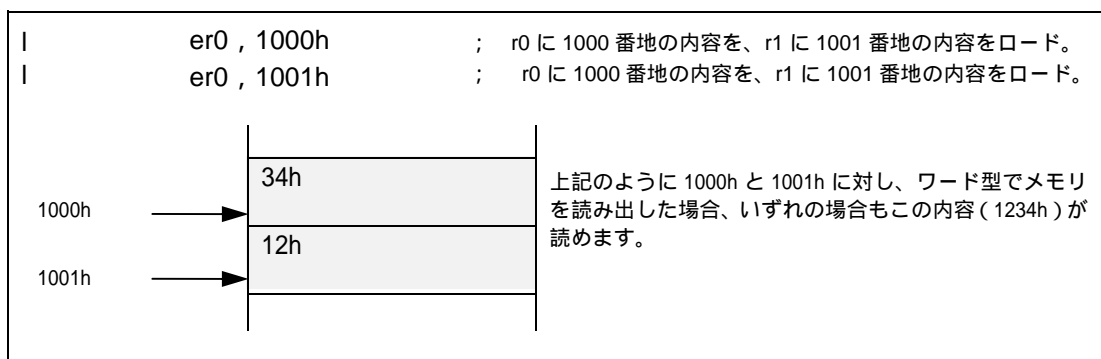
バイトアドレスは、メモリ上の各バイトに対して、個別のアドレスを割り当てたものです。64K バイト空間に最下位アドレス 0 より最上位アドレス 0FFFFH までの 65536 個のアドレスを割り付けています。

U8 には、プログラム・メモリ空間とデータ・メモリ空間の独立した 2 つの空間があり、それぞれの空間にバイトアドレスが割り付けられています。

1.3.5.3 ワードバウンダリ

U8 のデータメモリ空間上には、ワードバウンダリ (ワードアライメント) が存在します。ワードバウンダリとは、奇数番地に対するワード型/ダブルワード型/クワッドワード型のメモリアクセスで生じる仕組みのことです。U8 コアは、奇数番地に対してワード型/ダブルワード型/クワッドワード型でアクセスした場合、アドレスの最下位ビットは 0 とみなされ、1 番地前の偶数番地から始まるデータをアクセスします。この場合アドレスエラーは発生しません。従って、データメモリ空間上のワード長以上のデータはワードバウンダリに従って配置しなければなりません。なお、ダブルワードバウンダリ、クワッドワードバウンダリは存在しません。従ってダブルワード型/クワッドワード型でメモリをアクセスした場合もワードバウンダリ以外の制限はありません。

プログラム・メモリ空間についてもワードバウンダリが存在します。また、ROM ウィンドウ越しにアクセスするプログラムメモリ空間にもワードバウンダリは存在します。



メモリ空間上のワードバウンダリ

1.3.5.4ROM ウィンドウ機能

ROM ウィンドウとは、セグメント0の空間の、メモリが割り付けられていない領域に対してプログラムメモリ領域の内容を参照する機能です。U8 ではプログラムメモリ空間上のデータに対しアクセスするための特別な命令は存在しません。ROM ウィンドウを開いた場合、そこに見えるコードメモリに対するアクセスは、メモリに対する読み出し命令を用います。

ROM ウィンドウ領域を参照した場合の命令サイクル数は、同じ命令でデータメモリを参照したときよりも増加します。マシンサイクルについての詳細は、第3章“3.3 命令実行時間について”を参照してください。

ROM ウィンドウ領域に対するアクセスは読み出しのみ可能です。ROM ウィンドウ領域に対する書き込み動作の結果は保証されません。

1.3.6メモリモデル

U8 には、ハードウェア・メモリ・モデルという考え方があり、アクセス可能な最大プログラムメモリ容量を、64K バイト（32K ワード）と1M バイト（512K ワード）の2つから選択することができます。メモリモデルの選択の方法については、ターゲットチップのユーザーズマニュアルを参照してください。

メモリモデルの呼称と、それに対応するメモリモデルの状態を以下に記します。

メモリモデルの呼称	メモリの範囲	CSR CSR 退避レジスタ
SMALL	プログラムメモリ... 0H - FFFFH データメモリ ... 0H - FF:FFFFH	-
LARGE	プログラムメモリ... 0H - F:FFFFH データメモリ ... 0H - FF:FFFFH	有効

メモリモデルによって以下の項目が変化します。

- アクセスできるプログラムメモリの容量
- サブルーチン呼び出しおよびこれに対応する RT 命令の動作
- 割込みおよびこれに対応する RTI 命令の動作
- メモリへの復帰・待避命令の動作

これらの具体的な違いを以下の表に示します。

	SMALL	LARGE
アクセスできるプログラムの容量	64K バイト (0H - 0FFFFH)	1M バイト (0H - F:FFFFH)
サブルーチン呼び出しで待避される資源	PC	PC CSR
RT 命令で復帰する資源	PC	PC CSR
割込みで待避される資源	PC PSW	PC PSW CSR
RTI 命令で復帰する資源	PC PSW	PC PSW CSR
PUSH 命令で LR/ELR を指定したときにスタックメモリへ待避される資源	LR	LR LCSR
	ELR	ELR ECSR
POP 命令で LR/PC を指定した場合にスタックメモリから復帰される資源	LR	LR LCSR
	PC	PC CSR

1.3.7 割込み動作について

1.3.7.1 割込み成立の条件

割込み条件は、nmice, nmi, mi のいずれかの割込み要求が入力されている状態で、その割込みレベルが現在実行中の割込みレベルよりも等しいか高いときに成立します。割込みレベルとは、割込みの優先度を示す 0-3 の整数で、割込みの種類別にその値が次のように定められています。

各割込み要因に対応する割込みレベルを以下に示します。

割込みの種類	割込みレベル
エミュレータ専用割込み *1	3
ノンマスカブル割込み	2
ソフトウェア割込み	1
マスカブル割込み	1

*1 : エミュレータ専用割込みのため、通常のアプリケーションでは使用できません。

割込みレベルが 0 の場合は、割込みが入っていない状態を表します。

割込みレベル値が大きいほど、優先順位の高い割込みとなります。

割込みレベルは、割込み移行サイクル中に PSW の ELEVEL フィールドにセットされます。

ハードウェアは、発生した割込みのレベルと ELEVEL の値を比較し、発生した割込みのレベルが等しいか大きい場合に、割込み処理に移行します。以下に割込みベクタテーブルアドレスと、ハードウェアの動作の対応を示します。

割込みベクタ テーブルアドレス	備考
0000H	SP (スタックポインタ) の初期値として参照されます。
0002H	ハードウェアリセット入力時、または ELEVEL が 2 以上の状態で BRK 命令を実行した場合のリセットベクタアドレスとして参照されます。
0004H	ELEVEL の値が 1 以下の状態で BRK 命令を実行した場合のベクタアドレスとして参照されます。
0006H	エミュレータ専用割込みの処理に移行します。
0008H - 007EH	ノンマスカブル割込み、またはマスカブル割込みの処理に移行します
0080H - 00FEH	ソフトウェア割込みの処理に移行します。

各割込みの動作の詳細を以下に示します。

1.3.7.2 ノンマスカブル割込み (NMI)

ノンマスカブル割込みはいっさいのマスクができない割込みです。CPU は、NMI 割込み要求を検出すると直ちに NMI 割込み処理に移行します。NMI 割込み実行中に NMI 割込みを検出した場合も、CPU は NMI 割込みに移行します。例外として CPU の実行状態が以下の場合、NMI はマスクされません。

- ・リセット後（ハードウェアリセット入力，または ELEVEL が 3 の状態で BRK 命令の実行）最初の命令の実行を終了するまでの間。
- ・割込み移行サイクルと、割込みルーチンの先頭にある命令の間。
- ・DSR プリフィックスコードと次の命令の間。

NMI 割込みが発生すると、

PC を ELR2 へ転送。

CSR を ECSR2 へ転送。

PSW を EPSW2 へ転送。

PSW の ELEVEL フィールドへ 2 をセット。

CSR を 0 にリセット。

プログラムカウンタ (PC) に、ベクタテーブルに書かれている値をロード。

割込み先頭アドレス番地に対し、割込みの受け付け禁止。

などの一連のハードウェア処理を行い、NMI 割込み処理の最初の命令を実行します。これらのハードウェア処理に必要なサイクル数は 3 サイクルですが、[EA+]アドレッシングを持つ命令の直後に NMI が発生すると、CPU 内部で 1 サイクルのウェイトが挿入されたあと、ハードウェア処理が開始されます。詳細は、“3.3 命令実行時間について”を参照して下さい。

NMI ルーチンからの復帰方法は、NMI 割込みルーチンのプログラム構成により変化します。詳細は、“1.4 例外レベルと退避レジスタの取り扱いについて”を参照して下さい。

1.3.7.3 マスカブル割込み (MI)

マスカブル割込みは、内蔵されている周辺ハードウェアや、外部入力端子による様々な要因により発生します。PSW 内の MIE ビットが “1” の状態で割込み条件が成立すると MI 割込みが発生します。ただし、CPU の実行状態が以下の場合、MI はマスクされます。

- ・リセット後（ハードウェアリセット入力，または ELEVEL が 3 の状態で BRK 命令の実行）最初の命令の実行を終了するまでの間。
- ・割込み移行サイクルと、割込みルーチンの先頭にある命令の間。
- ・DSR プリフィックスコードと次の命令の間。
- ・ELEVEL の値が 2 以上の状態の間。

MI 割込みが発生すると、

PC を ELR1 へ転送。

CSR を ECSR1 へ転送。

PSW を EPSW1 へ転送。

PSW の ELEVEL フィールドへ 1 をセット。

MIE クリア。

CSR を 0 にリセット。

プログラムカウンタ (PC) に、ベクタテーブルに書かれている値をロード。

割込み先頭アドレス番地に対し、割込みの受付禁止。

などの一連のハードウェア処理を行い、MI 割込み処理の最初の命令を実行します。これらのハードウェア処理に必要なサイクル数は 3 サイクルですが、[EA+]アドレッシングを持つ命令の直後に MI が発生すると、CPU 内部で 1 サイクルのウェイトが挿入されたあと、ハードウェア処理が開始されます。詳細は、“3.3 命令実行時間について”を参照して下さい。

MI ルーチンから復帰方法は、MI 割込みルーチンのプログラム構成により変化します。詳細は後述の“1.4 例外レベルと退避レジスタの取り扱いについて”を参照して下さい。

1.3.7.4 ソフトウェア割込み (SWI)

ソフトウェア割込みはアプリケーションプログラム内で任意に発生させるものです。プログラム実行中に SWI 命令を実行すると SWI 割込みが発生します。ソフトウェア割込みは、割込み番号を SWI 命令のオペランド中で指定します。SWI 割込みが発生すると、

PC を ELR1 へ転送。

CSR を ECSR1 へ転送。

PSW を EPSW1 へ転送。

PSW の ELEVEL フィールドへ 1 をセット。

MIE クリア。

CSR を 0 にリセット。

プログラムカウンタ (PC) に、ベクタテーブルに書かれている値をロード。

割込み先頭アドレス番地に対し、割込みの受け付け禁止。

などの一連のハードウェア処理を行い、SWI 割込み処理の最初の命令を実行します。これらのハードウェア処理に必要なサイクル数は 3 サイクルですが、[EA+]アドレッシングを持つ命令の直後に SWI を実行すると、CPU 内部で 1 サイクルのウェイトが挿入されたあと、ハードウェア処理が開始されます。詳細は、“3.3 命令実行時間について”を参照して下さい。

割込みからの復帰方法は“1.4 例外レベルと退避レジスタの取り扱いについて”を参照して下さい。

1.4例外レベルと退避レジスタの取り扱いについて

U8 は、サブルーチンコール時と割込み発生時の戻り番地保持のため、PC 退避レジスタおよび CSR 退避用レジスタを備えています。また割込みエントリ時のプログラム実行状態保持のため、PSW 退避レジスタを備えています。各レジスタの名称と、用途を以下に記します。

PC 退避用レジスタ

名称	用途
LR	BL 命令によるサブルーチンコール時に、戻り PC 値を退避。
ELR1	マスカブル割込み発生時、あるいは SWI 命令実行時に、戻り PC 値を退避。
ELR2	ノンマスカブル割込み発生時に、戻り PC 値を退避。
ELR3	エミュレータ専用割込み発生時に、戻り PC 値を退避。

CSR 退避用レジスタ

名称	用途
LCSR	BL 命令によるサブルーチンコール時、あるいは SWI 命令実行時に、戻り CSR の値を退避。
ECSR1	マスカブル割込み発生時、あるいは SWI 命令実行時に、戻りセグメント値を退避。
ECSR2	ノンマスカブル割込み発生時に、戻りセグメント値を退避。
ECSR3	エミュレータ専用割込み発生時に、戻りセグメント値を退避。

PSW 退避用レジスタ

名称	用途
EPSW1	マスカブル割込み発生時、あるいは SWI 命令実行時に、割込み発生直前の PSW を退避。
EPSW2	ノンマスカブル割込み発生時に、割込み発生直前の PSW を退避。
EPSW3	エミュレータ専用割込み発生時に、割込み発生直前の PSW を退避。

LR,LCSR は、BL 命令実行によるサブルーチンコールおよび RT 命令によるサブルーチンからの復帰時にアクセスされます。

ELR,ECSR,および EPSW は、PSW の ELEVEL の値を指数とするいずれかひとつのレジスタが選択され、割込み発生時および RTI 命令実行時にアクセスされます。

退避レジスタは各要因に対して1つ備えてあります。通常、サブルーチンから復帰する場合は RT 命令を、割込みから復帰する場合は RTI 命令を使用しますが、サブルーチンがネストする場合あるいは多重割込みを許可する場合は、退避レジスタの内容が上書きされてしまうため、RT 命令および RTI 命令は使用できません。このような場合は、割込みルーチンまたはサブルーチンの先頭に PUSH 命令を置き、退避レジスタの内容をスタックメモリに退避します。そして RT 命令ではなく POP 命令を使用してサブルーチンおよび割込みルーチンから復帰します。

このように、アプリケーションプログラムにおける上記レジスタ群の取り扱い方法は、U8 の実行状態により異なりますので、アプリケーションプログラム設計時に注意を払う必要があります。次ページ以降に、U8 の実行状態とアプリケーションプログラムでの対応について記します。

U8 の実行状態を、ELEVEL の値ごとに分け、それぞれの状態下でサブルーチンを呼び出す場合と呼び出さない場合、および、多重割込みを許可する場合と禁止する場合に細分化した場合の、プログラミング時の注意事項を記します。

状態 A：割込みが実行されていない状態

ELEVEL が 0 の実行状態で、退避レジスタとして LR,LCSR が選択されます。

サブルーチンを呼び出すか呼び出さないかで、サブルーチンの開始直後の処理と終了処理が、以下のように異なります。

A-1：サブルーチンを呼び出さない場合

- ・ サブルーチン開始直後の処理
注意すべき取り扱いはありません。
- ・ サブルーチン終了処理
RT 命令を指定し、PC に LR レジスタの内容を復帰させます。

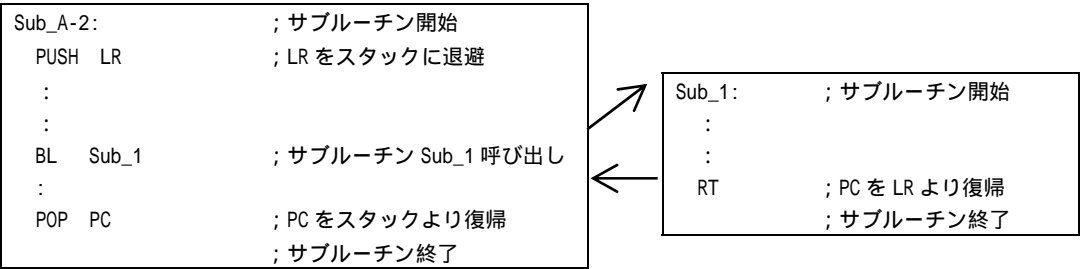
記述例：状態 A-1

```
Sub_A-1:                ; サブルーチン開始
:
:
RT                      ; PC を LR より復帰
                        ; サブルーチン終了
```

A-2：サブルーチンを呼び出す場合

- ・ サブルーチン開始直後の処理
“ PUSH LR ” 命令を指定し、戻り番地をスタックへ退避します。
- ・ サブルーチン終了処理
RT 命令の代わりに“ POP PC ” を指定し、PC にスタックの内容を復帰させます。

記述例：状態 A-2



状態 B : マスカブル割込み実行中

ELEVEL が 1 の実行状態で、退避レジスタとして ELR1,EPSW1,ECSR1 が選択されます。

サブルーチンを呼び出す場合と呼び出さない場合、および、多重割込みを禁止する場合と許可する場合で、以下のように処理が異なります。

B-1 : 割込みルーチン内でサブルーチンを呼び出さない場合

B-1-1 : 多重割込みを禁止する場合

- ・ 割込みルーチン実行開始直後の処理

注意すべき事項は特にありません。

- ・ 割込みルーチン実行終了時の処理

RTI 命令を配置し、PC に ELR1 レジスタの内容を、PSW に EPSW1 レジスタの内容を復帰させます。

記述例 : 状態 B-1-1

Intrpt_B-1-1 :	; 割込みルーチン開始
:	
:	
RTI	; PC を ELR より復帰
	; PSW を EPSW より復帰
	; 割込みルーチン終了

B-1-2 : 多重割込みを許可する場合

- ・ 割込みルーチン実行開始直後の処理

“ PUSH ELR、EPSW ” を指定し、割込みの戻り番地と PSW の状態をスタックに退避します。

- ・ 割込みルーチン実行終了時の処理

RTI 命令の代わりに “ POP PSW、PC ” を指定し、PC と PSW にスタックの内容を復帰させます。

記述例 : 状態 B-1-2

Intrpt_B-1-2:	; 割込みルーチン開始
PUSH ELR,EPSW	; 先頭で ELR,EPSW を退避
:	
:	
EI	; 割込み許可
:	
POP PSW,PC	; PC をスタックより復帰
	; PSW をスタックより復帰
	; 割込みルーチン終了

B-2：割込みルーチン内でサブルーチンを呼び出す場合

B-2-1：多重割込みを禁止する場合

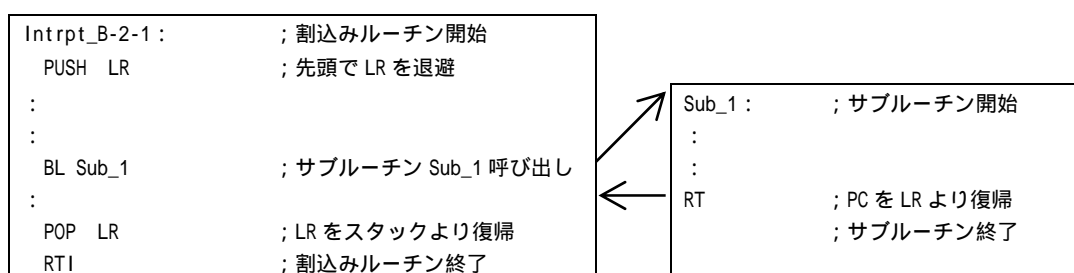
- ## ・ 割込みルーチン実行開始直後の処理

“ PUSH LR “ 命令を指定し、サブルーチンの戻り番地をスタックに退避します。

- ## ・ 割込みルーチン実行終了時の処理

RTI 命令の直前に “ POP LR ” を指定し、サブルーチンの戻り番地を LR に復帰させた後、割込みから復帰します。

記述例：状態 B-2-1



B-2-2：多重割込みを許可する場合

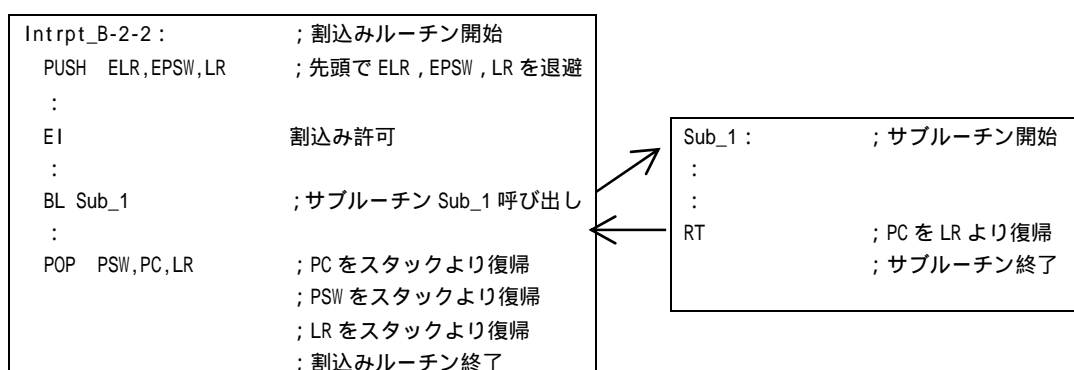
- ## ・ 割込みルーチン実行開始直後の処理

“PUSH ELR、EPSW、LR”を指定し、割込みの戻り番地、サブルーチンの戻り番地およびEPSW1の状態をスタックに退避します。

- ## ・ 割込みルーチン実行終了時の処理

RTI 命令の代わりに“ POP PSW、PC、LR ”を指定し、割込みの戻り番地の退避データはPCへ、EPSW1の退避データはPSWへ、LRの退避データはLRに復帰させます。

記述例：狀態 B - 2 - 2



状態 C：：ノンマスカブル割込み実行中

ELEVEL が 2 の状態で、退避レジスタとして ELR2,EPSW2,ECSR2 が選択されます。

サブルーチンを呼び出す場合と呼び出さない場合で、以下のように処理が異なります。

C-1：割込みルーチン内でサブルーチンを呼び出さない場合

- ・ 割込みルーチンの実行開始直後の処理

“ PUSH ELR、EPSW ” を指定し、割込みの戻り番地と PSW の状態を、スタックに退避します。
- ・ 割込みルーチンの実行終了処理

“ POP PSW,PC ” を指定し、PC と PSW にスタックの内容を復帰させます。

記述例：状態 C-1

Intrpt_C-1 :	; 割込みルーチン開始
PUSH ELR,EPSW	; 先頭で ELR , EPSW を退避
:	
:	
POP PSW,PC	; PC をスタックより復帰
	; PSW をスタックより復帰
	; LR をスタックより復帰
	; 割込みルーチン終了

C-2：割込みルーチン内でサブルーチンを呼び出す場合

- ・ 割込みルーチンの開始直後の処理

“ PUSH ELR , EPSW , LR ” を指定し、割込みの戻り番地、サブルーチンの戻り番地および EPSW の状態をスタックに退避します。
- ・ 割込みルーチンの終了処理

“ POP PSW , PC , LR ” を指定し、割込みの戻り番地の退避データは PC へ、EPSW の退避データは PSW へ、LR の退避データは LR に復帰させます。

記述例：状態 C-2

Intrpt_C-2 :	; 開始
PUSH ELR,EPSW,LR	; 先頭で ELR , EPSW , LR を退避
:	
:	
BL Sub_1	; サブルーチン Sub_1 呼び出し
:	
POP PSW,PC,LR	; PC をスタックより復帰
	; PSW をスタックより復帰
	; LR をスタックより復帰
	; 終了

Sub_1 :	
:	
:	
RT	; PC を LR より復帰
	; サブルーチン終了



このようにアプリケーションプログラムにおける上記レジスタ群の取り扱い方法は、割込み発生時、およびサブルーチンを呼び出し時の CPU の実行状態により異なりますので、アプリケーションプログラム設計時に注意を払う必要があります。

1.5 ノンマスカブル割込みに関する注意

本項では、C 言語でプログラム開発する際の、ノンマスカブル割込みに関する注意点を記します。ノンマスカブル割込み要求は禁止できないため、プログラム中でノンマスカブル割込みの多重割込み対策を実施しなければ、ノンマスカブル割込み実行中に他のノンマスカブル割込みに割り込まれ、プログラムが割込みルーチンから復帰できなくなる可能性があります。

対策として、ターゲットチップがノンマスカブル割込みを使用する場合は、INTERRUPT プラグマで category を 2 に設定してください。このようにすることで、多重割込みが発生しても問題なく動作するよう、割込みルーチンの入口、出口に ELR 等の退避・復帰コードが生成されます。

```
static void int_WDTINT(void);

#pragma interrupt int_WDTINT 0x08 2

static void int_WDTINT(void)
{
    /* 処理*/
}
```

上記の C プログラムに対し、コンパイラが出力するアセンブリコードは以下のようになります。

```
_int_WDTINT :
    push elr, epsw
    . . .
    pop psw, pc
```

割込みルーチン内にて関数呼び出しを伴う場合には、上記 ELR、EPSW の他に、LR および呼び出される関数によって変更される可能性のあるレジスタ (EA, R0 ~ R3) に対する退避・復帰コードが生成されます

1.6 割込み禁止状態について

U8 には、割込み条件が成立していても割込みを一切受け付けない動作状態があります。これを割込み禁止状態と呼びます。

割込み禁止状態と、その状態における割込みの取り扱いは次のようになります。

状態 1．割込み移行サイクルと、割込みルーチンの先頭にある命令の間

ここで割込み条件が成立した場合、すでに許可されている割込みと対応する割込みルーチンの先頭にある命令の実行直後に割込みが発生します。

状態 2．DSR プリフィックスコードと次の命令の間

ここで割込み条件が成立した場合、DSR プリフィックスコードと、次の命令実行直後に割込みが発生します。

DSR プリフィックスコードについては、“1.3.4 DSR プリフィックスコードについて”を参照してください。

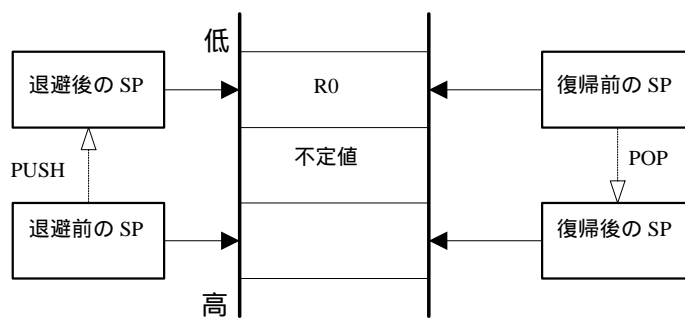
DSR プリフィックスコードを連続して配置すると割込み禁止状態は解除されますが、プログラムの意図しない CPU 動作となる可能性があるため、プログラム中でこのような記載をしないようにしてください。

1.7 スタックの変化について

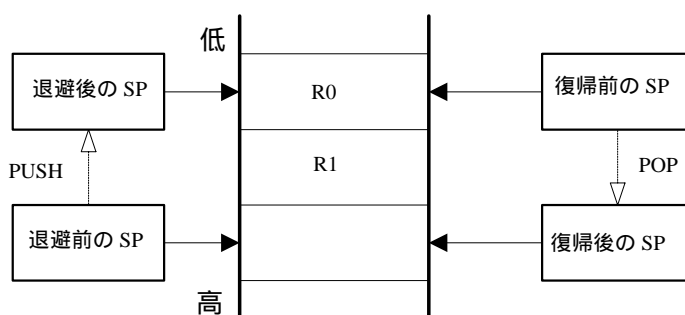
ここでは PUSH 命令および POP 命令実行時のスタックの変化をまとめます。各々の命令の詳細については、第 3 章をご覧ください。命令実行時スタックポインタは常に偶数バイト単位で変化します。奇数バイトのデータを退避する場合、レジスタの退避が行われる前に、1 サイクルのダミーサイクルが挿入されます。ダミーサイクル実行中は 1 バイトの不定値がメモリへの書き込まれます。奇数バイトのデータを復帰する場合には、レジスタ復帰の後、SP のみが 1 インクリメントされるダミーサイクルが挿入されます。ダミーサイクル実行中にスタックポインタの指す内容は復帰されません。また LR および ELR を選択した場合は、その動作がメモリモデルの影響を受けます。

以下に PUSH・POP 命令実行時のスタック動作例を関係を示します。

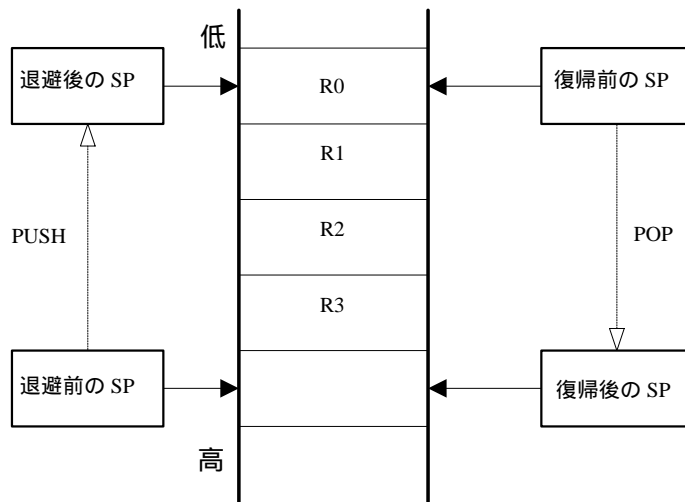
PUSH R0 / POP R0 の動作



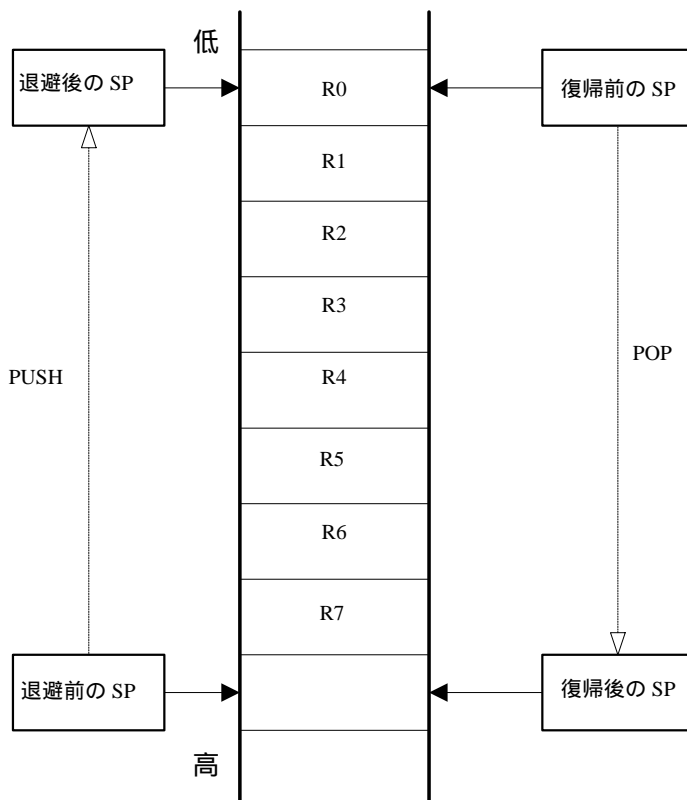
PUSH ER0 / POP ER0 の動作



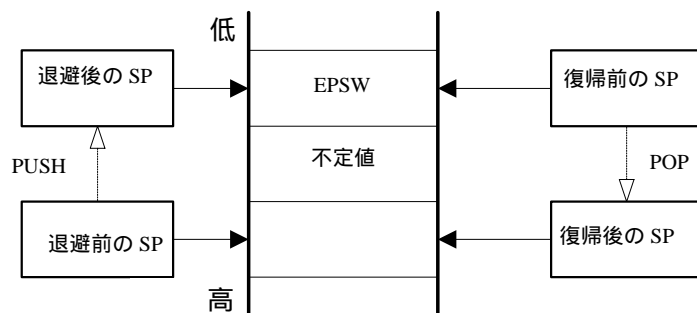
PUSH XR0 / POP XR0 の動作



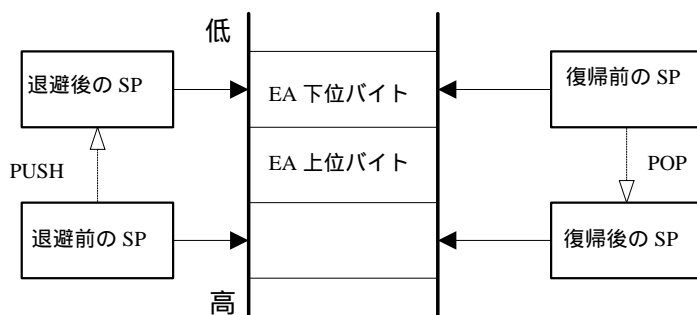
PUSH QR0 / POP QR0 の動作



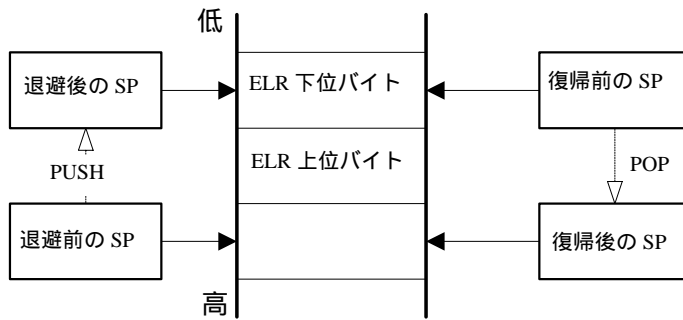
PUSH EPSW / POP PSW の動作



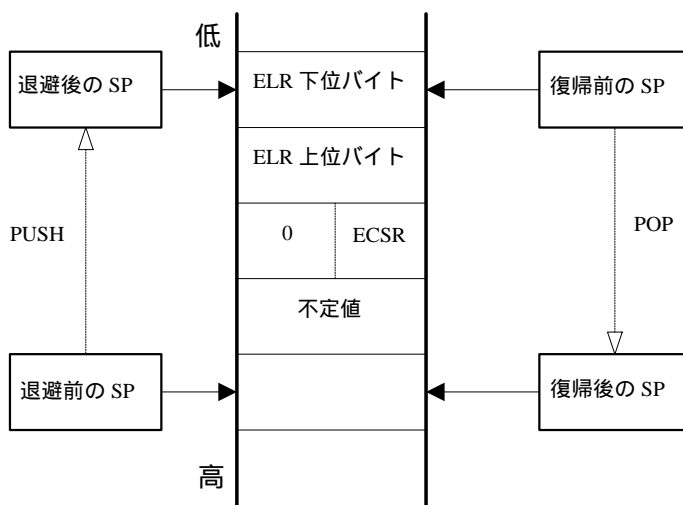
PUSH EA / POP EA の動作



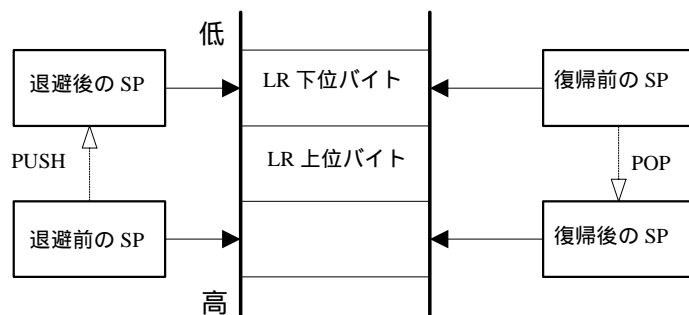
PUSH ELR / POP PC の動作 (スモールモデル)



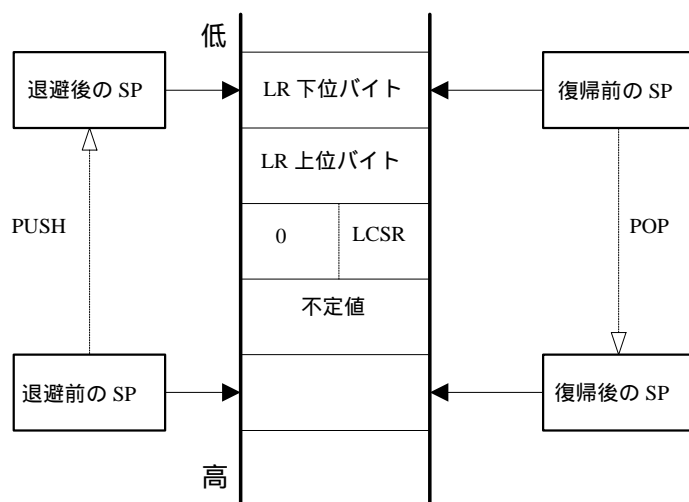
PUSH ELR / POP PC の動作 (ラージモデル)



PUSH LR / POP LR の動作 (スモールモデル)



PUSH LR / POP LR の動作 (ラージモデル)



2アドレスッシングモード

2.1 アドレッシングモード

U8 のアドレッシングモードは、U8 内部のレジスタとコプロセッサレジスタに対応するレジスタアドレッシング、データメモリ空間および ROM ウィンドウで設定された領域に対応するメモリアドレッシング、プログラムメモリ空間に対してのみ対応するプログラムメモリアドレッシング、および命令自身の中にオペランドを指定する即値アドレッシングの4種類があります。

2.2 レジスタアドレッシング

指定したレジスタそのものがアクセスの対象となります。レジスタアドレッシングの記述は、以下のとおりです。

アドレッシング記述	機能
R_n	バイト型の汎用レジスタ (R_n) がアクセスの対象となります。
ER_n	ワード型の汎用レジスタ (ER_n) がアクセスの対象となります。インストラクション表の記述で、オペランドに ER_n が記されている場合、 ER_{12} の代用として BP、 ER_{14} の代用として FP の記述が可能です。
XR_n	ダブルワード型の汎用レジスタ (XR_n) がアクセスの対象となります。
QR_n	クワッドワード型の汎用レジスタ (QR_n) がアクセスの対象となります。
CR_n	バイト型のコプロセッサレジスタ (CR_n) がアクセスの対象となります。
CER_n	ワード型のコプロセッサレジスタ (CER_n) がアクセスの対象となります。
CXR_n	ダブルワード型のコプロセッサレジスタ (CXR_n) がアクセスの対象となります。
CQR_n	クワッドワード型のコプロセッサレジスタ (CQR_n) がアクセスの対象となります。
PC	プログラムカウンタがアクセスの対象となります。
LR	リンクレジスタがアクセスの対象となります。
EA	EA レジスタがアクセスの対象となります。
SP	スタックポインタがアクセスの対象となります。
PSW	プログラムステータスワードがアクセスの対象となります。
ELR	例外処理用のリンクレジスタがアクセスの対象となります。
ECSR	CSR 退避レジスタがアクセスの対象となります。
EPSW	PSW 退避レジスタがアクセスの対象となります。
$R_n.bit_offset$	汎用レジスタ R_n の bit_offset で示されるビット位置のデータがアクセスの対象となります。

2.3 メモリアドレッシング

メモリアドレッシングは、データメモリ空間上のメモリをアクセスの対象とするものです。物理セグメント #0 以外のメモリをアクセスする場合は、DSR プリフィックスコードを使用して、アクセスすべき物理セグメントのアドレスを設定します。しかし DSR プリフィックスコードは、単独命令として取り扱うとプログラムの意図しない動作を行う可能性があるため、アセンブラでは、常にメモリアクセス命令と組み合わせて使用します。DSR プリフィックスコードと、オペランド記述は以下のように対応しています。

DSR プリフィックスコード	ハードウェアの動作	オペランド記述
1110_0011_iiii_iiii	DSR に、iiii_iiii で示される 8bit 即値をライトする。	<i>pseg_addr</i> : または FAR
1001_0000_dddd_1111	DSR に、dddd で指定される汎用レジスタの内容をライトする。	Rd:
1111_1110_1001_1111	現在の DSR の値を有効にする。	DSR:

DSR プリフィックスコードとメモリアクセス命令を組み合わせた場合の記述例を以下に示します。



オペランドの記述例	命令の動作
L R0, <u>1</u> :2345H	DSR 1 R0 [2345H]
L R0, <u>R1</u> :ER2	DSR R1 R0 [ER2]
ST R1, <u>DSR</u> :EA	[(DSR<<16) EA] R1

上の表の下線部分が、実際の DSR プリフィックスコードに対応する記述となります。


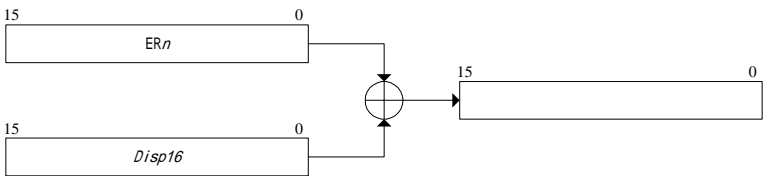
DSR プリフィックスコードを使用しない場合は、アクセスの対象となるメモリの物理セグメントは #0 となります。

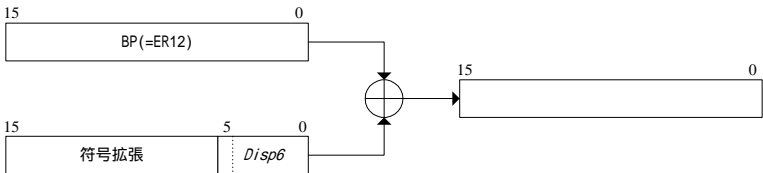
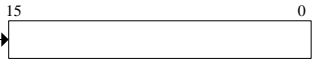
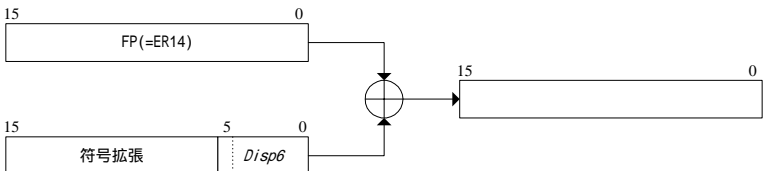
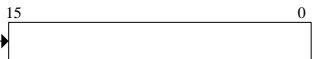
2.3.1 レジスタ間接アドレッシング

レジスタの内容をアドレスとするデータメモリ空間上のメモリが、アクセスの対象となります。レジスタ間接アドレッシングの記述は以下のとおりです。

アドレッシング記述	機能																							
[EA]	<p>EA レジスタの内容をアドレスとする物理セグメント#0 のデータメモリ空間メモリがアクセスの対象となります。</p> <div><div>実効アドレス計算方法</div><div>実効アドレス</div><div></div></div>																							
<i>pseg_addr</i> :[EA]	<p>物理セグメント指定付きのアドレッシングです。物理セグメント #<i>pseg_addr</i> のデータメモリ空間がアクセスの対象となります。</p>																							
DSR:[EA]	<p>物理セグメント指定付きのアドレッシングです。DSR によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。</p>																							
<i>Rd</i> :[EA]	<p>物理セグメント指定付きのアドレッシングです。汎用レジスタ <i>Rd</i> によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。</p>																							
[EA+]	<p>EA レジスタの内容をアドレスとする物理セグメント#0 のデータメモリ空間上のメモリが、アクセスの対象となります。命令実行終了後、EA の内容はオペランドサイズに応じて更新されますが、EA の内容とオペランドサイズにより、更新される値が以下のように異なります。</p> <table><tr><th>オペランドサイズ</th><th>EA の内容</th><th>加算される値</th></tr><tr><td rowspan="2">バイト型</td><td>偶数</td><td>1</td></tr><tr><td>奇数</td><td>1</td></tr><tr><td rowspan="2">ワード型</td><td>偶数</td><td>2</td></tr><tr><td>奇数</td><td>1</td></tr><tr><td rowspan="2">ダブルワード型</td><td>偶数</td><td>4</td></tr><tr><td>奇数</td><td>3</td></tr><tr><td rowspan="2">クワッドワード型</td><td>偶数</td><td>8</td></tr><tr><td>奇数</td><td>7</td></tr></table> <div><div>実効アドレス計算方法</div><div>実効アドレス</div><div></div><div>メモリアクセス後、 インクリメントされる</div></div>	オペランドサイズ	EA の内容	加算される値	バイト型	偶数	1	奇数	1	ワード型	偶数	2	奇数	1	ダブルワード型	偶数	4	奇数	3	クワッドワード型	偶数	8	奇数	7
オペランドサイズ	EA の内容	加算される値																						
バイト型	偶数	1																						
	奇数	1																						
ワード型	偶数	2																						
	奇数	1																						
ダブルワード型	偶数	4																						
	奇数	3																						
クワッドワード型	偶数	8																						
	奇数	7																						

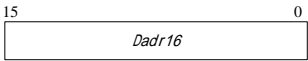
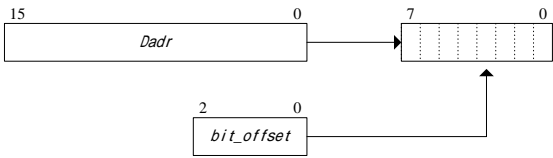
第2章 命令の詳細 アドレッシングモード

アドレッシング記述	機能
<i>pseg_addr</i> :[EA+]	物理セグメント指定付きのアドレッシングです。物理セグメント # <i>pseg_addr</i> のデータメモリ空間が指定されます。
DSR:[EA+]	物理セグメント指定付きのアドレッシングです。DSR によって示される物理セグメントのデータメモリ空間が指定されます。
Rd:[EA+]	物理セグメント指定付きのアドレッシングです。汎用レジスタ <i>Rd</i> によって示される物理セグメントのデータメモリ空間が指定されます。
[ER _{<i>n</i>}]	ワード型汎用レジスタ ER _{<i>n</i>} の内容をアドレスとする物理セグメント #0 のデータメモリ空間上のメモリが、アクセスの対象となります。[ER12]の代用として[BP]、[ER14]の代用として[FP]を記述することも可能です。 <div> <div>実効アドレス計算方法</div> <div>実効アドレス</div>  </div>
<i>pseg_addr</i> :[ER _{<i>n</i>}]	物理セグメント指定付きのアドレッシングです。物理セグメント # <i>pseg_addr</i> のデータメモリ空間がアクセスの対象となります。
DSR:[ER _{<i>n</i>}]	物理セグメント指定付きのアドレッシングです。DSR によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。
Rd:[ER _{<i>n</i>}]	物理セグメント指定付きのアドレッシングです。汎用レジスタ <i>Rd</i> によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。
<i>Disp16</i> [ER _{<i>n</i>}]	<i>Disp16</i> +ER _{<i>n</i>} をアドレスとするデータメモリ空間上のメモリが、アクセスの対象となります。 <div> <div>実効アドレス計算方法</div> <div>実効アドレス</div>  </div>
<i>pseg_addr</i> : <i>Disp16</i> [ER _{<i>n</i>}]	物理セグメント指定付きのアドレッシングです。物理セグメント # <i>pseg_addr</i> のデータメモリ空間がアクセスの対象となります。
DSR: <i>Disp16</i> [ER _{<i>n</i>}]	物理セグメント指定付きのアドレッシングです。DSR によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。
Rd: <i>Disp16</i> [ER _{<i>n</i>}]	物理セグメント指定付きのアドレッシングです。汎用レジスタ <i>Rd</i> によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。

アドレッシング記述	機能
<i>Disp6</i> [BP]	<p><i>Disp6</i>+BP をアドレスとするデータメモリ空間上のメモリがアクセスの対象となります。加算に際して、<i>Disp6</i> は符号拡張されます。DSR プリフィックスがない場合、物理セグメント#0 に限定したアドレッシングになります。</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p>実効アドレス計算方法</p>  </div> <div style="text-align: center;"> <p>実効アドレス</p>  </div> </div>
<i>pseg_addr:Disp6</i> [BP]	物理セグメント指定付きのアドレッシングです。物理セグメント # <i>pseg_addr</i> のデータメモリ空間がアクセスの対象となります
DSR: <i>Disp6</i> [BP]	物理セグメント指定付きのアドレッシングです。DSR によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。
<i>Rd:Disp6</i> [BP]	物理セグメント指定付きのアドレッシングです。汎用レジスタ <i>Rd</i> によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。
<i>Disp6</i> [FP]	<p><i>Disp6</i>+FP をアドレスとするデータメモリ空間上のメモリがアクセスの対象となります。加算に際して、<i>Disp6</i> は符号拡張されます。DSR プリフィックスがない場合、物理セグメント#0 に限定したアドレッシングになります。</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p>実効アドレス計算方法</p>  </div> <div style="text-align: center;"> <p>実効アドレス</p>  </div> </div>
<i>pseg_addr:Disp6</i> [FP]	物理セグメント指定付きのアドレッシングです。物理セグメント # <i>pseg_addr</i> のデータメモリ空間がアクセスの対象となります。
DSR: <i>Disp6</i> [FP]	物理セグメント指定付きのアドレッシングです。DSR によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。
<i>Rd:Disp6</i> [FP]	物理セグメント指定付きのアドレッシングです。汎用レジスタ <i>Rd</i> によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。

2.3.2ダイレクトアドレッシング

記述した値をアドレスとするデータメモリ空間上のメモリをアクセスの対象とします。ダイレクトアドレッシングの記述は以下のとおりです。

アドレッシング記述	機能
<i>Dadr</i>	<p>記述した値をバイトアドレスとするデータメモリ空間のメモリがアクセスの対象となります。</p> <p>実効アドレス</p> 
<i>pseg_addr:Dadr</i>	<p>物理セグメント指定付きのアドレッシングです。物理セグメント <i>pseg_addr</i> のデータメモリ空間がアクセスの対象となります。</p>
<i>DSR:Dadr</i>	<p>物理セグメント付きのアドレッシングです。DSR によって示される物理セグメントのデータメモリ空間がアクセスの対象となります。</p>
<i>Rd:Dadr</i>	<p>物理セグメント付きのアドレッシングです。汎用レジスタ <i>Rd</i> によって示される物理セグメントのデータメモリ空間が指定されます。</p>
<i>Dbitadr</i>	<p>記述した値をビットアドレスとするデータメモリ空間のメモリがアクセスの対象となります。記述形式は <i>Dadr.bit_offset</i> となります。<i>Dadr</i> は対象メモリのアドレス式を表し、<i>bit_offset</i> はメモリ内のビット位置を表します</p> <p>実効アドレス計算方法 実効アドレス</p> 
<i>pseg_addr:Dbitadr</i>	<p>物理セグメント付きのアドレッシングです。物理セグメント <i>pseg_addr</i> のデータメモリ空間がアクセスの対象となります。</p>
<i>DSR:Dbitadr</i>	<p>物理セグメント付きのアドレッシングです。DSR によって示される物理セグメントのデータメモリ空間が指定されます。</p>
<i>Rd:Dbitadr</i>	<p>物理セグメント付きのアドレッシングです。汎用レジスタ <i>Rd</i> によって示される物理セグメントのデータメモリ空間が指定されます。</p>

2.4 即値アドレッシング

記述した値そのものが対象となります。即値アドレッシングの記述は、以下のとおりです。

アドレッシング記述	機能
#imm8	記述した値は、8 ビット即値として扱われます。
#signed8	<p>記述した値は、符号付きの 8 ビット即値として扱われます。</p> <p>ADD SP, #imm8 を記述した場合に、imm8 は signed8 として扱われます。</p> <p>値の範囲は、$-128 \leq \text{signed8} \leq +127$ となります。</p>
#unsigned8	<p>記述した値は、符号なしの 8 ビット即値として扱われます。</p> <p>MOV PSW, #imm8 を記述した場合に、imm8 は unsigned8 として扱われます。</p> <p>値の範囲は、$0 \leq \text{unsigned8} \leq 0\text{FFH}$ となります。</p>
#width	<p>記述した値は、シフト幅として扱われます。</p> <p>値の範囲は、$0 \leq \text{width} \leq 7$ となります。</p>
#snum	<p>記述した値は、SWI 命令のベクタ番号として扱われます。</p> <p>値の範囲は、$0 \leq \text{snum} \leq 63$ となります。</p>
#imm7	<p>記述した値は、符号付きの 7 ビット即値として扱われます。</p> <p>値の範囲は、$-64 \leq \text{imm7} \leq +63$ となります。</p>

2.5 プログラムメモリアドレッシング

プログラムメモリ空間上のメモリがアクセスの対象となります。プログラムメモリアドレッシングの記述は、以下のとおりです。

アドレッシング記述	機能
<i>Cadr</i>	<p>B, BL の分岐先アドレスとなります。<i>Cadr</i> は物理セグメントアドレスを含みますので、異なる物理セグメントアドレスへの分岐が可能です。</p> <p>実効アドレス</p> <div><div>19</div><div>0</div><div><i>Cadr</i>[19:0]</div></div>
<i>Radr</i>	<p>条件分岐、または最適化分岐擬似命令の分岐先アドレスとなります。分岐先アドレスは、同一物理セグメント内に限定されます。</p> <p>実効アドレス計算方法</p> <div><div>15</div><div>0</div><div>PCの内容</div></div> <div><div>15</div><div>7</div><div>符号拡張</div></div> <div><div>7</div><div>0</div><div>命令コード 下位バイト</div></div> <div>符号ビット</div> <div><div>15</div><div>0</div><div><i>Radr</i></div></div>

3 命令の詳細

この章では nX-U8/100 コアの各命令の機能について詳細に説明します。

3.1概要

nX-U8/100 コアの命令には、オペランドを記述しない命令と、第1オペランドまたは第2オペランドを持つ命令があります。

[オペランド表記]

オペランド表記は

＜デスティネーション＞、＜ソース＞

の順番で表記します。

オペランドには、nX-U8/100 コアのアドレッシングモードに対応する記述を表記することができます。アドレッシング記述に関する詳細は、第2章を参照してください。

[記号表記]

命令の機能をわかりやすく表現するため、以下の記号を使用しています。

記号	意味
←	データの代入方向
+, または⊕	加算
-	減算
*	乗算
/	除算
>>	右シフト
<<	左シフト
=	等号
!=	不等号
&	論理積
	論理和
^	排他的論理和
~	反転論理

第3章 命令の詳細 インストラクションセット

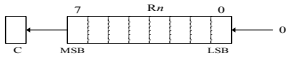
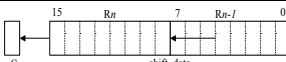
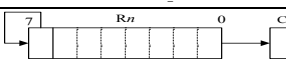
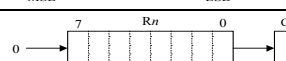

3.2nX-U8/100 コアの命令セット機能別分類表

各命令の詳細な動作については、3.4 各命令の説明 を参照して下さい。

演算命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
ADD	Rn	Rm $\#imm8$	*	*	*	*	*	*	加算命令 $Rn \leftarrow Rn + obj$
MOV	Rn	Rm $\#imm8$		*	*				転送命令 $Rn \leftarrow obj$
ADDC	Rn	Rm $\#imm8$	*	*	*	*	*	*	キャリ付き加算命令 $Rn \leftarrow Rn + obj + c$
CMP	Rn	Rm $\#imm8$	*	*	*	*	*	*	比較命令 $Rn - obj$
CMPC	Rn	Rm $\#imm8$	*	*	*	*	*	*	キャリ付き比較命令 $Rn - obj - c$
AND	Rn	Rm $\#imm8$		*	*				論理積命令 $Rn \leftarrow Rn \& obj$
OR	Rn	Rm $\#imm8$		*	*				論理和命令 $Rn \leftarrow Rn obj$
XOR	Rn	Rm $\#imm8$		*	*				排他的論理和命令 $Rn \leftarrow Rn \wedge obj$
SUB	Rn	Rm	*	*	*	*	*	*	減算命令 $Rn \leftarrow Rn - Rm$
SUBC	Rn	Rm	*	*	*	*	*	*	キャリ付き減算命令 $Rn \leftarrow Rn - Rm - c$
MOV	ERn	ERm $\#imm7$		*	*				転送命令 $ERn \leftarrow obj$
ADD	ERn	ERm $\#imm7$	*	*	*	*	*	*	加算命令 $ERn \leftarrow ERn + obj$
CMP	ERn	ERm	*	*	*	*	*	*	比較命令 $ERn - ERm$

シフト命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
SLL	Rn	Rm $\#width$	*						ハフト型左シフト命令 
SLLC	Rn	Rm $\#width$	*						論理左シフト継続命令 
SRA	Rn	Rm $\#width$	*						算術右シフト命令 
SRL	Rn	Rm $\#width$	*						論理右シフト命令 
SRLC	Rn	Rm $\#width$	*						論理右シフト継続命令 

ロード/ストア命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
L	Rn	[EA]		*	*				8 16 32 型転送命令 Rn ← [EA]
		pseg_addr:[EA]		*	*				
		DSR:[EA]		*	*				
		Rd:[EA]		*	*				
		[EA+]		*	*				8 16 32 型転送命令 Rn ← [EA] EA ← EA+1
		pseg_addr:[EA+]		*	*				
		DSR:[EA+]		*	*				
		Rd:[EA+]		*	*				
		[ERm]		*	*				8 16 32 型転送命令 Rn ← [ERm]
		pseg_addr:[ERm]		*	*				
		DSR:[ERm]		*	*				
		Rd:[ERm]		*	*				
		Disp16[ERm]		*	*				8 16 32 型転送命令 Rn ← Disp16[ERm]
		pseg_addr:Disp16[ERm]		*	*				
		DSR:Disp16[ERm]		*	*				
		Rd:Disp16[ERm]		*	*				
		Disp6[BP]		*	*				8 16 32 型転送命令 Rn ← Disp6[BP]
		pseg_addr:Disp6[BP]		*	*				
		DSR:Disp6[BP]		*	*				
		Rd:Disp6[BP]		*	*				
		Disp6[FP]		*	*				8 16 32 型転送命令 Rn ← Disp6[FP]
		pseg_addr:Disp6[FP]		*	*				
		DSR:Disp6[FP]		*	*				
		Rd:Disp6[FP]		*	*				
		Dadr		*	*				8 16 32 型転送命令 Rn ← Dadr
		pseg_addr:Dadr		*	*				
		DSR:Dadr		*	*				
		Rd:Dadr		*	*				
ERn	ERn	[EA]		*	*				ワード型転送命令 ERn ← [EA]
		pseg_addr:[EA]		*	*				
		DSR:[EA]		*	*				
		Rd:[EA]		*	*				
		[EA+]		*	*				ワード型転送命令 ERn ← [EA] EA ← EA+1
		pseg_addr:[EA+]		*	*				
		DSR:[EA+]		*	*				
		Rd:[EA+]		*	*				
		[ERm]		*	*				ワード型転送命令 ERn ← [ERm]
		pseg_addr:[ERm]		*	*				
		DSR:[ERm]		*	*				
		Rd:[ERm]		*	*				
		Disp16[ERm]		*	*				ワード型転送命令 ERn ← Disp16[ERm]
		pseg_addr:Disp16[ERm]		*	*				
		DSR:Disp16[ERm]		*	*				
		Rd:Disp16[ERm]		*	*				
		Disp6[BP]		*	*				ワード型転送命令 ERn ← Disp6[BP]
		pseg_addr:Disp6[BP]		*	*				
		DSR:Disp6[BP]		*	*				
		Rd:Disp6[BP]		*	*				
		Disp6[FP]		*	*				ワード型転送命令 ERn ← Disp6[FP]
		pseg_addr:Disp6[FP]		*	*				
		DSR:Disp6[FP]		*	*				
		Rd:Disp6[FP]		*	*				
		Dadr		*	*				ワード型転送命令 ERn ← Dadr
		pseg_addr:Dadr		*	*				
		DSR:Dadr		*	*				
		Rd:Dadr		*	*				

(次頁につづく)

第3章 命令の詳細 インストラクションセット

ロード/ストア命令 (続き)

ニーモニック	第 1 オペランド	第 2 オペランド	C	Z	S	OV	MIE	HC	機 能	
L	XRn	[EA]		*	*				ダブ'ワード'型転送命令	XRn←[EA]
		pseg_addr:[EA]		*	*					
		DSR:[EA]		*	*					
		Rd:[EA]		*	*					
		[EA+]		*	*				ダブ'ワード'型転送命令	XRn←[EA]
		pseg_addr:[EA+]		*	*					EA ←EA+1
		DSR:[EA+]		*	*					
		Rd:[EA+]		*	*					
	QRn	[EA]		*	*				クワッド'ワード'型転送命令	QRn←[EA]
		pseg_addr:[EA]		*	*					
		DSR:[EA]		*	*					
		Rd:[EA]		*	*					
		[EA+]		*	*				クワッド'ワード'型転送命令	QRn←[EA]
		pseg_addr:[EA+]		*	*					EA ←EA+1
		DSR:[EA+]		*	*					
		Rd:[EA+]		*	*					

ニーモニック	第 1 オペランド	第 2 オペランド	C	Z	S	OV	MIE	HC	機 能		
ST	Rn	[EA]							h' 16 型転送命令	[EA] ← Rn	
		pseg_addr:[EA]									
		DSR:[EA]									
		Rd:[EA]									
		[EA+]								h' 16 型転送命令	[EA] ←Rn
		pseg_addr:[EA+]								EA←EA+1	
		DSR:[EA+]									
		Rd:[EA+]									
		[ERm]								h' 16 型転送命令	[ERm]← Rn
		pseg_addr:[ERm]									
		DSR:[ERm]									
		Rd:[ERm]									
		Disp16[ERm]								h' 16 型転送命令	Disp16[ERm]← Rn
		pseg_addr: Disp16[ERm]									
	DSR:Disp16[ERm]										
	Rd:Disp16[ERm]										
	Disp6[BP]								h' 16 型転送命令	Disp6[BP]← Rn	
	pseg_addr:Disp6[BP]										
	DSR: Disp6[BP]										
	Rd: Disp6[BP]										
	Disp6[FP]								h' 16 型転送命令	Disp6[FP]← Rn	
	pseg_addr:Disp6[FP]										
	DSR: Disp6[FP]										
	Rd: Disp6[FP]										
	Dadr								h' 16 型転送命令	[Dadr]← Rn	
	pseg_addr: Dadr										
	DSR:Dadr										
	Rd:Dadr										
ERn	[EA]								ワード型転送命令	[EA] ← ERn	
	pseg_addr:[EA]										
	DSR:[EA]										
	Rd:[EA]										
	[EA+]								ワード型転送命令	[EA] ←ERn	
	pseg_addr:[EA+]								EA ←EA+1		
DSR:[EA+]											
Rd:[EA+]											

ロード/ストア命令 (続き)

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
ST	ERn	[ERm] pseg_addr:[ERm] DSR:[ERm] Rd:[ERm]							ワード型転送命令 [ERm] ← ERn
		Disp16[ERm] pseg_addr: Disp16[ERm] DSR: Disp16[ERm] Rd: Disp16[ERm]							ワード型転送命令 Disp16[ERm] ← ERn
		Disp6[BP] pseg_addr: Disp6[BP] DSR: Disp6[BP] Rd: Disp6[BP]							ワード型転送命令 Disp6[BP] ← ERn
		Disp6[FP] pseg_addr: Disp6[FP] DSR: Disp6[FP] Rd: Disp6[FP]							ワード型転送命令 Disp6[FP] ← ERn
		Dadr pseg_addr: Dadr DSR: Dadr Rd: Dadr							ワード型転送命令 [Dadr] ← ERn
	XRn	[EA] pseg_addr:[EA] DSR:[EA] Rd:[EA]							ダブワード型転送命令 [EA] ← XRn
		[EA+] pseg_addr:[EA+] DSR:[EA+] Rd:[EA+]							ダブワード型転送命令 [EA] ← XRn EA ← EA+1
	QRn	[EA] pseg_addr:[EA] DSR:[EA] Rd:[EA]							クワッドワード型転送命令 [EA] ← QRn
		[EA+] pseg_addr:[EA+] DSR:[EA+] Rd:[EA+]							クワッドワード型転送命令 [EA] ← QRn EA ← EA+1

第3章 命令の詳細 インストラクションセット

コントロールレジスタアクセス命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
ADD	SP	<i>#signed8</i>							加算命令 $SP \leftarrow SP + \text{signed8}$
MOV	ECSR	<i>Rm</i>							<ul style="list-style-type: none"> • ELEVEL が 0 $LCSR \leftarrow Rm$ <ul style="list-style-type: none"> • ELEVEL が 0 以外 $ECSR[ELEVEL] \leftarrow Rm$
									<ul style="list-style-type: none"> • ELEVEL が 0 $LR \leftarrow ERm$ <ul style="list-style-type: none"> • ELEVEL が 0 以外 $ELR[ELEVEL] \leftarrow ERm$
	ELR	<i>ERm</i>							<ul style="list-style-type: none"> • ELEVEL が 0 $LR \leftarrow ERm$ <ul style="list-style-type: none"> • ELEVEL が 0 以外 $ELR[ELEVEL] \leftarrow ERm$
	EPSW	<i>Rm</i>							<ul style="list-style-type: none"> • ELEVEL が 0 以外 $EPSW[ELEVEL] \leftarrow Rm$
	<i>ERn</i>	ELR							<ul style="list-style-type: none"> • ELEVEL が 0 $ERn \leftarrow LR$ <ul style="list-style-type: none"> • ELEVEL が 0 以外 $ERn \leftarrow ELR[ELEVEL]$
		SP							転送命令 $ERn \leftarrow SP$
	PSW	<i>Rm</i>	*	*	*	*	*	*	転送命令 $PSW \leftarrow Rm$
		<i>#unsigned8</i>	*	*	*	*	*	*	転送命令 $PSW \leftarrow \text{unsigned8}$
	<i>Rn</i>	ECSR							<ul style="list-style-type: none"> • ELEVEL が 0 $Rn \leftarrow LCSR$ <ul style="list-style-type: none"> • ELEVEL が 0 以外 $Rn \leftarrow ECSR[ELEVEL]$
									<ul style="list-style-type: none"> • ELEVEL が 0 以外 $Rn \leftarrow EPSW[ELEVEL]$
		PSW							転送命令 $Rn \leftarrow PSW$
	SP	<i>ERm</i>							転送命令 $SP \leftarrow ERm$

PUSH/POP 命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
PUSH	<i>ERn</i>								汎用レジスタ退避命令 $SP \leftarrow SP - n$ スタック領域 ← 汎用レジスタ
	<i>Rn</i>								
	<i>QRn</i>								
	<i>XRn</i>								
	<i>register_list</i>								コントロールレジスタ退避命令 $SP \leftarrow SP - n$ スタック領域 ← レジスタ群
POP	<i>ERn</i>								汎用レジスタ復帰命令 汎用レジスタ ← スタック領域 $SP \leftarrow SP + n$
	<i>Rn</i>								
	<i>QRn</i>								
	<i>XRn</i>								
	<i>register_list</i>		*	*	*	*	*	*	コントロールレジスタ復帰命令 レジスタ群 ← スタック領域 * 1 $SP \leftarrow SP + n$

*1 … *register_list* として PSW が指定されたときのみ PSW が影響を受ける

コプロセッサ転送

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能	
MOV	CR _{<i>n</i>}	R _{<i>m</i>}							バイト型転送命令	CR _{<i>n</i>} ← R _{<i>m</i>}
		[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]							バイト型転送命令	CR _{<i>n</i>} ← [EA]
	CER _{<i>n</i>}	[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]							バイト型転送命令	CR _{<i>n</i>} ← [EA+] EA ← EA+1
		[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]							ワード型転送命令	CER _{<i>n</i>} ← [EA]
		[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]							ワード型転送命令	CER _{<i>n</i>} ← [EA+] EA ← EA+1
		[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]							ダブルワード型転送命令	CXR _{<i>n</i>} ← [EA]
	CXR _{<i>n</i>}	[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]							ダブルワード型転送命令	CXR _{<i>n</i>} ← [EA+] EA ← EA+1
		[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]							クワッドワード型連続転送命令	CQR _{<i>n</i>} ← [EA]
		[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]							クワッドワード型連続転送命令	CQR _{<i>n</i>} ← [EA+] EA ← EA+1
		[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]							クワッドワード型連続転送命令	CQR _{<i>n</i>} ← [EA]
	CQR _{<i>n</i>}	[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]							クワッドワード型連続転送命令	CQR _{<i>n</i>} ← [EA+] EA ← EA+1
		[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]							クワッドワード型連続転送命令	CQR _{<i>n</i>} ← [EA]
		[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]							クワッドワード型連続転送命令	CQR _{<i>n</i>} ← [EA+] EA ← EA+1
		[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]							クワッドワード型連続転送命令	CQR _{<i>n</i>} ← [EA]

(次頁に続く)

第3章 命令の詳細 インストラクションセット

コプロセッサ転送 (前頁からのつづき)

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
MOV	R_n	CR_m							16 位型転送命令 $R_n \leftarrow CR_m$
	[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]	CR_m							16 位型転送命令 $[EA] \leftarrow CR_m$
	[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]	CR_m							16 位型転送命令 $[EA] \leftarrow CR_m$ $EA \leftarrow EA+1$
	[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]	CER_m							ワード型転送命令 $[EA] \leftarrow CER_m$
	[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]	CER_m							ワード型転送命令 $[EA] \leftarrow CER_m$ $EA \leftarrow EA+1$
	[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]	CXR_m							ダブルワード型転送命令 $[EA] \leftarrow CXR_m$
	[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]	CXR_m							ダブルワード型転送命令 $[EA] \leftarrow CXR_m$ $EA \leftarrow EA+1$
	[EA] <i>pseg_addr</i> : [EA] DSR: [EA] Rd: [EA]	CQR_m							クワッドワード型連続転送命令 $[EA] \leftarrow CQR_m$
	[EA+] <i>pseg_addr</i> : [EA+] DSR: [EA+] Rd: [EA+]	CQR_m							クワッドワード型連続転送命令 $[EA] \leftarrow CQR_m$ $EA \leftarrow EA+1$

EA レジスタ転送命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
LEA	[ER n]								EA への転送命令 $EA \leftarrow ER_n$
	<i>Disp16</i> [ER m]								$EA \leftarrow Disp16 + ER_m$
	<i>Dadr</i>								$EA \leftarrow Dadr$

ALU 命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
DAA	R_n		*	*	*		*		16 位型 10 進加算補正命令
DAS	R_n		*	*	*		*		16 位型 10 進減算補正命令
NEG	R_n		*	*	*	*	*		符号反転命令 $R_n \leftarrow 0 - R_n$

ビットアクセス命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
SB	$Rn.bit_offset$			*					セットビット命令 $z \leftarrow \sim Rn.bit_offset$ $Rn.bit_offset \leftarrow 1$
	$Dbitadr$			*					
	$pseg_addr: Dbitadr$			*					$z \leftarrow \sim [Dbitadr]$
	$DSR: Dbitadr$			*					$[Dbitadr] \leftarrow 1$
	$Rd: Dbitadr$			*					
RB	$Rn.bit_offset$			*					リセットビット命令 $z \leftarrow \sim Rn.bit_offset$ $Rn.bit_offset \leftarrow 0$
	$Dbitadr$			*					
	$pseg_addr: Dbitadr$			*					$z \leftarrow \sim [Dbitadr]$
	$DSR: Dbitadr$			*					$[Dbitadr] \leftarrow 0$
	$Rd: Dbitadr$			*					
TB	$Rn.bit_offset$			*					テストビット命令 $z \leftarrow \sim Rn.bit_offset$
	$Dbitadr$			*					
	$pseg_addr: Dbitadr$			*					
	$DSR: Dbitadr$			*					$z \leftarrow \sim [Dbitadr]$
	$Rd: Dbitadr$			*					

PSW アクセス命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
EI						*			割込み許可命令 $MIE \leftarrow 1$
DI						*			割込みマスク命令 $MIE \leftarrow 0$
SC				*					セットキャリ命令 $C \leftarrow 1$
RC				*					リセットキャリ命令 $C \leftarrow 0$
CPLC				*					キャリ反転命令 $C \leftarrow \sim C$

条件相対分岐命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
$Bcond$	$Radr$								条件分岐命令 $if\ cond ? Radr : PC+2$
BC	$cond$	$Radr$							

符号拡張命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
EXTBW	ERn			*	*				符号拡張命令 $ERn \leftarrow (\text{符号拡張})Rn$

第3章 命令の詳細 インストラクションセット

ソフトウェア割込み命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
SWI	# <i>snum</i>					*			ソフトウェア 割込み命令 address ← (<i>snum</i> << 1), PC ← Vector-table(address)
BRK									ブレーク命令 ・ ELEVEL が 2 以上 System reset ・ ELEVEL が 1 以下 PC ← (Vector-table 0004H)

分岐命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
B	<i>Cadr</i>								分岐命令 CSR ← <i>Cadr</i> [19:16] PC ← <i>Cadr</i> [15:0]
		<i>ERn</i>							PC ← <i>ERn</i>
BL	<i>Cadr</i>								分岐命令 LR ← 次の命令の先頭アドレス LCSR ← CSR CSR ← <i>Cadr</i> [19:16] PC ← <i>Cadr</i> [15:0]
		<i>ERn</i>							LR ← 次の命令の先頭アドレス LCSR ← CSR PC ← <i>ERn</i>

乗除算命令

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
MUL	<i>ERn</i>	<i>Rm</i>				*			乗算命令 $ERn \leftarrow Rn * Rm$
DIV	<i>ERn</i>	<i>Rm</i>				*	*		除算命令 $ERn \leftarrow ERn / Rm, Rm \leftarrow ERn \bmod Rm$

その他

ニーモニック	第1 オペランド	第2 オペランド	C	Z	S	OV	MIE	HC	機 能
INC	[EA] <i>pseg_addr</i> :[EA] DSR:[EA] <i>Rd</i> :[EA]			*	*	*	*	*	メモリ加算 [EA] ← [EA] + 1
DEC	[EA] <i>pseg_addr</i> :[EA] DSR:[EA] <i>Rd</i> :[EA]			*	*	*	*	*	メモリ減算 [EA] ← [EA] - 1
RT									サブルーチンからの復 帰命令 CSR ← LCSR PC ← LR
RTI			*	*	*	*	*	*	割込みからの復帰 命令 CSR ← ECSR[ELEVEL] PC ← ELR[ELEVEL] PSW ← EPSW[ELEVEL]
NOP									

3.3 命令実行時間について

本項では、nX-U8/100 コアの命令実行時間を示します。

クロック周波数への依存を避けるため、命令実行時間をクロック・サイクルで示します。

また、メモリのリードライトサイクルタイムはともに1サイクルであると仮定しています。それよりも長いメモリサイクルがあると待ち状態が発生するため、それを命令実行時間の合計に加える必要があります。

命令はそのマシンサイクルを単位として最低3マシンサイクルで実行されますが、nX-U8/100 コアは命令フェッチ、命令デコード、命令実行および演算の書き込みの3つのステートを並列処理するため、最も効率的に命令処理を行った場合、本来のマシンサイクルよりも早く命令実行を終了することができます。

このように命令がもっとも効率的に処理された場合の実行サイクル値を最小実行サイクルと呼びます。

命令実行中は、並列処理であるがゆえに、各ステート間で資源の競合が発生する場合があります。この場合 nX-U8/100 コアは、最低1マシンサイクルのウェイトサイクルを命令実行パイプラインに挿入し、一方のステートの処理開始を遅らせることで資源の競合を防ぎます。

以下にウェイトサイクルが挿入される条件をまとめます。

条件1. RomWindows 領域のアクセスを行うと、 $n \cdot m$ (n :アクセスバイト数 m :1バイトアクセス時のウェイト数)

のウェイトサイクルが挿入されます。

条件2. [EA+] のアドレッシングを持つ命令の直後に配置すると、CPU 内部バスが競合する命令があります。この場合、1サイクルのウェイトサイクルが付加されます。

条件3. [EA+] のアドレッシングを持つ命令の直後に割込みが発生すると、CPU 内部バスが競合します。この場合、1サイクルのウェイトサイクルが付加された後、割込み処理が開始されます。割込みについては、「1.3.7 割込み動作について」を参照して下さい。

すなわち、ある命令の全体の実行サイクルは、

(最小実行サイクル数+バスの競合によるウェイト数+データアクセス時のウェイト数)

となります。

次ページ以降に、nX-U8/100 コアの全命令と、それぞれが上記状態にある場合に付加されるサイクル数を記します。表中空白の欄は、その命令実行中には資源が競合する状態が存在しないか、メモリアクセスを行わないことを意味します。

第3章 命令の詳細 インストラクションセット

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレッシングの影響
ADD	ER _n	ER _m	2		
		#imm7	2		
ADD	R _n	R _m	1		
		#imm8	1		
	SP	#signed8	2		
ADDC	R _n	R _m	1		
		#imm8	1		
AND	R _n	R _m	1		
		#imm8	1		
B	Cadr		2		1
	ER _n		2		1
Bcond	Radr		1 / 3 ^(*1)		1
BL	Cadr		2		1
	ER _n		2		1
BRK			7		1
CMP	ER _n	ER _m	2		
	R _n	R _m	1		
		#imm8	1		
CMPC	R _n	R _m	1		
		#imm8	1		
CPLC			1		
DAA	R _n		1		
DAS	R _n		1		
DEC	[EA]		2		1
	pseg_addr:[EA]				
	DSR:[EA]		3		
	Rd:[EA]				
DI			3		
DIV	ER _n	R _m	17		
EI			1		
EXTBW	ER _n		1		
INC	[EA]		2		1
	pseg_addr:[EA]				
	DSR:[EA]		3		
	Rd:[EA]				

(*1) … (分岐条件非成立時 / 条件成立時)

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレッシング の影響
L	ERn	[EA]	2	2	
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	3	2	
		Rd: [EA]			
		[EA+]	2	2	
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	3	2	
		Rd: [EA+]			
		[ERm]	2	2	1
		<i>pseg_addr</i> : [ERm]			
		DSR: [ERm]	3	2	
		Rd: [ERm]			
		<i>Disp16</i> [ERm]	3	2	1
		<i>pseg_addr</i> : <i>Disp16</i> [ERm]			
		DSR: <i>Disp16</i> [ERm]	4	2	
		Rd: <i>Disp16</i> [ERm]			
		<i>Disp6</i> [BP]	3	2	1
		<i>pseg_addr</i> : <i>Disp6</i> [BP]			
		DSR: <i>Disp6</i> [BP]	4	2	
		Rd: <i>Disp6</i> [BP]			
		<i>Disp6</i> [FP]	3	2	1
		<i>pseg_addr</i> : <i>Disp6</i> [FP]			
		DSR: <i>Disp6</i> [FP]	4	2	
		Rd: <i>Disp6</i> [FP]			
		<i>Dadr</i>	2	2	1
		<i>pseg_addr</i> : <i>Dadr</i>			
		DSR: <i>Dadr</i>	3	2	
		Rd: <i>Dadr</i>			
	QRn	[EA]	8	8	
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	9	8	
		Rd: [EA]			
		[EA+]	8	8	
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	9	8	
		Rd: [EA+]			

第3章 命令の詳細 インストラクションセット

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレ ッシングの影響
L	Rn	[EA]	1	1	
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	2	1	
		Rd: [EA]			
		[EA+]	1	1	
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	2	1	
		Rd: [EA+]			
		[ERm]	1	1	1
		<i>pseg_addr</i> : [ERm]			
		DSR: [ERm]	2	1	
		Rd: [ERm]			
		<i>Disp16</i> [ERm]	2	1	1
		<i>pseg_addr</i> : <i>Disp16</i> [ERm]			
		DSR: <i>Disp16</i> [ERm]	3	1	
		Rd: <i>Disp16</i> [ERm]			
		<i>Disp6</i> [BP]	2	1	1
		<i>pseg_addr</i> : <i>Disp6</i> [BP]			
		DSR: <i>Disp6</i> [BP]	3	1	
		Rd: <i>Disp6</i> [BP]			
		<i>Disp6</i> [FP]	2	1	1
		<i>pseg_addr</i> : <i>Disp6</i> [FP]			
		DSR: <i>Disp6</i> [FP]	3	1	
		Rd: <i>Disp6</i> [FP]			
		<i>Dadr</i>	2	1	1
		<i>pseg_addr</i> : <i>Dadr</i>			
		DSR: <i>Dadr</i>	3	1	
		Rd: <i>Dadr</i>			
	XRn	[EA]	4	4	
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	5	4	
		Rd: [EA]			
		[EA+]	4	4	
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	5	4	
		Rd: [EA+]			

記号	第1オペランド	第2オペランド	最小実行サイクル	RomWindowアクセス	[EA+] アドレッシングの影響
LEA	[ERm]		1		
	<i>Disp16</i> [ERm]		2		
	<i>Dadr</i>		2		
MOV	CERn	[EA]	2	2	1
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	3	2	
		Rd: [EA]			
		[EA+]	2	2	1
		<i>pseg_addr</i> : [EA+]			
	CQRn	DSR: [EA+]	3	2	
		Rd: [EA+]			
		[EA]	8	8	1
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	9	8	
		Rd: [EA]			
	CRn	[EA+]	8	8	1
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	9	8	
		Rd: [EA+]			
	CRn	[EA]	1	1	1
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	2	1	
		Rd: [EA]			
	CXRn	[EA+]	1	1	1
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	2	1	
		Rd: [EA+]			
	CRn	Rm	1		
	CXRn	[EA]	4	4	1
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	5	4	
		Rd: [EA]			
	ECSR	[EA+]	4	4	1
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	5	4	
		Rd: [EA+]			
	ELR	ERm	3		
	EPSW	Rm	1		
	ERn	ELR	3		
		ERm	2		
		#imm7	2		
		SP	2		

第3章 命令の詳細 インストラクションセット

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレッシングの影響
MOV	[EA]	CERm	2	2	1
	<i>pseg_addr</i> : [EA]				
	DSR: [EA]	CERm	3	2	
	Rd: [EA]				
	[EA+]	CERm	2	2	1
	<i>pseg_addr</i> : [EA+]				
	DSR: [EA+]	CERm	3	2	
	Rd: [EA+]				
	[EA]	CQRm	8	8	1
	<i>pseg_addr</i> : [EA]				
	DSR: [EA]	CQRm	9	8	
	Rd: [EA]				
	[EA+]	CQRm	8	8	1
	<i>pseg_addr</i> : [EA+]				
	DSR: [EA+]	CQRm	9	8	
	Rd: [EA+]				
	[EA]	CRm	1	1	1
	<i>pseg_addr</i> : [EA]				
	DSR: [EA]	CRm	2	1	
	Rd: [EA]				
	[EA+]	CRm	1	1	1
	<i>pseg_addr</i> : [EA+]				
	DSR: [EA+]	CRm	2	1	
	Rd: [EA+]				
	[EA]	CXRm	4	4	1
	<i>pseg_addr</i> : [EA]				
	DSR: [EA]	CXRm	5	4	
	Rd: [EA]				
	[EA+]	CXRm	4	4	1
	<i>pseg_addr</i> : [EA+]				
	DSR: [EA+]	CXRm	5	4	
	Rd: [EA+]				
	PSW	Rm	1		
		<i>#unsigned8</i>	1		
	Rn	CRm	1		
		ECSR	2		
		EPSW	2		
		PSW	1		
		Rm	1		
		<i>#imm8</i>	1		
	SP	ERm	1		1

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレッシングの影響
MUL	ER n	R m	9		
NEG	R n		1		
NOP			1		
OR	R n	R m	1		
		#imm8	1		
POP	EA		4		1
	EA,LR		6 / 8 ^(*)		1
	EA,PC		8 / 9 ^(*)		1
	EA,PC,LR		10 / 13 ^(*)		1
	EA,PC,PSW		10 / 11 ^(*)		1
	EA,PC,PSW,LR		12 / 15 ^(*)		1
	EA,PSW		6		1
	EA,PSW,LR		8 / 10 ^(*)		1
	LR		2 / 4 ^(*)		1
	LR,PSW		4 / 6 ^(*)		1
	PC		4 / 5 ^(*)		1
	PC,LR		6 / 9 ^(*)		1
	PC,PSW		6 / 7 ^(*)		1
	PC,PSW,LR		8 / 11 ^(*)		1
	PSW		2		1
	ER n		2		1
	QR n		8		1
	R n		2		1
	XR n		4		1

(*) ... (メモリモデルが SMALL の時のサイクル/ LARGE の時のサイクル)

第3章 命令の詳細 インストラクションセット

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレッシングの影響
PUSH	EA		2		1
	ELR		2 / 4 ^(*)		1
	EA,ELR		4 / 6 ^(*)		1
	EPSW		2		1
	EPSW,EA		4		1
	EPSW,ELR		4 / 6 ^(*)		1
	EPSW,ELR,EA		6 / 8 ^(*)		1
	LR		2 / 4 ^(*)		1
	LR,EA		4 / 6 ^(*)		1
	LR,ELR		4 / 8 ^(*)		1
	LR,EA,ELR		6 / 10 ^(*)		1
	LR,EPSW		4 / 6 ^(*)		1
	LR,EPSW,EA		6 / 8 ^(*)		1
	LR,EPSW,ELR		6 / 10 ^(*)		1
	LR,ELR,EPSW,EA		8 / 12 ^(*)		1
	ER _n		2		1
	QR _n		8		1
	R _n		2		1
	XR _n		4		1
RB	<i>Dbitadr</i>		2		1
	<i>pseg_addr: Dbitadr</i>				
	DSR: <i>Dbitadr</i>		3		
	Rd: <i>Dbitadr</i>				
	Rn. <i>bit_offset</i>		1		
RC			1		
RT			2		1
RTI			2		1
SB	<i>Dbitadr</i>		2		1
	<i>pseg_addr: Dbitadr</i>				
	DSR: <i>Dbitadr</i>		3		
	Rd: <i>Dbitadr</i>				
	Rn. <i>bit_offset</i>		1		
SC			1		

(*) ... (メモリモデルが SMALL の時のサイクル/ LARGE の時のサイクル)

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレッシングの影響
SLL	Rn	Rm $\#width$	1 1		1 1
SLLC	Rn	Rm $\#width$	1 1		1 1
SRA	Rn	Rm $\#width$	1 1		1 1
SRL	Rn	Rm $\#width$	1 1		1 1
SRLC	Rn	Rm $\#width$	1 1		1 1
ST	ERn	[EA]	2		
		$pseg_addr:[EA]$			
		DSR:[EA]	3		
		Rd:[EA]			
		[EA+]	2		
		$pseg_addr:[EA+]$			
		DSR:[EA+]	3		
		Rd:[EA+]			
		[ERm]	2		1
		$pseg_addr:[ERm]$			
		DSR:[ERm]	3		
		Rd:[ERm]			
		$Disp16[ERm]$	3		1
		$pseg_addr: Disp16[ERm]$			
		DSR: $Disp16[ERm]$	4		
		Rd: $Disp16[ERm]$			
		$Disp6[BP]$	3		1
		$pseg_addr: Disp6[BP]$			
		DSR: $Disp6[BP]$	4		
		Rd: $Disp6[BP]$			
		$Disp6[FP]$	3		1
		$pseg_addr: Disp6[FP]$			
		DSR: $Disp6[FP]$	4		
		Rd: $Disp6[FP]$			
		$Dadr$	2		1
		$pseg_addr: Dadr$			
		DSR: $Dadr$	3		
		Rd: $Dadr$			

第3章 命令の詳細 インストラクションセット

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレッシングの影響
ST	QR <i>n</i>	[EA]	8		
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	9		
		Rd: [EA]			
		[EA+]	8		
		<i>pseg_addr</i> : [EA+]			
	R <i>n</i>	DSR: [EA+]	9		
		Rd: [EA+]			
		[EA]	1		
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	2		
		Rd: [EA]			
		[EA+]	1		
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	2		
		Rd: [EA+]			
		[ER <i>m</i>]	1		1
		<i>pseg_addr</i> : [ER <i>m</i>]			
		DSR: [ER <i>m</i>]	2		
		Rd: [ER <i>m</i>]			
		<i>Disp16</i> [ER <i>m</i>]	2		1
		<i>pseg_addr</i> : <i>Disp16</i> [ER <i>m</i>]			
		DSR: <i>Disp16</i> [ER <i>m</i>]	3		
		Rd: <i>Disp16</i> [ER <i>m</i>]			
		<i>Disp6</i> [BP]	2		1
		<i>pseg_addr</i> : <i>Disp6</i> [BP]			
		DSR: <i>Disp6</i> [BP]	3		
		Rd: <i>Disp6</i> [BP]			
		<i>Disp6</i> [FP]	2		1
		<i>pseg_addr</i> : <i>Disp6</i> [FP]			
		DSR: <i>Disp6</i> [FP]	3		
		Rd: <i>Disp6</i> [FP]			
		<i>Dadr</i>	2		1
		<i>pseg_addr</i> : <i>Dadr</i>			
		DSR: <i>Dadr</i>	3		
		Rd: <i>Dadr</i>			
	XR <i>n</i>	[EA]	4		
		<i>pseg_addr</i> : [EA]			
		DSR: [EA]	5		
		Rd: [EA]			
		[EA+]	4		
		<i>pseg_addr</i> : [EA+]			
		DSR: [EA+]	5		
		Rd: [EA+]			

ニーモニック	第1 オペランド	第2 オペランド	最小実行 サイクル	RomWindow アクセス	[EA+] アドレッシングの影響
SUB	Rn	Rm	1		
SUBC	Rn	Rm	1		
SWI	$\#snum$		3		1
TB	$Dbitadr$		2	1	1
	$pseg_addr: Dbitadr$				
	DSR: $Dbitadr$		3	1	
	$Rd: Dbitadr$				
	$Rn.bit_offset$		1		
XOR	Rn	Rm	1		
		$\#imm8$	1		

3.4各命令の説明

次ページより各命令の詳細について説明します。命令はアルファベット順に並べられており、ひとつの命令を1ページあるいは2ページで説明しています。命令フォーマットに記載されていないコードを、命令として実行した場合のCPU動作は保証できませんのでプログラマはご注意ください。以下に命令の説明の見方の例を示します。

命令の一般的な記述形式

ニーモニックおよびオペランドの一般系を略号を交えて表現します。

命令動作のメタ記述

命令動作を簡略な記号を用いて簡潔に表現します。

機能

SB Dbitadr

Z ← ~[Dbitadr]
[Dbitadr] ← 1

説明

命令の詳細な説明

動作の詳細とオペランドの記述および制限事項など。

- Dbitadr で指定されたメモリの1ビットの状態を調べ、その結果をZフラグに反映した後、そのビットをセットする命令です。
- Dbitadr は、ビットセットするメモリのアドレスを、Dadr16.bit の形式で記述します。
- bit は、0-7 までの整数で、セットするビットの、ビット位置を記述します。

フラグ

フラグ変化表

命令実行により変化するフラグの状態を下記のように分類します。
* 結果に従い変化
- 変化無し

C	Z	S	OV	MIE	HC
-	*	-	-	-	-

Z : ビットの内容が0の時1になり、それ以外の時は0になります。
- : 変化はありません。

命令フォーマット

命令コード表

オペランドに許されるアドレッシングの組み合わせと命令コードを示します

ニーモニック	第1オペランド	DSRプリフィックスコード	命令フォーマット			
			第1ワード		第2ワード	
SB	Dbitadr		A	0 1 bit 0	Dadr	
	*: Dbitadr	<word>	A	0 1 bit 0	Dadr	

DSR プリフィックスの命令コード表

直前の命令コード表中の、第1オペランド部の * と指定された個所に記述できます。

*	<word>			
pseg_addr	E	3	pseg_addr	
DSR	F	E	9	F
Rd	9	0	d	F

ADD ERn, ERm

加算命令

機能

$ERn \leftarrow ERn + ERm$

説明

- ERn レジスタの内容に、 ERm の内容を加算し、結果を ERn レジスタに格納します。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット 15 よりキャリが発生したときに 1 になり、それ以外の時は 0 になります。
- Z : 実行結果が 0 の時 1 になり、それ以外の時は 0 になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローが生じた場合 1 になり、それ以外の時は 0 になります。
- HC : ビット 11 にキャリあるいはボローが生じた時 1 になります。それ以外の時は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット			
			第 1 ワード			第 2 ワード
ADD	ERn	ERm	F	n	m	6

ADD ERn , #imm7

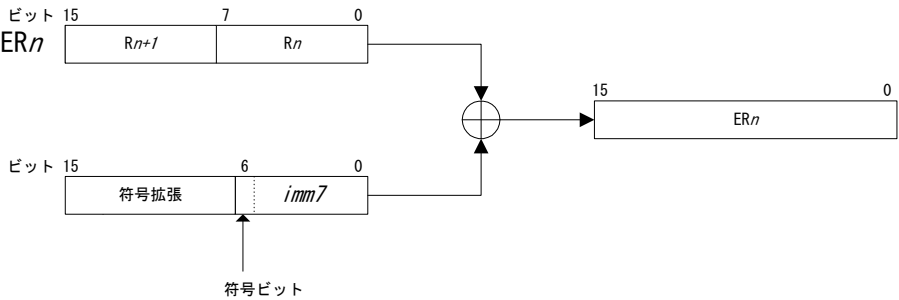
加算命令

機能

$$ERn \leftarrow ERn + (\text{signed})imm7$$

説明

・ERn レジスタの内容に、imm7 を符号拡張して加算し、結果を ERn レジスタに格納します。以下に計算手順を図示します。



フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット 15 よりキャリが発生したときに 1 になり、それ以外の時は 0 になります。
- Z : 実行結果が 0 の時 1 になり、それ以外の時は 0 になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローが生じた場合 1 になり、それ以外の時は 0 になります。
- HC : ビット 11 にキャリあるいはボローが生じた時 1 になります。それ以外の時は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
ADD	ERn	#imm7	E n	imm7

ADD Rn, obj

加算命令

機 能

$Rn \leftarrow Rn + obj$

説 明

・ Rn レジスタの内容に、 obj の内容を加算し、結果を Rn レジスタに格納します。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット7よりキャリが発生したときに1になり、それ以外の時は0になります。
- Z : 実行結果が0の時1になり、それ以外の時は0になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローがセットされた時1になり、それ以外の時は0になります。
- HC : ビット3にキャリあるいはボローが生じた時1になります。それ以外の時は0になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
ADD	Rn	Rm	8	n	m	1
		$\#imm8$	1	n	$imm8$	

ADD SP, #signed8

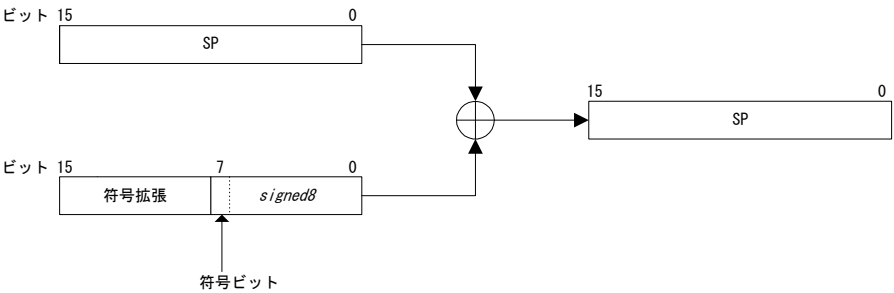
加算命令

機 能

$SP \leftarrow SP + signed\ 8$

説 明

- スタックポインタの内容に、*signed8* で示された即値を加算し、結果を SP に格納します。
- *signed8* はビット 7 を符号とする、-128 ～ +127 で示される範囲の整数値となります。計算手順を以下に図示します。



フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
ADD	SP	#signed8	E 1 signed8	

ADDC Rn, obj

キャリ付き加算命令

機能

$$Rn \leftarrow Rn + obj + C$$

説明

- Rn レジスタの内容に、 obj の内容と C フラグの内容を加算し、結果を Rn レジスタに格納します。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット7よりキャリが発生した時に1になり、それ以外の時は0になります。
- Z : 実行前のZの内容が0の時は、実行結果に関わらず0になります。
実行前のZの内容が1の時は、実行結果が0の時1になり、それ以外の時は0になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローがセットされた時1になり、それ以外の時に0になります。
- HC : ビット3にキャリ、あるいボローが生じた時1になります。それ以外の時は0になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
ADDC	Rn	Rm	8 n m 6	
		$\#imm8$	6 n $imm8$	

AND *Rn* , *obj*

論理積命令

機能

$Rn \leftarrow Rn \& obj$

説明

- *Rn* レジスタの内容と *obj* の内容の論理積をとり、結果を *Rn* レジスタに格納します。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : 実行結果が0の時1になり、それ以外の時は0になります。
S : 実行の結果の最上位ビットがセットされます。
— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
AND	<i>Rn</i>	<i>Rm</i>	8 <i>n</i> <i>m</i> 2	
		<i>#imm8</i>	2 <i>n</i> <i>imm8</i>	

B *Cadr*

ダイレクトジャンプ命令

機能

CSR \leftarrow *Cadr*[19:16]
PC \leftarrow *Cadr*[15:0]

説明

- ・プログラムメモリ空間の任意のアドレスにジャンプします。*Cadr* には分岐先アドレスを指定します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	命令フォーマット			
		第1ワード		第2ワード	
B	<i>Cadr</i>	F	<i>Cadr</i> [19:16]	0	0
					<i>Cadr</i> [15:0]

B *ERn*

インダイレクトジャンプ命令

機 能

PC ← *ERn*

説 明

- *ERn* レジスタの内容を飛び先アドレスとする、同一セグメント内の任意のアドレスにジャンプします。
- *ERn* は、ワード長のレジスタの内容です。*ERn* レジスタにはこの命令に先立って飛び先アドレスを設定しておく必要があります。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	命令フォーマット			
		第1ワード			
B	<i>ERn</i>	F	0	<i>n</i>	2

Bcond Radr BC cond , Radr

条件分岐命令

機 能

if (*cond* = 真) then PC ← *Radr*
ただし、(NextPC -128word) ≤ *Radr* ≤ (NextPC +127word)
NextPC: 次の命令の先頭アドレス

説 明

- *cond* で指定される条件が真であれば、*Radr* で指定されるアドレスへジャンプする命令です。
- 条件は、PSW に残されているフラグの状態を示します。従って、この命令に先立って、比較命令などの PSW に結果を残す命令が行われることを前提とし、この命令でその結果の評価を行います。
- *cond* をオペランドとして記述する方法と、ニーモニック文字列の一部として記述する方法があります。

例)

```
CMP    R0,#12H
BEQ    LABEL    ; cond を、ニーモニック文字列の一部として記述
CMP    R0,#56H
BC     NC,LABEL ; cond をオペランドとして記述
      :
```

LABEL:

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

第3章 命令の詳細 インストラクションセット

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット		
			第1ワード		
<i>Bcond</i>	<i>Radr</i>		<i>C</i>	<i>Condition</i>	<i>(Radr -NextPC)>>1</i>
<i>BC</i>	<i>cond</i>	<i>Radr</i>			

Condition

命令記述				
<i>Bcond</i>	<i>BC cond</i>	<i>Condition</i>	意味	フラグ条件
BGE	BC GE	0000	符号無し \geq	C=0
BNC	BC NC			
BLT	BC LT	0001	符号無し $<$	C=1
BCY	BC CY			
BGT	BC GT	0010	符号無し $>$	(C=0)&&(Z=0)
BLE	BC LE	0011	符号無し \leq	(Z=1) (C=1)
BGES	BC GES	0100	符号あり \geq	(OV^S)=0
BLTS	BC LTS	0101	符号あり $<$	(OV^S)=1
BGTS	BC GTS	0110	符号あり $>$	((OV^S) Z) = 0
BLES	BC LES	0111	符号あり \leq	((OV^S) Z) = 1
BNE	BC NE	1000	\neq	Z=0
BNZ	BC NZ			
BEQ	BC EQ	1001	$=$	Z=1
BZ	BC ZF			
BNV	BC NV	1010	オーバフロー無し	OV=0
BOV	BC OV	1011	オーバフローあり	OV=1
BPS	BC PS	1100	正	S=0
BNS	BC NS	1101	負	S=1
BAL	BC AL	1110	無条件	

BL Cadr

ブランチアンドリンク
命令

機 能

LR	← 次の命令の先頭アドレス
LCSR	← CSR
CSR	← Cadr[19:16]
PC	← Cadr[15:0]

説 明

- ・サブルーチン用リンクレジスタ（LR）に次の PC の値を、サブルーチン用 CSR 退避レジスタ（LCSR）に現在の CSR の値をストアした後、オペランド内で指定された任意のセグメントの任意のアドレスにジャンプします。Cadr には飛び先アドレスを記述します。
- ・この命令はサブルーチンの読み出しを行うために使われます。そのサブルーチンから復帰する場合は RT 命令を使用します。
- ・サブルーチンがネストする場合は、上位サブルーチンの先頭に PUSH 命令を配置し、現在の LR および LCSR の内容をメモリに退避する必要があります。この場合、サブルーチンから復帰する場合には、POP 命令を使用します。
- ・割込みルーチン内で本命令を使用する場合は、割込みの先頭番地で PUSH 命令を配置し、現在の LR、および LCSR の内容を、スタックメモリに退避する必要があります。この場合、サブルーチンから復帰する場合には、POP 命令を使用します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

—：変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	命令フォーマット			
		第 1 ワード	第 2 ワード		
BL	Cadr	F Cadr[19:16] 0 1	Cadr[15:0]		

BL ER_n

ブランチアンドリンク
命令

機 能

PC	← ER _n
LR	← 次の命令の先頭アドレス
LCSR	← CSR

説 明

- ・ サブルーチン用リンクレジスタ (LR) に次の PC の値をストアし、ER_n レジスタの内容を飛び先とする同一セグメント内の任意のアドレスにジャンプする命令です。
- ・ この命令はサブルーチンの読み出しを行うために使われます。そのサブルーチンから復帰する場合は RT 命令を使用します。
- ・ サブルーチンがネストする場合は、上位サブルーチンの先頭に PUSH 命令を配置し、現在の LR および LCSR の内容をメモリに退避する必要があります。この場合、サブルーチンから復帰する場合には、POP 命令を使用します。
- ・ 割込みルーチン内で本命令を使用する場合は、割込みの先頭番地で PUSH 命令を配置し、現在の LR、および LCSR の内容を、スタックメモリに退避する必要があります。この場合、サブルーチンから復帰する場合には、POP 命令を使用します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	命令フォーマット			
		第 1 ワード			
BL	ER _n	F	0	n	3

BRK

ブレーク命令
(ソフトウェアリセット)

機 能

- ELEVEL の値が 2 以上の場合
システムリセット
- ELEVEL の値が 1 以下の場合
 - ELR2 ← 次の命令の先頭アドレス
 - ECSR2 ← CSR
 - EPSW2 ← PSW
 - ELEVEL ← 2
 - PC ← (Vector table 0004H)

説 明

- ソフトウェアでシステムをリセットする命令です。
- ELEVEL が 2 以上の状態でこの命令が実行された場合、CPU は以下の手順でシステムリセット処理を行います。
 - ① CPU 内部のレジスタを全て初期化。
 - ② コード空間上の 0 番地から始まるワードデータを SP に転送
 - ③ コード空間上の 2 番地から始まるワードデータを PC に転送。
- ELEVEL が 1 以下の状態でこの命令が実行された場合は、NMI 割込み相当の動作を行った後、コード空間上の割込みベクタテーブルの 4 番地からのワードデータを PC に転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	命令フォーマット			
		第 1 ワード			
BRK		F	F	F	F

CMP ER_n , ER_m

比較命令

機 能

$ER_n - ER_m$

説 明

- ER_n レジスタの内容と、 ER_m の内容を比較します。
- 実際には ER_n の内容から ER_m の内容を減算し、その結果で各フラグを設定します。この結果は条件分岐などによって参照できます。
- ER_n の内容は変化しません。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット 15 にボローが発生したとき 1 になり、それ以外の時は 0 になります。
- Z : 実行結果が 0 の時 1 になり、それ以外の時は 0 になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローがセットされた時 1 になり、それ以外の時は 0 になります。
- HC : ビット 11 にキャリあるいはボローが生じた時 1 になります。それ以外の時は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット			
			第 1 ワード		第 2 ワード	
CMP	ER_n	ER_m	F	n	m	7

CMP Rn, obj

比較命令

機能

$Rn - obj$

説明

- Rn レジスタの内容と、 obj の内容を比較します。
- 実際には Rn の内容から obj の内容を減算し、その結果で各フラグを設定します。
この結果は条件分岐などによって参照できます。
- Rn の内容は変化しません。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット7にボローが発生したとき1になり、それ以外の時は0になります。
- Z : 実行結果が0の時1になり、それ以外の時は0になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローがセットされた時1になり、それ以外の時は0になります。
- HC : ビット3にキャリあるいはボローが発生した時1になります。それ以外の時は0になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
CMP	Rn	Rm	8	n	m	7
		$\#imm8$	7	n	$imm8$	

CMPC *Rn*, *obj*

キャリ付き比較命令

機能

Rn - *obj* - C

説明

- *Rn* レジスタの内容と、*obj* の内容をキャリ付きで比較します。
- 実際には *Rn* の内容から (*obj* + キャリ) の内容を減算し、その結果で各フラグを設定します。この結果は条件分岐などによって参照できます。
- *Rn* の内容は変化しません。
- この命令を CMP 命令の後に続けて実行することで、多バイトデータの比較が可能になります。

例) CMP R0, R4
 CMPC R1, R5 (ER0 と ER4 の比較完了)

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット 7 にボローが発生したとき 1 になり、それ以外の時は 0 になります。
- Z : 実行前の Z の内容が 0 の時は、実行結果に関わらず 0 になります。
 実行前の Z の内容が 1 の時は、実行結果が 0 の時 1 になり、それ以外の時は 0 になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローがセットされた時 1 になり、それ以外の時は 0 になります。
- HC : ビット 3 にキャリあるいはボローが生じた時 1 になります。それ以外の時は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
CMPC	<i>Rn</i>	<i>Rm</i>	8 <i>n</i> <i>m</i> 5	
		<i>#imm8</i>	5 <i>n</i> <i>imm8</i>	

CPLC

キャリ反転命令

機能

$C \leftarrow \sim C$

説明

・Cフラグの内容を反転します。

フラグ

C	Z	S	OV	MIE	HC
*	—	—	—	—	—

C : 反転した内容がセットされます。
— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
CPLC			F	E	C	F

DAA Rn

バイト型 10 進加算補正命令

機能

$Rn \leftarrow (10 \text{ 進補正})Rn$

説明

- Rn の内容を 2 桁の BCD (2 進化 10 進) に変換します。実際には直前の Rn、C、および HC の値を参照し、下表に従って Rn の内容を補正します。x はドントケアであることを示します。

C	Rn[7:4] の値	HC	Rn[3:0] の値	加算値	補正後の C
0	0-9	0	0-9	00	0
0	0-8	0	A-F	06	0
0	0-9	1	x	06	0
0	A-F	0	0-9	60	1
0	9-F	0	A-F	66	1
0	A-F	1	x	66	1
1	x	0	0-9	60	1
1	x	0	A-F	66	1
1	x	1	x	66	1

- 実行に先立って ADD Rn,obj により 2 進加算命令が実行され、その結果が Rn と PSW に残っている必要があります。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	—	—	*

- C : 命令実行前のキャリが 1 の場合は 1 になります。
そうでない場合は、10 進補正の結果、100 の位への桁上げがあれば 1 になり、なければ 0 になります。
- Z : 実行結果が 0 の時 1 になり、それ以外の時は 0 になります。
- S : 実行の結果の最上位ビットがセットされます。
- HC : ビット 3 にキャリあるいはボローが生じた時 1 になり、それ以外の時は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット			
			第 1 ワード		第 2 ワード	
DAA	Rn		8	n	1	F

DAS Rn

バイト型 10 進減算補正命令

機 能

$Rn \leftarrow (10 \text{ 進補正})Rn$

説 明

- Rn の内容を 2 桁の BCD (2 進化 10 進) に変換します。実際には直前の Rn 、 C 、および HC の値を参照し、下表に従って Rn の内容を補正します。x はドントケアであることを示します。

C	$Rn[7:4]$ の値	HC	$Rn[3:0]$ の値	減算値
0	0-9	0	0-9	00
0	0-9	0	A-F	06
0	0-9	1	x	06
0	A-F	0	0-9	60
0	A-F	1	x	66
0	A-F	0	A-F	66
1	x	0	0-9	60
1	x	1	x	66
1	x	0	A-F	66

- 実行に先立って SUB Rn,obj により 2 進減算命令が実行され、その結果が Rn と PSW に残っている必要があります。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	—	—	*

- C : 命令実行前のキャリが 1 の場合は 1 になります。
そうでない場合、10 進減算補正の結果 100 の位へのボローがあれば 1 になり、なければ 0 になります。
- Z : 実行結果が 0 の時 1 になり、それ以外の時は 0 になります。
- S : 実行の結果の最上位ビットがセットされます。
- HC : ビット 3 にキャリあるいはボローが生じた時 1 になります。それ以外の時は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
DAS	Rn		8 n 3 F	

DEC [EA]

デクリメント命令
(EA 間接)

機能

[EA] ← [EA] -1

説明

- EA レジスタが指すメモリの内容を-1 します。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	*	—	*

- Z : 実行結果が 0 の時 1 になり、それ以外の場合は 0 になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローがセットされた時 1 になり、それ以外の場合は 0 になります。
- HC : ビット 3 にキャリあるいはボローが生じた時 1 になります。それ以外の場合は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	DSR プリフィックスコード	命令フォーマット			
				第 1 ワード		第 2 ワード	
DEC	[EA]			F	E	3	F
	*:[EA]		<word>	F	E	3	F

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

DI

割込みマスク命令

機能

MIE ← 0

説明

・ MIE を 0 にして、マスカブル割込みを禁止する命令です。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	*	—

MIE : 0 になります。

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
DI			E	B	F	7

DIV ERn, Rm

除算命令

機能

$$ERn \leftarrow ERn / Rm$$
$$Rm \leftarrow ERn \bmod Rm$$

説明

- ・ワード型レジスタ ERn とバイト型レジスタ Rm の除算を行い、商 16 ビットと剰余 8 ビットを得る命令です。被除数はワード型レジスタ ERn で、除数はバイト型レジスタ Rm です。演算の結果、 ERn に商が、 Rm に剰余が入ります。
- ・0 で除算を行った場合は、キャリがセットされ、 ERn 、 Rm の値は不定となります。

フラグ

C	Z	S	OV	MIE	HC
*	*	—	—	—	—

- C : 0 による除算を行った場合 1、それ以外で 0 になります。
- Z : 商が 0 で 1、それ以外で 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット			
			第 1 ワード		第 2 ワード	
DIV	ERn	Rm	F	n	m	9

EI

割込み許可命令

機 能

MIE ← 1

説 明

- ・ MIE を 1 にし、マスカブル割込みを許可する命令です。
- ・ MIE は、この命令の実行サイクルを含めて 3 サイクル後に 1 になります。従って MIE が 0 の状態で EI 命令を実行しても、その直後の 2 サイクルはマスカブル割込みの禁止状態が継続しますので、アプリケーションプログラム設計時に注意を払う必要があります。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	*	—

MIE : 1 になります。
— : 変化はありません。

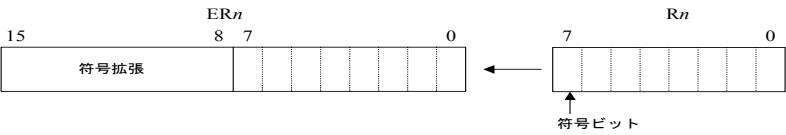
命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット			
			第 1 ワード		第 2 ワード	
EI			E	D 0 8		

EXTBW ERn

符号拡張

機 能



説 明

- Rn レジスタの内容を符号付きで 16 ビットに拡張し、結果を ERn レジスタに反映します。
- 動作としては、 $Rn+1$ の内容を Rn のビット 7 で埋めます。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : Rn の内容が 0 の時 1 になり、それ以外の場合は 0 になります。
- S : Rn のビット 7 の内容がセットされます。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
EXTBW	ERn		8 $n+1$ n F	

INC [EA]

インクリメント命令
(EA 間接)

機能

[EA] ← [EA] + 1

説明

・EA レジスタが指すメモリの内容を+1 します。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	*	—	*

- Z : 実行結果が 0 の時 1 になり、それ以外の場合は 0 になります。
S : 実行の結果の最上位ビットがセットされます。
OV : オーバフローがセットされた時 1 になり、それ以外の場合は 0 になります。
HC : ビット 3 にキャリあるいはボローが生じた時 1 になります。それ以外の場合は 0 になります。
— : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	DSR プリフィックスコード	命令フォーマット			
				第 1 ワード		第 2 ワード	
INC	[EA]			F	E 2	F	
	*:[EA]		<word>	F	E 2	F	

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

L ERn, obj

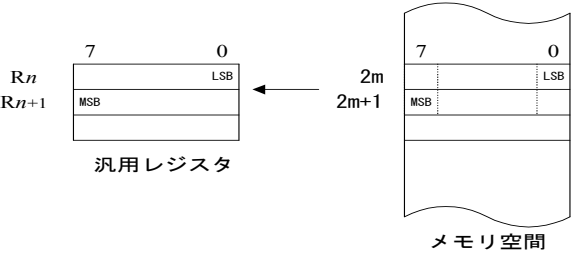
ワード型転送命令

機 能

$ERn \leftarrow obj$

説 明

- ERn レジスタに、obj をアドレスとするメモリの内容をワード長で転送します。



フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : ERn が 0 の時 1 になり、それ以外の場合は 0 になります。
S : 最上位ビットがセットされます。
— : 変化はありません。

命令フォーマット

次項に示します。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックス コード	命令フォーマット			
				第1ワード		第2ワード	
L	ER _n	[EA]		9	n	3	2
		*:[EA]	<word>	9	n	3	2
		[EA+]		9	n	5	2
		*:[EA+]	<word>	9	n	5	2
		[ER _m]		9	n	m	2
		*:[ER _m]	<word>	9	n	m	2
		Disp16[ER _m]		A	n	m	8
		*:Disp16[ER _m]	<word>	A	n	m	8
		Disp6[BP]		B	n	000	Disp6
		*:Disp6[BP]	<word>	B	n	000	Disp6
		Disp6[FP]		B	n	001	Disp6
		*:Disp6[FP]	<word>	B	n	001	Disp6
		Dadr		9	n	1	2
		*:Dadr	<word>	9	n	1	2

*	<word>			
pseg_addr	E	3	pseg_addr	
DSR	F	E	9	F
Rd	9	0	d	F

L QRn,obj

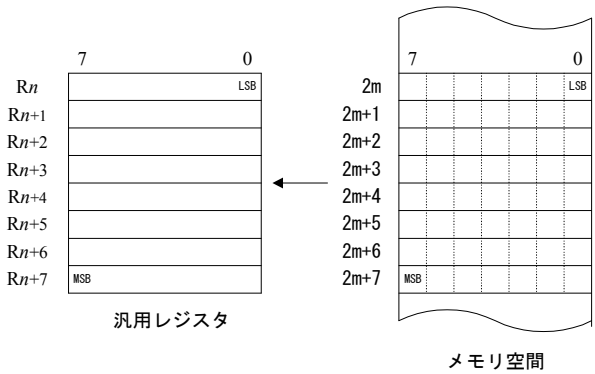
クワッドワード型
連続転送命令

機 能

$QRn \leftarrow obj$

説 明

- QRn レジスタに、obj で指定されるメモリデータをクワッドワード長で転送します。



フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : QRn が 0 の時 1 になり、それ以外の時は 0 になります。
S : 最上位ビットがセットされます。
— : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	DSR プリフィックスコード	命令フォーマット			
				第 1 ワード		第 2 ワード	
L	QRn	[EA]		9	n	3	6
		*:[EA]	<word>	9	n	3	6
		[EA+]		9	n	5	6
		*:[EA+]	<word>	9	n	5	6

*	<word>			
pseg_addr	E	3	pseg_addr	
DSR	F	E	9	F
Rd	9	0	d	F

L Rn, \textit{obj}

バイト型転送命令

機 能

$Rn \leftarrow \textit{obj}$

説 明

- ・ Rn レジスタに、 \textit{obj} で指定されるメモリデータをバイト長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : Rn が 0 の時 1 になり、それ以外の場合は 0 になります。
S : 最上位ビットがセットされます。
— : 変化はありません。

命令フォーマット

次頁に示します。

第3章 命令の詳細
 インストラクションセット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックスコード	命令フォーマット					
				第1ワード				第2ワード	
L	Rn	[EA]		9	n	3	0		
		*:[EA]	<word>	9	n	3	0		
		[EA+]		9	n	5	0		
		*:[EA+]	<word>	9	n	5	0		
		[ERm]		9	n	m	0		
		*:[ERm]	<word>	9	n	m	0		
		Disp16[ERm]		9	n	m	8	Disp16	
		*:Disp16[ERm]	<word>	9	n	m	8	Disp16	
		Disp6[BP]		D	n	00	Disp6		
		*:Disp6[BP]	<word>	D	n	00	Disp6		
		Disp6[FP]		D	n	01	Disp6		
		*:Disp6[FP]	<word>	D	n	01	Disp6		
		Dadr		9	n	1	0	Dadr	
		*:Dadr	<word>	9	n	1	0	Dadr	

*	<word>			
pseg_addr	E	3	pseg_addr	
DSR	F	E	9	F
Rd	9	0	d	F

L XRn, obj

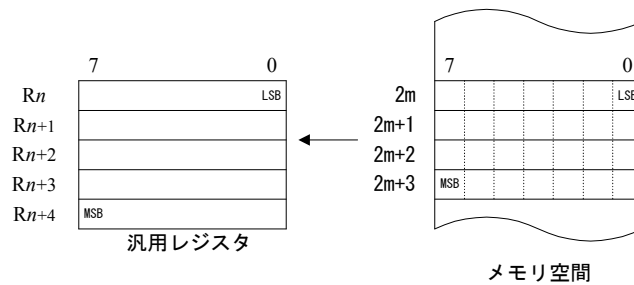
ダブルワード型
転送命令

機能

$XRn \leftarrow obj$

説明

• XRn レジスタに、 obj で示されるメモリデータをダブルワード長で転送します。



フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

Z : XRn が 0 の時 1 になり、それ以外の時は 0 になります。
 S : 最上位ビットがセットされます。
 — : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックス コード	命令フォーマット			
				第1ワード		第2ワード	
L	XRn	[EA]		9	n	3	4
		*:[EA]	<word>	9	n	3	4
		[EA+]		9	n	5	4
		*:[EA+]	<word>	9	n	5	4

*	<word>			
$pseg_addr$	E	3	$pseg_addr$	
DSR	F	E	9	F
Rd	9	0	d	F

LEA *obj*

EA への転送命令

機 能

EA ← *obj*

説 明

- ・ EA レジスタに、*obj* で示されたワード値をセットします。

フラグ

C	Z	S	OV	MIE	HC
－	－	－	－	－	－

－ ： 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
LEA	[ER <i>m</i>]		F	0	<i>m</i>	A
	<i>Dadr</i>		F	0	0	C
	<i>Disp16</i> [ER <i>m</i>]		F	0	<i>m</i>	B

MOV CERN, obj

コプロセッサ転送命令

機能

$CERN \leftarrow obj$

説明

- *obj* で示される内容を先頭とするメモリの内容を、*CERN* で示されるコプロセッサにワード長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックスコード	命令フォーマット			
				第1ワード		第2ワード	
MOV	CERN	[EA]		F	<i>n</i>	2	D
		*:[EA]	<word>	F	<i>n</i>	2	D
		[EA+]		F	<i>n</i>	3	D
		*:[EA+]	<word>	F	<i>n</i>	3	D

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

MOV CQRn , obj

コプロセッサ転送命令

機能

CQRn ←obj

説明

- ・obj で示される内容を先頭とするメモリの内容を、CQRn で示されるコプロセッサにクワッドワード長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSRプリフィックスコード	命令フォーマット			
				第1ワード		第2ワード	
MOV	CQRn	[EA]		F	n	6	D
		*:[EA]	<word>	F	n	6	D
		[EA+]		F	n	7	D
		*:[EA+]	<word>	F	n	7	D

*	<word>			
pseg_addr	E	3	pseg_addr	
DSR	F	E	9	F
Rd	9	0	d	F

MOV CRn, obj

コプロセッサ転送命令

機能

$CRn \leftarrow obj$

説明

- *obj* で示されるメモリの内容を、*CRn* で示されるコプロセッサにバイト長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックスコード	命令フォーマット			
				第1ワード		第2ワード	
MOV	CRn	[EA]		F	n	0	D
		*:[EA]	<word>	F	n	0	D
		[EA+]		F	n	1	D
		*:[EA+]	<word>	F	n	1	D

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

MOV CR*n*, R*m*

コプロセッサ転送命令

機能

CR*n* ←R*m*

説明

- ・CR*n* で示されるコプロセッサに、R*m* の結果を転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— ： 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
MOV	CR <i>n</i>	R <i>m</i>	A <i>n</i> <i>m</i> E	

MOV CXR*n*, *obj*

コプロセッサ転送命令

機 能

$CXRn \leftarrow obj$

説 明

- *obj* で示されるメモリの内容を、*CXRn* で示されるコプロセッサにダブルワード長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSRプリフィックスコード	命令フォーマット			
				第1ワード		第2ワード	
MOV	CXR <i>n</i>	[EA]		F	<i>n</i>	4	D
		*:[EA]	<word>	F	<i>n</i>	4	D
		[EA+]		F	<i>n</i>	5	D
		*:[EA+]	<word>	F	<i>n</i>	5	D

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

MOV ECSR, Rm

転送命令

機能

- ELEVEL が 0 の時
LCSR $\leftarrow Rm$
- ELEVEL が 0 以外の時
ECSR[ELEVEL] $\leftarrow Rm$

説明

- Rm の内容を、ELEVEL が 0 の場合は LCSR に、0 以外の場合は ECSR1-3 のいずれかに転送します。
- ELEVEL が 0 以外の場合に対象となる ECSR は、現在の ELEVEL の値を指数とする、いずれかひとつのレジスタです。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
MOV	ECSR	Rm	A	0	m	F

MOV ELR, ER_m

転送命令

機 能

- ELEVEL が 0 の時
LR ← ER_m
- ELEVEL が 0 以外の時
ELR[ELEVEL] ← ER_m

説 明

- ER_m の内容を、ELEVEL が 0 の場合は LR に転送し、0 以外の場合は ELR1-3 のいずれかのレジスタに転送します。
- ELEVEL が 0 以外の場合に対象となる ELR は、現在の ELEVEL の値を指数とする、いずれかひとつのレジスタです。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
MOV	ELR	ER _m	A	m	0	D

MOV EPSW , *Rm*

転送命令

機 能

- ELEVEL が 0 以外の場合
 EPSW[ELEVEL] ← *Rm*

説 明

- EPSW レジスタに、*Rm* レジスタの内容を転送します。
- 対象となる EPSW は、現在の ELEVEL の値を指数とする、いずれかひとつのレジスタです。ELEVEL が 0 の場合はレジスタへの書き込みを行わず、PC を次の命令の先頭へ進めるだけの動作となります。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
MOV	EPSW	<i>Rm</i>	A	0	<i>m</i>	C

MOV ER_n, ELR

転送命令

機 能

- ELEVEL が 0 の時
ER_n ← LR
- ELEVEL が 0 以外の時
ER_n ← ELR[ELEVEL]

説 明

- ER_n レジスタに、ELEVEL が 0 の場合は LR の内容を転送し、0 以外の場合は ELR1-3 のいずれかの内容を転送します。
- ELEVEL が 0 以外の場合、対象となる ELR は、現在の ELEVEL の値を指数とする、いずれかひとつのレジスタです。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
MOV	ER _n	ELR	A n 0 5	

MOV ERn , ERm

転送命令

機 能

ERn ←ERm

説 明

・ ERn レジスタに、ERm レジスタの内容を転送します。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : ERn が 0 の時 1 になり、それ以外の場合は 0 になります。
S : 実行の結果の最上位ビットがセットされます。
— : 変化はありません。

命令フォーマット

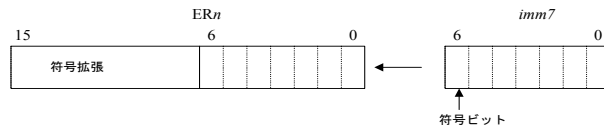
ニーモ ニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
MOV	ERn	ERm	F n m 5	

MOV ERn, #imm7

転送命令

機能

$ERn \leftarrow (\text{sign-extends})imm7$



説明

・ERn レジスタに、imm7 で示される内容を符号付きで転送します。具体的には、imm7 を Rn レジスタのビット 0-6 にライトし、imm7 のビット 6 の値で、Rn レジスタのビット 7 と、Rn+1 レジスタの全ビットを埋める動作となります。

実行例

```
MOV R0,#07Fh
MOV R1,#0h
MOV ER0,#-64 ; 実行の結果 1 が拡張され、R0=0C0h, R1=0FFH となります。
```

```
MOV R0,#03Fh
MOV R1,#0FFh
MOV ER0,#3Fh ; 実行の結果 0 が拡張され、R0=03fh, R1=0H となります。
```

フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

Z : ERn が 0 の時 1 になり、それ以外の場合は 0 になります。
S : 実行の結果の最上位ビットがセットされます。
— : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
MOV	ERn	#imm7	E n 0 imm7	

MOV ERn , SP

転送命令

機能

ERn ←SP

説明

- ・スタックポインタの内容を、ERn で示されるレジスタに転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— ： 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
MOV	ERn	SP	A : n : 1 : A	

MOV *obj*, CER*m*

コプロセッサ転送命令

機 能

(WORD) *obj* ←CER*m*

説 明

- ・ CER*m* で指定されるコプロセッサの内容を、現在の EA レジスタの内容を先頭アドレスとするメモリに、ワード長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックス コード	命令フォーマット			
				第1ワード		第2ワード	
MOV	[EA]	CER <i>m</i>		F	<i>m</i>	A	D
	*:[EA]	CER <i>m</i>	<word>	F	<i>m</i>	A	D
	[EA+]	CER <i>m</i>		F	<i>m</i>	B	D
	*:[EA+]	CER <i>m</i>	<word>	F	<i>m</i>	B	D

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

MOV *obj*, CQR*m*

コプロセッサ転送命令

機能

(QWORD)*obj* ← CQR*m*

説明

- ・ CQR*m* で指定されるコプロセッサの内容を、EA レジスタの内容を先頭アドレスとするメモリに、クワッドワード長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSRプリフィックスコード	命令フォーマット			
				第1ワード		第2ワード	
MOV	[EA]	CQR <i>m</i>		F	<i>m</i>	E	D
	*: [EA]	CQR <i>m</i>	<word>	F	<i>m</i>	E	D
	[EA+]	CQR <i>m</i>		F	<i>m</i>	F	D
	*: [EA+]	CQR <i>m</i>	<word>	F	<i>m</i>	F	D

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

MOV *obj*, CR*m*

コプロセッサ転送命令

機 能

(BYTE) *obj* ←CR*m*

説 明

- ・CR*m* で指定されるコプロセッサの内容を、EA レジスタの内容をアドレスとするメモリに、バイト長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックスコード	命令フォーマット			
				第1ワード		第2ワード	
MOV	[EA]	CR <i>m</i>		F	<i>m</i>	8	D
	*: [EA]	CR <i>m</i>	<word>	F	<i>m</i>	8	D
	[EA+]	CR <i>m</i>		F	<i>m</i>	9	D
	*: [EA+]	CR <i>m</i>	<word>	F	<i>m</i>	9	D

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
R <i>d</i>	9	0	<i>d</i>	F

MOV *obj*, CXR*m*

コプロセッサ転送命令

機能

(DOUBLE WORD) *obj* ← CXR*m*

説明

- ・CXR*m* で指定されるコプロセッサの内容を、EA レジスタの内容を先頭アドレスとするメモリに、ダブルワード長で転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSRプリフィックスコード	命令フォーマット			
				第1ワード		第2ワード	
MOV	[EA]	CXR <i>m</i>		F	<i>m</i>	C	D
	*: [EA]	CXR <i>m</i>	<word>	F	<i>m</i>	C	D
	[EA+]	CXR <i>m</i>		F	<i>m</i>	D	D
	*: [EA+]	CXR <i>m</i>	<word>	F	<i>m</i>	D	D

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
R <i>d</i>	9	0	<i>d</i>	F

MOV PSW , *obj*

転送命令

機 能

PSW ← *obj*

説 明

- ・ PSW に、*obj* で示されたバイト値の内容を転送します。
- ・ 本命令を使用して ELEVEL の値を書き換える場合、直後に NOP 命令を配置してください。

記述例

MOV PSW , #5Ah

NOP

RTI

NOP 命令を挿入しない場合、次に配置した命令が本命令で変化する前の ELEVEL を参照して実行されます。この動作がアプリケーションプログラムが誤動作する要因となる場合があります。

- ・ MIE の値を 1 から 0 に書き換える場合、本命令を使用せず、DI 命令を使用してください。本命令を使用した場合、実行終了後 1 サイクルの間、MIE が 1 の状態が継続します。これによりプログラムの意図せぬ割込みがアサートされる可能性があることが、アプリケーションプログラムが誤動作する要因となる場合があります。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	*	*

* : 対応するビットの値に変化します。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
MOV	PSW	#unsigned8	E	9	unsigned8	
		Rm	A	0	m	B

MOV Rn , CRm

コプロセッサ転送命令

機 能

$Rn \leftarrow CRm$

説 明

・ Rn レジスタに、コプロセッサの内容を 1 バイト転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— ： 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
MOV	Rn	CRm	A n m 6	

MOV R_n , ECSR

転送命令

機 能

- ELEVEL が 0 の時
 $R_n \leftarrow \text{LCSR}$
- ELEVEL が 0 以外の時
 $R_n \leftarrow \text{ECSR}[\text{ELEVEL}]$

説 明

- R_n レジスタに、ELEVEL が 0 の場合は LCSR の内容を転送し、0 以外の場合は ECSR1-3 のいずれかの内容を転送します。
- ELEVEL が 0 以外の場合、対象となる ECSR は、現在の ELEVEL の値を指数とするレジスタです。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット			
			第 1 ワード		第 2 ワード	
MOV	R_n	ECSR	A	n	0	7

MOV Rn, EPSW

転送命令

機能

- ELEVEL が 0 以外の時
 $Rn \leftarrow \text{EPSW}[\text{ELEVEL}]$

説明

- Rn レジスタに、EPSW の内容を転送します。
- 対象となる EPSW は、現在の ELEVEL の値を指数とするレジスタです。ELEVEL が 0 の場合は Rn レジスタへの書き込みを行わず、PC を次の命令の先頭へ進めるだけの動作となります。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
MOV	Rn	EPSW	A	n	0	4

MOV Rn, PSW

転送命令

機 能

$Rn \leftarrow \text{PSW}$

説 明

- ・ Rn レジスタに、PSW の内容を転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモ ニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
MOV	Rn	PSW	A n 0 3	

MOV *Rn* , *obj*

転送命令

機能

$Rn \leftarrow obj$

説明

- ・ *Rn* レジスタに、*obj* の内容を転送します。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : *Rn* が 0 の時 1 になり、それ以外の時は 0 になります。
S : 実行の結果の最上位ビットがセットされます。
— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
MOV	<i>Rn</i>	<i>Rm</i>	8 <i>n</i> <i>m</i> 0	
		<i>#imm8</i>	0 <i>n</i> <i>imm8</i>	

MOV SP, ERm

転送命令

機能

SP ← ERm

説明

- ・ ERm で示されるレジスタの内容をスタックポインタに転送します。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
MOV	SP	ERm	A : 1 : m : A	

MUL ER_n, R_m

乗算命令

機 能

$ER_n \leftarrow R_n * R_m$ (n は偶数)

説 明

- ・ バイト型レジスタ R_n と R_m の乗算を行い、積 16 ビットを得る命令です。被乗数は R_n の内容、乗数は R_m です。演算の結果、 ER_n に積が入ります。

フラグ

C	Z	S	OV	MIE	HC
—	*	—	—	—	—

- Z : 積がゼロの時 1 になり、それ以外の場合は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
MUL	ER_n	R_m	F n m 4	

NEG Rn

符号反転命令

機能

$Rn \leftarrow 0 - Rn$

説明

- ・バイト型レジスタ Rn の2の補数を取る命令です。結果は Rn に戻されます。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット7よりキャリが発生した時に1になり、それ以外の時は0になります。
- Z : 実行結果が0の時1になり、それ以外の時は0になります。
- S : 実行の結果の最上位ビットがセットされます。
- OV : オーバフローがセットされた時1になり、それ以外の時は0になります。
- HC : ビット3にキャリあるいボローが生じた時1になります。それ以外の時は0になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
NEG	Rn		8	n	5	F

NOP

ノーオペレーション命令

機能

No operation

説明

- ・プログラムカウンタを次に進めます。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
NOP			F E 8 F	

OR Rn, obj

論理和命令

機能

$$Rn \leftarrow Rn \mid obj$$

説明

- Rn レジスタの内容と obj で示される内容の論理 OR をとり、結果を Rn レジスタに格納します。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : 実行結果が 0 の時に 1 にセットされ、それ以外の時は 0 になります。
S : 実行の結果の最上位ビットがセットされます。
— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
OR	Rn	Rm	8 n m 3	
		$\#imm8$	3 n $imm8$	

POP レジスタリスト

コントロールレジスタ
復帰命令

機能

レジスタ群 ← (SP)
SP ← SP + n

説明

- スタックメモリに退避しているコントロールレジスタ群をレジスタリストで指定されたレジスタに復帰した後、スタックポインタ (SP) を指定したレジスタの個数に対応するバイト数だけインクリメントします。各々のレジスタが指定された時の実際のスタックの動作については、” 1.6 スタックの変化について” を参照してください。
- レジスタリストは、次のいずれかです。
 - ① プログラムステータスワード (PSW)
 - ② サブルーチン用 PC 退避レジスタ (LR)
 - ③ プログラムカウンタ (PC)
 - ④ EA レジスタ
- レジスタリストは、全て記述する必要はありません。ただし少なくとも一つのレジスタリストを記述する必要があります。
- オペランドのレジスタ記述は順不同ですが、復帰順は固定です。この命令を用いた場合のレジスタ復帰順は
EA → LR → PSW → PC となります。
- この命令実行中は、メモリ空間のワードバウンダリが発生しません。従って SP が奇数の場合も、そのアドレスから始まるデータが操作の対象になります。
- 通常、サブルーチンから復帰する場合は RT 命令を、割込みから復帰する場合は RTI 命令を使用しますが、サブルーチンがネストする場合、あるいは多重割込みが発生する可能性がある場合は、PUSH 命令で現在の退避レジスタの内容をスタックメモリに退避する必要があります。この場合 POP 命令を使用して、退避レジスタをコントロールレジスタに復帰させます。詳細は、“1.4 例外レベルと退避レジスタの取り扱いについて” を参照して下さい。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	*	*

- * : PSW の復帰が指定された場合は、対応するフラグが変化します。
PSW の復帰が指定されていない場合は変化ありません。

命令フォーマット

ニーモ ニック	第1 オペランド	命令フォーマット			
		第1 ワード			
POP	EA	F	1	8	E
	PC	F	2	8	E
	EA, PC	F	3	8	E
	PSW	F	4	8	E
	EA, PSW	F	5	8	E
	PC, PSW	F	6	8	E
	EA, PC, PSW	F	7	8	E
	LR	F	8	8	E
	EA, LR	F	9	8	E
	PC, LR	F	A	8	E
	EA, PC, LR	F	B	8	E
	LR, PSW	F	C	8	E
	EA, PSW, LR	F	D	8	E
	PC, PSW, LR	F	E	8	E
	EA, PC, PSW, LR	F	F	8	E

POP *obj*

汎用レジスタ
復帰命令

機能

レジスタ群 $\leftarrow (SP)$
 $SP \leftarrow SP + n$

説明

- ・スタックポインタの示すシステムスタックの内容を汎用レジスタに復帰したあと、SP を *obj* で指定したレジスタに対応するバイト数だけインクリメントします。各々のレジスタが指定された時の実際のスタックの動作については、”1.6 スタックの変化について”を参照してください。
- ・*obj* には、復帰するレジスタの型を指定します。
- ・スタックポインタは偶数バイト単位で変化します。*Rn* レジスタが指定された場合は、レジスタの復帰動作後、SP のみが1 インクリメントされるダミーサイクルが1 サイクル分挿入されます。ダミーサイクル実行中はレジスタの復帰は行われません。
- ・この命令実行中は、メモリ空間のワードバウンダリが発生しません。従って SP が奇数の場合も、そのアドレスから始まるデータが操作の対象になります。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

- : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	命令フォーマット					
		第1ワード					
POP	<i>Rn</i>	F	<i>n</i>	0	0	0	E
	<i>ERn</i>	F	<i>n</i>	0	0	1	E
	<i>XRn</i>	F	<i>n</i>	0	0	1	E
	<i>QRn</i>	F	<i>n</i>	0	0	1	E

PUSH レジスタリスト

コントロールレジスタ
退避命令

機 能

SP ← SP-n
(SP) ← コントロールレジスタ

説 明

- スタックポインタを レジスタリストで指定されたコントロールレジスタに対応するバイト数だけデクリメントした後、コントロールレジスタ群の内容をスタックポインタの示すシステムスタックに退避する命令です。各々のレジスタが指定された時の実際のスタックの動作については、“1.6 スタックの変化について”を参照してください。
- レジスタリストは、次のいずれかです。
 - ① PSW 退避レジスタ (EPSW)
 - ② 割込み用リンクレジスタ (ELR)
 - ③ サブルーチン用リンクレジスタ (LR)
 - ④ EA レジスタ (EA)
- レジスタリストは、全て記述する必要はありません。ただし少なくとも一つのレジスタリストを記述する必要があります。
- オペランドのレジスタ記述は順不同ですが、退避順は固定です。この命令を用いた場合のレジスタ退避順は
ELR → EPSW → LR → EA
となります。
- この命令実行中は、メモリ空間のワードバウンダリが発生しません。従って SP が奇数の場合も、そのアドレスから始まるデータが操作の対象になります。
- 通常、サブルーチンから復帰する場合は RT 命令を、割込みから復帰する場合は RTI 命令を使用しますが、サブルーチンがネストする場合、あるいは多重割込みが発生する可能性がある場合は、PUSH 命令で現在の退避レジスタの内容をスタックメモリに退避する必要があります。詳細は、“1.4 例外レベルと退避レジスタの取り扱いについて”を参照して下さい。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

- : 変化はありません。

命令フォーマット

ニーモ ニック	第 1 オペランド	命令フォーマット			
		第 1 ワード			
PUSH	EA	F	1	C	E
	ELR	F	2	C	E
	EA, ELR	F	3	C	E
	EPSW	F	4	C	E
	EPSW, EA	F	5	C	E
	EPSW, ELR	F	6	C	E
	EPSW, ELR, EA	F	7	C	E
	LR	F	8	C	E
	LR, EA	F	9	C	E
	LR, ELR	F	A	C	E
	LR, EA, ELR	F	B	C	E
	LR, EPSW	F	C	C	E
	LR, EPSW, EA	F	D	C	E
	LR, EPSW, ELR	F	E	C	E
	LR, EPSW, ELR, EA	F	F	C	E

PUSH *obj*

汎用レジスタ
退避命令

機能

SP ← SP - n
(SP) ← 汎用レジスタ

説明

- スタックポインタを *obj* で指定したレジスタに対応するバイト数だけデクリメントした後、汎用レジスタをスタックポインタの示すシステムスタックに退避します。*obj* には、退避するレジスタの型を指定します。
- スタックポインタは常に偶数バイト単位で変化します。 *Rn* レジスタが指定された場合は、レジスタの退避動作の前に、 *SP* のみが1デクリメントされるダミーサイクルが1サイクル分挿入されます。ダミーサイクル実行中はレジスタの退避は行われません。各々のレジスタが指定された時の実際のスタックの動作については、”1.6 スタックの変化について”を参照してください。
- この命令実行中は、メモリ空間のワードバウンダリが発生しません。従って *SP* が奇数の場合も、そのアドレスから始まるデータが操作の対象になります。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— ： 変化はありません。

命令フォーマット

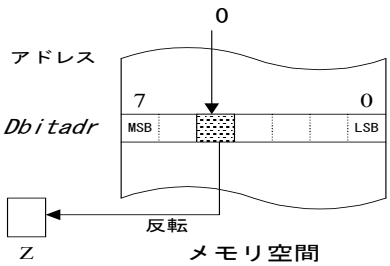
ニーモニック	第1オペランド	命令フォーマット									
		第1ワード									
PUSH	<i>Rn</i>	F	<i>n</i>	0	1	0	0	0	0	0	E
	<i>ERn</i>	F	<i>n</i>	0	0	1	0	0	0	0	E
	<i>XRn</i>	F	<i>n</i>	0	0	1	1	0	0	0	E
	<i>QRn</i>	F	<i>n</i>	0	0	1	1	1	0	0	E

RB *Dbitadr*

リセットビット命令

機 能

$Z \leftarrow \sim[Dbitadr]$
 $[Dbitadr] \leftarrow 0$



説 明

- 指定したメモリの 1 ビットの状態を読み出し、その結果を Z フラグに反映した後、そのビットをリセットする命令です。
- Dbitadr* には、ビットリセットするメモリのアドレスを *Dadr.bit* の形式で記述します。
- bit* は、0-7 までの整数で、リセットするビットの、ビット位置を記述します。

フラグ

C	Z	S	OV	MIE	HC
—	*	—	—	—	—

- Z : ビットの内容が 0 の時 1 になり、それ以外の場合は 0 になります。
— : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	DSR プリフィックスコード	命令フォーマット			
			第 1 ワード		第 2 ワード	
RB	<i>Dbitadr</i>		A	0	<i>bit</i>	2
	*: <i>Dbitadr</i>	<word>	A	0	<i>bit</i>	2

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

RB *Rn . bit_offset*

リセットビット命令

機 能

$$Z \leftarrow \sim Rn[bit_offset]$$

$$Rn[bit_offset] \leftarrow 0$$

説 明

- 指定したレジスタの1ビットの状態を読み出し、その結果をZフラグに反映した後、そのビットをリセットする命令です。
- bit_offset* には、リセットするビットの、ビット位置を記述します。

フラグ

C	Z	S	OV	MIE	HC
—	*	—	—	—	—

- Z : ビットの内容が0の時1になり、それ以外の時は0になります。
 — : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
RB	<i>Rn.bit_offset</i>		A n 0 bit 2	

RC

リセットキャリ命令

機 能

$C \leftarrow 0$

説 明

・Cフラグをリセットします。

フラグ

C	Z	S	OV	MIE	HC
*	—	—	—	—	—

C : 0 になります。
— : 変化はありません。

命令フォーマット

ニーモ ニック	第1 オペランド	第2 オペランド	命令フォーマット	
			第1 ワード	第2 ワード
RC			E B 7 F	

RT

サブルーチンからの復帰命令

機 能

CSR ← LCSR
PC ← LR

説 明

- ・ BL 命令より呼び出されたサブルーチンからの復帰命令です。
- ・ CSR にはサブルーチン用セグメントレジスタ (LCSR) の内容が復帰され、次の命令は復帰したコードセグメントからセグメント処理を行います。
- ・ PC にはサブルーチン用リンクレジスタ (LR) の内容が復帰され、次の命令は復帰した PC から処理を行います。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
RT			F E 1 F	

RTI

割込みからの
復帰命令

機 能

CSR ← ECSR[ELEVEL]
PC ← ELR [ELEVEL]
PSW ← EPSW[ELEVEL]

説 明

- ・割込みルーチンから復帰します。
- ・EPSW は例外処理用 PSW 退避レジスタです。現在の ELEVEL の値より、EPSW1(割込み用)、EPSW2(NMI 用)のいずれかが選択されて PSW に復帰されます。
- ・ELR は例外処理用リンクレジスタです。現在の ELEVEL の値より、ELR1(割込み用)、ELR2(例外処理用)のいずれかが選択されて PC に復帰されます。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	*	*

- * : EPSW の内容が書き込まれます
- : 変化はありません。

命令フォーマット

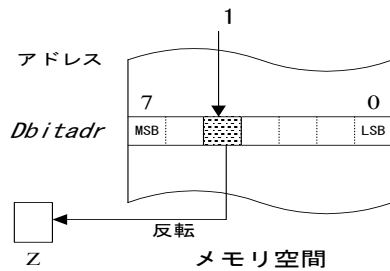
ニーモ ニック	第1 オペランド	第2 オペランド	命令フォーマット			
			第1 ワード		第2 ワード	
RTI			F	E	0	F

SB Dbitadr

セットビット命令

機 能

$Z \leftarrow \sim[Dbitadr]$
 $[Dbitadr] \leftarrow 1$



説 明

- *Dbitadr* で指定されたメモリの 1 ビットの状態を調べ、その結果を Z フラグに反映した後、そのビットをセットする命令です。
- *Dbitadr* は、ビットセットするメモリのアドレスを、*Dadr16.bit* の形式で記述します。
- *bit* は、0-7 までの整数で、セットするビットの、ビット位置を記述します。

フラグ

C	Z	S	OV	MIE	HC
—	*	—	—	—	—

- Z : ビットの内容が 0 の時 1 になり、それ以外の時は 0 になります。
 — : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	DSR プリフィックスコード	命令フォーマット	
			第 1 ワード	第 2 ワード
SB	<i>Dbitadr</i>		A 0 1 <i>bit</i> 0	<i>Dadr</i>
	*: <i>Dbitadr</i>	<word>	A 0 1 <i>bit</i> 0	<i>Dadr</i>

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

SB Rn . bit_offset

セットビット命令

機能

$Z \leftarrow \sim Rn[bit_offset]$
 $Rn[bit_offset] \leftarrow 1$

説明

- 指定したレジスタの1ビットの状態を調べ、その結果をZフラグに反映した後、そのビットをセットする命令です。
- bit_offset には、セットするビットの、ビット位置を記述します。

フラグ

C	Z	S	OV	MIE	HC
—	*	—	—	—	—

- Z : ビットの内容が0の時1になり、それ以外の場合は0になります。
— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
SB	Rn.bit_offset		A n 0 bit 0	

SC

セットキャリ命令

機能

$C \leftarrow 1$

説明

- ・キャリフラグを1にします。

フラグ

C	Z	S	OV	MIE	HC
*	—	—	—	—	—

C : 1になります。

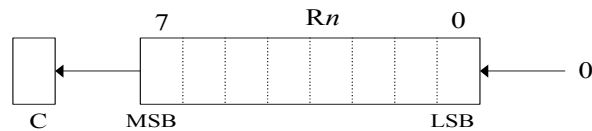
命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
SC			E : D : 8 : 0	

SLL *Rn* , *obj*

論理左シフト命令

機能



説明

- *Rn* の内容を第2オペランドで指定されたビット数だけ左に論理シフトし、最後にシフトアウトした値を *C* に格納します。
- シフト幅は 0-7 の範囲です。第2オペランドに *Rm* が指定された場合は、*Rm* のビット 2-0 のデータがシフト幅として読み込まれ、ビット 7-3 のデータは無視されます。シフト幅として 0 を指定した場合は NOP 命令と同じ動作となります。
- *LSB* には、シフト幅の数だけ 0 がシフトされて入ります。
- 本命令の直前に *SLLC* 命令を配置することにより、多バイトデータのシフト演算が可能となります。

フラグ

C	Z	S	OV	MIE	HC
*	—	—	—	—	-

- C* : シフト動作で、最後にキャリアウトしたデータがセットされます。
— : 変化はありません。

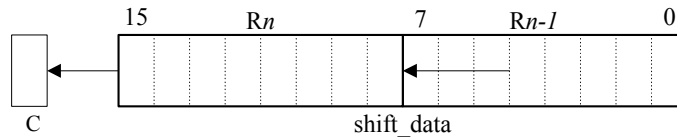
命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット				
			第1ワード				第2ワード
SLL	<i>Rn</i>	<i>Rm</i>	8	<i>n</i>	<i>m</i>	A	
		#width	9	<i>n</i>	0	width	A

SLLC Rn, obj

論理左シフト継続命令

機 能



説 明

- Rn と、 $Rn-1$ レジスタを、同時に左に最大7ビットシフトし、演算結果を Rn レジスタに格納します。
- シフトデータは、 Rn レジスタを上位バイト、 $Rn-1$ レジスタを下位バイトとするワード型データです。 $R0$ を上位バイトとして選択した場合、 $R15$ が下位バイトとなります。演算終了後は上位バイトが Rn レジスタに格納されます。
- obj の内容はシフトするビット幅で、命令に先立って 0-7 の値が格納されている必要があります。第2オペランドに Rm が指定された場合、 Rm のビット 2-0 のデータがシフト幅として読み込まれ、ビット 7-3 のデータは無視します。シフト幅として 0 を指定した場合、NOP 命令と同じ動作となります。
- 本命令を記述した直後に SLL 命令を配置することで、多バイトデータのシフト演算が可能となります。

例) ダブルワード長データの左シフト

```
SLLC R3, R5
SLLC R2, R5
SLLC R1, R5
SLL R0, R5    (XR0 のシフト演算完了)
```

フラグ

C	Z	S	OV	MIE	HC
*	—	—	—	—	—

- C : シフト動作で、最後にキャリアウトしたデータがセットされます。
— : 変化はありません。

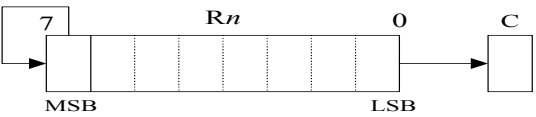
命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
SLLC	Rn	Rm	8	n	m	B
		$\#width$	9	n	0	$width$ B

SRA *Rn*, *obj*

算術右シフト命令

機 能



説 明

- *Rn* レジスタの内容を、第2オペランドで指定されたビット数だけ右算術シフトします。
- シフト幅は0-7の範囲です。第2オペランドに *Rm* が指定された場合、*Rm* のビット2-0のデータがシフト幅として読み込まれ、ビット7-3のデータは無視されます。シフト幅として0を指定した場合、NOP命令と同じ動作となります。
- *Rn* の最上位ビットは変化しません。

フラグ

C	Z	S	OV	MIE	HC
*	—	—	—	—	—

- C : 最後にビット0よりアウトした値が設定されます。
— : 変化はありません。

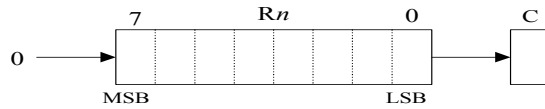
命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット				
			第1ワード				第2ワード
SRA	<i>Rn</i>	<i>Rm</i>	8	<i>n</i>	<i>m</i>	E	
		# <i>width</i>	9	<i>n</i>	0	<i>width</i>	E

SRL Rn, obj

論理右シフト命令

機 能



説 明

- Rn レジスタの内容を、第2オペランドで指定されたビット数だけ論理右シフトし、最後にシフトアウトされたビットをCに格納します。
- シフト幅は0-7の範囲です。第2オペランドに Rm が指定された場合、 Rm のビット2-0のデータがシフト幅として読み込まれ、ビット7-3のデータは無視されます。シフト幅として0を指定した場合、NOP命令と同じ動作となります。
- MSBにはシフト幅の値だけ0がシフトして入ります。
- 本命令の直前に SRLC 命令を配置することにより、多バイトデータのシフト演算が可能となります。

フラグ

C	Z	S	OV	MIE	HC
*	—	—	—	—	—

- C : 最後にビット0よりアウトした値が設定されます。
— : 変化はありません。

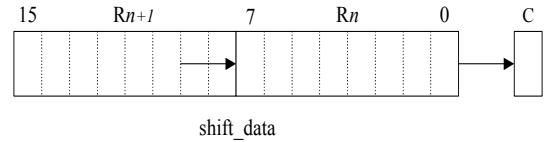
命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
SRL	Rn	Rm	8	n	m	C
		$\#width$	9	n	0	$width$ C

SRLC *Rn* , *obj*

論理右シフト継続命令

機 能



説 明

- *Rn* と、*Rn+1* レジスタを、同時に右に最大 7 ビットシフトし、演算結果を *Rn* レジスタに格納します。
- シフトデータは *Rn* レジスタを下位バイト、*Rn+1* レジスタを上位バイトとするワード型データです。演算終了後の下位バイトが *Rn* レジスタに格納されます。*R15* を下位バイトとして選択した場合、*R0* が上位バイトとなります。演算終了後は下位バイトが *Rn* レジスタに格納されます。
- *obj* の内容はシフトするビット幅で、命令に先立って 0-7 の値が格納されている必要があります。第 2 オペランドに *Rm* が指定された場合、*Rm* のビット 2-0 のデータがシフト幅として読み込まれ、ビット 7-3 のデータは無視されます。シフト幅として 0 を指定した場合、NOP 命令と同じ動作となります。
- 本命令を記述した直後に SRL 命令を配置することで、多バイトデータのシフト演算が可能となります。

例) ダブルワード長データの右シフト
SRLC *R0*, *R5*
SRLC *R1*, *R5*
SRLC *R2*, *R5*
SRL *R3*, *R5* (XR0 のシフト演算完了)

フラグ

C	Z	S	OV	MIE	HC
*	—	—	—	—	—

C : 最後にビット 0 よりアウトした値が設定されます。
— : 変化はありません。

命令フォーマット

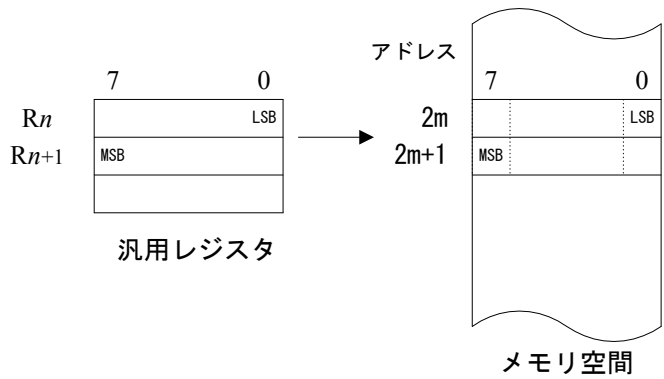
ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット			
			第 1 ワード		第 2 ワード	
SRLC	<i>Rn</i>	<i>Rm</i>	8	<i>n</i>	<i>m</i>	D
		#width	9	<i>n</i>	0 width	D

ST ERn , obj

ワード型転送命令

機 能

$obj \leftarrow ERn$



説 明

- ・ ERn レジスタの内容を、 obj で示される内容をアドレスとするメモリに、ワード型でストアします。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

次ページに示します。

第3章 命令の詳細
 インストラクションセット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックスコード	命令フォーマット				
				第1ワード			第2ワード	
ST	ER <i>n</i>	[EA]		9	<i>n</i>	3	3	
		*:[EA]	<word>	9	<i>n</i>	3	3	
		[EA+]		9	<i>n</i>	5	3	
		*:[EA+]	<word>	9	<i>n</i>	5	3	
		[ER <i>m</i>]		9	<i>n</i>	<i>m</i>	3	
		*:[ER <i>m</i>]	<word>	9	<i>n</i>	<i>m</i>	3	
		<i>Disp16</i> [ER <i>m</i>]		A	<i>n</i>	<i>m</i>	9	<i>Disp16</i>
		*: <i>Disp16</i> [ER <i>m</i>]	<word>	A	<i>n</i>	<i>m</i>	9	<i>Disp16</i>
		<i>Disp6</i> [BP]		B	<i>n</i>	10	<i>Disp6</i>	
		*: <i>Disp6</i> [BP]	<word>	B	<i>n</i>	10	<i>Disp6</i>	
		<i>Disp6</i> [FP]		B	<i>n</i>	11	<i>Disp6</i>	
		*: <i>Disp6</i> [FP]	<word>	B	<i>n</i>	11	<i>Disp6</i>	
		<i>Dadr</i>		9	<i>n</i>	1	3	<i>Dadr</i>
		*: <i>Dadr</i>	<word>	9	<i>n</i>	1	3	<i>Dadr</i>

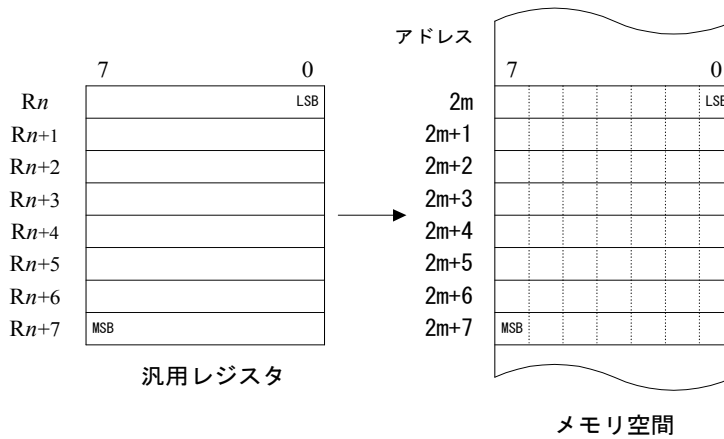
*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

ST QRn , obj

ダブルワード型
連続転送命令

機 能

$obj \leftarrow QRn$



説 明

- ・QRn レジスタの内容を、obj で指定されるメモリに、クワットワード型でストアします。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックス コード	命令フォーマット			
				第1ワード		第2ワード	
ST	QRn	[EA]		9	n	3	7
		*:[EA]	<word>	9	n	3	7
		[EA+]		9	n	5	7
		*:[EA+]	<word>	9	n	5	7

*	<word>			
pseg_addr	E	3	pseg_addr	
DSR	F	E	9	F
Rd	9	0	d	F

ST *Rn* , *obj*

バイト型転送命令

機能

obj ← *Rn*

説明

- ・ *Rn* レジスタの内容を、*obj* で示される値をアドレスとするメモリに、バイト型でストアします。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

次頁に示します。

ニーモニック	第1オペランド	第2オペランド	DSR プリフィックス コード	命令フォーマット				
				第1ワード				第2ワード
ST	Rn	[EA]		9	n	3	1	
		*:[EA]	<word>	9	n	3	1	
		[EA+]		9	n	5	1	
		*:[EA+]	<word>	9	n	5	1	
		[ERm]		9	n	m	1	
		*:[ERm]	<word>	9	n	m	1	
		Disp16[ERm]		9	n	m	9	Disp16
		*:Disp16[ERm]	<word>	9	n	m	9	Disp16
		Disp6[BP]		D	n	0	Disp6	
		*:Disp6[BP]	<word>	D	n	0	Disp6	
		Disp6[FP]		D	n		Disp6	
		*:Disp6[FP]	<word>	D	n		Disp6	
		Dadr		9	n	1	1	Dadr
		*:Dadr	<word>	9	n	1	1	Dadr

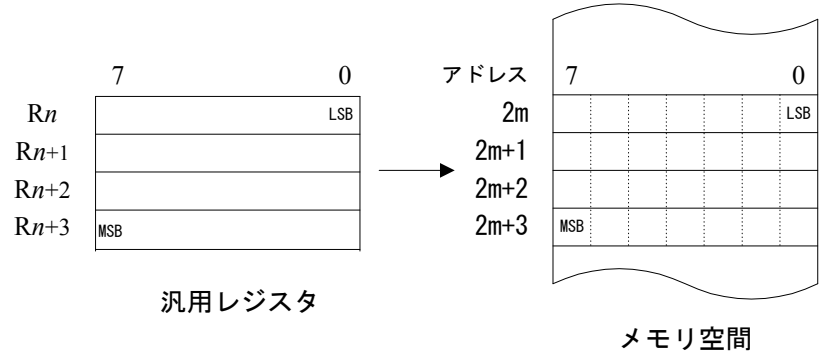
*	<word>			
pseg_addr	E	3	pseg_addr	
DSR	F	E	9	F
Rd	9	0	d	F

ST *XRn*, *obj*

ダブルワード型転送命令

機能

obj ← *XRn*



説明

- *XRn* レジスタの内容を、*obj* で指定されるメモリにダブルワード型でストアします。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	—	—

— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	DSRプリフィックスコード	命令フォーマット			
				第1ワード		第2ワード	
ST	<i>XRn</i>	[EA]		9	<i>n</i>	3	5
		*:[EA]	<word>	9	<i>n</i>	3	5
		[EA+]		9	<i>n</i>	5	5
		*:[EA+]	<word>	9	<i>n</i>	5	5

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

SUB Rn, Rm

減算命令

機能

$Rn \leftarrow Rn - Rm$

説明

- Rn レジスタの内容から、 Rm レジスタの内容を減算し、結果を Rn レジスタに格納します。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット7にボローが発生した時に1になり、それ以外の時は0になります。
- Z : 実行結果が0の時1になり、それ以外の時は0になります
- S : 実行の結果のビット7最上位ビットがセットされます。
- OV : オーバフローがセットされた時1になり、それ以外の時は0になります。
- HC : ビット3にキャリあるいボローが生じた時1になり、それ以外の時は0になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット			
			第1ワード		第2ワード	
SUB	Rn	Rm	8	n	m	8

SUBC Rn, Rm

キャリ付き減算命令

機能

$$Rn \leftarrow Rn - Rm - C$$

説明

・ Rn レジスタの内容から、 Rm レジスタの内容とキャリを減算します。

フラグ

C	Z	S	OV	MIE	HC
*	*	*	*	—	*

- C : 演算の結果、ビット 7 にボローが発生した時に 1 になり、それ以外の時は 0 になります。
- Z : 実行前の Z の内容が 0 の時は、実行結果に関わらず 0 になります。
実行前の Z の内容が 1 の時は、実行結果が 0 の時 1 になり、それ以外の時は 0 になります
- S : 実行の結果のビット 7 最上位ビットがセットされます。
- OV : オーバフローがセットされた時 1 になり、それ以外の時は 0 になります。
- HC : ビット 3 にキャリあるいはボローが生じた時 1 になり、それ以外の時は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット	
			第 1 ワード	第 2 ワード
SUBC	Rn	Rm	8 n m 9	

SWI #snum

ソフトウェア割込み

機 能

EPSW1 ←PSW
ELEVEL ← 1
ELR1 ← PC+2
ECSR1 ←CSR
MIE ←0
PC ←TABLE[snum<<1]

説 明

- ・ snum で指定されたベクタテーブルの内容を PC に書き込みます。
- ・ snum は 0-63 までの整数で、割込み先のアドレスを格納しているベクタテーブルの、ベクタ番号を指定します。
- ・ 割込みサイクル中に次の命令の先頭アドレスは ELR1 に退避されます。

フラグ

C	Z	S	OV	MIE	HC
—	—	—	—	*	—

MIE : 0 になります。
— : 変化はありません。

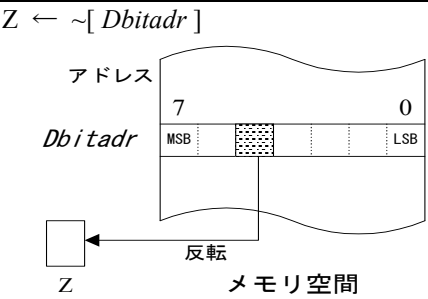
命令フォーマット

ニーモニック	第1オペランド	命令フォーマット			
		第1ワード			第2ワード
SWI	#snum	E	5	00	snum

TB *Dbitadr*

テストビット命令

機 能



説 明

- 指定したメモリの 1 ビットの内容をテストし、結果を Z フラグに格納します。
- Dbitadr* は、ビットテストするメモリのアドレスを、*Dadr16.bit* の形式で記述します。
- bit* は、0-7 までの整数で、リセットするビットのビット位置を記述します。

フラグ

C	Z	S	OV	MIE	HC
—	*	—	—	—	—

- Z : ビットの内容が 0 の時 1 になり、それ以外の場合は 0 になります。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	DSR プリフィックス コード	命令フォーマット	
			第 1 ワード	第 2 ワード
TB	<i>Dbitadr</i>		A 0 1 <i>bit</i> 1	<i>Dadr</i>
	*: <i>Dbitadr</i>	<word>	A 0 1 <i>bit</i> 1	<i>Dadr</i>

*	<word>			
<i>pseg_addr</i>	E	3	<i>pseg_addr</i>	
DSR	F	E	9	F
<i>Rd</i>	9	0	<i>d</i>	F

TB *Rn . bit_offset*

テストビット命令

機能

$$Z \leftarrow \sim Rn[bit_offset]$$

説明

- 指定したレジスタの1ビットの内容をテストし、結果をZフラグに格納します。
- bit_offset* には、テストするビットのビット位置を記述します。

フラグ

C	Z	S	OV	MIE	HC
—	*	—	—	—	—

- Z : ビットの内容が0の時1になり、それ以外の時は0になります。
— : 変化はありません。

命令フォーマット

ニーモニック	第1オペランド	第2オペランド	命令フォーマット	
			第1ワード	第2ワード
TB	<i>Rn</i> .bit		A <i>n</i> bit 1	

XOR *Rn* , *obj*

排他的論理輪命令

機 能

$Rn \leftarrow Rn \wedge obj$

説 明

- ・ *Rn* レジスタの内容と *obj* の内容の排他的論理和をとり、結果を *Rn* レジスタに格納します。

フラグ

C	Z	S	OV	MIE	HC
—	*	*	—	—	—

- Z : 実行結果が 0 の時 1 になり、それ以外の時は 0 になります。
- S : 実行結果の最上位ビットがセットされます。
- : 変化はありません。

命令フォーマット

ニーモニック	第 1 オペランド	第 2 オペランド	命令フォーマット			
			第 1 ワード		第 2 ワード	
XOR	<i>Rn</i>	<i>Rm</i>	8	<i>n</i>	<i>m</i>	4
		<i>#imm8</i>	4	<i>n</i>	<i>imm8</i>	

4付録

U8コアのインストラクション表、および、U8コアの旧バージョン（A2コア）のマニュアルと、現バージョン（A3コア）の差異を記載しています。

インストラクション表では、DSRプリフィックスコード指定は省略しています。
各命令の詳細な記述については、第3章を参照して下さい。

4.1 インストラクション表

演算命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
ADD	R_n	R_m	*	*	*	*		*	1000_ <i>nnnn</i> _ <i>mmmm</i> _0001		1
		$\#imm8$	*	*	*	*		*	0001_ <i>nnnn</i> _ <i>iiii</i> _ <i>iiii</i>		1
ADD	ER_n	ER_m	*	*	*	*		*	1111_ <i>nnn0</i> _ <i>mmm0</i> _0110		2
		$\#imm7$	*	*	*	*		*	1110_ <i>nnn0</i> _ <i>iii</i> _ <i>iiii</i>		2
ADDC	R_n	R_m	*	*	*	*		*	1000_ <i>nnnn</i> _ <i>mmmm</i> _0110		1
		$\#imm8$	*	*	*	*		*	0110_ <i>nnnn</i> _ <i>iiii</i> _ <i>iiii</i>		1
AND	R_n	R_m		*	*				1000_ <i>nnnn</i> _ <i>mmmm</i> _0010		1
		$\#imm8$		*	*				0010_ <i>nnnn</i> _ <i>iiii</i> _ <i>iiii</i>		1
CMP	R_n	R_m	*	*	*	*		*	1000_ <i>nnnn</i> _ <i>mmmm</i> _0111		1
		$\#imm8$	*	*	*	*		*	0111_ <i>nnnn</i> _ <i>iiii</i> _ <i>iiii</i>		1
CMPC	R_n	R_m	*	*	*	*		*	1000_ <i>nnnn</i> _ <i>mmmm</i> _0101		1
		$\#imm8$	*	*	*	*		*	0101_ <i>nnnn</i> _ <i>iiii</i> _ <i>iiii</i>		1
MOV	ER_n	ER_m		*	*				1111_ <i>nnn0</i> _ <i>mmm0</i> _0101		2
		$\#imm7$		*	*				1110_ <i>nnn0</i> _0 <i>iii</i> _ <i>iiii</i>		2
MOV	R_n	R_m		*	*				1000_ <i>nnnn</i> _ <i>mmmm</i> _0000		1
		$\#imm8$		*	*				0000_ <i>nnnn</i> _ <i>iiii</i> _ <i>iiii</i>		1
OR	R_n	R_m		*	*				1000_ <i>nnnn</i> _ <i>mmmm</i> _0011		1
		$\#imm8$		*	*				0011_ <i>nnnn</i> _ <i>iiii</i> _ <i>iiii</i>		1
XOR	R_n	R_m		*	*				1000_ <i>nnnn</i> _ <i>mmmm</i> _0100		1
		$\#imm8$		*	*				0100_ <i>nnnn</i> _ <i>iiii</i> _ <i>iiii</i>		1
CMP	ER_n	ER_m	*	*	*	*		*	1111_ <i>nnn0</i> _ <i>mmm0</i> _0111		2
SUB	R_n	R_m	*	*	*	*		*	1000_ <i>nnnn</i> _ <i>mmmm</i> _1000		1
SUBC	R_n	R_m	*	*	*	*		*	1000_ <i>nnnn</i> _ <i>mmmm</i> _1001		1

シフト命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
SLL	R_n	R_m	*						1000_ <i>nnnn</i> _ <i>mmmm</i> _1010		1
		$\#width$	*						1001_ <i>nnnn</i> _0 <i>www</i> _1010		1
SLLC	R_n	R_m	*						1000_ <i>nnnn</i> _ <i>mmmm</i> _1011		1
		$\#width$	*						1001_ <i>nnnn</i> _0 <i>www</i> _1011		1
SRA	R_n	R_m	*						1000_ <i>nnnn</i> _ <i>mmmm</i> _1110		1
		$\#width$	*						1001_ <i>nnnn</i> _0 <i>www</i> _1110		1
SRL	R_n	R_m	*						1000_ <i>nnnn</i> _ <i>mmmm</i> _1100		1
		$\#width$	*						1001_ <i>nnnn</i> _0 <i>www</i> _1100		1
SRLC	R_n	R_m	*						1000_ <i>nnnn</i> _ <i>mmmm</i> _1101		1
		$\#width$	*						1001_ <i>nnnn</i> _0 <i>www</i> _1101		1

ロード/ストア命令

ニーモニック	第 1	第 2	フラグの変化						命令コード		最小実行 サイクル
	オペランド	オペランド	C	Z	S	OV	MIE	HC	第 1 ワード	第 2 ワード	
L	ER <i>n</i>	[EA]	*	*					1001_ <i>nnn</i> 0_0011_0010		2
		[EA+]	*	*					1001_ <i>nnn</i> 0_0101_0010		2
		[ER <i>m</i>]	*	*					1001_ <i>nnn</i> 0_ <i>mmm</i> 0_0010		2
		Disp16[ER <i>m</i>]	*	*					1010_ <i>nnn</i> 0_ <i>mmm</i> 0_1000	DDDD_ DDDD_ DDDD_ DDDD	3
		Disp6[BP]	*	*					1011_ <i>nnn</i> 0_00DD_ DDDD		3
		Disp6[FP]	*	*					1011_ <i>nnn</i> 0_01DD_ DDDD		3
		Dadr	*	*					1001_ <i>nnn</i> 0_0001_0010	DDDD_ DDDD_ DDDD_ DDDD	2
	R <i>n</i>	[EA]	*	*					1001_ <i>nnnn</i> _0011_0000		1
		[EA+]	*	*					1001_ <i>nnnn</i> _0101_0000		1
		[ER <i>m</i>]	*	*					1001_ <i>nnnn</i> _ <i>mmm</i> 0_0000		1
		Disp16[ER <i>m</i>]	*	*					1001_ <i>nnnn</i> _ <i>mmm</i> 0_1000	DDDD_ DDDD_ DDDD_ DDDD	2
		Disp6[BP]	*	*					1101_ <i>nnnn</i> _00DD_ DDDD		2
		Disp6[FP]	*	*					1101_ <i>nnnn</i> _01DD_ DDDD		2
		Dadr	*	*					1001_ <i>nnnn</i> _0001_0000	DDDD_ DDDD_ DDDD_ DDDD	2
	XR <i>n</i>	[EA]	*	*					1001_ <i>nn</i> 00_0011_0100		4
		[EA+]	*	*					1001_ <i>nn</i> 00_0101_0100		4
	QR <i>n</i>	[EA]	*	*					1001_ <i>n</i> 000_0011_0110		8
		[EA+]	*	*					1001_ <i>n</i> 000_0101_0110		8

ニーモニック	第 1 オペランド	第 2 オペランド	フラグの変化							命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第 1 ワード	第 2 ワード		
ST	ER <i>n</i>	[EA]							1001_ <i>nnn</i> 0_0011_0011		2	
		[EA+]							1001_ <i>nnn</i> 0_0101_0011		2	
		[ER <i>m</i>]							1001_ <i>nnn</i> 0_ <i>mmm</i> 0_0011		2	
		Disp16[ER <i>m</i>]							1010_ <i>nnn</i> 0_ <i>mmm</i> 0_1001	DDDD_ DDDD_ DDDD_ DDDD	3	
		Disp6[BP]							1011_ <i>nnn</i> 0_10DD_ DDDD		3	
		Disp6[FP]							1011_ <i>nnn</i> 0_11DD_ DDDD		3	
		Dadr							1001_ <i>nnn</i> 0_0001_0011	DDDD_ DDDD_ DDDD_ DDDD	2	
	R <i>n</i>	[EA]							1001_ <i>nnnn</i> _0011_0001		1	
		[EA+]							1001_ <i>nnnn</i> _0101_0001		1	
		[ER <i>m</i>]							1001_ <i>nnnn</i> _ <i>mmm</i> 0_0001		1	
		Disp16[ER <i>m</i>]							1001_ <i>nnnn</i> _ <i>mmm</i> 0_1001	DDDD_ DDDD_ DDDD_ DDDD	2	
		Disp6[BP]							1101_ <i>nnnn</i> _10DD_ DDDD		2	
		Disp6[FP]							1101_ <i>nnnn</i> _11DD_ DDDD		2	
		Dadr							1001_ <i>nnnn</i> _0001_0001	DDDD_ DDDD_ DDDD_ DDDD	2	
	XR <i>n</i>	[EA]							1001_ <i>nn</i> 00_0011_0101		4	
		[EA+]							1001_ <i>nn</i> 00_0101_0101		4	
	QR <i>n</i>	[EA]							1001_ <i>n</i> 000_0011_0111		8	
		[EA+]							1001_ <i>n</i> 000_0101_0111		8	

コントロールレジスタアクセス命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
ADD	SP	<i>#signed8</i>							1110_0001_iiii_iiii		2
MOV	ECSR	<i>Rm</i>							1010_0000_mmmmm_1111		2
	ELR	<i>ERm</i>							1010_mmm0_0000_1101		3
	EPSW	<i>Rm</i>							1010_0000_mmmmm_1100		1
	<i>ERn</i>	ELR							1010_nnn0_0000_0101		3
		SP							1010_nnn0_0001_1010		2
	PSW	<i>Rm</i>	*	*	*	*	*	*	1010_0000_mmmmm_1011		1
		<i>#unsigned8</i>	*	*	*	*	*	*	1110_1001_iiii_iiii		1
	<i>Rn</i>	ECSR							1010_nnnn_0000_0111		2
		EPSW							1010_nnnn_0000_0100		2
		PSW							1010_nnnn_0000_0011		1
	SP	<i>ERm</i>							1010_0001_mmm0_1010		1

PUSH/POP 命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
PUSH	<i>ERn</i>								1111_nnn0_0101_1110		2
	<i>QRn</i>								1111_n000_0111_1110		8
	<i>Rn</i>								1111_nnnn_0100_1110		2
	<i>XRn</i>								1111_nn00_0110_1110		4
	<i>register_list</i>								1111_lepa_1100_1110		2-12
POP	<i>ERn</i>								1111_nnn0_0001_1110		2
	<i>QRn</i>								1111_n000_0011_1110		8
	<i>Rn</i>								1111_nnnn_0000_1110		2
	<i>XRn</i>								1111_nn00_0010_1110		4
	<i>register_list</i>		*	*	*	*	*	*	1111_lepa_1000_1110		2-15

コプロセッサ転送

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
MOV	CR _n	R _m							1010_nnnn_mmmm_1110		1
	CER _n	[EA]							1111_nnn0_0010_1101		2
		[EA+]							1111_nnn0_0011_1101		2
	CR _n	[EA]							1111_nnnn_0000_1101		1
		[EA+]							1111_nnnn_0001_1101		1
	CXR _n	[EA]							1111_nn00_0100_1101		4
		[EA+]							1111_nn00_0101_1101		4
	CQR _n	[EA]							1111_n000_0110_1101		8
		[EA+]							1111_n000_0111_1101		8
	R _n	CR _m							1010_nnnn_mmmm_0110		1
	[EA]	CER _m							1111_mmm0_1010_1101		2
	[EA+]	CER _m							1111_mmm0_1011_1101		2
	[EA]	CR _m							1111_mmmm_1000_1101		1
	[EA+]	CR _m							1111_mmmm_1001_1101		1
	[EA]	CXR _m							1111_mm00_1100_1101		4
	[EA+]	CXR _m							1111_mm00_1101_1101		4
	[EA]	CQR _m							1111_m000_1110_1101		8
	[EA+]	CQR _m							1111_m000_1111_1101		8

EA レジスタ転送命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
LEA	[ER _m]								1111_0000_mmm0_1010		1
	Disp16[ER _m]								1111_0000_mmm0_1011	DDDD_DDDD_DDDD_DDDD	2
	Dadr								1111_0000_0000_1100	DDDD_DDDD_DDDD_DDDD	2

ALU 命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
DAA	R _n		*	*	*			*	1000_nnnn_0001_1111		1
DAS	R _n		*	*	*			*	1000_nnnn_0011_1111		1
NEG	R _n		*	*	*	*		*	1000_nnnn_0101_1111		1

ビットアクセス命令

ニーモニック	第 1 オペランド	第 2 オペランド	フラグの変化							命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第 1 ワード	第 2 ワード		
SB	<i>Rn.bit_offset</i>			*					1010_nnnn_0bbb_0000		1	
	<i>Dbitadr</i>			*					1010_0000_1bbb_0000	DDDD_DDDD_DDDD_DDDD	2	
RB	<i>Rn.bit_offset</i>			*					1010_nnnn_0bbb_0010		1	
	<i>Dbitadr</i>			*					1010_0000_1bbb_0010	DDDD_DDDD_DDDD_DDDD	2	
TB	<i>Rn.bit_offset</i>			*					1010_nnnn_0bbb_0001		1	
	<i>Dbitadr</i>			*					1010_0000_1bbb_0001	DDDD_DDDD_DDDD_DDDD	2	

PSW アクセス命令

ニーモニック	第 1	第 2	フラグの変化							命令コード		最小実行 サイクル
	オペランド	オペランド	C	Z	S	OV	MIE	HC	第 1 ワード	第 2 ワード		
EI								*	1110_1101_0000_1000		1	
DI								*	1110_1011_1111_0111		3	
SC			*						1110_1101_1000_0000		1	
RC			*						1110_1011_0111_1111		1	
CPLC			*						1111_1110_1100_1111		1	

条件相対分岐命令

ニーモニック	第 1	第 2	フラグの変化					命令コード		最小実行	
	オペランド	オペランド	C	Z	S	OV	MIE	HC	第 1 ワード	第 2 ワード	サイクル
BGE	RadR								1100_0000_rrrr_rrrr		1/3
BLT									1100_0001_rrrr_rrrr		1/3
BGT									1100_0010_rrrr_rrrr		1/3
BLE									1100_0011_rrrr_rrrr		1/3
BGES									1100_0100_rrrr_rrrr		1/3
BLTS									1100_0101_rrrr_rrrr		1/3
BGTS									1100_0110_rrrr_rrrr		1/3
BLES									1100_0111_rrrr_rrrr		1/3
BNE									1100_1000_rrrr_rrrr		1/3
BEQ									1100_1001_rrrr_rrrr		1/3
BNV									1100_1010_rrrr_rrrr		1/3
BOV									1100_1011_rrrr_rrrr		1/3
BPS									1100_1100_rrrr_rrrr		1/3
BNS									1100_1101_rrrr_rrrr		1/3
BAL									1100_1110_rrrr_rrrr		3

符号拡張命令

ニーモニック	第1	第2	フラグの変化						命令コード		最小実行 サイクル
	オペランド	オペランド	C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
EXTBW	ERn			*	*				1000 nnn1 nnn0 1111		1

ソフトウェア割込み命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
SWI	# <i>snum</i>					*			1110_0101_00ii_iiii		3
BRK									1111_1111_1111_1111		7

分岐命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
B	<i>Cadr</i>								1111_gggg_0000_0000	cccc_cccc_cccc_cccc	2
	ER <i>n</i>								1111_0000_nnn0_0010		2
BL	<i>Cadr</i>								1111_gggg_0000_0001	cccc_cccc_cccc_cccc	2
	ER <i>n</i>								1111_0000_nnn0_0011		2

乗除算命令

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
MUL	ER <i>n</i>	R <i>m</i>			*				1111_nnn0_mmmmm_0100		9
DIV	ER <i>n</i>	R <i>m</i>			*	*			1111_nnn0_mmmmm_1001		17

その他

ニーモニック	第1 オペランド	第2 オペランド	フラグの変化						命令コード		最小実行 サイクル
			C	Z	S	OV	MIE	HC	第1ワード	第2ワード	
INC	[EA]			*	*	*		*	1111_1110_0010_1111		2
DEC	[EA]			*	*	*		*	1111_1110_0011_1111		2
RT									1111_1110_0001_1111		2
RTI			*	*	*	*	*	*	1111_1110_0000_1111		2
NOP									1111_1110_1000_1111		1

4.2 A2 コアと A3 コアのインストラクションマニュアル記載内容の相違

A2コアインストラクションマニュアルの名称:PJULU8-100-INST-02

A3コアインストラクションマニュアルの名称:FJUZ0317A0-U8-INST-01

■DI命令のマシンサイクルの相違

A3コア:3

A2コア:1

■DI命令の注意

A2コア:

MIE は、この命令の実行サイクルを含めて3サイクル後に0になります。従ってMIE が1 の状態でDI 命令を実行しても、その直後の2サイクルはマスカブル割込みの許可状態が継続しますので、アプリケーションプログラム設計時に注意を払う必要があります。

A3コア:

この注意の必要がありません。

■POP命令の、PUSH命令の説明

A2コア:

この命令に先だって、PUSH 命令で、対応するコントロールレジスタの内容がシステムスタックに退避されていることが必要です。

A3コア:

ROMWINDOWからのPOPを可能にしたため、この使用方法の制約はなくなりました。

改版履歷

改版履歴

ドキュメント No.	発行日	ページ		変更内容
		改版前	改版後	
PJULU8-100-INST-01	2000.11	-	-	暫定初版発行
PJULU8-100-INST-02	2001.4	1-6	1-6	H C フラグの変化条件を修正
		3-18	3-18	HC フラグの変化条件を修正
		3-19	3-19	HC フラグの変化条件を修正
		3-31	3-31	HC フラグの変化条件を修正
FJUZO317A0-U8-INST-01	2004.12.19	1-23~ 1-24	1-23~ 1-25	割り込み処理サイクル数を追記
		3-6	3-6	以下の命令の記載内容を修正 MOV ECSR , Rm MOV ELR , ERm MOV EPSW , Rm MOV ERm , ELR MOV Rn , ECSR PUSH POP
		3-9	3-9	以下の命令において、DSR プリフィックス指定の記述を追加。 SB Dbitadr RB Dbitadr TB Dbitadr EXTBW 命令の機能について、表現方法を修正。
		3-10	3-10	以下の命令の機能において、記載を修正。 分岐命令 INC [EA] DEC [EA] RT RTI INC・DEC 命令において、DSR プリフィックス指定の記述を追加。
		3-11~ 3-16	3-11~ 3-21	割り込み発生時のウェイト挿入を追記。 DSR プリフィックスコードを指定した場合の、命令実行時間を追加
		3-12	3-12	BRK 命令において、命令実行サイクル数を修正
		3-17	3-17	POP PC,LR 命令において、実行サイクル数を修正
		3-25	3-30	[ERn] という記載を ERn に修正
		3-27	3-32	BGE 命令において、フラグ条件の意味を以下のように修正 符号あり → 符号無し
		3-62~ 3-65	3-67~ 3-70	命令フォーマットの欄において DSR プリフィックスコードの表を追加
		3-9	3-9	DI命令のマシンサイクルを修整
		3-38	3-43	DI命令の命令において、動作機能を修整

ドキュメント No.	発行日	ページ		変更内容
		改版前	改版後	
FJUZ0317A0-U8-INST-01	2004.12.19	3-77	3-82	命令の説明を追加
		3-79	3-84	命令の説明を追加
		3-80	3-85	命令の説明を追加
		3-82	3-87	命令の説明を追加
FJUZ0317A0-U8-INST-02	2006.7.25	1-6	1-6	MIEフラグの表現を修正
		1-9	1-9	1.2.2.5項の誤記を修正
		1-15	1-15	プログラム記述例中の誤記を修正
		1-17	1-17	概要図中の誤記を修正
		1-19	1-19	説明図中の誤記を修正
		1-29	1-29	説明文中の誤記を修正
		3-12	3-12	不要な注釈を削除
		3-40	3-40	誤字修正
		3-85	3-85	誤字修正
FJUZ0317A0-U8-INST-03	2008.10.15	表紙	表紙	ロゴをOKIセミコンダクタに変更 バージョン及び日付変更
		はじめに	はじめに	OKIセミコンダクタオリジナル8ビット1チップマイクロ コントローラのCPUコアに変更
		ご注意	ご注意	OKIセミコンダクタに変更 バージョン及び日付変更
FJUZ0317A0-U8-INST-04	2009.06.01	1-27 1-30	1-27 1-31	ノンマスクابل割り込みの多重割り込みに対する対応を追記
			1-32	ノンマスクابل割り込みに対する注意項を追加
FJUZ0317A0-U8-INST-05	2011.10.11	3-12 3-17	3-12 3-17	以下の命令の実行サイクル数を修正 DIV ERn, ERm MOV EPSW, Rm MUL ERn, ERm
		3-16	3-16	以下の命令の[EA+]アドレッシングの影響を追記 MOV SP, ERm
		4-3 4-6	4-3 4-6	以下の命令の実行サイクル数を修正 DIV ERn, ERm MOV EPSW, Rm MUL ERn, ERm

ご注意

1. 本書に記載された内容は、製品改善及び技術改良等により将来予告なしに変更することがあります。したがって、ご使用の際には、その情報が最新のものであることをご確認ください。
2. 本書に記載された動作概要及び応用回路例は、本製品の標準的な動作や使い方を説明するためのものです。したがって、実際に本製品を使用される場合には、外部諸条件を考慮のうえ回路・実装設計をしてください。
3. **設計に際しましては、最大定格、動作電源電圧範囲、放熱特性など保証範囲内でお使いください。保証値を超えての使用など本製品の誤った使用または不適切な使用等に起因する本製品の具体的な運用結果につきましては、当社は責任を負いかねますのでご了承ください。**
4. 本製品及び本書に記載された情報や図面等の使用に関して、当社は、第三者の工業所有権・知的所有権及びその他の権利に対する保証または実施権の許諾を行うものではありません。したがって、その使用に起因する第三者の権利侵害に対し、当社は責任を負いかねますのでご了承ください。
5. 当社は品質、信頼性の向上に努めておりますが、部品の性格上、ある確率の欠陥、故障が不可避だと考えられます。当社製品をお使いの場合には、このような故障が生じても直接人命を脅かしたり、身体または財産に危害を生じさせないよう、装置やシステム上で十分な安全設計をお願いします。
6. 本書記載の製品は、一般電子機器(事務機器、通信機器、計測機器、家電製品など)に使用されることを意図しております。特別な品質・信頼性が要求され、その故障や誤動作が直接人命を脅かしたり、身体または財産に危害を及ぼす恐れのある装置やシステム(交通機器、安全装置、航空・宇宙機器、原子力制御、生命維持装置を含む医療機器など)に使用をお考えのお客様は、必ず事前に当社販売窓口までご相談願います。
7. 本書に記載された製品には、「外国為替及び外国貿易管理法」に基づく戦略物資等に該当するものがあります。したがって、該当製品またはその一部を輸出する場合には、同法に基づく日本国政府の輸出許可が必要となりますので、その申請手続きをお取りください。
8. 本書の内容については万全を期しておりますが、お気付きの点等がございましたら下記の弊社ホームページ URL からご連絡下さい。
9. 本書に記載された内容を、当社に無断で転載または複製することはご遠慮ください。

Copyright 2008 - 2011 LAPIS Semiconductor Co., Ltd.

ラピスセミコンダクタ株式会社

〒193-8550 東京都八王子市東浅川町 550-1
<http://www.lapis-semi.com/jp/>

nX-U8/100

インストラクションマニュアル

2011 年 10 月 第 5 版発行

FJUZ0317A0-U8-INST